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*

O'REILLY®

Lyza Danger Gardner, Jason Grigsby

Helion

Tytuł oryginału: Head First Mobile Web

Tłumaczenie: Aleksander Lamża

ISBN: 978-83-246-4864-1

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/mobweb
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

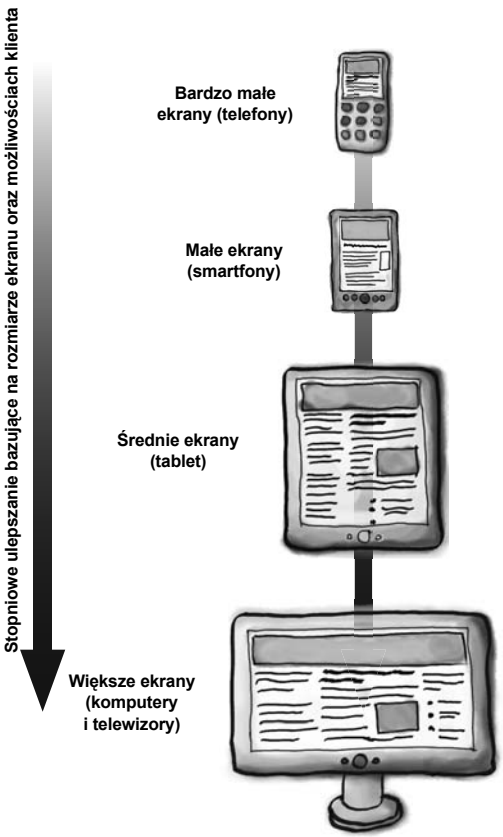
Wrażliwe projekty, czyli Responsive Web Design

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.

7

Mobilne aplikacje w prawdziwym świecie

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!

8 Tworzenie hybrydowych aplikacji mobilnych z PhoneGap

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!

9

Podejście „future friendly”

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

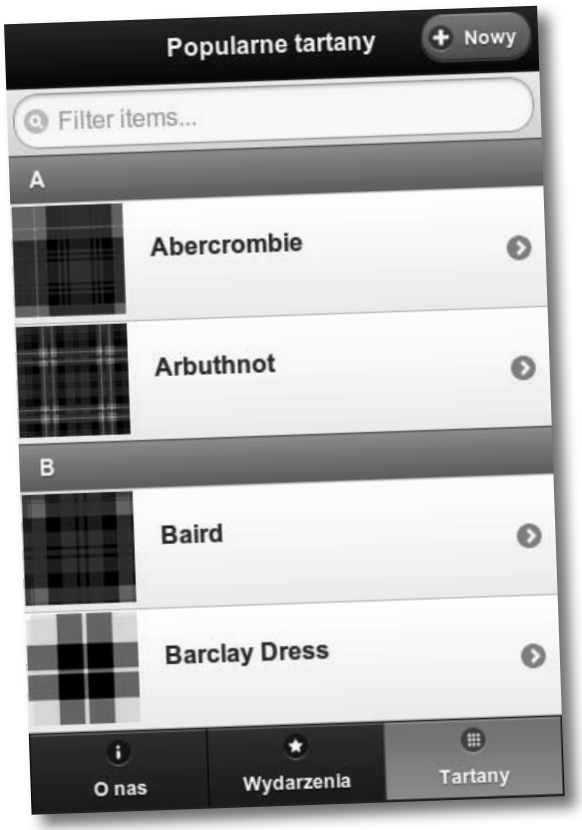
Aby gruntownie testować natywne aplikacje pod Androida, musisz przygotować sobie odpowiednie środowisko. Musisz dołączyć swój komputer do małego ekosystemu, do którego zagonisz wszystkie aplikacje Androida uruchamiane zarówno na wirtualnych (emulowanych), jak i rzeczywistych urządzeniach. Abyś mógł stać 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**.

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!

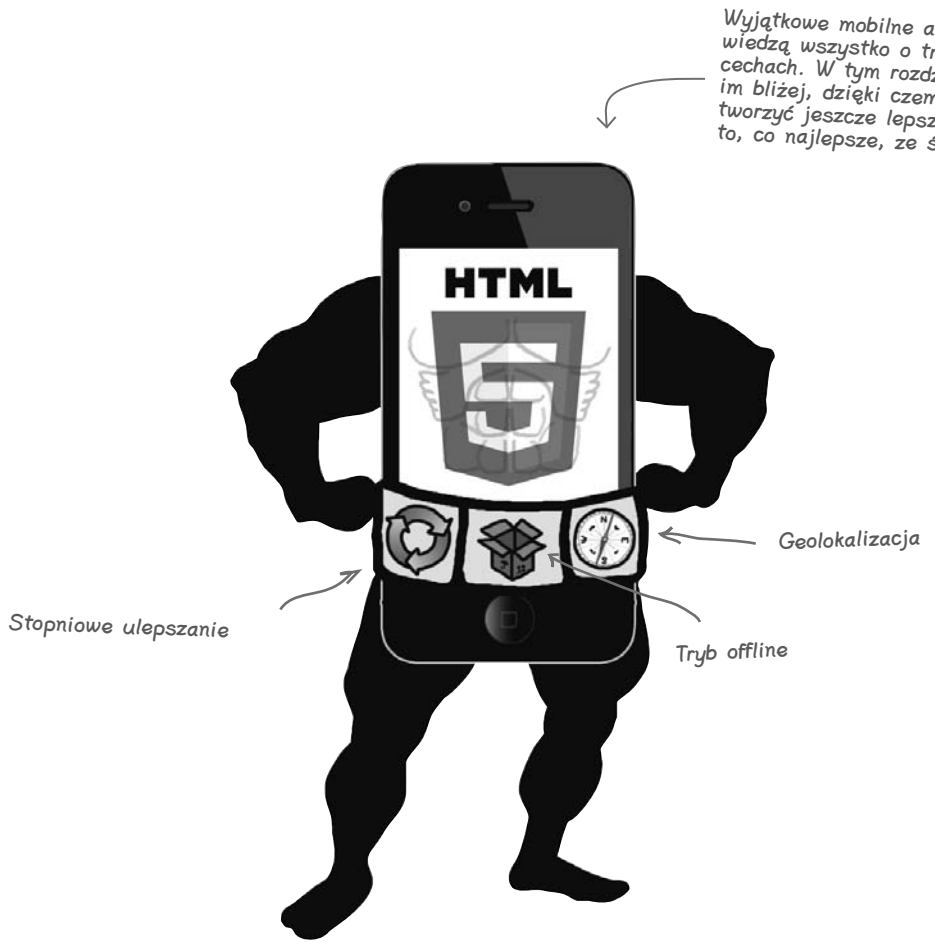
...a tu potrzeba działania!

