TWÓRZ ZAAWANSOWANE APLIKACJE!

Apress

HTML5 Zaawansowane programowanie

Peter Lubbers, Brian Albers, Frank Salim

Helion

Tytuł oryginału: Pro HTML5 Programming, Second Edition

Tłumaczenie: Jakub Hubisz

ISBN: 978-83-246-4809-2

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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/htm5zp.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/htm5zp
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 5

Wykorzystanie geolokalizacji

Załóżmy, że chcesz stworzyć aplikację oferującą zniżki i promocje na buty biegowe w sklepach znajdujących się w niewielkiej odległości od użytkownika. Korzystając z Geolocation API, możesz poprosić użytkownika, aby podał swoją aktualną pozycję. Jeżeli wyrazi zgodę, możesz przekazać mu instrukcje na temat tego, w jaki sposób dostać się do pobliskiego sklepu po nową parę butów w obniżonej cenie.

Innym przykładem wykorzystania geolokalizacji może być aplikacja mierząca odległość, jaką przebiegłeś (lub przeszedłeś). Wyobraź sobie aplikację działającą w przeglądarce Twojego telefonu komórkowego, którą włączasz, kiedy zaczynasz bieg. Aplikacja śledzi trasę, jaką przebyłeś. Koordynaty trasy mogą nawet zostać nałożone na mapę, być może wraz z profilem wysokościowym. Jeżeli ścigasz się z innymi uczestnikami biegu, aplikacja może nawet pokazywać pozycje Twoich przeciwników.

Pomysły na skorzystanie z możliwości geolokalizacji to na przykład aplikacja nawigująca przez kolejne etapy trasy, aplikacja społecznościowa pokazująca, gdzie dokładnie znajdują się Twoi znajomi, tak abyś mógł wybrać dogodną dla wszystkich kawiarnię, a także wiele innych, bardziej niecodziennych.

W tym rozdziale pokażemy, co można osiągnąć dzięki Geolocation API — ekscytującemu interfejsowi pozwalającemu użytkownikom na dzielenie się swoimi lokalizacjami w aplikacjach webowych, tak aby możliwe było wykorzystywanie uzależnionej od lokalizacji usługi. Najpierw zapoznamy się ze źródłem informacji geograficznych — szerokości, długości i innych atrybutów — i dowiemy się, skąd mogą one pochodzić (GPS, Wi-Fi, triangulacja itd.). Następnie omówimy problemy związane z prywatnością podczas korzystania z takich usług oraz opiszemy, jak przeglądarki obchodzą się z danymi geolokalizacyjnymi.

Potem zagłębimy się w praktyczną dyskusję na temat dwóch różnych funkcji (metod) żądania pozycji udostępnianych przez Geolocation API: jednorazowego żądania pozycji oraz ciągłej aktualizacji. Powiemy, kiedy i w jaki sposób z nich korzystać. W końcu pokażemy, jak zbudować praktyczną aplikację przy wykorzystaniu Geolocation API. Zakończymy opisem kilku ciekawych przypadków użycia oraz praktycznymi wskazówkami.

Informacje o lokalizacji

Interfejs Geolocation jest przejrzysty. Żądasz podania pozycji i jeżeli użytkownik wyrazi zgodę, przeglądarka zwraca informacje o lokalizacji. Pozycja podawana jest do wspierającej geolokalizację przeglądarki przez urządzenie (np. laptop lub telefon komórkowy), na którym została ona uruchomiona. Informacje obejmują zestaw koordynatów geograficznych (długość i szerokość geograficzna) wraz z towarzyszącymi metadanymi. Mając dostęp do tych informacji, możesz stworzyć aplikację zależną od lokalizacji użytkownika.

Szerokość i długość geograficzna

Informacje o lokalizacji składają się przede wszystkim z długości i szerokości geograficznej — podobnych do tych pokazanych w kolejnym przykładzie, które składają się na koordynaty pięknego Tahoe City, położonego na brzegu jeziora Tahoe, najpiękniejszego jeziora górskiego Ameryki:

```
Szerokość: 39.17222, Długość: -120.13778
```

W tym przykładzie szerokość geograficzna (wartość liczbowa reprezentująca odległość na północ lub południe od równika) wynosi `39.17222`, natomiast długość (wartość liczbowa reprezentująca odległość na wschód lub zachód od południka) ma wartość `–120.13778`.

Wartości te mogą być wyrażone w różny sposób:

- w formacie liczbowym — np. `39.17222`,
- jako „stopnie-minuty-sekundy" (DMS — *Degree Minute Second*) — np. `39° 10' 20"`,

■ **Uwaga.** W Geolocation API koordynaty podawane są zawsze w formacie liczbowym.

Oprócz długości i szerokości geograficznej udostępniana jest także *dokładność* koordynatów. Zależnie od urządzenia, na jakim uruchamiana jest przeglądarka, dostępne mogą być także dodatkowe metadane — mogą one zawierać *wysokość nad poziomem morza* wraz z *dokładnością*, *kąt odchylenia* i *prędkość*. Jeżeli dodatkowe informacje nie będą dostępne, przyjmą wartość `null`.

Skąd pochodzą informacje

Geolocation API nie definiuje, z jakiej technologii musi skorzystać urządzenie w celu określenia lokalizacji użytkownika — udostępnia jedynie możliwość pobrania tych informacji oraz poziomu dokładności, z jaką urządzenie zostało zlokalizowane. Nie ma żadnej gwarancji, że współrzędne zwrócone przez urządzenie są poprawne.

Lokalizacja, lokalizacja

Peter mówi: Oto ciekawy przykład. W domu korzystam z sieci bezprzewodowej. Otwarłem przykładową aplikację pokazaną w tym rozdziale w przeglądarce Firefox i dowiedziałem się, że jestem w Sacramento (ok. 120 km od mojego rzeczywistego położenia). Wynik jest błędny, ale nie zaskakujący — dostawca internetu, z którego usług korzystam, ma siedzibę w Sacramento.

Poprosiłem wtedy moich synów — Seana i Rocky'ego — aby otwarli stronę na iPhonie (przy wykorzystaniu tego samego połączenia bezprzewodowego). Safari pokazało, że znajdują się w Marysville w Kalifornii — mieście znajdującym się 50 km od Sacramento.

Dane o lokalizacji mogą pochodzić z następujących źródeł:

- adresu IP,
- triangulacji:
  - systemu globalnego pozycjonowania (GPS — *Global Positioning System*),
  - Wi-Fi z adresami MAC z sieci RFID, Wi-Fi lub Bluetooth,
  - identyfikatorów GSM lub CDMA,
- danych podanych przez użytkownika.

Wiele z urządzeń wykorzystuje kombinację różnych źródeł w celu zapewnienia większej dokładności. Każda z tych metod ma wady i zalety, które opiszemy w kolejnym podrozdziale.

Dane wynikające z adresu IP

Dane wynikające z adresu IP były dawniej jedyną możliwością uzyskania informacji o lokalizacji użytkownika, często jednak były niepoprawne. W tej metodzie automatycznie zwracany jest adres fizyczny rejestratora danego adresu IP. W związku z tym jeżeli adres IP uzyskujesz od dostawcy internetu, Twoja lokalizacja będzie często równoznaczna z jego fizycznym adresem, który może się znajdować wiele kilometrów od Ciebie. Tabela 5.1 przedstawia wady i zalety stosowania geolokalizacji opartej na adresie IP.

Tabela 5.1. *Wady i zalety stosowania geolokalizacji opartej na adresie IP*

Zalety	Wady
Dostępna wszędzie.	Niezbyt dokładna (często albo błędna, albo z dokładnością ograniczoną do miejscowości).
Przetwarzanie po stronie serwera.	Może być kosztowną operacją.

