Head Rush Ajax

Autor: Brett McLaughlin
Tłumaczenie: Piotr Rajca
ISBN: 978-83-246-0556-8
Tytuł oryginału: Head Rush Ajax
Format: B5, stron: 440

Poznaj technologię Ajax w szybki i niekonwencjonalny sposób!

- Napisz wydajny kod JavaScript generujący żądania asynchroniczne
- Dowiedz się, czym jest obiektowy model dokumentu
- Opanuj tajniki przetwarzania plików XML
- Twórz własne aplikacje w technologii Ajax

Jeśli masz dosyć czekania na przeładowanie strony po każdym kliknięciu łącza, zastanawiasz się, dlaczego potwierdzanie transakcji w sieci trwa tak długo i chcesz, aby tworzone przez Ciebie aplikacje internetowe były pozbawione tych irytujących cech, naprzeciw wychodzi Ci technologia Ajax. Jest to połączenie języka JavaScript i XML, dzięki któremu strony WWW działają naprawdę błyskawicznie. Jeśli obawiasz się, że nauka tej technologii wiąże się z koniecznością zaopatrzenia się w kilka opasłych tomisk, z analizowaniem przykładów liczących setki linii kodu i żmudnym zapamiętywaniem dziesiątek parametrów, jesteś w błędzie. Sięgnij po „Head Rush Ajax", otwórz swój umysł i przekonaj się, że nauka może być świetną zabawą!

Książka „Head Rush Ajax" to niezwykły podręcznik, za którego pomocą Ajax odsłoni przed Tobą wszystkie swoje sekrety. Autor książki, korzystając z najnowszych odkryć dotyczących metod przekazywania wiedzy, przedstawi Ci wszystkie zagadnienia, które są niezbędne, aby projektować i budować wydajne aplikacje sieciowe. Poznasz język JavaScript i nauczysz się pisać asynchroniczne żądania będące podstawą Ajaksa, użyjesz obiektowego modelu dokumentu (DOM) i znaczników XML. Zanim się zorientujesz, zostaniesz ekspertem specjalizującym się w Ajaksie.

- Podstawy technologii Ajax
- JavaScript i żądania asynchroniczne
- Aplikacje oparte na DOM
- Tworzenie interfejsów użytkownika
- Korzystanie z plików XML
- Możliwości technologii JSON
- Obsługa żądań POST

Zapomnij o powolnych witrynach WWW i nudnej nauce.
Zajmij się tworzeniem aplikacji internetowych następnej generacji!

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści (skrócony)

Spis treści (na serio)

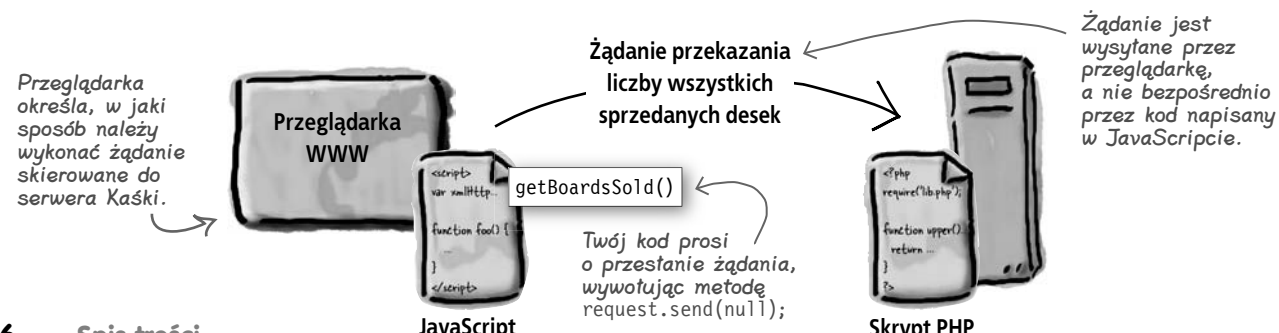
Wprowadzenie

Twój mózg koncentruje się na technologii Ajax. Podczas gdy Ty starasz się czegoś nauczyć, Twój mózg robi Ci przysługę i dba o to, abyś przez przypadek nie zapamiętał zdobywanych informacji. Twój mózg myśli sobie: „Lepiej zostawić trochę miejsca na bardziej istotne informacje, na przykład: jakich zwierząt unikać albo czy jeżdżenie na snowboardzie nago jest dobrym pomysłem". A zatem, w jaki sposób możesz oszukać swój mózg i przekonać go, że Twoje życie zależy od znajomości tworzenia aplikacji asynchronicznych?

Aplikacje internetowe dla nowego pokolenia

1 Zastosowanie technologii Ajax

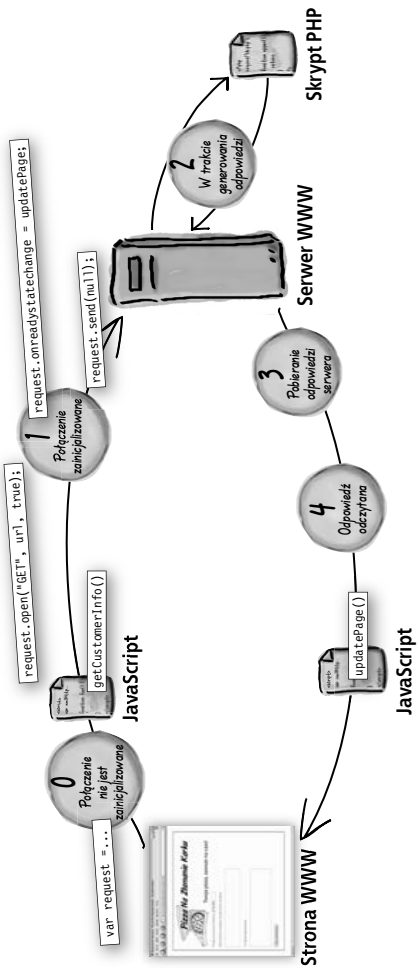
Nadaj nowego blasku swoim aplikacjom internetowym. Czy męczy Cię toporny interfejs aplikacji internetowych i konieczność ciągłego czekania na wyświetlenie kolejnych stron? Cóż, nadszedł zatem czas, byś nadał swoim aplikacjom internetowym tego samego sosnowego zapachu i wyglądu, jaki mają normalne aplikacje. O czym myślimy? Myślimy o najnowszej technologii, jaka pojawiła się w dziedzinie WWW: technologii **Ajax** (ang. *Asynchronous JavaScript and XML*, czyli asynchroniczny JavaScript i XML), która stanowi dla Ciebie przepustkę do tworzenia **wzbogaconych aplikacji internetowych** — bardziej *interaktywnych*, *szybciej reagujących na wykonywane operacje* i *łatwiejszych w obsłudze*. Sięgnij zatem po próbną buteleczkę Ajaksa dołączoną do niniejszej książki: zabieramy się do wypolerowania i nadania nowego blasku Twoim aplikacjom internetowym.



Zgłaszanie żądań w technologii Ajax

2 Zastosowanie odpowiedniego języka

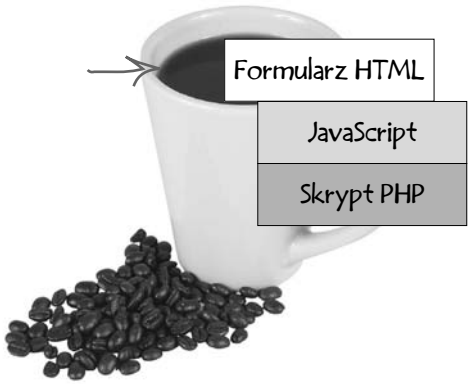
Czas dowiedzieć się, jak należy rozmawiać w sposób asynchroniczny. Jeśli chcesz napisać następną odlotową aplikację, będziesz musiał doskonale zrozumieć najdrobniejsze szczegóły technologii Ajax. W tym rozdziale znajdziesz szczegółowe informacje, a także plotki o pisaniu **asynchronicznych skryptów JavaScript**: dowiesz się, w jaki sposób wysyłać żądania z różnych przeglądarek, opanujesz **stany gotowości** i **kody statusu**, a nawet poznasz kilka dodatkowych sztuczek z repertuaru dynamicznego HTML-a. Kiedy skończysz lekturę tego rozdziału, będziesz zgłaszał żądania i obsługiwał odpowiedzi jak prawdziwy profesjonalista… A swoją drogą, czy wspominałem, że Twoi użytkownicy **nie będą musieli na Ciebie czekać**, kiedy Ty będziesz się uczył Ajaksa?



Aplikacje asynchroniczne

3 Oślepiła mnie asynchronicznością

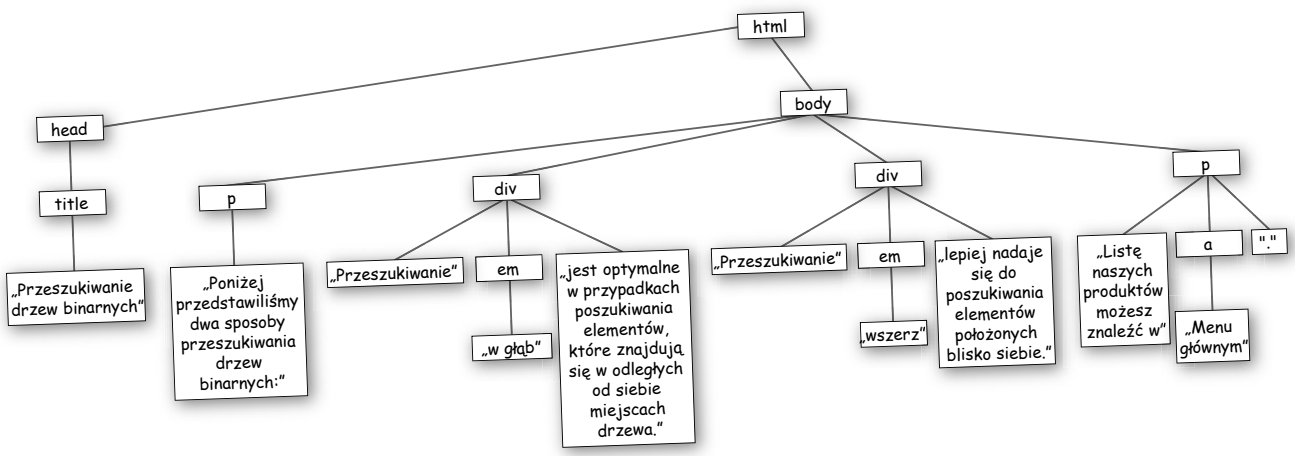
Poczekalnia? Przykro nam, ale u nas coś takiego nie istnieje. To jest WWW, a nie gabinet lekarski i nikt tu nie chce siedzieć i czekać na zakończenie pracy serwera, czytając ilustrowane magazyny sprzed sześciu miesięcy. Zobaczyłeś już, w jaki sposób technologia Ajax pozwoli Ci pozbyć się konieczności odświeżania stron, jednak teraz nadszedł czas, by do listy cech wyróżniających Twoje aplikacje dodać **wrażliwość** i **szybkość reakcji** na działania wykonywane przez użytkownika. W tym rozdziale dowiesz się, w jaki sposób przesyłać żądania użytkownika do serwera i zapewnić mu możliwość dalszego korzystania z aplikacji w trakcie oczekiwania na odpowiedź. Chociaż w sumie… źle się wyraziłem. W tym rozdziale nie będzie żadnego **oczekiwania**.



Obiektowy model dokumentu

4 Dendrologia stron WWW

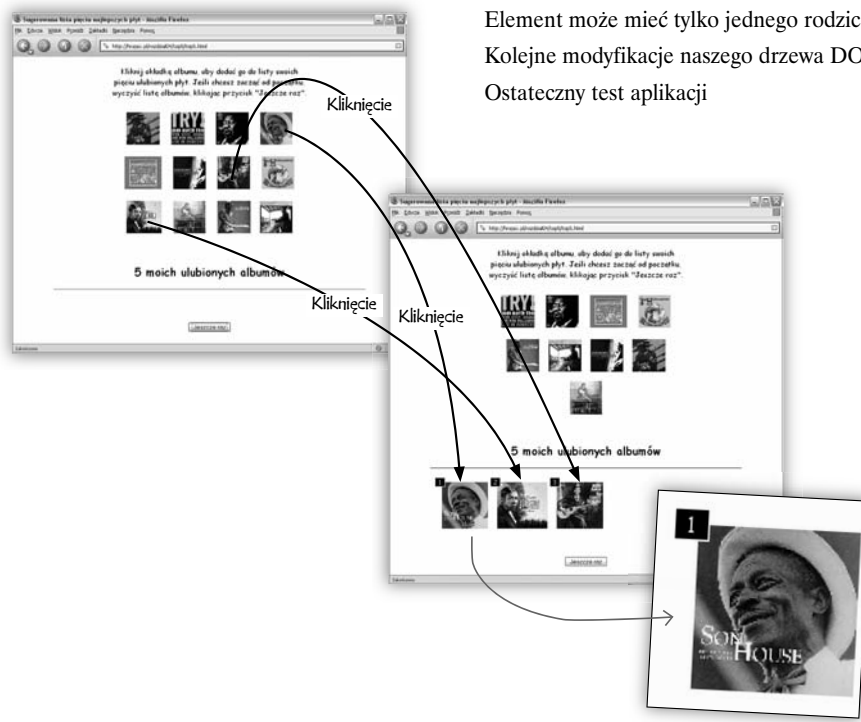
Poszukiwane: strony WWW zapewniające możliwość łatwej modyfikacji swojej zawartości. Czas wziąć sprawy w swoje ręce i rozpocząć pisanie kodu, który będzie dynamicznie modyfikował zawartość stron WWW. Dzięki wykorzystaniu **obiektowego modelu dokumentu** (w skrócie **DOM**, z ang. *Document Object Model*) Twoje strony mogą rozpocząć zupełnie nowe życie, odpowiadać na czynności wykonywane przez użytkowników i pomóc Ci raz na zawsze pozbyć się konieczności tego koszmarnego odświeżania. Kiedy zakończysz lekturę tego rozdziału, będziesz w stanie dodawać, usuwać i aktualizować zawartość stron WWW praktycznie we wszystkich jej miejscach.



Tworzenie aplikacji korzystających z DOM-u

4.5 Dodatkowa pomoc

Z niecierpliwością czekasz na kolejne informacje o DOM? W poprzednim rozdziale przeszedłeś błyskawiczny kurs najlepszego sposobu aktualizacji stron WWW: **obiektowego modelu dokumentu**, w skrócie DOM. Przypuszczam jednak, że chciałbyś dalej zajmować się tym tematem, dlatego też w tym rozdziale wykorzystasz poznane informacje i bazując na nich, napiszesz świetną aplikację wykorzystującą możliwości, jakie daje DOM. Jednocześnie poznasz kilka nowych **procedur obsługi zdarzeń**, dowiesz się, w jaki sposób **zmieniać style węzłów** i tworzyć **dynamiczne**, **przyjazne dla użytkownika** aplikacje internetowe. W tym rozdziale podniesiesz swoje umiejętności w posługiwaniu się DOM-em na zupełnie nowy poziom.



Żądania POST

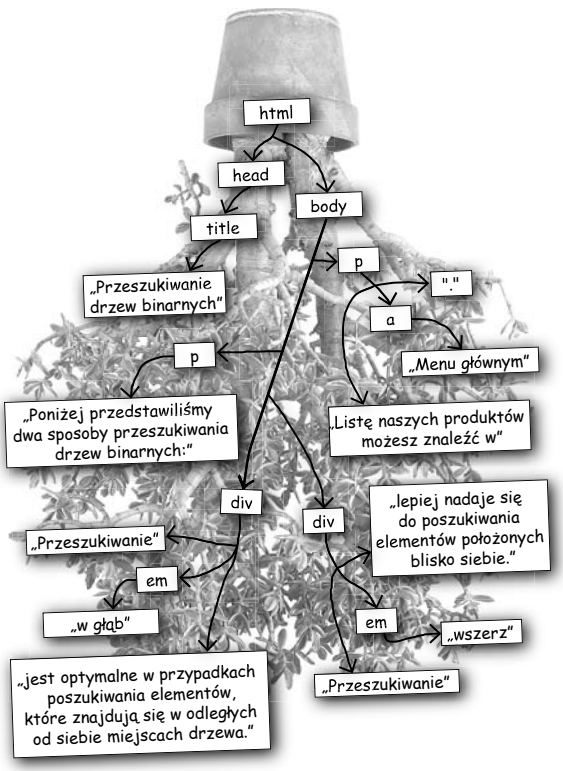
5 Powiedz więcej — użyj metody POST

To właśnie na ten rozdział czekałeś. Prosiłeś o niego i wiedziałeś, że kiedyś się go doczekasz: w końcu **porzucimy** wywołanie `send(null)` i dowiemy się, jak **przesyłać do serwera więcej danych**. Oczywiście będzie to wymagało od nas nieco większego nakładu pracy, jednak kiedy skończysz lekturę tego rozdziału, to będziesz w stanie asynchronicznie przesyłać do serwera znacznie więcej niż „nic". A zatem zapnij pasy, bo rozpoczynamy przejażdżkę po krainie typów zawartości i nagłówków żądań: jesteśmy w kraju rządzonym przez **metodę POST**.

Żądania i odpowiedzi XML

6 Więcej niż mogą wyrazić słowa

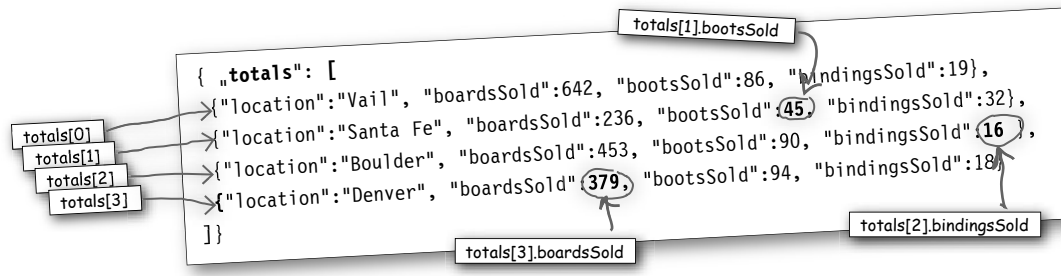
Czy kiedykolwiek miałeś uczucie, że nikt Ciebie nie słucha? Czasami zwykły, ludzki język nie wystarcza, kiedy **próbujesz się z kimś porozumieć**. Jak na razie we wszystkich żądaniach i odpowiedziach używałeś zwyczajnego tekstu. Nadszedł jednak czas, aby wyzwolić się z tej tekstowej klatki. W tym rozdziale **wypłyniemy na szerokie wody XML-a** i nauczymy nasze serwery, w jaki sposób mogą powiedzieć więcej, niż kiedykolwiek mogłyby powiedzieć, używając zwyczajnego tekstu. A jakby tego było mało, nauczysz posługiwania się językiem XML także swoje żądania, choć akurat to nie zawsze będzie dobrym pomysłem (więcej na ten temat dowiesz się w dalszej części rozdziału). Przygotuj się… Kiedy już skończysz lekturę tego rozdziału, *Twoje żądania i odpowiedzi nigdy już nie będą takie same*.



JSON a XML

7 Walka do samego końca

Nadszedł czas na powrót do szkoły podstawowej. Czy pamiętasz te czasy, gdy różnice poglądów były rozwiązywane przy użyciu ostrych słów, pięści i nieudolnych imitacji kung-fu? Kiedy nic bardziej nie pobudzało emocji niż okrzyk **„Biją się!”**, usłyszany w sali jakiejś knajpki? W tym rozdziale mamy zamiar wrócić właśnie do tamtych czasów i zostawić za sobą przyjacielskie słowa i dobre zasady. **XML** i **JSON** — dwa różne formaty zapisu danych, wykorzystywane do wysyłania i odbierania danych w asynchronicznych żądaniach, są już gotowe, by udowodnić swoją wartość na ringu. Przygotuj zatem tabliczki z ocenami i zajmij miejsce przy stoliku.



Dodatek 1.: Dodatki

D.1 Kilka specjalnych dodatków

Specjalnie dla Ciebie: prezent od naszego zespołu Head First Labs. A tak naprawdę w tym dodatku znajdziesz aż **pięć specjalnych dodatków**. Chcielibyśmy móc zostać nieco dłużej i powiedzieć Ci znacznie więcej, jednak nadszedł czas, kiedy musisz samemu wkroczyć w okrutny świat tworzenia aplikacji internetowych, wyposażony jedynie w swoją wiedzę zdobytą podczas lektury niniejszej książki. Nie mogliśmy jednak zostawić Cię bez **niewielkiego dodatkowego wyposażenia**, a zatem przyjrzyj się pięciu najważniejszym sprawom, jakie udało się nam jeszcze wcisnąć do tej książki.

Dodatek 2.: Narzędzia ułatwiające korzystanie z technologii Ajax i obsługę DOM

D.2 Najwyższy czas na małą nagrodę.

W niniejszym dodatku znajdziesz kod, który był nieco zbyt złożony, aby opisywać go we wcześniejszych rozdziałach książki, w miejscach, gdzie go używaliśmy, jednak obecnie powinieneś już być w stanie przeanalizować i zrozumieć działanie wszystkich prezentowanych tu funkcji, ułatwiających korzystanie z technologii Ajax oraz operowanie na DOM.

S Skorowidz 433

Zastosowanie technologii Ajax

Nadaj nowego blasku swoim aplikacjom internetowym. Czyż nie męczy Cię toporny interfejs aplikacji internetowych i konieczność ciągłego czekania na wyświetlenie kolejnych stron? Cóż, nadszedł zatem czas, byś nadał swoim aplikacjom internetowym tego samego sosnowego zapachu i wyglądu, jaki mają normalne aplikacje. O czym myślę? Myślę o najnowszej technologii, która pojawiła się w WWW: technologii Ajax (ang. *Asynchronous JavaScript and XML*, czyli asynchroniczny JavaScript i XML), która stanowi dla Ciebie przepustkę do tworzenia wzbogaconych aplikacji internetowych — bardziej *interaktywnych*, *szybciej reagujących na wykonywane operacje* i *łatwiejszych w obsłudze*. Sięgnij zatem po próbną buteleczkę Ajaksa dołączoną od niniejszej książki: zabieramy się do wypolerowania i nadania nowego blasku Twoim aplikacjom internetowym.