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.

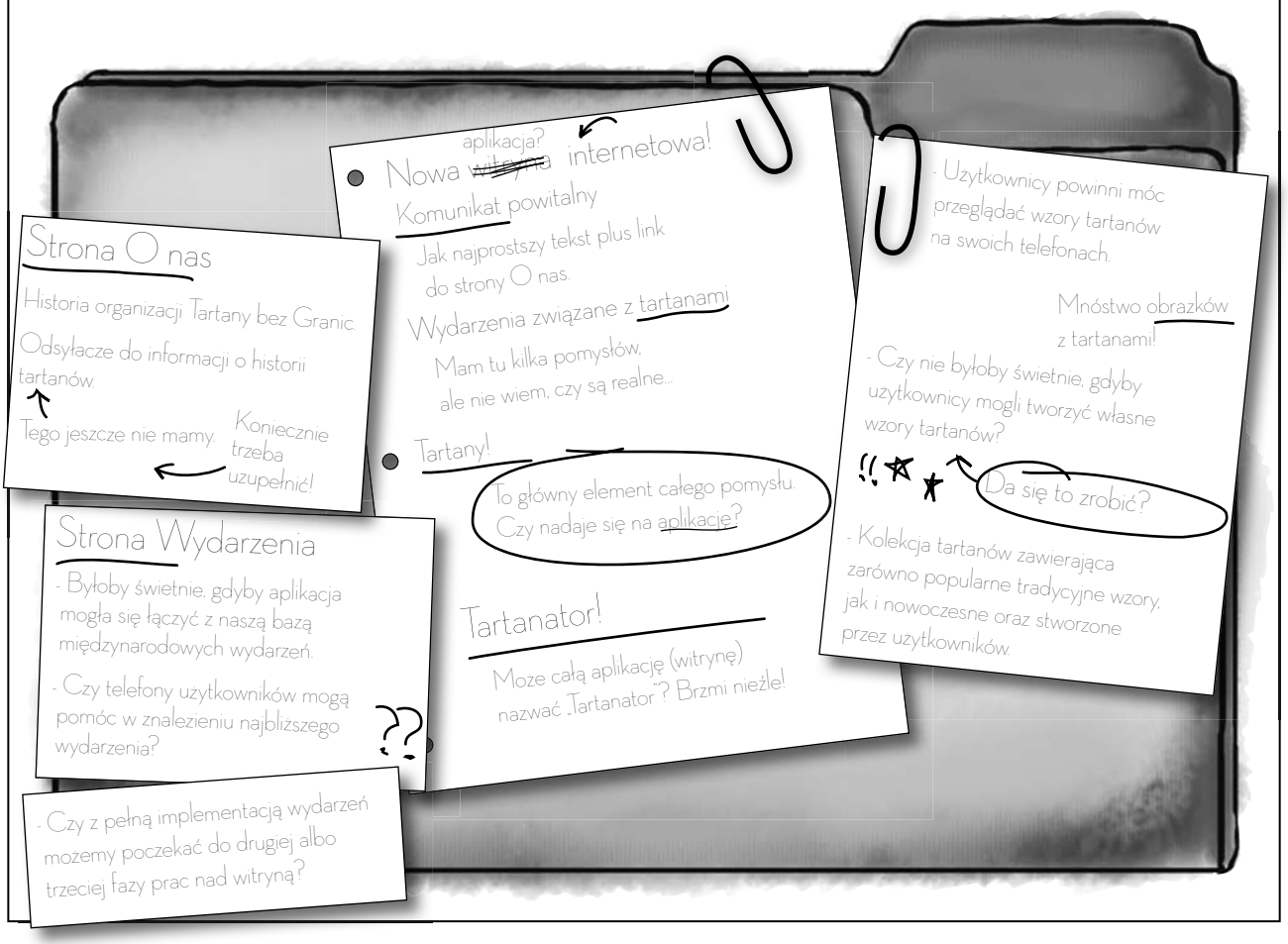
To, co można nazwać *wyjątkowymi mobilnymi aplikacjami internetowymi*, to aplikacje mobilne dostosowane do wymogów prawdziwego świata. Dopasowują się do możliwości konkretnych urządzeń użytkowników, korzystając z techniki **stopniowego ulepszania**. Jeśli chwilowo użytkownik nie ma dostępu do internetu, 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.