Wiele stron wyświetla reklamy w oparciu o lokalizację adresu IP. Możesz to zaobserwować podczas podróży do innego kraju, kiedy nagle zobaczysz reklamy lokalnych firm (zależnie od adresu IP regionu lub kraju, w którym się znalazłeś).

Dane GPS

Dopóki widać niebo, GPS może być źródłem bardzo dokładnych danych geolokalizacyjnych. Pozycja wyliczana jest na podstawie sygnału z wielu satelitów okrążających Ziemię. Sporo czasu może jednak zająć uzyskanie sygnału z odpowiedniej ich liczby, co nie jest dobre w przypadku aplikacji, które muszą rozpocząć działanie natychmiast.

Z powodu czasu potrzebnego na uzyskanie informacji można próbować wykonywać tę operację asynchronicznie. Aby pokazać użytkownikom, że pobierana jest ich lokalizacja, można wyświetlić pasek postępu. Tabela 5.2 pokazuje wady i zalety stosowania geolokalizacji opartej na GPS-ie.

Tabela 5.2. *Wady i zalety stosowania geolokalizacji opartej na GPS-ie*

Zalety	Wady
Bardzo dokładna.	Uzyskanie informacji o pozycji może zająć dużo czasu, co może doprowadzić do szybszego wyczerpania baterii urządzenia.
	Niezbyt dobrze działa wewnątrz budynków.
	Może wymagać dodatkowego sprzętu.

Dane pobierane w oparciu o Wi-Fi

Pobierane informacji w oparciu o sieć bezprzewodową działa na zasadzie triangulacji — sprawdzana jest odległość użytkownika od znanych punktów dostępowych (głównie w miastach). W przeciwieństwie do GPS-u dane uzyskiwane tą metodą są równie dokładne wewnątrz budynków. Tabela 5.3 przedstawia wady i zalety stosowania geolokalizacji opartej na Wi-Fi.

Tabela 5.3. *Wady i zalety stosowania geolokalizacji opartej na Wi-Fi*

Zalety	Wady
Dokładna.	Niezbyt dobrze działa na terenach z małą liczbą punktów dostępu.
Działa wewnątrz budynków.	
Namierzanie jest szybkie i tanie.	

Dane pobierane z telefonu komórkowego

Te dane także są pobierane w oparciu o triangulację — w tym przypadku jednak mierzona jest odległość użytkownika od nadajników telekomunikacyjnych. Dzięki temu otrzymujemy wynik o umiarkowanej dokładności. Metoda ta często wykorzystywana jest w połączeniu z metodą opartą na GPS-ie lub Wi-Fi. Tabela 5.4 przedstawia wady i zalety stosowania geolokalizacji działającej na podstawie telefonu komórkowego.

Tabela 5.4. *Wady i zalety stosowania geolokalizacji działającej na podstawie telefonu komórkowego*

Zalety	Wady
Dość dokładna.	Wymaga urządzenia z dostępem do telefonu lub modemu komórkowego.
Działa wewnątrz budynków.	Niezbyt dobrze działa na terenach z małą liczbą nadajników telekomunikacyjnych.
Namierzanie jest szybkie i tanie.	

Dane podawane przez użytkownika

Zamiast próbować uzyskać lokalizację użytkownika programistycznie, możesz pozwolić mu na jej samodzielne wprowadzenie. Aplikacja może pozwalać na wprowadzenie adresu, kodu pocztowego lub innych informacji; te dane możesz wykorzystać do zaserwowania użytkownikowi usług zależnych od lokalizacji. Tabela 5.5 przedstawia wady i zalety danych podawanych przez użytkownika.

Tabela 5.5. *Wady i zalety danych podawanych przez użytkownika*

Zalety	Wady
Użytkownik może dysponować danymi dokładniejszymi niż dostępne programistycznie.	Dane mogą być bardzo niedokładne, zwłaszcza gdy lokalizacja ulega zmianie.
Pozwala na wprowadzanie alternatywnych lokalizacji.	
Wprowadzenie danych przez użytkownika może być szybsze niż detekcja lokalizacji.	

Wsparcie geolokalizacji przez przeglądarki

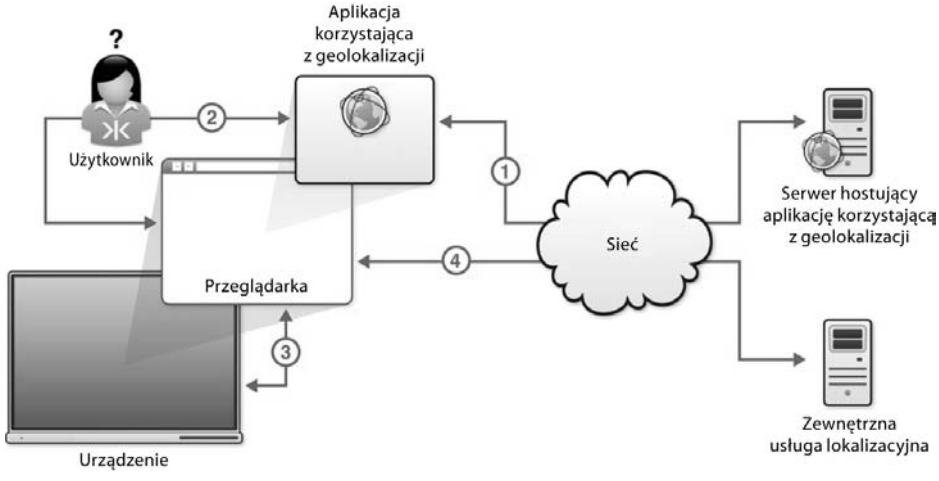
Interfejs Geolocation był jednym z pierwszych elementów HTML5, który został w pełni zaimplementowany, i wspierany jest obecnie przez wszystkie istotne przeglądarki. Jeżeli interesują Cię dokładniejsze informacje na temat wsparcia geolokalizacji, wejdź na stronę *http://caniuse.com/* i poszukaj hasła *Geolocation*.

Jeśli musisz zapewnić wsparcie dla starszych przeglądarek, przed skorzystaniem z API warto sprawdzić, czy interfejs Geolocation jest obsługiwany. W podrozdziale „Sprawdzenie wsparcia przeglądarki” pokażemy, w jaki sposób to zrobić.

Prywatność

Specyfika geolokalizacji wymaga, aby zapewniony został mechanizm ochrony prywatności użytkownika. Co więcej, informacje nie powinny być udostępniane, jeżeli użytkownik nie wyrazi na to zgody.

Jest to odpowiedź na problem „wielkiego brata", często poruszany przez użytkowników aplikacji wykorzystujących geolokalizację. Jak jednak widać z niektórych przypadków użycia Geolocation API w aplikacjach, użytkownicy zachęcani są do dzielenia się swoimi lokalizacjami. Na przykład użytkownik może chętniej podzielić się danymi geolokalizacyjnymi, jeżeli w zamian może uzyskać informację o rabacie na buty biegowe, dostępnym w sklepie oddalonym parę przecznic od miejsca, w którym właśnie pije kawę. Przyjrzyjmy się bliżej architekturze prywatności pokazanej na rysunku 5.1.



