Zdobądź nowych użytkowników smartfonów!

Rusz głową!

Mobile Web

Zbuduj raz,
uruchamiaj
wszędzie

Udzielaj większego wsparcia
(swoim użytkownikom)

Znajdź drogę
z geolokalizacją

Skieruj swoje
strony na
małoekranową
dietę

Twórz swoje projekty zgodnie
z *Responsive Web Design*

Lyza Danger Gardner, Jason Grigsby

O'REILLY®

Helion

Opinie o książce *Mobile Web. Rusz głową!*

„Jeśli kiedykolwiek planowałeś zakup książki na temat tworzenia treści dla urządzeń mobilnych, która sprawdza się dla wielu przeglądarek i urządzeń od różnych producentów, możesz zaprzestać poszukiwań i kupić *Mobile Web*. *Rusz głową!* Jest to książka napisana przez nieprawdopodobnie bystrych ludzi, którzy mają wielkie doświadczenie z urządzeniami mobilnymi i nie poprzestają na jednej platformie, lecz pracują na wszystkich. Wielu twórców oprogramowania bezustannie zastanawia się, czy tworzyć programy od razu z myślą o urządzeniach mobilnych, czy pisać normalne aplikacje internetowe. Ta książka gładko przechodzi od tematów wprowadzających do zaawansowanych, dostarczając wszystkich informacji potrzebnych do tworzenia ekscytujących treści dla urządzeń mobilnych”.

— Andrea Trasatti, lider projektu DeviceAtlas i współtwórca WURFL, biblioteki informacji o możliwościach urządzeń bezprzewodowych

„Praktyczne wprowadzenie w chaotyczny świat współczesnego programowania dla urządzeń mobilnych, dające próbkę tego, jak możemy i jak powinniśmy podchodzić do nich jutro. *Mobile Web. Rusz głową!* skutecznie przedstawia praktyczne techniki, które czytelnik może natychmiast zastosować, i jednocześnie daje mnóstwo fundamentów oraz zasobów, na których mogą bazować doświadczeni programiści”.

— Stephen Hay, projektant stron internetowych, programista, mówca i współzałożyciel konferencji Mobilism

„Od samego początku nastawiona na praktykę książka *Mobile Web. Rusz głową!* jest znakomitym wprowadzeniem w wyzwania i perspektywy ujawniające się podczas eksplorowania nowych obszarów projektowania treści internetowych”.

— Bryan i Stephanie Riegerowie, założyciele yiibu.com

Opinie o innych książkach z serii *Rusz głową!*

„*Analiza i projektowanie obiektowe. Rusz głową!* to odświeżające spojrzenie na projektowanie i analizę zorientowaną obiektowo. Tym, co odróżnia tę książkę od innych na rynku, jest nacisk na uczenie się. Autorzy sprawili, że temat ten stał się przystępny i dostępny dla praktyków".

— Ivar Jacobson, Ivar Jacobson Consulting

„Właśnie skończyłem czytać książkę *Analiza i projektowanie obiektowe. Rusz głową!* i jestem zachwycony! Najbardziej podobało mi się położenie nacisku na to, dlaczego właściwie zajmujemy się projektowaniem zorientowanym obiektowo — żeby pisać świetne programy".

— Kyle Brown, naczelny inżynier IBM

„Za zabawnymi ilustracjami i szalonymi czcionkami kryje się poważna, inteligentna, niezwykle starannie zaprojektowana prezentacja na temat projektowania i analizy zorientowanej obiektowo. Gdy czytałem tę książkę, czułem się, jakbym podglądał przy pracy doświadczonego projektanta, który dodatkowo wyjaśnia mi, co i dlaczego jest istotne na każdym kroku".

— Edward Sciore, adiunkt, Wydział Informatyki, Boston College

„*Head First Software Development. Edycja polska* to świetna książka dla każdego, kto chce utrwalić swoje umiejętności programistyczne na sposób, który stale angażuje na wielu poziomach".

— Andy Hudson, Linux Format

„Jeśli jesteś początkującym twórcą oprogramowania, *Head First Software Development. Edycja polska* od razu skieruje Cię na dobre tory. Jeśli zaś jesteś już doświadczonym programistą (czytaj: o długim stażu), nie rezygnuj z niej tak szybko..."

— Thomas Duff, Duffbert's Random Musings

„W *Java. Rusz głową!* jest coś dla każdego. Wzrokowcy, kinestetycy — każdy czegoś się z tej książki nauczy. Dzięki pomocom wizualnym łatwiej jest zapamiętywać materiał, a tekst jest napisany bardzo przystępnie, co odróżnia tę książką od większości podręczników Javy... *Java. Rusz głową!* to bezcenna książka. Wyobrażam sobie, że książki z serii *Rusz głową!* można wykorzystać jako tekst obowiązkowy na zajęciach, czy to na uczelni wyższej, czy też na kursach dla dorosłych. Z pewnością będę w przyszłości sięgał do tej książki i polecał ją innym".

— Warren Kelly, Blogcritics.org, marzec 2006

„W przeciwieństwie do typowego stylu podręcznikowego *Head First iPhone and iPad Development* wnosi do nauczania programowania w iOS element humoru, a nawet sprawia, że uczenie się jest interesujące i przyjemne. Omówione są wszystkie kluczowe technologie, również z warstwy jądra, a także tematy tak istotne jak projektowanie interfejsu — treść książki jest mądrze dobrana i opracowana na najwyższym poziomie. Gdzie indziej będziesz mógł podsłuchać rozmowy przy kominku pomiędzy UIWebView a UITextField?!"

— Sean Murphy, projektant i programista iOS

Opinie o innych książkach z serii *Rusz głową!*

„Kolejna zaleta drugiego wydania *Java. Rusz głową!* to fakt, że jej lektura zaostrza apetyt. Omówione pod koniec książki bardziej zaawansowane tematy, takie jak Swing i RMI, sprawiają, że nie możesz się doczekać zanurkowania w API i napisania takiego właśnie doskonałego programu na 100 000 linii kodu, który przyniesie Ci sławę i pieniądze. Jest tu też wiele materiału, a nawet wiedzy o dobrych praktykach, na temat połączeń sieciowych i wątków — co jest akurat moim słabym punktem. Tutaj, przyznam się, szczerze się uśmiałem, gdy autorzy użyli operatorki telefonicznej z lat 50. — tak, dobrze myślisz, to ta pani z trwałą na głowie, która ręcznie przepina kable — jako analogii dla portów TCP/IP... Naprawdę, powinieneś po prostu przejść się do księgarni i przejrzeć drugą edycję *Java. Rusz głową!*. Nawet jeśli już znasz Javę, na pewno nauczysz się czegoś nowego. A jeśli nie, to samo wertowanie tej książki to świetna zabawa”.

— Robert Eckstein, Java.sun.com

„*Java. Rusz głową!* nie wyróżnia się oczywiście zakresem poruszonych tematów, tylko stylem i nastawieniem autorów. Ta książka tak różni się od typowego podręcznika informatycznego lub poradnika technicznego, jak to tylko możliwe. Autorzy zamieścili w niej kreskówki, quizy i magnesy na lodówkę (tak, magnesy na lodówkę...). Tam, gdzie zwykle znalazłbyś przygotowane dla czytelnika ćwiczenia, tutaj masz się wcielić w kompilator i skompilować kod albo samemu złożyć segment kodu poprzez wypełnienie luk, albo... sam rozumiesz. Pierwsze wydanie tej książki polecaliśmy jako tekst wprowadzający dla osób rozpoczynających swoją przygodę z Javą i obiektami. Nowe wydanie nie rozczarowuje i jest udaną kontynuacją poprzedniego. Jeśli jesteś jedną z tych osób, które zasypiają nad tradycyjnym tekstem informatycznym, to przy tej książce pozostaniesz przytomny i gotów do nauki”.

— TechBookReport.com

„*Head First Web Design. Edycja polska* to Twoja przepustka do opanowania wszystkich tych złożonych tematów i zrozumienia, o co tak naprawdę chodzi w świecie projektowania stron internetowych. Jeśli nie przeszedłeś jeszcze inicjacji z czymś tak złożonym jak Dreamweaver, to ta książka będzie dla Ciebie świetnym sposobem na nauczenie się projektowania”.

— Robert Pritchett, MacCompanion

„Czy można naprawdę nauczyć się projektowania internetowego z książki? *Head First Web Design. Edycja polska* uczy projektowania przyjaznych dla użytkownika stron, od poznania wymagań klienta poczynając, przez ręcznie rysowane schematy stron, na gotowych stronach skutecznie funkcjonujących online kończąc. Tym, co odróżnia ten tekst od innych książek typu »jak stworzyć stronę internetową«, jest wykorzystanie przez autorów najnowszej wiedzy z zakresu nauk kognitywnych oraz technik nauczania i stworzenie bogatego wizualnie doświadczenia edukacyjnego zaprojektowanego zgodnie z tym, jak działa mózg i jak postępuje proces uczenia się. Efektem jest solidny tekst na temat podstaw projektowania internetowego, który powinien się znaleźć w każdej biblioteczce informatycznej i stać się kluczem do sukcesu”.

— Diane C. Donovan, California Bookwatch: The Computer Shelf

„Z pełnym przekonaniem polecam *Head First Web Design. Edycja polska* wszystkim moim kolegom programistom, którzy chcieliby zapoznać się z tą bardziej artystyczną częścią naszej pracy”.

— Claron Twitchell, UJUG

Inne książki z serii *Rusz głową!*

Analiza i projektowanie obiektowe. Rusz głową!

C#. Rusz głową!

Excel. Rusz głową!

Head First Ajax. Edycja polska

Head First Algebra. Edycja polska

Head First C

Head First Design Patterns. Edycja polska

Head First EJB. Edycja polska

Head First HTML with CSS & XHTML. Edycja polska

Head First JavaScript. Edycja polska

Head First Object-Oriented Analysis and Design. Edycja polska

Head First PHP & MySQL. Edycja polska

Head First Ruby on Rails. Edycja polska

Head First Servlets & JSP. Edycja polska. Wydanie II

Head First Software Development. Edycja polska

Head First Web Design. Edycja polska

Head First. Fizyka. Edycja polska

Head First. Sieci komputerowe. Edycja polska

Head First. Statystyka. Edycja polska

HTML5. Rusz głową!

Java. Rusz głową! Wydanie II

jQuery. Rusz głową!

SQL. Rusz głową!

Wzorce projektowe. Rusz głową!

Cudownym kobietom należącym do mojej rodziny: siostrze Maggie, mamie Fran, cioci Catherine, teściowej Christie, a przede wszystkim świętej pamięci babci Pearl, której zapał i niezależność zainspirowały kolejne pokolenia.

— Lyza

Moim rodzicom, dzięki którym wiele lat temu rozpocząłem przygodę z Commodore 64. Ukochanej żonie, Danie, za jej wsparcie oraz nieprzebrane pokłady cierpliwości i wyrozumiałości, a także Katie i Danny'emu — tak, teraz mogę się z wami pobawić.

— Jason

Lyza Danger Gardner (*@lyzadanger*) jest programistką. Tworzyła, psuła i hakowała w sieci od 1996 roku. Co ciekawe, Lyza urodziła się i wychowała w Portland w stanie Oregon — mieście, w którym wszyscy chcą się znaleźć, ale *z którego* nikt nie wydaje się pochodzić.

Lyza wcześnie poszła do college'u i dorobiła się prawdziwie eklektycznej edukacji: licencjat z nauk humanistycznych na Uniwersytecie Stanowym Portland, a następnie magisterka z informatyki na Uniwersytecie w Birmingham.

Lyza napisała masę aplikacji sieciowych (również po stronie serwera), walczyła z podstępnymi systemami zarządzania treścią, optymalizowała strony pod kątem zastosowań mobilnych, pracowała z licznymi API i poświęciła wiele uwagi bazom danych. Zafascynowana tym, jak technologie mobilne zmieniają świat, poświęca obecnie wiele czasu na zastanawianie się nad przyszłością technologii mobilnych i nie tylko.

Od czasu założenia firmy Cloud Four zajmującej się technologiami mobilnymi, mającej swoją siedzibę w Portland, Lyza dzielnie pokonuje niezbadane tereny Krainy Urządzeń, zgłębiając chaotyczny świat przeglądarek i urządzeń mobilnych. W wolnych chwilach oddaje się dziwacznym i anachronicznym zajęciom, między innymi wykonuje niebagatelne ilości fotografii. Jest właścicielką domeny *.com* o czteroliterowej nazwie. Gdy już zgadniesz tę nazwę, możesz wpaść i złożyć jej wizytę.

W 2000 roku **Jason Grigsby** dostał swój pierwszy telefon komórkowy. Od tego czasu obsesyjnie zajmuje go myśl, jak piękny byłby świat, gdyby wszyscy ludzie mieli w kieszeniach urządzenia umożliwiające dostęp do wszystkich informacji na świecie. W czasie, gdy poznała go Dana — jego przyszła żona — ściany jego mieszkania były pokryte szalonymi marzeniami na temat mobilnego świata. Do dziś Jason zastanawia się, jakim cudem Dana w końcu wyszła za niego za mąż.

Marzenia te zderzyły się niestety z rzeczywistością — WAP był po prostu fatalny. Jason pracował więc w branży sieciowej do 2007 roku, kiedy to iPhone pokazał wszystkim, że nadszedł już czas. Jason połączył siły z trzema najbystrzejszymi osobami, jakie znał, i założył Cloud Four.

Od czasu, gdy współzałożył Cloud Four, miał przyjemność pracować przy wielu fantastycznych projektach, m.in. Obama iPhone App. Jest założycielem i prezesem Mobile Portland, organizacji non profit zajmującej się współpracą ze społecznością mobilną w Portland w stanie Oregon.

Jason jest wziętym mówcą i konsultantem. Jest dziś jeszcze bardziej nakręcony na technologie mobilne, niż był w 2000 roku (przepraszam, kochanie!).

Jason bloguje na *http://cloudfour.com* oraz na swojej prywatnej stronie *http://userfirstweb.com*; jest też na Twitterze jako *@grigs*. Wpadnij i się przywitaj!

Spis treści (skrócony)

Spis treści (z prawdziwego zdarzenia)

Wprowadzenie

Twój mózg kontra technologie mobilne. Starasz się czegoś nauczyć, a mózg robi Ci „przysługę”, za wszelką cenę odciągając Twoją uwagę od nauki. Myśli: „Lepiej wyjdź i zajmij się czymś ciekawszym — wypatruj krwiożerczych bestii albo sprawdź, czy kiedy podpalisz swoje BlackBerry Bold, włączy się alarm pożarowy”. Jak w takim razie oszukać mózg, by uznał, że Twoje życie zależy od znajomości technologii mobilnych?

1 Wprowadzenie do mobilnych technologii webowych

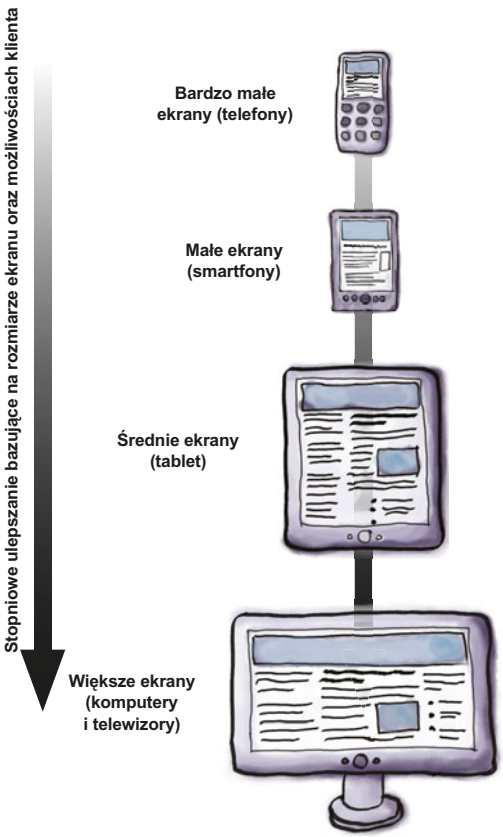
Wrażliwe projekty, czyli Responsive Web Design

Witajcie! Jesteście gotowi na mobilne technologie webowe? Tworzenie witryn na urządzenia mobilne jest naprawdę ekscytujące. Wiele w tym uroku, emocji i momentów, w których chciałoby się wykrzyknąć: ***Eureka!***. Ale z drugiej strony pełno tu tajemnic i trudności. Technologie mobilne rozwijają się w tak niewiarygodnym tempie, że cały czas jesteśmy trochę w tyle. Trzymaj się więc mocno! Naszą przygodę rozpoczynamy od ciekawego podejścia do tworzenia witryn internetowych, znanego jako ***Responsive Web Design (RWD)***. Dzięki niemu będziesz mógł sprawić, by strony wyglądały równie dobrze na wielu różnych urządzeniach mobilnych i, co ważne, przydadzą Ci się umiejętności, które już masz.

RWD na poważnie

2 Koncepcja Mobile First w podejściu Responsive Web Design

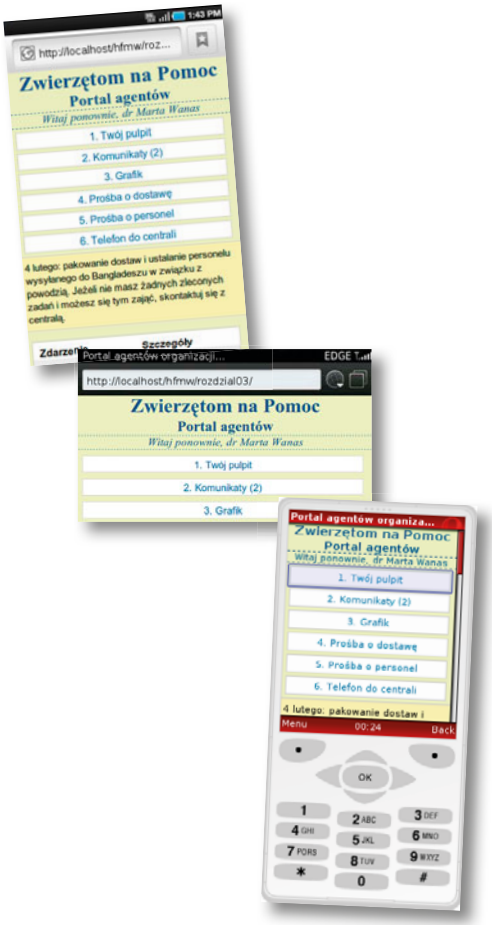
Oto śliczna mobilna witryna. Ale nie oceniaj jej tylko po pozorach. Pod tą piękną powłoką znajdziesz bowiem coś zupełnie innego. Być może wygląda jak mobilna witryna, ale to wciąż zwykła, desktopowa witryna, z tym że przebrana w mobilne ciuszki. Jeśli chcesz, żeby na urządzeniach mobilnych chodziła jak dobrze naoliwiona maszynka, musisz zastosować zasadę **Mobile First**. Jednak najpierw musimy przeprowadzić sekcję obecnej witryny, by odnaleźć ukrywający się w jej wnętrzu desktopowy szkielet. Następnie gruntownie posprzątamy i zaczniemy pracować na świeżo, zgodnie ze strategią **stopniowego ulepszania**, zaczynając od budowania podstawowych elementów, a kończąc na bogatej wersji desktopowej. Gdy skończymy, nasza strona będzie zoptymalizowana pod każdą możliwą rozdzielczość ekranu.



Oddzielna witryna mobilna

3 Stawiamy czoła niezupełnie sprzyjającym okolicznościom

Wizja jednego, wrażliwego projektu witryny jest cudowna... Mamy tylko jeden układ strony opracowany w zgodzie z koncepcją Mobile First, który dopasowuje się do specyfiki różnych przeglądarek i urządzeń. Brzmi pięknie. Co się jednak stanie, gdy tę wizję przyprawimy choćby szczyptą realizmu? Niezaktualizowane systemy, stare urządzenia lub ograniczenia budżetu klienta to tylko kilka przykładów. A co, jeśli zamiast łączyć wersję desktopową z mobilną, chciałbyś je rozdzielić? W tym rozdziale przyjrzymy się szczegółowo **wykrywaniu użytkowników korzystających z urządzeń mobilnych, wspieraniu starszych telefonów i tworzeniu odrębnych witryn dla urządzeń mobilnych.**



Komu wsparcie, komu?

4 Które urządzenia powinny być obsługiwane?

Przetestowanie witryny na wszystkich urządzeniach trwałoby wieczność. Musisz ustalić granicę wyodrębniającą urządzenia, które zamierzasz wspierać. **Ale jak podjąć tę decyzję?** Co z użytkownikami korzystającymi z telefonów, które już dawno powinny trafić na złomowisko? W jaki sposób stworzyć witrynę, by działała na urządzeniach, o których w ogóle nie słyszałeś? W tym rozdziale przygotujemy magiczną miksturę złożoną z **wymagań projektowych** i **informacji o odbiorcach**. Pomoże nam ona zdecydować, **które urządzenia mamy wspierać** i **co zrobić z tymi, których nie wspieramy**.

Bazy i klasy urządzeń

5 Zapoznaj się z grupą

Podczas wybierania wspieranych urządzeń nie wzięliśmy pod uwagę kilku dokuczliwych problemów. Jak mamy się dowiedzieć wystarczająco dużo o mobilnych przeglądarkach użytkowników, by przed dostarczeniem treści spełnić ich oczekiwania? Jak uniknąć tworzenia tylko podstawowej zawartości odpowiadającej najmniejszemu wspólnemu mianownikowi urządzeń? No i jak, pozostając przy zdrowych zmysłach, zapanować nad tym wszystkim? W tym rozdziale wkroczymy w świat **możliwości urządzeń** i nauczymy się korzystać z **baz danych urządzeń**, by wreszcie odkryć, jak te wszystkie urządzenia grupować w **klasy**.

6 Framework dla mobilnych aplikacji internetowych
Tartanator

„My chcemy aplikację!”. Jeszcze rok czy dwa lata temu tego typu hasło wiązało się nieodłącznie z jednym — tworzeniem natywnych aplikacji dla każdego urządzenia, które zamierzaliśmy wesprzeć. Na szczęście teraz nie jest to jedyne możliwe rozwiązanie. Aplikacje internetowe dla mobilnych przeglądarek są coraz doskonalsze, zwłaszcza ostatnio, kiedy wkroczył **HTML5** wraz z nieodłącznymi kompanami — **CSS3** i **JavaScriptem**. W świat mobilnych aplikacji internetowych wejdziemy wraz z **mobilnym frameworkiem**, czyli zbiorem gotowych rozwiązań programistycznych upraszczających i przyspieszających tworzenie aplikacji.

Mobilne aplikacje w prawdziwym świecie

7 Wyjątkowe mobilne aplikacje internetowe

Mobilne aplikacje są jak dzieci w klasie. Wiesz, chodzi o tę mieszankę fascynacji, przekonania o tym, że można zrobić niesamowite rzeczy, ale też tajemniczości i niepohamowanego rozrabiactwa. Staramy się trzymać pod kontrolą tych nadpobudliwych geniuszy, ustalając granice, a także dbając, by ich nie przekraczali. Przychodzi jednak czas zbierania korzyści z naturalnych talentów mobilnych aplikacji internetowych. Możemy zastosować metodę **stopniowego ulepszania**, by dopieścić interfejs na potrzeby lepiej rozwiniętych przeglądarek, a problemy z ciągłością dostępu do internetu rozwiązać za pomocą **trybu offline**. Możemy też wydobyć esencję mobilności, korzystając z **geolokalizacji**. Nie ma czasu do stracenia, zabierzmy się za tworzenie wyjątkowych mobilnych aplikacji internetowych!

Tworzenie hybrydowych aplikacji mobilnych z PhoneGap

8 Ustrzel tartan! — w stronę natywności

Czasem musisz się zdecydować na aplikację natywną. Być może potrzebujesz dostępu do czegoś, co nie jest (jeszcze) osiągalne z poziomu przeglądarki. A może klient chce, by aplikacja *koniecznie* znalazła się w sklepie. Nie możemy się już doczekać chwili, gdy przeglądarka będzie udostępniała wszystko, co jest potrzebne, by tworzyć pełnoprawne aplikacje mobilne. Jednak zanim do tego dojdzie, możemy skorzystać z możliwości **hybrydowego podejścia** — nadal będziemy tworzyć aplikacje z wykorzystaniem **technologii internetowych**, ale użyjemy **biblioteki, która pełni rolę mostu** między naszym kodem a natywnymi możliwościami urządzeń. **Międzyplatformowe natywne aplikacje zbudowane w oparciu o technologie internetowe?** Brzmi całkiem nieźle!

Podejście „future friendly”

9 Odnajdywanie (jakiegoś) sensu w chaosie

Responsive Web Design. Wykrywanie urządzeń. Mobilne aplikacje internetowe. PhoneGap. Chwila... czego właściwie powinienem użyć?

Obecnie istnieje wiele metod tworzenia aplikacji za pomocą technologii mobilnych. Bardzo często w projektach **łączy się wiele technik**. Nie ma jednego, najlepszego rozwiązania, ale nie przejmuj się, ponieważ kluczem do sukcesu jest nadążanie za rozwojem technologii. **Nie bój się wyzwań**. Wystarczy przyswoić **podejście „future friendly”** i dać się ponieść fali, będąc przeświadczonym o swojej elastyczności i gotowości na wszystko, co przyniesie przyszłość.

Ścinki

A

Sześć najważniejszych spraw (o których nie mówiliśmy)

Czujesz się, jakby coś Ci umknęło? Wiemy, co masz na myśli... Zawsze jest tak samo — myślisz, że to już koniec, a okazuje się, że jest tego więcej. Nie mogliśmy się pohamować, by nie przekazać Ci kilku dodatkowych szczegółów, o których nie wspomnieliśmy w treści książki. Gdybyśmy chcieli napisać o wszystkich ciekawych sprawach, książkę musiałbyś transportować w pancernej walizie na kołach. Rzuć okiem, co dla Ciebie wybraliśmy.

Postaw swój serwer

B

Gdzieś trzeba zacząć

„Mobilny internet" nie istnieje bez słowa „internet". Bez dwóch zdań. Jeśli chcesz się zająć tworzeniem witryn i aplikacji mobilnych, będziesz potrzebował serwera WWW. Prędzej czy później dojdzie do sytuacji, w której będziesz potrzebował części serwerowej swojej aplikacji. Możesz wtedy skorzystać z bezpłatnego lub komercyjnego hostingu albo uruchomić serwer na swoim komputerze. W tym dodatku opiszemy proces **stawiania lokalnego serwera WWW z obsługą PHP** z wykorzystaniem bezpłatnego oprogramowania.

Instalowanie WURFL

Jak wywęszyć urządzenia?

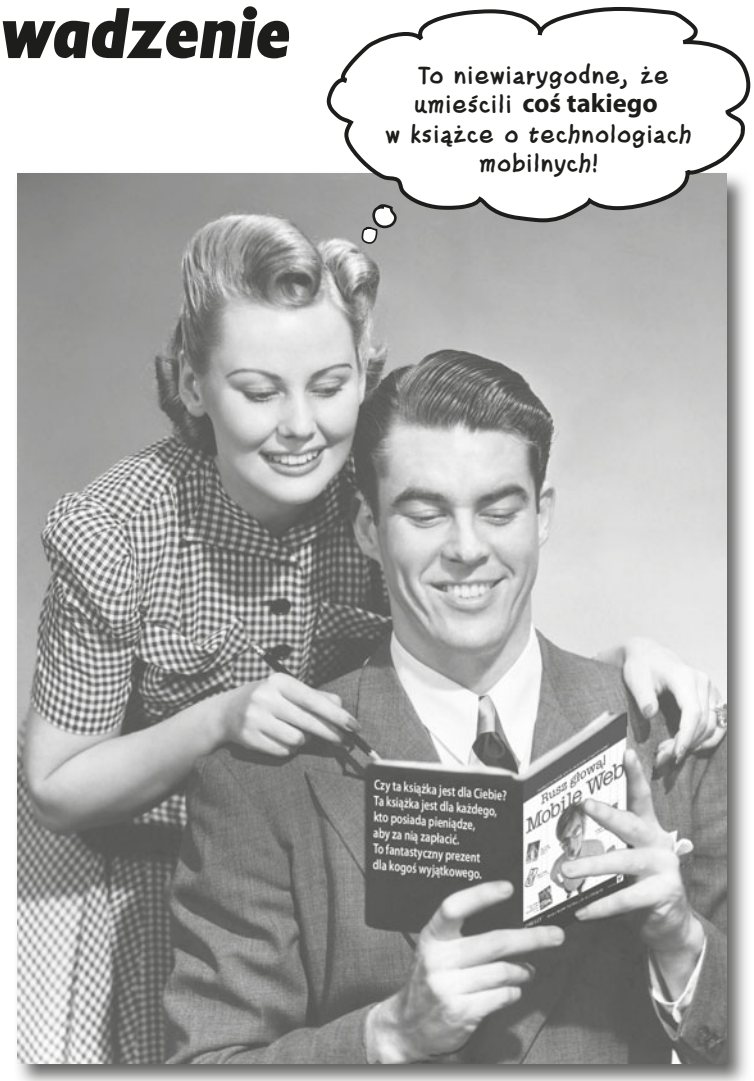
Pierwszy krok do rozwiązania tajemnicy wykrywania urządzeń wymaga trochę zachodu. Każdy przyzwoity gliniarz wie, że trzeba zbierać poszlaki i przesłuchiwać świadków. Musimy zacząć od mózgu całej operacji, którym jest **WURLF API dla PHP**. Później przyjdzie kolej na mięśniaka — jeden **plik XML** zawierający informacje o możliwościach tysięcy urządzeń. Nie będzie jednak łatwo zmusić ich do współpracy, więc będziemy musieli poświęcić im trochę czasu i pogrzebać w **konfiguracji**.

Instalowanie SDK i narzędzi dla Androida

Zadbaj o środowisko

Aby gruntownie testować natywne aplikacje pod Androida, musisz przygotować sobie odpowiednie środowisko. Musisz dołączyć swój komputer do małego ekosystemu, do którego zagonisz wszystkie aplikacje Androida uruchamiane zarówno na wirtualnych (emulowanych), jak i rzeczywistych urządzeniach. Abyś mógł stać się pasterzem tego stada, pokażemy Ci, jak pobrać **SDK (ang. *Software Development Kit*) Androida**, jak zainstalować niektóre **narzędzia dla tej platformy**, jak **utworzyć wirtualne urządzenia** i jak **instalować oraz odinstalowywać aplikacje**.

Wprowadzenie



*W tym rozdziale poznasz odpowiedź na palące pytanie: „Dlaczego autorzy **UMIEŚCILI** coś takiego w książce pod tytułem Mobile Web?”.*

Dla kogo jest ta książka?

Jeżeli możesz odpowiedzieć twierdząco na wszystkie poniższe pytania:

Na pewno Ci pomoże przynajmniej podstawowa wiedza na temat języków skryptowych. W książce nie korzystamy co prawda z jakichś wyjątkowo skomplikowanych skryptów, ale chodzi o to, byś nie wpadł w panikę na widok kawałka kodu w JavaScripcie.

1. Czy masz doświadczenie w projektowaniu i tworzeniu witryn internetowych?
2. Czy chcesz **opanować**, **zrozumieć**, **zapamiętać** i ***tworzyć*** internetowe projekty dla urządzeń mobilnych, tak by Twoje witryny były bardziej interaktywne i doskonalsze?
3. Czy od **drętwych i nudnych publikacji akademickich** wolisz **wciągające dyskusje przy posiłku**?

...ta książka jest właśnie dla Ciebie.

Kto raczej nie powinien sięgać po tę książkę?

Jeżeli możesz odpowiedzieć twierdząco na którekolwiek z poniższych pytań:

1. Czy jesteś **zupełnym nowicjuszem** w technologiach internetowych?
2. Tworzysz już mobilne aplikacje internetowe i szukasz raczej ***dokumentacji*** technologii mobilnych?
3. **Boisz się spróbować czegoś nowego?** Wolałbyś raczej poddać się leczeniu kanałowemu, niż przyjąć do wiadomości, że jest więcej niż jedna metoda tworzenia witryn i aplikacji internetowych? Czy naprawdę uważasz, że nie można traktować poważnie książki technicznej, w której pojawia się witryna pubu Pod Paradnym Morsem czy aplikacja o wdzięcznej nazwie Tartanator?

...raczej nie powinieneś po nią sięgać.

[Notatka z działu marketingu: ta książka jest dla wszystkich, którzy mają kartę kredytową. Ewentualnie może też być gotówka].

Wiemy, co sobie myślisz

„Że niby *to* ma być poważna książka o tworzeniu mobilnych witryn i aplikacji?!”

„Po co te wszystkie obrazki?”

„Czy w ten sposób naprawdę mogę się czegokolwiek *nauczyć*?”

Wiemy też, co sobie myśli Twój mózg

Twój mózg pragnie nowości. Zawsze szuka, przegląda i *wyczekuje* czegoś niezwykłego. Tak jest skonstruowany i dzięki temu masz szansę przeżyć.

Co w takim razie Twój mózg robi z tymi wszystkimi rutynowymi, zwyczajnymi, normalnymi informacjami, które do niego docierają? Otóż robi *wszystko*, co tylko może, aby nie przeszkadzały mu w jego najważniejszej funkcji — rejestrowaniu wszystkiego tego, co ma *prawdziwe znaczenie*. Nie traci czasu i energii na zapamiętywanie nudnych informacji, ponieważ stosuje filtr odrzucający od razu wszystko, co jest „całkowicie nieistotne”.

Ale w takim razie skąd mózg *wie*, co jest istotne? Wyobraź sobie, że wędrujesz sobie po lesie i nagle wyskakuje na Ciebie tygrys. Co dzieje się w Twojej głowie i Twoim ciele?

Neurony płoną. Emocje szaleją. *Adrenalina aż kipi.*

I właśnie dzięki temu Twój mózg wie, że...

To jest bardzo ważne! Nie zapomnij tego!

Wyobraź sobie teraz, że siedzisz w domu albo bibliotece. Jest bezpiecznie i miło, a w okolicy nie grasują żadne tygrysy. Uczysz się. Przygotowujesz się do egzaminu. A może starasz się zgłębić jakiś trudny techniczny problem, co — według szacunków Twojego szefa — nie powinno zająć więcej niż tydzień, góra dziesięć dni.

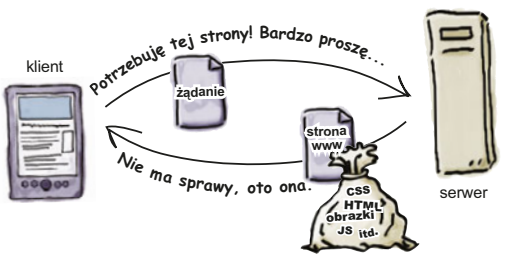
Jest tylko jeden problem. Twój mózg stara Ci się oddać przysługę. Stara się sprawić, by *bezdyskusyjnie* nieistotne informacje nie zajęły cennych zasobów w Twojej głowie. Zasobów, które powinny zostać wykorzystane na zapamiętanie *naprawdę ważnych* rzeczy. Takich jak tygrysy. Albo ogień. Albo tego, że już nigdy nie powinieneś jeździć na snowboardzie w krótkich spodenkach.

Niestety nie można po prostu powiedzieć mózgowi: „Hej, mózgu, jestem ci naprawdę wdzięczny, ale niezależnie od tego, jak nudna jest ta książka i jak mało emocji wyzwala, bardzo cię proszę, byś *pozwolił* mi zapamiętać wszystkie te informacje”.

Czytelnika tej książki traktujemy jak ucznia.

A zatem, czy chcesz się czegoś *nauczyć*? Jeśli tak, to w pierwszej kolejności powinieneś to *zrozumieć*, a później zadbać, by tego *nie zapomnieć*. Nie chodzi tu o wtłaczanie wiedzy do głowy. Najnowsze badania z zakresu przyswajania informacji, neurobiologii i psychologii nauczania jasno wskazują, że *uczenie się* wymaga czegoś więcej niż tylko czytania tekstu. My wiemy, co potrafi pobudzić Twój mózg do działania.

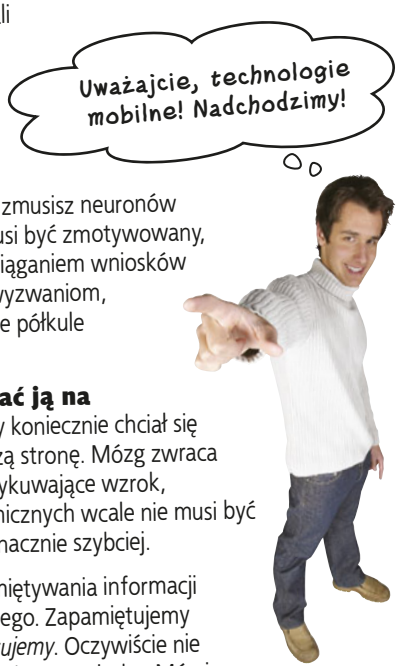
Oto wybrane zasady obowiązujące w książkach z serii „Rusz głową!":



Zwizualizuj to sobie. Obrazy są znacznie łatwiejsze do zapamiętania niż sam tekst i sprawiają, że nauka staje się o wiele bardziej efektywna (można osiągnąć nawet 89-procentowy wzrost efektywności w przypominaniu i przekazywaniu informacji). Co więcej, rysunki sprawiają, że przekazywane informacje stają się bardziej zrozumiałe.

Wystarczy umieścić słowa bezpośrednio na rysunku (lub w jego okolicy), a nie na następnej stronie, a dwukrotnie zwiększy się prawdopodobieństwo, że uczące się osoby będą w stanie rozwiązać problem, którego te słowa dotyczą.

Stosuj dialogi i personifikuj. Według najnowszych badań studenci uzyskiwali w testach o 40% lepsze wyniki, jeśli treści były przekazywane w sposób bezpośredni, w pierwszej osobie, z elementami dialogu, a nie w sposób formalny. wykładów lepsze są historyjki. Używaj języka potocznego. Nie traktuj się zbyt poważnie. Kiedy jesteś bardziej zainteresowany tematem: podczas ożywionej dyskusji przy obiedzie czy na wykładzie?



Zachęć czytelników do głębszych przemyśleń. Chodzi o to, że jeśli nie zmusisz neuronów do większej aktywności, w Twojej głowie nie zdarzy się nic ciekawego. Czytelnik musi być zmotywowany, zaangażowany, zaciekawiony i podekscytowany rozwiązywaniem problemów, wyciąganiem wniosków i zdobywaniem nowej wiedzy. A osiągnięcie tego wszystkiego jest możliwe dzięki wyzwaniom, ćwiczeniom, prowokującym pytaniom oraz innym aktywnościom angażującym obie półkule Twojego mózgu i jak najwięcej zmysłów.

Musisz zdobyć uwagę czytelnika i zachować ją na dłużej. Każdy z nas kiedyś znalazł się w sytuacji, gdy koniecznie chciał się czegoś nauczyć, ale nie mógł przebrnąć przez pierwszą stronę. Mózg zwraca uwagę na rzeczy niezwykłe, interesujące, dziwne, przykuwające wzrok, nieoczekiwane. Poznawanie nowych zagadnień technicznych wcale nie musi być nudne. I jeżeli takie nie jest, Twój mózg przyswoi je znacznie szybciej.

Wyzwól emocje. Jak już wiemy, zdolność zapamiętywania informacji jest w dużej mierze zależna od ładunku emocjonalnego. Zapamiętujemy to, na czym nam zależy. Zapamiętujemy, gdy coś *czujemy*. Oczywiście nie chodzi nam tu o łzawe historie o chłopcu i jego ukochanym piesku. Mówimy o emocjach takich jak zaskoczenie, zaciekawienie, podekscytowanie, „a niech to!", a także uczucie satysfakcji — „jestem wielki!" — którego doświadczamy po rozwiązaniu zagadki, zrozumieniu czegoś, co uważaliśmy za trudne, czy zdaniu sobie sprawy, że wiemy coś, o czym *nie ma pojęcia* Maciek z sąsiedniego działu.

Metapoznanie — myślenie o myśleniu

Kiedy naprawdę chcesz się czegoś nauczyć i chcesz to zrobić szybciej i bardziej dogłębnie, zwróć uwagę na to, jak zwracasz swoją uwagę. Pomyśl o tym, jak myślisz. Dowiaduj się, jak zdobywasz wiedzę.

Większość z nas nie uczestniczyła nigdy w kursach metapoznania ani teorii nauczania. Od dzieciństwa *oczekiwano* od nas, że będziemy się uczyć, ale rzadko mówiono, *jak* mamy to robić.

Zakładamy jednak, że skoro trzymasz w ręku tę książkę, naprawdę chcesz się nauczyć tworzenia mobilnych witryn i aplikacji internetowych. I zapewne nie chcesz na to poświęcić zbyt wiele czasu. A ponieważ w przyszłości będziesz najprawdopodobniej tworzył więcej aplikacji, koniecznie musisz *zapamiętać*, co przeczytałeś. Warunkiem zapamiętania jest jednak *zrozumienie*. Aby w jak największym stopniu skorzystać z tej książki, ale też z *każdej* innej, jak i innych form uczenia się, koniecznie musisz wziąć odpowiedzialność za swój mózg. Myśl *o tym*, czego się uczysz.

Sztuczka polega na tym, aby przekonać mózg, że poznawany materiał jest *naprawdę ważny*. Kluczowy dla Twojego samopoczucia. Nie mniej ważny od tygrysa. W przeciwnym przypadku będziesz nieustannie prowadził wojnę z własnym mózgiem, który za wszelką cenę będzie się starał unikać utrwalania nowej wiedzy.

Jak w takim razie zmusić mózg do traktowania witryn i aplikacji mobilnych jak głodnego tygrysa?

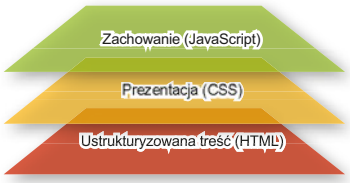
Są dwie metody: jedna powolna i męcząca, a druga szybka i skuteczna. Pierwsza polega na wielokrotnym powtarzaniu. Oczywiście wiesz, że *jesteś* w stanie zapamiętać nawet najnudniejsze zagadnienie, mozolnie je wkuwając. Po odpowiedniej liczbie prób Twój mózg stwierdzi: „Nie *wydaje* mi się, żeby to było dla niego szczególnie istotne, ale skoro tak *w kółko* to powtarza, to może jednak coś w tym jest”.

Szybsza metoda polega na robieniu ***czegokolwiek, co zwiększa aktywność mózgu***, a najlepiej czegoś, co wyzwala różne *typy* aktywności. Kluczowe są zasady, które opisaliśmy na poprzedniej stronie. Udowodniono, że wszystkie potrafią pomóc w nakłonieniu mózgu, by pracował na Twoją korzyść. Badania wykazują, że na przykład umieszczenie opisów *na* rysunkach (a nie w innych miejscach na stronie, na przykład w nagłówku lub wewnątrz akapitu) sprawia, że mózg stara się zrozumieć relację pomiędzy słowami a rysunkiem, co z kolei może rozgrzać nasze neurony do czerwoności. Większa aktywność neuronów to zaś większa szansa, że mózg uzna informacje za warte uwagi i, ewentualnie, zapamiętania.

Prezentowanie informacji w formie dialogu pomaga, ponieważ czytelnicy zwykle wykazują większe zainteresowanie w sytuacjach, gdy mają wrażenie uczestniczenia w dyskusji, ponieważ czują, że oczekuje się od nich śledzenia jej przebiegu i brania w niej czynnego udziału. Zadziwiające jest to, że mózg niekoniecznie *zwraca uwagę* na to, że rozmowa jest prowadzona między Tobą a książką! Z kolei jeśli informacje są przekazywane w sposób formalny i suchy, mózg traktuje je tak samo jak uczestniczenie w wykładzie na sali pełnej sennych słuchaczy. Nie ma potrzeby wykazywania jakiejkolwiek aktywności.

Ale rysunki i dialogi to tylko początek.

Oto, co MY zrobiliśmy



Wprowadziliśmy mnóstwo ***rysunków***, ponieważ Twój mózg zwraca większą uwagę na obrazy niż na tekst. Z punktu widzenia mózgu faktycznie jeden obraz *jest* wart więcej niż tysiąc słów. Wszędzie tam, gdzie tekstom towarzyszą rysunki, umieściliśmy tekst *na* rysunkach, ponieważ mózg działa efektywniej, gdy tekst jest wewnątrz tego, co opisuje, niż kiedy znajduje się w innym miejscu i stanowi część większego fragmentu.

Stosowaliśmy ***powtórzenia***, wielokrotnie podając tę samą informację na *różne* sposoby i przy wykorzystaniu różnych środków przekazu oraz odwołując się do *różnych zmysłów*. Wszystko po to, by zwiększyć szansę, że informacja zostanie zakodowana w większej liczbie obszarów mózgu.

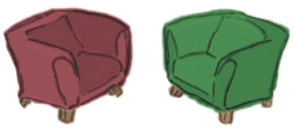
Jazda próbna

Korzystaliśmy z pomysłów i rysunków w ***nieoczekiwany*** sposób, ponieważ Twój mózg liczy na nowości i ich pragnie. Poza tym staraliśmy się zawrzeć w nich *chociaż odrobinę* ***emocji***, gdyż mózg jest skonstruowany w taki sposób, że zwraca uwagę na biochemię związaną z emocjami. Prawdopodobieństwo zapamiętania czegoś jest większe, jeśli „to coś" sprawia, że cokolwiek *poczujemy*, nawet jeśli to uczucie nie jest niczym więcej niż lekkim ***rozbawieniem***, ***zaskoczeniem*** lub ***zainteresowaniem***.

Używaliśmy bezpośrednich zwrotów i przekazywaliśmy treści ***w formie dialogu***, ponieważ mózg zwraca większą uwagę, jeśli myśli, że prowadzisz rozmowę, niż gdy jesteś jedynie biernym słuchaczem prezentacji. Mózg działa w ten sposób, nawet gdy *czytasz* rozmowę.

Zamieściliśmy w książce mnóstwo ***ćwiczeń***, ponieważ mózg uczy się i zapamiętuje zdecydowanie lepiej to, co ***robimy***, niż to, o czym *czytamy*. Poza tym podane ćwiczenia stanowią wyzwania, choć nie są przesadnie trudne, gdyż właśnie takie preferuje większość osób.

Pogawędki przy kominku



Zastosowaliśmy ***wiele stylów nauczania***, ponieważ tak jak *Ty* możesz preferować instrukcje opisujące sposób postępowania krok po kroku, tak ktoś inny woli ogólną analizę danego zagadnienia, a ktoś jeszcze inny przejrzenie przykładowego fragmentu kodu. Jednak niezależnie od ulubionego sposobu nauki *każdy* skorzysta na tym, że te same informacje będą przedstawiane kilkakrotnie w różny sposób.

Podaliśmy informacje przeznaczone dla ***obu półkul Twojego mózgu***, ponieważ im bardziej mózg będzie zaangażowany, tym większe jest prawdopodobieństwo nauczenia się i zapamiętania podanych informacji i tym dłużej możesz koncentrować się na nauce. Ponieważ angażowanie tylko jednej półkuli często oznacza, że druga może odpoczywać, będziesz mógł się uczyć bardziej produktywnie przez dłuższy czas.

Dołączyliśmy też ***historyjki*** i ćwiczenia prezentujące ***więcej niż jeden punkt widzenia***, ponieważ mózg uczy się dokładniej, gdy jest zmuszony do przetwarzania i podawania własnej opinii.

Postawiliśmy przed Tobą ***wyzwania*** — zarówno poprzez włączenie ćwiczeń, jak i stawiając ***pytania***, na które nie zawsze można odpowiedzieć w prosty sposób; a to dlatego, że mózg uczy się i pamięta, gdy musi *popracować* nad czymś (podobnie jak nie możemy zdobyć dobrej *formy fizycznej*, *obserwując* ćwiczenia w telewizji). Jednak dołożyliśmy wszelkich starań, aby zapewnić, że gdy pracujesz, robisz *dokładnie to, czego trzeba*, aby ***ani jeden dendryt nie musiał*** przetwarzać trudnego przykładu bądź analizować tekstu zbyt lapidarnego lub napisanego trudnym żargonem.

Personifikowaliśmy tekst. W historyjkach, przykładach, rysunkach i wszelkich innych możliwych miejscach tekstu staraliśmy się personifikować tekst, gdyż ***jesteś*** osobą, a Twój mózg zwraca większą uwagę na *osoby* niż na *rzeczy*.

Wytnij te porady i przyklej na lodówce.

Oto, co TY możesz zrobić, aby zmusić swój mózg do posłuszeństwa

A zatem zrobiliśmy, co w naszej mocy. Reszta zależy od Ciebie. Możesz zacząć od poniższych porad. Posłuchaj swojego mózgu i określ, które z nich sprawdzają się w Twoim przypadku, a które nie przynoszą pozytywnych rezultatów. Spróbuj czegoś nowego.

1 Zwolnij. Im więcej rozumiesz, tym mniej musisz zapamiętać.

Nie ograniczaj się jedynie do *czytania*. Przerwij na chwilę lekturę i pomyśl. Kiedy znajdziesz w tekście pytanie, nie zaglądaj od razu na stronę odpowiedzi. Wyobraź sobie, że ktoś *faktycznie* zadaje Ci pytanie. Im bardziej zmusisz swój mózg do myślenia, tym większa będzie szansa, że się nauczysz i zapamiętasz dane zagadnienie.

2 Wykonuj ćwiczenia. Rób notatki.

Umieszczaliśmy je w tekście, jednak jeśli zrobilibyśmy je za Ciebie, to niczym nie różniłoby się to od sytuacji, w której ktoś za Ciebie wykonywałby ćwiczenia fizyczne. I nie ograniczaj się jedynie do *czytania* ćwiczeń. **Używaj ołówka.** Można znaleźć wiele dowodów na to, że fizyczna aktywność *podczas* nauki może poprawić jej wyniki.

3 Czytaj fragmenty oznaczone jako „Nie istnieją głupie pytania".

Nie należy ich traktować jak fragmenty opcjonalne — ***stanowią część podstawowej zawartości książki!*** Nie pomijaj ich.

4 Niech lektura tej książki będzie ostatnią rzeczą, jaką robisz przed pójściem spać. A przynajmniej ostatnią rzeczą stanowiącą wyzwanie intelektualne.

Pewne elementy procesu uczenia się (a w szczególności przenoszenie informacji do pamięci długotrwałej) mają miejsce *po* odłożeniu książki. Twój mózg potrzebuje trochę czasu dla siebie i musi dodatkowo przetworzyć dostarczone informacje. Jeśli w czasie niezbędnym do tego dodatkowego „przetwarzania" zmusisz go do innej działalności, część z przyswojonych informacji może zostać utracona.

5 Pij wodę. Dużo wody.

Twój mózg pracuje najlepiej, gdy dostarczasz mu dużo płynów. Odwodnienie (które może następować, nawet zanim poczujesz pragnienie) obniża zdolność percepcji.

6 Mów o zdobywanych informacjach. Na głos.

Mówienie aktywuje odmienne fragmenty mózgu. Jeśli próbujesz coś zrozumieć lub zwiększyć szanse zapamiętania informacji na dłużej, powtarzaj je na głos. Jeszcze lepiej, gdy będziesz się starał je komuś wytłumaczyć. W ten sposób nie tylko będziesz się uczył szybciej, ale też przy odrobinie szczęścia odkryjesz to, czego nie dostrzegłeś, czytając treść książki.

7 Słuchaj swojego mózgu.

Uważaj, kiedy Twój mózg staje się przeciążony. Jeśli zauważysz, że zaczynasz czytać pobieżnie i zapominać to, o czym przeczytałeś przed chwilą, najwyższy czas na przerwę. Po przekroczeniu pewnego punktu nie będziesz się uczył szybciej, „wciskając" do głowy więcej informacji. Co gorsza, może to zaszkodzić całemu procesowi przyswajania wiedzy.

8 Poczuj coś.

Twój mózg musi wiedzieć, że to, czego się uczysz, ma znaczenie. Z zaangażowaniem śledź zamieszczone w tekście historyjki. Nadawaj własne tytuły zdjęciom. Zalewanie się łzami ze śmiechu po przeczytaniu głupiego dowcipu i tak jest lepsze od braku jakiejkolwiek reakcji.

9 Stwórz coś.

Wykorzystaj w codziennej pracy to, czego się nauczyłeś. Niech nowa wiedza ułatwia Ci podejmowanie decyzji projektowych. Rób więcej, niż od Ciebie oczekujemy w ćwiczeniach z tej książki. Wszystko, czego potrzebujesz, to ołówek i problem do rozwiązania... problem, który może przynieść korzyści dzięki wykorzystaniu narzędzi i technik potrzebnych Ci w codziennej pracy.

Kilka rzeczy, o których musisz wiedzieć

To książka do nauki, a nie encyklopedia. Celowo usunęliśmy wszystko, co mogłoby Ci przeszkadzać w nauce, niezależnie od tego, nad czym pracujesz w danym miejscu książki. Podczas pierwszej lektury książki należy zacząć od jej początku, gdyż kolejne rozdziały bazują na tym, co wiedziałeś i czego się dowiedziałeś wcześniej.

Zakładamy, że znasz języki HTML i CSS.

Jeśli nie znasz HTML-a ani CSS, najpierw sięgnij po książkę *Head First HTML with CSS & XHTML. Edycja polska*. A jeżeli te tematy nie są Ci obce, śmiało możesz rozpoczynać lekturę tej książki.

Zakładamy, że wiesz co nieco na temat języków skryptowych.

Nie oczekujemy, że jesteś mistrzem świata w JavaScripcie ani że zjadłeś zęby na projektach tworzonych w PHP, ale podstawowa wiedza na temat tych języków się przyda. Jeśli na widok najzwyklejszej instrukcji `for` dostajesz ataku paniki (albo nie masz pojęcia, o czym w ogóle tu mówimy), powinieneś zajrzeć najpierw do *Head First PHP & MySQL. Edycja polska* albo *Head First JavaScript. Edycja polska*, a później wrócić do lektury tej książki.

Spodziewamy się, że umiesz zdobywać nowe informacje i szukać rozwiązań problemów.

Będziemy szczerzy — technologie mobilne to niezwykle szeroki temat, który wymaga ciągłego *rozwijania* swoich umiejętności. Informacji jest tak dużo, że ich przyswojenie przekracza możliwości jednej osoby (wystarczy wspomnieć o szczegółach składni JavaScriptu czy wsparciu dla poszczególnych atrybutów elementów HTML5 w różnych przeglądarkach). Nie wymagaj od siebie zbyt wiele. Częścią pracy projektantów i programistów jest wyszukiwanie potrzebnych informacji i rozwiązań napotkanych problemów. Pamiętaj, że Google jest Twoim przyjacielem.

Liczymy na to, że się rozwiniesz.

Świat mobilnych technologii jest przeogromny. Mamy nadzieję, że dzięki tej książce zaczniesz go lepiej poznawać, ale to, jak daleko zajdziesz, zależy tylko od Ciebie. Szukaj aktywnych społeczności, czytaj blogi, dołączaj do list mailingowych oraz udzielaj się na warsztatach i konferencjach.

Ćwiczenia SĄ obowiązkowe.

Ćwiczenia oraz wszelkie dodatkowe polecenia nie są jedynie dodatkami — stanowią integralną część podstawowej treści książki. Niektóre z nich zostały umieszczone po to, by pomóc w zapamiętaniu informacji, inne, by pomóc w zrozumieniu opisywanego materiału, a jeszcze inne — by pomóc Ci w praktycznym zastosowaniu zdobytej wiedzy. ***Nie pomijaj żadnego ćwiczenia.*** Dzięki nim masz szansę przemyśleć to, czego się właśnie dowiedziałeś, oraz spojrzeć na przedstawione zagadnienia w innym kontekście.

Powtórzenia są celowe i ważne.

Jedną z cech, która wyróżnia serię książek *Rusz głową!*, jest to, iż naprawdę *bardzo, ale to bardzo* zależy nam na tym, abyś wszystko zrozumiał i przyswoił. Chcielibyśmy także, abyś zakończył lekturę tej książki, pamiętając zawarte w niej informacje. Autorzy większości książek informacyjnych i leksykonów nie stawiają sobie za cel przyswojenia i zapamiętania przez Ciebie prezentowanej treści; w tej książce jest inaczej, stąd wiele pojęć będzie się pojawiało kilka razy.

Ćwiczenia „Wysil szare komórki” nie mają odpowiedzi.

Na część postawionych w tych ćwiczeniach pytań nie ma dobrej odpowiedzi, na część sam sobie musisz odpowiedzieć i sprawdzić, czy i w jakich sytuacjach ta odpowiedź jest prawidłowa. W niektórych ćwiczeniach znajdziesz wskazówki, które pomogą Ci ukierunkować poszukiwania odpowiedzi.

Aby tworzyć witryny i aplikacje mobilne, potrzebujesz prostego edytora tekstu, przeglądarki, serwera WWW (który możesz zainstalować na swoim komputerze) oraz kodu źródłowego przykładów omawianych w książce.

Dla systemu Windows polecamy edytory tekstu takie jak: PSPad, TextPad czy EditPlus (jeżeli koniecznie chcesz, możesz używać systemowego notatnika). Dla systemu Mac OS proponujemy TextWrangler (albo jego starszego brata — BBEdit) lub TextMate. Warty polecenia jest też edytor Coda, który jest w większym stopniu nakierowany na pracę nad projektami webowymi.

Jeżeli pracujesz w Linuksie, masz do wyboru mnóstwo edytorów i wierzymy, że nie musimy Ci o nich opowiadać.

Skoro zamierzasz pracować nad projektami internetowymi, potrzebny Ci będzie serwer WWW. W dodatku B znajdziesz instrukcję instalacji serwera z obsługą PHP. Proponujemy, abyś zainstalował go jak najszybciej. Naprawdę — od razu przejdź do dodatku, postępuj zgodnie z instrukcjami, a następnie wróć do czytania tego wprowadzenia.

Na potrzeby ćwiczeń z rozdziału 5. będziesz musiał zainstalować API i dane WURFL (ang. *Wireless Universal Resource FiLe*). Z kolei w rozdziale 8. korzystamy z SDK Androida i kilku związanych z nim narzędzi. Zgadłeś — w dodatkach znajdziesz wszystkie niezbędne instrukcje.

Przez cały czas pracy z książką będziesz też potrzebował przeglądarki internetowej, a ściślej, ***tak wielu przeglądarek, jak to tylko możliwe***. Poza tym im więcej ***urządzeń mobilnych z przeglądarkami*** masz do dyspozycji, tym lepiej (spokojnie — jeśli nie masz sprzętu, możesz skorzystać z wielu dostępnych emulatorów).

Podczas tworzenia i testowania witryn oraz aplikacji internetowych na komputerze stacjonarnym używaj przeglądarki Google Chrome, która jest dostępna w systemach Mac, Windows i Linux. Warto poświęcić chwilę na bliższe zapoznanie się z konsolą JavaScriptu będącą częścią narzędzi programistycznych w tej przeglądarce. Potraktuj to jako pracę domową.

I na koniec sprawa kodu źródłowego i zasobów przykładów omawianych w książce. Możesz je pobrać ze strony ***ftp://ftp.helion.pl/przyklady/mobweb.zip***.

Kod źródłowy przykładów i wszystkie potrzebne zasoby możesz pobrać z ftp://ftp.helion.pl/przyklady/mobweb.zip.

Zajrzyj też na stronę oryginalnego wydania tej książki — http://hf-mw.com.

Zespół korektorów merytorycznych

Trevor Farlow w wolnych chwilach gra rekreacyjnie w piłkę nożną, jest piekarzem i opiekunem w schronisku dla zwierząt. Kiedy nie zajmuje go wyprowadzanie psów, strzelanie bramek i doskonalenie sernika po nowojorsku, uczy się sztuki zarządzania produktem w dynamicznym i prężnym zespole programistycznym w Clearwater Analytics, LLC.

Brad Frost jest strategiem do spraw technologii mobilnych i czołowym programistą w R/GA w Nowym Jorku, gdzie współpracuje z wielkimi markami przy projektach związanych z urządzeniami mobilnymi. Prowadzi stronę gromadzącą zasoby o nazwie Mobile Web Best Practices (*http://mobilewebbestpractices.com*), której misją jest pomoc ludziom w tworzeniu doskonałych doświadczeń mobilnych.

Stephen Hay zajmuje się tworzeniem stron internetowych od ponad 16 lat. Oprócz pracy z klientami, która w coraz większym stopniu skupia się na projektach i programowaniu wieloplatformowym, wypowiada się na spotkaniach branżowych i pisze dla wydawnictw takich jak „A List Apart" czy „.net Magazine". Jest też współorganizatorem Mobilism, poważanej konferencji na temat projektowania i programowania dla urządzeń mobilnych.

Ethan Marcotte to niezależny projektant i programista, którego pasją jest piękny projekt, elegancki kod i to, co powstaje na styku tych dwóch ideałów. Na przestrzeni lat wśród jego klientów znaleźli się m.in. „New York Magazine", Sundance Film Festival, „Boston Globe" i W3C. Ethan ukuł określenie Responsive Web Design, które oznacza nowy styl projektowania nastawiony na wciąż zmieniającą się sieć i, jeśli tylko mu na to pozwolić, będzie godzinami o nim opowiadał — posunął się w tym tak daleko, że napisał na ten temat książkę.

Bryan Rieger jest projektantem i nieśmiałym programistą wywodzącym się ze środowiska projektantów teatralnych i twórców klasycznych animacji. Bryan pracował przy wielu mediach — prasie, radiu, sieci i mediach mobilnych — i z wieloma klientami, m.in. Apple, Microsoft, Nokia i Fundacją Symbian. Jest niestrudzonym gawędziarzem i zapalonym majsterkowiczem; jest odpowiedzialny za stworzenie szerokiej gamy projektów dla Yiibu — niewielkiej firmy z Edynburga zajmującej się doradztwem projektowym.

Stephanie Rieger jest projektantką, pisarką i utajoną antropolożką, której pasją są interakcje pomiędzy człowiekiem a technologią. Stephanie projektuje treści dla urządzeń mobilnych od 2004 roku, a obecnie skupia się głównie na strategii sieciowej, dużych projektach i optymalizacji pod kątem wielu wyświetlaczy i sprzętów. Jest kompulsywną testerką i badaczką; jest zawsze chętna, by badać świat urządzeń mobilnych i dzielić się swoimi odkryciami na temat zachowań użytkowników i międzynarodowych trendów w technologiach mobilnych.

Andrea Trasatti zaczął tworzyć treści w środowisku WAP w 1999 roku na Nokii 7110, która w owym czasie była uznawana w Europie za szczyt technologii. Andrea od samego początku kierował projektami WURFL i DeviceAtlas i od lat gromadzi doświadczenia w dziedzinie wykrywania urządzeń i dopasowywania się treści do wykrytych zmian sprzętowych. Andrea pisuje na Twitterze jako *@AndreaTrasatti*, gdzie regularnie wypowiada się na temat sieci mobilnej i nowych trendów w tworzeniu treści dla urządzeń mobilnych i zarządzaniu nią.

Podziękowania

Courtney Nash

Nasza redaktor

Podziękowania (i gratulacje!) dla **Courtney Nash**, która skłoniła nas do napisania książki tak dobrej, jak tylko byliśmy w stanie. Dzielnie zniosła potężną powódź maili, pytań, narzekań i okazjonalnych epizodów dziwactwa. Cały czas stała po naszej stronie i przekonała nas, abyśmy zawierzyli naszym instynktom. Dzięki też dla **Briana Sawyera** za wkroczenie pod koniec i przeprowadzenie nas przez metę.

Zespół O'Reilly

Podziękowania dla **Lou Barr** za jej niezwykłą szybkość oraz mistrzostwo w projektowaniu i zarządzaniu układem graficznym. Jesteśmy pod wielkim wrażeniem. Dziękujemy. Na naszą wdzięczność zasłużyły też **Karen Shaner** i **Rachel Monaghan** ze względu na ich pomoc przy żonglowaniu wersjami roboczymi, komentarzami i wszystkimi ważnymi drobiazgami.

Lou Barr

Dziękujemy też reszcie ludzi z O'Reilly, dzięki którym czuliśmy się tak mile widziani: **Mike Hendrickson** jako pierwszy wpadł na ten szalony pomysł; **Brady Forrest** przedstawił nas i wypromował; **Tim O'Reilly** przez cały czas był takim właśnie szczerym, bystrym i fajnym facetem, jakim jest; a **Sara Winge** zasługuje na wdzięczność za jej hojność i ogólną cudowność.

Nasze podziękowania

Jason i Lyza pracują w Cloud Four z najmądrzejszymi ludźmi na świecie. Nasze nieskończone podziękowania należą się współzałożycielom, którymi są **Aileen Jeffries** i **John Keith**; na naszą wdzięczność zasługuje też reszta zespołu Cloud Four: **Matt Gifford**, **Chris Higgins** i **Megan Notarte**. Ta książka to tak naprawdę owoc naszej wspólnej obsesji na temat mobilnego internetu i to właśnie oni bardziej niż ktokolwiek inny promowali i wzniecali ten wysiłek. Dzięki wam po stokroć, tysiąckroć i zylionokroć, kochani.

Chcielibyśmy również podziękować całej społeczności zgromadzonej wokół mobilnej sieci. W szczególności są to: Josh Clark, Gail Rahn Frederick, Scott Jehl, Scott Jenson, Dave Johnson, Tim Kadlec, Jeremy Keith, Peter-Paul Koch, Bryan LeRoux, James Pearce, Steve Souders i Luke Wroblewski. Jesteśmy dumni, że możemy być częścią takiej społeczności.

Lyza dziękuje przyjaciołom i rodzinie

Dzięki dla **Bryana Christophera Foxa** (Other Dev), bez którego talentu programistycznego, wglądu, wsparcia i ogólnego supergeniuszu napisanie tej książki nie byłoby możliwe.

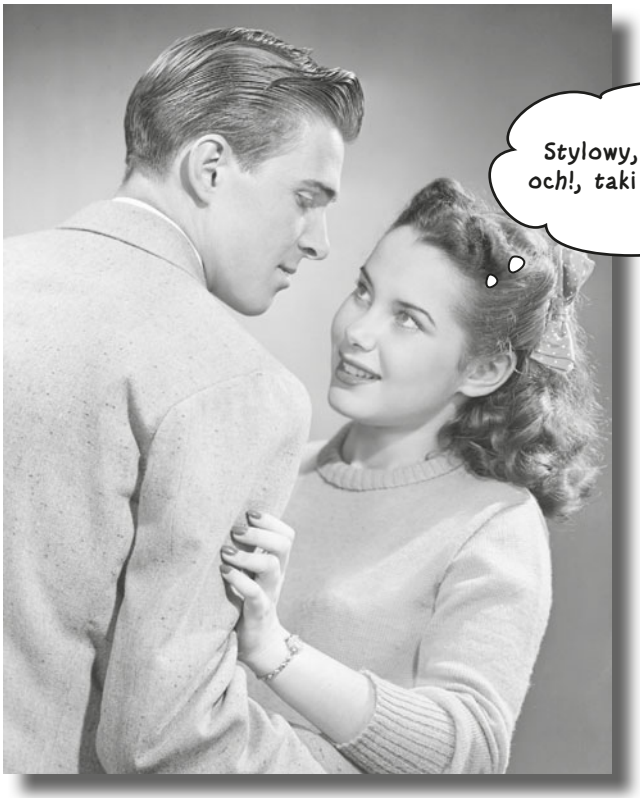
Potężne podziękowania dla moich **przyjaciół** i **rodziny**, którzy jakoś wciąż mnie znoszą mimo mojego długotrwałego pobytu w Krainie Książek. Dzięki dla Autumn i Amye, które wykazały wyjątkową wytrwałość w obliczu mojej ciągłej niedostępności. Mike — dzięki, jak zawsze. Dziękuję również Tacie, który zawsze pokazuje mi, jak odnaleźć piękno i nowe wyzwania. W końcu wielkie podziękowania dla Huw i Bethan z Plas-yn-Iâl w Llandegla (Walia), fantastycznego miejsca pełnego owiec i radości życia, gdzie zostało napisane około ćwierć tej książki.

Jason dziękuje przyjaciołom i rodzinie

Dziękuję całej mojej rodzinie za wsparcie. Nasi rodzice, **Jan**, **Carol**, **Mark** i **Doanne**, niezwykle pomogli nam w utrzymaniu zdrowia psychicznego, gdy próbowaliśmy połączyć pisanie książki, życie rodzinne i przeprowadzkę.

Szczególne podziękowania dla mojej żony, **Dany Grigsby**, za umożliwienie mi pisania książki w czasie, gdy wychowywaliśmy dzieci w wieku niemowlęcym i przedszkolnym i przeprowadzaliśmy się do nowego domu. Bez Ciebie nie byłoby to możliwe.

Wrażliwe projekty, czyli Responsive Web Design



Witajcie! Jesteście gotowi na mobilne technologie webowe? Tworzenie witryn na urządzenia mobilne jest naprawdę ekscytujące. Wiele w tym uroku, emocji i momentów, w których chciałoby się wykrzyknąć: ***Eureka!***. Ale z drugiej strony pełno tu tajemnic i trudności. Technologie mobilne rozwijają się w tak niewiarygodnym tempie, że cały czas jesteśmy trochę w tyle. Trzymaj się więc mocno! Naszą przygodę rozpoczynamy od ciekawego podejścia do tworzenia witryn internetowych, znanego jako ***Responsive Web Design (RWD)***. Dzięki niemu będziesz mógł sprawić, by strony wyglądały równie dobrze na wielu różnych urządzeniach mobilnych i, co ważne, przydadzą Ci się umiejętności, które już masz.

Wszyscy jedziemy na tym samym wózku. Wskakujesz?

Jest bardziej niż pewne, że masz telefon komórkowy. I wiemy to nie dlatego, że kupiłeś tę książkę (przy okazji — bardzo dobry pomysł!), ale dlatego, że trudno znaleźć kogokolwiek, kto by go nie miał.

Zupełnie nieistotne jest to, gdzie mieszkasz, bo telefony komórkowe używane są wszędzie. Rolnicy z Nigerii sprawdzają za ich pomocą najlepsze ceny skupu manioku, a połowa z dziesiątki najlepiej sprzedających się powieści w Japonii została przeczytana i napisana — *tak, napisana* — na telefonach komórkowych.

Na początku 2011 roku na wszystkich mieszkańców ziemi, czyli 7 miliardów ludzi, przypadało ponad 5,2 miliarda telefonów komórkowych. **To oznacza, że więcej ludzi ma dostęp do telefonów niż do toalety czy szczoteczki do zębów.**

Ten moment już nadszedł

Jak widzisz, mobilny świat jest ogromny, ale tak jest już od jakiegoś czasu. Dlaczego właśnie teraz postanowiłeś się tym zainteresować?

Ponieważ **iPhone wszystko zmienił**. Brzmi to może jak frazes, ale to prawda. Oczywiście przed iPhone'em były już sklepy z aplikacjami, ekrany dotykowe i przeglądarki internetowe na telefonach, ale firma Apple jako pierwsza połączyła to wszystko w taki sposób, że korzystanie z tego nie sprawia przeciętnemu użytkownikowi żadnego problemu.

iPhone jest fantastyczny, ale ludzie używają wielu innych telefonów z wielu różnych powodów. W dodatku utrzymanie pierwszego miejsca w tym wyścigu nie jest takie proste.

Nie umiemy przewidzieć, jakie telefony będą popularne w chwili, gdy będziesz czytał tę książkę. Trzy lata temu Android był zaledwie małym punktem na radarze, a w 2011 roku stał się wiodącą platformą dla smartfonów.

Co prawda technologie mobilne zmieniają się bardzo szybko, ale kilku spraw możemy być pewni:

1. **Każdy nowy telefon jest wyposażony w przeglądarkę internetową.**
   Być może udałoby Ci się znaleźć telefon bez przeglądarki, ale musiałbyś się sporo naszukać. Nawet najprostsze telefony mają na pokładzie przeglądarki. Przecież wszyscy chcą mieć na swoich telefonach dostęp do internetu.

2. **Popularność mobilnego internetu przebije dostęp za pośrednictwem komputerów stacjonarnych.**
   Niedługo więcej użytkowników będzie korzystało z sieci za pomocą telefonów komórkowych niż za pośrednictwem klasycznych komputerów. Już teraz wiele osób częściej korzysta z telefonu niż z peceta.

3. **Jedynie internet dostarcza prawdziwie międzyplatformowe technologie.**
   Android, iPhone, BlackBerry, Windows Phone, WebOS, Symbian, Bada — to tylko niektóre z platform dla telefonów, którymi możemy się zainteresować. Każda z nich ma własne środowisko programistyczne, co oznacza, że jeżeli chcielibyśmy stworzyć program dla każdego z nich, musielibyśmy za każdym razem zaczynać od zera.

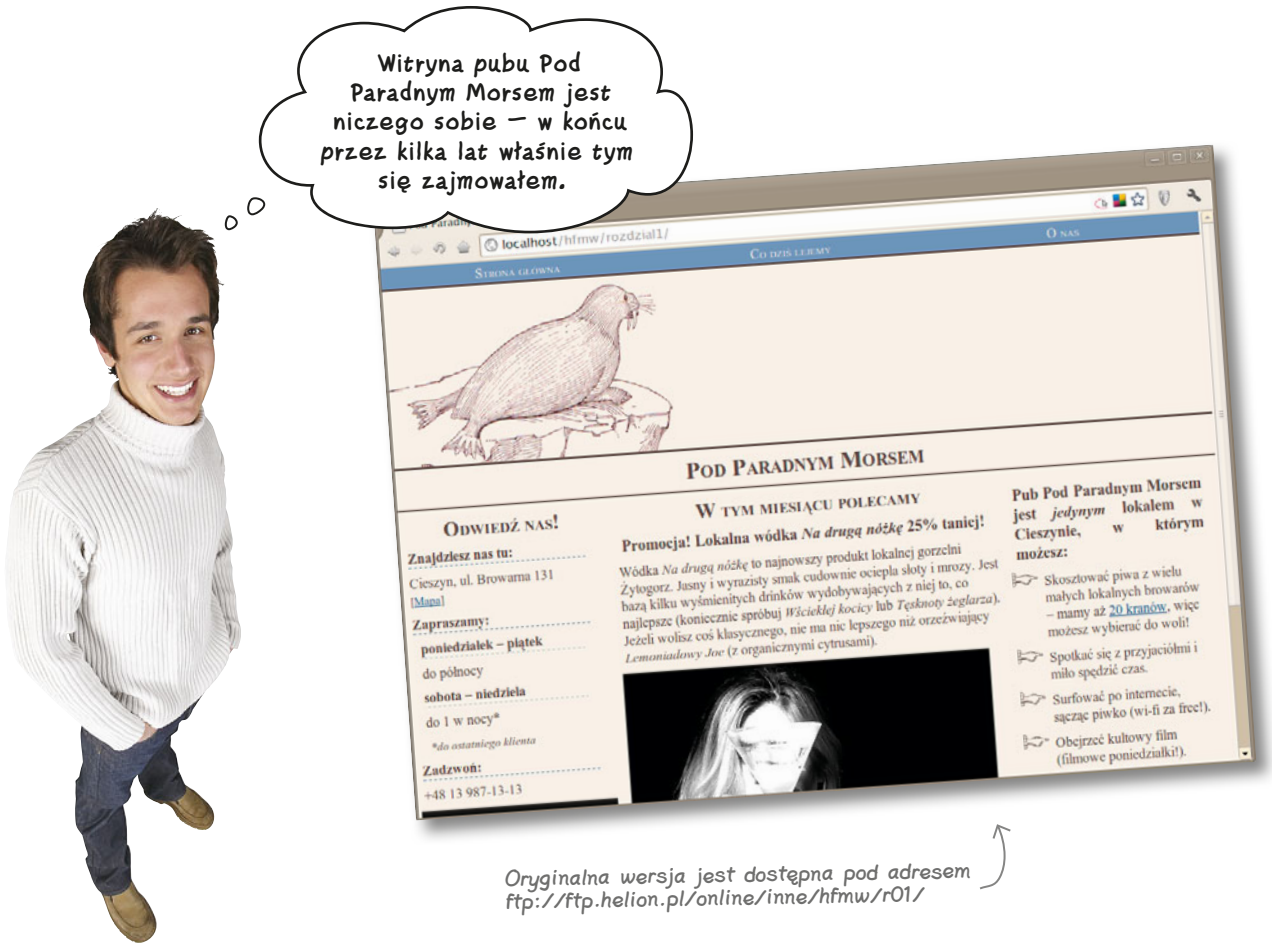
   Mobilne technologie webowe stawiają przed nami co prawda nowe wyzwania, ale nie ma innego rozwiązania, które umożliwiałoby tworzenie treści i aplikacji osiągalnych równocześnie na wszystkich platformach.

Zatem jesteś we właściwym miejscu we właściwym czasie. Mobilne technologie właśnie startują, więc nie czekaj i wskakuj na pokład. Zaczynamy!

Coś niedobrego stało się w drodze do pubu

Michał jest właścicielem pubu Pod Paradnym Morsem — miejsca o intrygującej nazwie i całkiem sporej rzeszy stałych klientów będących smakoszami tutejszego piwa. Michał zawsze stara się lać ciekawe piwa, a niektóre z nich promuje na swojej stronie internetowej.

Zanim zrealizowały się jego marzenia i został właścicielem pubu, Michał tworzył strony internetowe. Nie miał więc problemu z przygotowaniem witryny dla swojego pubu Pod Paradnym Morsem.



Oryginalna wersja jest dostępna pod adresem ftp://ftp.helion.pl/online/inne/hfmw/r01/

Skoro przeglądarki w telefonach komórkowych są takie świetne...

Michał stworzył witrynę pubu Pod Paradnym Morsem kilka lat temu, kiedy przeglądanie stron na urządzeniach mobilnych nie było powszechne. Michał robił ją z myślą o przeglądarkach desktopowych — przetestował ją zresztą w Firefoksie, Internet Explorerze i Safari.

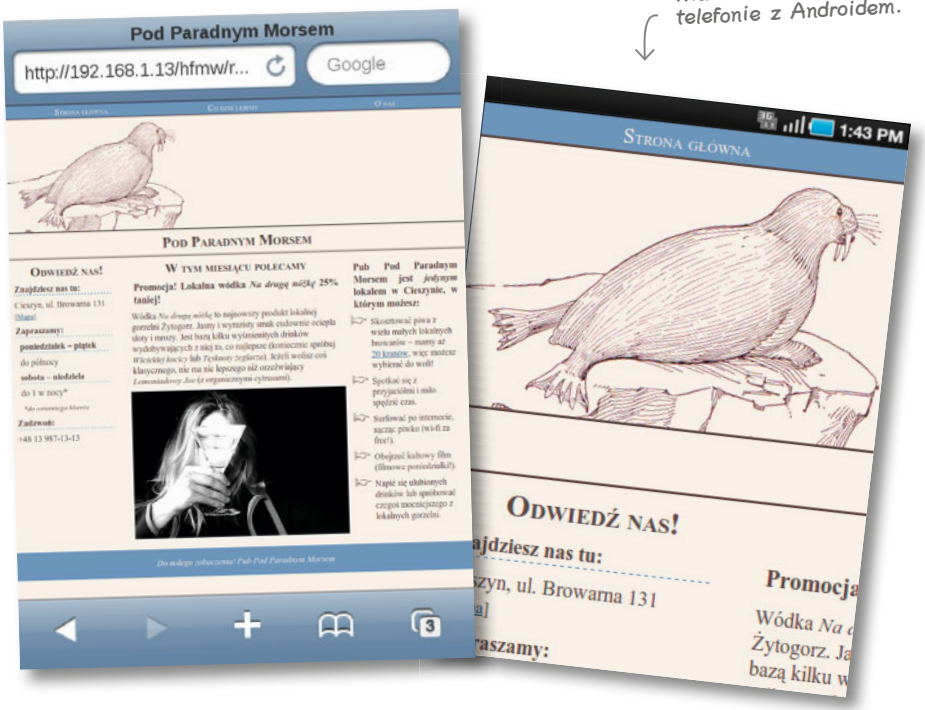
Wiele nowszych przeglądarek mobilnych jest ocenianych dobrze. Są coraz bardziej zaawansowane i wydajne, więc korzystanie z nich w coraz większym stopniu jest zbliżone do ich desktopowych odpowiedników.

...to czy nie powinno to po prostu działać?

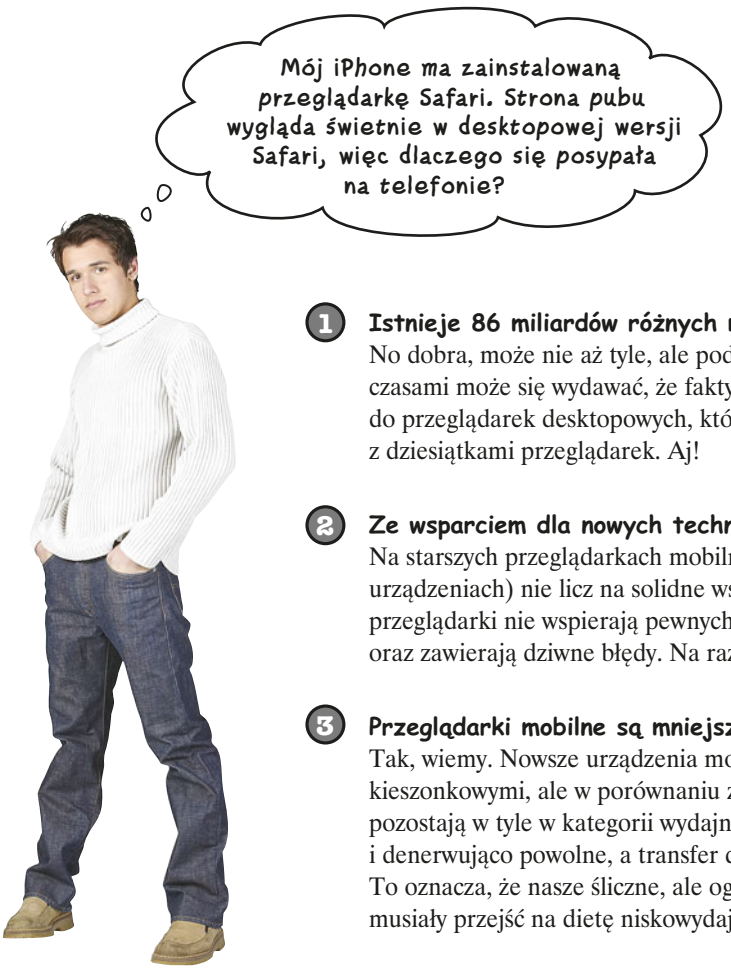
Niemiłym zaskoczeniem dla Michała był widok witryny na jego nowym iPhonie 4. Co gorsza, na Androidzie sytuacja nie wygląda lepiej.

Na iPhonie 4 witryna Pod Paradnym Morsem wygląda tak...

...a to widok strony na telefonie z Androidem.



Dlaczego na mobilne technologie webowe trzeba patrzeć inaczej?



1. **Istnieje 86 miliardów różnych mobilnych przeglądarek.**
No dobra, może nie aż tyle, ale podczas tworzenia stron dla urządzeń mobilnych czasami może się wydawać, że faktycznie tak jest. Tu, w przeciwieństwie do przeglądarek desktopowych, których jest kilka, mamy do czynienia z dziesiątkami przeglądarek. Aj!

A kiedy już Ci się wydaje, że wszystkie je ogarnąłeś, pojawia się nagle nowa przeglądarka.

2. **Ze wsparciem dla nowych technologii internetowych bywa różnie.**
Na starszych przeglądarkach mobilnych (a nawet nowszych, ale na słabszych urządzeniach) nie licz na solidne wsparcie dla CSS czy JavaScriptu. Nawet najnowsze przeglądarki nie wspierają pewnych technologii, obsługują je tylko częściowo lub inaczej oraz zawierają dziwne błędy. Na razie to prawdziwy dziki zachód!

3. **Przeglądarki mobilne są mniejsze i wolniejsze.**
Tak, wiemy. Nowsze urządzenia mobilne są pełnoprawnymi komputerami kieszonkowymi, ale w porównaniu z komputerami stacjonarnymi (i laptopami) nadal pozostają w tyle w kategorii wydajności. Sieci bezprzewodowe bywają niepewne i denerwująco powolne, a transfer danych niekoniecznie jest darmowy i nieograniczony. To oznacza, że nasze śliczne, ale ogromne, bogate w media i opasłe witryny będą musiały przejść na dietę niskowydajnościową.

4. **Interfejsy użytkownika stosowane w urządzeniach mobilnych wymuszają przemyślenie struktury witryn.**
To, że mobilne przeglądarki potrafią (z drobnymi potknięciami) wyrenderować strony przeznaczone dla ich desktopowych odpowiedników, nie oznacza, że koniecznie *powinny* to robić. Ekrany są mniejsze, a sposób interakcji i przewidywane zachowania są inne.

   Użytkownicy urządzeń mobilnych korzystają z wielu różnych sposobów interakcji: używają palców, rysików, a nawet miniaturowej klawiatury. Wprowadzanie tekstów i wypełnianie formularzy jest raczej żmudnym zajęciem. Poza tym wpatrywanie się w miniaturowe literki, które są dopasowane do desktopowych, ale nie mobilnych przeglądarek, może przyprawić o ból głowy i doprowadzić do szału. Chyba już wiesz, w czym rzecz.

Ćwiczenie

Przeglądarka z iPhone'a Michała tak renderuje stronę pubu Pod Paradnym Morsem.
Nie wygląda to dobrze. Czy umiesz wskazać problematyczne obszary?
Napisz o wszystkim, co Twoim zdaniem jest nie tak.

1 .....

.....

.....

.....

2 .....

.....

.....

.....

3 .....

.....

.....

.....

4 .....

.....

.....

.....

Czy zauważyłeś jakieś z tych problemów?

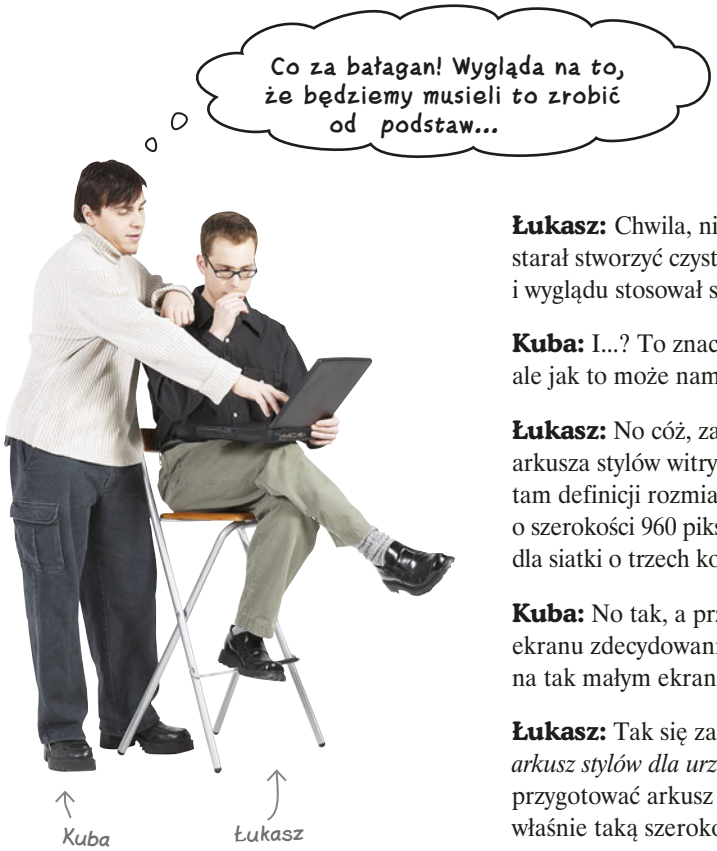
Rozwiązanie ćwiczenia

1. *Linki w pasku nawigacji są mikroskopijne, więc trudno je odczytać, nie mówiąc o ich kliknięciu.*

2. *Osadzone wideo z YouTube'a nie działa.*

3. *Przy takim rozmiarze ekranu trzykolumnowy układ strony jest nieporozumieniem. Poza tym tekst jest tak mały, że trudno go odczytać.*

4. *Z prawej strony ekranu pojawia się dziwna pusta przestrzeń.*



Łukasz: Chwila, nie tak szybko! Wiemy przecież, że Michał bardzo się starał stworzyć czysty, semantyczny kod HTML, a do układu graficznego i wyglądu stosował style.

Kuba: I...? To znaczy — świetnie, prawdziwie profesjonalna robota, ale jak to może nam pomóc w poprawieniu tej strony?

Łukasz: No cóż, zastanówmy się nad tym chwilę. Kiedy zajrzałem do arkusza stylów witryny Pod Paradnym Morsem zauważyłem, że pełno tam definicji rozmiarów różnych elementów dopasowanych do strony o szerokości 960 pikseli. Wygląda na to, że Michał zaprojektował tę stronę dla siatki o trzech kolumnach i całkowitej szerokości równej 960 pikseli.

Kuba: No tak, a przecież większość urządzeń mobilnych ma szerokości ekranu zdecydowanie mniejsze niż 960 pikseli. Poza tym trzy kolumny na tak małym ekranie to stanowczo za dużo.

Łukasz: Tak się zastanawiam... *a co, gdybyśmy zastosowali inny arkusz stylów dla urządzeń mobilnych*? A co byś powiedział na to, żeby przygotować arkusz dla 320 pikseli, bo ekrany wielu smartfonów mają właśnie taką szerokość? I może jeszcze zmniejszymy liczbę kolumn?

Kuba: Dobry pomysł, ale obawiam się, że nie obejdzie się bez masy kodu. Nie wiem po prostu, jak zmusić urządzenia mobilne do korzystania z innego arkusza stylów.

Łukasz: A wiesz, że Iza właśnie wróciła z warsztatów na temat mobilnych technologii webowych i jest zachwycona czymś, co zwie się *Responsive Web Design*?

Kuba: Jak mógłbym to przeoczyć? Przecież bez przerwy o tym mówi.

Łukasz: No właśnie. Wiem od niej, że cieszy się to dużym zainteresowaniem wśród programistów. Z tego, co pamiętam, częściowo jest to związane właśnie ze stosowaniem odpowiednich arkuszy stylów w różnych sytuacjach i wcale nie wymaga, jak to nazwałeś, „masy kodu". Poza tym najprawdopodobniej to podejście jest szczególnie przydatne podczas tworzenia witryn dla urządzeń mobilnych. Nie pamiętam żadnych szczegółów, ale na pewno uda nam się coś znaleźć na ten temat.

Wrażliwe projekty — Responsive Web Design

Responsive Web Design (RWD) jest zestawem technik zaproponowanych przez projektanta Ethana Marcotte'a. Witryny, które zostały zaprojektowane zgodnie z tym podejściem, dopasowują swój układ do środowiska przeglądarki użytkownika. W dużej mierze polega to na zmyślnym manipulowaniu stylami w arkuszach CSS.

Przeczytaj oryginalny artykuł Ethana na temat RWD, który jest dostępny pod adresem http://bit.ly/nRePnj.

W zależności od bieżącej wartości wybranych cech przeglądarki, takich jak rozmiar okna, orientacja czy współczynnik proporcji, możemy zastosować różne arkusze stylów. Jeśli przemyślimy sposób tworzenia układów stron, możemy uzyskać optymalne rozmieszczenie elementów w oknie przeglądarki, i to niezależnie od jego rozmiarów.

Podejście RWD jest jednym z prostszych i szybszych sposobów na tworzenie witryn, które dobrze wyglądają na wielu urządzeniach. Co ważne, wystarczą Ci do tego dotychczasowe umiejętności i wiedza o technologiach webowych.

Przepis na stosowanie RWD

Można wyróżnić trzy podstawowe techniki stosowane podczas tworzenia „wrażliwych" projektów witryn internetowych:

1. **Zapytania o media w CSS3**
   Wybranie odpowiedniego arkusza stylów na podstawie określonych właściwości bieżącego środowiska przeglądarki.

   W zależności od szerokości okna przeglądarki, jego współczynnika proporcji lub orientacji możemy zastosować różne reguły CSS.

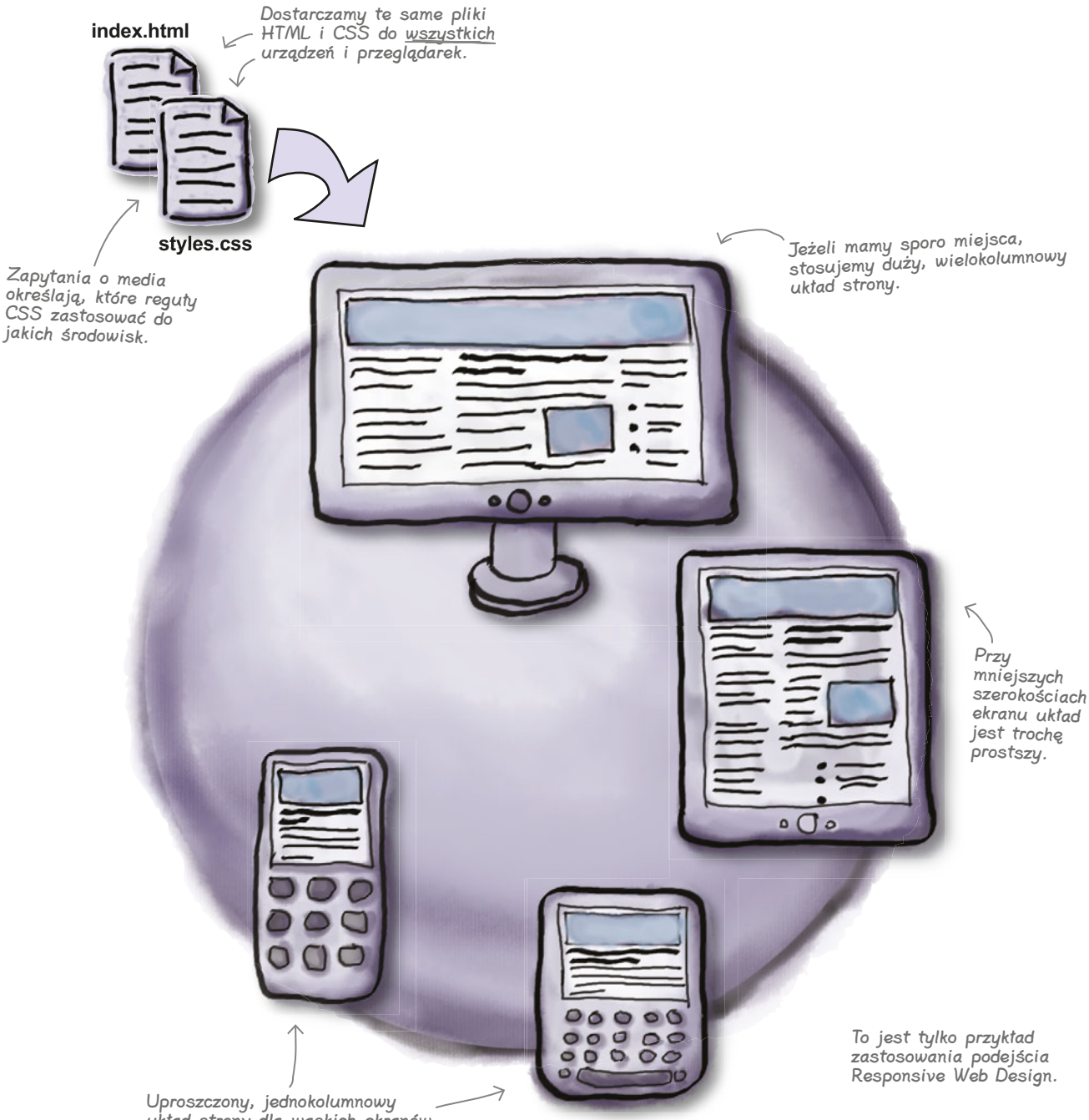
2. **Układy stron oparte na płynnych siatkach**
   Ustalanie względnych, a nie bezwzględnych rozmiarów elementów układu strony.

   Zgodnie z podejściem RWD do określania wymiarów kolumn i innych elementów układu strony jako jednostek używamy procentów, a nie pikseli.

3. **Dopasowujące się obrazy i inne media**
   Zastosowanie możliwości stylów do skalowania obrazów i innych mediów wraz ze zmianą wymiarów zawierających je kontenerów.

   Obrazy i media utrzymują się w granicach zawierających je elementów i skalują się proporcjonalnie do pozostałych elementów układu.

Przykład strony zaprojektowanej zgodnie z RWD



To jest tylko przykład zastosowania podejścia Responsive Web Design.

Różne arkusze stylów w różnych sytuacjach

Jeżeli zajmujesz się tworzeniem stron już od jakiegoś czasu (i dobrze znasz CSS), najprawdopodobniej spotkałeś się z **typami mediów w CSS**. **Reguł @media możemy użyć do selektywnego stosowania stylów.**

Deklaracje typów mediów umieszczone w arkuszu stylów wyglądają mniej więcej tak:

```
@media screen { /* Reguły CSS dla ekranu! */ }
```

Typ medium to „screen", czyli ekran.

Reguły umieszczone między nawiasami klamrowymi zostaną zastosowane tylko wtedy, gdy zawartość jest renderowana na ekranie.

Zbliżenie na typy mediów

Główne (i najczęściej stosowane) typy mediów to: `screen`, `print` oraz `all`. Są też inne, ale dużo rzadziej spotykane, na przykład: `aural`, `braille` czy `tv`.

Zaciekawiło Cię to? Jeżeli należysz do tej grupy ludzi, którzy czytają specyfikacje techniczne dla przyjemności bądź w celu zaspokojenia ciekawości, mamy dla Ciebie smaczny kąsek — definicje typów mediów z witryny W3C: *www.w3.org/TR/CSS2/media.html*.

Innym sposobem korzystania z typów mediów do selektywnego stosowania stylów jest określenie ich w znaczniku `<link>` w dokumencie HTML.

```
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
```

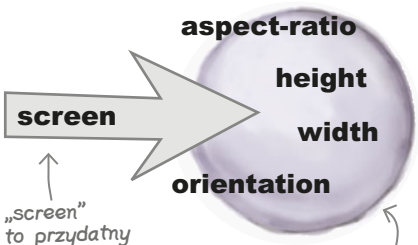
Reguły znajdujące się w tym zewnętrznym arkuszu stylów zostaną zastosowane tylko wtedy, gdy zawartość jest renderowana na urządzeniu drukującym (czyli, mówiąc po ludzku, na drukarce).

Kolejny typ medium to „print".

Przedstawiony tu sposób selektywnego dołączenia pliku CSS jest typowy dla medium `print`, czyli definicji stylów opracowanych tylko na potrzeby drukowania zawartości strony.

Typy mediów to nie wszystko — poznajcie cechy mediów

Siebie samego mógłbyś z pewnością określić pewnymi cechami — wiekiem, wzrostem itd. To samo dotyczy mediów. Podobnie jak moglibyśmy chcieć ustalić zasadę, by wstęp do pubu Pod Paradnym Morsem mieli tylko pełnoletni, równie dobrze można by zdefiniować różne arkusze stylów stosowane tylko do wybranych zakresów szerokości okna przeglądarki.



„screen" to przydatny typ mediów.

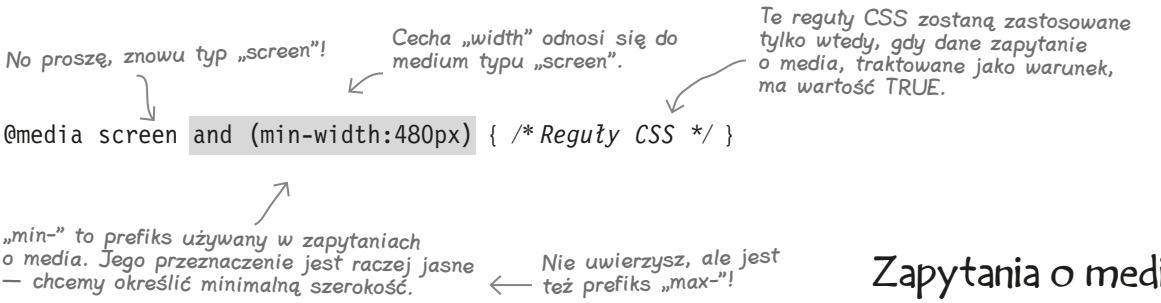
Kilka cech mediów typu „screen".

PS. Jest ich więcej, ale te są najczęściej używane.

Mamy dobrą wiadomość! Zarówno `width` (szerokość), `color` (kolor), jak i `orientation` (orientacja) są **cechami mediów** zdefiniowanymi w specyfikacji CSS3 dla wszystkich głównych typów mediów. Zatem ***typy*** mediów mają określone ***cechy***.

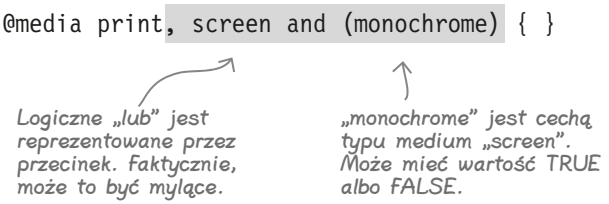
Same cechy mediów na niewiele nam by się zdały. Aby w jakiś sposób je wykorzystać, musimy mieć możliwość zapytania przeglądarki o stan wybranych cech. I tu z pomocą przychodzą nam wprowadzone w CSS3 ***zapytania o media*** (ang. *media queries*).

Zapytania o media w CSS



Przedstawione tu zapytanie można odczytać tak: **czy renderujemy zawartość na ekranie *ORAZ* czy *bieżąca* szerokość okna wynosi co najmniej 480 pikseli?** Tak? To świetnie! Stosujemy zdefiniowane tu reguły CSS.

Inny przykład:



Czy renderujemy na drukarkę LUB na monochromatyczny ekran (np. wyświetlający w odcieniach szarości)? Tak? Stosujemy te style!

Zapytania o media w CSS3 to wyrażenia logiczne wyznaczane w oparciu o bieżące wartości wybranych cech medium, z którego korzysta przeglądarka użytkownika. Jeśli to wyrażenie ma wartość TRUE, zostają zastosowane zdefiniowane reguły CSS.

Tłumaczenie zapytań o media. Spróbuj sam! Zapytania połącz z objaśnieniami.

Ćwiczenie

```
@media all and (orientation: landscape) {}
```

Stosuje reguły z zewnętrznego arkusza do ekranów wyświetlających w kolorze.

```
<link rel="stylesheet" type="text/css"
href="my.css" media="screen and (color)" />
```

Stosuje style do monochromatycznych drukarek.

```
@media print and (monochrome) {}
```

Stosuje style do ekranów wyświetlających w kolorze.

```
@media screen and (color) {}
```

Stosuje style do wszystkich typów mediów, ale tylko w przypadku poziomej orientacji.

Udało Ci się odszyfrować zapytania o media?

Rozwiązanie ćwiczenia

Zastanawiasz się, o co chodzi z tym określeniem typu „all"? Stosujemy je, gdy chcemy zdefiniować reguły dla określonych cech wszystkich typów mediów.

```
@media all and (orientation: landscape) {}
```

Stosuje reguły z zewnętrznego arkusza do ekranów wyświetlających w kolorze.

```
<link rel="stylesheet" type="text/css"
href="my.css" media="screen and (color)" />
```

Stosuje style do monochromatycznych drukarek.

```
@media print and (monochrome) {}
```

Stosuje style do ekranów wyświetlających w kolorze.

```
@media screen and (color) {}
```

Stosuje style do wszystkich typów mediów, ale tylko w przypadku poziomej orientacji.

W porządku, już rozumiem, czym są zapytania o media, i może nawet potrafiłabym sama jakieś wymyślić. Ale co z tym dalej zrobić? Jak zdefiniować style dla urządzeń mobilnych?

Na ile różne są różnice w CSS?

Znamy już narzędzie, które pozwoli nam stosować różne style w różnych sytuacjach. Ale co dalej?

Nie panikuj. Będziemy co prawda musieli napisać kilka reguł CSS przyjaznych urządzeniom mobilnym, ale nie będziemy zaczynać od zera. Jeżeli style dla urządzeń mobilnych nie muszą być zupełnie inne od standardowych, możemy współdzielić sporo tego, co już mamy.

Aby wygenerować układ graficzny odpowiedni dla urządzeń mobilnych, musimy:

- ☐ Przyjrzeć się dotychczasowemu układowi strony i przeanalizować jej strukturę.
- ☐ Znaleźć te fragmenty, które wymagają dostosowania do urządzeń mobilnych.
- ☐ Zdefiniować style dla wybranych elementów.
- ☐ Przebudować arkusz CSS i — za pomocą zapytań o media — wybiórczo stosować pewne style do urządzeń mobilnych i stacjonarnych.

Zdefiniujemy dodatkowe style tylko dla tych elementów układu graficznego, które są inne dla urządzeń mobilnych.

Dotychczasowa struktura witryny pubu Pod Paradnym Morsem

Rzuć okiem na plik *index.html* znajdujący się w katalogu *rozdzial1*. Jeżeli użyjesz wyobraźni i „potniesz” ten HTML, dostrzeżesz podstawową strukturę strony, która przedstawia się następująco:

Trzy główne kolumny na treść strony.

```
<div class="navigation">...</div>
<div class="header">...</div>
<h1>...</h1>
<div id="visit" class="column">...</div>
<div id="points" class="column">...</div>
<div id="main" class="column">...</div>
<div class="footer">...</div>
```

Umieszczenie kodu prawej kolumny przed kodem środkowej jest typowym trikiem stosowanym w projektowaniu stron, dzięki któremu łatwiej zapanować nad układem z zastosowaniem opływania.

Dotychczasowy arkusz stylów, dostosowany do stacjonarnych przeglądarek, rozmieszcza elementy strony w taki sposób:

Spójrzmy teraz na arkusz CSS, który definiuje ten układ strony, i sprawdźmy, co trzeba zmienić, by zaadaptować go na potrzeby urządzeń mobilnych.

Analiza dotychczasowego arkusza CSS

Otwórz plik *styles.css* dołączony do witryny pubu Pod Paradnym Morsem.

Na początku tego pliku jest sporo reguł, którymi nie musimy się w tej chwili przejmować. Są one związane z kolorami, typografią i innymi sprawami, które powinny zostać jednakowe w wersjach stacjonarnej i mobilnej.

To, czym się musimy zająć, to style związane ze **strukturą**, a odpowiadające za to reguły znajdują się na końcu pliku.

Skupiamy się teraz na strukturalnej części arkusza CSS.

```
/* Struktura */
body, .header, .navigation, .footer {
    width: 960px;
}
.header, .navigation, .footer {
    clear: both;
}
.column {
    margin: 10px 10px 0 0;
}
.navigation {
    min-height: 25px;
}
.navigation ul li {
    width: 320px; /* 960/3 */
}
.header {
    background:url(images/w.png) no-repeat;
    height: 200px;
}
#visit {
    width: 240px;
    float: left;
}
#points {
    width: 240px;
    float: right;
}
#main {
    margin: 10px 260px 0 250px;
    width: 460px;
}
```

Strona ma szerokość 960 pikseli. Nagłówek, stopka i elementy nawigacyjne rozciągają się na całą szerokość.

Ponieważ te elementy rozciągają się na całą szerokość, dbamy o to, by nic nie opływało ich z żadnej ze stron.

Dzięki deklaracji clear: both elementy są umieszczane „w nowym wierszu", czyli nic nie ma prawa znaleźć się obok nich.

Wszystkie kolumny (lewa, prawa oraz środkowa) mają 10-pikselowy margines na górze i z prawej strony (pełniący funkcję odstępu między kolumnami).

Odsyłacze w bloku nawigacji są elementami listy <ul>. Układamy je w poziomie, a każdy element <li> ma szerokość równą 1/3 szerokości strony.

Szerokość każdego elementu <li> jest równa 1/3 szerokości całej strony, ponieważ w bloku nawigacji są trzy odsyłacze.

Nagłówek ma w tle obraz, więc — by był w całości widoczny — konieczne jest ustalenie wysokości równej 200 pikseli.

Dwie kolumny (lewa i prawa) mają szerokość 240 pikseli i ustawione opływanie.

Główna kolumna nie ma ustawionego opływania, ponieważ jest pozycjonowana za pomocą marginesów.

Jej lewy margines ma 260, a prawy — 250 pikseli względem krawędzi strony.

Wydaje się, że środkowa kolumna powinna mieć szerokość 480 pikseli (960 minus dwie kolumny po 240 pikseli każda). Szerokość ma jednak 460 pikseli, ponieważ zostały uwzględnione dwa odstępy między kolumnami, po 10 pikseli każdy.

Co trzeba zmienić?

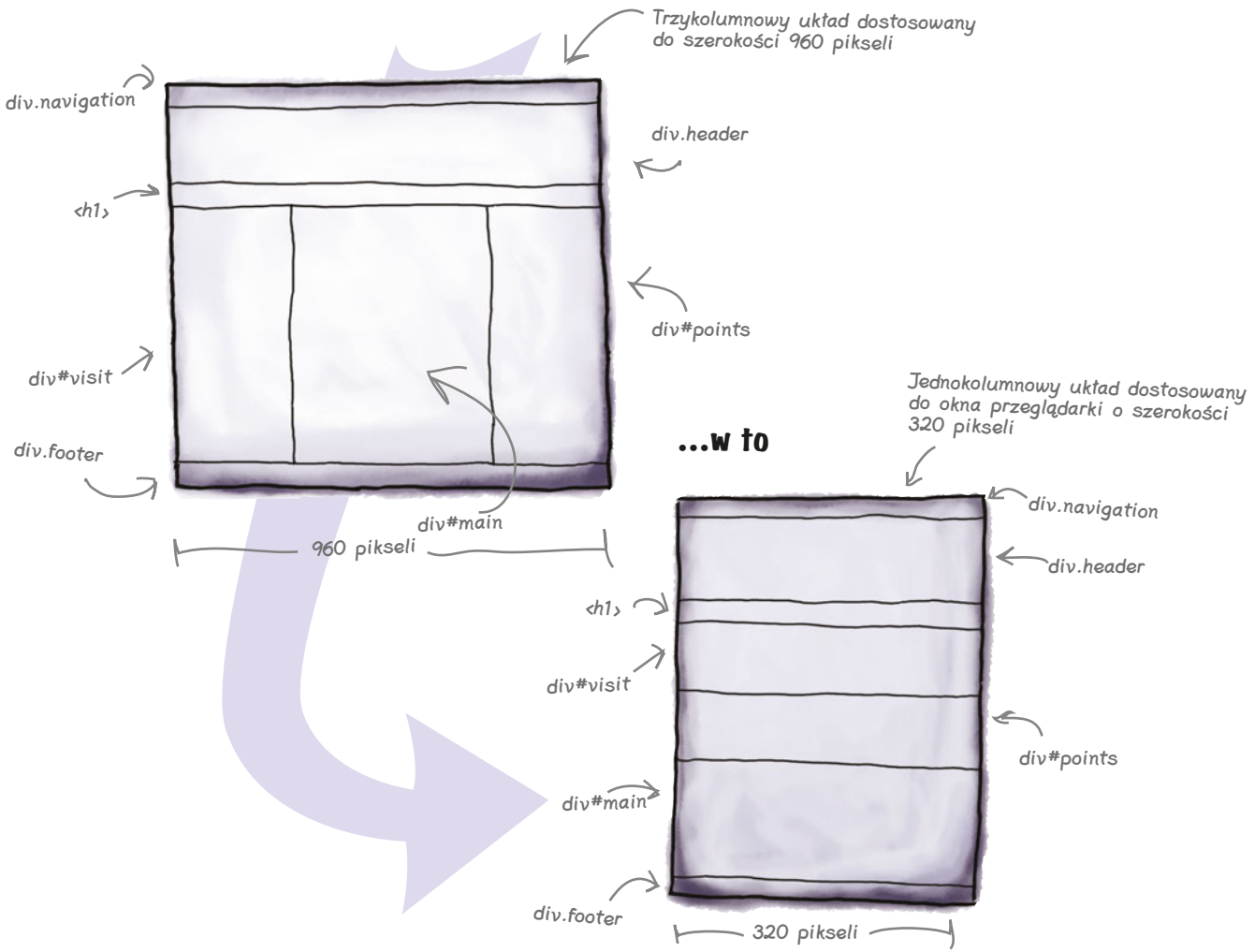
1. **Musimy zmienić stronę tak, aby jej elementy strukturalne zmieściły się przy szerokości 320 pikseli.** Wcześniej (na stronie 9) Łukasz stwierdził, że typową szerokością ekranów w urządzeniach mobilnych jest właśnie 320 pikseli.

2. **Musimy też zmniejszyć liczbę kolumn z trzech do jednej.** Na małym ekranie urządzenia mobilnego oryginalny trzykolumnowy układ byłby dość nieczytelny.

Nie musimy przepisywać na nowo całego arkusza CSS.

Wystarczy, że dostosujemy niektóre ze strukturalnych elementów układu strony. Pozostałe sprawy, takie jak typografia i kolory, mogą pozostać niezmienione.

Na potrzeby urządzeń mobilnych musimy zmienić to...



Szukamy stylów wymagających zmiany

Zaznaczony kod trzeba będzie dostosować do urządzeń mobilnych.


```
/* Struktura */
body, .header, .navigation, .footer {
    width: 960px;
}
.header, .navigation, .footer {
    clear: both;
}
.column {
    margin: 10px 10px 0 0;
}
.navigation {
    min-height: 25px;
}
.navigation ul li {
    width: 320px; /* 960/3 */
}
.header {
    background:url(images/w.png) no-repeat;
    height: 200px;
}
#visit {
    width: 240px;
    float: left;
}
#points {
    width: 240px;
    float: right;
}
#main {
    margin: 10px 260px 0 250px;
    width: 460px;
}
```

styles.css

Droga do stylów dostosowanych do urządzeń mobilnych

1. **Zmieniamy szerokości w zaznaczonych regułach stylów.**
2. **Pozbywamy się tych reguł, które są zbędne.**
3. **Wyodrębniamy wspólne reguły stylów.** ← *Tym zajmiemy się już za chwilę.*

Magnesiki z mobilnymi stylami

Skorzystaj z magnesików i skonstruuj CSS odpowiedni dla urządzeń mobilnych.

```
body, .header, .footer, .navigation {

}
.column {

   border-bottom: 1px dashed #7b96bc;
}
.navigation ul li {

}
#visit, #points, #main {

}
```

Ta krawędź służy do wizualnego oddzielenia kolumn (teraz umieszczonych w pionie).

`width: 320px;`

`width: 106.6667px;`

`margin: 10px 0;`

`width:320px;`

Magnesiki z mobilnymi stylami. Rozwiązanie

```
body, .header, .footer, .navigation {
  width: 320px;
}
.column {
  margin: 10px 0;
  border-bottom: 1px dashed #7b96bc;
}
.navigation ul li {
  width: 106.6667px;
}
#visit, #points, #main {
  width: 320px;
}
```

Tadam! Mobilny arkusz gotowy

Skończyliśmy! Te cztery reguły CSS to wszystko, czego potrzebujemy w mobilnej wersji układu strony. Teraz musimy tylko dopilnować, by zostały zastosowane na urządzeniach mobilnych.

A jak to zrobimy? Nasz stary druh, pan Zapytanie o'Media, przybywa na ratunek! Już za moment utworzymy zapytanie, dzięki któremu przygotowane reguły zostaną zastosowane w urządzeniach z ekranami o szerokości równej 480 pikseli i mniejszych.

Dlaczego 480 pikseli? Dlatego że jest to wymiar dłuższego boku ekranu (w orientacji poziomej) w wielu popularnych smartfonach.

One nie zniknęły...

...one po prostu nie muszą być umieszczone wśród stylów dla urządzeń mobilnych. Dlaczego? Ano dlatego, że zostaną zastosowane w przypadku obu układów stron: mobilnym i stacjonarnym.

Wspólne reguły CSS umieścimy poza zapytaniami o media, dzięki czemu nie będziemy musieli wstawiać ich w dwóch miejscach. Zajmijmy się tym.

Pozostała część strukturalnych stylów

Wspólne reguły

Widzisz? Mówiliśmy, że żaden fragment arkusza stylów tak naprawdę nie zniknął! Oto wspólne reguły CSS wydzielone na stronie 18.

```
.header, .footer, .navigation {
   clear: both;
}
.header {
   background:url(images/w.png) no-repeat;
   height: 200px;
}
.navigation {
   min-height: 25px;
}
```

Reguły dla stacjonarnych przeglądarek

Pamiętaj, że nadal są potrzebne prawidłowe style dla przeglądarek stacjonarnych!

Po usunięciu wspólnych reguł CSS pozostają nam **desktopowe reguły stylów**.

W tym przypadku musimy użyć zapytania o media, tak by style zostały zastosowane tylko na ekranach o szerokości większej niż 480 pikseli.

```
body, .header, .footer, .navigation {
    width: 960px;
}
.column {
    margin: 10px 10px 0 0;
}
.navigation ul li {
    width: 320px; /* 960/3 */
}
#visit {
    width: 240px;
    float: left;
}
#points {
    width: 240px;
    float: right;
}
#main {
    margin: 10px 260px 0 250px;
}
```

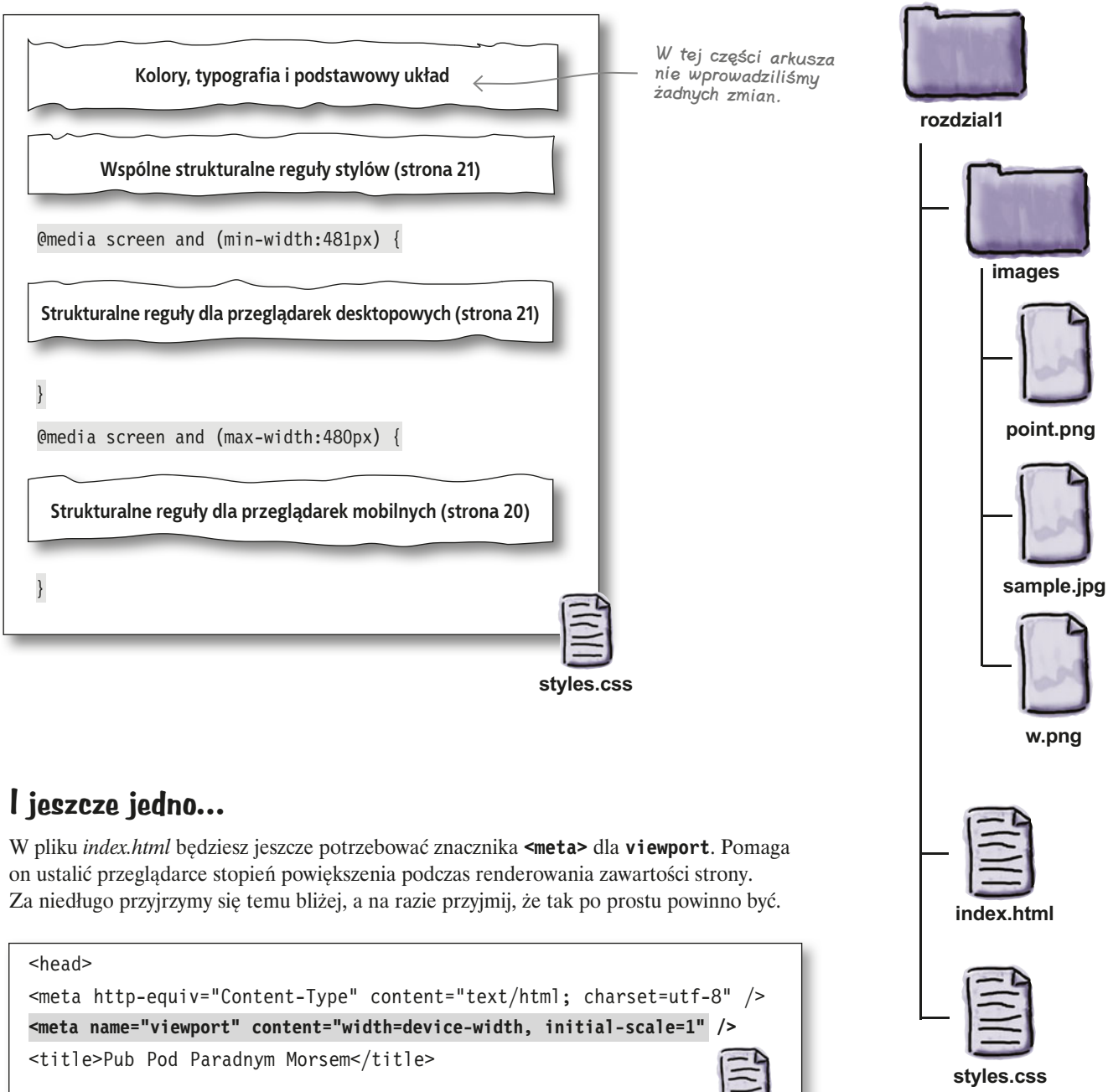
Co dalej?

Sprawdźmy, jak wygląda teraz nasza lista działań związanych z dostosowywaniem strukturalnych stylów do przeglądarek desktopowych i mobilnych:

- ☑ Przyjrzeć się dotychczasowemu układowi strony i przeanalizować jej strukturę.
- ☑ Znaleźć te fragmenty, które wymagają dostosowania do urządzeń mobilnych.
- ☑ Zdefiniować style dla wybranych elementów.
- ☐ **Przebudować arkusz CSS i — za pomocą zapytań o media — wybiórczo stosować pewne style do urządzeń mobilnych i stacjonarnych.**

Składanie w całość

Zaktualizowaną wersję arkusza stylów *styles.css* poskładamy w taki sposób:



I jeszcze jedno...

W pliku *index.html* będziesz jeszcze potrzebować znacznika **<meta>** dla **viewport**. Pomaga on ustalić przeglądarce stopień powiększenia podczas renderowania zawartości strony. Za niedługo przyjrzymy się temu bliżej, a na razie przyjmij, że tak po prostu powinno być.

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>Pub Pod Paradnym Morsem</title>
```

index.html

Zrób to sam!

1. **Zmodyfikuj plik index.html.**
   Dopisz znacznik `<meta>` dla `viewport` ze strony 22.

2. **Otwórz plik styles.css.**
   Musisz podmienić strukturalne reguły stylów znajdujące się na końcu pliku CSS. Usuń istniejące reguły elementów strukturalnych.

3. **Dodaj wspólne reguły.**
   Dodaj wspólne reguły stylów ze strony 21.

4. **Dodaj style dla przeglądarek desktopowych i mobilnych.**
   Dodaj style przeznaczone dla przeglądarek desktopowych (strona 21) i mobilnych (strona 20).

5. **Fragmenty przeznaczone dla przeglądarek desktopowych i mobilnych otocz zapytaniami o media.**
   Dodaj zapytania o media ze strony 22.

Jazda próbna

Po wprowadzeniu zmian otwórz desktopową przeglądarkę i załaduj plik *index.html*. Następnie zmniejsz szerokość okna do mniej niż 481 pikseli, aby zobaczyć wersję układu strony dla urządzeń mobilnych.

W desktopowej przeglądarce strona wciąż wygląda tak samo...

...a w oknie o szerokości mniejszej niż 481 pikseli zmienia się układ strony na ten, który zoptymalizowaliśmy pod kątem urządzeń mobilnych.

Dobrze zaczęliście, ale jest pewien problem. Kiedy na moim iPhonie przewinę stronę na dół, zobaczę zdjęcie, które jest zdecydowanie za duże i psuje cały układ.

Łukasz: To naprawdę frustrujące. Miałem nadzieję, że CSS, który przygotowaliśmy, załatwi tę sprawę.

Iza: Wasz arkusz stylów dostosowuje układ strony do mniejszych ekranów, ale i tak jest dosyć „sztywny". Zobacz, jak wygląda ta strona, kiedy obrócę telefon poziomo.

Łukasz: Widzę... Strona ma wciąż 320 pikseli szerokości, tyle że jest wyświetlana na ekranie o szerokości 480 pikseli. Czy to znaczy, że muszę przygotować style dla każdej możliwej szerokości ekranu?!

Iza: I właśnie w takich sytuacjach ujawniają się zalety podejścia **Responsive Web Design**. W tej chwili dla ekranów o szerokości 480 pikseli i węższych dostarczacie jeden układ ze sztywno określonymi rozmiarami elementów. Z całą pewnością nie jest to elastyczne rozwiązanie. Przecież nie wszystkie urządzenia mobilne mają ekrany o szerokości 320 pikseli.

Podejście RWD pomaga w dostosowaniu układu strony w różnych sytuacjach. Zamiast podawać dokładne wymiary elementów (tak jak robimy w tej chwili, określając wszystkie wielkości w pikselach), możemy zastosować proporcjonalne układy, które znacznie lepiej radzą sobie ze zróżnicowanymi warunkami.

Łukasz: Proporcjonalne? Chodzi ci o jednostki takie jak em i procenty?

Iza: Tak, w pewnym sensie. W regułach układu strony zamiast pikseli stosujemy procenty. Dzięki temu zawartość strony kurczy się i rozszerza, wypełniając dostępną przestrzeń. Zachowuje się podobnie jak woda wypełniająca wgłębienia, dlatego często mówi się o **płynnych układach**. Przy okazji uda nam się zapanować nad tym krnąbrnym zdjęciem.

Łukasz: Zatem cała ta zabawa z zapytaniami o media to strata czasu?

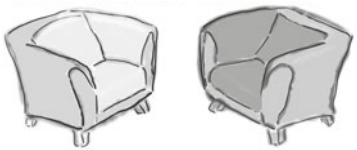
Iza: Gdzie tam! Zapytania o media to bardzo ważny składnik podejścia RWD.

To zdjęcie jest szersze niż 320 pikseli, więc psuje zaprojektowany układ strony.

Na iPhonie nie działa też wideo z YouTube'a.

Kolejnym krokiem na drodze do RWD, dzięki któremu strony będą się czuły dużo lepiej w większości przeglądarek i urządzeń, jest przekształcenie układu zdefiniowanego na sztywno z użyciem pikseli na proporcjonalny układ oparty na płynnych siatkach.

Pogawędki przy kominku



W dzisiejszym programie: sztywna siatka kontra płynna siatka

Płynna siatka

Cześć, sztywna siatko. Wiem, że siedzisz w tym biznesie już od dawna i niejedno widziałaś, ale — bez urazy — w twojej filozofii jest kilka problematycznych kwestii.

To znaczy... Chciałam powiedzieć, że odchodzisz w przeszłość. Przez te na sztywno określone wymiary jesteś skostniała i nieelastyczna. Jednym słowem, nie jesteś wrażliwa.

Nie reagujesz na zmiany w środowisku użytkownika. Zawsze jesteś taka sama.

Robisz to kosztem użytkowników. Sama zobacz, jak wyglądasz w oknach przeglądarek o innych rozmiarach. Albo czegoś nie ma, bo się nie zmieściło, albo jest masa wolnej przestrzeni.

Może to działało w twoich czasach, ale świat się zmienia — teraz jest mnóstwo różnych przeglądarek i urządzeń. Jeśli postanowisz zerwać z precyzyjnymi, sztywnymi układami — które stają się na naszych oczach reliktami przeszłości — i pozwolisz, by zawartość stron płynęła jak woda, wypełniając dostępną przestrzeń okna przeglądarki, możesz być w pełni przygotowana na różne sytuacje.

Sztywna siatka

Czy mnie słuch nie myli? Ma jeszcze mleko pod nosem, a patrzcie, jaka ważna. A tak w ogóle, to co ty wiesz o filozofii?

Wrażliwa?

A widzisz coś złego w niezmienności? Pozostaję wierna zamierzeniom projektanta! 960 pikseli szerokości, 240-pikselowe kolumny po lewej i prawej.

Niech by mi tylko jakiś użytkownik zamarudził, już ja bym mu dała do wiwatu! Korzystaliby ze standardowych przeglądarek, a nie jakichś tam wynalazków...

No dobrze, wymoczku, pokaż mi, na co cię stać.

Co jest nie tak z układami o stałej szerokości?

Strona w oknie o szerokości 960 pikseli.

Jeśli przeglądarki na całym świecie miałyby zawsze tę samą wielkość okna, byłoby sielsko i anielsko, a projektanci stron mogliby mieć pełną kontrolę nad wyglądem tworzonych przez siebie witryn.

Niestety to niemożliwe. Czasem staramy się projektować witryny dla okien o „standardowej" szerokości 640, 960 czy 1024 pikseli. Prawda jest jednak taka, że to tylko złudzenie, bo nie ma standardowych szerokości okna przeglądarki. A przecież jeszcze nie wspomnieliśmy o urządzeniach mobilnych.

Musimy się jednak zgodzić, że układ witryny pubu Pod Paradnym Morsem oparty na sztywnej siatce wygląda całkiem dobrze w oknach o szerokości 960 pikseli.

Układ nie dopasowuje się do okien o innej szerokości

Co się jednak stanie, jeżeli stronę wyświetlimy w węższym oknie? Szerokości kolumn pozostają bez zmian, w związku z czym zawartość zostaje ucięta i, by ją zobaczyć, użytkownik musi korzystać z poziomego paska przewijania. Paskudztwo!

Strona wyświetlona w oknie o szerokości 700 pikseli.

Żeby zobaczyć prawą kolumnę, musisz przewinąć stronę w poziomie.

Lewa i prawa kolumna mają szerokość 240 pikseli... zawsze.

W szerszych oknach cały układ ma nadal szerokość 960 pikseli, więc po prawej stronie powstaje pusta przestrzeń. Hm...

Szerokość jest ustawiona na sztywno na 960 pikseli.

Strona wyświetlona w oknie o szerokości 1200 pikseli.

Dlaczego płynne jest lepsze?

W układach opartych na płynnych siatkach przy określaniu szerokości zamiast pikseli stosuje się proporcjonalne jednostki (procenty). Możemy pozostać wierni wizji projektanta i ustalić szerokości lewej oraz prawej kolumny na jedną czwartą szerokości strony, ale zamiast ustalać je na 240 pikseli, definiujemy je jako 25%.

Płynna wersja układu w oknie o szerokości 960 pikseli. Wygląda tak samo jak wcześniej, prawda?

Układ dopasowuje się do bieżącej szerokości okna

W oknach o innej szerokości zawartość przepływa, wypełniając dostępną przestrzeń układu strony. Zarówno lewa, jak i prawa kolumna zawsze zajmują 25% szerokości całego okna, więc nie dochodzi do obcinania zawartości w węższych oknach, a w szerszych nie powstają puste przestrzenie.

Płynna wersja układu przy 700 pikselach.

Szerokości lewej i prawej kolumny są ustawione na 25% szerokości okna.

Płynna wersja układu przy 1200 pikselach.

WYTĘŻ UMYSŁ

W ten sam sposób zmodyfikujemy reguły zarówno dla desktopowych, jak i mobilnych przeglądarek. Jak sądzisz, czy pomoże to rozwiązać niektóre z problemów z przeglądarkami mobilnymi, o których wspomniała Iza?

W stronę płynności

Aby rozwiązać problemy wskazane przez Izę i zbliżyć się do realizacji zgodnej z podejściem RWD, musimy się zająć kilkoma sprawami.

- ☐ Trzeba przekształcić układ zwymiarowany w pikselach na płynny, w którym są określane proporcjonalne szerokości, a nie stałe.
- ☐ Domyślny rozmiar fontu musimy ustawić na 100%, aby było możliwe proporcjonalne skalowanie.
- ☐ Musimy naprawić niedziałające wideo z YouTube'a.
- ☐ Musimy poprawić zbyt szerokie zdjęcie.

Jeżeli zamierzamy postawić na płynny układ, fonty również muszą być elastyczne.

Wzór płynności

Aby przekształcić układ, w którym szerokości są definiowane za pomocą pikseli, na płynny z proporcjonalnymi wymiarami, musimy zastosować następujący wzór:

Wymiar elementu w pikselach. → **Element docelowy**

Element docelowy / Kontekst = Wynik

Wymiar „kontekstu", w którym jest umieszczony element, wyrażony w pikselach. → **Kontekst**

Wartość dla nowej reguły wyrażona w procentach. → **Wynik**

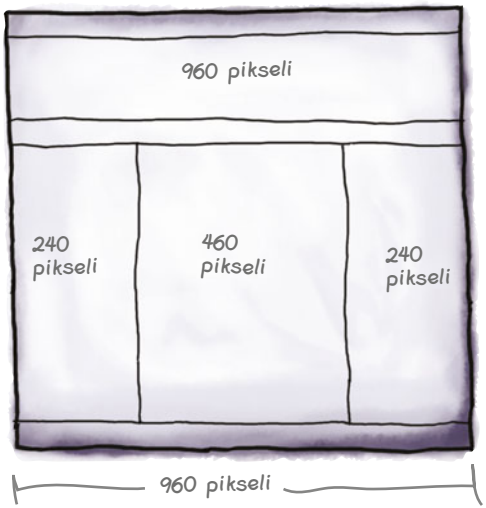
Bliższe spojrzenie

Weźmy teraz na warsztat układ strony pubu Pod Paradnym Morsem i zobaczmy, o co tak naprawdę chodzi z tym płynnym układem.

Zaczniemy od **kontekstu**, który jest podstawą w wyznaczaniu nowych wartości. W omawianym przypadku szerokością odniesienia jest 960 pikseli. Chcemy, by wynikowy układ zachował te same proporcje co jego dotychczasowy odpowiednik. Zatem w naszych obliczeniach przyjmiemy szerokość kontekstu wynoszącą 960 pikseli.

Zarówno pasek nawigacji, jak i nagłówek oraz stopka mają szerokość równą szerokości całej strony. Dzięki temu nasz „wzór płynności" jest naprawdę prosty.

$$\frac{\textbf{960 pikseli}}{\textbf{960 pikseli}} = \textbf{100\%}$$



Ciąg dalszy przekształceń

Szerokość obu bocznych kolumn wynosi 240 pikseli przy szerokości strony równej 960 pikseli. Aby wyznaczyć nową, proporcjonalną wartość, musimy ponownie zastosować znany wzór:

$$\frac{\text{240 pikseli}}{\text{960 pikseli}} = \text{25\%}$$

Szerokość każdej kolumny stanowi jedną czwartą szerokości strony. Wszystko jasne, prawda?

Z główną, środkową kolumną sprawa wygląda trochę inaczej. Ona nie opływa żadnego innego elementu, a do jej pozycjonowania użyliśmy marginesów. Nic to, przecież nic nie stoi na przeszkodzie, by do marginesów zastosować to samo podejście:

Dotychczasowa reguła dla środkowej kolumny.

$$\frac{\text{250 pikseli}}{\text{960 pikseli}} = \text{26.0416667\%}$$

```
#main {
  margin: 10px 260px 0 250px;
}
```

$$\frac{\text{260 pikseli}}{\text{960 pikseli}} = \text{27.0833333\%}$$

Uaktualniona reguła

```
#main {
   margin: 10px 27.0833333% 0 26.0416667%;
}
```

Ćwiczenie

Dokończ przekształcanie strukturalnych reguł w arkuszu *styles.css* na postać proporcjonalną. Będziesz musiał zaktualizować każdą z sześciu reguł zawartych w zapytaniu o media dla okien o szerokości większej niż 480 pikseli.

Nie istnieją głupie pytania

P: Dlaczego w tych wartościach jest tyle cyfr po przecinku? Czy naprawdę potrzebujemy takiej dokładności?

O: Demonstrujemy tu podejście do płynnej siatki w najczystszej postaci, w której wielkości odpowiadają dokładnie wynikom otrzymanym z obliczeń.

Tak naprawdę przeglądarki i tak zaokrąglają te wartości, ale niestety każda robi to w inny sposób.

Zatem wszystko zależy od Ciebie. Możesz zaokrąglić otrzymane wartości do jednego lub dwóch miejsc po przecinku albo zastosować inne podejście i postawić na pewną dowolność w układzie — w końcu jeden procent w tę czy w tamtą stronę nie robi wielkiej różnicy. Co prawda tak zdefiniowany układ strony nie będzie precyzyjny, ale ustrzeżesz się dzięki temu przed problemami z zaokrąglaniem, a obliczenia staną się prostsze.

P: Chwila. Dlaczego górny margines dla bloku `#main` jest nadal zdefiniowany jako `10px`? Czy to nie błąd?

O: Pionowy układ to zupełnie inna sprawa niż poziomy. Nie możemy użyć w tym przypadku 960-pikselowego kontekstu, ponieważ wysokość układu nigdy tak do końca nie jest znana. W związku z tym stosowanie wartości procentowych przy określaniu wysokości i pionowych odstępów nie jest wskazane i może powodować problemy (chociażby ze względu na różnice w poszczególnych przeglądarkach). Podawanie pionowych marginesów w pikselach jest jak najbardziej prawidłowe.

Rzućmy okiem na CSS dla płynnego układu desktopowego.

Rozwiązanie ćwiczenia

```
@media screen and (min-width: 481px) {
    body, .header, .footer, .navigation {
        width: 100%;
    }
    .column {
        margin: 10px 1.04166667% 0 0;
    }
    .navigation ul li {
        width: 33.333333%;
    }
    #visit {
        width: 25%;
        float: left;
    }
    #points {
        width: 25%;
        float: right;
    }
    #main {
        margin: 10px 27.0833333% 0 26.0416667%;
    }
}
```

960 / 960 = 100%

10 / 960 = 1.04166667%

320 / 960 = 33.333333... %

240 / 960 = 25%

Omówiliśmy to na stronie 29.

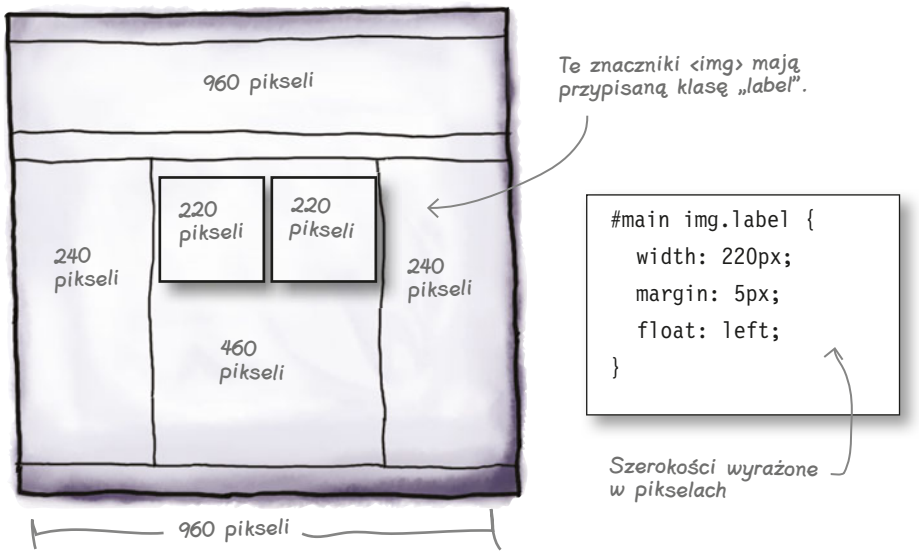
Czy dobrze rozumiem, że aby przekształcić szerokości na postać proporcjonalną, muszę podzielić szerokość elementu wyrażoną w pikselach przez całkowitą szerokość układu?

Nie tak szybko!

Kontekst lubi się czasem zmienić...

Przełączanie kontekstu

Michał chce co miesiąc umieszczać na stronie informacje o promowanych piwach. W związku z tym w głównej kolumnie, oprócz tekstu i zdjęcia, musi się znaleźć miejsce na dwie etykiety promowanych piw umieszczone obok siebie. W projekcie bazującym na pikselach wyglądałoby to tak:



Wydaje się, że wzór pozwalający na obliczenie szerokości tych obrazów powinien wyglądać tak:

$$\frac{\textbf{220 pixels}}{\textbf{960 pixels}} = \textbf{22.916667\%}$$

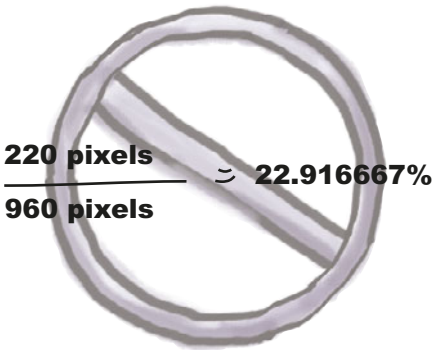
Co o tym sądzisz? Jest w porządku, prawda?

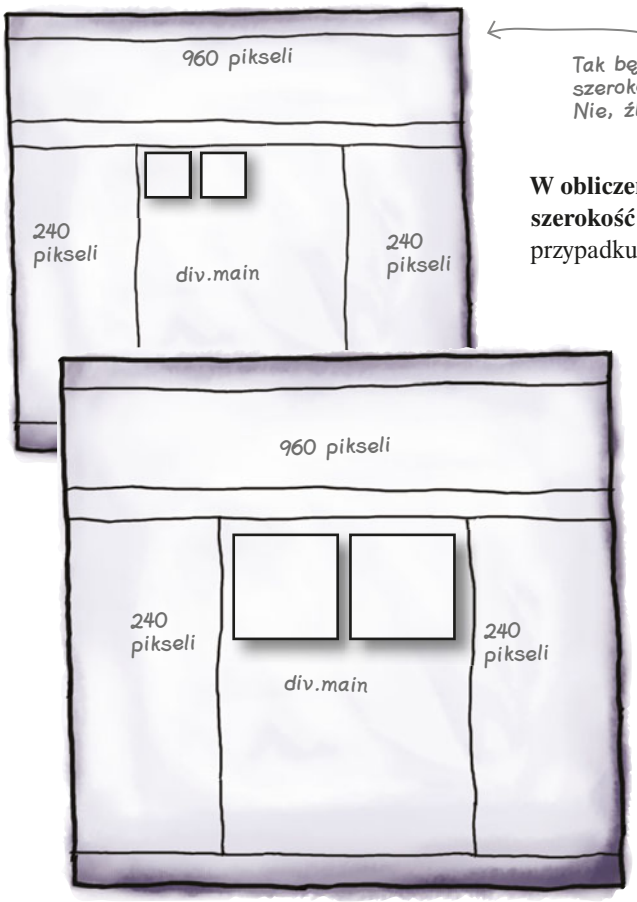
Co się stało z tymi obrazami?

Kontekst się zmienił!

Jeżeli ustalimy, by obrazy zajęły 22,916667% szerokości, zajmą właśnie tyle — 22,916667% szerokości *zawierającego je elementu*.

Te obrazy nie są umieszczone w elemencie `body` (o szerokości 100% lub 960 pikseli w projekcie, na którym bazujemy), tylko w bloku `div#main`, którego szerokość jest równa w przybliżeniu 460 pikseli (czyli, mówiąc językiem proporcjonalności, 47,91667% z 960 pikseli). Zatem tak naprawdę ustaliliśmy, że obrazy zajmują mniej niż 23% szerokości równej 460 pikseli, a to zdecydowanie za mało!





Tak będzie wyglądał układ strony, jeśli ustalimy szerokość elementów img.label na 22,916667%. Nie, źle!

Obrazy mają teraz szerokość równą niecałe 50% szerokości bloku div#main. Dużo lepiej!

W obliczeniach powinniśmy założyć, że kontekstem jest szerokość elementu zawierającego obrazy, czyli w tym przypadku uwzględniamy wartość 460 pikseli.

Zrób to sam!

220 pikseli / 460 pikseli ≈ 47.826087%

Nowy kontekst: szerokość elementu div#main, czyli elementu zawierającego.

Definiujemy szerokość obrazów w procentach? Dzięki temu uda nam się rozwiązać jeden z problemów z mobilnym układem. Pamiętasz to zdjęcie, które było za duże i psuło układ strony? Naprawimy to, stosując rozwiązanie podobne do powyższego!

Płynne obrazy

I media!

W tej niepozornej regule drzemie prawdziwa moc:

Tadam!