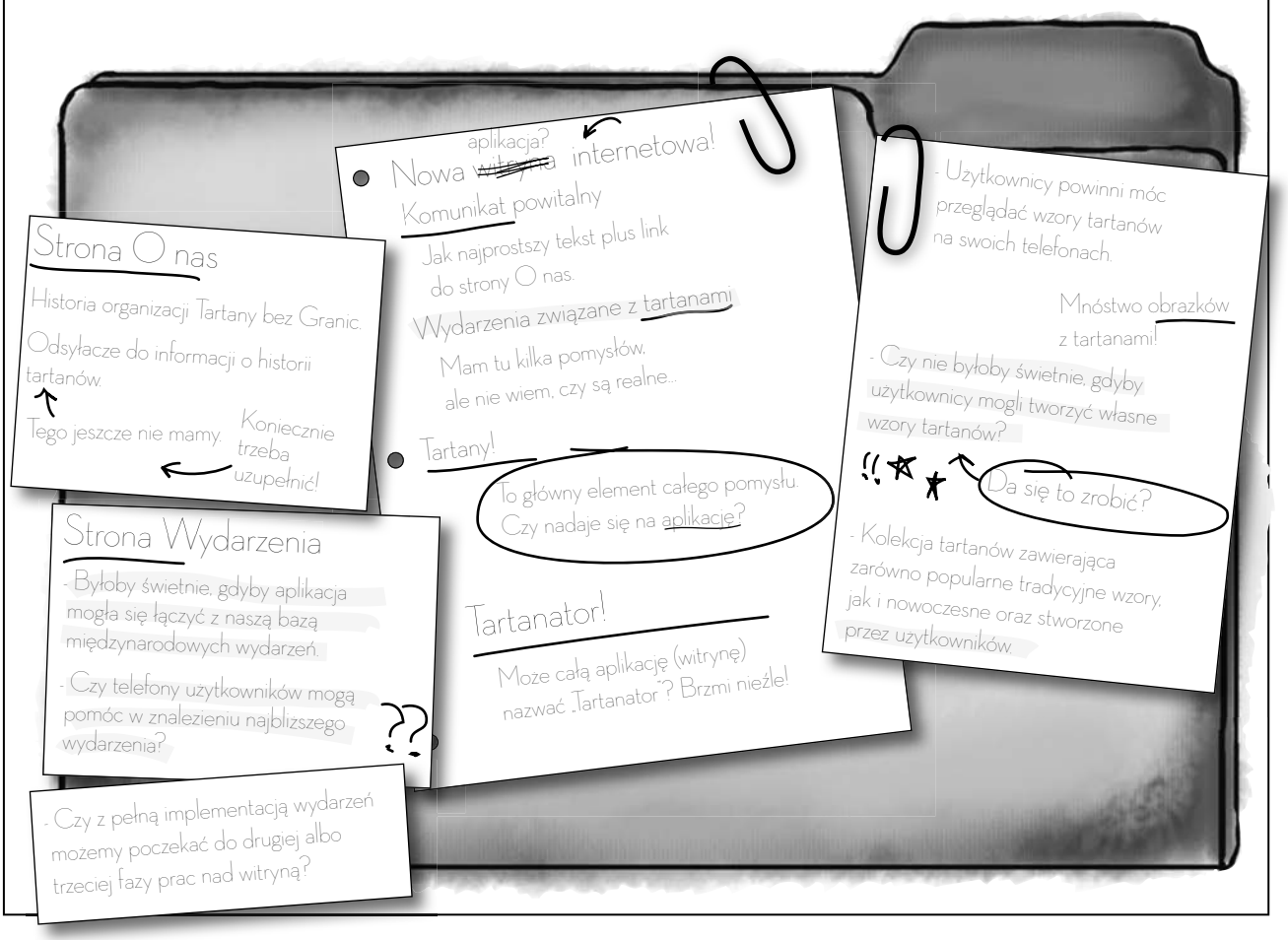


Zaostrz ołówek. Rozwiązanie

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

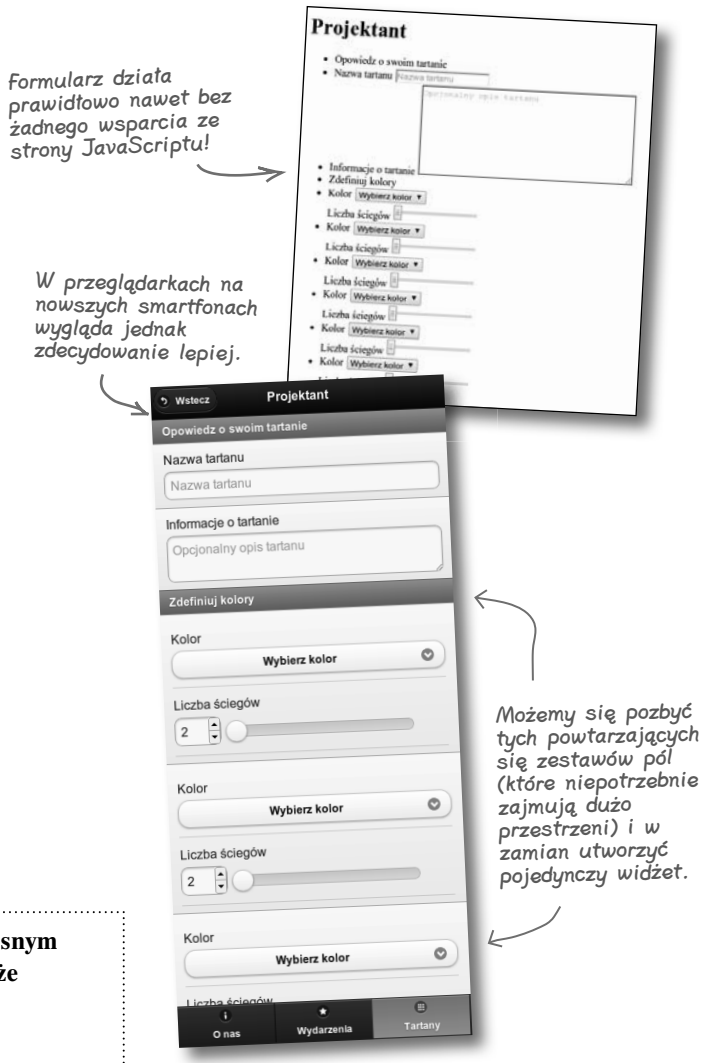
Zaprojektowaliśmy formularz, który spełnia minimalne wymagania. Żaden użytkownik nie pozostanie na lodzie (no dobra, może jakiś by się znalazł). Zobacz, jak wygląda formularz w przeglądarce z wyłączoną obsługą JavaScriptu.

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!

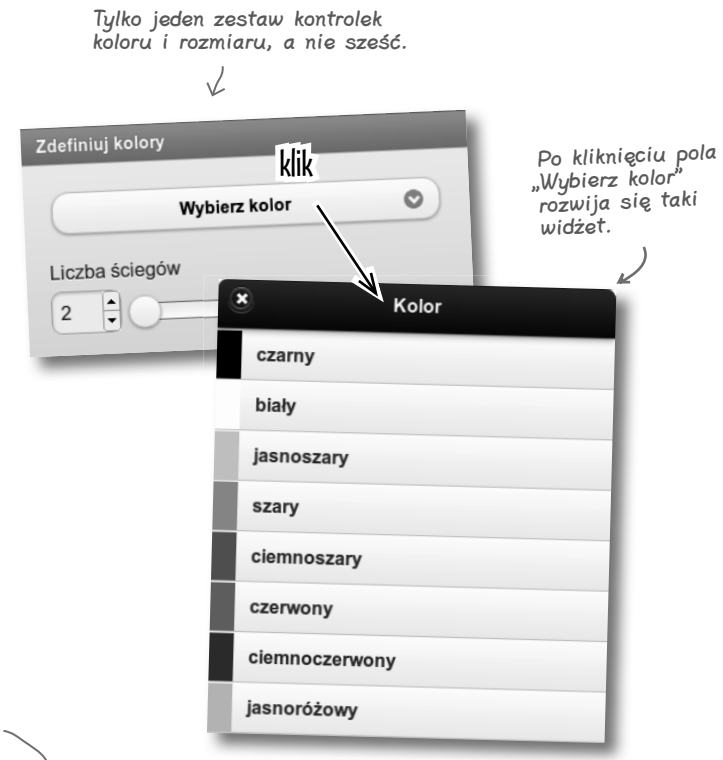
Ulepszamy formularz

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.