Rysunek 5.1. *Architektura prywatności przeglądarki i urządzenia*

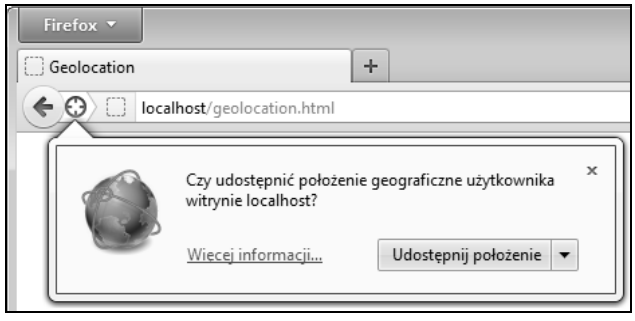
Na diagramie pokazano następujące kroki:

1. Użytkownik uruchamia w przeglądarce aplikację korzystającą z geolokalizacji.
2. Strona aplikacji ładuje się i przesyła prośbę o koordynaty poprzez wywołanie funkcji z Geolocation API. Przeglądarka przechwytuje zapytania i pyta użytkownika o pozwolenie. Załóżmy, że użytkownik udziela pozwolenia.
3. Przeglądarka pobiera koordynaty z urządzenia, na którym została uruchomiona. W tym przypadku jest to kombinacja danych pochodzących z adresu IP, Wi-Fi i GPS-u. Jest to wewnętrzna funkcjonalność przeglądarki.
4. Przeglądarka przesyła dane do zaufanej, zewnętrznej usługi lokalizacyjnej — zwracane koordynaty mogą być przekazane do aplikacji.

■ **Ważne.** Aplikacja *nie* ma bezpośredniego dostępu do urządzenia; może jedynie wysłać do przeglądarki prośbę o podanie informacji.

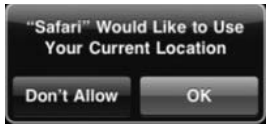
Uruchomienie mechanizmu ochrony prywatności

Kiedy uruchamiasz stronę korzystającą z Geolocation API, powinien zostać uruchomiony mechanizm ochrony prywatności. Rysunek 5.2 pokazuje, jak wygląda to w przeglądarce Firefox.



Rysunek 5.2. *Jeżeli w przeglądarce Firefox wykorzystywany jest interfejs Geolocation, pojawia się panel informacyjny*

Mechanizm uruchamiany jest w momencie uruchomienia kodu Geolocation. Dodanie kodu, który nie jest wywoływany (np. w zdarzeniu `onload`), nie spowoduje reakcji. Jeżeli jednak kod Geolocation zostanie uruchomiony, na przykład wywołaniem `navigator.geolocation.getCurrentPosition` (w dalszej części metoda zostanie omówiona dokładniej), użytkownik zostanie poproszony o podzielenie się informacjami o lokalizacji. Rysunek 5.3 pokazuje, co stanie się w przeglądarce Safari działającej na iPhonie.



Rysunek 5.3. *Jeżeli w przeglądarce Safari wykorzystywany jest interfejs Geolocation, pojawia się okno dialogowe z powiadomieniem*

W niektórych przypadkach (np. w Firefoksie) oprócz mechanizmu pytającego użytkownika o zezwolenie na pobranie lokalizacji możliwe jest także zapamiętanie decyzji użytkownika i uwzględnienie jej podczas kolejnych wizyt na stronie. Działa to podobnie do zapamiętywania haseł w przeglądarce.

■ **Uwaga.** Jeżeli w Firefoksie ustawiłeś, aby dla strony zawsze było udzielane pozwolenie, a później zmienisz zdanie, możesz łatwo odwołać pozwolenie. Musisz wejść na stronę, wybrać *Informacje o stronie* z menu *Narzędzia*, a następnie zmienić ustawienie dla opcji *Udostępnij położenie* na zakładce *Uprawnienia*.

Obchodzenie się z danymi geolokalizacyjnymi

Są to dane wrażliwe, więc jeżeli je otrzymasz, musisz być ostrożny podczas ich obsługi, przechowywania i powtórnej transmisji. Jeżeli użytkownik nie wyrazi zgody na przechowywanie danych, powinieneś się ich pozbywać zawsze po zakończeniu pracy z nimi.

Jeżeli retransmitujesz pozyskane dane, pamiętaj, że powinieneś je najpierw zaszyfrować. Twoja aplikacja powinna też zawsze w jasny sposób informować użytkownika:

- że zbierasz dane geolokalizacyjne,
- dlaczego je zbierasz,
- jak długo są przechowywane,
- w jaki sposób są zabezpieczone,
- z kim i jak dzielisz się tymi danymi,
- w jaki sposób użytkownik może sprawdzić i zaktualizować swoje dane.

Wykorzystanie Geolocation API

W tym podrozdziale bardziej szczegółowo omówimy wykorzystanie Geolocation API. Stworzymy prostą stronę — *geololocation.html*. Pamiętaj, że cały kod możesz pobrać pod adresem: *ftp://ftp.helion.pl/przyklady/htm5zp.zip*.

Sprawdzenie wsparcia w przeglądarce

Zanim wywołasz funkcje z Geolocation API, dobrze jest sprawdzić, czy to, co zamierzasz zrobić, wspierane jest przez przeglądarkę. Dzięki temu, jeżeli przeglądarka nie wspiera odpowiednich funkcji, możesz przekazać użytkownikowi odpowiednią informację lub poprosić go o zainstalowanie zewnętrznej wtyczki (np. Gears), która poprawi funkcjonalność przeglądarki. Na listingu 5.1 przedstawiony został jeden ze sposobów sprawdzenia wsparcia w przeglądarce.

Listing 5.1. *Sprawdzenie wsparcia w przeglądarce*

```
<script>
function loadDemo() {
    if(navigator.geolocation) {
        document.getElementById("support").innerHTML = "Geolokalizacja jest wspierana.";
    } else {
        document.getElementById("support").innerHTML = "Twoja przeglądarka nie wspiera
                                                        geolokalizacji.";
    }
}
</script>
```

Funkcja `loadDemo` sprawdza wsparcie dla geolokalizacji w przeglądarce — może zostać wywołana po załadowaniu strony. Jeżeli będzie to możliwe, wywołanie `navigator.geolocation` (można także wykorzystać Modernizr) zwróci obiekt `Geolocation` — w przeciwnym razie wykonany zostanie wariant braku wsparcia. Skrypt aktualizuje zdefiniowany wcześniej element `support` tekstem informującym o możliwości wykorzystania geolokalizacji w przeglądarce.

Prośby o podanie lokalizacji

Dostępne są dwa typy prośby o podanie lokalizacji:

- jednorazowa prośba,
- ciągła aktualizacja.

Jednorazowa prośba

W wielu przypadkach akceptowalne będzie jednorazowe pobranie lokalizacji użytkownika. Na przykład jeżeli ktoś szuka najbliższego kina pokazującego najnowszy film w ciągu najbliższej godziny, może zostać wykorzystana najprostsza forma geolokalizacji, pokazana na listingu 5.2.

Listing 5.2. *Jednorazowe pobranie lokalizacji*

```
void getCurrentPosition(in PositionCallback successCallback,
                        in optional PositionErrorCallback errorCallback,
                        in optional PositionOptions options);
```

Przyjrzyjmy się dokładniej wywołaniu tej funkcji.

Funkcja ta udostępniana jest przez obiekt `navigator.geolocation`, więc aby z niej skorzystać, należy najpierw pobrać sam obiekt. Jak już wcześniej wspominaliśmy, przed pobraniem obiektu warto sprawdzić, czy przeglądarka wspiera geolokalizację — jeżeli nie, należy zapewnić jakąś formę treści zastępczej.