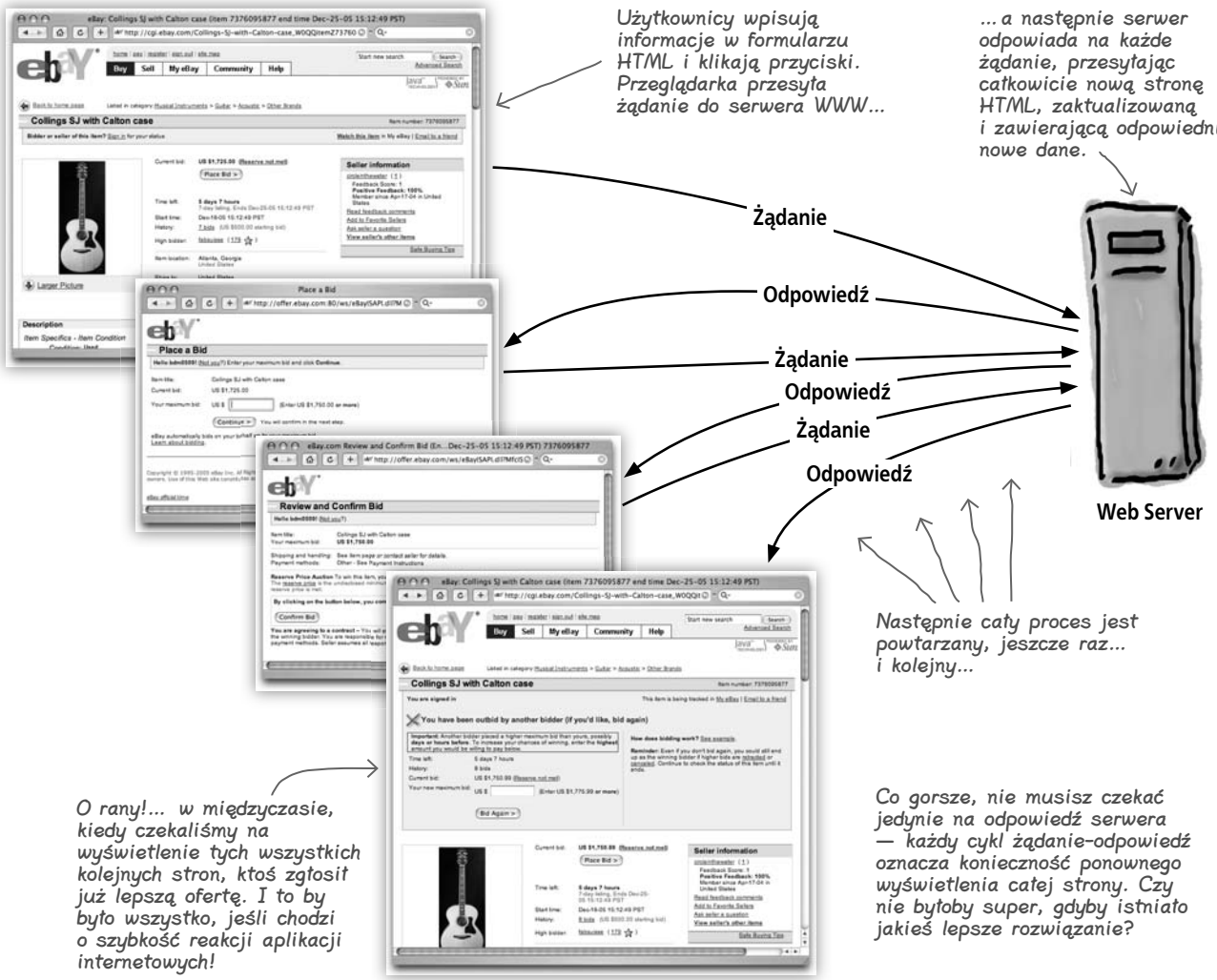
WWW, odświeżona

Prawdę rzekłszy, już w tym rozdziale powiem, w jaki sposób pozbędziesz się konieczności odświeżania i ponownego wyświetlania stron.

Czy klienci są zmęczeni czekaniem na wyświetlenie kolejnych stron podczas składania zamówień w Twojej witrynie? Czy ciągle słyszysz skargi, że każde kliknięcie przycisku powoduje odświeżenie strony? Jeśli właśnie tak się dzieje, to znak, iż nadszedł czas, by coś zrobić z programem — zmodyfikować go i podnieść na wyższy, lepszy poziom. Witam w świecie aplikacji internetowych nowej generacji, w którym **JavaScript**, odrobina **dynamicznego HTML-a** i nieco **XML-a** może nadać Twym aplikacjom internetowym zupełnie nowego, bardzie dynamicznego i interaktywnego charakteru, do tej pory właściwego jedynie zwykłym aplikacjom.

Przyjrzyjmy się aplikacjom internetowym, do których jesteś przyzwyczajony — zarówno Ty, jak i Twoi klienci.

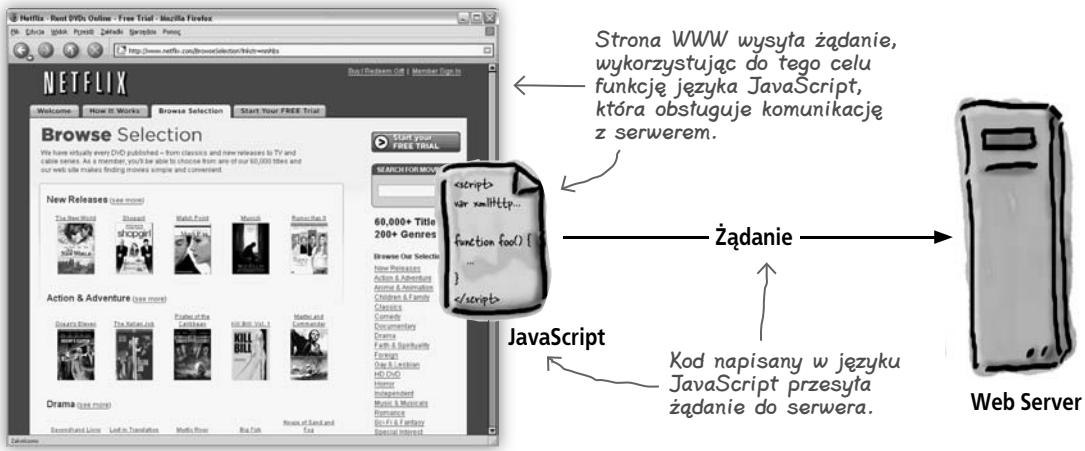
Stare rozwiązanie (pomyśl o roku 1999)



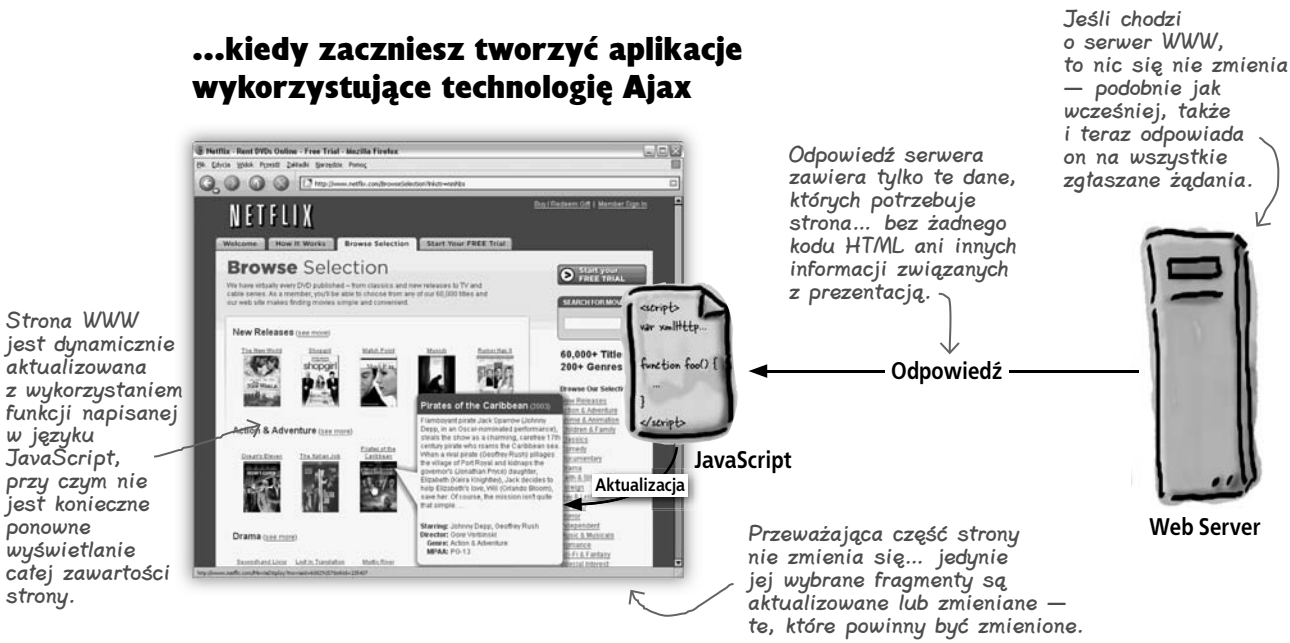
Witamy w nowym tysiącleciu!

Aplikacje wykorzystujące ten stary model żądanie-odpowiedź może tworzyć każdy, jeśli jednak chcesz tworzyć **szybsze aplikacje**, które zapewnią użytkownikom poczucie, że pracują z **normalnym oprogramowaniem biurkowym**, to będziesz potrzebował czegoś nowego — **Ajaksa**, czyli całkowicie nowego podejścia do zagadnień pisania aplikacji internetowych:

Żadnego oczekiwania...



...kiedy zaczniesz tworzyć aplikacje wykorzystujące technologię Ajax



Aplikacje utworzone w technologii Ajax są asynchroniczne. Jeśli jeszcze tego nie zrozumiałeś, to zwróć uwagę, iż w tej książce zajmujemy się wykorzystaniem technologii Ajax do tworzenia odlotowych aplikacji internetowych. Jak na razie, zobaczyłeś, w jaki sposób aplikacje korzystające z Ajaksa mogą wymieniać informacje z serwerem WWW bez konieczności odświeżania i ponownego wyświetlania całej strony, jednak Ajax to znacznie więcej niż jedynie możliwość usprawnienia interfejsu użytkownika.

Podstawowym narzędziem w walce z irytującą koniecznością odświeżania stron jest fakt, iż aplikacje wykorzystujące technologię Ajax mogą prowadzić wymianę danych z serwerem w sposób *asynchroniczny*. Innymi słowy, JavaScript może przesyłać żądanie do serwera i oczekiwać na odpowiedź, a Ty w tym samym czasie możesz dalej wypełniać kolejne pola formularza, a nawet klikać przyciski — w międzyczasie, w tle, serwer będzie realizował swoje zadania. Nieco później, kiedy serwer zakończy pracę, Twój kod może zaktualizować jedynie wybrane fragmenty strony, które uległy zmianie, jednak *użytkownik* nigdy nie czeka. Właśnie na tym polega potęga możliwości **asynchronicznej wymiany danych z serwerem**! Dodaj do tego aktualizację wybranych fragmentów stron bez konieczności ich całkowitego odświeżania, a otrzymasz aplikacje wykorzystujące technologię Ajax.

Nie przejmuj się, jeśli nie do końca rozumiesz, o co w tym wszystkim chodzi — w dalszych rozdziałach książki znacznie bardziej szczegółowo i wyczerpująco zajmę się zagadnieniami programowania asynchronicznego.

Najczęściej zadawane pytania

P. Hm... wszystko mi się pomieszało. Czyli teraz już nie korzystamy z modelu żądanie-odpowiedź?

O. Korzystamy — Twoje strony cały czas zgłaszają żądania i odbierają odpowiedzi. Zmienił się natomiast sposób wykonywania tych żądań oraz ich obsługi — teraz zamiast zwyczajnego przesyłania formularza, będą wykonywane funkcje napisane w języku JavaScript.

P. A dlaczego nie można w standardowy sposób przesłać formularza? Co nam w zasadzie daje zastosowanie technologii Ajax?

O. Kod JavaScriptu tworzący aplikację w technologii Ajax przesyła żądanie do serwera, *lecz nie czeka na otrzymanie odpowiedzi*. Co więcej, po odebraniu odpowiedzi przesłanej przez serwer, kod JavaScriptu może ją obsłużyć bez konieczności odświeżenia i ponownego wyświetlania całej strony.

P. A zatem w jaki sposób strona odbiera odpowiedź z serwera?

O. To właśnie w tym miejscu do akcji wkraczają asynchroniczne mechanizmy technologii Ajax. Kiedy serwer przesyła odpowiedź, kod napisany w języku JavaScript może umieścić na stronie przesłane wartości, zmienić obrazek, a nawet wyświetlić całkowicie nową stronę WWW. A podczas wykonywania tych wszystkich operacji użytkownik nigdy nie będzie musiał czekać.

P. Czy to znaczy, że powinienem używać technologii Ajax do obsługi wszystkich żądań?

O. Nie — wciąż istnieje wiele sytuacji, w których warto stosować tradycyjny model tworzenia aplikacji internetowych. Na przykład, kiedy użytkownik wypełnił wszystkie pola formularza, to można mu udostępnić możliwość kliknięcia przycisku *Wyślij zapytanie* i wysłania całego formularz do serwera, bez wykorzystywania Ajaksa.

Możliwości technologii Ajax powinieneś wykorzystywać do wykonywania i obsługi większości operacji związanych z dynamicznym przetwarzaniem stron, takich jak zmiana obrazków, aktualizacja pól formularzy czy też reagowanie na czynności wykonywane przez użytkownika. Jeśli chcesz zmienić jedynie fragment strony, to koniecznie powinieneś użyć do tego celu technologii Ajax.

P. Wspominałeś coś o XML-u...

O. Czasami kod JavaScript będzie wykorzystywał dane XML do wymiany informacji pomiędzy serwerem a aplikacją, jednak użycie XML-a nie zawsze będzie konieczne. W kolejnych rozdziałach poświęcimy wiele uwagi określaniu, kiedy i w jaki sposób należy korzystać z danych XML.

P. A AJAX jest jedynie akronimem pochodzącym od angielskich słów: Asynchronous JavaScript and XML?

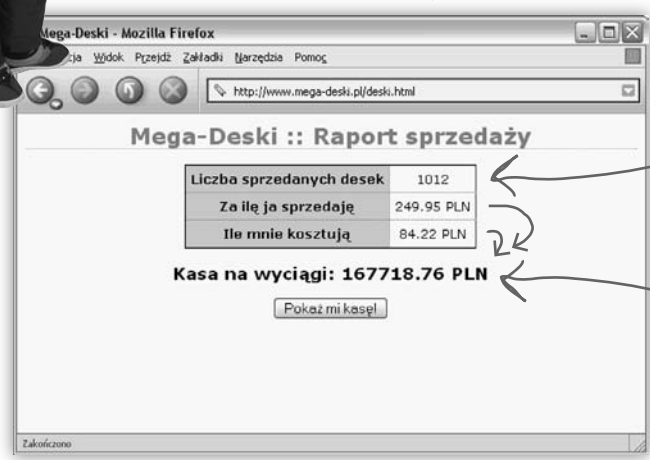
O. Poprawnie należy to zapisywać jako „Ajax" i prawdę rzekłszy, to nie jest akronim. Choć bez wątpienia na taki wygląda... i nawet pasuje... hm, nie pytaj mnie, to nie ja wymyśliłem ten termin!

Jesse James Garrett, który pierwszy użył terminu „Ajax" stwierdził, że nie wymyślił go jako skrótu. I bądź tu człowieku mądrym!

Teraz, kiedy sprzedaję ulepszone przeze mnie deski snowboardowe, na stronie WWW stworzyłam formularz, który prezentuje aktualną liczbę sprzedanych desek, jednak za każdym razem, gdy chcę zaktualizować wyniki, odświeżana jest cała strona wyświetlona w przeglądarce. Czy myślisz, że ten cały Ajax może w tym coś pomóc?

Kaśka, królowa snowboardu i internetowy przedsiębiorca.

Oto aktualna postać aplikacji Kaśki, w której nie jest używana technologia Ajax. Aplikacja ta składa się z prostego formularza, który przesyła żądanie do skryptu PHP.



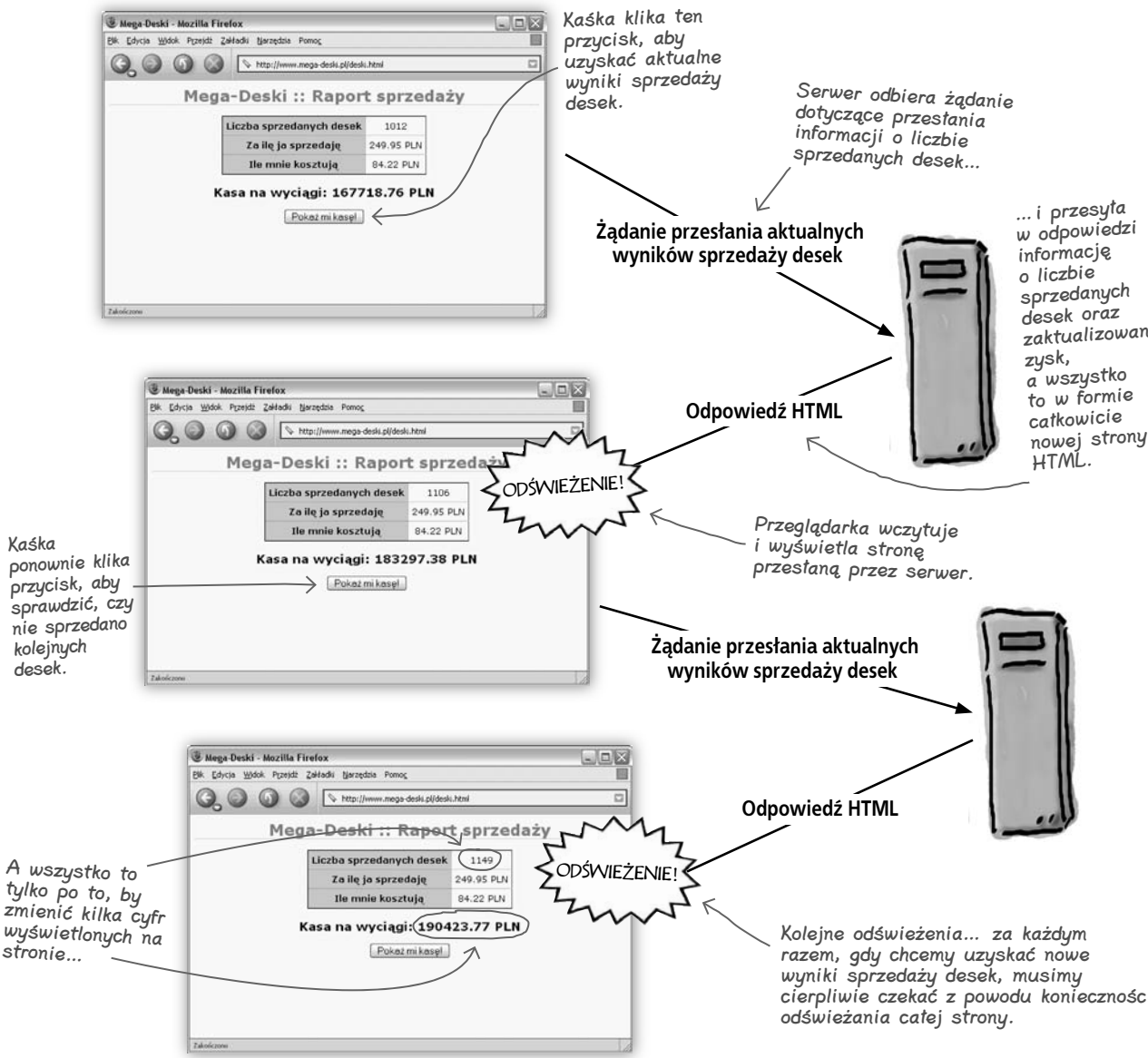
Skrypt PHP sprawdza, ile desek zostało sprzedanych.

Następnie, na podstawie ceny, za jaką Kaśka sprzedaje swoje deski oraz za ile je kupuje, skrypt wylicza ilość zarobionych przez nią pieniędzy.

„Odświeżanie? Nie potrzebujemy żadnego odświeżania!"

Nie ma chyba nic bardziej denerwującego niż aplikacja, która ponownie wyświetla całą stronę za każdym razem, gdy klikniemy jakiś przycisk lub wpiszemy jakąś wartość. W przypadku zestawienia na stronie Kaśki, za każdym razem zmienia się jedynie kilka wyświetlanych cyfr, a pomimo to konieczne jest odświeżenie i ponowne wyświetlenie całej strony.

W pierwszej kolejności określmy, dlaczego strona jest odświeżana…



Ajax spieszy na ratunek

Czy widzisz, na czym polega problem? Za każdym razem, gdy Kaśka chce uzyskać aktualne informacje o liczbie sprzedanych desek, konieczne jest ponowne wyświetlanie całej strony, a Kaśka musi znosić internetową wersję śnieżnej ślepoty.

Użyj technologii Ajax, aby usprawnić generowanie raportu ze sprzedaży...

Zmodyfikujmy zatem stronę z raportem ze sprzedaży w taki sposób, aby przesyłała ona żądania o aktualne informacje z wykorzystaniem technologii Ajax. Następnie będziemy mogli odebrać odpowiedź z serwera i zaktualizować stronę za pomocą JavaScriptu i dynamicznego HTML-a. Żadnego odświeżania strony… i Kaśka ponownie będzie radosną snowboarderką.

A może nawet zarobimy darmową deskę...

Rozpracuj to

Aby przerobić raport ze sprzedaży Kaśki na aplikację wykorzystującą technologię Ajax, będziesz potrzebował kilku funkcji napisanych w języku JavaScript. Poniżej podałem nazwy trzech funkcji. Narysuj linię łączącą nazwy funkcji z opisami określającymi, jakie będzie znaczenie danej funkcji w końcowej wersji aplikacji.

`getBoardsSold()`	Utworzenie nowego obiektu służącego do komunikacji z serwerem.
`updatePage()`	Przesłanie do serwera żądania o aktualne informacje o liczbie sprzedanych desek.
`createRequest()`	Wyświetlenie na stronie aktualnych informacji o liczbie desek, jakie udało się sprzedać Kaśce oraz o wysokości zarobionej kwoty.