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*).

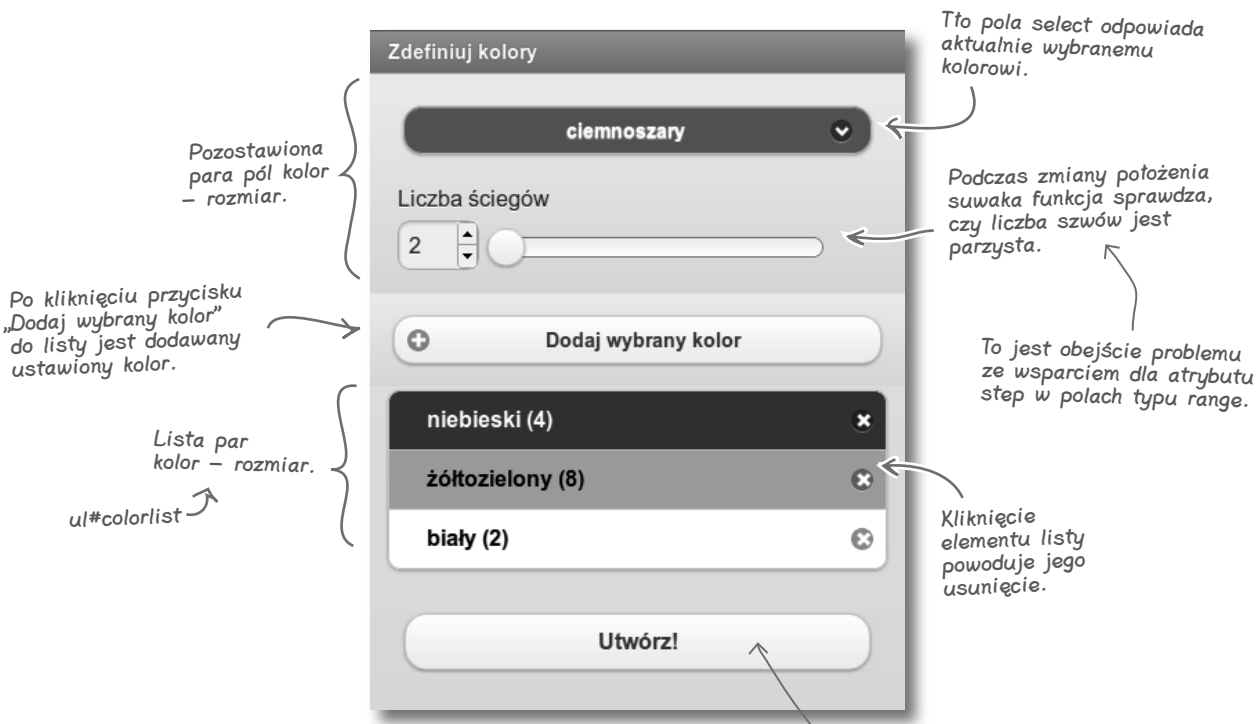
Widżet zarządzający listą kolorów i rozmiarów

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.

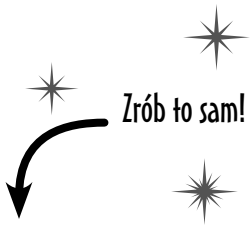
Nie zajęliśmy się jeszcze stroną serwerową. Kliknięcie przycisku „Utwórz!" na razie nie działa.

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.

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()	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`

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.