Funkcja przyjmuje jeden obowiązkowy i dwa opcjonalne argumenty:

- Parametr `successCallback` mówi przeglądarce, która funkcja ma zostać wywołana po udostępnieniu danych geolokalizacyjnych. Jest to ważne, ponieważ operacja uzyskania informacji o lokalizacji może być długotrwała. Użytkownik nie będzie chciał, aby przeglądarka była zablokowana podczas pobierania danych. Programista także nie będzie chciał, aby jego program był zatrzymany na nieznany okres — zwłaszcza że przedłużenie tego czasu może być spowodowane oczekiwaniem na udzielenie przez użytkownika pozwolenia. Funkcja `successCallback` jest miejscem, w którym dostaniesz właściwe dane i będziesz mógł z nich skorzystać.
- Jak w przypadku większości działań programistycznych, dobrze jest zabezpieczyć się przed wystąpieniem błędów. Możliwe jest, że proces pobierania lokalizacji zakończy się niepowodzeniem z przyczyn będących poza Twoją kontrolą — w takich przypadkach wykorzystywana jest funkcja `errorCallback`, która może przekazać użytkownikowi wyjaśnienia lub spróbować powtórnie pobrać dane. Mimo że parametr ten jest opcjonalny, warto z niego skorzystać.
- Można także przekazać obiekt opcji, aby zmodyfikować sposób pobierania informacji. Jest to parametr opcjonalny, który bardziej szczegółowo omówimy w dalszej części rozdziału.

Załóżmy, że stworzyłeś funkcję JavaScript o nazwie `updateLocation()`, która aktualizuje dane na stronie zgodnie z nową pozycją geograficzną, oraz funkcję `handleLocationError()` do obsługi ewentualnych błędów (szczegóły tych funkcji pokażemy w dalszej części). Oznacza to, że wywołanie żądania podania pozycji będzie wyglądało następująco:

```
navigator.geolocation.getCurrentPosition(updateLocation, handleLocationError);
```

Funkcja updateLocation

Co dzieje się wewnątrz funkcji `updateLocation()`? Funkcja zostanie wywołana, gdy tylko przeglądarka uzyska dostęp do danych geolokalizacyjnych. Zostanie do niej wówczas przekazany pojedynczy parametr: obiekt lokalizacji. Obiekt będzie zawierał koordynaty (atrybut `coords`) oraz stempel czasowy oznaczający moment pobrania danych. O ile stempel czasowy może być dla Ciebie nieprzydatny, o tyle atrybut `coords` zawiera najważniejsze informacje.

Koordynaty zawsze zawierają kilka atrybutów, jednak od przeglądarki i urządzenia, na którym została uruchomiona, zależy, czy atrybuty zostaną wypełnione wartościami. Pierwsze trzy atrybuty są następujące:

- `latitude` — szerokość geograficzna,
- `longitude` — długość geograficzna,
- `accuracy` — dokładność.

Te atrybuty muszą posiadać wartość, a ich znaczenie jest jasne. Atrybuty `latitude` i `longitude` będą przechowywać określoną przez usługę Geolocation lokalizację, wyrażoną w stopniach dziesiętnych. Atrybut `accuracy` informuje, w jakiej odległości (w metrach) od podanej lokalizacji znajduje się lokalizacja właściwa — informacja ta jest podawana z prawdopodobieństwem 95%, może więc zostać wykorzystana do zaprezentowania promienia odległości, który zobrazuje użytkownikowi poziom dokładności. Ze względu na specyfikę Geolocation API dane często będą podawane w przybliżeniu. Zawsze sprawdzaj dokładność uzyskanych danych, zanim zaprezentujesz wyniki. Polecenie użytkownikowi „pobliskiego” sklepu, do którego musiałby jechać wiele godzin, może mieć nieprzewidziane konsekwencje.

Pozostałe atrybuty nie muszą być podawane, a jeżeli nie zostaną uzupełnione, będą zawierały wartość `null` (np. jest mało prawdopodobne, że dostęp do tych informacji uzyskasz na komputerze stacjonarnym):

- `altitude` — wysokość nad poziomem morza wyrażona w metrach,
- `altitudeAccuracy` — dokładność wysokości w metrach lub `null`, jeżeli wysokość (`altitude`) nie zostanie podana,

- `heading` — wyprzedzenie, czyli kierunek w stopniach relatywny do kierunku północnego,
- `speed` — prędkość w metrach na sekundę.

Jeżeli nie jesteś pewien, czy użytkownicy dysponują urządzeniami podającymi takie informacje, nie powinieneś polegać na nich w swojej aplikacji. Urządzenia GPS prawdopodobnie będą udostępniać te dane, jednak prosta triangulacja już na to nie pozwoli.

Przyjrzyjmy się teraz implementacji funkcji `updateLocation` — wykonuje ona proste aktualizacje na podstawie pobranych informacji (listing 5.3).

Listing 5.3. *Przykład wykorzystania funkcji updateLocation*

```
function updateLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var accuracy = position.coords.accuracy;
    var timestamp = position.timestamp;

    document.getElementById("latitude").innerHTML = latitude;
    document.getElementById("longitude").innerHTML = longitude;
    document.getElementById("accuracy").innerHTML = accuracy;
    document.getElementById("timestamp").innerHTML = timestamp;
}
```

W przykładzie funkcja `updateLocation` aktualizuje tekst elementów na stronie. Długość, szerokość, dokładność i stempel czasowy wstawiamy w odpowiednie pola.

Funkcja handleLocationError

Bardzo istotnym elementem Geolocation API jest obsługa błędów; wiele jej części jest zmiennych i istnieje spore prawdopodobieństwo niedostępności usługi geolokalizacyjnej. Na szczęście API definiuje kody błędów dla wszystkich przypadków, z jakimi możesz mieć do czynienia. Do funkcji obsługi błędów przekazywany jest obiekt, którego atrybut `code` zawiera odpowiedni kod. Oto dostępne kody:

- `PERMISSION_DENIED` (kod 1) — użytkownik nie zezwolił na pobranie danych geolokalizacyjnych,
- `POSITION_UNAVAILABLE` (kod 2) — podjęta została próba pobrania danych, jednak zakończyła się niepowodzeniem,
- `TIMEOUT` (kod 3) — przekroczony został, zdefiniowany jako opcja, maksymalny czas oczekiwania na dane.

W przypadku wystąpienia któregoś z błędów będziesz najprawdopodobniej chciał dać użytkownikowi znać, że coś poszło nie tak. W dwóch ostatnich przypadkach możesz także spróbować ponownie pobrać dane.

Listing 5.4 zawiera przykład funkcji obsługującej błędy.

Listing 5.4. *Funkcja obsługująca błędy*

```
function handleLocationError(error) {
    switch(error.code){
        case 0:
            updateStatus("Podczas pobierania lokalizacji wystąpił błąd: " + error.message);
            break;
        case 1:
            updateStatus("Użytkownik nie zezwolił na dostęp do danych geolokalizacyjnych.");
            break;
        case 2:
            updateStatus("Przeglądarka nie była w stanie uzyskać informacji o lokalizacji: " +
                         blad.message);
            break;
        case 3:
            updateStatus("Maksymalny dozwolony czas pobierania danych został przekroczony.");
```

```
            break;
    }
}
```

Kody błędów udostępniane są w atrybucie `code` obiektu `error`, atrybut `message` zawiera bardziej szczegółowy opis napotkanego błędu. We wszystkich przypadkach wywołujemy stworzoną przez nas metodę aktualizującą status na stronie.

Opcjonalne atrybuty geolokalizacji

Obsłużyliśmy już zarówno poprawne, jak i niepoprawne pobranie lokalizacji. Możemy teraz przyjrzeć się możliwości wpływania na sposób pobierania danych przez usługę geolokalizacyjną poprzez przekazanie jednego z trzech atrybutów opcjonalnych. Mogą być one przekazane przy wykorzystaniu skróconej notacji obiektowej, dzięki czemu ich wykorzystanie jest bardzo proste.

- `enableHighAccuracy` — jest to informacja dla przeglądarki, że jeżeli jest to możliwe, powinna wykorzystać metodę pobierania lokalizacji gwarantującą najwyższą dokładność. Domyślne ustawienie przyjmuje wartość `false`, jednak jego włączenie może nie spowodować żadnej różnicy lub uzyskanie informacji może zająć więcej czasu. Opcji tej należy używać ostrożnie.