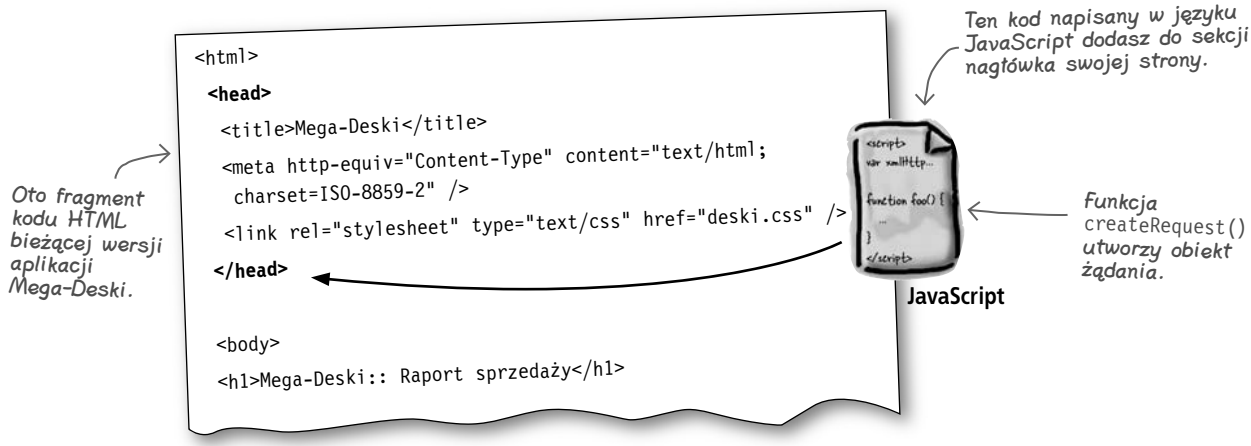
Odpowiedź znajduje się *na stronie 89.*

Modyfikacja strony raportu Kaśki

A zatem zastosujmy technologię Ajax do usprawnienia strony prezentującej wyniki sprzedaży internetowego sklepu Kaśki. Dzięki zastosowaniu Ajaksa możemy uniknąć konieczności odświeżania stron i znacząco ograniczyć ilość danych, jakie serwer musi przesłać w celu wyświetlenia aktualnego raportu. Poniżej opisałem, czym się zajmiesz podczas lektury dalszej części tego rozdziału:

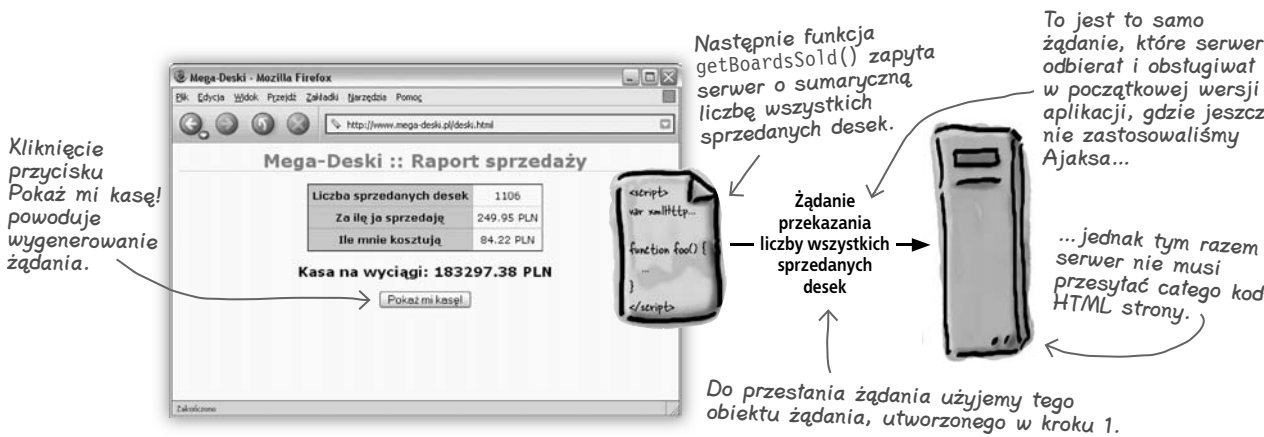
❶ Utworzysz nowy obiekt, który będzie przesyłał żądania do serwera.

W pierwszej kolejności będziesz potrzebował funkcji, napisanej w języku JavaScript, służącej do utworzenia obiektu, który pozwoli na przesyłanie żądań i odbieranie odpowiedzi serwera. Tej funkcji nadamy nazwę **`createRequest()`**.



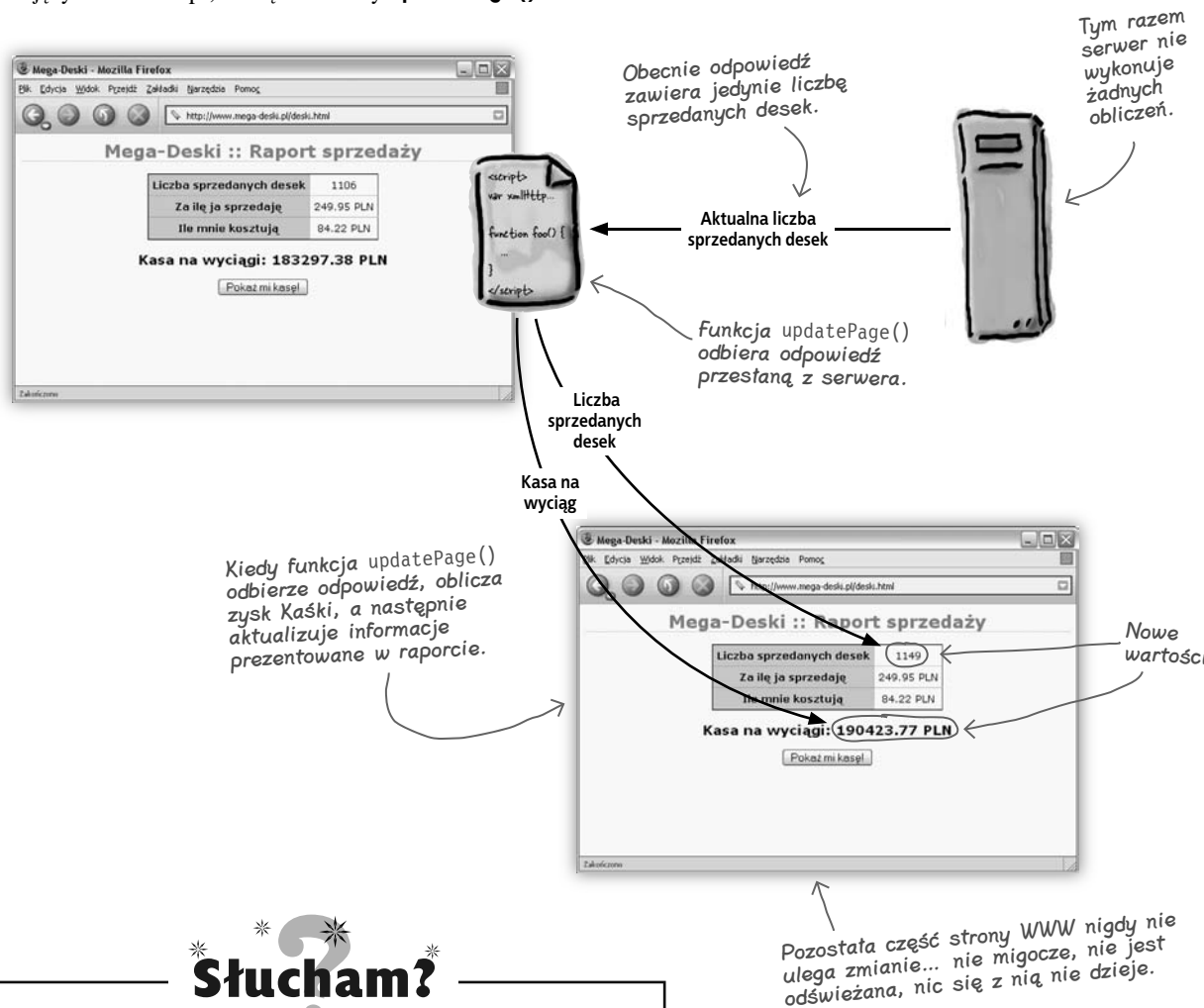
❷ Napiszesz funkcję, która będzie żądała od serwera przesłania aktualnych informacji o sprzedaży.

Następnie użyjesz obiektu utworzonego w kroku 1. do przesłania do serwera WWW żądania. Niezbędny kod umieścimy w kolejnej funkcji JavaScriptu, której nadamy nazwę **`getBoardsSold()`**. Funkcja ta będzie wywoływana w momencie, gdy Kaśka kliknie przycisk *Pokaż mi kasę!*.



❸ Zaktualizujesz raport sprzedaży Kaśki, używając kolejnej funkcji do wyświetlenia bieżących informacji.

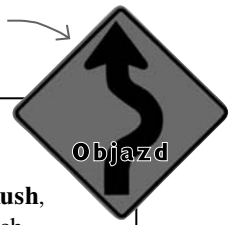
Teraz możesz już zaktualizować raport i wyświetlić w nim bieżące informacje na temat liczby sprzedanych desek oraz osiągniętego zysku. Do tego celu zastosujemy kolejną funkcję, napisaną w języku JavaScript, którą nazwiemy **`updatePage()`**.



Słucham?

Nie przejmuj się, jeśli nie rozumiesz wszystkiego, co napisałem... zwłaszcza informacji dotyczących żądań. Wszystkie te zagadnienia zostaną jeszcze szczegółowo opisane i wyjaśnione w dalszej części książki. Jak na razie, postaraj się zrozumieć podstawowe idee działania aplikacji korzystających z technologii Ajax. Zwróć szczególną uwagę na wykorzystanie funkcji JavaScriptu do zgłaszania żądań oraz na to, że serwer zwraca tylko konkretne informacje (w naszym przypadku — jedną liczbę), a nie cały kod HTML strony.

Już niedługo zabierzemy się za pisanie kodu. Najpierw jednak zrobimy mały „objazd" i zajmiemy się czym innym...



Film utrwalający: Rozdział 1.

Zapewne pomyślałeś sobie: „Co do diabła robi *Film utrwalający* prawie na samym początku rozdziału? To chyba nie jest normalne". I masz rację… jednak to jest **książka z serii Head Rush**, **pamiętasz?** Zanim zaczniesz dalszą lekturę, zatrzymaj się i przeczytaj na głos każdą z podanych poniżej uwag.

Następnie otwórz kolejną stronę i przygotuj się do wczytania podanych tu informacji do swojego mózgu. Każde z podanych zagadnień szczegółowo opiszemy i wyjaśnimy w dalszej części rozdziału.

- Aplikacje asynchroniczne wykonują żądania przy wykorzystaniu obiektu JavaScriptu, a nie poprzez przesyłanie formularza.
- Żądania i odpowiedzi są obsługiwane przez przeglądarkę, a nie bezpośrednio przez kod JavaScriptu.
- Kiedy przeglądarka odbierze odpowiedź na Twoje asynchroniczne żądanie, wywoła wskazaną przez Ciebie funkcję JavaScriptu, przekazując do niej odpowiedź otrzymaną z serwera.

Nie żartuję... nie zabieraj się za czytanie następnej strony, dopóki nie przeczytasz NA GŁOS trzech powyższych uwag. Uwierz mi, nikt sobie nie pomyśli, że Ci z lekka odbiło (no, a jeśli nawet ktoś taki się znajdzie, to jednocześnie będzie pod wrażeniem Twojej mądrości i ogromnej wiedzy!).

Ściąga z HTML-a

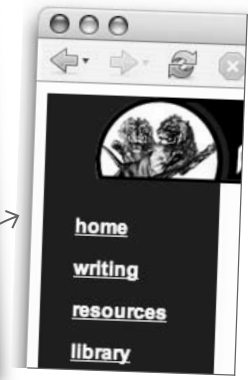
Czy czujesz, że przydałoby Ci się odświeżyć informacje na temat elementów ** ** i ****? Na następnej stronie zajmiemy się tworzeniem kodu HTML, a zatem, zanim tam dotrzesz, zamieszczam krótkie przypomnienie z podstawowymi informacjami o tych dwóch najciekawszych elementach HTML, z jakimi możesz się spotkać.

Element ** ** jest pojemnikiem, w którym można umieszczać inne, powiązane ze sobą elementy HTML i który pozwala na określanie ich wyglądu przy użyciu jednej reguły CSS.

```
<div id="menu">
  <a href="home.html">home</a>
  <a href="books.html">writing</a>
  <a href="links.html">resources</a>
  <a href="lib.html">library</a>
</div>
```

Identyfikator elementu możesz wykorzystać w arkuszu stylów CSS, aby w jednej regule określić postać całej zawartości konkretnego elementu.

Stosuj element , aby grupować elementy o podobnym przeznaczeniu.



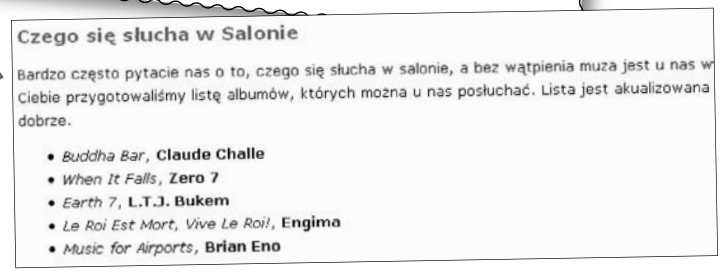
Elementy **** pozwalają na wyróżnianie fragmentów tekstu. Postać tych elementów można w prosty sposób określać przy użyciu stylów CSS.

```
<ul>
  <li><span class="cd">Buddha Bar</span>,
  <span class="artysta">Claude Challe</span></li>

  <li><span class="cd">When It Falls</span>,
  <span class="artysta">Zero 7</span></li>
```

Elementy mogą służyć do wydzielania fragmentów tekstu, jednak nie powodują utworzenia nowych akapitów lub bloków.

Elementy nie tworzą nowych bloków tekstu, choć zapewniają możliwość określania wyglądu swojej zawartości przy użyciu stylów CSS.



Prezentacja kodu HTML strony Mega-Deski

Zacznijmy od rzeczy najważniejszych… Kaśka dysponuje już stroną WWW, zatem przyjrzyjmy się jej konstrukcji. Później zabierzemy się za dodawanie do niej całego kodu JavaScriptu, o którym pisałem wcześniej.

Poniżej przedstawiłem aktualną postać kodu źródłowego strony prezentującej wyniki sprzedaży desek do snowbaordu Kaśki:

Kaśka przeczytała książkę Head First HTML with CSS & XHTML. Edycja polska, więc zna się na tych sprawach...

```
<html>
 <head>
  <title>Mega-Deski</title>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-2" />
  <link rel="stylesheet" type="text/css" href="deski.css" />
 </head>

 <body>
  <h1>Mega-Deski:: Raport sprzedaży</h1>
  <div id="boards">
   <table>
    <tr><th>Liczba sprzedanych desek</th>
     <td><span id="deski-sprzedane">1012</span></td></tr>
    <tr><th>Za ile ja sprzedaję</th>
     <td><span id="cena">249.95</span> PLN</td></tr>
    <tr><th>Ile mnie kosztują</th>
     <td><span id="koszt">84.22</span> PLN</td></tr>
   </table>
   <h2>Kasa na wyciągi:
    <span id="kasa">167718.76</span> PLN</h2>
   <form method="GET" action="pobierzAktualneDane.php">
    <input value="Pokaż mi kasę!" type="submit" />
   </form>
  </div>
 </body>
</html>
```

Kaśka używa zewnętrznego arkusza stylów.

Tutaj dodamy element `<script>`...

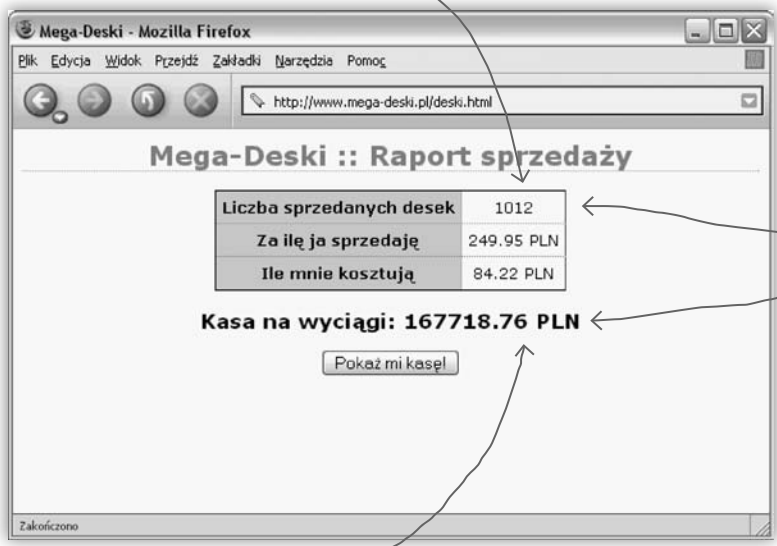
Tutaj jest wyświetlana całkowita liczba sprzedanych desek, którą będziesz musiał aktualizować...

...a później będziesz musiał także wyliczyć nowy zysk Kaśki.

A to jest przycisk, który Kaśka klika, żeby uaktualnić informacje wyświetlane na stronie raportu. W obecnej postaci kliknięcie tego przycisku powoduje wysłanie formularza, jednak już niedługo to zmienisz.

Kaśka podała w swoich elementach atrybuty id, dzięki czemu może określić ich postać w arkuszu stylów CSS...

1012



...a później będziemy mogli użyć tych identyfikatorów w naszych funkcjach JavaScriptu do aktualizacji liczb wyświetlanych w poszczególnych elementach .

167718.76

Po prostu to zrób

Czas przejść do działania. Pobierz przykłady do książki z serwera FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/hrajax.zip*) i odszukaj w nich katalog *rozdzial01/deski*. Następnie otwórz w przeglądarce plik *deski.html*. Strona wyświetlona w Twojej przeglądarce powinna wyglądać dokładnie tak jak ta przedstawiona na powyższym rysunku.

createRequest() getBoardsSold() updatePage()

Krok 1.: Utworzenie obiektu żądania

Wróćmy do modyfikacji raportu Kaśki. W pierwszej kolejności będziesz potrzebował funkcji tworzącej nowy obiekt, który pozwoli na przesyłanie żądań do serwera. Zadanie to jest dosyć trudne i skomplikowane, gdyż w różnych przeglądarkach taki obiekt jest tworzony w różny sposób. Aby ułatwić Ci to zadanie, napisałem „gotowy" skrypt wykonujący to zadanie. Za każdym razem, kiedy zobaczysz symbol Gotowy JavaScript, powinieneś założyć, że prezentowany kod robi właśnie to, o co chodzi, dokładnie tak, jak zamieszczony poniżej skrypt tworzący obiekt żądania. A zatem, na razie musisz mi zaufać — w kolejnych rozdziałach poznasz więcej szczegółów na temat poniższego fragmentu kodu. Na razie po prostu go przepisz do swojego pliku i zobacz, co dzięki niemu możesz zrobić.

Jeśli nie chce Ci się wpisywać tego kodu samodzielnie, to możesz go znaleźć w pliku tworzenie-zadania.txt w przykładach do książki, w katalogu rozdział01/deski...

```
var request = null;
function createRequest() {
 try {
   request = new XMLHttpRequest();
 } catch (trymicrosoft) {
   try {
     request = new ActiveXObject("Msxml2.XMLHTTP");
   } catch (othermicrosoft) {
     try {
       request = new ActiveXObject("Microsoft.XMLHTTP");
     } catch (failed) {
       request = null;
     }
   }
 }
 if (request == null)
   alert("Nie udało się utworzyć obiektu żądania!");
}
```

Oto zmienna, w której zostanie zapisany obiekt żądania.

W tym wierszu próbujemy utworzyć nowy obiekt żądania.

A to jest typ obiektu żądania.

Także w tych dwóch wierszach kodu próbujemy utworzyć obiekt żądania, jednak w sposób, który działa jedynie w przeglądarce Internet Explorer.

Jeśli próby utworzenia obiektu żądania nie powiodły się, to ten wiersz zapewni, że zmienna request będzie zawierała wartość null.

Teraz możesz sprawdzić, czy zmienna request wciąż ma wartość null. Jeśli faktycznie ma, to oznacza, że coś poszło nie tak...

...i należy wyświetlić użytkownikowi stosowny komunikat przy wykorzystaniu funkcji alert() języka JavaScript.

Gotowy JavaScript

...jednak poważnie rzecz biorąc, naprawdę powinieneś ten kod wpisać samemu. To pomoże Twojemu mózgowi przyzwyczaić się do pisania i myślenia o aplikacjach wykorzystujących technologię Ajax.

Do roboty! Dodaj przedstawiony kod do swojej strony deski.html, umieszczając go wewnątrz elementu <head>. Nie zapomnij o dodaniu znaczników <script> oraz </script>.

```
<head>
  <title>Mega-Deski</title>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-2" />
  <link rel="stylesheet" type="text/css" href="deski.css" />
  <script language="javascript" type="text/javascript">

  </script>
</head>
```

Te znaczniki informują przeglądarkę, iż na stronie jest umieszczony kod skryptu oraz że zostat on napisany w języku JavaScript.

Cały gotowy skrypt napisany w języku JavaScript ma zostać umieszczony w tym miejscu.

Najczęściej zadawane pytania

P. Czy powinienem już dokładnie rozumieć, jak działa przedstawiony kod?

O. Nie, nie musisz jeszcze rozumieć dokładnie tego kodu. Jak na razie wystarczy, żebyś mu się uważnie przyjrzał, opiszę go dokładniej w następnym rozdziale.

P. Co to jest `null`? Zobaczyłem to słowo w gotowym fragmencie kodu JavaScript, ale nie jestem pewny, co ono oznacza.

O. **`null`** to specjalna wartość, która faktycznie oznacza *wartość pustą* — nieistniejącą referencję. Innymi słowy, to jest coś, co można by określić jako „anty-wartość", jednak nie możesz mylić wartości **`null`** z wartością **0** lub **`false`** — gdyż one w żadnym przypadku nie są wartościami pustymi. Dlatego też nie mają one nic wspólnego z wartością **`null`**.

P. Czy obiekt żądania nosi nazwę „XMLHttpRequest" czy „ActiveXObject"?

O. Może to być każdy z tych obiektów. Znacznie dokładniej zajmiemy się tym zagadnieniem w następnym rozdziale, jednak najprostsze wyjaśnienie jest takie, iż w różnych przeglądarkach konieczne jest używanie różnych obiektów żądań.

P. A zatem, czy osoby, które używają moich aplikacji stosujących technologię Ajax, muszą wykorzystywać jakąś konkretną przeglądarkę?

O. Nie, o ile tylko mają w swojej przeglądarce włączoną obsługę języka JavaScript, to powyższy kod powinien być wykonywany bez najmniejszych problemów. A zatem, Twoje aplikacje korzystające z technologii Ajax będą równie dobrze działały w przeglądarce Firefox, Mozilla, Internet Explorer, Safari, Netscape oraz Opera.

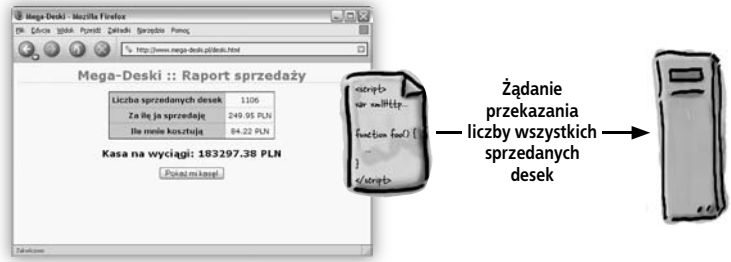
P. A co się stanie, jeśli użytkownik wyłączy w przeglądarce obsługę języka JavaScript?

O. Niestety, aplikacje korzystające z technologii Ajax wymagają zastosowania języka JavaScript. Oznacza to, że osoby, które wyłączą w swoich przeglądarkach obsługę JavaScriptu, nie będą mogły używać tych aplikacji.

Zazwyczaj we wszystkich przeglądarkach obsługa JavaScriptu jest domyślnie włączona, a zatem, jeśli ktoś ją wyłączył, to można przypuszczać, że wie, co robi i będzie w stanie ją ponownie włączyć, jeśli zechce używać aplikacji wykorzystującej technologię Ajax.

Krok 2.: Żądanie przesłania aktualnych wyników sprzedaży

Teraz, kiedy dysponujesz już funkcją **createRequest()**, możesz przejść do następnego etapu: napisania (oczywiście w języku JavaScript) funkcji **getBoardsSold()**. Funkcja ta użyje utworzonego wcześniej obiektu do przesłania do serwera żądania o zwrócenie sumarycznej liczby wszystkich sprzedanych desek. Określmy zatem, jak ta funkcja ma działać, a później zabierzemy się do napisania jej kodu. Czy pamiętasz nasz schemat? Oto krok, który obecnie realizujemy.



A oto, co będziesz musiał zrobić w celu zapewnienia poprawnego działania funkcji **getBoardsSold()**:

Do tego celu możesz wykorzystać gotowy kod JavaScript, przedstawiony na dwóch poprzednich stronach.

Kaśka podała już ten adres URL w swoim istniejącym formularzu, zatem wykonanie tego zadania będzie bardzo proste.

Aby wykonać tę operację, użyjesz obiektu żądania, zwróconego przez funkcję `createRequest()`.

Tej liczby użyjesz nieco później, w kroku 3., podczas wyświetlania aktualnych informacji na stronie raportu. Zajmiemy się tym już za chwilkę.

(a) Utworzyć nowy obiekt żądania poprzez wywołanie funkcji `createRequest()`.

(b) Określić adres URL, na jaki trzeba przesłać żądanie, aby otrzymać aktualne wyniki sprzedaży desek.

(c) Przygotować obiekt żądania do nawiązania połączenia z serwerem.

(d) Zgłosić żądanie przesłania aktualnych wyników sprzedaży desek.

Po prostu to zrób

Otwórz swój plik *deski.html* i dodaj do niego funkcję **getBoardsSold()**, umieszczając ją bezpośrednio poniżej funkcji **createRequest()**. Następnie spróbuj dodać do funkcji **getBoardsSold()** wiersz kodu, tworzący nowy obiekt żądania (patrz krok „a" na powyższej liście). Jeśli nie będziesz w stanie wykonać tego zadania, zajrzyj do odpowiedzi podanych na końcu tego rozdziału, na stronie *87*.

Zanim przejdziesz do lektury następnej strony, koniecznie musisz nie tylko wykonać to ćwiczenie, lecz także upewnić się, że wykonałeś je poprawnie. Nie zapomnij także o zapisaniu zmienionej wersji pliku *deski.html* na dysku.

Dodanie funkcji getBoardsSold()

Jeśli wykonałeś ćwiczenie zamieszczone na końcu poprzedniej strony, to w Twoim pliku *deski.html* powinien już znajdować się kod JavaScriptu podobny do przedstawionego poniżej:

Pamiętaj, że cały kod pisany w języku JavaScript musi być umieszczony pomiędzy znacznikami <script> oraz </script>.

```
<script language="JavaScript" type="text/javascript">
  var request = null;

  function createRequest() {
  // gotowy JavaScript
  }

  function getBoardsSold() {
    createRequest();
  }
</script>
```

To jest Twój obiekt żądania. Kiedy kod wywoła funkcję createRequest(), to w tej zmiennej zostanie zapisany nowy obiekt żądania.

A oto funkcja createRequest(), którą dodałeś do pliku podczas lektury jednej z poprzednich stron.

A to nowa funkcja...

...oraz miejsce, w jakim przy użyciu gotowego kodu JavaScriptu jest tworzony nowy obiekt żądania.

Przesyłanie żądania pod poprawny adres URL

No dobrze, a co dalej? Dysponujesz już obiektem, którego możesz użyć od przesłania do serwera żądania o zwrócenie sumarycznej liczby wszystkich sprzedanych desek, jednak w jaki sposób możesz zgłosić to żądanie? Przede wszystkim, musisz przekazać obiektowi informację o tym, na jaki adres należy przesłać żądanie — innymi słowy, obiekt ten wymaga nazwy programu przechowywanego na serwerze i używanego przez aplikację Kaśki do określania sumarycznej liczby sprzedanych desek. A zatem musisz przekazać adres URL skryptu działającego na serwerze.

Jednak czy ten skrypt PHP nie zwraca obecnie całego kodu HTML strony? Myślałem, że w nowej wersji aplikacji wykorzystującej technologię Ajax, będziemy chcieli, by serwer przesyłał w odpowiedzi jedynie sumaryczną liczbę wszystkich sprzedanych desek.

Ale skąd mamy wziąć ten adres URL? Możemy go znaleźć w oryginalnym formularzu Kaśki:

Oto adres URL skryptu PHP, do którego powinniśmy skierować żądanie.

```
<form method="GET" action="pobierzAktualneDane.php">
 <input value="Pokaż mi kasę!" type="submit" />
</form>
```

Oto element <form> umieszczony na stronie raportu Kaśki.

PHP ... na rzut oka

To dodatkowy bonus dla osób znających język PHP. Jeśli nie znasz tego języka, to i tak wszystko jest w porządku... po prostu przejrzyj poniższy skrypt i zabierz się za dalszą lekturę. Nie musisz rozumieć tego skryptu, aby uczyć się technologii Ajax lub analizować przykłady zamieszczane w tej książce.

Oto szybka prezentacja skryptu PHP używanego przez Kaśkę do generowania raportu ze sprzedaży desek. Pamiętaj, że nie mam zamiaru dokładnie wyjaśniać, jak on działa, jednak pokrótce opisałem, co się dzieje po odebraniu żądania o przesłanie aktualnych wyników sprzedaży.

Pierwsza część skryptu nawiązuje połączenie z bazą danych aplikacji Mega-Deski.

```
<?php
// Nawiązanie połączenia z bazą danych
$conn = @mysql_connect("mysql.megadeski.pl",
 "tajny", "super_tajne");
if(!$conn)
  die("Nie udało się nawiązać połączenia z serwerem MySQL: ". mysql_error());

if(!mysql_select_db("headfirst", $conn))
  die("Nie udało się wybrać bazy danych: ". mysql_error());

$select = 'SELECT boardsSold';
$from = ' FROM megadeski';
$queryResult = @mysql_query($select. $from);
if(!$queryResult)
  die('Nie udało się pobrać z bazy informacji o liczbie sprzedanych desek.');

while($row = mysql_fetch_array($queryResult)) {
  $totalSold = $row['boardsSold'];
}

$price = 249.95;
$cost = 84.22;
$cashPerBoard = $price - $cost;
$cash = $totalSold * $cashPerBoard;

mysql_close($conn);
?>

<html>
 <head>
  <title>Mega-Deski</title>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-2" />
  <link rel="stylesheet" type="text/css" href="deski.css" />
 </head>

 <body>
  <h1>Mega-Deski:: Raport sprzedaży</h1>
```

Ta część skryptu odpowiada za odczytanie z bazy aktualnych informacji o wielkości sprzedaży.

Gdybyś chciał, to także te wartości mógłbyś przechowywać i pobierać z bazy danych.

W tych wierszach wyliczamy zysk, jaki ma Kaśka na każdej sprzedanej desce.

A tu jest cały kod HTML, o którym już pisaliśmy...

```
  <div id="deski">
   <table>
    <tr><th>Liczba sprzedanych desek</th>
     <td><span id="deski-sprzedane">
<?php
  print $totalSold;
?>
    </span></td></tr>
    <tr><th>Za ile ja sprzedaję</th>
     <td><span id="cena">
<?php
  print $price;
?>
    PLN </span></td></tr>
    <tr><th>Ile mnie kosztują</th>
     <td><span id="koszt">
<?php
  print $cost;
?>
    PLN</span></td></tr>
   </table>
  <h2>Kasa na wyciągi:
    <span id="kasa">
<?php
  print $cash;
?>
    PLN </span></h2>
   <form method="GET" action="pobierzAktualneDane.php">
    <input value="Pokaż mi kasę!" type="submit" />
   </form>
  </div>
 </body>
</html>
```

Kolejny kod HTML generowany przez skrypt PHP...

Skrypt PHP wyświetla wartości pobrane z bazy danych oraz kod HTML, określający postać strony raportu.

A CO TAM?

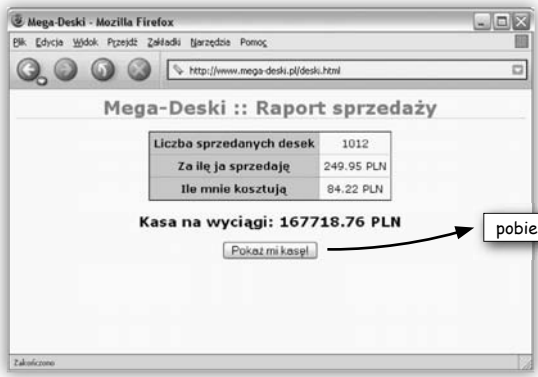
Wszystkie operacje wykonywane przez ten skrypt sprowadzają się do określenia liczby sprzedanych desek, wyliczenia zysku Kaśki i wyświetlenia strony HTML, prezentującej bieżące wyniki.

Nie przejmuj się jednak, jeśli nie znasz języka PHP lub jeśli jeszcze nigdy wcześniej nie obsługiwałeś baz danych. Już niedługo przedstawię inną wersję tego samego skryptu, która nie korzysta z bazy danych, dzięki temu samemu będziesz mógł wypróbować działanie raportu.

createRequest() getBoardsSold() updatePage()

Jak serwer pracował do tej pory...

Czy pamiętasz, w jaki sposób działała wersja raportu sprzedaży Kaśki, która nie korzystała z technologii Ajax? Za każdym razem, gdy otrzymywała zgłoszenie żądania skierowanego do skryptu PHP, musiała zwrócić zarówno bieżącą liczbę sprzedanych desek do snowbaordu, jak i cały kod HTML nowej strony. Zajrzyj na poprzednią stronę, a od razu zauważysz, że ponad połowa całego skryptu to kod HTML! Poniżej w nieco inny sposób pokazałem, jak działa ta wersja aplikacji:



Adres URL skryptu Kaśki, pobrany z jej formularza.

pobierzAktualneDane.php

Cały kod HTML, który serwer musi przesłać, odpowiadając na każde żądanie.

Serwer Kaśki, który musi obsługiwać ogromne ilości kodu HTML.

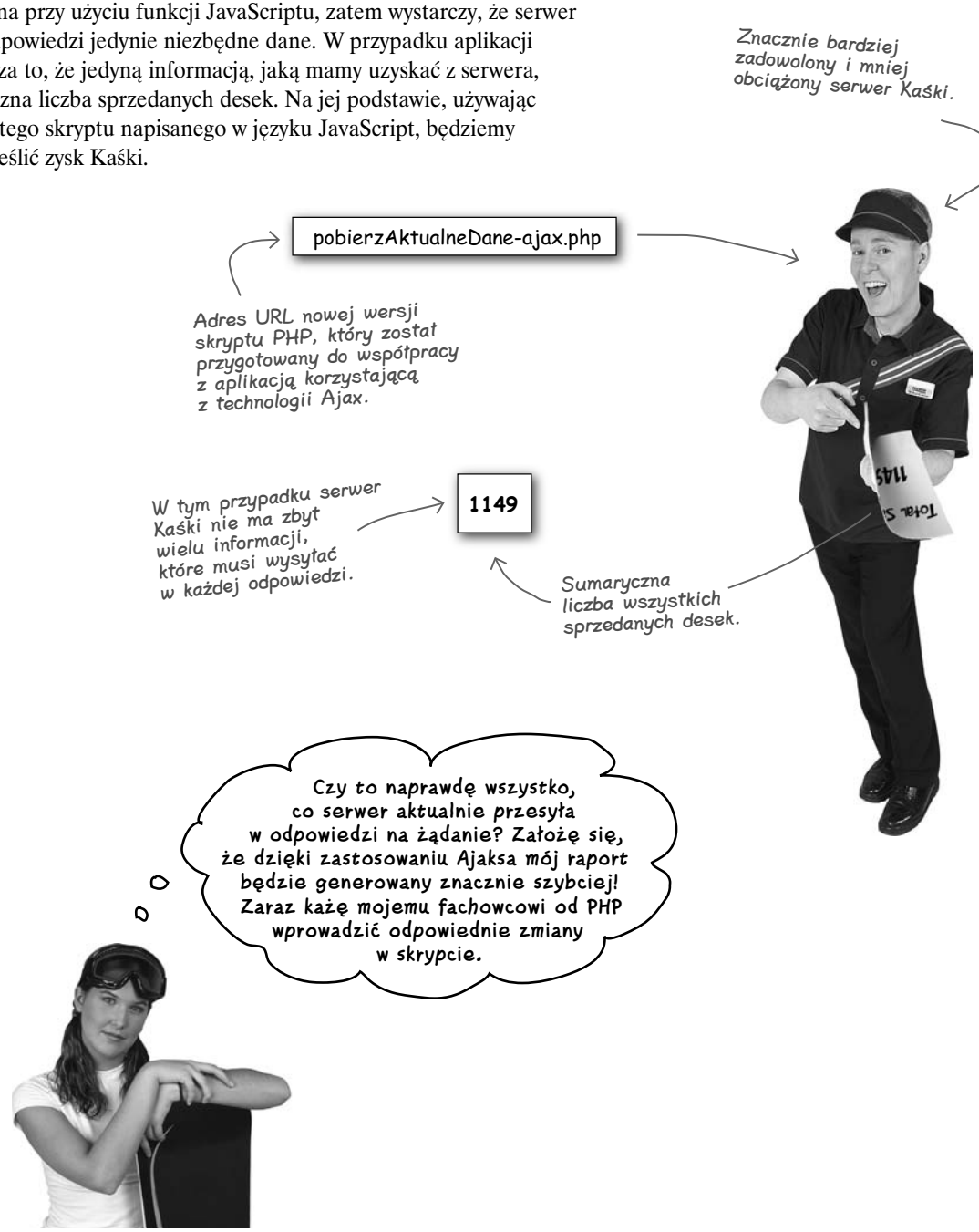
```
<html>
 <head>
  <title>Mega-Deski</title>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-2" />
  <link rel="stylesheet" type="text/css" href="deski.css" />
 </head>

 <body>
  <h1>Mega-Deski :: Raport sprzedaży</h1>
  <div id="boards">
   <table>
    <tr><th>Liczba sprzedanych desek</th>
     <td><span id="deski-sprzedane">1149</span></td></tr>
    <tr><th>Za ile ja sprzedaję</th>
     <td><span id="cena">249.95</span> PLN</td></tr>
    <tr><th>Ile mnie kosztują</th>
     <td><span id="koszt">84.22</span> PLN</td></tr>
   </table>
   <h2>Kasa na wyciągi:
    <span id="kasa">190423.27</span> PLN</h2>
   <form method="GET" action="pobierzAktualneDane.php">
    <input value="Pokaż mi kasę!" type="submit" />
   </form>
  </div>
 </body>
</html>
```

Taki rozbudowany kod HTML, a tak naprawdę zmieniają się tylko dwie wartości: sumaryczna liczba sprzedanych desek oraz zaktualizowany zysk Kaśki!

Co obecnie powinien robić serwer

W przypadku aplikacji korzystających z technologii Ajax, serwer nie musi przesyłać całego kodu HTML w każdej odpowiedzi. Zawartość strony jest aktualizowana przy użyciu funkcji JavaScriptu, zatem wystarczy, że serwer prześle w odpowiedzi jedynie niezbędne dane. W przypadku aplikacji Kaśki oznacza to, że jedyną informacją, jaką mamy uzyskać z serwera, jest sumaryczna liczba sprzedanych desek. Na jej podstawie, używając jedynie prostego skryptu napisanego w języku JavaScript, będziemy w stanie określić zysk Kaśki.



PHP... drugi rzut oka

Od momentu, gdy skrypt PHP wykorzystywany w aplikacji Kaśki mógł zwracać jedynie samą liczbę sprzedanych desek, stał się on znacznie krótszy. Zresztą przekonaj się sam:

Niemal cały początek nowej wersji skryptu jest taki sam jak w poprzedniej wersji.

```
<?php
// Nawiązanie połączenia z bazą danych
$conn = @mysql_connect("mysql.megadeski.pl",
                       "tajny", "super_tajne");
if (!$conn)
  die("Nie udało się nawiązać połączenia z serwerem MySQL: " . mysql_error());

if (!mysql_select_db("headfirst", $conn))
  die("Nie udało się wybrać bazy danych: " . mysql_error());

$select = 'SELECT boardsSold';
$from   = '  FROM megadeski ';
$queryResult = @mysql_query($select . $from);
if (!$queryResult)
  die('Nie udało się pobrać z bazy informacji o liczbie sprzedanych desek.');

while ($row = mysql_fetch_array($queryResult)) {
  $totalSold = $row['boardsSold'];
}

echo $totalSold;

mysql_close( $conn );
?>
```

Jednak teraz, zamiast całego kodu HTML, skrypt zwraca tylko jedną wartość: liczbę wszystkich sprzedanych desek.

✳ Zwróć uwagę, iż ze skryptu zniknął kod służący do wyliczania zysku Kaśki. Kalkulacje te będą teraz wykonywane przez kod umieszczony na stronie raportu i napisany w języku JavaScript.

createRequest() getBoardsSold() updatePage()

Po prostu to zrób

Nadszedł czas, by przystosować skrypt PHP, którego rola w aplikacji Kaśki polega na wykorzystywaniu technologii Ajax. Wyświetl przykłady dostępne w katalogu *rozdzial01/deski* i odszukaj wśród nich plik *pobierzAktualneDane-ajax.php*. Jest to wersja pliku *pobierzAktualneDane.php*, która zwraca jedynie liczbę sprzedanych desek, bez całego kodu HTML. Co więcej, ten skrypt nie wymaga korzystania z bazy danych, a zatem, aby go używać, nie trzeba instalować serwera MySQL. Upewnij się, że skrypt znajduje się w tym samym katalogu, co strona *deski.html*. Już za chwilkę zastosujemy ten nowy skrypt w aplikacji Kaśki.

Adres URL nowej wersji skryptu

Teraz, kiedy już dysponujesz skryptem PHP, który zwraca jedynie sumaryczną liczbę wszystkich sprzedanych desek bez całego kodu HTML strony, musisz w jakiś sposób przesłać do niego żądanie. Pamiętaj, że żądanie ma trafić do tego nowego skryptu PHP, oraz że nie zwraca on całego kodu HTML strony, a jedynie liczbę sprzedanych desek.

Adres URL nowego skryptu PHP zapisz w kodzie JavaScriptu, w nowej zmiennej:

```
function getBoardsSold() {
  createRequest();
  var url = „pobierzAktualneDane-ajax.php";
}
```

W pierwszej kolejności tworzy ona nowy obiekt żądania.

Oto funkcja napisana w języku JavaScript.

A to jest adres URL skryptu PHP przystosowanego do współpracy z aplikacją korzystającą z technologii Ajax.

Inicjalizacja połączenia

Dysponujesz już obiektem żądania oraz zmienną zawierającą adres URL, na jaki należy przesłać żądanie. Teraz musisz jedynie określić, w jaki sposób obiekt żądania ma nawiązać połączenie z serwerem i przekazać mu docelowy adres URL. Poniżej pokazałem, w jaki sposób to powinieneś zrobić:

```
function getBoardsSold() {

  createRequest();
  var url = "pobierzAktualneDane-ajax.php";
  request.open("GET", url, true);
}
```

Ten wiersz kodu inicjalizuje połączenie i przekazuje obiektowi żądania informacje o tym, w jaki sposób to połączenie należy utworzyć.

To właśnie w tym wierszu zaczynasz używać obiektu żądania utworzonego w funkcji `createRequest()`.

To dosyć zaawansowane pojęcia. Możesz sobie wpisać do dzienniczka piątkę z plusem, jeśli samemu wpadłeś na pomysł zadania któregokolwiek z tych pytań.

Rozbijmy to na czynniki pierwsze...

Metoda `open()` inicjalizuje połączenie...

```
request.open(
```

...a słowo `"GET"` określa, w jaki sposób należy przesłać dane do serwera.

```
„GET",
```

```
url,
```

To jest adres URL skryptu PHP działającego na serwerze Kaśki.

```
true);
```

Wartość `true` oznacza, że żądanie powinno być wykonane asynchronicznie... innymi słowy, wartość ta informuje skrypt, iż nie trzeba czekać na uzyskanie odpowiedzi z serwera. Strona z raportem sprzedaży Kaśki nie przestanie reagować podczas oczekiwania na odpowiedź z serwera.

Najczęściej zadawane pytania

P. Sądziłem, że żądania lepiej będzie wysyłać metodą „POST".

O. Metoda POST jest zazwyczaj używana w sytuacjach, gdy trzeba przesłać do serwera znaczne ilości danych, w przypadku przesyłania formularza związanego z operacjami finansowymi lub zakańczania procesu składania zamówienia. W naszym przypadku nie przesyłamy do skryptu PHP działającego na serwerze żadnych danych ani nie składamy zamówienia, zatem znacznie prościej będzie użyć metody GET. Wykorzystaniem metody POST zajmiemy się nieco później.

P. Czy w ogóle można używać metody `open()` do przesyłania żądań synchronicznych i przypisywać ostatniemu argumentowi wartość `false`?

O. Oczywiście! Czasami *nie będziemy chcieli*, by użytkownik mógł wykonywać jakiekolwiek czynności na stronie w czasie, gdy serwer odpowiada na przesłane żądanie. Na przykład można sądzić, że w przypadku składania zamówienia, nie będziesz chciał pozwolić użytkownikom na wykonywanie jakichkolwiek operacji aż do momentu potwierdzenia, iż zamówienie zostało zaakceptowane.

Wielki konkurs małżeństwa kodu i komentarza

Aby zaoszczędzić programistom czasu i problemów, na całym świecie fragmenty kodu są kojarzone z odpowiednimi komentarzami, zapewniając w ten sposób miłość, partnerstwo i dokumentację dla wszystkich.

Poniżej przedstawiłem kilka wierszy kodu JavaScript, zaczerpniętych z aplikacji korzystającej z technologii Ajax. Twoim zadaniem jest znalezienie pary — czyli odpowiedniego komentarza — dla każdego z tych wierszy.

Kod

```
request.open("POST", url, true);
```

```
var request = new XMLHttpRequest();
```

```
var url = "/servlet/GetMileageServlet";
```

```
request.open("GET", url, false);
```

Komentarze

```
/* Tworzy nowy obiekt komunikujący się protokołem HTTP z serwerem WWW. */
```

```
/* Tworzy nową zmienną i zapisuje w niej adres URL serwletu utworzonego w Javie. */
```

```
/* Inicjalizuje synchroniczne połączenie, wykorzystujące metodę GET */
```

```
/* Inicjalizuje asynchroniczne połączenie, wykorzystujące metodę POST */
```

Czy pamiętasz naszą listę zadań, dotyczącą pisania funkcji getBoardsSold()?

Prace nad pisaniem kodu postępują w doskonałym tempie. Może zatem spróbujesz zaznaczyć, jakie czynności już wykonałeś?

- ☐ Utworzyć nowy obiekt żądania poprzez wywołanie funkcji createRequest().
- ☐ Określić adres URL, na jaki trzeba przesłać żądanie, aby otrzymać aktualne wyniki sprzedaży desek.
- ☐ Przygotować obiekt żądania do nawiązania połączenia z serwerem.
- ☐ Zgłosić żądanie przesłania aktualnych wyników sprzedaży desek.

Wielki konkurs małżeństwa kodu i komentarza

Czy dobrałeś szczęśliwe pary? Upewnij się, że dobrałeś je dobrze, a jeśli popełniłeś jakieś błędy, to poświęć nieco czasu, by określić, co i dlaczego przegapiłeś.

Prezentujemy szczęśliwe pary...

Pamiętaj, że ten kod nie będzie działać w Internet Explorerze!

```
/* Tworzy nowy obiekt komunikujący się protokołem HTTP z serwerem WWW. */
var request = new XMLHttpRequest();

/* Tworzy nową zmienną i zapisuje w niej adres URL serwletu utworzonego w Javie. */
var url = "/servlet/GetMileageServlet";
```

To zwyczajna zmienna JavaScriptu.

```
/* Inicjalizuje synchroniczne połączenie, wykorzystujące metodę GET */
request.open("GET", url, false);
```

Żądania można przesyłać metodą GET lub POST.

Kiedy ostatni argument przekazywany w wywołaniu tej metody ma wartość `false`, to żądanie zostanie wykonane synchronicznie...

```
/* Inicjalizuje asynchroniczne połączenie, wykorzystujące metodę POST */
request.open("POST", url, true);
```

...jeśli natomiast argument przyjmie wartość `true`, to zostanie ono wykonane asynchronicznie.

Oto nasza lista zadań ze strony 54... pozostał nam do wykonania tylko jeden punkt.

- ☑ Utworzyć nowy obiekt żądania poprzez wywołanie funkcji createRequest().
- ☑ Określić adres URL, na jaki trzeba przesłać żądanie, aby otrzymać aktualne wyniki sprzedaży desek.
- ☑ Przygotować obiekt żądania do nawiązania połączenia z serwerem.
- ☐ Zgłosić żądanie przesłania aktualnych wyników sprzedaży desek.

Nawiązanie połączenia z serwerem WWW

No dobra! Dysponujesz już obiektem żądania, znasz docelowy adres URL, a połączenie jest skonfigurowane i gotowe do użycia. Teraz wystarczy nawiązać połączenie z serwerem i zażądać od niego zwrócenia aktualnej liczby sprzedanych desek. W tym celu wystarczy wywołać metodę **send()** obiektu **request**:

Właśnie teraz zabierzemy się za wykonanie tej ostatniej czynności.

```
function getBoardsSold() {
  createRequest();
  var url = "pobierzAktualneDane-ajax.php";
  request.open("GET", url, true);
  request.send(null);
}
```

Ta metoda powoduje wysłanie żądania...

...a null oznacza, że w żądaniu nie są przesyłane żadne dane. Nasz skrypt nie potrzebuje żadnych danych — każde jego wywołanie ma jedynie zwracać sumaryczną liczbę wszystkich sprzedanych desek.

Przekręć nieco swój mózg (i stronę też) i nie zapominaj o tym bardzo ważnym zagadnieniu.

Aplikacje asynchroniczne wykonują żądania przy użyciu obiektu JavaScriptu, a nie poprzez przesyłanie formularzy.

Żartujesz sobie ze mnie? Tak się namęczyliśmy, a ty mi teraz mówisz, że nic nie przesyłamy do serwera? I ty to nazywasz tworzeniem aplikacji nowej generacji?

Serwer nie potrzebuje żadnych danych.

W naszej aplikacji przesyłanie jakichkolwiek danych do serwera nie jest potrzebne. Każde żądanie kierowane do skryptu PHP ma na celu to samo: otrzymanie jednej liczby, określającej sumaryczną liczbę sprzedanych desek snowboardowych. Właśnie dlatego nie musimy przekazywać w żądaniu niczego więcej niż **null**.

Pamiętaj, że null oznacza „wartość pustą", czyli nic — a to jest dokładnie to, co chcemy przesłać do serwera.

Podsumowanie wszystkiego, co zdążyliśmy już zrobić

Wciąż zostało nam jeszcze wiele do zrobienia, jednak poświęćmy chwilkę na przyjrzenie się temu, co już udało się nam osiągnąć. Zrób sobie filiżankę dobrej kawy, pozwól swojemu mózgowi nieco odpocząć i przejrzyj wszystkie czynności, jakie wykonałeś w ramach realizacji dwóch pierwszych etapów zmiany zwykłego raportu Kaśki w aplikację korzystającą z technologii Ajax.

To było oryginalne zadanie, jakie mieliśmy wykonać...

...a to opis tego, co faktycznie wykonaliśmy.

❶ Utworzysz nowy obiekt, który będzie przesyłał żądania do serwera.

Dodałeś nową funkcję o nazwie **`createRequest()`**, która tworzy nowy obiekt żądania. Co więcej, ta funkcja może poprawnie działać w wielu różnych przeglądarkach.

Oto fragment kodu HTML bieżącej wersji aplikacji Mega-Deski.

```
<html>
 <head>
  <title>Mega-Deski</title>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-2" />
  <link rel="stylesheet" type="text/css" href="deski.css" />

 </head>

 <body>
  <h1>Mega-Deski:: Raport sprzedaży</h1>
```

Ten kod napisany w języku JavaScript dodasz do sekcji nagłówka swojej strony.

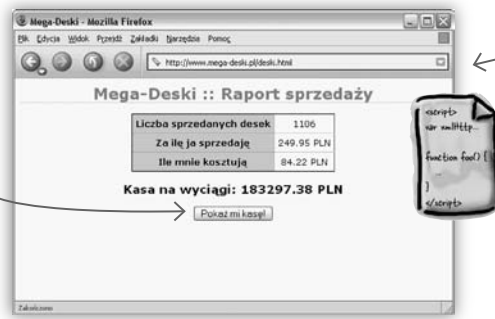
Funkcja `createRequest()` utworzy obiekt żądania.

❷ Napiszesz funkcję, która będzie żądała od serwera przesłania aktualnych informacji o sprzedaży.

Napisałeś także kolejną nową funkcję o nazwie **`getBoardsSold()`**. Wewnątrz niej, w celu utworzenia nowego obiektu żądania, jest wywoływana funkcja **`createRequest()`**. Następnie utworzyłeś zmienną, w której zapisałeś adres URL skryptu PHP działającego na serwerze Kaśki i przystosowanego do współpracy z nową wersją aplikacji, wykorzystującą technologię Ajax. Potem zainicjalizowałeś połączenie i przesłałeś żądanie do serwera.

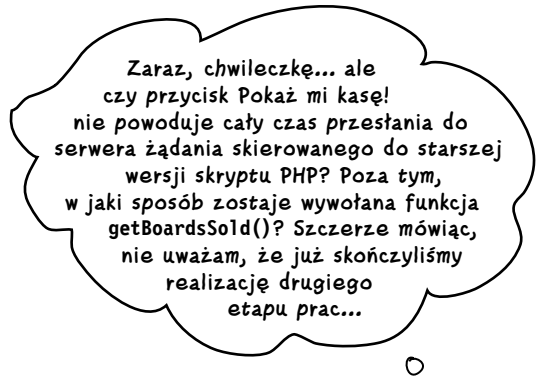
Nie musisz już przesyłać formularza... teraz funkcja `getBoardsSold()` zadba o przesłanie odpowiedniego żądania do serwera.

Kliknięcie przycisku Pokaż mi kasę! powoduje wygenerowanie żądania.



Żądanie przekazania liczby wszystkich sprzedanych desek

Na serwerze Kaśki umieściliśmy już nową wersję skryptu PHP, który jest już przygotowany do współpracy z aplikacją korzystającą z technologii Ajax.



Wciąż nam brakuje kilku rzeczy

Musimy mieć pewność, że formularz Kaśki nie będzie już wysyłany do starszego skryptu PHP, który nie jest przystosowany do współpracy z naszą nową wersją aplikacji, korzystającą z technologii Ajax. W przeciwnym razie nie uda nam się pozbyć tego strasznego odświeżania strony raportu, a nasza cała ciężka praca pozostanie niezauważona!

Jednak to nie wszystko — oprócz tego musimy się także upewnić, że kliknięcie przycisku *Pokaż mi kasę!* spowoduje wywołanie funkcji **`getBoardsSold()`**. Wygląda na to, że absolutnie nie jesteśmy jeszcze gotowi do rozpoczęcia aktualizowania zawartości strony… więc co trzeba teraz zrobić?

Otóż istnieją dwa podstawowe zagadnienia, którymi musimy się zająć. Poniżej zostawiłem nieco pustego miejsca, w którym masz zapisać co, według Ciebie, musimy zrobić w celu zakończenia realizacji drugiego etapu prac nad nową wersją aplikacji Kaśki.

STÓJ! Nie zaczynaj lektury następnej strony, zanim nie napiszesz czegoś w tych pustych wierszach

createRequest() getBoardsSold() updatePage()

Wracamy do kodu HTML

Czy już wiesz, co jeszcze musisz zrobić? Kiedy Kaśka klika przycisk *Pokaż mi kasę!*, cały formularz jest przesyłany do skryptu PHP. Jednak my nie chcemy przesyłać formularza, gdyż do przesłania żądania do serwera użyjemy możliwości Ajaksa. Na szczęście, wprowadzenie niezbędnych zmian nie powinno być szczególnie trudne.

A zatem, wróćmy do kodu HTML strony raportu sprzedaży aplikacji Mega-Deski:

```
<body>
<h1>Mega-Deski:: Raport sprzedaży</h1>
<div id="boards">
 <table>
  <tr><th>Liczba sprzedanych desek</th>
   <td><span id="deski-sprzedane">1012</span></td></tr>
  <tr><th>Za ile ja sprzedaję</th>
   <td><span id="cena">249.95</span> PLN</td></tr>
  <tr><th>Ile mnie kosztują</th>
   <td><span id="koszt">84.22</span> PLN</td></tr>
 </table>
 <h2>Kasa na wyciągi:
  <span id="kasa">167718.76</span> PLN</h2>
 <form method="GET" action="pobierzAktualneDane.php">
  <input value="Pokaż mi kasę!" type="submit" />
 </form>
</div>
</body>
```

Przedstawiłem tu jedynie kod HTML umieszczony wewnątrz elementu `<body>`, reszta pliku została pominięta.

Kliknięcie tego przycisku nie powinno już powodować przesłania formularza do serwera...

```
<input value="Pokaż mi kasę!" type="button" />
```

...a zatem można go zmienić na zwyczajny przycisk.

Najczęściej zadawane pytania

P: A czy zamiast zmieniania typu przycisku nie wystarczyłoby usunąć ze znacznika `<form>` atrybutu `action`?

O: To dobry pomysł — można to zrobić *oprócz* zmiany typu przycisku. Jeśli ograniczymy się *jedynie* do usunięcia atrybutu **`action`**, to kliknięcie przycisku *Pokaż mi kasę!* wciąż będzie powodowało przesłanie formularza do serwera... jednak w tym przypadku przesłanie formularza spowoduje jedynie odświeżenie strony. Dlatego też konieczna jest zmiana typu przycisku z **`"submit"`** na **`"button"`**.

Wywoływanie funkcji getBoardsSold() z poziomu formularza HTML

Nasz skrypt napisany w JavaScripcie jest gotowy, przycisk już nie powoduje przesyłania formularza do serwera… jednak także teraz funkcja **getBoardsSold()** nigdy nie jest wywoływana. Na szczęście rozwiązanie tego problemu nie powinno być szczególnie trudne:

Pytanie do JavaScriptu

JavaScripcie, musimy wywołać funkcję z poziomu formularza HTML. Czy możesz nam pomóc?

Och, no wiecie… jeśli chodzi o wywoływanie funkcji z poziomu kodu HTML, to moje możliwości są niezwykle duże i elastyczne. Wystarczy, że w kodzie HTML użyjecie jednej z **procedur obsługi zdarzeń**. Przykładowo, możecie użyć procedury **onBlur()**, aby wykonać jakąś operację, gdy użytkownik będzie wychodził z pola, **onClick()** — aby zrobić coś po kliknięciu przycisku… a co powiecie na **onChange()**, wywoływaną, kiedy zmieni się określona wartość… jest także **onFocus()**,… czekajcie, czekajcie, jest ich jeszcze więcej… czy to znaczy, że skończył się nam czas?

createRequest() getBoardsSold() updatePage()

Dodawanie procedury obsługi zdarzenia

Za każdym razem, gdy Kaśka kliknie przycisk *Pokaż mi kasę!*, powinna zostać wywołana funkcja **getBoardsSold()**. Do tego celu można zastosować **procedurę obsługi zdarzenia**. Taka procedura obsługi zdarzenia kojarzy fragment kodu JavaScriptu — na przykład funkcję **getBoardsSold()** — z pewnym zdarzeniem, takim jak kliknięcie przycisku umieszczonego na stronie.

W przypadku naszej aplikacji użyjemy procedury obsługi zdarzeń do połączenia przycisku *Pokaż mi kasę!* z funkcją **getBoardsSold()**. Należy dołączyć zdarzenie do kliknięcia przycisku, zatem musimy użyć procedury **onClick()**. Poniżej pokazałem, w jaki sposób to można zrobić:

```
<form method="GET" action="pobierzAktualneDane.php">
 <input value="Pokaż mi kasę!" type="button"
        onClick="getBoardsSold();"/>
</form>
```

W przypadku raportu wykorzystującego technologię Ajax, formularz nigdy nie powinien być przesyłany do serwera, zatem atrybut `action` możesz usunąć.

„onClick" oznacza, że za każdym razem, gdy Kaśka kliknie ten przycisk...

...zostanie wywołana ta funkcja.

Przegląd filmu utrwalającego

Czy pamiętasz nasz film utrwalający? Czy na bieżąco śledziłeś to, co udało się nam już zrobić oraz to, co jeszcze pozostało do zrobienia? Przyjrzyjmy się pokrótce temu, czego już nauczyłeś się podczas lektury tego rozdziału:

To właśnie z tego powodu zmodyfikowaliśmy przycisk Pokaż mi kasę! w taki sposób, by wywoływał funkcję JavaScriptu, a nie przesyłał formularz do serwera.

Do utworzenia tego obiektu żądania użyliśmy funkcji `createRequest()`.

Aplikacje asynchroniczne wykonują żądania przy wykorzystaniu obiektu JavaScriptu, a nie poprzez przesyłanie formularza.

Żądania i odpowiedzi są obsługiwane przez przeglądarkę, a nie bezpośrednio przez kod JavaScriptu.

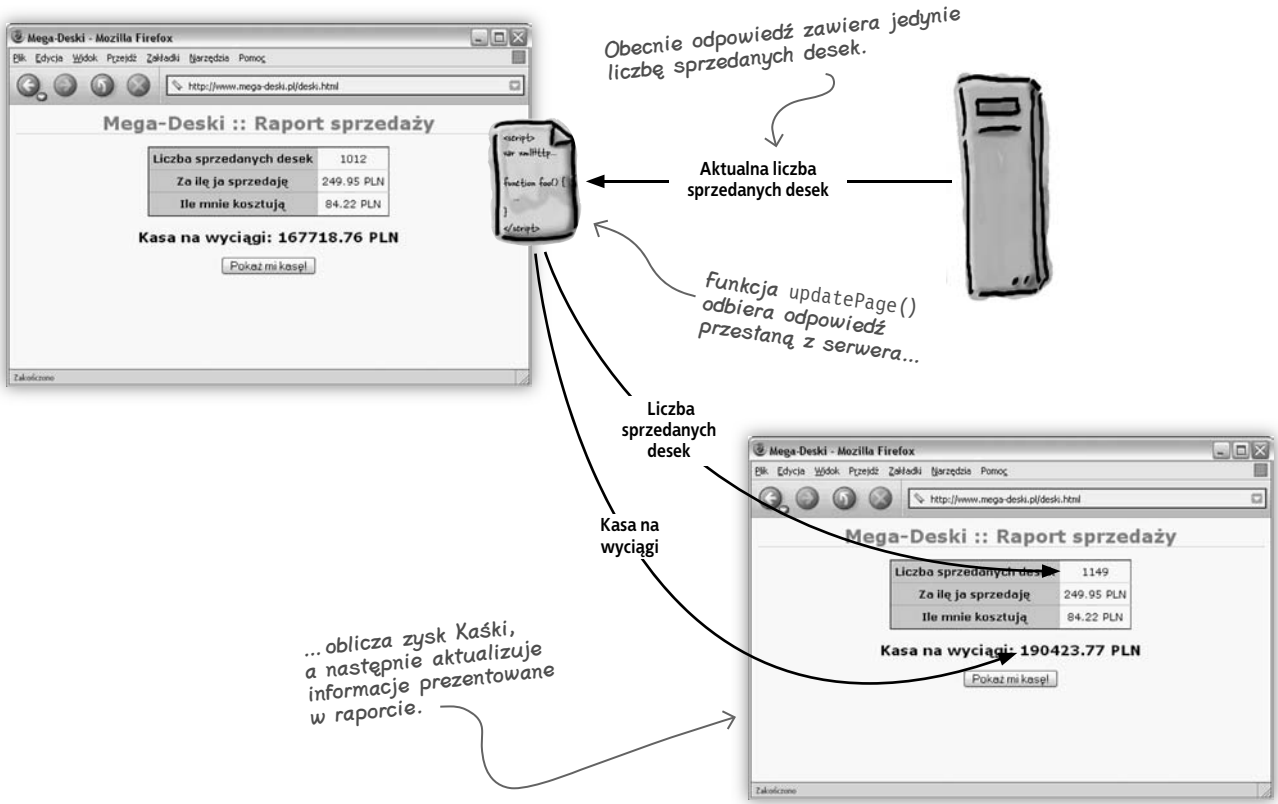
Do tych dwóch zagadnień jeszcze nie dotarliśmy.

Kiedy przeglądarka odbierze odpowiedź na Twoje asynchroniczne żądanie, wywoła wskazaną przez Ciebie funkcję JavaScriptu, przekazując do niej odpowiedź otrzymaną z serwera.

createRequest() getBoardsSold() updatePage()

Krok 3.: Pisanie kodu funkcji updatePage()

A zatem zadbałeś już o obsługę przesłania żądania do serwera i jesteś gotów, aby odebrać zwróconą przez niego odpowiedź i zaktualizować informacje wyświetlone w raporcie sprzedaży. A zatem nadszedł czas, aby napisać funkcję **`updatePage()`**. Czy pamiętasz, jakie zadania ma pełnić ta funkcja?



MOC UMYSŁU

W jaki sposób funkcja `updatePage()` może pobrać odpowiedź przesłaną przez skrypt, działający na serwerze Kaśki? Pamiętasz zapewne, że żądanie zostało wysłane w funkcji `getBoardsSold()`... oraz że nasza aplikacja ma działać *asynchronicznie*. To oznacza, że Kaśka powinna mieć możliwość wykonywania innych operacji na stronie podczas oczekiwania na przesłanie odpowiedzi z serwera.

A zatem, w jaki sposób funkcja `updatePage()` może uzyskać odpowiedź nadesłaną przez serwer? Zastanów się nad tym przez minutę lub dwie, a następnie zabierz się za lekturę kolejnej strony.

Uważam, że funkcję updatePage() powinniśmy wywołać na samym końcu funkcji getBoardsSold(). W ten sposób funkcja updatePage() będzie już mogła pobrać odpowiedź przesłaną z serwera, nieprawdaż?

Ajax jest technologią asynchroniczną

Pamiętaj, że funkcja **getBoardsSold()** wykonuje żądanie w sposób *asynchroniczny*. A to oznacza, że skrypt nie będzie czekał na odebranie odpowiedzi przesyłanej przez serwer.

W praktyce może się okazać, że działanie funkcji **getBoardsSold()** skończy się szybciej niż skrypt *pobierzAktualneDane-ajax.php* w ogóle zdąży zakończyć przetwarzanie żądania.

```
fuction
getBoardsSold() {
 createRequest();
 var url = "pobierzAktualneDane-ajax.php";
 request.open("GET", url, true);
 request.send(null);
 updatePage();
}
```

To nie będzie działać.... gdyż metoda send() nie będzie oczekiwała na odpowiedź z serwera, a zatem, w takim przypadku funkcja updatePage() zostałaby wywołana, zanim serwer zakończyłby przetwarzanie żądania i odesłał odpowiedź.

A zatem, jeśli działanie funkcji getBoardsSold() kończy się przed przesłaniem odpowiedzi przez serwer, to gdzie ta odpowiedź trafia? I w jaki sposób możemy zapewnić wywołanie metody updatePage() po odebraniu odpowiedzi nadesłanej przez serwer?

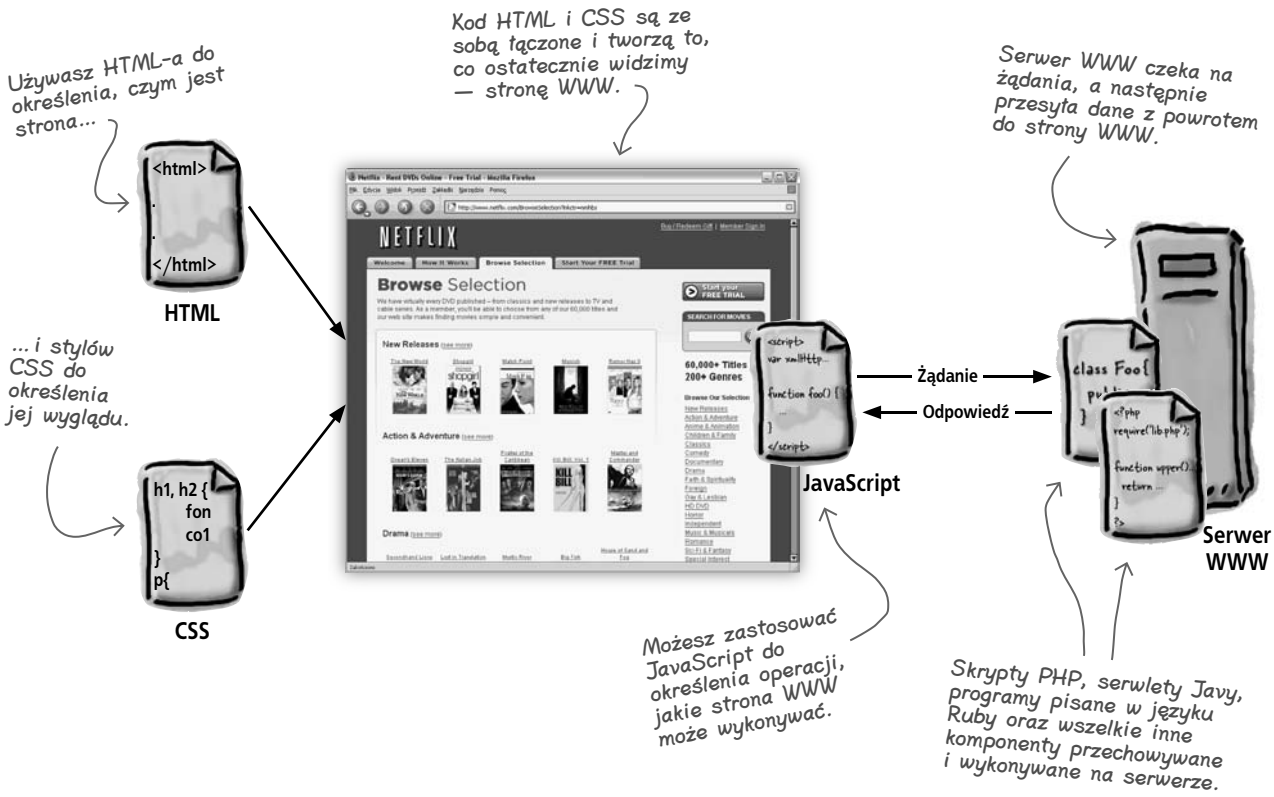
Gdzie trafia odpowiedź?

Właśnie dotarłeś do najtrudniejszego zagadnienia związanego z programowaniem asynchronicznych aplikacji internetowych: skoro asynchroniczne wywołanie nie czeka na odebranie odpowiedzi przysyłanej przez serwer, to w jaki sposób możemy ją *pobrać i zastosować* przesłane w niej informacje? A przede wszystkim, gdzie ta odpowiedź trafia?

Już wiesz, że na stronach WWW można wywoływać funkcje pisane w języku JavaScript przy wykorzystaniu procedur obsługi zdarzeń. Widziałeś także, w jaki sposób kod JavaScriptu wykonuje asynchroniczne żądanie. Aby dokończyć ten obrazek asynchronicznej aplikacji internetowej, musimy jednak dodać jeszcze jeden kawałek do układanki…

W jaki sposób postrzegamy aplikacje internetowe...

Aby określić, co się dzieje w naszej aplikacji, musimy się nieco cofnąć i wyobrazić sobie działanie aplikacji internetowych w sposób bardziej ogólny. Myśląc o aplikacji internetowej, zapewne wyobrażasz sobie stronę WWW, jakiś kod JavaScriptu, jakieś arkusze stylów CSS, może jeszcze jakiś serwer, na którym są wykonywane skrypty lub serwlety. Wszystkie te elementy współpracują ze sobą w sposób przedstawiony na poniższym rysunku:



Jednak w rzeczywistości dzieje się więcej niż widać na tym schemacie...

Kim jestem

Nadszedł czas, by przetestować umiejętności kojarzenia wysokiego poziomu. W naszej sztuce pod tytułem „Ajax" jest jedna bardzo ważna postać, której jeszcze nie spotkałeś i która właśnie przygotowuje się do swojego wielkiego wejścia na scenę. Sprawdź, czy będziesz w stanie zaskoczyć tę postać, odgadując jej tożsamość, jeszcze *zanim* przejdziesz na następną stronę. Udało nam się przedstawić kilka wypowiedzi tej postaci, które być może ułatwią Ci zadanie.

➡ **„Śpiew? Cóż, bardzo wiele osób nie zdaje sobie z tego sprawy, jednak przeszłam klasyczny trening. I kocham Wagnera."**

➡ **„Podaj mi swoje zmęczone, biedne, tulące się do siebie nawiasy, które nie mogą się doczekać własnego stylu."**

No cóż, nasza tajemnicza postać czasami jest cokolwiek przesadnie dramatyczna.

➡ **„Oczywiście, że uwielbiam sporty wodne. W rzeczywistości byłam zmiennikiem w filmie *Na fali* i prawie udało mi się dostać główną rolę w filmie *Blue Crush*.**

➡ **„Jestem tak twarda jak wymaga tego sytuacja... kilka lat temu brałam nawet udział w konfliktach zbrojnych".**

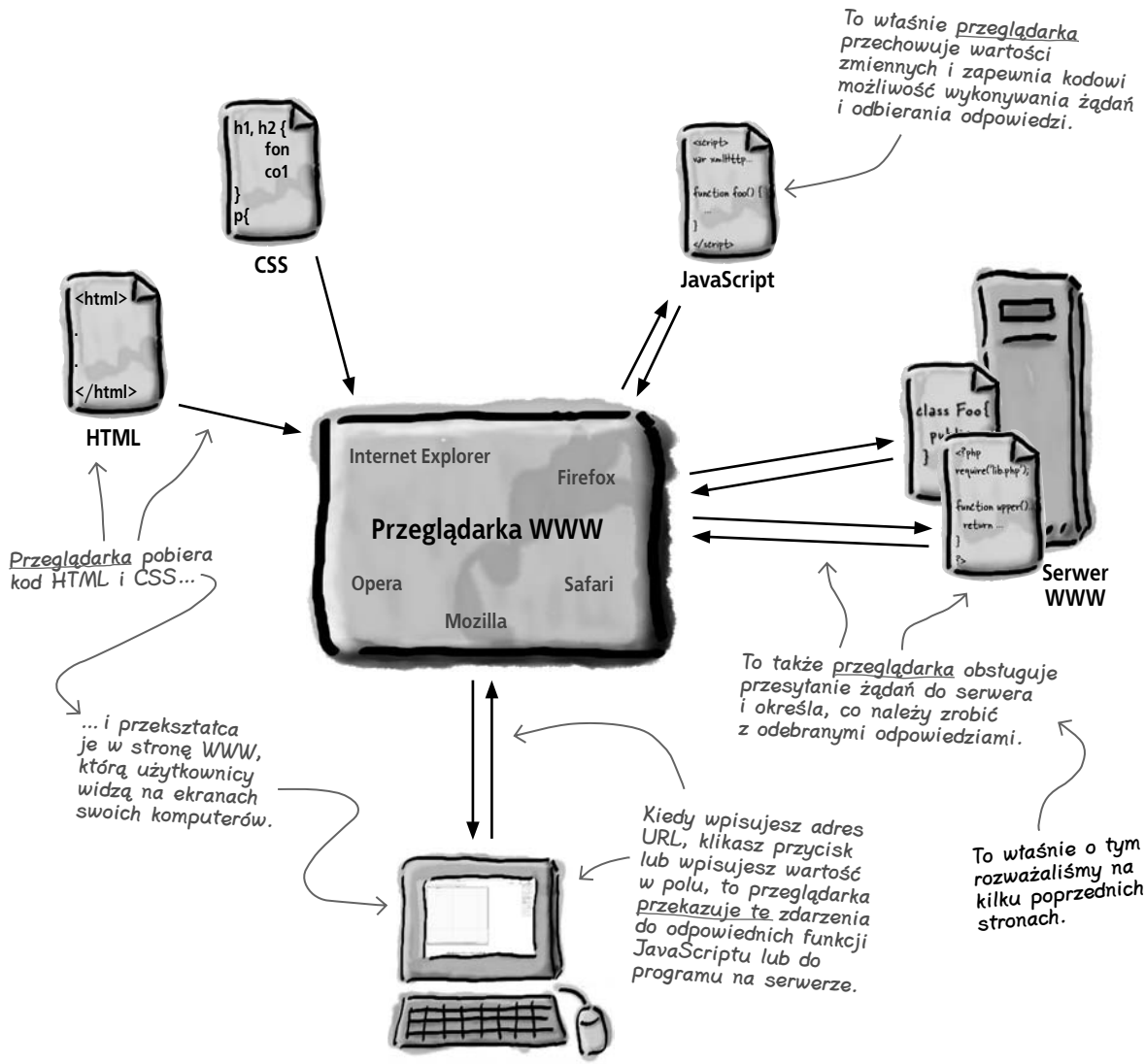
Kim jestem? _____

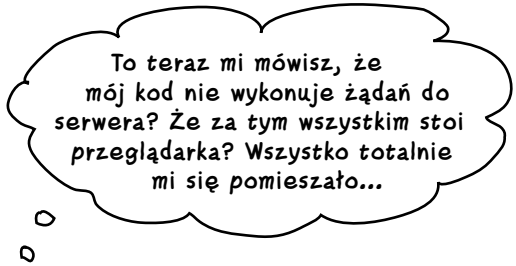
Tutaj wpisz, kim według Ciebie jest nasza tajemnicza postać, następnie przewróć stronę i sprawdź, czy Twoje przypuszczenia okazały się słuszne.

Przedstawiamy przeglądarkę WWW

Za kulisami musi być coś, co łączy te wszystkie elementy układanki w jedną całość. Tym „czymś" jest przeglądarka. To właśnie ona odczytuje kod HTML, style CSS i przekształca te wszystkie nawiasy kątowe w stronę pełną grafiki, przycisków i tekstów.

To także przeglądarka wykonuje cały kod JavaScriptu umieszczony na stronie… cicho i bez rozgłosu, ukryta za kulisami przeglądarka dba o realizację ważnych zadań, takich jak zapisywanie wartości w zmiennych, tworzenie nowych typów danych i obsługę wszelkich żądań sieciowych, jakie mogą się pojawić w Twoim kodzie.





Przeglądarka jedynie pomaga

Przeglądarka nie wykonuje żadnych tajemniczych zadań. Kiedy w kodzie skryptu musisz umieścić żądanie, piszesz coś takiego:

```
function getBoardsSold() {
  createRequest();
  var url = "pobierzAktualneDane-ajax.php";
  request.open("GET", url, true);
  request.send(null);
}
```

Przeglądarka obsługuje jedynie operacje sieciowe niskiego poziomu, konieczne do wykonania tego kodu. Połączenia sieciowe są obsługiwane w odmienny sposób w różnych systemach operacyjnych (wyobraź sobie Linuksa, Windows oraz Mac OS X), zatem to przeglądarka obsługuje operacje typowe dla każdego z nich. Dzięki temu skrypt będzie działać na każdym systemie, a przeglądarka zapewni przekształcenie kodu JavaScript do postaci, którą będzie w stanie wykorzystać konkretny komputer użytkownika.

Spójrz jeszcze raz na rysunek przedstawiony *na stronie 66* i zwróć uwagę, iż właśnie *przeglądarka* obsługuje wysyłanie żądań i odbieranie odpowiedzi przesyłanych z serwera. Kod „mówi” przeglądarce **co należy zrobić**, a ona robi wszystko, co ma zostać **wykonane**.

Twoje żądania i odpowiedzi są obsługiwane przez przeglądarkę WWW, a nie bezpośrednio przez kod JavaScriptu.

MOC UMYSŁU

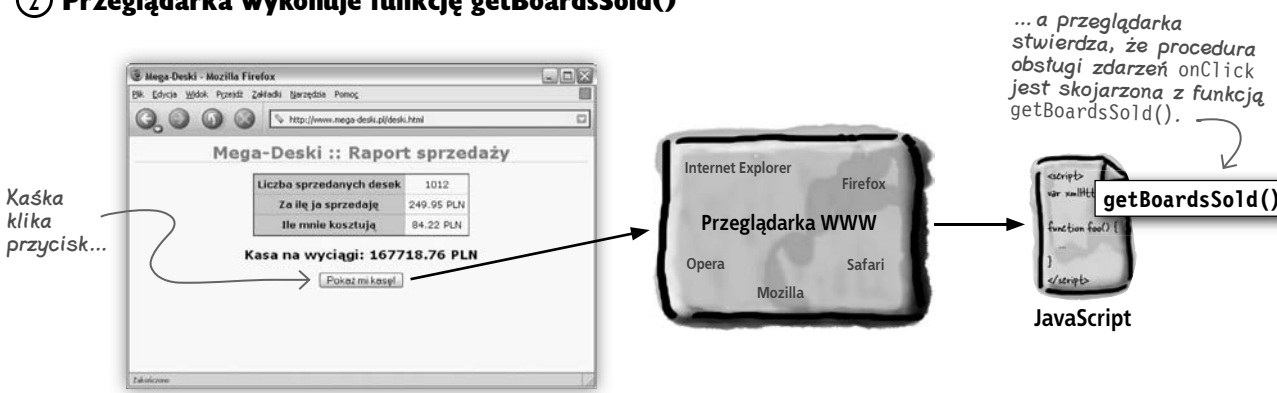
Jak myślisz, dlaczego w aplikacjach asynchronicznych konieczność obsługi żądań przez przeglądarkę WWW ma tak duże znacznie? Pamiętaj, że serwer WWW może odpowiedzieć tylko temu, kto przesłał do niego żądanie.

Przeglądarka przekazuje odpowiedź serwera do Twojego skryptu

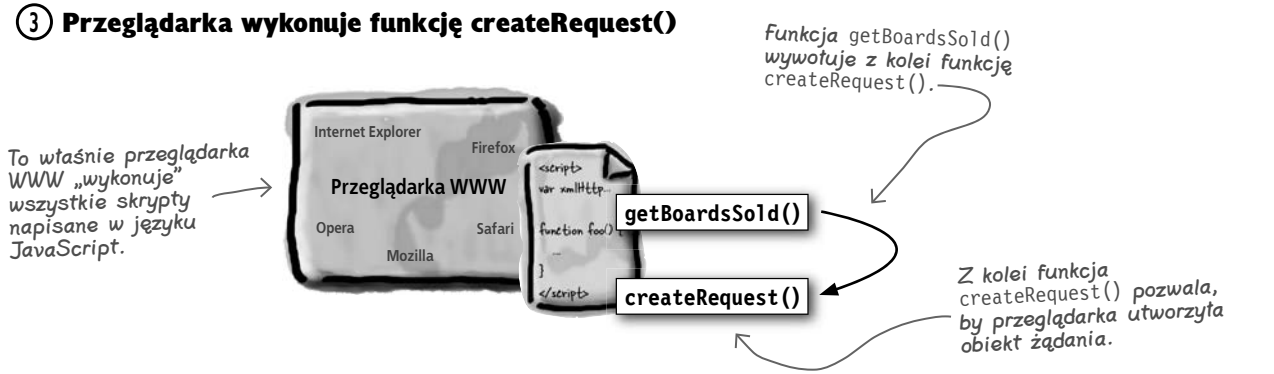
Przeglądarka obsługuje przesyłanie żądań, zatem także ona jest odpowiedzialna za odbieranie odpowiedzi przesyłanych przez serwer. Zobaczmy, co przeglądarka robi w naszej przykładowej aplikacji Mega-Deski:

① Przeglądarka wyświetla aplikację Mega-Deski

② Przeglądarka wykonuje funkcję getBoardsSold()



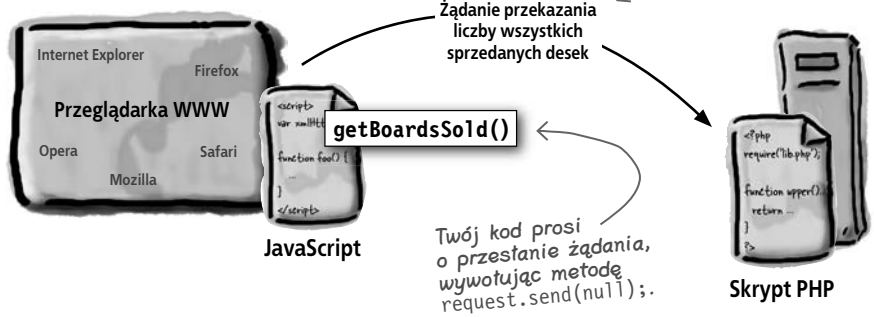
③ Przeglądarka wykonuje funkcję createRequest()



④ Przeglądarka przesyła żądanie do serwera Kaśki

Żądanie jest wysyłane przez przeglądarkę, a nie bezpośrednio przez kod napisany w JavaScripcie.

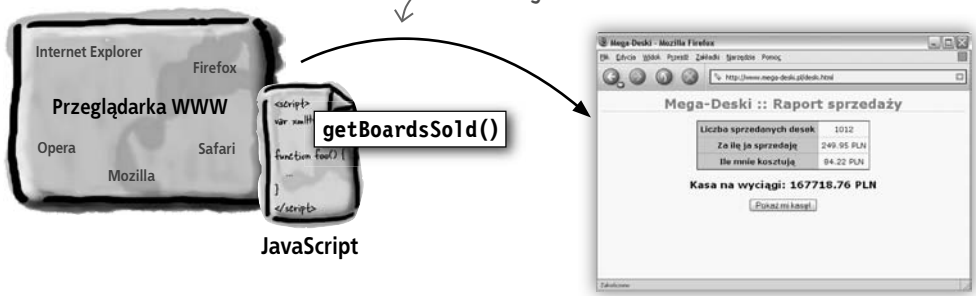
Przeglądarka określa, w jaki sposób należy wykonać żądanie skierowane do serwera Kaśki.



Twój kod prosi o przesłanie żądania, wywołując metodę `request.send(null);`.

⑤ Przeglądarka przekazuje „sterowanie" Kaśce

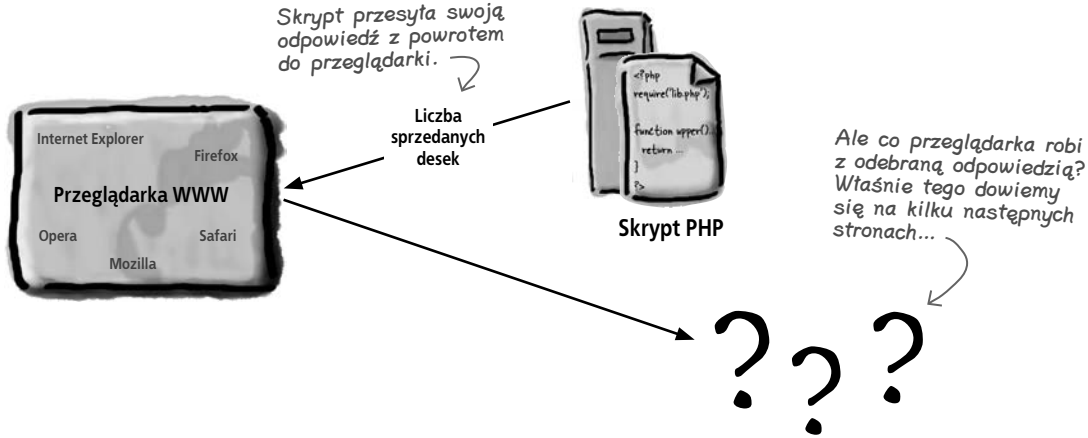
Ponieważ żądanie zostało wykonane w sposób asynchroniczny, zatem przeglądarka przekazuje sterowanie użytkownikowi... żądanie jest dalej obsługiwane w niewidoczny sposób.



Kaśka może zatem dalej używać strony raportu (choć w tym przypadku nie może z nim zrobić zbyt wielu interesujących rzeczy). Ale przynajmniej nie musi oglądać obracającej się piłki ani animowanej klepsydry!

⑥ Przeglądarka pobiera odpowiedź nadesłaną przez serwer

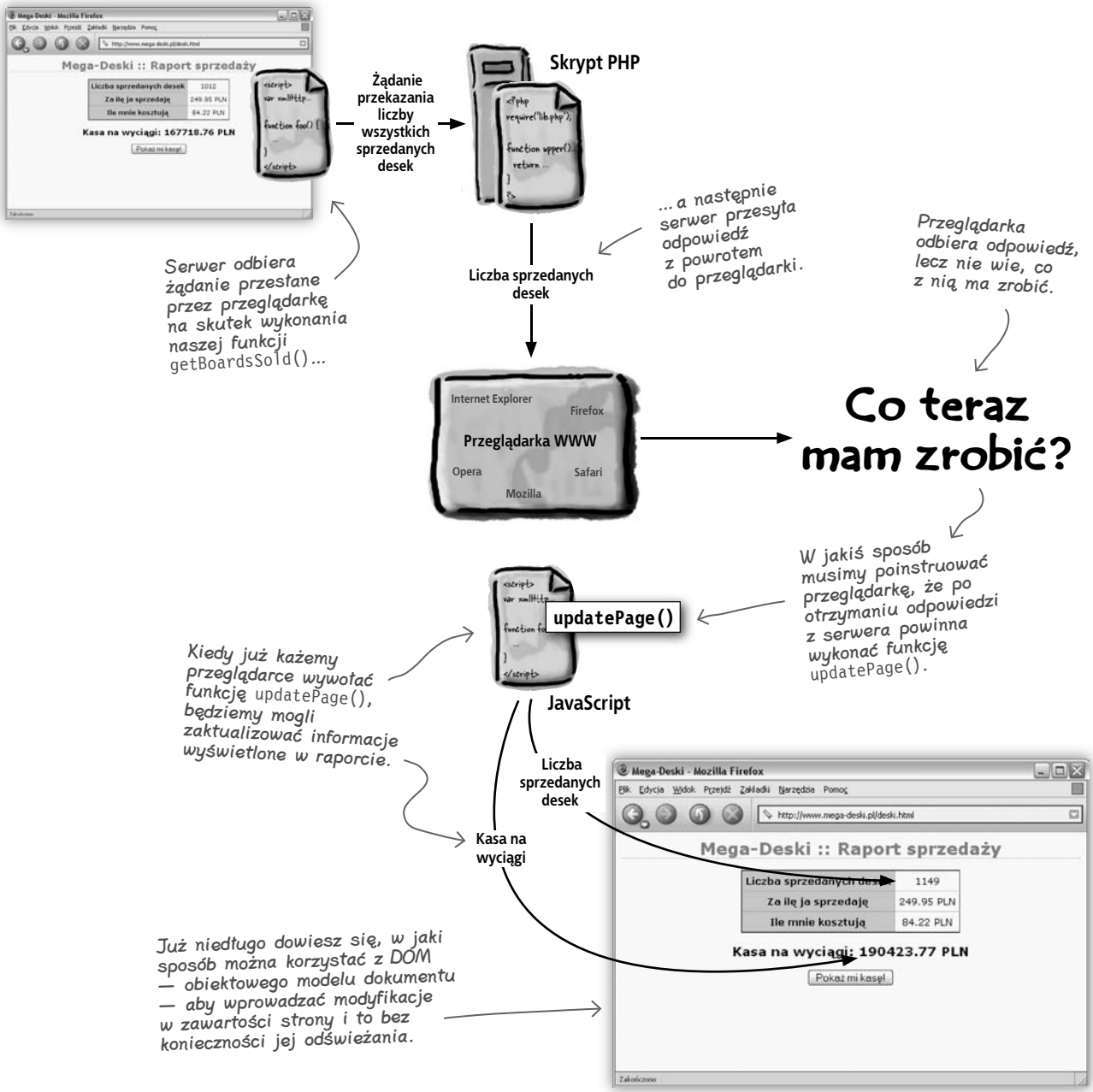
Skrypt przesyła swoją odpowiedź z powrotem do przeglądarki.



Ale co przeglądarka robi z odebraną odpowiedzią? Właśnie tego dowiemy się na kilku następnych stronach...

Co przeglądarka powinna zrobić z odpowiedzią przesłaną z serwera?

Przeglądarka odbierze odpowiedź nadesłaną z serwera, jednak do niczego jej nie wykorzysta, chyba że jej każesz. A zatem musimy dowiedzieć się, w jaki sposób można poinformować przeglądarkę o tym, iż w momencie odebrania odpowiedzi z serwera powinna wywołać funkcję **`updatePage()`**.



Rozmowy przy espresso

Dzisiejszymi kochającymi i pożądającymi kawy gośćmi są:
Przeglądarka WWW oraz Strona HTML

Przeglądarka: Siemasz, stara. Miło cię znowu widzieć. Właśnie rozmawiałam o twoim kumplu — CSS.

Strona: O, naprawdę? Ostatnio przebywam z nim znacznie częściej. Wygląda na to, że ludzie zaczynają więcej czytać o używaniu CSS na stronach... obojgu nam się to bardzo podoba.

Przeglądarka: Tak, ja też jestem waszą fanką. Chociaż, jeśli o mnie chodzi, to wasza dwójka dodaje mi nieco roboty, gdyż muszę pobierać dwa pliki zamiast jednego.

Strona: *Ty* masz coś do roboty? Co masz na myśli? To ja muszę, z bólem serca, żegnać się ze starymi znajomymi, takimi jak atrybut `align` czy też znacznik `font`. I nie rozumiem, co to ma wspólnego z tobą.

Przeglądarka: Chyba sobie żartujesz? Ach te języki znacznikowe... ciągle takie ciemne i tępe. Jak widzę XHTML, wcale nie jest lepszy... Beze mnie nikt by was w ogóle nie zobaczył!

Strona: Niby dlatego, że jesteś programem, który pozwala ludziom na nas patrzeć? Słuchaj, jedynym celem twojego istnienia jest zapewnienie ludziom możliwości oglądania mojego kodu.

Przeglądarka: Dziecinko, nie masz pojęcia o czym gadasz... Ludzie „mogą" oglądać kod HTML w pierwszym lepszym edytorze... jednak nikogo nie interesują twoje nawiasy kątowe i znaczniki `<head>`. Ludzie chcą oglądać cię w takiej postaci jaką ja tworzę — w postaci wizualnej, z tymi wszystkimi obrazkami, tabelami i łączami...

Strona: Hm... mania wielkości... rany! Jestem pewna, że wszyscy ustawiają się w kolejce do twoich drzwi, by zobaczyć, na ile sposobów jesteś w stanie wyświetlić tę samą stronę. Tak... po prostu *ubóstwiam* przeglądarki i to, w jaki cudowny sposób potrafią schrzanić mój wygląd.

Przeglądarka: Wojny przeglądarek zbliżają się do końca, ty niedoinformowana misko nawiasów kątowych. Poza tym, jeśli ludzie pisaliby strony zgodne ze standardami, to nie byłoby żadnych problemów.

Strona: Och, doprawdy? Szkoda, że JavaScript mówił mi coś zupełnie innego. Ostatnio narzekał na konieczność używania dwóch różnych obiektów do przesłania prostego żądania do serwera.

Przeglądarka: Ech... dwóch to i tak znacznie lepiej niż pięciu lub sześciu... a mogę się założyć, że wraz z pojawieniem się nowych wersji przeglądarek, takich jak Internet Explorer czy też Mozilla, będziemy przesyłali żądania, używając tylko jednego obiektu szybciej niż się spostrzeżesz.

Strona: Uważasz, że na wszystko masz gotową odpowiedź.

Przeglądarka: Może nie znam odpowiedzi na wszystkie pytania, jednak na pewno to do mnie trafiają wszystkie odpowiedzi przesyłane przez serwer. I to jest jeszcze jeden powód, dla którego mnie potrzebujesz.

Strona: Wszystkie odpowiedzi? O czym ty mówisz?

Przeglądarka: Może powinnaś zapytać o to swojego przyjaciela JavaScript. Jak widać, wszyscy uważają, że to strona WWW i kod wykonują żądania...

Strona: Bo właśnie tak jest!

Przeglądarka: ...jednak to właśnie ja, w dyskretny i niewidoczny sposób, dbam o to, by strony wyglądały jak należy, żądania były przesyłane a odpowiedzi odbierane i przekazane do odpowiedniego kodu. Beze mnie wy wszystkie byłybyście jedynie zbiorami śmiesznych znaczków, zapisanych w plikach tekstowych.

Strona: ...*zrzędzi*, *zrzędzi*...

createRequest() getBoardsSold() updatePage()

Teraz, kiedy dowiedzieliśmy się czegoś więcej o przeglądarkach WWW, czas ponownie zająć się funkcją getBoardsSold().

A zatem, w jaki sposób możemy dogadać się z przeglądarką? Jak na razie, cały napisany przez nas kod posługuje się jedynie utworzonym obiektem żądania. Ale... chwila... to jest właśnie to! Czy nie możemy użyć obiektu żądania, by dogadać się z przeglądarką?

Przekazywanie instrukcji przeglądarce

Pamiętaj, że musimy powiedzieć przeglądarce, co ma zrobić z odpowiedzią, *zanim* prześle ona żądanie do serwera… W przeciwnym razie, po wysłaniu żądania, działanie metody `getBoardsSold()` skończy się, a nasz kod już nie będzie mógł się dowiedzieć, co należy zrobić z odpowiedzią. Na szczęście, obiekt żądania, którego używamy, dysponuje odpowiednią właściwością:

```
function getBoardsSold() {
  createRequest();
  var url = "pobierzAktualneDane-ajax.php";
  request.open("GET", url, true);
  request.onreadystatechange = updatePage;
  request.send(null);
}
```

Pamiętaj, by określić wartość tej właściwości przed wywołaniem metody `send()`, gdyż w przeciwnym razie funkcja `updatePage()` nie zostanie wywołana.

Jeśli podasz tu nazwę funkcji, to przeglądarka ją wywoła po odebraniu odpowiedzi przesłanej przez serwer.

JavaScript wymaga, by podając nazwę funkcji, w tym miejscu pominąć nawias.

Najczęściej zadawane pytania

P: Serwer komunikuje się z przeglądarką, przeglądarka z JavaScriptem, a JavaScript aktualizuje naszą stronę... Chyba się pogubiłem. Czy możesz mi to jeszcze raz wytłumaczyć?

O: Pamiętaj, że żądania są przesyłane *asynchronicznie*. Kiedy Twój kod każe przeglądarce przesłać żądanie do serwera, serwer musi odpowiedzieć na to żądanie, przesyłając odpowiedź, jednak kod nie czeka, aby się przekonać, jaka odpowiedź zostanie nadesłana. A zatem, kiedy serwer prześle odpowiedź, to przeglądarka będzie musiała określić, co należy z nią zrobić. W momencie odbierania odpowiedzi nie ma żadnego działającego kodu, który czekałby na nią, zatem przeglądarka wykona funkcję podaną we właściwości **`onreadystatechange`** obiektu żądania.

P: Przypomnij mi jeszcze raz, do czego służy właściwość `onreadystatechange`. Nie bardzo to rozumiem.

O: Tak naprawdę, nie jest to takie trudne, jak mogłoby się wydawać. Właściwość ta informuje przeglądarkę, iż w przypadku zmiany stanu żądania — na przykład w momencie odebrania odpowiedzi przesłanej przez skrypt PHP — przeglądarka powinna wywołać funkcję JavaScriptu wskazaną w obiekcie żądania. W naszym przypadku będzie to funkcja **`updatePage()`**.

P: A zatem istnieją także inne czynniki, oprócz zakończenia skryptu, które mogą spowodować wywołanie funkcji `updatePage()`?

O: Owszem. Przeglądarka WWW może się znajdować w kilku różnych „stanach gotowości", a za każdym razem, gdy jej stan ulegnie zmianie, zostanie wywołana funkcja **`updatePage()`**. Tym zagadnieniem zajmiemy się znacznie bardziej szczegółowo w następnym rozdziale, a zatem pamiętaj o nim.

Kiedy przeglądarka WWW odbierze odpowiedź na wygenerowane wcześniej asynchroniczne żądanie, to wywoła wskazaną funkcję JavaScriptu, przekazując do niej odpowiedź serwera.

createRequest() getBoardsSold() updatePage()

Pobieranie odpowiedzi z serwera

W końcu jesteśmy gotowi, by zabrać się za pisanie kodu funkcji **updatePage()**, prawda? Określiłeś już wartość właściwości **onreadystatechange** obiektu żądania, zatem przeglądarka wywoła funkcję **updatePage()** po odebraniu odpowiedzi na żądanie. Jednak czegoś jeszcze nam brakuje…

Już rozumiem, w jaki sposób mogę zmusić przeglądarkę do wywołania funkcji updatePage() w momencie odebrania odpowiedzi, jednak jak pobrać dane przesłane przez serwer w tej odpowiedzi? W jaki sposób mogę się do nich dobrać w kodzie funkcji updatePage()?

Odpowiedź serwera możesz pobrać, używając do tego celu właściwości responseText obiektu żądania.

I tym razem przeglądarka jest pomocna.

Dowiedziałeś się już, w jaki sposób możesz zmusić przeglądarkę do wywołania funkcji **updatePage()** po odebraniu odpowiedzi nadesłanej przez serwer, jednak przeglądarka — oraz używany obiekt odpowiedzi — robią znacznie więcej, aby ułatwić Ci życie.

Kiedy przeglądarka odbierze odpowiedź nadesłaną przez serwer, określa, co dalej należy zrobić, sprawdzając wartość właściwości **onreadystatechange**. Co więcej, skoro zapewne zechcesz w jakiś sposób wykorzystać informacje przesłane z serwera, przeglądarka zapisuje je we właściwości obiektu żądania. Właściwość ta nosi nazwę **responseText**.

A zatem, za każdym razem, gdy będziesz chciał przekonać się, jakie informacje serwer zwrócił w odpowiedzi, wystarczy, że odczytasz wartość właściwości **responseText** obiektu żądania.

Okazuje się, że przed wywołaniem wskazanej przez Ciebie funkcji JavaScriptu, przeglądarka podaje wartości kilku właściwości obiektu żądania. Czytaj uważnie, aby nie przegapić informacji o tym, jak się te właściwości nazywają i do czego służą...

Rozpracuj to

Poznałeś już dwie właściwości używanego przez nas obiektu żądania. Teraz należałoby sprawdzić wiedzę, którą już zdobyłeś. Poniżej, z lewej strony podałem nazwy kilku właściwości obiektu żądania, natomiast z prawej — umieściłem opisy tych właściwości. Sprawdź, czy będziesz w stanie dopasować opisy do nazw właściwości.

`responseText`	Kod statusu HTTP zwrócony przez serwer.
`readyState`	Funkcja, jaką przeglądarka ma wywołać po odebraniu odpowiedzi z serwera.
`onreadystatechange`	Liczba określająca, w jakim stanie znajduje się żądanie: w trakcie wczytywania, w trakcie realizacji, zakończone i tak dalej.
`status`	Dane zwrócone przez serwer w odpowiedzi na żądanie.

Dwóch spośród tych właściwości jeszcze nie poznałeś, ale nie przejmuj się tym i spróbuj odgadnąć, jakie jest ich przeznaczenie.

Planowanie funkcji updatePage()

Przeglądarka zapisuje odpowiedź przesłaną przez serwer w obiekcie żądania, a następnie wywołuje funkcję **updatePage()**. *W końcu* jesteśmy gotowi, by zabrać się za napisanie kodu tej funkcji. Zobaczmy, co musimy w tym celu zrobić:

Po pierwsze: pobrać liczbę określającą zaktualizowaną liczbę sprzedanych desek

Serwer przesyła tę liczbę do przeglądarki, a następnie przeglądarka zapisuje ją we właściwości **responseText**.

```
function updatePage() {
  var newTotal = request.responseText;
}
```

To jest właściwość obiektu żądania, której przeglądarka używa do przekazania danych przesłanych w odpowiedzi do naszego skryptu.

Tu mamy obiekt żądania.

Po drugie: pobrać elementy HTML, których zawartość należy zmienić

Na stronie raportu istnieją dwa elementy, których zawartość musimy zmienić: liczba sprzedanych desek snowboardowych oraz zysk Kaśki. Każdy z tych elementów można pobrać przy użyciu funkcji **getElementById()** JavaScriptu, wykorzystując przy tym ich identyfikatory.

Zapisz każdy z pobranych elementów w zmiennej, dzięki czemu bez problemu będziesz mógł używać ich w dalszej części kodu.

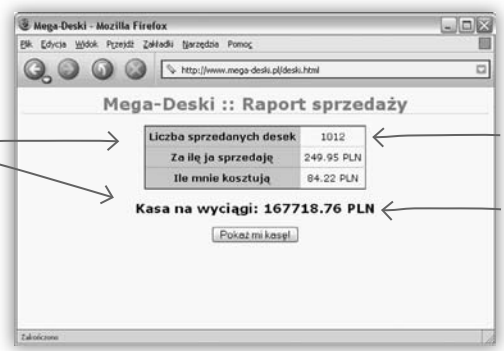
```
function updatePage() {
  var newTotal = request.responseText;
  var boardsSoldEl = document.getElementById("deski-sprzedane");
  var cashEl = document.getElementById("kasa");
}
```

document reprezentuje całą stronę HTML...

...a metoda getElementById() odnajduje na stronie element, którego identyfikator (wartość atrybutu id) odpowiada wartości podanej w wywołaniu.

To są identyfikatory elementów, które chcemy pobrać

Czy pamiętasz, że Kaśka dodała atrybuty id do elementów na stronie raportu, aby móc określić ich postać przy użyciu CSS? Okazuje się, że te same atrybuty ułatwiają dostęp do tych elementów w kodzie JavaScriptu.



1012

167718.76

Po trzecie: dodać odwołanie do pliku JavaScriptu zawierającego funkcje, które ułatwią nam operacje na tekście

Teraz już możesz przystąpić do zmiany liczby sprzedanych desek. Zmiana tekstu prezentowanego wewnątrz elementu **** będzie wymagała napisania dosyć zaawansowanego skryptu. Techniki i rozwiązania, jakie należy w nim zastosować, opiszę szczegółowo w rozdziale 4., poświęconym obiektowemu modelowi dokumentu (DOM). Do tego czasu skorzystamy z Gotowy JavaScript — pliku *text-utils.js*.

Plik text-utils.js możesz znaleźć w przykładach do książki, w katalogu rozdział01/deski.

Plik ten zawiera kilka przydatnych funkcji, które mogą Ci się przydać, jednak w pierwszej kolejności będziesz musiał przekazać aplikacji informację, gdzie ma go szukać. Odwołanie do pliku JavaScript Kaśki możesz dodać w strony HTML przy użyciu elementu **<script>**. Poniżej pokazałem, w jaki sposób można to zrobić:

Cały ten kod jest umieszczony w sekcji nagłówka Twojej strony HTML.

```
<head>
  <title>Mega-Deski</title>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-2" />
  <link rel="stylesheet" type="text/css" href="deski.css" />
  <script type="text/javascript" src="text-utils.js"> </script>
  <script language='javascript' type="text/javascript">
```

To jest początek całego kodu JavaScript, jaki wcześniej dodałeś do swojej strony.

Kiedy dodasz ten wiersz kodu, będziesz mógł używać w swoim skrypcie wszystkich funkcji zdefiniowanych w pliku text-utils.js.

Po czwarte: zmodyfikować raport poprzez zapisanie w nim aktualnych informacji

W pliku *text-utils.js* zdefiniowano funkcję o nazwie **replaceText()**. Możesz ją zastosować do aktualizacji informacji wyświetlonych na stronie raportu:

Wszystkie funkcje zdefiniowane w pliku text-utils.js modyfikują zawartość strony przy wykorzystaniu DOM — obiektowego modelu dokumentu.

Ta funkcja została zdefiniowana w pliku text-utils.js. Wszystkie funkcje znajdujące się w tym pliku opiszę szczegółowo w dalszej części książki.

```
function updatePage() {
  var newTotal = request.responseText;
  var boardsSoldEl = document.getElementById("deski-sprzedane");
  var cashEl = document.getElementById("kasa");
  replaceText(boardsSoldEl, newTotal);
}
```

To jest element, którego tekstowa zawartość zostanie zmieniona...

... a to jest nowa wartość, która zostanie umieszczona w elemencie.

W wyniku wykonania tego wiersza kodu strona raportu zostanie zaktualizowana — pojawi się na niej nowa liczba sprzedanych desek.

Po prostu to zrób

Teraz możesz samodzielnie dokończyć pisanie kodu funkcji updatePage(). Poniżej przedstawiłem jej kod, jednak pozostawiłem w nim wiele pustych miejsc, które musisz uzupełnić. Spróbuj swoich sił i w puste miejsca wpisz kod, jaki powinien się w nich znajdować.

Kiedy już uznasz, że wykonałeś ćwiczenie, porównaj swoje odpowiedzi z rozwiązaniem zamieszczonym na następnej stronie.

```
function updatePage() {
  var newTotal = request.responseText;
  var boardsSoldEl = document.getElementById("deski-sprzedane");
  var cashEl = document.getElementById("kasa");
  replaceText(boardsSoldEl, newTotal);

  /* Określenie zysku Kaśki */
  var priceEl = document.getElementById("_______");
  var price = getText(_______);
  var _______ = document._______("koszt");
  var cost = getText(costEl);
  var cashPerBoard = _______ - _______;
  var cash = _______ * _______;

  /* Aktualizacja zysku Kaśki - kasy na wyciągi */
  cash = Math.round(cash * 100) / 100;
  replaceText(cashEl,_______);
}
```

Funkcja `getText()` została zdefiniowana w pliku text-utils.js. W jej wywołaniu można przekazać dowolny element, a funkcja zwróci umieszczony wewnątrz niego tekst.

Ta prosta sztuczka zapewnia, że liczba będzie mieć jedynie dwa miejsca dziesiętne, podobnie jak wszystkie prezentowane wartości monetarne.

Najczęściej zadawane pytania

P: Kilka razy wspominałeś o DOM — obiektowym modelu dokumentu. Co to jest? I co to coś ma wspólnego z raportem Kaśki?

O: DOM (ang. Document Object Model — obiektowy model dokumentu) to sposób, w jaki przeglądarka przechowuje i reprezentuje strony HTML. Poprzez modyfikowanie DOM można na bieżąco aktualizować wygląd strony, dokładnie w taki sposób, w jaki chcemy to zrobić w raporcie Kaśki. Szczegółowe informacje na temat DOM znajdziesz w rozdziale 4., a na razie możesz korzystać z pliku text-utils.js i nie przejmować się tymi wszystkimi wzmiankami o DOM.

Po prostu to zrób. Rozwiązanie

Teraz możesz już samodzielnie dokończyć pisanie kodu funkcji `updatePage()`. Poniżej przedstawiłem jej kod, jednak pozostawiłem w nim wiele pustych miejsc, które musisz uzupełnić. Spróbuj swoich sił i w puste miejsca wpisz kod, jaki powinien się w nich znajdować.

```
function updatePage() {
  var newTotal = request.responseText;
  var boardSoldEl = document.getElementById("deski-sprzedane");
  var cashEl = document.getElementById("kasa");
  replaceText(boardSoldEl, newTotal);

  /* Określenie zysku Kaśki */
  var priceEl = document.getElementById(" cena ");
  var price = getText( priceEl );
  var costEl = document. getElementById ("koszt");
  var cost = getText(costEl);
  var cashPerBoard = price - cost ;
  var cash = cashPerBoard * newTotal ;

  /* Aktualizacja zysku Kaśki - kasy na wyciągi */
  cash = Math.round(cash * 100) / 100;
  replaceText(cashEl, cash );
}
```

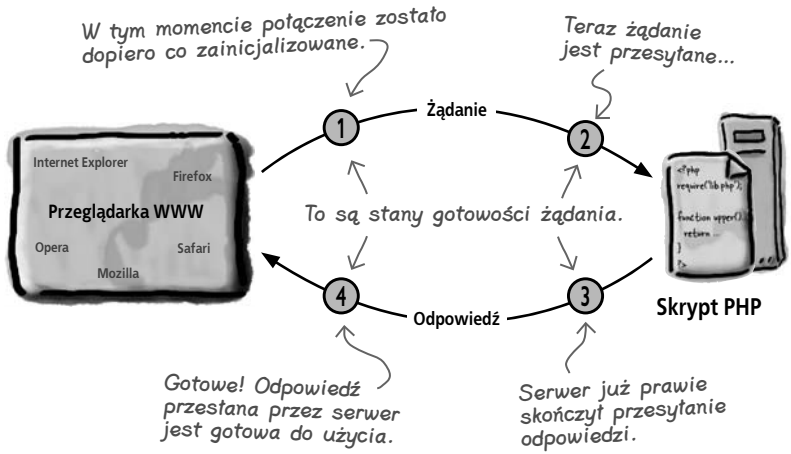
Funkcja `getText()` została zdefiniowana w pliku text-utils.js. W jej wywołaniu można przekazać dowolny element, a funkcja zwróci umieszczony wewnątrz niego tekst.

Ta prosta sztuczka zapewnia, że liczba będzie mieć jedynie dwa miejsca dziesiętne, podobnie jak wszystkie prezentowane wartości monetarne.

createRequest() getBoardsSold() updatePage()

Sprawdź, czy serwer zakończył przesyłanie odpowiedzi

Obecnie założyliśmy, że w chwili wywołania funkcji **updatePage()** serwer zakończył już przesyłanie odpowiedzi… jednak może się okazać, że tak wcale nie jest! Aby zrozumieć, co się dzieje z odpowiedzią serwera, musimy poznać zagadnienia związane ze **stanami gotowości**. Używany w skrypcie obiekt żądania charakteryzuje się odpowiednią właściwością, która przekazuje przeglądarce dość dużo informacji o tym, w jakim **stanie** znajduje się żądanie.



Stan gotowości informuje przeglądarkę, w jakim stanie znajduje się żądanie.

Stany gotowości są powiązane z właściwością onreadystatechange obiektu żądania

Czy pamiętasz właściwość zastosowaną w funkcji **getBoardsSold()**, której zadaniem było przekazanie przeglądarce informacji, co należy zrobić w momencie odebrania odpowiedzi z serwera? Zerknij na poniższy kod, by odświeżyć swoją pamięć:

```
fuction getBoardsSold() {
 createRequest();
 var url = "pobierzAktualneDane-ajax.php";
 request.open("GET", url, true);
 request.onreadystatechange = updatePage;
 request.send(null);
}
```

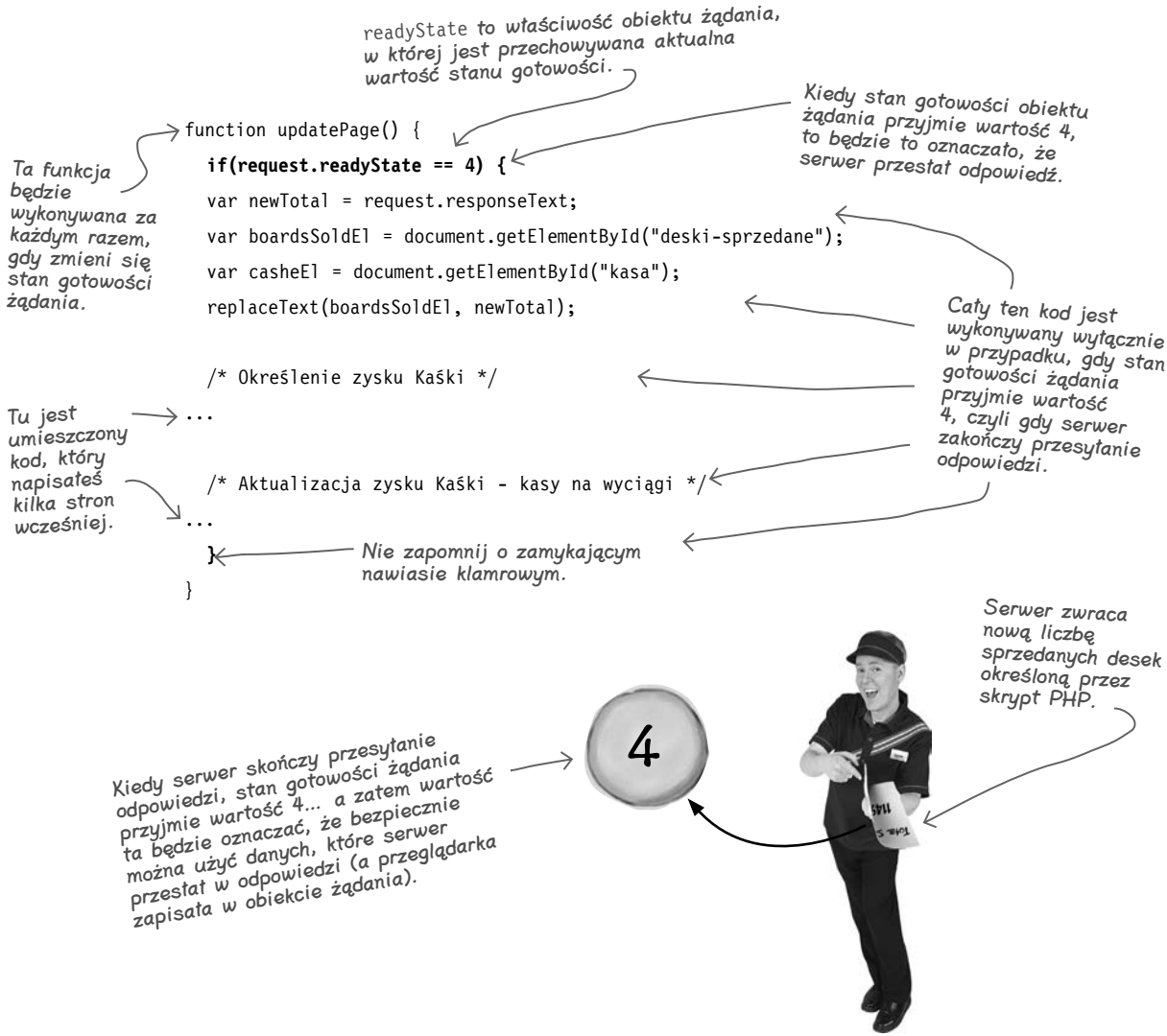
Ta właściwość określa funkcję, która powinna być wykonana za każdym razem, gdy zmieni się stan gotowości żądania.

Ta właściwość dotyczy *każdego* z wyróżnianych stanów gotowości, a nie tylko tego, który oznacza zakończenie obsługi żądania.

Sprawdzanie stanu gotowości żądania

Już wiesz, że przeglądarka wywoła funkcję **updatePage()** wtedy, kiedy odbierze odpowiedź nadesłaną z serwera. Ale jest pewien problem: otóż przeglądarka wywoła tę funkcję za *każdym razem*, gdy zmieni się stan gotowości żądania, a nie tylko po zakończeniu przesyłania żądania z serwera. Spójrz na schemat przedstawiony *na stronie 80* i zwróć uwagę, iż w trakcie obsługi żądania zmienia się także wartość stanu gotowości.

Wygląda na to, że w momencie, gdy serwer skończy przesyłanie odpowiedzi, stan gotowości przyjmuje wartość 4. A zatem, sprawdzając stan gotowości w naszym kodzie, możemy go porównywać właśnie z tą wartością. Dzięki temu możemy mieć pewność, że próba zaktualizowania raportu sprzedaży nastąpi wyłącznie w przypadku, gdy serwer skończył już przesyłanie odpowiedzi.



Po prostu to zrób

Otwórz swój plik *deski.html* i dodaj do niego cały nowy kod JavaScriptu. Upewnij się, że funkcja **updatePage()** zawiera instrukcję warunkową, sprawdzającą stan gotowości żądania, kod służący do aktualizacji liczby sprzedanych desek oraz wyznaczenie i zmianę zysku Kaśki. Oprócz tego musisz się także upewnić, że w tym samym katalogu, w którym są umieszczone pliki *deski.html* i *deski.css*, będzie się także znajdować plik *text-utils.js*.

Najczęściej zadawane pytania

P. Nie rozumiem rozwiązania umieszczonego na samym końcu funkcji `updatePage()`, *na stronie 78*. Chodzi mi o to pomnożenie a następnie podzielenie zysku Kaśki przez 100. Czy w ten sposób nie uzyskasz wartości początkowej?

O. JavaScript może dosyć dziwnie mnożyć liczby. Bardzo często do wynikowej wartości może dodać kilka miejsc dziesiętnych. A zatem, zamiast uzyskać wynik 59,95, JavaScript może zwrócić wartość taką jak 59,9499995. Bez wątpienia Kaśka nie chciałaby zobaczyć takiej liczby w swoim raporcie sprzedaży.

Aby poprawić taką liczbę jak 59,9499995, w pierwszej kolejności należy ją pomnożyć przez 100. W tym przypadku uzyskamy wartość 5994,99995. Następnie można zastosować metodę **`Math.round()`**, aby zaokrąglić uzyskaną wartość do najbliższej liczby całkowitej, dzięki czemu uzyskamy wartość 5994. Ostatnią operacją będzie podzielenie otrzymanej wartości przez 100, dzięki czemu uzyskamy wartość 59,94... czyli dokładnie to, co chciałaby otrzymać Kaśka.

P. Jak działa funkcja `getText()`? Widziałem, że używaliśmy jej w funkcji `updatePage()` *na stronie 78*.

O. **`getText()`** jest funkcją pomocniczą, podobnie jak **`replaceText()`**. W jej wywołaniu przekazuje się element strony WWW, a funkcja zwraca tekst umieszczony wewnątrz tego elementu. W przypadku raportu Kaśki funkcja ta jest używana do pobrania ceny, za jaką Kaśka kupuje deski snowboardowe oraz ceny ich sprzedaży. Obie te wartości są pobierane z elementów **`<span>`**.

P. A zatem co jest z tymi wszystkimi funkcjami pomocniczymi zdefiniowanymi w pliku *text-utils.js*? Czy mam się nimi przejmować?

O. O ile tylko umieścisz na swojej stronie element **`<script>`** odwołujący się do tego pliku, to Twój kod powinien działać bez najmniejszych problemów. Funkcje te wykonują dosyć zaawansowane operacje na DOM, a ich działanie wyjaśnię w rozdziale 4.

Cały kod pliku *text-utils.js* zostanie także zamieszczony w *dodatku 1*, jednak jeśli nie rozumiesz działania tych funkcji, nie musisz się tym na razie przejmować. Kiedy skończysz lekturę niniejszej książki, działanie tych funkcji będzie dla Ciebie jasne i proste.

P. Czyli wykonując jakieś modyfikacje na stronach WWW, wykorzystujemy do tego celu DOM?

O. Tak. Przeglądarki WWW używają DOM do reprezentowania stron HTML. Pisany przez Ciebie kod JavaScript może używać DOM do aktualizowania na bieżąco informacji wyświetlanych na stronie.

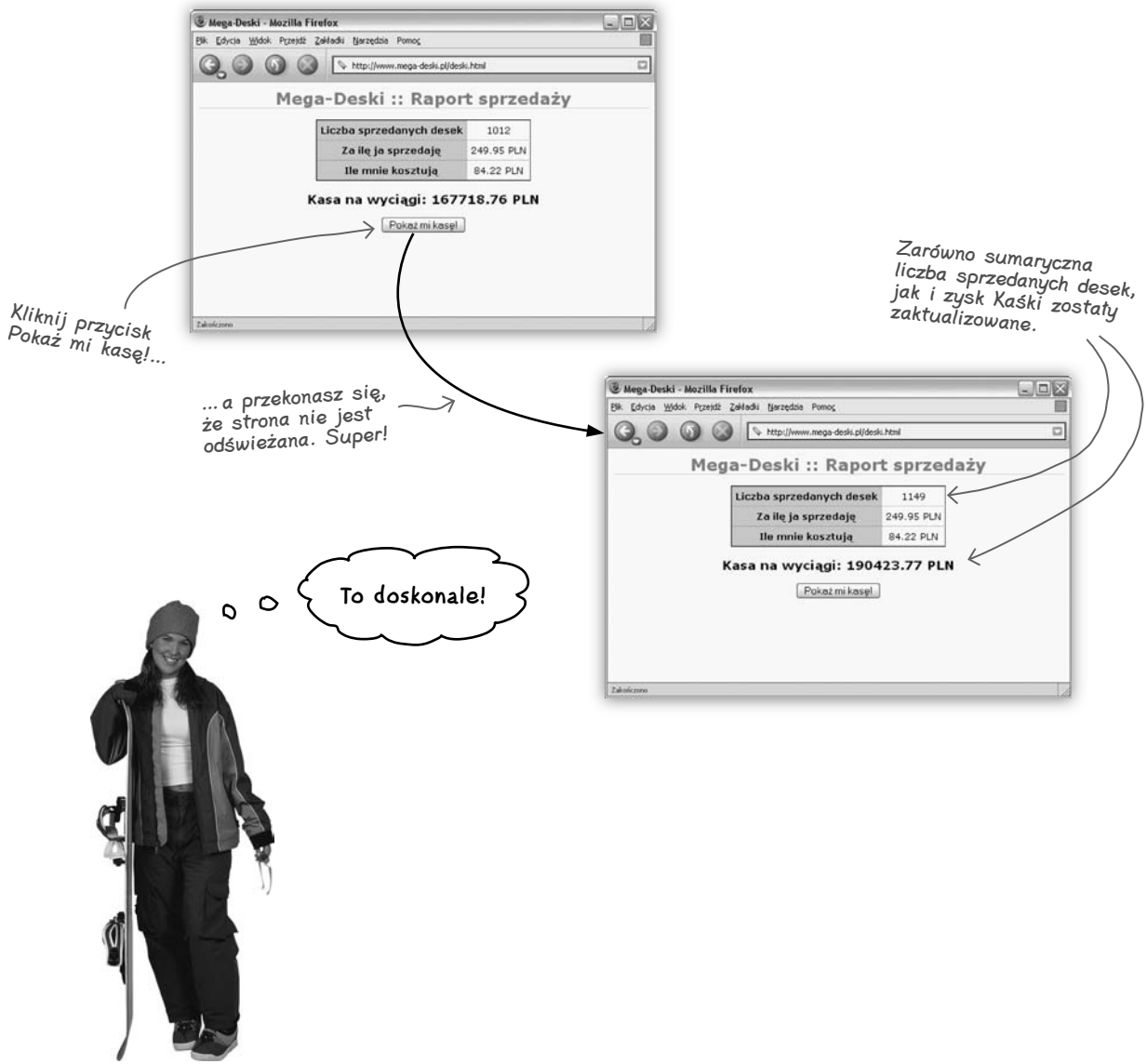
Okazuje się, że już wcześniej używałeś DOM! Za każdym razem, gdy w kodzie JavaScript odwoływałeś się do obiektu **`document`** lub wywoływałeś metodę **`getElementById()`**, używałeś DOM.

P. A właściwość readyState, czy mógłbyś opisać ją jeszcze raz?

O. **`readyState`** jest właściwością obiektu żądania, która pozwala nam na określenie, w jakim stanie aktualnie znajduje się żądanie. Znacznie więcej czasu i uwagi poświęcimy tej właściwości w następnym rozdziale. Na razie wystarczy, abyś wiedział, że kiedy wartość tej właściwości wyniesie 4, to będzie to oznaczało, że serwer przesłał całą swoją odpowiedź.

Pokazujemy Kaśce czary Ajaksa

Czas przekonać się, jakie są efekty naszej ciężkiej pracy nad zmianą raportu Kaśki w aplikację korzystającą z technologii Ajax. Upewnij się, że w pliku *deski.html* znalazł się cały niezbędny kod JavaScriptu i dwukrotnie sprawdź, czy kliknięcie przycisku *Pokaż mi kasę!* powoduje wywołanie metody **`getBoardsSold()`**, a nie przesłanie całego formularza do serwera. Następnie wyświetl stronę *deski.html* w przeglądarce i przekonaj się, co potrafi Ajax!

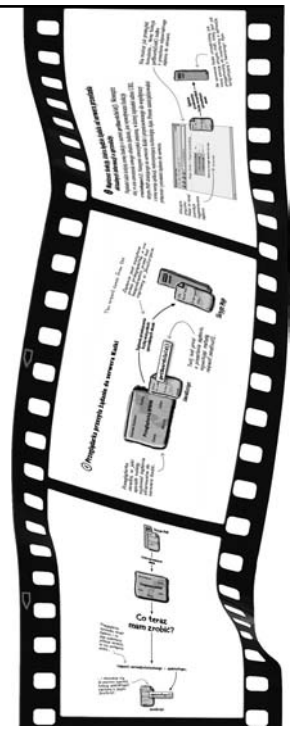


Udało Ci się! Nudną stronę z raportem sprzedaży zmieniłeś na dynamiczną aplikację, używającą technologii Ajax!

Dobra robota. Aplikacja raportu sprzedaży Kaśki z powrotem działa na WWW, dzięki czemu Kaśka wie, ile pieniędzy może przeznaczyć na wyciągi — i to wszystko bez niepotrzebnego oczekiwania na serwer lub narażania się na denerwujące odświeżanie strony. W ramach podziękowania, Kaśka zaoferowała się udzielić Ci jednej darmowej lekcji jazdy na desce.

Przegląd filmu utrwalającego

- Aplikacje asynchroniczne wykonują żądania z wykorzystaniem obiektu JavaScriptu, a nie poprzez przesyłanie formularza.
- Żądania i odpowiedzi są obsługiwane przez przeglądarkę, a nie bezpośrednio przez kod JavaScriptu.
- Kiedy przeglądarka odbierze odpowiedź na Twoje asynchroniczne żądanie, wywoła wskazaną przez Ciebie funkcję JavaScriptu, przekazując do niej odpowiedź otrzymaną z serwera.



Czekajcie! Nie naciskajcie!

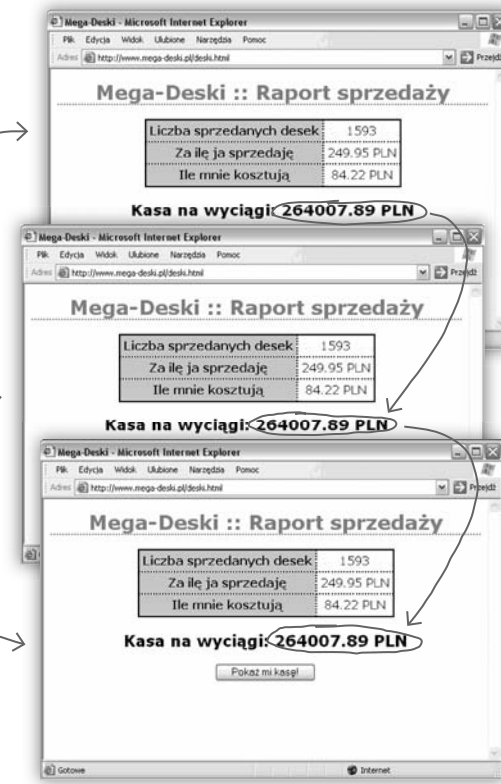
Na Maku Kaśki wszystko działa świetnie, ale na moim komputerze z systemem Windows informacje są aktualizowane tylko za pierwszym razem, gdy kliknę przycisk Pokaż mi kasę!. Później już cały czas wyświetlane są te same liczby... o co w tym chodzi?

Brat Kaśki i kierownik sklepu. Jest zagorzałym zwolennikiem systemu Windows, choć Kaśka woli Maka.

Oto raport po pierwszym kliknięciu przycisku Pokaż mi kasę!... jak widać pojawiły się na nim nowe, zaktualizowane wartości.

W przypadku używania przeglądarki Internet Explorer, po pierwszym kliknięciu przycisku wszystko działa doskonale...

...jednak potem każde kliknięcie przycisku Pokaż mi kasę! powoduje zwrócenie tych samych wyników. Nie ma wątpliwości, że ten problem występuje jedynie w systemie Windows.



Co się dzieje?

Czy popełniliśmy jakiś błąd?

Czy aplikacje wykorzystujące technologię Ajax działają w przeglądarce Internet Explorer?

Aby znaleźć odpowiedzi na te oraz wiele innych pytań, musisz poczekać aż dotrzemy do rozdziału 2....

60-sekundowe podsumowanie

- Tradycyjny model programowania aplikacji internetowych wiąże się z wykonywaniem żądań skierowanych do serwera i odbieraniem odpowiedzi zawierających zaktualizowane informacje, które zazwyczaj są zapisane wewnątrz zupełnie nowej strony WWW.
- Aplikacje korzystające z technologii Ajax używają asynchronicznych skryptów JavaScript.
- Aplikacje korzystające z technologii Ajax mogą wykonywać żądania i odbierać odpowiedzi bez konieczności odświeżania całych stron WWW.
- Asynchroniczne skrypty JavaScript nie czekają na otrzymanie odpowiedzi z serwera. Użytkownicy wciąż będą mogli korzystać ze strony, nawet wtedy, gdy serwer wciąż będzie przetwarzał żądanie.
- Przeglądarki WWW przekształcają kod HTML i CSS na strony WWW, które użytkownik może oglądać na ekranie swojego komputera. Oprócz tego to przeglądarki wykonują skrypty pisane w języku JavaScript i umieszczane na stronach.
- W aplikacjach korzystających z technologii Ajax serwer zazwyczaj przesyła w odpowiedzi tylko te dane, które są żądane przez aplikację, bez żadnego dodatkowego kodu HTML bądź kodu określającego sposób prezentacji danych.
- JavaScriptu można użyć do wykonania zarówno żądania synchronicznego, jak i asynchronicznego.
- JavaScript udostępnia kilka procedur obsługi zdarzeń, które można zastosować do wykonywania kodu w przypadku wystąpienia różnego typu zdarzeń. Dwoma popularnymi przykładami takich procedur obsługi zdarzeń są: `onClick()` oraz `onChange()`.
- Przeglądarka cały czas zna bieżący stan gotowości żądania i pozwala odczytywać go w kodzie JavaScript.
- Przy zastosowaniu właściwości `onreadystatechange` obiektu żądania można nakazać przeglądarce wykonywanie wskazanej funkcji JavaScriptu za każdym razem, gdy zmieni się stan gotowości żądania.
- Kiedy stan gotowości żądania przyjmie wartość 4, będzie to oznaczało, że żądanie zostało obsłużone, a serwer wysłał odpowiedź do przeglądarki.

Rozwiązanie ćwiczenia

Po prostu to zrób. Rozwiązanie

Otwórz swój plik *deski.html* i dodaj do niego funkcję **`getBoardsSold()`**, umieszczając ją bezpośrednio poniżej funkcji **`createRequest()`**. Następnie przekonaj się, czy będziesz potrafił dodać do funkcji **`getBoardsSold()`** wiersz kodu tworzący nowy obiekt żądania (patrz krok „a" na liście przedstawionej *na stronie 44*).

```
<script language='javascript' type="text/javascript">
  var request = null;

  function createRequest() {
   try {
     request = new XMLHttpRequest();
   } catch (trymicrosoft) {
     try {
       request = new ActiveXObject("Msxml2.XMLHTTP");
     } catch (othermicrosoft) {
       try {
         request = new ActiveXObject("Microsoft.XMLHTTP");
       } catch (failed) {
         request = null;
       }
     }
   }

   if (request == null)
     alert("Nie udało się utworzyć obiektu żądania!");
  }

  function getBoardsSold() {
    createRequest();
  }
</script>
```

To wszystko jest gotowy JavaScript, który samodzielnie powinieneś wpisać w kodzie strony.

Oto początek Twojej funkcji `getBoardsSold()`...

... która używa funkcji `createRequest()` do utworzenia obiektu żądania.

Kim jestem

Nadszedł czas, by przetestować umiejętności kojarzenia wysokiego poziomu. W naszej sztuce pod tytułem „Ajax" jest jedna bardzo ważna postać, której jeszcze nie spotkałeś i która właśnie przygotowuje się do swojego wielkiego wejścia na scenę. Sprawdź, czy będziesz w stanie zaskoczyć tę postać, odgadując jej tożsamość jeszcze zanim przejdziesz na następną stronę. Udało nam się przedstawić kilka wypowiedzi tej postaci, które być może ułatwią Ci zadanie.

➡ **„Śpiew? Cóż, bardzo wiele osób nie zdaje sobie z tego sprawy, jednak przeszłam klasyczny trening. I kocham Wagnera."**

To oczywiście przeglądarka Opera, która nieco przesadza, opowiadając o swoich umiejętnościach wokalnych.

➡ **„Podaj mi swoje zmęczone, biedne, tulące się do siebie nawiasy, które nie mogą się odczekać własnego stylu."**

Zdecydowanie chodzi o przeglądarkę WWW, żądającą kodu HTML.

➡ **„Oczywiście, że uwielbiam sporty wodne. W rzeczywistości byłam zmiennikiem w filmie Na fali i prawie udało mi się dostać główną rolę w filmie Blue Crush.**

„Surfowanie" po WWW... przeglądarki właśnie do tego służą.

Netscape a Internet Explorer, ktoś coś wie na ten temat? Pamiętacie wojny przeglądarek?

➡ **„Jestem tak twarda jak wymaga tego sytuacja... kilka lat temu brałam nawet udział w konfliktach zbrojnych".**

Kim jestem? Przeglądarką WWW

Rozpracuj to. Odpowiedzi

Aby przerobić raport ze sprzedaży Kaśki na aplikację wykorzystującą technologię Ajax, będziesz potrzebował kilku funkcji napisanych w języku JavaScript. Poniżej podaliśmy nazwy trzech funkcji. Narysuj linię łączącą nazwy funkcji z opisami określającymi, jakie będzie znaczenie danej funkcji w końcowej wersji aplikacji.

`getBoardSold()` Utworzenie nowego obiektu służącego do komunikacji z serwerem.

`updatePage()` Przesłanie do serwera żądania o aktualne informacje o liczbie sprzedanych desek.

`createRequest()` Wyświetlenie na stronie aktualnych informacji o liczbie desek jakie udało się sprzedać Kaśce oraz o wysokości zarobionej kwoty.

Rozpracuj to. Odpowiedzi

Poznałeś już dwie właściwości używanego przez nas obiektu żądania. Teraz należałoby sprawdzić wiedzę, którą już zdobyłeś. Poniżej, z lewej strony podałem nazwy kilku właściwości obiektu żądania, natomiast z prawej — umieściłem opisy tych właściwości. Sprawdź, czy będziesz w stanie dopasować opisy do nazw właściwości.

`responseText` Kod statusu HTTP zwrócony przez serwer.

`readyState` Funkcja, jaką przeglądarka ma wywołać po odebraniu odpowiedzi z serwera.

`onreadystatechange` Liczba określająca, w jakim stanie znajduje się żądanie: w trakcie wczytywania, w trakcie realizacji, zakończone i tak dalej.

`status` Dane zwrócone przez serwer w odpowiedzi na żądanie.