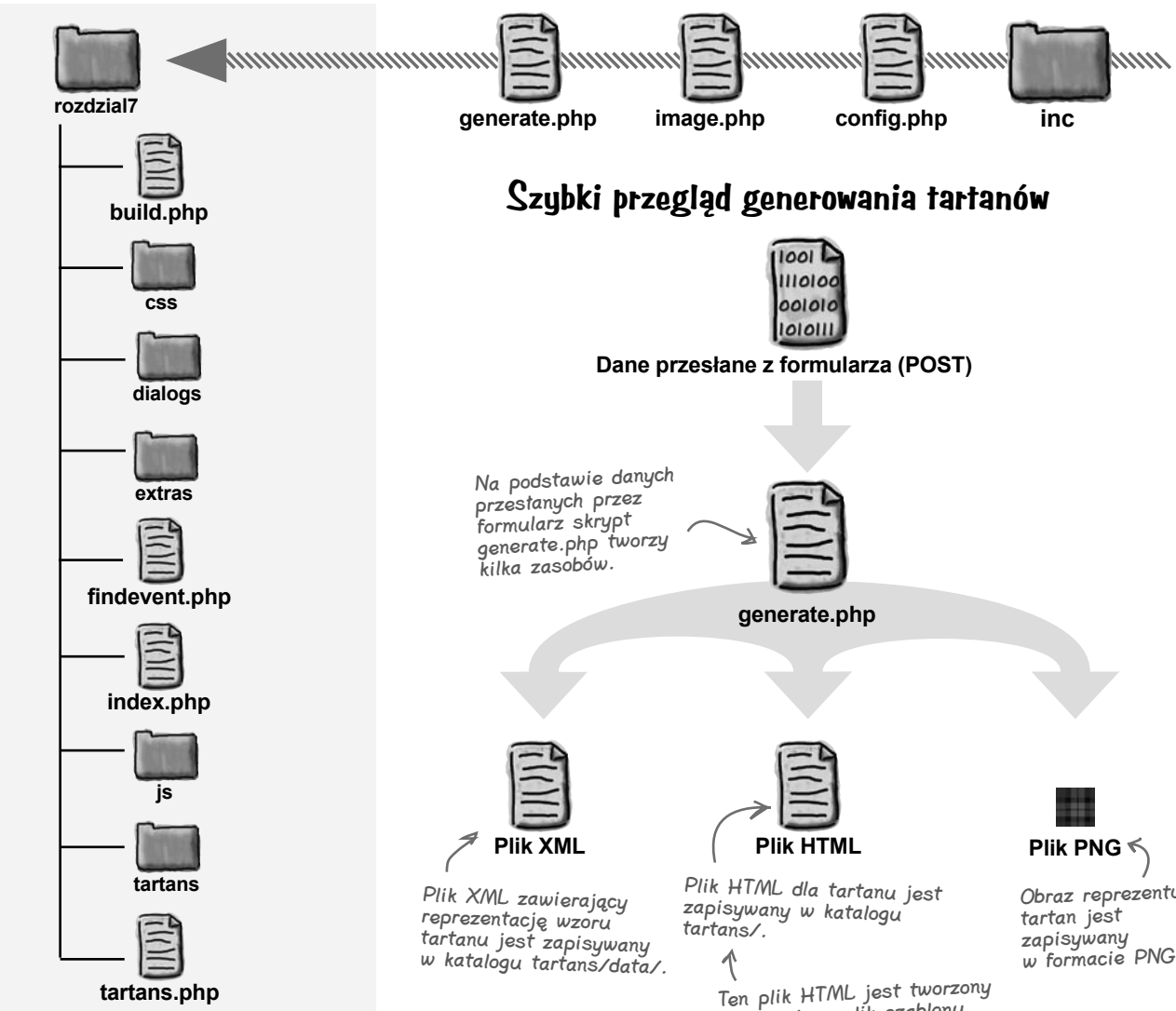
Zwróć jednak uwagę, że kod HTML odpowiadający za to pole (`<select>`) nadal jest semantycznie poprawny. Ja je tylko trochę poprawiłem, by stało się bardziej użyteczne. Podsumowując, masz rację, że do zmiany natywnych kontrolek formularza należy podchodzić z dużą ostrożnością.

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).



Dwie twarze skryptu generate.php

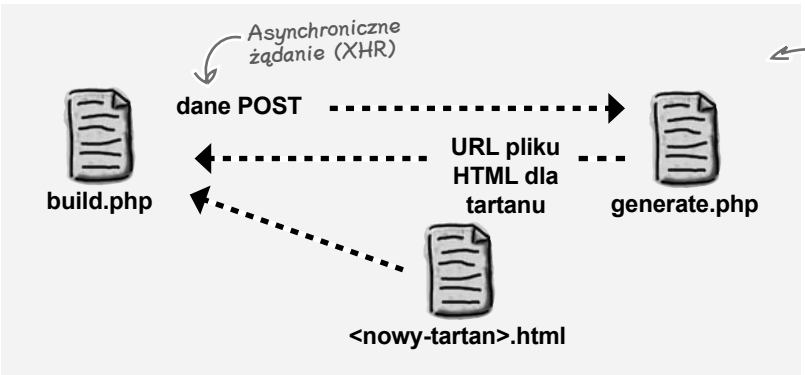
Sposób, w jaki skrypt *generate.php* zachowuje się po utworzeniu zasobów związanych z tartanem, zależy od sposobu jego wywołania.

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



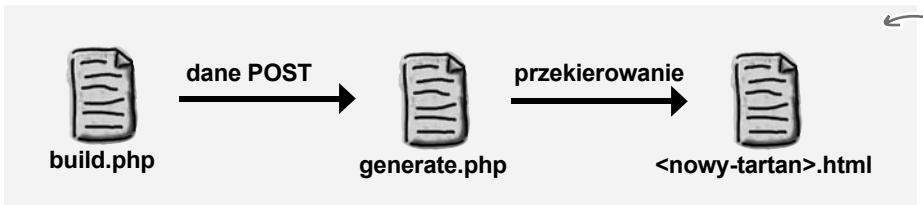
W tej metodzie nigdy nie dochodzi do pełnego przeładowania strony. Zawartość strony nowego tartanu jest wstawiana do struktury DOM strony edytora tartanów.

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

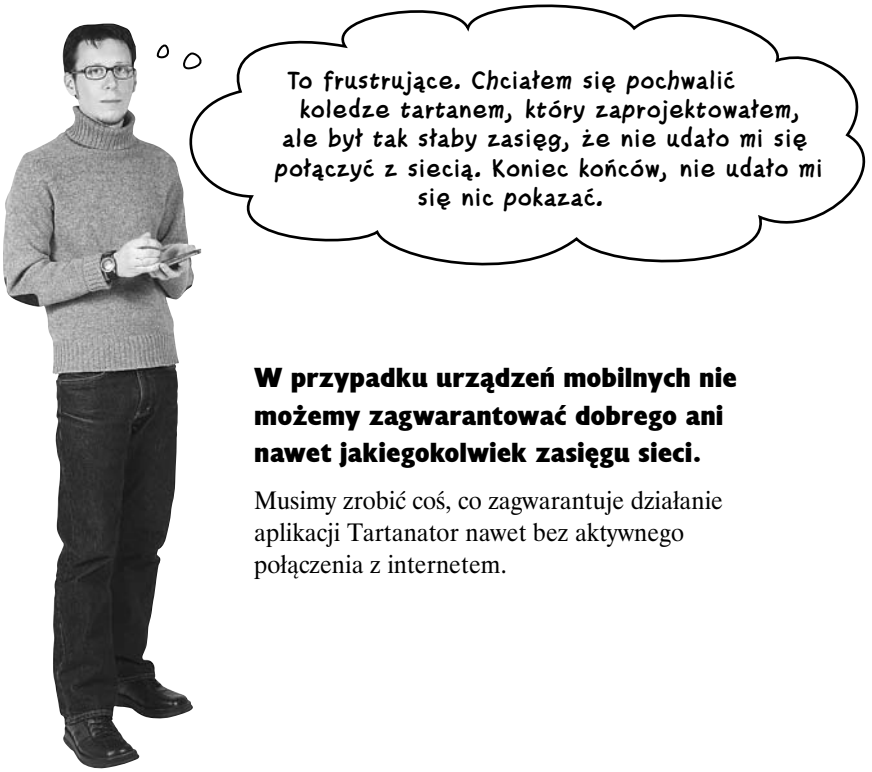
Coraz bardziej zbliżamy się do wyjątkowej mobilnej aplikacji internetowej. Udoskonaliliśmy już interfejs, z czego skorzystają użytkownicy lepszych przeglądarek. Zadbaliśmy też w końcu o to, by aplikacja coś robiła!