■ **Uwaga.** Ustawienie wysokiej dokładności jest tylko przełącznikiem `true/false`. API nie pozwala na definiowanie różnych poziomów dokładności za pomocą wartości numerycznych. Być może zostanie to zmienione w kolejnych wersjach specyfikacji.

- `timeout` — wartość podawana jest w milisekundach — oznacza maksymalny czas pobierania danych geolokalizacyjnych. Jeżeli obliczenia nie zostaną zakończone w podanym przedziale czasu, wywoływana jest funkcja obsługująca błędy. Domyślnie limit nie jest ustawiany.
- `maximumAge` — wartość ta określa, po jakim czasie dane geolokalizacyjne muszą być powtórnie przeliczone — podawana jest w milisekundach. Domyślna wartość to 0, co oznacza, że przeglądarka musi przeliczyć lokalizację natychmiast.

■ **Uwaga.** Być może zastanawiasz się, jaka jest różnica pomiędzy `timeout` a `maximumAge`. Mimo że ich nazwa jest podobna, ich zastosowanie jest różne. Wartość `timeout` odnosi się do czasu potrzebnego na pobranie danych, natomiast `maximumAge` określa *częstotliwość* pobierania lokalizacji. Jeżeli czas pobierania lokalizacji będzie dłuższy niż podany w parametrze `timeout`, wywoływany jest błąd. Jeżeli jednak przeglądarka nie posiada aktualnej lokalizacji (młodszej niż wynikająca z parametru `maximumAge`), będzie musiała pobrać ją ponownie. Możliwe jest przypisanie specjalnych wartości: ustawienie wartości na `0` powoduje jej każdorazowe pobranie, ustawienie na `Infinity` oznacza, że wartość nigdy nie będzie powtórnie pobierana.

Geolocation API nie pozwala Ci na poinstruowanie przeglądarki, jak często ma aktualizować pozycję, leży to całkowicie w jej gestii. Jedyne, co możemy zrobić, to przekazać jej parametry `maximumAge`. Nie możemy niestety dokładnie kontrolować czasu pobierania danych geolokalizacyjnych.

Zmodyfikujmy wcześniejsze wywołanie, dodając opcje przy wykorzystaniu skróconej notacji obiektowej:

```
navigator.geolocation.getCurrentPosition(updateLocation, handleLocationError, {timeout:10000});
```

Nowe wywołanie gwarantuje, że jeżeli pobieranie lokalizacji będzie trwało dłużej niż 10 sekund (10 000 milisekund), wywołany zostanie błąd — funkcja `handleLocationError` zostanie uruchomiona z kodem błędu `timeout`. Możemy połączyć powyższe fragmenty kodu w celu wyświetlenia danych na stronie (rysunek 5.4).

Ciągła aktualizacja lokalizacji

Czasami musisz cyklicznie powtarzać pobieranie lokalizacji. Dzięki projektantom Geolocation API modyfikacja aplikacji w celu ciągłego pobierania danych geolokalizacyjnych jest niezwykle prosta. Wystarczy tylko podmienić wywołanie głównej metody, jak pokazano w następujących przykładach.

Rysunek 5.4. *Dane geolokalizacyjne wyświetlane w telefonie komórkowym*

Jednorazowa aktualizacja:

```
navigator.geolocation.getCurrentPosition(updateLocation, handleLocationError);
```

Ciągła aktualizacja:

```
navigator.geolocation.watchPosition(updateLocation, handleLocationError);
```

Ta prosta zmiana spowoduje, że usługa geolokalizacji będzie wywoływała metodę `updateLocation`, jeżeli lokalizacja klienta ulegnie zmianie. Działa, jakby aplikacja *obserwowała* lokalizację klienta — zostaniesz poinformowany, jeżeli dane geograficzne się zmienią.

Po co miałbyś z tego korzystać?

Wyobraź sobie aplikację, która podaje kolejne wskazówki dotarcia do jakiegoś miejsca, lub stronę, która jest ciągle aktualizowana, pokazując Ci najbliższe stacje benzynowe, podczas gdy poruszasz się po mieście lub jedziesz autostradą. Można nawet stworzyć aplikację, która zapisuje Twoje pozycje i pozwoli Ci później prześledzić przebytą trasę. Wszystkie te aplikacje są łatwe do napisania, jeżeli przeglądarka będzie automatycznie informować o zmianach lokalizacji.

Wyłączenie aktualizacji jest równie proste. Jeżeli aplikacja przestanie potrzebować aktualnej pozycji użytkownika, wystarczy wywołać funkcję `clearWatch()`, zgodnie z następującym przykładem:

```
navigator.geolocation.clearWatch(watchId);
```

Funkcja ta poinformuje usługę geolokalizacji, że już nie potrzebujesz ciągłych aktualizacji lokalizacji. Czym jednak jest zmienna `watchId` i skąd się wzięła? Jest to wartość zwracana z funkcji `watchPosition()`. Zwraca identyfikator usługi obserwacji lokalizacji, dzięki czemu możemy z niej zrezygnować. Jeżeli więc Twoja aplikacja będzie musiała zrezygnować z ciągłej aktualizacji, powinieneś utworzyć kod podobny do tego z listingu 5.5.

Listing 5.5. *Wykorzystanie funkcji watchPosition*

```
var watchId = navigator.geolocation.watchPosition(updateLocation, handleLocationError);
// wykonanie operacji aktualizujących pozycję!
// teraz jesteśmy gotowi do zatrzymania usługi aktualizacji
navigator.geolocation.clearWatch(watchId);
```

Budowa aplikacji wykorzystującej geolokalizację

Do tej pory skupialiśmy się na jednokrotnym pobieraniu lokalizacji. Zobaczmy, co możemy osiągnąć, wykorzystując ciągłą aktualizację odległości — zbudujemy małą, lecz użyteczną aplikację: stronę pozwalającą na śledzenie odległości.

Jeżeli kiedykolwiek potrzebowałeś informacji, jaką odległość przebyłeś w danym czasie, wykorzystywałeś prawdopodobnie urządzenie dedykowane, na przykład nawigację GPS lub krokomierz. Dzięki wykorzystaniu geolokalizacji możesz stworzyć aplikację, która zmierzy odległość, jaką przebyłeś od momentu jej uruchomienia w przeglądarce. Aplikacja nie będzie zbyt użyteczna dla użytkowników komputerów stacjonarnych, jednak dla milionów właścicieli telefonów wspierających geolokalizację będzie idealnym narzędziem. Wystarczy uruchomić naszą przykładową aplikację, udzielić pozwolenia na dostęp do informacji o lokalizacji, a co kilka minut przebyta odległość zostanie zaktualizowana i doliczona do wartości sumarycznej (rysunek 5.5).

Rysunek 5.5. *Przykład aplikacji wykorzystującej geolokalizację*

Przykład wykorzystuje omawianą wcześniej metodę `watchPosition()`. Za każdym razem, kiedy zostanie przekazana nowa lokalizacja, porównamy ją z poprzednią i obliczymy przebytą odległość. Do tego celu wykorzystamy funkcję haversin, pozwalającą obliczyć odległość między dwoma punktami na kuli, reprezentowanymi przez długość i szerokość. Listing 5.6 przedstawia wzór.

Listing 5.6. *Funkcja haversin*

$$d = 2R\arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right)+\cos\varphi_1\cos\varphi_2\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)$$

Jeżeli chciałbyś dowiedzieć się, jak działa ta funkcja, to się zawiedziesz. Zamiast tego pokażemy jej implementację w JavaScripcie, którą możesz wykorzystać do obliczenia odległości pomiędzy dwoma lokalizacjami (listing 5.7).

Listing 5.7. *Implementacja funkcji haversin*

```
Number.prototype.toRadians = function() {
    return this * Math.PI / 180;
}
function distance(latitude1, longitude1, latitide2, longitude2) {
    // R oznacza promień Ziemi w kilometrach
    var R = 6371;

    var deltaLatitude = (latitude2- latitude 1).toRadians();
    var deltaLongitude = (longitude2-longitude1).toRadians();
    latitude = latitude1.toRadians(),latitude2 = latitude2.toRadians();

    var a = Math.sin(deltaLatitude/2) *
      Math.sin(deltaLatitude/2) +
      Math.cos(latitude1) *
      Math.cos(latitude2) *
      Math.sin(deltaLongitude/2) *
      Math.sin(deltaLongitude/2);
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    var d = R * c;
    return d;
}
```

Jeżeli chcesz się dowiedzieć, czy lub dlaczego ten wzór działa, zajrzyj do podręcznika matematyki z liceum. Na nasze potrzeby dokonaliśmy konwersji ze stopni na radiany oraz stworzyliśmy funkcję `distance()` wyliczającą odległość pomiędzy dwoma punktami.

Jeżeli pozycja użytkownika będzie sprawdzana w regularnych i krótkich odstępach czasu, suma obliczanych odległości da w dobrym przybliżeniu długość przebytej trasy. Na potrzeby przykładu zakładamy, że użytkownik pomiędzy poszczególnymi pomiarami przemieszcza się w linii prostej.

Stworzenie kodu HTML

Zacznijmy od kodu HTML — jest stosunkowo prosty, ponieważ tym, co nas naprawdę interesuje, jest kod JavaScript. Na stronie wyświetlane są informacje o pozycji użytkownika. Dodatkowo umieścimy też informację pozwalającą użytkownikowi na sprawdzenie, jaką odległość przebył (listing 5.8).

Listing 5.8. *Kod HTML dla aplikacji mierzącej przebytą odległość*

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" >
    <title>Geolokalizacja</title>
    <link rel="stylesheet" href="geo-html5.css" >
</head>

<body onload="loadDemo()">

    <header>
        <h1>Przebyta trasa</h1>
        <h4>Aktualne informacje!</h4>
```

```
    </header>

    <div id="container">

    <section>
        <article>
            <header>
                <h1>Twoja lokalizacja</h1>
            </header>

            <p class="info" id="status">Twoja przeglądarka nie wspiera geolokalizacji.</p>

            <div class="geostatus">
                <p id="latitude">Szerokość: </p>
                <p id="longitude">Długość: </p>
                <p id="accuracy">Dokładność: </p>
                <p id="timestamp">Stempel czasowy: </p>
                <p id="currDist">Aktualnie przebyta odległość: </p>
                <p id="totalDist">Całkowita przebyta odległość: </p>
            </div>
        </article>
    </section>

    <footer>
        <h2>Zasilane przez HTML5 i Twoje stopy!</h2>
    </footer>

    </div>
    .
    .
    .
</body>
</html>
```

Na razie wstawione zostały wartości domyślne, które zostaną zastąpione wartościami pobranymi z API.

Przetwarzanie danych geolokalizacyjnych

Pierwszy fragment kodu JavaScript jest już znany. Po załadowaniu strony uruchamiany funkcję `loadDemo()`. Skrypt sprawdzi, czy geolokalizacja jest wspierana, i za pośrednictwem funkcji aktualizujących status wyświetli wynik sprawdzenia na stronie. Następnie rozpocznie obserwację lokalizacji użytkownika (listing 5.9).

Listing 5.9. *Funkcja loadDemo() oraz funkcje aktualizujące status*

```
var totalDistance = 0.0;
var lastLat;
var lastLong;

function updateErrorStatus(message) {
    document.getElementById("status").style.background = "papayaWhip";
    document.getElementById("status").innerHTML = "<strong>Błąd</strong>: " + message;
}

function aktualizujStatus(message) {
    document.getElementById("status").style.background = "paleGreen";
    document.getElementById("status").innerHTML = message;
}
```

```
function loadDemo() {
    if(navigator.geolocation) {
        document.getElementById("status").innerHTML = "Geolokalizacja jest wspierana.";
        navigator.geolocation.watchPosition(updateLocation, handleLocationError, {maximumAge:20000});
    }
}
```

Zauważ, że wykorzystaliśmy ustawienie `{maximumAge:20000}` — jest to informacja dla usługi geolokalizacji, że nie interesują nas dane starsze niż 20 sekund (20 000 milisekund). Dzięki temu dane będą regularnie aktualizowane; możesz poeksperymentować z różnymi wartościami atrybutu.

Na potrzeby obsługi błędów wykorzystamy funkcję, którą stworzyliśmy we wcześniejszej części rozdziału — jest wystarczająco ogólna, abyśmy mogli jej użyć. Jeżeli wystąpi błąd, wewnątrz funkcji sprawdzimy jego kod i zaktualizujemy informacje na stronie (listing 5.10).

Listing 5.10. *Kod obsługujący błędy*

```
function handleLocationError(error) {
    switch(error.code){
        case 0:
            updateErrorStatus("Podczas pobierania lokalizacji wystąpił błąd: " + error.message);
            break;
        case 1:
            updateErrorStatus("Użytkownik nie zezwolił na dostęp do danych geolokalizacyjnych.");
            break;
        case 2:
            updateErrorStatus("Przeglądarka nie była w stanie uzyskać informacji o lokalizacji: " +
                              error.message);
            break;
        case 3:
            updateErrorStatus("Maksymalny dozwolony czas pobierania danych został przekroczony.");
            break;
    }
}
```

Większość pracy będzie wykonywała funkcja `updateLocation()`. Wewnątrz tej funkcji zaktualizujemy wartości na stronie zgodnie z aktualnymi danymi oraz obliczymy przebytą odległość (listing 5.11).

Listing 5.11. *Funkcja updateLocation*

```
function updateLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var accuracy = position.coords.accuracy;
    var timestamp = position.timestamp;

    document.getElementById("latitude").innerHTML = "Szerokość: " + latitude;
    document.getElementById("longitude").innerHTML = "Długość: " + longitude;
    document.getElementById("accuracy").innerHTML = "Dokładność: " + accuracy + " metrów";
    document.getElementById("timestamp").innerHTML = "Stempel czasowy: " + timestamp;
```

Pierwszą rzeczą, jaką robimy po uzyskaniu nowego zestawu współrzędnych, jest ich zapisanie. Pobieramy szerokość, długość, dokładność i stempel czasowy, a następnie wyświetlamy te informacje na stronie.

W swojej aplikacji możesz nie wyświetlać stempla czasowego. Data w takiej formie jest użyteczna przede wszystkim dla komputerów — dla użytkownika taki zapis jest niezrozumiały. Możesz zastąpić ją datą w bardziej czytelnej formie lub usunąć.

Dokładność podawana jest w metrach i na pierwszy rzut oka może się wydawać niepotrzebna. Wartość każdych danych jest jednak zależna od ich dokładności. Nawet jeżeli nie pokazujesz użytkownikowi, jaka jest dokładność, powinieneś brać ją pod uwagę podczas obliczeń. Wyświetlanie niedokładnych wartości mogłoby

dać użytkownikowi błędne informacje na temat lokalizacji. W związku z tym wszystkie aktualizacje z małą dokładnością nie będą w naszym kodzie brane pod uwagę (listing 5.12).

Listing 5.12. *Ignorowanie niedokładnych danych*

```
// test zdrowego rozsądku... nie obliczamy odległości,
// jeżeli wartość dokładności jest zbyt duża
if (accuracy >= 30000) {
    updateStatus("Do obliczenia odległości potrzebne są dokładniejsze wartości.");
    return;
}
```

Najprostszy sposób podróżowania

Brian mówi: Sprawdzanie dokładności danych geolokalizacyjnych jest kluczowe. Nie masz dostępu do metod, z jakich przeglądarka korzysta, aby te dane uzyskać, masz jednak dostęp do informacji o dokładności. Wykorzystaj to!

Siedząc w hamaku w swoim ogrodzie, sprawdziłem swoją lokalizację za pośrednictwem telefonu komórkowego. Zdziwiłem się, ponieważ dowiedziałem się, że przez zaledwie kilka minut przemieściłem się o wiele kilometrów z różnymi prędkościami. Należy pamiętać, że dane są tylko tak dokładne, jak pozwala na to ich źródło.

Na koniec obliczymy przebytą odległość, zakładając, że otrzymaliśmy wcześniej przynajmniej jedną wystarczająco dokładną lokalizację. Zaktualizujemy całkowitą przebytą odległość i wyświetlimy odpowiednią informację na stronie. Aby uzyskać większą przejrzystość interfejsu, warto zaokrąglić obliczone wartości (listing 5.13).

Listing 5.13. *Kod obliczający odległość*

```
    // obliczenie odległości
    if ((lastLat != null) && (lastLong != null)) {
        var currentDistance = distance(latitude, longitude, lastLat, lastLong);

        document.getElementById("currDist").innerHTML =
            "Aktualnie przebyta odległość: " + currentDistance.toFixed(2) + " km";

        totalDistance += currentDistance;
        document.getElementById("totalDist").innerHTML =
            "Całkowita przebyta odległość: " + totalDistance.toFixed(2) + " km";
        updateStatus("Aktualizacja lokalizacji zakończona poprawnie.");
    }

    lastLat = latitude;
    lastLong = longitude;
}
```

I to wszystko. Aby stworzyć aplikację śledzącą lokalizację użytkownika i demonstrującą praktycznie całe API geolokalizacji, potrzebowaliśmy mniej niż 200 linii kodu. Przykład jest mało interesujący na urządzeniach stacjonarnych — wypróbuj go w telefonie wspierającym geolokalizację i sprawdź, jaką odległość przebywasz w ciągu dnia.

Kompletny kod

Na listingu 5.14 przedstawiony został kompletny kod aplikacji.

Listing 5.14. *Kompletny kod aplikacji śledzącej przebytą odległość*

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" >
    <title>Geolokalizacja</title>
    <link rel="stylesheet" href="geo-html5.css" >
</head>

<body onload="loadDemo()">

    <header>
        <h1>Przebyta trasa</h1>
        <h4>Aktualne informacje!</h4>
    </header>

    <div id="container">

    <section>
        <article>
            <header>
                <h1>Twoja lokalizacja</h1>
            </header>

            <p class="info" id="status">Twoja przeglądarka nie wspiera geolokalizacji.</p>

            <div class="geostatus">
                <p id="latitude">Szerokość: </p>
                <p id="longitude">Długość: </p>
                <p id="accuracy">Dokładność: </p>
                <p id="timestamp">Stempel czasowy: </p>
                <p id="currDist">Aktualnie przebyta odległość: </p>
                <p id="totalDist">Całkowita przebyta odległość: </p>
            </div>
        </article>
    </section>

    <footer>
        <h2>Zasilane przez HTML5 i Twoje stopy!</h2>
    </footer>

    </div>

    <script>
        var totalDistance = 0.0;
        var lastLat;
        var lastLong;

        Number.prototype.toRadians = function() {
            return this * Math.PI / 180;
        }

        function distance(latitude1, longitude1, latitude2, longitude2) {
            // R oznacza promień Ziemi w kilometrach
            var R = 6371;

            var deltaLatitude = (latitude2-latitude1).toRadians();
```

```
    var deltaLongitude = (longitude2-longitude1).toRadians();
    latitude1 = latitude1.toRadians(), latitude2 = latitude2.toRadians();
    var a = Math.sin(deltaLatitude/2) *
      Math.sin(deltaLatitude/2) +
      Math.cos(latitude1) *
      Math.cos(latitude2) *
      Math.sin(deltaLongitude/2) *
      Math.sin(deltaLongitude/2);

    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    var d = R * c;
    return d;
}

function updateErrorStatus(message) {
    document.getElementById("status").style.background = "papayaWhip";
    document.getElementById("status").innerHTML = "<strong>Błąd</strong>: " + message;
}

function updateStatus(message) {
    document.getElementById("status").style.background = "paleGreen";
    document.getElementById("status").innerHTML = message;
}

function loadDemo() {
    if(navigator.geolocation) {
        document.getElementById("status").innerHTML = "Geolokalizacja jest wspierana.";
        navigator.geolocation.watchPosition(updateLocation, handleLocationError,
                  {maximumAge:20000});
    }
}

function updateLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var accuracy = position.coords.accuracy;
    var timestamp = position.timestamp;

    document.getElementById("latitude").innerHTML = "Szerokość: " + latitude;
    document.getElementById("longitude").innerHTML = "Długość: " + longitude;
    document.getElementById("accuracy").innerHTML = "Dokładność: " + accuracy + " metrów";
    document.getElementById("timestamp").innerHTML = "Stempel czasowy: " + timestamp;

    // test zdrowego rozsądku... nie obliczamy odległości,
    // jeżeli wartość dokładności jest zbyt duża
    if (accuracy >= 30000) {
        updateStatus("Do obliczenia odległości potrzebne są dokładniejsze wartości.");
        return;
    }

    // obliczenie odległości
    if ((lastLat != null) && (lastLong != null)) {
        var currentDistance = distance(latitude, longitude, lastLat, lastLong);

        document.getElementById("currDist").innerHTML =
            "Aktualnie przebyta odległość: " + currentDistance.toFixed(2) + " km";

        totalDistance += currentDistance;
```

```
                document.getElementById("totalDist").innerHTML =
                    "Całkowita przebyta odległość: " + totalDistance.toFixed(2) + " km";
                updateStatus("Aktualizacja lokalizacji zakończona poprawnie.");
            }

            lastLat = latitude;
            lastLong = longitude;
        }

        function handleLocationError(error) {
            switch(error.code){
                case 0:
                    aktualizujStatus("Podczas pobierania lokalizacji wystąpił błąd: " +
                    error.message);
                    break;
                case 1:
                    aktualizujStatus("Użytkownik nie zezwolił na dostęp do danych
                    geolokalizacyjnych.");
                    break;
                case 2:
                    aktualizujStatus("Przeglądarka nie była w stanie uzyskać informacji
                    o lokalizacji: " + error.message);
                    break;
                case 3:
                    aktualizujStatus("Maksymalny dozwolony czas pobierania danych został
                    przekroczony.");
                    break;
            }
        }
    </script>
</body>
</html>
```

Praktyczne dodatki

Są techniki, które nie pasują do pokazywanych przykładów, ale są bardzo przydatne podczas tworzenia aplikacji w HTML5. Poniżej pokażemy klika prostych, lecz przydatnych technik.

Jaki jest mój status?

Na pewno zauważyłeś już, że znaczna część Geolocation API dotyczy wartości czasowych. To nie powinno dziwić. Techniki sprawdzające lokalizację — triangulacja telefoniczna, GPS, sprawdzanie adresu IP — mogą sprawiać problemy: pobieranie lokalizacji może trwać bardzo długo lub w ogóle się nie zakończyć. Na szczęście API dostarcza narzędzia pozwalające na zbudowanie paska postępu.

Programista, ustawiając opcję `timeout` przy sprawdzaniu danych geolokalizacyjnych, zawiadamia przeglądarkę, że powinna wygenerować błąd, jeżeli operacja przekroczy zadany czas. Podczas sprawdzania lokalizacji możemy pokazać użytkownikowi informację mówiącą o aktualnym postępie. Wyświetlanie rozpoczyna się w momencie wysłania żądania pobrania pozycji, natomiast jego zakończenie jest definiowane przez wartość atrybutu `timeout`, niezależnie od tego, czy żądanie zakończy się sukcesem.

Na listingu 5.15 uruchamiamy licznik aktualizujący dane na stronie nową wartością oznaczającą postęp pobierania lokalizacji.

Listing 5.15. *Monitorowanie statusu pobierania danych*

```
function updateStatus(message) {
```

```
    document.getElementById("status").innerHTML = message;
}

function endRequest() {
    updateStatus("Zakończono.");
}

function updateLocation(position) {
    endRequest();
    // obsłużenie danych geolokalizacyjnych
}

function handleLocationError(error) {
    endRequest();
    // obsłużenie błędów
}

navigator.geolocation.getCurrentPosition(updateLocation,
                                         handleLocationError,
                                         {timeout:10000}); // maksymalny czas pobierania – 10 sekund

updateStatus("Pobieranie danych geolokalizacyjnych...");
```

Przyjrzyjmy się poszczególnym częściom. Jak poprzednio, mamy funkcję aktualizującą status na stronie:

```
function updateStatus(message) {
    document.getElementById("status").innerHTML = message;
}
```

Status podajemy w formie tekstu, jednak w podobny sposób moglibyśmy zbudować bardziej skomplikowaną, graficzną reprezentację postępu (listing 5.16).

Listing 5.16. *Wyświetlenie statusu*

```
navigator.geolocation.getCurrentPosition(updateLocation,
                                         handleLocationError,
                                         {timeout:10000}); // maksymalny czas pobierania – 10 sekund

updateStatus("Pobieranie danych geolokalizacyjnych...");
```

Po raz kolejny wykorzystujemy API geolokalizacji w celu pobrania aktualnej lokalizacji użytkownika, tym razem ustawiliśmy jednak maksymalny czas dla żądania. Po upływie podanego czasu żądanie zakończy się sukcesem lub błędem.

Natychmiast po tym wywołaniu aktualizujemy status na stronie, aby poinformować użytkownika, że pobieranie się rozpoczęło. Po pobraniu lokalizacji lub upływie 10 sekund modyfikujemy status, informując użytkownika o zakończeniu operacji (listing 5.17).

Listing 5.17. *Końcowa aktualizacja statusu*

```
function endRequest() {
    updateStatus("Zakończono.");
}

function updateLocation(position) {
    endRequest();
    // obsłużenie danych geolokalizacyjnych
}
```

Jest to prosty skrypt, łatwo go jednak rozbudować.

Ta metoda jest wygodna w przypadku jednorazowego pobierania lokalizacji. Wtedy programista dokładnie wie, kiedy rozpoczyna się pobieranie — oczywiście w momencie uruchomienia funkcji `getCurrentPosition()`. W przypadku ciągłej aktualizacji lokalizacji (funkcja `watchPosition()`) programista nie ma kontroli nad tym, kiedy rozpoczyna się pobieranie.

Co więcej, czas jest liczony dopiero od momentu udzielenia przez użytkownika zgody na pobieranie. Implementowanie aktualizacji statusu w przypadku ciągłej aktualizacji jest niepraktyczne, ponieważ strona nie jest informowana o momencie, w którym użytkownik udzielił pozwolenia.

Pokaż mnie na mapie

Częstym zadaniem przy korzystaniu z geolokalizacji jest pokazanie lokalizacji użytkownika na mapie — na przykład podczas korzystania z popularnej usługi Mapy Google. Jest to tak popularne, że Google wbudowało możliwości geolokalizacji w swój produkt. Wystarczy nacisnąć przycisk *Pokaż moją lokalizację* (rysunek 5.6); po jego kliknięciu wykorzystane zostanie Geolocation API (jeżeli będzie dostępne) do pobrania i wyświetlenia lokalizacji na mapie.

Rysunek 5.6. *Mapy Google i pobieranie aktualnej lokalizacji*

Możliwe jest także samodzielne wykonanie takiej operacji. Google Maps API zostało zaprojektowane tak, aby pobierać szerokość i długość geograficzną w formie liczb zmiennoprzecinkowych (i nie jest to zbieg okoliczności). W łatwy sposób możesz więc przekazać pobrane wartości do API (listing 5.18). Więcej na ten temat dowiesz się z książki *Beginning Google Maps Applications* (Apress, 2010, wydanie drugie).

Listing 5.18. *Przekazanie lokalizacji do Google Maps API*

```
// Załączenie biblioteki Google Maps
<script src="http://maps.google.com/maps/api/js?sensor=false"></script><script>

// Utworzenie mapy…
var map = new google.maps.Map(document.getElementById("map"));

function updateLocation(position) {
    // przekazanie współrzędnych do Google Maps API
    map.setCenter(new google.maps.LatLng(
                      parseFloat(position.coords.latitude),
                      parseFloat(position.coords.longitude));
```

```
}
navigator.geolocation.getCurrentPosition(updateLocation, handleLocationError);
```

Podsumowanie

W tym rozdziale omówiliśmy geolokalizację. Dowiedziałeś się, z czego składają się dane geolokalizacyjne (szerokość, długość i inne atrybuty) oraz z jakich źródeł mogą pochodzić. Dyskutowaliśmy także na temat problemów z prywatnością związanych z geolokalizacją. Pokazaliśmy również, jak, używając API, tworzyć aplikacje wykorzystujące informacje o lokalizacji użytkownika.

W następnym rozdziale pokażemy, w jaki sposób HTML5 pozwala na komunikację pomiędzy zakładkami i oknami, a także pomiędzy stronami i serwerami działającymi w różnych domenach.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

Ś

T

U

V

W

X

Z

Ź

HTML5. Zaawansowane programowanie

HTML5 szturmem zdobywa rynek stron (a w zasadzie już aplikacji) WWW. Dziś ten odświeżony, bogaty w nowe możliwości język ma ogromny potencjał. Dlatego warto poświęcić mu trochę czasu i zacząć tworzyć jeszcze bardziej zaawansowane strony. To na pewno się opłaci!

Dzięki tej książce poznasz najbardziej zaawansowane możliwości HTML5. Nauczysz się wykorzystywać mechanizm geolokalizacji i będziesz w stanie dostarczać użytkownikom spersonalizowane treści w zależności od ich miejsca pobytu. Ponadto przekonasz się, jaki potencjał kryje element *canvas*. Za jego pomocą stworzysz niezwykle atrakcyjny interfejs bez konieczności stosowania dodatków do przeglądarki. W dalszych rozdziałach sprawdzisz, jak prosta może być komunikacja przeglądarki z serwerem za pośrednictwem WebSockets oraz jak zachować funkcjonalność Twojej aplikacji w przypadku braku połączenia z siecią. Dzięki tej książce zdążysz na czas opanować nowości z HTML5 i wykorzystać je w Twoim kolejnym projekcie!

Sięgnij po tę książkę i:

- dostarcz użytkownikowi odpowiednią zawartość strony w zależności od jego lokalizacji
- przechowuj niezbędne dane w bazie danych przeglądarki
- twórz atrakcyjny interfejs dzięki elementowi *canvas*
- zastosuj najnowsze technologie!

WYKORZYSTAJ MOC NAJNOWSZYCH TECHNOLOGII WCHODZĄCYCH W SKŁAD HTML5!

Apress

sięgnij po WIĘCEJ

KOD KORZYŚCI

Nr katalogowy: 10476

Księgarnia internetowa: http://helion.pl

helion.pl księgarnia internetowa

Zamówienia telefoniczne: 0 801 339900 0 601 339900

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

Helion

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

Cena 59,00 zł

ISBN 978-83-246-4809-2

9 788324 648092

Informatyka w najlepszym wydaniu