```
img, object {
  max-width: 100%;
}
```

Płynne obrazy i media, podobnie jak płynne układy, skalują się proporcjonalnie wraz z całym układem.

Dzięki tej małej wstawce dbamy o to, by obrazy i osadzone media nie były szersze niż zawierające je elementy. Ponieważ ich szerokość jest ograniczona do 100% — 100% szerokości zawierającego je elementu — słuchają swoich „rodziców” i nie próbują wyjść poza ich granice. Fajnie!

Ostatnie pożegnanie

Doskonałe rozwiązania wymagają zazwyczaj jakichś poświęceń. Jeśli chcemy zastosować płynne techniki na obrazach i mediach, musimy zapomnieć o naszych starych przyjaciołach: atrybutach `width` i `height`.

Przedstawiona powyżej reguła CSS przesłoni atrybut `width`, ale nie wpłynie na atrybut `height`. To oznacza, że jeśli zdefiniujemy oba atrybuty (`width` i `height`), obraz będzie skalowany w poziomie, ale nie w pionie. Uzyskamy więc brzydkie rozciągnięte lub ściśnięte obrazy o złych proporcjach.

Na szczęście są na to sposoby. Co prawda całkowite usunięcie atrybutów `width` i `height` nie jest najlepszym wyjściem, ale na razie musimy się ich pozbyć.

Obejrzyj to!

Technika płynnych obrazów nie jest lekiem na całe zło.

To, że na węższych ekranach obraz jest odpowiednio pomniejszany, nie oznacza, że tak naprawdę staje się mniejszy. Na przykład 800-kilobajtowy plik JPEG pozostaje 800-kilobajtowym plikiem JPEG, nawet jeśli zmieścił się w 120-pikselowej kolumnie.

W rozdziale 2. opowiemy o technikach umożliwiających dostarczanie odpowiednich obrazów dla różnych urządzeń i przeglądarek, dzięki którym można uniknąć marnotrawienia przepustowości łącza i mocy procesora (wymaganej do właściwego przeskalowania obrazu).

Mimo to opisana technika ma duże możliwości i z całą pewnością powinieneś mieć ją w swoim arsenale.

Czy to już odpowiedni czas?

Poczyniliśmy już postępy na drodze do urzeczywistnienia wrażliwych projektów, które dopasowują się do możliwości urządzeń. Zostało jeszcze jednak kilka spraw do załatwienia:

Wciąż musimy zmodyfikować mobilny CSS (na razie zajęliśmy się tylko desktopowym).

- ☐ Trzeba przekształcić układ zwymiarowany w pikselach na płynny, w którym są określane proporcjonalne szerokości, a nie stałe.
- ☐ Domyślny rozmiar fontu musimy ustawić na 100%, aby było możliwe proporcjonalne skalowanie.
- ☐ Musimy naprawić niedziałające wideo z YouTube'a.
- ☑ Musimy poprawić zbyt szerokie zdjęcie.

Zrobiliśmy to dzięki zastosowaniu techniki płynnych obrazów.

Pucujemy mobilny CSS

Jest tylko kilka reguł mobilnego CSS, które trzeba przekształcić na płynną postać.

```
@media screen and (max-width: 480px) {
    body, .header, .footer, .navigation
    {
        width: 320px;
    }
.column {
    margin: 1em 0;
    border-bottom: 1px dashed #7b96bc;
}
.navigation ul li {
    width: 106.6667px;
}
#visit, #points, #main {
    width: 320px;
}
```

dotychczasowa wersja (na sztywno)

```
@media screen and (max-width: 480px) {
    body, .header, .footer, .navigation {
        width: 100%;
    }
.column {
    margin: 1em 0;
    border-bottom: 1px dashed #7b96bc;
}
.navigation ul li {
    width: 33.333333%;
}
#visit, #points, #main {
    width: 100%;
}
```

No proszę, te dwie reguły są identyczne z tymi, które ustaliliśmy dla wersji desktopowej (patrz strona 30). W związku z tym, żeby ich nie dublować, lepiej je przenieść do wspólnej części arkusza.

zaktualizowana wersja (płynna)

Diabeł tkwi w szczegółach

Teraz zajmiemy się rozwiązaniem kilku drobnych problemów, dzięki czemu nasz projekt będzie w pełni zgodny z podejściem RWD.

Elastyczne definicje rozmiaru czcionek

Udało nam się stworzyć adaptujący się układ strony, ale rozmiary czcionek wciąż są ustawione na sztywno. Dla szerokości definiowanych w pikselach płynnym odpowiednikiem są procenty, a dla rozmiaru czcionek odpowiednikiem są proporcjonalne jednostki **em**. Michał zastosował je w swoim projekcie, więc wystarczy dodać do elementu `<body>` następującą definicję:

```
body {
    background: #f9f3e9;
    color: #594846;
    font: 100% "Adobe Caslon Pro", "Georgia",
    "Times New Roman", serif;
}
```

Domyślny rozmiar fontu ustawiamy na wszelki wypadek. Ustalamy w ten sposób punkt odniesienia, względem którego określamy rozmiar czcionek na stronie. To nic wielkiego, ale pozwala zachować porządek.

Zbliżenie na rozmiar fontu

Ta definicja w regule dla elementu `<body>` ustala bazowy rozmiar fontu na 100%. Ale co to oznacza? Oto krótka (ale i *uproszczona*) zasada przeliczania jednostek:

1em = 100% ≈ 12pt ≈ 16px

Pamiętaj jednak, że chcemy dopasować zawartość strony do środowiska użytkownika. Jeśli użytkownik sam zmienił rozmiar fontu w przeglądarce, 100% będzie odpowiadało innemu rozmiarowi bezwzględnemu.

Pamiętaj również o tym, że fonty na urządzeniach mobilnych rządzą się swoimi prawami, więc w niektórych sytuacjach `1em` może odpowiadać różnym (czasem nawet bardzo różnym) wielkościom wyrażonym w punktach czy pikselach.

Naprawiamy wideo z YouTube'a

Wiele urządzeń mobilnych nie oferuje wsparcia dla Flasha. Znajdujący się na stronie kod osadzający wideo z YouTube'a jest zdecydowanie nieaktualny. Obecnie stosuje się rozwiązanie wykorzystujące ramkę `iframe`, które działa na iPhonie (i innych nowszych urządzeniach). W pliku *index.html* musimy zmienić fragment odpowiedzialny za osadzanie wideo.

```
<object width="230" height="179" type="application/x-
shockwave-flash" data="http://www.youtube.com/v/0-
jOEAufDQ4?fs=1&amp;hl=en_US&amp;rel=0"><embed src=...
/></object>
```

Zamiast tego kodu (ograniczonego do Flasha)...

Ta technika nie ogranicza się do osadzania Flasha! Inne media też mogą się stać płynne.

...użyj tego!

```
<iframe src="http://www.youtube.com/embed/0-jOEAufDQ4"
style="max-width:100%"></iframe>
```

Kod osadzający wideo udostępniany obecnie przez serwis YouTube zapewnia dostarczenie materiału wideo w formacie odpowiednim dla przeglądarki użytkownika. Poza tym wspiera element `video` z HTML5 zamiast Flasha, co jest istotne w przypadku wielu urządzeń mobilnych (na przykład w iPhonie). Zaprezentowany tu fragment kodu skopiowaliśmy po prostu ze strony z filmem w serwisie YouTube.

Pamiętaj, by być wrażliwym

Podobnie jak płynne skalowanie obrazu nie zmienia tak naprawdę rozmiaru pliku z obrazem, tak skalowanie mediów nie wpływa na ich faktyczny rozmiar. Od Ciebie zależy, czy dostarczenie wideo (lub innych multimediów) do urządzeń mobilnych jest warte tych danych, które trzeba pobrać, i mocy procesora, którą trzeba przeznaczyć na ich odtworzenie.

Długie ćwiczenie

Do dzieła! Wprowadź wszystkie opisane zmiany, by witryna pubu Pod Paradnym Morsem stała się przyjazna dla urządzeń mobilnych i była zgodna z podejściem RWD.

Zmodyfikuj plik styles.css

1. Zmień reguły w zapytaniu o media dla urządzeń o mniejszych ekranach. Zamień jednostki stosowane w definicjach szerokości na proporcjonalne (strona 34).

2. Odszukaj te reguły, które — po wprowadzeniu proporcjonalnych szerokości — stały się identyczne dla wersji mobilnej i desktopowej (strona 34). Usuń je z sekcji zapytań o media i umieść wśród wspólnych reguł.

3. Dodaj reguły wprowadzające technikę płynnych obrazów (i mediów), przedstawione na stronie 33.

4. Zaktualizuj reguły dla elementu `<body>`, tak by były stosowane proporcjonalne rozmiary czcionek (strona 35).

Zmodyfikuj plik index.html

5. Zamień kod osadzający wideo z wersji tylko dla Flasha na wersję z elementem `iframe` przedstawioną na stronie 35.

Sprawdź to!

Spróbuj zmienić rozmiar okna przeglądarki i obserwuj, jak dopasowuje się do niego zawartość strony.

6. Zapisz zmiany i otwórz plik *index.html* w dowolnej przeglądarce.

Nie istnieją głupie pytania

P: W porządku, w arkuszu stylów widziałem już jednostki em, ale nie do końca rozumiem, o co w tym chodzi. Dlaczego je stosujemy?

O: Wielkość `1em` odpowiada bieżącemu rozmiarowi fontu w aktualnym kontekście. Trzeba przyznać, że taka definicja nie jest zbyt porywająca. Ale zaczyna być to interesujące, kiedy definiujesz rozmiar fontu względem tej jednostki. Jeśli więc ustawisz rozmiar elementu `<h1>` na `1.5em`, będzie to stanowiło 150% domyślnego rozmiaru fontu w obrębie zawierającego go elementu.

Do wyznaczania rozmiarów fontu możesz użyć tej samej zależności co do elementów strony. Odpowiednikiem 18-punktowego fontu w kontekście, w którym bazą jest 16-pikselowy font, będzie: `18/16 = 1.125em` (element docelowy/kontekst = wynik).

P: Moment, a dlaczego mówimy o `1.125 em`, a nie `112.5%`?

O: Głównie ze względu na to, że tak się przyjęło, co się wiąże również z czytelnością, ale też dlatego, że jednostki em trochę lepiej są obsługiwane na różnych platformach. Zwyczajowo szerokości elementów blokowych określa się w procentach, a rozmiary fontu w jednostkach em. W zdecydowanej większości przypadków procenty są wymienne z jednostkami em. Jednym z wyjątków jest określenie atrybutu `font-size` dla elementu `<body>` na `100%`.

P: Czy zapytania o media, poza dostosowaniem arkuszy stylów do urządzeń mobilnych, znajdują jeszcze jakieś zastosowanie?

O: Oczywiście. Jeśli uznamy, że urządzenia mobilne znajdują się na jednym końcu skali wielkości ekranu, to nowsze monitory i telewizory mogą się znaleźć na drugim. Niekiedy mogłoby być uzasadnione dopasowanie układu do większych szerokości ekranu, na przykład poprzez dodanie kolejnych kolumn.

P: Z czym jest związane pojawienie się tej pustej przestrzeni z prawej strony ekranu widocznej na rysunku na stronie 7?

O: To jest cecha iPhone'ów. Gdy w mobilnej przeglądarce Safari wyświetla się przeskalowane strony (by wyglądały tak jak w przeglądarkach desktopowych), odgórnie ustalana jest szerokość strony równa 980 pikseli. Przed modyfikacjami strona pubu Pod Paradnym Morsem miała szerokość 960 pikseli, co powoduje pojawienie się 20-pikselowej pustej przestrzeni z prawej strony.

P: Jakie znaczenie ma to „3” w nazwie „zapytania o media w CSS3”?

O: Jest kilka wersji CSS. Można by powiedzieć, że są trzy, ale sytuacja jest trochę bardziej skomplikowana. Wersją najlepiej znaną przez projektantów i twórców stron była CSS2, wydana w 1998 roku jako rekomendacja. I właśnie w tej wersji zostały wprowadzone zapytania o media.

CSS3 to zupełnie nowa jakość i trudno tę wersję porównać z wcześniejszymi. Jest silnie zmodularyzowana (składa się z około czterdziestu różnych modułów) i nie jest opisana jedną, złożoną specyfikacją. Na szczęście moduł zapytań o media jest jednym z tych, które są prawie w pełni kompletne i stabilne.

P: Skoro prace nad standardem CSS3 nie zostały jeszcze zakończone, jak wygląda sprawa wsparcia go w przeglądarkach?

O: Tak jak już wspomnieliśmy, panuje tu drobny chaos. Niektóre elementy CSS3 są wspierane w większości przeglądarek, ale są i takie, których stopień realizacji pozostawia wiele do życzenia (o czym przekonasz się niebawem).

P: Dlaczego w mobilnym układzie zawartość prawej kolumny jest wyświetlana przed główną?

O: W kodzie HTML blok `div#points` znajduje się przed blokiem `div#main`. W desktopowym układzie do pozycjonowania kolumn jest stosowane opływanie, w związku z czym kolumna `#points` jest wyświetlana z prawej strony bloku `#main`. W mobilnym układzie opływanie nie jest stosowane, więc zawartość jest wyświetlana zgodnie z kolejnością występowania w kodzie HTML.

P: W zapytaniach o media pominęliście instrukcję `@import`. Czy mimo to mogę jej użyć?

O: Instrukcja `@import` nie cieszy się dużym uznaniem. Właśnie z tego względu nawet o niej nie wspomnieliśmy. Ale oczywiście możesz jej użyć w zapytaniach o media do zaimportowania odpowiednich arkuszy stylów.

Rozwiązanie długiego ćwiczenia

Przeanalizujmy wynikowy plik arkusza stylów i zobaczmy, co się zmieniło.

To znajduje się na samym początku pliku CSS.

```
body {
    background: #f9f3e9;
    color: #594846;
    font: 100% "Adobe Caslon Pro", Georgia,
    "Times New Roman", serif;
}
```

Aby nie zaciemniać obrazu, pomijamy ten fragment.

(Wspólne reguły dla typografii, kolorów, krawędzi itp., czyli niestrukturalne style...)

Wspólne style strukturalne

```
.header, .footer {
    clear: both;
}
.header {
    background:url(images/w.png) no-repeat;
    height: 200px;
}
.navigation {
    min-height: 25px;
}
img, object {
    max-width: 100%;
}
.navigation ul li {
    width: 33.333%;
}
.header, .footer, .navigation {
    width: 100%;
}
```

A teraz strukturalny CSS powiązany z wymiarami ekranu.

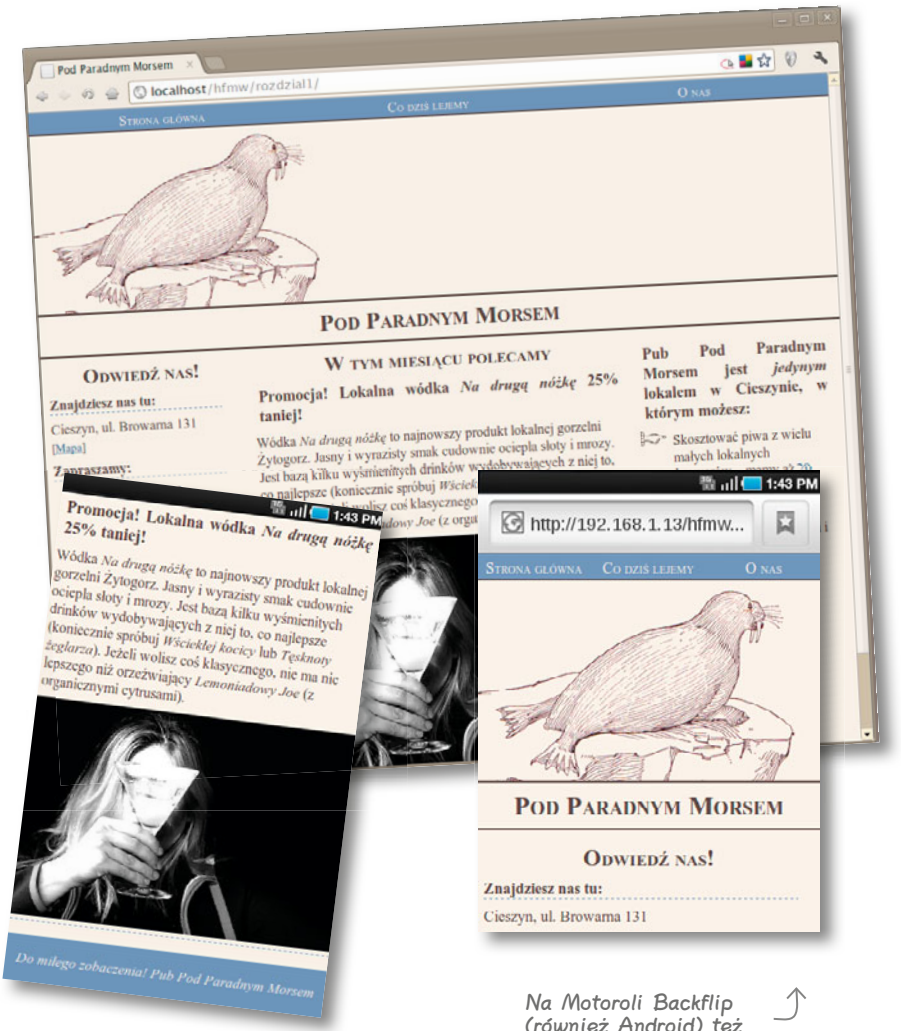
Strukturalny CSS dla większych ekranów (na przykład w przeglądarkach desktopowych)

```
@media screen and (min-width:481px) {
    .column {
        margin: 10px 1.04166667% 0 0;
    }
    #visit {
        width: 25%;
        float: left;
    }
    #points {
        width: 25%;
        float: right;
    }
    #main {
        margin: 10px 27.0833333% 0 26.0416667%;
    }
}
```

Strukturalny CSS dla mniejszych ekranów (na przykład w przeglądarkach mobilnych)

```
@media screen and (max-width:480px) {
    .column {
        margin: 1em 0;
        border-bottom: 1px dashed #7b96bc;
    }
    #visit, #points, #main {
        width: 100%;
    }
}
```

Oto strona w stylu RWD!



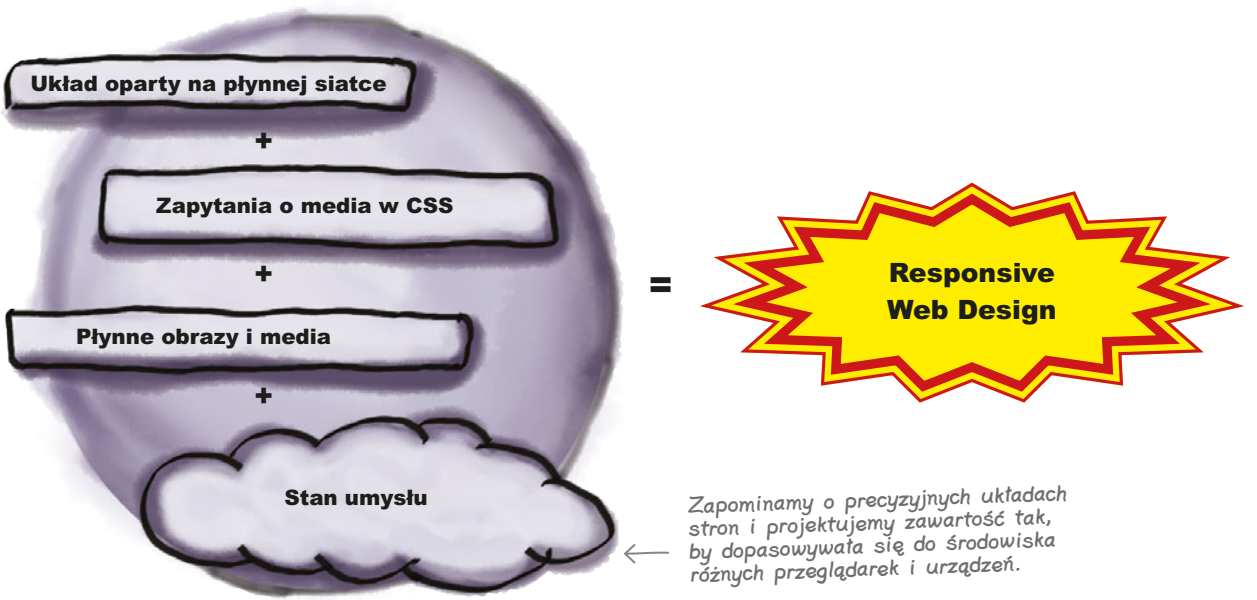
Dzięki technice płynnych obrazów na smartfonie Nexus S z Androidem zdjęcie jest wyświetlane prawidłowo.

Na Motoroli Backflip (również Android) też wszystko wygląda dobrze.

WYSIL SZARE KOMÓRKI

W arkuszu zoptymalizowanym pod kątem urządzeń mobilnych stosujemy ten sam obraz dla tła nagłówka, chociaż bywa, że zostaje on przycięty. W jaki sposób mógłbyś skorzystać z technik RWD, by w takich przypadkach był stosowany *inny* obraz (aby nie obciążać łącza i zwiększyć wydajność)?

Podejście Responsive Web Design to również stan umysłu



CELNE SPOSTRZEŻENIA

- Mobilne technologie webowe to w pewnym sensie dziki zachód — pełno w nich niespodzianek i przygód. Krajobraz mobilnych przeglądarek jest różnorodny, co czasami doprowadza do furii.
- To, że możemy zastosować ten sam układ dla przeglądarek mobilnych co dla desktopowych, nie oznacza, że powinniśmy to robić.
- Podejście **Responsive Web Design** (RWD) to zbiór zaleceń i technik, dzięki którym to strony (i inne treści internetowe) dopasowują się do środowiska użytkownika, a nie użytkownik do nich (godząc się na oglądanie źle sformatowanych stron).
- RWD to połączenie **zapytań o media w CSS3** oraz **układów opartych na płynnych siatkach** i **płynnych obrazach**. Uogólniając, jest to sposób myślenia o układzie i zawartości.
- **Zapytania o media w CSS3** pozwalają na wybiórcze stosowanie reguł CSS w różnych środowiskach użytkowników na podstawie bieżących wartości wybranych cech mediów.
- **Typy mediów** (na przykład: `screen`, `print`, `projection`) mają określone **cechy mediów** (takie jak: `width`, `color`, `monochrome` czy `orientation`). To właśnie za pomocą tych cech definiujemy zapytania.
- **Zapytanie o medium w CSS** jest wyrażeniem logicznym. Jeśli jego wartość to TRUE, zawarte w nim reguły są stosowane do dokumentu.
- W **płynnych układach** stosuje się proporcjonalne wartości szerokości, a nie ustalone na sztywno, tak by zawartość strony skalowała się i przepływała wraz ze zmianą szerokości okna przeglądarki.
- **Płynne obrazy** to technika, dzięki której zbyt duże obrazy (i media) trzymają się granic elementów nadrzędnych, nawet jeśli te elementy są węższe niż obraz. Obrazy są pomniejszane wraz z elementem nadrzędnym.
- Ustalenie domyślnej wartości atrybutu `font-size` elementu `<body>` i definiowanie rozmiaru czcionek w jednostkach em lub procentach pozwala na zachowanie płynnych rozmiarów czcionek.

Koncepcja Mobile First w podejściu Responsive Web Design

Oto śliczna mobilna witryna. Ale nie oceniaj jej tylko po pozorach. Pod tą piękną powłoką znajdziesz bowiem coś zupełnie innego. Być może wygląda jak mobilna witryna, ale to wciąż zwykła, desktopowa witryna, z tym że przebrana w mobilne ciuszki. Jeśli chcesz, żeby na urządzeniach mobilnych chodziła jak dobrze naoliwiona maszynka, musisz zastosować zasadę **Mobile First**. Jednak najpierw musimy przeprowadzić sekcję obecnej witryny, by odnaleźć ukrywający się w jej wnętrzu desktopowy szkielet. Następnie gruntownie posprzątamy i zaczniemy pracować na świeżo, zgodnie ze strategią **stopniowego ulepszania**, zaczynając od budowania podstawowych elementów, a kończąc na bogatej wersji desktopowej. Gdy skończymy, nasza strona będzie zoptymalizowana pod każdą możliwą rozdzielczość ekranu.

Gdy właśnie zamierzałeś zacząć świętować swój sukces...

Michał wpadł w panikę. Jako twórca stron internetowych zwykle opiera się pokusie majstrowania przy swoim dziele, ale tym razem postanowił jednak wprowadzić kilka zmian. Niestety skończyło się to klęską, więc pilnie potrzebuje naszej pomocy.

Na stronie Co dziś lejemy Michał wstawił etykiety wszystkich oferowanych piw. Nie zmienił żadnego fragmentu istniejącego kodu — dodał tylko kilka znaczników wstawiających obrazy. Od tego momentu strona ładuje się bardzo wolno na wszystkich telefonach i innych urządzeniach mobilnych. Na tyle wolno, że klienci zaczęli narzekać.

Stronę Co dziś lejemy możesz znaleźć pod adresem ftp://ftp.helion.pl/online/inne/hfmw/r02/ontap.html.

Czy to naprawdę jest problem? Skąd to wiadomo?

Kuba: Biedny Michał. Niestety narobił sobie kłopotu.

Łukasz: Zgadzam się, ale nie sądzę, żeby w tym przypadku popełnił jakiś błąd. Telefony zwykle mają dostęp do powolnego łącza internetowego, a poza tym mają słabsze procesory. To jasne, że strona będzie się ładowała powoli.

Kuba: To brzmi całkiem rozsądnie, ale faktem jest, że strona ładuje się zdecydowanie wolniej, niż Michał się spodziewał. Ostatnio mi powiedział, że nawet kiedy korzysta z wi-fi, tempo jest nie do zaakceptowania. A on przecież sprawdzał to na całkiem nowym i wydajnym smartfonie.

Przemek: Wygląda na to, że musimy się temu przyjrzeć i sprawdzić, co tak spowalnia ładowanie.

Łukasz: A jak chcesz to sprawdzić? To przecież może być wina sieci albo tysiąca innych elementów pośredniczących między telefonem a serwerem.

Przemek: Korzystałem kiedyś z dodatku do Firefoksa, który pomaga analizować wydajność strony. Może użyjemy czegoś takiego?

Kuba: Sprawdźmy to cudo!

Łukasz: A uda nam się zainstalować ten dodatek w przeglądarce mobilnej?

Przemek: W tym problem, że nie da się go zainstalować na telefonie. Większość moich ulubionych narzędzi to dodatki do przeglądarki. A jak bez nich sprawdzić, co się dzieje z tą stroną?

Kuba: Kiedyś Kaśka pokazała mi, jak obserwować żądania w sieci bezprzewodowej za pomocą jakiegoś narzędzia zainstalowanego na routerze. Moglibyśmy z czegoś takiego skorzystać i zobaczyć, jakie dane pobiera nasz telefon.

Łukasz: To świetny pomysł, ale chyba mam lepszy. Co powiecie na użycie *serwera proxy*? Jego działanie jest bardzo podobne do tego, co pokazywała ci Kasia, ale służy właśnie do tego, czym się musimy zająć. Jeśli z takim serwerem połączymy telefon, będziemy mogli zobaczyć wszystkie wykonywane przez niego żądania.

Dodatek YSlow pozwala przeanalizować wydajność strony na podstawie jej budowy. Możesz go pobrać z yhoo.it/yslow. Dla każdego twórcy stron, a zwłaszcza stron mobilnych wydajność jest jedną z ważniejszych spraw.

Skoro o tym mowa, poświęcimy trochę czasu na przyjrzenie się sprawom wydajności. Nie martw się — już wkrótce zajmiemy się koncepcją Mobile First w podejściu Responsive Web Design, a uwierz, że o wydajności mówimy nie bez przyczyny.

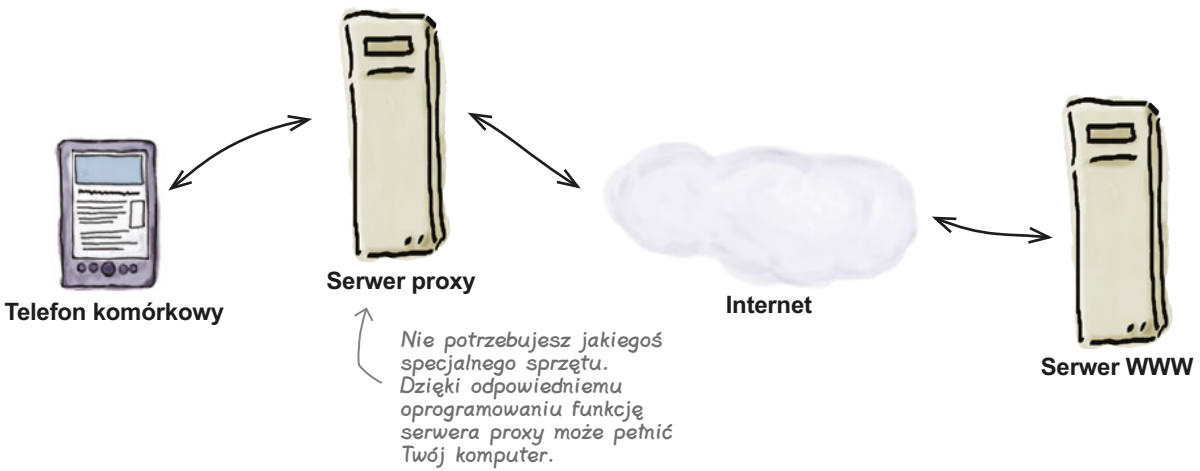
Czy może pani przyjąć moje zamówienie?

Pewnego razu znany zawodnik w przeddzień walki ze swoim odwiecznym rywalem nabawił się nieprzyjemnego zatrucia pokarmowego. Policja podejrzewała nieczyste zagranie ze strony tego drugiego, więc zaangażowała w śledztwo doświadczonego detektywa. Detektyw szybko przeszedł do rzeczy i przesłuchał główną podejrzaną — kelnerkę.

Kelnerka zeznała, że przyjęła zamówienie, zapisała je w notesiku i zaniosła kucharzowi. Kiedy posiłek był gotowy, wzięła go z kuchni i zaniosła do stolika gościa. Widziała wszystko i nic nie umknęło jej uwadze.

Przez większość czasu spędzonego w sieci rozmawiasz bezpośrednio z kucharzem. Z perspektywy stron, które oglądasz, nie ma nic pomiędzy Twoją przeglądarką a serwerem.

Ale możemy skorzystać z serwera pośredniczącego, czyli *proxy*, który pełni funkcję kelnerki z przytoczonej opowieści — rejestruje zarówno to, czego żąda klient, jak i to, co jest mu dostarczane. Dzięki temu możemy sami pobawić się w detektywa i zbadać, co się tak naprawdę stało.



Czy w postawieniu serwera proxy pomoże mi jakiś pośrednik?

Jeśli na poważnie zajmujesz się projektowaniem mobilnych stron, najprawdopodobniej warto, byś poświęcił chwilę na bliższe zapoznanie się z serwerami proxy. Pamiętaj, że to najlepszy sposób na sprawdzenie, co dzieje się między telefonem a serwerem.

Zła wiadomość jest taka, że postawienie i skonfigurowanie serwera proxy nie jest takie proste. Ale mamy też dobrą wiadomość — mili ludzie z firmy Blaze, zajmującej się wydajnością rozwiązań mobilnych, udostępnili bezpłatny serwis, który pomoże nam rozwiązać nasz problem.

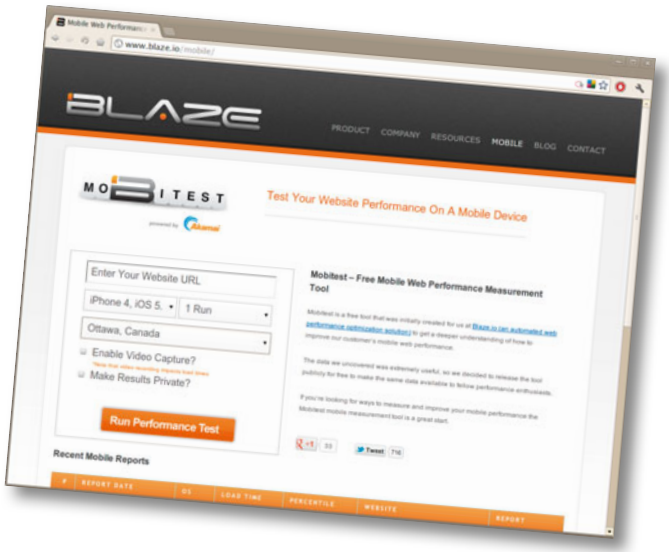
Co zrobić, gdy nie śmiga?

Firma Blaze udostępnia bezpłatny test wydajności dla iPhone'ów i telefonów z Androidem — Mobitest, który znajdziesz pod adresem *www.blaze.io/mobile*.

Mobitest działa jak serwer proxy. Podajesz adres URL strony, którą chcesz przetestować, oraz urządzenie mobilne. Tak zdefiniowany test jest przekazywany do kolejki testów dla danego urządzenia.

Gdy żądany telefon jest już dostępny, rozpoczyna się test, podczas którego jest analizowany cały ruch w sieci między telefonem i serwerem.

To narzędzie oferuje też ciekawą funkcję polegającą na nagrywaniu ekranu urządzenia podczas ładowania strony, dzięki czemu możesz zobaczyć, jak ładowanie Twojej strony będzie wyglądało na różnych telefonach.



Jazda próbna

Gotowy na małe dochodzenie? Sprawdź, czemu strona Co dziś lejemy ładuje się tak wolno.

1. **Stronę Co dziś lejemy przetestuj w serwisie www.blaze.io/mobile.**
   Strona jest dostępna pod adresem *ftp://ftp.helion.pl/online/inne/hfmw/r02/2/ontap.html*.

2. **Zwróć uwagę na czas ładowania i rozmiar strony.**
   Czas ładowania (ang. *load time*) wskazuje, jak długo trwało załadowanie strony w urządzeniu podczas przeprowadzania testu. Rozmiar strony (ang. *page size*) to całkowity rozmiar wszystkich zasobów powiązanych ze stroną, włączając w to pliki HTML, CSS, JavaScript, obrazy, fonty itd.

3. **Przeprowadź test na dwóch urządzeniach i porównaj wyniki.**
   Wydajność zależy nie tylko od sieci. Wpływ może mieć samo urządzenie oraz to, jak szybko zainstalowana na nim przeglądarka pobiera i wyświetla strony.

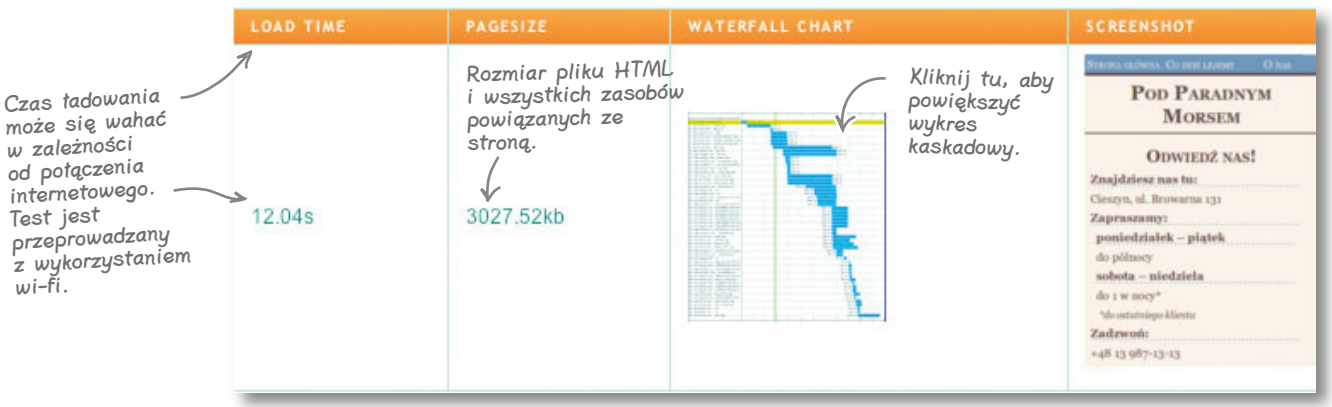
Nie ma co się oszukiwać, to jest WIELKA strona

Fatalnie! Z raportu wynika, że strona ma blisko 3 megabajty. Mogłaby się ewentualnie sprawdzić jako desktopowa strona, ale na telefonie osiąga tempo leniwego żółwia.

Tak naprawdę to jest zdecydowanie za duża nawet dla komputerów stacjonarnych. Z tego, że mamy duży ekran, nie wynika wcale, że mamy szybkie połączenie z internetem. Wydajność jest istotna w każdym przypadku.

Nic więc dziwnego, że klienci Michała żalą się na szybkość ładowania strony — załadowanie jej w iPhonie zajmuje ponad 10 sekund.

Strona serwisu Mobitest z wynikami testu dla iPhone'a.



Co to jest wykres kaskadowy? Rzuciłam na niego okiem, ale nie mam pojęcia, jak się z niego dowiedzieć, co na tej stronie jest tak duże. Czy ten wykres w ogóle może się przydać?

Wyniki testów wydajności zwykle przedstawia się na wykresach kaskadowych.

Na wykresie widać pliki pobierane przez przeglądarkę podczas ładowania strony. Słupki reprezentują czas potrzebny na załadowanie kolejnych zasobów. Zasoby są umieszczone w kolejności, w jakiej przeglądarka zgłasza żądania do serwera.

Nie staraj się jednak wywnioskować wszystkiego na podstawie tego wykresu. Nie ma tu szczegółów, które są nam niezbędne do rozwiązania problemu. Pokażemy Ci bardziej przydatne narzędzie.

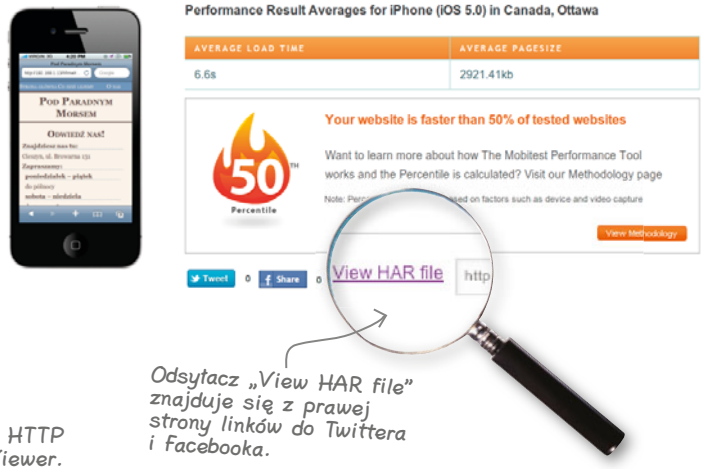
Dobrodziejstwa pliku HAR

Na stronie z wynikami testu Mobitest jest pewien skarb ukryty pod odsyłaczem ***View HAR file***.

Po kliknięciu tego odsyłacza przejdziesz na kolejną stronę zatytułowaną *HTTP Archive Viewer*, na której zobaczysz wykres kaskadowy o wiele bardziej szczegółowy niż poprzedni.

Na wykresie widać każdy zasób pobrany przez przeglądarkę wraz z dodatkowymi informacjami, których nie było na wykresie przedstawionym w raporcie testu Mobitest.

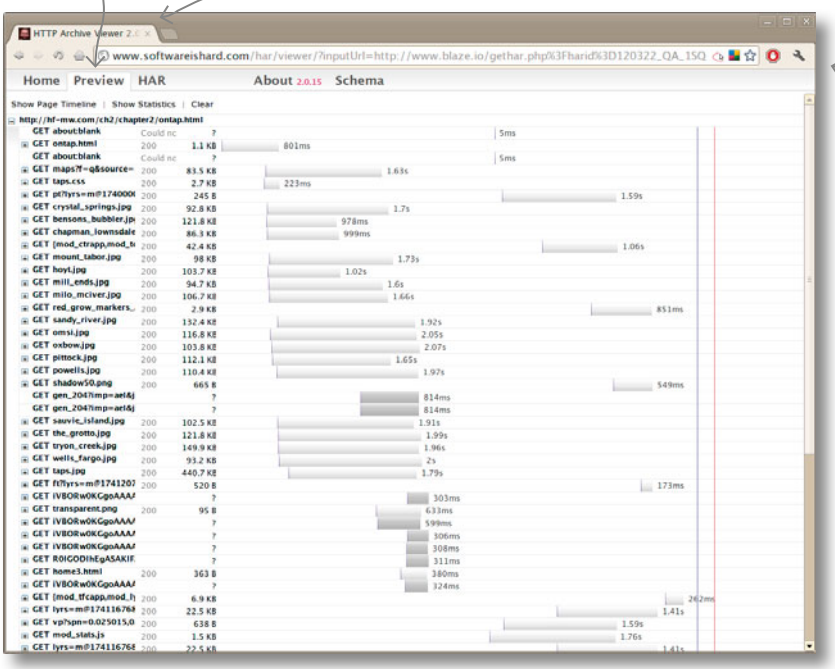
Górna część strony z raportem testu Mobitest.



Odsyłacz „View HAR file" znajduje się z prawej strony linków do Twittera i Facebooka.

W zakładce Preview jest wyświetlany wykres kaskadowy dla testowanej strony. Jest on generowany na podstawie pliku HAR.

Narzędzie HTTP Archive Viewer.



Plik HAR może posłużyć do wygenerowania wykresu kaskadowego. Więcej na temat HAR możesz znaleźć na stronie httparchive.org.

Skrót HAR pochodzi od HTTP Archive. Jest to specyfikacja opisująca standard rejestrowania tego, co dzieje się w chwili zgłoszenia przez przeglądarkę żądania strony z serwera.

WYSIL SZARE KOMÓRKI

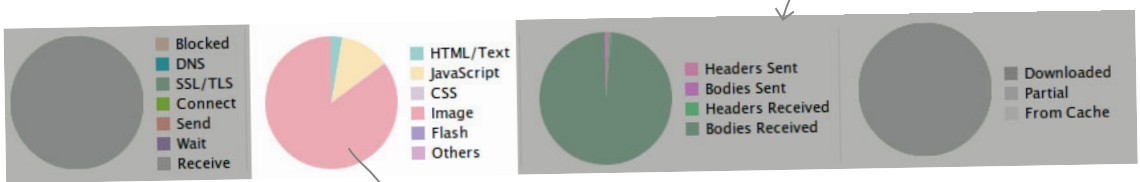
Czy podczas oglądania wykresu rzuciły Ci się w oczy jakieś podejrzane pliki?

Widok z lotu ptaka — opcja Show Statistics

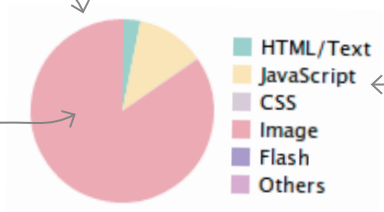
Na wykresie kaskadowym wygenerowanym z pliku HAR widać pobrane pliki, kod odpowiedzi serwera oraz czas wymagany do ich pobrania. Zanim jednak zaczniesz zgłębiać wykres, rzuć okiem na bardziej ogólne statystyki.

Na stronie *HTTP Archive Viewer* kliknij odsyłacz *Show Statistics*. Twoim oczom ukażą się cztery wykresy kołowe. Najistotniejszym z nich z naszego punktu widzenia jest drugi od lewej, który przedstawia rozkład czasów pobierania ze względu na typ plików. Po najechaniu wskaźnikiem myszy każdego z wyszczególnionych typów plików zostaje wyświetlona informacja o ich liczbie oraz całkowitym rozmiarze.

Aby zobaczyć takie wykresy kołowe, na stronie narzędzia HTTP Archive Viewer musisz kliknąć Show Statistics.



A niech to! Same obrazy (a jest ich 35) ważą aż 2,4 MB!

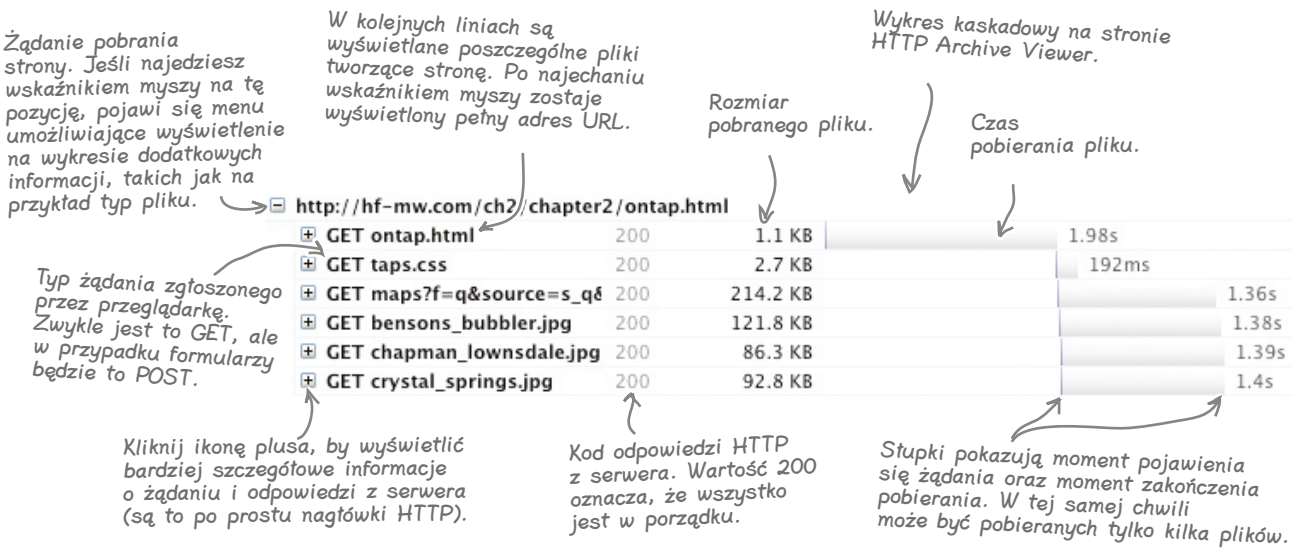


Aż osiem plików JavaScript o łącznym rozmiarze 351 kB!

A więc to jest przyczyną spowolnienia strony. Na stronie nie ma nic, co korzystałoby ze skryptów JavaScript, a mimo to jest pobieranych aż osiem takich plików. No i najważniejsze — prawie 2,4 MB obrazków! Musimy sprawdzić, dlaczego obrazki i skrypty są aż tak duże.

Wytęż wzrok i znajdź zawalidrogę

Najwyższy czas, by bliżej przyjrzeć się wykresowi kaskadowemu i dowiedzieć się czegoś więcej o tych dużych obrazach i skryptach. Oto klucz ułatwiający odczytywanie wykresu.



Liczba połączeń między przeglądarką i serwerem może przytłaczać, ale nie przejmuj się tym zanadto. Twoim zadaniem jest znalezienie odpowiedzi na tylko dwa pytania: *które zasoby są największe?* oraz *skąd pochodzą pliki JavaScript?*

Ćwiczenie

Przejrzyj wykres kaskadowy dla strony Co dziś lejemy. Znajdź pięć największych plików i sprawdź je. Dla każdego z nich odpowiedz na te trzy pytania:

1. **Jakiego typu jest ten plik?**
2. **Z jakiej domeny pochodzi plik?**
3. **Jeśli jest to plik obrazu, jaką ma szerokość i wysokość?**
   Wskazówka: aby sprawdzić wymiary obrazu, będziesz musiał otworzyć go w nowym oknie przeglądarki lub pobrać go na dysk.

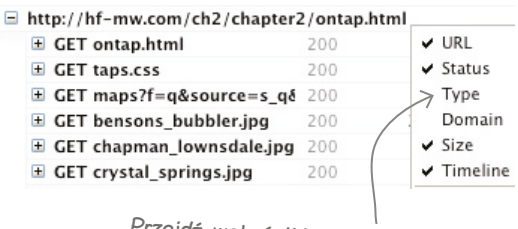
Czy te informacje naprowadziły Cię na możliwy sposób przyspieszenia działania strony? Co możesz zrobić?

Znalazłeś przyczynę problemu ze stroną? Omówimy to wspólnie.

Rozwiązanie ćwiczenia

1 **Jakiego typu jest ten plik?**

Typ pliku możesz sprawdzić po najechaniu wskaźnikiem myszy na wybraną nazwę pliku — pojawią się wtedy pełny adres URL oraz rozszerzenie. Możesz też dodać kolumnę, w której są wyświetlane informacje o typie, dzięki czemu dużo szybciej uda Ci się przejrzeć całą listę plików.



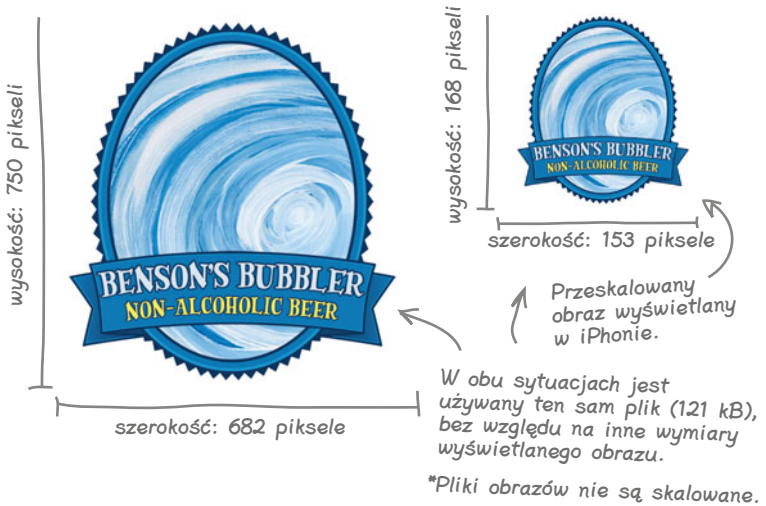
2 **Z jakiej domeny pochodzi plik?**

Przechodzimy dalej — sprawdź, skąd jest pobierany duży (174,8 kB) plik JavaScript.

Maps.gstatic.com to domena usługi Mapy Google. Przeglądarka pobiera skrypt mapy, która nie jest wyświetlana w urządzeniach mobilnych. A z tym skryptem jest powiązanych wiele innych tajemniczych plików.

3 **Jeśli jest to plik obrazu, jaką ma szerokość i wysokość?**

Znajdź plik obrazu o największym rozmiarze. Skopiuj jego adres URL i otwórz w nowym oknie przeglądarki. Nawet nie musisz sprawdzać dokładnych wartości szerokości i wysokości, ponieważ gołym okiem widać, że ten obraz jest zdecydowanie za duży i nie ma szans zmieścić się na małym ekranie.



Skąd pochodzi skrypt map Google'a?

Gdy wyświetlisz stronę Co dziś lejemy na telefonie, nie zobaczysz na niej mapy. Dlaczego w takim razie jest ładowany plik JavaScript? Aby to sprawdzić, otworzymy stronę w przeglądarce desktopowej.

To ciekawe — mapa została wyświetlona. Widocznie Michał tak ją ustawił, by pojawiała się tylko na szerszych ekranach.

Ukrywanie mapy na urządzeniach mobilnych jest uzasadnione, ponieważ jest trochę za duża jak na tak małe ekrany. Poza tym starsze telefony mogą nie być w stanie obsłużyć skomplikowanego skryptu związanego z mapą.

W jaki sposób Michał ukrył mapę?

O! Tu jest nasza mapa!

Jedna linia ładująca wszystko

Mapa jest umieszczana na stronie za pomocą ramki `iframe`. Ramka ładuje wszystkie składniki niezbędne do wyświetlenia mapy.

Zajrzyj do pliku ontap.html i znajdź ten fragment kodu.

```
<iframe id="map" width="300" height="300" frameborder="0" scrolling="no" marginheight="0"
marginwidth="0" src="http://mapy.google.pl..."></iframe>
```

Ta jedna ramka iframe ładuje aż 47 plików!

Skróciliśmy bardzo długi adres URL.

Michał ukrywa mapę za pomocą CSS

Michał poznał zasady tworzenia zapytań o media i zmodyfikował mobilny układ. W zapytaniu dodał regułę ustalającą wartość właściwości `display` ramki na `none`.

Niestety powoduje to jedynie ukrycie mapy, ale nie zapobiega pobieraniu plików.

taps.css

```
@media screen and (max-width:480px) {
  .
  .
  .
  #map {display:none;}
}
```

W tym pliku CSS jest znacznie więcej reguł.

Ramka iframe ma identyfikator „map". Ustawienie właściwości display na none pozwala ukryć ramkę z mapą Google'a.

A co zrobić z dużymi obrazami?

Obrazy z tej strony muszą zrzucić kilka kilo. Przyjrzyj się wykresowi kaskadowemu i znajdź największe obrazy oraz zastanów się, dlaczego są aż tak duże.

Największy jest plik taps.jpg — ma rozmiar 440,7 kB.

Ten ogromny plik zawiera obraz nagłówka, który przecież nie jest wyświetlany na urządzeniach mobilnych.

Racja, ale to nie oznacza, że nie jest ładowany.

Obraz *taps.jpg* został ukryty na stronie w taki sam sposób jak mapa, czyli za pomocą właściwości `display` ustawionej na `none`. Ale, co widzieliśmy już w przypadku mapy, ustawienie `display:none` nie anuluje pobierania zawartości.

```
@media screen and (max-width:480px) {
    [Tu znajdują się pozostałe reguły]
    .header {display:none;}
}
```

To jest obraz zapisany w pliku taps.jpg. Otwórz stronę Co dziś lejemy w przeglądarce desktopowej, aby zobaczyć, gdzie ten obraz jest wyświetlany.

Płynne obrazy to duże obrazy

Z wykresu kaskadowego możemy jeszcze wyczytać, że wszystkie obrazy z etykietami piwa są duże. Sprawa wygląda tak, że są to desktopowe obrazy przeskalowane na potrzeby mniejszych ekranów za pomocą techniki płynnych obrazów, którą omówiliśmy w rozdziale 1.

W sumie obrazy wszystkich szesnastu etykiet zajmują prawie 2 MB. Obmyślenie sposobu optymalizacji tych obrazów jest kluczem do przyspieszenia działania strony.

To nie jest nowy problem, ale wcześniej go nie dostrzegliśmy, ponieważ ładowaliśmy co najwyżej kilka obrazów. Teraz sprawa wygląda inaczej, ponieważ na tej stronie pobieramy aż 16 obrazów, przez co problemy techniki płynnych obrazów stały się widoczne.

Wygląda przyjaźnie, ale takie nie jest

Kuba: Ale porażka... Też myślicie, że chociaż to wygląda całkiem dobrze, coś nadal jest nie tak?

Łukasz: Przynajmniej możemy powiedzieć Michałowi, że nie popsuł swojej strony.

Przemek: Tak, każdy z nas mógł popełnić te same błędy. Problemy ujawniają się akurat na tej stronie, ale pewnie podobnie sprawa wygląda na wszystkich pozostałych.

Łukasz: No to co robimy? Tworzymy od podstaw osobną wersję strony na urządzenia mobilne? Zapominamy o podejściu Responsive Web Design?

Przemek: Spokojnie, bez paniki. Musi istnieć jakiś sposób naprawienia tej strony. Czuję, że jesteśmy całkiem blisko. Czy problemy z obrazami i JavaScriptem mają jakiś wspólny mianownik?

Kuba: Wygląda na to, że te wszystkie problemy biorą się stąd, że zaczęliśmy od desktopowej wersji i na potrzeby mobilnej wersji tylko ukrywamy niektóre elementy, takie jak mapę, obrazki itp.

Łukasz: Właśnie. Mamy sporo dużych obrazów ładowanych przez przeglądarkę, a w CSS tylko staramy się je ukryć lub pomniejszyć. Problem w tym, że te duże pliki są pobierane również na urządzeniach mobilnych. To nie jest najlepsze rozwiązanie.

Przemek: A może odwrócilibyśmy sytuację i domyślnie udostępnialibyśmy mniejsze obrazy?

Kuba: Hm... To ciekawe i może zadziałać. Rozpoczniemy od mobilnego szablonu strony, a później będziemy go uzupełniać o zawartość dla wersji desktopowej.

Łukasz: To, co opisałeś, nazywa się chyba *stopniowym ulepszaniem*?

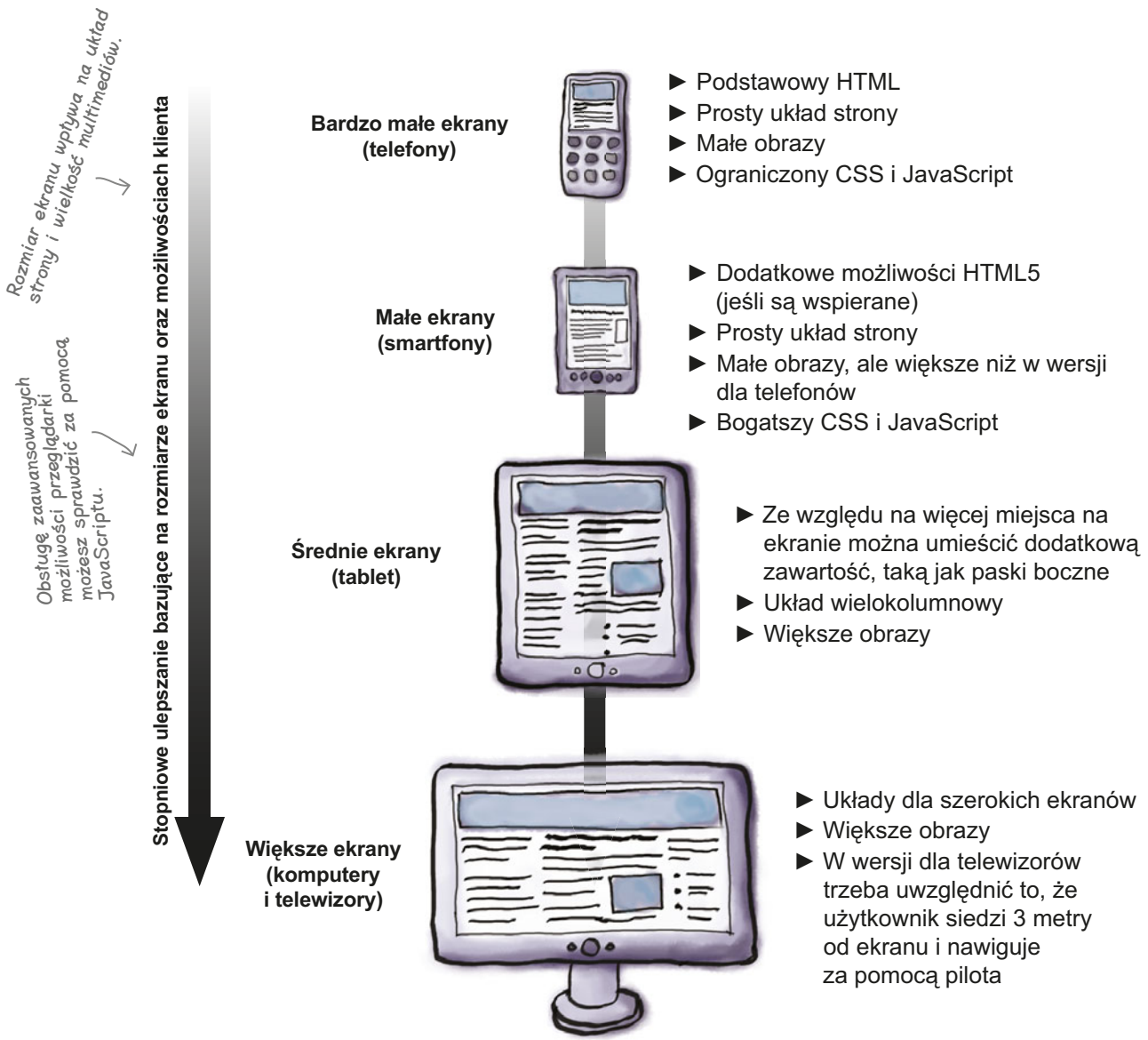
Przemek: Masz rację. Przecież korzystamy z tego podejścia już od lat. W tym przypadku rozpoczniemy od wersji mobilnej i będziemy stopniowo ulepszać dokument z myślą o wersji desktopowej.

Kuba: Wydaje się, że to dobry pomysł. Sprawdźmy to!

Progressive enhancement, czyli stopniowe ulepszanie, jest podejściem nakierowanym na tworzenie wielowarstwowych stron internetowych. Niezbędnym minimum jest treść strony, do której musi mieć dostęp każdy użytkownik. W przeglądarkach o większych możliwościach są dokładane dodatkowe warstwy stylów i interaktywności, dzięki którym strona staje się bogatsza.

Koncepcja Mobile First w podejściu Responsive Web Design

Założenia koncepcji Mobile First („najpierw mobilne”) wynikają wprost z nazwy — jest to technika RWD polegająca na rozpoczynaniu projektu od szablonu dla urządzeń mobilnych. Mimo swojej prostoty to rozwiązanie daje naprawdę spore możliwości.



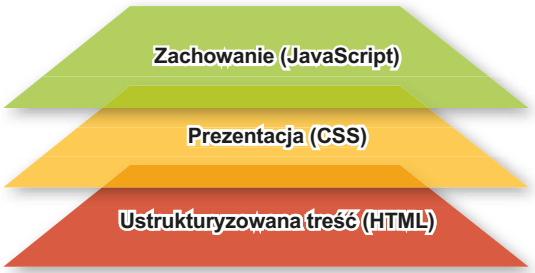
*To są tylko przykłady ulepszeń. Konkretne rozwiązania zależą od projektu.

Na czym polega stopniowe ulepszanie?

W metodzie **stopniowego ulepszania** projekt strony jest traktowany jako seria warstw. Pierwszą warstwę stanowi treść. Jeśli „ubierzesz” ją w strukturalne znaczniki HTML, otrzymasz ustrukturyzowaną treść. Gdybyś zatrzymał prace na tym etapie, tak przygotowany dokument można by bez problemu otworzyć w każdej przeglądarce.

Następnym krokiem jest dodanie warstw prezentacji (CSS) i zachowania (JavaScript). Nigdy nie możesz zakładać, że przeglądarka obsługuje te technologie, ale jeśli tak jest, użytkownik zobaczy bogatszą wersję strony.

Przez wiele lat twórcy stron internetowych skupiali się na wybranych, najbardziej zaawansowanych przeglądarkach i starali się, by strony — zgodnie z zasadą „degradacji z wdziękiem” (ang. *graceful degradation*) — wyświetlały się w miarę poprawnie w starszych przeglądarkach. Metoda stopniowego ulepszania jest dokładnym przeciwieństwem tego podejścia.



Stopniowe ulepszanie jest jak wielowarstwowy tort. Mm... Tort...

Korzyści ze stosowania techniki Mobile First

Technika Mobile First nie różni się bardzo od stopniowego ulepszania. Biorąc to pod uwagę, wielu projektantów mówi raczej o podejściu „najpierw treść” (ang. *content first*), ponieważ w technice stopniowego ulepszania to właśnie treść jest pierwszą, podstawową warstwą.

Bez względu na to, jak nazwiesz to podejście, rozpoczynanie od podstawowego dokumentu nie tylko umożliwia dotarcie do większej grupy odbiorców, ale również ma kilka pozytywnych efektów ubocznych.

O technice Mobile First możesz pomyśleć jak o diecie „małego talerza”, która polega na tym, że jeśli podczas posiłków korzystasz z mniejszego talerza, faktycznie mniej zjesz.

Typowa strona desktopowa przypomina szwedzki stół, na którym znajdziesz wszystko i w dużych ilościach.

Strona mobilna to talerzyk deserowy. Musisz się dobrze zastanowić i ustalić priorytety, zanim coś na nim położysz.

A kiedy już masz mobilną stronę, jesteś lepiej przygotowany do odpowiedzi na pytanie, czy to, czego nie umieściłeś w wersji mobilnej, jest na tyle istotne, że powinno się znaleźć w wersji desktopowej.

Zbliżenie na semantyczne znaczniki

Semantyczne znaczniki to te znaczniki i atrybuty HTML-a, które są nośnikiem znaczenia treści.

Na przykład treść umieszczona w znaczniku `<h1>` jest bardziej istotna na stronie niż treść umieszczona w znacznikach `<h2>` czy `<p>`.

Atrybuty `class` oraz `id` również mogą wnosić do dokumentu znaczenie, jeśli ich wartości odnoszą się do funkcji pełnionej przez element (np. `kalendarz`), a nie do formy prezentacji (np. `lewy` czy `dolny`). Wielu twórców stron w celu dostarczenia bardziej semantycznego dokumentu korzysta z klas w standardowy sposób, nazywany **mikroformatami**. Więcej na ten temat możesz znaleźć pod adresem www.microformats.org.

Podejście semantyczne nie oznacza, że należy za wszelką cenę unikać znaczników niewprowadzających znaczenia, takich jak `<div>` czy `<span>`. Powinno się raczej w jak największym stopniu stosować odpowiednie znaczniki i atrybuty semantyczne.

Przerabiamy stronę

Ponieważ już stosowaliśmy podejście RWD, przekształcenie strony zgodnie z techniką Mobile First nie powinno zająć za dużo czasu. Oto krótka lista zmian, które zamierzamy wprowadzić.

- ☐ Do maksimum uprościmy dokument HTML i zmienimy kolejność w arkuszu CSS, tak by wersja dla urządzeń mobilnych była pierwsza.
- ☐ Poprawimy obrazy tła w arkuszu stylów, tak by na jeden obraz był pobierany tylko jeden plik. Upewnimy się również, czy poprawnie stosujemy `display:none`.
- ☐ Dostarczymy różne pliki dla znaczników `<img>` w zależności od rozmiaru ekranu. Upewnimy się, czy są pobierane prawidłowe pliki.
- ☐ Do osadzenia mapy Google'a użyjemy JavaScriptu, tak by mapa była umieszczana tylko wtedy, gdy przeglądarka potrafi ją obsłużyć, a ekran jest na tyle szeroki, by ją wyświetlić.

Wyjaśnimy to niebawem.

Bieżąca struktura strony Co dziś lejemy

Otwórz plik *ontap.html* z katalogu *rozdzial2*. Zawartość pliku jest bardzo podobna do dokumentu, z którym pracowaliśmy w rozdziale 1.

Dwie kolumny zamiast trzech.

```
<div class="navigation">...</div>
<div class="header">...</div>
<h1>...</h1>
<div id="visit" class="column">...</div>
<div id="ontap" class="column">...</div>
<div class="footer">...</div>
```

Zamiast bloku o nazwie „main" mamy tu kolumnę o identyfikatorze „ontap", która zawiera listę oferowanych piw.

Ponieważ przyłożyliśmy się do tworzenia semantycznego kodu znacznikowego, dokument jest czysty i prosty. Wygląda na to, że pierwszym krokiem w stronę Mobile First będzie usunięcie fragmentu odpowiedzialnego za osadzanie mapy Google'a.

Ponieważ kod osadzający przyda nam się później, umieścimy go na razie w komentarzu, tak by ramka `iframe` nie pojawiała się na stronie.

Ramkę iframe z kodem osadzającym mapę otaczamy znacznikami komentarza <!-- i -->.

Ramkę iframe znajdziesz w bloku o identyfikatorze visit.

```
<!--
<iframe id="map" width="300" height="300" frameborder="0" scrolling="no"
marginheight="0" marginwidth="0" src="http://mapy.google.pl..."></iframe>
-->
```

Jestem już na nowej stronie czy nie?

Strona Co dziś lejemy wygląda prawie tak samo jak strona główna. Jak poinformować odwiedzających ją użytkowników, że są na innej stronie?

Masz rację! Mamy problem z kolejnością wyświetlanych treści.

Bardzo dobrze, że na stronie głównej jako pierwsza pojawia się sekcja *Odwiedź nas*. Ale jeśli te same informacje są widoczne na każdej stronie, odwiedzający może nie zauważyć, że zmieniła się wyświetlana strona.

W związku z tym musimy zreorganizować stronę, tak by informacje bezpośrednio związane ze stroną Co dziś lejemy znalazły się nad sekcją *Odwiedź nas*.

Zrób to sam!

Skopiuj cały blok `<div>` o identyfikatorze `visit` i wklej go pod blokiem `ontap`.

```
<div class="navigation">...</div>
<div class="header">...</div>
<h1>...</h1>
<div id="ontap" class="column">...</div>
<div id="visit" class="column">...</div>
<div class="footer">...</div>
```

WYSIL SZARE KOMÓRKI

Czy na tej stronie sekcja *Odwiedź nas* jest naprawdę niezbędna? Czy nie byłoby lepiej przenieść ją na osobną stronę i dodać tylko odsyłacz? A może opuścić ją w wersji mobilnej i zastosować skrypt JavaScript, który doda sekcję, gdy strona jest renderowana na większych ekranach?

Poprawiamy pływanie elementów

Zmiana kolejności bloków z treścią spowodowała problemy z układem strony wyświetlanej w przeglądarce desktopowej. Sekcja *Odwiedź nas* znajduje się teraz na dole strony.

W rozdziale 1. wspomnieliśmy, że umieszczenie prawej kolumny przed główną zawartością ułatwia zapanowanie nad opływaniem. Jeśli chcesz, by jakiś blok opływał inny, musisz umieścić go wcześniej w kodzie.

W tej chwili nie powinieneś się jednak tym przejmować, ponieważ jest na to prosta rada. Do tej pory sekcja *Odwiedź nas* była opływana przez zawartość głównego bloku z lewej strony, a musimy sprawić, by główna opływała sekcję *Odwiedź nas* z prawej strony.

Otwórz plik *taps.css* i wprowadź dwie poniższe zmiany.

Sekcja Odwiedź nas opływa etykiety piwa.

Przed

```
@media screen and (min-width:481px) {
    .column {
        margin: 10px 1.04166667% 0 0;
    }
    #visit {
        width: 31.25%;
        float: left;
    }
    #points {
        width: 25%;
        float: right;
    }
    #main {
        margin: 10px 27.0833333% 0
26.0416667%;
    }
    #ontap {
        margin: 10px 0 0 32%;
    }
}
```

Ten fragment zmień...

...na taki.

Po

```
@media screen and (min-width:481px) {
    .column {
        margin: 10px 1.04166667% 0 0;
    }
    #visit {
        margin: 0 68.75% 0 0;
    }
    #points {
        width: 25%;
        float: right;
    }
    #main {
        margin: 10px 27.0833333% 0
26.0416667%;
    }
    #ontap {
        width: 67%;
        float: right;
        margin: 10px 0 0 0;
    }
```

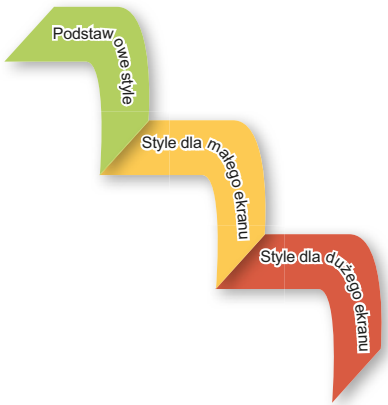
Ten fragment zmień...

...na taki.

Zastosowanie 67% zamiast 68,75% daje nam pewne pole manewru kolumnami.

Zapytania o media w technice Mobile First

Przyszła pora na wiosenne porządki. W rozdziale 1. wyszliśmy od desktopowej witryny i przekształciliśmy ją w mobilną. Teraz odwrócimy sytuację i zaczniemy od najprostszej wersji, by na jej podstawie opracować wersję desktopową (a nawet więcej).



Najpierw jednak musimy coś wyjaśnić. W przypadku stylów pojęcie „Mobile First” nie jest tak do końca właściwe. Zanim zastosujemy jakiekolwiek zapytania o media dla małych ekranów, zajmiemy się podstawowymi stylami (dla kolorów, fontów itp.), by później je rozbudowywać.

To całkiem rozsądne podejście. Wiele przeglądarek mobilnych w ogóle nie rozpoznaje zapytań o media, więc musimy zadbać o to, by zastosowały chociaż podstawowe reguły stylów.

Układamy style w odpowiedniej kolejności

Kod plików CSS bardzo często przypomina zawartość kuchennej szuflady „na wszystko”. Na początku jest przeważnie uporządkowany i logiczny, ale w miarę upływu czasu chaos bierze górę. Aby zapytania o media znalazły się we właściwym miejscu, będziesz musiał wyciągnąć podstawowe style z tych związanych z układem.

Kaskada stylów przepływa od małych ekranów do dużych.

Na szczęście arkusz, który powstał w rozdziale 1., jest w nie najgorszym stanie. Większość podstawowych reguł stylów znajduje się na początku pliku, a zapytania o media dodające reguły dla układu i formatowania są umieszczone poniżej. Wszystko, co musimy teraz zrobić, to umieścić zapytania o media dla urządzeń mobilnych przed zapytaniami dla wersji desktopowej.

Zrób to sam!

```
/* Szersze ekrany/wyższe rozdzielczości (np. komputery) */
@media screen and (min-width:481px) {
[Reguły układu dla wersji desktopowej]
}
/* Urządzenia mobilne/niższe rozdzielczości */
@media screen and (max-width:480px) {
[Reguły układu dla wersji mobilnej]
}
```

Przenieś blok zapytania o media dla urządzeń mobilnych nad zapytanie dla wersji desktopowej. Dzięki temu będziemy mieć pewność, że efekt kaskady stylów jest zgodny z podejściem stopniowego ulepszania w duchu Mobile First.

Jazda próbna

W kodzie strony wprowadziliśmy kilka zmian:

- Usunęliśmy mapę Google’a.
- Zreorganizowaliśmy kod HTML.
- Poprawiliśmy opływanie.
- Zreorganizowaliśmy zapytania o media.

Lepiej sprawdźmy, czy strona wciąż działa. Otwórz ją w desktopowej i mobilnej przeglądarce, by sprawdzić, jak wygląda. Koniecznie sprawdź też w Internet Explorerze.

Niespodzianka! W Internet Explorerze strona się rozsypała

Tylko nie mów, że kiedy wspomnieliśmy, byś sprawdził stronę w Internet Explorerze (IE), nie przeczuwałeś najgorszego. Niestety dla każdego twórcy stron walka z IE to codzienność. Całkiem możliwe, że po ostatnio stoczonym boju spodziewałeś się, że i tym razem gdzieś za rogiem czai się zło...

No i co mamy tym razem? Jasne... *IE nie obsługuje zapytań o media.*

Jednak zanim rzucisz tę książkę w kąt i zaczniesz ciskać w nas gromy za to, że uczymy Cię czegoś, co nie działa w ponoć najpopularniejszej przeglądarce świata, weź głęboki oddech i się uspokój. Na szczęście są sposoby na obejście (bardzo) wielu błędów w IE.

Może byliśmy trochę za ostrzy. IE9 już wspiera zapytania o media, więc nie jest aż tak źle.

Ponieważ IE8 (i wcześniejsze wersje) nie obsługuje zapytań o media, reguły tworzące kolumny nie są stosowane.

Wyjście ewakuacyjne w IE — komentarze warunkowe

Firma Microsoft udostępniła przydatne narzędzie pomagające twórcom stron dostarczać kod tylko do Internet Explorera za pomocą komentarzy warunkowych.

Tu sprawdzamy, czy przeglądarka to IE w wersji wcześniejszej (lt) niż 9 i czy nie jest to wersja mobilna (!IEMobile). Nie uwzględniamy wersji mobilnej, ponieważ chodzi nam tu o desktopową wersję układu strony. IE9 i nowsze wersje obsługują zapytania o media, więc nie musimy się nimi w tym momencie przejmować.

Pełną składnię komentarzy warunkowych możesz znaleźć na stronie http://bit.ly/ie-comments.

```
<!--[if (lt IE 9)&(!IEMobile)]>
<link rel="stylesheet" type="text/css" href="layout.css" media="all" />
<![endif]-->
```

Przyjrzyj się uważnie. Komentarz HTML rozpoczyna się w pierwszej linii, a kończy w ostatniej (-->). Inne przeglądarki uznają cały ten fragment za komentarz, więc go zignorują.

Jeśli przeglądarka IE natrafi na taki komentarz, wykona wszystko, co znajduje się między otwierającą instrukcją [if] i zamykającą [endif]. W tym przykładzie jest to akurat dołączenie pliku CSS, ale może być to cokolwiek, co tylko może się znaleźć w dokumencie HTML.

Komentarze warunkowe z zapytaniami o media

Zauważyłeś pewnie, że komentarz warunkowy dołącza plik *layout.css*. Czas, aby utworzyć ten plik.

Część reguł skopiujemy z bieżącego arkusza stylów. Nazwaliśmy ten plik *layout.css*, ponieważ będzie stosowany tylko przez przeglądarki wyświetlane na dużych ekranach, na których bez problemu zmieści się układ wielokolumnowy.

1. **Utwórz pusty plik tekstowy, zapisz go jako layout.css i skopiuj do niego reguły dla desktopowej wersji układu.**
   Musisz skopiować wszystko, co znajduje się między początkiem i końcem zapytania o media, ale bez samej reguły `@media`.

taps.css

```
/* Szersze ekrany/wyższe rozdzielczości
(np. komputery) */
@media screen and (min-width:481px) {
  .column {
    margin: 10px 1.04166667% 0 0;
  }
  #visit {
    margin: 0 68.75% 0 0;
  }
  #points {
    width: 25%;
    float: right;
  }
  #main {
    margin: 10px 27.0833333% 0
26.0416667%;
  }
  #ontap {
    width: 67%;
    float: right;
    margin: 10px 0 0 0;
  }
}
```

Skopiuj te reguły do nowego pliku.

Po skopiowaniu tego fragmentu usuń reguły oraz zapytanie o media z pliku taps.css. Trochę później ponownie je zastosujemy do dokumentu HTML.

layout.css

```
.column {
   margin: 10px 1.04166667% 0 0;
}
#visit {
   margin: 0 68.75% 0 0;
}
#points {
   width: 25%;
   float: right;
}
#main {
   margin: 10px 27.0833333% 0
26.0416667%;
}
#ontap {
   width: 67%;
   float: right;
   margin: 10px 0 0 0;
}
```

2 Dodaj odsyłacz do nowego arkusza stylów.
Dodajemy odsyłacz do pliku *layout.css*, który zostanie załadowany przez przeglądarki obsługujące zapytania o media w przypadku odpowiedniej szerokości ekranu.

```
<link rel="stylesheet" type="text/css" href="taps.css" />
<link rel="stylesheet" type="text/css" href="layout.css" media="all and
(min-width: 481px)" />
```

Ten znacznik umieść w pliku ontap.html.

Wartość 481px dla atrybutu min-width została skopiowana z zapytania o media usuniętego z pliku taps.css.

Tę składnię zapytania o media umieszczonego w znaczniku <link> poznałeś w rozdziale 1.

3 Dodaj komentarz warunkowy dla IE.
W tej chwili działa to w większości przeglądarek desktopowych. Aby zadziałało we wszystkich, musimy jeszcze dodać komentarz warunkowy, który przygotowaliśmy wcześniej.

```
<link rel="stylesheet" type="text/css" href="taps.css" />
<link rel="stylesheet" type="text/css" href="layout.css" media="all and
(min-width: 481px)">
<!--[if (lt IE 9)&(!IEMobile)]>
<link rel="stylesheet" type="text/css" href="layout.css" media="all" />
<![endif]-->
```

W komentarzu warunkowym osadzamy ten sam arkusz co powyżej, tak by przeglądarka IE dołączyła plik layout.css.

4 Czas na ponowny test.
Sprawdź stronę w przeglądarce, która obsługuje zapytania o media, i w różnych wersjach Internet Explorera. Wygląda dobrze, prawda?

Ten układ jest już wyświetlany prawidłowo nawet przez starego upierdliwca IE.

Nie istnieją głupie pytania

P: Na horyzoncie już widać superszybkie telefony 4G. Czy w związku z tym taka dbałość o wydajność jest naprawdę konieczna?

O: Oczywiście. Przecież nawet telefonom 4G zdarza się czasem korzystać z połączenia EDGE (to starsze, znacznie wolniejsze połączenie). Badania pokazują, że powolne strony mają mniejszą oglądalność, co bezpośrednio wpływa na obniżenie ewentualnych zysków.

P: Dlaczego uzyskałem inne wyniki w teście Blaze Mobitest?

O: Jest wiele możliwych przyczyn. Czas ładowania strony może się zmieniać w związku z obciążeniem sieci. Kod osadzający mapę Google'a różni się dla poszczególnych systemów operacyjnych i może się czasem zmieniać. Zachowanie telefonów również może ulec zmianie po wprowadzeniu nowszej wersji systemu operacyjnego. W tej książce korzystaliśmy z testowych urządzeń z systemami iOS 4.3, Android 2.2 oraz Android 2.3.

Nie przejmuj się tymi drobnymi różnicami. Istotne jest to, że niepotrzebnie pobierane są kod JavaScript i pliki obrazów.

P: Czy podzielenie arkusza stylów na dwa pliki nie spowoduje, że strona będzie się ładowała wolniej?

O: To prawda, że liczba wykonywanych żądań HTTP wpływa na prędkość ładowania. Nie powinniśmy więc lekkomyślnie dodawać kolejnych żądań. W tym przypadku uznaliśmy jednak, że wydzielenie fragmentu arkusza dla IE jest uzasadnione.

P: Wspomnieliście, że postawienie serwera proxy może być dobrym pomysłem. Jakie serwery polecacie?

O: Jest wiele serwerów proxy, w tym kilka naprawdę ciekawych, które są dostępne na zasadach open source. Tak się jednak składa, że jesteśmy fanami komercyjnego serwera Charles Proxy.

P: Wydaje się, że brak dodatków może być problemem. Jak sobie poradzić bez takich narzędzi jak Firebug czy Web Inspector?

O: To faktycznie nie jest proste, ale tak się składa, że większość prac związanych z debugowaniem możesz przeprowadzić w przeglądarce desktopowej, pod warunkiem że wykonasz też testy na rzeczywistych urządzeniach.

Poza tym cały czas powstają narzędzia umożliwiające obejście ograniczenia związanego z dodatkami. Zestaw Mobile Perf Bookmarklet (*http://bit.ly/mw-perf*) zawiera wiele przydatnych narzędzi, a weinre (*http://bit.ly/mweinre*) oraz Opera Dragonfly (*http://opera.com/dragonfly*) umożliwiają uruchomienie Web Inspectora na komputerze i sprawdzanie działania strony wyświetlanej w telefonie.

P: Szczerze mówiąc, niewiele się zmieniło w kodzie po przejściu na zapytania o media zgodne z podejściem Mobile First. Po co się tym przejmować?

O: W przypadku tej strony w pliku CSS nie było dużych różnic między wersją desktopową i mobilną. Jednak z naszego doświadczenia wynika, że to jest wyjątek. W bardziej złożonych arkuszach często zachodzi potrzeba przesłonięcia niektórych (ale nie wszystkich) reguł dla mniejszych ekranów przez te dla większych. Zreorganizowanie zapytań o media jest istotne, ponieważ zapewniamy w ten sposób prawidłowe zachowanie kaskady stylów, co jest niezbędne do prawidłowej implementacji zgodnej z podejściem stopniowego ulepszania.

P: Wygląda na to, że kolejność elementów wyświetlanych na stronie często różni się między wersją mobilną i desktopową. Jak sobie z tym poradzić w przypadku bardziej skomplikowanych stron?

O: O! Zauważyłeś to, prawda? Tak, to jest jedno z większych wyzwań związanych z podejściem Responsive Web Design. Być może rozwiązaniem będzie opracowywany model wyświetlania Flexible Box Layout (w skrócie flexbox) dla CSS3. Wystarczy połączyć flexbox z zapytaniami o media, a reorganizacja strony w związku ze zmianą warunków wyświetlania staje się banalna. Niestety model flexbox jest jeszcze młody i dlatego nie jest w pełni obsługiwany przez przeglądarki. Twórcom stron pozostaje więc uciekanie się do sztuczek w JavaScripcie lub połączenie RWD z wykrywaniem urządzenia (patrz rozdział 5.). Tak naprawdę rozmieszczanie zawartości i obsługa obrazów to dwa największe wyzwania stojące przed RWD.

P: Czy przeglądarka IE wyświetli prawidłowo projekt stworzony zgodnie z podejściem RWD, skoro wiele jej wersji nie obsługuje zapytań o media? Czy zapytania o media nie są konieczne?

O: Internet Explorer wyświetli wersję desktopową. Będą działały układy oparte na płynnej siatce oraz płynne obrazy. Niestety nie będą brane pod uwagę żadne zmiany wynikające z zapytań o media. Jeśli jednak w danym projekcie są niezbędne, można się posłużyć biblioteką Respond.js, dzięki której starsze wersje IE mogą obsługiwać zapytania o media. Musisz jednak wiedzieć, że ten skrypt mocno obciąża przeglądarkę, więc zanim go użyjesz, gruntownie go przetestuj.

Jak nam idzie?

Zajęliśmy się już uporządkowaniem kodu HTML i CSS. Co jest dalej na naszej liście?

- [x] Do maksimum uprościmy dokument HTML i zmienimy kolejność w arkuszu CSS, tak by wersja dla urządzeń mobilnych była pierwsza.
- [ ] Poprawimy obrazy tła w arkuszu stylów, tak by na jeden obraz był pobierany tylko jeden plik. Upewnimy się również, czy poprawnie stosujemy `display:none`.
- [ ] Dostarczymy różne pliki dla znaczników `<img>` w zależności od rozmiaru ekranu. Upewnimy się, czy są pobierane prawidłowe pliki.
- [ ] Do osadzenia mapy Google'a użyjemy JavaScriptu, tak by mapa była umieszczana tylko wtedy, gdy przeglądarka potrafi ją obsłużyć, a ekran jest na tyle szeroki, by ją wyświetlić.

Bierzemy na warsztat obraz z nagłówka

Z wykresu kaskadowego mogliśmy wyczytać, że mamy jeden duży obraz tła, który jest ukrywany za pomocą `display:none`. Mimo że obraz nie jest wyświetlany na stronie, przeglądarka i tak go pobiera.

Pamiętasz naszego kolegę, plik taps.jpg, który był pobierany, ale nigdy nie był wyświetlany na stronie?

Dużo lepszym rozwiązaniem byłoby pobieranie obrazu tylko wtedy, gdy ma zostać wyświetlony. Ale jak to zrobić? Wystarczy umieścić go w zapytaniu o media, tak by był ładowany tylko dla ekranów szerszych niż 480 pikseli.

Ale zamiast tworzyć całkiem nowe zapytanie o media, umieścimy odpowiednie reguły w pliku *layout.css*, który już jest dołączony do strony za pomocą znacznika `<link>` uzupełnionego o zapytanie o media.

Skopiuj ten fragment z arkusza taps.css i umieść go na końcu pliku layout.css.

```
.header {
    background:URL('images/taps.jpg') repeat-x;
    height: 300px;
}
```

Po dodaniu do pliku layout.css usuń ten fragment z arkusza taps.css.

Jazda próbna

Sprawdź stronę Co dziś lejemy za pomocą narzędzia Blaze Mobitest i upewnij się, że plik *taps.jpg* nie jest już pobierany. Wykonaj test dla iPhone'a i Androida. Jeśli nie możesz umieścić przygotowanej przez siebie strony na serwerze, skorzystaj z adresu *ftp://ftp.helion.pl/online/inne/hfmw/r02/3/ontap.html*.

Hm... działa na iPhonie, ale na Androidzie obraz ciągle jest pobierany.

GET sauvie_island.jpg	200 OK	102.5 KB
⊞ GET tryon_creek.jpg	200 OK	149.9 KB
⊞ GET taps.jpg	200 OK	440.7 KB
20 Requests		2.1 MB

Wygląda na to, że na Androidzie plik taps.jpg jest ciągle pobierany.

Z wyników testu Mobitest wynika, że na Androidzie obraz jest pobierany, ale to nieprawda.

Firma Blaze zmodyfikowała telefony używane podczas testów i dostosowała je do zdalnej pracy. To niestety powoduje czasem dziwne i niewłaściwe zachowania.

Gdybyś użył zwykłego smartfonu z Androidem, obraz *taps.jpg* nie byłby pobierany.

Stara dobra szkoła optymalizacji obrazów

W dawnych czasach twórcy stron poświęcali dużo uwagi optymalizacji obrazów. W miarę wzrostu przepustowości łączy internetowych przestali jednak walczyć z każdym nadmiarowym bajtem. Wraz z upowszechnieniem się urządzeń mobilnych ten problem znów stał się aktualny.

Wydaje się, że moglibyśmy zmniejszyć plik obrazu *taps.jpg* za pomocą prostych metod optymalizacyjnych. Po zmianie jakości JPEG z 80 do 45 rozmiar pliku maleje z 440 do 78 kB, a pogorszenie jakości nie będzie się w tym przypadku rzucało w oczy.

Można by się spierać, czy optymalizacja obrazów przestała być istotnym zagadnieniem. Nie wszyscy użytkownicy dysponują szybkimi łączami, nawet w komputerach stacjonarnych.

Więcej informacji na temat optymalizacji obrazów znajdziesz w rozdziale 5. książki Head First HTML with CSS & XHTML[1].

Pliki obrazu są wciąż za duże? Ściśnij je jeszcze mocniej na www.smushit.com.

Zoptymalizowaną wersję pliku *taps.jpg* skopiuj z folderu *extras* i umieść w folderze *images*, zamieniając oryginalny, duży plik na jego mniejszą wersję.

Do optymalizacji zdjęcia używamy Photoshopa, ale możesz skorzystać z dowolnego innego narzędzia tego typu.

[1] E. Freeman, E. Freeman, *Head First HTML with CSS & XHTML. Edycja polska*, Helion, Gliwice 2007 — *przyp. tłum.*

Problemy z jednym atrybutem src

Obrazy definiowane w arkuszu CSS to dopiero początek nieszczęść z obrazami. Znacznik `<img>` stanowi problem we wszystkich projektach realizowanych zgodnie z podejściem Responsive Web Design, ponieważ możemy podać tylko jedną wartość atrybutu `src` bez względu na rozmiar ekranu. Jak w takim razie dostarczyć obraz odpowiedniej wielkości?

Na stronie Co dziś lejemy znajduje się 16 etykiet piwa, które są umieszczane za pomocą znacznika <img>. To jest przykład tego znacznika dla piwa Benson's Bubbler.

Chcemy pobierać obraz dostosowany do rozmiaru ekranu, jednak nie jest to możliwe z poziomu HTML-a, ponieważ do atrybutu src można przypisać tylko jedną wartość.

```
<img src="brews_images/bensons_bubbler.jpg" alt="Benson's Bubbler">
```

Można pomyśleć, że dobrym rozwiązaniem byłaby zmiana wartości atrybutu `src` z poziomu JavaScriptu. Niestety większość przeglądarek, zanim wykona skrypty JavaScript, analizuje całą stronę i pobiera pliki obrazów. W związku z tym zmiana wartości `src` powodowałaby tak naprawdę podwójne ładowanie pliku, a dodatkowo jeszcze przebudowanie układu strony przez przeglądarkę.

Z poziomu CSS też nie możemy nadpisać wartości atrybutu src.

Na ratunek przybywa serwer obrazów

Jeśli przeglądarka nie może poinformować serwera o odpowiednim obrazie do pobrania, to serwer powinien sam to rozwiązać. Tak właśnie działa usługa **Sencha.io Src**.

Pełna nazwa tej usługi skalowania obrazów to TinySRC.

Usługi Sencha.io Src możemy użyć do dostarczania przeglądarce obrazów dopasowanych do możliwości dowolnego urządzenia.

Pierwszą część adresu w atrybucie src ustaw na http://src.sencha.io/.

Tu umieść nazwę domeny i ścieżkę do katalogu z plikami obrazu.

Po ukośniku wpisz pełny URL pliku obrazu, który ma być przeskalowany.

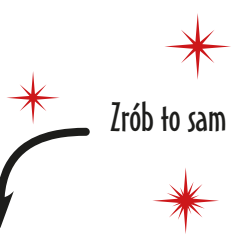
```
<img src="http://src.sencha.io/http://[DOMENA]/[ŚCIEŻKA]/brews_images/bensons_bubbler.jpg"
alt="Benson's Bubbler">
```

Obejrzyj to!

Usługa Sencha.io Src jedynie *zmniejsza* obrazy.

Nigdy ich nie powiększa, ponieważ powoduje to pogorszenie jakości. Z tego względu źródłowe obrazy powinny być wysokiej jakości.

Usługa Sencha.io Src przeskaluje obraz do wymiarów ekranu urządzenia. Jeśli odwiedzasz stronę na przykład za pomocą iPhone'a, obraz zostanie przeskalowany do ekranu o wymiarach 320 na 480 pikseli.



Zmodyfikuj wszystkie znaczniki obrazów przedstawiających etykiety piwa w pliku *ontap.html*, tak by używały usługi Sencha.io Src.

Jak działa usługa Sencha.io Src?

Wygląda to na magię! Zgłaszasz żądanie pobrania obrazu z iPhone'a, a usługa przekazuje odpowiednio przeskalowany obraz. Korzystasz ze zwykłego telefonu? Nie ma sprawy — Sencha.io Src udostępni malutki obraz. Siedzisz przy zwykłym komputerze — proszę, obraz w pełnych wymiarach.

Skąd ta usługa wie, jaki jest rozmiar okna w urządzeniu, z którego jest zgłaszane żądanie? Korzysta z *opisu agenta użytkownika*, czyli identyfikatora, który jest dostarczany do serwera przez każdą przeglądarkę. Ten identyfikator służy do odszukania w obszernej bazie danych obejmującej tysiące urządzeń tego, z którego zostało zgłoszone żądanie. A jedną z informacji zawartych w bazie jest rozmiar ekranu.

Co to jest opis agenta użytkownika? Więcej na ten temat powiemy w kolejnym rozdziale.

Kiedy jest już znany rozmiar ekranu, usługa przeskalowuje obraz do maksymalnej szerokości ekranu danego urządzenia. Utworzony obraz jest przechowywany przez 30 minut, tak aby kolejne żądania mogły być obsługiwane szybciej.

Jeśli chcesz, by usługa Sencha.io Src dostarczyła obraz o określonych wymiarach, możesz to oczywiście zrobić. Więcej informacji znajdziesz na stronie http://bit.ly/senchasrc.

Nie ma idealnego rozwiązania dla znaczników <img>

Używanie usługi Sencha.io Src nie jest idealnym rozwiązaniem. Bazuje na wykrywaniu urządzeń, co czasami może się nie powieść (ten problem opisaliśmy w rozdziale 5.). Poza tym wszystkie obrazy muszą przejść przez zewnętrzny serwer.

Niestety prawda jest taka, że do tej pory nie pojawiło się doskonałe rozwiązanie problemu dostarczania obrazów o wymiarach dopasowanych do ekranu urządzenia. Jednak sytuacja zmienia się bardzo szybko, ponieważ nad znalezieniem lepszego rozwiązania pracuje wielu ludzi.

Dlaczego znacznik <img> sprawia takie problemy? Poczytaj artykuły na stronie http://bit.ly/rwdimgs1.

Ostatni szlif — zoptymalizowane obrazy etykiet piwa

Podobnie jak w przypadku obrazu *taps.jpg*, pliki z etykietami piwa również możemy trochę zmniejszyć, stosując niższą jakość kompresji JPEG. Nie martw się, zrobiliśmy to za Ciebie. Znajdziesz je w katalogu *extras/labels-optimized*.

Skopiuj wszystkie zoptymalizowane pliki do katalogu *brew_images*, nadpisując ich poprzednie wersje.

Jazda próbna

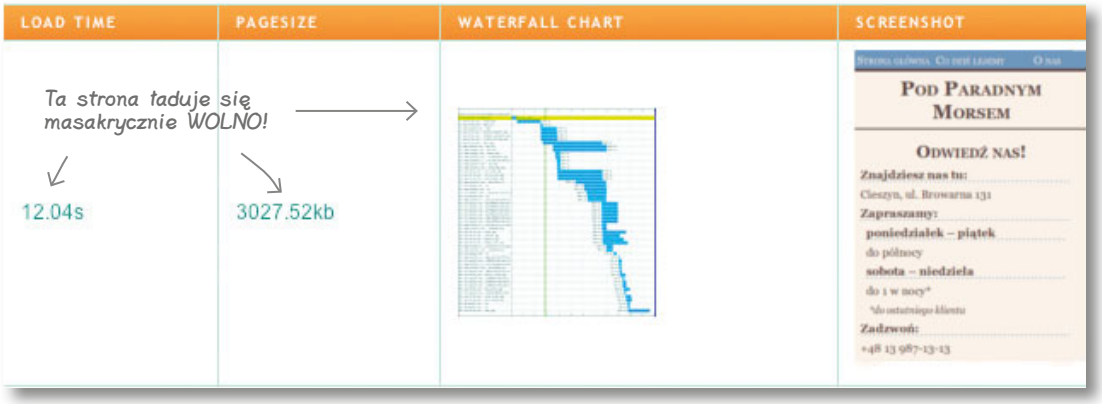
Powinniśmy już mieć wydajną i szybką stronę zoptymalizowaną pod kątem urządzeń mobilnych. Przetestuj ją ponownie za pomocą narzędzia Blaze Mobitest (*www.blaze.io/mobile*), aby porównać wyniki. Wybierz wykonanie trzech testów, z których jest wyciągana średnia. Porównaj całkowity rozmiar strony oraz czas ładowania nowej wersji strony z poprzednimi wynikami.

Jeśli nie jesteś w stanie umieścić strony na publicznym serwerze, skorzystaj z adresu *ftp://ftp.helion.pl/online/inne/hfmw/r02/4/ontap.html*.

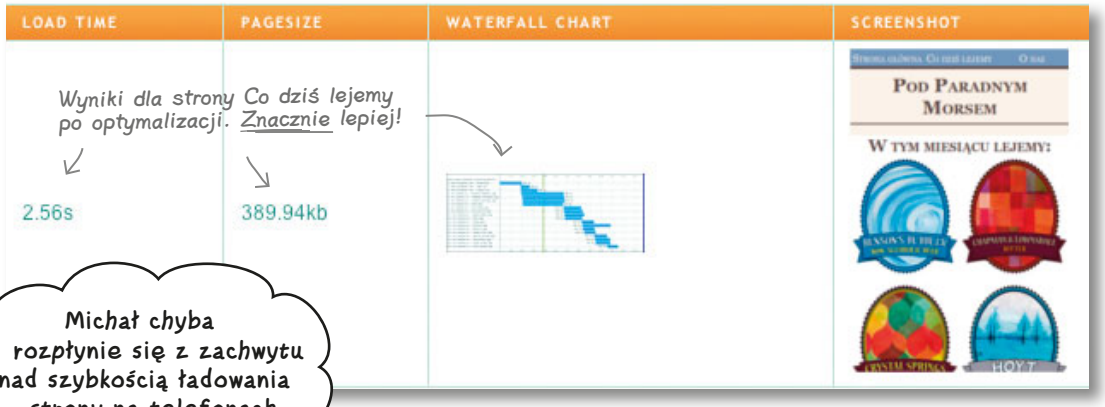
Superszybka strona mobilna

Zastosowana przez nas dieta zdziałała cuda! Strona Co dziś lejemy jest o 87% lżejsza niż wcześniej. Czas pobierania na iPhonie spadł z 12 sekund do prawie 3.

Przed



Po



WYSIL SZARE KOMÓRKI

Jest wiele innych sposobów na optymalizację wydajności stron. W jaki inny sposób mógłbyś poprawić szybkość ładowania tej strony?

Nie istnieją głupie pytania

P: Dlaczego przeglądarki pobierają pliki obrazów, które nie są w ogóle wyświetlane?

O: Zwykle przeglądarka nie jest w stanie ze stuprocentową pewnością stwierdzić, że dany obraz nie będzie wyświetlany. Obraz może być przecież wyświetlany z poziomu JavaScriptu na przykład w odpowiedzi na jakieś zdarzenie. Przeglądarki pobierają obrazy z wyprzedzeniem, aby użytkownicy nie musieli czekać na ich załadowanie po zajściu jakiegoś zdarzenia powodującego ich wyświetlenie.

P: W porządku, ale w takim razie dlaczego nie pobierają obrazów zdefiniowanych w zapytaniach o media.

O: Początkowo i te obrazy były pobierane. Jednak twórcy przeglądarek obserwowali i analizowali sposób wykorzystania zapytań o media i dopasowali zachowania przeglądarek do oczekiwań twórców stron. To są stosunkowo nowe zagadnienia, wciąż rozwijane, więc niektóre przeglądarki nadal pobierają zasoby umieszczone w nieaktywnych w danej sytuacji zapytaniach o media.

P: Czy przekazywanie obrazów przez zewnętrzny serwis, taki jak na przykład Sencha.io Src, jest bezpieczne? Nie czuję się z tym dobrze.

O: Zawsze należy zachować ostrożność. Za każdym razem, gdy integrujesz krytyczne elementy swojej strony z zewnętrznym serwisem, stajesz się zależny od tego serwisu.

Twórcy serwisu Sencha.io Src zapewniają, że zamierzają go utrzymywać i pozostanie nieodpłatny. Musisz sobie jednak zdawać sprawę, że jeśli okazałoby się, że z serwisu zacznie korzystać niewyobrażalna liczba stron, mogą się zmienić zasady i serwis może stać się płatny lub w ogóle przestanie działać.

Jeśli z jakichś względów nie odpowiada Ci serwis Sencha.io Src, możesz sam stworzyć coś podobnego, korzystając z tego, czego dowiesz się w rozdziale 5.

P: Czy są jakieś alternatywy dla Sencha.io Src? Czy są dostępne rozwiązania problemu znacznika `<img>` działające po stronie klienta?

O: Jest wiele innych sposobów obsługi znacznika `<img>` zgodnych z podejściem RWD. Obecnie podejmowanych jest wiele działań zmierzających do opracowania rozwiązania niewymagającego wykrywania urządzenia. Zastosowanie każdej metody wiąże się z pewnymi kompromisami, co dotyczy również metody wykorzystanej w tym projekcie. Obszerny przegląd tego typu technik znajdziesz pod adresem *http://bit.ly/rwdimgs2*.

P: A co z innymi mediami? Czy podczas stosowania elementów `video` i `audio` napotkamy te same problemy?

O: Jednym słowem: tak. Sytuacja jest tu jednak trochę lepsza, ponieważ elementy `video` i `audio` pozwalają na zdefiniowanie wielu wersji mediów w różnych formatach. Jeśli przeglądarka nie obsługuje pierwszego wskazanego pliku, szuka następnego.

To podejście, chociaż lepsze niż w przypadku obrazów, nie rozwiązuje jednak problemu prędkości pobierania czy rozdzielczości. A przecież użytkownik telefonu komórkowego ze słabym łączem raczej nie potrzebuje wideo w jakości HD. Są jednak na to sposoby, czego przykładem może być technologia QuickTime firmy Apple, która umożliwia udostępnianie materiałów wideo w jakości odpowiedniej dla różnych przepustowości łącza.

P: Czy tylko mi się wydaje, czy w podejściu Responsive Web Design jest wiele niewiadomych i nierozwiązanych dotąd problemów?

O: Z całą pewnością jest jeszcze sporo wyzwań. Tak jak w każdej innej nowej technologii, tak i tu trwają próby sprawdzenia, co działa, a co nie. Podejście RWD wymaga jeszcze gruntownego przetestowania. Właśnie dlatego w książce opisujemy wiele różnych technik. Najprawdopodobniej w swoich projektach będziesz musiał łączyć różne rozwiązania, by osiągnąć zamierzony efekt.

Mimo wyzwań nadzieje związane z RWD zainspirowały wielu ludzi do opracowania bardziej kompletnych rozwiązań. Jeśli jesteś zainteresowany tą tematyką, musisz uważnie śledzić aktualny stan rozwoju, ponieważ sytuacja zmienia się bardzo szybko.

Powiększanie, pomniejszanie...

Bardzo was przepraszam, chłopaki. Zwykle nie wyskakuję z nowymi pomysłami i wymaganiami w trakcie prac nad projektem, ale jedna z moich najlepszych klientek ma problem z odczytaniem drobnego tekstu na stronie i narzeka, że nie może powiększyć widoku. Możecie to poprawić?

Pamiętasz znacznik `<meta> viewport` z pierwszego rozdziału? Czas, by mu się bliżej przyjrzeć.

Znacznik ten umożliwia określenie pożądanych wymiarów i skali (czyli stopnia powiększenia) strony. Daje również możliwość zabezpieczenia przed zmianą rozmiaru strony przez użytkownika.

Powiększanie w znaczniku `<meta> viewport`

Znacznik `<meta> viewport` znajdziesz w sekcji `<head>` dokumentu *ontap.html*. Jego składnia jest wyjątkowo prosta.

Jakiego typu jest ten znacznik <meta>?

Atrybut content zawiera listę instrukcji dla przeglądarki oddzielonych przecinkami. Wszystkie dostępne opcje znajdziesz na stronie http://bit.ly/metaviewport.

Szerokość widoku. Można ją określić w pikselach lub za pomocą stałej „device-width", co spowoduje ustalenie szerokości równej szerokości ekranu urządzenia.

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1" />
```

Ustala początkową skalę (poziom powiększenia). Podanie wartości 1 oznacza, że dokument powinien zostać wyświetlony w normalnej skali.

Określa maksymalny stopień powiększenia strony. Jest też dostępne ustawienie „minimum-scale".

To z powodu tego ustawienia nie można powiększać naszej strony.

Czy powinno się umożliwiać skalowanie?

Wydaje mi się, że umożliwienie skalowania jest ważne ze względu na dostępność. Dlaczego ktoś chciałby z tego zrezygnować?

To faktycznie jest istotne z punktu widzenia dostępności. Są twórcy stron, którzy posuwają się nawet do stwierdzenia, że możliwość skalowania stron na urządzeniach mobilnych to jedno z podstawowych praw człowieka.

Niekoniecznie musimy podchodzić do sprawy aż tak serio, ale trzeba się zgodzić, że skalowanie jest ważne, więc na pewno trzeba się dwa razy zastanowić, zanim się pozbawi użytkowników takiej możliwości.

Dlaczego projektanci stron blokują tę możliwość? Jest co najmniej kilka przyczyn. Jeśli na stronie są używane złożone gesty, zablokowanie skalowania ułatwia korzystanie z nich.

Poza tym w systemie iOS jest błąd powodujący zmianę powiększenia po ustawieniu urządzenia w orientacji poziomej. Strona jest powiększana, co powoduje przycięcie jej prawej strony.

Gdy obrócisz urządzenie z systemem iOS, strona nie będzie się mieściła w oknie, a prawa strona zawartości zostanie przycięta.

Przywracamy możliwość skalowania

Aby przywrócić możliwość skalowania, musimy usunąć ustawienie `maximum-scale` ze znacznika `<meta> viewport`.

Zrób to sam!

Ze znacznika <meta> viewport usuń ustawienie „maximum-scale=1”.

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

Po przywróceniu możliwości skalowania obróć iPhone lub iPod Touch, aby zobaczyć błąd systemu iOS w akcji.

Nie zapomnij wykasować przecinka po „initial-scale=1”.

Wracamy do naszego planu

Po rozwiązaniu palącego problemu ze skalowaniem rzućmy okiem na postęp prac. Jesteśmy już całkiem blisko spełnienia założeń koncepcji Mobile First w podejściu RWD. Zostało jedynie dodanie mapy w sytuacji, gdy ekran jest wystarczająco duży.

- [x] Do maksimum uprościmy dokument HTML i zmienimy kolejność w arkuszu CSS, tak by wersja dla urządzeń mobilnych była pierwsza.
- [x] Poprawimy obrazy tła w arkuszu stylów, tak by na jeden obraz był pobierany tylko jeden plik. Upewnimy się również, czy poprawnie stosujemy `display:none`.
- [x] Dostarczymy różne pliki dla znaczników `<img>` w zależności od rozmiaru ekranu. Upewnimy się, czy są pobierane prawidłowe pliki.
- [ ] Do osadzenia mapy Google'a użyjemy JavaScriptu, tak by mapa była umieszczana tylko wtedy, gdy przeglądarka potrafi ją obsłużyć, a ekran jest na tyle szeroki, by ją wyświetlić.

Z pomocą JavaScriptu przywracamy mapę

Pozostało jedynie umieszczenie na stronie mapy, jeśli ekran jest wystarczająco szeroki. Już wcześniej mogliśmy się na własne oczy przekonać, że samo ukrywanie mapy nie zapobiega pobieraniu związanych z nią zasobów.

W związku z tym musimy użyć JavaScriptu, by mapa była umieszczana na stronie, gdy pozwalają na to warunki. To rozwiązanie możesz potraktować jak javascriptową wersję zapytań o media, które już poznałeś i z pewnością pokochałeś.

Powróćmy na chwilę do kodu ramki `iframe` osadzającej mapę Google'a, który tymczasowo wrzuciliśmy do komentarza. Już za moment będziemy musieli go z powrotem umieścić w dokumencie. Najpierw jednak musimy się mu bliżej przyjrzeć.

Pamiętasz ten fragment kodu wrzucony w komentarz? Użyjemy teraz JavaScriptu, by z powrotem umieścić go na stronie.

```
<!--
<iframe id="map" width="300" height="300" frameborder="0" scrolling="no"
marginheight="0" marginwidth="0" src="http://mapy.google.pl..."></iframe>
-->
```

Tym razem mapa będzie użyteczna

Lokalizacja, lokalizacja i jeszcze raz lokalizacja.

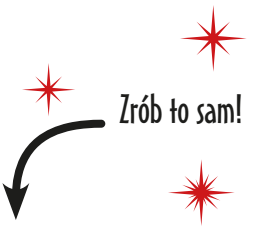
To stare powiedzenie agentów nieruchomości zyskuje nowe znaczenie w kontekście urządzeń mobilnych. Ponieważ większość z nich oferuje możliwość określenia położenia za pomocą GPS-u lub innych metod lokalizacyjnych, coraz powszechniejsze staje się dostarczanie treści dostosowanych do lokalizacji użytkownika.

Michał usunął mapę ze strony w wersji dla urządzeń mobilnych, ponieważ była zbyt duża. Teraz, kiedy już wiemy, jak wiele zasobów jest pobieranych w związku z mapą, najlepiej byłoby z niej zrezygnować.

Jednak z drugiej strony mapa by się przydała. Proponujemy kompromis dla urządzeń o wąskich ekranach — zamiast umieszczać mapę, dołączymy tylko odsyłacz do niej.

Dodajemy odsyłacz do mapy

Będzie nam potrzebny blok `<div>`, do którego możemy się odwołać z poziomu JavaScriptu. W tym bloku umieścimy znacznik `<p>` zawierający odsyłacz do mapy. Jak myślisz, dlaczego zastosowaliśmy akurat takie rozwiązanie?



Będzie nam potrzebny kontener, do którego możemy się odwołać z poziomu JavaScriptu, więc dodajemy blok .

Dzięki identyfikatorowi znacznika <p> będziemy mogli wstawić ramkę iframe nad linkiem w przypadku szerszych ekranów. Zajmiemy się tym już wkrótce.

Nowy znacznik wstaw przed kodem ramki iframe umieszczonej w komentarzu.

```
<div id="mapcontainer">
   <p id="maplink">
      <a href="http://mapy.google.pl/maps?q=Browarna+131,+Cieszyn&hl=pl&ie=UTF
8&sll=49.749306,18.63049&t=m&view=map&z=18">Mapa</a>
   </p>
</div>
<!--
<iframe id="map" width="300" height="300" frameborder="0"
scrolling="no" marginheight="0" marginwidth="0"
src="http://mapy.google.pl..."></iframe>
-->
```

Budujemy pseudozapytanie o media w JavaScripcie

Rzuć okiem na kod JavaScript, którego zamierzamy użyć do umieszczenia ramki `iframe` na stronie. Ten kod działa podobnie jak proste zapytanie o media.

Ta zmienna przechowuje identyfikator elementu, w którym chcemy umieścić mapę. Korzystamy ze zmiennej, by ułatwić ewentualną zmianę kontenera w przyszłości.

Ustalamy wartość zmiennej breakpoint na 481 pikseli. Wartość ta to szerokość, powyżej której na stronie jest umieszczana mapa.

Sprawdzamy, czy szerokość okna jest większa niż wartość zmiennej breakpoint.

Ten fragment odpowiada za ustawienie wszystkich niezbędnych atrybutów elementu iframe. Nazwy atrybutów i ich wartości zostały skopiowane z dotychczasowego kodu osadzającego mapę.

Na samym końcu utworzony element iframe (mapElement) wstawiamy do bloku o identyfikatorze mapcontainer przed akapitem zawierającym odsyłacz do mapy (maplink).

Tworzymy nowy element iframe i przypisujemy go do zmiennej mapElement.

Ten adres URL nie wygląda zbyt przyjaźnie. Zamiast go przepisywać, skopiuj go z pliku extras/map.js.

```
<script type="text/javascript">
var breakpoint = 481,
    id = 'mapcontainer',
    viewportWidth = window.innerWidth;
if (viewportWidth > breakpoint) {
    var mapElement = document.createElement('iframe');
    mapElement.id = 'map';
    mapElement.width = '300';
    mapElement.height = '300';
    mapElement.frameborder = '0';
    mapElement.scrolling = 'no';
    mapElement.marginheight = '0';
    mapElement.marginwidth = '0';
    mapElement.src = 'http://mapy.google.pl/maps?f=q&amp;source=
s_q&amp;hl=pl&amp;geocode=&amp;q=Browarna+131,+Cieszyn&amp;aq=&a
mp;sll=49.749306,18.63049&amp;sspn=0.002378,0.003809&amp;ie=UTF8
&amp;hq=&amp;hnear=Browarna,+43-400+Cieszyn,+cieszy%C5%84ski,+%C
5%9Bl%C4%85skie&amp;ll=49.749269,18.630416&amp;spn=0.002378,0.00
3809&amp;t=m&amp;z=14&amp;output=embed';
    document.getElementById(id).insertBefore(mapElement,
maplink);
}
</script>
```

Usuwamy kod ramki umieszczony w komentarzu

Oryginalny kod wstawiający mapę nie jest już nam potrzebny, więc usuń go z dokumentu HTML.

Usuń ten fragment!

```
<!--
    <iframe id="map" width="300" height="300" frameborder="0" scrolling="no"
marginheight="0" marginwidth="0" src="http://maps.google.com..."></iframe>
-->
```

Wstawiamy skrypt na stronę

Teraz musimy wstawić ten skrypt JavaScript na stronę. Ponieważ mapa jest miłym, ale niekoniecznie niezbędnym dodatkiem, kod, który służy do jej umieszczenia, będzie jednym z ostatnich elementów strony.

Otwórz plik ontap.html i przejdź na koniec dokumentu.

Skrypt umieszczamy na samym końcu, tuż przed zamykającym znacznikiem </body>.

```
<div class="footer">
   <p>Do miłego zobaczenia! Pub Pod Paradnym Morsem</p>
</div>
[TU WSTAW SKRYPT]
</body>
</html>
```

Umieszczanie mniej ważnych skryptów JavaScript na końcu dokumentu HTML sprawia, że strona ładuje się szybciej. Dzięki temu przeglądarka najpierw parsuje HTML i CSS, a dopiero później zajmuje się JavaScriptem. W ten sposób użytkownicy szybciej dostaną działającą stronę, bez konieczności oczekiwania na załadowanie mapy.

Ćwiczenie

Pora na sprawdzenie efektów pracy. Przetestuj zaktualizowany plik *ontap.html* i odpowiedz na następujące pytania:

1. **Czy pliki JavaScript są pobierane na urządzeniach mobilnych?**
   Załaduj stronę na iPhonie i Androidzie, korzystając z serwisu *www.blaze.io/mobile/*. Przejrzyj wykres kaskadowy, by sprawdzić, czy są pobierane skrypty powiązane z mapą Google'a. Jeśli nie możesz umieścić opracowanej samodzielnie strony na publicznym serwerze, skorzystaj z wersji dostępnej pod adresem: *ftp://ftp.helion.pl/online/inne/hfmw/r02/5/ontap.html*.

2. **Czy mapa pojawia się na większych ekranach?**
   Stronę Co dziś lejemy otwórz w ulubionej przeglądarce desktopowej. Czy mapa jest widoczna w oknach szerszych niż 480 pikseli?

3. **Jak mapa wpasowuje się w nasz projekt zgodny z RWD?**
   Zmieniaj rozmiar okna przeglądarki. Czy mapa skaluje się wraz z pozostałą częścią projektu? Czy pojawiają się jakieś problemy z mapą?

I jak? Jakieś problemy?

1 **Czy pliki JavaScript są pobierane na urządzeniach mobilnych?**

Rozwiązanie ćwiczenia

GET ontap.html	200	1.4 KB
GET style.css	200	975 B
GET layout.css	200	279 B
GET bensons_bubbler.jpg	200	26.8 KB
GET chapman_lownsdale.jpg	200	18.6 KB
GET crystal_springs.jpg	200	21.4 KB
GET hoyt.jpg	200	22.8 KB
GET mill_ends.jpg	200	22.7 KB
GET milo_mciver.jpg	200	24.3 KB
GET mount_tabor.jpg	200	21.3 KB
GET omsi.jpg	200	27.1 KB
GET oxbow.jpg	200	23.8 KB
GET pittock.jpg	200	23.9 KB
GET powells.jpg	200	22.2 KB
GET sandy_river.jpg	200	29 KB
GET sauvie_island.jpg	200	21.1 KB
GET the_grotto.jpg	200	25.9 KB
GET tryon_creek.jpg	200	28.6 KB
GET wells_fargo.jpg	200	22.7 KB
19 Requests		384.8 KB

Spójrzcie, nie jest pobierany żaden plik mapy Google'a!

Załaduj stronę na iPhonie i Androidzie, korzystając z narzędzia Mobitest. Aby sprawdzić, jakie pliki zostały pobrane, przejdź na stronę szczegółowego wykresu kaskadowego, klikając odsyłacz do pliku HAR.

Pliki powiązane z mapą nie są pobierane. Świetnie!

A co z Androidem? Z wykresu wynika, że pliki są pobierane, ale to jest, podobnie jak ostatnio, fałszywa informacja.

Wspomnieliśmy, że firma Blaze musiała zmodyfikować telefony, by umożliwiały zdalne przeprowadzanie testów. Jednym z efektów ubocznych modyfikacji jest to, że skrypty JavaScript uruchamiane na tym urządzeniu informują o znacznie szerszym ekranie (800 pikseli) niż w rzeczywistości.

Musisz nam uwierzyć na słowo, że wszystko działa jak należy i pliki mapy nie są pobierane.

2 **Czy mapa pojawia się na większych ekranach?**

Tak. W przeglądarce desktopowej mapa prezentuje się wyśmienicie.

Mapa pojawia się na stronie, co oznacza, że przygotowany skrypt JavaScript działa.

3 **Jak mapa wpasowuje się w nasz projekt zgodny z RWD?**

Nie jest dobrze — mamy z tym kilka problemów. Po pierwsze, mapa nie jest skalowana wraz z pozostałymi elementami projektu. A po drugie, przy pewnych szerokościach ekranu mapa zasłania etykiety piwa.

A niech to! Po zmniejszeniu szerokości okna przeglądarki mapa zaczyna zasłaniać etykiety piwa.

Dlaczego mapa nie skaluje się tak samo jak obrazki?

Ten widżet nie jest zgodny z RWD

Kod ramki iframe ładującej mapę Google'a nie jest zaprojektowany tak, by była płynna. Szerokość jest ustalona na stałe w kodzie i wynosi 300 pikseli. Założę się, że wystarczy zmodyfikować ramkę w taki sposób, by korzystała z CSS, a od razu stanie się płynna.

Podejście Responsive Web Design jest na tyle nowe, że wiele widżetów, w tym również mapa Google'a, nie jest domyślnie płynnych.

Firmy dostarczające widżety starają się opracować je tak, by wyglądały dobrze bez względu na układ strony, na której są osadzane. To często pociąga za sobą konieczność ustalania na stałe szerokości (`width`) oraz wysokości (`height`) w kodzie HTML.

Posługiwanie się nie najlepiej opracowanymi widżetami stwarza problemy w zdecydowanej większości mobilnych witryn. Zastosowanie podejścia RWD wiąże się z koniecznością spełnienia wielu warunków, nie tylko tych związanych z płynnością widżetów.

Szerokość i wysokość są ustalone na stałe, co uniemożliwia skalowanie mapy.

```
<iframe id="map" width="300" height="300" frameborder="0" scrolling="no"
marginheight="0" marginwidth="0" src="http://mapy.google.pl..."></iframe>
```

Wiele z tych atrybutów można przenieść do CSS.

Byłoby idealnie, gdyby kod HTML zawierał jedynie treść i znaczniki. Na pewno nie powinien zawierać informacji dotyczących prezentacji.

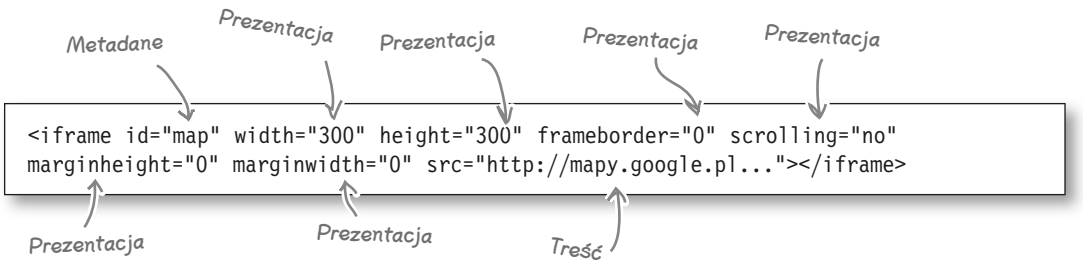
WYSIL SZARE KOMÓRKI

Jakie właściwości CSS są odpowiednikami atrybutów zastosowanych w znaczniku `iframe`?

Przenosimy atrybuty do CSS

Przeniesiemy teraz jak najwięcej atrybutów ramki `iframe` do arkusza stylów, zmieniając je przy okazji tak, by mapa stała się płynna.

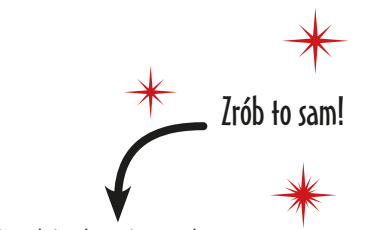
Przede wszystkim musimy określić listę atrybutów, które możemy przenieść do arkusza stylów. Na liście powinny się znaleźć tylko te, które są związane z prezentacją, natomiast pozostałe — powiązane z treścią i metadanymi — powinny pozostać w dokumencie HTML.



Wszystkie zaznaczone atrybuty związane z prezentacją przeniesiemy do CSS.

Atrybuty zamieniamy na właściwości stylów

Niektóre atrybuty mają takie same nazwy jak ich odpowiedniki w arkuszu stylów. Znalezienie CSS-owych wersji atrybutów `width` i `height` nie jest szczególnie trudne, ale w przypadku `frameborder` możemy już mieć pewne wątpliwości. Na szczęście w większości przypadków właściwości CSS mają dosyć oczywiste nazwy.



Atrybuty elementu iframe.

```
<iframe
    id="map"
    width="300"
    height="300"
    frameborder="0"
    scrolling="no"
    marginheight="0"
    marginwidth="0"
    src="http://mapy.
google.pl...">
</iframe>
```

Nowe reguły CSS dla ramki zawierającej mapę.

```
#map {
    width:100%;
    height:100%;
    border:none;
    overflow:hidden;
    margin:0;
}
```

Aby ramka z mapą stała się płynna, zamiast ustalać wymiary na stałe w pikselach, stosujemy wartości procentowe (mówiliśmy o tym w rozdziale 1.).

W arkuszu stylów zachowanie podczas przewijania jest kontrolowane przez właściwość overflow. W taki sposób wskazujemy, że zawartość niemieszcząca się w elemencie jest ukrywana i nie pojawiają się paski przewijania. Rezultat jest taki sam jak w przypadku ustawienia atrybutu scrolling="no".

Usuwamy atrybuty z JavaScriptu

Mamy już odpowiednie reguły CSS, więc ze skryptu JavaScript możemy usunąć fragment odpowiedzialny za ustawianie atrybutów powiązanych z prezentacją. Usuń wszystkie te wiersze kodu, w których są definiowane wskazane atrybuty.

Ten kod JavaScript znajdziesz na końcu pliku ontap.html.

Usuń ten fragment ze skryptu.

```
<script type="text/javascript">
var breakpoint = 481,
    id = 'mapcontainer',
    viewportWidth = window.innerWidth;
if (viewportWidth > breakpoint) {
    var mapElement = document.createElement('iframe');
    mapElement.id = 'map';
    mapElement.width = '300';
    mapElement.height = '300';
    mapElement.frameborder = '0';
    mapElement.scrolling = 'no';
    mapElement.marginheight = '0';
    mapElement.marginwidth = '0';
    mapElement.src = 'http://mapy.google.pl/maps?f=q&amp;source
=s_q&amp;hl=pl&amp;geocode=&amp;q=Browarna+131,+Cieszyn&amp;aq
=&amp;sll=49.749306,18.63049&amp;sspn=0.002378,0.003809&amp;ie
=UTF8&amp;hq=&amp;hnear=Browarna,+43-400+Cieszyn,+cieszy%C5%84
ski,+%C5%9Bl%C4%85skie&amp;ll=49.749269,18.630416&amp;spn=0.00
2378,0.003809&amp;t=m&amp;z=14&amp;output=embed';
    document.getElementById(id).insertBefore(mapElement,
maplink);
}
</script>
```

Jazda próbna

Zapisz pliki *layout.css* i *ontap.html*. W ulubionej przeglądarce otwórz stronę Co dziś lejemy. Jeżeli wolisz skorzystać z naszej, poprawionej wersji, przejdź na *ftp://ftp.helion.pl/online/inne/hfmw/r02/6/ontap.html*. Jak teraz wygląda mapa?

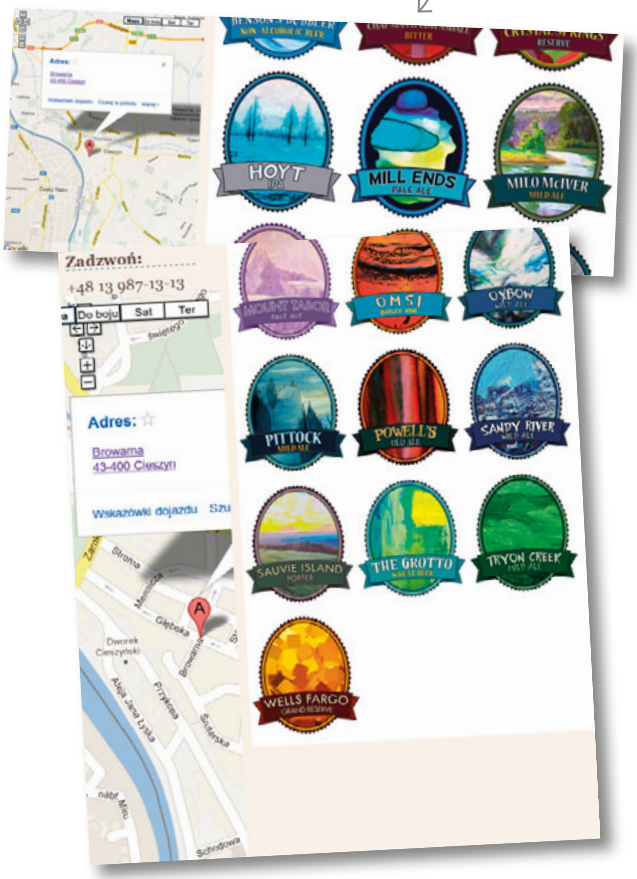
Teraz nikt nie powinien mieć problemu ze znalezieniem pubu

Nie wydaje Ci się, że na tej stronie jest trochę za dużo mapy? Zajmuje prawie całą lewą kolumnę. Niedługo zacznie krzyczeć „Halo, to ja, Twoja mapa!”, żeby zwrócić Twoją uwagę, chyba że ją skutecznie przystopujesz.

Żeby się dowiedzieć, dlaczego mapa miałaby zacząć krzyczeć, wystarczy, że zmniejszysz szerokość okna przeglądarki. Mapa się kurczy, aż w końcu zostaje z niej tylko wąski, zupełnie bezużyteczny pasek.

Nadal chcemy, by mapa się skalowała, ale musimy określić rozsądne granice, w których mogą się zmieniać jej wymiary.

Przedstawiliśmy tu zrzuty ekranu z przeglądarki Chrome. Inne przeglądarki mogą się zachowywać inaczej, ale na żadnej z nich mapa nie zachowuje się właściwie.



1 **Wysokość mapy jest za duża.**
Z powodu ustawienia wysokości mapy na 100% jej wysokość jest przeważnie większa niż szerokość. Spróbujmy to skorygować, ustalając na stałe wysokość na 400 pikseli.

Umieść ten wiersz w regule CSS dla elementu #map.

```
height: 400px;
```

2 **Patrzcie, mapa nosi przyciasne spodnie.**
Podczas zmniejszania szerokości okna przeglądarki mapa staje się na tyle wąska, że niewiele na niej widać. Musimy określić minimalną szerokość, żeby mapa nie stawała się na naszych oczach chuda jak tyczka.

Po dodaniu opisanych dwóch wierszy reguła dla elementu #map powinna wyglądać tak.

```
#map {
    width:100%;
    height:400px;
    border:none;
    overflow:hidden;
    margin:0;
    min-width: 200px;
}
```

Mapa znów zasłania treść strony

Myślałam, że głównym powodem, dla którego ramka miała stać się płynna, było uniknięcie zachodzenia mapy na inne obszary strony. Mamy już wszystko w arkuszu stylów, ale mapa nadal zasłania etykiety piwa, gdy okno przeglądarki jest wąskie.

Przeniesienie atrybutów elementu iframe powiązanych z prezentacją do arkusza stylów było pierwszym etapem. Teraz musimy spojrzeć jeszcze raz na zapytania o media.

W rozdziale 1. stosowaliśmy zapytania o media do przełączania układu przy szerokości równej 480 pikseli. Określiliśmy tę wartość na podstawie szerokości ekranu popularnych smartfonów.

Obecne zachowanie mapy wskazuje, że przed podjęciem decyzji o tym, w jakiej sytuacji zastosować zapytania o media, musimy się przyjrzeć zawartości strony.

Gdy ekran jest wąski, mapa znów zasłania etykiety piwa.

Niech prowadzi Cię zawartość strony

Ustalenie wartości granicznej dla zapytania o media równej 480 pikseli rodzi pewne problemy. Telefony mają ekrany o różnej szerokości. A nawet jeśli w tej chwili w większości przypadków jest to 480 pikseli, to skąd wiesz, czy w przyszłości standardem nie będzie na przykład 540 pikseli?

Lepszym podejściem jest zdanie się podczas zmiany układu na zawartość strony.

Nie prosimy Cię jednak, byś zjednoczył się z zawartością strony i odczytywał jej wewnętrzny przekaz. Ale jeśli będziesz ustalał rozmiar okna przeglądarki, dopóki nie uzyskasz zadowalającego efektu, zawartość strony może Ci powiedzieć całkiem sporo.

Być może obrazy są zbyt małe? A może kolumny za wąskie?

Kiedy zauważysz tego typu problemy, oznacza to, że przy tej szerokości powinien się zmienić układ. Dzięki temu wiesz, jak zdefiniować zapytania o media.

Rozciągamy i zmniejszamy przeglądarkę

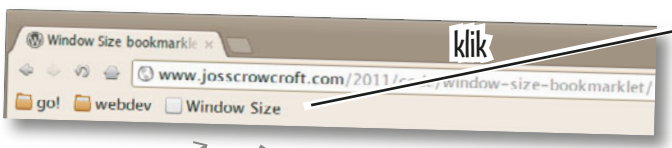
Musimy sprawdzić zachowanie strony, maksymalnie ją rozciągając i zwężając, a przy tym cały czas obserwując układ.

Jednak zanim przystąpimy do działania, musimy znaleźć sposób na sprawdzenie, przy jakich wymiarach ekranu z układem strony dzieje się coś złego. Najprostszym sposobem jest zainstalowanie tzw. **bookmarkletu**, który będzie na bieżąco wyświetlał wymiary okna.

Bookmarklet to niewielki skrypt JavaScript zapisany w zakładkach przeglądarki.

Zainstaluj bookmarklet w przeglądarce

Aby utworzyć bookmarklet, przejdź na stronę *http://bit.ly/window-resize* i przeciągnij odsyłacz *Window Size* do paska zakładek w swojej przeglądarce, a następnie kliknij, aby go uruchomić. W lewym górnym rogu okna przeglądarki jest teraz wyświetlany jego aktualny rozmiar.



Zakładka Window Size w pasku zakładek w przeglądarce Chrome.

Po kliknięciu bookmarkletu Window Size w lewym górnym rogu okna przeglądarki pojawia się informacja o wymiarach.

Inna możliwość — zainstaluj dodatek

W przeglądarce Google Chrome można zainstalować dodatek, który nie tylko pokazuje rozmiar okna, ale umożliwia również przeskalowanie go do wybranych rozdzielczości ekranu. Dodatek możesz pobrać ze strony *http://bit.ly/chrome-resizer*.

Z kolei dodatek Web Developer Toolkit (*http://bit.ly/webdevtoolkit*) wyświetla rozmiar strony w pasku tytułowym przeglądarki, a także oferuje masę innych przydatnych narzędzi. Działa w przeglądarkach Firefox i Chrome.

Zaostrz ołówek

Otwórz stronę Co dziś lejemy w przeglądarce z zainstalowanym bookmarkletem Window Size (lub innym tego typu dodatkiem). Uruchom bookmarklet i przeskaluj okno przeglądarki.

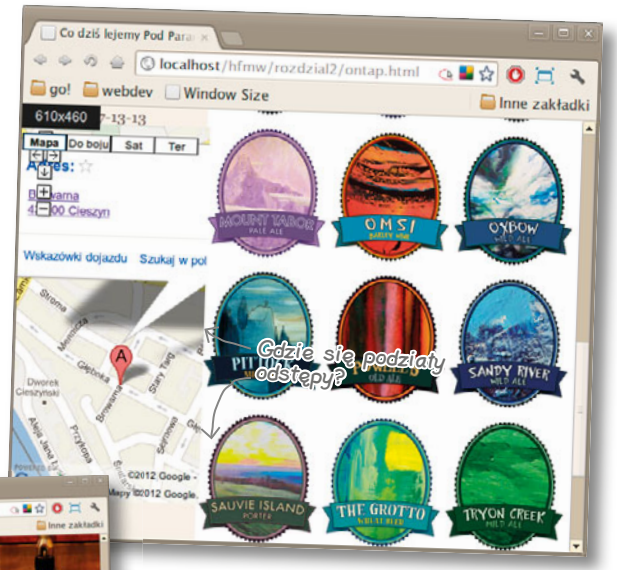
Zapisz szerokość okna przeglądarki, przy której układ strony się psuje lub zawartość przestaje wyglądać dobrze.

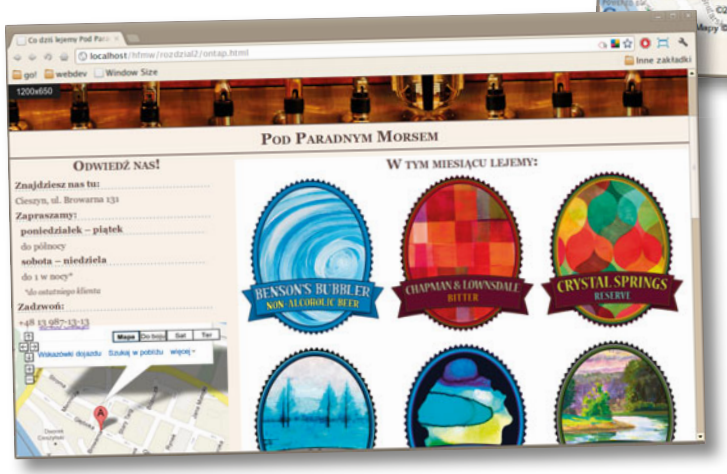
Zaostrz ołówek
Rozwiązanie

Omówmy niektóre z problemów, które pojawiają się podczas skalowania okna przeglądarki.

610 pikseli — etykiety piwa stykają się z mapą.

Przy szerokości okna w okolicach 610 pikseli etykiety piwa stykają się z mapą. Gdybyśmy chcieli zdefiniować nową wartość graniczną rozwiązującą ten problem, musielibyśmy uwzględnić margines poprawnego wyświetlania. To oznacza, że zamiast wartości 610 powinniśmy zastosować 640 pikseli.





1200 pikseli — ogromne etykiety piwa.

W miarę zwiększania szerokości okna przeglądarki etykiety piwa stają się przesadnie duże. *Moment*, w którym stają się za duże, to sprawa gustu. Naszym zdaniem jest to szerokość w okolicach 1200 pikseli.

WYSIL SZARE KOMÓRKI

Czy podczas skalowania okna przeglądarki dostrzegłeś inne problemy? Jak sądzisz, na ile istotne muszą one być, by należało im przeciwdziałać za pomocą dodatkowych zapytań o media?

Wartości graniczne przybywają na ratunek

W gruncie rzeczy nie jest aż tak źle. Kilka sprytnych poprawek w arkuszu stylów powinno pomóc.

Zmniejszamy ogromne etykiety piwa

W tej chwili w jednym wierszu wyświetlamy trzy etykiety. Gdy strona staje się szersza, bez problemu znalazłoby się miejsce na jeszcze jedną.

W związku z tym utworzymy kolejne zapytanie o media dla okien szerszych niż 1200 pikseli, w którym ustalimy, że w jednym wierszu mają zostać wyświetlone cztery etykiety.

Ta zmiana zostanie zastosowana tylko na ekranach o szerokości większej niż 1200 pikseli.

Dzięki ustaleniu szerokości elementu listy (li) zawierającego etykietę piwa na 25% w jednym wierszu zostaną wyświetlone cztery etykiety.

```
@media screen and (min-width:1201px) {
    .taplist li {
        width: 25%;
    }
}
```

Tę regułę dodaj do pliku layout.css.

Bardzo często w miarę wzrostu szerokości ekranu zmniejsza się rozmiar obrazów.

Wcześniej przechodzimy na układ z jedną kolumną

Nawet jeśli etykiety piwa nie są zasłaniane przez mapę, przy szerokości rzędu 640 pikseli układ staje się bardzo „ciasny”. Zamiast definiować kolejną wartość graniczną, możemy zmodyfikować istniejące zapytanie o media, tak by zmieniało układ przy 640, a nie przy 480 pikselach.

Wartość 480 pikseli mamy w HTML-u, JavaScripcie i arkuszu stylów, więc musimy to zmienić we wszystkich trzech miejscach.

Wprowadzenie tej zmiany spowoduje przełączanie na układ jednokolumnowy przy wartości 640 pikseli. Ma to tę zaletę, że jednokolumnowy układ zostanie zastosowany również na telefonach o większych ekranach.

Plik ontap.html

```
<link rel="stylesheet" type="text/css" href="layout.css"
media="all and min-width: 641px)">
```

Właściwość min-width ustawiamy na 641 pikseli.

```
<script type="text/javascript">
var breakpoint = 641,
    ...
</script>
```

Wartość graniczną ustawiamy na 641 pikseli.

Plik taps.css

```
/* Urządzenia mobilne z mniejszymi ekranami
*/
@media screen and (max-width:640px)
{
```

Właściwość max-width ustawiamy na 640 pikseli.

Zakończyliśmy nasz projekt zgodny z techniką Mobile First

- Do maksimum uprościmy dokument HTML i zmienimy kolejność w arkuszu CSS, tak by wersja dla urządzeń mobilnych była pierwsza.
- Poprawimy obrazy tła w arkuszu stylów, tak by na jeden obraz był pobierany tylko jeden plik. Upewnimy się również, czy poprawnie stosujemy `display:none`.
- Dostarczymy różne pliki dla znaczników `<img>` w zależności od rozmiaru ekranu. Upewnimy się, czy są pobierane prawidłowe pliki.
- Do osadzenia mapy Google'a użyjemy JavaScriptu, tak by mapa była umieszczana tylko wtedy, gdy przeglądarka potrafi ją obsłużyć, a ekran jest na tyle szeroki, by ją wyświetlić.

Nie istnieją głupie pytania

P: Jak dokładnie rozumieć, czym jest widoczna część ekranu?

O: Wyobraź sobie kartonowy arkusz z wyciętym pośrodku prostokątem. Przyłóż ten karton do monitora, tak byś widział tylko część wyświetlanej strony. To jest właśnie widoczna część ekranu.

P: Zatem znacznik `<meta> viewport` określa rozmiar widocznej części ekranu, czy tak?

O: Właśnie. Domyślnie system iOS ustawia szerokość na 980 pikseli. Jeśli optymalizujesz stronę na potrzeby mniejszych ekranów, powinieneś to zasygnalizować przeglądarce właśnie za pomocą znacznika `<meta> viewport`.

P: Do czego służą wartości graniczne?

O: Za pomocą wartości granicznych możesz określić wymiary okna, przy których zmienia się układ strony. Zwykle realizuje się to poprzez zapytania o media, które sprawdzają, czy bieżące okno jest szersze, czy węższe niż wielkość ustalona w pikselach.

W złożonych projektach może występować wiele wartości granicznych. Niektóre z nich mogą powodować gruntowne zmiany układu, a inne mogą wprowadzać drobne zmiany poprawiające niewielkie błędy w układzie strony.

P: Nie chcę blokować możliwości skalowania strony, ale błąd w systemie iOS mi w tym skutecznie przeszkadza. Czy są sposoby na obejście tego problemu?

O: Jedno z rozwiązań, wykorzystujące JavaScript, znajdziesz pod adresem *https://gist.github.com/901295*.

P: Dlaczego mapa zasłania pozostałe elementy strony?

O: Ponieważ mapa jest elementem, który nie jest skalowany wraz z oknem przeglądarki. Gdy okno jest małe, przeglądarka nie może zmniejszyć mapy, więc w rezultacie lewa kolumna zaczyna zasłaniać prawą.

P: Czy dodanie właściwości `min-width` do mapy jest prawidłowe z punktu widzenia podejścia RWD? Przecież w ten sposób tworzymy element, który nie jest skalowany wraz z oknem przeglądarki.

O: Technicznie rzecz biorąc, tak. Jednak w tym przypadku jest to dopuszczalne rozwiązanie, ponieważ zmodyfikowaliśmy zapytania o media w celu uniknięcia zasłaniania zawartości. Inną możliwością byłoby zastosowanie zapytań o media ustalających wymiary mapy i proporcje szerokości kolumn.

CELNE SPOSTRZEŻENIA

- Dodanie zapytań o media do istniejącej witryny desktopowej może sprawić, że zacznie wyglądać dobrze na urządzeniach mobilnych, ale wcale nie oznacza, że staje się **zoptymalizowana pod kątem tych urządzeń**.
- Ponieważ **większość przeglądarek mobilnych nie obsługuje dodatków**, podczas tworzenia mobilnych stron mamy do dyspozycji znacznie mniej narzędzi.
- Użycie **serwera proxy** lub narzędzia takiego jak Blaze Mobitest może pomóc w sprawdzeniu, **co tak naprawdę jest pobierane przez mobilną przeglądarkę**.
- Podstawowymi narzędziami umożliwiającymi poprawę wydajności są **pliki HAR** (ang. *HTTP Archive*) i **wykresy kaskadowe**.
- Koncepcja **Mobile First** w podejściu **Responsive Web Design** pomaga w zoptymalizowaniu stron internetowych, tak by **domyślnie były pobierane mniejsze zasoby**.
- Mobile First RWD jest jedną z form **stopniowego ulepszania**, w której bazuje się na rozmiarze ekranu.
- Podczas projektowania w zgodzie z koncepcją Mobile First **skupiamy się na tym, co jest naprawdę istotne**, a to pomaga usunąć ze stron niepotrzebny balast.
- Internet Explorer 8 i jego wcześniejsze wersje nie obsługują zapytań o media. Możliwym obejściem są **komentarze warunkowe**.
- Za pomocą **JavaScriptu można wspomóc zapytania o media**, sprawdzając rozmiar ekranu i dodając do strony wymagane elementy.
- Zamiast stosować w projekcie wartości graniczne wynikające z typowych rozmiarów ekranu, **lepiej określić je na podstawie zachowania zawartości strony**, obserwując zmiany układu.

3. Oddzielna witryna mobilna

Stawiamy czoła niezupełnie sprzyjającym okolicznościom

Wizja jednego, wrażliwego projektu witryny jest cudowna... Mamy tylko jeden układ strony opracowany w zgodzie z koncepcją Mobile First, który dopasowuje się do specyfiki różnych przeglądarek i urządzeń. Brzmi pięknie. Co się jednak stanie, gdy tę wizję przyprawimy choćby szczyptą realizmu? Niezaktualizowane systemy, stare urządzenia lub ograniczenia budżetu klienta to tylko kilka przykładów. A co, jeśli zamiast łączyć wersję desktopową z mobilną, chciałbyś je rozdzielić? W tym rozdziale przyjrzymy się szczegółowo **wykrywaniu użytkowników korzystających z urządzeń mobilnych, wspieraniu starszych telefonów i tworzeniu odrębnych witryn dla urządzeń mobilnych.**

Agenci organizacji Zwierzętom na Pomoc patrolują pola

Zwierzętom na Pomoc to stowarzyszenie non profit stawiające za cel niesienie pomocy chorym i okaleczonym zwierzętom hodowlanym, a także wspomaganie rolników, którzy ucierpieli w następstwie kataklizmów. Do tej pory organizacja bazowała na bezpośredniej komunikacji, choć od czasu do czasu agenci otrzymywali specjalne, wzmocnione laptopy odporne na działanie czynników atmosferycznych, ułatwiające zarządzanie personelem i dostawami.

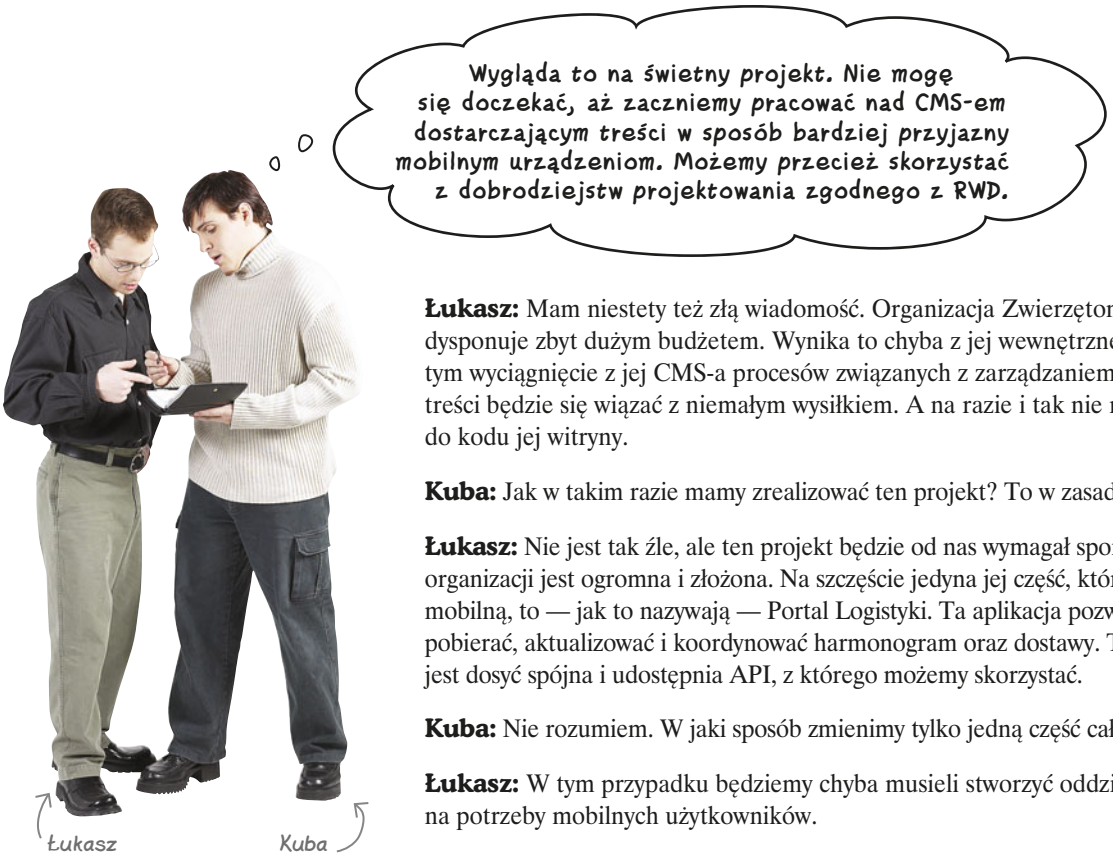
Jak agenci mogą otrzymywać i przekazywać informacje?

Zwierzętom na Pomoc nie jest nową organizacją — jej historia sięga dwóch dekad wstecz. Większość wewnętrznej struktury jest już dawno ustalona. Jest też obecna w sieci, ale jej witryna jest oparta na tradycyjnym systemie zarządzania treścią (CMS).

Rosnąca potrzeba witryny mobilnej

Coraz częściej okazuje się, że do komunikacji w terenie personel organizacji Zwierzętom na Pomoc korzysta z jedynej dostępnej metody, czyli telefonii komórkowej. W takich warunkach trudno jest znaleźć połączenie z internetem, poza tym wymaga to dodatkowego wyposażenia, a obecna witryna zupełnie nie odpowiada możliwościom urządzeń mobilnych.

Jednym słowem, organizacja potrzebuje mobilnej witryny. Takiej, z której można korzystać na bardzo odmiennych urządzeniach przy różnej jakości połączeniach.



Łukasz: Mam niestety też złą wiadomość. Organizacja Zwierzętom na Pomoc nie dysponuje zbyt dużym budżetem. Wynika to chyba z jej wewnętrznej polityki. Poza tym wyciągnięcie z jej CMS-a procesów związanych z zarządzaniem i publikowaniem treści będzie się wiązać z niemałym wysiłkiem. A na razie i tak nie możemy zajrzeć do kodu jej witryny.

Kuba: Jak w takim razie mamy zrealizować ten projekt? To w zasadzie niemożliwe...

Łukasz: Nie jest tak źle, ale ten projekt będzie od nas wymagał sporo pracy. Witryna organizacji jest ogromna i złożona. Na szczęście jedyna jej część, która ma mieć wersję mobilną, to — jak to nazywają — Portal Logistyki. Ta aplikacja pozwala agentom pobierać, aktualizować i koordynować harmonogram oraz dostawy. Ta część witryny jest dosyć spójna i udostępnia API, z którego możemy skorzystać.

Kuba: Nie rozumiem. W jaki sposób zmienimy tylko jedną część całej witryny?

Łukasz: W tym przypadku będziemy chyba musieli stworzyć oddzielną witrynę na potrzeby mobilnych użytkowników.

Kuba: Nie wygląda to najlepiej...

Łukasz: Tworzenie mobilnej witryny faktycznie nie będzie łatwe. Wiesz to tak samo dobrze jak ja. Ale musimy się tym zająć i zrobić to porządnie — w końcu będzie z tego korzystać wiele osób z całego świata. Niestety już widzę pewne problemy. Nie licz na to, że przyszli użytkownicy mają najnowsze smartfony z najwyższej półki. Agenci często korzystają z urządzeń otrzymanych przez organizację z darowizn, więc często są to stare, wysłużone telefony, a z dostępem do internetu na niektórych terenach bywa różnie. Do naszego projektu nie możemy tak po prostu przenieść istniejących desktopowych rozwiązań.

Czasami utworzenie oddzielnej mobilnej witryny jest uzasadnione.

Boisz się programowania? Bez stresu...

Co prawda w rozmowie Łukasza z Kubą pojawiły się pojęcia „API ” i „aplikacja internetowa”, ale nic się nie bój — programowaniem zajmiemy się my. Ty nam pomożesz w poprawieniu wyglądu i dopilnowaniu, by aplikacja dobrze działała na urządzeniach mobilnych.

Nie istnieją głupie pytania

P: Czym jest system zarządzania treścią (CMS)?

O: CMS to narzędzie umożliwiające edytowanie i publikowanie treści udostępnianych w internecie. Niektóre CMS-y są rozbudowanymi systemami dostarczającymi środowisko dla programistów, dzięki któremu możliwe jest szybkie tworzenie aplikacji (tego typu rozwiązania nazywa się czasem *Content Management Frameworks* — CMF, czyli frameworkami zarządzania treścią).

CMS-y umożliwiają użytkownikom — którzy niekoniecznie są zaznajomieni z HTML-em i projektowaniem witryn internetowych — tworzenie treści i zarządzanie nią.

Dostępne są rozwiązania stworzone w przeróżnych technologiach (i to zarówno na zasadach open source, jak i w wersjach komercyjnych). Przykładami mogą być: WordPress, Drupal, DotNetNuke, Joomla! czy SharePoint. Na potrzeby większych lub bardziej wyspecjalizowanych organizacji często są tworzone dedykowane CMS-y.

Większość systemów zarządzania treścią korzysta z mechanizmu szablonów lub innych tego typu rozwiązań. To może powodować problemy podczas przechodzenia na wersję mobilną, ponieważ często warstwa treści jest wymieszana z warstwą prezentacji.

Tak właśnie jest w przypadku witryny Zwierzętom na Pomoc. Ich CMS, zaprojektowany wiele lat temu, oferuje tylko jeden szablon. Modyfikowanie systemu od podstaw jest w tej chwili kompletnie nieopłacalne.

P: Czy zaadaptowanie każdego systemu zarządzania treścią na potrzeby urządzeń mobilnych jest trudne?

O: Liczba dostępnych CMS-ów jest przeogromna. Niektóre z nich łatwiej zaadoptować na potrzeby urządzeń mobilnych niż inne. Musisz wiedzieć, że większość popularnych CMS-ów boryka się z tym samym problemem wymieszania warstw treści, logiki i prezentacji.

Społeczności programistów i firmy stojące za wieloma CMS-ami aktywnie pracują nad dopasowaniem ich produktów do udostępniania treści dla różnych urządzeń. Wizjonerzy rozwoju mobilnych technologii internetowych obmyślają nowe sposoby budowania struktury dokumentów, tak by treść była traktowana w bardziej uporządkowany sposób, podobnie jak w przypadku danych w aplikacjach. Dzięki temu wielokrotne wykorzystanie danych na wielu różnych platformach stałoby się dużo prostsze.

Zbliżenie na API

API (ang. *Application Programming Interface*), czyli **interfejs programowania aplikacji**, to usystematyzowany i zdefiniowany interfejs umożliwiający wzajemną komunikację między oprogramowaniem. Jednym z popularnych interfejsów API dostępnych w internecie jest API Twittera. Jest w nim zdefiniowany zbiór metod, za pomocą których programiści mogą pobierać i modyfikować dane z serwisu Twitter.

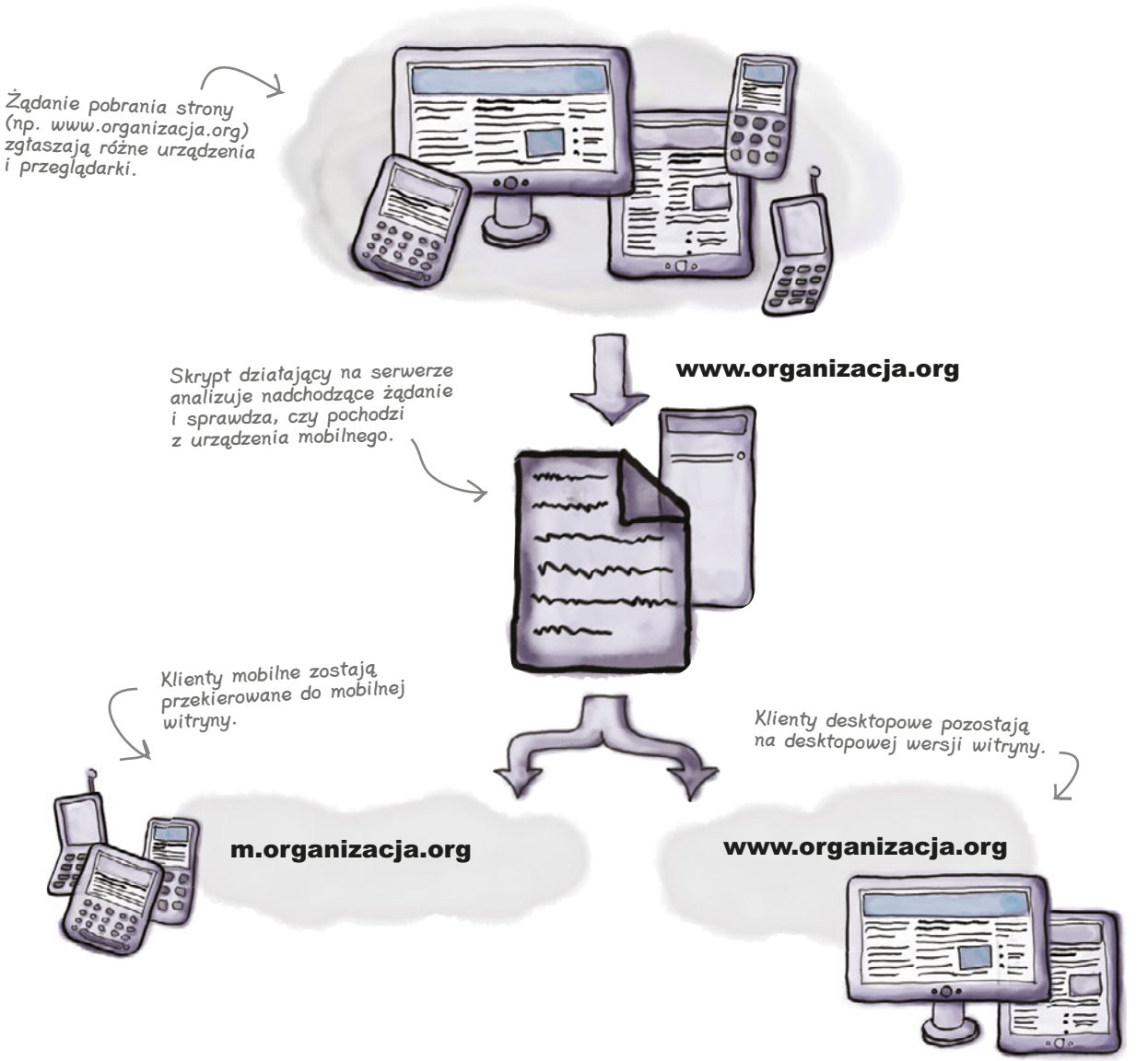
Zespół, który stworzył witrynę organizacji Zwierzętom na Pomoc, opracował również moduł pozwalający na zarządzanie agentami i dostawami rozrzuconymi po całym świecie. Powstał więc interfejs API, dzięki któremu można odczytywać i modyfikować dane dotyczące personelu i dostarczanej pomocy.

Dane wymieniane z API są odpowiednio ustrukturyzowane, dzięki czemu możemy z nich skorzystać podczas tworzenia mobilnej wersji witryny.

W przeciwieństwie do CMS-a organizacji, udostępniane API nie łączy logiki i prezentacji. Dzięki temu planowana optymalizacja przebiegnie znacznie prościej.

Wysyłamy mobilnych użytkowników na zoptymalizowaną witrynę

Czasem przygotowanie oddzielnej witryny dla urządzeń mobilnych i przeglądarek desktopowych jest koniecznością. W takich sytuacjach dobrym rozwiązaniem jest automatyczne przekierowanie użytkowników urządzeń mobilnych ze standardowego adresu, np. *www.organizacja.org*, na adres mobilnej wersji witryny.



Jak wywęszyć mobilnych użytkowników?

Aby przekierowanie mobilnych urządzeń do witryny zoptymalizowanej pod ich kątem zaczęło działać, serwer WWW musi wiedzieć, czy przychodzące żądanie pochodzi z urządzenia mobilnego, czy nie.

W celu sprawdzenia, czy użytkownik korzysta z urządzenia mobilnego, czy nie, zajrzymy do nagłówka HTTP przesłanego przez przeglądarkę i sprawdzimy identyfikator `User-Agent`.

Stosowane są też inne techniki, ale sprawdzanie pola `User-Agent` jest jednym z częściej wybieranych rozwiązań.

Ta technika jest określana terminem „user-agent sniffing".

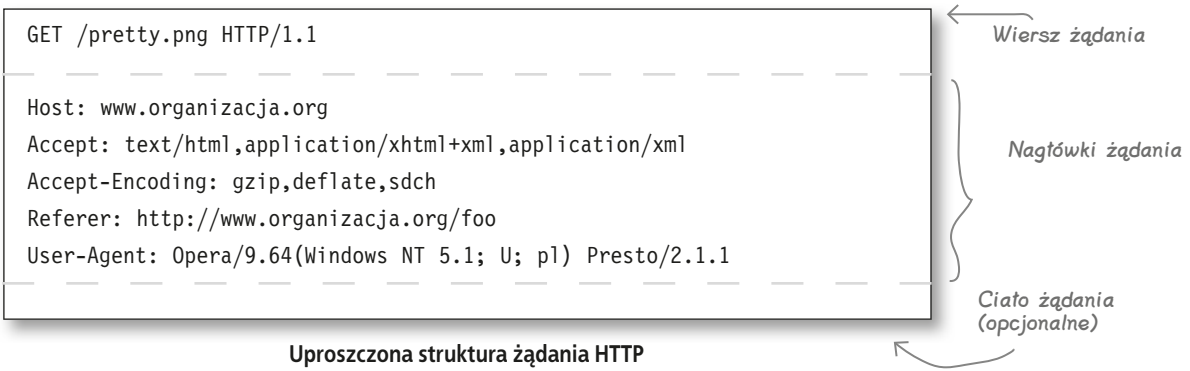
Łańcuch user-agent jest fragmentem tekstu, który pełni funkcję identyfikatora aplikacji klienckiej (w naszym przypadku jest to przeglądarka). Każda przeglądarka ma własny łańcuch user-agent.

Nazwa „user-agent" jest często skracana do „UA".

Rozpoznajemy agenta użytkownika

Przeglądarki internetowe, w tym także przeglądarki mobilne, przesyłają wraz z **żądaniem HTTP** strony lub innego zasobu **nagłówek `User-Agent`**.

Łańcuchy user-agent są od dawana używane (lepiej lub gorzej) przez webmasterów do zidentyfikowania przeglądarek (poprawnego lub nie). W latach 90. zeszłego wieku technika *user-agent sniffing* była zmorą użytkowników stykających się bez przerwy z wszechobecnymi komunikatami typu „Tę stronę najlepiej oglądać w Internet Explorerze” (albo Mozilli, Netscapie czy jakiejkolwiek innej przeglądarce, która akurat w danym czasie była preferowana).



Uproszczona struktura żądania HTTP

Archeologia stosowana — user-agent

Łańcuch user-agent wygląda często jak dziwna i zastanawiająca mieszanina konwencji, chaosu i różnego typu obejść. Nigdy nie udało się narzucić jednolitej budowy tego łańcucha, więc można się spotkać na przykład z czymś takim:

```
Mozilla/5.0 (Windows NT 6.0) AppleWebKit/535.1 (KHTML, like Gecko)
Chrome/14.0.792.0 Safari/535.1
```

Przeglądarka Chrome 14 działająca w systemie Windows Vista — przecież to oczywiste!

Dlaczego przeglądarka Chrome działająca na Windowsie wspomina o Safari? Co oznacza „KHTML” i w jakim sensie jest podobne (ang. *like*) do „Gecko”? To coś jak wyrostek robaczkowy u ludzi albo szczątkowe kończyny u wielorybów, czyli typowe cechy atawistyczne.

Informacja o kompatybilności z Mozillą (w tym przypadku chodzi o fragment Mozilla/5.0) jest obecna w zasadzie we wszystkich łańcuchach user-agent, więc tym samym niewiele wnosi. Wzmianka o Mozilli, KHTML, Gecko czy WebKit oznacza tylko, że dany agent użytkownika (czyli przeglądarka) ma silnik porównywalny z wymienionymi (lub od nich lepszy). W chwili pisania tej książki wszystkie przeglądarki bazujące na WebKit, które działają na urządzeniach mobilnych (z wyjątkiem tej z Androida), wymieniają w łańcuchu UA nazwę „Safari” (WebKit został stworzony przez firmę Apple na podstawie silnika KHTML).

Zbliżenie na łańcuch user-agent

Rzućmy okiem na fragmenty wybranych łańcuchów user-agent.

Przeglądarka Chrome 14 działająca w systemie Windows Vista (już to widzieliśmy na poprzedniej stronie)

```
Mozilla/5.0 (Windows NT 6.0) AppleWebKit/535.1 (KHTML,
like Gecko) Chrome/14.0.792.0 Safari/535.1
```

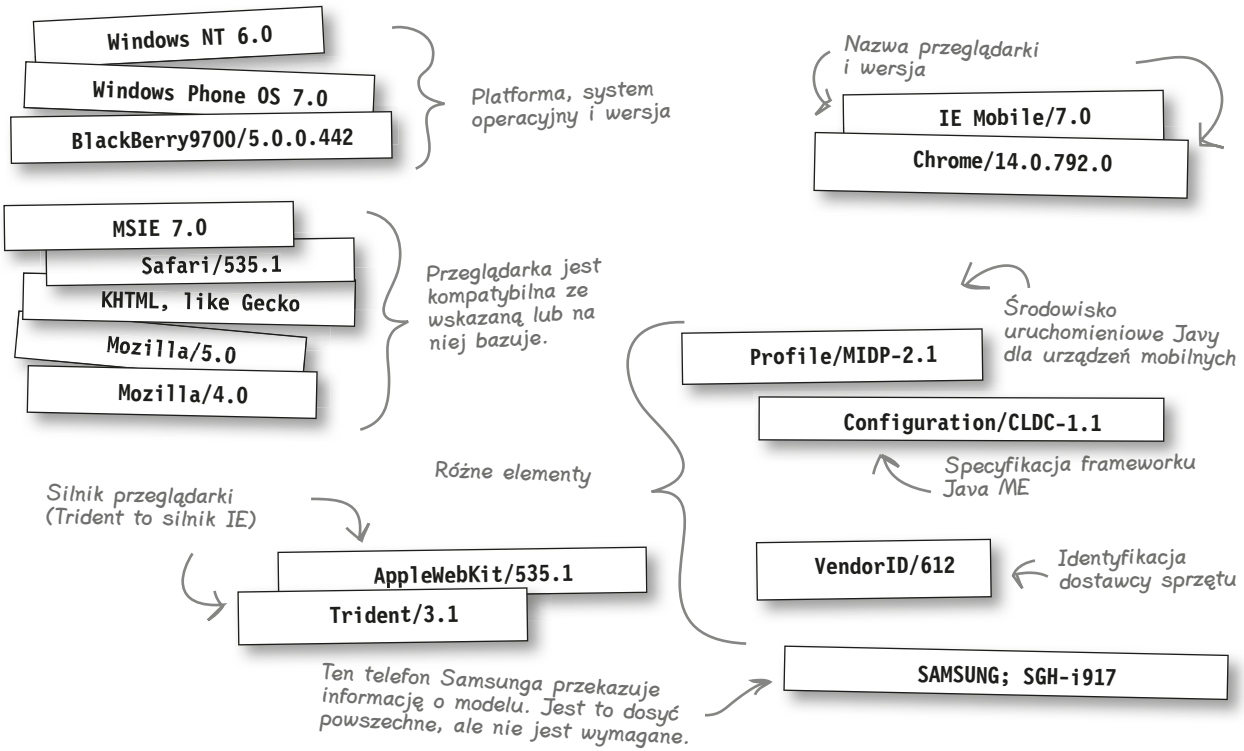
```
Mozilla/4.0 (compatible; MSIE 7.0; Windows Phone OS 7.0; Trident/3.1;
IEMobile/7.0; SAMSUNG; SGH-i917)
```

Telefon Samsunga z Windows Phone 7

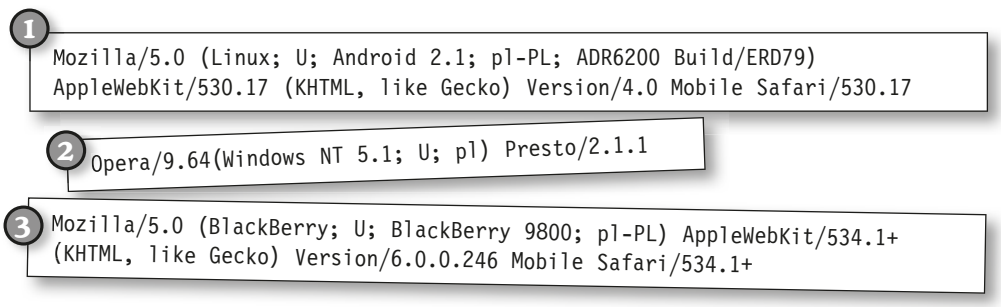
BlackBerry 9700 (Bold)

```
BlackBerry9700/5.0.0.442 Profile/MIDP-2.1 Configuration/CLDC-1.1
VendorID/612
```

Przeanalizujmy kilka fragmentów



JAKIE JEST MOJE PRZEZNACZENIE?



Byłeś już świadkiem analizy kilku łańcuchów user-agent. Teraz spróbuj sam.
Poniższe fragmenty wydzielone z przedstawionych łańcuchów UA połącz z ich przeznaczeniem.

Presto/2.1.1	Platforma, system operacyjny i wersja
Apple WebKit/534.1+	Historyczna informacja o kompatybilności
Windows NT 5.1	Nazwa przeglądarki i wersja
Mozilla/5.0	Silnik przeglądarki
Version/6.0.0.246	Historyczna informacja o kompatybilności
KHTML, like Gecko	Silnik przeglądarki
Opera/9.64	Wersja przeglądarki

→ Odpowiedź znajdziesz na stronie 102.

Łańcuch user-agent — dzieło szatana?

Czy jeśli używamy techniki user-agent sniffing do sprawdzenia, czy żądanie pochodzi z urządzenia mobilnego, nie powtarzamy tego samego błędu, który w przeszłości spowodował tyle zamieszania?

No cóż, w pewnym sensie tak.

Powiedz na głos „user-agent sniffing” w pokoju pełnym programistów, a z całą pewnością niejeden spojrzy na Ciebie spode łba, pokręci z dezaprobatą głową i wyda z siebie stłumione ostrzegawcze dźwięki.

Technika user-agent sniffing sprowadziła wielu programistów na manowce. Pokazaliśmy już, jak skomplikowane mogą być łańcuchy user-agent. Poza tym istnieje coś takiego jak *user-agent* ***spoofing***, czyli podszywanie się pod inną przeglądarkę, do którego dochodzi, gdy użytkownik skonfiguruje (świadomie lub nie) swoją przeglądarkę w taki sposób, że przesyła inny nagłówek UA. Trzeba też wziąć pod uwagę to, że funkcjonują setki możliwych łańcuchów UA, a ta liczba stale rośnie.

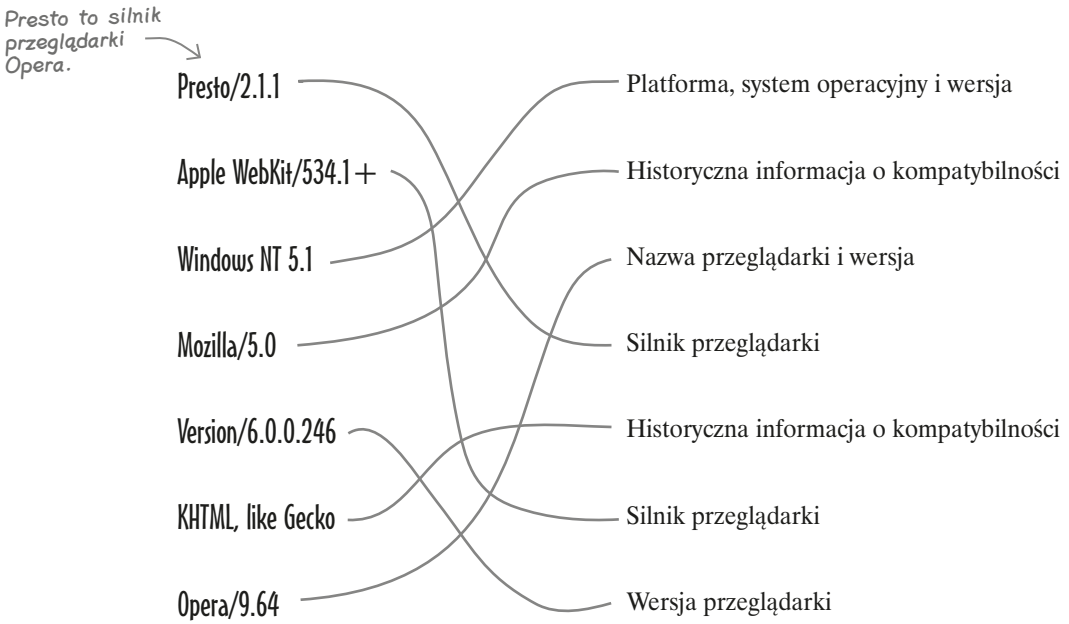
Wykrywanie mobilnych przeglądarek za pomocą tej techniki może wyglądać na nieeleganckie i niewłaściwe rozwiązanie, ale czasami jest to najlepsze, co możemy zrobić.

WYTĘŻ UMYSŁ

Pomyśl, z jakich powodów technika user-agent sniffing może być uznana za złą.

JAKIE JEST MOJE PRZEZNACZENIE?

ROZWIĄZANIE



Co to za przeglądarki?

Jesteś ciekawy, z jakich przeglądarek pochodzą łańcuchy user-agent z tego ćwiczenia (ze strony 100)? Zdemaskowaliśmy je za Ciebie:

1. **Smartfon Droid Eris (czyli HTC Desire) z systemem Android 2.1.**
2. **Przeglądarka Opera działająca na Windowsie XP.**
3. **BlackBerry 9800 (Torch) z systemem BlackBerry OS 6.**

Porównaj łańcuch UA z BlackBerry Torch (OS 6) ze strony 100 z łańcuchem z BlackBerry Bold (OS 5) ze strony 99. Różnią się, prawda? W BlackBerry OS 6 przeglądarka używa silnika WebKit i łańcuch UE wygląda podobnie jak w innych przeglądarkach WebKit.

Obejrzyj to!

Nie wszystkie łańcuchy UA są tworzone zgodnie ze spójnym szablonem.

Jedne łańcuchy są bardziej spójne od innych. Nie we wszystkich znajdują się te same elementy, a w niektórych można znaleźć bardzo nietypowe fragmenty. Co prawda w większości przypadków jest stosowany jeden z kilku podstawowych szablonów, ale nie jest to regułą. Nie przejmuj się! Na szczęście są dostępne dobre narzędzia, na które można (przeważnie) liczyć.

Nie istnieją głupie pytania

P: A co z innymi elementami łańcucha `user-agent` ze strony 99, których nie opisaliście? Co na przykład oznacza `pl-PL`? A co znaczy U?

O: Fragment `pl-PL` to informacja o języku stosowanym w przeglądarce i elementach jej interfejsu użytkownika. Z kolei `U` to informacja o poziomie bezpieczeństwa przeglądarki — w tym przypadku wysokim (`I` oznacza niski poziom bezpieczeństwa, a `N` — brak zabezpieczeń). Jeśli chcesz poznać znaczenie pozostałych elementów łańcucha UA, zajrzyj na stronę *www.useragentstring.com*.

P: Czy naprawdę wszystkie wersje lub nawet rewizje przeglądarek, platformy i urządzenia są opisywane innym łańcuchem `user-agent`?

O: Przeważnie tak właśnie jest.

P: Czy nagłówki `User-Agent` przesyłają tylko przeglądarki?

O: Przesyła je wiele innych aplikacji wykonujących żądania do serwerów internetowych. Zaliczają się do nich na przykład wyszukiwarki, roboty indeksujące czy różne boty. Czasem serwery zapisują łańcuchy UA w przypadku nietypowych żądań, jeśli stronę chcą pobrać na przykład przeglądarka wbudowana w desktopową aplikację do sporządzania raportów, narzędzie do zarządzania zakładkami czy usługa monitorująca działanie stron internetowych.

P: Dlaczego niektórzy użytkownicy fałszują łańcuch user-agent przesyłany przez przeglądarkę?

O: Pytasz, dlaczego niektórzy umyślnie „proszą" przeglądarkę o przekazywanie informacji niezgodnych z prawdą? Można wymienić kilka powodów, a najbardziej typowe to zachowanie prywatności oraz wymuszenie pożądanego zachowania przeglądarki. Niektórzy użytkownicy nie chcą się dzielić informacją o tym, jakiej używają przeglądarki. Inni z kolei chcą zobaczyć stronę wyświetlaną w określony sposób. Dobrym przykładem jest „tryb desktopowy" dostępny w wielu przeglądarkach mobilnych. Po jego włączeniu (czasem nieświadomym) przeglądarka przesyła inny niż standardowy nagłówek `User-Agent`, który przypomina bardziej nagłówek przesyłany przez przeglądarki desktopowe. Czasem jest to realizowane lepiej, jak choćby w przypadku Windows Phone 7, a czasem gorzej. Spójrz na poniższy łańcuch UA przesyłany w trybie desktopowym przez przeglądarkę Skyfire działającą w jednej z wersji Androida:

```
Mozilla/5.0 (Macintosh; U;
Intel Mac OS X 10_5_7; pl-pl)
AppleWebKit/530.17 (KHTML, like
Gecko) Version/4.0 Safari/530.17
```

Przypomina to łańcuch z przeglądarki Safari działającej w systemie OS X. Tak właściwie jest to dokładnie taki sam łańcuch UA, jaki przesyła desktopowa przeglądarka Safari.

P: Ale czy w sytuacji, gdy ktoś w mobilnej przeglądarce celowo podmienił łańcuch UA, nie powinniśmy uszanować jego wyboru i przesłać mu desktopowej wersji strony?

O: Co prawda jest to pytanie natury filozoficznej, ale możemy odpowiedzieć „tak", a ewentualnym problemom zapobiec za pomocą technik RWD (które pozwalają na dopasowanie elementów strony do warunków, niezależnie od tego, czy przeglądarka jest mobilna, czy desktopowa, byleby obsługiwała zapytania o media). Co prawda nie ma pewności, czy użytkownik w pełni świadomie podmienił łańcuch UE, ale lepiej się trzymać zasady „nasz klient, nasz pan". Pamiętaj jednak, że w podejściu do tej sprawy twórcy stron się różnią, więc możesz się spotkać z krytyką.

P: Chwila, a co z UAProf? W ogóle o tym nie wspomnieliście.

O: Wiele urządzeń mobilnych w żądaniach przesyła nagłówek `X-Wap-Profile` zawierający opis UAProf (ang. *User-Agent Profile*). Specyfikacja UAProf daje wytwórcom mobilnych urządzeń możliwość dostarczenia szczegółowych informacji o tych urządzeniach. W nagłówku zwykle jest przesyłany odsyłacz do pliku UAProf.

Może się wydawać, że to jest doskonałe rozwiązanie, ale niestety nie wszystkie urządzenia i przeglądarki przesyłają nagłówki UAProf. Poza tym odsyłacze zawarte w nagłówku wielokrotnie nie są prawidłowe. Nie ma też przyjętego standardu opisu urządzeń, więc pliki UAProf dla poszczególnych urządzeń zawierają różną ilość informacji.

P: No dobrze, podsumujmy — są tysiące łańcuchów UA, a pliki UAProf nie są dostępne dla wszystkich urządzeń. Czy nadal twierdzicie, że na podstawie UA można wykrywać urządzenia mobilne?

O: W rozdziale 5. poznasz organizacje i projekty, których jedynym celem jest śledzenie powstających łańcuchów UA i budowanie baz danych zawierających metadane opisujące różne urządzenia oraz inne struktury, takie jak choćby UAProf. Do tego czasu zastosujemy najprostsze podejście, czyli skrypt działający po stronie serwera, który szuka w łańcuchach UA fragmentów wskazujących na mobilne przeglądarki.

Mówiąc wprost — większość dużych witryn ma swoją wersję mobilną

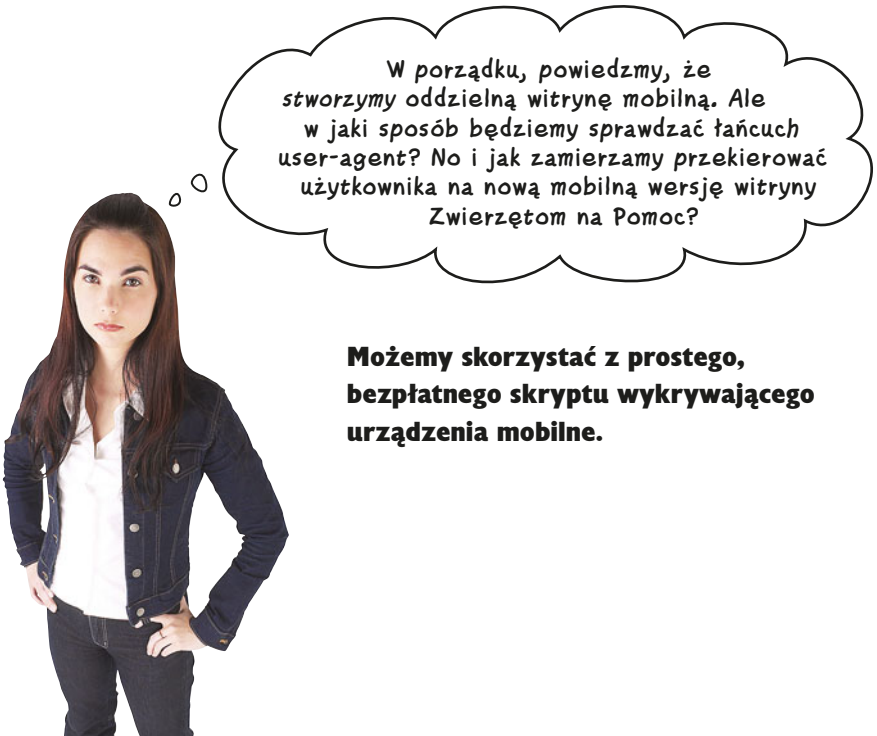
Duża część największych na świecie witryn internetowych ma oddzielne wersje mobilne, a wiele z nich do przekierowania do tej wersji korzysta z jakiejś formy techniki user-agent sniffing. To jak to w końcu jest — ta technika jest zła czy nie? Można przypuszczać, że ma *jakieś* zalety.

Niektóre rozwiązania lepiej się czują na serwerze...

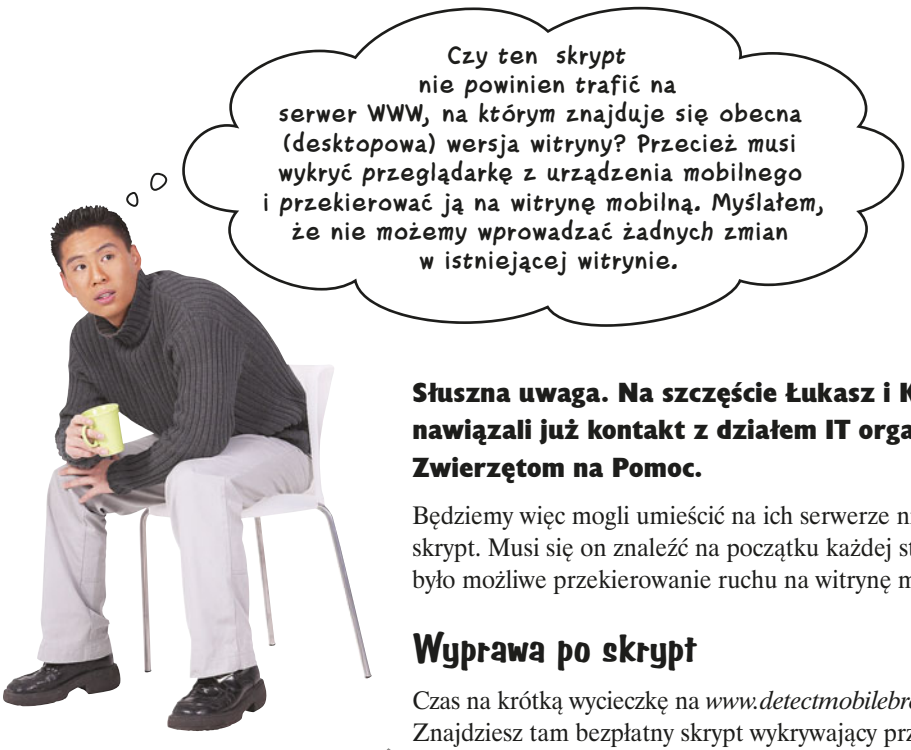
Przekierowanie ruchu sieciowego do mobilnej witryny jest czymś, co doskonale pasuje do środowiska serwera. Jeśli żądania zgłaszane przez urządzenia mobilne mają kierować do innej witryny (czyli tak jak w przypadku naszego projektu), **decyzję o przekierowaniu najlepiej podjąć przed przesłaniem odpowiedzi do klienta**.

...a nagłówek User-Agent jest tego najlepszym przykładem

Chociaż użycie nagłówka `User-Agent` nie jest pozbawione wad, jest z pewnością **najprostszą i najbardziej niezawodną wskazówką dotyczącą przeglądarek zgłaszających żądania**. Jeszcze kilka innych nagłówków żądania HTTP może pomóc w stwierdzeniu, czy mamy do czynienia z przeglądarką mobilną, ale żaden z nich nie jest tak popularny jak `User-Agent`. Jest to po prostu najlepszy wybór z możliwych. Opiszemy sytuacje, w których ta metoda detekcji się nie sprawdza (trzeba być czasem adwokatem diabła), ale większość czasu poświęcimy na dopracowanie naszego projektu.



Kiedy jedynym rozwiązaniem jest przekierowanie...

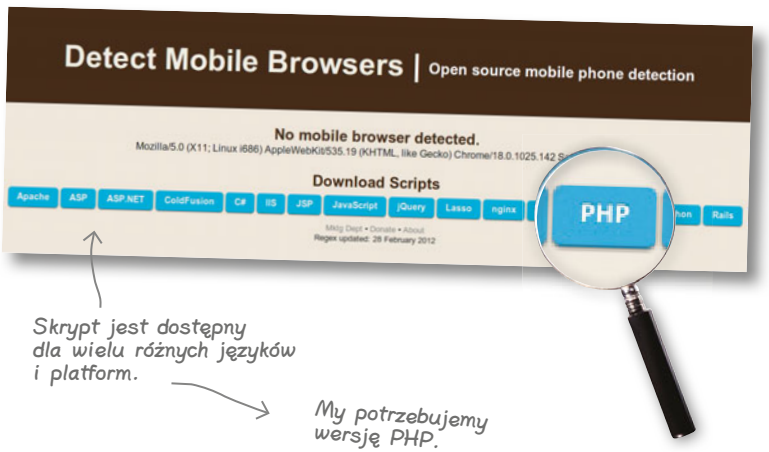


Słuszna uwaga. Na szczęście Łukasz i Kuba nawiązali już kontakt z działem IT organizacji Zwierzętom na Pomoc.

Będziemy więc mogli umieścić na ich serwerze niewielki skrypt. Musi się on znaleźć na początku każdej strony, tak by było możliwe przekierowanie ruchu na witrynę mobilną.

Wyprawa po skrypt

Czas na krótką wycieczkę na *www.detectmobilebrowsers.com*. Znajdziesz tam bezpłatny skrypt wykrywający przeglądarki mobilne. Pobierz najnowszą wersję tego skryptu — koniecznie w wersji dla PHP.



Rzuć okiem na skrypt

```
<?php
$useragent=$_SERVER['HTTP_USER_AGENT'];
if(preg_match('/android|avantgo|blackberry|blazer|compal|elaine|fennec|hiptop|iem
obile|ip(hone|od)|iris|kindle|lge |maemo|midp|mmp|opera m(ob|in)i|palm( os)?|phon
e|p(ixi|re)\/|plucker|pocket|psp|symbian|treo|up\.(browser|link)|vodafone|wap|wi
ndows (ce|phone)|xda|xiino/i',$useragent)|| preg_match('/1207|6310|6590|3gso|4thp
|50[1-6]i|770s|802s|a wa|abac|ac(er|oo|s\-)|ai(ko|rn)|al(av|ca|co)|amoi|an(ex|ny
|yw)|aptu|ar(ch|go)|as(te|us)|attw|au(di|\-m|r |s )|avan|be(ck|ll|nq)|bi(lb|rd)|
bl(ac|az)|br(e|v)w|bumb|bw\-(n|u)|c55\/|capi|ccwa|cdm\-|cell|chtm|cldc|cmd\-|co(
mp|nd)|craw|da(it|ll|ng)|dbte|dc\-s|devi|dica|dmob|do(c|p)o|ds(12|\-d)|el(49|ai)
|em(l2|ul)|er(ic|k0)|esl8|ez([4-7]0|os|wa|ze)|fetc|fly(\-|_)|g1 u|g560|gene|gf\-
5|g\-mo|go(\.w|od)|gr(ad|un)|haie|hcit|hd\-(m|p|t)|hei\-|hi(pt|ta)|hp( i|ip)|hs\-
c|ht(c(\-| |_|a|g|p|s|t)|tp)|hu(aw|tc)|i\-(20|go|ma)|i230|iac( |\-|\/)|ibro|idea|
ig01|ikom|im1k|inno|ipaq|iris|ja(t|v)a|jbro|jemu|jigs|kddi|keji|kgt( |\/)|klon|kpt
|kwc\-|kyo(c|k)|le(no|xi)|lg( g|\/(k|l|u)|50|54|e\-|e\/|\-[a-w])|libw|lynx|m1\-
w|m3ga|m50\/|ma(te|ui|xo)|mc(01|21|ca)|m\-cr|me(di|rc|ri)|mi(o8|oa|ts)|mmef|mo(
01|02|bi|de|do|t(\-| |o|v)|zz)|mt(50|p1|v )|mwbp|mywa|n10[0-2]|n20[2-3]|n30(0|2
)|n50(0|2|5)|n7(0(0|1)|10)|ne((c|m)\-|on|tf|wf|wg|wt)|nok(6|i)|nzph|o2im|op(ti
|wv)|oran|owg1|p800|pan(a|d|t)|pdxg|pg(13|\-([1-8]|c))|phil|pire|pl(ay|uc)|pn\-
2|po(ck|rt|se)|prox|psio|pt\-g|qa\-a|qc(07|12|21|32|60|\-[2-7]|i\-)|qtek|r380|r60
0|raks|rim9|ro(ve|zo)|s55\/|sa(ge|ma|mm|ms|ny|va)|sc(01|h\-|oo|p\-)|sdk\/|se(c(\-
|0|1)|47|mc|nd|ri)|sgh\-|shar|sie(\-|m)|sk\-0|sl(45|id)|sm(al|ar|b3|it|t5)|so(f
t|ny)|sp(01|h\-|v\-|v )|sy(01|mb)|t2(18|50)|t6(00|10|18)|ta(gt|lk)|tcl\-|tdg\-
|tel(i|m)|tim\-|t\-mo|to(pl|sh)|ts(70|m\-|m3|m5)|tx\-9|up(\.b|g1|si)|utst|v40
0|v750|veri|vi(rg|te)|vk(40|5[0-3]|\-v)|vm40|voda|vulc|vx(52|53|60|61|70|80|8
1|83|85|98)|w3c(\-| )|webc|whit|wi(g |nc|nw)|wmlb|wonu|x700|xda(\-|2|g)|yas\-
|your|zeto|zte\-/i',substr($useragent,0,4)))
header('Location: http://detectmobilebrowser.com/mobile');
?>
```

(1) – `$_SERVER['HTTP_USER_AGENT'];`
(2) – `if(preg_match(`
(3) – `header('Location: http://detectmobilebrowser.com/mobile');`

detectmobilebrowser.php.txt

To jest właśnie ten skrypt! Najnowsza wersja, którą pobrałeś, może nie wyglądać dokładnie tak samo, ale ogólna budowa z pewnością nie uległa zmianie.

Spokojnie

Przecież ten skrypt to jakaś dzika bestia! Spokojnie, wbrew pozorom uda nam się go poskromić.

Przyznaj szczerze, że kiedy pierwszy raz otworzyłeś plik ze skryptem, trochę się przeraziłeś. Teraz nie musisz się już bać. Opiszemy pokrótce jego działanie, ale tak naprawdę jedyne, co trzeba zrobić, to wprowadzić kilka drobnych zmian. Oczywiście pokażemy Ci jakich.

Zrób to sam!

Zmień nazwę tego pliku na *redirect.php* i zapisz w katalogu rozdzial3.

Jak działa skrypt?

Skrypt sprawdza, czy łańcuch user-agent pochodzi z mobilnej przeglądarki i, jeżeli tak, przekierowuje na podany adres. Odbywa się to w trzech krokach:

1. **Najpierw jest pobierany łańcuch user-agent przesłany przez przeglądarkę.**
   W PHP realizuje się to poprzez globalną zmienną `$_SERVER['HTTP_USER_AGENT']`.

2. **Na łańcuchu zostają zastosowane misternie skonstruowane wyrażenia regularne.**
   W skrypcie znajdują się dwa wyrażenia regularne. Oba działają na łańcuchu user-agent i sprawdzają, czy zawiera on wartości wskazujące na mobilną przeglądarkę. Najprawdopodobniej wyłowiłeś niektóre oczywiste fragmenty, takie jak „hiptop” czy „symbian”, ale poza nimi znajduje się tu wiele ukrytych wzorców. Te wyrażenia opracowali ludzie, którzy naprawdę mocno siedzą w świecie urządzeń mobilnych.

   Wyrażenia regularne (ang. regular expressions, znane także pod nazwami regex lub regexp) umożliwiają sprawdzanie występowania wzorców w łańcuchach tekstowych.

3. **Jeśli łańcuch UA odpowiada wyrażeniu regularnemu, następuje przekierowanie przeglądarki na inny adres.**
   Jeśli w łańcuchu user-agent został znaleziony wzorzec odpowiadający jakiemuś urządzeniu mobilnemu, skrypt ustawia nagłówek `Location` z adresem, na który ma nastąpić przekierowanie.

W skrypcie domyślnym adresem przekierowania jest www.detectmobilebrowser.com/mobile. Musimy to zmienić.

Sprawdź, czy jesteś gotowy

Zanim przejdziesz dalej, sprawdź, czy masz wszystko, co będzie potrzebne. W kilku kolejnych sekcjach, a także w wielu innych kolejnych rozdziałach będziesz potrzebował serwera WWW, który obsługuje PHP. Możesz korzystać ze swojego komputera lub z zewnętrznego serwera — to bez znaczenia. Jeśli jeszcze się tym nie zająłeś, przejdź do dodatku B, w którym znajdziesz niezbędne informacje.

Przygotowujemy makietę wersji mobilnej

Kuba: Cześć pracy! Jak tam postępy nad mobilną wersją witryny organizacji Zwierzętom na Pomoc?

Łukasz: Pomyślałem, że napisanie kodu współpracującego z udostępnianym interfejsem API może na razie poczekać, więc zająłem się projektem witryny. Przygotowałem kilka makiet, od których możemy zacząć. Nie wszystko naraz, prawda?

Łukasz: Prawda. Mam nadzieję, że pamiętałeś o podejściu RWD?

Kuba: No cóż, i tak, i nie. Użyłem proporcjonalnych szerokości, ale przypomniałem sobie, że to musi działać też na starszych telefonach, nieobsługujących zapytań o media ani innych technik, dzięki którym projekty mogą być zgodne z RWD. Wziąłem na warsztat szablon głównej strony organizacji, uprościłem go i zmodyfikowałem CSS, tak by nie sprawiał problemów na urządzeniach mobilnych. Skorzystałem z przykładowych danych, więc na razie nie musimy się przejmować funkcjonalnością i interfejsem API.

Łukasz: A co z wykrywaniem urządzeń mobilnych i skryptem przekierowującym? Jak go przetestujemy?

Kuba: Słyszałem, że musimy go umieścić na serwerze organizacji, by ludzie z działu IT podpięli go do każdej strony witryny. Na razie przetestujemy to lokalnie, by sprawdzić, jak to przekierowanie działa.

Kilka zmian w skrypcie przekierowującym

W skrypcie musimy wprowadzić kilka zmian, aby zobaczyć go w akcji. Dodamy **nowy wiersz**, w którym zdefiniujemy lokalizację mobilnej wersji witryny. Przyda nam się zmienna `$_SERVER['HTTP_HOST']` przechowująca w PHP adres hosta, na którym działa skrypt.

Nazwa hosta (np. www.organizacja.org, „localhost", „10.0.0.2" lub cokolwiek innego, jeśli pracujesz na swoim komputerze).

Musimy też **zmodyfikować wiersz, w którym jest przesyłany** nagłówek `Location` odpowiedzi. Urządzenia mobilne chcemy przecież przekierować do naszej makiety, a nie na stronę *www.detectmobilebrowser.com*.

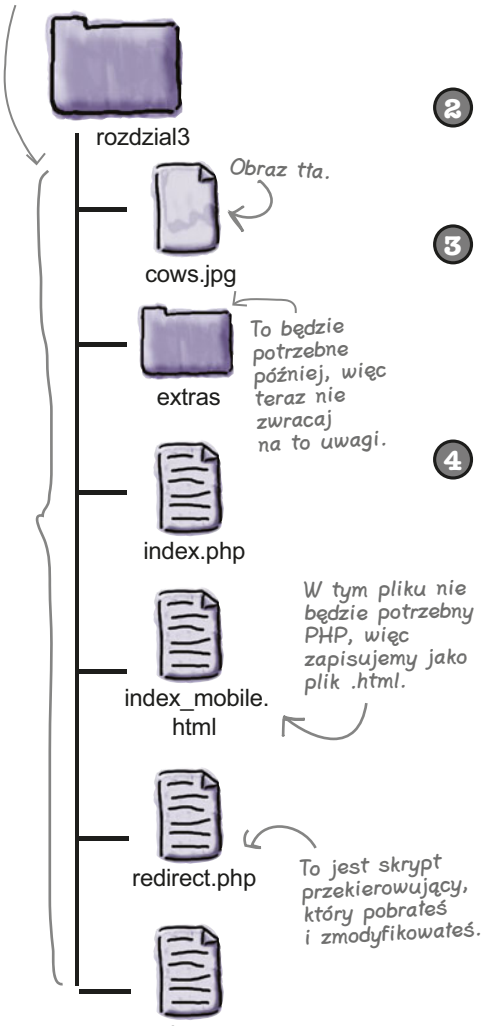
Nagłówki znajdują się zarówno w żądaniach (jak choćby User-Agent), jak i w odpowiedziach HTTP (odsyłanych przez serwer).

```
<?php
$useragent = $_SERVER['HTTP_USER_AGENT'];
$mobile_location = 'http://' . $_SERVER['HTTP_HOST'] . '/index_mobile.html';
if(preg_match(...))
   header('Location: ' . $mobile_location);
?>
```

redirect.php

Jazda próbna

Skopiuj całą zawartość katalogu rozdzial3 do głównego katalogu dokumentów serwera WWW.



1. **Zawartość katalogu rozdzial3 umieść w głównym katalogu dokumentów serwera WWW.**
Jeżeli nie chcesz umieszczać plików w tym katalogu, nie musisz. Jednak w takim przypadku koniecznie zaktualizuj ścieżkę w deklaracji zmiennej `$mobile_location`.

2. **W skrypcie przekierowującym nanieś zmiany opisane na stronie 108.**
Ponownie otwórz skrypt przekierowujący w edytorze (jeżeli zdążyłeś już go zamknąć). Wprowadź opisane zmiany i **je zapisz**.

3. **Uzupełnij plik index.php.**
Na samym początku pliku *index.php* dodaj poniższy kod:

```
<?php require_once('redirect.php'); ?>
```

Nie zapomnij zapisać zmian.

4. **Otwórz makietę w przeglądarce (w emulatorze lub urządzeniu).**

Makieta wyświetlona w iPhonie 4 — prosta, ale działa.

To tylko makiety.

Spokojnie Makiety stron mobilnych, z których będziemy korzystać w tym rozdziale, są tylko, no cóż, makietami. Wpisane są w nich na stałe pewne dane, tak by było wiadomo, jak strona będzie wyglądać po podpięciu do prawdziwego źródła danych. Wpisane dane są wyssane z palca, a wiele odsyłaczy nie działa. Nie przejmuj się tym!

Specjalna dostawa... spraw komplikujących życie

Cześć, Łukaszu. Zrobiłam mały wywiad wśród personelu i dowiedziałam się, że na wielu telefonach mają przeglądarkę Opera Mini. Przygotowałam też dla ciebie paczkę z kilkoma telefonami, których najczęściej używamy, żebyś mógł przeprowadzić na nich testy.

Że co? Opera Mini?

Co prawda to odpicowane przeglądarki, takie jak WebKit w BlackBerry czy Safari w iOS, zbierają laury, ale nie można zapominać o Operze Mini. Dwie siostry — **Opera Mini i Opera Mobile — są najpopularniejszymi mobilnymi przeglądarkami na świecie.**

W połowie 2011 roku Opera miała 110 milionów użytkowników. Dla przykładu: z Opery pochodzi prawie połowa ruchu sieciowego z urządzeń mobilnych w Indiach i blisko 90% ruchu w Nigerii.

Opera Mini jest popularna zwłaszcza w tych miejscach, gdzie łącza są słabe, a transfer drogi i limitowany.

Opera Mini nie jest zwykłą przeglądarką, jest **transkoderem**. Kiedy użytkownik chce otworzyć jakąś stronę, żądanie jest prowadzone przez serwery Opery, na których wszystkie dokumenty i zasoby są kompresowane i optymalizowane pod kątem urządzeń mobilnych. Dzięki temu zmniejsza się ilość danych dostarczanych do urządzenia, a tym samym zwiększa prędkość (i często również zmniejsza koszty).

Opera Mini działa inaczej niż typowe przeglądarki instalowane w smartfonach, ponieważ założeniem przyświecającym jej twórcom było dostosowanie do możliwości wielu różnych urządzeń (również tych mniej wydajnych).

Popularność Opery Mini nie jest ograniczona do krajów rozwijających się. Jest obecna również w wielu popularnych telefonach używanych w Europie i USA.

Świetnie. Telefony z pewnością przydadzą nam się podczas testów. Ale widzę, że część z nich lata świetności ma już dawno za sobą, a części z nich w ogóle nie kojarzę. Ciekawe, jak nasza makieta będzie wyglądała na tych „złomkach"...

Zrób to sam!

Postaramy się wyświetlić makietę w Operze Mini. Opera udostępnia wygodny symulator (aplet Javy), w którym można testować strony. Znajdziesz go pod adresem:

http://www.opera.com/mobile/demo/

Na wskazanej stronie jest dostępny symulator najnowszej wersji Opery Mini (obecnie jest to 6.5), która już całkiem dobrze radzi sobie z wieloma, dawniej problematycznymi sprawami. Trzeba się jednak liczyć z tym, że na wielu telefonach funkcjonują starsze, a zarazem uboższe wersje tej przeglądarki. Z tego względu najlepiej będzie przetestować naszą makietę w symulatorze Opery Mini w wersji 4.2. Niestety symulator nie jest już dostępny na oficjalnej stronie producenta, więc trzeba się posiłkować innymi źródłami. Oto jedno z możliwych:

http://ariefew.com/simulator/opera-mini-42-simulator/

Zaostrz ołówek

Zobaczmy, jak przygotowana makieta wygląda w symulatorze Opery Mini 4.2. Otwórz stronę symulatora i przejdź na adres makiety: *ftp://ftp.helion.pl/online/inne/hfmw/r03/1*.

Zanotuj, co wygląda inaczej (i — niestety — źle). Czy umiesz znaleźć cztery różnice między tym, jak makieta wygląda w symulatorze Opery Mini, a tym, jak w symulatorze iPhone'a oraz Androida (patrz strona 109)?

1. .....
2. .....
3. .....
4. .....

Zaostrz ołówek

Rozwiązanie

W Operze Mini 6.5 makieta wygląda inaczej, prawda? Znaleźliśmy pięć istotnych różnic. Oto one:

1. *Do elementów <h1> i <h2> nie zostały zastosowane style, więc tekst jest za mały. Z tego względu obraz w tle wydaje się za duży, a cały nagłówek wygląda dziwnie.*

   Tak naprawdę w Operze Mini mamy bardzo ograniczone możliwości stosowania stylów.

2. *Cała strona jest za szeroka, więc wychodzi poza prawą krawędź ekranu.*

   To jest najgorsze.

3. *Formatowanie tekstu też nie wygląda najlepiej. Z założenia pochylony tekst wcale nie jest pochylony. To samo dotyczy zresztą pogrubienia.*

4. *Wierzchołki są proste, a powinny być zaokrąglone.*

5. *Właściwość overflow bloku komunikatu jest ustawiona na scroll, co jest niedopuszczalne w przypadku urządzeń mobilnych. Zawartość tego bloku została przycięta, a paski przewijania nie działają.*

   Punkt dla Ciebie, jeśli zauważyłeś to samo na iPhonie i pod Androidem!

Nie istnieją głupie pytania

P: Dlaczego muszę sprawdzać działanie tej witryny, korzystając z adresu *ftp://ftp.helion.pl/online/inne/hfmw/r03*? Nie mogę wyświetlić swojej wersji?

O: Opera Mini to *przeglądarka proxy*, co oznacza, że cały ruch sieciowy przechodzi przez serwer Opery. Zatem makieta naszego projektu musi być dostępna dla serwerów Opery.

Twój komputer, na którym pracujesz nad stroną, ma zapewne adres IP niewidoczny w internecie (czyli jest to wewnętrzny adres IP dostępny tylko w lokalnej sieci).

Jeśli masz możliwość umieszczenia tworzonej strony na ogólnodostępnym serwerze WWW, nie ma też problemu, byś testował ją za pomocą symulatora przeglądarki Opera Mini.

Nie wszystkie telefony to smartfony...

W wielu krajach, w których działa organizacja Zwierzętom na Pomoc, widok smartfonu to rzadkość. Na przykład w Indiach w 2011 roku smartfony stanowiły 6% łącznej sprzedaży telefonów. Niewiele, prawda?

A ile telefonów jest w Indiach?

W sierpniu 2011 roku w Indiach było zarejestrowanych **850 milionów** telefonów. To drugi wynik na świecie, tuż za Chinami.

Mimo że w Indiach smartfony to rzadkość, telefony komórkowe są łatwiej dostępne niż toalety!

To nie jest głupi żart. W jednym z raportów ONZ dostępność telefonów komórkowych została zestawiona z dostępnością infrastruktury sanitarnej.

A co, jeśli użytkownicy nie znajdują się w Indiach? Przecież na przykład w Polsce smartfony są bardzo popularne. Może nie powinnam się w ogóle przejmować starszymi telefonami?

Nie tylko w krajach rozwijających się liczba zwykłych telefonów komórkowych przekracza liczbę smartfonów.

W Stanach Zjednoczonych i wielu krajach europejskich dopiero niedawno sprzedaż smartfonów przekroczyła sprzedaż zwykłych telefonów. Szacuje się, że smartfony będą stanowić większość rynku telefonów dopiero pod koniec 2012 roku.

Przeglądając raporty dotyczące wzrostu popularności smartfonów, musisz pamiętać, że minie trochę czasu, zanim wszystkie stare telefony ustąpią miejsca nowym smartfonom.

Poza tym zwykłe telefony komórkowe nie są takie złe

Jasne, przyprawiają nas o ból głowy ze swoimi starymi, kapryśnymi przeglądarkami i często słabym łączem, ale pamiętaj o tym, że dzięki nim poprawił się komfort życia bardzo wielu ludzi na całym świecie.

Jeśli chcesz poznać prawdziwe znaczenie słowa „innowacja”, wybierz się w podróż po krajach rozwijających się i zobacz, jak tamci ludzie korzystają z telefonów komórkowych i co dzięki nim potrafią załatwić.

Dla wielu ludzi telefon nie jest tylko fajną, kieszonkową wersją nowoczesnego komputera. Liczy się przede wszystkim możliwość nawiązywania kontaktu ze światem.

Prostota przede wszystkim — poznaj XHTML-MP

Zajrzyj jeszcze raz do pliku *index_mobile.html* i sprawdź, jak jest ustawiony typ dokumentu (DOCTYPE). Powinieneś zobaczyć to:

```
<!DOCTYPE html>
```

To jest typ dokumentu w HTML5. To doskonały wybór w przypadku współczesnych bogatych stron i aplikacji internetowych. Jeśli jednak chcemy przygotować stronę obsługiwaną dobrze przez urządzenia, jakimi dysponują agenci organizacji Zwierzętom na Pomoc, niekoniecznie jest to najlepsze rozwiązanie. Najwyższy czas, byś poznał nowego przyjaciela — **XHTML Mobile Profile** (w skrócie **XHTML-MP**).

XHTML-MP to wariant XHTML-a opracowany z myślą o telefonach komórkowych i ich niedoskonałych przeglądarkach. Ma już kilka dobrych lat i jest coraz silniej wypierany przez młodszego i seksowniejszego kuzyna — HTML5 — jednak w takich projektach jak nasz sprawdza się idealnie.

XHTML Mobile Profile jest podzbiorem desktopowego standardu (X)HTML, który znasz i kochasz.
To uproszczona wersja HTML-a dostosowana do możliwości wielu różnych przeglądarek mobilnych, które — w większości — obsługują ten standard.

DOCTYPE gotowy do użycia

Przekształcimy makietę strony na dokument XHTML-MP. W tym celu **otwórz plik *index_mobile.html*** i zamień definicję typu dokumentu dla HTML5 (pierwszy wiersz pliku) na takie dwa wiersze:

```
<?xml version="1.0" encoding="UTF-8" ?>
< !DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.2//EN"
"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">
```

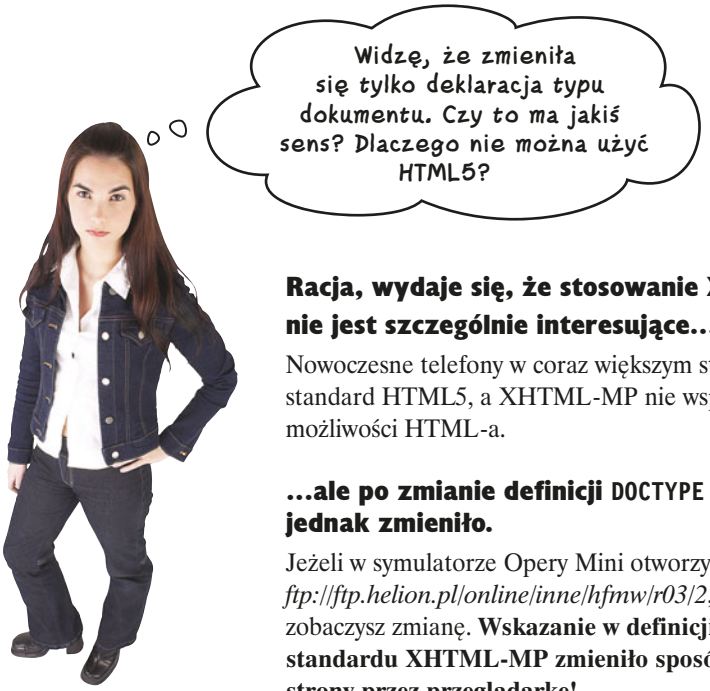
XHTML-MP bazuje na XML-u, więc ta deklaracja jest niezbędna.

To jest definicja typu dokumentu dla najnowszej wersji XHTML-MP, czyli wersji 1.2.

index_mobile.html

Tak, wiemy, że napisaliśmy „dwa wiersze", ale w dokumencie zrobią się z tego najprawdopodobniej trzy.

Dlaczego chcemy użyć tak starego rozwiązania?



Racja, wydaje się, że stosowanie XHTML-MP nie jest szczególnie interesujące...

Nowoczesne telefony w coraz większym stopniu obsługują standard HTML5, a XHTML-MP nie wspiera niektórych możliwości HTML-a.

...ale po zmianie definicji `DOCTYPE` coś się jednak zmieniło.

Jeżeli w symulatorze Opery Mini otworzysz stronę *ftp://ftp.helion.pl/online/inne/hfmw/r03/2*, z pewnością zobaczysz zmianę. **Wskazanie w definicji `DOCTYPE` standardu XHTML-MP zmieniło sposób wyświetlania strony przez przeglądarkę!**

Zawartość strony już nie wychodzi poza ekran — przyciski i inne elementy wypełniają całą szerokość strony.

Przeglądarka Opera Mini traktowała definicję `DOCTYPE` dla HTML5 jako wskazówkę, by zachowywać się bardziej jak przeglądarka desktopowa. Z tego względu układ strony oparty o procentowe szerokości (zgodny z RWD) może nie być prawidłowo wyświetlany.

Jeszcze nie skończyliśmy — sama zmiana definicji `DOCTYPE` nie wystarczy.

Zmieniliśmy definicję `DOCTYPE` na XHTML-MP, ale nie oznacza to, że dokument jest zgodny ze specyfikacją XHTML-MP. Musimy wprowadzić kilka zmian, dzięki którym strona stanie się prawdziwym dokumentem XHTML-MP.

Spójrz tylko na to! Strona mieści się na ekranie.

XHTML-MP pomaga unikać problemów

Być może specyfikacja XHTML-MP nie jest zachwycająca, ale wskazanie jej w definicji DOCTYPE ma kilka zalet:

1. **Zapewnia wsparcie w wielu starszych przeglądarkach mobilnych.**
   Mimo że większość mobilnych przeglądarek nie powinna się udławić dokumentem HTML5, trzeba się jednak z tym liczyć. Poza tym wiele możliwości oferowanych przez HTML5 najzwyczajniej w świecie nie zadziała na starszych telefonach.

2. **Przypomina nam o potencjalnych problemach z przeglądarkami mobilnymi i pozwala uniknąć wpadek.**
   Jeżeli coś nie jest uwzględnione w specyfikacji XHTML-MP, widocznie był ku temu jakiś powód. Tworzenie dokumentów zgodnych z XHTML-MP powstrzymuje nas od wkraczania na niebezpieczne terytorium. Podobnie jak podejście Mobile First zgodne z RWD wymusza dbałość o porządek i branie pod uwagę ograniczeń, tak samo specyfikacja XHTML-MP narzuca nam ramy, w których możemy się poruszać, by utworzony dokument był obsługiwany przez wiele różnych przeglądarek mobilnych.

Nie istnieją głupie pytania

P: Chyba coś mi umknęło. Czym jest DOCTYPE?

O: DOCTYPE (czyli *definicja typu dokumentu*) jest fragmentem kodu umieszczonym na początku dokumentu SGML i XML, informującym klienta (to znaczy przeglądarkę) o DTD, zgodnie z którym należy przetwarzać ten dokument.

P: To jakieś szaleństwo — same skróty! Dokumenty SGML i XML? DTD?!

O: HTML został stworzony w oparciu o SGML (ang. *Standard Generalized Markup Language* — standardowy uogólniony język znaczników), a XHTML jest typem XML-a (i z tego względu jest bardziej rygorystyczny niż jego kuzyni bez „X" na początku). W obu przypadkach stosuje się definicje DOCTYPE.

Deklaracje typu dokumentu (DTD) są formalnymi dokumentami opisującymi dostępne elementy (znaczniki, atrybuty itd.) i możliwości ich stosowania. Każda specyfikacja HTML i XHTML ma swoją własną deklarację DTD.

Teoretycznie powiązanie dokumentu z DTD powinno wymuszać na kliencie przeprowadzenie walidacji tego dokumentu pod kątem zgodności ze wskazaną deklaracją DTD. Jednak w praktyce przeglądarki tego nie robią.

P: Jaki jest więc cel stosowania definicji DOCTYPE i czemu przeglądarki w ogóle biorą je pod uwagę?

O: Przeglądarki traktują definicje DOCTYPE jedynie jako podpowiedzi co do trybu, w jakim mają renderować dokumenty.

Przypomnij sobie sytuację z przeglądarką Opera Mini, gdy zmiana definicji DOCTYPE z HTML5 na XHTML-MP (lub XHTML-Basic) wpłynęła na sposób wyświetlania strony. To dobry przykład ilustrujący sposób traktowania DOCTYPE przez przeglądarki.

Wiele przeglądarek korzysta z definicji DOCTYPE do ustalenia, czy stosować tryb standardowy, czy tzw. tryb „quirk", czyli tryb dziwactw i osobliwości, w którym przeglądarka stara się być bardziej tolerancyjna i kompatybilna z rozwiązaniami stosowanymi dawniej.

W HTML5 zrezygnowano ze sprawdzania DTD przez przeglądarkę, dzięki czemu dało się skrócić definicję typu dokumentu do `<!DOCTYPE html>` (bez adresu URL do deklaracji DTD).

Wszystko o XHTML-MP

Wywiad tygodnia:
Po co zawracać sobie głowę XHTML-MP?

Zrzędliwy projektant mobilnych stron: Witaj, XHTML-MP. Latka lecą... Ile jeszcze pociągniesz, rok, dwa?

XHTML-MP 1.2: Jeszcze nie umarłem! Co prawda niektórzy sądzą, że już od dawna wącham kwiatki od spodu, ale się mylą — jestem starym, ale przydatnym dziadem. Całe rzesze starszych przeglądarek są mi wdzięczne. Na mnie możesz liczyć — dzięki mnie uda ci się uniknąć wielu problemów.

ZPMS: O jakiego typu problemach mówisz?

XHTML-MP 1.2: Chcesz przykładów? W porządku. Jaki sens ma stosowanie wartości `_new` albo `_blank` w atrybucie `target` znacznika `<a>` w przypadku urządzeń mobilnych? Wiele mobilnych przeglądarek nie pozwala na otwieranie nowego okna z poziomu odsyłacza. W związku z tym nie wspieram atrybutu `target`. O, i jeszcze na przykład ramki. To niebezpieczne ustrojstwo, więc na wszelki wypadek nie obsługuję ich w ogóle. Tak, nawet tych fajnych `iframe`.

ZPMS: Nie obsługujesz ramek `iframe`? To chyba żart!

XHTML-MP 1.2: Nie zapominaj, że w rodzinie XHTML-MP oferujemy funkcję klawiszy dostępu, czyli atrybut `accesskey`. Miał go nawet mój dziadek XHTML-MP 1.0, świeć panie nad jego duszą...

ZPMS: Atrybut `accesskey`?

XHTML-MP 1.2: Cała linia rodu XHTML-MP oferuje atrybut `accesskey` dla znaczników `<a>`. Pozwala on na przypisanie klawiszy cyfr (0 – 9) do konkretnych odsyłaczy. To może być całkiem przydatne w telefonach z klawiaturą numeryczną. W ten sam sposób można rozwiązać problem przewijania zawartości strony.

ZPMS: Czyli klawisze dostępu zostały wynalezione przez XHTML-MP?

XHTML-MP 1.2: No cóż, niezupełnie. To spadek po poprzednich generacjach, od WML zaczynając, przez C-HTML, a na XHTML-MP kończąc. To prawdziwa pamiątka rodowa.

ZPMS: WML? C-HTML?

XHTML-MP 1.2: Ja jeszcze nie umarłem, ale ci dwaj odeszli już dawno temu. WML (ang. *Wireless Markup Language*) był stopniowo wycofywany, aż zupełnie zniknął. Był zupełnie inny niż HTML, więc moja rodzina wymieniła go na mobilny wariant języka znaczników.

Z kolei C-HTML (ang. *Compact HTML*) był stosowany głównie w Japonii przez firmę NTT DoCoMo. Niekiedy mówi się o nim iMode. Potrzebne ci emotikonki? Nie ma sprawy — C-HTML da ci je od ręki. Ale jest kilka „ale”: brak obsługi tabel, stylów i obrazków. No i brak kolorów. Tak jest, stare dobre czasy...

Obejrzyj to!

Musimy dbać o porządek w kodzie.

Całkiem poważnie mówiliśmy o udławieniu się kodem. Mobilne przeglądarki (zwłaszcza te starsze) mają mniejszą odporność na błędy w strukturze dokumentu niż ich desktopowe odpowiedniki. Musisz dbać o to, by kod był poprawny, ponieważ pewne błędy mogą spowodować wysypanie się przeglądarki, a w niektórych przypadkach nawet wyłączenie się telefonu. Oj!

Kuba: Jak sądzisz, Łukaszu, powinniśmy skorzystać z tego XHTML-MP czy nie?

Łukasz: Wygląda na to, że to najbezpieczniejsze wyjście. Dla dobra starszych przeglądarek powinniśmy zadbać o porządek i prostotę.

Kuba: Czy to znaczy, że musimy wprowadzić jeszcze jakieś zmiany?

Łukasz: Fajnie, że zmiana definicji `DOCTYPE` na XHTML-MP rozwiązała problem pływających elementów `<div>` w Operze Mini, ale zaczynam się zastanawiać, czy w ogóle z nich nie zrezygnować. **To straszne, ale pomyślałem, że układ tej strony moglibyśmy zrobić na tabelach.**

Kuba: Bez przesady, przecież układ realizowany na tabelach to relikt lat 90. zeszłego wieku!

Łukasz: Zwykle unikam ich jak ognia, ale w tym przypadku ich użycie nie będzie aż takim dużym błędem. Przecież treść strony przypomina dane tabelaryczne. Po prostu nie jestem pewien, czy układ oparty na pływających elementach `<div>` będzie prawidłowo wyświetlany na urządzeniach, które mamy obsłużyć.

Kuba: Tak, takie stosowanie tabel nie jest fajne, ale przynajmniej wiemy, że to będzie działać. Tak przy okazji — a co ze stylami, które zdefiniowaliśmy dla naszej makiety?

Łukasz: Nie do końca wiem, których elementów HTML-a i CSS możemy bez obaw użyć.

Najlepiej będzie się zająć jedną sprawą naraz. Zacznę od sprawdzenia, które elementy HTML-a są obsługiwane na tych telefonach. **W związku z tym wytnę na razie cały CSS i zajmę się nim później.**

Chłopaki! Jeszcze jedna sprawa, zanim zabierzecie się za kolejną makietę. Dostaliśmy informację z organizacji Zwierzętom na Pomoc, że wielu użytkowników ma telefony z bardzo małymi ekranami, więc muszą przewijać zawartość strony, by dostać się do poszczególnych elementów pulpitu. Macie pomysł, co z tym zrobić?

Do odsyłaczy z bloku `<div>` #tools możemy przypisać atrybuty `accesskey`, dzięki czemu dostęp do poszczególnych sekcji strony będzie szybszy.

Bez konieczności przewijania strony.

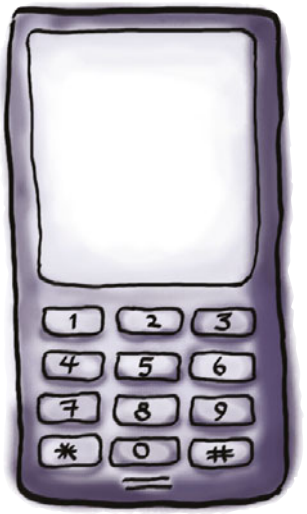
Przy okazji — przewijanie jest do bani

Nie wszyscy mogą korzystać z dobrodziejstw ekranu dotykowego. Na wielu telefonach do nawigowania po stronach i ich przewijania trzeba korzystać z klawiszy. W wielu interfejsach użytkownika poszczególne odsyłacze są podświetlane w miarę przewijania strony, tak by można je było „kliknąć". Im więcej odsyłaczy i treści, tym więcej przewijania.

...na szczęście mamy klawisze dostępu

Atrybut `accesskey` znacznika `<a>` pozwala na przypisanie do odsyłaczy numerycznego klawisza skrótu, dzięki czemu użytkownik nie musi przewijać strony, ponieważ wystarczy, że wciśnie odpowiedni klawisz na swoim telefonie. Atrybut `accesskey` umieszcza się w kodzie w taki sposób:

```
<a href="#dashboard" accesskey="1">Twój pulpit</a>
```



Dzięki klawiszom dostępu użytkownicy mogą użyć klawiszy numerycznych jako skrótów!

Liczba przypisana do odsyłacza nie jest wyświetlana, więc często dobrym rozwiązaniem jest zastosowanie do zestawu odsyłaczy listy numerowanej (`<ol>`) w miejsce nienumerowanej (`<ul>`), oczywiście przy założeniu, że do atrybutu `accesskey` pierwszego odsyłacza jest przypisana wartość 1.

Ostatni problem

W porządku, jest jeszcze coś, o czym Ci nie wspomnieliśmy. Zrozumiemy, jeśli spowoduje to gwałtowną reakcję Twojego organizmu wraz z towarzyszącymi niepokojącymi odgłosami. Otóż:

W 2008 roku XHTML-MP 1.2 został zastąpiony przez XHTML-Basic. Przepraszamy.

Specyfikacja XHTML-Basic 1.1 daje Ci wszystko, co było w XHTML-MP 1.2, i dodaje kilka bonusów. Masz do dyspozycji na przykład atrybut `target` (nie sugerujemy bynajmniej, że musisz go używać). Możesz też korzystać ze znaczników `<sup>` i `<sub>`. Jednym słowem, nie tracisz nic, co oferowała specyfikacja XHTML-MP 1.2.

Poza tym XHTML-MP 1.2 nadal działa — i to dobrze — w przeglądarkach mobilnych.

Jest całe mnóstwo możliwości stosowania języka znaczników w urządzeniach mobilnych. W tym rozdziale pozostaniemy jednak przy specyfikacji XHTML-Basic.

Przykro nam, że musiałeś przez to przejść, ale to dla Twojego dobra. Świat znaczników dla technologii mobilnych jest złożony, a Ty powinieneś się mniej więcej w nim orientować. Od teraz nie będziemy się już miotać i pozostaniemy przy XHTML-Basic. To nie będzie takie trudne: wystarczy zidentyfikować znaczniki, które nie są obsługiwane, i ich nie stosować. W tej chwili nie musimy się nawet przejmować stylami i układem strony.

Jazda próbna

1. **Przekonwertuj dokument na XHTML-Basic.**
Bieżącą definicję DOCTYPE zamień na poniższą:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
"http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
```

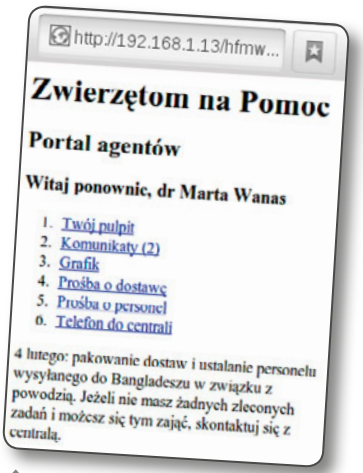
2. **Usuń odwołanie do arkusza stylów.**
Usuń znacznik <link> odwołujący się do arkusza stylów.
CSS zajmiemy się później.

3. **Znacznik <ul> znajdujący się w bloku #tools zamień na <ol>.**
Dzięki temu przy każdym odsyłaczu pojawi się kolejny numer, co ułatwi nawigację (patrz kolejny krok).

4. **Do znaczników <a> znajdujących się w bloku #tools dodaj atrybut accesskey.**
Jeżeli masz wątpliwości co do składni, zajrzyj na stronę 119.

5. **Pływający, oparty na elementach blok #dashboard przekonwertuj na tabelę.**
Zawartość bloku o identyfikatorze #dashboard zamień na tabelę.
To dosyć uciążliwe, więc przygotowaliśmy gotowy kod, który znajdziesz na stronie 121 (a także w pliku *table.txt* umieszczonym w katalogu *rozdzial3/extras*).

6. **Zmieniony dokument zapisz w pliku index.html.**
Dokument zapisz w pliku *index.html*. Przetestowaliśmy już przekierowanie do mobilnej strony, więc wygodniej będzie używać pliku *index.html* (zamiast *index_mobile.html*).



Nasza odchudzona, ale nadal funkcjonująca makieta wyświetlona w Androidzie.

Uważaj, żeby nie usunąć elementu #dashboard. Masz usunąć tylko jego zawartość.

Nie istnieją głupie pytania

P: Zatem zamierzamy teraz maksymalnie uprościć makietę. Ale zanim zmienię definicję DOCTYPE na XHTML-Basic, chciałbym wiedzieć, czy makieta ze strony 115 nadal działa w Operze Mini. Jak to sprawdzić?

O: Jasne! Jeżeli jesteś ciekawy, zobacz sam: *ftp://ftp.helion.pl/online/inne/hfmw/r03/2a*.

P: Czy Opera Mini 4.2 nie jest już całkiem przestarzała? Ilu użytkowników z niej korzysta?

O: Trzeba szczerze przyznać, że nie jest to najnowsza przeglądarka. Musisz sobie jednak zdawać sprawę, że w świecie technologii mobilnych będziesz miał do czynienia również z takimi „staruszkami". To dobry przykład *specyficznych* przeglądarek ze *specyficznymi* ograniczeniami działających na starszych, prostych telefonach komórkowych, co jest bardzo powszechne w krajach rozwijających się.

P: Ale czy nie wspomnieliście przed chwilą, że sprzedaż smartfonów rośnie bardzo szybko i niedługo przekroczy sprzedaż zwykłych telefonów?

O: To, że rośnie sprzedaż, nie znaczy, że smartfony wyparły telefony. Cały czas w użyciu jest wiele starszych smartfonów (które — jak na dzisiejsze standardy — nie są szczytem techniki) oraz telefonów komórkowych. Musi jeszcze minąć trochę czasu, zanim smartfony zdominują ten rynek.

KOD TABELI gotowy do użycia

```
<table>
   <thead>
   <tr>
      <th>Typ</th>
      <th>Szczegóły</th>
   </tr>
   </thead>
   <tbody>
      <tr>
         <td><p>Komunikat<br />3 lut 20:54 CET</p></td>
         <td><p><a href="#">Prośba o dostawę #493-C4 &ndash; zaakceptowana</a><br />
         Cześć, Marto. Dobre nowiny! Chcę Ci powiedzieć, że już sprowadzamy te butelki, o które
prosiłaś.</p></td>
      </tr>
      <tr>
         <td><p>Komunikat<br />3 lut 13:47 CET</p></td>
         <td><p><a href="#">Prośba o dostawę #493-C4 &ndash; odebrana</a><br />
         To jest wiadomość wygenerowana automatycznie. Twoja prośba o dostawę została odebrana
i czeka na rozpatrzenie...</p></td>
      </tr>
      <tr>
         <td><p>Grafik<br />3 lut 8:22 CET</p></td>
         <td><p><a href="#">Plan 39924 &ndash; zaakceptowany</a><br />
         Twój plan dla zdarzenia "Powódź w Bangladeszu" został zaakceptowany. Twój kalendarz został
zaktualizowany...</p></td>
      </tr>
      <tr>
         <td><p>Personel <br />2 lut 21:23 CET</p></td>
         <td><p><a href="#">Re: Potwierdzenie wyjazdu 05.03 - 15.03</a><br />
         Marto! Bardzo dziękujemy, że zgłosiłaś się do tego zadania! Z tego, co wiem, jedzie też
dr Doktor i...</p>
         </td>
      </tr>
      <tr>
         <td colspan="2"><div class="morelink"><p><a href="#">Więcej &gt;&gt; </p>
         </a></div></td>
      </tr>
   </tbody>
</table>
```

Ćwiczenie

W której specyfikacji są poprawne następujące elementy? Skorzystaj z wiedzy, którą zdobyłeś do tej pory, wspomóż się intuicją i dla każdego znacznika zaznacz specyfikacje, w których występuje. Uwaga: niektóre elementy mogą być poprawne w więcej niż jednym standardzie.

	XHTML-MP 1.2	XHTML-Basic 1.1	HTML5
<font>	☐	☐	☐
<table>	☐	☐	☐
<a target="">	☐	☐	☐
<sup>	☐	☐	☐
<a accesskey="">	☐	☐	☐
<u>	☐	☐	☐
<tbody>	☐	☐	☐
<link>	☐	☐	☐
	☐	☐	☐
<video>	☐	☐	☐
<iframe>	☐	☐	☐

→ Odpowiedź znajdziesz na stronie 126.

Przysięgamy, że to działa na większości telefonów, ale nie musisz nam wierzyć na słowo.

Klawisze dostępu są obsługiwane również w większości desktopowych przeglądarek.

Klawisze dostępu w akcji

Jeśli chcesz przetestować klawisze dostępu, możesz otworzyć plik *index.html* w przeglądarce desktopowej. Aby wywołać skróty, użyj następujących kombinacji klawiszy:

Chrome i Safari: *Ctrl+Opt [accesskey]*
Firefox: *Ctrl+[accesskey]*

MacOS

Chrome i Safari: *Alt+[accesskey]*
Firefox: *Alt+Shift+[accesskey]*
Internet Explorer: *Alt+[accesskey]*

Windows

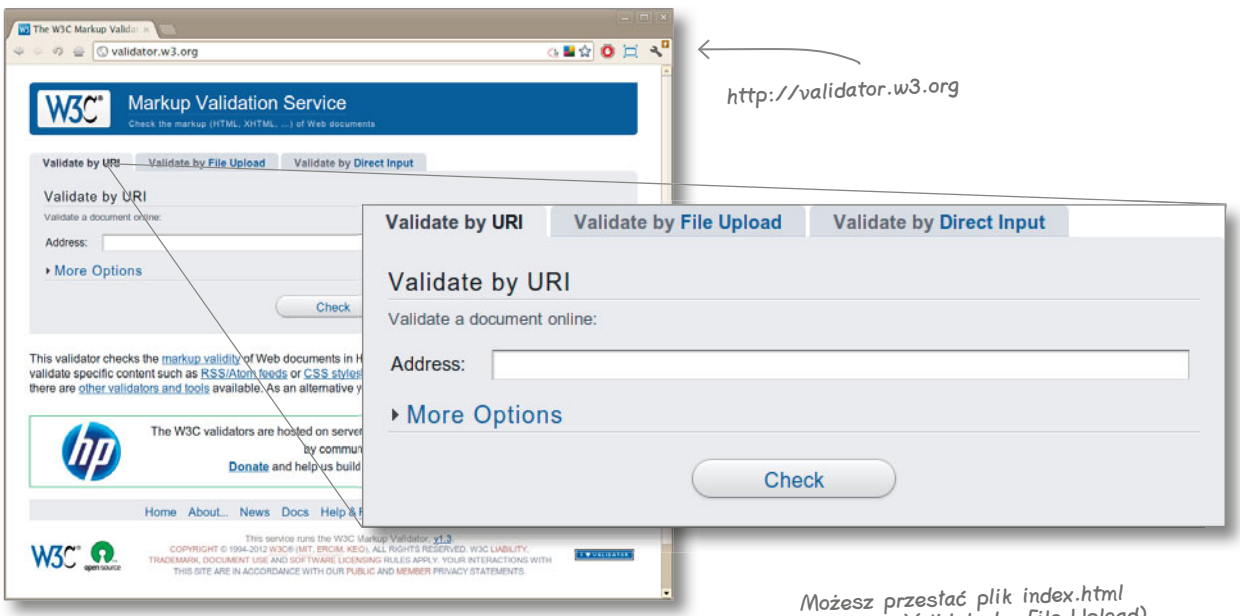
Chrome: *Alt+[accesskey]*
Firefox: *Alt+Shift+[accesskey]*

Linux

W związku z różnymi konfiguracjami systemu operacyjnego i zainstalowanym oprogramowaniem niektóre skróty klawiaturowe mogą być zajęte.

Odrobina walidacji

Pamiętasz, jak mówiliśmy, że musisz dbać o poprawność kodu dokumentu? Czas przejść od słów do czynów — przeprowadzimy walidację kodu makiety, by się upewnić, że wszystko jest w porządku. Skorzystamy z nieocenionego narzędzia, jakim jest **W3C Markup Validation Service**.

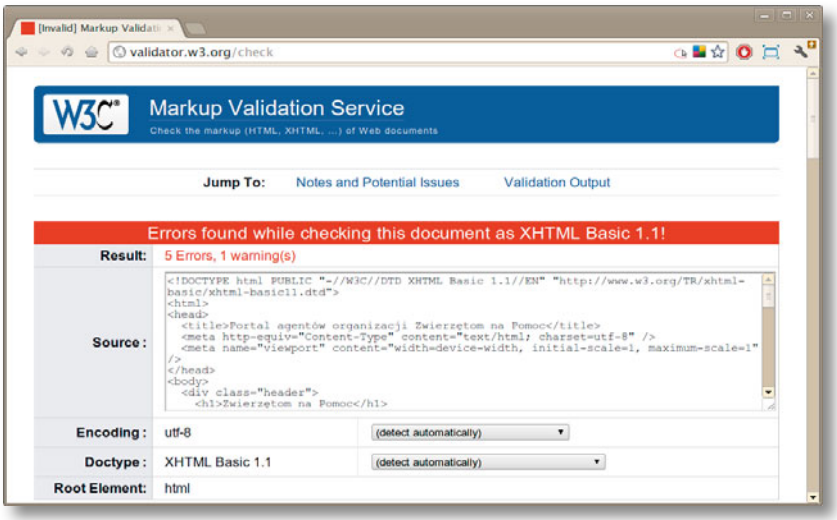


http://validator.w3.org

Możesz przesłać plik index.html (zakładka Validate by File Upload) lub wkleić jego zawartość (zakładka Validate by Direct Input).

I jak tam? Wszystko gra?

O nie!



Co poszło nie tak?

Przewiń stronę z wynikami walidacji, by zobaczyć szczegółowe informacje o błędach. W czym zatem tkwi problem?

1. **Znaczniki <thead> i <tbody> nie są obsługiwane w specyfikacji XHTML-Basic (ani w XHTML-MP).**
   To jest największe niedociągnięcie w specyfikacjach XHTML-Basic i XHTML-MP. Możesz używać tabel, ale nie możesz stosować znaczników `<thead>`, `<tbody>`, `<tfoot>`, `<col>` i `<colgroup>`.

Te trzy błędy są związane z brakiem obsługi znaczników <thead> i <tbody>.

Line 35, Column 10: element "thead" undefined

<thead>

Line 41, Column 10: element "tbody" undefined

<tbody>

Line 68, Column 8: end tag for "table" which is not finished

</table>

Trzeci błąd jest efektem ubocznym problemu ze znacznikami <thead> i <tbody>. Kiedy usuniemy oba znaczniki, ten błąd zniknie.

Naprawiamy błędy

2 **Jest kilka nieprawidłowo zagnieżdżonych znaczników.**
Nawet profesjonalistom zdarza się czasem popełnić takie błędy. To jeden z powodów, dla których warto korzystać z walidatora, nawet jeżeli uważasz się za mistrza.

Z pliku index.html.

```
<td colspan="2"><div class="morelink">
<p><a href="#">Więcej &gt;&gt;</p></a></div>
</td>
```

Line 64, Column 82: **end tag for "a" omitted, but OMITTAG NO was specified**

```
…      <td colspan="2"><div class="morelink"><p><a href="#">Więcej &gt;&gt; </p>
```

Line 65, Column 13: **end tag for element "a" which is not open**

```
          </a></div></td>
```

Line 64, Column 51: start tag was here

```
…      <td colspan="2"><div class="morelink"><p><a href="#">Więcej &gt;&gt; </p>
```

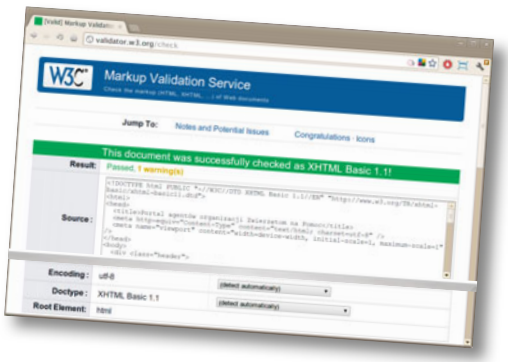
Oba te błędy są spowodowane złym umieszczeniem znacznika .

To nie jest tak naprawdę błąd — to informacja z walidatora z podpowiedzią dotyczącą wykrytego problemu.

Jazda próbna

Popraw kod powodujący błędy w pliku *index.html*.

1. Usuń znaczniki `<thead>`, `</thead>`, `<tbody>` i `</tbody>`.
2. Popraw nieprawidłowo zagnieżdżone znaczniki. Znacznik `</a>` z bloku `<div>` `.morelink` umieść wewnątrz bloku akapitu.
3. Ponownie sprawdź zaktualizowany kod w walidatorze W3C.



Super! Zielony kolor oznacza sukces.

Jakie znaczniki są poprawne w podanych standardach?

Rozwiązanie ćwiczenia

	XHTML-MP 1.2	XHTML-Basic 1.1	HTML5
<font>	☐	☐	☐
<table>	☑	☑	☑
<a target="">	☐	☑	☑
<sup>	☐	☑	☑
<a accesskey="">	☑	☑	☑
<u>	☐	☐	☐
<tbody>	☐	☐	☑
<link>	☑	☑	☑
	☑	☑	☑
<video>	☐	☐	☑
<iframe>	☐	☐	☑

Znaczniki <font> i <u> są przestarzałe — w ich miejsce stosuj style!

W porządku. Mamy już poprawny kod i podstawową strukturę dokumentu, ale wygląda to dosyć blado. Może przywrócimy trochę stylów?

CSS dostosowany do przeglądarek mobilnych

Specyfikacje **CSS Mobile Profile 2.0** (CSS-MP) i XHTML-Basic (lub XHTML-MP) są jak papużki nierozłączki. Specyfikacja CSS-MP została opracowana z myślą o telefonach z dolnej i średniej półki.

Co mamy ciekawego w CSS Mobile Profile? Nie jest źle — znajdziesz tu większość tego, czego mógłbyś się spodziewać z CSS2, ale też kilka elementów z CSS3. Chyba nie musimy tłumaczyć, że wszystkiego tu nie znajdziesz.

W teorii brzmi to nieźle

Przeglądarki, w których jest zaimplementowana specyfikacja CSS Mobile Profile 2.0, powinny obsługiwać wymagane właściwości. Niestety w praktyce nie wygląda to już tak pięknie. Wsparcie dla CSS różni się w poszczególnych przeglądarkach, a cały ciężar związany ze sprawdzaniem zgodności i wymyślaniem obejść napotkanych problemów spada oczywiście na Ciebie — nieustraszonego projektanta.

Poza tym niektóre właściwości stylów i ich wartości są *opcjonalne* w specyfikacji CSS-MP, co oznacza, że twórcy przeglądarek nie muszą ich implementować.

Porada dnia: mniej gadania, więcej pisania!

Zamiast złorzeczyć nad lekturą listy obsługiwanych i nieobsługiwanych właściwości CSS w specyfikacji CSS Mobile Profile, **weź się do roboty i postaraj się dostosować dotychczasowy arkusz stylów do wymagań stawianych przez tę specyfikację.**

Długie ćwiczenie

Zapnij pasy i przygotuj się na ciężką pracę oraz nieprzespane noce. Wybieramy się w podróż. Podróż, która — przy pomyślnych wiatrach — zakończy się szczęśliwym małżeństwem XHTML-Basic z CSS Mobile Profile. Będziemy pracować zarówno z plikiem *index.html*, jak i *styles.css*.

1 W pliku index.html umieść z powrotem znacznik <link>.

Chcemy znowu użyć arkusza stylów, więc potrzebujemy znacznika `<link>`.

```
<head>
    <title>Portal agentów organizacji Zwierzętom na Pomoc</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1,
maximum-scale=1" />
    <link rel="stylesheet" type="text/css" href="styles.css" />
</head>
```

index.html

2 Zrefaktoryzuj arkusz styles.css, tak by był zgodny ze specyfikacją CSS Mobile Profile.

Co prawda właściwości `background-image` i `background-repeat` są obsługiwane w CSS-MP, ale nagłówek z obrazkiem w tle zajmuje dużo cennej przestrzeni ekranu, więc z niego rezygnujemy.

Wyrażanie wartości właściwości background-position w procentach i pikselach nie jest wspierane w CSS-MP, więc musielibyśmy to zmienić, gdybyśmy zostawili obrazek w tle.

Ale nie zostawiamy tego obrazka, ponieważ z jego obsługą bywa różnie. Nawet wsparcie dla właściwości background-color nie jest pewne.

```
* {
    padding: 0;
    margin: 0;
}
body {
    font-family: "Helvetica", "Arial", san-serif;
    font-size: 100%;
    width: 100%;
    background-color: #f3ffc2;
    background-image: url('cows.jpg');
    background-repeat: no-repeat;
    background-position: 50% 0px;
}
p {
    font-size: .95em;
    margin: 0.25em 0;
}
h1, h2, h3, h4, h5 {
    font-family: "Times New Roman", serif;
    margin: 0;
    color: #10508c;
    text-align: center;
}
```

styles.css

Pozostałe elementy w tym fragmencie CSS są zgodne ze specyfikacją, więc lecimy dalej!

styles.css

```
h3 {
    font-style: italic;
    font-weight: 100;
    font-size: 1.15em;
}
ol {
    width: 100%;
}
a {
    text-decoration: none;
    color: #096c9f;
}
.header {
    height: 150px;
}
#tools ol {
    list-style-type: none;
}
#tools ol li a {
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    border-radius: 5px;
    display: block;
    height: 1.1em;
    width: 94%;
    background-color: #fff;
    margin: 3%;
    border: 1px solid #ccc;
    text-align: center;
    padding: .6em 0;
}
.greeting {
    border: 1px dashed #10508c;
    border-width: 1px 0;
}
#dashboard {
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    border-radius: 5px;
    background-color: #fff;
    border: 1px solid #ccc;
    margin: 1em 3%;
    padding: .5em 0;
    width: 94%;
}
```

Od czasu, kiedy ostatni raz korzystaliśmy z tego CSS, zmieniliśmy znacznik <ul> na <ol>. Musimy więc zaktualizować te dwie reguły, żeby wszystko pasowało.

Te właściwości, odpowiedzialne za zaokrąglanie wierzchołków, są niepoprawne w CSS-MP. Zresztą zdecydowana większość starszych przeglądarek i tak ich nie obsługuje. Usuń cały kod związany z zaokrąglaniem.

Ciąg dalszy na kolejnej stronie.

Długie ćwiczenie

Wracamy do pliku index.html: do wybranych elementów tabeli trzeba dodać klasy.

Hej, mądralo! Zastosuj te same klasy do pozostałych trzech wierszy.

Ale ustaw na przemian parzyste (tr.even) i nieparzyste (tr.odd) wiersze.

```
<tr class="even">
<td class="event event_meta"><p><strong>Komunikat</strong><br
/>
3 lut 20:54 CET</p>
</td>
<td class="event"><p><a href="#"><strong>Prośba o dostawę #493-C4 &ndash;
zaakceptowana</strong></a><br />
Cześć, Marto. Dobre nowiny! Chcę Ci powiedzieć, że już sprowadzamy te
butelki, o które prosiłaś.</p></td>
</tr>
```

index.html

4 **A teraz kolej na plik styles.css: dostosuj style do układu opartego na tabelach.** Układ oparty na blokach `<div>` zmieniliśmy na zbudowany w oparciu o tabelę. Będzie trzeba pozbyć się niektórych właściwości, które nie mają już sensu, a kilka innych trzeba będzie zmodyfikować.

Usuń te trzy właściwości.

Żegnamy się z właściwością float, a poza tym ustawiamy szerokość i marginesy.

```
#dashboard td.event {
    border: 1px dashed #ddd;
    border-width: 1px 0 0 0;
    margin: .5em 0;
    padding: 0.5em 0;
    width: 100%;
    clear: both;
    overflow: hidden;
}
#dashboard .odd {
    background-color: #fff;
}
#dashboard .even {
    background-color: #eee;
}
#dashboard td.event_meta {
    width: 30%;
    margin: 0 2%;
    float: left;
}
```

styles.css

5 **Dodaj tę regułę do pliku styles.css...**

Dodaj tę nową regułę (gdziekolwiek w pliku styles.css).

```
#dashboard table {
    border-collapse: collapse;
    width: 100%;
}
```

styles.css

6 **...i usuń kilka niepotrzebnych.**

Odszukaj i wykasuj poniższe reguły:

Usuń wszystkie te reguły!

```
#dashboard div
#dashboard .event_time
#dashboard .event_details
#dashboard .event_subject
#dashboard .event_summary
```

styles.css

7 **Napraw element #status_message.**

I, na koniec, pozbądź się tego.

```
#status_message {
    background-color: #f8f1b2;
    padding: 0.25em;
    line-height: 1.3em;
    border: 1px solid #ccc;
    border-width: 1px 0;
    height: 70px;
    overflow: scroll;
}
```

styles.css

Strzeż się właściwości `overflow`!

W świecie technologii mobilnych `overflow: scroll` jest tabu. Ta właściwość nie jest obsługiwana w zasadzie przez żadną platformę, chociażby dlatego, że zmuszanie do przewijania zawartości w urządzeniach mobilnych nie jest mile widziane.

Zachowanie właściwości `overflow` w przeglądarkach implementujących CSS-MP jest różne. Jedyną wartością, która musi być obsługiwana, jest `auto`. W związku z tym za wszelką cenę unikaj stosowania tej właściwości.

8 **Zapisz pliki. To wszystko!**

Hm... czegoś tu brakuje

Spójrz na ten zrzut ekranu z symulatora Opery Mini 4.2. Zauważyłeś coś niepokojącego? Zniknęły numery z elementów listy `<ol>`, do których przypisaliśmy atrybuty `accesskey`. O nie!

Gdzie się podziały numery?

Numery nadal tam są, ale zastosowaliśmy CSS, który definiuje wygląd elementów `<li>`, tak by wyglądały jak przyciski zajmujące całą szerokość ekranu. W związku z tym numery znalazły się poza lewą krawędzią widocznego obszaru strony.

Standardowym rozwiązaniem jest zastosowanie czegoś takiego:

```
#tools ol {
   list-style-position: inside;
}
```

Tak wygląda teraz nasza makieta. Na przyciskach nie ma numerów.

Ustawienie `list-style-position: inside` przenosi numery do wnętrza elementu `<li>` i umieszcza je (w tym przypadku) obok zawartości tego elementu (czyli odsyłacza). Zgadza się, prawda?

Niestety właściwość **`list-style-position` nie jest obsługiwana w CSS-MP**. Mamy więc problem. Moglibyśmy zmienić to menu w listę zwykłych odsyłaczy, tak by nie wyglądały jak przyciski. Numery powinny się wtedy znów pokazać.

Pa, pa, przyciski

Aby ustalić prawidłowo wyglądającą listę odsyłaczy, musimy zmienić reguły dla `#tools ol` i `#tools ol li` na taką regułę:

```
#tools ol {
   margin: 0.5em 1.5em;
}
```

Jeśli zamienimy dotychczasowe reguły na tę...

PS. Właściwości padding i margin dla wszystkich elementów (*) zostały ustawione na 0. W związku z tym dla tej listy musimy dodać niewielki margines, aby numery były widoczne na ekranie.

=

Witaj ponownie, dr Marta Wanas

1. Twój pulpit
2. Komunikaty (2)
3. Grafik
4. Prośba o dostawę
5. Prośba o personel
6. Telefon do centrali

...numerowana lista `<ol>` powróci.

Jednak kiedy przedstawiciele organizacji Zwierzętom na Pomoc spojrzeli na nowy układ...

Bardzo nam brakuje tych przycisków!

Hm... Zdecydowanie bardziej podobały mi się odsyłacze w kształcie przycisków.

Pracownicy organizacji Zwierzętom na Pomoc bardzo się przyzwyczaili do przycisków, więc chcą, by powróciły.

Jedynym rozwiązaniem tego problemu, przy zachowaniu poprawności pod kątem CSS-MP, jest przywrócenie listy `<ul>`, dodanie numerów do treści odsyłaczy i poprawienie reguł CSS.

Ech, te kompromisy... Tak to niestety wygląda w świecie technologii mobilnych!

Jazda próbna

Zmień selektory w pliku *styles.css*, a w pliku *index.html* znacznik `<ol>` zmień z powrotem na `<ul>`. Nie zapomnij dopisać do treści odsyłaczy kolejnych numerów odpowiadających wartościom atrybutów `accesskey`.

Plik styles.css.

W tym miejscu umieść kod CSS, który stosowaliśmy dla #tools ol (ze strony 129).

```
#tools ul {
    ...
}
#tools ul li a {
    ...
}
```

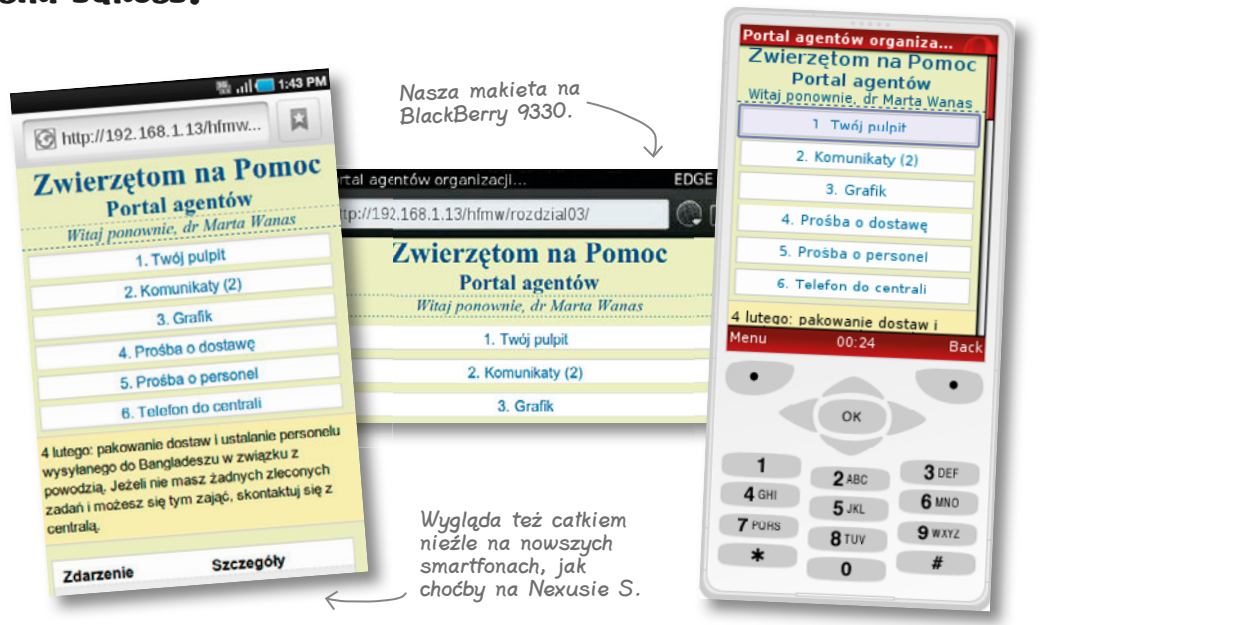
A tu umieść kod CSS, który był wpisany dla #tools ol li.

W pliku index.html znacznik <ol> zmień z powrotem na <ul>.

```
<div id="tools">
    <ul>
        <li><a href="#dashboard" accesskey="1">1. Twój pulpit</a></li>
        <li><a href="#" accesskey="2">2. Komunikaty (2)</a></li>
        <li><a href="#" accesskey="3">3. Grafik</a></li>
        <li><a href="#" accesskey="4">4. Prośba o dostawę</a></li>
        <li><a href="#" accesskey="5">5. Prośba o personel</a></li>
        <li><a href="#" accesskey="6">6. Telefon do centrali</a></li>
    </ul>
</div>
```

index.html

Wielki sukces!



CELNE SPOSTRZEŻENIA

- Świat jest całkiem spory i są na nim miliony (serio) telefonów komórkowych. Nie wszystkie to najnowsze smartfony. Czasem będziesz musiał stworzyć stronę lub aplikację, która działa na tych „innych" telefonach.
- W rzeczywistych sytuacjach możesz napotkać wiele ograniczeń: starsze systemy, oporni użytkownicy lub specyficzne wymagania projektowe, które wymuszają utworzenie **oddzielnej witryny mobilnej**.
- Jednym ze sposobów na przekierowanie ruchu sieciowego na oddzielną witrynę mobilną jest skorzystanie z **wykrywania urządzeń mobilnych po stronie serwera**.
- Popularna technika **user-agent sniffing** pozwala na sprawdzenie, czy przychodzące żądanie zostało zgłoszone przez przeglądarkę mobilną.
- Technika **user-agent sniffing** polega na sprawdzeniu nagłówka `User-Agent` przesyłanego przez przeglądarkę wraz z każdym żądaniem HTTP. Użytkownicy mogą zmodyfikować (świadomie lub nie) nagłówek `User-Agent`. To jest najpoważniejsza słabość tej metody.
- Niektóre starsze przeglądarki (ale również nowsze, ale działające na słabszych urządzeniach) implementują różne standardy HTML-a i CSS.
- **XHTML Mobile Profile** (XHTML-MP) jest standardem stosowanym przez wiele przeglądarek mobilnych. Jest podobny do XHTML-a, ale nie wspiera wszystkiego.
- Podobnie sprawa wygląda ze standardem **CSS Mobile Profile** (CSS-MP), który jest mobilnym odpowiednikiem CSS.
- XHTML-MP został zastąpiony przez **XHTML-Basic 1.1**, który — poza wsparciem dla kilku nowych elementów — nie różni się zbytnio od poprzednika.
- Bardzo ważne jest wybranie w projekcie **właściwego typu dokumentu (DOCTYPE)** i dbanie o **poprawność kodu**. Błędy w kodzie mogą spowodować poważne problemy z urządzeniem mobilnym.

Nie istnieją głupie pytania

P: Zamiast HTML-a stosujemy XHTML-Basic lub XHTML-MP, a zamiast CSS — CSS Mobile Profile. Czy jest jakiś odpowiednik JavaScriptu dla urządzeń mobilnych?

O: Tak, jest mobilna wersja JavaScriptu, która nazywa się ECMAScript Mobile Profile, ale za bardzo nie możesz na nią liczyć. Pracowaliśmy już z telefonami, które nie pozwalają na zmianę czegokolwiek po załadowaniu strony z poziomu skryptu, a przecież do tego przeważnie wykorzystuje się JavaScript. Zatem jeśli wiesz, że Twoim celem jest zapewnienie wsparcia dla starszych telefonów, lepiej w ogóle nie stosuj JavaScriptu.

P: A co ze standardem Wireless CSS?

O: Wireless CSS to standard bardzo podobny do CSS Mobile Profile, ale obsługuje mniej właściwości. Wygląda na to, że kończy swój żywot, więc nie widzimy żadnych powodów, dla których miałbyś go używać zamiast CSS-MP.

P: Powiedzmy, że korzystam z techniki user-agent sniffing i zdarzyło się, że źle zidentyfikowałem przeglądarkę użytkownika — czy to ze względu na „oszukany" nagłówek `User-Agent`, czy ze względu na błąd po mojej stronie. Czy użytkownik tej przeglądarki utknie na mobilnej witrynie i nie będzie mógł wyświetlić pełnej, desktopowej wersji?

O: Bardzo dobre pytanie! I bardzo istotny problem. W rzeczywistych projektach powinieneś uwzględnić „koło ratunkowe", najczęściej w postaci odsyłacza, który prowadzi do desktopowej wersji witryny.

Jednak sam odsyłacz nie wystarczy — musisz jeszcze poinformować skrypt przekierowujący o tym, że użytkownik chce zobaczyć desktopową witrynę. Można to zrealizować na przykład za pomocą ciasteczka, które — przekazane wraz z żądaniem na serwer — poinformuje o tym, że użytkownik nie chce zostać przekierowany na mobilną witrynę.

P: A co, jeśli chcąc zachować pełną funkcjonalność mojej witryny, nie mogę doprowadzić do sytuacji, w której jest ona w pełni zgodna z XHTML-Basic, CSS-MP czy innym standardem? Czy świat legnie w gruzach? Czy telefony na całym świecie padną?

O: Sukces w świecie technologii mobilnych bardzo często zależy od gotowości do kompromisów i ustępstw. Chociaż dbałość o poprawność kodu jest bardzo ważna, czasem nie jest możliwe zapewnienie stuprocentowej zgodności ze standardami. Jednak zanim złamiesz reguły, musisz je znać i wiedzieć, z czym się to wiąże.

P: Czy jeśli chcę, by moja witryna była dostępna dla wielu różnych przeglądarek mobilnych, zawsze muszę stosować standard XHTML-Basic? To mnie strasznie ogranicza.

O: No cóż, na to pytanie udzielimy Ci klasycznej odpowiedzi: „to zależy". W przypadku witryny dla organizacji Zwierzętom na Pomoc musieliśmy się liczyć z tym, że będą z niej korzystać użytkownicy wielu starszych telefonów o małych możliwościach. Naszym celem było więc takie przygotowanie witryny, by była wyświetlana w tak wielu typach telefonów, jak to tylko możliwe.

Jednak w wielu sytuacjach, nawet w przypadku starszych telefonów, rozsądne wydaje się zastosowanie HTML5. W tym rozdziale chcieliśmy Ci jednak pokazać, z jakimi wyzwaniami musisz się zmierzyć i na co uważać w projektach realizowanych dla starszych telefonów.

P: Na stronie 127 wspominacie o ograniczeniach standardu CSS-MP. Gdzie mogę znaleźć pełną listę wspieranych i niewspieranych właściwości?

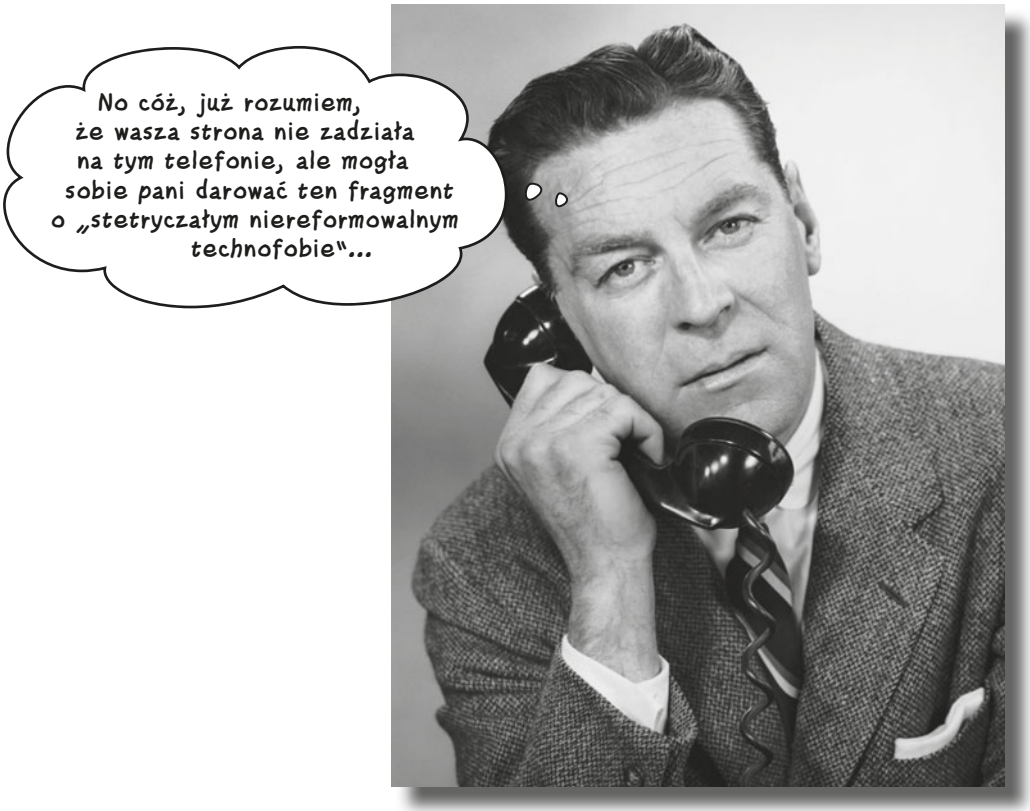
O: Masz szczęście, ponieważ specyfikacja CSS Mobile Profile 2.0 jest stosunkowo krótka i łatwo się ją czyta. Możesz ją znaleźć pod adresem *http://www.w3.org/TR/css-mobile*.

WYSIL SZARE KOMÓRKI

Hej, superprogramisto! Jak rozwiążemy problem dostępności wersji desktopowej? Czego będziemy potrzebować, by dodać odsyłacz kierujący z mobilnej witryny na desktopową, i co należy zmienić w skrypcie *redirect.php*, by sprawdzać w nim ciasteczko z ustawieniami dotyczącymi użytkownika?

Niełądnie jest więzić użytkownika na ograniczonej witrynie mobilnej, skoro chciałby uciec do jej pełnej, desktopowej wersji.

Które urządzenia powinny być obsługiwane?

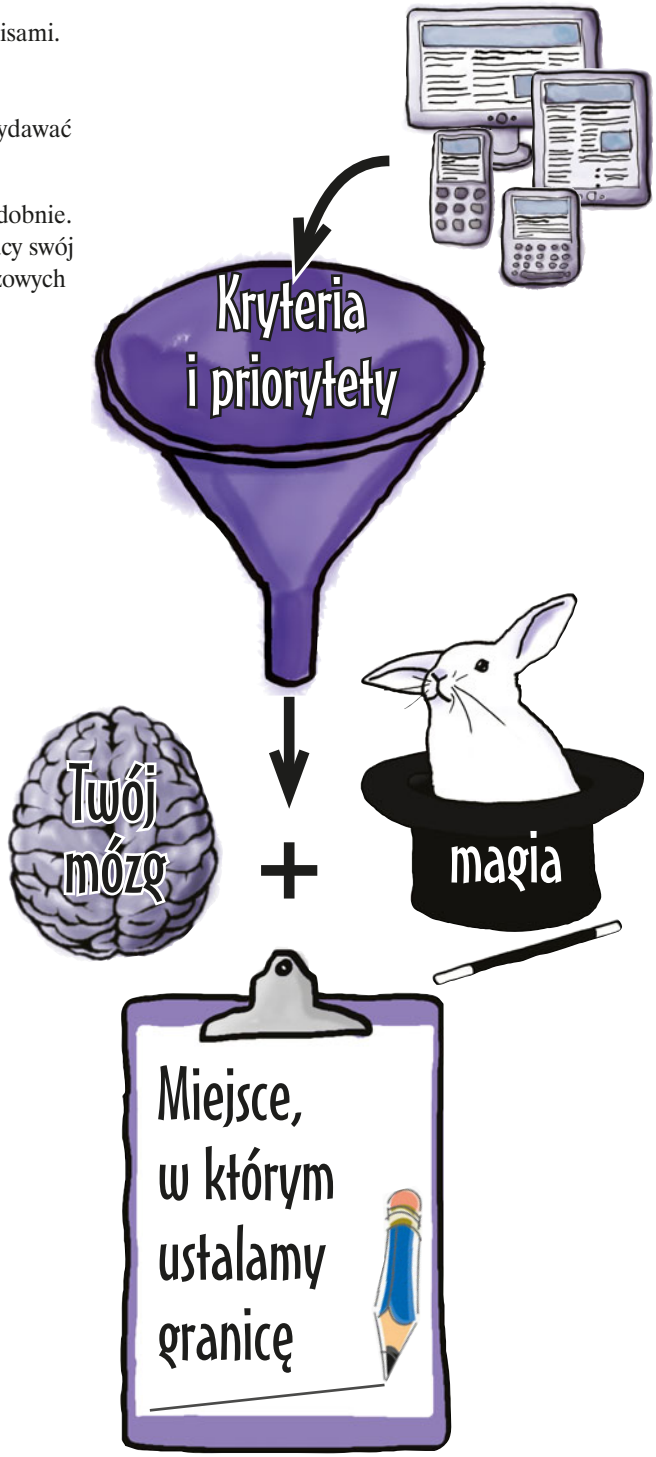


Przetestowanie witryny na wszystkich urządzeniach trwałoby wieczność. Musisz ustalić granicę wyodrębniającą urządzenia, które zamierzasz wspierać. **Ale jak podjąć tę decyzję?** Co z użytkownikami korzystającymi z telefonów, które już dawno powinny trafić na złomowisko? W jaki sposób stworzyć witrynę, by działała na urządzeniach, o których w ogóle nie słyszałeś? W tym rozdziale przygotujemy magiczną miksturę złożoną z **wymagań projektowych** i **informacji o odbiorcach**. Pomoże nam ona zdecydować, **które urządzenia mamy wspierać** i **co zrobić z tymi, których nie wspieramy**.

Skąd wiedzieć, gdzie ustalić granicę?

Realizacja każdego projektu wiąże się z wieloma kompromisami. Oczekiwania współpracowników. Informacje zwrotne od użytkowników. Optymalizacja pod kątem wyszukiwarek. Określenie najistotniejszych aspektów projektu może się wydawać praktycznie niewykonalne.

Z decyzją dotyczącą wsparcia dla różnych urządzeń jest podobnie. Bierzesz pod uwagę kryteria i priorytety, zaprzęgasz do pracy swój mózg, dodajesz szczyptę magii i już! — dostajesz listę kluczowych urządzeń.



Odejdź na chwilę od komputera

Do tej pory staraliśmy się mieć wszystko pod kontrolą. Jednak w każdym filmie dochodzi do momentu, w którym aktor zapatruje się w nicość i kontempluje zastany problem. Nagle, jak grom z jasnego nieba, spada na niego rozwiązanie, więc rzuca się w wir akcji.

Tutaj to Ty jesteś aktorem. Masz już wszystko, czego potrzebujesz, by wyznaczyć granicę.

Potrzebujesz jeszcze inspiracji, a jej nie znajdziesz przy klawiaturze. W związku z tym zrobimy krótką przerwę, podczas której pomówimy o tym, jak założone kryteria i priorytety przekuć na konkretną listę urządzeń, które należy obsłużyć. Pomożemy Ci się skupić na najistotniejszych sprawach, byś wyszedł z tego zadania obronną ręką.

Nie martw się, już niedługo będziesz mógł znowu zasiąść przy klawiaturze. Najlepsi reżyserzy wiedzą doskonale, że wystarczy dać aktorom wskazówki, a później pozwolić im grać.

Co oddziela granica, którą mamy wyznaczyć?

Chęć dostarczenia wsparcia wszystkim istniejącym urządzeniom jest doprawdy godna podziwu, ale testowanie na nich wszystkich stworzonej witryny jest pewną drogą do szaleństwa. Możesz starać się zapewnić dostęp jak największej liczbie użytkowników, ale — by nie popaść w obłęd — musisz najpierw odpowiedzieć na trzy proste pytania.

1. **Które urządzenia wspierasz?**
   Na których urządzeniach lub typach urządzeń zamierzasz przeprowadzić testy, by sprawdzić, czy witryna działa zgodnie z oczekiwaniami?

2. **Co się stanie na urządzeniach, których NIE wspierasz?**
   Co możesz zrobić, by na urządzeniach, których nie przetestowałeś, witryna działała prawidłowo?

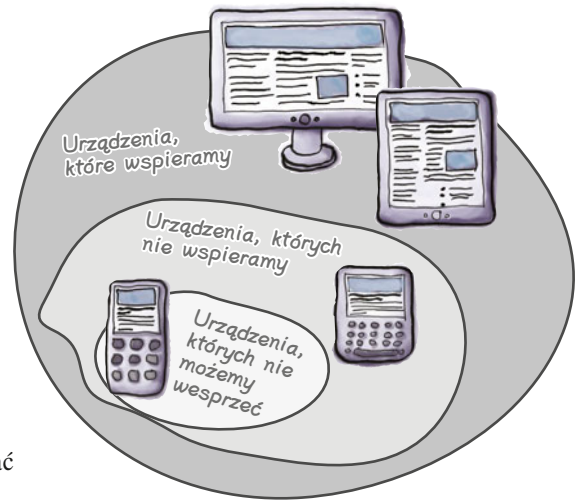
3. **Co się stanie na urządzeniach, których NIE MOŻESZ wspierać?**
   Jaki komunikat zobaczą użytkownicy urządzeń, na których witryna nie będzie działać?

WYSIL SZARE KOMÓRKI

Czy rozumiesz różnicę między urządzeniami, których *nie* wspierasz, a tymi, których *nie możesz* wspierać?

Urządzenia, których nie wspierasz, kontra te, których nie możesz wspierać

Skoro już wybrałeś urządzenia, które *zamierzasz* wspierać, wszystkie pozostałe nie będą wspierane. Po co więc tworzyć podział na te, których *nie* wspierasz, i na te, których *nie możesz* wspierać?



Mimo że nie dołączysz jakiegoś urządzenia do listy obsługiwanych, nie znaczy to wcale, że Twoja witryna *nie będzie* na nim działać. Jeśli stworzysz dobrze sformatowany, semantyczny kod HTML i zastosujesz metodę stopniowego ulepszania, strona będzie dostępna dla wielu przeglądarek i urządzeń, których nie uwzględniłeś na swojej liście.

Czasami jednak przeglądarka jest na tyle stara i słaba, że *nie możesz* jej wesprzeć.

Jeśli na przykład tworzysz witrynę internetowego sklepu obuwniczego, będziesz musiał skorzystać z protokołu HTTPS, by bezpiecznie obsługiwać transakcje. Niektóre starsze telefony nie wspierają protokołu HTTPS. I co wtedy? Najlepiej poinformować użytkownika o tym, że za pomocą jego telefonu nie da się zrealizować zakupów.

Nie bądź niegrzeczny

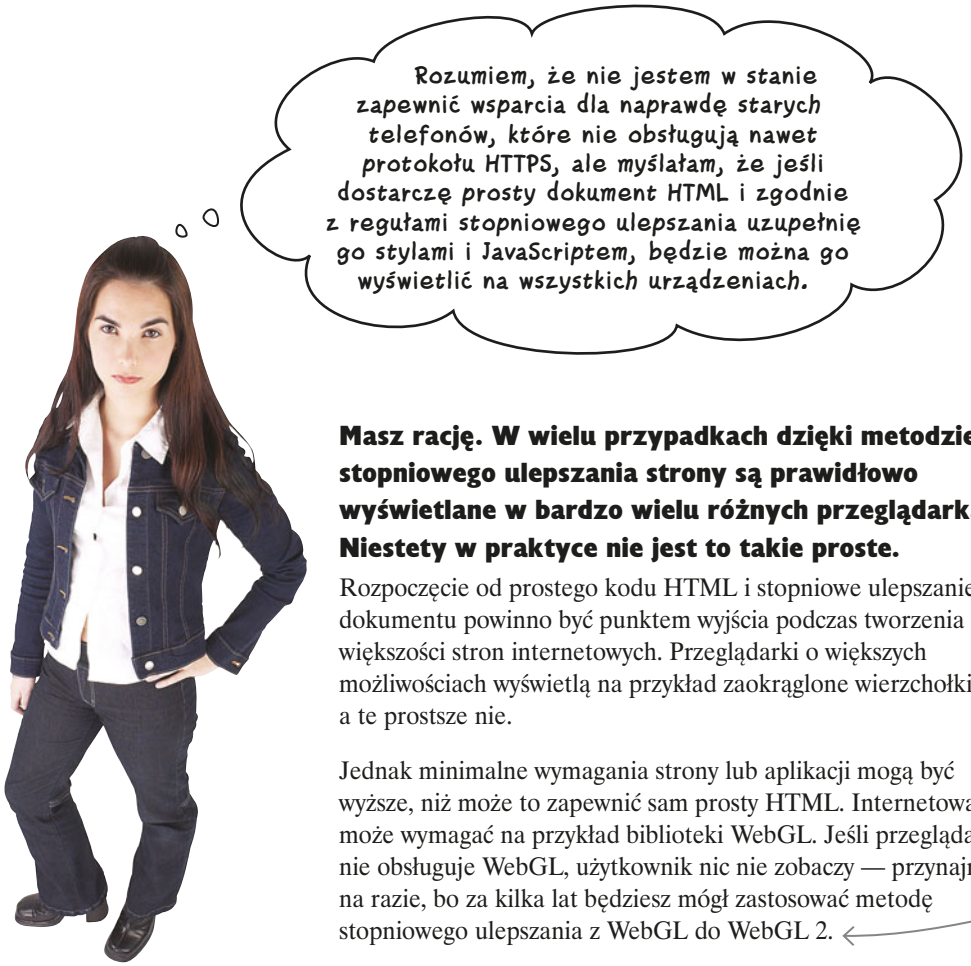
Jeśli czyjaś przeglądarka nie spełnia nałożonych przez Ciebie wymagań, bądź miły. Chociaż użytkownik nie ma w kieszeni najnowszego telefonu, może mieć dostęp do lepszej przeglądarki na innym urządzeniu.

Poza tym nikogo nie ucieszy stwierdzenie, że jego telefon to kupa złomu. Bądź uprzejmy dla swoich gości.

Obejrzyj to!

Nie wykluczaj niewspieranych przeglądarek.

Tylko dlatego, że nie jesteś w stanie sprawdzić działania swojej witryny w danej przeglądarce, nie możesz wykluczać jej użytkowników. Jeśli tworzysz poprawny i semantyczny kod, Twoje strony będą poprawnie wyświetlane w wielu przeglądarkach, włączając w to te, których nie jesteś w stanie przetestować, oraz te, które nie są tak istotne z perspektywy Twojego projektu.



Masz rację. W wielu przypadkach dzięki metodzie stopniowego ulepszania strony są prawidłowo wyświetlane w bardzo wielu różnych przeglądarkach. Niestety w praktyce nie jest to takie proste.

Rozpoczęcie od prostego kodu HTML i stopniowe ulepszanie dokumentu powinno być punktem wyjścia podczas tworzenia większości stron internetowych. Przeglądarki o większych możliwościach wyświetlą na przykład zaokrąglone wierzchołki, a te prostsze nie.

Jednak minimalne wymagania strony lub aplikacji mogą być wyższe, niż może to zapewnić sam prosty HTML. Internetowa gra może wymagać na przykład biblioteki WebGL. Jeśli przeglądarka nie obsługuje WebGL, użytkownik nic nie zobaczy — przynajmniej na razie, bo za kilka lat będziesz mógł zastosować metodę stopniowego ulepszania z WebGL do WebGL 2.

WebGL 2 na razie nie istnieje, ale jest raczej pewne, że za kilka lat się pojawi.

Tak jak już powiedzieliśmy — trzeba znać zasady, zanim zacznie się je łamać. Powinieneś zawsze postępować zgodnie z regułami stopniowego ulepszania, rozpoczynając od najprostszego kodu HTML, chyba że znajdziesz dobry powód, by tego nie robić.

Ale nawet w takiej sytuacji powinieneś zadbać o to, by w razie konieczności pojawił się komunikat: „Przykro nam, ale ta gra wymaga biblioteki WebGL”.

Zadawaj dużo pytań o swój projekt

Być może uznasz to za oczywistą oczywistość, ale pierwszym krokiem ustalania granicy wsparcia dla urządzeń powinno być przemyślenie projektu. ***Kto*** będzie z niego korzystał? ***Jaka*** funkcjonalność jest kluczowa? ***Które*** funkcje są opcjonalne? ***Jaki*** interfejs będzie najbardziej odpowiedni?

Nasza witryna — Zabawne koty w akcji — powinna się dobrze sprawdzić na urządzeniach mobilnych, ale pod warunkiem, że będą obsługiwały wideo. Które telefony obsługują wideo i oferują dużą przepustowość połączenia z internetem?

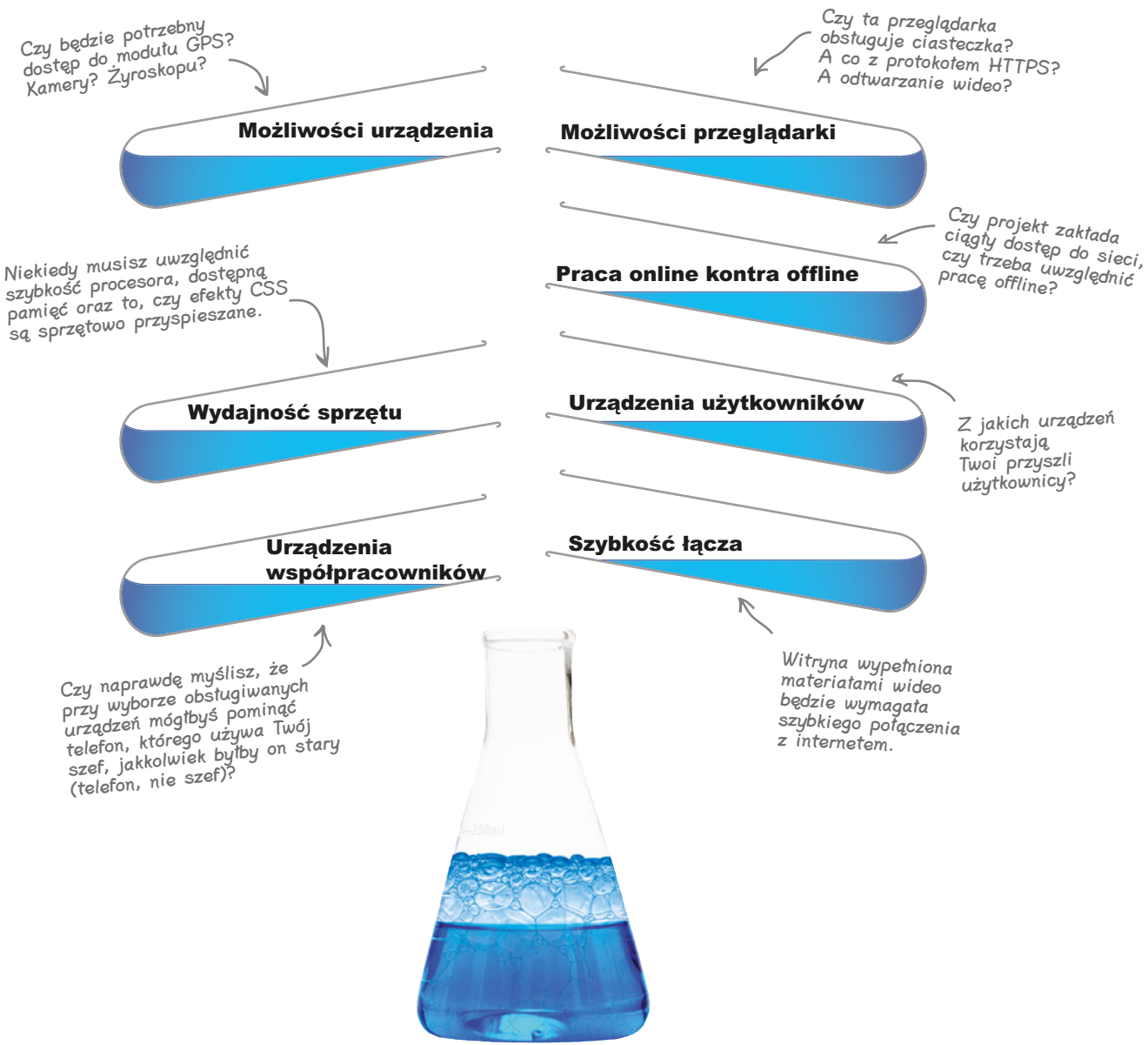
Jesteśmy gotowi na wkroczenie na rynek japoński i koreański. Czy oni używają tych samych telefonów co my? Słyszałam, że w Japonii jest wiele telefonów niedostępnych nigdzie indziej.

Nasza najnowsza gra — Ptasia furia — ma duże wymagania co do grafiki. Nie obejdzie się bez biblioteki WebGL, ale zastanawiamy się, czy na większości telefonów nie będzie to działało za wolno.

Składniki magicznej mobilnej mikstury

Kiedy już zadasz pytania o projekt, zaczną się pojawiać wskazówki pomocne w określeniu, które urządzenia powinieneś wziąć pod uwagę. Kim są przyszli użytkownicy i jakie mają przyzwyczajenia? Które funkcje muszą być, a które tylko mogą?

W tym momencie wkracza magia. Wymieszaj wszystkie składniki, tworząc magiczną miksturę, z której wyczytasz, jakie urządzenia musisz wziąć pod uwagę.



Zajrzyj do szafki z narzędziami i danymi

Chcesz poznać urządzenia użytkowników? Zajrzyj do słoja ze statystykami sieciowymi i sprawdź, z czego korzystają dotychczasowi użytkownicy. Wchodzisz na nowy rynek? Wyciągnij dane o najpopularniejszych urządzeniach mobilnych w tym kraju i sprawdź, jaki telefon najlepiej się sprzedaje na przykład w Singapurze.

Analiza może się opierać na statystykach internetowych lub danych pochodzących z innych form śledzenia zachowania użytkowników.

Jeżeli Twoja witryna nie jest często przeglądana z poziomu urządzeń mobilnych, do statystyk podchodź z dużą ostrożnością, bo otrzymane dane będą niemiarodajne.

Występują bardzo duże różnice w popularności telefonów w różnych częściach świata.

Nigdy nie zaszkodzi zapytać!

W odpowiedzi na pytanie, jakie możliwości mają poszczególne urządzenia, mogą pomóc takie bazy jak Device Anywhere Data Explorer czy Browserscope.

Dochody, wiek i (niekiedy) płeć to czynniki, które mogą wpływać na wybór określonego telefonu.

Twoja szafka na narzędzia świeci pustkami? Masz szczęście. Koniecznie rzuć okiem na wyczerpujący przewodnik po mobilnych statystykach: http://bit.ly/m-stats.

Ćwiczenie

Czas włożyć czarodziejski kapelusz i trochę poczarować. Dla każdego studium przypadku stwórz listę wymagań, opierając się na zakładanej funkcjonalności aplikacji i charakterystyce przyszłych użytkowników.

Lista powinna zawierać zarówno wymagania niezbędne, jak i opcjonalne.

Przykład

Studium przypadku

Galeria To-Sz-Tu-Ka przygotowuje aplikację umożliwiającą zwiedzającym wyświetlenie na ekranie telefonu dodatkowych informacji o eksponacie, na który kierują kamerę. Oglądany eksponat jest identyfikowany na podstawie lokalizacji telefonu. Zasięg w galerii może być nie najlepszy.

Wymagania:

- Niezbędne: obsługa JavaScriptu
- Niezbędne: dostęp do kamery
- Niezbędne: dostęp do GPS-u
- Opcjonalne: obsługa trybu offline

To jest lista wymagań określona na podstawie informacji zawartych w studium przypadku.

Studium przypadku

Pewna pani polityk chce wygrać najbliższe wybory. Potrzebuje aplikacji dla wolontariuszy, którzy chodzą od drzwi do drzwi i zachęcają do głosowania. Aplikacja musi dostarczać adresy ludzi, których należy odwiedzić (najlepiej wraz z mapą), oraz listę pytań, które wolontariusze mają zadać. Trzeba założyć, że wolontariusze to zwykle biedni studenci, więc aplikacja musi działać na bardzo wielu różnych, niekoniecznie najnowszych telefonach.

Wymagania:

Studium przypadku

Kręgle bokserów to gra, której pomysł został oparty na obrazach C.M. Coolidge'a z serii „Psy grające w pokera". Gracz steruje bokserem, przypominającym tego z obrazów, który bierze udział w zawodach kręglarskich. Rysowanie postaci i pozostałych elementów gry wyciska z większości telefonów ostatnie poty.

Wymagania:

Studium przypadku

Wielka Korporacja zanotowała wzrost sprzedaży w Indiach. Wiceprezes ds. Sprzedaży uważa, że Indie mogą być w przyszłości ogromnym rynkiem zbytu dla Wielkiej Korporacji. Chce, by powstała zlokalizowana wersja witryny firmy, która działałaby na większości telefonów używanych w Indiach. Poza tym chce, by do maksimum uprościć proces składania zamówień przez telefon.

Wymagania:

WYSIL SZARE KOMÓRKI

Które wymagania można zaliczyć do minimalnych, a które mogą być spełnione za pomocą metody stopniowego ulepszania?

Rozwiązanie ćwiczenia

Pewnie nie przepadasz za sytuacją, w której ktoś Ci mówi, że nie ma poprawnej odpowiedzi? No cóż, przykro nam, ale kiedy mówimy o wymaganiach, nie ma jednej poprawnej odpowiedzi.

Nie martw się więc, jeśli Twoje odpowiedzi różnią się od naszych. Twoje doświadczenie mogło Ci podpowiedzieć coś, co nam umknęło. Zaufaj doświadczeniu i instynktowi.

Przykład

Studium przypadku

Galeria To-Sz-Tu-Ka przygotowuje aplikację umożliwiającą zwiedzającym wyświetlenie na ekranie telefonu dodatkowych informacji o eksponacie, na który kierują kamerę. Oglądany eksponat jest identyfikowany na podstawie lokalizacji telefonu. Zasięg w galerii może być nie najlepszy.

Wymagania:

- Niezbędne: obsługa JavaScriptu
- Niezbędne: dostęp do kamery
- Niezbędne: dostęp do GPS-u
- Opcjonalne: obsługa trybu offline

Interfejs API umożliwiający dostęp do kamery pojawił się stosunkowo niedawno. Na razie tylko najnowsze wersje Androida częściowo go wspierają.

Studium przypadku

Pewna pani polityk chce wygrać najbliższe wybory. Potrzebuje aplikacji dla wolontariuszy, którzy chodzą od drzwi do drzwi i zachęcają do głosowania. Aplikacja musi dostarczać adresy ludzi, których należy odwiedzić (najlepiej wraz z mapą), oraz listę pytań, które wolontariusze mają zadać. Trzeba założyć, że wolontariusze to zwykle biedni studenci, więc aplikacja musi działać na bardzo wielu różnych, niekoniecznie najnowszych telefonach.

Wymagania:

- Niezbędne: połączenie internetowe
- Opcjonalne: możliwość wyświetlania obrazów (na potrzeby mapy)
- Opcjonalne: obsługa JavaScriptu na potrzeby interaktywnych map

Świetnie! Widać tu możliwości zastosowania stopniowego ulepszania!

Studium przypadku

Kręgle bokserów to gra, której pomysł został oparty na obrazach C.M. Coolidge'a z serii „Psy grające w pokera". Gracz steruje bokserem, przypominającym tego z obrazów, który bierze udział w zawodach kręglarskich. Rysowanie postaci i pozostałych elementów gry wyciska z większości telefonów ostatnie poty.

Wymagania:

- Niezbędne: obsługa JavaScriptu
- Niezbędne: wsparcie dla WebGL
- Niezbędne: szybki procesor (CPU i GPU); w przeciwnym przypadku gra nie będzie działać płynnie

Studium przypadku

Wielka Korporacja zanotowała wzrost sprzedaży w Indiach. Wiceprezes ds. Sprzedaży uważa, że Indie mogą być w przyszłości ogromnym rynkiem zbytu dla Wielkiej Korporacji. Chce, by powstała zlokalizowana wersja witryny firmy, która działałaby na większości telefonów używanych w Indiach. Poza tym chce, by do maksimum uprościć proces składania zamówień przez telefon.

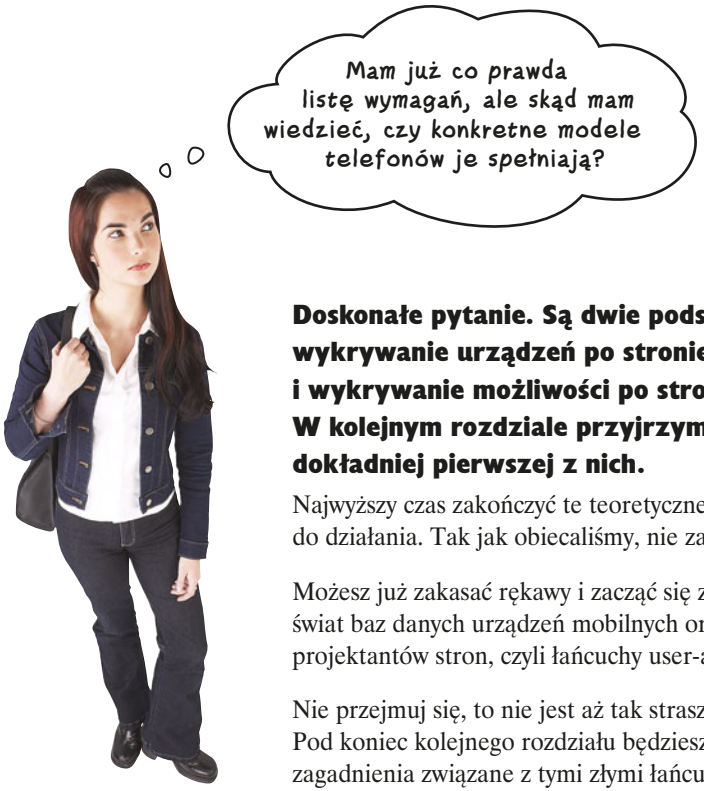
Wymagania:

- Niezbędne: wsparcie podstawowego kodu HTML dla starszych telefonów
- Niezbędne: obsługa protokołu HTTPS (składanie zamówień)
- Niezbędne: obsługa ciasteczek (logowanie)

Podejście iteracyjne pomaga odkryć dodatkowe wymagania.

Gdybyś zajrzał za kulisy projektów mobilnych witryn, zobaczyłbyś, że ich twórcy bardzo często nie są pewni, jakie urządzenia powinny być wspierane. Zaczynają od wytypowania najbardziej ich zdaniem odpowiednich, a następnie w iteracjach — w miarę prac nad projektem — doprecyzowują listę urządzeń.

Skąd mam wiedzieć, czy moi klienci używają odpowiednich urządzeń?



Doskonałe pytanie. Są dwie podstawowe metody: wykrywanie urządzeń po stronie serwera i wykrywanie możliwości po stronie klienta. W kolejnym rozdziale przyjrzymy się dokładniej pierwszej z nich.

Z wykrywaniem możliwości zetknęliśmy się w rozdziale 2., kiedy mówiliśmy o stopniowym ulepszaniu.

Najwyższy czas zakończyć te teoretyczne rozważania i przystąpić do działania. Tak jak obiecaliśmy, nie zajęło to dużo czasu.

Możesz już zakasać rękawy i zacząć się zagłębiać w szalony świat baz danych urządzeń mobilnych oraz zmorę wszystkich projektantów stron, czyli łańcuchy user-agent.

Nie przejmuj się, to nie jest aż tak straszne, jak się wydaje. Pod koniec kolejnego rozdziału będziesz miał w małym palcu zagadnienia związane z tymi złymi łańcuchami user-agent.

CELNE SPOSTRZEŻENIA

- W każdym projekcie trzeba **wyznaczyć granicę** określającą wspierane urządzenia.
- Granicę wyznacza się na podstawie **doświadczenia**, **analizy zebranych danych** oraz **instynktu**.
- **Jest różnica między urządzeniami, których nie wspierasz, a tymi, których nie możesz wspierać,** ponieważ brak kluczowych możliwości uniemożliwia działanie witryny na danym urządzeniu.
- W wyznaczaniu granicy może pomóc zestawienie **wymagań projektowych** z **charakterystyką przyszłych użytkowników**.
- **Domyślnym podejściem** do projektu powinno być **stopniowe ulepszanie**. Dzięki niemu tworzona witryna będzie działać na wielu urządzeniach, nawet takich, których nie wymienisz na oficjalnej liście wsparcia.
- **Nie wykluczaj danego urządzenia, dopóki z całą pewnością nie stwierdzisz, że nie możesz go obsługiwać.** Cały czas pojawiają się nowe przeglądarki — daj szansę się im wykazać.
- Nie pozwól, by złożoność problematyki technologii mobilnych Cię przytłoczyła. Masz już wszystkie narzędzia niezbędne do prawidłowego wyznaczania granicy wsparcia. **Zaufaj doświadczeniu i instynktowi**.

Zapoznaj się z grupą

Podczas wybierania wspieranych urządzeń nie wzięliśmy pod uwagę kilku dokuczliwych problemów. Jak mamy się dowiedzieć wystarczająco dużo o mobilnych przeglądarkach użytkowników, by przed dostarczeniem treści spełnić ich oczekiwania? Jak uniknąć tworzenia tylko podstawowej zawartości odpowiadającej najmniejszemu wspólnemu mianownikowi urządzeń? No i jak, pozostając przy zdrowych zmysłach, zapanować nad tym wszystkim? W tym rozdziale wkroczymy w świat **możliwości urządzeń** i nauczymy się korzystać z **baz danych urządzeń**, by wreszcie odkryć, jak te wszystkie urządzenia grupować w **klasy**.

Przycisk awaryjny dla spanikowanych studentów

DaRadę! Przygotowania do testów to szczególny rodzaj usługi, której celem jest dostarczenie wszystkim studentom, a zwłaszcza tym, którzy — czy to ze względu na przyszłą karierę, czy dla idei — zarywają noce, przygotowując się do egzaminów, możliwości bezpośredniego kontaktu z nauczycielami specjalizującymi się w wybranych dziedzinach.

Czasami studenci panikują. Właśnie z myślą o nich firma DaRadę! chce na swojej witrynie umieścić specjalną stronę — Spanikowałem!. Jej cel jest prosty — bezpośrednio połączyć użytkowników z nauczycielami dostępnymi pod telefonem.

Przycisk jest tylko dla urządzeń mobilnych

Desktopowe przeglądarki nie mogą wykonać połączenia telefonicznego, więc wielki czerwony przycisk awaryjny nie ma w nich za bardzo sensu. Stefan chce, by przycisk pojawiał się tylko na telefonach komórkowych.

Mamy w firmie informatyka, który zajmuje się naszą witryną, ale w tym przypadku będziemy chyba potrzebować pomocy.

Ale skąd mamy wiedzieć, że użytkownik korzysta z telefonu komórkowego?

Łukasz: Przecież możemy to załatwić w arkuszu stylów za pomocą zapytań o media bazujących na szerokości okna przeglądarki.

Iza: Nie sądzę, żeby tym razem to było dobre rozwiązanie. To, że okno jest wąskie, nie znaczy, że mamy do czynienia z mobilną przeglądarką.

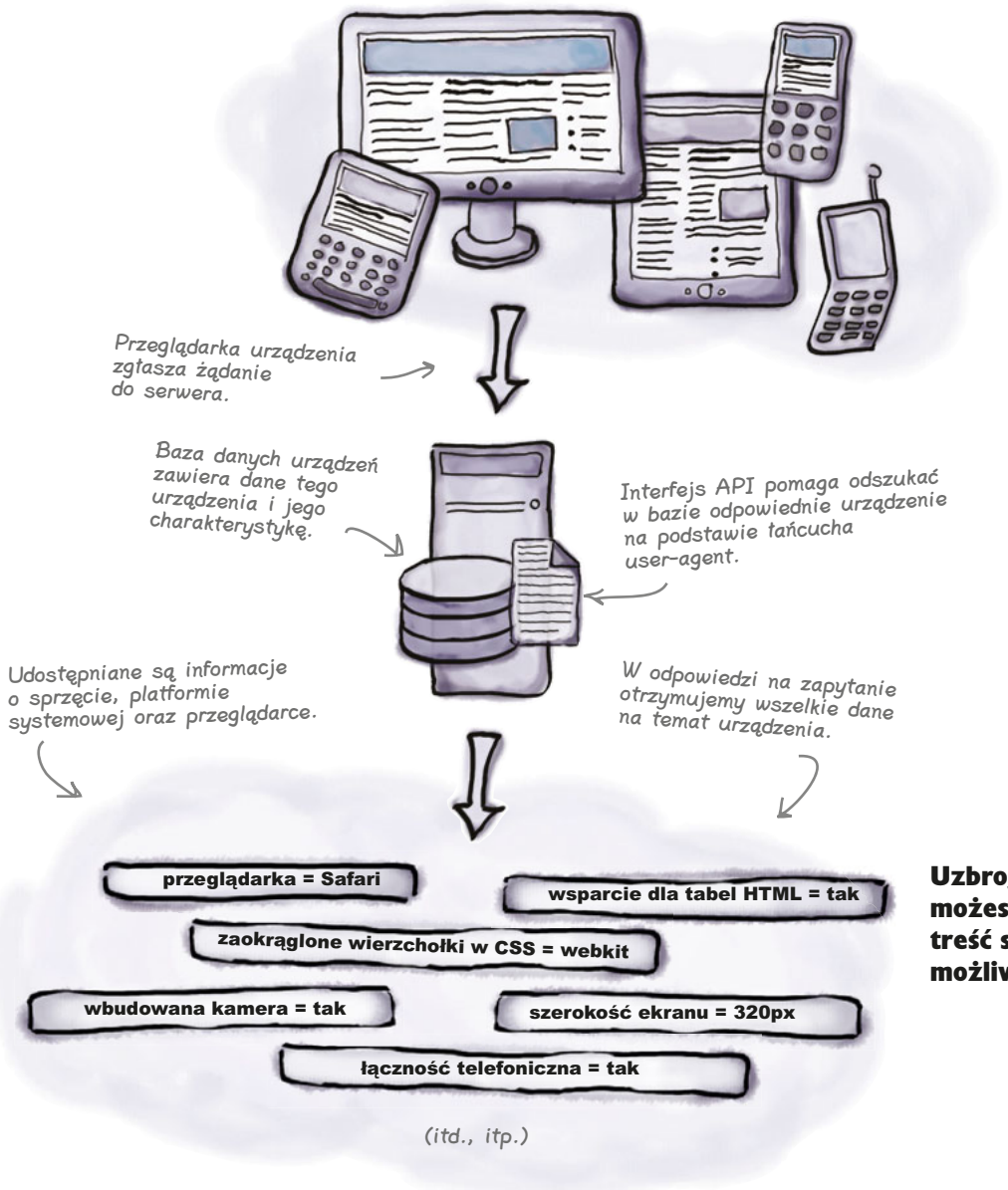
Łukasz: Prawda. Gdybyśmy oparli to o zapytania o media, przycisk pojawiałby się w wąskich oknach przeglądarek desktopowych, na tabletach i innych urządzeniach, które wcale nie są telefonami. Co nam w takim razie pozostaje?

Nie tylko telefony mają wąskie ekrany!

Iza: Idealnie by było, gdybyśmy mieli możliwość zdobycia szczegółowych informacji o przeglądarce i urządzeniu, tak by jednoznacznie można było stwierdzić, że dane urządzenie to telefon komórkowy, a nie jakikolwiek sprzęt z małym ekranem.

Źródła danych o urządzeniach mobilnych spieszą na ratunek

Bazy danych urządzeń mobilnych zawierają szczegółowe informacje o przeglądarkach, platformach i możliwościach sprzętowych oferowanych przez wiele różnych urządzeń mobilnych. Wystarczy przesłać do nich zapytanie z unikalnym identyfikatorem (zwykle łańcuchem user-agent), by dostać w odpowiedzi zbiór cech opisujących urządzenie i zainstalowaną w nim przeglądarkę.



Uzbrojony w te dane możesz dostosować treść strony do możliwości urządzenia.

Poznaj WURFL

WURFL (ang. *Wireless Universal Resorce FiLe*) to baza danych dostępna na zasadach open source, udostępniająca dane o możliwościach niezliczonej liczby urządzeń i przeglądarek mobilnych.

Ostatnio główni twórcy WURFL założyli firmę ScientiaMobile, która oferuje komercyjne wsparcie dla tej bazy.

W zasobach WURFL można znaleźć szczegółowe informacje o urządzeniach mobilnych, zaczynając od formatu dźwięku dzwonka, a na fizycznych wymiarach ekranu kończąc.

Nie wszystkie dostarczane właściwości urządzeń są nam — twórcom stron — potrzebne, ale znajdzie się tam kilka naprawdę przydatnych.

Urządzenie to nie platforma ani przeglądarka.

Nie ma jednego dobrego określenia na „charakterystyka sprzętowa urządzenia mobilnego, jego platformy i zintegrowanej z nią przeglądarki". A przydałoby się. Baza WURFL łączy te trzy sprawy i zawiera informacje o sprzęcie, platformie oraz przeglądarce.

Stosowana terminologia czasem sprawia problemy — my też jesteśmy w pewnym stopniu winni. Pamiętaj więc, że kiedy używamy słowa „urządzenie", mamy na myśli najczęściej zarówno sprzęt, jak i platformę oraz przeglądarkę.

Ćwiczenie

Sprawdź sam i przekonaj się, czym jest WURFL. Firma ScientiaMobile udostępniła eksplorator danych WURFL, który działa w przeglądarce.

1. Przejdź na stronę *http://www.tera-wurfl.com/explore* w przeglądarce mobilnej.

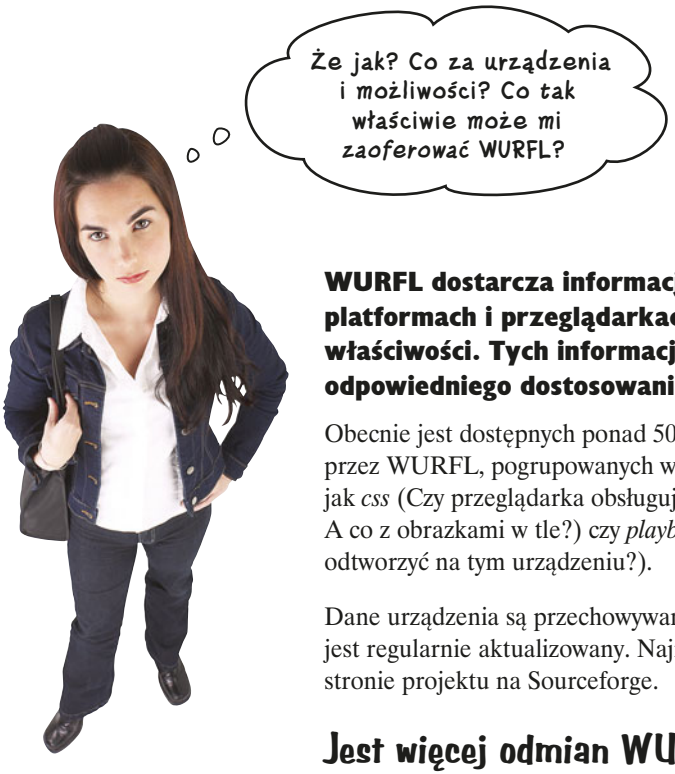
2. Przejdź do sekcji ***See my capabilities*** i przejrzyj możliwości Twojego urządzenia i przeglądarki.

Eksplorator jest zaprojektowany z myślą o przeglądarkach mobilnych. Jeżeli uruchomisz go w przeglądarce desktopowej, zobaczysz właściwości losowo wybranego urządzenia mobilnego.

Możesz też uruchomić eksplorator w przeglądarce desktopowej i podać dowolny łańcuch user-agent, by poznać możliwości wybranej przeglądarki. To jest szybki i wygodny sposób na sprawdzenie możliwości urządzenia i jego przeglądarki, pod warunkiem że znasz odpowiednie łańcuchy UA.

Niektóre z właściwości wyświetlonych w eksploratorze na iPhonie 4.

WURFL i jego możliwości



WURFL dostarcza informacje o urządzeniach, platformach i przeglądarkach w postaci zbioru właściwości. Tych informacji możesz użyć do odpowiedniego dostosowania zawartości stron.

Obecnie jest dostępnych ponad 500 możliwości uwzględnianych przez WURFL, pogrupowanych w kilkadziesiąt kategorii, takich jak *css* (Czy przeglądarka obsługuje zaokrąglone wierzchołki? A co z obrazkami w tle?) czy *playback* (Jakiego typu media mogę odtworzyć na tym urządzeniu?).

Dane urządzenia są przechowywane w ogromnym pliku XML, który jest regularnie aktualizowany. Najnowsza wersja jest dostępna na stronie projektu na Sourceforge.

Ku zaskoczeniu wszystkich znajduje się pod adresem wurfl.sourceforge.net.

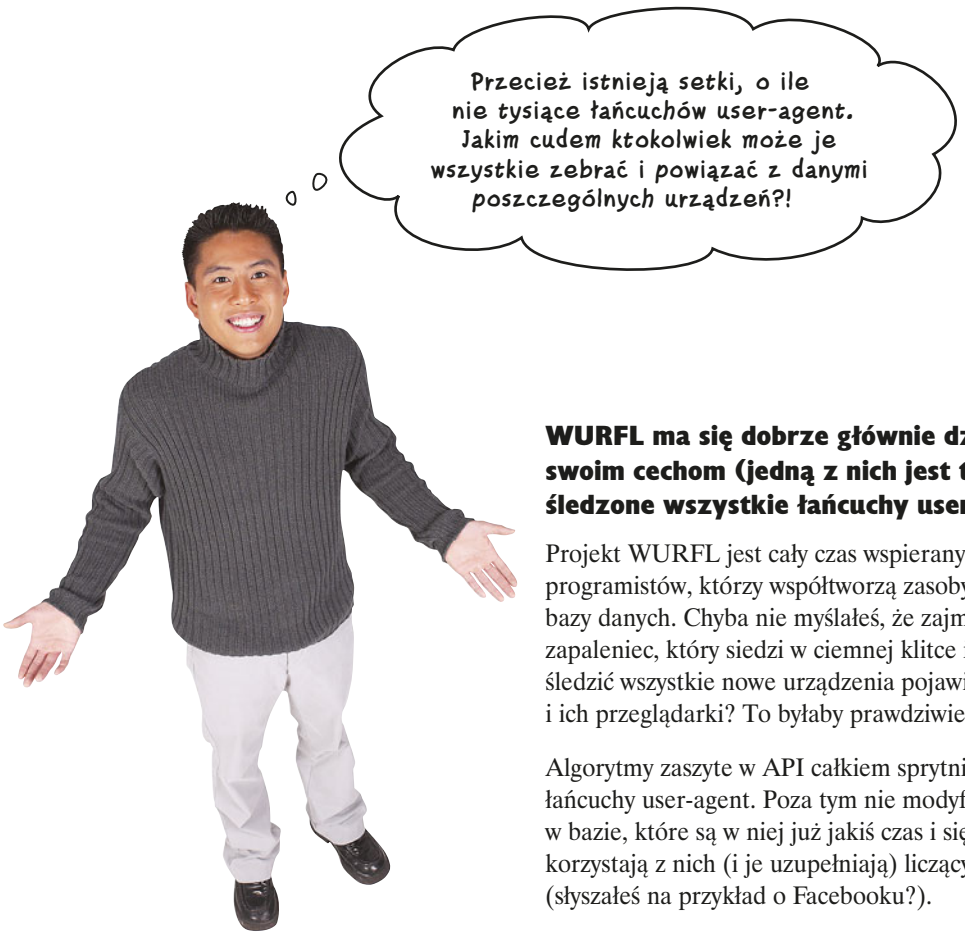
Jest więcej odmian WURFL

Eksplorator firmy ScientiaMobile bazuje na jednej z **implementacji** WURFL (odmianie *Database Edition*, znanej też jako *Tera-WURFL*), w której dane o urządzeniach są przechowywane w bazie danych, a nie w pliku XML. Z założenia Tera-WURFL może być usługą internetową lub biblioteką PHP, z której mobilne witryny internetowe mogą pobierać dane o urządzeniach na podstawie łańcuchów user-agent.

Jest dostępnych wiele interfejsów API i implementacji WURFL, ale łączy je jedno — ogromny plik XML zawierający wszystkie dane urządzenia mobilnego. Aby zatem w pełni skorzystać z możliwości WURFL, potrzebne są plik XML, wiedza o sposobie przechowywania w nim danych oraz interfejs API umożliwiający dostęp do nich.

My skorzystamy z API dla PHP

Dostępne są API dla wielu popularnych języków programowania. My skorzystamy z API dla PHP oraz danych zapisanych w pliku XML.



WURFL ma się dobrze głównie dzięki niektórym swoim cechom (jedną z nich jest to, że nie są śledzone wszystkie łańcuchy user-agent).

Projekt WURFL jest cały czas wspierany przez społeczność programistów, którzy współtworzą zasoby tej ogromnej bazy danych. Chyba nie myślałeś, że zajmuje się tym jeden zapaleniec, który siedzi w ciemnej klitce i stara się na bieżąco śledzić wszystkie nowe urządzenia pojawiające się na rynku i ich przeglądarki? To byłaby prawdziwie syzyfowa praca.

Algorytmy zaszyte w API całkiem sprytnie dopasowują łańcuchy user-agent. Poza tym nie modyfikują tych wpisów w bazie, które są w niej już jakiś czas i się sprawdziły lub korzystają z nich (i je uzupełniają) liczący się gracze na rynku (słyszałeś na przykład o Facebooku?).

Jeśli jednak WURFL z jakiegoś względu nie spełnia Twoich oczekiwań, jest wiele innych baz danych urządzeń, takich jak DeviceAtlas, DetectRight czy MobileAware. Która z nich sprawdzi się w Twoim projekcie, zależy od potrzeb, budżetu i wymagań dotyczących licencjonowania.

Nie istnieją głupie pytania

P: Czym jest ScientiaMobile?

O: ScientiaMobile to firma, którą w 2011 roku założyli Steve Kamerman, Luca Passani i Krishna Guda. Luca był jednym z założycieli projektu (drugim był Andrea Trasatti) oraz programistą utrzymującym open source'owy projekt WURFL od 2001 roku. Steve utworzył odmianę Tera-WURFL, która przekształciła się w WURFL Database Edition.

P: Czy korzystanie z WURFL jest bezpłatne?

O: To zależy. Technicznie rzecz biorąc, WURFL jest typu open source. API jest dostępne w ramach licencji Affero General Public License v3 (AGPL), czyli licencji open source. W licencji na bazę WURFL XML jest zastrzeżenie, że może być używana jedynie z udostępnianym API.

Zatem jeśli spełniasz wymogi licencji AGPL, nie musisz płacić. Jednak licencja AGPL jest bardziej restrykcyjna od swojej kuzynki GPL — uruchomienie oprogramowania na serwerze jest traktowane jako dystrybucja, co wymaga otwarcia źródeł wszystkich projektów pochodnych.

Jeśli licencja AGPL jest w Twoim przypadku nieodpowiednia (a jest to bardzo prawdopodobne), firma ScientiaMobile oferuje licencje komercyjne.

P: Jeśli zintegruję WURFL z (tu wstaw swój ulubiony framework typu open source) i w oparciu o nie zbuduję swoją witrynę, to czy muszę otworzyć źródła całej witryny?

O: Prosta sprawa: kup licencję komercyjną lub skontaktuj się z prawnikiem. Możesz też przejrzeć FAQ na stronie *ScientialMobile.com*.

P: Czy dostępne są inne rozwiązania oferujące to co WURFL?

O: Tak. Co prawda WURFL jest jedyną bazą danych urządzeń z możliwością dostępu na zasadach open source, ale jest jeszcze kilka komercyjnych baz. Najczęściej wymienianą alternatywą jest DeviceAtlas. Inne to MobileAware i DetectRight.

P: Jakie inne interfejsy API oferuje WURFL?

O: Poza PHP są jeszcze API dla Javy, .NET i baz danych.

P: W jaki sposób są aktualizowane dane WURFL?

O: Ludzie z firmy ScientiaMobile cały czas obserwują rynek urządzeń mobilnych. Czasem informacje dostarczają sami producenci, a czasem użytkownicy korzystający z WURFL. Większość pracy jest związana z weryfikacją dostarczonych informacji.

P: Kto decyduje o tym, jakie możliwości uwzględnić w WURFL?

O: Firma ScientiaMobile wybiera możliwości na podstawie sugestii społeczności skupionej wokół projektu.

P: A gdzie znajdę tę społeczność, o której wspominacie?

O: Główna aktywność społeczności przejawia się na liście dyskusyjnej. Projekt WURFL jest rozwijany już dosyć długo i ma bardzo aktywną listę — *wml-programming*. Również od czasu do czasu są organizowane dyskusje na forum witryny *ScientiaMobile.com*.

Listę wml-programming możesz znaleźć pod adresem http://tech.groups.yahoo.com/group/wmlprogramming/.

WURFL — sprytny interfejs API

Gdy API WURFL stara się dopasować otrzymany łańcuch user-agent do jakiegoś znanego urządzenia, nie ogranicza się do zwykłego sprawdzenia, czy ten konkretny łańcuch znajduje się w bazie. Ponieważ zebranie w bazie wszystkich istniejących łańcuchów UA jest praktycznie niemożliwe, algorytmy dopasowujące zaimplementowane w tym API muszą stosować sprytne sztuczki.

Drzewo urządzeń i ich rodzin

Podczas analizowania łańcucha user-agent dochodzi do wielu stale poszerzanych uogólnień, których celem jest zaklasyfikowanie danego urządzenia do właściwej rodziny podobnych urządzeń.

Dane WURFL możesz sobie wyobrażać jako drzewo urządzeń, którego pień to uogólniona przeglądarka, a konary, gałęzie i liście to coraz bardziej uszczegółowione urządzenia lub ich grupy. Algorytm dopasowujący stara się dojść do konkretnego liścia, ale jeżeli to się nie uda, ma możliwość odwrotu do poprzednich elementów drzewa.

Dane WURFL dla konkretnego urządzenia (lub jego grupy) zawierają tylko te możliwości, które są różne od możliwości nadrzędnego urządzenia. Dzięki temu plik XML z danymi WURFL jest stosunkowo niewielki (nieco ponad 18 MB w połowie 2012 roku) w porównaniu z ogromem informacji, które zawiera.

Dla maniaków

W API WURFL jest stosowanych wiele metod zabezpieczających przed błędami dopasowania w przypadku nietypowych elementów znajdujących się w łańcuchach user-agent.

Jedną z metod jest stosowanie **algorytmów dopasowujących** zoptymalizowanych pod kątem różnych rodzin przeglądarek i urządzeń. Najnowsze API dla PHP zawiera dziesiątki algorytmów analizujących konkretne łańcuchy UA.

Pierwszym etapem jest wybranie algorytmu, który jest najbardziej odpowiedni dla danego łańcucha user-agent. Na przykład łańcuch zawierający fragment „BlackBerry” zostanie skierowany do algorytmu BlackBerryHandler.

Każdy z algorytmów jest wyczulony na drobne różnice w łańcuchach występujących w danej rodzinie, dzięki czemu udało się zmniejszyć liczbę precyzyjnie określonych łańcuchów w bazie WURFL.

W API WURFL dla PHP jest stosowane dopasowywanie **RIS** (ang. *Reduction In String*), które polega na usuwaniu nierozpoznawalnych fragmentów łańcucha do momentu trafienia na znany.

Czas zainstalować WURFL na komputerze

Szczegóły dotyczące instalacji znajdziesz w dodatku C.

My też możemy zbudować eksplorator

Kiedy już zainstalujesz na swoim komputerze API WURFL dla PHP, wystarczy odrobina kodu, by stworzyć stronę niewiele się różniącą od eksploratora firmy ScientiaMobile. Tak naprawdę to zrobimy to właśnie teraz.

Dzięki samodzielnemu zbudowaniu eksploratora poznasz API WURFL i skorzystasz z danych z własnego pliku.

Chodzi o plik XML, który pobrałeś podczas instalowania WURFL.

Nauka przez praktykę — oto najlepsze rozwiązanie. Zbudujemy fundamenty, na których będziesz mógł oprzeć inne, bardziej funkcjonalne projekty.

Eksplorator firmy ScientiaMobile jest oparty na edycji Database Edition WURFL. Sposób przechowywania danych oraz dopasowywania łańcuchów UA trochę się różni od tego, które zamierzamy zastosować, czyli plików XML i API dla PHP. Najistotniejsze jest to, że **różnią się również dane** (czasem dość mocno).

Dzięki temu, że zbudujesz własny eksplorator, będziesz pewien, że pobierasz dane z własnego pliku XML, a nie z jakiejś starszej lub nowszej jego wersji.

Etapy tworzenia eksploratora

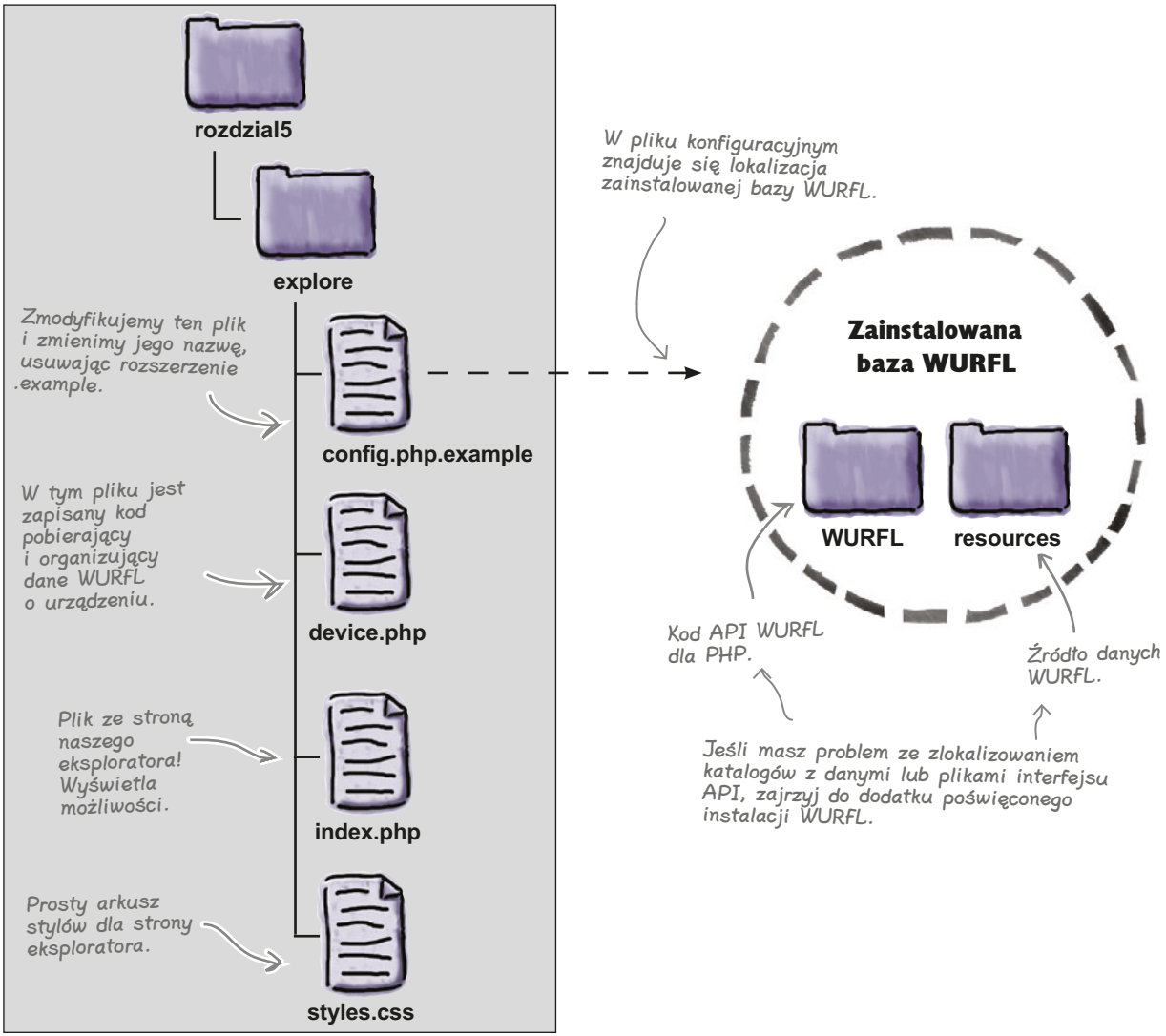
- ☐ Przygotujemy środowisko robocze i pliki, a także skonfigurujemy WURFL.
- ☐ Napiszemy (ramowy) kod PHP inicjalizujący niektóre obiekty WURFL, tak aby można było uzyskać dostęp do informacji o możliwościach bieżącej przeglądarki i urządzenia.
- ☐ Opracujemy strukturę informacji o możliwościach, stworzymy stronę i wyświetlimy te dane w tabeli.

Eksplorator — przygotowanie środowiska

Pierwszym, czym się musimy zająć, jest przygotowanie **struktury katalogów** i utworzenie **pliku konfiguracyjnego**. Niech punktem wyjścia będzie zawartość podkatalogu *explore* katalogu *rozdzial5*.

Powinieneś mieć już zainstalowany i uruchomiony interfejs API WURFL dla PHP. Aby utworzyć plik konfiguracyjny (to znaczy taki, który *działa*), musisz znać miejsce, w którym zainstalowałeś API.

Struktura katalogów strony eksploratora



Długie ćwiczenie

WURFL jest już zainstalowany (przynajmniej mamy taką nadzieję), ale trzeba go odpowiednio skonfigurować, aby działał z naszą stroną. W tym celu trzeba stworzyć **plik konfiguracyjny**.

W edytorze otwórz plik *config.php.example*. Zmodyfikuj wskazane ścieżki i zapisz plik w katalogu *explore* pod nazwą ***config.php*** (koniecznie usuń rozszerzenie *.example*!).

```
/* WURFL_DIR musi wskazywać lokalizację instalacji WURFL. */
define("WURFL_DIR", '/ścieżka/do/WURFL/');
/* RESOURCES_DIR musi wskazywać katalog resources, którego zamierzasz użyć. */
define("RESOURCES_DIR", '/ścieżka/do/WURFL/resources/');
```

Koniecznie to zmień! Obie ścieżki muszą być poprawne i wskazywać katalog z plikami interfejsu API WURFL oraz katalog zasobów.

Napisz raz, używaj zawsze. W pozostałych przykładach z tego rozdziału będziemy korzystać z tego samego pliku konfiguracyjnego.

config.php.example

Nie zapomnij usunąć rozszerzenia .example.

- [x] Przygotujemy środowisko robocze i pliki, a także skonfigurujemy WURFL.
- [ ] Napiszemy (ramowy) kod PHP inicjalizujący niektóre obiekty WURFL, tak aby można było uzyskać dostęp do informacji o możliwościach bieżącej przeglądarki i urządzenia.
- [ ] Opracujemy strukturę informacji o możliwościach, stworzymy stronę i wyświetlimy te dane w tabeli.

Jeśli masz problem z określeniem ścieżek instalacyjnych WURFL, zajrzyj do dodatku C.

Teraz możemy napisać kod inicjalizujący obiektu WURFL, by zacząć odczytywać właściwości.

W porządku — mamy już plik konfiguracyjny. Teraz musimy napisać kod inicjalizujący elementy bazy WURFL, by mogła w końcu podać nam jakieś dane o możliwościach urządzenia. W edytorze otwórz plik *device.php*. Na razie nie wygląda zbyt okazale, więc trzeba coś z tym zrobić!

Skrypt *device.php* ma dwa podstawowe zadania: **zainicjalizować obiekty WURFL** za pomocą nagłówka `User-Agent` pochodzącego z żądania (zgłoszonego przez przeglądarkę użytkownika) oraz **przygotować dane z możliwościami urządzenia** w sposób, który nadawałby się do wyświetlenia na stronie.

Dołączamy plik konfiguracyjny.

```
<?php
require_once('config.php');
$user_agent = $_SERVER['HTTP_USER_AGENT'];

$wurflConfig = new WURFL_Configuration_XmlConfig($wurflConfigFile);
$wurflManagerFactory = new WURFL_WURFLManagerFactory($wurflConfig);
$wurflManager = $wurflManagerFactory->create();
$device = $wurflManager->getDeviceForUserAgent($user_agent);
```

device.php

W tych wierszach tworzymy obiekty WURFL i wypełniamy obiekt reprezentujący urządzenie.

Większość tego kodu jest uniwersalna, więc możemy go użyć w innych projektach korzystających z WURFL. Jeżeli PHP znasz nie od dziś i rozumiesz, co się dzieje w tym kodzie, szacun, a jeżeli nie — nie musisz się za bardzo przejmować.

Ciąg dalszy na kolejnej stronie.

Długie ćwiczenie

Przyjrzymy się teraz bliżej ostatniemu wierszowi kodu z poprzedniej strony. Wskazujemy tu sposób utworzenia przez WURFL obiektu urządzenia. W opisywanym przypadku korzystamy ze zmiennej `$user_agent`, która przechowuje łańcuch zawierający nagłówek `User-Agent` otrzymany z żądania.

W ten sposób informujemy WURFL, by skorzystał z łańcucha user-agent przeglądarki użytkownika i spróbował dopasować go do danych z pliku XML. Jeśli ta operacja się powiedzie, otrzymamy obiekt zawierający możliwości przeglądarki i urządzenia.

```
$device = $wurflManager->getDeviceForUserAgent($user_agent);
```

Od tego momentu, jeżeli wszystko się powiedzie, można odczytywać informacje o możliwościach urządzenia.

Metoda `getDeviceForUserAgent()` klasy `WURFL_WURFLManager` przyjmuje łańcuch user-agent i zwraca odpowiadające mu urządzenie (określone na podstawie danych WURFL) w postaci obiektu klasy `WURFL_CustomDevice`.

Tłumaczenie: podajemy łańcuch user-agent, a dostajemy obiekt reprezentujący urządzenie, wypełniony wartościami odpowiadającymi poszczególnym możliwościom.

- [x] Przygotujemy środowisko robocze i pliki, a także skonfigurujemy WURFL.
- [x] Napiszemy (ramowy) kod PHP inicjalizujący niektóre obiekty WURFL, tak aby można było uzyskać dostęp do informacji o możliwościach bieżącej przeglądarki i urządzenia.
- [ ] Opracujemy strukturę informacji o możliwościach, stworzymy stronę i wyświetlimy te dane w tabeli.

Skoro mamy już wypełniony obiekt reprezentujący urządzenie, przyszedł czas na przygotowanie informacji o możliwościach w takiej formie, by dało się je wyświetlić w postaci tabeli.

Drugim zadaniem skryptu *device.php* jest przygotowanie informacji o możliwościach na potrzeby ich wyświetlenia. Dopiszemy kod, który będzie kategoryzował możliwości względem grup określonych w WURFL.

Poniższy fragment umieść w pliku *device.php* i zapisz zmiany. Na razie zakończyliśmy pracę nad tym plikiem.

Metoda getListOfGroups() zwraca tablicę nazw grup WURFL...

...a metoda getCapabilitiesNameForGroup() zwraca nazwy możliwości umieszczonych w tej grupie.

```
$device = $wurflManager->getDeviceForUserAgent($user_agent);
if ($device) {
    $groups = $wurflManager->getListOfGroups();
    $grouped_capabilities = array();
    foreach($groups as $a_group) {
        $grouped_capabilities[$a_group] = array();
        $capabilities = $wurflManager->getCapabilitiesNameForGroup($a_group);
        foreach ($capabilities as $cap) {
            $grouped_capabilities[$a_group][$cap] = $device->getCapability($cap);
        }
    }
}
```

device.php

Ten fragment kodu przygotowuje możliwości urządzenia i przeglądarki, łącząc je w grupy.

W ten sposób odczytujemy wartości poszczególnych możliwości. Już wkrótce przyjrzymy się temu bliżej.

Nie przejmuj się, jeżeli nie masz pojęcia, co robi ten kod PHP. Wystarczy, że umieścisz go w pliku device.php.

Jeszcze nie skończyliśmy — ciąg dalszy na kolejnej stronie.

Długie ćwiczenie

Na razie skończyliśmy pracę nad plikiem *device.php*. Teraz zajmiemy się plikiem *index.php* i dodamy do niego trochę kodu.

- ☑ Przygotujemy środowisko robocze i pliki, a także skonfigurujemy WURFL.
- ☑ Napiszemy (ramowy) kod PHP inicjalizujący niektóre obiekty WURFL, tak aby można było uzyskać dostęp do informacji o możliwościach bieżącej przeglądarki i urządzenia.

To już mamy!

- ☐ Opracujemy strukturę informacji o możliwościach, stworzymy stronę i wyświetlimy te dane w tabeli.

Czas na to.

Kod PHP/HTML gotowy do użycia

Iterujemy po kolekcji danych opisujących możliwości, które są umieszczone w grupach.

Dla każdej grupy generujemy znaczniki <dl> z nazwami i wartościami możliwości umieszczonych w tej grupie.

Wyświetlamy tańcuch user-agent oraz identyfikator urządzenia (określony przez WURFL).

```
<div id="devicedata">
   <h2>Dane urządzenia</h2>
   <p>Dane urządzenia dla <?php print $user_agent; ?></p>
   <p><strong>Identyfikator WURFL urządzenia: <?php print $device->id; ?></strong></p>
   <?php foreach($grouped_capabilities as $group_name => $my_caps): ?>
      <h3 class="group"><?php print $group_name; ?></h3>
      <dl>
         <?php foreach($my_caps as $cap_name => $cap): ?>
            <dt><?php print $cap_name; ?></dt>
            <dd><?php print ($cap) ? $cap : '[no value]'; ?></dd>
         <?php endforeach; ?>
      </dl>
   <?php endforeach; ?>
</div>
```

Nazwę możliwości wyświetlamy w znaczniku <dt>, a przypisaną mu wartość w <dd>.

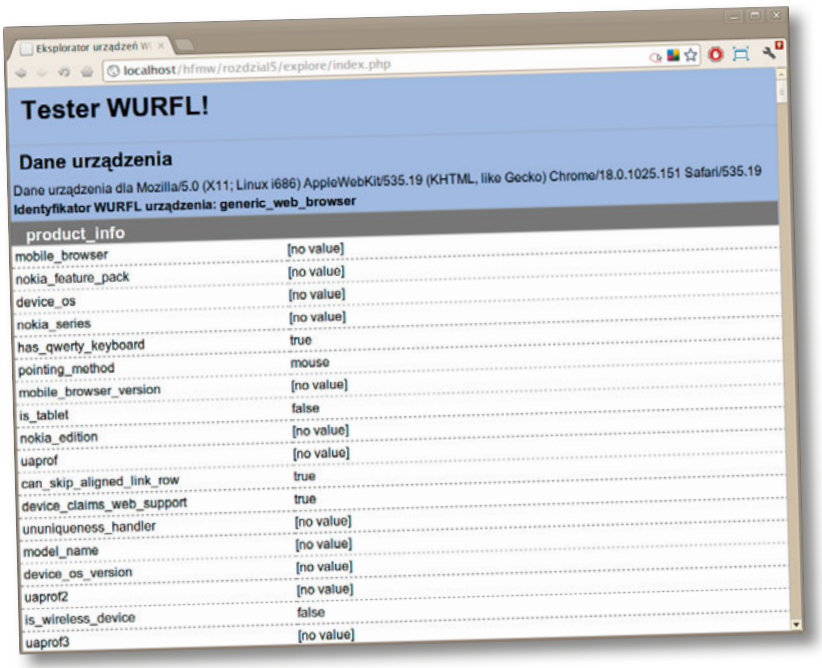
No dobra, sprawdźmy to! Zapisz plik index.php i otwórz go w przeglądarce.

index.php

Dobrze zaczęliśmy!

Możesz zobaczyć wszystkie możliwości swojej przeglądarki desktopowej pogrupowane według kategorii narzuconych przez WURFL.

- Przygotujemy środowisko robocze i pliki, a także skonfigurujemy WURFL.
- Napiszemy (ramowy) kod PHP inicjalizujący niektóre obiekty WURFL, tak aby można było uzyskać dostęp do informacji o możliwościach bieżącej przeglądarki i urządzenia.
- Opracujemy strukturę informacji o możliwościach, stworzymy stronę i wyświetlimy te dane w tabeli.



Ale możemy to jeszcze ulepszyć

Zamiast wyświetlać możliwości bieżącej przeglądarki (zwłaszcza że informacje o przeglądarce desktopowej są średnio interesujące), możemy zmodyfikować nasz eksplorator tak, by wyświetlał informacje dla dowolnego łańcucha user-agent. Wystarczy, że w pliku *index.php* umieścisz krótki formularz HTML i wprowadzisz drobną zmianę w pliku *device.php*.

Szast-prast i eksplorator ulepszony

Wystarczy kilka drobnych zmian, a nasz eksplorator stanie się bardziej użyteczny.

Zmienna $_SERVER['PHP_SELF'] zawiera lokalizację aktualnie wykonywanego skryptu. Dzięki temu formularz zostanie przesłany do bieżącej strony.

```
<div id="testform">
<form method="post" action="<?php print $_SERVER['PHP_SELF']; ?>"
id="useragentform">
   <p>Sprawdź ten łańcuch user-agent:</p>
   <input type="text" name="useragent" id="useragent_field"
      value="<?php print $user_agent; ?>" /><br />
   <input type="submit" name="submit" value="Sprawdź łańcuch UA" id="submit" />
</form>
</div>
```

Ten formularz pozwala na wprowadzenie dowolnego łańcucha user-agent.

index.php

```
require_once('config.php');
$user_agent = (isset($_POST['useragent'])) ? $_POST['useragent'] : $_SERVER['HTTP_USER_
AGENT'];
$wurflConfig = new WURFL_Configuration_XmlConfig($wurflConfigFile);
```

Zmodyfikuj ten wiersz.

W skrypcie device.php sprawdzamy, czy istnieje wartość łańcucha user-agent (w zmiennej $_POST). Jeżeli nie istnieje, używany jest łańcuch bieżącej przeglądarki.

device.php

Jazda próbna

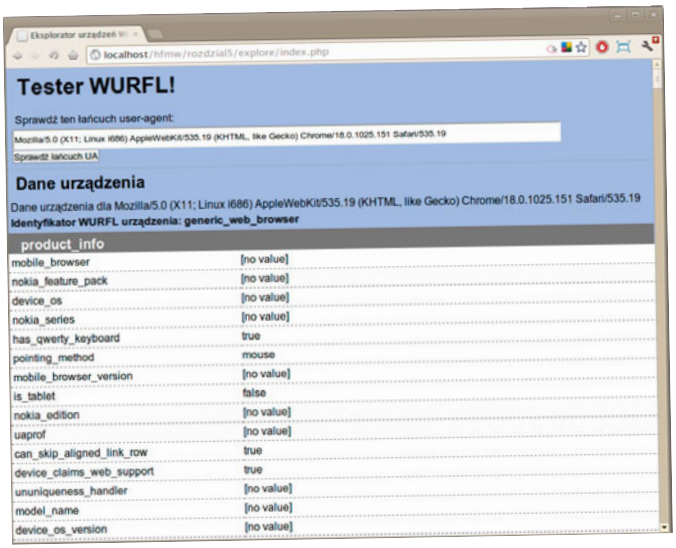
1. **Zapisz wszystkie zmiany i w przeglądarce otwórz plik index.php.**
   Po pierwszym załadowaniu strony powinieneś zobaczyć możliwości swojej przeglądarki desktopowej.

2. **Sprawdź kilka łańcuchów user-agent mobilnych przeglądarek.**
   W polu formularza wpisz łańcuch user-agent dowolnej mobilnej przeglądarki i przejrzyj wyniki.

```
Mozilla/5.0 (Linux; U; Android 2.2.1; en-us; DROIDX Build/
VZW) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1 480X854 motorola DROIDX
```

Sprawdź te. →

```
Mozilla/5.0 (webOS/1.4.5; U; en-US)
AppleWebKit/532.2 (KHTML, like Gecko)
Version/1.0 Safari/532.2 Pre/1.0
```



Teraz możesz wpisać dowolny łańcuch user-agent i sprawdzić możliwości danego urządzenia.

Łańcuch UA swojej przeglądarki możesz szybko sprawdzić na stronie http://whatsmyuseragent.com.

Masz już dość wpisywania łańcuchów user-agent?

Tak właśnie sobie pomyśleliśmy. W katalogu *rozdzial5* znajdziesz plik *useful_user_agents.txt*, a w nim kilka przykładowych łańcuchów, również te, o których wspominamy w tym rozdziale.

Czas zaprząc możliwości do pracy

Masz rację, czas się zająć konkretami.

Zastanowimy się, jak możemy użyć możliwości otrzymanych z WURFL do podjęcia decyzji, czy na stronie DaRadę! wyświetlić przycisk awaryjny, czy też tego nie robić.

(No tak, przecież od tego się to wszystko zaczęło!)

Korzystamy z WURFL do zróżnicowania zawartości stron

Wielki czerwony przycisk chcemy wyświetlić tylko w tych przeglądarkach, które zostaną przez WURFL zaklasyfikowane jako **przeglądarki mobilne** (co nie jest równoważne jedynie z małym ekranem, jak to było w przypadku zapytań o media). W tym celu skorzystamy oczywiście z możliwości zwracanych przez WURFL (nie mów, że się *tego* nie spodziewałeś).

Trzy kroki do sukcesu

- ☐ Utworzymy plik konfiguracyjny i dodamy trochę kodu inicjalizującego obiekty WURFL. W ten sposób zdobędziemy informacje o bieżącym urządzeniu i będziemy mogli z nich skorzystać.

  To już opisaliśmy w poprzednim ćwiczeniu!

- ☐ Poprosimy WURFL o podanie określonych możliwości (jednej lub wielu).

- ☐ Do makiety strony Spanikowałem! dodamy kod, dzięki któremu urządzeniom o różnych możliwościach będzie udostępniana odpowiednia zawartość.

Zadaj odpowiednie pytania

Na stronie 165 wspomnieliśmy o metodzie `getCapability` obiektu `$device`.

To bardzo przydatna metoda. Nawet jeśli to całe zamieszanie z klasami i obiektami jest dla Ciebie czarną magią, uzyskanie informacji o możliwościach nie jest wcale takie skomplikowane. Pokażemy Ci zresztą, jak to zrobić.

Pobieranie wartości dla określonej możliwości wygląda tak:

Zmienna $value przyjmie wartość wskazanej możliwości urządzenia lub NULL, jeśli nie ma takiej możliwości.

Może to być na przykład „is_tablet" czy „brand_name".

Łańcuch znaków odpowiadający nazwie możliwości.

```
$value = $device->getCapability($nazwa_możliwości);
```

Obiekt WURFL klasy CustomDevice (wcześniej utworzony i wypełniony danymi).

Zajmiemy się tym już za moment (w pliku device.php).

Kilka przykładów:

Czy to jest urządzenie mobilne?

```
$value = $device->getCapability('is_wireless_device');
```

```
$value = $device->getCapability('cookie_support');
```

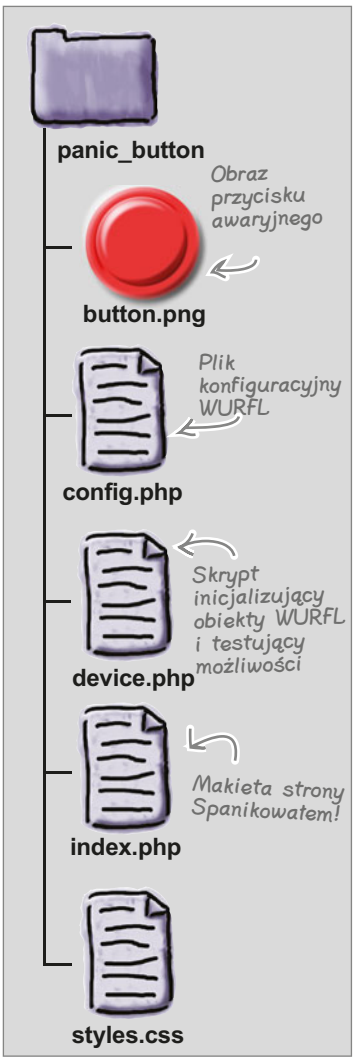
Czy przeglądarka na tym urządzeniu obsługuje ciasteczka?

Zwróć uwagę, że kiedy pobierasz wartość możliwości, nie musisz się przejmować grupą, do której należy ta możliwość. Grupy pozwalają na uporządkowanie wszystkich możliwości, ale nie trzeba ich uwzględniać podczas odczytywania wartości, ponieważ nazwy możliwości są unikatowe.

Pełną informację na temat możliwości obsługiwanych przez WURFL znajdziesz pod adresem http://wurfl.sourceforge.net/help_doc.php.

Zrób to sam!

Plik *config.php* skopiuj z katalogu *explore* i umieść go w katalogu *panic_button*.



Inicjalizacja obiektu urządzenia i przygotowanie danych

Zajrzyj do katalogu *panic_button*, a zobaczysz, że przygotowaliśmy wszystko, co niezbędne, byś bez większych problemów mógł zacząć integrować WURFL ze stroną Spanikowałem!. W pliku *device.php* znajduje się już większość kodu inicjalizującego.

Zastosujemy inną metodę tworzenia obiektu reprezentującego urządzenie, niż to miało miejsce w przypadku strony eksploratora. Nie wywołamy metody `getDeviceForUserAgent` z przekazanym łańcuchem user-agent, ale pozwolimy bibliotece WURFL zdecydować, z jakich danych z bieżącego żądania ma skorzystać, przekazując jej zmienną `$_SERVER`.

Tłumaczenie: podaj nam dane urządzenia, które zgłosiło żądanie do serwera.

```
$wurflManager = $wurflManagerFactory->create();
$device = $wurflManager->getDeviceForHttpRequest($_SERVER);
```

device.php

Ta metoda korzysta z kilku nagłówków HTTP innych niż User-Agent, jednak to właśnie on jest podstawowym źródłem informacji.

Czy to jest urządzenie mobilne?

Obiekt WURFL reprezentujący urządzenie jest już zwarty i gotowy. Czas zadać mu pytanie.

Wygląda to na boolowską właściwość typu TRUE/FALSE.

WURFL oferuje przydatną właściwość `is_wireless_device`, która udziela odpowiedzi na podstawowe pytanie: „Czy to jest urządzenie mobilne?”. ***Wydaje się*** więc, że powinniśmy zrobić coś takiego:

```
$device = $wurflManager->getDeviceForHttpRequest($_SERVER);
$is_phone = $device->getCapability('is_wireless_device');
```

device.php

Świetne, prawda?

Diabeł tkwi w szczegółach

W API dla PHP wszystkie wartości przypisane możliwościom są zwracane jako łańcuchy tekstowe. Oznacza to, że przykładowa boolowska możliwość `is_wireless_device` może mieć jedną z trzech wartości: `'true'`, `'false'` albo `NULL` (`NULL` oznacza, że nie udało się określić wartości dla tej możliwości).

W związku z tym musimy dokładnie określić sprawdzany warunek, ponieważ — ze względu na potraktowanie wartości `'false'` jako `true` — w przeciwnym przypadku stacjonarne urządzenie zostanie uznane za mobilne. Musimy zatem uwzględnić te dodatkowe uwarunkowania. Zmień drugi wiersz ostatnio omawianego kodu:

W PHP łańcuch 'false' jest traktowany jako true. Nie przejmuj się, jeżeli nie do końca rozumiesz, o co w tym chodzi. Wystarczy, że odpowiednio zmodyfikujesz fragment kodu ze strony 172.

```
$is_phone = $device->getCapability('is_wireless_device');
$is_phone = ($device->getCapability('is_wireless_device') === 'true') ? true : false;
```

Zastosowany tu operator identyczności (===) oznacza, że wartość musi być łańcuchem tekstowym 'true', a nie tylko wartością odpowiadającą true.

device.php

A teraz użyjemy tej wartości

W pliku *index.php* dołączyliśmy skrypt *device.php*, więc mamy dostęp do kodu, który właśnie napisaliśmy. Do pliku *index.php* wprowadź poniższą instrukcję warunkową:

index.php

```
<div id="content">
   <h1>Spanikowałem!</h1>
   <?php if ($is_phone): ?>
   <div id="panic_button">
      <img src="button.png" alt="POMOCY!" />
   </div>
   <?php else: ?>
   <h2>Wystarczy jeden telefon i po problemie!</h2>
   <div id="big_number">
      777-222-313
   </div>
   <?php endif; ?>
   <p>Stresujesz się przed egzaminem? Masz problem z zadaniem i nie możesz ruszyć z miejsca?
Nasi nauczyciele na telefon, eksperci w wielu dziedzinach, tylko czekają, by Ci pomóc!</p>
</div>
```

Dla żądań, które zostały zaklasyfikowane do urządzeń mobilnych (przez możliwość is_wireless_device), wyświetlamy przycisk awaryjny...

...a w przeciwnym przypadku wyświetlamy tylko numer telefonu, ale za to ogromny (rozmiar jest zdefiniowany w arkuszu CSS).

Jazda próbna

W plikach *device.php* i *index.php* wprowadź zmiany opisane na stronach 172 i 173, zapisz pliki, a następnie wyświetl stronę Spanikowałem! zarówno w przeglądarce desktopowej, jak i mobilnej.

Wszystko o WURFL

Wywiad tygodnia:
Do czego może nam się przydać WURFL?

Redaktor: Witaj, WURFL. Co ze sobą przyniosłeś?

WURFL: Ot takie małe cudeńko. Powie ci, czy przeglądarka jest mobilna, czy nie.

Redaktor: Przecież są na to inne sposoby — wykrywanie po stronie klienta, proste sprawdzanie po stronie serwera...

WURFL: Ale ograniczenie się do określenia mobilne – niemobilne to dziecinada. Nie skorzystałeś z mojego potencjału.

Redaktor: W porządku, wyjaśnij mi to.

WURFL: Dobrze. Spróbuj wyświetlić stronę w iPodzie Touch.

Redaktor: Chwileczkę... [*dłuższa przerwa, podczas której redaktor przekopuje szufladę z urządzeniami mobilnymi*] ...już mam. I co widzę? Wielki czerwony przycisk.

WURFL: No cóż, nie wygląda to za dobrze. Pewnie nie tego się spodziewałeś. Udało ci się kiedyś zadzwonić z iPoda?

Redaktor: Świetnie... Przecież ten przycisk nie ma sensu na iPodzie Touch. Jednym słowem, wracamy do punktu wyjścia?

WURFL: Nie, nie i jeszcze raz nie! Tak jak powiedziałem, nie skorzystałeś z całego mojego potencjału. Musisz kopać głębiej, by dostrzec potęgę moich możliwości. Sądzę, że co najmniej jedna lub dwie cię mile zaskoczą.

Redaktor: Ale przecież oferujesz ponad 500 możliwości! Jak mam znaleźć tę właściwą?

WURFL: Ech! Narzekasz, bo jestem zbyt skomplikowany? Poddaję się... Dobra, dam ci podpowiedź: to, że przycisk jest wyświetlany na urządzeniach mobilnych, nie znaczy, że możesz zadzwonić. Przecież przycisk nie jest jeszcze w żaden sposób obsługiwany — to tylko zwykły rysunek, który nie pełni żadnej konkretnej funkcji. Przemyśl to na spokojnie, a potem jeszcze raz rzuć okiem na moje możliwości.

Ćwiczenie

Otwórz stronę naszego eksploratora i przejrzyj możliwości urządzenia iPod Touch (na którym przycisk nie ma być wyświetlany). Porównaj je z możliwościami smartfonu Nexus S z Androidem (na którym ma się pojawić przycisk). Możesz też sprawdzić — dla porównania — możliwości przeglądarki desktopowej.

Czy umiesz wskazać możliwość, która może nam pomóc rozróżnić małe lub mobilne urządzenia od telefonów?

Łańcuch UA iPoda Touch

```
Mozilla/5.0 (iPod; U; CPU iPhone OS 4_3_2 like Mac
OS X; pl-pl) AppleWebKit/533.17.9 (KHTML, like Gecko)
Version/5.0.2 Mobile/8H7 Safari/6533.18.5
```

```
Mozilla/5.0 (Linux; U; Android 2.3.3; pl-pl; Nexus S Build/
GRI40) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1
```

Łańcuch UA Nexusa S

Przycisk pojawia się na iPodzie Touch — przecież to nie ma sensu.

Rozwiązanie ćwiczenie

W grupie `bearer` znajduje się przydatna możliwość `has_cellular_radio`. Skorzystamy z niej, by sprawdzić, czy żądanie pochodzi z telefonu, czy tylko z urządzenia mobilnego.

Zaprzyjaźnimy się również z możliwością `xhtml_make_phone_call_string`, dzięki której wybierzemy optymalny sposób realizacji odsyłacza do numeru telefonu dla danego urządzenia.

Dzięki WURFL strona staje się sprytniejsza

Sprawdzając wartość możliwości `has_cellular_radio`, możemy stwierdzić, czy bieżący użytkownik korzysta z telefonu. Musimy się jeszcze dowiedzieć, jak stworzyć odsyłacz z numerem telefonu, tak aby po jego kliknięciu rozpoczęło się połączenie.

Wartość możliwości `xhtml_make_phone_call_string` jest wskazówką składni, którą musimy zastosować w odsyłaczu do numeru telefonu. W przypadku telefonu Nexus S wartość jest równa `tel:`.

Dla urządzenia iPod Touch wartością tej możliwości jest `none`, co oznacza, że wykonanie połączenia telefonicznego nie jest możliwe. Podobnie jest w przypadku przeglądarek desktopowych.

Dzięki połączeniu wartości `xhtml_make_phone_call_string` z numerem telefonu możemy otrzymać działający odsyłacz umożliwiający nawiązywanie połączeń telefonicznych.

Jeśli właściwość has_cellular_radio ma wartość 'true', a w xhtml_make_phone_call_string znajduje się wartość inna niż none, mamy do czynienia z telefonem!

Nawiązywanie połączeń za pomocą odsyłaczy

Przeglądarki w telefonach komórkowych rozpoznają pewne wzorce URI umożliwiające nawiązywanie połączeń telefonicznych. Gdy użytkownik kliknie tego typu odsyłacz, pojawia się okienko z prośbą o potwierdzenie, czy na pewno wykonać połączenie.

Najbardziej typowym schematem URI dla numerów telefonów, działającym w większości nowych smartfonów, jest `tel:`. Nie zaszkodzi jednak skorzystać z wartości możliwości `xhtml_make_phone_call_string` — przecież tak czy inaczej już ją masz. Efekt końcowy ma wyglądać tak:

```
<a href="tel:+48777222313">Zadzwoń</a>
```

Chociaż w odsyłaczach tel: nie trzeba umieszczać międzynarodowego numeru kierunkowego (np. +48), powinniśmy zawsze go dodawać, by umożliwić dodzwonienie się z zagranicy.

Przycisk awaryjny — tylko w telefonach

W skrypcie *device.php* możemy umieścić dodatkowy warunek sprawdzający wartości możliwości `has_cellular_radio` i `xhtml_make_phone_call_string`.

Najpierw pobieramy możliwości

Tej linii kodu już nie potrzebujemy.

```
$is_phone = ($device->getCapability('is_wireless_device') === 'true') ? true : false;
$has_radio = $device->getCapability('has_cellular_radio');
$phone_string = $device->getCapability('xhtml_make_phone_call_string');
```

device.php

Później określamy ich wartości

```
$phone_string = $device->getCapability('xhtml_make_phone_call_string');
$is_phone = false;
if ($has_radio === 'true' && $phone_string && $phone_string !== 'none') {
    $is_phone = true;
}
```

device.php

To dlatego musimy sprawdzać zarówno to, czy zmienna $phone_string ma wartość, jak i to, czy ta wartość jest różna od łańcucha 'none'.

Obejrzyj to!

Wartość 'none' też ma znaczenie!

Na stronie 173 *wspomnieliśmy, że metoda* `CustomDevice->getCapability()` *z API dla PHP zwraca łańcuch tekstowy (jeśli istnieje wartość dla tej możliwości) albo NULL (jeśli WURFL nie potrafi określić wartości). Ta różnica jest bardzo istotna.*

Dla urządzeń takich jak iPod Touch i przeglądarek desktopowych wartością zwracaną dla możliwości `xhtml_make_phone_call_string` *jest łańcuch* `'none'`. *Jeśli potraktujesz go w PHP jako wartość boolowską, otrzymasz* `TRUE`.

Nie istnieją głupie pytania

P: Czy przeglądarki w urządzeniach mobilnych nie rozpoznają automatycznie numerów telefonów i czy nie tworzą z nich odsyłaczy?

O: Większość tak. Nie jest to jednak takie proste. Zastanówmy się choćby nad naszą sytuacją — elementem, który ma być odsyłaczem, jest obrazek, a nie numer telefonu (w postaci tekstu). Z całą pewnością przeglądarce telefonu nie uda się go rozpoznać.

Poza tym rozpoznawanie numerów telefonów w różnych przeglądarkach działa różnie, czasem nie do końca poprawnie. Na przykład Opera Mini często traktuje kod pocztowy jako numer telefonu. Łatwo sobie wyobrazić, że telefonowanie na numer kodu pocztowego nie przyniesie ciekawych efektów.

Zwróć też uwagę, że numer telefonu zapisaliśmy w bardzo formalnym formacie (+48777222313). Całkiem możliwe, że na stronie ten numer powinien być zaprezentowany w bardziej przyjaznej formie, z odstępami lub łącznikami, natomiast sam odsyłacz powinien być zapisany bez odstępów.

P: Dlaczego po kliknięciu odsyłacza z numerem telefonu musi się pojawić okienko z prośbą o potwierdzenie wybrania tego numeru? Czy nie możemy wykonać połączenia bezpośrednio po kliknięciu?

O: Chodzi tu o zabezpieczenie przed oszustwami. Przecież numer zapisany w odsyłaczu może się różnić od tego, który jest wyświetlany na stronie. Niczego nieświadomy użytkownik mógłby więc wykonać bardzo kosztowne połączenie, myśląc, że dzwoni w sprawie prenumeraty lokalnej gazety.

W naszym przypadku nie wyświetlamy nawet numeru telefonu, co można uznać za nie najlepsze rozwiązanie. Można by zamienić obrazek wstawiony w kodzie HTML na tło ustawione w arkuszu stylów, a numer telefonu wpisać jako tekst odsyłacza (i ustawić duży ujemny margines, by ukryć ten tekst).

P: A po polsku? W szkole nie byłem prymusem z CSS...

O: Ustawienie dużego ujemnego lewego marginesu (np. `margin-left: -10000px`) powoduje przesunięcie tekstu poza widoczny obszar, a więc jego ukrycie. Tekst znajduje się 10 000 pikseli od elementu, czyli lewituje gdzieś w naszym pokoju. Ależ jesteśmy niesamowici!

Na ekranie iPoda Touch już nie ma przycisku! Bardzo dobrze, że został na iPhonie!

P: Jeśli możliwość `xhtml_make_phone_call_string` ma wartość inną niż `none`, możemy przypuszczać, że chodzi o telefon komórkowy, ale czy wiemy na pewno, że można z niego dzwonić?

O: WURFL udostępnia wiele pożytecznych danych, ale nie jest aż tak sprytny. Nie ma żadnego sposobu na to, by stwierdzić z całą pewnością, że użytkownik ma dostęp do sieci komórkowej (a nie na przykład do sieci WiFi lub czegoś jeszcze innego) lub że nie zaistniały jakieś tymczasowe przeszkody uniemożliwiające nawiązanie połączenia telefonicznego. Mimo wszystko jest to i tak najlepsze z możliwych rozwiązań.

P: Czy nie wystarczyłoby sprawdzić, czy wartość `xhtml_make_phone_call_string` jest różna od 'none'? Sprawdzanie możliwości `has_cellular_radio` wydaje się niepotrzebne.

O: Wygląda na to, że zaczynasz łapać, mądralo.

P: Skoro mowa o mądralach — bardzo pobieżnie przelecieliście po możliwościach oferowanych przez WURFL. Chciałbym dowiedzieć się więcej na temat obiektów WURFL, metod API i...

O: Dobra, dobra. Przecież nic nie stoi na przeszkodzie, abyś przejrzał kod klas PHP tworzących API i zobaczył, jak to wszystko działa. Po zainstalowaniu API masz wszystkie pliki u siebie na dysku, więc do dzieła!

Kuba: To się powoli zaczyna wymykać z rąk! W bardziej złożonym projekcie, a nawet w przypadku witryny DaRadę! dodawanie kolejnych warunków doprowadzi do powstania strasznie poszatkowanego kodu, a to zawsze się kończy tak samo — godzinami spędzonymi na dochodzeniu do tego, co jest czym, i nieodłącznym bólem głowy. Chodzi mi o to, że będzie to wyglądało tak, że w jednym miejscu sprawdzamy jedną możliwość, w innym miejscu kolejną i tak dalej...

Łukasz: Zgadzam się. Wygląda na to, że dla każdej możliwej kombinacji możliwości będziemy dostarczać inną stronę. Testowanie czegoś takiego to istny koszmar!

Kuba: Dwa słowa, które mrożą krew w żyłach każdego programisty: „spaghetti code”...

Łukasz: No tak, ale nie chciałbym porzucać samego pomysłu, bo wydaje się dobry. Mimo że w niektórych sytuacjach wystarczy wykrywanie po stronie klienta, nadal dążymy do tego, żeby nasza strona jak najlepiej dopasowywała się do możliwości urządzenia. A to może nam zapewnić tylko baza danych urządzeń, czyli na przykład WURFL, bo żadna inna technika nie pozwoli uzyskać tak wielu szczegółowych informacji.

Kuba: Ale jak sobie z tym poradzić i nie zwariować?

Łukasz: Może zamiast sprawdzać każdą z możliwości pogrupowalibyśmy urządzenia? Pamiętasz, jak niedawno zajmowaliśmy się wyborem urządzeń, które zamierzamy wspierać? Wybraliśmy istotne cechy i możliwości, a następnie wyznaczyliśmy granicę.

Kuba: Jasne, że pamiętam. Przez to ćwiczenie jeszcze teraz kręci mi się w głowie!

Łukasz: Moglibyśmy pójść o krok dalej i — w ramach wspieranych urządzeń — zdefiniować grupy urządzeń o zbliżonych możliwościach.

Kuba: Ciekawe... Całkiem nieźle to wymyśliłeś.

Ćwiczenie

Przykład tańcucha UA znajdziesz na stronie 175.

Zaktualizuj plik *device.php* — wprowadź zmiany opisane na stronie 173 i dodaj fragmenty ze strony 177. Pora przetestować kod. Tylko jak to zrobić, jeżeli nie masz pod ręką iPoda Touch?

Aby sprawdzić, jak strona wygląda na iPodzie Touch, zmodyfikuj lekko skrypt *device.php*. Zamiast metody `getDeviceForHttpRequest()` zastosuj metodę `getDeviceForUserAgent()` i podaj łańcuch user-agent iPoda Touch.

Rozwiązanie znajdziesz na stronie 180.

Zaganianie urządzeń

Klasa urządzeń tworzy pewnego rodzaju logiczną „zagrodę”, do której możesz zagonić urządzenia mające wspólne cechy.

Podobnie jak hodowca może umieścić czarne krowy w kropki bordo na jednym pastwisku, konie na innym, a prosiaki na jeszcze innym (nie mów, że nigdy nie zaganiałeś bydła...), tak i my możemy rozmieścić urządzenia w wirtualnych klatkach.

Klasa urządzeń to abstrakcyjny zbiór cech definiujący grupę urządzeń (i ich przeglądarek).

Pogrupuj raz, a później korzystaj

Kiedy nasz hodowca zidentyfikuje jakieś zwierzę jako należące do jednej z tych trzech grup, podjęcie dalszych decyzji nie wymaga analizy szczegółowych informacji. Może dostarczyć owies wprost do zagrody z końmi bez sprawdzania, czy zwierzęta z tej zagrody mają nogi odpowiedniej długości, bujną kiść ogona i czy wspierają przezroczyste obrazki PNG (przecież każde dziecko wie, że konie nie wspierają przezroczystych PNG!).

Hodowca wie, że to są konie, ponieważ znajdują się w zagrodzie dla koni. Nie musi dostarczać jedzenia każdemu zwierzęciu z osobna (musiałby zapanować nad pokaźną liczbą posiłków). Nie może jednak nakarmić owsem prosiaków.

Rozwiązanie ćwiczenia

Aby sprawdzić, jak wygląda strona na określonym urządzeniu w konkretnej przeglądarce, musisz przekazać jej łańcuch user-agent podczas inicjalizowania obiektu WURFL reprezentującego urządzenie.

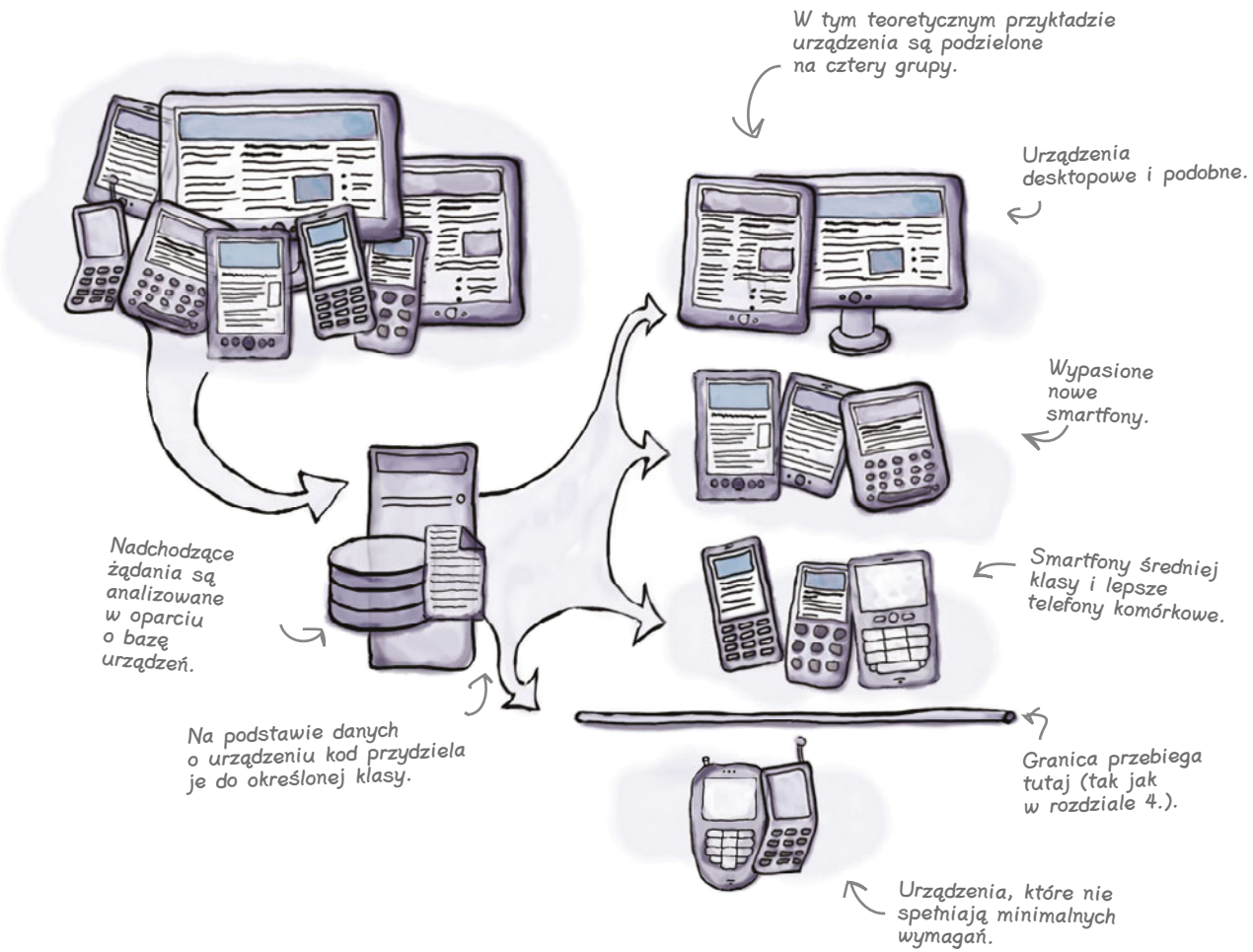
device.php

```
$wurflManager = $wurflManagerFactory->create();
~~$device = $wurflManager->getDeviceForHttpRequest($_SERVER);~~
$user_agent = "Mozilla/5.0 (iPod; U; CPU iPhone OS 4_3_2 like Mac
OS X; pl-pl) AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2
Mobile/8H7 Safari/6533.18.5";
$device = $wurflManager->getDeviceForUserAgent($user_agent);
```

Klasy urządzeń

Już wcześniej analizowaliśmy nadchodzące żądania w oparciu o bazę urządzeń i odczytywaliśmy informacje o konkretnym urządzeniu. Dzięki pogrupowaniu możliwości oraz wartości istotnych z punktu widzenia tworzonej strony możemy dostosować jej zawartość do całej klasy urządzeń, zamiast skupiać się na każdej możliwości z osobna.

Te **definicje klas urządzeń**, przedstawione w postaci kodu, przyporządkowują urządzenia do określonych grup. Po pogrupowaniu urządzeń możemy wykonać pożądane operacje bez potrzeby zwracania uwagi na poszczególne możliwości i ich wartości.



Kolejne zadanie, tym razem poważniejsze

Cześć, chłopaki! Nieźle sobie poradziliście z tym przyciskiem awaryjnym. Może pomoglibyście zoptymalizować pod kątem urządzeń mobilnych naszą nową witrynę? Dacie radę?

Kolejny lukratywny kontrakt z firmą DaRadę!

Firma DaRadę! tworzy witrynę z pomocami dydaktycznymi, które chce sprzedawać studentom. Okazuje się, że plansze, ćwiczenia, podręczniki i tym podobne materiały idą jak ciepłe bułeczki, ale do tej pory jedynym sposobem na ich zamówienie było skorzystanie z katalogu w wersji papierowej.

Programiści z DaRadę! tworzą to z wykorzystaniem JavaScriptu i innych standardowych technologii internetowych.

Czas na zmiany. Witryna firmy DaRadę! ma nie tylko umożliwiać kupienie dowolnego produktu online, ale również interaktywne skorzystanie z części z nich, na przykład z plansz, bezpośrednio na stronie.

Szybki rzut oka na wstępny projekt

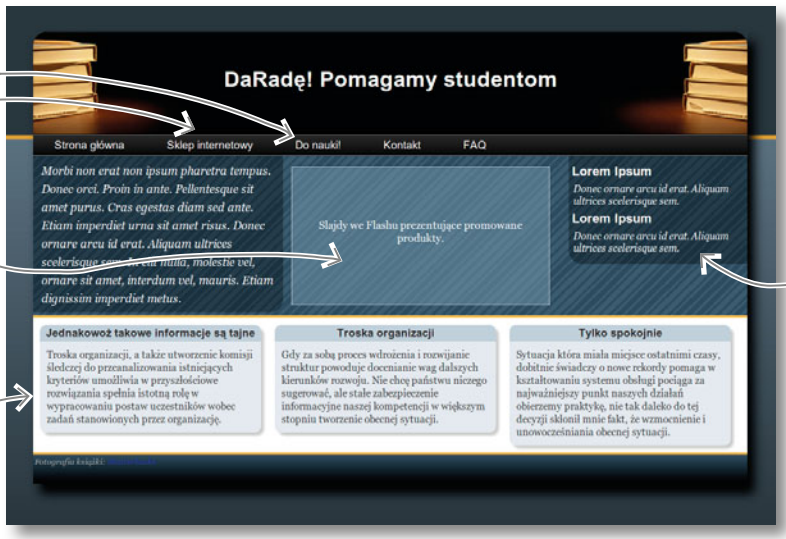
Pomysły na witrynę dopiero kiełkują, a realizacja jest w początkowej fazie. Jeszcze nawet nie powstał projekt logo, treści są powoli uzupełniane, a układ całej witryny jest na razie na etapie szkicu makiety. Zobaczmy, co do tej pory zostało ustalone:

Ludzie odpowiedzialni w firmie DaRadę! za sprawy internetowe przestali tę makietę głównej strony nowej witryny.

Użytkownicy mogą teraz nie tylko kupować online, ale i studiować dzięki interaktywnym produktom dostępnym na stronie.

Firma DaRadę! chce reklamować swoje produkty w filmie flashowym.

Na głównej stronie mają być wyświetlane fragmenty wpisów z firmowego bloga.



Tu znajdą się reklamy usług powiązanych z tematyką witryny (np. sprawdzanie prac, wspólna nauka itp.).

Jak strona główna wygląda przez mobilne okulary?

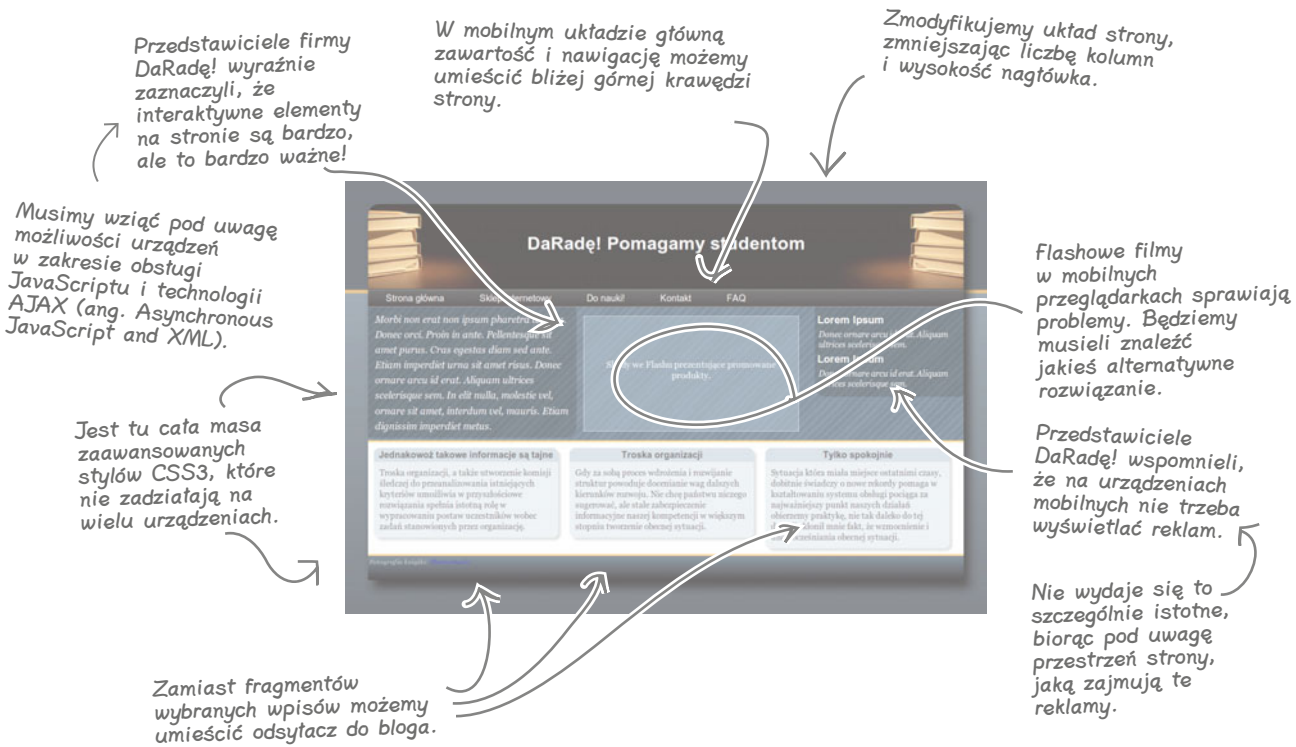
Zadanie stawiane nowej witrynie z pomocami dla studentów jest znacznie poważniejsze niż pojedyncza strona z przyciskiem awaryjnym. Gdybyśmy zajęli się każdym aspektem witryny z osobna, możliwość po możliwości, skończylibyśmy najpewniej w wariatkowie. Jedynym rozsądnym rozwiązaniem, dzięki któremu uda nam się zapewnić, by witryna wyglądała dobrze na różnego typu urządzeniach, jest **pogrupowanie oferowanych możliwości z wykorzystaniem klas urządzeń**.

Powiązanie danych o urządzeniu z logicznymi grupami

Wiemy, wiemy... Na razie mamy ogólny szkic głównej strony tej witryny. Jednak z całą pewnością jej funkcjonalność znacznie się rozszerzy.

Proces definiowania klas urządzeń jest powiązany z tym, co zrobiliśmy w rozdziale 4. Bierzemy jakąś funkcjonalność, określamy jej najistotniejsze elementy, wypowiadamy kilka zaklęć i z kapelusza wyciągamy ogólne kryteria.

Tak, wiemy. Sformułowanie „pogrupowanie oferowanych możliwości z wykorzystaniem klas urządzeń" jest bardzo mało konkretne. Zacznijmy od ponownego przyjrzenia się makiecie i określenia priorytetów poszczególnych funkcjonalności z uwzględnieniem tego, co już wiemy o możliwościach i ograniczeniach mobilnych technologii internetowych.

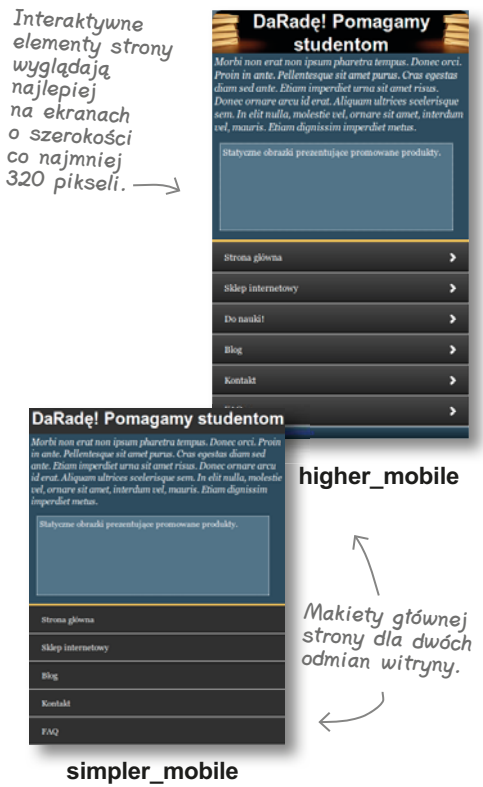


Zdefiniowanie odmian witryny w zależności od wymagań

Funkcjonalności oferowane przez interaktywne witryny zwiększają wymagania. Mobilne urządzenia muszą zapewnić obsługę JavaScriptu, HTML-a i CSS, a to jest możliwe tylko w przeglądarkach nowszych i lepszych smartfonów. Programiści z firmy DaRadę! skupiają się na urządzeniach z przeglądarkami bazującymi na silniku WebKit i ekranami nie węższymi niż 320 pikseli.

Nie oznacza to jednak, że użytkownicy pozostałych urządzeń mają zostać pominięci. Mniejsze i nie tak wydajne telefony też są mile widziane. Za ich pomocą można przecież zamawiać produkty w sklepie internetowym i przeglądać wiele innych stron witryny. Nie mogą jednak wyświetlić tej zawartości, która wymaga dużej wydajności i zaawansowanych możliwości, na przykład obsługi CSS3.

Zamiast dostarczać wszystkim treści spełniające wymagania najsłabszych urządzeń albo odrzucać nie dość dobre urządzenia, **utworzymy dwie oddzielne odmiany witryny**, które będą odpowiednie dla każdej z grup urządzeń.



Bogatsza odmiana witryny

Dzięki utworzeniu klasy urządzeń spełniających wygórowane wymagania możemy zaprojektować całkiem bogatą witrynę i nie musimy się przejmować tym, że na słabszych urządzeniach nie będzie prawidłowo działała.

Dzięki określeniu, że urządzenia w tej klasie muszą mieć przeglądarkę opartą na silniku WebKit, możemy użyć takich dobrodziejstw CSS jak na przykład gradienty. Możemy też liczyć na przyzwoite wsparcie dla JavaScriptu.

Uproszczona odmiana dla słabszych telefonów

Dla urządzeń o węższych ekranach i mniejszych możliwościach uszczuplimy nieco założenia projektu. Musimy też się pogodzić z tym, że nie możemy skorzystać z wielu możliwości nowoczesnych przeglądarek.

W prostszym układzie nie umieścimy odsyłaczy do produktów działających online, ale nadal będzie istniała możliwość kupowania w sklepie internetowym. Ta wersja układu strony powinna dobrze wyglądać na ekranach o szerokości nawet 176 pikseli, bo przy mniejszych zdjęciach produktów byłyby one całkiem nieczytelne.

Mamy zatem dwie ogólnie zdefiniowane klasy urządzeń: `higher_mobile` i `simpler_mobile`.

Przybliżamy klasy urządzeń

Udało nam się ogólnie zdefiniować dwie klasy urządzeń. Coś jeszcze?

No tak — tablety

Programiści z DaRadę! pracują też nad wersją dla tabletów z eleganckim dotykowym interfejsem użytkownika dla produktów działających online. Jeszcze nie skończyli, ale chcą być gotowi na dołączenie tej wersji, więc zależy im na rozpoznawaniu **tabletów**.

Gdzie wyznaczyć granicę?

Dla ludzi z firmy DaRadę! bardzo ważne jest to, by użytkownicy mogli kupować w sklepie internetowym i korzystać z produktów działających online. Jeśli którakolwiek z tych funkcjonalności nie jest zapewniona z powodu ograniczeń urządzenia, takie urządzenie (i jego przeglądarka) są traktowane jako nieobsługiwane.

Oprogramowanie sklepu internetowego wymaga **obsługi ciasteczek** oraz podstawowego **wsparcia dla JavaScriptu** (nie tak dużego jak produkty działające online). Ze względów bezpieczeństwa jest też konieczne zapewnienie wsparcia dla **SSL**.

Podsumowanie wymagań

W oparciu o priorytety i wymagania udało się wyznaczyć granicę wsparcia. Urządzenia mobilne muszą oferować następujące możliwości:

- wsparcie dla SSL,
- obsługa ciasteczek,
- podstawowe wsparcie dla JavaScriptu,
- szerokość ekranu co najmniej 176 pikseli.

Nasze klasy urządzeń

Wszystko o KLASACH URZĄDZEŃ

Wywiad tygodnia:
Od teorii do praktyki. Czym tak naprawdę jest klasa urządzeń?

Trochę zagubiony twórca stron: Nie umiem do końca zrozumieć, o co chodzi z tymi klasami urządzeń. Zbiór możliwości... Czy to ma związek z oferowaną funkcjonalnością?

Klasa urządzeń: Całkowicie cię rozumiem, na początku trudno dojść, o co tu chodzi. Gdybym był obrazem, byłbym abstrakcyjnym bohomazem. Jestem koncepcją umożliwiającą takie zaplanowanie witryny, by trzeba było utworzyć tylko kilka jej odmian, a nie dziesiątki czy setki.

TZTS: Zatem klasa urządzeń to, inaczej mówiąc, zbiór możliwości WURFL...?

Klasa urządzeń: Nie tak szybko! Pamiętaj, że jestem abstrakcyjną koncepcją. Owszem, skorzystamy z WURFL, ale nie jest to niezbędne. Tak naprawdę można użyć dowolnej bazy urządzeń.

TZTS: To zaczyna mnie przerastać. Pomożesz mi zrozumieć, na czym to wszystko polega?

Klasa urządzeń: Jasne! Zaplanujemy to wspólnie. W przypadku witryny DaRadę! zdefiniowaliśmy pięć klas urządzeń.

TZTS: Czy to znaczy, że musimy przygotować osobną wersję witryny dla każdej z tych klas? Przecież to masa roboty!

Klasa urządzeń: Na szczęście nie musimy. Wystarczy się skupić na różnicach. Weźmy na przykład wersję dla tabletów — jeśli pominiemy sprawę interaktywnych plansz z interfejsem dotykowym, ta wersja różni się od desktopowej tylko tym, że nie obsługujemy w niej filmów Flasha.

TZTS: W porządku. Mamy zatem klasę urządzeń stacjonarnych (o oczywistym przeznaczeniu) oraz klasę tabletów, która różni się od niej tylko drobiazgami. A co z różnicami między klasami `higher_mobile` i `simpler_mobile`?

Klasa urządzeń: Skoro wiemy, że klasa `higher_mobile` reprezentuje urządzenia z większymi ekranami i przeglądarkami bazującymi na silniku WebKit, możemy założyć, że są one również na tyle wydajne, by udźwignąć niektóre z bardziej wymagających funkcji witryny.

Z kolei klasa `simpler_mobile` reprezentuje urządzenia z mniejszymi ekranami i zapewne mniej nowoczesnymi przeglądarkami. W związku z tym możemy na przykład dostarczyć mniejsze obrazki (dzięki czemu mniej obciążymy łącze). Wiemy też, że w pewnym stopniu obsługują JavaScript, ale z całą pewnością trzeba będzie przygotować mniej bogatą wersję witryny.

TZTS: Chwilkę, a skąd wiemy, że urządzenia z klasy `simpler_mobile` obsługują JavaScript?

Klasa urządzeń: Wykonamy test sprawdzający, czy przeglądarka daje minimalne wymagane wsparcie dla JavaScriptu. Urządzenia, które nie pozwalają na modyfikowanie struktury DOM, po załadowaniu strony zostaną przerzucone do klasy niewspieranych urządzeń (`unsupported`).

TZTS: W takim razie czym się teraz trzeba zająć?

Klasa urządzeń: Musimy przyporządkować odpowiednie możliwości WURFL i ich wartości do klas urządzeń, które zamierzamy stworzyć. Kiedy będziemy już mieli logiczną reprezentację klas urządzeń, utworzymy kod wykonujący testy i przypisujący konkretne urządzenia do zdefiniowanych klas.

Zagadkowy basen

Najwyższy czas, by znaleźć możliwości WURFL i ich wartości odpowiadające zdefiniowanym przez nas klasom urządzeń. **Twoim zadaniem** jest wyłowienie możliwości i wartości WURFL z basenu i umieszczenie ich w odpowiednich klasach urządzeń. **Nie możesz** użyć dwa razy tego samego elementu.

higher_mobile

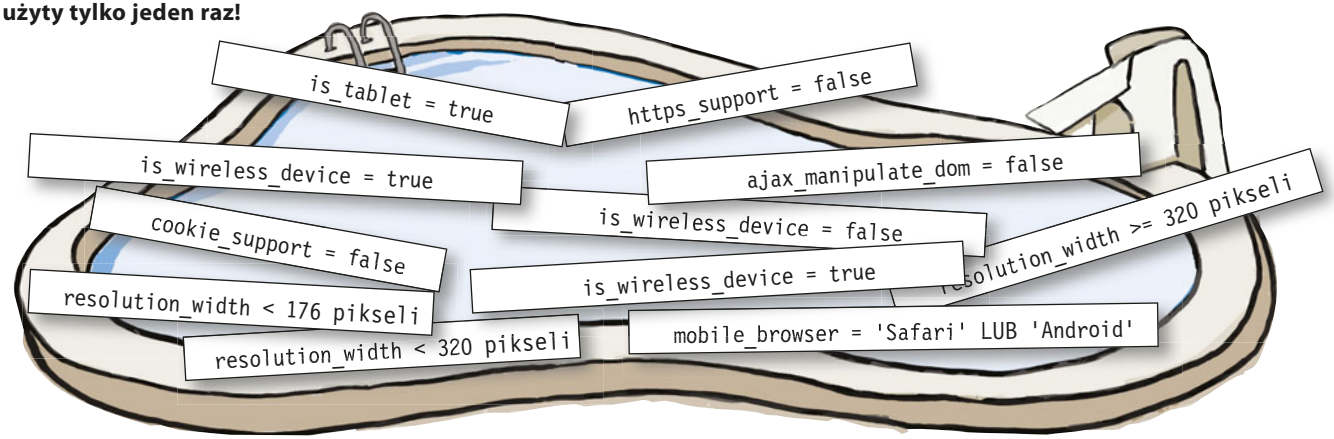
desktop

tablet

simpler_mobile

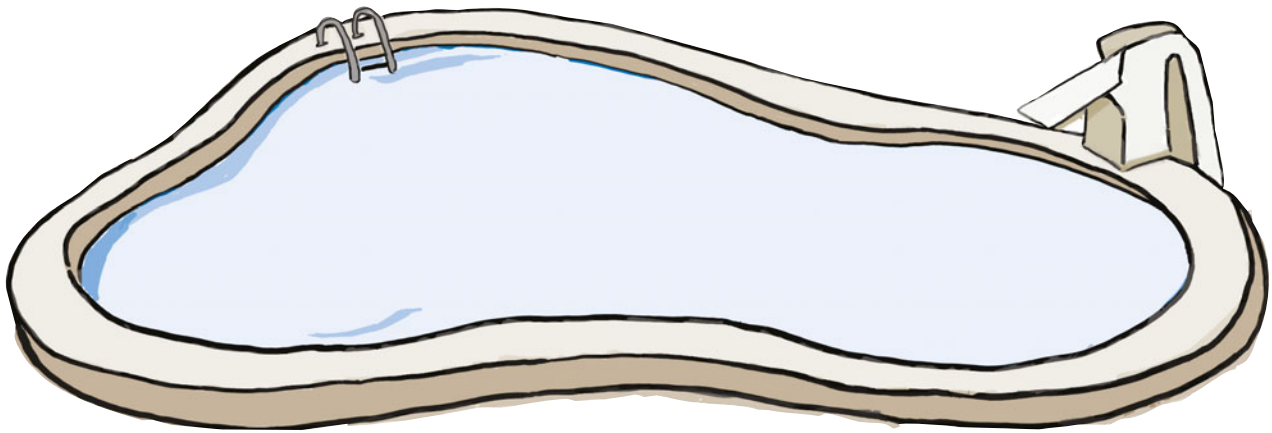
unsupported

Uwaga: każdy element z basenu może być użyty tylko jeden raz!



Zagadkowy basen. Rozwiązanie

Wszystkie możliwości i wartości WURFL przypisaliśmy do klas urządzeń.



Nie istnieją głupie pytania

P: Dlaczego stawiamy na przeglądarki oparte na silniku WebKit? Czy programiści unikają innych, całkiem przyzwoitych przeglądarek mobilnych?

O: Przeglądarki oparte na silniku WebKit są doceniane przez programistów ze względu na ich możliwości i zgodność ze standardami. Przeglądarki te przeważnie oferują wsparcie dla technologii HTML5, JavaScript i CSS3.

Zwróć uwagę, że użyliśmy słowa *przeważnie*. Problem w tym, że powszechne przekonanie o zgodności tych przeglądarek ze standardami jest nie do końca słuszne. Smutna prawda bowiem jest taka, że tu również panuje chaos.

Znany strateg platform mobilnych, Peter-Paul Koch, zajął się problemem trudności wynikających z założenia zgodności przeglądarek bazujących na silniku WebKit i opisał je na swoim blogu QuirksBlog (*http://bit.ly/uWnFLa*) w poście *There is no WebKit on Mobile*.

P: Skoro mobilny silnik WebKit nie jest idealny, dlaczego opieramy na nim klasę urządzeń?

O: Zgodność i niezawodność (lub ich brak) to jedno, ale trzeba wziąć pod uwagę, że programiści tworzą i testują oprogramowanie najczęściej właśnie z wykorzystaniem przeglądarek opartych na silniku WebKit. Na tym etapie pracy nad projektem jest za późno, by poszerzać zakres wsparcia. Skupiamy się zatem na wiodącym i ogólnie stosowanym rozwiązaniu.

P: Czy porównując nazwę przeglądarki z łańcuchami `'Safari'` i `'Android'`, uda nam się wykryć wszystkie przeglądarki bazujące na silniku WebKit?

O: Pewnie pamiętasz z rozdziału 3., że łańcuchy user-agent to podstępne stworzenia. W przypadku silnika WebKit (który został opracowany przez firmę Apple i jest stosowany m.in. w przeglądarce Safari) łańcuchy UA zachowują historycznie uwarunkowane zapisy. W związku z tym wszystkie znane przeglądarki bazujące na WebKit jako swoją nazwę podają „Safari" bądź „Android" (tak, nawet przeglądarki działające na BlackBerry, telefonach Nokii i jakichkolwiek innych urządzeniach).

P: Jeśli zdecyduję się na zastosowanie w projekcie klas urządzeń, czy zawsze będzie ich pięć?

O: Nie. Możesz mieć 10, 6 lub 0 klas. Im jest ich mniej, tym mniejsza złożoność, a im więcej, tym większa szczegółowość dopasowania treści do możliwości urządzeń.

P: Zwróciłem uwagę na nazwy `higher_mobile` i `simpler_mobile`. Czy jest ustalona jakaś konwencja nazewnicza, o której powinienem wiedzieć?

O: Nie, akurat takie nazwy przyszły nam do głowy, bo dobrze oddają przeznaczenie klas. Zamiast spacji użyliśmy znaków podkreślenia, by można je było bez zmian umieścić w kodzie. Klasom urządzeń możesz nadawać dowolne nazwy.

P: Czy kiedy już będę miał zdefiniowane klasy urządzeń, nie będę musiał sprawdzać pojedynczych możliwości urządzenia?

O: Nadal możesz to robić. Rozwiązanie, które zastosowaliśmy w przykładzie z przyciskiem awaryjnym, wydaje się właściwe — sprawdzaliśmy wartość konkretnej możliwości, w dodatku takiej, która może być różna dla różnych urządzeń w tej samej klasie (na przykład para iPhone – iPod Touch).

P: Czy może się zdarzyć sytuacja, w której jakieś urządzenie będzie pasowało do więcej niż jednej klasy?

O: Tak. Z tego względu musimy zadbać o odpowiednią kolejność przeprowadzania testów. Urządzenie zostanie przypisane do pierwszej pasującej klasy.

P: W przykładowych fragmentach kodu pojawia się obiekt `CustomDevice`. Co to za obiekt?

O: `CustomDevice` to po prostu nazwa klasy w API WURFL dla PHP, która reprezentuje urządzenie i jego cechy.

P: Wygląda na to, że klasy urządzeń nie dotyczą tylko mobilnego internetu, mam rację?

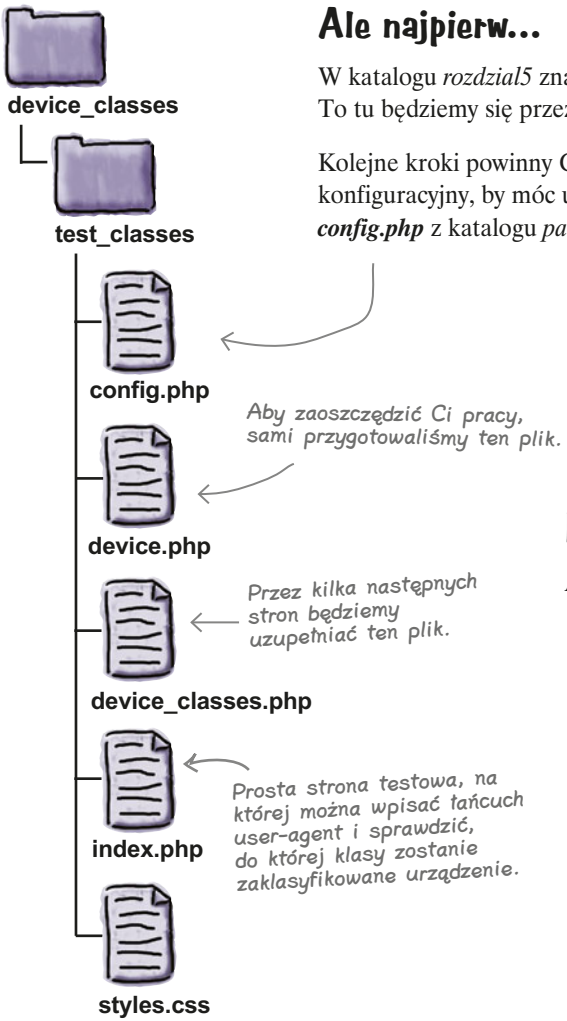
O: Oczywiście! Dopasowywanie treści i układu jest istotne we wszystkich projektach internetowych. Nie jest to tylko domena urządzeń mobilnych.

P: W porządku. Mam kilka okręgów z wypisanymi możliwościami i ich pożądanymi wartościami. Co dalej?

O: Cierpliwości! Odwróć stronę — zaczynamy przekształcać zdefiniowane klasy urządzeń w działający kod.

Czas na realizację

Musimy przekształcić abstrakcyjne zapisy dotyczące grupowania urządzeń na prawdziwy, działający kod, który wykryje i przyporządkuje urządzenia do odpowiednich klas. Do dzieła!



Ale najpierw...

W katalogu *rozdzial5* znajduje się katalog *device_classes*, a w nim *test_classes*. To tu będziemy się przez chwilę bawić.

Kolejne kroki powinny Ci się wydać znajome — musimy przygotować plik konfiguracyjny, by móc uruchomić WURFL. Zacznij od **skopiowania pliku *config.php*** z katalogu *panic_button* do *device_classes/test_classes*.

Przygotowania zakończone!

A oto, co teraz zrobimy:

- ☐ **Napiszemy funkcję (PHP)** testującą poszczególne możliwości urządzenia umieszczone w definicjach klas urządzeń.
- ☐ Przekształcimy zdefiniowane wymagania dla wszystkich klas urządzeń w **instrukcje warunkowe (`if/else`)**, dzięki czemu będzie można określić klasę, do której należy przypisać bieżące urządzenie.
- ☐ **Napiszemy prostą stronę testową** (podobną do tej z eksploratora WURFL), aby sprawdzić, jak różne łańcuchy user-agent są przydzielane do poszczególnych klas urządzeń.

Zapoznajemy się z funkcją dopasowującą

Funkcja dopasowująca czeka już na Ciebie w pliku *device_classes.php*. Skorzystamy z niej podczas tworzenia poszczególnych testów dla każdej z klas urządzeń.

Jednak zanim zaczniemy jej używać, chcemy ją pokrótce opisać. Uwaga, nadchodzi PHP!

Kod PHP gotowy do użycia

Pożądana wartość

Operator porównania

Nazwa możliwości do przetestowania

Sprawdzamy, czy istnieje obiekt urządzenia.

Pobieramy wartość możliwości z obiektu urządzenia.

Bieżącą wartość odczytaną z obiektu urządzenia porównujemy z pożądaną wartością.

```
function device_match($capability, $comparison, $value) {
    global $device;
    if (!$device) {
        return FALSE;
    }
    $device_value = $device->getCapability($capability);
    switch ($comparison) {
        case '==':
        case '===':
            return ($device_value === $value);
        case '!=':
        case '!==':
            return ($device_value !== $value);
        case '>=':
            return ($device_value >= $value);
        case '<=':
            return ($device_value <= $value);
        case '>':
            return ($device_value > $value);
        case '<':
            return ($device_value < $value);
        case 'LIKE':
            return (strpos($device_value, $value) !== FALSE);
        case 'NOT LIKE':
            return (strpos($device_value, $value) === FALSE);
        default:
            return FALSE;
    }
}
```

Ta funkcja zwraca boolowską wartość TRUE albo FALSE odpowiadającą wynikowi przeprowadzonego testu.

device_classes.php

O co chodzi w tej instrukcji switch?

Zmienna $device_value przechowuje wartość możliwości, która jest testowana dla bieżącego urządzenia.

```
switch ($comparison) {
    case '==':
    case '===':
        return ($device_value === $value);
    case '!=':
    case '!==':
        return ($device_value !== $value);
    case '>=':
        return ($device_value >= $value);
    case '<=':
        return ($device_value <= $value);
    case '>':
        return ($device_value > $value);
    case '<':
        return ($device_value < $value);
    case 'LIKE':
        return (strpos($device_value, $value) !== FALSE);
    case 'NOT LIKE':
        return (strpos($device_value, $value) === FALSE);
    default:
        return FALSE;
}
```

Zmienna $value przechowuje pożądaną wartość.

W zależności od przekazanego operatora porównania wartość jest porównywana na różne sposoby.

Operatory LIKE oraz NOT LIKE umożliwiają wykonywanie porównań cząstkowych (na fragmentach łańcuchów).

Jeśli zmienna $comparison nie zawiera jednego ze zdefiniowanych operatorów, zwracamy FALSE.

Kilka przykładów

Ten test przechodzi (funkcja zwraca TRUE), jeśli dla bieżącego urządzenia wartość resolution_width jest mniejsza niż lub równa 240 pikseli.

```
device_match('resolution_width', '<=', '240');
```

Ten test przechodzi, jeśli wartość is_wireless_device dla bieżącego urządzenia jest równa false.

```
device_match('is_wireless_device', '===', 'false');
```

Ten test przechodzi, jeśli wartość mobile_browser zawiera łańcuch 'Safari'.

```
device_match('mobile_browser', 'LIKE', 'Safari');
```

Używamy funkcji dopasowującej do testowania możliwości

Mamy już funkcję sprawdzającą możliwości. Czas na przekształcenie zdefiniowanych wymagań na testy.

- [x] **Napiszemy funkcję (PHP)** testującą poszczególne możliwości urządzenia umieszczone w definicjach klas urządzeń.
- [ ] Przekształcimy zdefiniowane wymagania dla wszystkich klas urządzeń w **instrukcje warunkowe** (`if/else`), dzięki czemu będzie można określić klasę, do której należy przypisać bieżące urządzenie.
- [ ] **Napiszemy prostą stronę testową** (podobną do tej z eksploratora WURFL), aby sprawdzić, jak różne łańcuchy user-agent są przydzielane do poszczególnych klas urządzeń.

Dla przypomnienia: zdefiniowaliśmy pięć klas urządzeń: `desktop`, `tablet`, `higher_mobile`, `simpler_mobile` oraz `unsupported`. W ćwiczeniu z puzzlami ze strony 188 określiliśmy, co trzeba testować dla każdej z nich.

Co prawda funkcje dopasowujące napisaliśmy za Ciebie, ale Ty też się musisz czymś zająć. Do których klas urządzeń należą te testy?

Ćwiczenie

.....
```
if (device_match('resolution_width', '>=', '320'))
```

.....
```
if (device_match('resolution_width', '<', '176'))
```

.....
```
if (device_match('is_wireless_device', '===', 'false'))
```

.....
```
if (device_match('mobile_browser', 'LIKE', 'Safari') ||
    device_match('mobile_browser', 'LIKE', 'Android'))
```

.....
```
if (device_match('cookie_support', '===', 'false'))
```

.....
```
if (device_match('is_tablet', '===', 'true'))
```

.....
```
if (device_match('ajax_manipulate_dom', '===', 'false'))
```

.....
```
if (device_match('https_support', '===', 'false'))
```

.....
```
if (device_match('resolution_width', '<', '320'))
```

Testy są już przyporządkowane do klas! Teraz kolej na przekształcenie ich w grupy zaimplementowane w PHP. Czytaj dalej!

Rozwiązanie ćwiczenia

Może się wydawać, że ten test mógłby być przeprowadzany dla klasy desktop, ale przeglądarki desktopowe wykryjemy szybciej za pomocą możliwości is_wireless_device.

higher_mobile	`if (device_match('resolution_width', '>=', '320'))`
unsupported	`if (device_match('resolution_width', '<', '176'))`
desktop	`if (device_match('is_wireless_device', '===', 'false'))`
higher_mobile	`if (device_match('mobile_browser', 'LIKE', 'Safari') ||` `  device_match('mobile_browser', 'LIKE', 'Android'))`
unsupported	`if (device_match('cookie_support', '===', 'false'))`
tablet	`if (device_match('is_tablet', '===', 'true'))`
unsupported	`if (device_match('ajax_manipulate_dom', '===', 'false'))`
unsupported	`if (device_match('https_support', '===', 'false'))`
simpler_mobile	`if (device_match('resolution_width', '<', '320'))`

Czas poskładać kod...

Zaostrz ołówek

Jak się czujesz w PHP? Spróbuj uzupełnić puste miejsca odpowiednimi fragmentami kodu. To są zgrupowane testy przyporządkowujące urządzenie do odpowiedniej klasy.

```
$device_class = NULL;

if (device_match('is_wireless_device', '===', '...............')) {
   $device_class = 'desktop';
}
else if (device_match('https_support', '...............', '...............') ||
          device_match('..............._support', '...............', '...............') ||
          device_match('ajax_manipulate_dom', '===', 'false') ||
          device_match('...............', '...............', '176')) {
   $device_class = '...............';
}
else if (device_match('is_tablet', '===', '...............')) {
   $device_class = '...............';
}
else if (device_match('is_wireless_device', '===', 'true') &&
          device_match('resolution_width', '>=', '...............') &&
          (device_match('mobile_browser', '...............', 'Safari') ||
          device_match('...............', 'LIKE', 'Android') ))
{
   $device_class = '...............';
}
else if (device_match('is_wireless_device', '===', '...............') &&
          device_match('...............', '<', '...............')) {
   $device_class = 'simpler_mobile';
}
```

Zaostrz ołówek. Rozwiązanie

W porządku — testy już gotowe!

```
$device_class = NULL;

if (device_match('is_wireless_device', '===', 'false')) {
   $device_class = 'desktop';
}
else if (device_match('https_support', '===', 'false') ||
         device_match('cookie_support', '===', 'false') ||
         device_match('ajax_manipulate_dom', '===', 'false') ||
         device_match('resolution_width', '<', '176')) {
   $device_class = 'unsupported';
}
else if (device_match('is_tablet', '===', 'true')) {
   $device_class = 'tablet';
}
else if (device_match('is_wireless_device', '===', 'true') &&
         device_match('resolution_width', '>=', '320') &&
         (device_match('mobile_browser', 'LIKE', 'Safari') ||
         device_match('mobile_browser', 'LIKE', 'Android') ))
{
   $device_class = 'higher_mobile';
}
else if (device_match('is_wireless_device', '===', 'true') &&
         device_match('resolution_width', '<', '320')) {
   $device_class = 'simpler_mobile';
}
```

Po pierwsze i najważniejsze: czy to w ogóle jest urządzenie mobilne?

Jeśli nie są spełnione minimalne wymagania...

Czy to może jest tablet?

A czy przeglądarka i szerokość ekranu są w porządku?

A może urządzenie jest wystarczająco dobre, ale bez szału?

W porządku! Dwa punkty z trzech załatwione! Już prawie skończyliśmy!

- [x] **Napiszemy funkcję (PHP)** testującą poszczególne możliwości urządzenia umieszczone w definicjach klas urządzeń.
- [x] Przekształcimy zdefiniowane wymagania dla wszystkich klas urządzeń w **instrukcje warunkowe (`if/else`)**, dzięki czemu będzie można określić klasę, do której należy przypisać bieżące urządzenie.
- [ ] **Napiszemy prostą stronę testową** (podobną do tej z eksploratora WURFL), aby sprawdzić, jak różne łańcuchy user-agent są przydzielane do poszczególnych klas urządzeń.

Ostatnia prosta

Najwyższy czas zakończyć prace nad projektem. Potrzebna nam będzie strona z prostym formularzem umożliwiającym podanie łańcucha user-agent. Po kliknięciu przycisku pojawi się informacja o klasie, do której zostało przyporządkowane dane urządzenie. Ponieważ strona jest bardzo podobna do tej, którą stworzyliśmy na potrzeby eksploratora WURFL, przygotowaliśmy ją za Ciebie (hura!).

Kod tego prostego formularza znajduje się w pliku index.php.

Na stronie są wyświetlane identyfikator urządzenia oraz wyznaczona klasa urządzeń, do której został przypisany podany łańcuch user-agent.

index.php

Jazda próbna

1. **Umieść kod w pliku device_classes.php.**
W pliku pod funkcją `device_match()` umieść kod ze strony 196, a następnie zapisz plik.

2. **Wyświetl dokument index.php w przeglądarce.**
Domyślnie na stronie pojawią się informacje o klasie urządzeń odpowiadające bieżącej przeglądarce.

3. **Przeprowadź próby dla kilku różnych łańcuchów user-agent.**
W polu formularza wpisz przykładowe łańcuchy user-agent, by sprawdzić, do której klasy zostaną zaklasyfikowane.

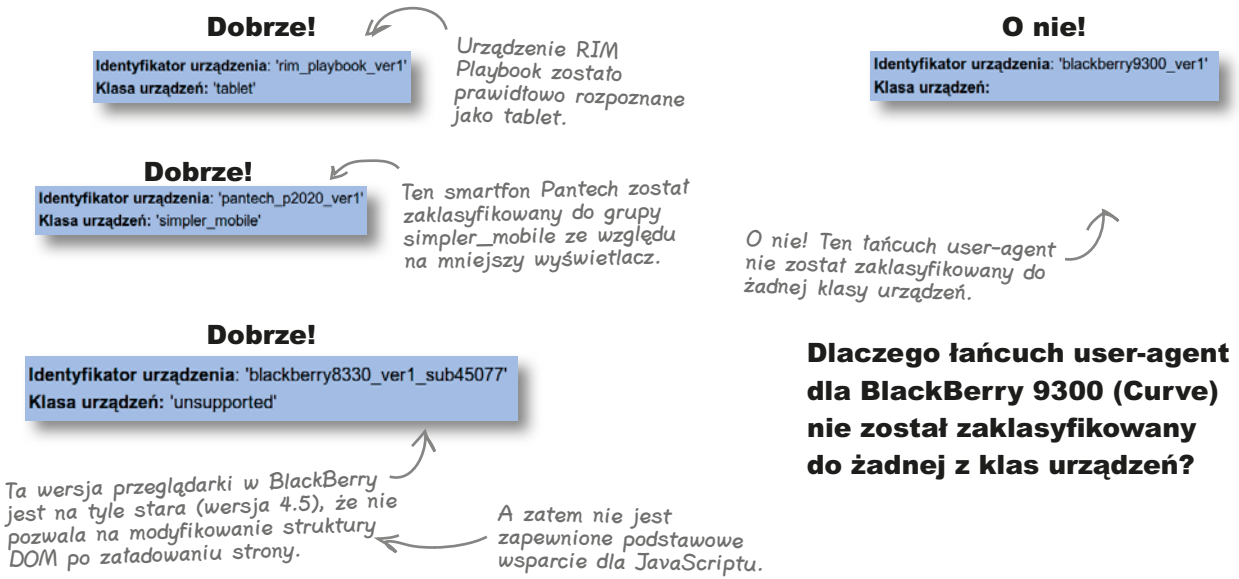
```
Mozilla/5.0 (PlayBook; U;
RIM Tablet OS 1.0.0; pl-PL)
AppleWebKit/534.8+ (KHTML,
like Gecko) Version/0.0.1
Safari/534.8+
```

```
PantechP2020/JIUS05172010R; Mozilla/5.0
(Profile/MIDP-2.0 Configuration/CLDC-
1.1; Opera Mini/att/4.2.19039; U;
pl-PL) Opera 9.50
```

```
BlackBerry8330/4.5.0.77 Profile/
MIDP-2.0 Configuration/CLDC-1.1
VendorID/105
```

```
BlackBerry9300/5.0.0.794
Profile/MIDP-2.1 Configuration/
CLDC-1.1 VendorID/245
```

No cóż, zobaczmy, jak to działa



Dlaczego łańcuch user-agent dla BlackBerry 9300 (Curve) nie został zaklasyfikowany do żadnej z klas urządzeń?

Wygląda na to, że coś jest nie tak

Problem w tym, że ten telefon ma ekran o szerokości równej 320 pikseli (czyli jest zbyt szeroki, by urządzenie zaklasyfikować do klasy `simpler_mobile`), ale nie ma przeglądarki opartej na silniku WebKit (czyli nie kwalifikuje się do klasy `higher_mobile`). Ten sam problem wystąpi w przypadku innych urządzeń, które mają duże ekrany, ale nie mają odpowiedniej przeglądarki.

Musimy to poprawić i sprawdzić, czy nie ma jeszcze innych luk w algorytmie przydzielającym do klas urządzeń.

Wypełniamy lukę

Mamy trzy możliwości poprawienia błędu pojawiającego się przy urządzeniach o dużych ekranach, ale bez przeglądarki opartej na silniku WebKit:

1. Możemy zdefiniować kolejną klasę urządzeń obsługującą taką sytuację.
2. Możemy zmienić klasę `higher_mobile`, tak by przyjmowała urządzenia bez przeglądarki WebKit. ← *Grupowanie pod kątem rozdzielczości.*
3. W klasie `simpler_mobile` możemy usunąć ograniczenie maksymalnej szerokości ekranu do 240 pikseli. ← *Grupowanie pod kątem przeglądarki.*

Każda z tych możliwości jest dobra. Musimy zdecydować, która z nich będzie najlepsza w przypadku witryny DaRadę!.

Przemek: Czy naprawdę musimy dodawać kolejną klasę urządzeń? Przecież to masa dodatkowej roboty...

Łukasz: Zgadzam się. Musimy znaleźć złoty środek między szczegółowością a liczbą klas urządzeń. Wydaje mi się, że teraz mamy w sam raz klas.

Przemek: No to co robimy?

Łukasz: Podsumujmy — problem polega na tym, że urządzenia o większych ekranach, ale bez przeglądarki opartej na silniku WebKit nie są przypisywane do żadnej ze zdefiniowanych klas.

Przemek: A czy nie są to aż dwa problemy? Przede wszystkim nie przewidzieliśmy takiej możliwości, więc nie wiemy, jaką treść należy dostarczyć tego typu urządzeniom. Ale jest jeszcze druga sprawa — nie zapewniliśmy czegoś w rodzaju domyślnej klasy urządzeń. Przecież zawsze się może zdarzyć, że coś pójdzie nie tak z detekcją urządzenia albo wydarzy się jeszcze coś innego, czego w ogóle nie przewidzieliśmy. Myślę, że potrzebujemy klasę „bezpieczeństwa" dla urządzeń mobilnych.

Łukasz: Masz rację. Zacznijmy jednak od pierwszego problemu. Jeżeli, jak ustaliliśmy, nie dodajemy kolejnej klasy urządzeń, musimy jakoś wypełnić tę lukę. Ale pojawia się pytanie o to, co jest ważniejsze: szerokość ekranu czy możliwości przeglądarki?

Przemek: Firma DaRadę! położyła duży nacisk na interaktywne elementy witryny, a z tego wniosek, że bardziej powinny się liczyć możliwości przeglądarki. Jednak z drugiej strony planowaliśmy, że urządzeniom z niewielkimi ekranami będziemy dostarczać pomniejszone obrazki.

Łukasz: Myślę, że masz rację co do priorytetów firmy DaRadę!. A co powiesz na taki kompromis — zmodyfikujemy klasy urządzeń w taki sposób, by w klasie `simpler_mobile` nie było wymagania dotyczącego szerokości ekranu?

Oznacza to, że niektóre urządzenia z szerszymi ekranami otrzymają uproszczone treści, ale — o ile dobrze pamiętam — zespół programistów pracujący nad interaktywnymi planszami korzysta z frameworku przystosowanego przede wszystkim do przeglądarek opartych na silniku WebKit. Tak sobie teraz pomyślałem, że właśnie dlatego testy dla klasy `higher_mobile` rozpoczęliśmy od sprawdzenia typu przeglądarki. Chyba zaczyna mnie zawodzić pamięć. Muszę się porządnie wyspać...

Przemek: A co z obrazkami?

Łukasz: Sądzę, że będzie dobrze, jeśli wszystkim mobilnym urządzeniom dostarczymy zoptymalizowane obrazki, których szerokość (ani wysokość) nie przekracza 320 pikseli. Następnie skorzystamy z jednej z technik RWD, aby uzyskać idealny wymiar dla danego urządzenia.

Wypełniamy luki w testach klas urządzeń

W kodzie przyporządkowującym urządzenia do klas wprowadzimy następujące zmiany:

Usuwamy to ograniczenie szerokości ekranu dla klasy simpler_mobile.

```
 $device_class = 'higher_mobile';
}
else if (device_match('is_wireless_device', '===','true') &&
         device_match('resolution_width', '<', '320')) {
   $device_class = 'simpler_mobile';
}
else {
   $device_class = 'desktop';
}
```

Ostrożnie: tutaj usuwamy tylko jeden nawias!

Dodajemy domyślną klasę.

device_classes.php

Jazda próbna

Wprowadź zmiany w pliku *device_classes.php* i jeszcze raz przetestuj łańcuchy user-agent ze strony 197.

Pięknie! Problematyczny łańcuch user-agent dla urządzenia BlackBerry został przydzielony do klasy simpler_mobile.

Zróbmy wreszcie użytek z klas urządzeń

Najwyższy czas, by skorzystać z klas, które zdefiniowaliśmy, i zrobić z nimi coś pożytecznego.

Nasz cel: dostarczyć różne odmiany głównej strony do użytkowników różnych urządzeń na podstawie zdefiniowanych klas urządzeń.

Użyjemy dotychczas napisanego kodu do przydzielania urządzeń do odpowiednich klas i dostarczenia na tej podstawie różnych odmian makiety głównej strony witryny firmy DaRadę!, zgodnie z założeniami przedstawionymi na stronie 184.

Skopiuj plik konfiguracyjny z ostatniego ćwiczenia do katalogu *adapt_content*. Tak, znowu to samo! Obiecujemy, że już ostatni raz.

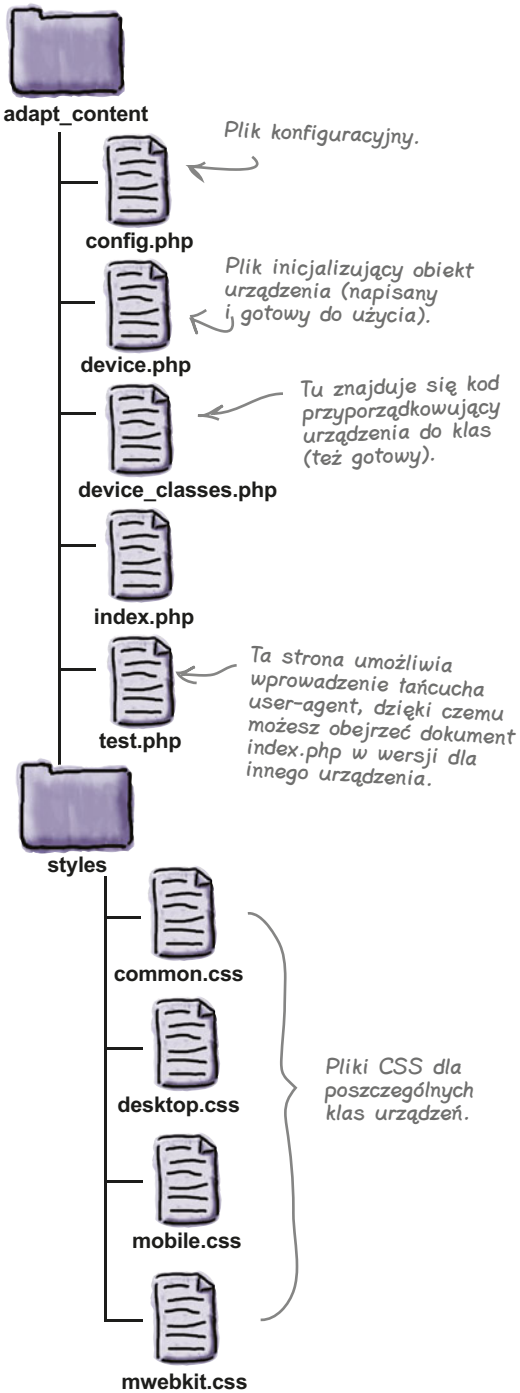
Kod HTML i CSS, z którym będziemy mieć do czynienia, jest całkiem prosty. Firma DaRadę! jest jeszcze we wstępnej fazie prac nad nową witryną, więc na razie pracujemy na podstawowej wersji makiety.

Niewspierane urządzenia traktujemy na razie podobnie do desktopowych

Niewspieranym urządzeniom (przypisanym do klasy `unsupported`) ma być dostarczana ta sama zawartość co urządzeniom stacjonarnym (z klasy `desktop`), ale bez odsyłaczy do sekcji z interaktywnymi planszami — i dotyczy to zarówno witryny, jak i sklepu internetowego.

Robi się już późno i wszyscy jesteśmy zmęczeni. Z tego względu, by zaoszczędzić trochę czasu, zrobiliśmy za Ciebie całkiem sporo.

Na kilku kolejnych stronach opisaliśmy, co trzeba zrobić, ale na końcu pokażemy Ci, skąd wziąć gotowy kod.



Zaostrz ołówek

Początek kodu dokumentu *index.php* jest jednakowy dla **wszystkich** wersji dostarczanej treści (czyli dla wszystkich klas urządzeń). Nadszedł czas, by sprawdzić, które fragmenty dokumentu mają zostać dostarczone urządzeniom z konkretnych klas, a które nie. W każdym z poniższych punktów uzupełnij lukę, wpisując nazwy klas odpowiadające fragmentom kodu znajdującym się z prawej strony.

1. Tej deklaracji typu DOCTYPE użyj w klasie urządzeń .....

2. Tego arkusza stylów użyj w klasach urządzeń: ....., ...

   oraz .....

3. Tego arkusza stylów użyj w klasach urządzeń: ..... i .....

4. Tego arkusza stylów użyj w klasie urządzeń .....

5. Tego bloku nawigacji użyj w klasach urządzeń: ....., ...

   oraz .....

6. Nie wyświetlaj tych dwóch odsyłaczy w klasie urządzeń .....

Poniżej znajduje się kod z pliku *index.php*. Podczas odpowiadania na pytania z lewej strony skorzystaj z zaznaczonych i ponumerowanych fragmentów.


index.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
1    "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
<!DOCTYPE html>
<html>
<head>
   <title>DaRadę! Pomoce naukowe dla studentów</title>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1" />
   <link rel="stylesheet" type="text/css" href="../assets/common.css" />
2  <link rel="stylesheet" type="text/css" href="../assets/desktop.css" />
3  <link rel="stylesheet" type="text/css" href="../assets/mobile.css" />
4  <link rel="stylesheet" type="text/css" href="../assets/mwebkit.css" />
</head>
<body>

<div id="header">
   <h1>DaRadę! Pomoce naukowe dla studentów</h1>
</div>
<div id="navigation">
5  <ul>
      <li><a href="#">Strona główna</a></li>
      <li><a href="#">Sklep internetowy</a></li>
      <li><a href="#">Do nauki!</a></li>
      <li><a href="#">Kontakt</a></li>
      <li><a href="#">FAQ</a></li>
   </ul>
</div>
```

Cii... Sekcja Do nauki! zawiera interaktywne plansze oparte na skryptach JavaScript.

6

Ciąg dalszy na kolejnej stronie.

Zaostrz ołówek

Myślałeś, że już skończyłeś? Jesteś dopiero w połowie zadania!

7. Flashową zawartość dostarcz w klasie urządzeń .....

8. Nie używaj Flasha w klasach urządzeń: ....., ..., ... i .....

9. Nie wyświetlaj tych sekcji w klasach urządzeń: ..... i .....

10. Tę wersję nawigacji zastosuj w klasach urządzeń: ..... i .....

11. Ukryj odsyłacz do interaktywnych plansz w klasie urządzeń .....



index.php

```
<div id="intro">
<p>Morbi non erat non ipsum pharetra tempus. Donec orci. Proin in ante.
Pellentesque sit amet purus. Cras egestas diam sed ante. Etiam imperdiet
urna sit amet risus...</p>
</div>
<div id="feature">
   <div id="featured_product">
7  <p>Slajdy we Flashu prezentujące promowane produkty.</p>
8  <p>Statyczne obrazki prezentujące promowane produkty.</p>
   </div>
</div>
<div id="ads">
9     ...
</div>
<div id="fromtheblog">
   ...
</div>
<div id="navigation">
   <ul>
10       <li><a href="#">Strona główna</a></li>
         <li><a href="#">Sklep internetowy</a></li>
         <li><a href="#">Do nauki!</a></li>
         <li><a href="#">Blog</a></li>
         <li><a href="#">Kontakt</a></li>
         <li><a href="#">FAQ</a></li>
   </ul>
</div>
<div id="footer">
   <p>Bieżąca klasa urządzeń: <?php print $device_class; ?></p>
</div>
</body>
</html>
```

To są teksty zastępcze — w tych miejscach znajdą się flashowe filmy i statyczne obrazki.

Aby zaoszczędzić miejsce na stronie, pominęliśmy dłuższe fragmenty tekstu.

11

Zaostrz ołówek. Rozwiązanie

Przyjrzyj się różnicom w kodzie pomiędzy poszczególnymi klasami urządzeń. Przyszedł czas na przekształcenie ustalonych reguł w działający kod.

1. Tej deklaracji typu DOCTYPE użyj w klasie urządzeń ..... *simpler_mobile* .....

Typu dokumentu XHTML-Basic używamy w klasie simpler_mobile. W pozostałych klasach stosujemy HTML5.

```
<?php if($device_class = = 'simpler_mobile'): ?>
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN" "http://www.w3.org/TR/
xhtml-basic/xhtml-basic11.dtd">
<?php else: ?>
   <!DOCTYPE html>
<?php endif; ?>
```

2. Tego arkusza stylów użyj w klasach urządzeń: ..... *desktop* ....., ... *tablet* .....
   oraz ..... *unsupported* .....

Pamiętaj, że klasę unsupported traktujemy podobnie jak desktop.

3. Tego arkusza stylów użyj w klasach urządzeń: ..... *simpler_mobile* ..... i ..... *higher_mobile* .....

4. Tego arkusza stylów użyj w klasie urządzeń ..... *higher_mobile* .....

Wszystkie klasy korzystają z arkusza common.css.

```
<link rel="stylesheet" type="text/css" href="../assets/common.css" />
<?php if ($device_class == 'desktop'
2       || $device_class == 'tablet'
        || $device_class == 'unsupported'): ?>
   <link rel="stylesheet" type="text/css" href="../assets/desktop.css" />
<?php endif; ?>
<?php if ($device_class == 'higher_mobile'
3       || $device_class == 'simpler_mobile'): ?>
   <link rel="stylesheet" type="text/css" href="../assets/mobile.css" />
<?php endif; ?>
<?php if ($device_class == 'higher_mobile'): ?>
4  <link rel="stylesheet" type="text/css" href="../assets/mwebkit.css" />
<?php endif; ?>
```

W tym arkuszu znajdują się elementy wspólne dla tych dwóch klas urządzeń mobilnych.

W tym arkuszu znajdują się definicje gradientów i innych bajerów (zapisane zgodnie ze specyfiką przeglądarek opartych na silniku WebKit).

Ta nawigacja jest umieszczona na górze strony (w stylu desktopowym).

5 Tego bloku nawigacji użyj w klasach urządzeń: *desktop*, *tablet* oraz *unsupported*

6 Nie wyświetlaj tych dwóch odsyłaczy w klasie urządzeń *unsupported*

```
<?php if ($device_class == 'desktop'
      || $device_class == 'tablet'
      || $device_class == 'unsupported'): ?>
   <div id="navigation">
      <ul>
         <li><a href="#">Strona główna</a></li>
         <?php if ($device_class != 'unsupported'): ?>
            <li><a href="#">Sklep internetowy</a></li>
            <li><a href="#">Do nauki!</a></li>
         <?php endif; ?>
         <li><a href="#">Kontakt</a></li>
         <li><a href="#">FAQ</a></li>
      </ul>
   </div>
<?php endif; ?>
```

5 6

Sklep internetowy i interaktywne plansze nie są dostępne dla urządzeń z klasy unsupported.

7 Flashową zawartość dostarcz w klasie urządzeń *desktop*

Flash i urządzenia mobilne nie przepadają za sobą. To samo dotyczy tabletów.

8 Nie używaj Flasha w klasach urządzeń: *tablet*, *higher_mobile*, *simpler_mobile* i *unsupported*

```
<?php if ($device_class == 'desktop'): ?>
   <p>Slajdy we Flashu prezentujące promowane produkty.</p>
<?php else: ?>
   <p>Statyczne obrazki prezentujące promowane produkty.</p>
<?php endif; ?>
```

7 8

Ciąg dalszy na kolejnej stronie.

Zaostrz ołówek. Rozwiązanie

W mobilnym układzie nie wyświetlamy reklam, a zamiast fragmentów wpisów z bloga umieszczamy odsyłacz do strony bloga.

9 Nie wyświetlaj tych sekcji w klasach urządzeń: *higher_mobile* i *simpler_mobile*

```
<?php if ($device_class == 'desktop'
      || $device_class == 'tablet'): ?>
9  <div id="ads">
       ...
   </div>
   <div id="fromtheblog">
       ...
   </div>
<?php endif; ?>
```

To jest mobilna wersja nawigacji (umieszczona na dole strony).

10 Tę wersję nawigacji zastosuj w klasach urządzeń: *higher_mobile* i *simpler_mobile*

Na telefonach nie ma dostępu do interaktywnych plansz!

11 Ukryj odsyłacz do interaktywnych plansz w klasie urządzeń *simpler_mobile*

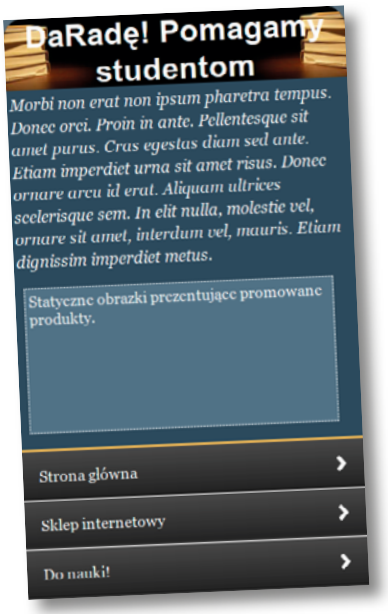
```
<?php if ($device_class == 'higher_mobile'
      || $device_class == 'simpler_mobile'): ?>
10  <div id="navigation">
      <ul>
         <li><a href="#">Strona główna</a></li>
         <li><a href="#">Sklep internetowy</a></li>
         <?php if ($device_class == 'higher_mobile'): ?> 11
            <li><a href="#">Do nauki!</a></li>
         <?php endif; ?>
         <li><a href="#">Blog</a></li>
         <li><a href="#">Kontakt</a></li>
         <li><a href="#">FAQ</a></li>
      </ul>
   </div>
<?php endif; ?>
```

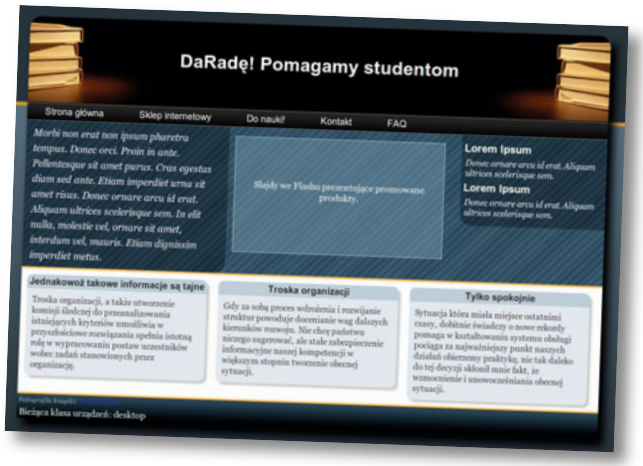
Tylko w klasie higher_mobile dajemy odsyłacz do plansz.

Jazda próbna

Sprawdź, czy to działa. Ukończona wersja pliku *index.php* znajduje się w *index_solution.php*. Zamień bieżący plik *index.php* na *index_solution.php*.

W przeglądarce otwórz stronę *test.php* i przetestuj kilka łańcuchów user-agent z pliku *useful_user_agents.txt*, by sprawdzić różne wersje klas urządzeń. Zobacz, jak wygląda strona *index.php* w przeglądarce desktopowej.





WYTĘŻ UMYSŁ

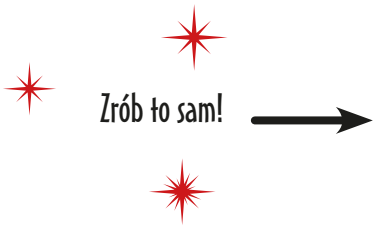
Co prawda poprawiliśmy odkryte błędy w klasach urządzeń (patrz strona 200), ale może przyszło Ci do głowy, co jeszcze może powodować problemy?

Obchodź się z tym ostrożnie i bądź przygotowany

W rozdziale 3. zwróciliśmy uwagę na niektóre problemy związane z detekcją urządzeń po stronie serwera z wykorzystaniem łańcuchów user-agent. Istnieją pewne niebezpieczeństwa, o których nie możesz zapominać:

1. **Nie zawsze przeglądarki przesyłają właściwe łańcuchy user-agent.** Niekiedy użytkownicy celowo zmieniają łańcuch user-agent, a czasem to sama przeglądarka wysyła dziwny łańcuch. W takich przypadkach baza danych urządzeń może zwrócić nieodpowiednie dane.

2. **WURFL nie zawsze potrafi znaleźć (konkretne) urządzenie.** W niektórych przypadkach nie udaje się odnaleźć konkretnego urządzenia, a czasem w ogóle nie dochodzi do dopasowania. W rezultacie możesz uzyskać ogólny identyfikator urządzenia (`generic`) lub pusty łańcuch.



Otwórz na chwilę stronę eksploratora, którą opracowaliśmy wcześniej w tym rozdziale, i sprawdź, jaki będzie wynik dla łańcucha `facebookexternalhit/1.1 (+http://www.facebook.com/externalhit_uatext.php)`.

Identyfikator generic jest naprawdę ogólny

Jeśli sprawdziłeś w eksploratorze łańcuch user-agent dla `facebookexternalhit` (jeżeli jeszcze tego nie zrobiłeś, zrób to teraz!), otrzymałeś informację, że urządzenie ma identyfikator `generic`. Jednym słowem, WURFL wzruszył ramionami i powiedział: „Słuchaj, naprawdę się starałem, ale nie mam pojęcia, co to za jeden”.

I co z tym zrobić?

Wartości możliwości przypisane ogólnemu urządzeniu `generic` nie są na tyle konkretne, by móc na ich podstawie podejmować jakiekolwiek decyzje. Nie jest raczej możliwe, by — na przykład — dla klienta `facebookexternalhit` szerokość ekranu (wartość `resolution_width`) wynosiła 90 pikseli. Po prostu taka wartość została określona w definicji urządzenia `generic`.

Musimy bardziej zadbać o bezpieczeństwo

Podczas projektowania witryny, w której została zastosowana detekcja urządzeń po stronie serwera, musisz poświęcić chwilę na przemyślenie tego, jak witryna powinna się zachować, gdy jest zwracany ogólny identyfikator lub urządzenie nie zostało w ogóle określone. *Podpowiedź: witryna nadal powinna działać.*

Przypadkowy sukces to za mało

W tej chwili w witrynie DaRadę! łańcuch user-agent, który powoduje zwrócenie identyfikatora `generic`, zostaje zaklasyfikowany do klasy `unsupported` (sprawdź to sam, podając łańcuch dla `facebookexternalhit`). To nie jest niewłaściwe zachowanie, ale przypomina raczej zdanie się na ślepy los.

display	
physical_screen_height	27
columns	11
dual_orientation	false
physical_screen_width	27
rows	6
max_image_width	90
resolution_height	40
resolution_width	90
max_image_height	

Szczegółowe informacje ze strony eksploratora. Wartości z grupy display są bardzo ogólne — na pewno nie powinniśmy na nich polegać.

Dane WURFL mogą się różnić.

Porównajmy możliwości określane dla łańcucha `facebookexternalhit` *przez nasz eksplorator z tymi, które podaje eksplorator ScientiaMobile.*

Przejdź na stronę http://www.tera-wurfl.com/explore i wpisz łańcuch `facebookexternalhit`*. Pierwsze, co rzuca się w oczy, to to, że wariant Database Edition jako* **ogólny** *identyfikator podaje* `generic_web_browser`.

Przyjrzyj się szczegółowym informacjom zwracanym przez ten eksplorator, a dojrzysz jeszcze więcej różnic. Spójrz na przykład na grupę `display` *i porównaj wyniki z tymi, które uzyskaliśmy w naszym eksploratorze dla urządzenia* `generic`.

Różnią się, prawda? Dobrze jest zapoznać się z różnymi odmianami WURFL (API dla PHP czy oparty na plikach). Dobrze też jest mieć własną stronę eksploratora.

Szczegółowe informacje ze strony eksploratora ScientiaMobile dla łańcucha user-agent facebookexternalhit.

display	
physical_screen_height	400
columns	120
dual_orientation	false
physical_screen_width	400
rows	200
max_image_width	600
resolution_height	600
resolution_width	800
max_image_height	

Lepiej zapobiegać, niż leczyć

API dla PHP dostarcza kilka metod umożliwiających określenie dokładności przeprowadzonego dopasowania. Rzućmy na nie okiem.

Jeśli nie ma żadnego identyfikatora...

```
if (!$device->id
    || (!$device->isSpecific()
     && $device->fallBack === 'root')) {
    /* PORAŻKA - upewnij się, że masz przygotowany plan awaryjny na taką sytuację */
}
```

...lub jeśli dopasowanie nie jest ściste oraz zostało uogólnione do podstawowego urządzenia.

Przesadzacie! Czemu nie mogę po prostu poprzestać na wykryciu konkretnego urządzenia?

Z dwóch powodów: dobrodziejstw sprawdzania błędów i łatki dla przeglądarek desktopowych.

Ze sprawdzaniem braku identyfikatora urządzenia jest jak z utrzymywaniem domu w czystości. Chcemy, żeby lśniło.

Łatka dla przeglądarek desktopowych, w której korzystamy z WURFL, pozwala nam zdobyć podstawowe informacje zarówno o przeglądarkach desktopowych, jak i mobilnych.

Jednak w przypadku wszystkich dopasowań dla przeglądarek desktopowych metoda `isSpecific()` zwraca wartość `FALSE`, ponieważ nie ma możliwości dokładnego określenia możliwości urządzenia. Sprawdzenie, czy dokonane uogólnienie dopasowania (właściwość `fallBack`) nie osiągnęło poziomu `root` (najbardziej ogólne), pozwala uniknąć potraktowania błędnie zidentyfikowanej przeglądarki desktopowej jako niepowodzenia.

Nie istnieją głupie pytania

P: Czym jest łatka dla przeglądarek desktopowych, o której mówicie?

O: Domyślna instalacja WURFL zawiera łatkę, która umożliwia ogólne zidentyfikowanie zbioru przeglądarek desktopowych oraz mobilnych.

P: Nie rozumiem, które części WURFL należą do API.

O: WURFL to tylko dane. Z kolei API to kod umożliwiający dostęp do tych danych. Dostępne są interfejsy API dla kilku głównych języków programowania.

P: Co to za łańcuch user-agent `facebookexternalhit`?

O: Gdy ktoś udostępnia jakiś odsyłacz na Facebooku, serwis często pobiera fragmenty treści lub tworzy obrazek z podglądem, i to właśnie w takich sytuacjach stosowany jest ten łańcuch user-agent.

P: Czy kod testujący odpowiadający za klasy urządzeń nie mógłby być odrobinę lepszy?

O: Oczywiście, że by mógł. W rzeczywistych projektach kod powinien być bardziej elegancki, lepiej zorganizowany i wydajniejszy. W omawianym przykładzie chcieliśmy tylko wyjaśnić Ci podstawowe zagadnienia.

P: Wybaczcie, ale jeśli się nie mylę, zmienna `$device` jest globalna. Przecież to okropne.

O: Patrz wyżej.

Ćwiczenie

Do skryptu *device_classes.php* z katalogu *adapt_content* dołącz kod odpowiedzialny za sprawdzanie ogólnego identyfikatora i błędnych dopasowań. Takie przypadki należy zaklasyfikować do klasy `unsupported`.

Podpowiedź: kod umieść w tym miejscu.

```
$device_class = NULL;

if (device_match('is_wireless_device', '===', 'false')) {
    $device_class = 'desktop';
}
```

device_classes.php

Załataliśmy już wszystko, co wymagało załatania! Gotowe!

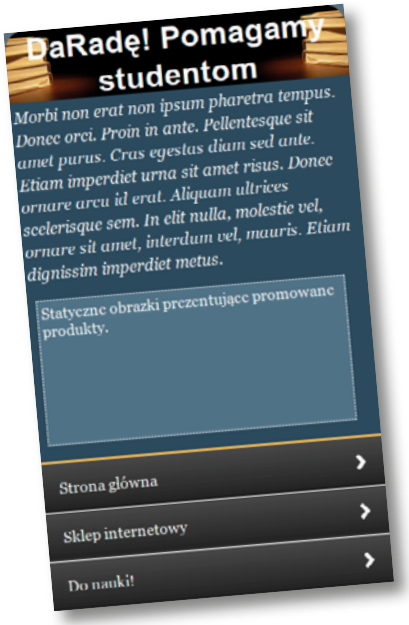
Rozwiązanie ćwiczenia

```
$device_class = NULL;
if (!$device->id ||
    (!$device->isSpecific() && $device->fallBack === 'root')) {
    $device_class = 'unsupported';
}
else if (device_match('is_wireless_device', '===', 'false')) {
    $device_class = 'desktop';
}
```

device_classes.php

Gratulacje! Jesteś prawdziwym mistrzem w poskramianiu możliwości i klas urządzeń.

To bardzo złożony problem, ale co to dla Ciebie! Za dowód może posłużyć witryna z dopasowującą się zawartością. Świetna robota!



Jak mam wybrać jedną, najlepszą metodę? Mamy detekcję po stronie serwera, ale jest niepewna ze względu na problemy z łańcuchami user-agent. Możemy też skorzystać z techniki Responsive Web Design i wykrywania możliwości po stronie klienta, ale nie wszystkie mobilne przeglądarki je obsługują...

Nie musisz się decydować na jedną metodę. Nie musisz porzucać RWD tylko dlatego, że korzystasz z detekcji po stronie serwera.

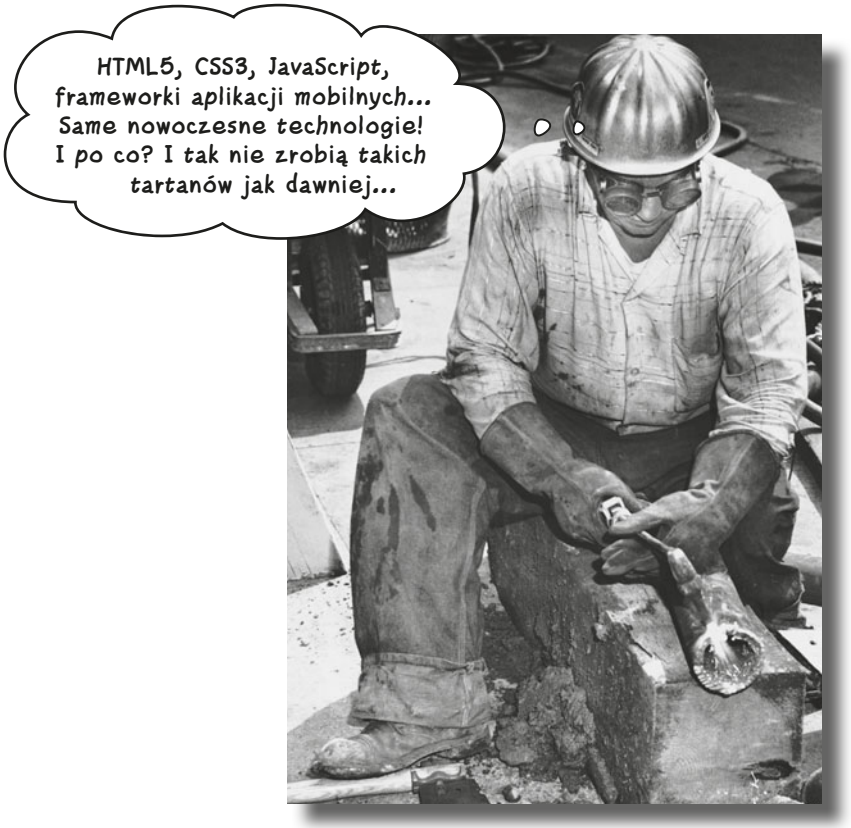
Czasem najlepsze efekty można uzyskać dzięki połączeniu tych dwóch podejść. Każde z nich ma swoje zalety, ale też wady. Połączenie ich pozwoli wydobyć z nich to, co najlepsze, ale o tym powiemy więcej w rozdziale 9.

CELNE SPOSTRZEŻENIA

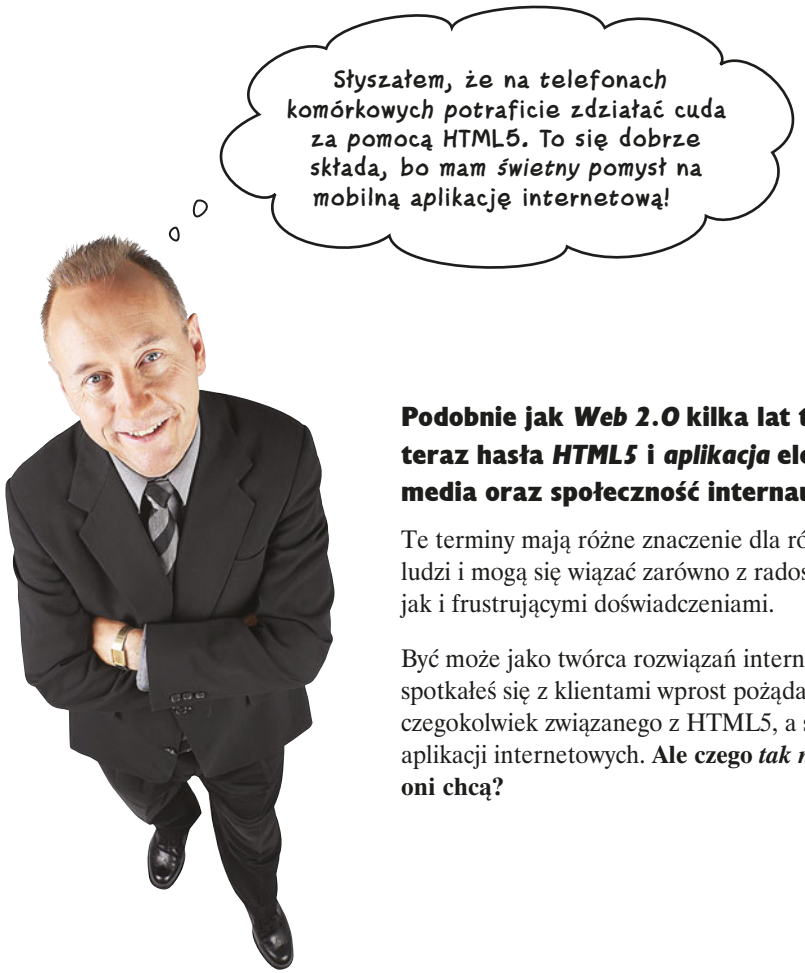
- **Repozytoria danych o urządzeniach**, takie jak **WURFL** (ang. *Wireless Universal Resource File*), dostarczają bardzo szczegółowe informacje na temat całego mnóstwa urządzeń.
- Dane WURFL obejmują ponad 500 możliwości podzielonych na dziesiątki grup (dla każdego urządzenia).
- Bazy danych urządzeń możemy użyć do określenia wartości przypisanej danej **możliwości**.
- **API WURFL dla PHP** to jeden z kilku interfejsów umożliwiających uzyskanie dostępu do danych WURFL. Poszczególne API zachowują się trochę inaczej i dostarczają dane w różnych formatach.
- Firma ScientiaMobile została założona w 2011 roku przez głównych twórców bazy WURFL. Firma dostarcza licencje na korzystanie z WURFL — zarówno typu open source, jak i komercyjne. Jeśli z jakichś powodów WURFL Ci nie odpowiada, możesz skorzystać z jednego z alternatywnych rozwiązań.
- Podczas pracy nad większym projektem może się okazać pożądane **pogrupowanie wybranych możliwości w klasy urządzeń**.
- Klasy urządzeń to abstrakcyjne grupy urządzeń powiązanych **wspólnymi możliwościami**.
- Dzięki zaklasyfikowaniu urządzenia do klasy urządzeń możemy wykonać pewne operacje (np. dostarczyć dopasowaną treść) bez potrzeby ciągłego śledzenia poszczególnych możliwości.
- Podczas korzystania przy identyfikacji urządzeń z baz danych bardzo ważne jest, by stworzyć **domyślną klasę urządzeń, zapewnić sprawdzanie błędów i obsługę ogólnego urządzenia**.
- Podobnie jak większość internetowych rozwiązań, tak i detekcja urządzeń po stronie serwera nie jest w stu procentach pewną metodą.
- **Metody detekcji urządzeń po stronie serwera i dopasowywania zawartości można łączyć z technikami realizowanymi po stronie klienta**, ponieważ te dwa podejścia nie wykluczają się wzajemnie. Rozwiniemy ten temat w dalszej części książki.

6. Framework dla mobilnych aplikacji internetowych

Tartanator



„My chcemy aplikację!”. Jeszcze rok czy dwa lata temu tego typu hasło wiązało się nieodłącznie z jednym — tworzeniem natywnych aplikacji dla każdego urządzenia, które zamierzaliśmy wesprzeć. Na szczęście teraz nie jest to jedyne możliwe rozwiązanie. Aplikacje internetowe dla mobilnych przeglądarek są coraz doskonalsze, zwłaszcza ostatnio, kiedy wkroczył **HTML5** wraz z nieodłącznymi kompanami — **CSS3** i **JavaScriptem**. W świat mobilnych aplikacji internetowych wejdziemy wraz z **mobilnym frameworkiem**, czyli zbiorem gotowych rozwiązań programistycznych upraszczających i przyspieszających tworzenie aplikacji.



Podobnie jak *Web 2.0* kilka lat temu, tak teraz hasła *HTML5* i *aplikacja* elektryzują media oraz społeczność internautów.

Te terminy mają różne znaczenie dla różnych ludzi i mogą się wiązać zarówno z radosnymi, jak i frustrującymi doświadczeniami.

Być może jako twórca rozwiązań internetowych spotkałeś się z klientami wprost pożądającymi czegokolwiek związanego z HTML5, a szczególnie aplikacji internetowych. **Ale czego *tak naprawdę* oni chcą?**

HTML5, aplikacja internetowa... Co te słowa znaczą?

HTML5 to konkretna technologia...

HTML5 *jest* konkretną technologią. To stale rozwijany standard, który wyewoluował z HTML-a — języka, który znamy i kochamy, bo bez niego nie byłoby internetu. Standard HTML5 oczyszcza i wzbogaca technologię funkcjonującą już ponad dwie dekady, umożliwiając między innymi tworzenie aplikacji internetowych bez wystawiania wstecznej kompatybilności na zbytnie ryzyko.

W HTML5 zostało wprowadzonych kilka elementów semantycznych, takich jak choćby `<section>`, `<article>`, `<nav>` czy `<header>`. Składnia niektórych znaczników została uproszczona, dostaliśmy do dyspozycji znaczniki do obsługi mediów: `<audio>` i `<video>`, a także wprost nieograniczone możliwości wprowadzania interaktywności poprzez skrypty i interfejsy API JavaScriptu (takie jak geolokalizacja czy lokalne składowanie danych).

...ale reprezentuje znacznie więcej

Przeważnie ludzie, mówiąc o HTML5, mają na myśli połączenie samego języka HTML z dobrodziejstwami superwydajnego JavaScriptu i bogatego CSS3. Technologie te tworzą drużynę wprost stworzoną do budowania nowoczesnych, interaktywnych aplikacji internetowych.

Niektórzy sądzą nawet, że to najwspanialsza rzecz, z którą mieli kiedykolwiek do czynienia.

No cóż, to może być trochę mylące...

A czym tak naprawdę jest aplikacja internetowa?

Jak rozumieć hasło *aplikacja internetowa* w zwrocie *aplikacja internetowa HTML5*? Już samo wyjaśnienie znaczenia terminu *HTML5* jest wystarczająco skomplikowane, a tu jeszcze mowa o jakichś ***aplikacjach***. Wkraczamy na terytorium mrocznego świata, pełnego smoków i tajemnic.

Przeważnie wystarczy rzut oka na stronę w przeglądarce, by wyczuć, czy mamy do czynienia z czymś, co zachowuje się jak aplikacja, czy nie. Skupienie uwagi na wykonywaniu zadań, układ strony mieszczący się w jednym oknie, akcje niewymagające przeładowania całej strony, interaktywność — to propozycje kryteriów pozwalających na odróżnienie aplikacji od klasycznej witryny internetowej skupionej na treści.

Jednym słowem, jeszcze nikt nie zdefiniował słowa *aplikacja* w sposób, który zadowoliłby wszystkich.

Trudno zdefiniować, czym są aplikacje internetowe, ale mają one kilka cech związanych z interaktywnością, do których HTML5 i powiązane z nim technologie wydają się stworzone.

Jak się zachowują klasyczne witryny internetowe?

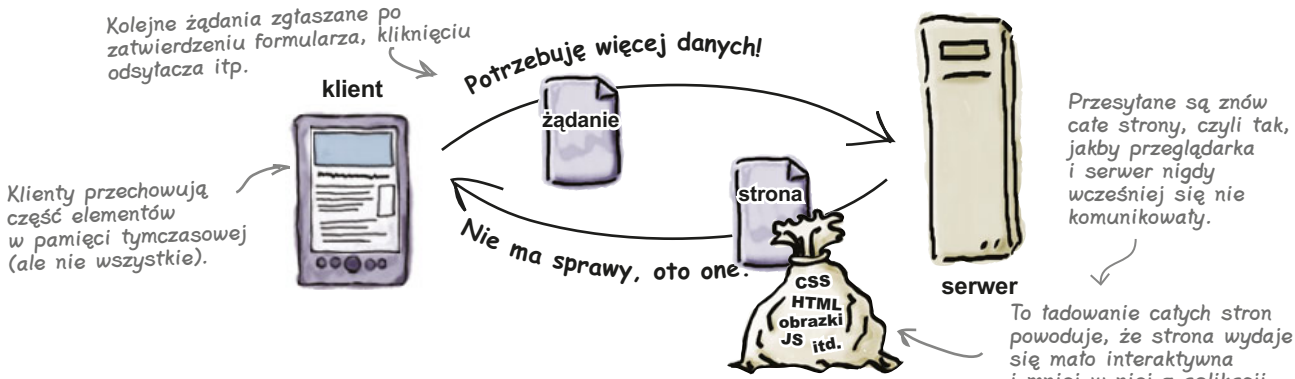
W tradycyjnym modelu witryn internetowych żądanie strony HTML powoduje przesłanie przez serwer wszystkich składników tej strony: dokumentu HTML, skryptów JavaScript, arkuszy CSS, obrazów itd.

Przeglądarki zwykle dobrze sobie radzą z przechowywaniem elementów w pamięci tymczasowej, a serwery można tak skonfigurować, by również składowały część danych w tego typu pamięci. Jednak bez względu na to klasyczny model zakłada, że każda interakcja ze stroną (przesłanie formularza, kliknięcie odsyłacza czy jakakolwiek inna tego typu operacja) wymaga przeładowania całej strony.

Pierwsze żądanie strony

Wadą takiego rozwiązania — zwłaszcza w przypadku urządzeń mobilnych o ograniczonej liczbie równoczesnych połączeń HTTP, małej przepustowości i niewielkiej mocy obliczeniowej — jest konieczność pobierania zasobów, które mogły już zostać wcześniej pobrane przez przeglądarkę, oraz wymuszenie na przeglądarce ponownego wyświetlenia zawartości strony oraz wykonania kodu, który nie uległ żadnej zmianie. Poza tym tego typu strona *wydaje się* mniej interaktywna.

Kolejne żądania



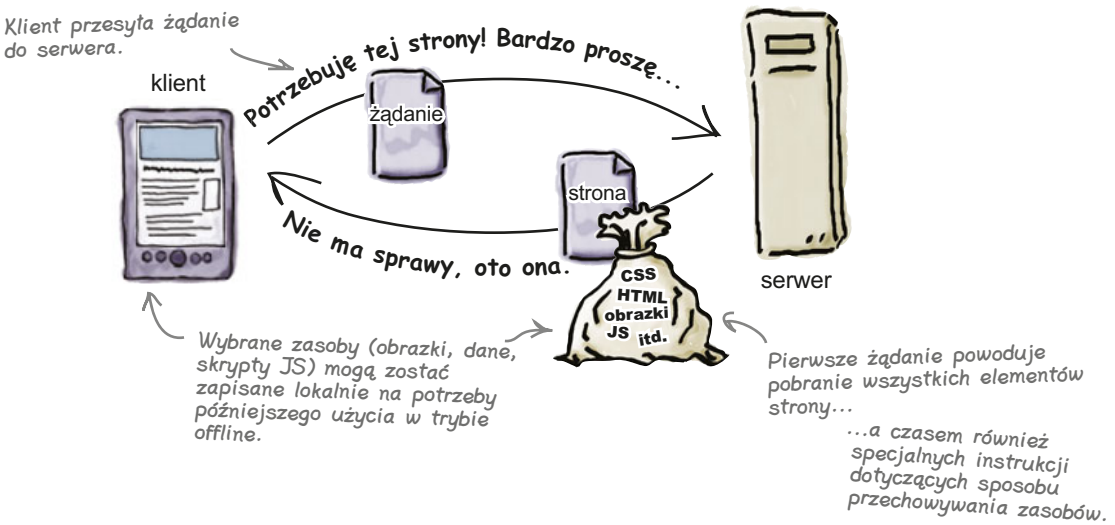
Jak się zachowują witryny przypominające aplikacje?

W modelu witryny internetowej przypominającym aplikację klient odgrywa znacznie większą rolę, a w odpowiedzi na żądania jest przesyłanych mniej zasobów. Dokumenty, kod i inne zasoby mogą być przechowywane lokalnie. Żądania dla zmienianej zawartości strony lub danych mogą być wykonywane asynchronicznie z wykorzystaniem AJAX-a i, co ważne, nie wymagają pełnego przeładowania strony.

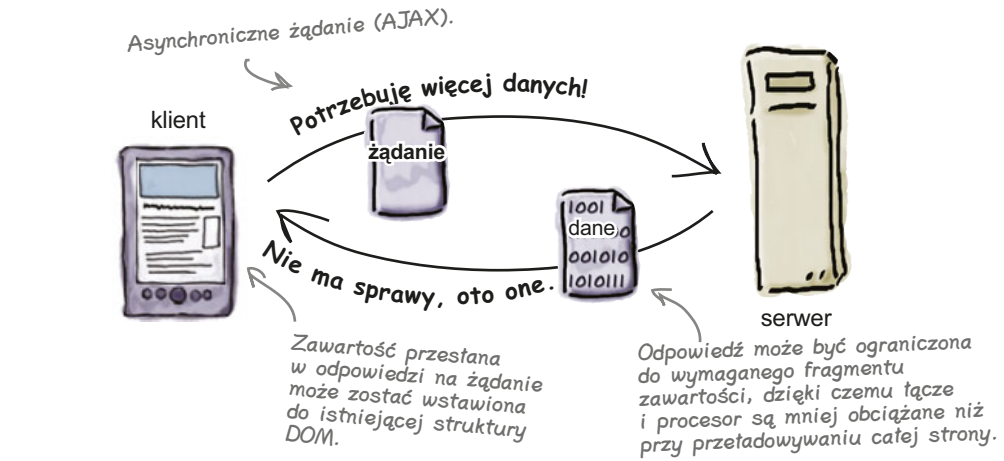
W większości za wzmożone wrażenia interaktywności odpowiadają właśnie te asynchroniczne żądania.

Dzięki wymianie zawartości tylko wybranych elementów w zamian za przeładowanie całej strony zmniejszamy obciążenie łącza i procesora, a przy okazji — z punktu widzenia użytkowników — strona staje się bardziej interaktywna.

Pierwsze żądanie strony

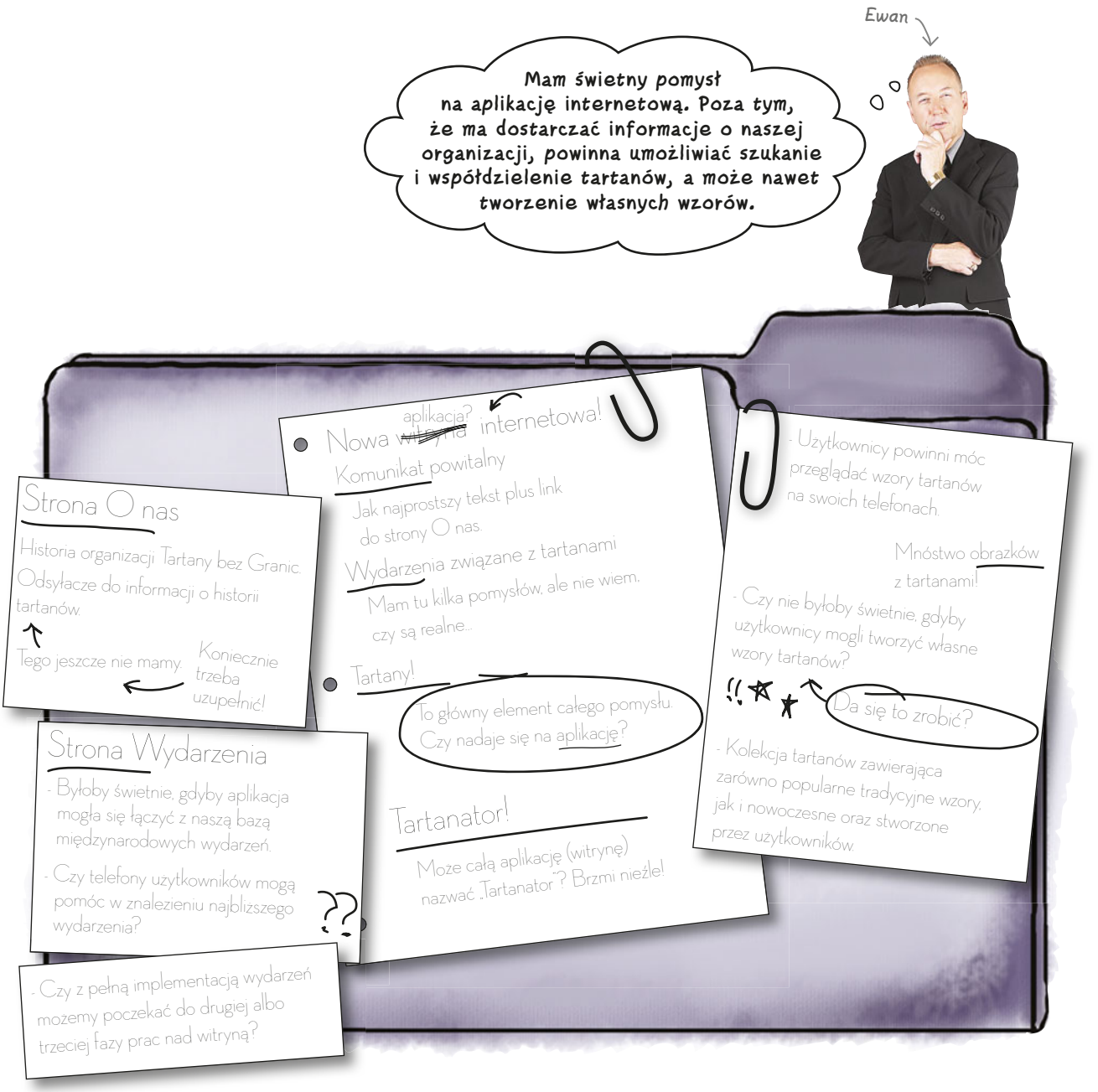


Kolejne żądania



Internetowa aplikacja mobilna HTML5

Tartany bez Granic to międzynarodowa organizacja starająca się popularyzować historię i kulturę szkockich tartanów na całym świecie.



Iza: Cześć, chłopaki! Jak widzę, wymagania nie są zbyt precyzyjne. Kiedy rozmawiałam z Ewanem, zrozumiałam, że są dwa główne wyzwania stojące przed tą witryną..., a raczej aplikacją czy jak to nazwać. Część ma prezentować **treści**, na przykład informacje o organizacji czy historię tartanów. Ale jest też odrębna sekcja, którą Ewan nazywa **Tartanatorem**...

Przemek: Tartanator? Naprawdę?! He, he...

Iza: Właśnie. Tak naprawdę to chce tak nazwać całą witrynę, a nie tylko tę sekcję. Tak czy inaczej, ma to być coś w rodzaju listy wzorów tartanów, którą można swobodnie przeglądać. Ewan ma też nadzieję, że uda się zrealizować jego plan polegający na tym, żeby użytkownicy mogli sami tworzyć wzory za pomocą formularza.

Łukasz: No ładnie! To wszystko wygląda dosyć dziwacznie, ale może być ciekawym wyzwaniem programistycznym.

Przemek: Po kolei — strony prezentujące treści, sekcja Tartanatora... A co ze stroną Wydarzenia?

Iza: Moment! Zaraz o tym powiem! Postanowiliśmy, że zrealizujemy ten projekt w dwóch fazach. W pierwszej fazie stworzymy podstawową strukturę dla stron prezentujących treści i zaimplementujemy listę tartanów. Ewan chciałby też, żebyśmy przemyśleli wygląd interfejsu użytkownika w narzędziu do tworzenia własnych wzorów. Najlepiej by było, gdybyśmy stworzyli prototyp frontendu.

Przemek: Dobrze. W takim razie mamy zaprojektować formularz do tworzenia wzorów tartanów, ale na razie nie ma nic robić?

Iza: Tak, coś w tym rodzaju. W drugiej fazie zajmiemy się konkretną implementacją tego narzędzia i wrócimy do strony Wydarzenia.

Łukasz: No to do roboty. Wygląda na to, że zabieramy się za tworzenie mobilnej aplikacji internetowej.

Plan pierwszej fazy projektu Tartanator

☐ **Zbudować strony prezentujące treści i strukturę witryny.**
Musimy stworzyć główne sekcje i strony oraz podstawową strukturę witryny.

To właśnie musimy zrobić!

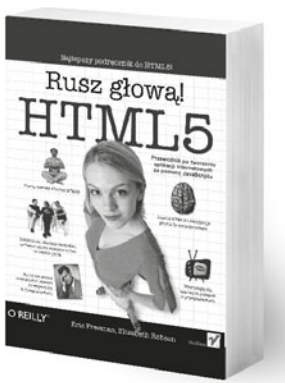
☐ **Stworzyć listę tartanów.**
W pierwszej fazie utworzymy listę istniejących popularnych wzorów tartanów. Listę powinno dać się łatwo przeglądać i — co oczywiste — powinna wyglądać i zachowywać się jak aplikacja, i to w dodatku mobilna.

☐ **Zaprojektować prototyp formularza służącego do tworzenia tartanów.**
Ewan chce, by użytkownicy mogli projektować własne wzory tartanów za pomocą interfejsu przypominającego aplikacje mobilne. Ponieważ chce jak najszybciej wiedzieć, jak to może wyglądać, dostarczymy mu prototyp.

Wygląda na to, że zbudujemy tę aplikację w HTML5 i powiązanych technologiach. Nie mam pojęcia, od czego zacząć. Czy muszę wszystko zbudować od podstaw?

No cóż, jasne, że można by tę aplikację zbudować od podstaw...

Jeśli chcesz się dowiedzieć, jak za pomocą technologii HTML, CSS i JavaScript tworzyć wspaniałe aplikacje internetowe, koniecznie sięgnij po książkę *HTML5. Rusz głową!*.



Koniecznie ją kup! Jest świetna! Najlepsza na rynku książka o programowaniu w HTML5. Naprawdę, nie pożałujesz!

Dobra, dobra. Wystarczy tej bezwstydnej reklamy! — red.

...ale możesz też użyć frameworku dla mobilnych aplikacji internetowych

W pierwszej fazie pracy nad Tartanatorem włączymy turbodoładowanie dzięki zastosowaniu frameworku dla mobilnych aplikacji internetowych.

Nadal będziemy używać HTML-a i CSS, ale framework pomoże nam szybciej zbudować mobilny interfejs użytkownika.

Po co używać frameworków?

Przyjrzyjmy się temu. Tworzenie skomplikowanych interaktywnych aplikacji internetowych (a zwłaszcza mobilnych aplikacji internetowych) od podstaw jest dosyć przerażającą wizją. Zastosowanie frameworku, czyli zestawu interaktywnych elementów i pomocniczych narzędzi, może nam pomóc wykonać pierwszy krok.

1. **Framework może nam pomóc sprawić, by witryna lub aplikacja internetowa wyglądała na mobilną.**
   Frameworki dla mobilnych aplikacji internetowych pomagają przede wszystkim tak zmodyfikować elementy HTML-a, by wyglądały i zachowywały się zgodnie z oczekiwaniami użytkowników urządzeń mobilnych. Dzięki temu możemy zaoszczędzić sporo czasu.

2. **Framework może nam pomóc sprawić, by witryna lub aplikacja internetowa zachowywała się jak mobilna.**
   Frameworki biorą na siebie sprawy związane z przejściami i innymi efektami, dzięki którym witryny lub aplikacje internetowe przypominają natywne aplikacje, a przynajmniej nie odróżniają się za bardzo od nich.

3. **Framework może nam pomóc poradzić sobie z problemami wynikającymi z niezgodności między różnymi platformami.**
   Twórcy frameworków biorą na siebie wszystkie nieznośne oraz uciążliwe dziwactwa różnych przeglądarek i w kodzie frameworków implementują obejścia tych problemów, dzięki czemu my nie musimy się tym zajmować.

W niektórych projektach być może nie będziesz chciał odkrywać na nowo typowych elementów interaktywnej aplikacji internetowej. To właśnie wtedy frameworki okazują się bardzo pomocne.

Obejrzyj to!

Frameworki — wybieraj je i używaj ich z rozwagą.

Frameworki to potężne narzędzia, ale z ich stosowaniem mogą się wiązać pewne problemy. Wiele z nich sporo waży, więc mogą zapchać łącze setkami kilobajtów. Niektóre mają w sobie wszystko, co tylko możliwe, czyli masę widżetów i efektownych animacji, które nie tylko powodują przesadny rozrost plików, ale również negatywnie wpływają na wydajność aplikacji, zwłaszcza tych uruchamianych na słabszych urządzeniach. Zawsze musisz sprawdzić, jakie urządzenia są wspierane przez dany framework. Zdarza się, że do obsługi niektórych wymagane są najnowsze platformy i najwydajniejsze urządzenia.

Dla projektu Tartanator wybraliśmy framework jQuery Mobile

Podczas tworzenia Tartanatora skorzystamy z frameworku **jQuery Mobile**.

Jest to framework dla interfejsu użytkownika zoptymalizowany pod kątem urządzeń mobilnych. Jest zbudowany na bazie niezwykle popularnej javascriptowej biblioteki jQuery.

W naszym projekcie zdecydowaliśmy się zastosować właśnie jQuery Mobile głównie ze względu na jego prostotę oraz architekturę silnie związaną z HTML5. Framework jQuery Mobile jest tak zaprojektowany, że bardzo łatwo (czasem wręcz niewidocznie!) wtapia się w dobrze sformatowany dokument HTML5.

Poza tym, jeśli już wcześniej korzystałeś z jQuery, z całą pewnością wiesz, jak intuicyjna i prosta jest ta biblioteka.

Framework jQuery Mobile (jQM), podobnie jak pozostałe projekty z rodziny jQuery, jest rozpowszechniany na zasadach open source.

Przecież są setki sposobów na tworzenie rozwiązań dla mobilnego internetu. Koncepcja Responsive Web Design. Oddzielne strony. Wykrywanie urządzeń i możliwości. Mam rozumieć, że zapominamy teraz o tym wszystkim i rozpoczynamy od nowa z frameworkami dla mobilnych aplikacji internetowych?

To wszystko jest trochę bardziej skomplikowane...

W świecie mobilnego internetu nie ma prostych odpowiedzi. W przypadku złożonych i skomplikowanych projektów bardzo często dochodzi do sytuacji, w której musimy połączyć kilka rozwiązań i technik.

Zatem nie ma sensu odrzucać wszystkiego, czego się do tej pory nauczyłeś. Żadne z omówionych wcześniej technik nie wykluczają się wzajemnie, a każda z nich lepiej lub gorzej nadaje się do rozwiązania konkretnych problemów.

Życie jest skomplikowane. To samo dotyczy mobilnego internetu!

Nie istnieją głupie pytania

P: Czy są jakieś inne frameworki dla mobilnych aplikacji internetowych?

O: I to ile! Rosną jak grzyby po deszczu. Niektóre z nich to: Sencha Touch, Wink, iUI, DHTMLX Touch czy JQTouch.

P: Z czego tak naprawdę złożony jest framework dla mobilnych aplikacji internetowych?

O: To zależy od konkretnego rozwiązania, ale w większości to kombinacja JavaScriptu, CSS i grafiki (lub innych zasobów), dzięki którym interfejs użytkownika wygląda i zachowuje się jak mobilny. Niektóre frameworki zawierają też część serwerową, która generuje (a nie dopasowuje) zawartość.

P: Chwila, a co z zepto.js albo XUI?

O: Zepto.js (bardzo lekki skrypt JavaScript oferujący składnię przypominającą jQuery) i XUI (też niewielki) plasują się bardziej po stronie bibliotek, a nie frameworków. Frameworki zawierają komponenty interfejsu użytkownika, a biblioteki składają się tylko z kodu (w tym przypadku JavaScript). Nie ma jednak ściśle wytyczonej granicy między frameworkami a bibliotekami.

P: Zatem jQuery Mobile to mobilna wersja oryginalnej biblioteki jQuery, tak?

O: Nie tak szybko! Framework jQuery Mobile został zbudowany na bazie biblioteki jQuery, ale jej nie zastępuje. Kiedy zaczniemy budować aplikację za pomocą jQuery Mobile, zauważysz, że pierwszym skryptem, który dołączymy do projektu, będzie właśnie bazowa biblioteka jQuery.

P: W takim razie jQuery Mobile jest frameworkiem napisanym w JavaScripcie, który rozszerza bibliotekę jQuery. Mam rację?

O: Powoli, wszystko po kolei. Tak, we frameworku jQuery Mobile znajdziesz JavaScript, ale nie jest to framework *javascriptowy*. Jest czymś więcej. To framework służący do budowania interfejsów użytkownika, a to oznacza, że zawiera też arkusze stylów, ikony i inne niezbędne elementy.

P: Czy naprawdę musimy korzystać z frameworku?

O: Czy musimy? Z technicznego punktu widzenia? Nie, wcale nie musimy! Tak naprawdę nawet Cię zachęcamy do zbudowania aplikacji od podstaw, jeżeli uznasz, że w Twoim przypadku będzie to najlepsze wyjście.

Musisz jednak wziąć pod uwagę, że jedną z lepszych cech frameworków, a jQuery Mobile w szczególności, jest to, że biorą na siebie sprawy wszystkich błędów i niekonsekwencji działania różnych platform. Zespoły programistów pracujące nad frameworkami skupiają się też na niedociągnięciach wielu mobilnych przeglądarek.

Gdybyśmy chcieli opisać wszystko, co należałoby zrobić w projekcie Tartanator bez użycia frameworku, z pewnością zabrakłoby nam miejsca, a efekt nie byłby najlepszy. Nie wspominamy nawet o chaosie związanym z przeprowadzaniem wymaganych testów i problemach ze sprawdzaniem działania aplikacji na różnych platformach.

P: Wciąż nie do końca to wszystko rozumiem. Dlaczego Tartanator jest bardziej aplikacją niż witryną internetową?

O: Bo tak powiedział Ewan. A tak naprawdę różnice między aplikacją i witryną są na tyle subtelne, że jednoznaczne wskazanie właściwego określenia jest umowną kwestią.

Ewan ma wizję Tartanatora jako funkcjonalnego internetowego „czegoś". Skupia się na możliwości wyszukiwania i tworzenia tartanów, a także powiadamiania o wydarzeniach. W jego mniemaniu dzięki temu to „coś" można nazwać aplikacją.

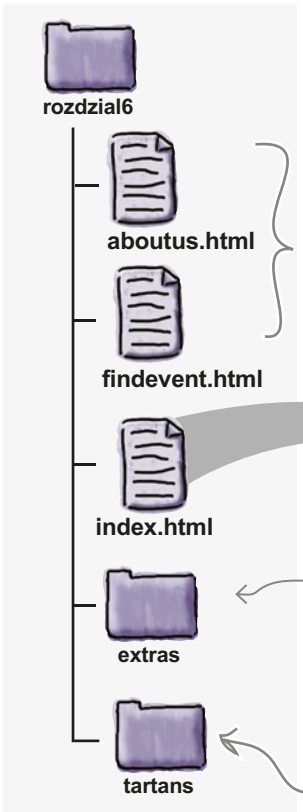
P: A co, gdyby to była jednak witryna? Nie moglibyśmy wtedy użyć frameworku jQuery Mobile?

O: jQuery Mobile to framework służący do tworzenia interfejsów użytkownika. Nie ma znaczenia, czy nasz projekt nazwiemy aplikacją, czy witryną. Jego zadaniem jest dostarczenie użytecznych elementów (opartych na CSS, JavaScripcie i HTML5), które będą się dobrze zachowywały na różnych platformach.

P: No dobrze, a które przeglądarki mobilne wspierają HTML5?

O: Żadna przeglądarka nie wspiera (w całości) HTML5. Bez paniki! HTML5 składa się z wielu modułów, a przeglądarki mobilne obsługują coraz więcej z nich. Tak samo jak starsze przeglądarki desktopowe nie w pełni wspierają CSS 2.1, tak i nie wszystkie przeglądarki będą wspierać HTML5 w sposób opisany w specyfikacji. Poza tym specyfikacja cały czas się rozwija. Jeśli szukasz informacji o wsparciu dla konkretnych możliwości HTML5 i innych głównych technologii internetowych, koniecznie zajrzyj na stronę *http://www.caniuse.com*.

Tworzenie prostej strony z jQuery Mobile



Struktura katalogu rozdzial6 — od tego zaczynamy.

Zanim zaczniesz prawdziwą przygodę z Tartanatorem, musisz się dowiedzieć, jak utworzyć prostą stronę z wykorzystaniem frameworku jQuery Mobile, tak byś mógł opracować strony z projektu. Będziesz zaskoczony, jakie to proste.

Zaczynamy od podstaw

Zaczniemy od najprostszej możliwej strony HTML, dzięki czemu wyraźnie zobaczysz, na czym polega praca z jQuery Mobile.

Używamy znacznika <DOCTYPE> z HTML5.

Oto cała zawartość pliku index.html, naprawdę!

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8" />
   <title>Tartanator</title>
</head>
<body>
<h1>Tartanator</h1>
<p>Tartanator to organizacja łącząca
stowarzyszenia, firmy i zwykłych ludzi, którym
leży na sercu utrzymanie dziedzictwa Szkotów
poprzez popularyzowanie historii i kultury
<strong>tartanów</strong> na całym świecie.</p>.
</body>
</html>
```

index.html

Dołączamy składniki jQuery Mobile

Otwórz plik *index.html* w edytorze i przygotuj się na dorzucenie jQuery Mobile. Zaczynamy od **dołączenia** skryptów JavaScript i arkusza CSS.

Dzięki dołączeniu skryptów jQuery i jQuery Mobile nasza aplikacja stanie się bardziej „mobilna".

Dołączamy arkusz CSS z jQuery Mobile, dzięki czemu elementy strony będą wyglądały na „mobilne".

Pamiętaj, że jQuery Mobile jest niczym bez bazowej biblioteki jQuery.

```
<title>Tartanator</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.1.0/jquery.mobile-
1.1.0.min.css" />
<script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
<script src="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.js"></
script>
</head>
```

index.html

Kod pozostałych elementów strony

Dołączyliśmy trzy główne składniki jQuery Mobile, korzystając z CDN jQuery (ang. *Content Delivery Network*), co oznacza, że są one pobierane z serwera *code.jquery.com*. To pewne i szybkie źródło plików.

Po dołączeniu tych trzech plików — arkusza CSS, jQuery i jQuery Mobile — musimy wprowadzić kilka zmian w kodzie HTML znajdującym się w sekcji `<body>`.

Poprzez umieszczenie fragmentów zawartości w blokach `<div>` z atrybutami `data-*` informujemy jQuery Mobile, w jaki sposób mają być traktowane:

Atrybuty data-* opiszemy już za chwilę.

Atrybut data-role równy „page" oznacza, że zawartość tego bloku ma być traktowana jako cała strona.

Atrybut data-role równy „header" oznacza, że ten element ma wyglądać jak nagłówek.

Atrybut data-role równy „content" oznacza, że to jest główna zawartość strony.

Wrzucamy tutaj stopkę, by zobaczyć, co z nią zrobi jQuery Mobile.

```
<body>
<div data-role="page">
   <div data-role="header">
      <h1>Tartanator</h1>
   </div><!-- /header -->
   <div data-role="content">
      <p>Tartanator to organizacja łącząca stowarzyszenia,
firmy i zwykłych ludzi, którym leży na sercu utrzymanie
dziedzictwa Szkotów poprzez popularyzowanie historii i kultury
<strong>tartanów</strong> na całym świecie.</p>
   </div><!-- /content -->
<div data-role="footer">
   <h4>Bring forrit the tartan!</h4>
</div><!-- /footer -->
</div><!--/page-->
</body>
```

index.html

„Bring forrit the tartan!" to okrzyk wojenny przypisywany sir Colinowi Campbellowi podczas oblężenia Lucknow w 1850 roku.

Jazda próbna

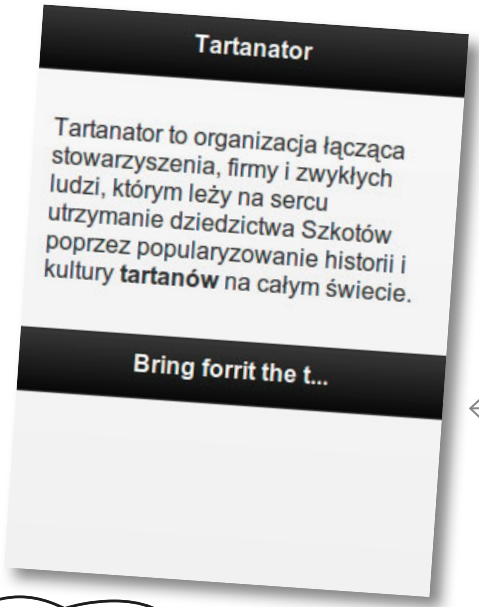
Stworzenie prostej strony z jQuery Mobile jest naprawdę łatwe!

Uzupełnij kod w sekcjach `<head>` i `<body>` pliku *index.html* i go zapisz, a następnie otwórz w przeglądarce (jeśli chcesz, możesz wyświetlić tę stronę w przeglądarce mobilnej).

No to załatwione!

Nie mamy jeszcze wszystkiego (no dobra, nie mamy prawie nic), ale główna strona wygląda już dosyć „mobilnie”.

Zwróć uwagę, że jQuery Mobile dodał gradienty, ustawił rozmiary i fonty, dzięki czemu elementy dokumentu wyglądają jak nagłówek i stopka.



Nasza pierwsza strona jQuery Mobile wyrenderowana na iPhonie.

Hm... Tekst stopki nie mieści się w całości, ale tym zajmiemy się trochę później.

Co ma niby znaczyć „wygląda mobilnie”? Przecież ta strona wygląda jak natywna aplikacja w iOS!

Domyślny wygląd oferowany przez jQuery Mobile do złudzenia przypomina iOS.

Może Ci się to podobać lub nie, ale zaokrąglone przyciski i nagłówki z delikatnym gradientem występujące w natywnych aplikacjach na platformę iOS stały się charakterystyczną cechą mobilnych aplikacji internetowych. Wielu użytkowników taki wygląd utożsamia z mobilnością.

Jednak poza stylizacją elementów zmienia się coś jeszcze. Powiększają się odstępy między słowami, a rozmiar fontu jest optymalizowany do wyświetlania na małych ekranach. Niedługo utworzymy formularz i wtedy zobaczysz, że jQM powiększa pola i zmienia przyciski, dostosowując w ten sposób interfejs do obsługi za pomocą dotyku.

To tylko kilka przykładów.

Atrybuty data-*

Framework jQuery Mobile robi użytek z atrybutów `data-*` wprowadzonych w HTML5. W naszej prostej stronie użyliśmy atrybutu `data-role`, by poinformować framework o roli, jaką w strukturze strony pełni dany element. Teraz sprawa jest bardzo prosta: mamy nagłówek (`header`), jakąś zawartość (`content`) i stopkę (`footer`).

Kolejne składniki Tartanatora

Położyliśmy już solidny fundament, więc możemy zacząć dobudowywać kolejne składniki Tartanatora. Mamy już stronę główną, ale — by ruszyć z projektem do przodu — musimy ją jeszcze połączyć z innymi stronami.

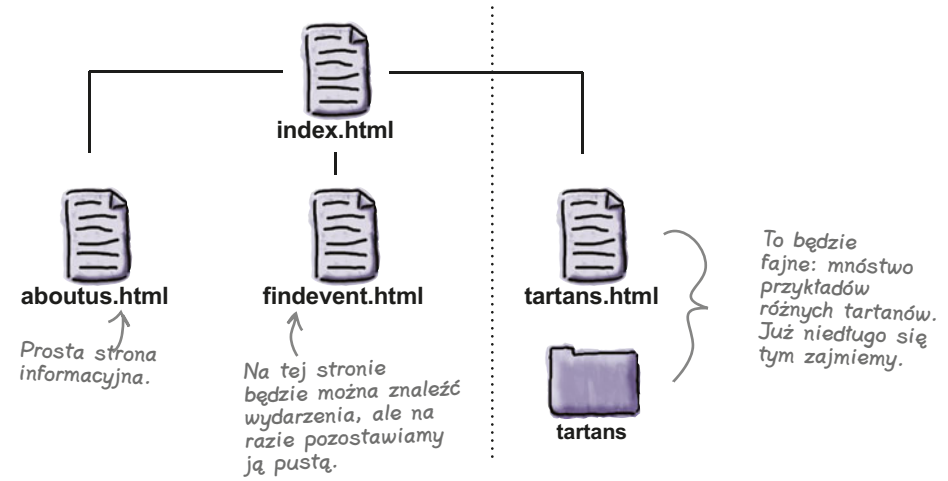
Zbliżenie na atrybuty data-*

Atrybuty `data-*` zostały wprowadzone w HTML5, by umożliwić programistom wiązanie z elementami HTML niewielkich, ale istotnych danych.

W przeszłości w tym samym celu były często stosowane inne atrybuty, takie jak `class` czy `title`, które — jak wiadomo — do tego nie służą.

Teraz mamy do dyspozycji atrybuty stworzone specjalnie w tym celu, a jQuery Mobile robi z nich świetny użytek.

Podczas dalszej pracy nad projektem spotkasz się jeszcze z niejednym atrybutem `data-*`, który przekazuje frameworkowi jQuery Mobile istotne informacje.



Ćwiczenie

Nadszedł czas, by dodać nawigację w postaci listy linków do trzech podstron. W pliku *index.html*, pod akapitem ze wstępem w bloku `<div>` głównej zawartości, umieść bardzo prosty element `<ul>`. Kolejne pozycje listy to: **O nas**, **Znajdź wydarzenie** oraz **Popularne tartany**.

Odsyłacze dodamy trochę później. Na razie wystarczy sam tekst.

Rozwiązanie ćwiczenia

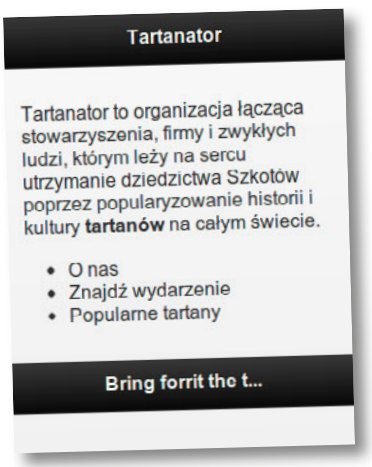
Prosty HTML rządzi!

```
   <p>Tartanator to organizacja łącząca stowarzyszenia, firmy i zwykłych ludzi,
którym leży na sercu utrzymanie dziedzictwa Szkotów poprzez popularyzowanie
historii i kultury <strong>tartanów</strong> na całym świecie.</p>
   <ul>
      <li>O nas</li>
      <li>Znajdź wydarzenie</li>
      <li>Popularne tartany</li>
   </ul>
</div><!-- /content -->
```

index.html

Robimy z tego listę w stylu jQuery Mobile

Po zapisaniu zmian w pliku *index.html* i wyświetleniu go w przeglądarce zobaczysz, że przygotowana przez nas lista nie wygląda zbyt ciekawie (ani tym bardziej mobilnie). To wszystko dlatego, że nie poinformowaliśmy jQuery Mobile o jej istnieniu. **Musimy znów skorzystać z atrybutu `data-role`!**



Ta lista jest dosyć, hm..., pospolita.

```
<ul data-role="listview">
```

Użyliśmy atrybutu data-role do poinformowania jQuery Mobile, by traktował to jako widok listy.

Dzięki temu jQuery Mobile zastosuje odpowiednie style i dowie się o istnieniu tej listy.

Framework jQuery Mobile zamienia zwykłą listę na taką, która wygląda i zachowuje się w sposób zbliżony do jej mobilnych, interaktywnych odpowiedników.

Jazda próbna

W pliku *index.html* do elementu `<ul>` dopisz atrybut `data-role` i wyświetl stronę w przeglądarce.

Co sądzisz o zmianie wyglądu zastosowanej przez jQuery Mobile?

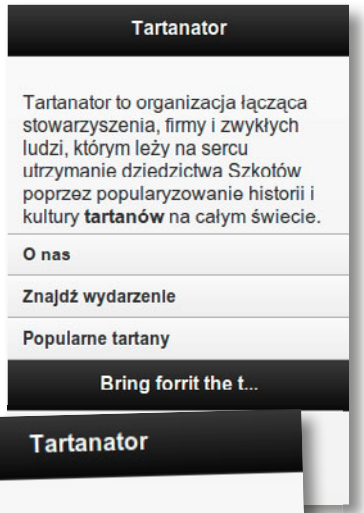
Nasza lista jest lepsza, ale nie idealna

Framework jQuery Mobile domyślnie traktuje `listview` jak zawartość strony, czyli wypełnia elementami `<li>` całą dostępną szerokość. W przypadku naszej strony głównej wygląda to trochę topornie.

Element nawigacyjny powinien być tylko częścią zawartości, a nie stanowić jej w całości. Mamy na szczęście możliwość przedstawienia listy jako *wstawki* (ang. *inset*), dzięki czemu lista jest umieszczana na stronie z uwzględnieniem jej pozostałej zawartości. Sprawdźmy, jak to działa.

Pierwsze podejście

Lista wygląda tak sobie.



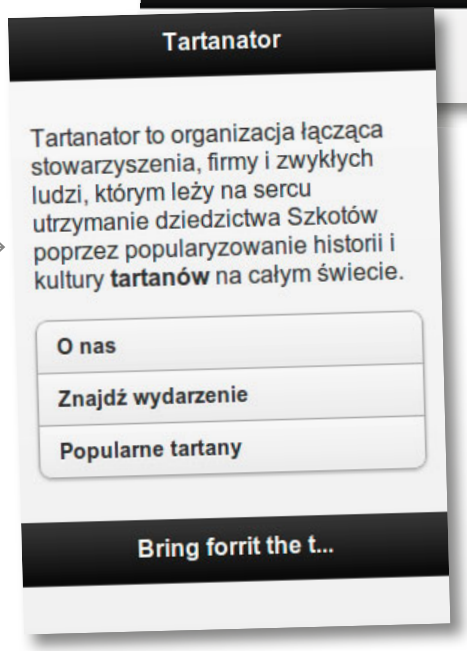

```
<ul data-role="listview" data-inset="true">
```

Zrób to sam!

Do znacznika listy w pliku *index.html* dodaj atrybut `data-inset`, a następnie ponownie wyświetl stronę w przeglądarce.

Drugie podejście

Lista typu „wstawka" (inset) wygląda znacznie lepiej.



Lista wygląda świetnie i w ogóle, ale żadnego z jej elementów nie można kliknąć. Ale chwilkę, czy z elementów listy nie powinniśmy zrobić odsyłaczy?

Masz rację. Dodajmy odsyłacze do pozostałych stron!

Czas, by Tartanator stał się witryną złożoną z więcej niż jednej strony.

Odsyłacze do wielu stron w jQuery Mobile

Tworzenie odsyłaczy do innych stron jest niezwykle proste. Wystarczy do każdej pozycji listy dodać znacznik <a>.

```
<ul data-role="listview" data-inset="true">
    <li><a href="aboutus.html">O nas</a></li>
    <li><a href="findevent.html">Znajdź wydarzenie</a></li>
    <li><a href="tartans.html">Popularne tartany</a></li>
</ul>
```

index.html

Zrób to sam!

W pliku *index.html* do wszystkich pozycji listy dodaj odsyłacze i je przetestuj. Co się dzieje?

Po dodaniu odsyłaczy automatycznie pojawiły się ikony ze strzałkami, które wyraźnie wskazują, że są to odsyłacze.

Na razie nie ma pliku tartans.html. Tak reaguje jQuery Mobile, gdy zostanie kliknięty niepoprawny odsyłacz.

Wszystko o jQuery Mobile

Wywiad tygodnia: Jak jQuery Mobile obsługuje ładowanie stron?

Redaktor: Doszły mnie słuchy, że chcesz nam powiedzieć, czym są dla ciebie „strony".

jQuery Mobile: Zgadza się! To jedna z ciekawszych spraw na mój temat, więc koniecznie chcę o tym opowiedzieć.

Pamiętacie, jak wyglądała struktura prostej strony? Chodzi mi przede wszystkim o blok `<div>` z atrybutem `data-role` równym `page`.

Redaktor: Jasne.

jQuery Mobile: Dla mnie stroną są znaczniki umieszczone właśnie w tym bloku `<div>`, czyli najczęściej nagłówek, jakaś zawartość i stopka.

Redaktor: W takim razie po co są pozostałe elementy dokumentu HTML? Można się ich pozbyć?

jQuery Mobile: W żadnym wypadku! W aplikacji internetowej zaprojektowanej zgodnie z moimi zasadami każdy plik HTML jest samodzielnym, autonomicznym dokumentem, który możesz wyświetlić w przeglądarce.

Redaktor: Pogubiłem się. Jeśli każdy plik HTML jest niezależny, po co to całe zamieszanie z blokiem `<div>` z atrybutem `data-role` równym `page`?

jQuery Mobile: Już wyjaśniam. Gdy pierwszy raz trafiasz na stronę z mojej witryny lub aplikacji, ładuje się ona jak każda inna zwykła strona HTML.

Jednak po załadowaniu tej strony sytuacja się zmienia. Gdy klikniesz odsyłacz, odszukuję zawartość strony (czyli to, co znajduje się w bloku `<div>` z atrybutem `data-role` równym `page`) z dokumentu HTML pobranego za pomocą AJAX-a i wstrzykuję do istniejącej struktury DOM...

Redaktor: Ale po co te wszystkie kombinacje?

jQuery Mobile: Gdybyś dał mi dokończyć, wszystkiego byś się dowiedział...

Zamiast pobierać całe strony, razem ze skryptami, obrazkami i stylami, oraz ponownie tworzyć od zera strukturę DOM, wymieniam tylko to, co faktycznie się liczy. Dzięki temu jest zgłaszanych mniej żądań HTTP, minimalizuje się wykorzystanie łącza i mocy obliczeniowej, a witryna lub aplikacja zachowuje się w bardziej naturalny sposób, zbliżony do natywnych aplikacji.

Redaktor: Jeśli fragmenty zawartości są ładowane dynamicznie, jaki adres jest wyświetlany w pasku adresu przeglądarki?

jQuery Mobile: Moja skromna powierzchowność kryje bardzo zaawansowany model nawigacji. Zawsze staram się dążyć do podawania prawdziwych adresów URL stron aplikacji, nawet jeśli te strony są generowane dynamicznie albo w jednym dokumencie HTML znajduje się wiele bloków `<div>` z atrybutem `data-role` równym `page`.

Ponieważ mam unikalne adresy URL dla fragmentów zawartości pobranej za pomocą AJAX-a, mogę sprawić, by wyglądały i służyły jak pełnoprawne strony internetowe. Jeśli przeglądarka mi na to pozwoli (a nie wszystkie pozwalają), po asynchronicznym pobraniu zawartości mogę nawet zaktualizować URL wyświetlany w pasku adresu.

Redaktor: Jeśli dobrze zrozumiałem, mogę uzyskać bezpośredni dostęp do każdego pliku HTML w mojej aplikacji lub witrynie, ale równocześnie zawartość tych samych dokumentów może być pobierana niezależnie i dynamicznie, jeśli są połączone odsyłaczami, co korzystnie wpływa na wydajność i responsywność. Mam rację?

jQuery Mobile: Doskonale to ująłeś!

Redaktor: Bardzo ci dziękuję, jQuery Mobile, za naprawdę dynamiczną rozmowę.

Jeśli zawartość stron jest ładowana dynamicznie za pomocą AJAX-a, co się stanie w starszych przeglądarkach z ograniczonym wsparciem dla JavaScriptu lub w przeglądarkach, które w ogóle nie obsługują JavaScriptu? Czy to ma szansę zadziałać?

jQuery Mobile może działać nawet wtedy, gdy przeglądarka w ogóle nie obsługuje JavaScriptu.

To jakieś szaleństwo!

Przypomnij sobie, czym w kodzie HTML są odsyłacze z naszej strony: zwykłymi elementami <a>. To kod JavaScript zawarty we frameworku jQuery Mobile zamienia je na AJAX-owe cudeńka, ale tylko w przeglądarkach, które go wspierają.

W przeglądarkach bez wsparcia dla JavaScriptu nawigacja między stronami będzie działała tak samo jak w przypadku zwykłych stron internetowych.

Nie istnieją głupie pytania

P: Jeśli strony są ładowane za pomocą AJAX-a, po co w ogóle tworzyć oddzielne strony? Nie wystarczyłoby umieścić całej zawartości aplikacji w jednym dokumencie i użyć jQuery Mobile do wyświetlania i ukrywania pożądanych fragmentów?

O: To możliwe, ale traci się w ten sposób możliwości stopniowego ulepszania, jakie oferuje jQuery Mobile. W ten sposób starsze przeglądarki mobilne (które nie potrafią wystarczająco dobrze obsłużyć JavaScriptu) zostaną na lodzie.

Poza tym powstałby w ten sposób ogromny, skomplikowany plik, który byłby trudny w utrzymaniu. Struktura DOM też byłaby przerośnięta, więc słabsze urządzenia mogłyby mieć z nią problemy. Framework jQuery Mobile zachęca, by tworzyć witryny i aplikacje internetowe tak samo jak do tej pory, czyli z odrębnych plików HTML, które są niezależne, ale równocześnie mogą zostać pobrane za pomocą AJAX-a.

P: Skąd jQuery Mobile wie, którego atrybutu `data-*` użyć? Gdzie mogę znaleźć informację o wszystkich dostępnych atrybutach `data-*`?

O: Programiści mogą sami wybrać nazwy atrybutów `data-*`, które im się podobają (warunek jest taki, że muszą się rozpoczynać literą). Nie ma jakiegoś z góry narzuconego zestawu nazw. Cała sprawa polega na tym, że atrybuty `data-*` odnoszą się do funkcjonowania witryny, w której występują, czyli nie są przeznaczone do przekazywania informacji do zewnętrznych aplikacji. Właśnie dlatego programiści mają względną dowolność tworzenia własnych nazw `data-*`.

P: Której wersji jQuery Mobile używamy?

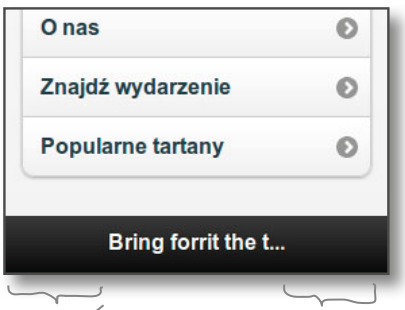
O: Zespół programistów odpowiadający za framework jQuery Mobile pracuje niezwykle szybko. Pierwsza wersja pojawiła się we wrześniu 2010 roku. Kolejne były wypuszczane co kilka miesięcy (czasem nawet częściej). W chwili pisania tej książki była dostępna stabilna wersja 1.1.0 i właśnie ona jest używana w projekcie Tartanator.

RE(KONSTRUKCJA) STOPKI

Jak widzieliśmy na stronie 230, tekst stopki jest przycinany na węższych ekranach. Wynika to stąd, że jQuery Mobile zostawia trochę miejsca po bokach zawartości nagłówków i stopek z przeznaczeniem na przyciski. Ale przecież my nie mamy żadnych przycisków w stopce. Naprawmy to!

Nasza stopka przed rekonstrukcją. Jest przycięta.

Jest też trochę zbyt „ciężka". Można by ją rozjaśnić.



jQuery Mobile zostawia domyślnie pewną przestrzeń na przyciski wokół elementów nagłówkowych (<h1>, <h2> itd.) w nagłówkach i stopkach.

1 Usuń znacznik <h4> otaczający tekst stopki.
W tej chwili tekst stopki znajduje się w elemencie <h4>. Usunięcie tego elementu zapobiegnie rezerwowaniu przez jQuery Mobile miejsca dla przycisków i związanemu z tym przycinaniu tekstu.

```
<div data-role="footer">
  <h4>Bring forrit the tartan!</h4>
</div><!-- /footer -->
```

index.html

2 Dopisz regułę CSS wypośrodkowującą tekst stopki i ustawiającą niewielki padding.
Niestety wykasowanie znacznika <h4> powoduje również usunięcie wypośrodkowania i paddingu. Musimy temu zapobiec we własnym arkuszu CSS.

Otwórz plik *tartans/tartans.css* w edytorze i na samym początku dodaj poniższą regułę (w pliku znajduje się już trochę kodu CSS):

```
[data-role="footer"] { text-align: center; padding: 5px 0;}
#abercrombie { background-image:url('icons/abercrombie.png'); }
```

Ten selektor odnosi się do elementów z atrybutem data-role ustawionym na footer.

tartans.css

Odwróć stronę, by zobaczyć ciąg dalszy.

RE(KONSTRUKCJA) STOPKI

3 **Dołącz arkusz stylów.**
Ponownie zajrzyj do pliku *index.html* i dołącz odsyłacz do arkusza stylów.

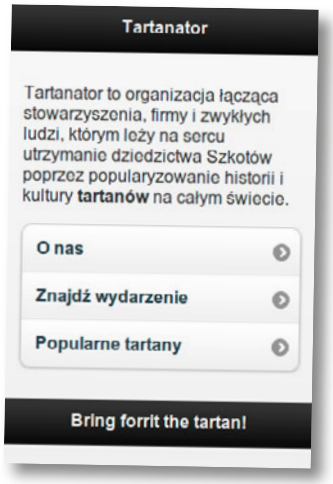
```
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.0rc1/
jquery.mobile-1.0rc1.min.css" />
<link rel="stylesheet" href="tartans/tartans.css" />
<script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
```

index.html

4 **Sprawdź efekty zmian.**
Zapisz plik *index.html* i wyświetl stronę w przeglądarce na smartfonie lub w symulatorze.

Poprawienie obu tych spraw wymaga wprowadzenia drobnych zmian w bloku `<div>` stopki.

Musimy przekazać frameworkowi jQuery Mobile, by zastosował *stałe pozycjonowanie*, dzięki czemu stopka będzie „przyklejona” do dolnej krawędzi ekranu. Musimy też zmodyfikować *schemat kolorystyczny*, tak by stopka nie rzucała się aż tak w oczy.



Nie zawsze stopka znajduje się na samym dole strony — czasami pojawia się odstęp.

5 **Do stopki zastosuj stałe pozycjonowanie, tak by zawsze była wyświetlana w tym samym miejscu.**
Poprzez dodanie atrybutu `data-position` i ustawienie go na `fixed` informujemy jQuery Mobile, by do elementu stopki zostało zastosowane stałe pozycjonowanie. Dzięki temu stopka będzie wyświetlana zawsze na samym dole strony.

6 **Zastosuj inny schemat kolorystyczny, tak by stopka mniej rzucała się w oczy.**
Początkowy arkusz stylów jQuery Mobile oferuje pięć domyślnych grup kolorów, tzw. *próbek*. Pięciu próbkom przyporządkowane są litery od a do e.

Domyślny schemat kolorystyczny dla próbek od a do e jest zdefiniowany w arkuszu CSS należącym do frameworku jQuery Mobile.

Domyślnie elementy nagłówka i stopki korzystają z próbki a, która jest najbardziej kontrastową grupą kolorów.

Stopka nie jest wyjątkowo istotnym elementem strony, a teraz zbyt mocno rzuca się w oczy. Wystarczy przypisać jej próbkę c, a przestanie się tak wybijać.

Atrybut `data-theme` możesz dodać do każdego elementu i w ten sposób zmienić próbkę domyślnie zastosowaną przez jQuery Mobile.

```
<div data-role="footer" data-position="fixed" data-theme="c">
   Bring forrit the tartan!
</div><!-- /footer -->
```

To by było na tyle. Stopka powinna wyglądać dużo lepiej. Zapisz zmiany i wyświetl stronę w przeglądarce mobilnej lub w symulatorze.

Wraz z wydaniem jQuery Mobile w wersji 1.0 zostało udostępnione narzędzie Theme Roller, które upraszcza tworzenie własnych schematów kolorystycznych. Koniecznie zajrzyj na http://jquerymobile.com/themeroller/.

Sedno Tartanatora — tartany jako takie

Sprawdźmy, co udało nam się zrealizować z pierwszej fazy projektu Tartanator:

- [x] **Zbudować strony prezentujące treści i strukturę witryny.**

  Musimy stworzyć główne sekcje i strony oraz podstawową strukturę witryny.

  Mamy już podstawowy układ i utworzyliśmy zarys głównych stron.

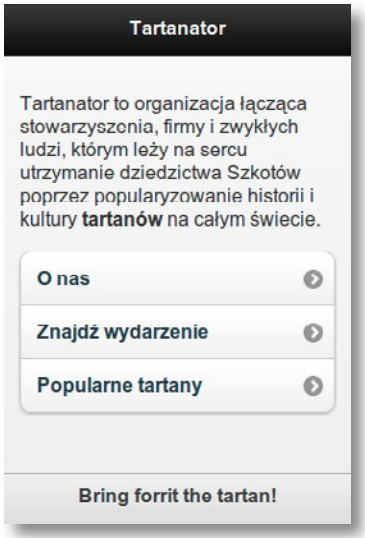
- [ ] **Stworzyć listę tartanów.**

  W pierwszej fazie utworzymy listę istniejących popularnych wzorów tartanów. Listę powinno dać się łatwo przeglądać i — co oczywiste — powinna wyglądać i zachowywać się jak aplikacja, i to w dodatku mobilna.

  Następne na liście!

- [ ] **Zaprojektować prototyp formularza służącego do tworzenia tartanów.**

  Do tego dojdziemy w swoim czasie.



Położenie stopki jest już prawidłowe, a ona sama nie rzuca się już tak mocno w oczy. Wielki sukces!

Mamy już główną zawartość stron, a przynajmniej ich szkielety. Teraz skupimy się na czymś znacznie ciekawszym — tartanach.

W pierwszej fazie projektu musimy zadbać o to, by użytkownicy mogli **przeglądać kolekcję popularnych i nietypowych tartanów**. Wyobraź to sobie jako miniencyklopedię tartanów.

Zrób to sam! → Z katalogu *extras* skopiuj plik *tartans.html* i umieść go w katalogu *rozdział6* (inaczej mówiąc, przenieś go o jeden poziom wyżej).

Katalog tartans, który zawiera strony HTML poszczególnych tartanów, powinien się już znajdować w katalogu rozdział6.

Do roboty! Bierzemy się za plik tartans.html

W pliku *tartans.html* znajduje się zalążek listy (`<ul>`) popularnych tartanów. Abyś miał na czym pracować, lista zawiera tartany, których nazwy rozpoczynają się na A i B. Po przeniesieniu pliku *tartans.html* we właściwe miejsce załaduj aplikację Tartanator w przeglądarce (mobilnej lub desktopowej) i zobacz, jak to działa. Po kliknięciu nazwy jakiegoś tartanu pojawia się powiązana z nim strona.

Listę przygotowaliśmy za Ciebie

Zróbmy coś ciekawego z tą listą

Lista tartanów nie robi zbyt dużego wrażenia, zwłaszcza jeśli porówna się ją ze stronami poszczególnych tartanów. Dobre wieści! Framework jQuery Mobile pozwala na proste dodanie miniaturek do listy, dzięki czemu przy każdej pozycji znajdzie się mała ikona (o niebo lepiej!). Oto przykład:

Tak, wszystko, co musisz zrobić, to dodać znacznik obrazka. jQuery Mobile zajmuje się resztą.

```
<li><a href="tartans/abercrombie.html">
   <img src="tartans/icons/abercrombie.png" alt="Abercrombie" />
   <h3>Abercrombie</h3>
</a></li>
```

Ćwiczenie

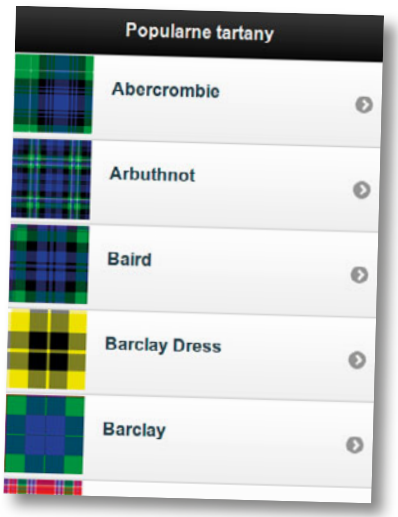
Dodaj miniaturki do wszystkich elementów listy w pliku *tartans.html*. Ikony znajdują się w katalogu *tartans/icons* i mają takie same nazwy jak pliki HTML (ale z rozszerzeniem *.png*).

Rozwiązanie ćwiczenia

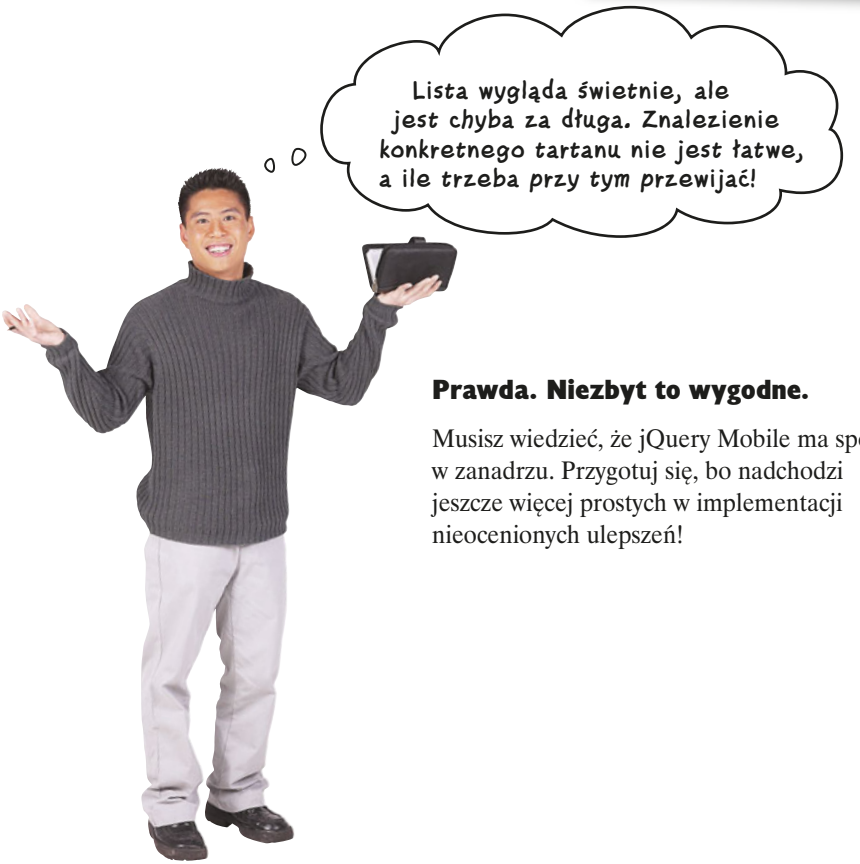
```
<ul data-role="listview">
   <li><a href="tartans/abercrombie.html">
      <img src="tartans/icons/abercrombie.png" alt="Abercrombie" />
      <h3>Abercrombie</h3>
   </a></li>
   <li><a href="tartans/arbuthnot.html">
      <img src="tartans/icons/arbuthnot.png" alt="Arbuthnot" />
      <h3>Arbuthnot</h3>
   </a></li>
   <li><a href="tartans/baird.html">
      <img src="tartans/icons/baird.png" alt="Baird" />
      <h3>Baird</h3>
   </a></li>
   <li><a href="tartans/barclay-dress.html">
      <img src="tartans/icons/barclay-dress.png" alt="Barclay Dress" />
      <h3>Barclay Dress</h3>
   </a></li>
   <li><a href="tartans/barclay.html">
      <img src="tartans/icons/barclay.png" alt="Barclay" />
      <h3>Barclay</h3>
   </a></li>
   <li><a href="tartans/birrell.html">
      <img src="tartans/icons/birrell.png" alt="Birrell" />
      <h3>Birrell</h3>
   </a></li>
   <li><a href="tartans/blair.html">
      <img src="tartans/icons/blair.png" alt="Blair" />
      <h3>Blair</h3>
   </a></li>
   <li><a href="tartans/borthwick-dress.html">
      <img src="tartans/icons/borthwick-dress.png" alt="Borthwick Dress" />
      <h3>Borthwick Dress</h3>
   </a></li>
   <li><a href="tartans/borthwick.html">
      <img src="tartans/icons/borthwick.png" alt="Borthwick" />
      <h3>Borthwick</h3>
   </a></li>
   <li><a href="tartans/bruce.html">
      <img src="tartans/icons/bruce.png" alt="Bruce" />
      <h3>Bruce</h3>
   </a></li>
   <li><a href="tartans/buchanan.html">
      <img src="tartans/icons/buchanan.png" alt="Buchanan" />
      <h3>Buchanan</h3>
   </a></li>
</ul>
```

Wrzucamy pozostałe tartany

W kolejce czeka jeszcze całe mnóstwo tartanów, które chcą się znaleźć na liście (razem ze ślicznymi ikonami). Nie martw się, nie będziesz musiał dużo pisać. W katalogu *extras* odszukaj plik *tartans-list.txt*. W pliku znajduje się kod HTML, który jest listą (`<ul>`) wszystkich tartanów. Skopiuj całość i wklej w pliku *tartans.html*, zamieniając dotychczasowy kod listy.



Teraz mamy już listę zawierającą wszystkie tartany z naszej kolekcji. Fajne ikony!



Prawda. Niezbyt to wygodne.

Musisz wiedzieć, że jQuery Mobile ma sporo w zanadrzu. Przygotuj się, bo nadchodzi jeszcze więcej prostych w implementacji nieocenionych ulepszeń!

Filtrowanie i porządkowanie listy

1. **Do listy możemy dodać elementy rozdzielające.**
Możemy **podzielić listę** na sekcje, korzystając z elementów rozdzielających. Dzięki temu pogrupujemy tartany według pierwszej litery ich nazwy.

2. **Do listy możemy dodać filtr.**
Zastosowanie do listy filtru jest w jQuery Mobile bajecznie proste. Filtr wygląda jak pole wyszukiwania (z małą lupą i wszystkim, co jest jeszcze potrzebne!) i **filtruje listę zgodnie z tym, co wpisze użytkownik**. Nie musisz nic pisać w JavaScripcie!

Aby do listy zastosować filtr, wystarczy do znacznika <ul> dodać atrybut data-filter ustawiony na wartość „true”.

Tak, to wszystko!

1
```
<ul data-role="listview" data-filter="true">
```

Aby podzielić wizualnie listę na grupy, wystarczy dodać element <li> z atrybutem data-role z wartością „list-divider”.

2
```
<ul data-role="listview" data-filter="true">
    <li data-role="list-divider">A</li>
    <li><a href="tartans/abercrombie.html">
    <img src="tartans/icons/abercrombie.png" alt="Abercrombie" />
    <h3>Abercrombie</h3>
</a></li>
```

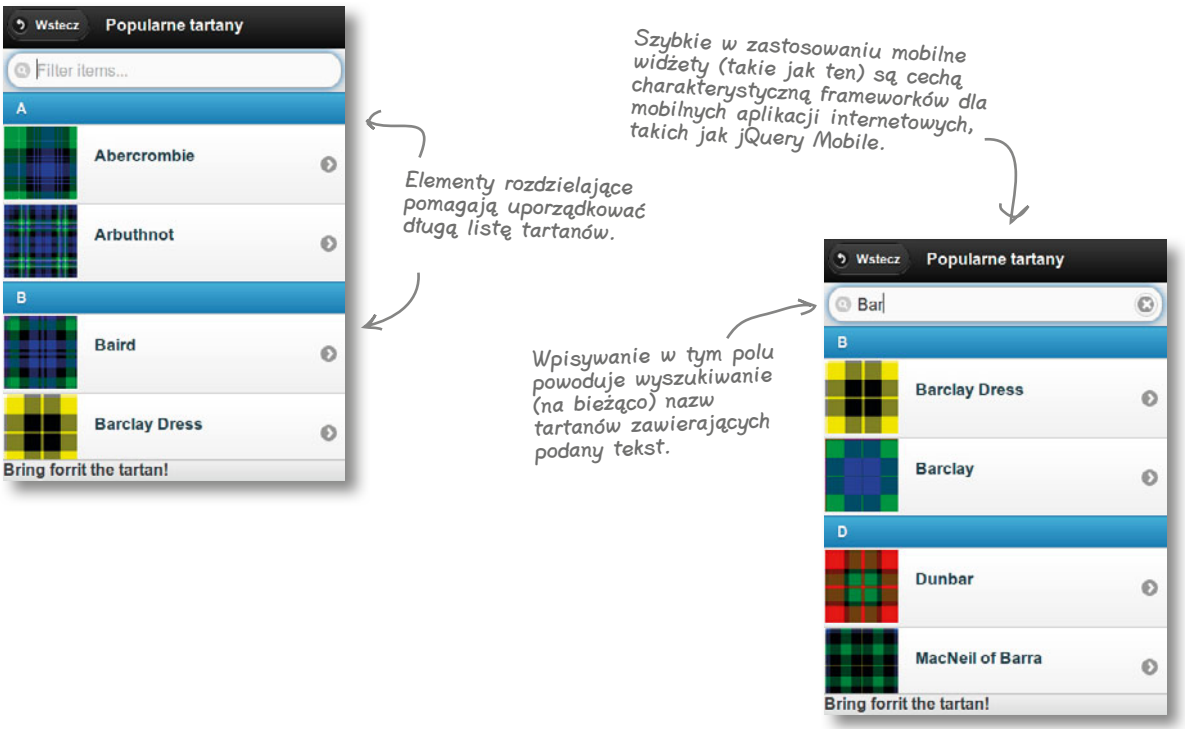
tartans.html

Jazda próbna

Do listy <ul> w pliku *tartans.html* dodaj filtr i elementy rozdzielające (dla każdej litery, oczywiście z wyjątkiem tych, które nie występują, takich jak Q czy X).

Zapisz zmiany i wyświetl stronę w przeglądarce. Nieźle, prawda?

Lista tartanów jest teraz zdecydowanie fajniejsza



Nie istnieją głupie pytania

P: Zauważyłem, że gdy zmieniają się strony w aplikacji, pojawia się efekt przejścia. Skąd się wziął?

O: Aby aplikacje internetowe jeszcze bardziej naśladowały interfejs użytkownika aplikacji natywnych, jQuery Mobile stosuje efekt przejścia podczas zmiany stron.

Domyślnie jest stosowany efekt „wjazdu" (`slide`), ale możesz go zmienić, dodając atrybut `data-transition` do znacznika odsyłacza. Dostępnych jest co najmniej kilka innych efektów. Więcej na ten temat znajdziesz w dokumentacji (na stronie frameworku).

P: Czy to zadziała na każdym urządzeniu?

O: Tego nie może zagwarantować żaden framework. jQuery Mobile stara się obsługiwać tyle platform, ile to tylko możliwe. Listę wspieranych urządzeń i przeglądarek (oraz stopień wsparcia) możesz znaleźć na stronie *http://jquerymobile.com/gbs*.

P: Co się w takim razie stanie, gdy będziemy chcieli wyświetlić aplikację na urządzeniu, które nie jest wspierane? Albo gdy jest wyłączona obsługa JavaScriptu?

O: Filozofia frameworku jQuery Mobile jest ściśle powiązana z ideą stopniowego ulepszania. Pamiętaj o kodzie HTML, który za tym wszystkim stoi!

Czemu nie sprawdzisz tego sam? Załaduj stronę Tartanator w przeglądarce internetowej z wyłączoną obsługą JavaScriptu. Jasne, efekt nie powala, ale to działa!

Nadszedł czas, by pokazać Ewanowi wczesną wersję Tartanatora

Jak wygląda nasz harmonogram?

- [x] **Zbudować strony prezentujące treści i strukturę witryny.**
- [x] **Stworzyć listę tartanów.**
- [ ] **Zaprojektować prototyp formularza służącego do tworzenia tartanów.**

Zanim zaczniemy pracować nad formularzem, sprawdźmy, co o naszych dokonaniach sądzi klient.

Kuba: Co ma niby znaczyć to „by zachowywało się jak prawdziwa natywna aplikacja”?

Łukasz: To dosyć względne, prawda? Myślę, że Ewanowi chodzi o to, żeby to, co zrobiliśmy, w jeszcze większym stopniu przypominało aplikację.

Kuba: Czyli...?

Łukasz: Zgaduję, że chodzi o zakładki, elementy nawigacyjne, więcej przycisków i ikon. Ewanowi spodobał się na przykład efekt przejścia. Właśnie takie rzeczy jego zdaniem świadczą o byciu aplikacją.

Kuba: Ale czy nie przesadzimy w tym naśladowaniu konkretnej platformy kosztem innych?

Łukasz: Też o tym pomyślałem. Kiedy ludzie mówią „natywna”, często są niebezpiecznie blisko stwierdzenia: „Zróbcie to tak, żeby wyglądało jak w iOS”. Przecież platformy różnią się między sobą.

Kuba: To powoli zaczyna przypominać *Paragraf 22*. Żeby nasz klient był zadowolony, musimy upodobnić aplikację do jej natywnych odpowiedników, co może z kolei spowodować niezadowolenie użytkowników innych platform, takich jak Android czy BlackBerry.

Łukasz: Myślę, że musimy wybrać kilka rozwiązań, które upodobnią Tartanator do aplikacji, ale bez ślepego naśladowania konkretnej platformy, na przykład *iPhone'a*. Trzeba jednak przyznać, że jQuery Mobile stylizuje elementy w sposób mocno przypominający iOS. Pójdziemy więc tą drogą.

Kuba: Zatem naszym celem jest teraz zbliżenie wyglądu i zachowania Tartanatora do aplikacji, ale niekoniecznie natywnej?

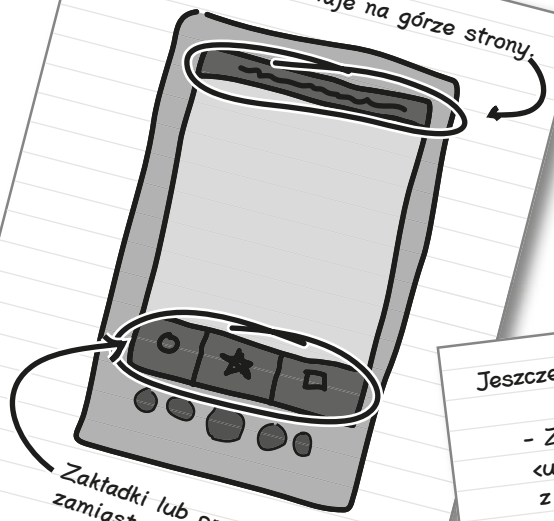
Łukasz: Właśnie tak. Naszkicuję kilka pomysłów zmian, które mogą nas zbliżyć do celu.

Nasz nowy cel: zbliżenie wyglądu i zachowania Tartanatora do aplikacji, ale niekoniecznie natywnej.

Triki, które przybliżą nas do aplikacji

Oto kilka prostych modyfikacji, które pomogą sprawić, by Tartanator wyglądał i zachowywał się jak aplikacja.

Nagłówek, który zostaje na górze strony.



Zakładki lub przyciski zamiast odsyłaczy.

Jeszcze bardziej podobne do aplikacji!

- Zamiana dotychczasowej listy <ul> odsyłaczy na stały pasek z przyciskami umieszczony na dole ekranu.

- Do aplikacji jeszcze bardziej zbliżą nas przyciski z ikonami.

- Nagłówek powinien zostawać na górze strony.

A co ze stroną O nas?

- Z ostatniego e-maila od Ewana: treść nie rozkłada się tak dobrze, jak chciałem.

- Może połączyć główną stronę ze stroną O nas? W ten sposób główna strona pełniłaby funkcję strony informacyjnej.

Przybliżanie się do aplikacji — plan działania

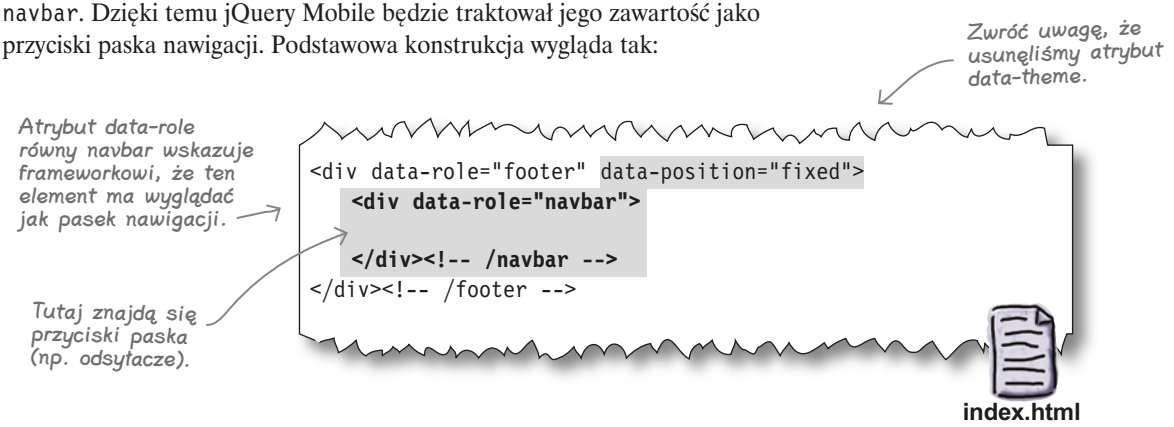
- ☐ Musimy przekonwertować odsyłacze z głównej strony na stały pasek narzędzi z ikonami.
- ☐ Trzeba wyrzucić osobną stronę O nas, ponieważ jest zbędna. Warto pomyśleć o zmianie głównej strony *de facto* na stronę O nas.
- ☐ Nagłówek musi być cały czas na górze strony.

Dodajemy pasek narzędzi w stopce

W jQuery Mobile bardzo łatwo dodać pasek w nagłówku lub stopce. Aby przybliżyć się do aplikacji, umieścimy taki pasek w stopce. Zastąpi on listę `<ul>` odsyłaczy do poszczególnych sekcji Tartanatora, znajdującą się na głównej stronie.

Konstruujemy pasek nawigacji

W kontenerze nagłówka lub stopki (oznaczonym odpowiednim atrybutem `data-role`) możemy umieścić kolejny blok `<div>` z atrybutem `data-role` równym `navbar`. Dzięki temu jQuery Mobile będzie traktował jego zawartość jako przyciski paska nawigacji. Podstawowa konstrukcja wygląda tak:



Umieszczamy przyciski w pasku nawigacji

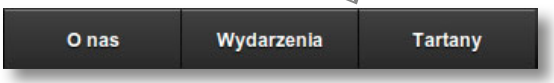
Zamiast odsyłaczy umieszczonych na pionowej liście zamierzamy uzyskać pasek nawigacji z przyciskami, które będą kierowały do głównych sekcji Tartanatora. To rozwiązanie jest często stosowane w aplikacjach.

jQuery Mobile automatycznie przekonwertuje listę elementów umieszczoną w bloku `navbar` na przyciski.

Upiększamy pasek narzędzi

Wszystko pięknie, ale przyciski na pasku narzędzi są jakieś takie nieciekawe. jQuery Mobile oferuje około dwudziestu domyślnych ikon, z których możemy w prosty sposób skorzystać. Wrzućmy kilka ikon za pomocą atrybutu `data-icon`.

Ujęcie pierwsze: tak wyglądają przyciski.



Wyglądają co prawda jak przyciski, ale mogłyby być trochę ciekawsze.

Listę dostępnych domyślnych ikon możesz znaleźć w dokumentacji frameworku jQuery Mobile.

Atrybut data-icon wskazuje ikonę, która ma zostać wyświetlona w przycisku.

```
<div data-role="footer" data-position="fixed">
   <div data-role="navbar">
      <ul>
      <li><a href="index.html" data-icon="info">O nas</a></li>
      <li><a href="findevent.html" data-icon="star">Wydarzenia</a></li>
      <li><a href="tartans.html" data-icon="grid">Tartany</a></li>
      </ul>
   </div><!-- /navbar -->
</div><!-- /footer -->
```

index.html

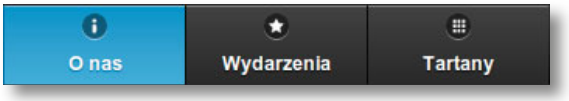
Druga sprawa to zaznaczenie sekcji, którą aktualnie przegląda użytkownik. W tym celu wystarczy do odpowiedniego odsyłacza dodać klasę `ui-btn-active`. Dzięki temu przycisk aktualnej sekcji zostanie wyświetlony w innej kolorystyce.

Ten kod pochodzi z pliku index.html. W pozostałych plikach stron musisz przypisać tę klasę odpowiedniemu odsyłaczowi.

```
<ul>
<li><a href="index.html" data-icon="info" class="ui-btn-active">O nas</a></li>
<li><a href="findevent.html" data-icon="star">Wydarzenia</a></li>
<li><a href="tartans.html" data-icon="grid">Tartany</a></li>
</ul>
```

index.html

Finalizowanie pracy nad strukturą

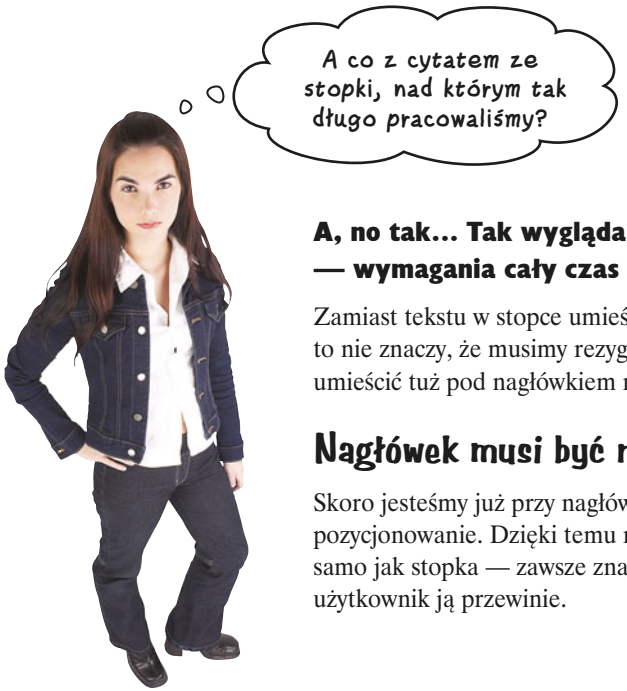


Wygląd paska nawigacji po wprowadzeniu zmian w kodzie strony

Sprawdźmy naszą listę:

- [x] Musimy przekonwertować odsyłacze z głównej strony na stały pasek narzędzi z ikonami.
- [x] Trzeba wyrzucić osobną stronę O nas, ponieważ jest zbędna. Warto pomyśleć o zmianie głównej strony *de facto* na stronę O nas.
- [ ] Nagłówek musi być cały czas na górze strony.

Teraz musimy tylko trochę popracować nad nagłówkiem...



A, no tak... Tak wygląda życie w sieci — wymagania cały czas się zmieniają!

Zamiast tekstu w stopce umieściliśmy pasek nawigacji, ale przecież to nie znaczy, że musimy rezygnować z tego cytatu. Możemy go umieścić tuż pod nagłówkiem na głównej stronie (ale tylko tam).

Nagłówek musi być na górze każdej strony

Skoro jesteśmy już przy nagłówku, poprawimy trochę jego pozycjonowanie. Dzięki temu nagłówek będzie się zachowywał tak samo jak stopka — zawsze znajdzie się na górze strony, nawet jeśli użytkownik ją przewinie.

```
<div data-role="header" data-position="fixed">
    <h1>Tartanator</h1>
</div><!-- /header -->
<div data-role="header" data-theme="b" class="forrit">Bring forrit the tartan!</div>
```

index.html

Jazda próbna

Ach, ten proces iteracyjny... To zdecydowanie najlepszy sposób na projekty. Zaktualizujmy Tartanator.

1. **Zaimplementuj nowy pasek nawigacji w stopce.**
W plikach *index.html*, *findevent.html* i *tartans.html* zamień kod nagłówka na ten, który przygotowaliśmy na stronie 250.

   Nie zapomnij dodać klasy `ui-btn-active` do odpowiednich odsyłaczy (zależnie od pliku).

2. **Wypozycjonuj nagłówek na górze strony.**
W pliku każdej strony do nagłówka dodaj atrybut `data-position`.

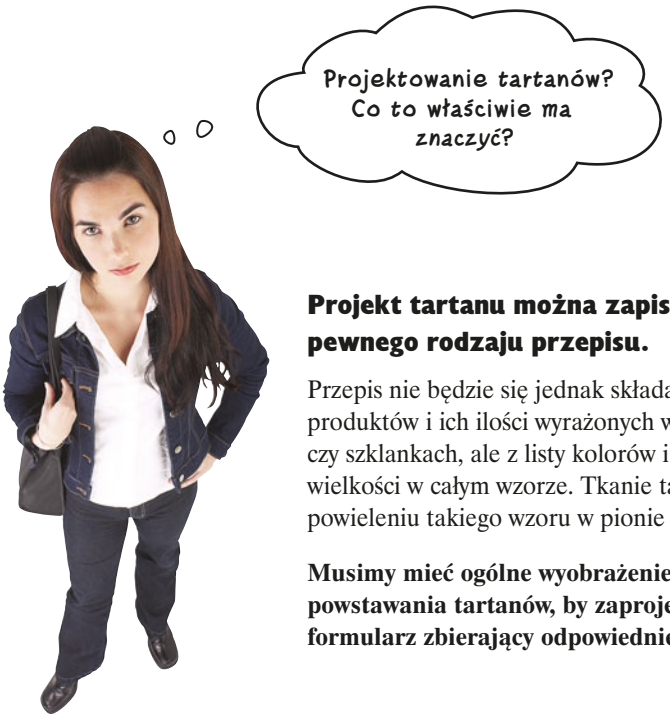
3. **Na głównej stronie (w pliku index.html) umieść cytat.**
Pod nagłówkiem w pliku *index.html* (tylko w nim) dodaj cytat.

Zarówno nagłówek, jak i stopka są już odpowiednio wypozycjonowane. Cały czas są wyświetlane na górze i na dole strony, nawet jeśli użytkownik ją przewinie.

Czas na przygotowanie formularza do tworzenia tartanów

Kolejnym krokiem w projekcie jest stworzenie prototypu formularza, który pozwoli użytkownikom na projektowanie w mobilnej przeglądarce własnych tartanów.

- [x] **Zbudować strony prezentujące treści i strukturę witryny.**
- [x] **Stworzyć listę tartanów.**
- [ ] **Zaprojektować prototyp formularza służącego do tworzenia tartanów.**
  Ewan chce, by użytkownicy mogli projektować własne wzory tartanów za pomocą interfejsu przypominającego aplikacje mobilne.



Projekt tartanu można zapisać w postaci pewnego rodzaju przepisu.

Przepis nie będzie się jednak składał z listy produktów i ich ilości wyrażonych w gramach, łyżkach czy szklankach, ale z listy kolorów i ich względnej wielkości w całym wzorze. Tkanie tartanów polega na powieleniu takiego wzoru w pionie i poziomie.

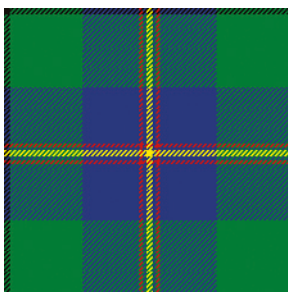
Musimy mieć ogólne wyobrażenie na temat powstawania tartanów, by zaprojektować formularz zbierający odpowiednie informacje.

Przepisy na wzory tartanów

Spójrz na poniższy przykład. Aby stworzyć tartan klanu Carmichael, potrzebny jest następujący wzór:

Każdy składnik wzoru to para kolor – rozmiar.

Kolor	Rozmiar (Liczba ściegów)
czarny	6
zielony	72
niebieski	56
czerwony	4
niebieski	4
żółty	6



Tartan klanu Carmichael

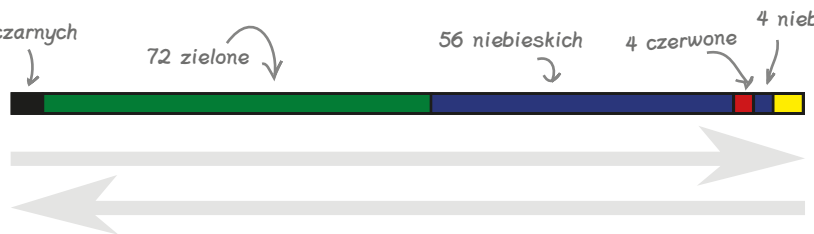
Zróbmy to według przepisu

Podczas tkania tartanu wzór jest przeglądany w stałej kolejności (my będziemy „tkać” z pikseli, a nie z wełny).

W przypadku tartanu klanu Carmichael po 6 czarnych ściegach pojawiają się 72 zielone ściegi, a po nich 56 niebieskich i tak dalej.

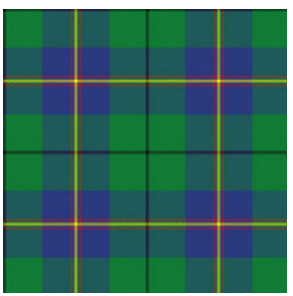
Kiedy zostanie zakończone tkanie ostatniego koloru z wzoru (w naszym przykładzie po sześciu żółtych ściegach), wzór jest odwracany (niebieski, czerwony, niebieski, zielony). Po osiągnięciu pierwszego koloru z wzoru (czarnego) cykl się powtarza. Wzór jest tkany w poziomie i pionie (osnowa i wątek), dzięki czemu powstaje charakterystyczny wzór tartanu.

Pierwszy i ostatni kolor (czarny i żółty) nie są powtarzane. We wzorze te kolory są nazywane osiami.



To ten sam wzór co wyżej, ale w układzie poziomym, a nie pionowym.

Kiedy różne kolory nakładają się w pionie i poziomie, powstaje charakterystyczny wzór.



Musimy zaprojektować formularz, w którym użytkownik będzie mógł definiować składniki za pomocą par różnych kolorów i ich rozmiarów.

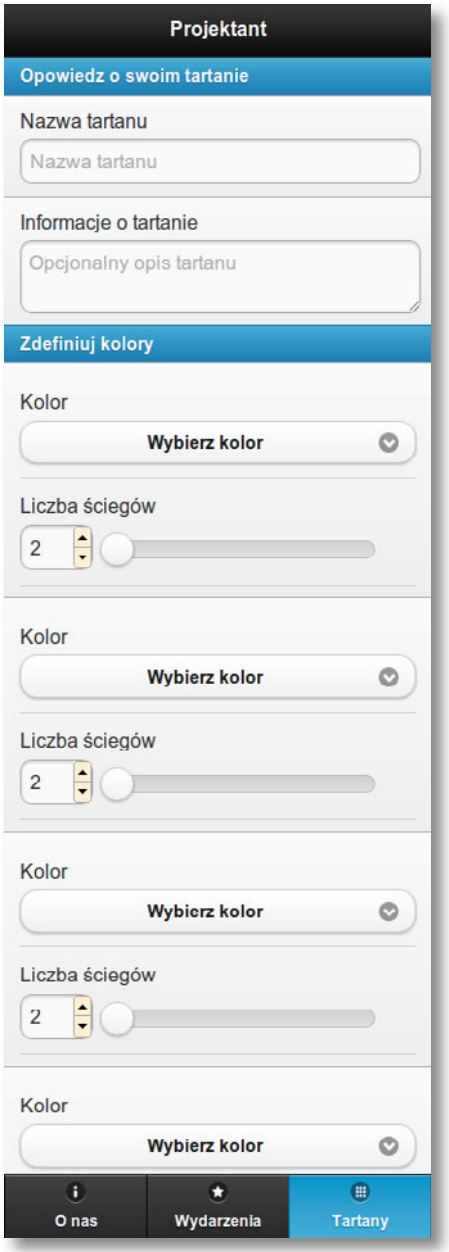
Tłumaczymy wzory tartanów na formularz

Rzuć okiem na prototyp formularza, który zamierzamy stworzyć. Potrzebne nam będą pola umożliwiające użytkownikom definiowanie par kolor – rozmiar, które tworzą przepis na tartan.

Podstawowe informacje na temat tworzonego tartanu: nazwa i opcjonalny opis.

Pola dla par kolor – rozmiar umożliwią użytkownikom zdefiniowanie listy składników tartanu.

Takich kompletów pól jest więcej.



W porządku, mniej więcej wiemy, jaki formularz chcemy zbudować. Na co czekamy? Do roboty!

Ćwiczenie

Utwórz pustą stronę wykorzystującą jQuery Mobile, która będzie zawierała formularz. Możesz w tym celu skorzystać na przykład z pliku *tartans.html*. Użyj tego samego nagłówka i stopki, ale zmień tytuł i nagłówek na „Projektant". Plik zapisz pod nazwą *build.php*.

Tworzymy formularz w HTML5

Zastosujemy standardowy formularz napisany w HTML5. Framework jQuery Mobile przekształci go do postaci bardziej odpowiedniej dla przeglądarek mobilnych. W rozdziale 7. zrobimy z tego formularza jakiś użytek, a także uzupełnimy go o elementy interaktywności za pomocą JavaScriptu oraz dodamy podgląd wzoru projektowanego tartanu.

Na razie zajmiemy się położeniem fundamentów pod formularz, który ma działać w prawie wszystkich mobilnych przeglądarkach.

Przepis na tartan

- Nazwa tartanu
- Opis tartanu (opcjonalny)
- Seria par kolor – rozmiar

Te informacje musi zebrać formularz.

Struktura formularza

Chcemy, by użytkownicy musieli podać nazwę swojego tartanu i mogli dodać jego opis. Następnie mogą zdefiniować pary kolor – rozmiar tworzące wzór. Formularz będzie się składać z głównych sekcji: górnej, która będzie zawierać dane na temat tartanu, oraz głównej z definicjami kolorów oraz ich rozmiarów.

Struktura formularza gotowa do użycia

Skorzystamy z widoku listy opartego na elemencie <ul>, co pomoże nam zachować prawidłowy układ formularza.

Atrybuty identyfikatorów przydadzą się później, podczas uzupełniania o skrypty.

```
<div data-role="content">
    <form id="tartanator_form">
      <ul data-role="listview" id="tartanator_form_list">
        <li data-role="list-divider">Opowiedz o swoim tartanie</li>
        <li data-role="list-divider">Zdefiniuj kolory</li>
      </ul>
    </form>
</div><!-- /content -->
```

build.php

Te dwa elementy rozdzielające wyodrębniają dwie sekcje strony.

Dodajemy podstawowe pola

Pola formularza odpowiadające za dane dotyczące tartanu, czyli za nazwę i opis, są naprawdę proste. Użyjemy pól tekstowych `input` i `textarea`. W kodzie może Cię zastanowić jedynie zastosowany przez nas atrybut `placeholder`, który został wprowadzony w HTML5. Poza tym użyliśmy wskazanych ze względu na semantykę i dostępność elementów `label`.

Atrybut placeholder umożliwia podanie początkowej wartości w polach tekstowych. Ten tekst zastępczy znika, gdy użytkownik kliknie pole i zmieni jego zawartość.

Pola formularza gotowe do użycia

```
<li data-role="list-divider">Opowiedz o swoim tartanie</li>
    <li data-role="fieldcontain">
       <label for="tartan_name">Nazwa tartanu</label>
       <input type="text" name="name" id="tartan_name"
placeholder="Nazwa tartanu" />
    </li>
    <li data-role="fieldcontain">
       <label for="tartan_info">Informacje o tartanie</label>
       <textarea cols="40" rows="8" name="tartan_info" id="tartan_info"
placeholder="Opcjonalny opis tartanu"></textarea>
    </li>
<li data-role="list-divider">Zdefiniuj kolory</li>
```

build.php

jQuery Mobile potrzebuje podpowiedzi

Być może zauważyłeś, że każde pole znajduje się w elemencie z atrybutem `data-role` ustawionym na `fieldcontain`. Jest to wskazówka dla jQuery Mobile, że w tych elementach znajdują się pola formularza, o które należy szczególnie zadbać.

Formularz budujemy na bazie listy <ul>, więc atrybut fieldcontain ustawiamy w każdym elemencie <li> tej listy, który zawiera pole formularza.

Dodawanie kolorów umożliwiają użytkownikom listy w listach

Każdy element wzoru tartanu jest kombinacją koloru i jego rozmiaru (szerokości). W układzie formularza zgrupujemy pola dla tych dwóch cech w jednym elemencie `<li>`. Ponieważ musimy wziąć pod uwagę przeglądarki mobilne, które nie obsługują JavaScriptu, generujemy sześć grup pól za pomocą PHP (dzięki czemu nie musimy wpisywać dużo kodu!).

Zaczynamy od czegoś takiego:

Pola dla każdej pary kolor – rozmiar będą przechowywane w jednym elemencie <li>.

```
<li data-role="list-divider">Zdefiniuj kolory</li>
  <?php for ($i = 0; $i < 6; $i++): // pola dla 6 kolorów ?>
  <li class="colorset">

  </li>
  <?php endfor; ?>
</ul>
```

Ta pętla for w PHP generuje sześć jakichkolwiek elementów HTML, których kod znajdzie się w pętli.

Tu umieścimy pola koloru i rozmiaru.

build.php

Nie istnieją głupie pytania

P: Dlaczego nie dodaliśmy atrybutu `data-role` równego `fieldcontain` do tego elementu `<li>`? Przecież w nim też znajdą się pola formularza.

O: Każdy z tych elementów `<li>` (`li.colorset`) będzie zawierał tak naprawdę *dwa* pola. Kiedy będziemy je dodawać, każde z nich umieścimy w osobnym bloku `<div>` z atrybutem `data-role` ustawionym na `fieldcontain`.

P: Nie rozumiem. Jakie elementy powinny posiadać atrybut `data-role` równy `fieldcontain`?

O: Jakiekolwiek elementy, ale warunkiem jest, by bezpośrednio w nich znajdowały się pola formularza. Może to być `<li>`, `<div>` czy `<fieldset>` albo nawet `<p>`.

Dzięki atrybutowi `data-role` równemu `fieldcontain` jQuery Mobile grupuje pole wraz z etykietą z elementu `label`. Z tego względu element posiadający ten atrybut może zawierać tylko jedno pole (i jego etykietę).

P: Czy do pracy nad tą częścią projektu potrzebuję PHP?

O: Tak! Kiedy zajmiemy się bardziej funkcjonalnymi elementami Tartanatora, będziemy potrzebować PHP do wielu zadań.

Pary kolor – rozmiar: pole wyboru koloru

W każdym zestawie do definicji pojedynczego koloru znajdują się pole koloru (`<select>`) oraz rozmiaru (`<input>`). Klasy `color-input` oraz `size-input`, które są przypisane do bloków zawierających te pola, będą potrzebne później, gdy będziemy dodawać skrypty JavaScript.

To oczywiste, że chcemy więcej możliwości wyboru koloru niż tylko czarny i biały. Jak to zrobić, dowiesz się za chwilę.

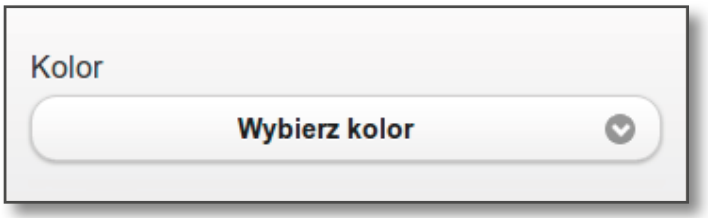
Byłoby świetnie, gdybyśmy mogli zastosować pole wyboru koloru wprowadzone w HTML5, ale niestety nie jest jeszcze obsługiwane w większości przeglądarek.

```
<li class="colorset">
   <div data-role="fieldcontain" class="color-input">
   <label class="select" for="color-<?php print $i ?>">Kolor</label>
      <select name="colors[]" id="color-<?php print $i ?>" >
         <option value="">Wybierz kolor</option>
         <option value="#000000">czarny</option>
         <option value="#ffffff">biały</option>
      </select>
   </div>
</li>
```

Pamiętaj, że to jest wnętrze pętli for w kodzie PHP. W ten sposób przypisujemy każdemu polu unikalny identyfikator.

build.php

Pole wygląda w ten sposób w nowoczesnych przeglądarkach opartych na silniku WebKit.



Para kolor – rozmiar: pole rozmiaru

Zastosowaliśmy nowy rodzaj pola wejściowego, wprowadzony w HTML5 — `range`. Jest ono wspierane przez stale rosnącą rzeszę nowoczesnych przeglądarek (ale niestety nie przez wszystkie). W przeglądarkach, które obsługują to pole, pojawi się kontrolka suwaka, a w pozostałych — pole tekstowe. Jak zwykle framework jQuery Mobile dostosowuje kontrolkę do możliwości urządzeń mobilnych.

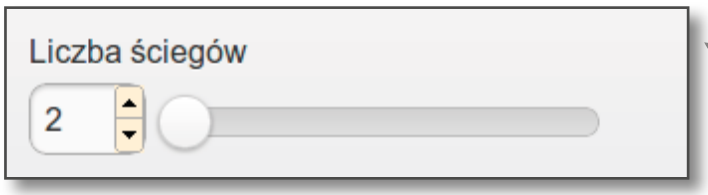
Ten fragment musi się znaleźć bezpośrednio po kodzie pola <select>.

Atrybuty min i max odpowiadają za to, czego można się po nich spodziewać, czyli definiują minimalną i maksymalną wartość dla pola.

Wzory tartanów mogą się składać tylko z parzystej liczby szwów. W tej sytuacji atrybut step (krok) jest niezastąpiony.

```
        </select>
    </div>
    <div data-role="fieldcontain" class="size-input">
        <label for="size-<?php print $i ?>">Liczba ściegów</label>
        <input id="size-<?php print $i ?>" type="range" min="2" step="2" max="72"
autocomplete="off" name="sizes[]" value="2" />
    </div>
</li>
```

build.php



Pole wygląda w ten sposób w nowoczesnych przeglądarkach opartych na silniku WebKit.

Jazda próbna

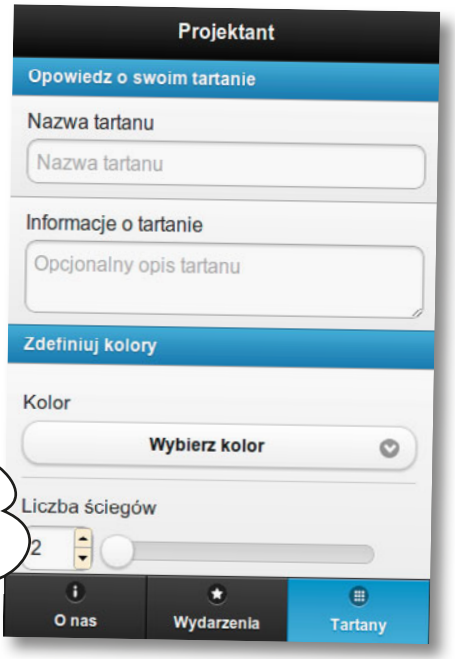
Nadszedł czas, by w końcu poskładać wszystko w całość i stworzyć mobilny (chociaż jeszcze niefunkcjonalny) formularz.

1. **W pliku build.php umieść kod pól formularza.** Skorzystaj z fragmentów kodu ze stron 256 – 260, aby stworzyć podstawowy formularz.

2. **W polach wyboru koloru dodaj więcej opcji.** Możesz użyć wartości kolorów (w postaci szesnastkowej), które zamieściliśmy w pliku *extras/color-list.txt*, ale możesz też zdefiniować własne.

Zapisz zmiany w pliku *build.php*, a następnie załaduj stronę w przeglądarce mobilnej.

Wygląda (i działa) jak aplikacja.



Musimy dodać odsyłacz do formularza oraz przycisk powrotu. Do dzieła!

Odsyłacz do formularza

Formularz ma być częścią sekcji *Tartany*. W związku z tym musimy umożliwić użytkownikom przejście do formularza z głównej strony Tartany (tej z długą listą tartanów). Najlepszym rozwiązaniem jest chyba dodanie przycisku do nagłówka:

```
<div data-role="header" data-position="fixed">
   <h1>Popularne tartany</h1>
   <a href="build.php" data-role="button" data-icon="plus"
class="ui-btn-right" data-theme="b">Nowy</a>
</div><!-- /header -->
```

Dodajemy do przycisku ikonę plusa.

tartans.html

Ustawiamy zestaw kolorystyczny „b", aby przycisk się wyróżniał.

Dzięki dodaniu klasy ui-btn-right przycisk zostanie umieszczony z prawej strony (zamiast z lewej, co jest domyślnym zachowaniem).

Możliwość powrotu

W pliku z formularzem (*build.php*) dodajemy przycisk powrotu:

```
<div data-role="header" data-position="fixed">
   <a href="tartans.html" data-rel="back" data-icon="back"
data-role="button">Wstecz</a>
   <h1>Projektant</h1>
</div><!-- /header -->
```

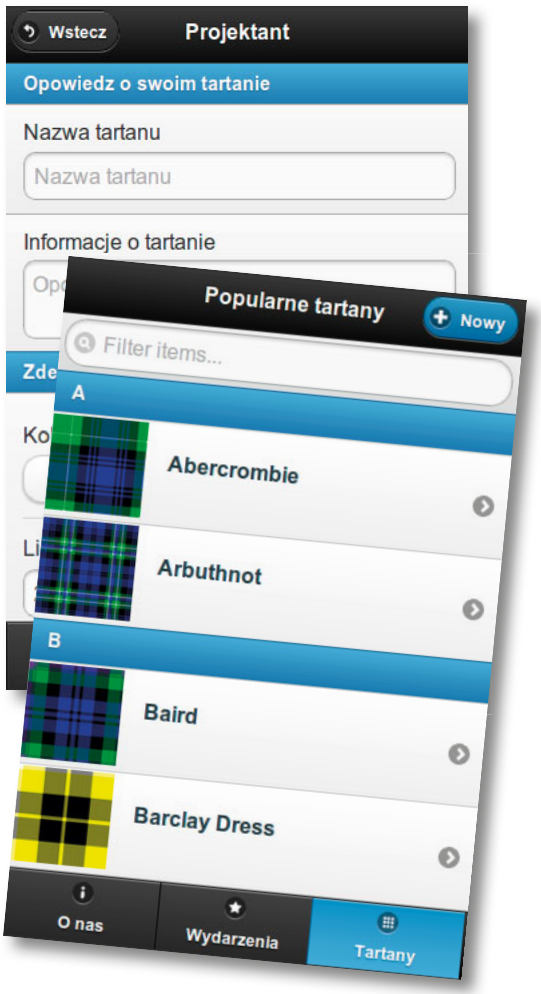
build.php

Dodanie atrybutu `data-rel="back"` informuje jQuery Mobile, by traktował ten odsyłacz jako odsyłacz powrotu. Kiedy tylko jest to możliwe, użytkownik zostanie odesłany do ostatniej lokalizacji zapisanej w historii przeglądania. W przeglądarkach, które tego nie obsługują, zostanie zastosowany klasyczny atrybut `href`, który wskazuje w tym przypadku plik *tartans.html*.

Jazda próbna

1. **W pliku tartans.html dodaj odsyłacz do formularza.**
W nagłówku w pliku *tartans.html* dodaj przycisk *Nowy*.

2. **Na stronie formularza (build.php) dodaj przycisk powrotu.**
W nagłówku strony Projektant dodaj przycisk powrotu.



- [ ] **Zbudować strony prezentujące treści i strukturę witryny.**
Musimy stworzyć główne sekcje i strony oraz podstawową strukturę witryny.

- [ ] **Stworzyć listę tartanów.**
W pierwszej fazie utworzymy listę istniejących popularnych wzorów tartanów. Listę powinno dać się łatwo przeglądać i — co oczywiste — powinna wyglądać i zachowywać się jak aplikacja, i to w dodatku mobilna.

- [ ] **Zaprojektować prototyp formularza służącego do tworzenia tartanów.**
Ewan chce, by użytkownicy mogli projektować własne wzory tartanów za pomocą interfejsu przypominającego aplikacje mobilne. Ponieważ chce jak najszybciej wiedzieć, jak to może wyglądać, dostarczymy mu prototyp.

Super! Zakończyliśmy pierwszą fazę pracy nad Tartanatorem. Teraz czas na opinię klienta.

Pomoc jQuery Mobile

W tym rozdziale ledwie musnęliśmy powierzchnię frameworku jQuery Mobile i skorzystaliśmy jedynie z ułamka wszystkich możliwości. Aby nauczyć się więcej lub zdobyć dodatkowe informacje, zapoznaj się z dokumentacją dostępną na stronie www.jquerymobile.com. Jest bardzo szczegółowa.

CELNE SPOSTRZEŻENIA

- Ludzie chcą aplikacji! Wyraźne wskazanie tego czegoś, co odróżnia witrynę od **aplikacji internetowej**, jest trudne. Witryny pretendujące do miana aplikacji zwykle są bardziej interaktywne od klasycznych stron prezentujących treści. Często dochodzi do asynchronicznego pobierania fragmentów zawartości oraz danych i wstawiania ich do istniejącej struktury DOM, dzięki czemu rzadziej dochodzi do pełnego przeładowania strony.
- **HTML5** to konkretna specyfikacja reprezentująca ewolucję języka znaczników HTML (ang. *HyperText Markup Language*), ale termin *HTML5* jest często używany w szerszym znaczeniu do określenia rodziny technologii, dzięki którym w rozwiązaniach internetowych można uzyskać zachowanie przypominające aplikacje.
- Stworzenie mobilnej aplikacji internetowej od podstaw może być bardzo trudne. Zachęcamy, byś spróbował! Jednak w naszym przypadku lepszym rozwiązaniem było zastosowanie **frameworku**, który ułatwia tworzenie interfejsu użytkownika.
- Jest wiele frameworków nastawionych na mobilne aplikacje internetowe (i z każdym dniem coraz więcej!). Zastosowano w nich różne podejścia i położono nacisk na różne aspekty funkcjonowania.
- **jQuery Mobile** jest popularnym frameworkiem. Jest silnie powiązany z HTML5, dzięki czemu stosunkowo łatwo tworzyć w nim mobilne interfejsy użytkownika z poziomu kodu HTML.
- W pierwszej fazie prac nad Tartanatorem skorzystaliśmy z jQuery Mobile do stworzenia struktury aplikacji. Użyliśmy widoków listy, nagłówków, stopek, pasków nawigacji i elementów formularza.
- Sprawienie, by aplikacja internetowa wyglądała i zachowywała się podobnie jak natywna, jest trudnym zadaniem i wymaga podjęcia wielu przemyślanych decyzji projektowych.
- Przy budowie formularza użyliśmy kilku atrybutów **elementów formularza wprowadzonych w HTML5**. Framework jQuery Mobile pomaga dostosować je do możliwości i ograniczeń mobilnych przeglądarek. Dzięki temu formularz zadziała prawidłowo nawet w przeglądarkach, które nie obsługują niektórych elementów i atrybutów.

Nie istnieją głupie pytania

P: Ile kodu w Tartanatorze to prawdziwy HTML5? Chodzi mi o to, ile ze znaczników i atrybutów to nowe elementy wprowadzone w specyfikacji HTML5?

O: Dosyć często korzystamy z atrybutów `data-*`. To jest nowe. Skorzystaliśmy też z pola formularza typu `range` oraz atrybutów `placeholder`.

P: Do pola range dodałem atrybut `step`, ale mogę wpisać dowolną wartość z zakresu od 2 do 72, nawet liczby nieparzyste. O co chodzi?

O: Niestety może się tak zdarzyć, że w niektórych przeglądarkach widżet `slider` nie obsługuje prawidłowo atrybutu `step`. W rozdziale 7. pokażemy, jak obejść ten problem.

P: Wybór koloru z listy rozwijanej — nie wydaje mi się, żeby to rozwiązanie było najlepsze.

O: W rozdziale 7., kiedy zajmiemy się ulepszaniem formularza za pomocą JavaScriptu, dodamy kod wyświetlający wybrany kolor.

P: Co się stanie w przeglądarkach, które nie obsługują pola typu `range`?

O: Jeśli przeglądarka dobrze wspiera JavaScript i CSS, jQuery Mobile zamieni pole `range` na widżet suwaka (tak naprawdę w tej chwili robi to także w przeglądarkach, które obsługują to pole).

Z kolei jeśli przeglądarka nie wspiera ani pola `range`, ani JavaScriptu (lub jego obsługa jest wyłączona), zamiast tej kontrolki pojawi się najprawdopodobniej zwykłe pole tekstowe.

P: Czasem zdarza się, że nagłówek i stopka błyskają albo ładują się wolniej od pozostałych fragmentów strony. Zdarza się też, że przewijanie działa bardzo wolno. Co się dzieje?

O: Jednym ze sposobów, w jaki jQuery Mobile emuluje zachowanie natywnych aplikacji, jest stałe pozycjonowanie nagłówków i stopek (oczywiście jeśli sam to ustawisz, a tak jest w naszym przypadku). Sposób, w jaki jest to realizowane, zależy w dużej mierze od platformy.

W skrócie: chociaż sytuacja zmienia się bardzo szybko, stałe pozycjonowanie (realizowane w CSS za pomocą `position:fixed`) nie jest najlepiej obsługiwane w przeglądarkach mobilnych. Jest wiele obejść tego problemu, ale żadne z nich nie jest pozbawione wad. Niewielkie spowolnienia i sporadyczne dziwactwa są objawem prób podejmowanych przez framework w celu znalezienia złotego środka między zachowaniem w stylu aplikacji i zgodności ze standardami.

Gratulacje! Właśnie stworzyłeś mobilną aplikację internetową z pomocą HTML5 i jego przyjaciół.

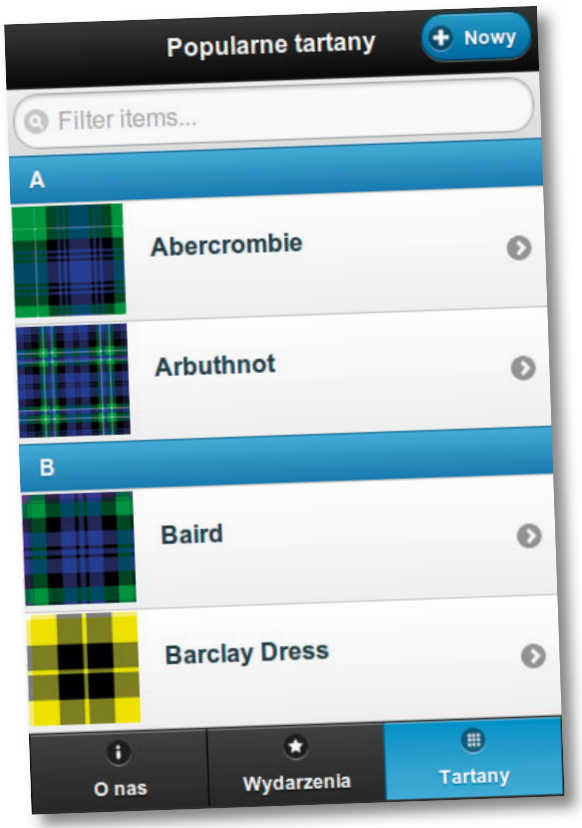
Widzisz? Droga od HTML-a i CSS do mobilnej witryny przypominającej aplikację wcale nie jest taka długa!

Wyjątkowe mobilne aplikacje internetowe

Mobilne aplikacje są jak dzieci w klasie. Wiesz, chodzi o tę mieszankę fascynacji, przekonania o tym, że można zrobić niesamowite rzeczy, ale też tajemniczości i niepohamowanego rozrabiactwa. Staramy się trzymać pod kontrolą tych nadpobudliwych geniuszy, ustalając granice, a także dbając, by ich nie przekraczali. Przychodzi jednak czas zbierania korzyści z naturalnych talentów mobilnych aplikacji internetowych. Możemy zastosować metodę **stopniowego ulepszania**, by dopieścić interfejs na potrzeby lepiej rozwiniętych przeglądarek, a problemy z ciągłością dostępu do internetu rozwiązać za pomocą **trybu offline**. Możemy też wydobyć esencję mobilności, korzystając z **geolokalizacji**. Nie ma czasu do stracenia, zabierzmy się za tworzenie wyjątkowych mobilnych aplikacji internetowych!

Wygląda nieźle...

Po zakończeniu pierwszej fazy projektu Tartanator wygląda już jak prawdziwa mobilna aplikacja internetowa.



Pusta gadanina...

Są przyciski. I pasek nawigacji. Nie zabrakło też fajnych efektów. Ładujemy fragmenty zawartości za pomocą AJAX-a. Redukujemy obciążenie łącza i liczbę żądań oraz ograniczamy przetwarzanie struktury DOM do minimum. Framework jQuery Mobile pomaga nam dostosować elementy formularza przygotowanego w HTML5 do możliwości urządzeń mobilnych.

Stylowa, to prawda. Mobilna — jak najbardziej. Szkopuł w tym, że nic nie robi! Czy aplikacje nie powinny czegoś robić?

Spokojnie. Może się wydawać, że pierwsza faza prac nad projektem była pełna spektakularnych akcji, a tu nic. Jednak trzeba uczciwie przyznać, że położyliśmy całkiem solidne fundamenty pod aplikację.

W tej chwili mamy prostą strukturę opartą na HTML5. Czas przejść na kolejny poziom!

...a tu potrzeba działania!

W drugiej fazie prac będziemy bazować na tym, co udało nam się stworzyć w pierwszej, i zamienimy solidną (chociaż nie do końca funkcjonalną) aplikację internetową w to, co można by nazwać wyjątkową mobilną aplikacją!

Aplikacje mobilne w prawdziwym świecie

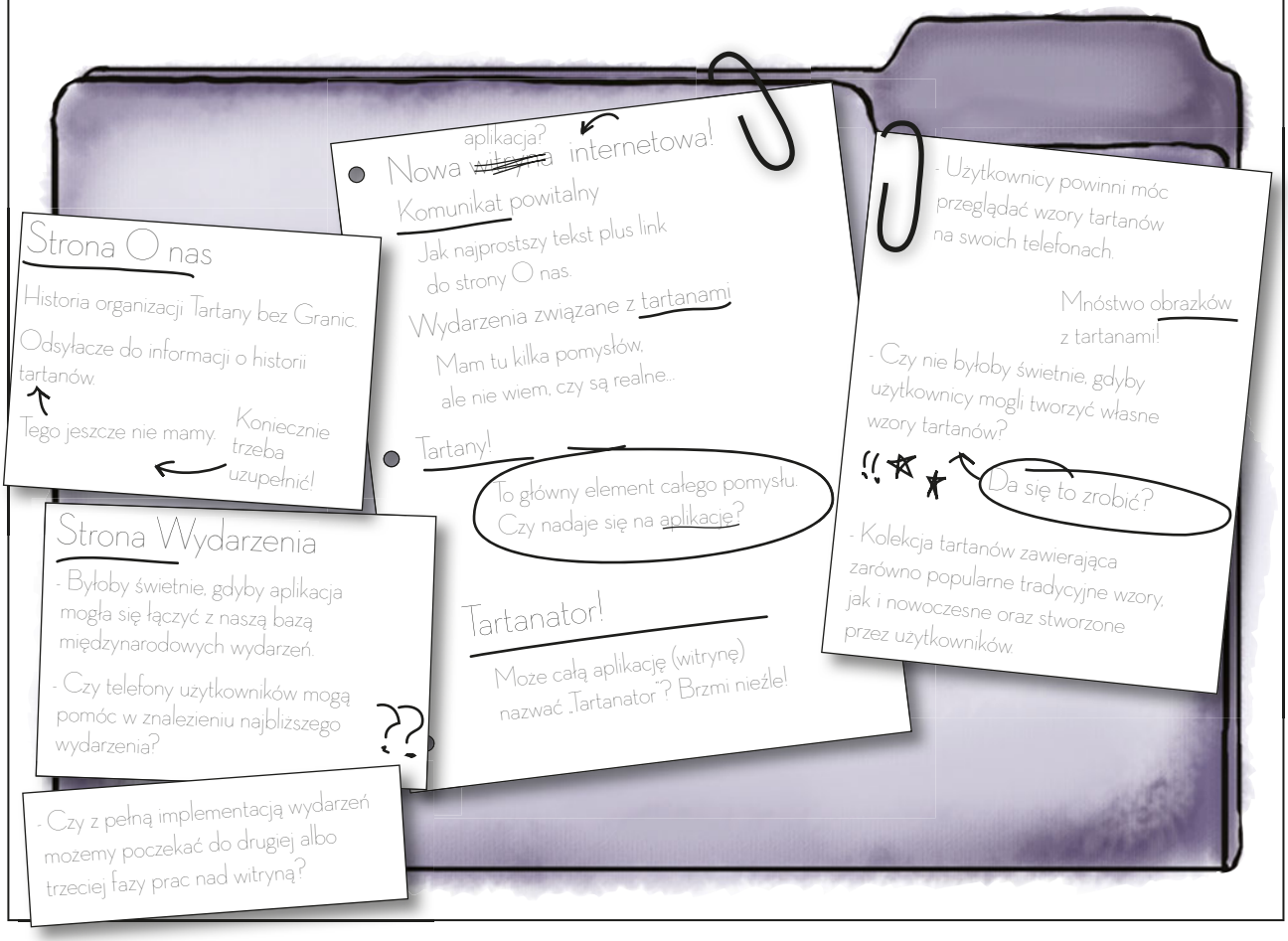
Mobilne aplikacje internetowe, które korzystają z dobrych wzorców mobilnego świata, mają kilka cech wspólnych.

To, co można nazwać *wyjątkowymi mobilnymi aplikacjami internetowymi*, to aplikacje mobilne dostosowane do wymogów prawdziwego świata. Dopasowują się do możliwości konkretnych urządzeń użytkowników, korzystając z techniki **stopniowego ulepszania**. Jeśli chwilowo użytkownik nie ma dostępu do internetu, działają dalej w **trybie offline**. W wielu sytuacjach używają **geolokalizacji** oferowanej przez przeglądarki mobilne, by dostarczać treści powiązane z lokalizacją użytkownika.

Zaostrz ołówek

Co zrobimy, żeby zakończyć drugą fazę prac i sprawić, by Tartanator stał się wyjątkową mobilną aplikacją internetową?

Przede wszystkim musimy się wziąć za to, co zostało do zaimplementowania zgodnie ze specyfikacją, którą podał Ewan w rozdziale 6. Zaznacz te wymagania, którymi się jeszcze nie zajęliśmy. Za chwilę sformułujemy na ich podstawie kluczowe zadania dla drugiej fazy projektu.

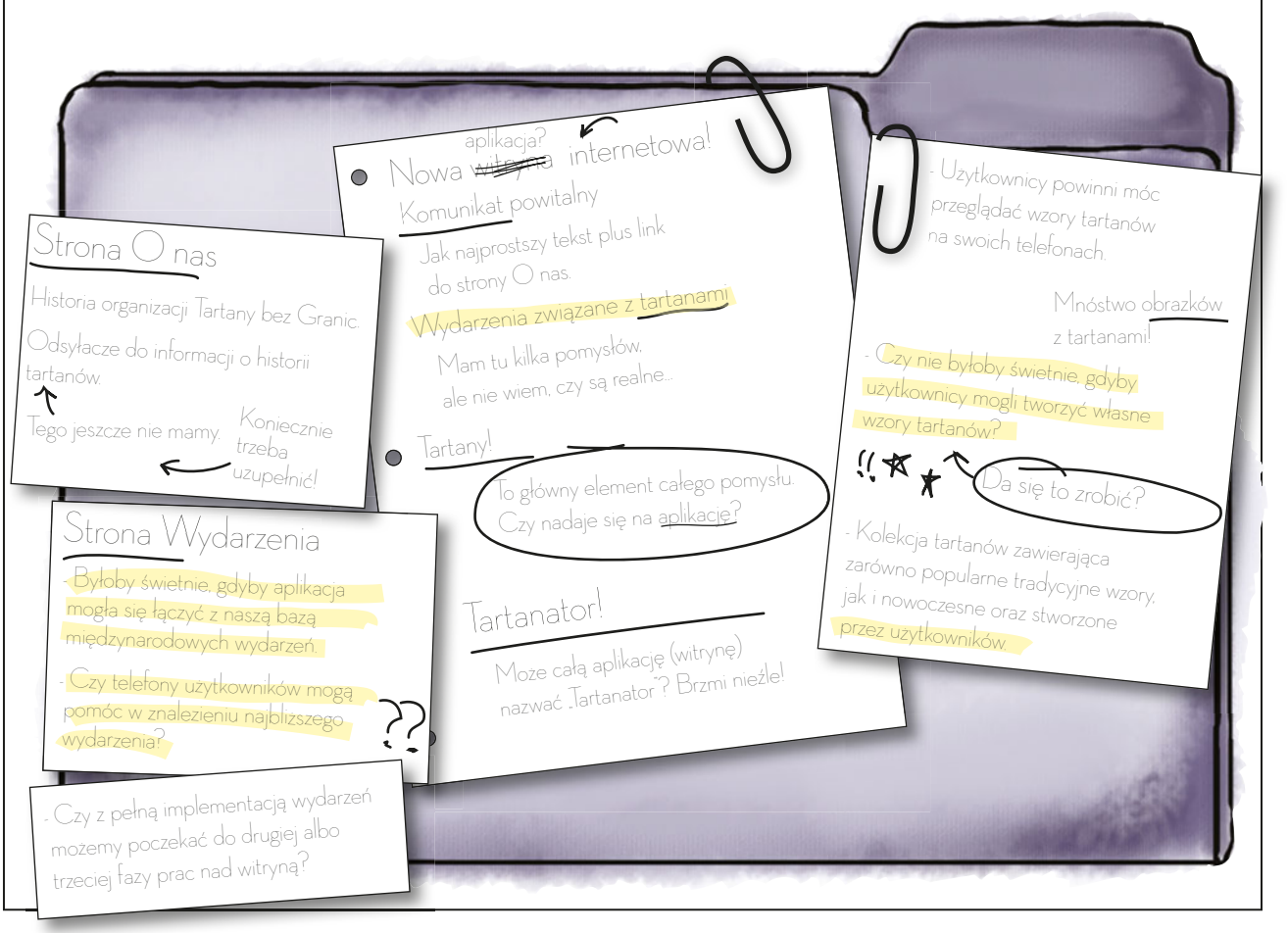


Zaostrz ołówek. Rozwiązanie

Kluczowe zadania drugiej fazy projektu tak naprawdę służą osiągnięciu dwóch głównych celów:

1. **Dokończenie prac nad edytorem własnych tartanów.**
Mamy już prototyp formularza. Teraz musimy przejść do konkretów i sprawić, by formularz zaczął działać. Musimy też ulepszyć formularz, by był bardziej użyteczny w lepszych smartfonach (przy okazji dodamy kilka wodotrysków).

2. **Stworzenie dynamicznej strony Wydarzenia, w której da się wyszukiwać i która „zna" lokalizację użytkownika.**
Musimy umożliwić wyszukiwanie wydarzeń powiązanych z tartanami, które odbywają się najbliżej aktualnej lokalizacji użytkownika. W tym celu skorzystamy ze źródła danych z informacjami o wydarzeniach oraz geolokalizacji.



Łukasz: To pierwsze duże wymaganie wygląda groźnie, nie sądzicie? Może rozbijemy je na mniejsze zadania?

Kuba: Ale chwila, zanim się tym zajmiemy, co z obiecanymi tekstami na stronę O nas i opisem historii tartanów?

Łukasz: A, właśnie! Sytuacja wygląda tak, że żona człowieka, który miał napisać te teksty dla Tartanów bez Granic, właśnie urodziła. Świeżo upieczony tatuś obudził się, nomen omen, z ręką w nocniku — nic wcześniej nie napisał, a teraz odkrył, że dziecko jest bardziej zajmujące, niż sądził, więc w najbliższym czasie nie będzie mógł brać udziału w projekcie. Niespodzianka, co? Jednym słowem, tę sprawę zostawiamy na później.

Kuba: No cóż, przynajmniej jedno zadanie mniej. Pomówmy o pracach nad edytorem tartanów. Nie chcę się chwalić, ale już to trochę przemyślałem. Pomajstrowałem nawet przy formularzu i dodałem trochę JavaScriptu.

Przemek: Świetnie. Formularz będzie śliczny i w ogóle, ale musimy się przecież jeszcze zająć częścią serwerową, która wykona całą ciężką pracę...

Kuba: Oj tam, część serwerowa... Uwierzcie, ulepszenie interfejsu formularza to wielka rzecz.

Przemek: Jasne, jasne. No dobra — zgłaszam się na ochotnika do pisania skryptów PHP. Kto się zajmie sprawą wydarzeń?

Kuba: A czy moglibyśmy się skupić tylko na tworzeniu nowych tartanów? Kiedy tylko z tym skończymy, wrócimy do tematu wyszukiwania wydarzeń. Mój mózg pozwala mi na skupienie się tylko na jednym zadaniu naraz.

Przemek: Może tak być, ale pod warunkiem że uporamy się z tym szybko. Mamy dosyć napięty harmonogram.

Do zrobienia — implementacja edytora tartanów

To nasz plan działania związany z edytorem tartanów.

Zaczniemy od tego!

- ☐ Ulepszyć istniejący formularz, by mógł wykorzystać możliwości oferowane przez nowsze przeglądarki mobilne.
- ☐ Napisać kod strony serwerowej, który jest odpowiedzialny za przetwarzanie danych z formularza oraz generowanie zasobów (obrazów, kodu HTML itp.) na potrzeby tartanów utworzonych przez użytkownika.
- ☐ Zapewnić możliwość pracy w trybie offline w tej części aplikacji.

Na miejsca, gotowi, ulepszać!

Zapnijcie pasy! Zamierzamy w rekordowo krótkim czasie wprowadzić sporo ulepszeń do formularza Tartanatora.

Zrobiliśmy, co było trzeba

Zaprojektowaliśmy formularz, który spełnia minimalne wymagania. Żaden użytkownik nie pozostanie na lodzie (no dobra, może jakiś by się znalazł). Zobacz, jak wygląda formularz w przeglądarce z wyłączoną obsługą JavaScriptu.

Teraz możemy ulepszać

Co prawda formularz działa na każdym sprzęcie, ale ma kilka drobnych wad. Na przykład sześć pól z rozwijaną listą do wyboru koloru sprawdzi się w przypadku lepszych urządzeń, ale już niekoniecznie na tych nie tak dobrych.

Skoro podstawowa wersja formularza działa, możemy się zająć dodawaniem ulepszeń, dzięki którym korzystanie z niego na smartfonach będzie prawdziwą przyjemnością.

Pierwszy krok do wyjątkowych mobilnych aplikacji internetowych: dopracuj interfejs użytkownika na potrzeby przeglądarek, które go dobrze wspierają.

Nie wpadaj w panikę na myśl o własnym widżecie. JavaScript już się nie może doczekać, by Ci w tym pomóc.

Kilku genialnych programistów frontendowych opracowało już te ulepszenia. To ciekawe, że kiedy trzeba pracować nad mobilnymi aplikacjami internetowymi, wszyscy są tacy podekscytowani i produktywni.

Pokażemy Ci najistotniejsze elementy (oczywiście będziesz mógł poświęcić trochę czasu na dogłębniejszą analizę kodu), ale sam nie będziesz musiał napisać ani linijki!

Ulepszamy formularz

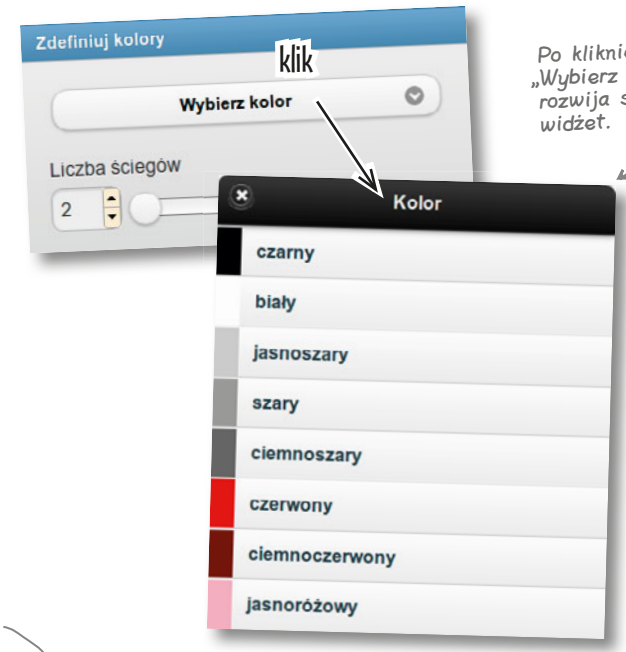
Zamiast sześciu pól — zajmujących dużo miejsca i średnio funkcjonalnych — zastosujemy jeden widżet. Dzięki niemu użytkownicy będą mogli dodać tyle kolorów, ile chcą. Zastosujemy w tym celu skrypt JavaScript, który **usunie wszystkie pola koloru i rozmiaru z wyjątkiem pierwszego zestawu**.

Własny widżet dla pola wyboru koloru

Domyślny interfejs pola wyboru możemy zastąpić widżetem pochodzącym z frameworku jQuery Mobile. Dzięki temu w każdej pozycji listy rozwijanej będziemy mogli wyświetlić próbkę koloru.

Widżet, który zamierzamy zastosować, jest wyświetlany podobnie jak okno dialogowe, tyle że zawiera dostępne opcje koloru.

Tylko jeden zestaw kontrolek koloru i rozmiaru, a nie sześć.



Po kliknięciu pola „Wybierz kolor" rozwija się taki widżet.

Dla zainteresowanych jQuery i JavaScriptem.

Dla maniaków

Jeśli nie jest Ci obca biblioteka jQuery, z pewnością wielokrotnie się spotkałeś z konstrukcją `$(document).ready` (lub nawet z niej korzystałeś). Metoda `ready()` przypomina (w pewnym stopniu) zdarzenie przeglądarki `body.onload` — jest wywoływana po załadowaniu i utworzeniu struktury DOM. Programiści korzystający z jQuery zazwyczaj otaczają swój kod właśnie tą konstrukcją, by został wykonany dopiero wtedy, gdy struktura DOM jest gotowa.

Podczas korzystania z jQuery Mobile (jQM) zwykle nie będziesz używał konstrukcji `$(document).ready`. jQM wprowadza bowiem kilka nowych zdarzeń związanych z ładowaniem strony, a najistotniejsze z nich to `pageinit` oraz `pagecreate`. Zdarzenie `pagecreate` jest zgłaszane po zakończeniu inicjalizacji struktury DOM danej strony przez jQuery i jQM, ale zanim zostaną wyświetlone widżety. Z kolei zdarzenie `pageinit` jest zgłaszane po wyrenderowaniu całej strony, łącznie z widżetami. Pamiętaj, że później wyświetlane strony są często wstawiane do istniejącej struktury DOM, co znaczy, że zdarzenia `pagecreate` i `pageinit` mogą być wielokrotnie zgłaszane podczas pojedynczego ładowania strony.

Framework jQuery Mobile wprowadza także wiele „mobilnych" zdarzeń związanych na przykład z dotykiem, zmianą orientacji urządzenia czy z przejściami.

Nie wszystko wydaje Ci się jasne? Nie przejmuj się, nam też zajęło chwilę ogarnięcie tego. Więcej na ten temat znajdziesz w dokumentacji jQM (*http://jquerymobile.com/demos*).

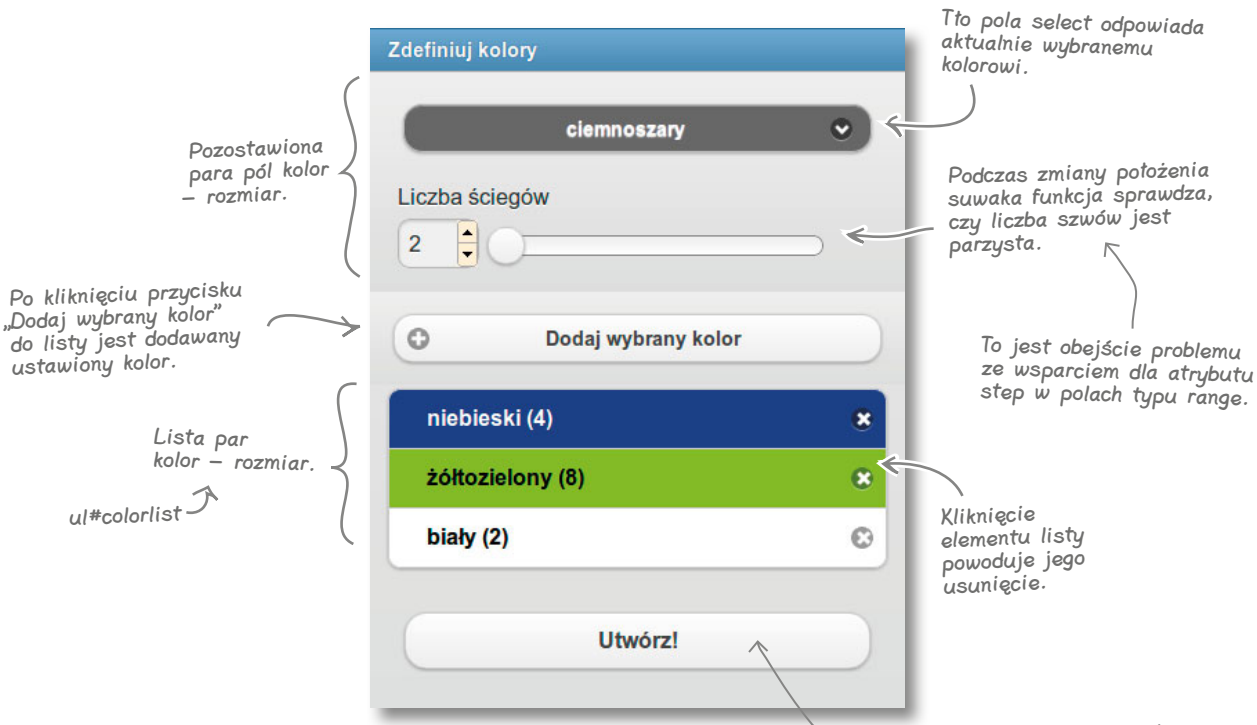
Widżet zarządzający listą kolorów i rozmiarów

Usunęliśmy wszystkie zestawy pól kolor – rozmiar z wyjątkiem jednego. Musimy sprawić, by za pomocą pozostawionej pary pól można było zdefiniować dowolną liczbę kolorów.

Aby to zrobić, musimy dodać nowy przycisk — *Dodaj wybrany kolor*. Po jego kliknięciu wybrany kolor i liczba szwów muszą zostać wstawione do listy (`<ul>`) jako element `<li>`. W tym elemencie umieścimy też ukryte pole formularza, w którym zapiszemy kolor i rozmiar.

Tym zajmie się JavaScript.

Kliknięcie elementu `<li>` znajdującego się na liście ma spowodować jego usunięcie. Z kolei przycisk *Utwórz!* ma służyć do wygenerowania wzoru tartanu na podstawie zdefiniowanych kolorów.



Ulepszona wersja formularza z pliku build.php (fragment)

Weź to w obroty

Mamy dobrą wiadomość. Nie musisz nic sam robić, by udoskonalić formularz. Zrobiliśmy to za Ciebie.

Punktem wyjścia jest katalog *rozdzial7*, w którym umieściliśmy skrypt JavaScript z zaimplementowanymi usprawnieniami. Sam sprawdź! Zobacz też, jak to działa w mobilnej przeglądarce.

Pamiętaj, że formularz znajduje się w pliku build.php.

Nie zajęliśmy się jeszcze stroną serwerową. Kliknięcie przycisku „Utwórz!" na razie nie działa.

Zaglądamy pod przykrywkę

Obiecaliśmy, że nie będziesz musiał napisać ani kawałka kodu JavaScript, ale może uda nam się skłonić Cię do zainteresowania się, co właściwie ten kod robi.

W skrócie: jest kilka istotnych działań, które mają miejsce po załadowaniu strony z formularzem i utworzeniu struktury DOM (zdarzenie `pagecreate`) oraz po zakończeniu operacji przeprowadzanych przez framework jQuery Mobile (zdarzenie `pageinit`).

W tych funkcjach inicjalizujących przypisaliśmy funkcje do różnych zdarzeń (zmiany pola formularza, kliknięć, zatwierdzenia formularza itp.). Dzięki temu udało się wzbogacić interakcję i poprawić funkcjonalność widżetu.

Zaprzyjaźnij się z kodem. Przeznacz trochę czasu na zapoznanie się z aktualizacjami w plikach aplikacji Tartanator, a zwłaszcza ze skryptem *tartanator.js*.

KTO ZA CO ODPOWIADA?

Dopasuj każdą z funkcji JavaScript umieszczonych w pliku *tartanator.js* do realizowanych zadań oraz zdarzeń, które je wywołują. W główkowaniu mogą Ci pomóc komentarze umieszczone w skrypcie.

Zadanie dodatkowe!

Nazwa funkcji	Realizowane zadanie	Zdarzenie, które ją wywołuje
onStitchSizeChange()	Odświeża elementy `<ul>` i `<li>` bieżącej listy kolorów.	`pageinit` oraz `change` (pola wyboru koloru)
styleColorListItem()	Tworzy przycisk dodający parę kolor – rozmiar i wstawia go do struktury DOM.	`click` (przycisk *Dodaj wybrany kolor*)
buildAddButton()	Ustawia kolor tła pola wyboru koloru na aktualnie wybrany.	`pageinit`
onColorListChange()	Tworzy ukryte pole formularza, umieszcza je w nowym elemencie `<li>`, a następnie dołącza ten element do istniejącej listy kolorów.	`click` (element bieżącej listy kolorów) **oraz** po dodaniu nowego koloru do listy
addColor()	Za pomocą CSS umieszcza kolorową ramkę (próbkę) z lewej strony każdego elementu opcji rozwijanej listy kolorów.	`change` (po zaktualizowaniu wartości pola rozmiaru)
setColorSelectStyle()	Sprawdza, czy ustawiony rozmiar jest liczbą parzystą.	`pagecreate`

No tak, to było ulepszanie frontendu

- [x] Ulepszyć istniejący formularz, by mógł wykorzystać możliwości oferowane przez nowsze przeglądarki mobilne. ← *Zrobione!*
- [ ] Napisać kod strony serwerowej, który jest odpowiedzialny za przetwarzanie danych z formularza oraz generowanie zasobów (obrazów, kodu HTML itp.) na potrzeby tartanów utworzonych przez użytkownika. ← *Następne na warsztat!*
- [ ] Zapewnić możliwość pracy w trybie offline w tej części aplikacji.

KTO ZA CO ODPOWIADA? ROZWIĄZANIE

Dopasuj każdą z funkcji JavaScript umieszczonych w pliku *tartanator.js* do realizowanych zadań oraz zdarzeń, które je wywołują. W główkowaniu mogą Ci pomóc komentarze umieszczone w skrypcie.

Nazwa funkcji	Realizowane zadanie	Zdarzenie, które ją wywołuje
onStitchSizeChange()	Sprawdza, czy ustawiony rozmiar jest liczbą parzystą.	`pagecreate`
styleColorListItem()	Za pomocą CSS umieszcza kolorową ramkę (próbkę) z lewej strony każdego elementu opcji rozwijanej listy kolorów.	`change` (po zaktualizowaniu wartości pola rozmiaru)
buildAddButton()	Tworzy przycisk dodający parę kolor – rozmiar i wstawia go do struktury DOM.	`click` (przycisk *Dodaj wybrany kolor*)
onColorListChange()	Ustawia kolor tła pola wyboru koloru na aktualnie wybrany.	`pageinit`
addColor()	Tworzy ukryte pole formularza, umieszcza je w nowym elemencie `<li>`, a następnie dołącza ten element do istniejącej listy kolorów.	`click` (element bieżącej listy kolorów) **oraz** po dodaniu nowego koloru do listy
setColorSelectStyle()	Odświeża elementy `<ul>` i `<li>` bieżącej listy kolorów.	`pageinit` oraz `change` (pola wyboru koloru)

Wszystko o ulepszeniach interfejsu

Wywiad tygodnia:
Ulepszenia: czy to jedynie kosmetyczne poprawki?

Redaktor: Cześć, muszę o to zapytać. Zrobiliśmy właśnie szybką rundkę po ulepszeniach interfejsu za pomocą JavaScriptu w projekcie, który wykorzystuje framework jQuery Mobile. Czy naprawdę było warto?

Ulepszenia interfejsu: To, na co pozwala jQuery Mobile i jak ułatwia tworzenie interfejsów wprost stworzonych dla mobilnych przeglądarek, jest naprawdę niesamowite, ale rozumiem, o co ci chodzi. Chciałbym zarysować szerszy sens tego typu działań.

Redaktor: Słuchamy...

Ulepszenia: Bez względu na to, czy stosujesz jakiś framework, czy piszesz od podstaw, najlepszym rozwiązaniem jest projektowanie z myślą o minimalnych wymaganiach, a następnie wprowadzanie ulepszeń.

Redaktor: Za pomocą JavaScriptu?

Ulepszenia: Otwórz swój umysł... to wykracza daleko poza JavaScript. Mówię tu o sytuacji, w której zaczynasz od zdefiniowania podstawowej funkcjonalności i zawartości, a następnie stopniowo je ulepszasz. W ten sposób wychodzisz naprzeciw możliwościom oferowanym przez różne urządzenia mobilne.

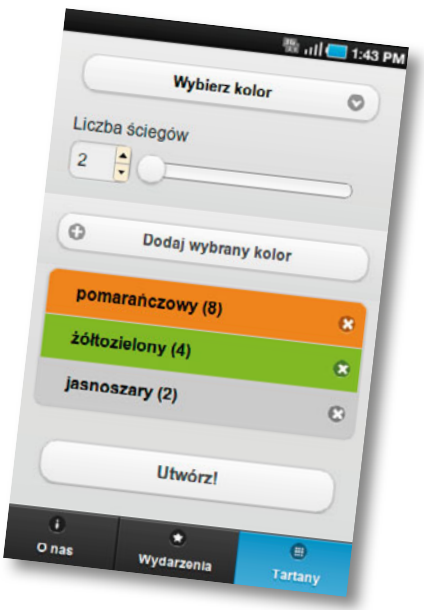
Redaktor: Tak, znam to. Dodaje się zwykle jakieś wodotryski...

Ulepszenia: Chciałbym wierzyć, że mój wkład w sprawę to coś więcej niż tylko zaokrąglone wierzchołki i gradienty. Przed chwilą zaczęliśmy pracę od dosyć ograniczonego i nieszczególnego przypadku, a skończyliśmy na nie dość, że ładnym, to jeszcze funkcjonalnym formularzu.

Redaktor: Zmieniłeś sposób renderowania natywnych elementów formularza. Pole wyboru koloru używa teraz widżetu jQuery Mobile. Czy nie przesadzasz?

Ulepszenia: Zgodzę się, że mieszanie w natywnych kontrolkach formularza jest trochę kontrowersyjne. Nie kłócę się z tym jako ogólną zasadą. Jednak w przypadku pola wyboru koloru nie byłoby możliwe osiągnięcie takiego efektu, czyli kolorowych próbek przy nazwach kolorów, za pomocą tradycyjnych pól wyboru.

Zwróć jednak uwagę, że kod HTML odpowiadający za to pole (`<select>`) nadal jest semantycznie poprawny. Ja je tylko trochę poprawiłem, by stało się bardziej użyteczne. Podsumowując, masz rację, że do zmiany natywnych kontrolek formularza należy podchodzić z dużą ostrożnością.

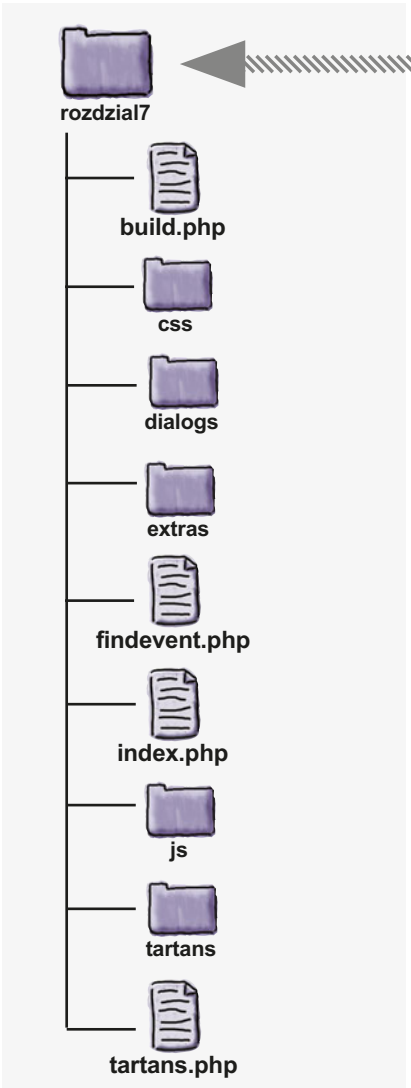


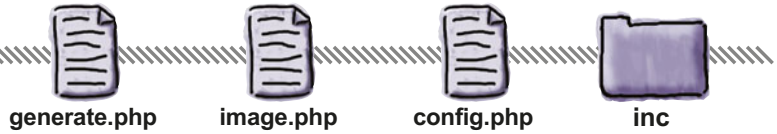
Formularz do tworzenia tartanów w przeglądarce Androida

...a teraz pora na backend

Punktem wyjścia w rozdziale 7. jest struktura plików przedstawiona poniżej. Dołączenie elementów działających po stronie serwera wymaga przeprowadzenia kilku prostych operacji.

Skopiuj całą zawartość katalogu *extras/scripts* do katalogu *rozdzial7*. Po zakończeniu kopiowania w katalogu *rozdzial7* powinny się pojawić trzy nowe pliki: *config.php*, *generate.php* i *image.php* oraz katalog *inc* (zawierający dwa pliki).





Szybki przegląd generowania tartanów

Dane przesłane z formularza (POST)

Na podstawie danych przesłanych przez formularz skrypt generate.php tworzy kilka zasobów.

generate.php

Plik XML

Plik XML zawierający reprezentację wzoru tartanu jest zapisywany w katalogu tartans/data/.

Plik HTML

Plik HTML dla tartanu jest zapisywany w katalogu tartans/.

Ten plik HTML jest tworzony w oparciu o plik szablonu tartan-template.php zapisany w katalogu tartans/.

Plik PNG

Obraz reprezentujący tartan jest zapisywany w formacie PNG.

Dwie twarze skryptu generate.php

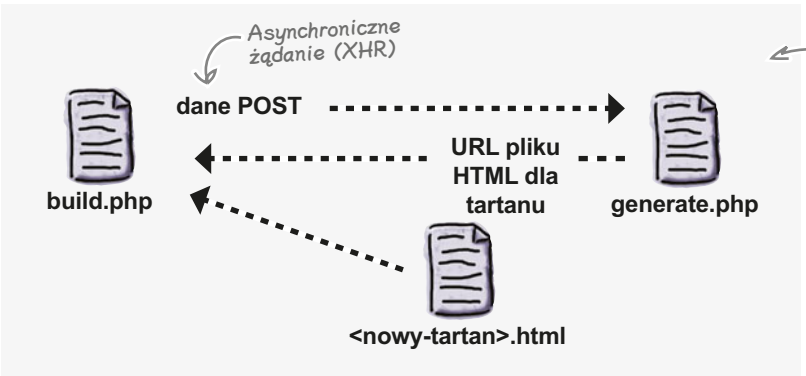
Sposób, w jaki skrypt *generate.php* zachowuje się po utworzeniu zasobów związanych z tartanem, zależy od sposobu jego wywołania.

Postaraliśmy się zapewnić wsparcie zarówno dla przeglądarek obsługujących AJAX-a, jak i tych, które go nie obsługują.

Żądanie za pomocą AJAX-a

W przeglądarkach, które prawidłowo obsługują AJAX-a, skrypt JavaScript znajdujący się w pliku *build.php* przesyła dane z formularza do skryptu *generate.php* za pomocą obiektu `XMLHttpRequest` (w skrócie XHR). Jeśli operacja się powiedzie, skrypt *generate.php* w odpowiedzi przesyła adres URL nowo utworzonego pliku HTML dla zaprojektowanego tartanu. Zawartość tej strony jest następnie umieszczana w strukturze DOM bieżącej strony.

Asynchroniczne żądanie zrealizowane za pomocą XHR



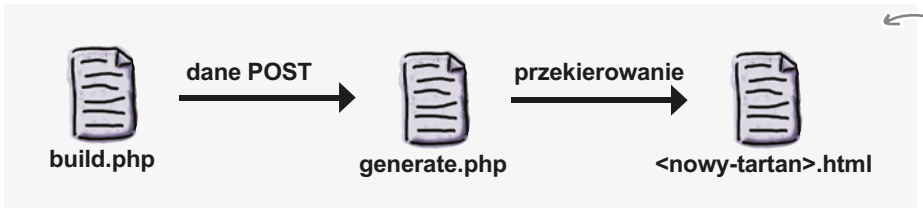
W tej metodzie nigdy nie dochodzi do pełnego przeładowania strony. Zawartość strony nowego tartanu jest wstawiana do struktury DOM strony edytora tartanów.

Formularz przesłany bezpośrednio

W przypadku przeglądarek, które nie obsługują JavaScriptu ani obiektu `XMLHttpRequest`, formularz jest przesyłany do skryptu *generate.php* tradycyjną metodą. Po utworzeniu w skrypcie zasobów związanych z tartanem przeglądarka jest przekierowywana do nowo utworzonej strony tartanu.

Bez względu na metodę podstawowe zadanie skryptu generate.php jest jedno: utworzyć zasoby dla nowego tartanu.

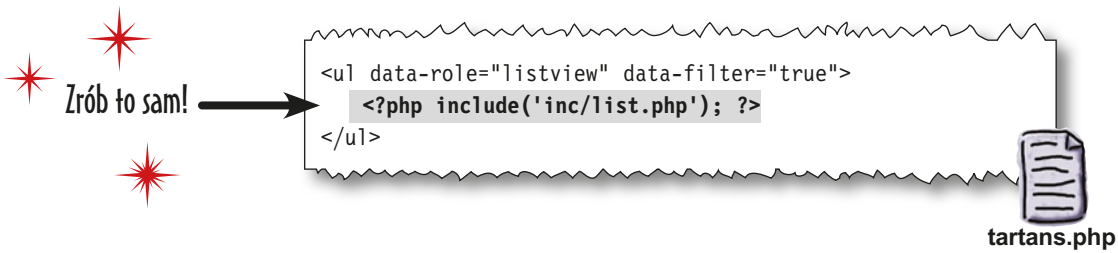
Tradycyjna metoda — przekierowanie do nowo utworzonej strony tartanu



W tej metodzie klient jest przekierowywany do nowo utworzonej strony tartanu (dochodzi do pełnego przeładowania strony).

Jeszcze tylko jedno...

Teraz, kiedy użytkownicy mogą już tworzyć własne wzory, lista tartanów powinna się zmieniać. Umieść poniższy kod w pliku *tartans.php* — dołącza on skrypt generujący elementy `<li>` dla każdego istniejącego tartanu.



Ćwiczenie

Czas utworzyć trochę tartanów! Za pomocą formularza ze strony *build.php* zaprojektuj kilka lub kilkanaście wzorów tartanów. Twoje projekty powinny się pojawić na liście tartanów na głównej stronie (*tartans.php*).

Nie istnieją głupie pytania

P: Pomocy! Coś tu nie działa prawidłowo!

O: Jeśli masz problem z uruchomieniem formularza lub nie udaje Ci się utworzyć nowego tartanu, powinieneś sprawdzić kilka spraw. Przede wszystkim sprawdź, czy istnieje katalog *tartans* i czy masz możliwość zapisywania w jego podkatalogach. Sprawdź też, czy w katalogu *tartans* znajduje się plik *tartan-template.php*. Na koniec powinieneś trzy razy sprawdzić znacznik `<form>` w pliku *build.php* pod kątem atrybutów `action` i `method`.

P: Jak działa strona z listą tartanów?

O: Plik *list.php* dołączony do strony szuka w katalogu *tartans/* plików HTML i tworzy z nich listę. Dla każdego z nich pobiera powiązany z nim plik XML (z katalogu *tartans/data/*) i odczytuje z niego dodatkowe informacje, jak choćby nazwę. Następnie tworzy kod elementu `<li>` dla każdego tartanu, zawierający odsyłacz do strony HTML tartanu.

P: O co dokładnie chodzi z tym plikiem XML?

O: Dane opisujące wzory tartanów postanowiliśmy umieścić w pliku XML z kilku powodów. Po pierwsze, to fajny, prosty i przenośny format danych. Po drugie, dzięki temu unikamy konieczności używania bazy danych (co oznacza mniej pracy dla Ciebie!).

P: A tak przy okazji — czym są te wspaniałe mobilne aplikacje internetowe, o których mówiliście wcześniej? Czy to jakiś standard, inicjatywa, czy coś jeszcze innego?

O: Nic z tych rzeczy. Po prostu tak nazwaliśmy mobilne aplikacje internetowe, które wykorzystują dobrodziejstwa oferowane przez urządzenia mobilne i ich przeglądarki.

Dobra robota — dwa zadania z trzech załatwione

Coraz bardziej zbliżamy się do wyjątkowej mobilnej aplikacji internetowej. Udoskonaliliśmy już interfejs, z czego skorzystają użytkownicy lepszych przeglądarek. Zadbaliśmy też w końcu o to, by aplikacja coś robiła!

- [x] Ulepszyć istniejący formularz, by mógł wykorzystać możliwości oferowane przez nowsze przeglądarki mobilne.
- [x] Napisać kod strony serwerowej, który jest odpowiedzialny za przetwarzanie danych z formularza oraz generowanie zasobów (obrazów, kodu HTML itp.) na potrzeby tartanów utworzonych przez użytkownika.
- [ ] Zapewnić możliwość pracy w trybie offline w tej części aplikacji.

Musimy sobie poradzić jeszcze z tym.

Ale to jeszcze nie wszystko

Teraz musimy się zająć trzecim elementem układanki — Tartanator musi działać offline.

To frustrujące. Chciałem się pochwalić koledze tartanem, który zaprojektowałem, ale był tak słaby zasięg, że nie udało mi się połączyć z siecią. Koniec końców, nie udało mi się nic pokazać.

W przypadku urządzeń mobilnych nie możemy zagwarantować dobrego ani nawet jakiegokolwiek zasięgu sieci.

Musimy zrobić coś, co zagwarantuje działanie aplikacji Tartanator nawet bez aktywnego połączenia z internetem.

Tryb offline to ważna sprawa

Chcąc zapewnić wysoki poziom funkcjonalności i użyteczności aplikacji, musimy się zająć zachowaniem witryny lub aplikacji, w przypadku gdy nie jest dostępne połączenie z internetem.

Musimy się teraz poważnie zastanowić nad tym, co zrobić, by Tartanator działał lepiej w przypadku braku połączenia. Z całą pewnością kilka elementów musi być dostępnych offline.

Możemy użyć tzw. *pliku manifestu*, w którym definiuje się składniki aplikacji dostępne offline.

Zamanifestuj to!

Podręczna pamięć aplikacji (ang. *Application Cache*) jest częścią specyfikacji HTML5. Pozwala na ustalanie, które zasoby są przechowywane na potrzeby trybu offline. Służy do tego ***plik manifestu*** (ang. *cache manifest*).

Plik manifestu jest umieszczany na serwerze, a jego zadaniem jest przekazywanie szczegółowych informacji na temat przechowywania określonych zasobów w pamięci podręcznej przeglądarki użytkownika. W ten sposób będziemy mogli określić, które pliki aplikacji Tartanator mają być dostępne w trybie offline.

Przeglądarki, które obsługują pamięć podręczną aplikacji (często określa się ją w skrócie jako *appCache*), udostępniają obiekt `window.applicationCache` i powiązane z nim zdarzenia, z których możemy korzystać z poziomu JavaScriptu.

Większość nowoczesnych przeglądarek obsługuje ten mechanizm. Niechlubnymi wyjątkami są Internet Explorer 9 i Opera Mini. Zastosowanie pliku manifestu w żaden sposób nie wpłynie jednak na ich pracę — po prostu go zignorują.

WYSIL SZARE KOMÓRKI

Czy umiałbyś wyjaśnić, dlaczego przeglądarka Opera Mini może mieć problem ze wsparciem dla tego mechanizmu?

Prosty przepis na plik manifestu

Aby utworzyć plik manifestu i zastosować go w witrynie lub aplikacji, trzeba wykonać trzy kroki:

1. Napisać plik manifestu.
2. Do znacznika `<html>` wszystkich pożądanych stron dodać atrybut `manifest` i przypisać mu adres URL pliku manifestu (może być względny).
3. Sprawić, by plik manifestu był udostępniany z prawidłowym nagłówkiem typu.

Plik musi mieć nagłówek content-type równy text/cache-manifest. W przeciwnym razie nie będzie działać.

Podejrzanie prosta składnia pliku manifestu

Zawartość pliku manifestu wygląda na mało skomplikowaną. W sekcji `CACHE:` (nie jest to wymagane, ale powinno się jednoznacznie określać tę sekcję) wypisuje się zasoby, które mają być dostępne offline.

Ten wiersz jest wymagany (dokładnie w takiej postaci).

Adresy URL mogą być względne (jak w tym przypadku) lub bezwzględne (np. http://...).

1

```
CACHE MANIFEST
# tak się umieszcza komentarze

CACHE:

index.html
foo/bar.html
baz.html
css/styles.css
icons/plus.png
```

Plik nie musi się nazywać „manifest", ale powinien mieć rozszerzenie „.appcache".

manifest.appcache

Następnie trzeba zaktualizować stronę (lub strony), dodając atrybut `manifest` odwołujący się do pliku manifestu. Trzeba też zadbać o to, by plik był udostępniany z prawidłowym nagłówkiem `content-type` (lub inaczej `MIME-type`). W przypadku serwera Apache wystarczy dodać odpowiedni wpis do pliku konfiguracyjnego lub, co jest lepszym rozwiązaniem, do pliku `.htaccess` umieszczonego w głównym katalogu witryny.

2

```
<!DOCTYPE html>
<html manifest="manifest.appcache">
<head>
```

3

```
AddType text/cache-manifest .appcache
```

.htaccess

Ten wpis informuje serwer WWW, by udostępniał pliki o rozszerzeniu .appcache z nagłówkiem content-type równym text/cache-manifest.

Plik musi być tego typu, bo w przeciwnym przypadku przeglądarka go nie rozpozna.

Jak zwykle diabeł tkwi w szczegółach

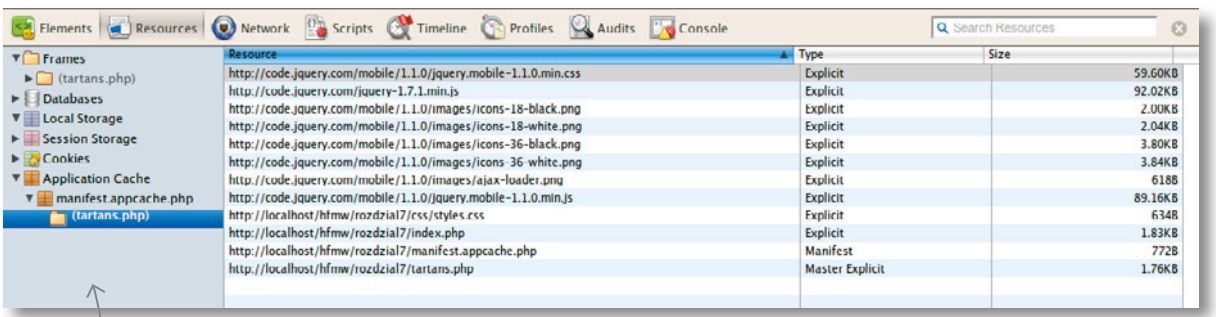
W teorii zarządzanie zasobami w trybie offline za pomocą pliku manifestu jest proste. Wystarczy wypisać pliki, które mają być dostępne offline, dodać atrybut `manifest` do znacznika `<html>` na stronach i już!

Sprawy nie wyglądają jednak tak różowo. Zmuszenie całego mechanizmu, by działał tak, jak chcemy, może być trudne, a czasem wręcz frustrujące. Po drodze czai się wiele niebezpieczeństw, które mogą Cię sprowadzić na manowce. Koniecznie musisz się zaopatrzyć w dobre narzędzie, które pomoże Ci zapanować nad tym, co się dzieje.

Na ratunek przybywają narzędzia

Silnik WebKit dostarcza narzędzie Web Inspector, które jest dostępne zarówno w przeglądarce Chrome, jak i Safari (która też jest oparta na tym silniku). Szczegóły dotyczące mechanizmu Application Cache znajdziesz w zakładce *Resources*.

Szczegółowe informacje o podręcznej pamięci aplikacji znajdują się na zakładce Resources w przeglądarce Chrome.



W przeglądarce Safari wygląda to niemal identycznie.

Obejrzyj to!

Podczas pracy z plikami manifestu łatwo jest się zaciąć.

Zachowanie mechanizmu podręcznej pamięci aplikacji może być momentami niezrozumiałe i trudne w debugowaniu. Niepoprawne pliki manifestu mogą sporo namieszać, a znalezienie przyczyny nieprawidłowego zachowania bez odpowiednich narzędzi może być trudne. Proponujemy, byś podczas tworzenia i testowania plików manifestu dla aplikacji (również dla Tartanatora) korzystał z przeglądarek Chrome lub Safari. Musisz też wiedzieć, że mechanizm podręcznej pamięci aplikacji nie jest obsługiwany w Internet Explorerze do wersji 10. (która w chwili pisania książki jest nadal w fazie testów).

Dobrym pomysłem jest też tworzenie i testowanie aplikacji w przeglądarce desktopowej, a nie mobilnej.

Usuwanie źle zdefiniowanych ustawień z plików manifestu w przeglądarce mobilnej może być bardzo przykrym doświadczeniem.

Udostępniaj pliki manifestu z prawidłowym nagłówkiem content-type

Jest bardzo prawdopodobne, że możesz używać plików `.htaccess` na swoim serwerze — zarówno hostowanym, jak i lokalnym. My jednak obejdziemy się bez nich.

Wiemy, że możesz używać PHP. Skoro tak, wygenerujemy plik manifestu za pomocą PHP i przy okazji ustawimy nagłówek `content-type` na `text/cache-manifest`.

Ćwiczenie

Zaimplementuj podstawowy plik manifestu dla Tartanatora.

1. Utwórz nowy plik i zapisz go w katalogu *rozdzial7* pod nazwą *manifest.appcache.php*.

   Rozszerzenie .php jest potrzebne, by mógł być wykonany kod PHP umieszczony w tym pliku.

2. Na samym początku pliku umieść poniższy wiersz kodu:

```
<?php header('Content-type: text/cache-manifest'); ?>
```

   W ten sposób ustawiamy nagłówek content-type na text/cache-manifest.

3. Dodaj sekcję `CACHE:` i wypisz główne strony witryny, a także pliki JavaScript i CSS.

   Zastosuj składnię przedstawioną na stronie 285.

4. Do znacznika `<html>` w plikach *index.php*, *build.php*, *tartans.php* i *findevent.php* dodaj atrybut `manifest`.

Rozwiązanie ćwiczenia

Gotowy plik *manifest.appcache.php* powinien wyglądać tak jak poniżej. Zwróć uwagę, że dołączyliśmy też pliki CSS i JavaScript z serwisu CDN (ang. *Content Delivery Network*) hostowanego przez jQuery.

```
<?php header('Content-type: text/cache-manifest'); ?>
CACHE MANIFEST
CACHE:
index.php
build.php
tartans.php
findevent.php
css/styles.css
js/tartanator.js
http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.css
http://code.jquery.com/jquery-1.7.1.min.js
http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.js
```

Poza względnymi adresami URL zasobów witryny...

...możesz też używać bezwzględnych adresów zasobów znajdujących się w innych domenach.

manifest.appcache.php

Znaczniki <html> na stronach składających się na aplikację Tartanator powinny wyglądać w ten sposób.

```
<html manifest="manifest.appcache.php">
```

W związku z tym, że do generowania pliku manifestu użyliśmy PHP, nie możemy zastosować rozszerzenia .appcache.

Wszystko o pliku manifestu

Wywiad tygodnia:
Jak zmusić appCache, by zrobił, co mu każemy

Sfrustrowany programista: Wrr... Przejrzałem poszczególne strony Tartanatora i widzę, że brakuje mnóstwa obrazów i ikon, nawet kiedy pracuję online.

appCache: No cóż, to dlatego, że do listy w sekcji `CACHE` nie dołączyłeś tych zasobów. Powinieneś je koniecznie dodać do pliku manifestu.

SP: Czyli muszę dołączyć wszystkie zasoby witryny, bo w przeciwnym razie się nie pojawią?

appCache: Nie, musisz dodać te zasoby, które są potrzebne plikom HTML zapisanym w pamięci podręcznej.

SP: Nic z tego nie rozumiem. Co masz na myśli, mówiąc o plikach HTML zapisanych w pamięci podręcznej?

appCache: Znasz już jeden sposób na uwzględnianie plików HTML w pamięci podręcznej aplikacji — wypisanie ich w sekcji `CACHE` pliku manifestu. Jednak wszystkie pliki HTML, które w znaczniku `<html>` mają atrybut `manifest`, są uwzględniane przez mechanizm pamięci podręcznej, nawet jeśli nie są wypisane w pliku manifestu. Tylko dla tych stron, które są wypisane w pliku manifestu lub mają atrybut `manifest`, trzeba dołączyć wszystkie zasoby.

SP: Zatem jeśli zapomnę dodać wszystkie zasoby w sekcji `CACHE` dla którejkolwiek strony uwzględnionej w pamięci podręcznej, nie będą one wyświetlane nawet w trybie online?

appCache: No cóż, nie do końca tak jest. Wiesz już co nieco o sekcji `CACHE`, ale nie słyszałeś jeszcze o sekcji `NETWORK`.

SP: A do czego służy?

appCache: Sekcja `NETWORK` pozwala zdefiniować zasoby, które nie mają być przechowywane w podręcznej pamięci aplikacji. Przeglądarka powinna pobierać zawsze ich świeżą kopię (oczywiście przy dostępnym połączeniu). W tej sekcji można użyć poręcznego symbolu wieloznacznego *, który reprezentuje wszystko z wyjątkiem zasobów wypisanych w sekcji `CACHE`.

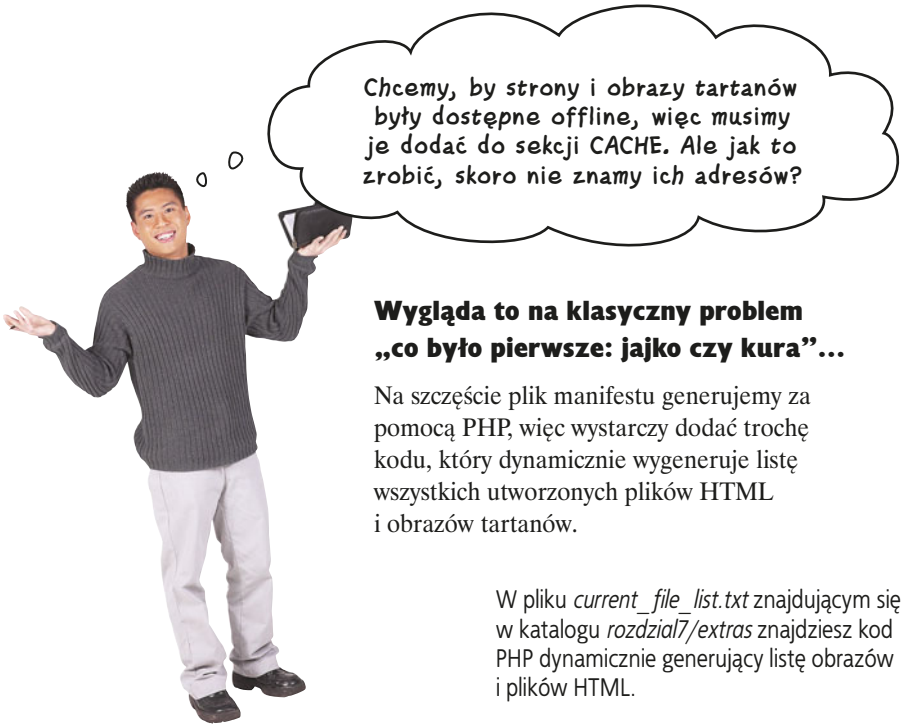
SP: Eureka! Tego właśnie potrzebowałem.

appCache: Musisz jednak uważać. To, co umieścisz w sekcji `NETWORK` (bezpośrednio lub za pomocą symbolu *), zawsze będzie pobierane z serwera. Zachowaj ostrożność podczas definiowania ścieżek.

Musisz zdecydować, które zasoby koniecznie powinny być dostępne offline, i umieścić je w sekcji `CACHE`. Powinny się tam znaleźć na przykład pliki ikon i obrazów. Jednak w przypadku dynamicznych zasobów, takich jak ekrany logowania czy wywołania API, powinieneś skorzystać z sekcji `NETWORK` i symbolu *.

WYTĘŻ UMYSŁ

Czy masz pomysł, które elementy aplikacji Tartanator nie powinny być dostępne offline? Dlaczego właśnie one?



Wygląda to na klasyczny problem „co było pierwsze: jajko czy kura”...

Na szczęście plik manifestu generujemy za pomocą PHP, więc wystarczy dodać trochę kodu, który dynamicznie wygeneruje listę wszystkich utworzonych plików HTML i obrazów tartanów.

W pliku *current_file_list.txt* znajdującym się w katalogu *rozdział7/extras* znajdziesz kod PHP dynamicznie generujący listę obrazów i plików HTML.

Zrób to sam!

Obejrzyj to!

Zmiany w plikach zasobów wymienionych w sekcji `CACHE` pliku manifestu nie zostaną pobrane przez przeglądarkę do czasu zmiany samego pliku manifestu.

W sekcji `CACHE` *pliku manifestu uwzględniamy plik css/styles.css. Możemy wprowadzać w nim zmiany, ale przeglądarka ich nie zobaczy. Po zapisaniu jakiegoś zasobu przez mechanizm podręcznej pamięci aplikacji jedyną możliwością jego odświeżenia jest zmiana samego* **pliku manifestu**.

Popularnym sposobem rozwiązania tego problemu jest umieszczenie w komentarzu numeru wersji. Wystarczy zmienić tę wartość, by przeglądarka uznała, że plik manifestu został zmodyfikowany, co powoduje jego pobranie oraz sprawdzenie, czy pliki zasobów uległy zmianie.

Pamiętaj, że komentarze oznacza się symbolem #.

Wspomniany kod PHP automatycznie generuje listę plików HTML i obrazów utworzonych tartanów, czego efektem ubocznym jest ładowanie przez przeglądarkę pliku manifestu za każdym razem, gdy zostanie dodany nowy tartan. Jednak jeśli wprowadziliśmy zmiany w innych plikach, musimy samodzielnie powiększyć numer wersji zapisany w pliku manifestu.

Bardzo łatwo jest utworzyć plik manifestu, który — jakby to delikatnie ująć — nie do końca odpowiada naszym zamierzeniom.

Okiełznanie krnąbrnych plików manifestu jest najprostsze w przeglądarce Chrome. Wystarczy w pasku adresu wpisać URL `chrome://appcache-internals`, a zostanie wyświetlona strona z informacjami o bieżących danych mechanizmu appCache. Przy zestawieniu dla każdej witryny znajduje się odsyłacz *Remove* służący do usuwania.

W przeglądarce Safari z menu ustawień trzeba wybrać pozycję *Preferencje* i przejść na zakładkę *Prywatność*. Można tu albo usunąć wszystko za pomocą przycisku *Usuń wszystkie dane witryn*, albo wcisnąć przycisk *Szczegóły* i usunąć dane wybranej witryny.

W przeglądarce Firefox z menu *Edycja* trzeba wybrać pozycję *Preferencje* i w oknie przejść na zakładkę *Zaawansowane*/*Sieć*. W sekcji *Pamięć trybu offline* znajduje się lista witryn, dla których są przechowywane dane. Z tego poziomu można wybrać odpowiednią witrynę i usunąć powiązane z nią dane.

Ćwiczenie

Najwyższy czas, by nasz plik manifestu zachowywał się trochę lepiej. Musimy uwzględnić wszystko to, o czym do tej pory mówiliśmy, i zastosować w pliku *manifest.appcache.php*.

1. **Wyśledź brakujące ikony i obrazy.**
   Jest kilka ikon i obrazów używanych przez jQuery Mobile, które musimy dodać do sekcji `CACHE`.

   Czapki z głów, jeśli udało Ci się już to zrobić wcześniej.

2. **Dodaj sekcję NETWORK z symbolem wieloznacznym *.**
   Umieść to przed sekcją `CACHE`:

```
NETWORK:
*
```

3. **Dołącz fragment kodu PHP.**
   Na końcu pliku manifestu umieść kod z pliku *extras/current_file_list.txt*.

4. **Z pliku manifestu usuń pliki build.php oraz findevent.php.**
   Po przemyśleniu sprawy uznaliśmy, że te dwie strony powinny być dostępne tylko w trybie online. Co prawda na stronie Wydarzenia jeszcze nic ciekawego nie ma, ale już niedługo.

   Nie zapomnij usunąć atrybutu manifest ze znacznika <html> w tych dwóch plikach.

Rozwiązanie ćwiczenia

Plik *manifest.appcache.php* powinien wyglądać jak poniżej.

```
<?php header('Content-type: text/cache-manifest'); ?>
CACHE MANIFEST

NETWORK:
*

CACHE:
http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.css
http://code.jquery.com/jquery-1.7.1.min.js
http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.js
http://code.jquery.com/mobile/1.1.0/images/ajax-loader.png
http://code.jquery.com/mobile/1.1.0/images/icons-18-white.png
http://code.jquery.com/mobile/1.1.0/images/icons-18-black.png
http://code.jquery.com/mobile/1.1.0/images/icons-36-white.png
http://code.jquery.com/mobile/1.1.0/images/icons-36-black.png
index.php
tartans.php
css/styles.css

<?php // W tym miejscu ma się znaleźć kod PHP z pliku current_file_list.txt.
// Nie zamieszczamy go tu, by zaoszczędzić trochę miejsca i ocalić kilka drzew.?>
```

Uważaj, żeby między header() i CACHE MANIFEST nie było żadnego pustego wiersza.

Teraz zapisz plik i sprawdź, jak działa aplikacja!

Obejrzyj to!

Jeden błąd w pliku manifestu może spowodować problemy z całą aplikacją.

Pewnie się tego spodziewałeś — jest jeszcze więcej pułapek. Wystarczy, że żądanie jednego zasobu z pliku manifestu zakończy się kodem 404 (brak pliku), a cały plik manifestu jest ignorowany. Taki sam efekt może spowodować zwykła niewinna literówka.

Powinieneś się odpowiednio zabezpieczyć. Możesz skorzystać z walidatora http://manifest-validator.com/ oraz na bieżąco śledzić informacje w przeglądarkowych narzędziach dla programistów, by kontrolować, czy żaden zasób nie został pominięty.

Mimo że plik manifestu został zaktualizowany, aby zobaczyć nowo dodany wzór, musisz przeładować stronę z listą tartanów (*tartans.php*). Dlaczego?

Łukasz: Poczytałem o tym trochę. Przyznaję, bardzo mnie wciągnęło. Chcesz się dowiedzieć, jak to działa? Powiedzmy, że przeglądarka ma już plik manifestu dla twojej witryny. Wchodzisz na stronę z listą tartanów po utworzeniu kilku wzorów. Przeglądarka zauważa, że plik manifestu się zmienił — różni się w porównaniu z tym, który ma listę plików powiązanych z tartanami. W związku z tym natychmiast pobiera nowy plik manifestu i sprawdza, co się zmieniło.

Kuba: To czemu, do jasnej ciasnej, nie pojawiają się zaktualizowane lub nowe elementy?

Łukasz: Przeglądarka nie czeka z renderowaniem strony na załadowanie wszystkich zaktualizowanych i nowych elementów. Po pobraniu zasoby siedzą zwarte i gotowe w pamięci podręcznej, ale nie zostaną wyświetlone aż do ponownego przeładowania strony.

Kuba: W takim razie jedyną możliwością zaktualizowania strony jest jej przeładowanie?

Łukasz: No cóż, mam chyba pomysł na obejście tego problemu w tej sytuacji. Napisałem taki mały skrypcik JavaScript...

Kuba: No jasne, jak zwykle JavaScript!

Łukasz: Dynamiczna część strony jest generowana przez skrypt PHP zawarty w pliku *list.php*. Dla każdego tartanu jest tworzony jeden element `<li>`. Pozostała część strony jest statyczna. Tak naprawdę powinniśmy się zatroszczyć tylko o zawartość listy `<ul>` z tartanami.

Kuba: Prawda.

Łukasz: W moim kodzie korzystam z obiektu `window.applicationCache` i jego metod, by sprawdzić, czy coś się zmieniło w pamięci podręcznej, a tak jest w przypadku modyfikacji pliku manifestu. Zgadza się? Jeśli doszło do zmiany, trzeba pobrać zaktualizowaną zawartość ze skryptu *list.php*.

Kuba: Wszystko jasne. Jeśli plik manifestu został zmieniony, powinniśmy też zaktualizować listę tartanów. A najlepiej, żeby nie wymagało to przeładowania całej strony.

Łukasz: Zgadza się. Zatem jeśli doszło do zmiany pliku manifestu, mój kod wysyła żądanie AJAX do skryptu *list.php*. Ten skrypt nie znajduje się na liście manifestu i w przypadku bezpośredniego wywołania zwraca kod HTML zawierający listę aktualnych wzorów tartanów. Otrzymany kod wstawiam w miejsce dotychczasowego, czyli w miejsce listy tartanów. I to wszystko!

Jazda próbna

Skorzystaj z kodu JavaScript przygotowanego przez Łukasza i sprawdź, czy pomoże w poradzeniu sobie z problemami mechanizmu podręcznej pamięci aplikacji.

1. **Skopiuj plik z kodem.**
   Plik *cache-manager.js* z katalogu *extras/js* skopiuj do katalogu *rozdzial7/js*.

2. **Dołącz skrypt do stron Tartanatora.**
   W plikach *index.php*, *build.php*, *findevent.php* i *tartans.php* dołącz skrypt *cache-manager.js* (bezpośrednio pod znacznikiem dodającym framework jQuery Mobile).

```
<script src="js/cache-manager.js" type="text/javascript"></script>
```

3. **W pliku tartans.php umieść dodatkowy kod JavaScript.**
   Ten kod zapewnia aktualną zawartość elementu `#tartans-list` znajdującego się w bloku `<div>` o identyfikatorze `#tartans_page` na podstawie danych zwracanych przez skrypt *inc/list.php*.

```
<div data-role="page" id="tartans_page">
    <script type="text/javascript" charset="utf-8">
        var tartanPage = $('#tartans_page');
        tartanPage.live('pageinit', function () {
            if (!tartanPage.data.cacheManager) {
                tartanPage.data.cacheManager = new CacheManager('#tartans_page');
                tartanPage.data.cacheManager.ensureFreshContent('#tartans-list', 'inc/list.php');
            }
        });
    </script>
<div data-role="header" data-position="fixed">
```

tartans.php

Nie istnieją głupie pytania

P: Skoro mogę użyć symbolu wieloznacznego w sekcji NETWORK pliku manifestu, to czy nie mógłbym skorzystać z niego w sekcji CACHE, by do trybu offline dołączyć wszystkie zasoby witryny?

O: Na szczęście (albo na nieszczęście) nie możesz użyć symboli wieloznacznych w sekcji CACHE. Symbol * możesz zastosować tylko w sekcji NETWORK i tak naprawdę to nie jest symbol wieloznaczny. Nie możesz go umieścić w dowolnej ścieżce — występuje zawsze samotnie i ma szczególne znaczenie (które można wyjaśnić tak: jeśli dany zasób nie jest uwzględniony w innej sekcji, pobierz go bezpośrednio z serwera).

P: Dlaczego strony z atrybutem `manifest` w znaczniku `<html>` są przechowywane w pamięci podręcznej nawet wtedy, gdy nie są uwzględnione w pliku manifestu?

O: Faktycznie może to być nie do końca zrozumiałe i powodować problemy. Chodzi tu o to, by zamiast pobierania i zapisywania w podręcznej pamięci całego mnóstwa stron i zasobów za jednym razem umożliwić ich „leniwe" dodawanie, czyli takie, w którym dana strona jest umieszczana w pamięci offline wtedy, gdy użytkownik odwiedza ją po raz pierwszy.

P: Co macie na myśli, mówiąc „za jednym razem"?

O: Kiedy przeglądarka odczyta nowy lub zaktualizowany plik manifestu, natychmiast zabiera się za pobieranie wszystkich nowych zasobów i sprawdzanie zmian w tych już istniejących. Koniecznie musisz to uwzględnić podczas definiowania plików manifestu.

P: W naszym pliku manifestu są uwzględnione pliki HTML i obrazy wszystkich tartanów. Czy to nie jest mnóstwo danych do pobrania „za jednym razem"?

O: Trochę się tym martwiliśmy. Wygeneruje to całkiem sporo żądań HTTP. Jednak pliki są stosunkowo małe: większość obrazów ma rozmiar poniżej 1 kB, a pliki HTML jeszcze mniej. Obciążenie łącza nie będzie więc zbyt duże, zwłaszcza jeśli się weźmie pod uwagę to, że pliki są pobierane asynchronicznie. Użytkownik nie powinien tego raczej odczuć.

P: Czy przeglądarka ponownie pobiera wszystkie zasoby z pliku manifestu za każdym razem, gdy jest on aktualizowany?

O: Na szczęście nie. Przeglądarka sprawdza, czy zasoby, które znajdują się już w pamięci trybu offline, uległy zmianie. Jeśli nie, nie są ponownie pobierane.

P: Wszystko pięknie i ładnie, ale co się stanie, gdy przeglądarka użytkownika nie obsługuje mechanizmu podręcznej pamięci aplikacji?

O: Witryna będzie się zachowywała dokładnie tak samo jak wcześniej, zanim dodaliśmy plik manifestu. Nie dzieje się nic złego, ale brakuje oczywiście funkcjonalności trybu offline.

P: Nic nie wspomnieliście o sekcji FALLBACK.

O: Poza sekcjami CACHE i NETWORK jest jeszcze opcjonalna sekcja FALLBACK, która daje możliwość zdefiniowania wersji offline plików, kiedy są niedostępne ich wersje online (na przykład plik *login.html* może zostać podmieniony na *offline.html*).

P: Mechanizm appCache jest fajny, ale co z localStorage? W tej chwili to wygląda tylko na połowę opowieści.

O: Ej, mądralo, wyprzedziłeś materiał! Do tematu lokalnego składowania danych, czyli localStorage, wrócimy w rozdziale 8. (nie przejmuj się, jeżeli jeszcze o tym nie słyszałeś).

Obejrzyj to!

Uważaj, by nie nadużywać atrybutu `manifest` na stronach.

Już o tym wspomnieliśmy, ale musimy to powtórzyć: każda strona, która w znaczniku `<html>` *ma atrybut* `manifest`*, trafia do pamięci trybu offline bez względu na to, czy znajduje się w pliku manifestu. Nie ma na to wpływu również sekcja* NETWORK *ani inne sztuczki.*

Zwyciężyliśmy (w końcu)

Uff. Było z tym trochę problemów, ale w końcu Tartanator działa w trybie offline. To znaczy, że zakończyliśmy prace nad edytorem tartanów z drugiej fazy projektu.

- [x] Ulepszyć istniejący formularz, by mógł wykorzystać możliwości oferowane przez nowsze przeglądarki mobilne.
- [x] Napisać kod strony serwerowej, który jest odpowiedzialny za przetwarzanie danych z formularza oraz generowanie zasobów (obrazów, kodu HTML itp.) na potrzeby tartanów utworzonych przez użytkownika.
- [x] Zapewnić możliwość pracy w trybie offline w tej części aplikacji.

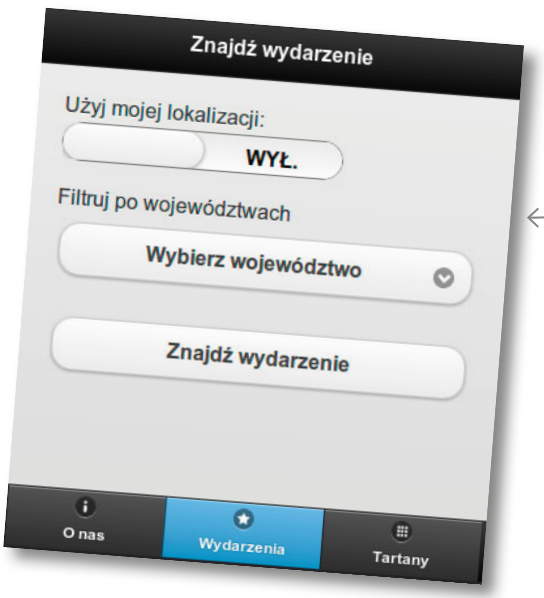
Gratulacje! Udało Ci się zmusić tryb offline do prawidłowego działania.

Zdefiniowanie pliku manifestu w taki sposób, by tryb offline działał tak, jak tego chcesz, jest trudne. Tobie się udało!

w Twojej okolicy

Teraz kolej na szukanie wydarzeń

Kolejnym elementem drugiej fazy projektu jest stworzenie strony, która pozwala wyszukiwać wydarzenia związane z tartanami w najbliższej okolicy użytkowników.

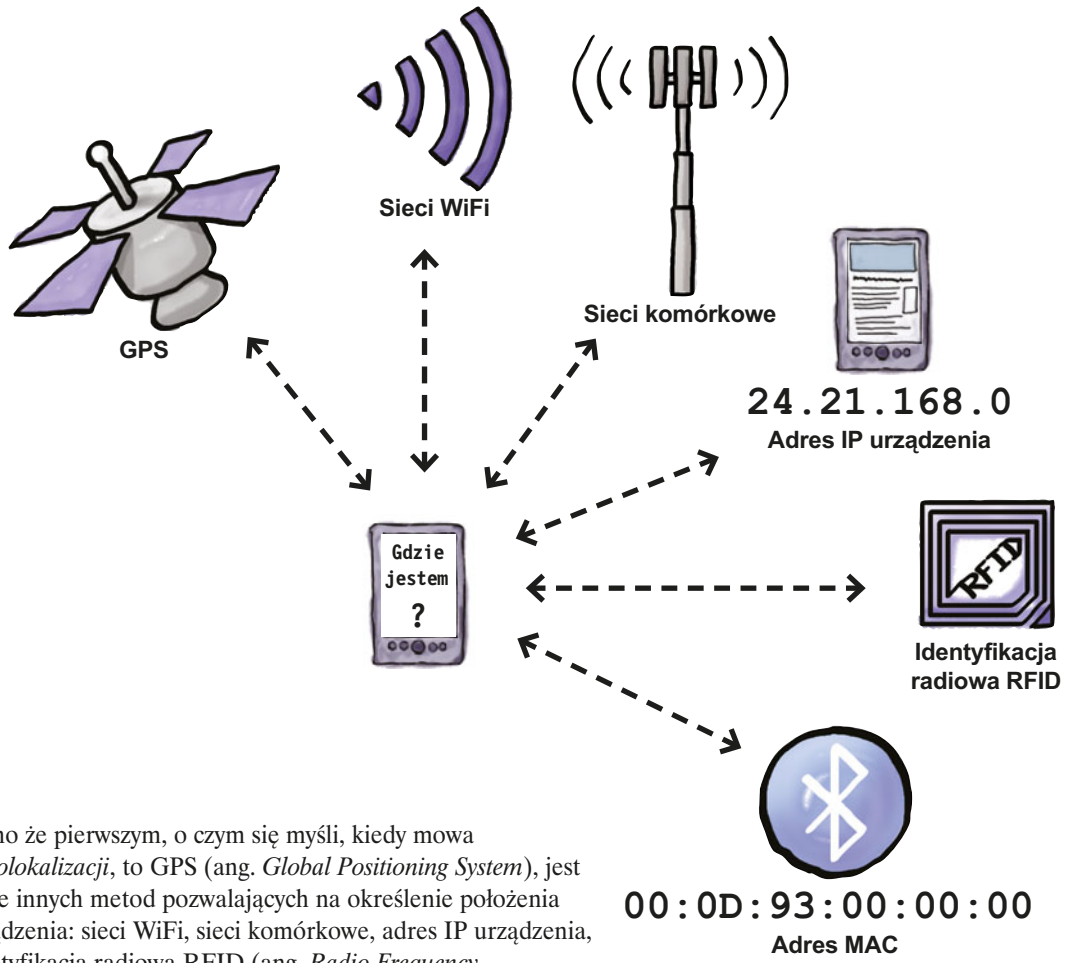


Formularz „Znajdź wydarzenie", który zamierzamy stworzyć.

Chcemy wyszukiwać wydarzenia, które odbywają się blisko bieżącej lokalizacji użytkownika. To oznacza, że musimy porozmawiać o geolokalizacji.

Jak działa geolokalizacja?

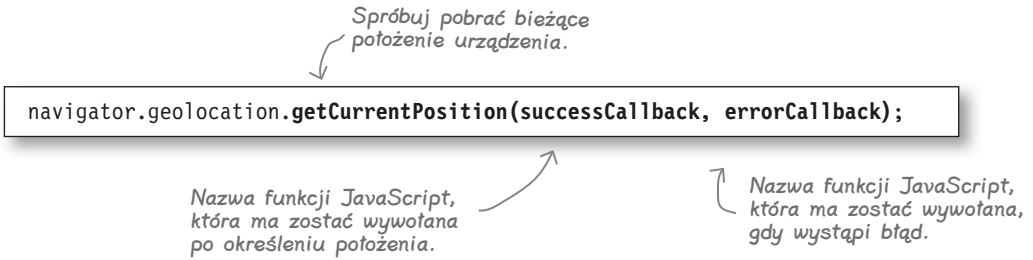
Skorzystanie z ***geolokalizacji*** w przeglądarce internetowej wymaga użycia JavaScriptu do pobrania bieżącej lokalizacji urządzenia. Wiele nowoczesnych przeglądarek mobilnych ma zaimplementowane wsparcie dla interfejsu API geolokalizacji zgodnego ze specyfikacją W3C (ang. *World Wide Web Consortium*), umożliwiającego stosunkowo proste odczytywanie danych geolokalizacyjnych. Jest też kilka innych rozwiązań pozwalających na geolokalizację, jak choćby Google Gears API czy interfejsy API konkretnych przeglądarek. Przeglądarki w starszych telefonach, a nawet w niektórych nowszych smartfonach w ogóle nie obsługują geolokalizacji.



Mimo że pierwszym, o czym się myśli, kiedy mowa o *geolokalizacji*, to GPS (ang. *Global Positioning System*), jest wiele innych metod pozwalających na określenie położenia urządzenia: sieci WiFi, sieci komórkowe, adres IP urządzenia, identyfikacja radiowa RFID (ang. *Radio Frequency IDentification*), a także adresy MAC (ang. *Media Access Control*) urządzeń WiFi i Bluetooth.

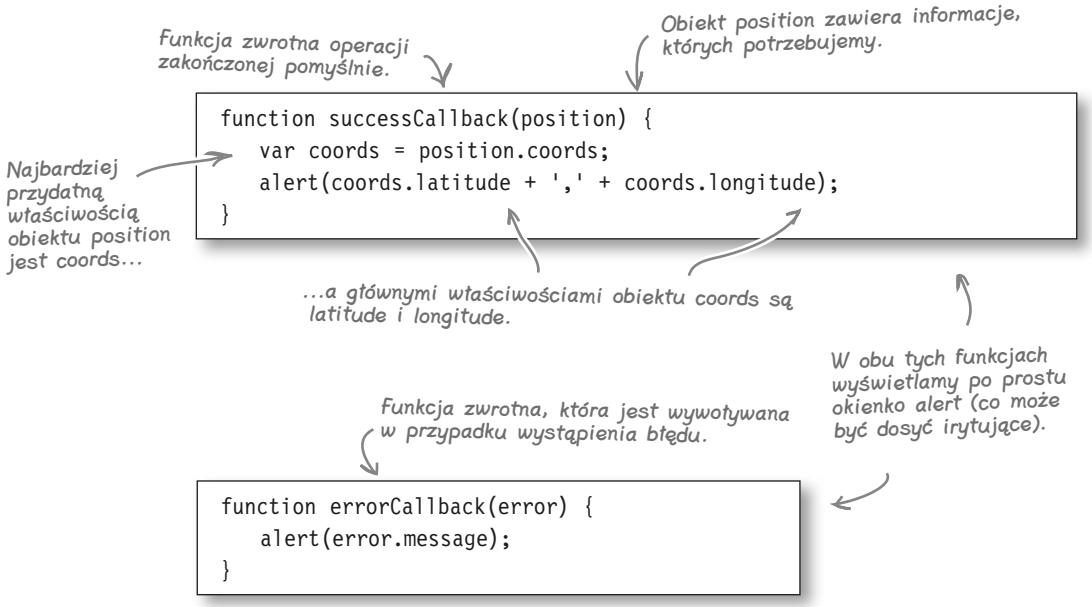
Jak poprosić przeglądarkę o dane geolokalizacyjne?

Przeglądarki, które mają zaimplementowany interfejs API geolokalizacji zgodny z W3C, udostępniają obiekt `navigator.geolocation`. Najważniejszą metodą tego obiektu jest `getCurrentPosition`, która powoduje odczytanie bieżącej lokalizacji.



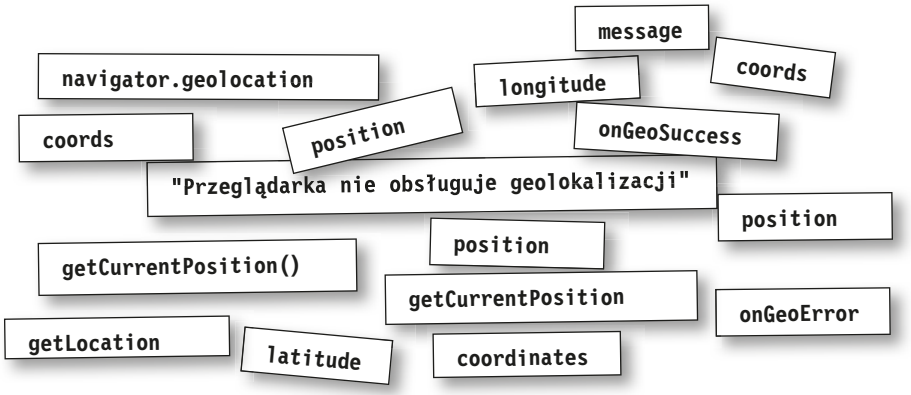
Korzystanie z danych dostarczanych przez getCurrentPosition

Po pomyślnym określeniu położenia jest wywoływana funkcja zwrotna, której nazwę przekazaliśmy metodzie `getCurrentPosition`. Funkcja ta otrzymuje jako parametr obiekt `position`. Najbardziej przydatnymi danymi z tego obiektu są szerokość i długość geograficzna zapisane we właściwościach `latitude` i `longitude`, które znajdują się z kolei we właściwości `coords` obiektu `position`.



Magnesiki z kodem geolokalizacji w JavaScripcie

Magnesiki z kodem umieść we właściwych miejscach na listingu przedstawionym poniżej. Każdego magnesu możesz użyć tylko raz, ale uważaj, bo niektóre z nich są zbędne.



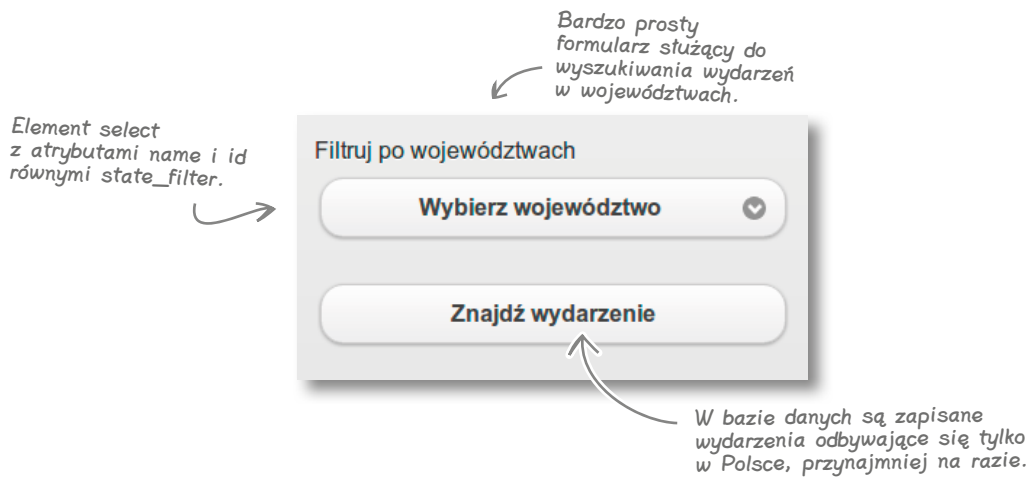

```
function getLocation () {
    // Sprawdzamy, czy przeglądarka obsługuje geolokalizację
    if (.....) {
        navigator.geolocation.....(....., onGeoError);
    } else { // Nie obsługuje
        onGeoError(new Error(.....));
    }
}

function onGeoSuccess (.....) {
    var ..... = ..... coords;
    alert(coordinates..... + ',' + coordinates.....);
}

function ..... (error) {
    alert(error.....);
}
.....();
```

Odpowiedzi na stronie 302

Początek pracy nad stroną Znajdź wydarzenie — podstawy

Głównym zadaniem strony Znajdź wydarzenie jest... znajdowanie wydarzeń. Mamy już ogólny obraz tego, jak za pomocą JavaScriptu można określić aktualną lokalizację urządzenia. Musimy jednak zacząć od stworzenia podstawowej wersji strony. Co będzie, jeśli przeglądarka użytkownika nie obsługuje JavaScriptu lub geolokalizacji?

Zaczniemy od utworzenia formularza służącego do wyszukiwania wydarzeń, który nie wymaga ani jednego, ani drugiego. Punktem wyjścia będzie pole wyboru z listą województw.



Ćwiczenie

Zbuduj prosty formularz do wyszukiwania wydarzeń w dokumencie *findevent.php*, który posłuży jako baza. Kod formularza umieść w bloku `<div>` z zawartością strony.

1. W znaczniku `<form>` ustaw atrybuty `method` na `GET`, `id` na `search_form`, a `action` na `events.php`.

   Zaraz Ci powiemy, gdzie możesz znaleźć plik events.php.

2. Do formularza dodaj element `select`. Opcjami tego elementu mają być nazwy wszystkich 16 województw, a wartościami poszczególnych opcji powinny być trzyliterowe skróty nazw.

   Gotową listę znajdziesz w pliku wojewodztwa.txt umieszczonym w katalogu extras/events.

3. Przycisk zatwierdzający formularz powinien mieć identyfikator `search_submit`.

Nie zapomnij umieścić pól w bloku z atrybutem data-role ustawionym na fieldcontain (ze względu na jQM).

Magnesiki z kodem geolokalizacji w JavaScripcie. Rozwiązanie

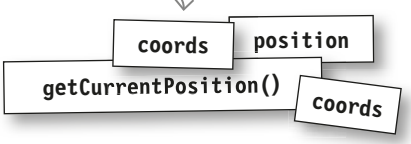
Udało Ci się umieścić magnesiki we właściwych miejscach?

```
function getLocation () {
    // Sprawdzamy, czy przeglądarka obsługuje geolokalizację
    if ( navigator.geolocation ) {
        navigator.geolocation. getCurrentPosition ( onGeoSuccess , onGeoError);
    } else { // Nie obsługuje
        onGeoError(new Error( "Przeglądarka nie obsługuje geolokalizacji" ));
    }
}

function onGeoSuccess ( position ) {
    var coordinates = position .coords;
    alert(coordinates. latitude + ',' + coordinates. longitude );
}

function onGeoError (error) {
    alert(error. message );
}
getLocation ();
```

Kiedy już skompletujesz ten kod, **umieść go w pliku o nazwie** *geolocation.js* i zapisz w katalogu *rozdzial7/js*. Następnie dołącz go do pliku *findevent.php* za pomocą znacznika `<script>` umieszczonego w bloku `<div>` z identyfikatorem `data-role="page"`.

Powinien się znaleźć tuż nad formularzem.

Te magnesiki nie były potrzebne.

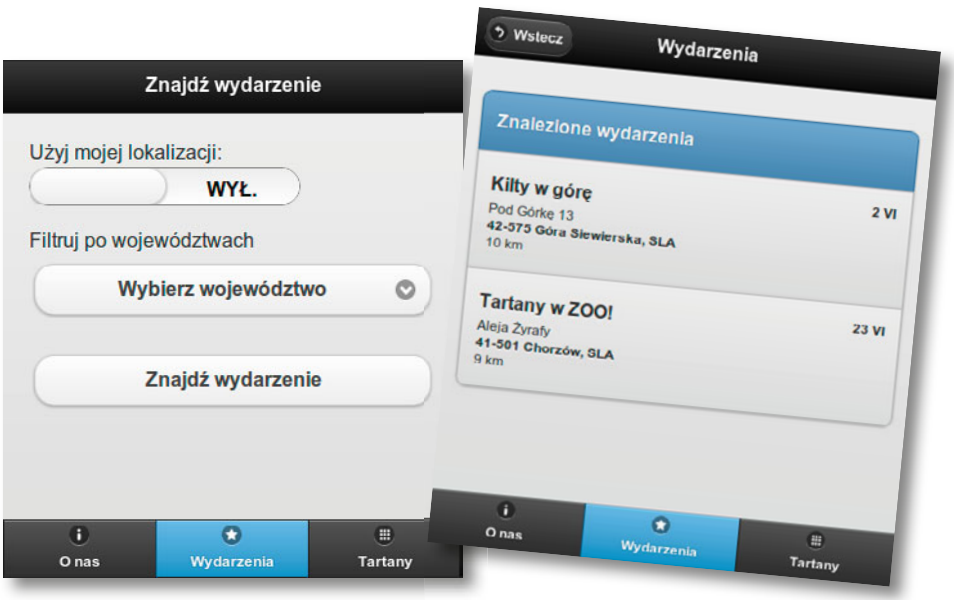


Dołączamy geolokalizację

Mamy już podstawową wersję formularza. A, i jeszcze jedno — bardzo denerwujące okienka wyświetlane przez JavaScript.

Teraz zajmiemy się ciekawszą częścią — zastosujemy geolokalizację w przeglądarkach, które ją obsługują. Po wprowadzeniu kilku zmian formularz będzie w nich wyglądać tak:

Kiedy już dopieścimy formularz, użytkownicy przeglądarek obsługujących geolokalizację będą mogli użyć swojego aktualnego położenia podczas wyszukiwania wydarzeń.



Ćwiczenie. Rozwiązanie

Kod podstawowej wersji formularza wygląda jak poniżej.

```
<div data-role="content">
   <script type="text/javascript" src="js/geolocation.js"></script>
   <form method="get" action="events.php" id="search_form">
      <div data-role="fieldcontain">
         <label for="state_filter">Filtruj po województwach</label>
         <select id="state_filter" name="state_filter">
            <option value="">Wybierz województwo</option>
            <option value="DOL">dolnośląskie</option>
            <!-- itd. -->
         </select>
      </div>
      <div data-role="fieldcontain">
         <input type="submit" value="Znajdź wydarzenie" id="search_submit" />
      </div>
   </form>
</div><!-- /content -->
```

GEOLOKALIZACYJNE PRACE KONSTRUKCYJNE

Dopracujemy teraz kod skryptu *geolocation.js*. Zwróć szczególną uwagę na wyróżnione miejsca.

```
(function () {
   var $page, $searchForm, $submitButton, $stateFilter;

   $page = $('#event_page');
   if (!$page.data.initialized) {
      $page.live('pagecreate', initGeo);
      $page.data.initialized = true;
   }

   function initGeo() {
      $searchForm = $('#search_form');
      $submitButton = $('#search_submit');
      $stateFilter = $('#state_filter');
      if (navigator.geolocation) {
         initGeoOptions();
      }
   }
   function initGeoOptions() {
      var $latField, $longField, $flipSwitch;
      $flipSwitch = $('<select name="usegeo" id="usegeo"
data-role="slider"><option value="off">WYŁ.</option><option
value="on">WŁ.</option></select>').change(toggleLocation);
      $flipSwitch.prependTo($searchForm).wrap('<div
datarole="fieldcontain"></div>');
      $flipSwitch.before('<label for="usegeo">Użyj mojej
lokalizacji</label>');
      $latField = $('<input type="hidden" />').attr({ name :
'latitude', id : 'latitude'});
      $longField = $('<input type="hidden" />').attr({name:
'longitude', id : 'longitude' });
      $latField.appendTo($searchForm);
      $longField.appendTo($searchForm);
}
```

Po zainicjalizowaniu strony wywołujemy funkcję initGeo.

Jeśli przeglądarka obsługuje geolokalizację, inicjalizujemy elementy formularza związane z lokalizowaniem.

Dodaliśmy suwak stylizowany na przełącznik, dzięki któremu użytkownicy mogą skorzystać z informacji o aktualnym położeniu.

Dodaliśmy dwa ukryte pola, które przechowują wartości szerokości i długości geograficznej pobrane z API geolokalizacji.

Funkcja toggleLocation jest podpięta do zdarzenia change suwaka.

```
function toggleLocation(event) {
    var geoActivated = ($(event.target).val() == 'on') ? true : false;
    if (geoActivated) {
        $submitButton.button('disable');
        $stateFilter.selectmenu('disable');
        $.mobile.showPageLoadingMsg();
        navigator.geolocation.getCurrentPosition(onGeoSuccess, onGeoError);
    } else {
        $stateFilter.selectmenu('enable');
        $submitButton.button('enable');
    }
}
function onGeoSuccess(position) {
    var coordinates = position.coords;
    $('#latitude').val(coordinates.latitude);
    $('#longitude').val(coordinates.longitude);
    $.mobile.hidePageLoadingMsg();
    $submitButton.button('enable');
}
function onGeoError(error) {
    $('#usegeo').val('off').trigger('change');
    alert(error.message);
}
})();
```

Jeśli użytkownik włączy geolokalizację, dezaktywowane są pole wyboru województwa i przycisk zatwierdzający (przycisk zostanie ponownie aktywowany po określeniu lokalizacji).

W tym miejscu uruchamiamy geolokalizację.

Po wyłączeniu przełącznika ponownie aktywujemy pole wyboru województwa.

To jest funkcja zwrotna wywoływana po pomyślnym określeniu położenia. Zmieniliśmy ją tak, by aktualizowała ukryte pola #latitude i #longitude otrzymanymi wartościami.

W przypadku błędu przełącznik jest wyłączany, a następnie jest wyzwalane zdarzenie change (do którego jest podpięta funkcja toggleLocation).

Sprawdź sam, czy to działa! Pobaw się chwilę formularzem, by poczuć na własnej skórze to, czego dokonaliśmy dzięki JavaScriptowi.

Przedstawiony tu kod możesz znaleźć w pliku *extras/js/enhanced_geo_form.js*.

Zaktualizuj plik *geolocation.js*, zapisz go, a następnie przejdź na zaktualizowaną stronę Znajdź wydarzenie.

Kuba: Przed chwilą sprawdzałem, jak na różnych urządzeniach działa nasz nowy formularz. Wiem, że przeglądarka w BlackBerry OS 5 obsługuje geolokalizację, ale w formularzu nie pojawił się przełącznik *Użyj mojej lokalizacji*.

Łukasz: Tak, przeglądarka w BlackBerry OS 5 obsługuje geolokalizację, ale nie jest ona zgodna ze specyfikacją W3C. Poszperałem trochę na ten temat. Starsze przeglądarki w Androidzie też nie mają obsługi zgodnej z W3C, bo bazują na Google Gears.

Kuba: To wszystko wygląda na strasznie skomplikowane...

Łukasz: Bo takie jest. A nam się zbliża termin. Nie sądzę, żebyśmy mieli dość czasu na przyjrzenie się wszystkim niuansom, przetestowanie na wszystkich możliwych urządzeniach i poprawienie najdrobniejszych błędów. Znalazłem coś, co może nam pomóc. To mała javascriptowa biblioteka typu open source, która pośredniczy w dostępie do geolokalizacji na wielu różnych platformach — zarówno tych zgodnych z W3C, jak i niezgodnych. Nazywa się dosyć prosto: *geo-location-javascript*.

Kuba: Ale czy to przypadkiem nie znaczy, że musimy zrefaktoryzować cały do tej pory napisany kod JavaScript?

Łukasz: Nie. Interfejs API tej biblioteki jest zgodny ze specyfikacją W3C. Nazwy wszystkich ważniejszych metod i właściwości są identyczne. Sądzę, że wystarczy zmienić kilka linijek kodu, by przestać się martwić o międzyplatformową kompatybilność.

Kuba: Skoro już będziemy grzebać przy tym kodzie, może zrobimy coś z tymi okropnymi okienkami z komunikatami o błędach?

Łukasz: Słusznie. Myślę, że powinniśmy użyć okienek dialogowych z frameworku jQuery Mobile, z których korzystałem już w formularzu edytora tartanów.

Kuba: Nic mi o tym nie wspomniałeś! Jakie okienka?

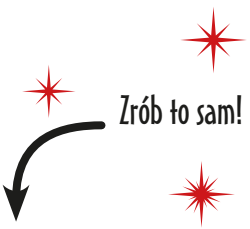
Łukasz: Jeśli użytkownik chce zatwierdzić formularz z tartanem, a nie poda jego nazwy, pojawia się okienko dialogowe informujące o błędzie. Wygląda całkiem ładnie — podobnie jak pozostałe elementy z jQM.

Kuba: A jak ci się udało wyświetlić te okna?

Łukasz: To są oddzielne strony jQM. Zresztą zajrzyj do katalogu *dialogs*, a sam zobaczysz. Wyświetlam je z poziomu JavaScriptu.

Kuba: Świetnie! Koniecznie muszę się temu przyjrzeć.

Łukasz: W porządku. Ja w tym czasie szybko dorzucę tę bibliotekę geolokalizacji.



Plik *extras/hs/geo.js* skopiuj do katalogu *rozdzial7//js*. To jest właśnie wspomniana biblioteka geo-location-javascript. Możesz też odwiedzić stronę projektu, jeśli chcesz się dowiedzieć więcej: *http://bit.ly/sx7JrH*.

Ćwiczenie

Zaktualizuj stronę Znajdź wydarzenie oraz kod JavaScript geolokalizacji, tak by była wykorzystywana międzyplatformowa biblioteka, a błędy pojawiały się w okienkach jQM.

1 Do pliku findevent.php dołącz skrypt JavaScript nowej biblioteki geolokalizacji.

```
<div data-role="content">
<script src="http://code.google.com/apis/gears/gears_init.js" type="text/
javascript" charset="utf-8"></script>
<script src="js/geo.js" type="text/javascript" charset="utf-8"></script>
<script type="text/javascript" src="js/geolocation.js"></script>
```

2 Zmodyfikuj plik geolocation.js, by wykorzystywał nową bibliotekę.

Znajdź poniższy kod:

```
~~if (navigator.geolocation) {~~
~~    initGeoOptions();~~
~~}~~
```

i zmień go na:

```
if (geo_position_js.init()) {
   initGeoOptions();
}
```

Zmień również to:

```
~~navigator.geolocation.getCurrentPosition(onGeoSuccess, onGeoError);~~
```

na to:

```
geo_position_js.getCurrentPosition(onGeoSuccess, onGeoError);
```

3 Utwórz okno dla błędów geolokalizacji i dodaj kod wywołujący.

Posłuż się plikami z katalogu *dialogs* jako wzorem i utwórz ładne okno informujące o błędach geolokalizacji. Zapisz je w pliku *geolocation_error.html*.

Następnie wywołanie `alert` w funkcji `onGeoError` zamień na wyświetlenie utworzonego okna. Przykład odpowiedniego kodu znajdziesz w pliku *js/tartanator.js*. Wystarczy, że zmienisz adres URL.

Rozwiązanie ćwiczenia

Funkcja `onGeoError` po zamianie zwykłego okienka `alert` na okno jQuery Mobile powinna wyglądać jak poniżej. Nie zapomnij utworzyć pliku HTML dla tego okna!

```
function onGeoError(error) {
   $('#usegeo').val('off').trigger('change');
   $.mobile.changePage( "dialogs/geolocation_error.html", {
      transition: "pop",
      reverse: false,
      role: 'dialog'
   });
}
```

Przedarliśmy się biegiem przez gąszcz JavaScriptu.

Przyznajemy, że ostatnie przeżycia były dosyć intensywne. Jest też kod, którego dokładnie nie omówiliśmy. Ale to nie jest przecież książka *Head First JavaScript. Edycja polska*, a my nie chcemy odchodzić zanadto od głównego zadania: wygrania walki z mechanizmem podręcznej pamięci aplikacji. Chcemy też zająć się interfejsem mediaCapture dostępnym w technologii PhoneGap (nawet jeśli jeszcze nie wszystkie przeglądarki go wspierają).

Jazda próbna

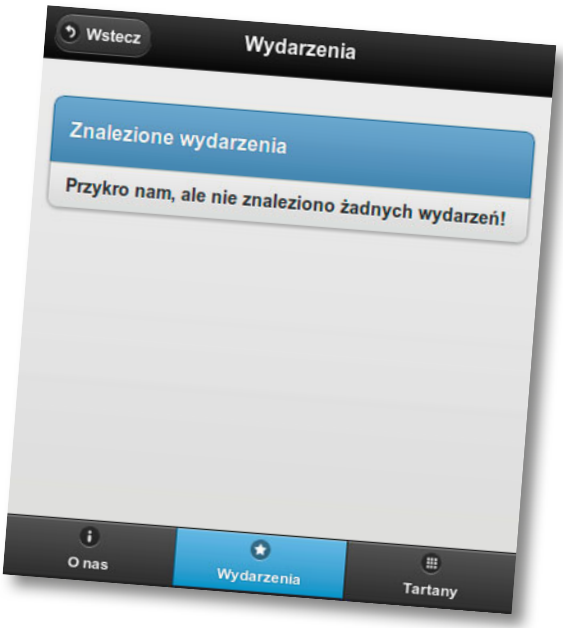
Na razie wystarczy! Zerknijmy, jak to się sprawdza w praktyce. Dołożenie kodu wyszukującego wydarzenia wraz z przykładowymi danymi nie zajmie Ci wiele czasu.

1. **Skopiuj kilka niezbędnych plików z katalogu extras/events.**
   Do katalogu *rozdzial7/inc* skopiuj pliki *event_search.inc* i *event_list.inc*. Plik *events.php* umieść w katalogu *rozdzial7*.

2. **Sprawdź, jak to działa!**
   W przeglądarce mobilnej przejdź na stronę Znajdź wydarzenie i włącz przełącznik *Użyj mojej lokalizacji*.

Nic nie znalazł

Ha, ha! Przepraszamy za to. Najprawdopodobniej otrzymasz taki wynik, chyba że jesteś akurat w okolicy miejsc, które zdefiniowaliśmy w pliku *event_list.inc*.



Najprawdopodobniej otrzymasz właśnie taki wynik.

Jest to spowodowane doborem przykładowych danych oraz ustalonym promieniem poszukiwań (50 km). Co Ty na to, żeby dodać własne definicje wydarzeń, by otrzymać jakieś wyniki?

Ćwiczenie

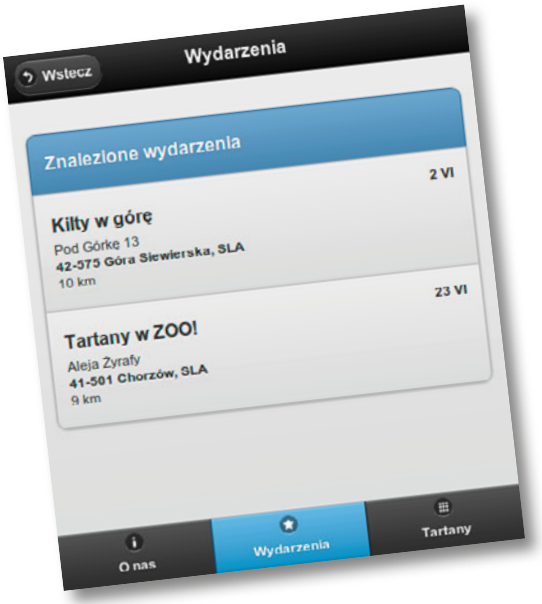
Dodaj kilka przykładowych definicji wydarzeń odbywających się w Twojej okolicy. Otwórz plik *event_list.inc* umieszczony w katalogu *rozdzial7/inc/*. Znajduje się w nim tablica zawierająca definicje wydarzeń. Do tablicy dodaj kilka elementów z adresami i współrzędnymi odpowiadającymi Twojej najbliższej okolicy.

Jeżeli nie najlepiej czujesz się z PHP, możesz jedynie zmodyfikować istniejące definicje. Najważniejsze dane używane do wyszukiwania to szerokość i długość geograficzna (`latitude` i `longitude`) oraz województwo (`state`).

W szybkim określeniu współrzędnych geograficznych może pomóc narzędzie dostępne pod adresem http://itouchmap.com/latlong.html.

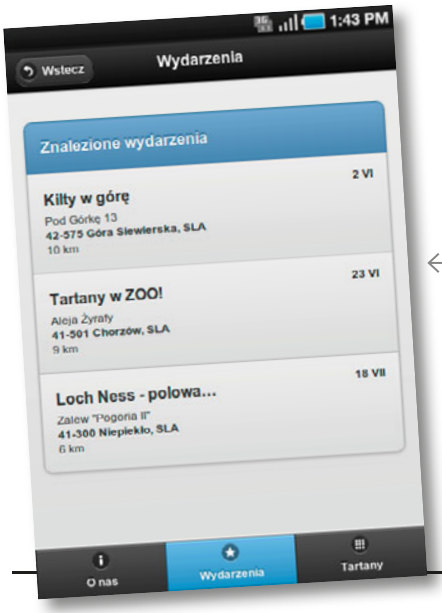
Chłopaki, to wygląda wyśmienicie! Właśnie tak to sobie wyobrażałem, kiedy pierwszy raz wpadł mi do głowy pomysł na mobilną aplikację.

Super! Udało nam się zintegrować kilka elementów, dzięki którym możemy o naszej mobilnej aplikacji powiedzieć „wyjątkowa": stopniowe ulepszanie, tryb offline i wykorzystanie geolokalizacji.



WYSIL SZARE KOMÓRKI

Zadanie dodatkowe: w jaki sposób można by dodać do formularza pole typu `range`, by użytkownik mógł ustalać promień poszukiwań? Kiedy pole powinno się pojawiać? Kod działający po stronie serwera jest już przygotowany, więc wystarczy, że dodasz pole o nazwie `radius`.



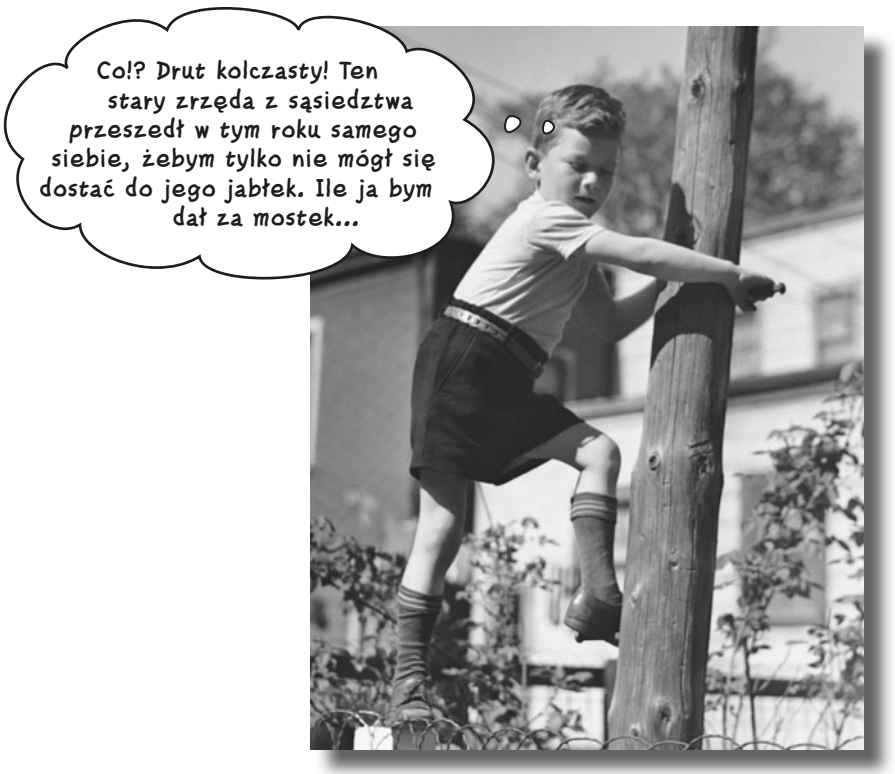
Wygląda dobrze w wielu różnych przeglądarkach mobilnych.

Tak wygląda w Androidzie.

CELNE SPOSTRZEŻENIA

- Mobilne aplikacje internetowe, które są wygodne i zachowują się podobnie jak aplikacje natywne, często charakteryzują się **stopniowym ulepszaniem, dobrze działającym trybem offline oraz wykorzystaniem geolokalizacji**.
- Rozpoczynanie od **podstawowej wersji** pozwala na dotarcie do tak wielu użytkowników, jak to tylko możliwe. Dzięki zastosowaniu **stopniowego ulepszania** możemy przygotować olśniewającą aplikację dla bardziej wydajnych urządzeń.
- Dostarczenie możliwości **pracy offline** jest bardzo ważne, ponieważ w przypadku urządzeń mobilnych trudno zapewnić stały dostęp do internetu.
- Aby skorzystać z mechanizmu **podręcznej pamięci aplikacji**, musimy utworzyć **plik manifestu** informujący przeglądarkę o zasobach, które mają być dostępne offline.
- Budowa pliku manifestu jest prosta, ale trzeba uważać na **wiele pułapek** tego mechanizmu.
- W pliku manifestu możemy wskazać zasoby, które mają zostać pobrane do podręcznej pamięci i być dostępne offline (**sekcja `CACHE`**), jak również te, które za każdym razem — przy aktywnym połączeniu — mają być pobierane z serwera (**sekcja `NETWORK`**).
- **Geolokalizacja** jest obsługiwana przez większość nowoczesnych smartfonów. W wielu z nich zaimplementowano **interfejs API zgodny ze specyfikacją W3C**, który udostępnia z poziomu JavaScript obiekt `navigator.geolocation`.
- Formularz Znajdź wydarzenie zintegrowaliśmy z geolokalizacją w taki sposób, by była dostępna na tych urządzeniach, które ją obsługują. Zrobiliśmy to w podobny sposób jak w przypadku formularza edytora tartanów.
- Niektóre przeglądarki mobilne, zwłaszcza dostępne na starszych smartfonach, **mają zaimplementowaną geolokalizację niezgodną z W3C**. Aby zapewnić ich obsługę, skorzystaliśmy z zewnętrznej biblioteki geo-location-javascript.
- **Biblioteka geo-location-javascript** emuluje API W3C, więc nie wymaga wprowadzania większych zmian w istniejącym kodzie JavaScript nastawionym na API zgodne z W3C.

Ustrzel tartan! — w stronę natywności

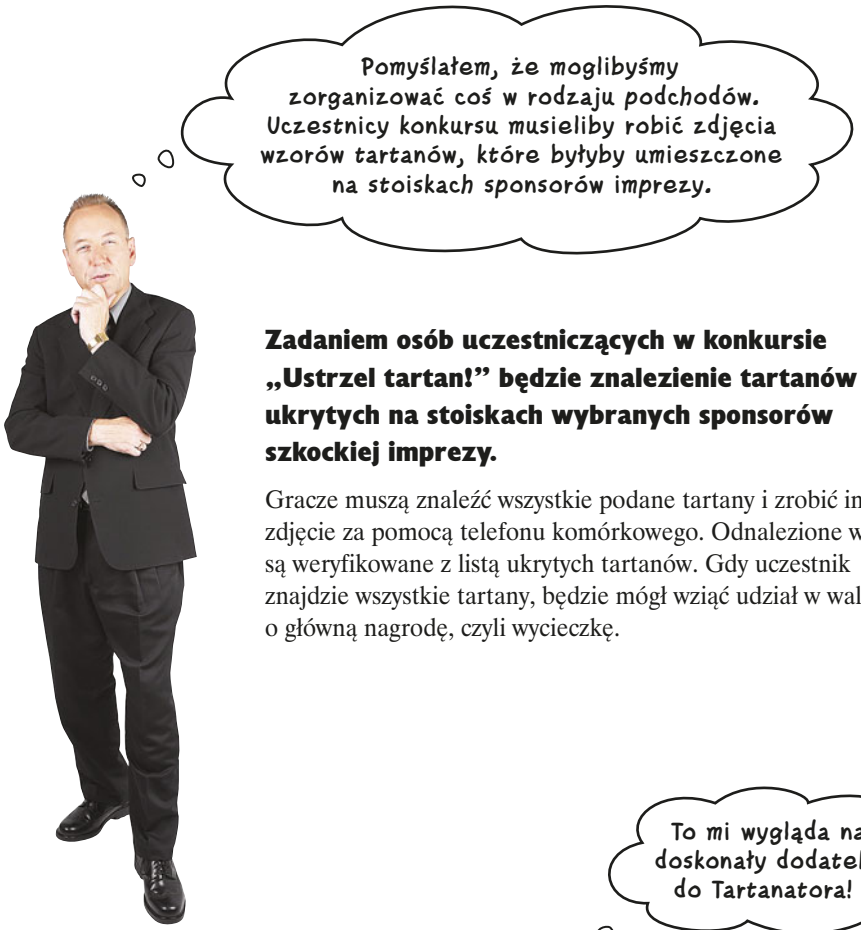


Czasem musisz się zdecydować na aplikację natywną. Być może potrzebujesz dostępu do czegoś, co nie jest (jeszcze) osiągalne z poziomu przeglądarki. A może klient chce, by aplikacja *koniecznie* znalazła się w App Store. Nie możemy się już doczekać chwili, gdy przeglądarka będzie udostępniała wszystko, co jest potrzebne, by tworzyć pełnoprawne aplikacje mobilne. Jednak zanim do tego dojdzie, możemy skorzystać z możliwości **hybrydowego podejścia** — nadal będziemy tworzyć aplikacje z wykorzystaniem **technologii internetowych**, ale użyjemy **biblioteki, która pełni rolę mostu** między naszym kodem a natywnymi możliwościami urządzeń. **Międzyplatformowe natywne aplikacje zbudowane w oparciu o technologie internetowe?** Brzmi całkiem nieźle!

Nowe możliwości

Szkockie linie lotnicze Loch Air zgodziły się zasponsorować główną nagrodę — wycieczkę dla dwojga do Edynburga.

Jedną z atrakcji Święta Szkota, które ma się odbyć za dwa miesiące, jest konkurs organizowany przez sponsora, linie lotnicze Loch Air, we współpracy z organizacją Tartany bez Granic kierowaną przez Ewana.



Zadaniem osób uczestniczących w konkursie „Ustrzel tartan!" będzie znalezienie tartanów ukrytych na stoiskach wybranych sponsorów szkockiej imprezy.

Gracze muszą znaleźć wszystkie podane tartany i zrobić im zdjęcie za pomocą telefonu komórkowego. Odnalezione wzory są weryfikowane z listą ukrytych tartanów. Gdy uczestnik znajdzie wszystkie tartany, będzie mógł wziąć udział w walce o główną nagrodę, czyli wycieczkę.

Niestety jest jeden problem.

Z poziomu przeglądarki internetowej dostęp do kamery telefonu nie jest możliwy. Podobnie sprawa wygląda z innymi możliwościami urządzeń mobilnych, takimi jak nagrywanie dźwięku, operacje na systemie plików, informacje o dostępie do sieci komórkowej, lista kontaktów itp.

Hm... Zdjęcia. Przecież z poziomu przeglądarki internetowej nie mamy dostępu do kamery. Co zrobimy? Może musimy napisać aplikację natywną?

Wiele możliwości oferowanych przez urządzenia mobilne, takie jak na przykład robienie zdjęć, nie jest dostępnych z poziomu przeglądarek internetowych.

To się powoli zmienia na lepsze, ale pozostaje jeszcze wiele spraw, których nie da się załatwić z poziomu przeglądarki.

Łukasz: Nie ma mowy. Nie mamy ani wystarczająco dużo czasu, ani budżetu, żeby napisać od nowa Tartanatora jako aplikację natywną dla kilku platform, a poza tym jeszcze dodać to, o czym ostatnio wspomniał Ewan.

Kuba: Czy to znaczy, że musimy sobie dać z tym spokój? Byłoby szkoda...

Przemek: Zastanawiam się, czy nie znalazłoby się jakieś kompromisowe rozwiązanie. Może zastanowilibyśmy się nad *aplikacją hybrydową*?

Kuba: A co to takiego?

Przemek: Aplikacja hybrydowa to natywna aplikacja utworzona za pomocą technologii internetowych. Ponieważ jest natywna, mamy dostęp do urządzeń, jak choćby kamery, ale sam kod jest pisany zgodnie ze standardami technologii internetowych, dzięki czemu istnieje możliwość jego współdzielenia dla wielu platform.

Łukasz: Ale jak to działa, skoro opieramy się na technologiach internetowych, a jakieś funkcjonalności nie są obsługiwane przez przeglądarkę?

Przemek: Jest kilka frameworków, które stanowią coś w rodzaju mostu między aplikacjami internetowymi a natywnym kodem. Wygląda to tak: biorą aplikację webową, chwilę ją przeżuwają i wypluwają natywne aplikacje dla różnych platform. Biorą na siebie wszystkie sprawy związane ze zgodnością i specyfiką poszczególnych platform, więc my nie musimy się tym przejmować.

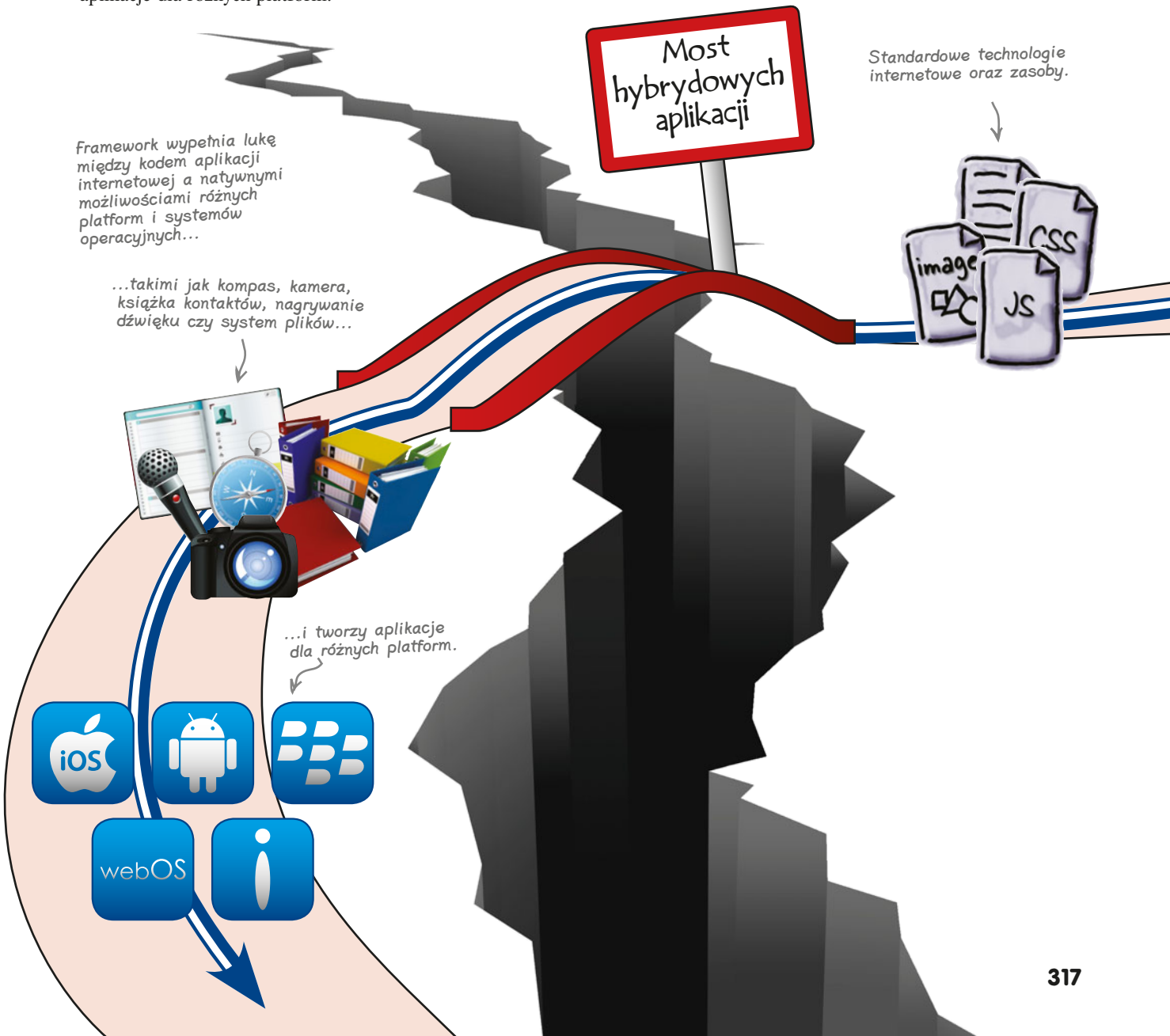
Kuba: Zatem powinniśmy zmienić Tartanatora w jedną z tych hybrydowych aplikacji?

Przemek: Nie mamy czasu, by zamieniać całą aplikację i ją na nowo testować. Mamy krótki termin, więc może zaproponujemy Ewanowi, że przygotujemy natywną aplikację tylko na potrzeby konkursu?

Łukasz: Podoba mi się ten pomysł. Nie będziemy ryzykować, że podczas przenoszenia Tartanatora coś pójdzie nie tak. Zainteresowane osoby będą mogły po prostu pobrać hybrydową aplikację na czas konkursu. Słuchaj, Przemek, jeżeli jesteś pewien, że to zadziała na kilku platformach, przedstaw Ewanowi nasz pomysł.

Jak działają aplikacje hybrydowe?

Piszesz kod, korzystając ze standardowych technologii internetowych, takich jak HTML5, CSS i JavaScript, czyli postępujesz dokładnie tak samo jak przy budowie witryny internetowej. Framework stanowi most dostarczający interfejs API, z którego możesz korzystać, by uzyskać dostęp do natywnych funkcjonalności różnych platform. W ten sposób jest wypełniana luka między Twoim kodem JavaScript a natywnym kodem działającym na urządzeniu (jeśli dane urządzenie i platforma mają określoną funkcjonalność). Wynikiem tych działań są natywne aplikacje dla różnych platform.



Budowanie mostu za pomocą PhoneGap

PhoneGap to platforma HTML5 typu open source, która umożliwia tworzenie natywnych aplikacji dla różnych systemów operacyjnych działających na urządzeniach mobilnych. Wykorzystywane jest tu rozwiązanie, o którym mówiliśmy wcześniej. Piszesz kod tak jak zwykle, używając HTML-a, CSS i JavaScriptu. PhoneGap udostępnia spójne javascriptowe API, które spina aplikację internetową z natywnym kodem.

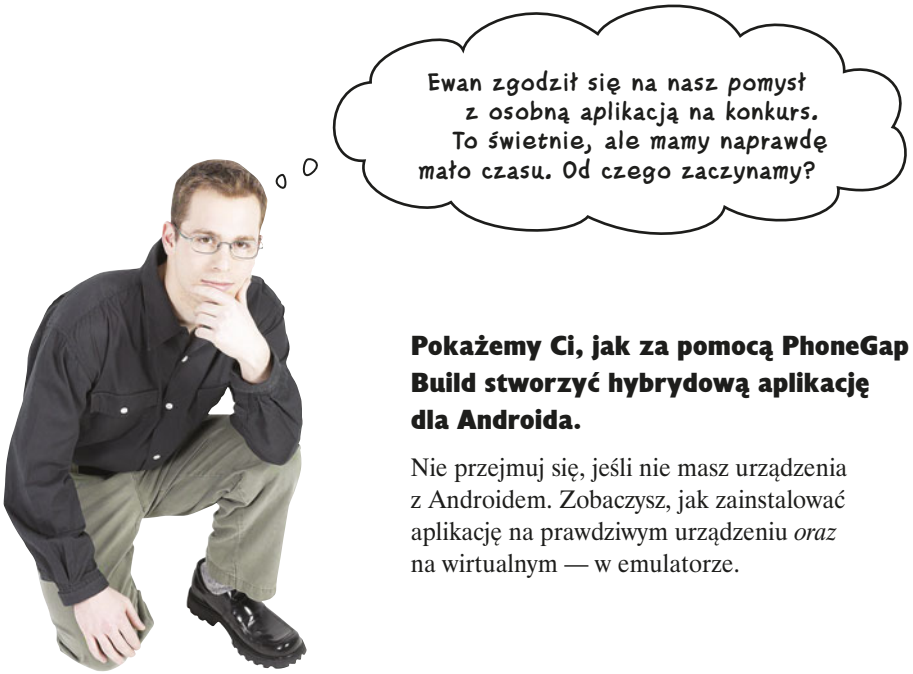
Platforma PhoneGap powstała w firmie Nitobi z Vancouver, która w październiku 2011 roku została przejęta przez Adobe. Nie martw się — firma Adobe wykonała już niezbędne kroki, by platforma PhoneGap pozostała na zasadach open source, przekazując prawa fundacji non profit Apache Software Foundation, która utworzyła na potrzeby platformy projekt Apache Cordova.

PhoneGap Build

PhoneGap Build to internetowy serwis umożliwiający kompilowanie kodu w chmurze równocześnie dla kilku platform. Dzięki temu nie musisz się męczyć z instalowaniem SDK, wtyczek i środowisk programistycznych dla każdej platformy, którą zamierzasz wspierać.

PhoneGap wspiera siedem platform, ale w tym rozdziale utworzymy aplikacje dla Apple iOS, Androida, BlackBerry, WebOS i Symbiana.

Chociaż w standardowym procesie tworzenia aplikacji z PhoneGap kompiluje się ją samodzielnie, z PhoneGap Build wystarczy, że prześlesz spakowane źródła (w pliku ZIP) lub wskażesz adres repozytorium, a on zrobi resztę.



Pokażemy Ci, jak za pomocą PhoneGap Build stworzyć hybrydową aplikację dla Androida.

Nie przejmuj się, jeśli nie masz urządzenia z Androidem. Zobaczysz, jak zainstalować aplikację na prawdziwym urządzeniu *oraz* na wirtualnym — w emulatorze.

Na przykładową platformę wybraliśmy Androida z kilku powodów.

Przede wszystkim Android to popularna platforma. Z drugą z najbardziej popularnych platform, czyli iOS, jest pewien problem — aby tworzyć dla niej aplikacje, trzeba być członkiem programu iOS Developer Program (a to się wiąże z kosztem 99 dolarów na rok). Nie chcieliśmy Cię narażać na taki wydatek.

Android ma ogólnie dostępne SDK działające na wielu platformach, które zawiera między innymi emulator wielu urządzeń z różnymi wersjami systemu Android. Z kolei w przypadku systemu iOS mamy do czynienia ze środowiskiem programistycznym Xcode, które zadziała tylko na Macu.

Czy Twoje środowisko programistyczne jest gotowe?

Jeśli jeszcze tego nie zrobiłeś, wstrzymaj się z czytaniem rozdziału i przejdź do dodatku D. Zanim przejdziesz dalej, musisz zainstalować oprogramowanie oraz nauczyć się instalować (i odinstalowywać) aplikacje na wirtualnych i rzeczywistych urządzeniach.

Obejrzyj to!

Naprzód, JavaScripcie!

Nie musisz być mistrzem JavaScriptu, by przejść przez ten rozdział, ale trudno zaprzeczyć, że w dzisiejszych czasach znajomość tego języka jest niezbędna, jeśli się myśli o tworzeniu aplikacji internetowych.

Aby komunikować się z natywnymi funkcjonalnościami, takimi jak API kamery, musimy rozwiązać sprawę po stronie klienta (czyli przeglądarki), a to wymusza użycie JavaScriptu.

Nie istnieją głupie pytania

P: Jakie inne narzędzia, produkty lub usługi mogą pomóc w budowaniu hybrydowych aplikacji?

O: Dwie z najbardziej znanych alternatyw dla PhoneGap to Appcelerator Titanium oraz Sencha Touch 2.

P: Jakie platformy są wspierane przez PhoneGap?

O: PhoneGap obecnie wspiera siedem platform: Android, Bada, BlackBerry, iOS, Symbian, WebOS i Windows Phone 7. Szczegółowe informacje dotyczące obsługiwanych funkcjonalności na poszczególnych platformach można znaleźć na stronie *http://phonegap.com/about/feature*.

P: Czy usługa PhoneGap Build wspiera te same platformy co PhoneGap?

O: Niezupełnie. W chwili pisania książki nie były jeszcze obsługiwane platformy Windows Phone 7 i Bada.

P: Czy za korzystanie z PhoneGap nic się nie płaci?

O: Tak. PhoneGap, a właściwie Apache Cordova, to całkowicie darmowy projekt open source.

P: A co z usługą PhoneGap Build? Czy ona też jest za darmo?

O: To zależy. PhoneGap Build oferuje kilka poziomów usług. Bezpłatny wariant obejmuje jedną prywatną aplikację oraz dowolną liczbę publicznych. W omawianym tu przykładzie aplikacji Ustrzel tartan! skorzystamy właśnie z tej możliwości. Pozostałe warianty oferują więcej prywatnych aplikacji oraz możliwość tworzenia projektu przez wieloosobowe zespoły.

P: Dlaczego muszę zainstalować SDK Androida?

O: Nie będziemy co prawda kompilować aplikacji Ustrzel tartan! (zrobi to za nas PhoneGap Build), ale musimy ją zainstalować i przetestować w emulatorze, a najlepiej też na urządzeniu (jeżeli masz telefon z Androidem). Aby to zrobić, będziemy musieli skorzystać z narzędzi dostępnych w SDK.

Nie musimy jednak instalować środowiska programistycznego Eclipse, które byłoby przydatne, gdybyśmy tworzyli i kompilowali aplikację samodzielnie. Jak widać, dzięki PhoneGap udało nam się uniknąć przynajmniej jednego zadania.

P: Dlaczego niektóre funkcjonalności nie są dostępne z poziomu przeglądarki internetowej?

O: Jest kilka powodów, dla których na urządzeniach mobilnych nie wszystko można zrobić z poziomu przeglądarki.

Jednym z ważniejszych jest bezpieczeństwo. Umożliwienie przeglądarce dostępu do elementów, takich jak choćby system plików albo lista kontaktów, wiązałoby się z koniecznością zapewnienia odpowiedniego poziomu bezpieczeństwa. Przygotowanie mechanizmów, które w odpowiedni sposób zabezpieczą tego typu dane, wymaga czasu i rozwiązania wielu problemów.

Poza tym niektóre z proponowanych standardów są zupełnie nowe i niekompletne, a niekiedy zakładają wiele wariantów. W związku z tym, kiedy wszystkie sprawy zostaną już ustalone, z całą pewnością pojawią się problemy z obsługą tych funkcjonalności w różnych przeglądarkach, których twórcy zastosowali różne rozwiązania.

To wszystko są problemy, które będą rozwiązywane w najbliższej przyszłości, więc miej oczy otwarte i staraj się nadążyć za zmianami.

WYSIL SZARE KOMÓRKI

Jeśli masz urządzenie z systemem innym niż Android, ale *wspieranym* przez usługę PhoneGap Build, albo jesteś członkiem programu iOS Developer Program, możesz zbudować i testować omawianą aplikację na swojej platformie.

Podczas pracy nad tym projektem z powodzeniem przetestowaliśmy aplikację Ustrzel tartan! na różnych wersjach systemów Android i iOS.

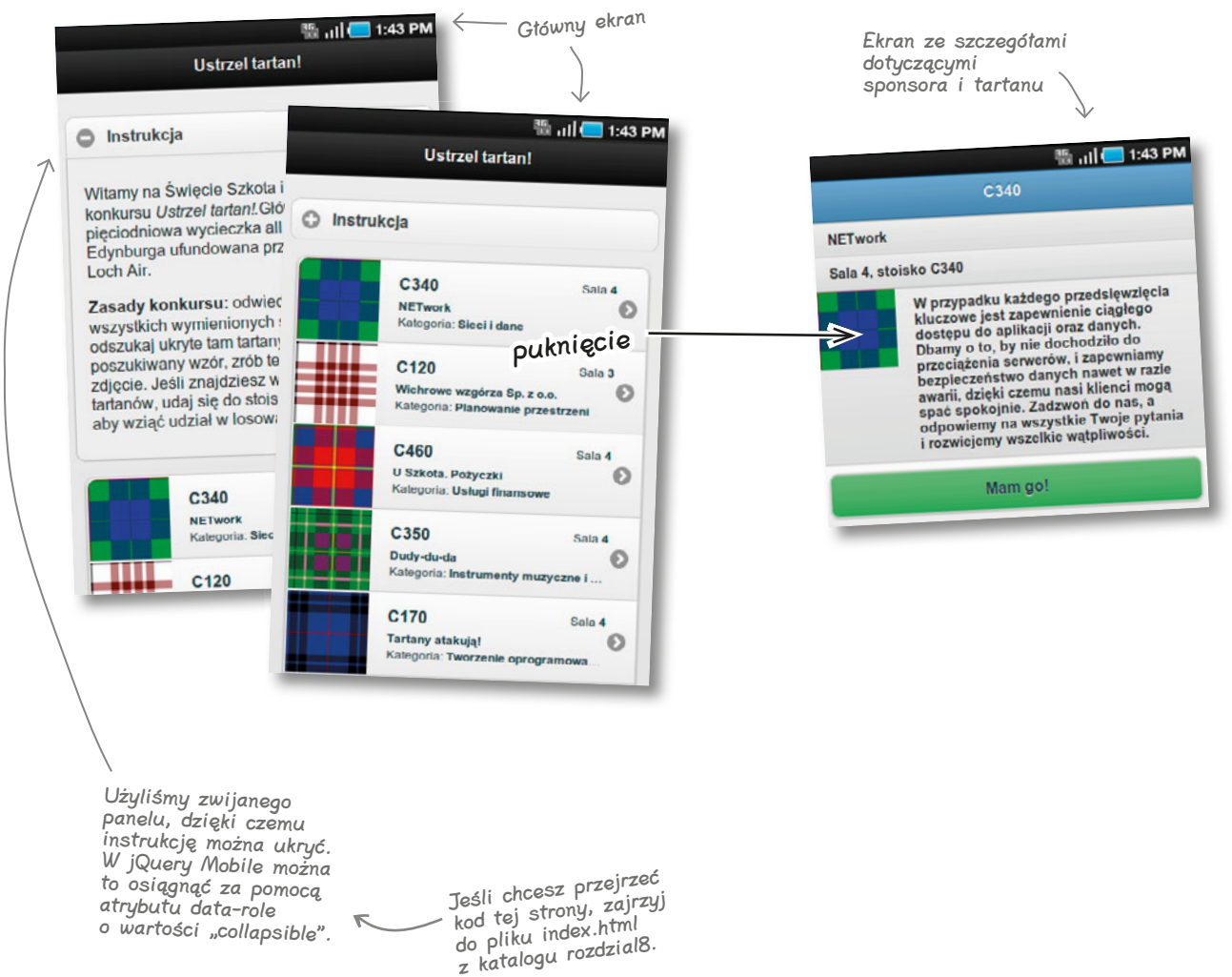
Dołącz do PhoneGap Build

Zrób to sam!

Przejdź na stronę *https://build.phonegap.com* i utwórz konto. Zajmuje to tylko chwilę i nie wiąże się z żadnymi opłatami.

Jak ma działać aplikacja?

Na głównym ekranie naszej aplikacji ma być wyświetlany opis konkursu i lista tartanów, które trzeba znaleźć. Dotknięcie wybranego elementu powoduje wyświetlenie bardziej szczegółowych informacji o stoisku sponsora, na którym można znaleźć dany tartan.



Śledzenie ustrzelonych tartanów

W ukończonej aplikacji kliknięcie przycisku *Mam go!* spowoduje uruchomienie aparatu telefonu, dzięki czemu gracz będzie mógł zrobić zdjęcie upolowanego tartanu. Zdjęcie zostanie wyświetlone na stronie ze szczegółowymi informacjami, a na liście wszystkich wzorów na głównym ekranie zostanie zaznaczone jako znalezione.

Mamy już podstawowy układ strony w HTML-u, arkusz CSS i obrazki. **Teraz musimy sprawić, by aplikacja coś robiła, oraz za pomocą PhoneGap Build zbudować prawdziwą natywną aplikację dla Androida.**

Co musimy zrobić?

- ☐ Trzeba stworzyć i skonfigurować projekt PhoneGap Build, a następnie spakować wszystkie pliki projektu (HTML, CSS i obrazki), zbudować aplikację oraz zainstalować ją na urządzeniu z Androidem lub na emulatorze.
- ☐ Musimy umożliwić użytkownikom zaznaczanie tartanów, które udało im się znaleźć.
- ☐ Musimy dodać funkcjonalność zapisywania zdjęć znalezionych tartanów.

Anatomia projektu Ustrzel tartan!

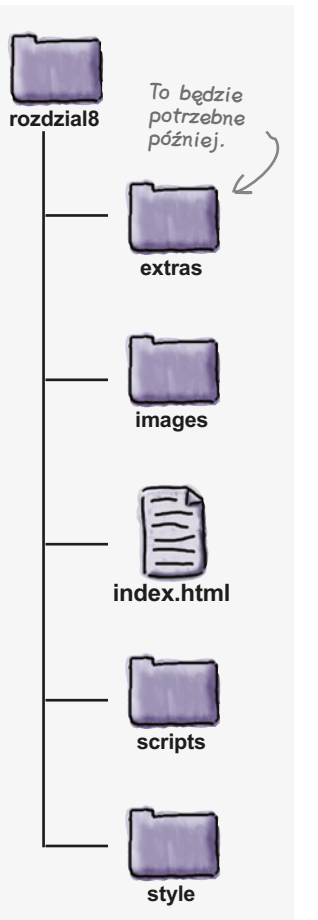
Poniżej przedstawiliśmy uproszczoną strukturę bloku `<div>` z treścią znajdującą się w pliku *index.html*.

Strukturę zawartości tworzy lista `<ul>` z zagnieżdżonymi kolejnymi listami `<ul>`.

```
<div data-role="content">
   <div data-role="collapsible">...</div>
   <ul id="vendors">
      <li>
         <ul class="details">
            <li>...</li>
            <li>...</li>
            ...
         </ul>
      </li>
      <li>
         <ul class="details">
            <li>...</li>
            <li>...</li>
            ...
         </ul>
      </li>
      ...
   </ul>
</div>
```

Lista ul#vendors ma po jednym elemencie `<li>` na każdy tartan, który trzeba znaleźć.

Każdy element `<li>` reprezentujący tartan ma zagnieżdżoną listę (ul.details) zawierającą dodatkowe informacje o sponsorze oraz przycisk „Mam go!".



W tym projekcie znów skorzystamy z frameworku jQuery Mobile, dzięki czemu wygląd aplikacji będzie spójny z pozostałą częścią Tartanatora.

Jesteśmy już prawie gotowi na PhoneGap Build

To, co już mamy, jest prawie gotowe na wrzucenie do PhoneGap Build.

Projekty PhoneGap Build mają strukturę bazującą na **specyfikacji organizacji W3C dla widżetów internetowych**. Widżety to pewnego rodzaju aplikacje internetowe, które są traktowane jak samodzielne aplikacje. W najprostszym przypadku zawierają co najmniej jeden *plik startowy* (u nas jest to plik *index.html*) oraz *plik konfiguracyjny* (w formacie XML). Projekt można uzupełnić o dowolną liczbę plików wymaganych przez aplikację — obrazków, arkuszy CSS czy skryptów JavaScript.

Chcesz przeczytać tę specyfikację? Znajdziesz ją tu: www.w3.org/TR/widgets/.

Do celu mamy już całkiem niedaleko. Musimy jeszcze **utworzyć plik konfiguracyjny**, aby przekazać PhoneGap Build kilka informacji na temat aplikacji.

PhoneGap Build wymaga, aby w projekcie znajdował się plik *config.xml*. Zajmijmy się nim teraz!

Ćwiczenie

Stwórzmy w końcu naszą pierwszą aplikację z PhoneGap Build. Potrzebny nam będzie plik konfiguracyjny. Utwórz plik i zapisz go w katalogu *rozdzial8* pod nazwą *config.xml*. W pliku wprowadź informacje o aplikacji, korzystając z poniższego szablonu.

Tu wpisz nazwę i opis aplikacji.

```
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns     = "http://www.w3.org/ns/widgets"
        xmlns:gap = "http://phonegap.com/ns/1.0"
        id        = ""
        version   = "1.0.0">
   <name></name>
   <description></description>
   <author href=""
      email="">TU WPISZ IMIĘ I NAZWISKO
   </author>
   <icon src="images/touch-icon-iphone4.png" height="114" width="114" />
</widget>
```

Identyfikatorem aplikacji może być com.hfmw.tartanhunt.

Adres Twojej witryny internetowej i e-mail. Jeśli nie masz własnej witryny, możesz wpisać adres http://helion.pl.

Dodajmy ikony! Pojawią się na głównym ekranie użytkownika. Pierwszą ikonę dodaliśmy za Ciebie. Znajdź pozostałe pliki w katalogu images i też je dodaj.

config.xml

Pamiętaj, że projekt PhoneGap Build może zostać zbudowany albo z pliku ZIP, albo z repozytorium. W prezentowanym przykładzie skorzystamy z pierwszej możliwości.

Spakuj do formatu ZIP całą zawartość katalogu *rozdzial8*. Nie ma znaczenia, jak będzie się nazywał plik archiwum, pod warunkiem że będzie miał rozszerzenie *.zip*.

Rozwiązanie ćwiczenia

Utworzony plik konfiguracyjny *config.xml* powinien wyglądać podobnie jak poniższy. Oczywiście możesz wpisać swoje imię i nazwisko oraz inne związane z Tobą dane.

```
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns      = "http://www.w3.org/ns/widgets"
        xmlns:gap = "http://phonegap.com/ns/1.0"
        id         = "com.hfmw.tartanhunt"
        version    = "1.0.0">
   <name>Ustrzel tartan!</name>
   <description>Aplikacja do konkursu organizowanego w związku ze Świętem Szkota:
wygraj wycieczkę do Edynburga dla dwojga ufundowaną przez linie lotnicze Loch Air!</
description>
   <author href="http://www.swietoszkota.org"
      email="biuro@swietoszkota.org">
      Maria Stewart
   </author>
   <icon src="images/touch-icon-iphone4.png" height="114" width="114" />
   <icon src="images/touch-icon-ipad.png" height="72" width="72" />
   <icon src="images/touch-icon-iphone.png" height="57" width="57" />
   <icon src="images/icon-48.png" height="48" width="48" />
   <icon src="images/icon-36.png" height="36" width="36" />
</widget>
```

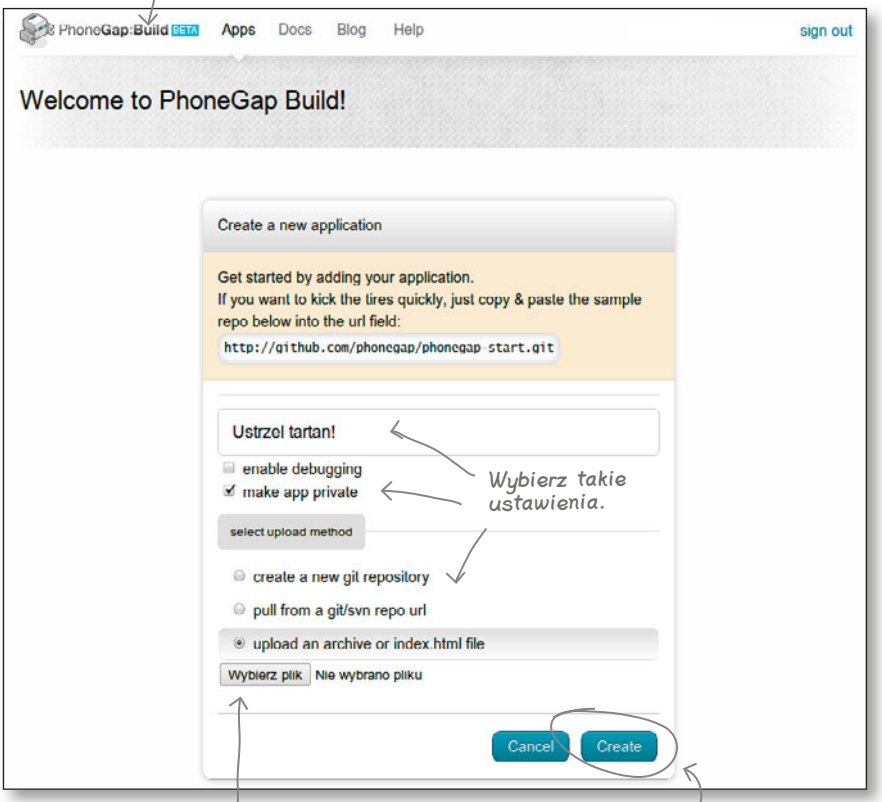
PhoneGap Build przyporządkuje prawidłową ikonę do danego urządzenia.

config.xml

Zaloguj się do PhoneGap Build i przejdź na zakładkę Apps. Czas utworzyć aplikację!

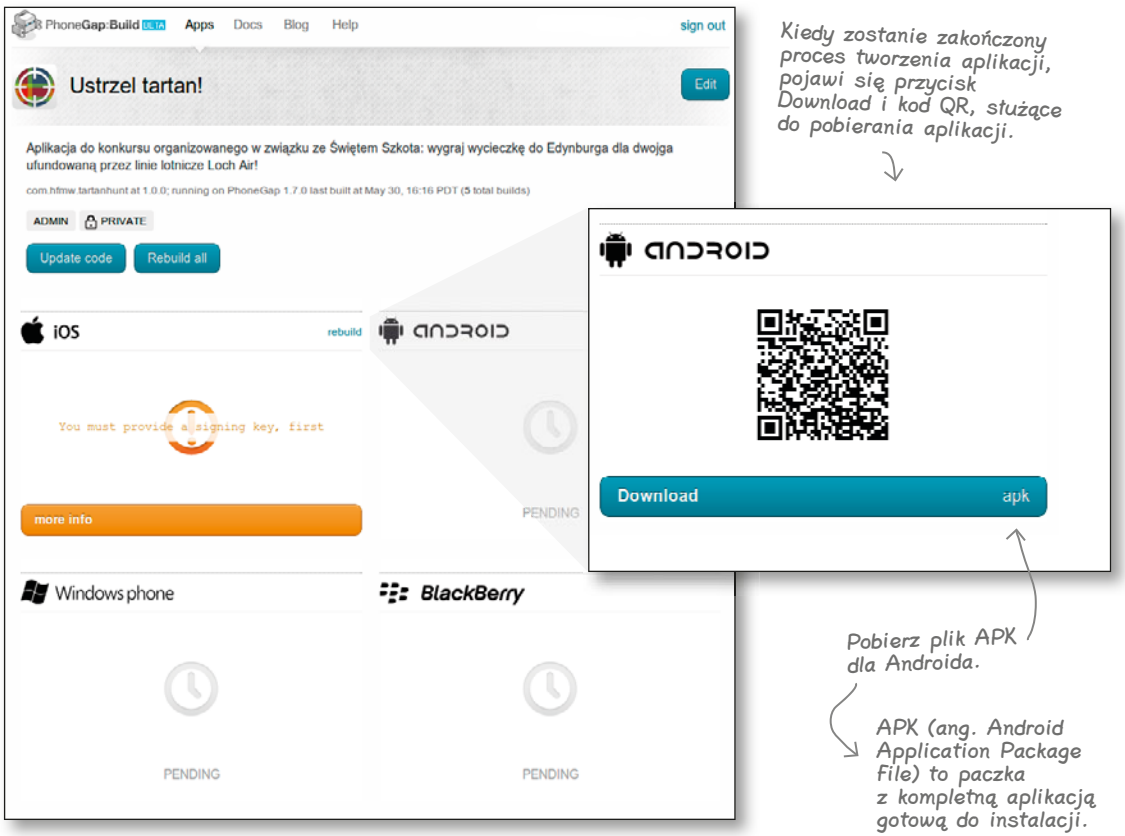
Rozwiązanie ćwiczenia

https://build.phonegap.com



Pobieranie utworzonej aplikacji

Po przesłaniu pliku ZIP PhoneGap Build kolejkuje aplikację w procesie budowania. Poczekaj chwilę i odśwież stronę. Powinieneś zobaczyć przycisk *Download* przy wszystkich platformach, dla których aplikacja została prawidłowo zbudowana. Chcemy pobrać paczkę dla platformy Android (plik APK).



No dobra, zainstalujmy tę aplikację.

Wybierz drogę

Zainstaluj na emulatorze

Aby przygotować się do instalowania aplikacji na wirtualnym urządzeniu, musisz uruchomić narzędzie Android SDK Manager. Z menu *Tools* wybierz polecenie *Manage AVDs*, aby pojawiła się lista zdefiniowanych wirtualnych urządzeń Androida (ang. *Android Virtual Device*, AVD). Uruchom urządzenie z najnowszą wersją platformy. Podczas instalowania aplikacji musi być uruchomione emulowane urządzenie, ale uważaj, żeby nie instalować podczas trwania procesu uruchamiania.

LUB

Zainstaluj na prawdziwym urządzeniu

Jeśli masz urządzenie mobilne z Androidem, to super! Podłącz je kablem USB do komputera. Tak, to wszystko.

Cel, pal!

Proces instalowania aplikacji wygląda tak samo w przypadku wirtualnego i rzeczywistego urządzenia. Otwórz okno terminala (w systemach Mac lub Linux), a w systemie Windows w polu wyszukiwania w menu *Start* wpisz polecenie **cmd**, aby zostało wyświetlone okno wiersza poleceń. Następnie przejdź (za pomocą polecenia **cd**) do katalogu, w którym znajduje się pobrany plik APK.

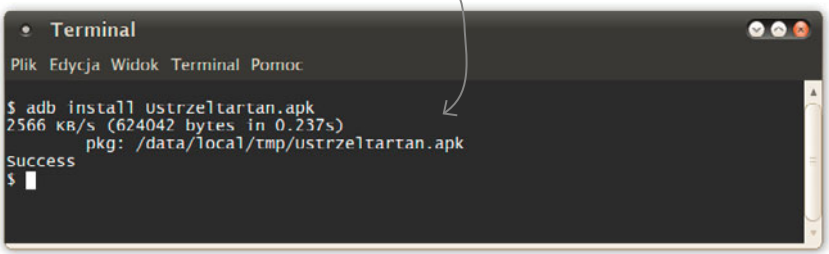
Polecenie powodujące zainstalowanie aplikacji:

```
adb install <nazwa pliku APK>
```

Więcej na ten temat znajdziesz w dodatku D.

Pamiętaj, że zanim wykonasz to polecenie, musisz podłączyć (prawdziwe) urządzenie lub w pełni uruchomić emulator.

W naszym przypadku plik APK nosi nazwę Ustrzeltartan.apk.



Powinieneś zobaczyć coś podobnego.

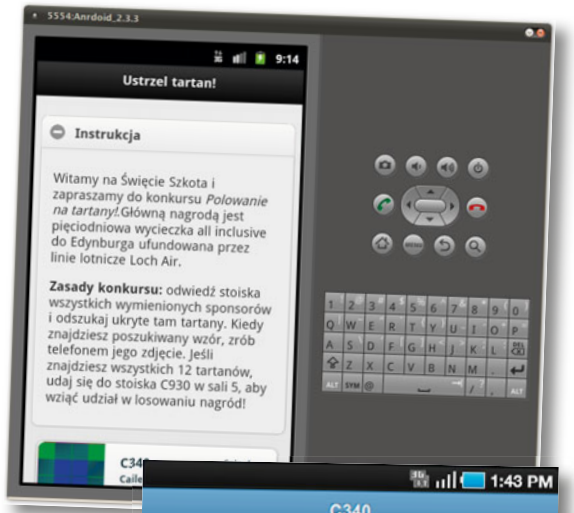
Najlepiej, jeśli użyjesz wirtualnego urządzenia dla Androida 2.3 lub 4.0. Wcześniejsze wersje platformy działają w emulatorze niezbyt dobrze i mają problemy z obsługą kamery (która będzie nam potrzebna później).

W emulatorze lub na urządzeniu powinieneś po chwili zobaczyć wśród innych aplikacji ikonę Ustrzel tartan!. Uruchom ją.

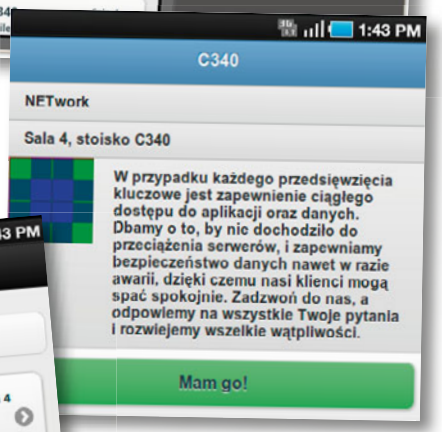
Pierwsze uruchomienie aplikacji może zająć trochę więcej czasu — nawet około 30 sekund.

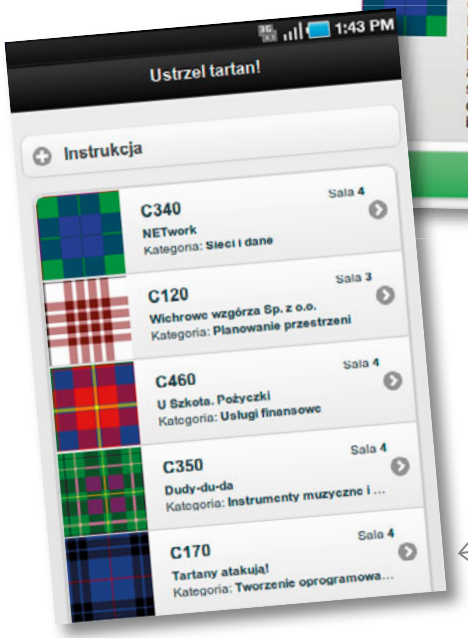
Ikona aplikacji Ustrzel tartan! pojawi się wśród innych aplikacji.

Główny ekran aplikacji w emulatorze Androida w wersji 2.3.3.



Jedna z zagnieżdżonych podstron wyświetlona w urządzeniu Nexus One.





Główny ekran ze zwiniętą instrukcją wyświetlony w emulatorze Androida.

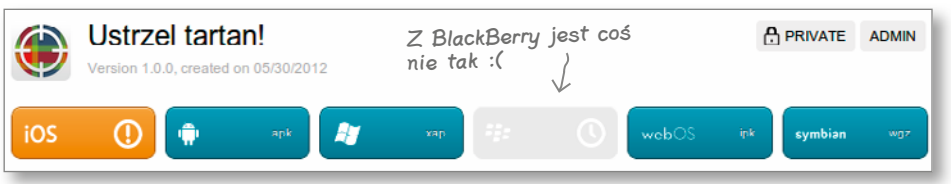
Główny ekran na platformie iOS. Jeśli masz klucz do podpisywania aplikacji, możesz tworzyć aplikacje pod ten system.

Framework jQuery Mobile automatycznie konwertuje zagnieżdżone listy <ul> na oddzielne „strony”.

Dzieje się tak dlatego, że każda zagnieżdżona lista jest zamieniana na blok `<div>` z atrybutem `data-role` równym `page`. Kiedy dany element listy zostanie kliknięty, aktywowana jest odpowiednia „strona”. Tak naprawdę z technicznego punktu widzenia jest tylko jedna *strona*. Zwróciłeś pewnie też uwagę na to, że jako tytuł zagnieżdżonej strony jest używany tytuł nadrzędnego elementu listy (w naszym przypadku jest to numer stoiska sponsora).

Hej, a co z BlackBerry?

Kiedy przejdziesz na zakładkę aplikacji w PhoneGap Build, zobaczysz pewnie coś takiego:



Sprawę iOS możemy wyprostować, dostarczając klucz do podpisywania aplikacji (jeżeli taki mamy).

O co chodzi z tym BlackBerry? Niestety PhoneGap Build ma problem w przypadku BlackBerry z nazwami plików zawierających łączniki. Problem w tym, że wiele nazw plików składających się na framework jQuery Mobile zawiera te znaki.

Ponieważ nie chcemy zwariować, a Ewan wahał się w sprawie wsparcia dla BlackBerry, nie będziemy się tym zajmować.

WYSIL SZARE KOMÓRKI

Aby aplikacja Ustrzel tartan! zbudowana przez PhoneGap Build działała na platformie BlackBerry, powinieneś pozmieniać nazwy plików ikon oraz zaktualizować wszystkie odwołania w kodzie źródłowym jQuery Mobile. Chociaż nie zajęłoby to zbyt wiele czasu, tego typu „hakowanie” frameworku nie jest najlepszym pomysłem.
Co o tym myślisz? Czy gra jest warta świeczki?

Ćwiczenie

Dodamy teraz ekran powitalny, by podczas ładowania aplikacji użytkownicy nie musieli patrzeć na pusty, nieciekawy ekran. Odpowiedź na pytanie, jak to zrobić, znajdziesz w dokumentacji pliku *config.xml* dostępnej na stronie PhoneGap Build pod adresem *https://build.phonegap.com/docs/config-xml*. Obrazy ekranu powitalnego znajdziesz w katalogu *rozdzial8/extras/images*. Przenieś je do katalogu *rozdzial8/images*.

Zaktualizuj plik *config.xml*, spakuj ponownie cały katalog *rozdzial8* i prześlij do PhoneGap Build. Następnie przeinstaluj aplikację, by sprawdzić, czy pojawi się ekran powitalny.

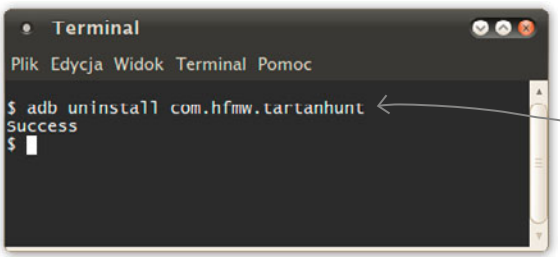
Aby ponownie przesłać spakowany plik, wciśnij przycisk Update code. Aplikacja zostanie automatycznie przebudowana po załadowaniu pliku.

Rozwiązanie na stronie 334

Obejrzyj to!

Zanim ponownie zainstalujesz aplikację, musisz ją najpierw odinstalować.

Za każdym razem, gdy przebudowujesz aplikację w PhoneGap Build i chcesz ją ponownie zainstalować na urządzeniu lub w emulatorze, musisz ją najpierw odinstalować. W tym celu trzeba wykonać poniższe polecenie (po podłączeniu urządzenia lub uruchomieniu emulatora).



Zwróć uwagę, że podczas odinstalowywania używasz identyfikatora pakietu (com.hfmw.tartanhunt), który został zdefiniowany w pliku config.xml, a nie — jak podczas instalowania — nazwy pliku APK.

Niezła robota!

Rządzimy! Mamy już aplikację. Teraz wypadałoby zaimplementować właściwą funkcjonalność. Zadanie podzieliliśmy na dwa etapy.

Przede wszystkim chcemy, by aplikacja zapamiętywała tartany, które zostały znalezione. Kiedy użytkownik kliknie przycisk *Mam go!*, musimy to zapamiętać.

Gratulacje! Właśnie zbudowałeś natywną aplikację mobilną.

Nie było tak źle, prawda? Teraz już wiesz, że tworzenie natywnych aplikacji wcale nie musi być bardzo skomplikowane.

Zapisywanie odnalezionych tartanów

- [x] Trzeba stworzyć i skonfigurować projekt PhoneGap Build, a następnie spakować wszystkie pliki projektu (HTML, CSS i obrazki), zbudować aplikację oraz zainstalować ją na urządzeniu z Androidem lub na emulatorze.

Najpierw zajmiemy się możliwością zapamiętywania tartanów, które zostały odnalezione.

- [ ] **Musimy umożliwić użytkownikom zaznaczanie tartanów, które udało im się znaleźć.**

Później zadbamy o to, by aplikacja umożliwiała robienie zdjęć.

- [ ] Musimy dodać funkcjonalność zapisywania zdjęć znalezionych tartanów.

W jaki sposób mamy „zapamiętać" w aplikacji, które tartany zostały oznaczone przez użytkownika jako odnalezione? Czy będziemy musieli zakasać rękawy i zabrać się za natywny kod?

Na szczęście nie, ponieważ można to zrobić z poziomu JavaScriptu dzięki wprowadzonemu w HTML5 mechanizmowi lokalnego składowania danych, czyli localStorage.

Ten mechanizm jest obsługiwany przez przeglądarki mobilne na wszystkich platformach, które zamierzamy wspierać.

Ten mechanizm nie jest wspierany przez BlackBerry w wersji starszej niż 6., ale przecież i tak nie nastawiamy się na wspieranie BlackBerry.

Kto co widział? Zapisujemy znalezione tartany

Proste dane — w naszym przypadku informacje o znalezionych tartanach — możemy zapisać po stronie klienta, czyli w przeglądarce, dzięki interfejsowi **API localStorage**.

Dlaczego mechanizm localStorage jest taki wyjątkowy?

W przeszłości w sytuacji, gdy było konieczne przechowywanie danych po stronie klienta, twórcy stron często bazowali na **ciasteczkach HTTP**. Ciasteczka mają jednak kilka wad.

Za każdym razem, gdy klient zgłasza żądanie do serwera, przesyłana jest cała zawartość wszystkich ciasteczek dla danej domeny. Może się zdarzyć sytuacja, w której chcielibyśmy przechować po stronie klienta większą ilość danych, np. obrazki lub bogate informacje o konfiguracji, ale ciasteczka na to nie pozwalają (albo przynajmniej bardzo silnie wpływa to na wydajność).

Poza tym do pracy z ciasteczkami jest niezbędny JavaScript, a ich obsługa jest niezbyt zgrabna. Jest jeszcze jedna bardzo ważna sprawa — a co w sytuacji, gdy chcemy zapisać dane, o których serwer wcale nie musi wiedzieć (albo, jak w naszym przypadku, *w ogóle nie ma serwera*)?

Mechanizm lokalnego składowania danych został zaprojektowany tak, by zapisywanie i odczytywanie danych było możliwie proste. Są one zapisywane po stronie klienta w postaci tekstowych par klucz – wartość. Interfejs API udostępnia metody służące do ustawiania, pobierania i czyszczenia tych danych.
I to w zasadzie wszystko na ten temat.

Nasza aplikacja ma już ekran powitalny! Tak to wygląda na iPodzie Touch.

Rozwiązanie ćwiczenia

Oto końcówka pliku *config.xml*, w której dodajemy ekran powitalny. Dostarczamy obraz w dwóch wielkościach, dzięki czemu ekran będzie wyglądać dobrze również na urządzeniach z większymi ekranami.

```
    <icon src="images/icon-48.png" height="48" width="48" />
    <icon src="images/icon-36.png" height="36" width="36" />

    <gap:splash src="images/splash2x.png" width="640" height="960" />
    <gap:splash src="images/splash.png" width="320" height="480" />
</widget>
```

config.xml

W czym nam może pomóc localStorage?

Gdy użytkownik kliknie przycisk *Mam go!*, dodamy do `localStorage` nowy wpis. Podczas sprawdzania, które tartany użytkownik już znalazł, odczytamy z `localStorage` wcześniej zapisane dane.

Poznaj gettery i settery

W API obiektu `localStorage` dwie metody są najważniejsze. Pierwsza z nich ustawia wartość:

Zarówno klucz, jak i wartość muszą być łańcuchami tekstowymi.

Klucz będący nazwą porcji danych, którą zapisujemy.

Wartość, którą chcemy zapisać.

```
localStorage.setItem(klucz, wartość);
```

Później możemy odczytać tak zapisaną wartość, posługując się jej kluczem:

Oto klucz — poproszę o powiązane z nim dane.

```
var storedValue = localStorage.getItem(klucz);
```

Jeśli dla podanego klucza istnieje wartość, zostanie ona przypisana zmiennej storedValue.

Jeśli wartość nie istnieje, zmienna storedValue będzie równa null.

Gdy użytkownik znajdzie na stoisku sponsora, powiedzmy, tartan Douglas i oznaczy go przyciskiem *Mam go!*, możemy zrobić coś takiego:

```
localStorage.setItem('douglas', 'true');
```

Następnie, gdy będziemy chcieli sprawdzić, czy tartan Douglas został znaleziony, wystarczy wywołać metodę `getItem`:

```
var isFound = localStorage.getItem('douglas');
```

Magnesiki z kodem JavaScript dla localStorage

Musimy zaktualizować skrypt *scripts/app.js* (czyli główny kod JavaScript aplikacji), tak by zapisywał znalezione tartany. Uzupełniony kod znajdziesz na kolejnej stronie. Twoim zadaniem jest wstawienie magnesików z komentarzami ponad wierszami kodu, do których się odnoszą.

Każdego magnesu możesz użyć tylko raz, a kilka z nich jest niepotrzebnych!

```
// Odczytaj wpis z localStorage
```

```
// Funkcja obsługi kliknięcia przycisku "Mam go!"
```

```
// Usuń wszystkie wpisy z localStorage
```

```
// Wyłącz przejścia jQM między stronami
```

```
// Utwórz element <a> stylizowany na przycisk
```

```
// Do stron zagnieżdżonej listy dodaj przycisk powrotu
```

```
// Sprawdź, czy przeglądarka obsługuje localStorage
```

```
// Wstaw na stronę przycisk zerowania
```

```
// Zapisz informację, że ten tartan został znaleziony
```

```
// Do przycisku "Mam go!" dodaj funkcję obsługi kliknięcia
```

```
// Gdy struktura DOM jest gotowa, wywołaj funkcję initDevice
```

```
// Pobierz identyfikator klikniętego przycisku
```

```
// Do przycisku zerowania dodaj funkcję obsługi kliknięcia
```

```
// Odśwież ekran, by wyświetlić znalezione tartany
```

```
(function() {
   $(document).bind("mobileinit", function() {

      $.extend($.mobile, { defaultPageTransition: 'none' });

      $.mobile.page.prototype.options.addBackBtn = true;
   });

   var initDevice = function() {

      if (typeof(window.localStorage) == 'object') {

         $('.foundTartan').click(tartanFound);

         addResetButton();
      }
   };

   $(document).ready(initDevice);

   var tartanFound = function(event) {

      var tartanKey = $(event.currentTarget).attr('id');

      localStorage.setItem(tartanKey, 'true');
   };

   var addResetButton = function() {

      var $resetButton = $('<a></a>').attr('data-role','button').html('Zacznij od nowa!');

      $resetButton.click(function() {

         localStorage.clear();

      });

      $resetButton.appendTo($('#booths'));
   };
})();
```

app.js


```
(function() {
  $(document).bind("mobileinit", function() {
    // Wyłącz przejścia jQM między stronami
    $.extend($.mobile, { defaultPageTransition: 'none' });
    // Do stron zagnieżdżonej listy dodaj przycisk powrotu
    $.mobile.page.prototype.options.addBackBtn = true;
  });

  var initDevice = function() {
    // Sprawdź, czy przeglądarka obsługuje localStorage
    if (typeof(window.localStorage) == 'object') {
      // Do przycisku "Mam go!" dodaj funkcję obsługi kliknięcia
      $('.foundTartan').click(tartanFound);

      addResetButton();
    }
  };
  // Gdy struktura DOM jest gotowa, wywołaj funkcję initDevice
  $(document).ready(initDevice);
// Funkcja obsługi kliknięcia przycisku "Mam go!"
  var tartanFound = function(event) {
    // Pobierz identyfikator kliknięty przycisku
    var tartanKey = $(event.currentTarget).attr('id');
  // Zapisz informację, że ten tartan został znaleziony
    localStorage.setItem(tartanKey, 'true');
  };

  var addResetButton = function() {
    // Utwórz element <a> stylizowany na przycisk
    var $resetButton = $('<a></a>').attr('data-role','button').html('Zacznij od nowa!');
    // Do przycisku zerowania dodaj funkcję obsługi kliknięcia
    $resetButton.click(function() {
    // Usuń wszystkie wpisy z localStorage
      localStorage.clear();
    });
  // Wstaw na stronę przycisk zerowania
    $resetButton.appendTo($('#booths'));
  };
})();
```

Wyłączamy przejścia między stronami, ponieważ na niektórych urządzeniach z Androidem są dosyć powolne.

Funkcję obsługi kliknięcia przypisujemy tylko w tych przeglądarkach, które obsługują localStorage.

To samo dotyczy przycisku zerowania.

Wybraliśmy akurat wartość 'true', ale chodzi tu tylko o zapisanie "czegokolwiek".

Jak widzisz, dodajemy przycisk umożliwiający wyzerowanie zapamiętanych tartanów i rozpoczęcie gry od nowa.

Hej, jeszcze nic nie powiedzieliśmy o metodzie clear() — jak myślisz, do czego służy?

Łatwo się domyślić — metoda czyści localStorage, czyli usuwa wszystkie klucze i powiązane z nimi wartości.

app.js

Sprawdzamy, co obsługuje przeglądarka

W rozdziale 2. wspomnieliśmy o **wykrywaniu funkcjonalności wspieranych po stronie klienta**. Tu, w funkcji `initDevice`, zrobimy to samo. Aby sprawdzić, czy przeglądarka obsługuje mechanizm localStorage, wystarczy sprawdzić, czy `window.localStorege` jest obiektem.

```
var initDevice = function() {
   if (typeof(window.localStorage) == 'object')
   {
      $('.foundTartan').click(tartanFound);
      addResetButton();
   }
}
```

Przeprowadzamy tu detekcję możliwości po stronie klienta, by przed wyświetleniem przycisku zerowania i podpięciem zdarzenia upewnić się, że przeglądarka obsługuje localStorage.

Funkcja initDevice jest wywoływana przez $(document).ready(). Co to oznacza? Funkcja zostanie wywołana po zainicjalizowaniu przez jQuery struktury DOM strony.

Wykrywanie możliwości po stronie klienta może być tak mało skomplikowane jak w tym przypadku, ale w sytuacji gdy są wymagane szczegółowe informacje na temat wsparcia, może się okazać pomocna javascriptowa biblioteka Modernizr (*http://modernizr.com*).

W rozdziale 5. przeprowadzaliśmy detekcję możliwości urządzenia za pomocą WURFL, ale ona odbywała się po stronie serwera.

Moment... jeszcze nie skończyliśmy

Magnesiki, które nie zostały nigdzie przypięte, dają nam wskazówkę dotyczącą dalszych działań. Zapisujemy znalezione tartany i umożliwiamy wyczyszczenie ich wszystkich, ale jak na razie nic się nie zmienia w interfejsie użytkownika. Musimy napisać **kod aktualizujący wyświetlaną zawartość ekranu, tak by użytkownicy mogli zobaczyć, które tartany zostały już znalezione**. Do dzieła!

```
// Odczytaj wpis z localStorage
```

```
// Odśwież ekran,
// by wyświetlić
// znalezione tartany
```

WYSIL SZARE KOMÓRKI

Dodawanie przycisku zerowania i podpinanie zdarzeń jest realizowane tylko wtedy, gdy przeglądarka obsługuje mechanizm localStorage. Takie podejście stosowaliśmy już w rozdziale 4., gdzie ustalaliśmy granicę wsparcia. Jak sądzisz, czy to rozwiązanie jest dobre? Czy umiesz sobie wyobrazić sytuację, w której takie podejście jest niewłaściwe?

Używamy funkcji wyświetlającej znalezione tartany

Wygląda na to, że będzie nam potrzebna kolejna funkcja JavaScript. Jej zadaniem jest aktualizowanie zawartości strony, tak by znalezione tartany były wyraźnie oznaczone. Jak to zrobić? Wywołamy nową funkcję — `refreshTartans` — która zmienia wygląd listy tartanów i stron ze szczegółowymi informacjami.

Każda z zagnieżdżonych list (`ul.details`) z pliku *index.html* zawiera informacje o poszczególnych sponsorach i tartanach, które należy znaleźć. Do określenia klucza wpisu w `localStorage` odpowiadającego danemu tartanowi możemy użyć atrybutu `id` przypisanego do listy. Jeśli w `localStorage` z tym kluczem jest powiązana jakakolwiek wartość, oznacza to, że tartan został znaleziony, więc trzeba odświeżyć wyświetlaną zawartość strony.

Kod JavaScript gotowy do użycia

Zatem dla każdego elementu ul.details w dokumencie...

To jest kod jQuery, który iteruje w pętli each przez wszystkie elementy odpowiadające selektorowi ul.details (czyli elementy ul z przypisaną klasą „details").

app.js

```
$(document).ready(initDevice);
refreshTartans = function() {
  $('ul.details').each(function() {
    var myID = $(this).attr('id');
    var tartanKey = 'found-' + myID;
    var foundValue = localStorage.getItem(tartanKey);
    var isFound = Boolean (foundValue);
    $('#vendor-'+ myID).toggleClass('found', isFound);
    $('[data-url*="'+myID+'"]').toggleClass('found',isFound);
    $('#'+tartanKey).closest('li').toggle(!isFound);
  });
  $ ('ul').each(function() {
    if ($(this).data('listview')) {
      $(this).listview('refresh');
    }
  });
};
tartanFound = function (event) {
```

1. *Na podstawie atrybutu id elementu ul określa nazwę klucza wartości zapisanej w localStorage i dodaje do niej przedrostek „found-", czyli na przykład „found-douglas".*
2. *Sprawdza, czy dla określonego klucza istnieje wartość w localStorage.*
3. *Przełącza widoczność i klasy niektórych elementów, tak by widok odzwierciedlał to, czy dany tartan został znaleziony, czy nie.*
4. *Odświeża elementy listview z jQuery Mobile w dokumencie.*

Gdybyśmy tego nie zrobili, zawartość strony nie zostałaby zmieniona.

Pamiętaj, że framework jQuery Mobile jest zbudowany na bazie jQuery, więc mamy do dyspozycji wszystkie metody z tej biblioteki.

Metody `toggle` i `toggleClass` są częścią jQuery. Zaraz się im przyjrzymy.

Metody toggle i toggleClass

Metody `toggle` i `toggleClass` pochodzą z biblioteki jQuery. Pierwsza z nich przełącza widoczność elementu, a druga — przypisanie klasy CSS do elementu.

Dzięki zastosowaniu klasy „found" do wybranych elementów za pomocą metody toggleClass możemy oznaczać znalezione tartany za pomocą stylów...

...a przycisk „Mam go!", gdy już nie jest potrzebny, możemy ukryć za pomocą metody toggle.

(Tu nie ma przycisku!)

Spójrzmy jeszcze raz na kod

Jak pamiętasz, metoda `localStorage.getItem(tartanKey)` może zwrócić wartość zapisaną dla podanego klucza (w naszym przypadku jest to `'true'`) albo `null`. To, co otrzymujemy, zamieniamy na wartość boolowską (`true` albo `false`), tak byśmy mogli w prosty sposób użyć metod jQuery `toggle` i `toggleClass`.

Aby użyć tych metod, potrzebujemy prawdziwej wartości boolowskiej, a nie łańcucha 'true' i null.

Zmienna isFound przyjmuje wartość true, jeśli w localStorage istnieje jakakolwiek wartość dla danego tartanu, a wartość false w przeciwnym przypadku.

```
var foundValue = localStorage.getItem(tartanKey);
var isFound = Boolean (foundValue);
$('#vendor-'+ myID).toggleClass('found', isFound);
$('[data-url*="'+myID+'"]').toggleClass('found',isFound);
$('#'+tartanKey).closest('li').toggle(!isFound);
```

Metoda `toggleClass` *dodaje* wskazaną klasę CSS (`found`) do elementów odpowiadających podanemu selektorowi, jeśli zmienna `isFound` ma wartość `true` (jeżeli ma wartość `false`, klasa jest *usuwana*).

Analogicznie — metoda `toggle` wyświetli dany element `<li>`, jeśli zostanie jej przekazana boolowska wartość `true`. Tak naprawdę robimy tu coś sprytniejszego, ponieważ przekazujemy zanegowaną wartość zmiennej `isFound` (to oznacza wyrażenie `!isFound`). Dlaczego? No cóż, chcemy ukryć element `<li>` — zawierający przycisk *Mam go!* — jeśli tartan został znaleziony (czyli zmienna `isFound` ma wartość `true`). Przecież w takiej sytuacji przycisk nie jest już do niczego potrzebny.

Podsumowanie: jeśli tartan został odnaleziony, dodaj klasę found do dwóch elementów i ukryj przycisk „Mam go!". W przeciwnym przypadku usuń klasę i wyświetl przycisk.

Ćwiczenie

Do pliku *scripts/app.js* dodaj pełny kod z ćwiczenia z magnesikami oraz funkcję `refreshTartans`. Funkcja ta musi zostać wywołana w chwili inicjalizowania strony i za każdym razem, gdy zmienia się zawartość `localStorage`. Spróbuj ustalić trzy miejsca w kodzie, w których należy umieścić wywołanie funkcji `refreshTartans`.

Oto trzy miejsca w skrypcie *app.js*, w których trzeba wywołać funkcję `refreshTartans`.

Rozwiązanie ćwiczenia

Gdy inicjalizujemy...

```
var initDevice = function() {
   if (typeof(window.localStorage) == 'object') {
      $('.foundTartan').click(tartanFound);
      refreshTartans();
      addResetButton();
   }
};
```

...gdy tartan zostaje znaleziony...

```
var tartanFound = function(event) {
   var tartanKey = $(event.currentTarget).attr('id');
   localStorage.setItem(tartanKey, 'true');
   refreshTartans();
};
```

...i gdy znalezione tartany są zerowane.

```
var addResetButton = function() {
   var $resetButton = $('<a></a>').attr('data-role','button').
html('Rozpocznij!');
   $resetButton.click(function() {
      localStorage.clear();
      refreshTartans();
   });
   $resetButton.appendTo($('#booths'));
};
```

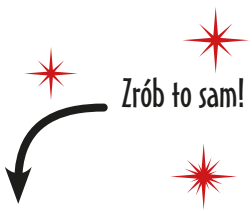
app.js

Jazda próbna

Uzupełnij skrypt *app.js* zgodnie z omówionymi zmianami, zapisz go, a następnie wyświetl plik *index.html* w przeglądarce desktopowej (aplikacja powinna działać poprawnie). Na kilku tartanach kliknij przycisk *Mam go!*. Powinieneś zauważyć zmianę wyglądu znalezionych tartanów (zazielenią się).

Jeśli masz jakieś problemy, użyj narzędzia Web Inspector (w przeglądarce Chrome bądź Safari) lub Konsoli błędów (w Firefoksie), aby sprawdzić, czy wystąpiły jakieś błędy w skrypcie JavaScript. Jeśli naprawdę utknąłeś, zajrzyj do pliku *rozdzial8/extras/scripts/app.localStorage.js*, gdzie znajdziesz ukończoną wersję skryptu.

Jeśli chcesz użyć tego pliku, musisz go zapisać zamiast dotychczasowego pliku app.js.



Ponownie spakuj do pliku ZIP zawartość całego katalogu *rozdzial8* i przebuduj go w PhoneGap Build. Następnie utworzoną aplikację zainstaluj w emulatorze i sprawdź, jak działa.

Nie istnieją głupie pytania

P: Które z najnowszych przeglądarek obsługują mechanizm localStorage?

O: Odpowiedź jest krótka: prawie wszystkie. Wyjątkiem jest Opera Mini. Jeśli z jakiegoś powodu używasz Internet Explorera 7, musisz pomyśleć nad zmianą przeglądarki.

P: Czy klucze muszą mieć jakieś konkretne nazwy?

O: Zarówno klucze, jak i wartości są łańcuchami tekstowymi. To jest jedyne ograniczenie — klucze mogą być dowolne.

P: Ile danych mogę zapisać?

O: Specyfikacja W3C dla mechanizmu localStorage jest czarująco niekonkretna. Oto cytat: „Zalecany jest umowny limit pięciu megabajtów na domenę. Wszelkie uwagi od programistów zajmujących się implementacją w przeglądarkach są mile widziane i zostaną uwzględnione w kolejnych wersjach specyfikacji".

Większość przeglądarek zapewnia obszar o pojemności od 2 do 5 MB. Niektóre przeglądarki, jak choćby Safari, proszą użytkownika o zgodę na powiększenie przestrzeni, w sytuacji gdy zajęta już została większość przydziału.

P: Czy inne witryny lub aplikacje mają dostęp do moich danych zapisanych w `localStorage`?

O: Nie. Istotny fragment specyfikacji jest poświęcony tematowi bezpieczeństwa i ustaleniu zasad mających zabezpieczyć domeny (czyli, upraszczając, inne witryny) przed nieuprawnionym dostępem do danych zapisanych w `localStorage`.

P: Była mowa o tym, że w `localStorage` można zapisywać tylko łańcuchy tekstowe. Wcześniej jednak wspomnieliście, że istnieje możliwość zapisania obrazów. Jak niby miałoby się to udać?

O: Tak, łańcuchy. Nikt i nic nas jednak nie powstrzyma przed zapisywaniem długich łańcuchów, prawda? Obrazy mogą zostać przekonwertowane za pomocą schematu kodowania Base64 na łańcuch tekstowy, dzięki czemu można je bezpośrednio przekazać jako wartość atrybutu `src` elementu `<img>` lub jako wartość `url()` dla obrazów tła w arkuszach CSS. Obsługa adresów URI dla danych jest w nowoczesnych przeglądarkach całkiem przyzwoita, no, może z jednym, jakże typowym wyjątkiem — Internet Explorerem. Więcej na ten temat znajdziesz w artykule *http://bit.ly/sWe7HS* autorstwa Nicholasa Zakasa.

P: Chwileczkę! Przeglądałem właśnie kod i zwróciłem uwagę na to, że na zagnieżdżonych stronach dodajemy przycisk powrotu. To nie ma sensu, ponieważ zdecydowana większość urządzeń z Androidem ma sprzętowy przycisk powrotu.

O: A niech to, zauważyłeś! Tak naprawdę ten przycisk nie tylko wygląda nieszczególnie w Androidzie, on w dodatku źle działa — zamyka aplikację wygenerowaną przez PhoneGap Build! To *nie za dobrze*. Niebawem wrócimy do tego problemu i go rozwiążemy.

P: Jeśli mechanizm localStorage jest dostępny bezpośrednio w przeglądarce, po co stosujemy PhoneGap Build? Czy nie moglibyśmy poprzestać na zwykłej aplikacji internetowej?

O: Cierpliwości! Właśnie o tym chcieliśmy pomówić. Najwyższy czas dołączyć do naszej aplikacji obsługę kamery.

Znalazłeś tartan? Udowodnij to!

- [x] Trzeba stworzyć i skonfigurować projekt PhoneGap Build, a następnie spakować wszystkie pliki projektu (HTML, CSS i obrazki), zbudować aplikację oraz zainstalować ją na urządzeniu z Androidem lub na emulatorze.

Zrobiliśmy to dzięki localStorage!

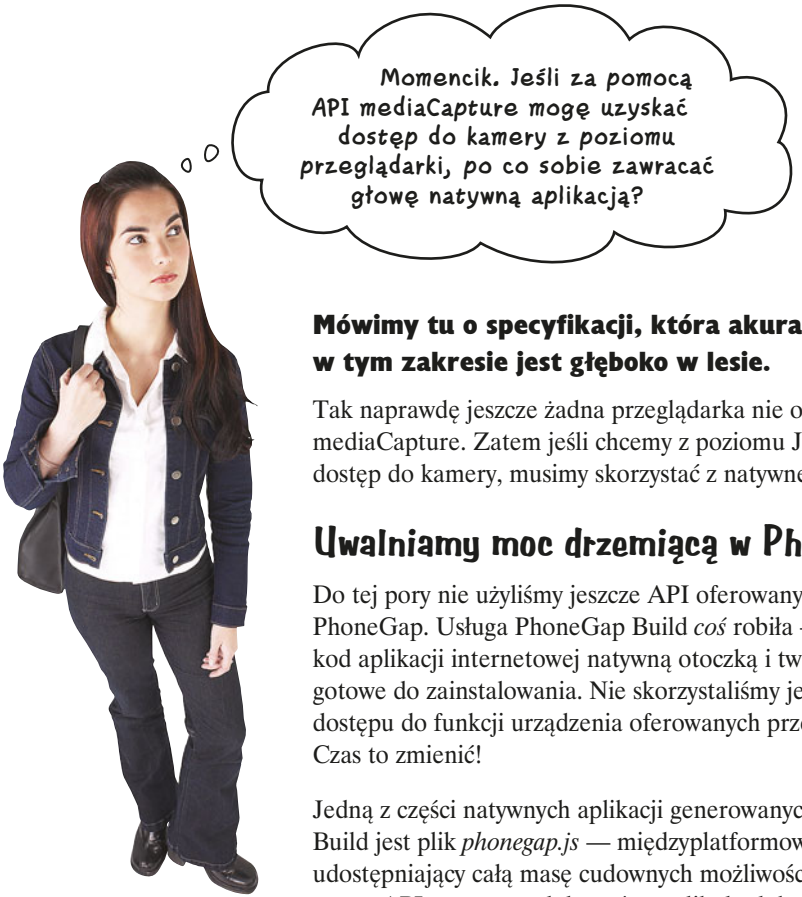
- [x] Musimy umożliwić użytkownikom zaznaczanie tartanów, które udało im się znaleźć.

- [ ] **Musimy dodać funkcjonalność zapisywania zdjęć znalezionych tartanów.**

No dobrze, a jak niby mamy to zrobić?

Z odsieczą przybywa API

Opisany w specyfikacji **W3C interfejs API mediaCapture** umożliwia nagrywanie dźwięku i wideo oraz robienie zdjęć. Jest to możliwe z poziomu JavaScriptu dzięki obiektom `navigator.device` oraz `navigator.device.capture`.



Mówimy tu o specyfikacji, która akurat w tym zakresie jest głęboko w lesie.

Tak naprawdę jeszcze żadna przeglądarka nie obsługuje API mediaCapture. Zatem jeśli chcemy z poziomu JavaScriptu uzyskać dostęp do kamery, musimy skorzystać z natywnego wsparcia.

Uwalniamy moc drzemiącą w PhoneGap

Do tej pory nie użyliśmy jeszcze API oferowanych przez PhoneGap. Usługa PhoneGap Build *coś* robiła — opakowywała kod aplikacji internetowej natywną otoczką i tworzyła paczki gotowe do zainstalowania. Nie skorzystaliśmy jednak z możliwości dostępu do funkcji urządzenia oferowanych przez PhoneGap. Czas to zmienić!

Jedną z części natywnych aplikacji generowanych przez PhoneGap Build jest plik *phonegap.js* — międzyplatformowy interfejs API udostępniający całą masę cudownych możliwości. Aby skorzystać z tego API, wystarczy dołączyć ten plik do dokumentu HTML.

Zaprzęgamy PhoneGap do robienia zdjęć

Javascriptowe API PhoneGap oferuje całkiem sporo możliwości, włącznie z kilkoma nowymi zdarzeniami. W naszym przypadku najistotniejszym z nich jest `deviceready`, które jest zgłaszane po przygotowaniu przez PhoneGap danego urządzenia. Możemy nasłuchiwać tego zdarzenia bezpośrednio w kodzie skryptu *app.js*.

Powiążemy to zdarzenie z funkcją zwrotną, w której będziemy mogli zająć się obsługą kamery.

Musimy zacząć od dołączenia pliku phonegap.js i nasłuchiwania zdarzenia deviceready. Potem możemy już działać.

Dzięki oczekiwaniu na zdarzenie deviceready możemy być pewni, że urządzenie jest gotowe do pracy.

Przeznaczenie tego argumentu jest dosyć skomplikowane. Z grubsza chodzi o to, że chcemy odebrać zdarzenie w fazie bąbelkowania, a nie w fazie przechwytywania.

```
document.addEventListener('deviceready', initPhoneGap, false);
```

Nie przejmuj się tym zanadto.

Chcemy, by w wyniku wystąpienia zdarzenia deviceready została wywołana funkcja initPhoneGap.

Hej, musimy napisać tę funkcję! Zrobimy to niebawem.

Ćwiczenie

Zajmij się uzupełnieniem aplikacji o elementy wymagane przez PhoneGap.

1. **Zaktualizuj plik index.html w celu dołączenia skryptu phonegap.js.**
   Zapewne zauważyłeś, że w katalogu *rozdzial8* nie ma pliku *phonegap.js*. PhoneGap Build sam dodaje odpowiednią wersję tego skryptu do każdej generowanej aplikacji. Dodaj znacznik `<script>`, dzięki któremu wszystko będzie prawidłowo działało po zbudowaniu aplikacji.

2. **W pliku app.js nad funkcją initDevice utwórz pustą funkcję initPhoneGap.**
   Za chwilę to uzupełnimy.

3. **Na samym końcu funkcji initDevice umieść kod definiujący funkcję nasłuchującą zdarzenia deviceready (kod znajdziesz na górze tej strony).**
   Dzięki temu funkcja `initPhoneGap` zostanie wywołana po zainicjalizowaniu urządzenia przez PhoneGap.

Poniżej znajdziesz fragmenty zaktualizowanego skryptu *app.js* oraz pliku *index.html*, do którego dołączyliśmy *phonegap.js*. Będziemy musieli załatwić jeszcze kilka drobnych spraw, by w końcu móc cieszyć się obsługą kamery.

Rozwiązanie ćwiczenia

```
(function() {
    $(document).bind("mobileinit", function() {
        $.extend($.mobile, { defaultPageTransition: 'none' });
        $.mobile.page.prototype.options.addBackBtn = true;
    });
    var initPhoneGap = function() {
    };
    var initDevice = function() {
        if (typeof(window.localStorage) == 'object') {
            $('.foundTartan').click(tartanFound);
            refreshTartans();
            addResetButton();
        }
        document.addEventListener('deviceready', initPhoneGap, false);
    };
    $(document).ready(initDevice);
```

app.js

```
<link rel="stylesheet" href="style/jquerymobile1_1_0_min.css" />
<link rel="stylesheet" href="style/app.css" />
<script src="phonegap.js"></script>
<script src="scripts/jquery1_7_min.js"></script>
<script src="scripts/app.js"></script>
<script src="scripts/jquerymobile1_1_0_min.js"></script>
```

index.html

Może Cię to trochę zdziwić, ponieważ nie mamy pliku phonegap.js, jednak po zbudowaniu aplikacji plik się pojawi. To będzie po prostu... działać.

Integracja z PhoneGap jest już prawie gotowa

Znowu jesteśmy w tym samym miejscu skryptu *app.js*, bo mamy tu jeszcze coś do zrobienia. Zacznij od wypełnienia pustej funkcji `initPhoneGap`.

Umieściliśmy tę zmienną w tym miejscu, by była dostępna w pozostałych fragmentach skryptu.

Musimy sprawdzić, czy urządzenie obsługuje przechwytywanie obrazu z kamery. Na początku zakładamy, że nie (zmiennej przypisujemy false).

```
(function() {
    var imageCaptureSupported = false;
    $(document).bind("mobileinit", function() {
        $.extend($.mobile, { defaultPageTransition: 'none' });
        $.mobile.page.prototype.options.addBackBtn = true;
    });

    var initPhoneGap = function() {
        if (!navigator.device || !navigator.device.capture) { return; }
        imageCaptureSupported = true;
        if (device.platform && device.platform == 'Android') {
            $('body').addClass('android');
        }
    };
    var initDevice = function() {
```

app.js

W porządku, te testy przeszły pomyślnie, funkcjonalność mediaCapture jest obsługiwana.

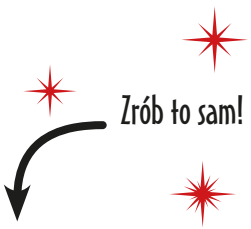
Jeśli nie są dostępne niezbędne funkcjonalności, nie możemy kontynuować.

Świetnie! We właściwości device.platform znajduje się nazwa platformy, dzięki czemu możemy ukryć przycisk powrotu dla Androida.

W arkuszu stylów style/app.css znajduje się reguła ukrywająca przycisk powrotu w elemencie <body> z przypisaną klasą android.

W funkcji `initPhoneGap` (która jest funkcją zwrotną dla zdarzenia `deviceready`) upewniamy się, czy mamy dostęp do `navigator.device.capture`, a jeśli tak, ustawiamy zmienną `imageCaptureSupported` na `true`.

Następnie, w przypadku Androida, ukrywamy niepotrzebny przycisk powrotu (który w dodatku źle działa). Większość urządzeń z Androidem ma sprzętowy przycisk powrotu, więc powielanie go prowadzi do zamieszania i może denerwować użytkowników. Przycisk ten jest jednak potrzebny na urządzeniach iOS, ponieważ bez niego użytkownikom nie udałoby się powrócić do poprzedniego ekranu. Sprawę załatwia ten krótki fragment kodu.



W porządku, wszystko gotowe! Zaktualizuj skrypt *app.js* i... światła, kamera, akcja!

Teraz jesteśmy gotowi na zgłębienie API mediaCapture

Interfejs API mediaCapture z PhoneGap, podobnie jak inne dostarczane API urządzeń, imituje oryginalną specyfikację W3C dla mediaCapture. We wspomnianej specyfikacji można znaleźć całkiem sporo informacji, ale nam przyda się w zasadzie tylko jedna metoda — przywitaj się serdecznie z `CaptureImage`.

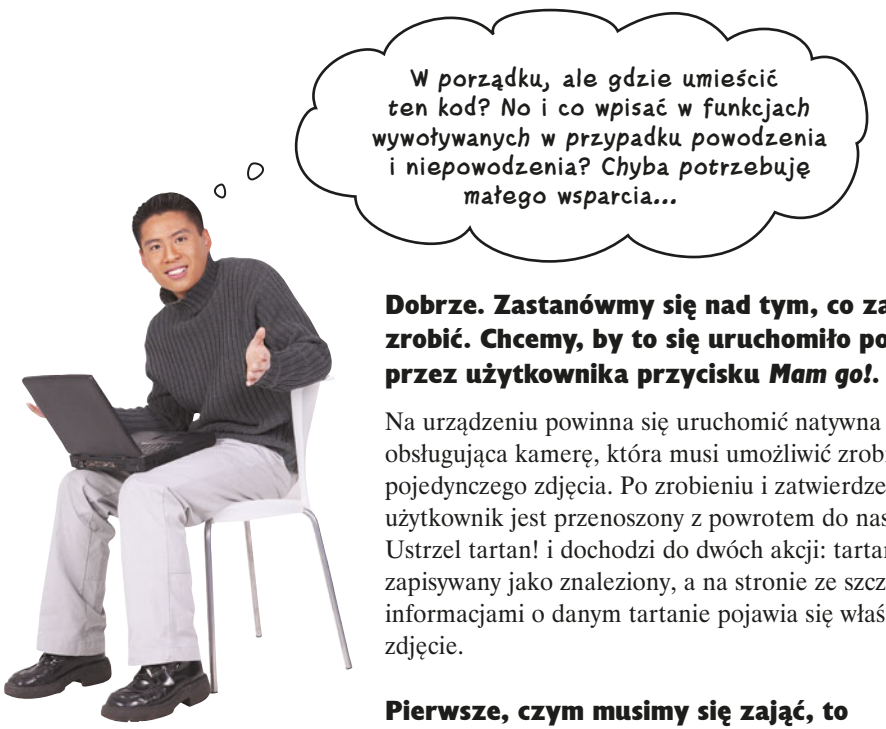
Funkcja, która zostanie wywołana w przypadku powodzenia.

Opcje. Możemy na przykład ustalić, by użytkownik mógł zrobić tylko jedno zdjęcie (w tym celu ustawimy odpowiednią opcję).

```
navigator.device.capture.captureImage(successCallback, errorCallback, options);
```

Metoda captureImage uruchamia kamerę urządzenia i robi zdjęcia.

Funkcja, która zostanie wywołana w przypadku niepowodzenia.



Dobrze. Zastanówmy się nad tym, co zamierzamy zrobić. Chcemy, by to się uruchomiło po kliknięciu przez użytkownika przycisku *Mam go!*.

Na urządzeniu powinna się uruchomić natywna aplikacja obsługująca kamerę, która musi umożliwić zrobienie pojedynczego zdjęcia. Po zrobieniu i zatwierdzeniu zdjęcia użytkownik jest przenoszony z powrotem do naszej aplikacji Ustrzel tartan! i dochodzi do dwóch akcji: tartan jest zapisywany jako znaleziony, a na stronie ze szczegółowymi informacjami o danym tartanie pojawia się właśnie zrobione zdjęcie.

Pierwsze, czym musimy się zająć, to zaktualizowanie funkcji `tartanFound`, która obsługuje zdarzenie kliknięcia przycisku *Mam go!*, tak by była w niej wywoływana metoda `captureImage` pobierająca zdjęcie wykonane przez użytkownika. Zatem do roboty!

W jaki sposób obsłużymy akcję zakończoną powodzeniem?

Kiedy użytkownik kliknie przycisk *Mam go!* ma zostać wywołana metoda `captureImage` (zakładając, że ma urządzenie z kamerą, które obsługuje API mediaCapture dostarczone przez PhoneGap Build).

Z naszego pierwszego spotkania z tą metodą powinieneś pamiętać, że pierwszy argument to funkcja, która zostanie wywołana, jeśli wszystko się uda. W ogólnym zarysie ta funkcja zwrotna ma wyglądać mniej więcej tak:

Hm... co to za mediaFiles?

Wyrażenie mediaFiles[0].fullPath daje nam ścieżkę do pierwszego obrazu z tablicy mediaFiles.

```
function itWorked(mediaFiles) {
   localStorage.setItem(tartanKey, mediaFiles[0].fullPath);
}
```

Ponieważ ustawimy opcję wymuszającą robienie tylko jednego zdjęcia (za chwilę powiemy na ten temat więcej), w tablicy i tak powinien się znaleźć tylko jeden element.

Jeśli działanie metody `captureImage` zakończy się powodzeniem, zostanie wywołana podana funkcja zwrotna z przekazaną listą (tablicą JavaScript) obiektów `MediaFile`. Obiekty te nie zawierają plików, a jedynie zestaw metadanych. Spokojnie, nie rozpaczaj. Znajdziemy tam to, czego potrzebujemy — **ścieżkę do pliku obrazu w systemie plików urządzenia**. Dzięki temu możemy ją zapisać w `localStorage` zamiast stosowanej do tej pory wartości `true`.

A co, jeśli coś pójdzie nie tak?

Musimy się przygotować na sytuację, w której operacje związane z kamerą zakończą się niepowodzeniem (użytkownik anuluje wykonywanie zdjęcia albo coś się popsuje). Będzie więc nam potrzebna kolejna funkcja zwrotna. Ponieważ niemożność wykonania zdjęcia nie jest powodem do wszczynania czerwonego alarmu, funkcja ta będzie stosunkowo prosta: wypiszemy w niej do konsoli informację o błędzie, tak by jakaś ciekawska dusza mogła ją odczytać podczas debugowania. W związku z tym funkcja przyjmie taką postać:

To jest tylko przykład funkcji zwrotnej dla niepowodzenia operacji.

```
function ohSad(error) {
   console.log(error);
}
```

W praktyce zawsze wygląda to trochę inaczej

Jesteś gotowy na spotkanie z rozszerzoną wersją funkcji `tartanFound`? Pamiętaj, że ta funkcja zostaje wywołana po kliknięciu przez użytkownika przycisku *Mam go!* dla wybranego tartanu. Spójrz poniżej — oto ona. Przyznajemy, że ostateczna wersja kodu nieco odbiega od tego, co pokazaliśmy przed momentem.

Zmienną imageCaptureSupported ustawiliśmy w funkcji initPhoneGap, pamiętasz?

Jeśli mediaCapture jest obsługiwany i dostępny, przechodzimy do robienia zdjęcia.

W przypadku urządzeń, które nie obsługują mediaCapture, oznaczamy dany tartan jako znaleziony za pomocą wartości true.

W tym miejscu ustawiamy opcję ograniczającą liczbę wykonywanych zdjęć do jednego.

Funkcja zwrotna dla niepowodzenia.

```
var tartanFound = function(event) {
    var tartanKey = $(event.currentTarget).attr('id');
    if(imageCaptureSupported) {
        navigator.device.capture.captureImage(function(mediaFiles) {
            localStorage.setItem(tartanKey, mediaFiles[0].fullPath);
            refreshTartans();
        }, captureError, {limit:1});
    }
    else {
        localStorage.setItem(tartanKey, 'true')
        refreshTartans();
    }
};
var captureError = function(error) { console.log(error); }
addResetButton = function() {
```

app.js

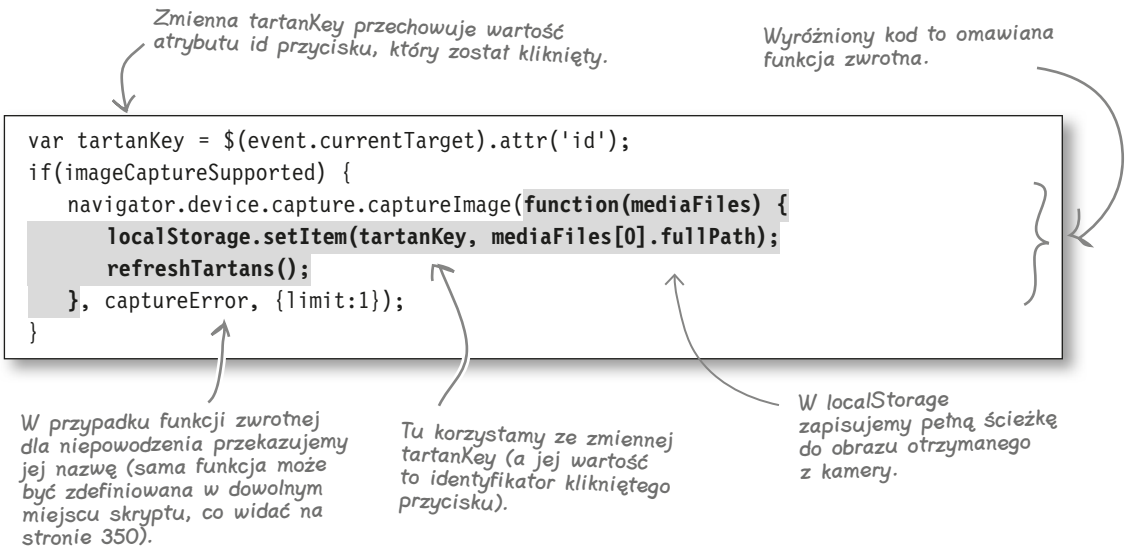
Co tu się dzieje? To wygląda zupełnie inaczej niż metoda captureImage, którą opisaliście wcześniej...

Ponieważ potrzebna nam jest aktualna wartość `tartanKey`, funkcja zwrotna dla powodzenia jest funkcją anonimową.

Odrobina anonimowości

Zamiast przekazywać metodzie `captureImage` nazwę funkcji zwrotnej, możemy całą tę funkcję zdefiniować bezpośrednio w miejscu przekazania.

A czemu to ma służyć? Zwróć uwagę, jak przypisujemy atrybut `id` elementu, który został kliknięty (`event.currentTarget`, czyli przycisku *Mam go!*), do zmiennej `tartanKey`. Dzięki zdefiniowaniu funkcji zwrotnej jako ***funkcji anonimowej*** mamy dostęp do odpowiedniej wartości zmiennej `tartanKey` w zależności od kliknięty przycisku. Gdybyśmy użyli funkcji zdefiniowanej poza `tartanFound`, nie mielibyśmy dostępu do tej wartości w odpowiednim kontekście.



WYSIL SZARE KOMÓRKI

W kodzie aktualizujemy `localStorage` z poziomu funkcji zwrotnej dla powodzenia działania metody `imageCapture`. Jeśli nie uda się zrobić zdjęcia, tartan nie zostanie oznaczony jako znaleziony.

Jak sądzisz, czy to dobrze, że tak się dzieje? A może istnieje lepszy sposób na obsługę takiej sytuacji?

No dobrze, widzę, że w localStorage zapisujemy ścieżkę do obrazu, ale jak zamierzamy z niej skorzystać?

Dobrze, że spytałeś. Musimy zaktualizować funkcję `refreshTartans`.

Jeśli użytkownik zrobił zdjęcie danego tartanu, chcemy je wyświetlić na stronie z opisem stoiska sponsora.

Najistotniejsza informacja to ta, że zapisanej ścieżki możesz użyć jako wartości atrybutu `src` elementu `img`. Nie było to takie skomplikowane...

Kod JavaScript gotowy do użycia

Jeśli w localStorage mamy zapisaną ścieżkę obrazu, za pomocą jQuery tworzymy element <img>, ustawiamy jego atrybut src na tę ścieżkę, a następnie umieszczamy nowy element na stronie.

```
var refreshTartans = function() {
   $('ul.details').each(function() {
      var myID = $(this).attr('id');
      var tartanKey = 'found-' + myID;
      var foundValue = localStorage.getItem(tartanKey);
      var isFound = Boolean (foundValue);
      $('#vendor-'+ myID).toggleClass('found', isFound);
      $('[data-url*="'+ myID +'"]').toggleClass('found', isFound);
      $('#'+tartanKey).closest('li').toggle(!isFound);
      var hasPhoto = (isFound && foundValue != 'true');
      if (hasPhoto) {
         if (!$(this).find('.tartanImage').length) {
            var $tartanHolder = $('<p></p>').append($('<img>').attr({
               'src' : foundValue,
               'class' : 'tartanImage'
            }));
            $(this).append('<li data-role="list-divider">Moje zdjęcie tartanu!</li>');
            $('<li></li>').append($tartanHolder).appendTo($(this));
         }
      }
   });
   $('ul').each(function() {
      if ($(this).data('listview')) { $(this).listview('refresh'); }
   });
};
```

Jeśli dla tego tartanu jest już zapisana wartość i NIE jest ona łańcuchem 'true', oznacza to, że musi to być zdjęcie.

Atrybutowi src elementu img przypisujemy zapisaną ścieżkę.

Jazda próbna

To jak, jesteś gotowy? Musieliśmy trochę popisać w JavaScripcie, ale nie było chyba tak źle. Wynikowy plik *app.js* ma około 75 linii kodu, więc — jak na kod całej aplikacji — jest wyjątkowo zwięzły.

Już ostatnia sprawa!

Mała wskazówka: lepiej dodaj poniższe dwa wiersze do pliku *config.xml*. Ustawienia orientacji zabezpieczają przed przełączeniem się urządzenia (a zwłaszcza emulatora) w tryb poziomy po uruchomieniu aplikacji do robienia zdjęć. W drugim wierszu informujemy PhoneGap, że chcemy użyć API kamery (zostanie to automatycznie zamienione na odpowiednie **ustawienie pozwoleń** w Androidzie).

```
<preference name="orientation" value="portrait" />
<feature name="http://api.phonegap.com/1.0/camera" />
```

config.xml

W porządku. Weź głęboki oddech i postaraj się uspokoić po nawałnicy kodu z ostatnich kilku stron. Sprawdź, czy wprowadziłeś wszystkie omówione zmiany w plikach *app.js* i *config.xml*, a następnie spakuj ponownie katalog *rozdzial8*, przejdź do PhoneGap Build i prześlij zaktualizowany kod.

Po zakończeniu procesu budowania aplikacji pobierz plik APK i zainstaluj go w emulatorze lub na urządzeniu.

Nie zapomnij najpierw odinstalować poprzedniej wersji!

Jeśli zainstalujesz aplikację na emulatorze, aplikacja kamery może się zachowywać dziwacznie. Nie przejmuj się tym zbytnio.

Zauważyliśmy, że kamery w emulatorze dla wersji Androida wcześniejszych niż 2.3 zachowują się niestabilnie — blokują się lub zawieszają. W związku z tym najlepiej, żebyś korzystał z nowszej wersji Androida w AVD (ang. *Android Virtual Device*).

Jednak nawet w nowszych wersjach emulatora działanie kamery jest dalekie od ideału. Niektóre emulatory wyświetlają tylko biały ekran, a na innych może się pojawić krótka animacja, ale mimo to generują *jakieś* obrazy. Przeważnie...

Utknąłeś?

Pełną wersję kodu skryptu *app.js* znajdziesz w pliku *extras/scripts/app.final.js*. Z kolei ostateczna wersja paczki APK z aplikacją dla Androida jest dostępna pod adresem *ftp://ftp.helion.pl/online/inne/hfmw/r08/TartanHunt.apk*.

Jeśli masz znajomego, który ma telefon z Androidem, pora się tadnie uśmiechnąć. Kamera działa o niebo lepiej na prawdziwych urządzeniach.

Daliśmy radę!

- Trzeba stworzyć i skonfigurować projekt PhoneGap Build, a następnie spakować wszystkie pliki projektu (HTML, CSS i obrazki), zbudować aplikację oraz zainstalować ją na urządzeniu z Androidem lub na emulatorze.

- Musimy umożliwić użytkownikom zaznaczanie tartanów, które udało im się znaleźć.

- Musimy dodać funkcjonalność zapisywania zdjęć znalezionych tartanów.

CELNE SPOSTRZEŻENIA

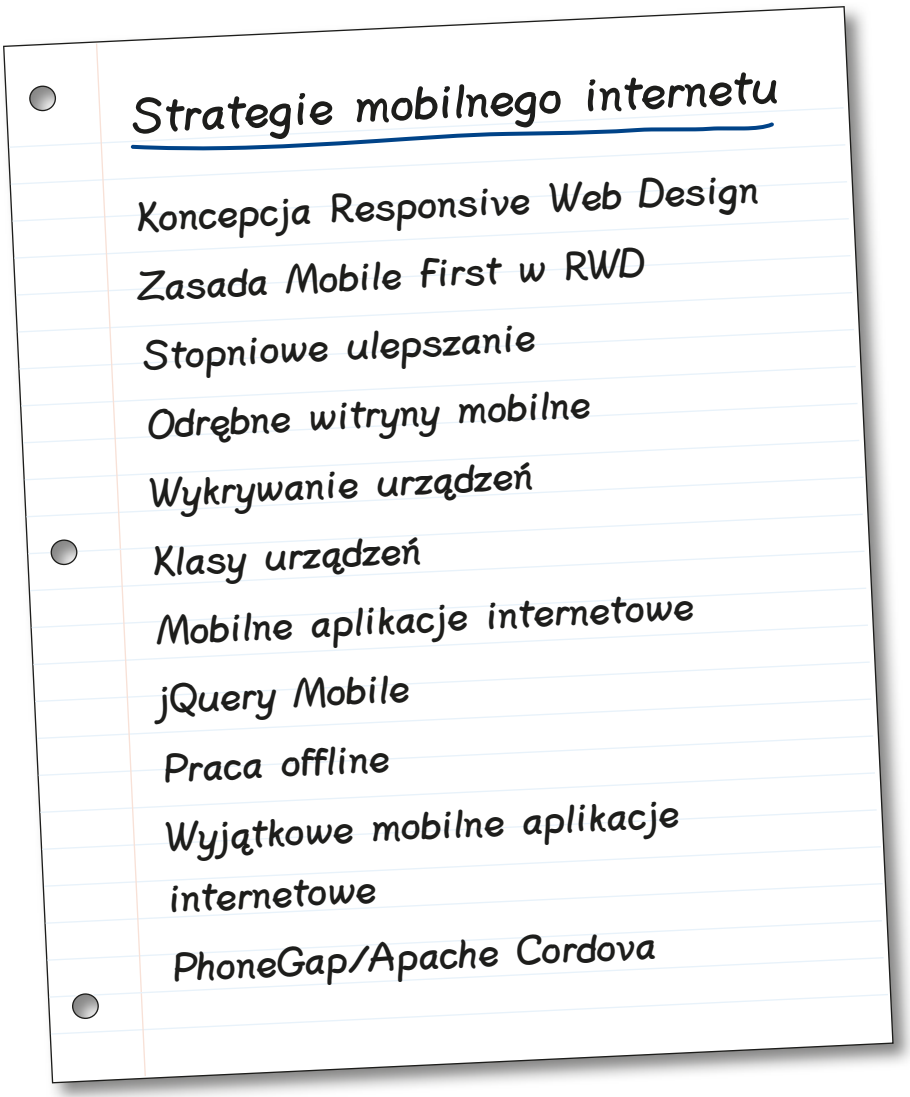
- Mobilne witryny przebyły długą drogę. Mimo to nadal jest **kilka rzeczy, do których nie mamy dostępu z poziomu przeglądarki**.
- **Hybrydowe aplikacje** pozwalają na **utworzenie natywnych aplikacji z kodu zgodnego ze standardami HTML5, CSS i JavaScript**. Dzięki temu możemy znacznie zmniejszyć kod wytwarzania aplikacji i ich złożoność, ponieważ ten sam kod jest współdzielony przez wiele platform.
- **PhoneGap to platforma HTML5 służąca do budowania natywnych aplikacji**. Została utworzona przez agencję Nitobi, która jesienią 2011 roku została przejęta przez firmę Adobe.
- **PhoneGap Build** to serwis internetowy umożliwiający proste tworzenie projektów PhoneGap.
- Struktura projektów PhoneGap Build jest zgodna ze specyfikacją **W3C dla widżetów internetowych**. Minimalne wymagania to plik startowy (*index.html*) oraz plik konfiguracyjny (*config.xml*).
- **Interfejs API localStorage** pozwala na **składowanie danych w postaci par klucz – wartość po stronie klienta**. W naszej aplikacji korzystamy z tego mechanizmu do przechowywania informacji o znalezionych tartanach.
- Plik *phonegap.js*, dający dostęp do **javascriptowych interfejsów API PhoneGap**, jest automatycznie dodawany do projektu PhoneGap Build. W omówionym projekcie opisaliśmy sposób nasłuchiwania **zdarzenia `deviceready`**, które informuje, że pożądana funkcjonalność urządzenia jest gotowa do użycia.
- Aby w aplikacji skorzystać z możliwości oferowanych przez kamerę, używamy **interfejsu API mediaCapture**, który odpowiada interfejsowi opisanemu w specyfikacji W3C.
- W celu uruchomienia natywnej aplikacji obsługującej kamerę i zrobienia zdjęcia użyliśmy metody **`navigator.device.capture.captureImage`**.
- **Utworzyliśmy międzyplatformową natywną aplikację z kodu złożonego z 75 linii JavaScriptu i pojedynczego pliku HTML!**

Odnajdywanie (jakiegoś) sensu w chaosie

Responsive Web Design. Wykrywanie urządzeń. Mobilne aplikacje internetowe. PhoneGap. Chwila... czego właściwie powinienem użyć? Obecnie istnieje wiele metod tworzenia aplikacji za pomocą mobilnych technologii. Bardzo często w projektach **łączy się wiele technik**. Nie ma jednego, najlepszego rozwiązania, ale nie przejmuj się, ponieważ kluczem do sukcesu jest nadążanie za rozwojem technologii. **Nie bój się wyzwań**. Wystarczy przyswoić **podejście „future friendly"** i dać się ponieść fali, będąc przeświadczonym o swojej elastyczności i gotowości na wszystko, co przyniesie przyszłość.

I co teraz?

Omówiliśmy sporo tematów — od witryn po aplikacje, od koncepcji Responsive Web Design do wykrywania urządzeń. Jest cała masa różnych podejść do tworzenia na potrzeby mobilnych technologii. Jak wybrać to najlepsze rozwiązanie?

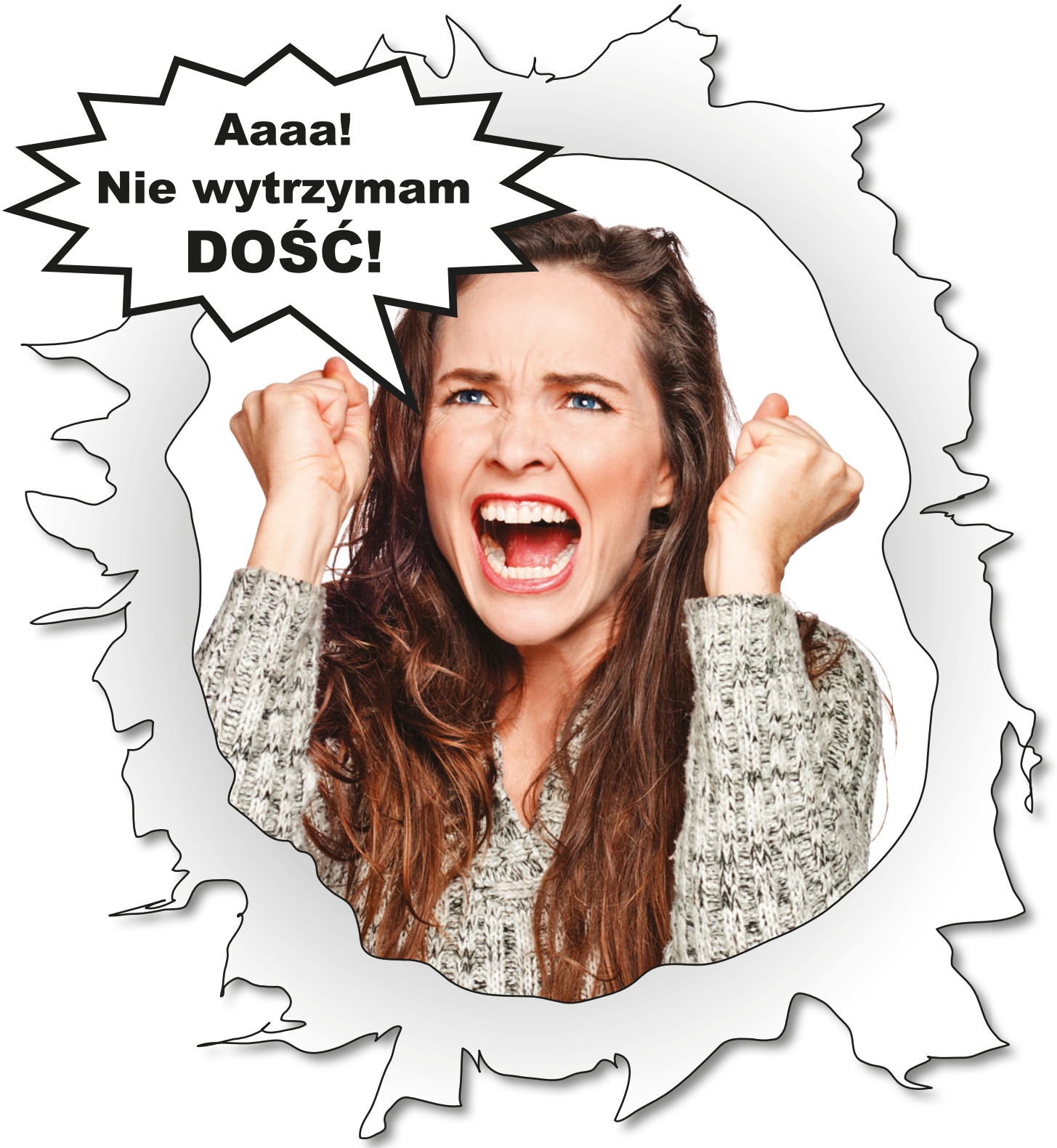


Chcielibyśmy skorzystać z koncepcji Responsive Web Design, ale nie możemy użyć serwera dla wszystkich obrazków, a żadne z obecnie dostępnych rozwiązań w JavaScripcie nie odpowiada naszym wymaganiom. Czy to sytuacja bez wyjścia?

Każdego dnia pojawiają się nowe urządzenia. Wygląda na to, że technika wykrywania urządzeń to nigdy niekończący się wyścig zbrojeń.

A co z tabletami? Czy musimy czekać na książkę „Tablety i internet. Rusz głową!”, by dowiedzieć się, jak tworzyć witryny internetowe na iPada?

Potrzebujemy witryny i aplikacji. Testowanie każdej z osobna będzie kosztowało krocie.


Aaaa!
Nie wytrzymam
DOŚĆ!

To nie takie proste

Nie ma drogi na skróty. Wytwarzanie oprogramowania internetowego staje się wraz z każdym nowym urządzeniem coraz bardziej skomplikowane. Nawet gorzej, to się staje coraz trudniejsze, jeszcze zanim było proste.

Szybkość zmian jest niebywała. Tu chodzi nie tylko o tempo pojawiania się nowych urządzeń, ale również związaną z tym ewolucję technologii i oprogramowania.

Zmienia się sposób interakcji z komputerami — w użyciu są interfejsy dotykowe, stosowane są obrazy z kamer oraz rozpoznawanie dźwięku. To wszystko każe nam inaczej myśleć o komunikacji człowiek – komputer.

Co z tym zrobić? Jak zapewnić użytkownikom fascynujące wrażenia z korzystania z naszych aplikacji, skoro jest tyle niejasności i niepewności?

Czas rozwiać zbiorowe złudzenie co do sprawowania kontroli

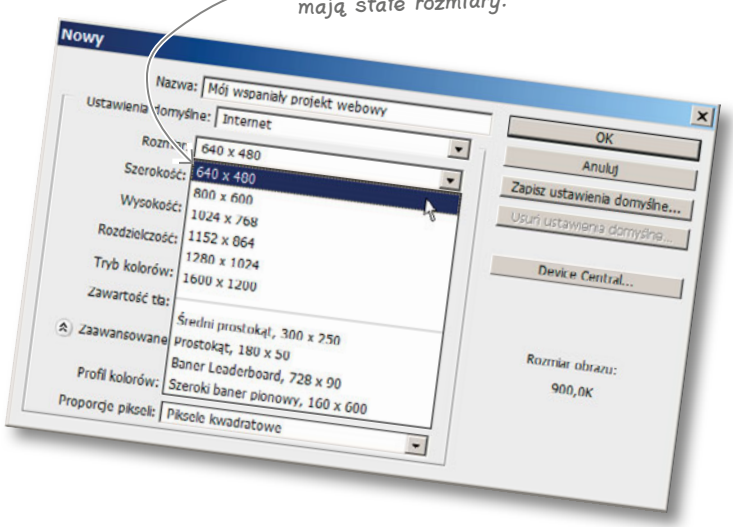
Przez dłuższy czas oszukiwaliśmy się, sądząc, że mamy kontrolę nad siecią.

Te złudzenia są widoczne w stosowanych narzędziach i procesach. Bardzo często projekt rozpoczyna się w Photoshopie. Pierwsze pytanie, jakie się pojawia, dotyczy rozmiaru płótna, na którym zamierzamy pracować.

Ale przecież nie ma jednego, uniwersalnego rozmiaru. Nawet w przypadku komputerów stacjonarnych.

Im szybciej zdamy sobie sprawę z wymogów otaczającej nas rzeczywistości internetowej, tym łatwiej będzie nam się do nich dostosować oraz zacząć tworzyć witryny i aplikacje internetowe, które dostosowują się do środowiska i użytkowników.

Podczas tworzenia nowego projektu w Photoshopie pojawia się okno, którego zawartość sugeruje, że projekty webowe mają stałe rozmiary.



Musimy zmienić sposób rozumowania — akceptujemy różnorodność i staramy się dostosować do wszystkiego, co przyniesie nam przyszłość.

Manifest „future friendly"

Nie tylko Ty czujesz się tym wszystkim przytłoczony. Pewnego pięknego dnia zebrała się grupa twórców rozwiązań mobilnych, by podczas szalonej burzy mózgów określić drogi wiodące do możliwości sprostania wymogom stale zmieniającego się internetowego krajobrazu. W wyniku tego spotkania powstał manifest podejścia „future friendly".

Oryginał manifestu możesz znaleźć na stronie http://futurefriend.ly.

FUTURE ☆ FRIENDLY

W dzisiejszych czasach oszałamiającego, ale i przytłaczającego rozwoju cyfrowych urządzeń istnieją prawdy, które uznajemy za niezmienne:

Zamęt będzie się nieodwołalnie powiększał. Ilość i rozmaitość urządzeń – w tym takich, których jeszcze sobie nawet nie wyobrażamy – eksploduje, podobnie jak liczba i różnorodność ludzi ich używających. **Dzisiejsze standardy, procesy i infrastruktura tego nie wytrzymają.** Mający obecnie miejsce szturm urządzeń zbliża je do punktu granicznego. Nie uniosą ciężaru tego, co nastąpi. **Na początku będą dominować opatentowane rozwiązania.** Koniecznością jest, by innowacje poprzedzały standaryzację. Twórcy technologii będą utajniali te rozwiązania, zanim zdadzą sobie sprawę (po raz kolejny), że ustandaryzowana platforma jest warunkiem zachowania jako takiej normalności. **Proces standaryzacji będzie potwornie powolny.** Będziemy walczyć z własnościowymi standardami. W tym czasie sieć wpadnie w jeszcze większym stopniu w ręce opatentowanych rozwiązań.

★ A NEW HOPE ★

Jest jednak nadzieja. Mimo że nie jesteśmy w stanie przewidzieć, co przyniesie przyszłość, możemy:

1. Uznać i przyjąć nieprzewidywalne.
2. Myśleć i zachowywać się zgodnie z zasadą „future friendly”.
3. Pomóc innym zrobić to samo.

To my tworzymy przyszłość – przyjaźnie.

★ UNDERSIGNUMS ★

LUKE WROBLEWSKI	SCOTT JENSON
BRAD FROST	JEREMY KEITH
LYZA D. GARDNER	SCOTT JEHL
STEPHANIE RIEGER	JASON GRIGSBY
BRYAN RIEGER	JOSH CLARK

★ Home ★ Thinking ★ Resources ★

We're Creative Commons friendly. Talk to us.

Jeśli nie możesz się uodpornić na przyszłość, zaprzyjaźnij się z nią

Ponieważ żyjemy w takiej niepewności co do przyszłego kształtu internetu, najważniejsze jest to, byśmy tworzyli witryny i aplikacje tak elastyczne, jak to tylko możliwe. To oznacza, że pracując nad projektami tu i teraz, musimy myśleć o przyszłości.

Istne **namnażanie urządzeń** bynajmniej w tym nie pomaga. Ponieważ mamy do czynienia z taką złożonością, kusi — bardziej niż kiedykolwiek — przymknięcie oczu i budowanie aplikacji, które działają teraz, na obecnie używanych urządzeniach, i pozostawienie na później problemów związanych z uniwersalnym dostępem oraz wsparciem dla przyszłych urządzeń.

Praca nad internetowymi projektami zawsze wymagała **kompromisów**. Cała sztuka polega na tym, by znaleźć takie rozwiązanie, które nie blokuje treści i danych w sposób uniemożliwiający ich późniejsze wykorzystanie i przekształcenie.

Nie ma rozwiązań idealnych

Weźmy na przykład obrazy — żadne rozwiązanie wynikające z podejścia Mobile First w koncepcji Responsive Web Design nie jest idealne. Powód, dla którego wybraliśmy Sencha.io Src (pomijając to, że bazuje na wykrywaniu urządzeń), jest prosty — to rozwiązanie nie wymaga ingerencji w dokument HTML. Dzięki temu możemy teraz bez problemu zamienić Sencha.io Src na jakiekolwiek inne rozwiązanie.

Podobnie sprawa wygląda, w przypadku gdy musisz zrobić coś, co jeszcze nie jest obsługiwane przez przeglądarki. Lepiej, żebyś wybrał technologie takie jak PhoneGap, które są opracowane w zgodzie ze standardami, niż decydował się na własne rozwiązania różnych firm, ponieważ przeważnie ich życie nie jest zbyt długie.

Podstawą sukcesu w przyszłości są semantyczny kod dokumentu oraz stopniowe ulepszanie. Jeśli zwrócisz uwagę na te zasady, Twoje życie będzie prostsze.

Jazda próbna

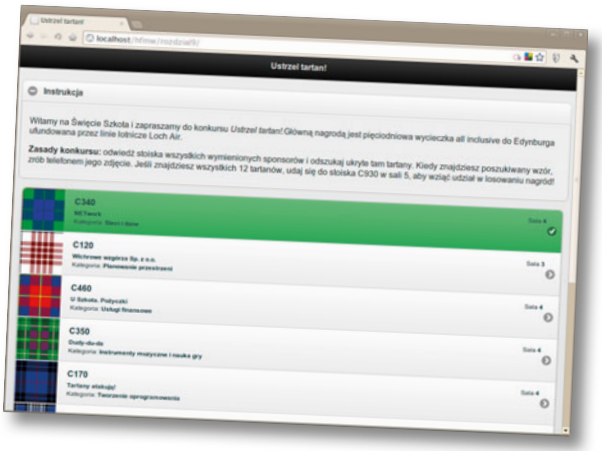
Myślenie zgodne z podejściem „future friendly” wkroczyło do aplikacji Ustrzel tartan! z rozdziału 8., mimo że efektem końcowym była natywna aplikacja. Pobierz kod aplikacji i otwórz plik *index.html* w przeglądarce desktopowej.

Czy aplikacja działa? Czy czegoś brakuje?

Jeśli potrzebujesz kopię ostatecznej wersji kodu, pobierz ją z *ftp://ftp.helion.pl/online/inne/hfmw/r09/rozdzial9.zip*.

Dziś aplikacja, jutro witryna

Mimo że projekt Ustrzel tartan! tworzyliśmy z myślą o aplikacji natywnej, można bez problemu wyświetlić go w przeglądarce. Odnalezione tartany są zapisywane i podświetlane na zielono. Brakuje jedynie kamery.



Dzięki temu, że podczas projektowania tej aplikacji postępowaliśmy zgodnie z zasadą „future friendly” —korzystaliśmy z proponowanych standardów W3C — są duże szanse, że ten fragment aplikacji, który nie działa w tej chwili (chodzi o kamerę), zacznie działać w przyszłości.

Usuwamy odwołania do PhoneGap

Są dwie sprawy, które powinniśmy załatwić, by aplikacja działała lepiej w przeglądarce.

1. **Usuwamy odwołanie do biblioteki phonegap.js.**
Gdy przeglądarki osiągną stan nirwany i zostanie zasypana przepaść między tym, co natywne, a tym, co internetowe, nie będziemy już potrzebować biblioteki *phonegap.js*. Opuszczenie odwołania do pliku *phonegap.js* nie wywoła żadnych niekorzystnych efektów. Aby przeglądarka nie zgłaszała błędu o kodzie 404 (brak pliku), musimy usunąć element `<script>` dołączający bibliotekę.

Usuń ten wiersz z pliku index.html.

```
<script src="phonegap.js"></script>
```

Przedstawiona tu taktyka jest podstawą myślenia zgodnego z podejściem „future friendly”.

2. **Zmieniamy funkcje inicjalizujące.**
PhoneGap, jQuery i jQuery Mobile ścigają się w inicjalizacji i tworzeniu zdarzeń JavaScript. Stąd wniosek, że nie możesz zakładać, iż zostaną uruchomione w jakiejś konkretnej kolejności. Z tego względu w kodzie naszej aplikacji musieliśmy zastosować pewne sztuczki.

   Nie musisz się jednak za bardzo tym przejmować, a jedyne, co powinieneś zrobić, to zmienić jedną linię w kodzie.

Zmień to...

```
document.addEventListener('deviceready', initPhoneGap, false);
```

...na to:

```
initPhoneGap();
```

Przy okazji lepiej byłoby nazwać tę funkcję tak, by nie zawierała nazwy PhoneGap (ups!).

Aplikacja Ustrzel tartan! jest gotowa na sytuację, w której przeglądarki zaczną obsługiwać API mediaCapture!

Czeka nas długa droga. Przyda się kilka wskazówek

Szukasz podpowiedzi, jak manifest „future friendly” zastosować w praktyce? Oto kilka myśli ze strony *http://futurefriend.ly*, nad którymi warto się zastanowić. To nie są gotowe przepisy, ponieważ nikt nie wie, co przyniesie nam przyszłość. To raczej wskazówki określające obszary, którymi warto się zainteresować, oraz problemy, które należy wziąć pod uwagę podczas pracy nad projektami.

★ PRECYZYJNE OGNISKOWANIE ★

Nie wszystko może się udać na każdym urządzeniu. Aby poradzić sobie w świecie wciąż rosnącego skomplikowania urządzeń, musimy się skupiać na najistotniejszych – z punktu widzenia klientów – sprawach. I nie chodzi tu o rozwiązania oparte na najmniejszym wspólnym mianowniku, ale o tworzenie treści i usług mających znaczenie. Użytkownicy są w coraz większym stopniu zmęczeni rosnącym szumem informacyjnym i koniecznością szukania sposobów na upraszczanie rzeczywistości. Musisz precyzyjnie zogniskować swoje usługi, zanim klienci i rosnąca różnorodność zrobią to za Ciebie.

★ UNIWERSALNA ZAWARTOŚĆ ★

Nadrzędnym nakazem jest tworzenie dobrze ustrukturyzowanej zawartości. Zastanów się, jak opracowana zawartość będzie się zachowywała w różnych kontenerach, uwzględniając ich ograniczenia i możliwości. Bądź odważny i korzystaj z nowych możliwości, ale pamiętaj, że przyszłość może podążyć w różnych kierunkach.

Na naszą przyszłość składają się zarówno zaawansowane urządzenia o ogromnych możliwościach, jak i proste urządzenia z minimalną funkcjonalnością. Tworząc strukturę zawartości, bierz to pod uwagę.

★ ORBITOWANIE WOKÓŁ DANYCH ★

Ekosystem urządzeń wymaga możliwie prostych międzyplatformowych i wydajnych mechanizmów wymiany danych. Dostosowuj się do istniejących, ale i dopiero co powstających możliwości, definiując dane w taki sposób, by:

- Był możliwy wielokrotny (i elastyczny) dostęp do danych.
- Były stosowane międzyplatformowe standardy.
- Dane były nakierowane na długoterminową integralność.
- Dane zawierały znaczące i stałe odwołania do całej zawartości.
- Były wspierane operacje odczytu i zapisu.

★ IDENTYFIKACJA OKRĘTÓW WIDMO ★

Uwzględnienie w projekcie wszystkich możliwych konfiguracji urządzeń jest nie lada wyzwaniem. Proces adaptacji można uprościć, stosując wysokopoziomowe i dobrze dobrane zbiory standardów dla różnych typów urządzeń. Standardy te można uzupełnić szczegółowymi informacjami o profilu urządzenia.

Tego typu taksonomia może pomóc uszeregować obecnie dostępne urządzenia różnych producentów, pozwalając na dołączenie nowych typów urządzeń, które pojawią się w przyszłości.

★ DOWODZENIE FLOTĄ ★

Korzystanie z wielu różnych urządzeń w codziennym życiu pozwala nam rozdzielać zadania i informacje pomiędzy nimi. Gdy jakieś zadanie jest zarządzane przez kolekcję urządzeń, każde z nich może zastosować interakcje, z którymi radzi sobie najlepiej. Dzięki temu nie musimy dostarczać wszystkich aspektów danego zadania do każdego urządzenia, ponieważ nastawiamy się raczej na pracę w ekosystemie możliwości urządzeń.

To jest dosyć przyszłościowy problem. Wyobraź sobie na przykład, że zegarek jest drugim ekranem Twojego telefonu.

KTO ZA CO ODPOWIADA?

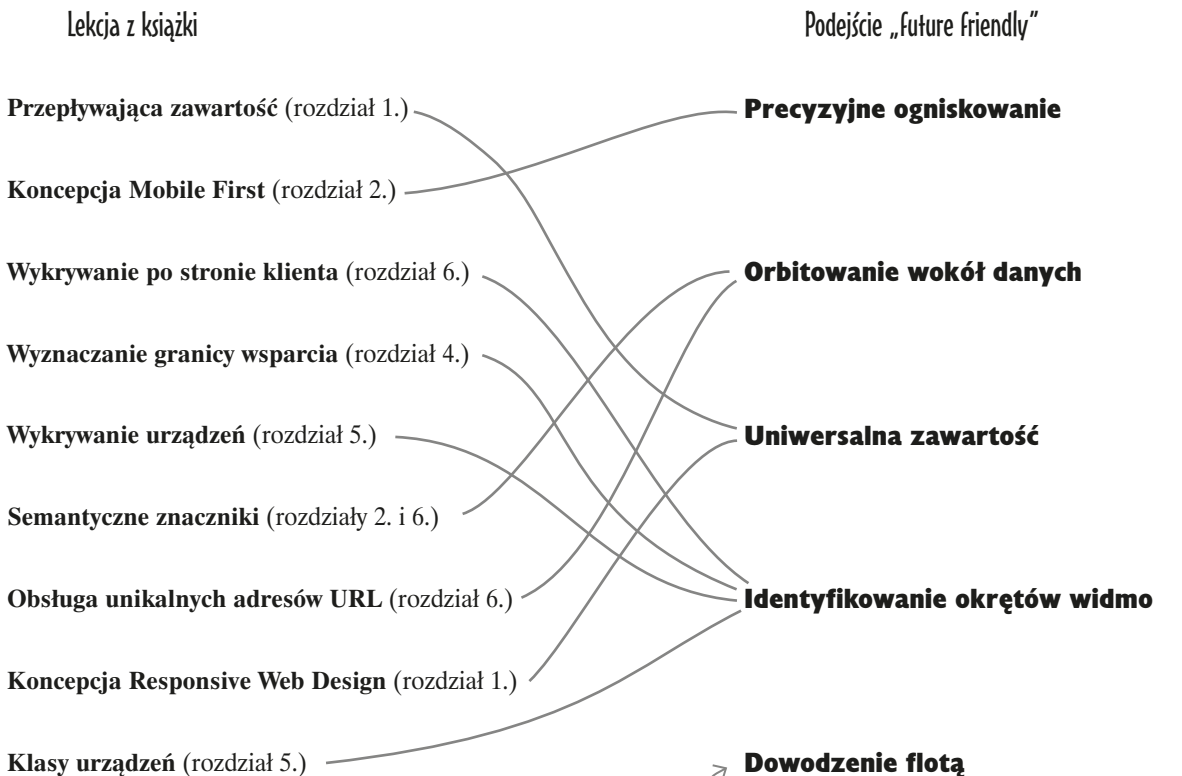
Dopasuj każdą lekcję z tej książki do którejś ze wskazówek podejścia „future friendly”. Wiele lekcji może odpowiadać tym samym koncepcjom.

Lekcja z książki	Podejście „Future Friendly”
Przepływająca zawartość (rozdział 1.)	**Precyzyjne ogniskowanie**
Koncepcja Mobile First (rozdział 2.)	
Wykrywanie po stronie klienta (rozdział 6.)	**Orbitowanie wokół danych**
Wyznaczanie granicy wsparcia (rozdział 4.)	
Wykrywanie urządzeń (rozdział 5.)	**Uniwersalna zawartość**
Semantyczne znaczniki (rozdziały 2. i 6.)	
Obsługa unikalnych adresów URL (rozdział 6.)	**Identyfikowanie okrętów widmo**
Koncepcja Responsive Web Design (rozdział 1.)	
Klasy urządzeń (rozdział 5.)	**Dowodzenie flotą**

KTO ZA CO ODPOWIADA?

ROZWIĄZANIE

Być może znalazłeś trochę inne elementy podejścia „future friendly" w poszczególnych lekcjach. My proponujemy takie rozwiązanie.



To hasło było podstępne. Ta wskazówka jest na tyle przyszłościowa, że żadna z lekcji bezpośrednio nie poruszała związanych z nią tematów. Wciąż czekamy na zegarki w stylu Dicka Tracy'ego.

Mieszamy mobilne składniki

Ćwiczenie

Przygotowanie mobilnej uczty wymaga połączenia odpowiednich składników dla uzyskania niepowtarzalnego smaku. Które przyprawy sprawią, że poniższe dania wzniosą się na wyżyny smaku i staną się „future friendly"? Wpisz je na kartach z przepisami.

Składniki

Uniwersalna zawartość

Nie próbuj naśladować natywnych aplikacji

Wykrywanie urządzeń po stronie klienta

Używaj mikroformatu hCard i znacznika `address`

hCard to standardowy zestaw znaczników przeznaczonych do zapisywania adresów. Więcej na ten temat dowiesz się na stronie http://microformats.org/wiki/hcard.

Elementy semantyczne HTML5 (`article`, `section` itd.)

Znaczniki obrazów z atrybutem `alt` poprawiającym dostępność

Odsyłacze ustawiające ciasteczka z preferencjami użytkownika

Tartanator

Problemy

- Na BlackBerry 5.0 działa strasznie wolno.
- Przewijanie zawartości ekranu jest powolne.
- Są dostępne dwa przyciski powrotu (jeden przeglądarki, a drugi utworzony przez aplikację).
- Tartany na stronach ze szczegółami powinny być zawartością, a są ustawiane jako obraz tła.

Zwierzętom na Pomoc

Problemy

- Trzeba zapewnić przejście z witryny mobilnej do desktopowej.
- Zawartość witryny mobilnej jest uboższa od wersji desktopowej.

Pod Paradnym Morsem

Problemy

- Cała zawartość jest umieszczona w blokach .
- Adres pubu to zwykły tekst oddzielony znacznikami
. Kod dokumentu jest pozbawiony elementów semantycznych.

Rozwiązanie ćwiczenia

Oto, do czego doszliśmy. Może wpadłeś na inne pomysły ulepszenia tych projektów i sprawienia, by były bardziej „future friendly”?

Zwierzętom na Pomoc

Problemy

- Trzeba zapewnić przejście z witryny mobilnej do desktopowej.
- Zawartość witryny mobilnej jest uboższa od wersji desktopowej.

Jeśli musisz utworzyć oddzielne witryny, pozwól użytkownikom zdecydować, którą chcą oglądać.

Odsyłacze ustawiające ciasteczka z preferencjami użytkownika

Uniwersalna zawartość

Nawet na oddzielnej mobilnej witrynie użytkownicy będą chcieli robić wszystko to, co mogą na jej wersji desktopowej.

BlackBerry 5.0 ma obsługę wymaganych możliwości JavaScriptu. Problem w tym, że działa powoli. Lepszym rozwiązaniem będzie wykrycie tych urządzeń i dostarczenie im wersji aplikacji bez jQuery Mobile.

Tartanator

Wykrywanie urządzeń po stronie klienta

Problemy

- Na BlackBerry 5.0 działa strasznie wolno.
- Przewijanie zawartości ekranu jest powolne.
- Są dostępne dwa przyciski powrotu (jeden przeglądarki, a drugi utworzony przez aplikację).

Nie próbuj naśladować natywnych aplikacji

- Tartany na stronach ze szczegółami powinny być zawartością, a są ustawiane jako obraz tła.

Znaczniki obrazów z atrybutem alt poprawiającym dostępność

Powielony przycisk powrotu i powolne przewijanie bierze się stąd, że w przeglądarce staraliśmy się odtworzyć wygląd i zachowanie natywnej aplikacji. Skoro ta aplikacja nie jest dostępna w sklepie App Store ani Google Play, po co udawać, że to jest aplikacja?

Obrazy z CSS przenieś do znaczników <img>.

Kod dokumentu będzie niósł więcej znaczeń, jeśli użyjemy elementów semantycznych HTML5.

Pod Paradnym Morsem

Problemy

- Cała zawartość jest umieszczona w blokach .

Elementy semantyczne HTML5 (article, section itd.)

- Adres pubu to zwykły tekst oddzielony znacznikami
. Kod dokumentu jest pozbawiony elementów semantycznych.

Używaj mikroformatu hCard i znacznika address

O mikroformatach jedynie wspomnieliśmy. To kolejny sposób na uzupełnienie treści o semantykę. Niektóre przeglądarki i wyszukiwarki rozpoznają mikroformaty na stronie i używają ich w celu dostarczenia dodatkowych funkcjonalności.

Spójrz w przyszłość

Nie martw się o urządzenia i przeglądarki, które będą dostępne w przyszłości. Masz do dyspozycji mnóstwo różnych narzędzi. Od małych do ogromnych ekranów, od telefonów komórkowych do lodówek podpiętych do internetu — jesteś gotowy na wszystko!

Jeśli masz podstawę w postaci standardów sieciowych i techniki stopniowego ulepszania, możesz tworzyć witryny i aplikacje, które są dostępne dla ogromnej rzeszy użytkowników.

Przyjmij z otwartymi ramionami wszystko to, co przyniesie nam w przyszłości internet. Jest to co prawda wielka niewiadoma, ale z całą pewnością świat mobilnych urządzeń dostarczy nam mnóstwo możliwości i — o czym nie trzeba wspominać — radości.

Zatem do dzieła! Bądź mobilny, twórz przyszłość!

Dodatek A Ścinki

Sześć najważniejszych spraw (o których nie mówiliśmy)

Czujesz się, jakby coś Ci umknęło? Wiemy, co masz na myśli...
Zawsze jest tak samo — myślisz, że to już koniec, a okazuje się, że jest tego więcej. Nie mogliśmy się pohamować, by nie przekazać Ci kilku dodatkowych szczegółów, o których nie wspomnieliśmy w treści książki. Gdybyśmy chcieli napisać o wszystkich ciekawych sprawach, książkę musiałbyś transportować w pancernej walizie na kołach. Rzuć okiem, co dla Ciebie wybraliśmy.

1. Testowanie na urządzeniach mobilnych

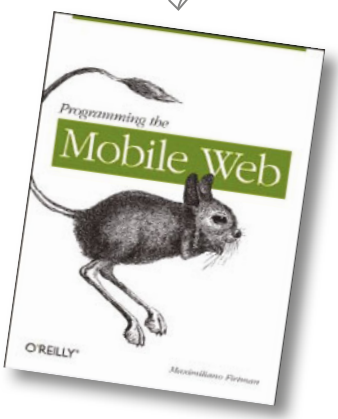
Testowanie na urządzeniach mobilnych jest do kitu. Właśnie tak, do kitu. Najgorsze, że nie ma na to rady. Kiedy zdasz sobie sprawę z tego, że Twoja witryna albo aplikacja może działać różnie na tym samym telefonie, ale pracującym w innej sieci, dojdziesz do smutnego wniosku, że na niczym nie możesz polegać.

Jak nie zwariować? Oto kilka sposobów, które ułatwią Ci życie.

1. **Zacznij od poprawnego kodu w przeglądarce desktopowej.**
Jeśli coś pójdzie nie tak w przeglądarce desktopowej, na pewno się wysypie w mobilnej. Równie dobrze możesz zacząć testować tam, gdzie jest to znacznie prostsze i dostępne są lepsze narzędzia. Im szybciej znajdziesz błędy, tym lepiej.

2. **Korzystaj z emulatorów i symulatorów urządzeń mobilnych.**
Prawie wszystkie mobilne platformy dostarczają emulatory lub symulatory. Maximiliano Firtman umieścił w swojej książce, ***Programming the Mobile Web***, listę emulatorów i symulatorów, którą cały czas uaktualnia na stronie *http://www.mobilexweb.com/emulators*.

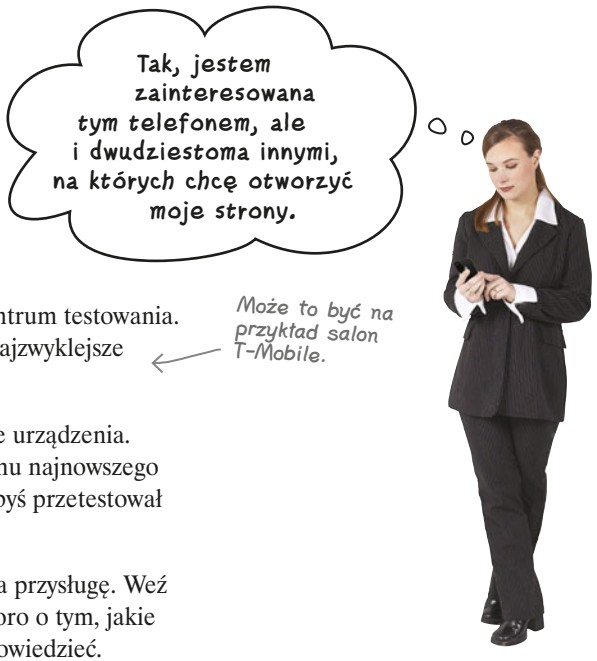
W książce Maksa znajdziecie doskonały opis mobilnych emulatorów i symulatorów.



3. **Zainwestuj w kilka urządzeń.**
Nie da się uniknąć kupna kilku urządzeń. To, które z nich kupić, zależy od typu projektów oraz docelowego rynku. Sprawdź, z czego korzystają główni interesariusze projektu, i koniecznie zdobądź takie telefony. W budżecie każdego projektu uwzględniaj koszt zakupu urządzeń.

Peter-Paul Koch opublikował w magazynie internetowym A List Apart artykuł zawierający wiele cennych wskazówek dotyczących wyboru telefonów do testów: http://www.alistapart.com/articles/smartphone-browser-landscape/.

4. **Proś, pożyczaj i kradnij.**
Skontaktuj się z innymi twórcami oprogramowania mobilnego i wymieniaj z nimi urządzenia. Dobrym pomysłem może być też utworzenie bazy urządzeń w Twoim środowisku, tak by uprościć znalezienie pożądanego telefonu. Możesz nawet pójść dalej i stworzyć prawdziwe laboratorium testujące urządzenia mobilne, podobne do tego, które założyliśmy w Portland.



5 Odwiedź lokalne mobilne centrum testowania.
W prawie każdym mieście możesz znaleźć mobilne centrum testowania. No dobrze, nie są to jakieś laboratoria naukowe, ale najzwyklejsze w świecie salony operatorów sieci komórkowej.

Może to być na przykład salon T-Mobile.

W większości salonów klienci mogą wypróbować różne urządzenia. Pracownicy salonu nie pozwolą Ci raczej wziąć do domu najnowszego smartfonu, ale nie powinni mieć nic przeciwko temu, byś przetestował swoją witrynę na kilku urządzeniach.

Przedstaw sprawę jasno i nie zapomnij podziękować za przysługę. Weź pod uwagę, że pracownicy salonów wiedzą całkiem sporo o tym, jakie telefony najlepiej się sprzedają, i mogą Ci sporo podpowiedzieć.

6 Korzystaj ze zdalnych usług testowania.
Czasem naprawdę musisz przetestować jakiś określony scenariusz. Możesz na przykład chcieć sprawdzić, jak Twoja witryna działa na konkretnym modelu telefonu, który jest dostępny tylko w jednej sieci komórkowej na drugim końcu świata. W takich przypadkach zdalne usługi testowania są niezastąpione.

Device Anywhere (*http://deviceanywhere.com*) i Perfecto Mobile (*http://perfectomobile.com*) umożliwiają zdalny dostęp do prawdziwych telefonów rozlokowanych w centrach na całym świecie. Oznacza to, że możesz przeprowadzać testy na konkretnych urządzeniach mających dostęp do internetu dostarczanego przez lokalnych operatorów, dzięki czemu możesz sprawdzić, jak Twoja witryna będzie działać w rzeczywistych warunkach.

W centrum Device Anywhere w San Mateo w Kalifornii telefony są częściowo rozmontowane i podpięte do serwerów. Dzięki temu masz do nich dostęp z dowolnego miejsca na ziemi.

W przypadku obu wspomnianych usług płaci się za czas przeprowadzania testów, więc musisz być efektywny, bo w przeciwnym razie słono zapłacisz.

7 Ustal priorytety testów.
Nawet gdybyś miał dostęp do każdego istniejącego telefonu, nie starczyłoby Ci czasu na testy na każdym z nich. Wybierz reprezentatywne urządzenie i ustal priorytety testów, opierając się na założeniach projektowych wynikających z preferencji przyszłych użytkowników.

2. Zdalne debugowanie

Już wspomnieliśmy, jak trudno jest sprawdzić, co się dzieje w mobilnej przeglądarce. Jednak nawet gdybyśmy mogli sprawdzić, co się stało, trudno sobie wyobrazić, byśmy — jako programiści — mogli cokolwiek zrobić na tak małym ekranie.

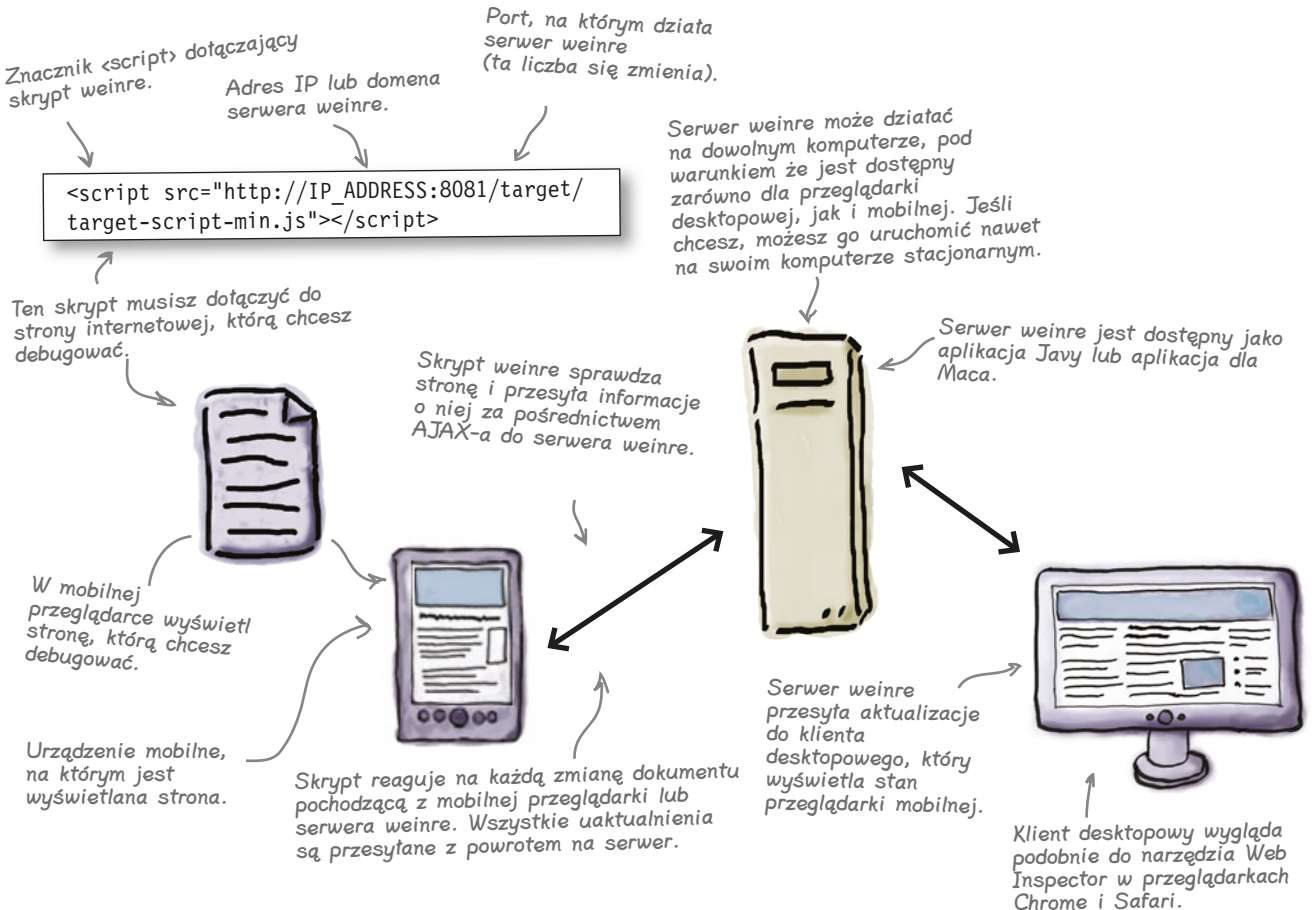
Zdalne debugowanie rozwiązuje ten problem. Cała sprawa polega na połączeniu mobilnej przeglądarki z desktopowym narzędziem do debugowania, które pozwala na śledzenie działania mobilnej przeglądarki.

Narzędzie WEb INspector REmote (weinre)

Istnieje kilka możliwych rozwiązań zdalnego debugowania. Jednym z nich, które może być stosowane z większością urządzeń i przeglądarek, jest narzędzie weinre (wymawiane jak angielskie słowo *winery*). Jest to projekt typu open source zapoczątkowany przez jednego z współtwórców silnika WebKit, Patricka Muellera, a później wzięty pod skrzydła PhoneGap.

Narzędzie możesz pobrać ze strony http://phonegap.github.com/weinre/.

Jak działa weinre?



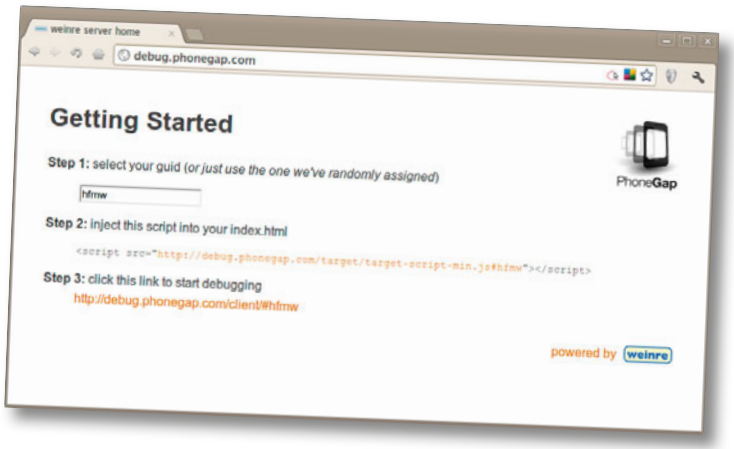
Uruchomienie serwera weinre

Uruchomienie serwera weinre to nie operacja na otwartym sercu, ale nie jest też banalnie proste. Na szczęście dobrzy ludzie z PhoneGap postawili publiczny serwer, z którego można korzystać. Wszystko, czego potrzebujesz, to desktopowa i mobilna przeglądarka oraz dostęp do internetu.

Jednym z minusów korzystania z publicznego serwera jest to, że jest dostępny dla wszystkich. W związku z tym musisz wybrać taki identyfikator, który z dużym prawdopodobieństwem jest unikalny.

1 Za pomocą przeglądarki Chrome lub Safari przejdź na stronę *debug.phonegap.com*.

Na stronie zostały opisane trzy proste kroki, które należy wykonać. W pierwszym kroku musisz dostarczyć tzw. *guid* (ang. *globally unique identifier*), czyli globalny unikalny identyfikator. Chodzi o to, by podać słowo lub zdanie, które pozwoli na jednoznaczną identyfikację Twojej usługi testowej weinre, tak by nikt inny z niej nie skorzystał.



2 Utwórz przykładowy dokument HTML, który chcesz testować.

Utwórz nowy dokument HTML, umieść w nim poniższy kod i zapisz pod nazwą *weinre.html*. Dodaj znacznik `<script>` podany na stronie *debug.phonegap.com*, a następnie umieść zaktualizowany plik na publicznym serwerze.

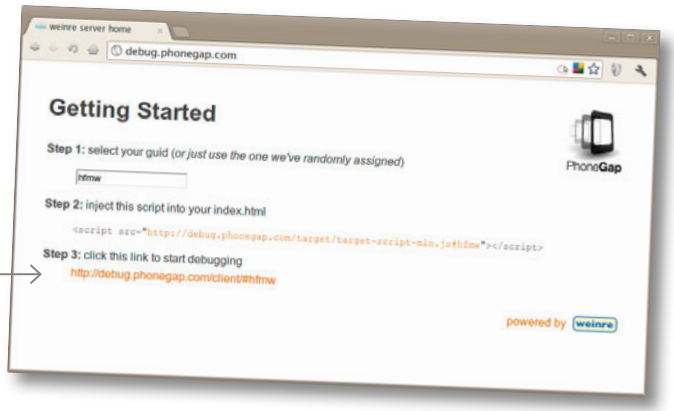
```
<!DOCTYPE html>
<html lang="pl">
<head>
    <title>Przykład weinre</title>
    <style type="text/css">
    h1 {color:red}
    #box {background:#ffc;padding:10px;width:70%;height:200px;}
    </style>
    <script src="http://debug.phonegap.com/target/target-script-min.
js#hfmw"></script>
</head>
<body>
<h1>Strona testowa weinre</h1>
<p id="box">Na początku to jest żółty prostokąt.</p>
</body>
</html>
```

Umieść plik weinre.html na publicznym serwerze.

Dodaj znacznik <script> wyświetlony na stronie debug.phonegap.com.

Ze względów bezpieczeństwa nie możemy zaoferować Ci hostingu.

3 **Uruchom debugowanie.**
Kliknij odsyłacz znajdujący się w trzecim kroku na stronie *debug.phonegap.com*, aby uruchomić debugowanie.



Kliknij ten przycisk, aby uruchomić debugowanie.

Po uruchomieniu debugowania zobaczysz taką stronę.

W pasku tytułowym okna przeglądarki widoczny jest tekst „target not connected", który oznacza, że żadne urządzenie nie połączyło się z debuggerem.

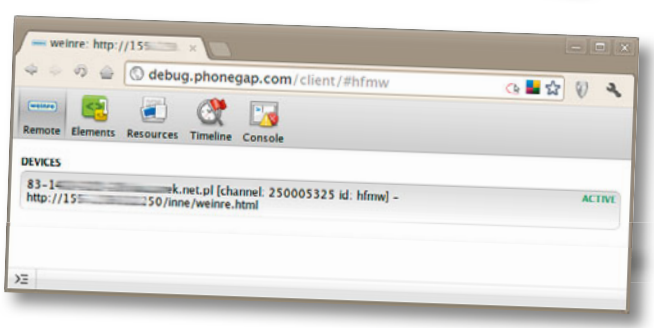
W sekcji DEVICES są wyświetlane urządzenia połączone z serwerem weinre. Do tej pory się nie połączyliśmy, więc pora to zmienić.

4 **Otwórz plik weinre.html w przeglądarce mobilnej.**
Sprawdź, czy przeglądarka mobilna obsługuje połączenia AJAX (urządzenia z Androidem i iPhone'y oraz nowsze smartfony powinny działać). W oknie przeglądarki powinieneś zobaczyć testową stronę z dużym żółtym prostokątem i czerwonym nagłówkiem.

Strona weinre.html na iPhonie.

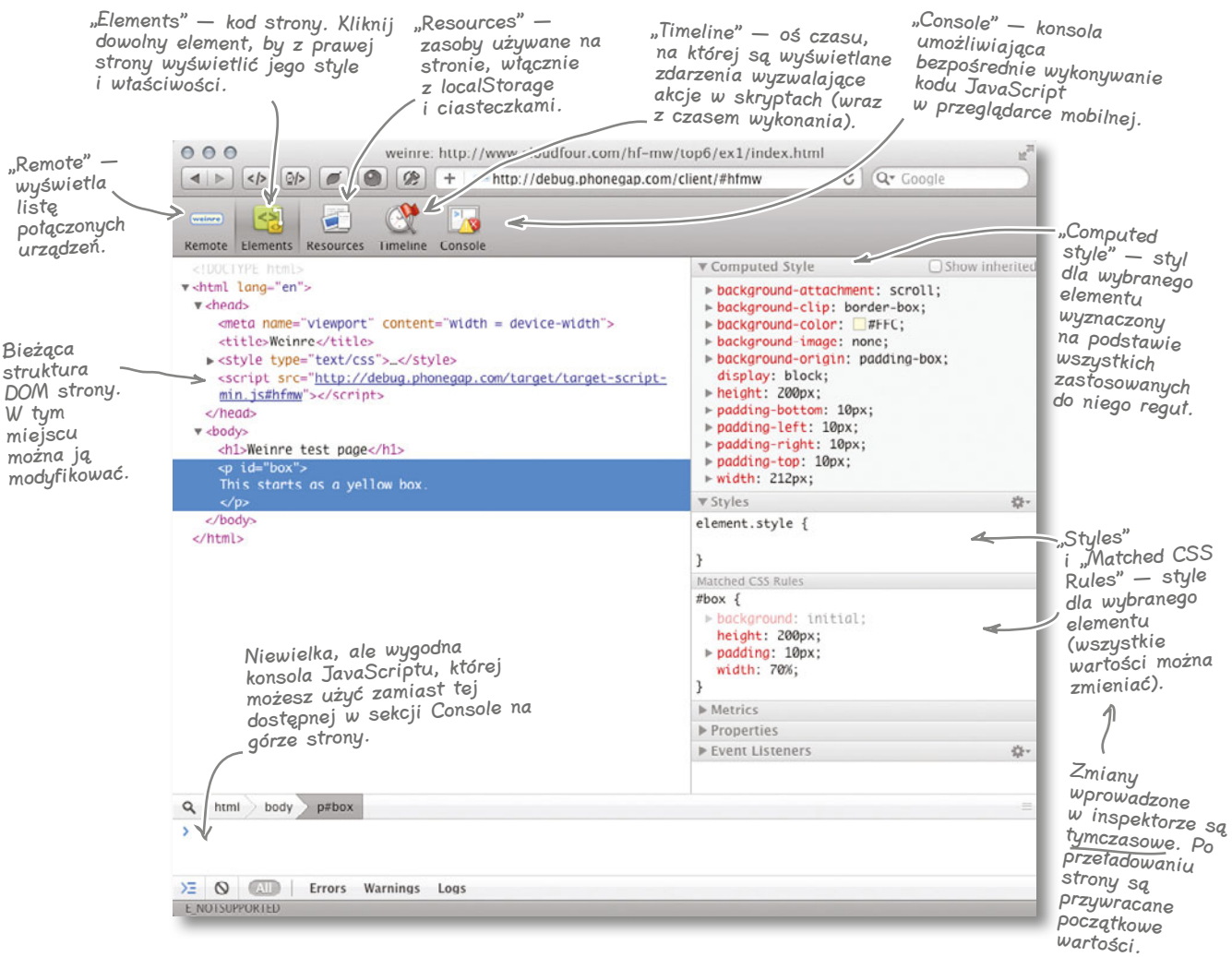
5 **Sprawdź, czy urządzenie połączyło się z debuggerem.**
Na liście wyświetlanej na stronie debuggera powinieneś zobaczyć swoje urządzenie. Jeśli go nie widzisz, odśwież stronę.

Testowe urządzenie jest już widoczne na liście.



Poznajemy inspektora weinre

Teraz, kiedy udało już się nam nawiązać połączenie, w przeglądarce desktopowej widać uproszczoną wersję narzędzia Web Inspector z Google Chrome lub Safari.



Ćwiczenie

Pewnie myślałeś, że masz już spokój z ćwiczeniami? My też tak sądziliśmy, więc potraktuj to ćwiczenie raczej jako zabawę. Wprowadź zmiany na stronie za pomocą inspektora i obserwuj efekty na telefonie.

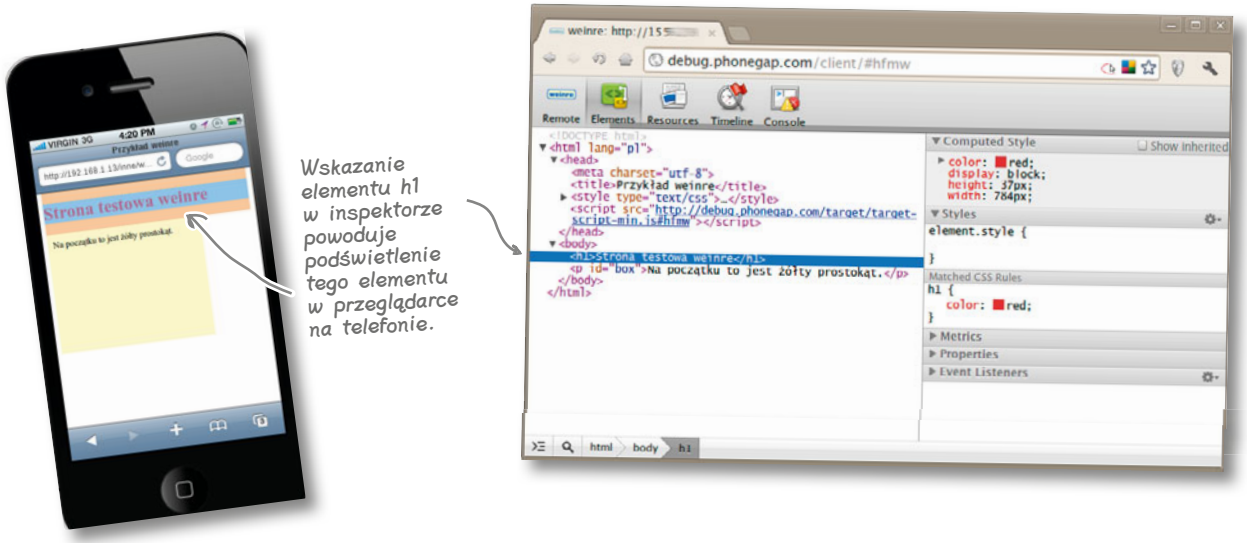
1. Przejedź wskaźnikiem myszy nad elementami na zakładce *Elements*. Co się dzieje w przeglądarce mobilnej?
2. W inspektorze zmień kolor tła z żółtego na dowolny inny.
3. Za pomocą konsoli wyświetl okienko `alert` na telefonie.

Rozwiązanie ćwiczenia

Przyznasz chyba, że jest coś z magii w tym, że efekty zmian wprowadzanych w inspektorze weinre można obserwować w przeglądarce na telefonie, prawda?

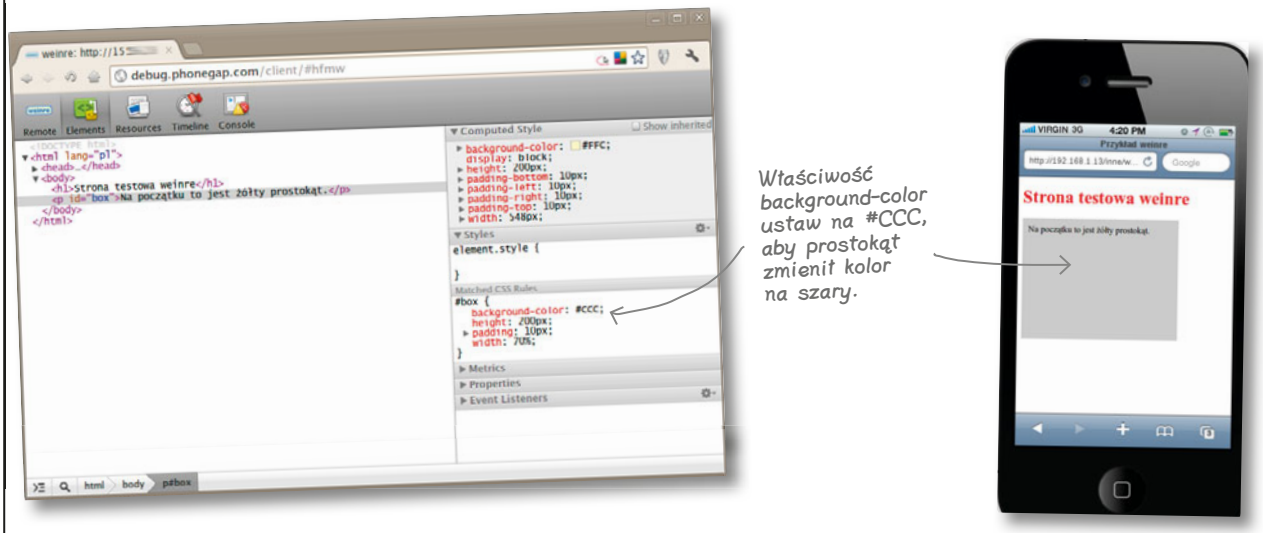
1. Przejedź wskaźnikiem myszy nad elementami na zakładce *Elements*. Co się dzieje w przeglądarce mobilnej?

Elementy wyświetlane w przeglądarce mobilnej są podświetlane po wskazaniu ich w inspektorze uruchomionym w przeglądarce desktopowej.



2. W inspektorze zmień kolor tła z żółtego na dowolny inny.

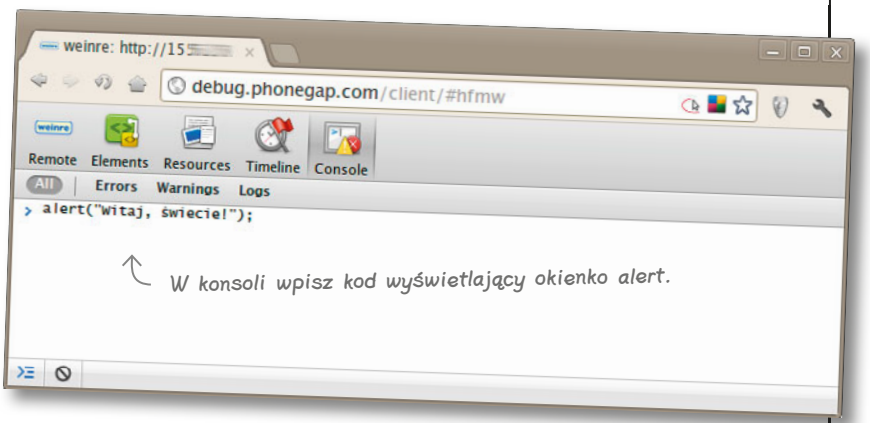
W inspektorze kliknij znacznik <p> i w sekcji „Matched CSS Rules" zmień wartość właściwości background-color.



3. Za pomocą konsoli wyświetl okienko `alert` na telefonie.

Wpisz kod JavaScript wyświetlający klasyczne okienko „Witaj, świecie!”.

```
alert("Witaj, świecie!");
```



Tak jak się można było spodziewać, okienko zostało wyświetlone na telefonie.

Nie istnieją głupie pytania

P: Czy usługa debug.phonegap.com jest bezpieczna? Dlaczego nie chcecie hostować strony testowej?

O: Problemy z bezpieczeństwem są raczej rzadkością. Ktoś musiałby znać Twój adres serwera weinre i wiedzieć, że akurat w tej chwili z niego korzystasz, by przesłać złośliwy kod JavaScript do Twojej przeglądarki.

Tak jak wspomnieliśmy, jest bardzo mało prawdopodobne, by do tego doszło, chyba że opublikujesz adres URL serwera w książce, będziesz hostował testową stronę i powiesz wszystkim, by z niej korzystali. Publikowanie strony jest w tym przypadku niebezpieczne i możesz stać się celem ataku.

P: Wspomnieliście o innych debuggerach. Powiedzcie coś więcej.

O: Opera Dragonfly (*http://opera.com/dragonfly*) oferuje zdalny debugger, który można stosować z przeglądarką Opera. BlackBerry Playbook i BlackBerry OS od wersji 7. mają wbudowane wsparcie dla zdalnego debugowania. Silnik WebKit zawiera podobne narzędzie, ale w większości przeglądarek jest wyłączone. Wspomniany już Maximiliano Firtman stworzył narzędzie iWebInspector (*http://iwebinspector.com*), które jest zdalnym debuggerem dla iOS.

P: Czy po przejęciu firmy Nitobi przez Adobe, czyli ludzi stojących za PhoneGap, usługa *debug.phonegap.com* nie zniknie?

O: Usłyszeliśmy zapewnienie, że usługa będzie nadal dostępna. Jeśli jednak coś się zmieni, nadal będziesz mógł pobrać weinre i uruchomić go na własnym komputerze.

3. Sprawdź, co obsługują przeglądarki

Bardzo dobrze, jeśli stosujesz technikę stopniowego ulepszania, by obsłużyć lepsze przeglądarki, ale zastanów się, jaki sens ma spędzanie masy czasu na tworzeniu czegoś, z czego może skorzystać zaledwie kilka urządzeń.

Być może kiedyś chciałbyś sprawdzić, które przeglądarki będą w stanie obsłużyć stworzony przez Ciebie system nawigacji wykorzystujący żyroskop wbudowany w telefon.

Nie rób tego! Nawigacja sterowana żyroskopem to przerażający pomysł. To sto razy gorsze niż użycie znacznika <blink> ;). Zapomnij, że o tym w ogóle wspomnieliśmy.

Listy możliwości przeglądarek

Na tych stronach znajdziesz katalog możliwości poszczególnych przeglądarek.

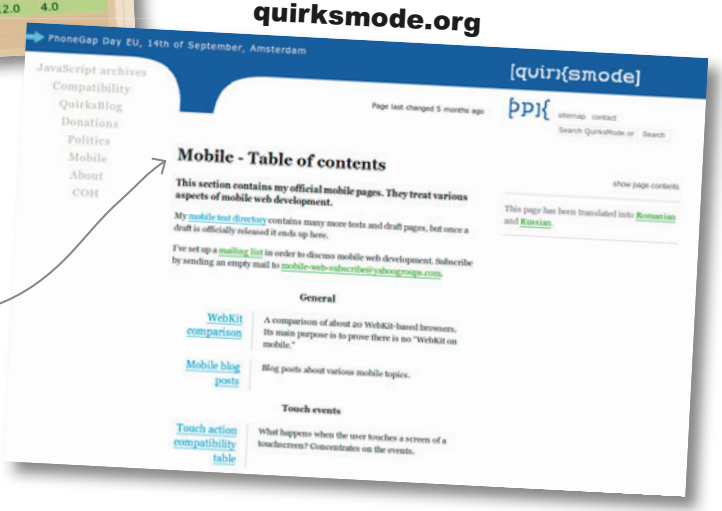
Odpowiada na pytanie: „Gdzie mogę użyć...?".

caniuse.com

Możemy tu znaleźć wykaz kompatybilności dla HTML-a, CSS oraz JavaScriptu zarówno dla desktopowych, jak i mobilnych przeglądarek.

Prababcia wszystkich stron z wykazami kompatybilności. Peter-Paul Koch (PPK) rozpoczął testowanie przeglądarek w 1998 roku, a do QuirksMode.org dołączył w 2003.

quirksmode.org



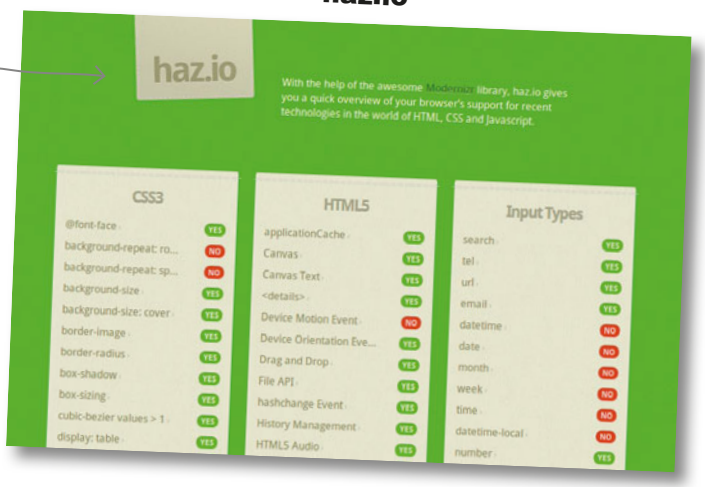
PPK w tej chwili poświęcił się w całości przeglądarkom mobilnym. Proponujemy, byś czytał jego blog i zainteresował się konferencją Mobilism, którą organizuje.

Sprawdź, co potrafi Twoja przeglądarka

Te strony są z kolei przydatne, jeśli chcesz sprawdzić, jakie możliwości ma przeglądarka, z której korzystasz. Wystarczy jedna wizyta, byś wiedział wszystko.

haz.io

Serwis haz.io korzysta z biblioteki Modernizr (www.modernizr.com), by sprawdzić, w jakim stopniu przeglądarka wspiera HTML, CSS i JavaScript.



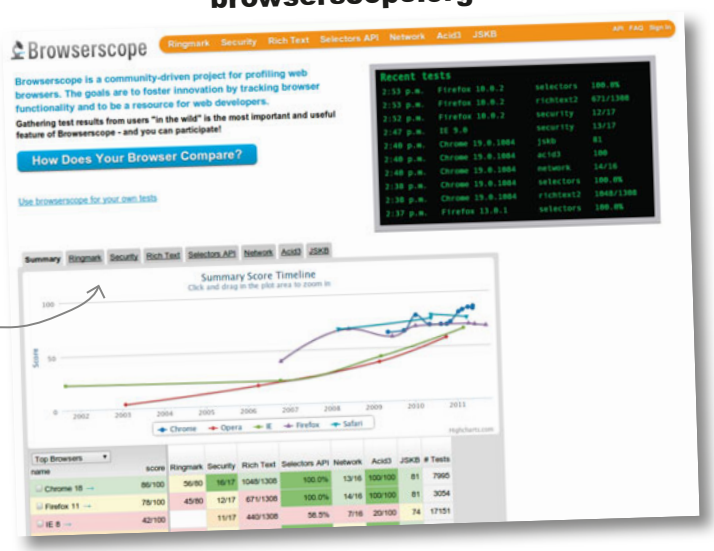
html5test.com

Sprawdź swoją przeglądarkę pod kątem wsparcia dla nowych możliwości HTML5. Życzymy Ci jak najwyższych wyników!

Możesz też sprawdzić wszystko naraz

browserscope.org

Za pomocą serwisu Browserscope przetestujesz swoją przeglądarkę pod kątem wsparcia dla różnych możliwości, podobnie jak haz.io, ale zostanie uwzględnione również to, co wcześniej nie było obsługiwane.



Możesz opracować własne testy i zapisywać dane w Browserscope. Programiści związani z tym serwisem wciąż tworzą nowe testy, dzięki czemu uzyskiwane tu dane są coraz bardziej wartościowe.

4. Interfejsy API urządzeń

Mimo swoich niewielkich rozmiarów telefony komórkowe i smartfony to całkiem wydajne urządzenia z mnóstwem możliwości, takich jak książka adresowa, kalendarz, przesyłanie wiadomości, wibracje czy sensory, i wieloma nowoczesnymi technologiami nakierowanymi na komunikację. W przykładach omawianych w książce ledwie musnęliśmy temat wykorzystania ich możliwości — użyliśmy na przykład kamery w aplikacji PhoneGap.

Trzeba sobie zadać pytanie, co będzie możliwe, gdy przeglądarki zaczną obsługiwać interfejsy API urządzeń.

Czym są API urządzeń?

Pierwotnie termin ***API urządzenia*** odnosił się do zbioru specyficznych funkcjonalności dostępnych na urządzeniach mobilnych, które nie były oferowane przez komputery stacjonarne. Typowym przykładem może być kamera telefonu.

To, co początkowo było ograniczone do wąskiego zakresu typowo mobilnych możliwości, obejmuje obecnie w zasadzie wszystkie funkcjonalności — od definiowania zdarzeń dla kalendarza do odczytywania informacji o systemie, takich jak wykorzystanie procesora czy prędkość połączenia sieciowego.

Chociaż zakres się powiększył, nadal chodzi o to samo — udostępnienie przeglądarkom tego, co w urządzeniach mobilnych jest najlepsze.

Standardy według DAP i WAC

Nad standardami dostępu do API urządzeń pracują dwie grupy: Device APIs Working Group (DAP) działająca pod patronatem W3C oraz Wholesale Applications Community (WAC).

O W3C z pewnością już słyszałeś, bo to ta organizacja odpowiada za standardy HTML-a i CSS. Mogłeś nie słyszeć jednak o WAC.

Grupa WAC została powołana przez operatorów sieci oraz producentów telefonów, którzy postawili sobie za cel utworzenie wspólnej platformy dla twórców aplikacji. Oparli się na HTML5 i połączyli dwa istniejące standardy w jeden — WAC 2.0.

Między WAC i DAP jest wiele podobieństw, które można zauważyć zarówno w specyfikacjach, jak i członkach komitetów obu grup. Grupy współpracują ze sobą bardzo intensywnie.

To świetnie. Dlaczego nie wspomnieliście o tym w książce?

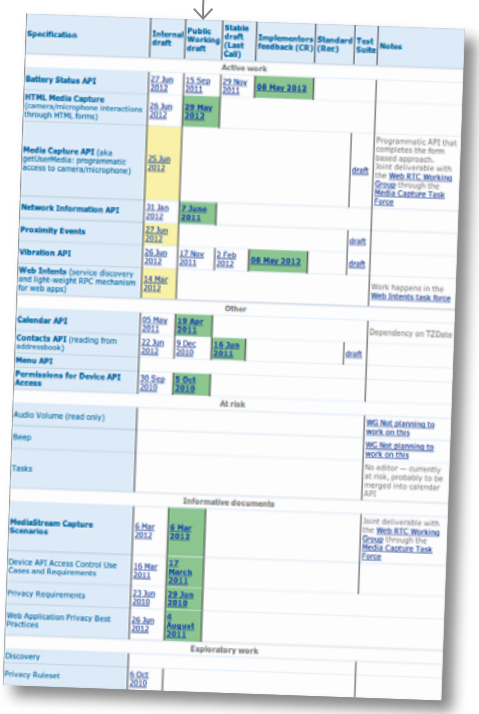
Wynika to stąd, że większość przeglądarek nie wspiera na razie interfejsów API urządzeń, a wiele ze standardów jest dopiero we wczesnej fazie opracowywania.

Miejmy nadzieję, że do czasu przygotowywania kolejnego wydania tej książki wspomniane standardy będą już opracowane, tak byśmy mogli im poświęcić kilka ciekawych rozdziałów.

W3C®

WAC

Proponujemy, byś śledził postępy prac grupy Device APIs Working Group prezentowane w harmonogramie dostępnym na stronie http://w3.org/2009/dap.



5. Sklepy z aplikacjami oraz dystrybucja

Stworzyłeś olśniewającą aplikację, opakowałeś ją w PhoneGap i jesteś gotowy ją sprzedawać. Jak zarobić kokosy w sklepach z aplikacjami?

Smutna prawda: większość aplikacji nie daje zarobić

Nie oznacza to oczywiście, że Twoja nie stanie się chlubnym wyjątkiem. Powinieneś jednak zdawać sobie sprawę ze swoich szans na zostanie milionerem.

Jednak w wielu sytuacjach aplikacje mogą się sprawdzić, zwłaszcza w przypadku firm, w których ich sprzedaż nie jest jedynym źródłem dochodu, ale dodatkowym.

Sklepów jest bez liku...

Obecnie jest za dużo sklepów. Firma Wireless Industry Partnership (WIP) prowadzi bazę danych wszystkich znalezionych sklepów z aplikacjami — w tej chwili jest ich ponad 120.

W każdym sklepie obowiązują inne zasady oraz strategie promowania aplikacji. Mnogość możliwych opcji obezwładnia.

...a teraz dołączają do nich aplikacje internetowe

Ostatnio można zaobserwować nowy trend związany z aplikacjami HTML5. Google Chrome, Mozilla, a także cała masa innych, mniejszych firm tworzy nowe sklepy z aplikacjami internetowymi.

Kilka wskazówek od nie-eksperta

Nie jesteśmy ekspertami w sprawach związanych ze sprzedażą aplikacji. Fachowych porad musisz więc poszukać gdzie indziej. Chcemy Ci jednak powiedzieć o trzech sprawach:

1. **Badaj rynek i przeszukuj sklepy z aplikacjami.**
   Musisz wiedzieć, co robi konkurencja. Dowiedz się więcej o sklepie, w którym zamierzasz wystawić aplikację, zwracając szczególną uwagę na profil oferowanych aplikacji. Odrób to zadanie bardzo sumiennie.

2. **Przygotuj klasyczny plan marketingowy.**
   Nie licz na to, że odniesiesz sukces tylko dlatego, że sprzedajesz aplikację w sklepie Apple'a. To nie film, a twarda rzeczywistość.

3. **Spraw, by Twoja aplikacja zachowywała się jak najbardziej natywnie.**
   Mimo że tworzysz aplikację za pomocą międzyplatformowych narzędzi, nie możesz wymagać od użytkowników Androida, by wiedzieli, jak działają aplikacje na iPhonie. Postaraj się tak dopracować każdą aplikację, by wyglądała jak natywna na konkretnej platformie.

6. RESS: REsponsive design + komponenty Server-Side

Nie wydaje się, by w dłuższej perspektywie techniki Responsive Web Design lub wykrywania urządzeń stały się na tyle dominujące, by wyeliminowały inne podejścia. Jeśli już, to w większości przypadków największy sens ma połączenie tych technik.

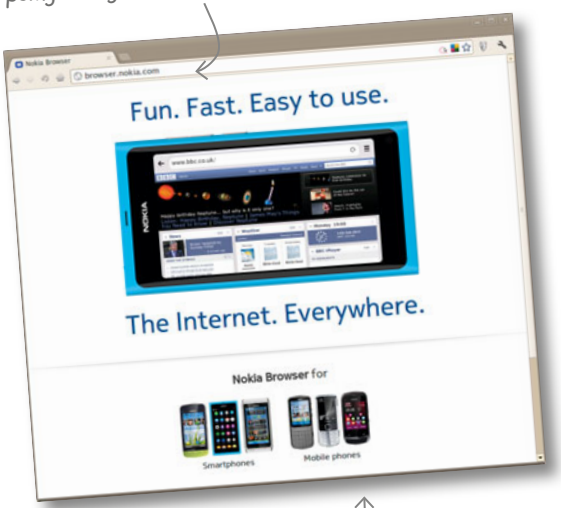
Luke Wroblewski nadał nawet nazwę tej kombinacji koncepcji Responsive Web Design i wykrywania urządzeń po stronie serwera — RESS.

To, co Luke nazwał RESS, niewiele się różni od innowacyjnych technik zastosowanych przez właścicieli serwisu Yiibu.com — Bryana i Stephanie Riegerów.

Zastosowali oni wykrywanie urządzeń, by w najlepszym stopniu dostosować przesyłaną zawartość do możliwości przeglądarki, a następnie — po stronie klienta — za pomocą JavaScriptu weryfikują trafność decyzji i poprawiają ewentualne problemy. Skrypt zapisuje również ciasteczko, które dostarcza serwerowi informacji o błędnych założeniach wynikających z nietrafnej detekcji urządzenia.

http://www.lukew.com/ff/entry.asp?1392

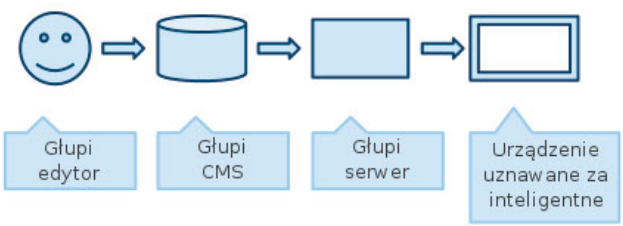
Na stronie http://browser.nokia.com możesz sprawdzić, jak w praktyce sprawdza się pomysł Bryana i Stephanie.



Więcej na ten temat znajdziesz na stronach http://www.slideshare.net/yiibu oraz http://yiibu.com.

Przyszłość RESS

Trudno wyrokować, czy nazwa RESS się przyjmie, ale bez względu na to, sam pomysł połączenia koncepcji Responsive Web Design i technik serwerowych wydaje się ciekawy. Dobrą ilustracją zaniechania tego pomysłu jest poniższy rysunek Jona Arnesa Sæteråsa.



Oryginał możesz znaleźć na stronie http://mpulp.mobi/2011/05/next-steps-of-responsive-web-design/.

Naszym celem powinno być doprowadzenie do sytuacji, w której każdy z tych elementów jest jak najbardziej „inteligentny”. Rozwiązania takie jak RESS zakładają, że klient i serwer dzielą odpowiedzialność za dostarczenie takiej treści, by była jak najlepiej dostosowana do urządzenia.

Dodatek B Posław swój serwer

Gdzieś trzeba zacząć

„Mobilny internet" nie istnieje bez słowa „internet". Bez dwóch zdań. Jeśli chcesz się zająć tworzeniem witryn i aplikacji mobilnych, będziesz potrzebował serwera WWW. Prędzej czy później dojdzie do sytuacji, w której będziesz potrzebował części serwerowej swojej aplikacji. Możesz wtedy skorzystać z bezpłatnego lub komercyjnego hostingu albo uruchomić serwer na swoim komputerze. W tym dodatku opiszemy proces **stawiania lokalnego serwera WWW z obsługą PHP** z wykorzystaniem bezpłatnego oprogramowania.

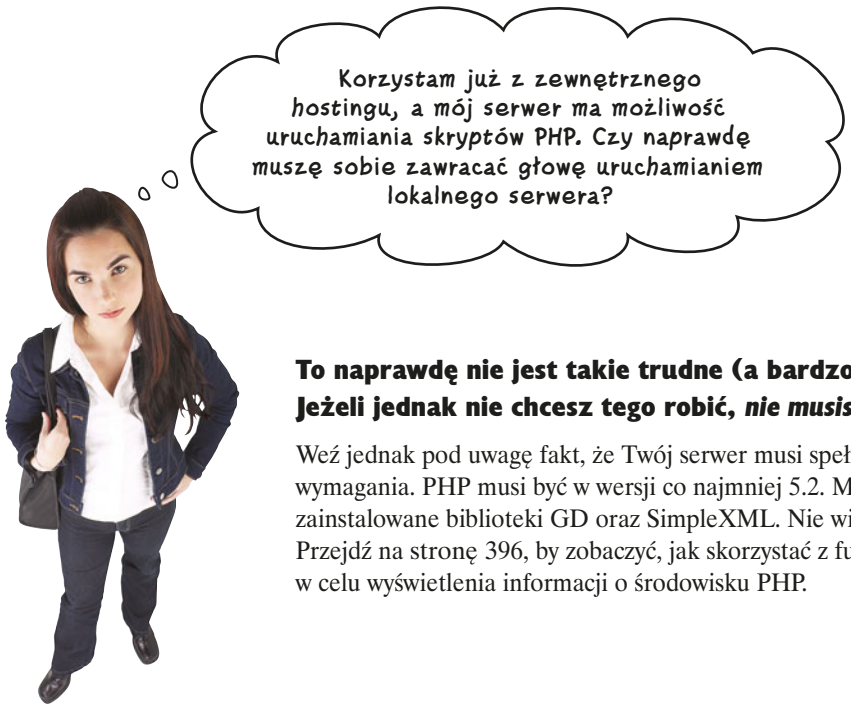
Czego będziesz potrzebował?

Aby wykonać niektóre ćwiczenia z tej książki, będziesz musiał uruchomić serwer WWW. Nie martw się, to wcale nie jest takie trudne, a my pokażemy, jak to zrobić.

1. **Musisz mieć możliwość udostępniania stron internetowych (na przykład plików HTML) i zasobów (obrazów, arkuszy CSS, skryptów JavaScript itp.).**
   To są podstawowe wymagania stawiane serwerom WWW, prawda? Musimy zatem postawić taki serwer i go skonfigurować.

2. **Musisz mieć możliwość uruchamiania na serwerze skryptów PHP.**
   W przykładach z tej książki do dynamicznego generowania zawartości korzystamy z języka skryptowego PHP (ang. *Hypertext Preprocessor*). Nie wymagamy, byś był biegły w *programowaniu* w tym języku, ale musisz umieć *uruchomić* skrypty.

3. **Musisz mieć choćby minimalne prawa administracyjne do serwera.**
   Musisz mieć możliwość modyfikowania uprawnień do katalogów oraz kopiowania i przenoszenia plików.

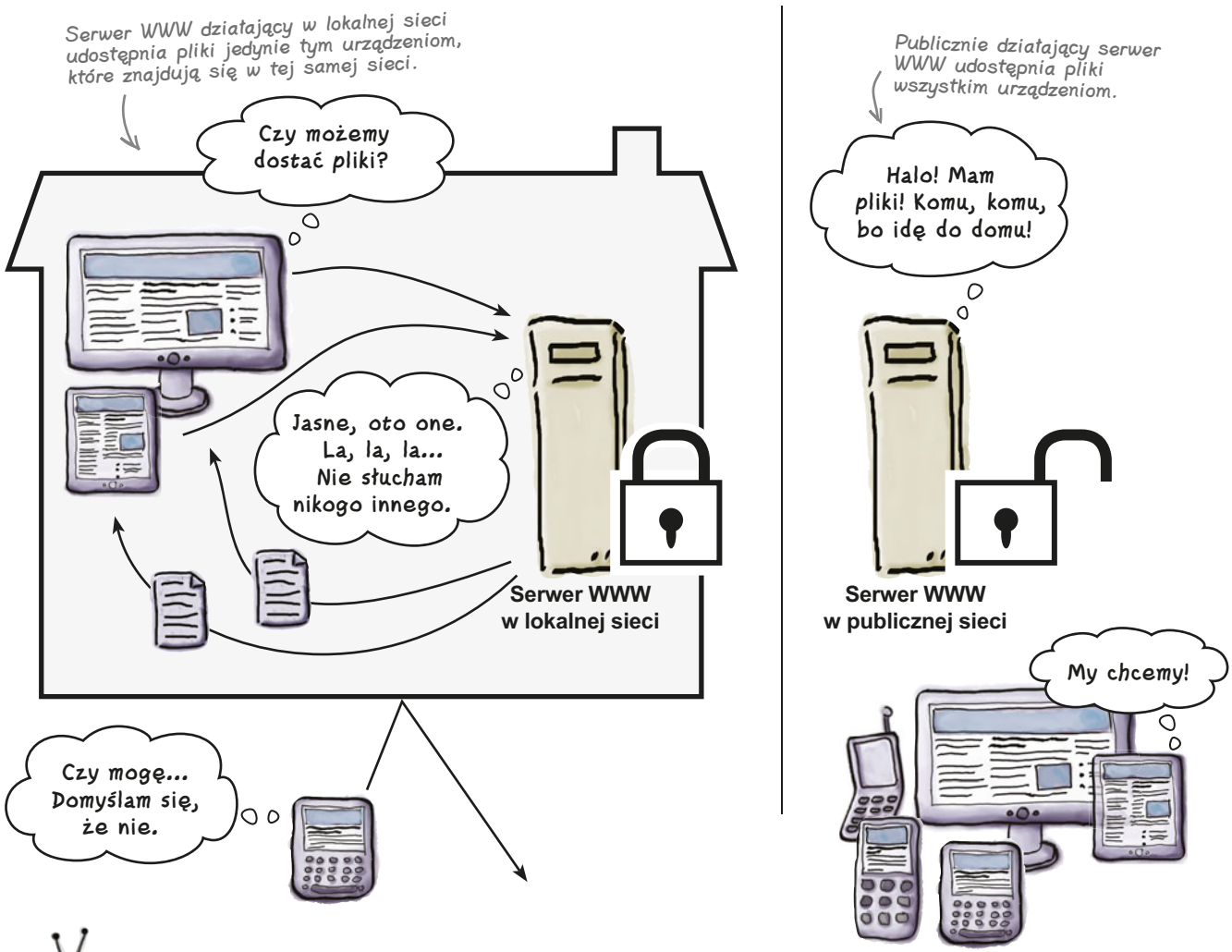


To naprawdę nie jest takie trudne (a bardzo użyteczne). Jeżeli jednak nie chcesz tego robić, *nie musisz*.

Weź jednak pod uwagę fakt, że Twój serwer musi spełnić pewne wymagania. PHP musi być w wersji co najmniej 5.2. Musisz mieć zainstalowane biblioteki GD oraz SimpleXML. Nie wiesz, czy masz? Przejdź na stronę 396, by zobaczyć, jak skorzystać z funkcji `phpinfo` w celu wyświetlenia informacji o środowisku PHP.

Dostępny tylko lokalnie

Pokażemy Ci krok po kroku, jak postawić serwer WWW na własnym komputerze.



Zintegrowane pakiety oprogramowania WWW, które będziemy instalować, nie nadają się do pracy na serwerach dostępnych publicznie.

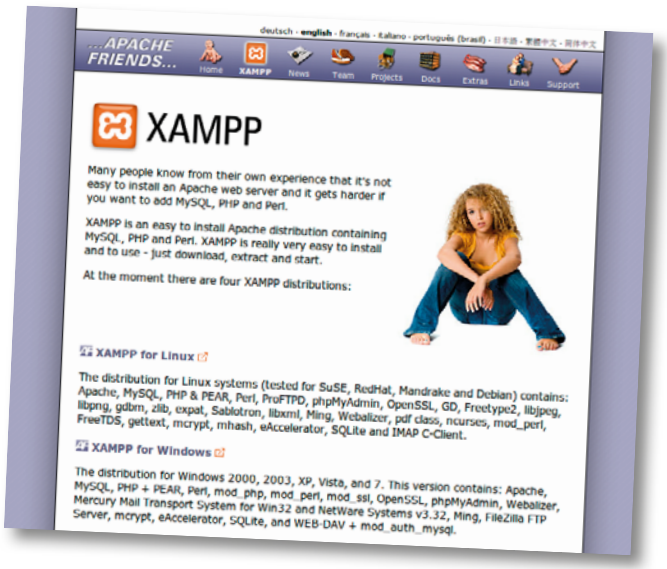
Ponieważ w przypadku tych narzędzi głównym założeniem jest maksymalne uproszczenie procesu stawiania i konfigurowania serwera, nie zostały zachowane wszystkie aspekty bezpieczeństwa. Z tego względu nie powinno się ich używać na publicznych serwerach. Bądź ostrożny!

Windows i Linux — zainstaluj i skonfiguruj XAMPP-a

Pobierz pakiet kompletnego serwera WWW

Użyjemy **XAMPP**-a, czyli prostego w użyciu pakietu zawierającego między innymi **serwer Apache** oraz **PHP**, czyli to, czego potrzebujemy. W pakiecie znajduje się jeszcze MySQL — system zarządzania relacyjną bazą danych (RDBMS), oraz Perl, który również niekiedy może się przydać (chociaż nie korzystamy z niego w tej książce).

Przejdź na stronę *www.apachefriends.org/en/xampp.html* i kliknij odsyłacz do odpowiedniej wersji systemu operacyjnego (Windows lub Linux). Pobierz najnowszą wersję.

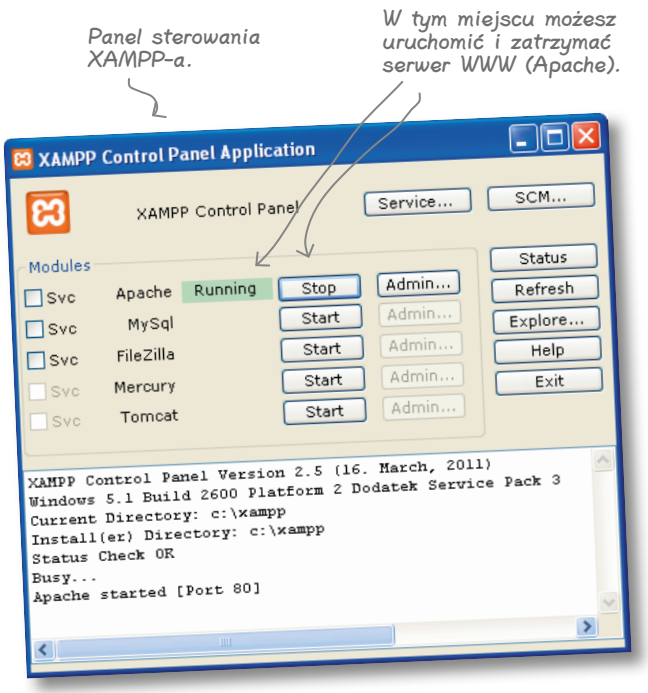


Na początek Windows

Uruchom pobrany plik. Zostanie otwarte okno instalatora, w którym musisz wskazać katalog instalacji. Domyślnie jest to *c:\xampp* — i tak możesz zostawić. Następnie postępuj zgodnie z instrukcjami instalatora.

Po zainstalowaniu możesz uruchomić serwer, wybierając z menu *Start/Programy* pozycję *XAMPP*.

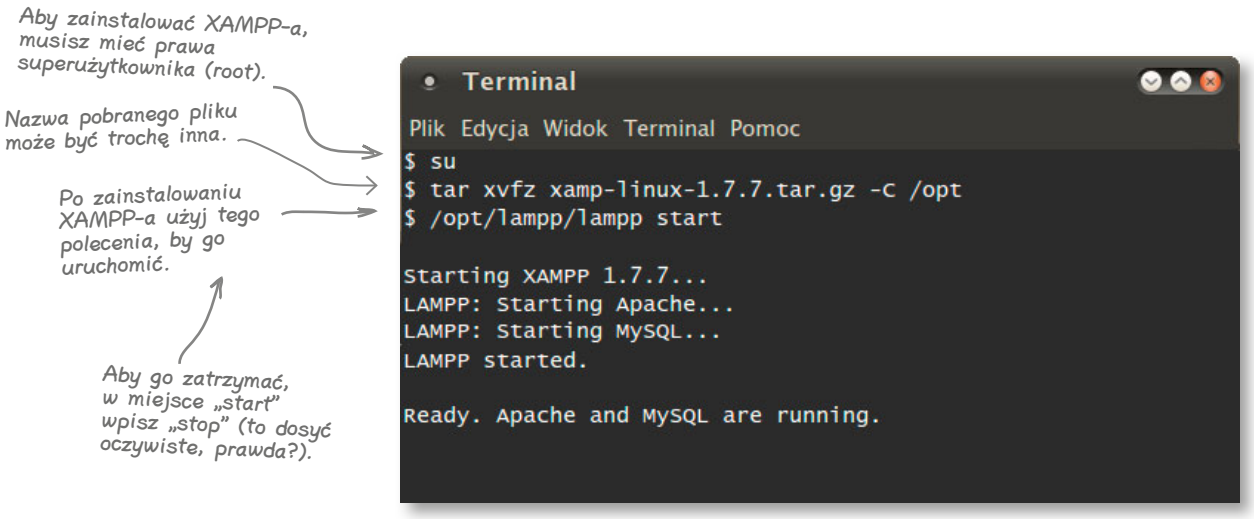
Nie możesz jej znaleźć? Sprawdź w sekcji Wszystkie programy menu Start.



Ciąg dalszy XAMPP-a

Teraz kolej na Linuksa

Po pobraniu paczki z oprogramowaniem otwórz terminal i wykonaj poniższe polecenia.



W internecie znajdziesz mnóstwo dodatkowych informacji

Jeśli chcesz poznać więcej poleceń, opcji konfiguracyjnych, ogólnych informacji oraz porad, zajrzyj do dokumentacji na stronie *www.apachefriends.org/en/xampp-windows.html* (jeżeli pracujesz na Windowsie) lub *www.apachefriends.org/en/xampp-linux.html* (jeżeli wolisz Linuksa).

Na koniec Mac — MAMP

Dla Maca proponujemy bardzo prosty i intuicyjny serwer MAMP.
Jego instalacja jest niezwykle prosta.

Nam w zupełności wystarczy zwykła wersja MAMP-a.

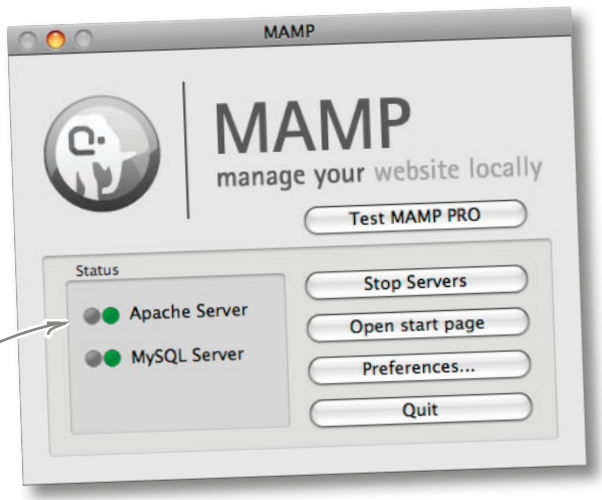
MAMP? Bułka z masłem...

Przejdź na stronę *http://mamp.info* i pobierz pakiet oprogramowania. Dwukrotnie kliknij, by zamontować pobrany obraz dysku i przeciągnij aplikację MAMP do folderu aplikacji.

Po uruchomieniu MAMP-a będziesz mógł uruchamiać i zatrzymywać serwery Apache oraz MySQL. W naszym przypadku istotny jest tylko Apache. Musisz go włączyć, by pomyślnie uruchomić i przetestować przykłady z tej książki.

Panel sterowania MAMP-a pozwala na uruchamianie i zatrzymywanie usług.

Jeśli jeszcze tego nie zrobiłeś, uruchom serwer Apache.



Sprawdź, czy zadokowałeś we właściwym porcie

Na Macu...

W najnowszych wersjach systemu Mac OS X serwer Apache jest już zainstalowany. Działa na porcie 80, czyli domyślnym porcie protokołu HTTP. Jeśli serwer działa, jest dostępny pod adresem *http://localhost*. Tak jak wspomnieliśmy, port 80 jest domyślny, więc wpisanie adresów *http://localhost* i *http://localhost:80* daje ten sam efekt (czyli dopisywanie portu nie jest potrzebne).

Serwer WWW na Macu jest sterowany opcją Web Sharing znajdującą się w panelu System Preferences/Sharing.

Zainstalowaliśmy MAMP-a, gdyż potrzebujemy PHP. Ponieważ większości Maców ma serwer WWW, MAMP działa domyślnie na porcie 8888. To oznacza, że musisz używać adresu *http://localhost:8888*. Jeśli chciałbyś zmienić port w MAMP-ie, zrobisz to bez problemu w ustawieniach.

...i w Windowsie

XAMPP domyślnie działa na porcie 80. To oznacza, że jeśli miałeś już wcześniej zainstalowany jakiś serwer zajmujący ten port, pojawią się problemy. W przypadku Windowsa musisz zatrzymać serwer Windows IIS (jeżeli jest zainstalowany i działa).

1. Otwórz Menedżer IIS. Aby to zrobić, najwygodniej jest otworzyć menu *Start* i w polu wyszukiwania wpisać `IIS`. Pojawi się pozycja *Menedżer internetowych usług informacyjnych (IIS)*, którą należy kliknąć.

2. Prawym przyciskiem myszy kliknij domyślą witrynę sieci web i z menu *Akcja* wybierz polecenie *Zatrzymaj*.

Równie dobrze możesz skonfigurować serwer Apache tak, by działał na innym porcie.

Dostań się do swojego serwera

Całkiem możliwe, że będziesz chciał przetestować niektóre przykłady z tej książki na prawdziwych urządzeniach mobilnych, korzystając ze swojego komputera jako serwera WWW. To powinno się udać, nawet jeśli serwer jest dostępny tylko lokalnie. Sprawdź, czy jesteś na to gotowy.

Musisz poznać adres swojego *komputera*, a nie *zewnętrzny* adres IP.

Ponieważ w przypadku tych narzędzi głównym założeniem jest maksymalne uproszczenie procesu stawiania i konfigurowania serwera, nie zostały zachowane wszystkie aspekty bezpieczeństwa. Z tego względu nie powinno się ich używać na publicznych serwerach. Bądź ostrożny!

1. **Zdobądź aktualny adres IP swojego komputera.**
Adres `localhost` jest względny. Odwołuje się do tego komputera, z którego aktualnie korzystasz. Zatem jeśli będziesz chciał się dostać do `localhost` z innego urządzenia (mobilnego lub jakiegokolwiek innego), będziesz się odwoływał do tego samego urządzenia.

   Aby poznać aktualny adres IP komputera w systemach Mac i Linux, w terminalu uruchom polecenie `ifconfig`. Użytkownicy Windowsa muszą w wierszu poleceń uruchomić polecenie `ipconfig` (wiersz poleceń najłatwiej włączyć, wpisując `cmd` w polu wyszukiwania w menu *Start*).

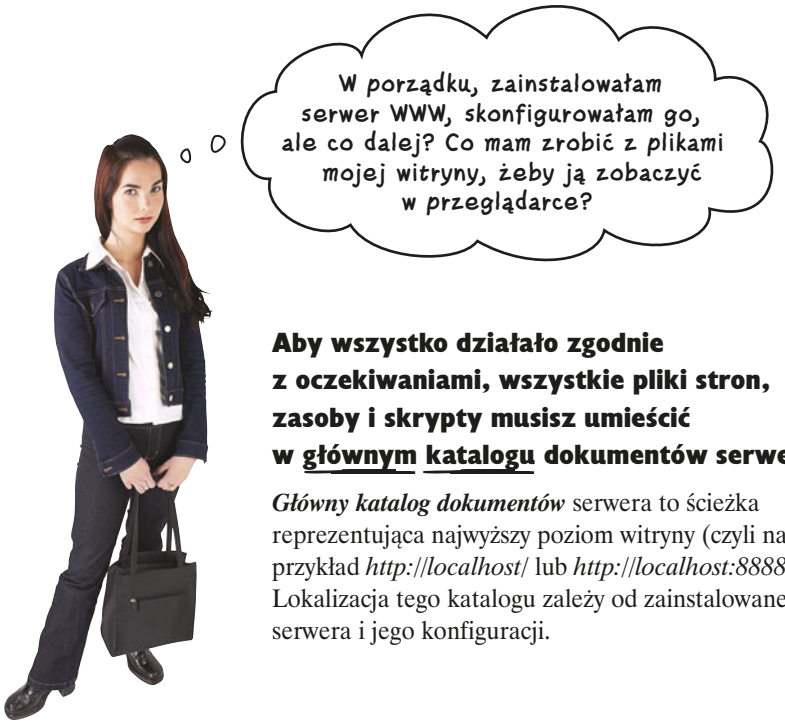
   Narzędzie `ifconfig` (lub `ipconfig`) wyświetla informacje o każdym interfejsie sieciowym komputera. Na liście szukaj interfejsów rozpoczynających się od `eth` lub `en` (Linux i Mac) lub `IP4` (Windows), ponieważ to one odpowiadają połączeniom ethernetowym. Kiedy już taki zlokalizujesz, znajdziesz bieżący adres IP.

   Wewnętrzne adresy IP zwykle rozpoczynają się od 10, 127 lub 192.

2. **Sprawdź, czy urządzenie mobilne znajduje się w tej samej sieci.**
Pamiętaj, że wewnętrzny adres IP jest dostępny tylko w ramach tej samej sieci. Jeśli z urządzenia mobilnego chcesz uzyskać dostęp do lokalnego serwera WWW, musisz się przyłączyć (najprawdopodobniej za pomocą WiFi) do tej samej sieci co serwer.

   Nie zapomnij dopisać numeru portu, jeśli nie jest domyślny, czyli na przykład http://10.0.0.2:8888.

Pracujesz w systemie Windows i nadal masz problemy z serwerem XAMPP? Poczytaj posty na Apache Friends Support Forum, które pomogą Ci je obejść (*http://bit.ly/sG3Qa0*).



Aby wszystko działało zgodnie z oczekiwaniami, wszystkie pliki stron, zasoby i skrypty musisz umieścić w głównym katalogu dokumentów serwera.

Główny katalog dokumentów serwera to ścieżka reprezentująca najwyższy poziom witryny (czyli na przykład *http://localhost/* lub *http://localhost:8888/*). Lokalizacja tego katalogu zależy od zainstalowanego serwera i jego konfiguracji.

Dojdź do korzeni

Użytkownicy Maca z zainstalowanym MAMP-em znajdą główny katalog dokumentów w */Applications/MAMP/htdocs*. Możesz zostawić tę lokalizację lub ją zmienić z poziomu okna preferencji serwera MAMP.

W przypadku Windowsa i serwera XAMPP katalog znajduje się w lokalizacji *\xampp\htdocs*, a dla Linuksa jest to */opt/xampp/htdocs*.

Wszystkie pliki i katalogi, do których ma uzyskać dostęp serwer WWW, powinny się znaleźć w głównym katalogu dokumentów.

Informacje od phpinfo

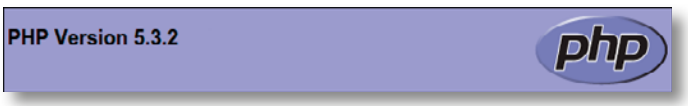
Aby poznać wersję PHP, różne opcje, zainstalowane dodatki i masę innych spraw, utwórz plik *test.php* i zapisz go w głównym katalogu dokumentów, tak by był dostępny pod adresem *http://localhost/test.php*.

W pliku umieść tylko jeden wiersz kodu:

```
<?php phpinfo(); ?>
```

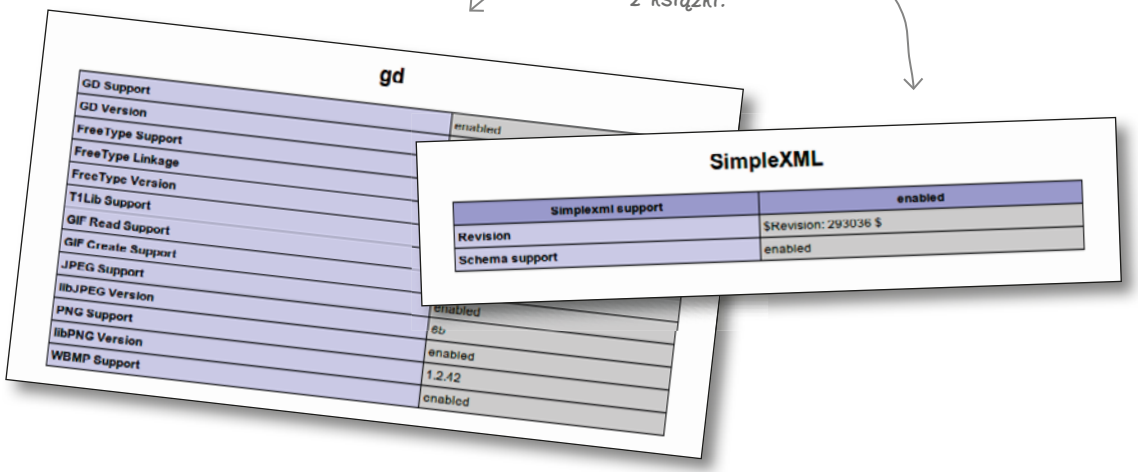
Nie zapomnij dopisać numeru portu, jeśli nie jest domyślny, czyli na przykład http://localhost:8888.

Zapisz plik i wyświetl go w przeglądarce, wpisując adres *http://localhost/test.php*. W ten sposób sprawdzisz, czy PHP działa, i zdobędziesz wiele cennych informacji na jego temat.



W sekcji na samej górze strony znajdziesz numer wersji PHP.

Przewiń stronę w dół i znajdź sekcje dotyczące gd i SimpleXML. Te dwa dodatki będą Ci potrzebne, by uruchomić przykłady z książki.



Dodatek C Instalowanie WURFL

Jak wywęszyć urządzenia?

Pierwszy krok do rozwiązania tajemnicy wykrywania urządzeń wymaga trochę zachodu. Każdy przyzwoity gliniarz wie, że trzeba zbierać poszlaki i przesłuchiwać świadków. Musimy zacząć od mózgu całej operacji, którym jest **WURLF API dla PHP**. Później przyjdzie kolej na mięśniaka — jeden **plik XML** zawierający informacje o możliwościach tysięcy urządzeń. Nie będzie jednak łatwo zmusić ich do współpracy, więc będziemy musieli poświęcić im trochę czasu i pogrzebać w **konfiguracji**.

Skąd wziąć mózg?

Musisz zmusić do współpracy WURFL API i jego dane. Oto nasz plan.

- ☐ Pobierz i zainstaluj WURFL API dla PHP.
- ☐ Pobierz i zainstaluj dane WURFL o urządzeniach zapisane w pliku XML.
- ☐ Wprowadź drobne zmiany w konfiguracji.

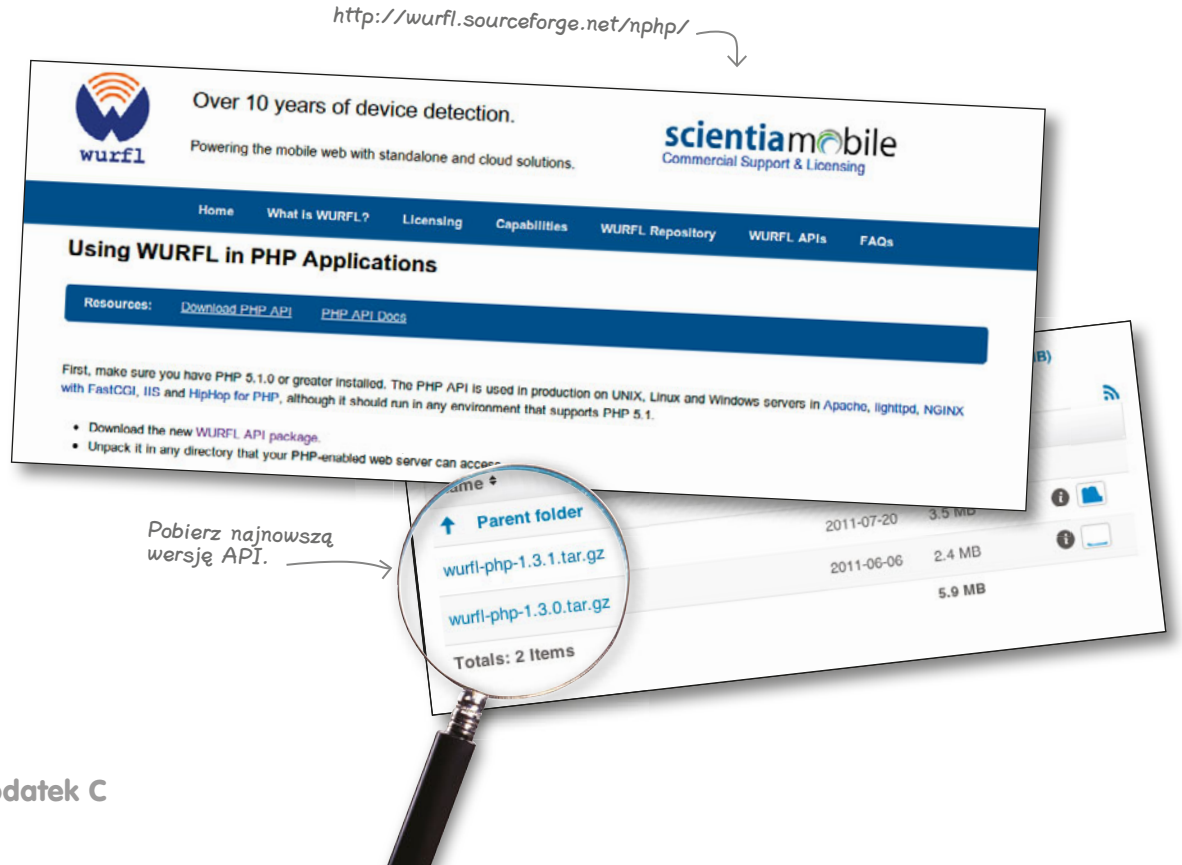
Pobierz API

Przejdź na stronę projektu WURFL API dla PHP znajdującą się pod adresem *http://wurfl.sourceforge.net/nphp/*. Odszukaj odsyłacz do paczki WURFL API i pobierz jego najnowszą wersję.

To jest interfejs API dla PHP. Rozpakuj pobrany plik i umieść jego zawartość (pojedynczy katalog) w wybranej lokalizacji na dysku. Nie jest ważne, gdzie się znajdzie, o ile będziesz pamiętał lokalizację.

WURFL (ang. Wireless Universal Resource FiLe) to plik XML z danymi na temat możliwości oferowanych przez mobilne urządzenia i ich przeglądarki. Dostępne są również interfejsy API umożliwiające korzystanie z tych danych. Tego też będziemy potrzebować!

Powtórz to z nami: „WURRR-full".



A co z mięśniakiem?

Interfejs API dla PHP jest całkiem sprytny, ale jest nic niewart bez swojego napakowanego pomocnika, czyli samych danych WURFL.

Aby pobrać najnowszą wersję pliku XML z danymi WURFL, przejdź na stronę *http://wurfl.sourceforge.net/wurfl_download.php*. Zanim pobierzesz plik, powinieneś przeczytać i zaakceptować umowę licencyjną.

Po pobraniu pliku rozpakuj go, zmień nazwę pliku wynikowego na *wurfl.xml* i przenieś go do katalogu *examples/resources/* znajdującego się w lokalizacji WURFL API dla PHP (katalog już powinien istnieć).

Pobrany plik będzie miał w nazwie najprawdopodobniej numer wersji. Najwygodniej będzie zmienić tę nazwę na wurfl.xml.

Jak zmusić tę dwójkę do współpracy?

Cały czas idziemy do przodu. Teraz przyszła kolej na wprowadzenie drobnych modyfikacji w pliku konfiguracyjnym, tak by API i dane zaczęły ze sobą współpracować.

Otwórz w edytorze plik *examples/resources/wurfl-config.xml* znajdujący się w lokalizacji WURFL API. Musisz w nim zmienić jedną linię.

- ☑ Pobierz i zainstaluj WURFL API dla PHP.
- ☑ Pobierz i zainstaluj dane WURFL o urządzeniach zapisane w pliku XML.
- ☐ **Wprowadź drobne zmiany w konfiguracji.**

Zmień to...

```
<wurfl-config>
    <wurfl>
        <main-file>wurfl.zip</main-file>
        <patches>
            <patch>web_browsers_patch.xml</patch>
        </patches>
    </wurfl>
```

wurfl-config.xml

...na to:

```
<wurfl-config>
    <wurfl>
        <main-file>wurfl.xml</main-file>
        <patches>
            <patch>web_browsers_patch.xml</patch>
        </patches>
    </wurfl>
```

wurfl-config.xml

Dzięki temu interfejs API będzie wiedział, gdzie może znaleźć dane.

Czas na porządki w systemie plików

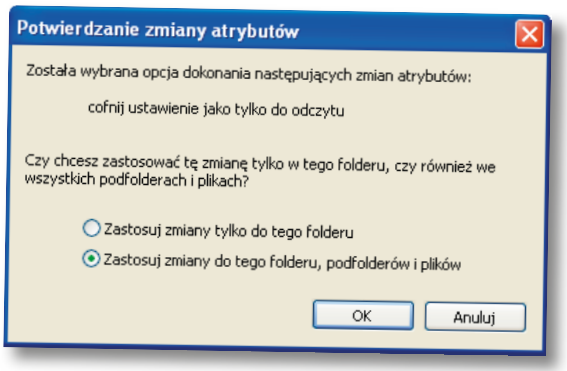
Teraz sprawdzimy, czy WURFL ma dostęp do wszystkich zasobów. W domyślnej konfiguracji jest stosowane **cache'owanie plików**. Musimy więc zadbać o to, by system plików był na to gotowy.

Przede wszystkim musisz sprawdzić, czy w lokalizacji WURFL API znajduje się katalog *examples/resources/storage* i czy zawiera dwa podkatalogi: *cache* i *persistence*.

Jeśli nie ma tych katalogów, utwórz je sam.

Aby WURFL działał prawidłowo, serwer WWW musi mieć prawo do zapisu w tych katalogach. Ten warunek jest spełniony w przypadku MAMP-a, ponieważ serwer Apache działa domyślnie z prawami zalogowanego użytkownika. Jeśli więc sam masz prawo zapisu do tych katalogów, Apache również je ma.

Nie dotyczy to jednak systemów Windows i Linux. W **Windowsie** musisz w oknie eksploratora plików kliknąć prawym przyciskiem myszy folder *storage* i wyłączyć zaznaczenie pola *Tylko do odczytu* (jeżeli jest zaznaczone). Pojawi się wówczas okno podobne do przedstawionego obok, w którym musisz potwierdzić, że chcesz wprowadzić zmiany do tego folderu, jego podfolderów i plików.



Spokojnie

Teraz wszystko już powinno być gotowe do pracy. Jeśli jednak napotkasz jakieś problemy, podążaj za wskazówkami wyświetlanymi przez PHP.

Jeśli podczas porządków w systemie plików nie pozbyłeś się całego bałaganu albo coś nie gra w konfiguracji WURFL, możesz skorzystać z informacji, ostrzeżeń i błędów generowanych przez PHP.

Biały ekran w przeglądarce oznacza najczęściej, że jest wyłączone zgłaszanie błędów w PHP. Aby je włączyć, utwórz plik *.htaccess* (jeśli jeszcze go nie ma) w tym samym katalogu co problematyczny plik PHP i wpisz w nim poniższą linię:

```
php_flag display_errors on
```

(Użytkownicy Linuksa mogą użyć poleceń *chmod* i *chown*).

Zwróć na to uwagę!

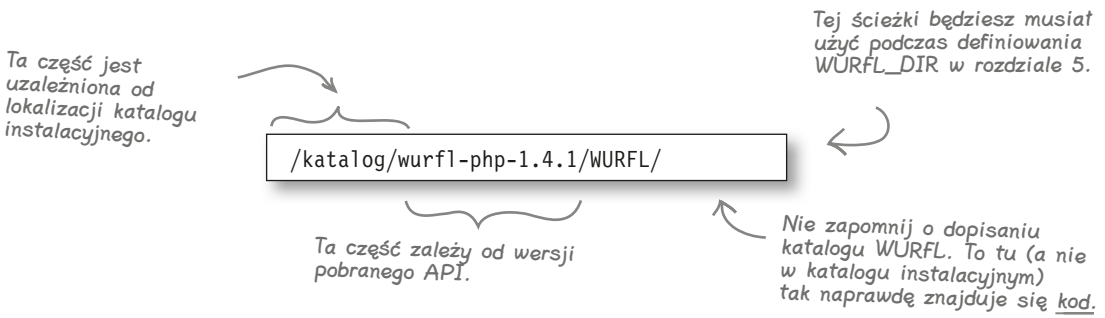
Aby rozpocząć pracę z projektem z rozdziału 5., musisz **zwrócić uwagę na kilka ścieżek**.

Ścieżki dla systemów Mac i Linux wyglądają jak w zaprezentowanych przykładach. W ścieżkach dla Windowsa musisz zastosować lewe ukośniki.

Ścieżka do kodu WURFL API dla PHP

Najważniejsza jest pełna ścieżka do lokalizacji, w której znajduje się kod WURFL API dla PHP. Jest on umieszczony w podkatalogu *WURFL* wewnątrz katalogu instalacyjnego.

Jeśli masz na przykład wersję 1.4.1 API i zainstalowałeś go w katalogu, powiedzmy, */katalog*, ścieżka będzie wyglądała tak:



Ścieżka do zasobów

Kolejna sprawa to pełna ścieżka do podkatalogu *examples/resources/* w katalogu instalacyjnym WURFL API dla PHP. Może wyglądać tak:

Sprawa zamknięta!

- ☑ Pobierz i zainstaluj WURFL API dla PHP.
- ☑ Pobierz i zainstaluj dane WURFL o urządzeniach zapisane w pliku XML.
- ☑ Wprowadź drobne zmiany w konfiguracji.

Dodatek D Instalowanie SDK i narzędzi dla Androida

Zadbaj o środowisko

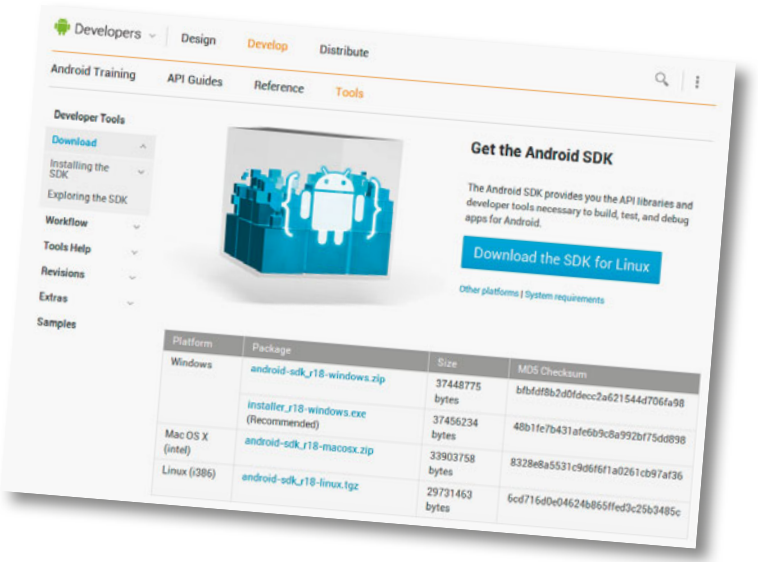
Aby gruntownie testować natywne aplikacje pod Androida, musisz przygotować sobie odpowiednie środowisko. Musisz dołączyć swój komputer do małego ekosystemu, do którego zagonisz wszystkie aplikacje Androida uruchamiane zarówno na wirtualnych (emulowanych), jak i rzeczywistych urządzeniach. Abyś mógł stać się pasterzem tego stada, pokażemy Ci, jak pobrać **SDK (ang. *Software Development Kit*) Androida**, jak zainstalować niektóre **narzędzia dla tej platformy**, jak **utworzyć wirtualne urządzenia** i jak **instalować oraz odinstalowywać aplikacje**.

Pobieramy SDK Androida

Zrób to sam!

Przejdź na stronę *http://developer.android.com/sdk* i znajdź odsyłacz odpowiedni dla swojego systemu operacyjnego.

http://developer.android.com/sdk



Pobierz plik i go rozpakuj. Wynikowy katalog z SDK umieść tam, gdzie na dysku przechowujesz aplikacje.

Użytkownicy Windowsa mogą pobrać wygodny instalator, który ułatwia ten proces.

Dobra robota! Udało Ci się samodzielnie zainstalować SDK dla Androida (nie było tak źle, prawda?).

Teraz do SDK dodamy kilka pakietów i narzędzi, które pomogą nam w pracy.

Udoskonalamy środowisko SDK dla Androida

- [x] Pobierz i zainstaluj SDK dla Androida odpowiednie dla swojego systemu operacyjnego.

To już zrobiliśmy.

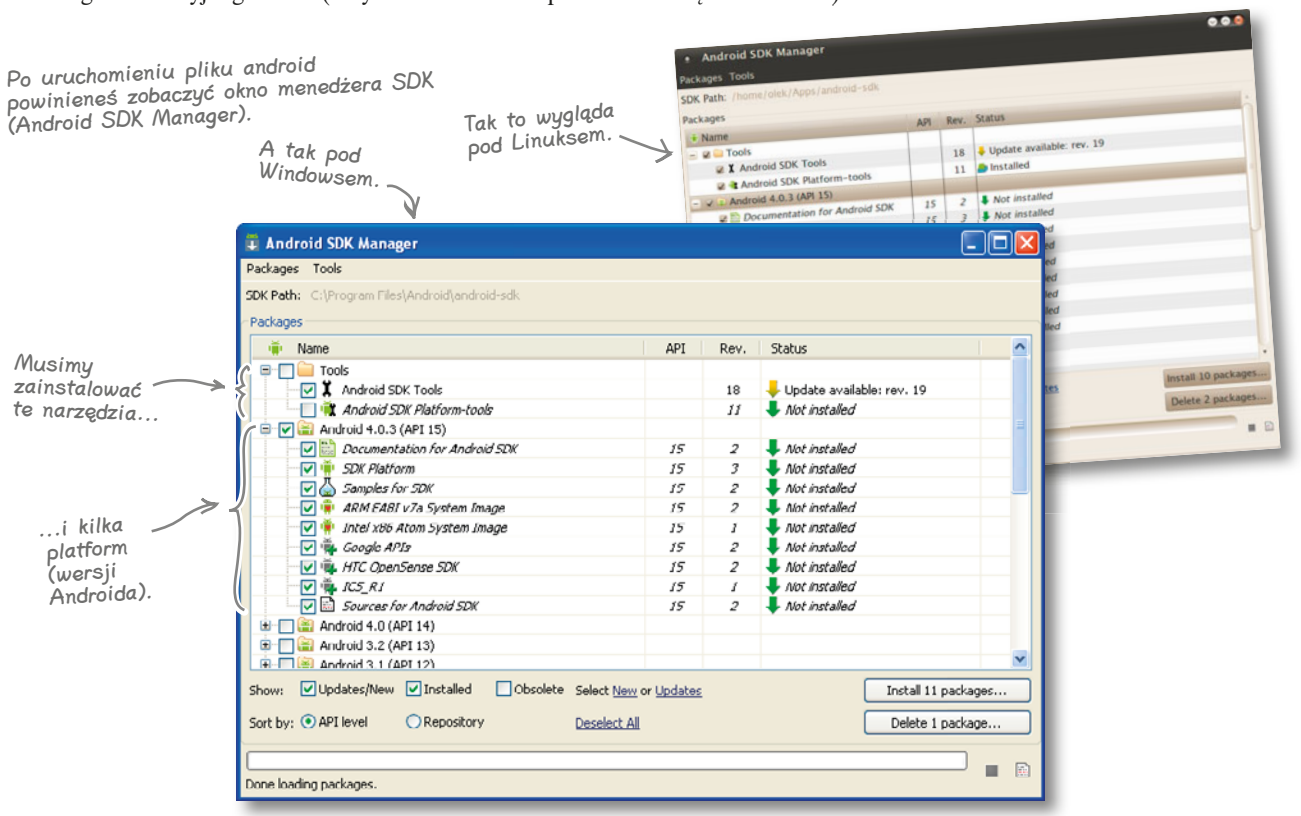
- [ ] **Zainstaluj kilka narzędzi i platform Androida (różnych wersji API).**
- [ ] Utwórz kilka wirtualnych urządzeń (AVD). Pełnią one rolę emulatorów, na których można uruchomić aplikacje.
- [ ] Dowiedz się, jak zainstalować i odinstalować aplikacje na emulatorach i w urządzeniach.
- [ ] Skonfiguruj ścieżki systemowe (zmienna `PATH`), by uruchamianie narzędzi z SDK było wygodniejsze.

To nie jest wymagane, ale bardzo ułatwia pracę.

Wybierz odpowiednie narzędzia

Zainstalowanie SDK było prostym zadaniem, ale to nie koniec prac przygotowawczych. Musisz jeszcze doinstalować kilka platform Androida i narzędzi.

Odszukaj plik *android* i go uruchom. Powinien się znajdować w podkatalogu *tools* katalogu instalacyjnego SDK (w systemie Windows plik nosi nazwę *Android.bat*).



Instalowanie platform i narzędzi

Chcemy doinstalować kilka platform (czyli wersji API Androida) oraz narzędzi. Aby to zrobić, wystarczy zaznaczyć odpowiednie pola w menedżerze SDK. Po kolei:

1. **Zainstaluj dwie lub trzy ostatnie wersje Androida.**
   W chwili pisania książki najnowsza wersja Androida miała numer 4.0.3. Zainstalowaliśmy również wersje 2.2 oraz 2.3.3, by mieć możliwość przetestowania aplikacji na starszych wersjach platformy. Jeśli chcesz, możesz zainstalować ich jeszcze więcej.

2. **Zainstaluj pakiet „Tools".**
   Przede wszystkim jest nam potrzebne narzędzie `adb`, które jest częścią pakietu *Android SDK Platform Tools*. Zaznacz pole przy folderze *Tools*.

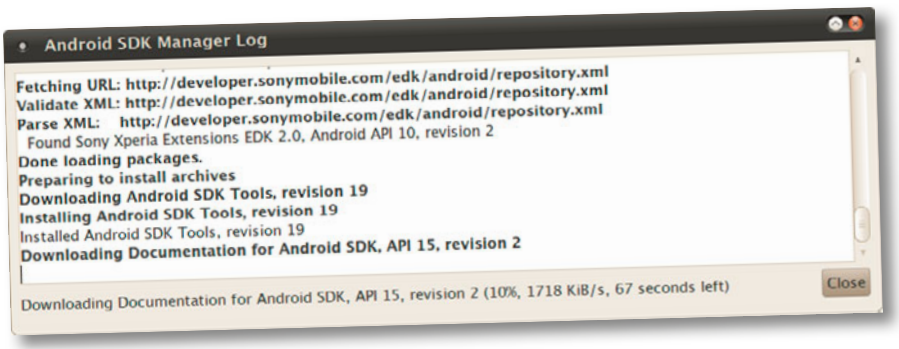
Nie wszystkie wersje Androida są dostępne w SDK.

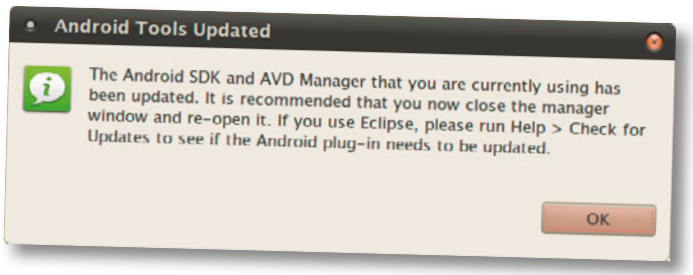
Android 3.0 (Honeycomb) jest wersją przeznaczoną tylko na tablety. Możesz ją zainstalować, ale mogą wystąpić problemy z emulatorem.

Kliknij przycisk Install i zrób sobie kawę

Po zaznaczeniu pożądanych platform i pakietu *Tools* kliknij przycisk *Install*, by rozpocząć instalację. Przygotuj się na to, że to chwilę zajmie — w zależności od systemu i zaznaczonych opcji może to trwać nawet 10 – 20 minut. Zrób sobie w tym czasie przerwę na kawę.

Najprawdopodobniej pojawi się okno wyglądające podobnie do tego, w którym wyświetlane są wszystkie komunikaty instalatora.





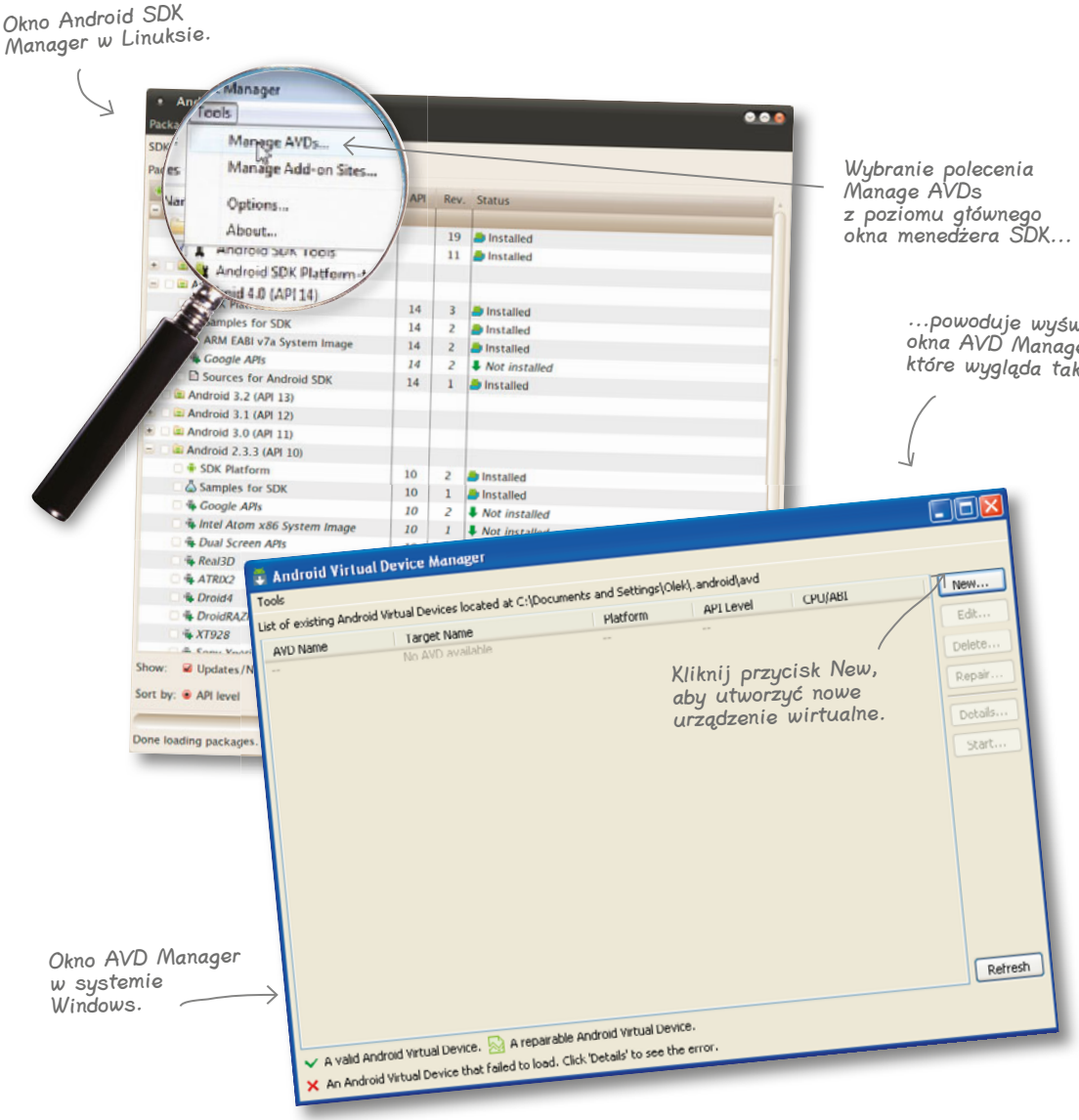
Po zakończeniu procesu instalowania może się pojawić komunikat podobny do tego. Wciśnij przycisk OK.

- [x] Pobierz i zainstaluj SDK dla Androida odpowiednie dla swojego systemu operacyjnego.
- [x] Zainstaluj kilka narzędzi i platform Androida (różnych wersji API).
- [ ] **Utwórz kilka wirtualnych urządzeń (AVD). Pełnią one rolę emulatorów, na których można uruchomić aplikacje.**
- [ ] Dowiedz się, jak zainstalować i odinstalować aplikacje na emulatorach i w urządzeniach.
- [ ] Skonfiguruj ścieżki systemowe (zmienna `PATH`), by uruchamianie narzędzi z SDK było wygodniejsze.

Musimy teraz utworzyć kilka wirtualnych urządzeń na potrzeby przyszłych testów.

Czy wirtualne urządzenia liczą elektroniczne owce?

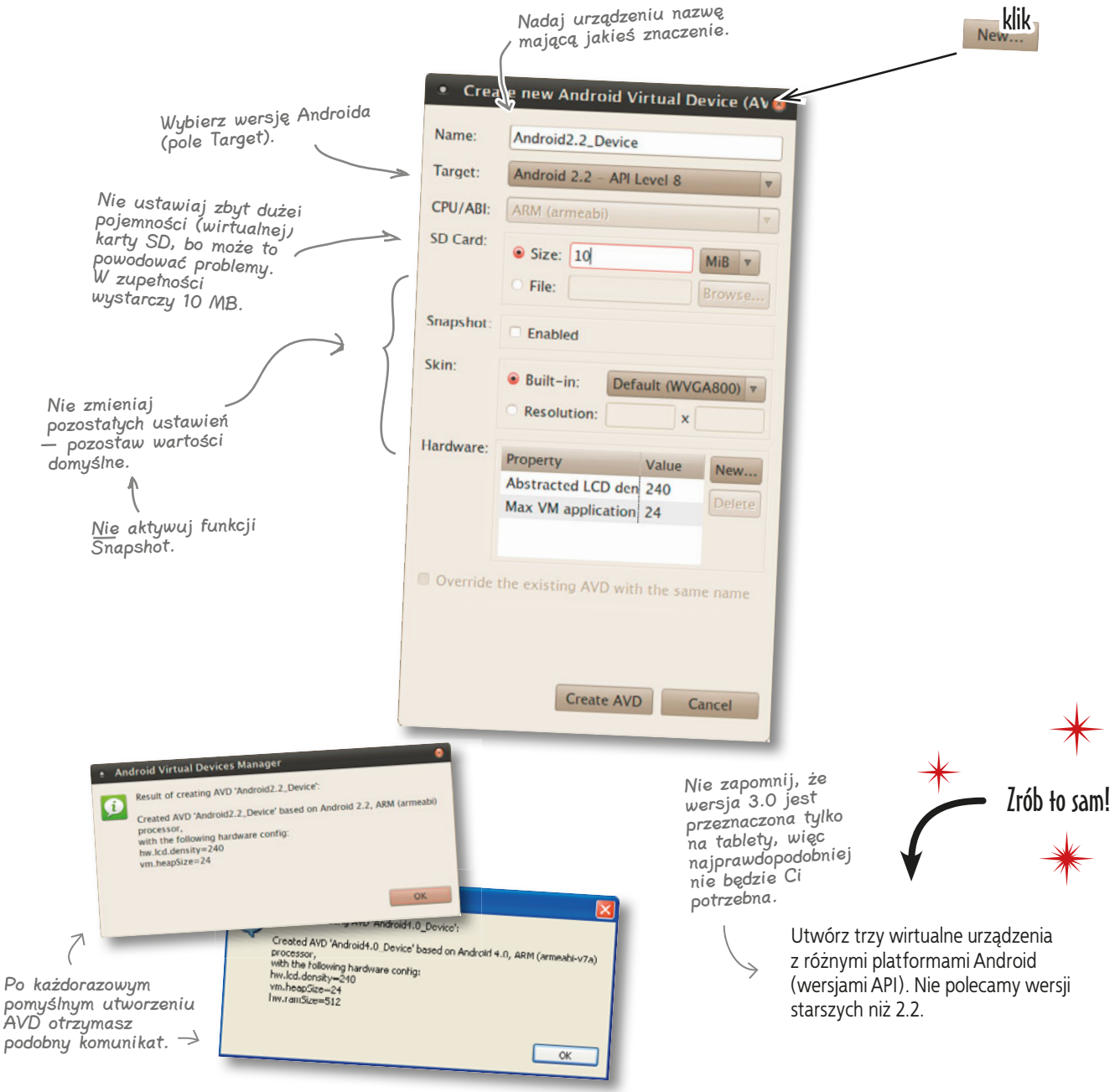
Nasz cel: utworzenie wirtualnych urządzeń. W oknie *Android SDK Manager* z menu *Tools* wybierz pozycję *Manage AVDs*.



Musimy dodać kilka urządzeń. Kliknij przycisk Start, by rozpocząć.

Tworzenie nowego urządzenia wirtualnego

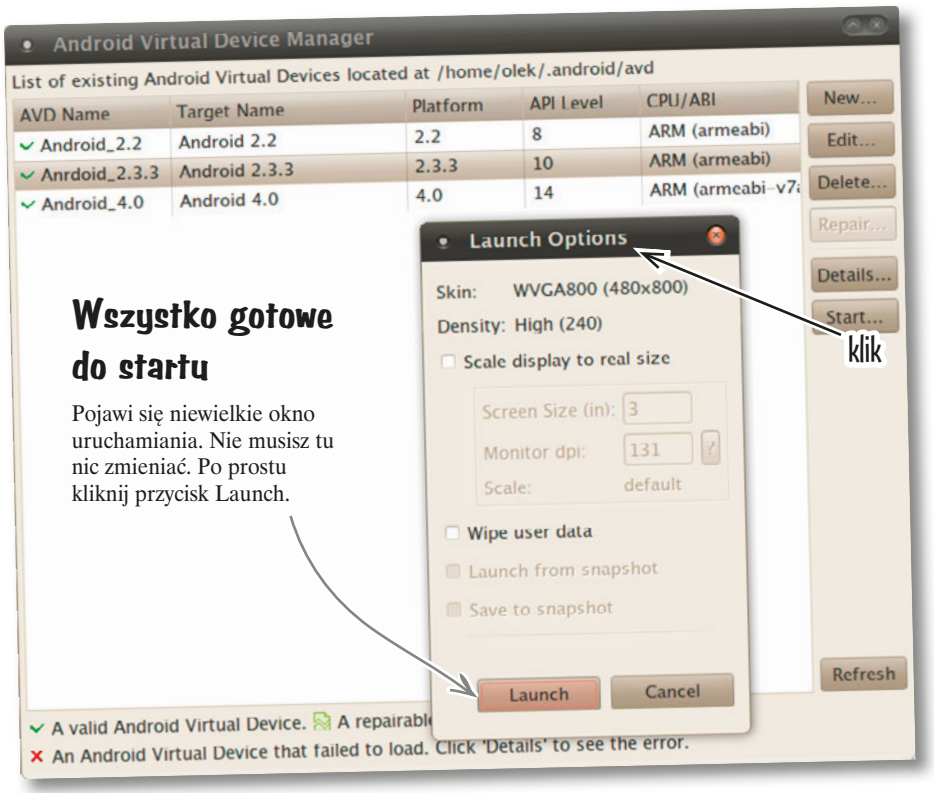
Po kliknięciu przycisku *New* w oknie menedżera AVD pojawi się okno podobne do przedstawionego niżej (ten zrzut pochodzi akurat z Linuksa). Zmień ustawienia zgodnie z opisem, a następnie kliknij przycisk *Create AVD*.



Odpalamy maszynę!

Po utworzeniu kilku urządzeń możesz je uruchomić z poziomu okna *Android Virtual Device Manager* (jak pamiętasz, można go włączyć poleceniem *Manage AVDs* z menu *Tools*).

Aby włączyć wybrane urządzenie, zaznacz je na liście i kliknij przycisk *Start*.



Włączanie wirtualnego urządzenia może trochę potrwać.

Uruchamianie się emulatora jest powolnym procesem, zwłaszcza za pierwszym razem. Spokojnie poczekaj i nie wpadaj w panikę, jeżeli Ci się wydaje, że emulator się zawiesił.

Jeśli jednak emulator nie chce się uruchomić lub po chwili się „wysypuje”, sprawdź, czy nie ustawiłeś zbyt dużej pojemności wirtualnej karty SD lub nie włączyłeś opcji *Snapshot*. Może też być tak, że akurat wybrana wersja platformy nie działa w Twoim systemie operacyjnym. Jeżeli wyczerpałeś już wszystkie możliwości, spróbuj uruchomić inną wersję API.

Wirtualne urządzenie z Androidem 2.3.3 uruchamiające się na Linuksie

Wirtualne urządzenie z Androidem 4.0 działające pod Windowsem

Instalowanie i odinstalowywanie aplikacji

- [x] Pobierz i zainstaluj SDK dla Androida odpowiednie dla swojego systemu operacyjnego.
- [x] Zainstaluj kilka narzędzi i platform Androida (różnych wersji API).
- [x] Utwórz kilka wirtualnych urządzeń (AVD). Pełnią one rolę emulatorów, na których można uruchomić aplikacje.
- [ ] **Dowiedz się, jak zainstalować i odinstalować aplikacje na emulatorach i w urządzeniach.**
- [ ] Skonfiguruj ścieżki systemowe (zmienna `PATH`), by uruchamianie narzędzi z SDK było wygodniejsze.

Pamiętasz, jak po zainstalowaniu SDK Androida doinstalowaliśmy pakiet *Tools*? W pakiecie znajduje się program `adb`, dzięki któremu możemy instalować i odinstalowywać aplikacje zarówno na wirtualnych, jak i rzeczywistych urządzeniach.

Proces instalowania i odinstalowywania aplikacji jest bardzo prosty:

1. Poczekaj na załadowanie wirtualnego urządzenia (AVD) **lub** podłącz urządzenie z Androidem do portu USB komputera.
2. Użyj polecenia `adb install` lub `adb uninstall`, aby — odpowiednio — zainstalować lub odinstalować pożądaną aplikację.

Skrót adb pochodzi od nazwy Android debug bridge. To narzędzie umożliwia instalowanie, odinstalowywanie i debugowanie aplikacji na emulatorach i urządzeniach.

Aby zainstalować

Plik z paczką aplikacji

```
$ adb install MojaAplikacja.APK
```

Aby odinstalować

```
$ adb uninstall com.foo.mojaAplikacja
```

Pakiet aplikacji

W aplikacjach pod Androida stosuje się taki sam sposób nazywania pakietów jak w Javie, który wygląda jak nazwa domenowa zapisana od końca.

Nie wiem, od czego zacząć. Gdzie znajdę plik APK, który mogłabym zainstalować? Gdzie mam wpisać polecenie adb install?

Trochę się pospieszyliśmy, przepraszamy!

Pokażemy Ci, gdzie znaleźć przykładową aplikację, ale najpierw musisz się dowiedzieć więcej na temat polecenia `adb install`.

Poszukiwania ścieżki — zmienna PATH

Jest całkiem możliwe, że narzędzia z SDK Androida, włączając w to `adb`, nie są uwzględnione w ścieżce środowiskowej `PATH`. W związku z tym, jeśli otworzysz terminal (lub wiersz polecenia w systemie Windows) i spróbujesz uruchomić polecenie `adb` bezpośrednio, najprawdopodobniej otrzymasz komunikat o błędzie. Twój system musi znać lokalizację narzędzia `adb`, by móc je uruchomić.

Sugerujemy zaktualizowanie zmiennej `PATH` poprzez dołączenie do niej ścieżki do katalogu z narzędziami SDK Androida, tak by uruchamianie ich z terminala było prostsze.

W systemie Windows są oddzielone średnikami.

Zmienna środowiskowa $PATH to lista oddzielonych dwukropkami ścieżek, w których system operacyjny może szukać plików wykonywalnych wywoływanych z terminala.

Zwykle zmienna ta zawiera wiele ścieżek, ale akurat nie te, które wskazywałyby lokalizację SDK Androida.

Jeśli nie umiesz sobie poradzić ze zmienną $PATH, masz inne możliwości.

W chwili wywoływania polecenia możesz podać pełną ścieżkę (na przykład */Applications/android-sdk/platform-tools/sdk/adb*) lub zmienić katalog na *platform-tools* i uruchomić polecenie *./adb* (./ jest konieczne).

Znajdź właściwą ścieżkę

Użytkownicy Maca

W terminalu wpisz poniższe polecenia. Aby zmiany zostały uwzględnione, musisz **zamknąć okno terminala** i uruchomić je ponownie.

Pierwsze polecenie przenosi Cię do katalogu użytkownika.

Ten fragment wpisz dokładnie. Zwróć uwagę na dwukropek (:) i nawiasy klamrowe ({}).

Nie zapomnij w tym miejscu o apostrofie (musisz go umieścić też przed słowem eksport).

```
Plik  Edycja  Widok  Termina  Pomoc
$ cd ~
$ echo 'export PATH=${PATH}:/Applications/.../platform-tools' >> .profile
```

Drugie polecenie dołącza ścieżkę do katalogu z narzędziami SDK Androida do zmiennej środowiskowej $PATH zapisanej w profilu użytkownika.

Ustawienia z pliku .profile są uwzględniane za każdym razem, gdy jest otwierane okno terminala.

W tym miejscu musisz podać prawidłową ścieżkę do katalogu platform-tools SDK Androida w Twoim systemie.

Użytkownicy Windowsa

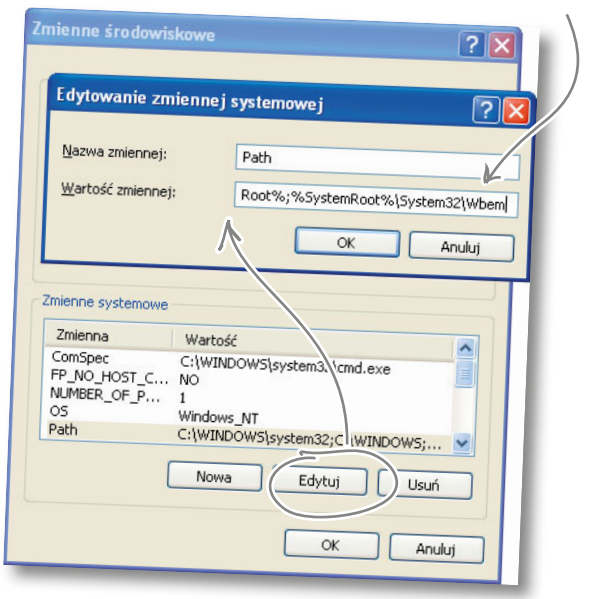
1. W menu *Start* kliknij prawym przyciskiem myszy pozycję *Komputer* i z podręcznego menu wybierz pozycję *Właściwości*. Przejdź na zakładkę *Zaawansowane*.

2. W zakładce *Zaawansowane* kliknij przycisk *Zmienne środowiskowe*. W sekcji *Zmienne systemowe* zaznacz pozycję *Path* i kliknij przycisk *Edytuj*.

3. Umieść kursor na samym **końcu** zawartości pola *Wartość zmiennej*, dodaj znak średnika (**;**) i wprowadź pełną ścieżkę folderu *platform-tools*. Po ścieżce nie musisz wstawiać średnika. Potwierdź zmianę przyciskiem *OK*.

4. Musisz się wylogować i ponownie zalogować, by wprowadzone zmiany zostały uwzględnione.

Ścieżkę do katalogu platform-tools dodaj na końcu bieżącej wartości zmiennej Path.



Użytkownikom Linuksa najprawdopodobniej nie trzeba mówić, jak zmodyfikować zmienną PATH.

Jazda próbna

W końcu nadszedł ten moment, na który od dawna czekaliśmy. Najpierw zainstalujemy testową aplikację na emulatorze, a później ją odinstalujemy.

1. Pobierz przykładową aplikację PhoneGap Build dla Androida ze strony *http://hf-mw.com/ch8/PhoneGap.apk*.
2. Włącz menedżer SDK (jeśli wcześniej tego nie zrobiłeś).
3. Uruchom AVD (czyli wirtualne urządzenie). W tym celu przejdź do menu *Tools*, wybierz polecenie *Manage AVDs*, zaznacz na liście pożądane urządzenie, a następnie je uruchom przyciskiem *Start*.
4. Poczekaj, aż wirtualne urządzenie się **w pełni** uruchomi.
5. Otwórz okno terminala lub wiersza poleceń. Zmień bieżący katalog na ten, w którym znajduje się pobrany plik APK.
6. Wykonaj poniższe polecenie:

Zakładamy, że ustawiłeś zmienną PATH tak, by zawierała ścieżkę do katalogu platform-tools. W przeciwnym przypadku musisz podać pełną ścieżkę do narzędzia adb.

```
$ adb install PhoneGap.apk
```

7. Po chwili na ekranie wirtualnego urządzenia powinieneś zobaczyć przykładową aplikację PhoneGap. Możesz ją uruchomić i chwilę się nią pobawić.
8. Aby odinstalować aplikację, z wiersza poleceń uruchom poniższe polecenie:

```
$adb uninstall com.phonegap.getting.started
```

To jest identyfikator pakietu przykładowej aplikacji PhoneGap.

Nie istnieją głupie pytania

P: Czym jest plik pakietu dla Androida?

O: Plik pakietu dla Androida (ang. *Android Package File*, APK) to plik zawierający natywną aplikację dla Androida. Znajduje się w nim skompilowany kod oraz wszystkie zasoby aplikacji.

Pliki APK są bardzo podobne do plików JAR (ang. *Java Archive File*). Jeśli jesteś ciekawy, co „siedzi" w takim pliku, zamień jego rozszerzenie na *.zip*, dzięki czemu będziesz mógł go rozpakować i zobaczyć zawartość.

P: Skąd się wzięło to *com.foo.mojaAplikacja*?

O: Pakiety z aplikacjami Androida są identyfikowane przez pełne nazwy pakietów (tak samo jak w Javie). Wygląda to trochę jak nazwa domenowa zapisana od końca.

P: Skąd mam wiedzieć, jaką pełną nazwę pakietową ma moja aplikacja?

O: W rozdziale 8. pokazaliśmy, jak należy zmodyfikować plik XML zawierający konfigurację dla PhoneGap Build. Wśród ustawień znajduje się również nazwa pakietowa aplikacji.

P: Skąd narzędzie adb wie, na jakim urządzeniu zainstalować aplikację?

O: Narzędzie `adb` instaluje aplikację na aktualnie uruchomionym urządzeniu (to samo dotyczy odinstalowania aplikacji). Jeśli masz prawdziwe urządzenie z Androidem, możesz je podłączyć do portu USB, a wtedy `adb` zainstaluje aplikację na tym urządzeniu.

P: Czy mogę odinstalować aplikację z poziomu wirtualnego lub rzeczywistego urządzenia?

O: Oczywiście. Możesz odinstalować aplikację dokładnie w ten sam sposób, w jaki robisz to z innymi aplikacjami. Przejdź do menu urządzenia i znajdź narzędzie do zarządzania aplikacjami, dzięki któremu usuniesz niechcianą aplikację.

P: Gdy próbuję zainstalować lub odinstalować aplikację, pojawia się komunikat o błędzie.

O: Przede wszystkim sprawdź, czy emulator się w pełni włączył. Możesz wykonać polecenie `adb devices`, aby wyświetlić listę urządzeń lub emulatorów, do których ma dostęp narzędzie `adb`. Jeśli urządzenie lub emulator działają poprawnie, powinieneś zobaczyć jedną pozycję na liście.

Podczas instalowania i odinstalowywania aplikacji nie możesz mieć równocześnie uruchomionych więcej urządzeń (prawdziwego lub wirtualnego), ponieważ narzędzie `adb` zawsze odwołuje się do bieżącego (z założenia jednego).

Nie możesz też zainstalować aplikacji, która już jest zainstalowana. Wcześniej musisz ją odinstalować.

P: Mam prawdziwe urządzenie z Androidem. Jak mam na nim instalować i odinstalowywać aplikacje?

O: Używasz tych samych poleceń, ale zamiast uruchamiać emulator, podłączasz swoje urządzenie do portu USB komputera. Po wykonaniu polecenia `adb device` zobaczysz je na liście.

Narzędzie adb ma całkiem sporo przydatnych funkcji pomagających w debugowaniu i analizowaniu działania aplikacji oraz urządzeń.

Więcej informacji na jego temat możesz znaleźć w pomocy wyświetlanej poleceniem `adb help`.

Gratulacje! Dałeś radę!

- ☑ Pobierz i zainstaluj SDK dla Androida odpowiednie dla swojego systemu operacyjnego.
- ☑ Zainstaluj kilka narzędzi i platform Androida (różnych wersji API).
- ☑ Utwórz kilka wirtualnych urządzeń (AVD). Pełnią one rolę emulatorów, na których można uruchomić aplikacje.
- ☑ Dowiedz się, jak zainstalować i odinstalować aplikacje na emulatorach i w urządzeniach.
- ☑ Skonfiguruj ścieżki systemowe (zmienna `PATH`), by uruchamianie narzędzi z SDK było wygodniejsze.

Skorowidz

D

E

F

G

N

O

P

X

Y

Z