- [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.



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://...).

```
CACHE MANIFEST
# tak się umieszcza komentarze

CACHE:

index.html
foo/bar.html
baz.html
css/styles.css
icons/plus.png
```

manifest.appcache

Plik nie musi się nazywać „manifest", ale powinien mieć rozszerzenie „.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.

```
<!DOCTYPE html>
<html manifest="manifest.appcache">
<head>
```

```
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

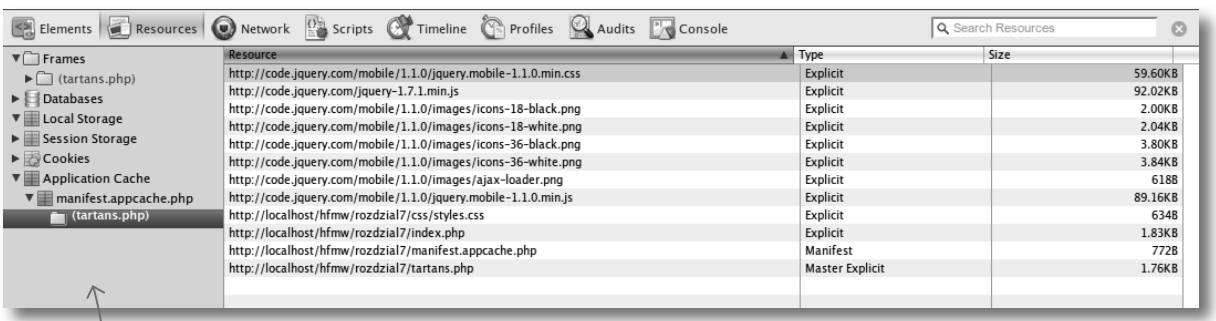
W teorii zarządzanie zasobami w trybie offline za pomocą pliku manifestu jest proste. Wystarczy wypisać pliki, które mają być dostępne offline, dodać atrybut `manifest` do znacznika `<html>` na stronach i już!

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

Nigdy się nie bawiłam plikami .htaccess i nie wiem nawet, czy mój hosting na to pozwala.

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!

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.

- ☑ 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.

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

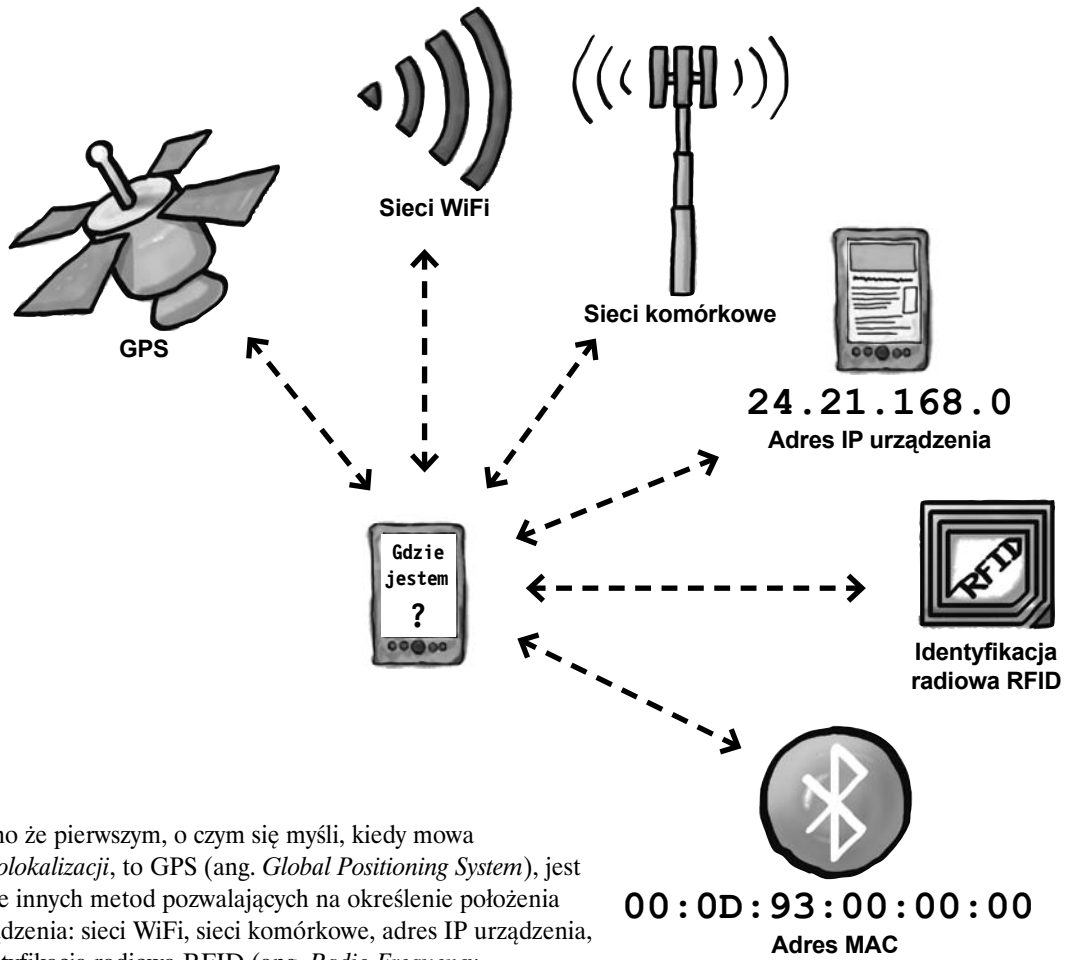
Kolejnym elementem drugiej fazy projektu jest stworzenie strony, która pozwala wyszukiwać wydarzenia związane z tartanami w najbliższej okolicy użytkowników.

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?

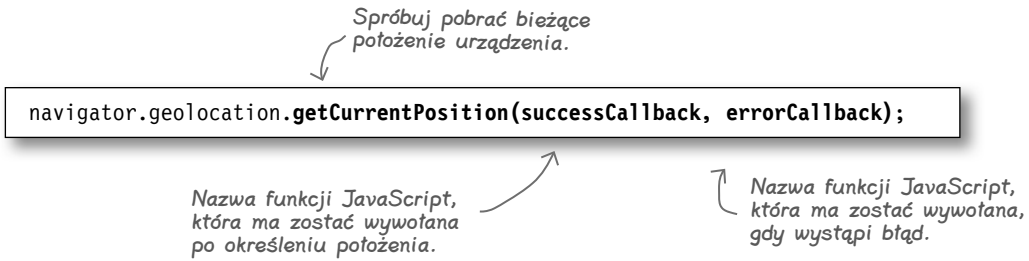
Skorzystanie z ***geolokalizacji*** w przeglądarce internetowej wymaga użycia JavaScriptu do pobrania bieżącej lokalizacji urządzenia. Wiele nowoczesnych przeglądarek mobilnych ma zaimplementowane wsparcie dla interfejsu API geolokalizacji zgodnego ze specyfikacją W3C (ang. *World Wide Web Consortium*), 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.

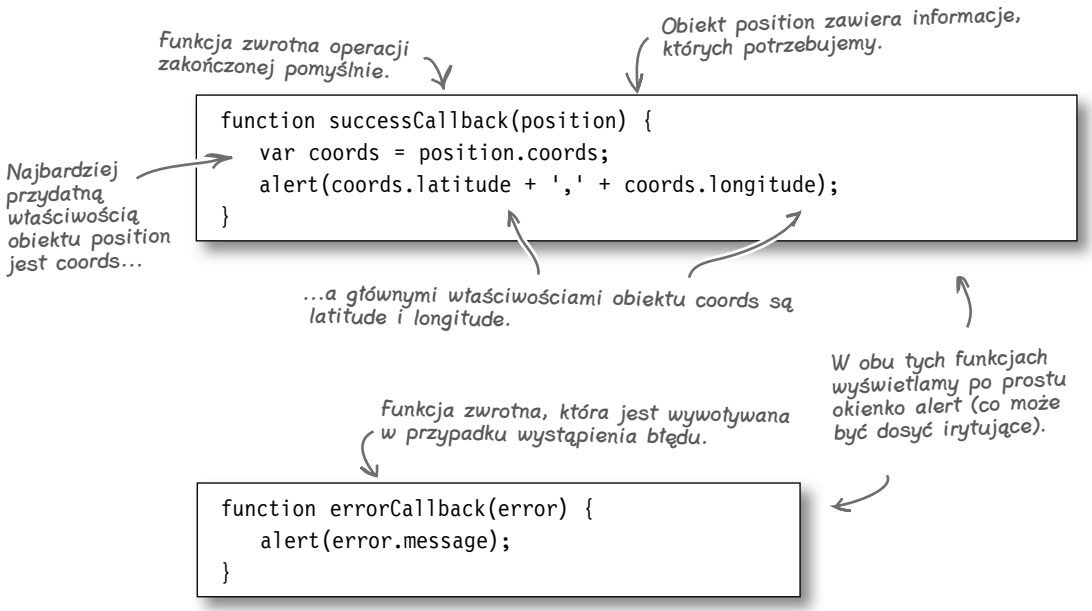
Jak poprosić przeglądarkę o dane geolokalizacyjne?

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.



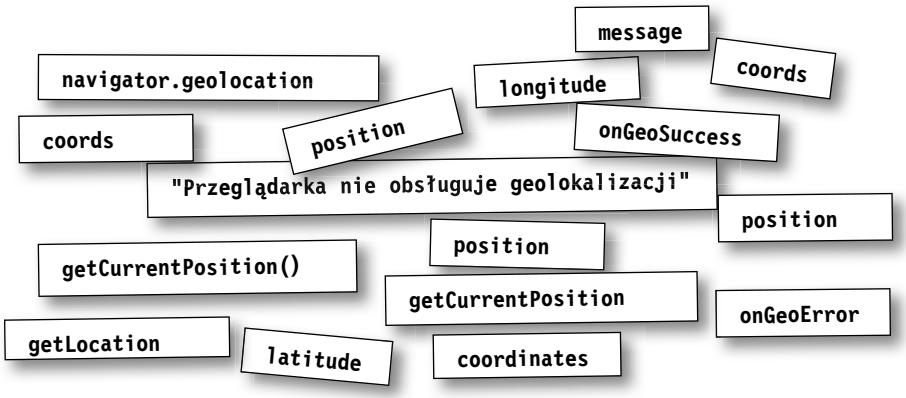
Korzystanie z danych dostarczanych przez getCurrentPosition

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.



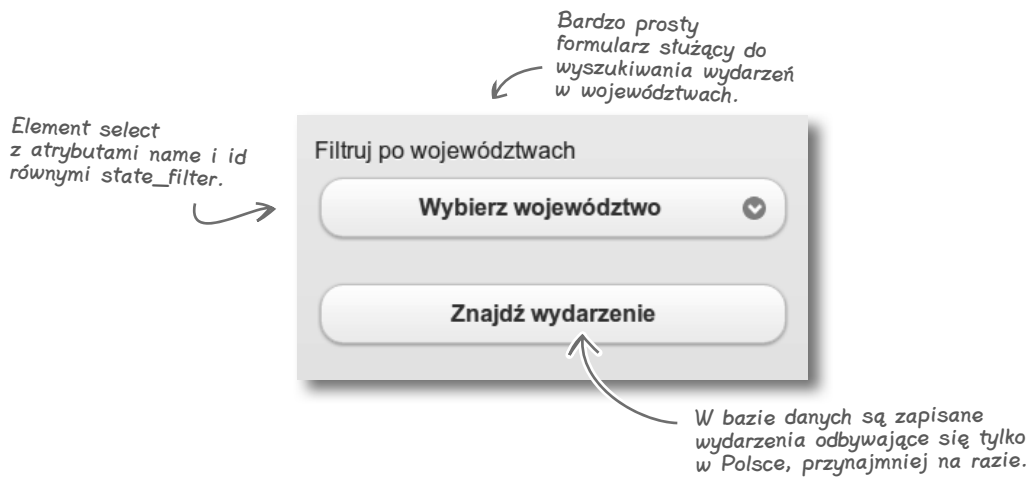

```
function getLocation () {
    // Sprawdzamy, czy przeglądarka obsługuje geolokalizację
    if (.....) {
        navigator.geolocation.....(....., onGeoError);
    } else { // Nie obsługuje
        onGeoError(new Error(.....));
    }
}

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

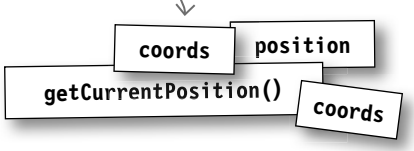
Udało Ci się umieścić magnesiki we właściwych miejscach?

```
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?

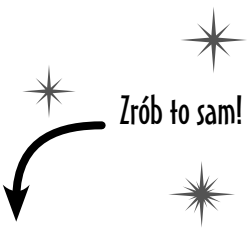
Łukasz: Jeśli użytkownik chce zatwierdzić formularz z tartanem, a nie poda jego nazwy, pojawia się okienko dialogowe informujące o błędzie. Wygląda całkiem ładnie — podobnie jak pozostałe elementy z jQM.

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.

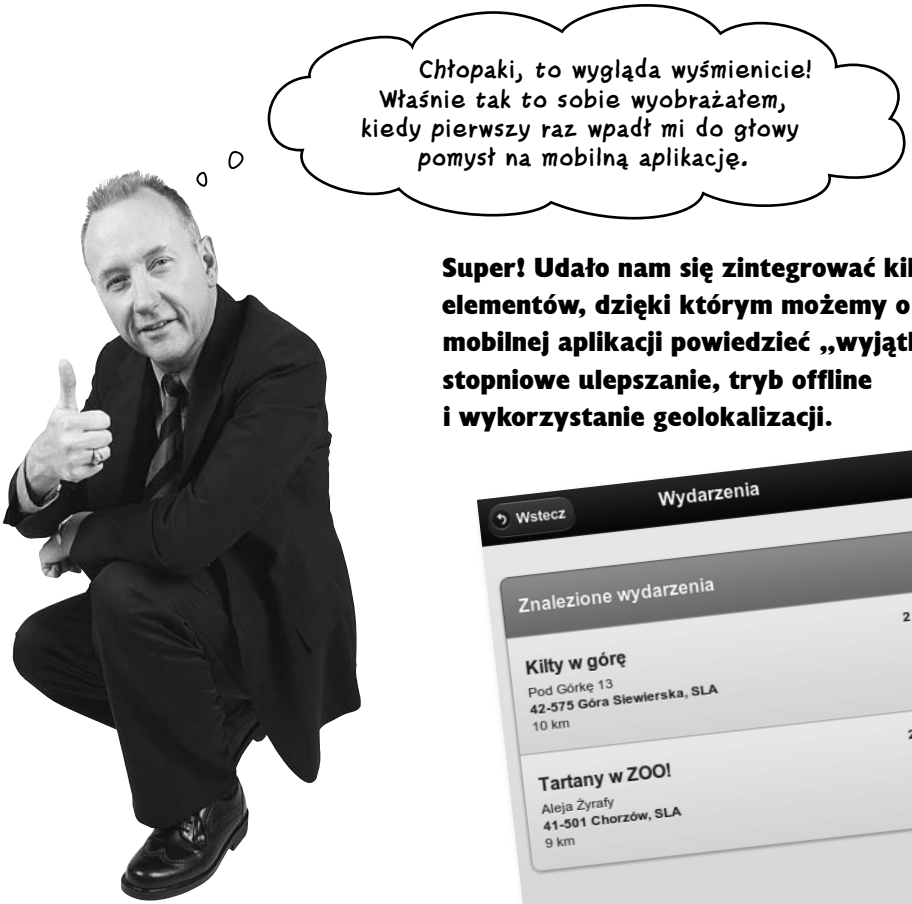
Jest to spowodowane doborem przykładowych danych oraz ustalonym promieniem poszukiwań (50 km). Co Ty na to, żeby dodać własne definicje wydarzeń, by otrzymać jakieś wyniki?

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.



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`.

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.

Skorowidz

D

E

F

G

X

Y

Z

Zdobądź nowych użytkowników smartfonów!

Mobile Web. Rusz głową!

Liczba użytkowników internetu przeglądających strony internetowe za pomocą smartfonów rośnie lawinowo. Powszechnie sądzi się, że wkrótce będzie ich więcej niż tych, którzy korzystają z tradycyjnych komputerów. Dlatego już dziś należy przygotować się na rewolucję i zmienić podejście do tworzenia stron internetowych. Skorzystaj z nowości HTML5 oraz CSS3 i przekonaj się, że to wcale nie musi być trudne.

Jeżeli dołożysz do tego kolejną książkę z serii *Rusz głową!*, całe zagadnienie może okazać się wręcz banalne. Dzięki nowatorskim technikom nauczania będziesz chłonął wiedzę niczym gąbka. W trakcie lektury nauczysz się korzystać z podejścia *Responsive Web Design* oraz sprawdzisz, jak rozpoznać, że użytkownik korzysta ze smartfona. Ponadto poznasz biblioteki, które wspomogą Cię w trakcie realizacji postawionych zadań — jQuery Mobile to jedna z nich, a jej możliwości są oszałamiające. Twoją ciekawość powinien wzbudzić też projekt PhoneGap. Dzięki niemu będziesz mógł skonwertować swoją aplikację napisaną przy użyciu HTML5 do natywnego formatu, a to da Ci parę nowych możliwości. Książka ta jest obowiązkową pozycją dla wszystkich projektantów stron internetowych, którzy chcą być na czasie! Zaliczasz się do nich?

Wykorzystaj potencjał HTML5 i CSS3!

Poznaj:

- przyszłość internetu i stron WWW
- podejście *Responsive Web Design*
- możliwości biblioteki jQuery i projektu PhoneGap
- rozwiązania typowych problemów

sięgnij po WIĘCEJ

KOD KORZYŚCI

ISBN 978-83-246-4864-1

9 788324 648641

Cena 79,00 zł

helion.pl
księgarnia internetowa

Nr katalogowy: 11711

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu