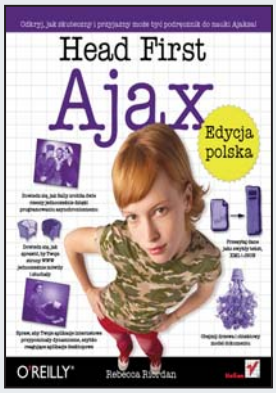
Head First Ajax. Edycja polska



Autor: Rebecca Riordan
Tłumaczenie: Marcin Rogóż
ISBN: 978-83-246-1778-4
Tytuł oryginału: Head First Ajax
Format: 200×230, stron: 516

Technologia AJAX (skrót od ang. Asynchronous JavaScript and XML) tchnęła nowe życie w strony internetowe. Sprawiła, że stały się one interaktywne, przyjazne w użyciu oraz dynamiczne. Dzięki niej aplikacje internetowe coraz bardziej przypominają te standardowe, znane z codziennej pracy. Jeśli jednak ogrom możliwości tej technologii wydaje Ci się trudny do opanowania – jesteś w błędzie! Przekonaj się, że z dobrym podręcznikiem nic nie bywa trudne! „Head First Ajax. Edycja polska" to kolejny przyjazny podręcznik z cieszącej się wielkim uznaniem Czytelników serii Head First. Znajdziesz tu nowatorskie i skuteczne techniki nauki, a przy tym dużo praktycznych informacji i humoru. Trudno wyobrazić sobie lepsze warunki do zdobywania nowej wiedzy. Dzięki tej książce dowiesz się, jak myśleć „po ajaksowemu", obsługiwać zdarzenia, okiełznać asynchroniczność oraz wykorzystać model DOM i format JSON. Te oraz wiele innych ciekawych wiadomości, dzięki którym szybko opanujesz tajniki AJAX-a, znajdziesz właśnie w tym wyjątkowym podręczniku.

- Przeznaczenie technologii AJAX
- Skutki asynchroniczności żądań w AJAX-ie
- Obsługa zdarzeń w języku JavaScript
- Wykorzystanie wielu procedur dla jednego zdarzenia
- Operacje na drzewie DOM
- Zastosowanie frameworków oraz innych bibliotek
- Użycie formatu XML w żądaniach i odpowiedziach
- Format JSON
- Tworzenie formularzy i ich walidacja
- Żądania POST

Odkryj, jak skuteczny i przyjazny może być podręcznik do nauki AJAX-a!

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

Spis treści (skrócony)

Spis treści (z prawdziwego zdarzenia)

Wprowadzenie

Skierujmy mózg na Ajaksa. *Ty* starasz się czegoś *nauczyć*, natomiast *mózg* stara ci się wyświadczyć przysługę, pilnując, aby to, czego się uczysz, nie zostało *zapamiętane*. Mózg myśli sobie: „Lepiej zostawię miejsce na bardziej istotne sprawy, na przykład których zwierząt unikać, czy też dlaczego uprawianie snowboardingu nago jest złym pomysłem”. A więc *jak* można zmylić mózg, aby myślał, że twoje życie zależy od znajomości Ajaksa?

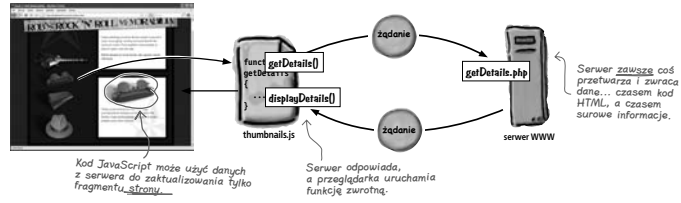
Ajax — zastosowanie

1 Aplikacje internetowe dla nowego pokolenia

Zmęczony czekaniem na przeładowanie strony? Sfrustrowany pokracznymi interfejsami aplikacji internetowych? Czas, aby zaczęły one przypominać przyjemne w użyciu aplikacje desktopowe. Jak to zrobić? Za pomocą **Ajaksa** — twojej przepustki do tworzenia *bardziej interaktywnych, sprawniej reagujących* i *łatwiejszych w użyciu aplikacji internetowych*. Pomiń drzemkę, musisz dopieścić swoje witryny. Czas na zawsze pozbyć się niepotrzebnych i długotrwałych przeładowań strony.

Strony ajaksowe rozmawiają z serwerem tylko wtedy, kiedy muszą... i tylko o tym, o czym serwer wie.

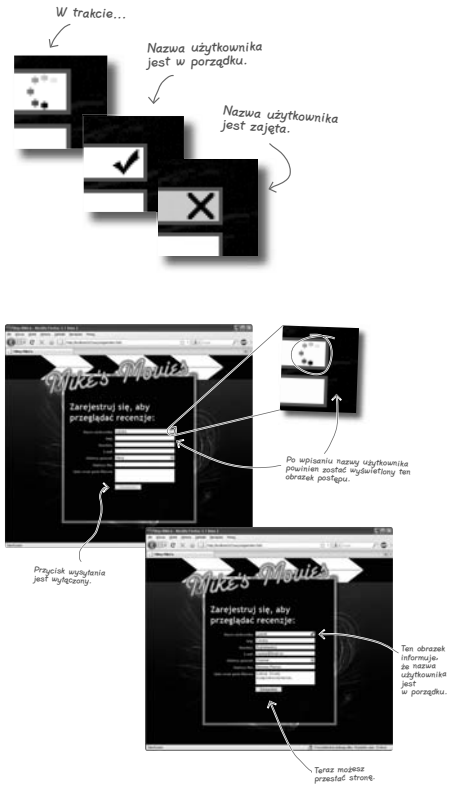
Problemem witryny Roba nie są zbyt wolne serwery, ale fakt, że strony *ciągle* wysyłają żądania... nawet wtedy, gdy nie muszą.



Projektowanie aplikacji w metodologii Ajax

2 Myślenie „po ajaksowemu”

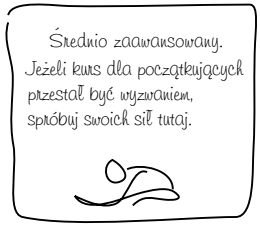
Witamy wśród aplikacji ajaksowych — to zupełnie nowy świat internetowy. Utworzyłeś już swoją pierwszą aplikację w metodologii Ajax i zaczynasz zastanawiać się, jak wprowadzić żądania asynchroniczne do pozostałych swoich witryn. Ale w Ajaksie nie wszystko sprowadza się do programowania. Musisz ***inaczej myśleć o aplikacji***. Samo wykonywanie żądań asynchronicznych nie sprawia, że staje się ona bardziej przyjazna dla użytkownika. To ty powinieneś mu pomóc **uniknąć popełnienia błędów**, a to oznacza konieczność **powtórnego przemyślenia projektu** całej aplikacji.



Zdarzenia w JavaScripcie

3 Reagowanie na użytkowników

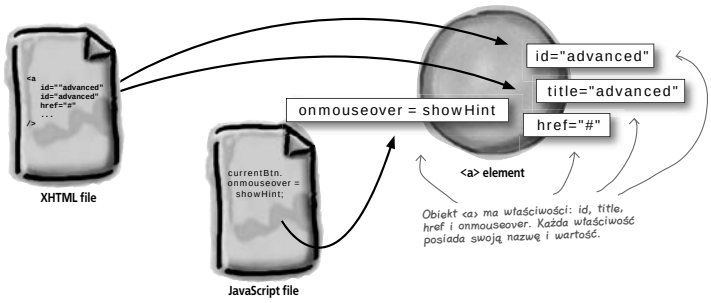
Czasami kod musi reagować na inne rzeczy dziejące się w aplikacji internetowej... W takiej sytuacji przydatne są **zdarzenia**. Zdarzenie jest *czymś, co dzieje się* na stronie, w przeglądarce, a nawet na serwerze. Nie wystarczy jedynie wiedzieć o zdarzeniach... Czasami trzeba na nie odpowiedzieć. Tworząc kod i rejestrując go jako **procedurę obsługi zdarzenia**, możesz sprawić, aby przeglądarka wykonywała go zawsze, gdy wystąpi określone zdarzenie. Połącz zdarzenia i ich procedury obsługi, a otrzymasz **interaktywne aplikacje internetowe**.



Kilka procedur obsługi zdarzeń

4 Dwoje to już towarzystwo

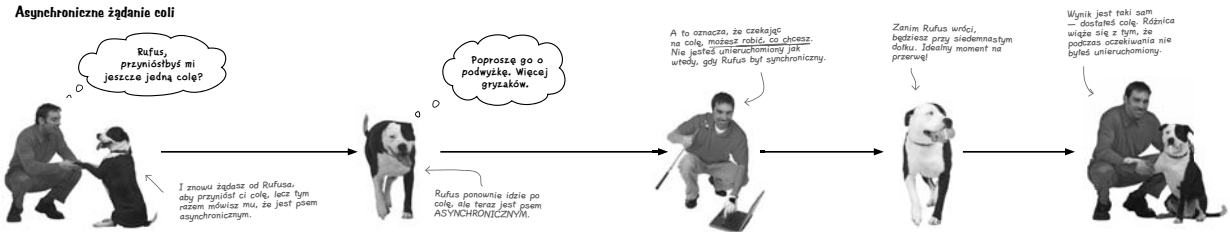
Jedna procedura obsługi zdarzenia nie zawsze wystarcza. Czasami zdarzenie musi wywołać kilka procedur obsługi. Być może musisz utworzyć jakieś operacje specyficzne dla zdarzenia oraz kod generyczny; wówczas umieszczenie tego wszystkiego w jednej procedurze obsługi nie uda się. Albo po prostu chcesz utworzyć **jasny kod wielokrotnego użytku** i masz **dwa fragmenty funkcjonalności** wyzwalane przez to **samo zdarzenie**. Na szczęście możemy użyć metod z **DOM Poziom 2,** aby przypisać kilka procedur obsługi do **jednego zdarzenia.**



Aplikacje asynchroniczne

5 To jak odnawianie prawa jazdy

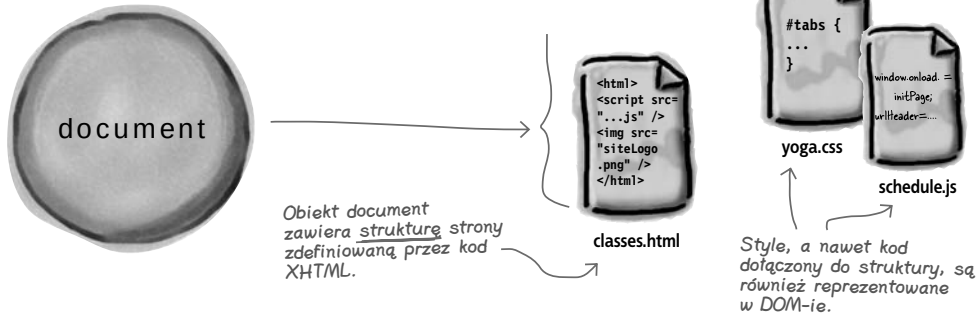
Masz dość czekania? Nienawidzisz opóźnień? Możesz coś z tym zrobić, wykorzystując asynchroniczność! Utworzyłeś już kilka stron wykonujących żądania asynchroniczne do serwera, aby użytkownik nie musiał czekać na odświeżenie strony. W tym rozdziale jeszcze dokładniej omówimy szczegóły budowania aplikacji asynchronicznych. Dowiesz się, **co tak naprawdę oznacza asynchroniczność**, nauczysz się używać **kilku żądań asynchronicznych**, a nawet utworzysz funkcję **monitorującą**, aby asynchroniczność nie wprawiała w zakłopotanie ani ciebie, ani użytkowników.



Obiektowy model dokumentu

6 Leśnictwo na stronie WWW

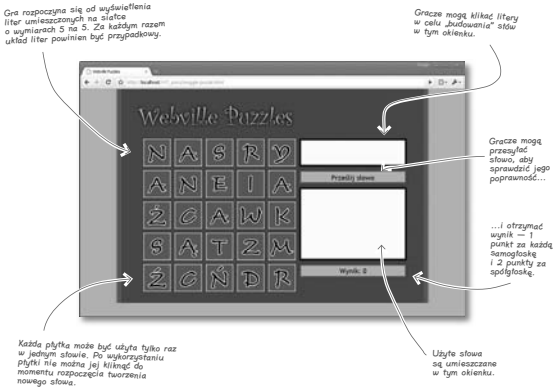
Poszukiwany: Łatwe w aktualizacji strony WWW. Czas wziąć sprawy w swoje ręce i zacząć pisać kod aktualizujący strony WWW w trakcie ich przeglądania. Dzięki **obiektowemu modelowi dokumentu** twoje aplikacje będą w całkowicie nowy sposób odpowiadały na działania użytkowników, a ty na zawsze wyeliminujesz niepotrzebne przeładowania stron. Po przeczytaniu tego rozdziału będziesz potrafił wyszukać, przenieść i zaktualizować zawartość znajdującą się dosłownie w dowolnym miejscu strony. A więc przewróć kartkę i udajmy się na przechadzkę po Szkółce Leśnej Webville.

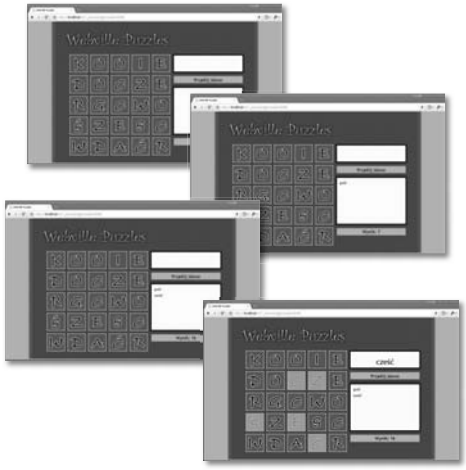


Manipulowanie DOM-em

7 Moje życzenie jest dla ciebie rozkazem

Czasami potrzebujesz tylko trochę kontroli nad ~~umysłem~~ DOM-em. Dobrze wiedzieć, że przeglądarki przekształcają kod XHTML w drzewa DOM. Możesz wiele osiągnąć, poruszając się po nich. Ale potężne możliwości daje dopiero **przejęcie kontroli nad drzewem DOM** i sprawienie, by wyglądało ono tak, jak *Ty* tego chcesz. Zdarza się, że musisz **dodać nowy element** i trochę tekstu lub **usunąć ze strony element** taki jak `<img>`. To i wiele innych rzeczy możesz osiągnąć za pomocą DOM-u, a przy okazji **pozbyć się tej kłopotliwej właściwości `innerHTML`**. Wynik? Kod, który potrafi zrobić ze stroną więcej, *bez* mieszania prezentacji i struktury w JavaScripcie.





Frameworki i zestawy narzędzi

8

Nie ufaj nikomu

A więc czym tak naprawdę są te frameworki ajaksowe? Jeżeli spędziłeś trochę czasu, pracując nad aplikacjami internetowymi, prawdopodobnie natknąłeś się na przynajmniej jeden framework JavaScript lub Ajax. Niektóre frameworki oferują **wygodne metody pracy z DOM-em**. Inne ułatwiają **walidację** i **wysyłanie żądań**. Jeszcze inne zawierają biblioteki z gotowymi **efektami wizualnymi** w JavaScripcie. Ale który powinieneś wybrać? I skąd masz wiedzieć, co tak naprawdę dzieje się w takim frameworku? Musisz zrobić coś więcej, niż tylko skorzystać z kodu napisanego przez innych... Czas ***przejąć kontrolę nad swoimi aplikacjami***.

Zalety frameworków

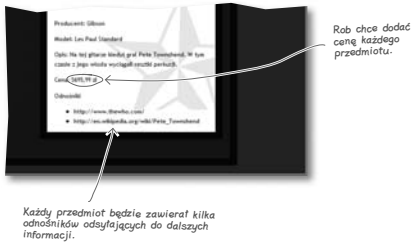
Wady frameworków

Żądania i odpowiedzi XML

9

Więcej niż mogą wyrazić słowa

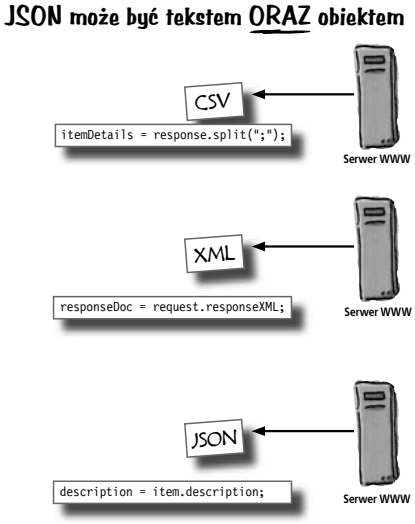
Jak opisałbyś siebie za 10 lat? A za 20? Czasami potrzebujesz danych, **które zmieniają się razem z twoimi potrzebami...** lub potrzebami twoich klientów. Dane, z których teraz korzystasz, mogą się zmienić za kilka godzin, dni lub miesięcy. Dzięki XML-owi, ***rozszerzalnemu językowi znaczników***, dane mogą **same się opisywać**. Oznacza to, że twoje skrypty nie będą wypełnione instrukcjami `if`, `else` i `switch`. Zamiast tego możesz wykorzystać dostarczane przez XML opisy do określenia, jak użyć danych zawartych w XML-u. Wynik: **większa elastyczność** i **łatwiejsza obsługa danych**.



10 JSON

Syn JavaScriptu

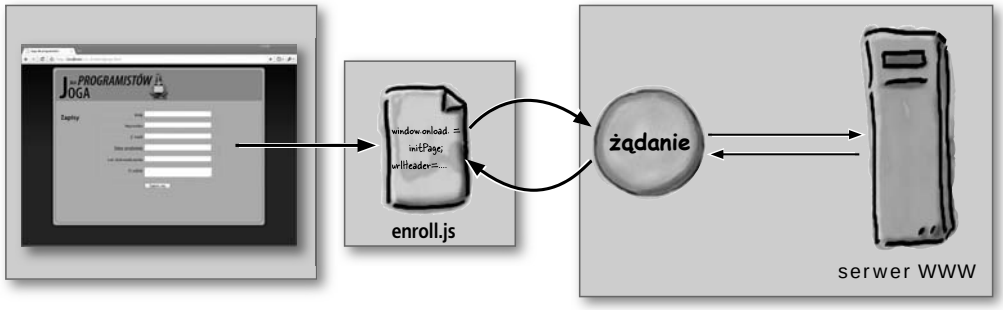
JavaScript, obiekty i notacja, o rany! Jeżeli będziesz musiał kiedyś reprezentować obiekty w JavaScripcie, pokochasz JSON, ***JavaScript Standard Object Notation.*** Dzięki JSON-owi możesz **reprezentować złożone obiekty i odwzorowania** za pomocą tekstu i kilku nawiasów klamrowych. Co więcej, możesz **wysyłać i odbierać JSON** w innych językach, takich jak PHP, C# i Ruby. Ale jak JSON sprawuje się jako format danych? Obróć kartkę i się przekonaj...



Formularze i walidacja

11 Powiedz to, co chciałeś powiedzieć

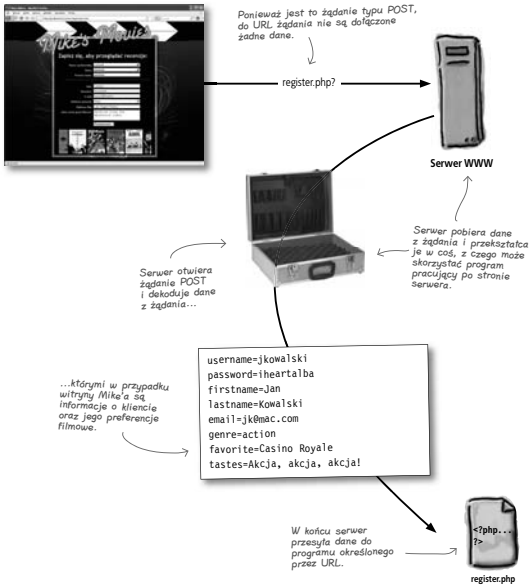
Od czasu do czasu każdy popełnia błędy. Pozwól człowiekowi mówić (lub pisać) przez kilka minut, a prawdopodobnie popełni przynajmniej jeden **błąd** lub dwa. A jak mają **odpowiadać na błędy** nasze aplikacje internetowe? Musimy **sprawdzać poprawność** wprowadzanych przez użytkowników danych i reagować, gdy występują w nich jakieś usterki. Ale co powinno się czym zajmować? Co powinna robić strona WWW? Co powinien robić JavaScript? I jaka jest rola serwera w **walidacji** i **integralności danych**? Przewróć kartkę, aby poznać odpowiedzi nie tylko na te pytania.



12 Żądania POST

Paranoja to twoja przyjaciółka

Ktoś cię obserwuje. Teraz. Naprawdę. *Freedom of Information Act*? Czy to nie ta ustawa, która ułatwiła powstanie internetu? Obecnie wszystko, co użytkownik wpisze do formularza lub kliknie na stronie WWW, podlega **kontroli**. A to przez administratora sieci, a to przez firmę próbującą poznać preferencje klientów, a to przez hakera lub spamera — twoje ***informacje nie są bezpieczne, jeżeli ich nie zabezpieczasz***. W przypadku stron WWW musisz **chronić dane użytkowników** po kliknięciu przycisku *Wyślij*.



Pozostałości

A

Pięć najważniejszych tematów (których nie omówiliśmy)

To była długa podróż... i niemal dotarłeś do końca. Trudno jest nam pogodzić się z myślą, że nas opuścisz, ale zanim odejdziesz, musimy omówić jeszcze kilka zagadnień. Nie mogliśmy zmieścić całej tematyki związanej z Ajaksem w jednej 600-stronicowej książce. Cóż, *próbowaliśmy*... ale dział marketingu stwierdził, że 14-kilogramowa książka techniczna nie będzie zbyt dobrze prezentowała się na półce. Więc wyrzuciliśmy wszystko, co nie jest niezbędne, a pozostałe ważne informacje zamieściliśmy w tym dodatku.

Funkcje narzędziowe

B

Po prostu daj mi kod

Czasami chcesz mieć po prostu wszystko w jednym miejscu. Spędziłeś wiele czasu, używając *utils.js*, naszej małej klasy funkcji narzędziowych, które obsługują Ajax, DOM i zdarzenia. Na kolejnych stronach wszystkie te funkcje zostaną umieszczone w jednym miejscu i będziesz mógł ich używać we własnych skryptach narzędziowych oraz aplikacjach. Po raz ostatni przyjrzyj się tym funkcjom i bierz się do pracy nad własnymi narzędziami!

S

Myślenie „po ajaksowemu”

Witamy wśród aplikacji ajaksowych — to zupełnie nowy świat internetowy. Utworzyłeś już swoją pierwszą aplikację w metodologii Ajax i zaczynasz zastanawiać się, jak wprowadzić żądania asynchroniczne do pozostałych swoich witryn. Ale w Ajaksie nie wszystko sprowadza się do programowania. Musisz ***inaczej myśleć o aplikacji***. Samo wykonywanie żądań asynchronicznych nie sprawia, że staje się ona bardziej przyjazna dla użytkownika. To ty powinieneś mu pomóc **uniknąć popełnienia błędów**, a to oznacza konieczność **powtórnego przemyślenia projektu** całej aplikacji.

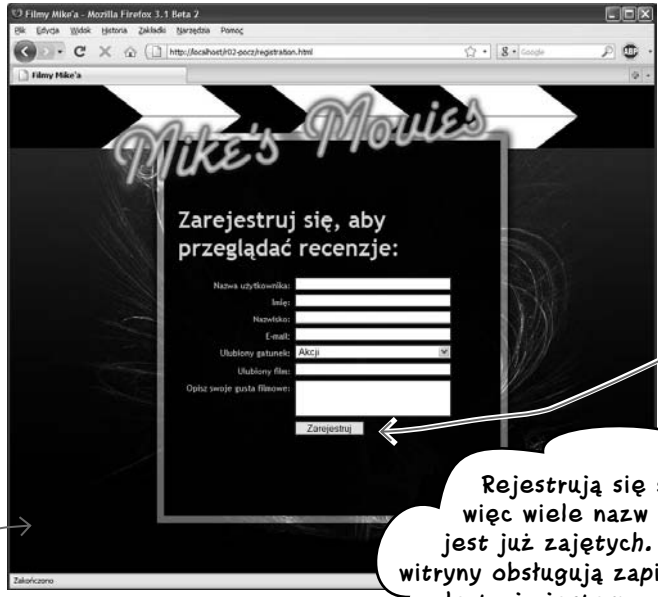
Tradycyjna witryna Mike'a ~~jest do bani~~

Uwaga od działu HR: Czy możemy użyć mniej obraźliwego określenia? Na przykład „irytuje wszystkich użytkowników"?

Mike pisze świetne recenzje filmowe i zapragnął przenieść je do internetu. Niestety, pojawiły się problemy ze stroną rejestracji. Internauci odwiedzają witrynę, wybierają nazwę użytkownika, wpisują jeszcze kilka informacji i wysyłają formularz, aby uzyskać dostęp do recenzji.

Jednak gdy nazwa użytkownika jest już zajęta, pojawia się problem — serwer zwraca ponownie początkową stronę, komunikat o błędzie... bez żadnych informacji wpisanych przez użytkownika. Ponadto internauci denerwują się, że muszą czekać na nową stronę, a potem nie otrzymują niczego poza komunikatem o błędzie. Oni chcą recenzji filmów!

Użytkownicy nie powinni być zmuszani do wypełnienia ośmiu pól tylko po to, aby dowiedzieć się, czy dane w pierwszym polu są poprawne.



Teraz użytkownik wypełnia formularz i klika przycisk „Zarejestruj"... Następnie czeka, mając nadzieję, że wszystko się uda.

Witryna wygląda świetnie i zawiera mnóstwo świetnych recenzji... ale tylko dla użytkowników, którym uda się zarejestrować i przejść dalej, poza stronę rejestracji.

Rejestrują się setki osób, więc wiele nazw użytkowników jest już zajętych. Wszystkie inne witryny obsługują zapisy w ten sposób, ale to ja jestem zalewany skargami. Możesz mi pomóc?

Ćwiczenie

Mike jest w prawdziwych tarapatach. Ty jednak, nawet po napisaniu jednej aplikacji ajaksowej, powinieneś mieć kilka pomysłów na usprawnienie jego witryny. Spójrz na schemat obecnego sposobu działania aplikacji i napisz, co twoim zdaniem ***powinno*** się dziać. Następnie odpowiedz na pytania znajdujące się na dole strony.

1 Użytkownik wypełnia formularz rejestracji.

2 Formularz jest przesyłany do serwera WWW.

serwer WWW

3 Program po stronie serwera weryfikuje informacje podane przez użytkownika podczas rejestracji...

4 ...i zwraca do przeglądarki nową stronę WWW.

Serwer wyświetla ekran powitalny...

lub

...lub ponownie wyświetla stronę rejestracji z komunikatem o błędzie.

Wszystkie informacje wpisane przez użytkownika zostają utracone... Wszystkie pola są puste.

Jak sądzisz, jaki jest największy problem z witryną Mike'a? .....

.....

.....

Co byś zrobił, aby usprawnić witrynę Mike'a?.....

.....

.....

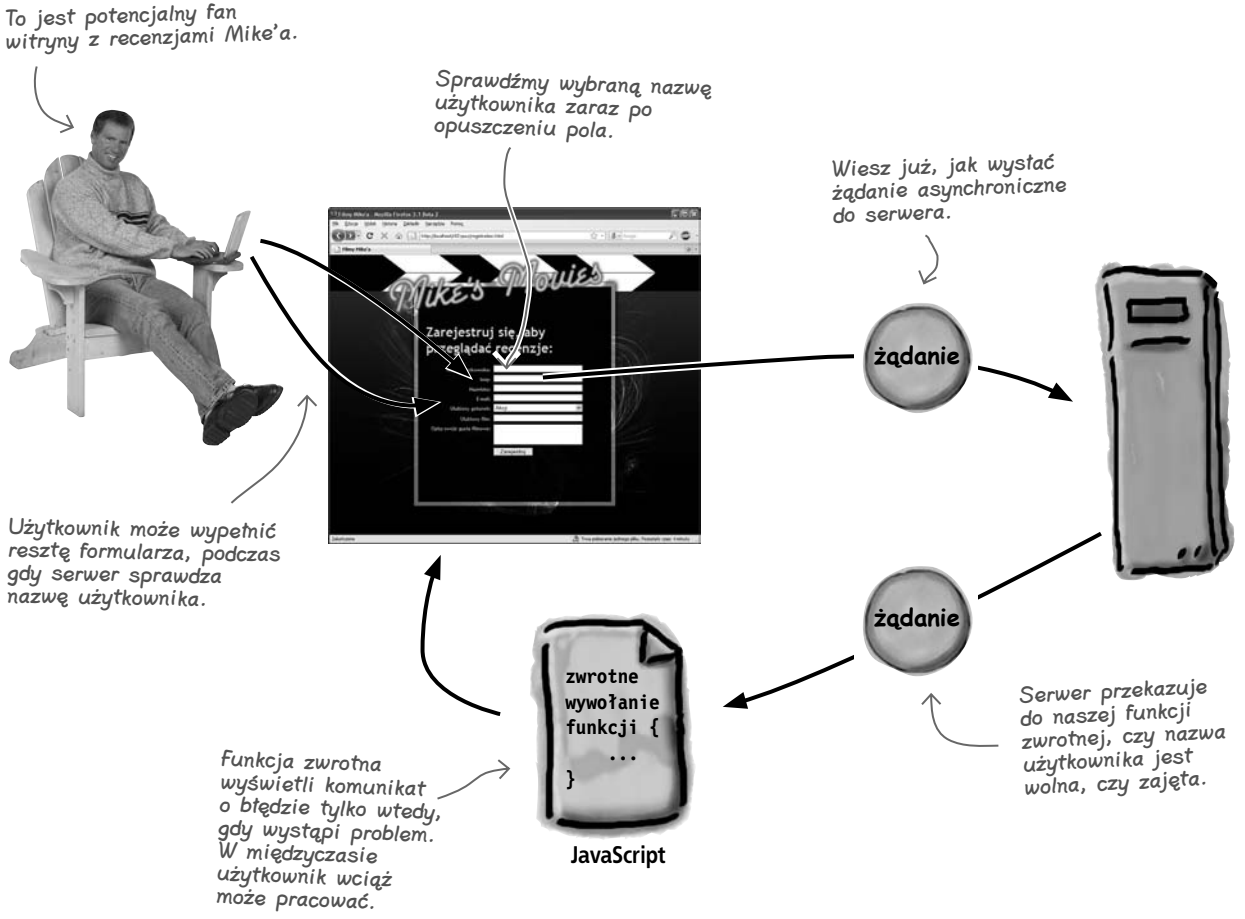
Użyjmy Ajaksa do ASYNCHRONICZNEGO przesyłania żądań rejestracji

Ajax jest idealnym narzędziem do rozwiązania problemu ze stroną Mike'a. W tej chwili największy problem polega na tym, że użytkownik musi czekać na pełne przeładowanie strony tylko po to, aby dowiedzieć się, że nazwa użytkownika jest już zajęta. Co więcej, jeżeli będzie chciał wybrać inną nazwę, musi ponownie wpisać pozostałe informacje. Obydwa problemy możemy rozwiązać, korzystając z metodologii Ajax.

Czy jako największy problem witryny Mike'a opisałeś coś podobnego?

Wciąż musimy porozumieć się z serwerem, aby dowiedzieć się, czy nazwa użytkownika jest już zajęta, ale po co czekać, aż użytkownik wypełni cały formularz? Gdy internauta skończy wpisywać nazwę, możemy wysłać **żądanie asynchroniczne** do serwera, sprawdzić nazwę i zakomunikować problemy bezpośrednio na stronie — ***bez konieczności*** jej przeładowania i ***bez*** utraty innych informacji wprowadzonych przez użytkownika.

Nic złego się nie stało, jeżeli nie pomyślałeś o wysłaniu żądania tuż po zakończeniu wpisywania nazwy przez użytkownika... Dostajesz jednak dodatkowy punkt, jeżeli o tym pomyślałeś!



A to wszystko po to, żeby jakiś miłośnik filmów nie musiał jeszcze raz wpisywać swojego nazwiska i adresu e-mail? Czy to nie lekka przesada?

Nie irytuj użytkowników... nigdy!

W internecie twoja konkurencja jest oddalona tylko o jedno kliknięcie. Jeżeli od razu nie powiadomisz użytkowników o problemie czy też zmusisz ich do ponownego wykonania czegokolwiek, prawdopodobnie stracisz ich na zawsze.

Witryna Mike'a może nie przynosi (jeszcze) zbyt dużych dochodów ani nawet nie wydaje ci się zbyt ważna... ale jest ważna dla jego fanów. Pewnego dnia jeden z użytkowników, którego nie zirytujecie, może mu zlecić pisanie recenzji filmowych dla „The New York Times" za sześciocyfrową sumę. Ale Mike nadal nie będzie wiedział, że jego witryna odstrasza internautów. Tutaj pomocne będą twoje umiejętności wykorzystania metodologii Ajax.

Ważna zasada projektowa w Ajaksie

Nie irytuj użytkowników.

Jeżeli wystąpi problem z aplikacją internetową, poinformuj o tym użytkowników możliwie szybko i zrozumiale. Nie powinieneś też nigdy wyrzucać tego, co użytkownicy już zrobili, nawet jeżeli stało się coś, czego oni (ani ty) się nie spodziewali.

Nie istnieją głupie pytania

P: Ta: zasada projektowa nie odnosi się tylko do Ajaksa, prawda?

O: Nie, stosuje się ona do wszystkich aplikacji internetowych, tak naprawdę do wszystkich rodzajów aplikacji. Jednak w przypadku aplikacji ajaksowych, a zwłaszcza żądań asynchronicznych, wiele rzeczy może pójść nie po naszej myśli. Częścią twojej pracy, jako dobrego programisty, jest ochrona użytkowników przed takimi problemami, a przynajmniej powiadomienie internautów o tym, co się dzieje.

Ćwiczenie

Czas popracować nad witryną Mike'a. Poniżej spisano 5 kroków, które należy wykonać, aby witryna działała, ale brakuje szczegółowych informacji na ich temat. Ponadto pomieszano kolejność czynności. Uporządkuj je i napisz jedno lub dwa zdania o tym, co powinno się wydarzyć na każdym z etapów.

Utwórz i skonfiguruj nowy obiekt żądania.

.....
.....
.....
.....

Ustaw procedury obsługi zdarzeń dla pól formularza.

.....
.....
.....
.....

Sprawdź wybraną nazwę użytkownika.

.....
.....
.....
.....

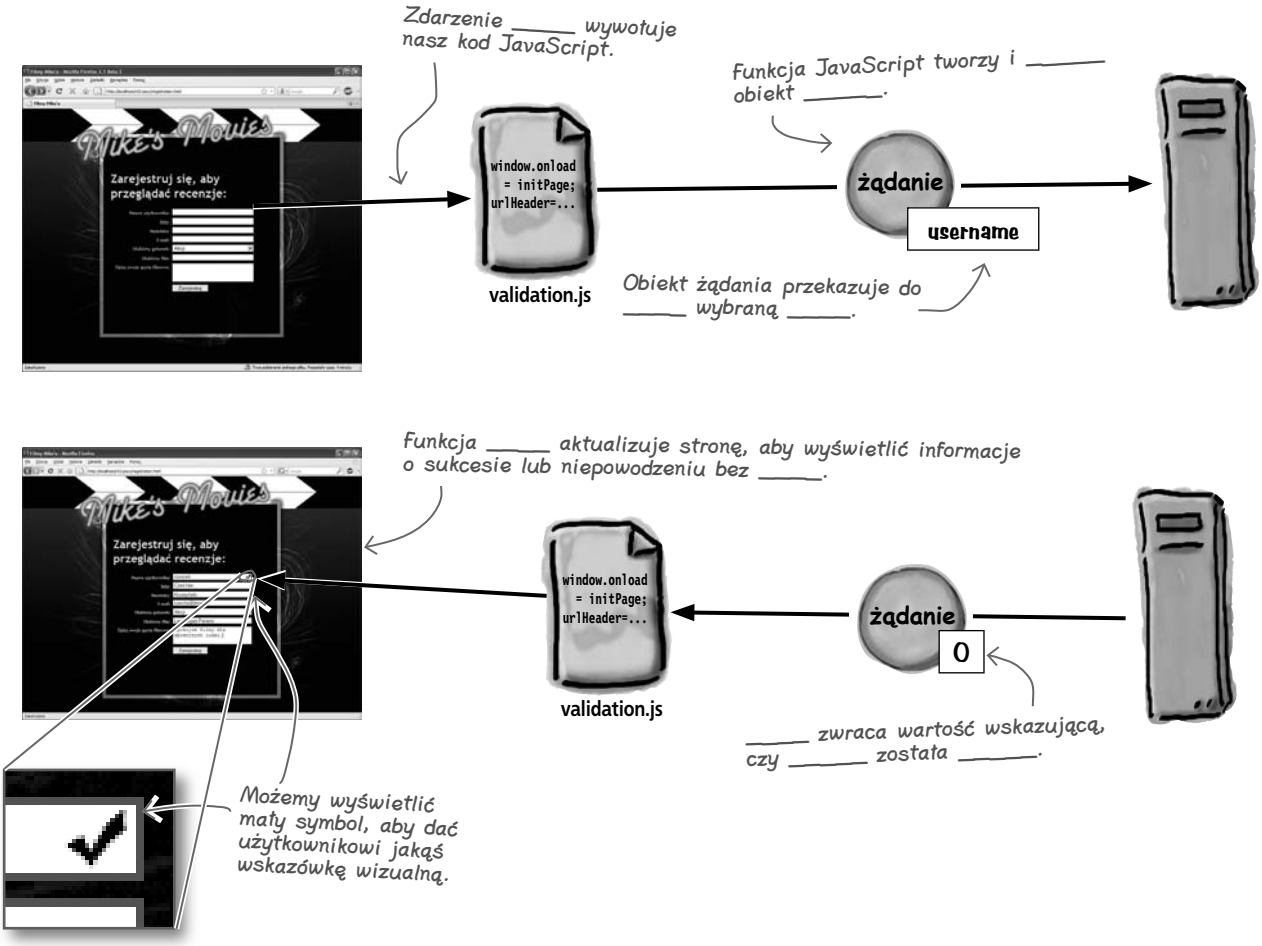
Zgłoś wszelkie problemy z wybraną nazwą użytkownika.

.....
.....
.....
.....

Zaktualizuj kod XHTML i CSS strony rejestracji.

.....
.....
.....
.....

Po uporządkowaniu etapów pracy nad witryną przyjrzyj się poniższym schematom przedstawiającym wybrane interakcje w ajaksowej wersji aplikacji Mike'a. Sprawdź, czy potrafisz uzupełnić puste pola.



Ćwiczenie: Rozwiązanie

Twoje zadanie polegało na uporządkowaniu etapów tworzenia ajaksowej wersji witryny z recenzjami i na dodaniu brakujących opisów poszczególnych czynności. Miałeś również uzupełnić brakujące słowa w schematach.

1 Zaktualizuj kod XHTML i CSS strony rejestracji.

Musimy dopisać elementy `<script>` do formularza rejestracji w celu odwołania się do kodu JavaScript, który napiszemy.

Kod dla poszczególnych etapów możesz napisać w dowolnej kolejności, ale w taki sposób będzie działać aplikacja i zgodnie z nim będziemy aktualizowali aplikację w tym rozdziale.

2 Ustaw procedury obsługi zdarzeń dla pól formularza.

Potrzebujemy trochę inicjalizującego kodu, który ustawia zdarzenie `onblur` dla pola z nazwą użytkownika. Dzięki temu, gdy użytkownik opuści pole, rozpoczniemy przetwarzanie żądania.

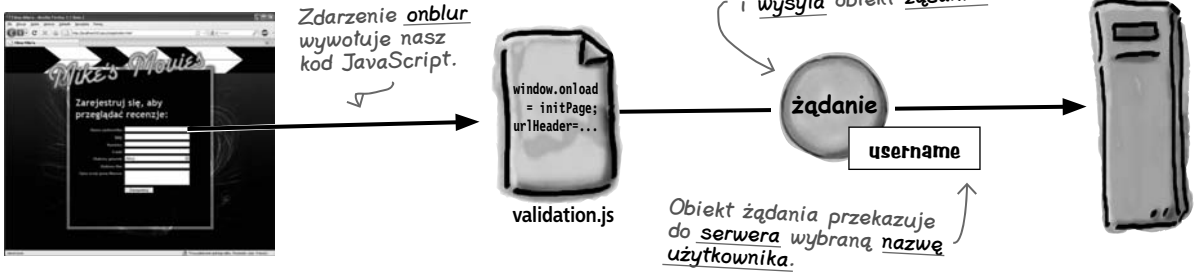
3 Utwórz i skonfiguruj nowy obiekt żądania.

W celu utworzenia żądania możemy wykorzystać funkcję `createRequest()` napisaną w rozdziale 1., następnie do łańcucha URL dodamy nazwę wybraną przez użytkownika, aby przesłać ją do serwera.

W poprzednim rozdziale prześlizgnęliśmy się przez tę funkcję, ale teraz omówimy ją szczegółowo.

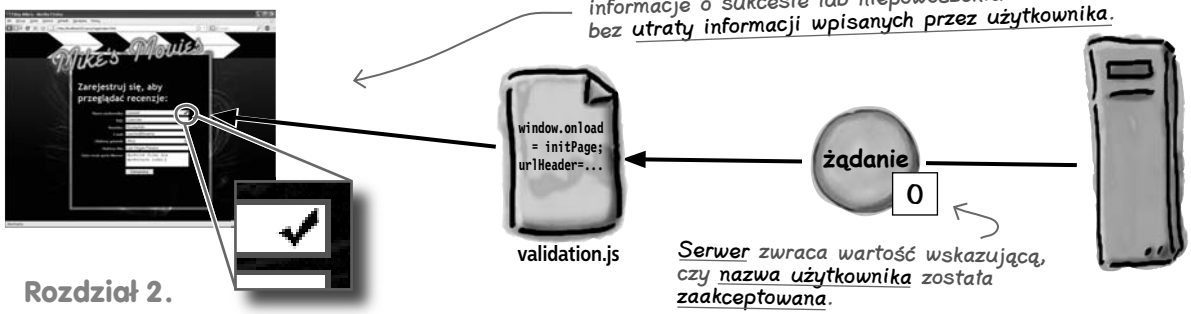
4 Sprawdź wybraną nazwę użytkownika.

Po utworzeniu obiektu żądania musimy go przesłać do serwera, aby sprawdzić, czy wybrana nazwa użytkownika jest wolna. Możemy to zrobić asynchronicznie, dzięki czemu internauta będzie mógł wypełniać pozostałe pola formularza podczas weryfikacji nazwy użytkownika przez serwer.



5 Zgłoś wszelkie problemy z wybraną nazwą użytkownika.

Po powrocie obiektu żądania funkcja zwrotna może zaktualizować stronę, aby wyświetlić wynik weryfikacji nazwy użytkownika.



Aktualizacja strony rejestracji

Podstawowa struktura strony rejestracji jest już gotowa, więc przejdźmy dalej i dodajmy znacznik `<script>` ładujący kod JavaScript, który napiszemy. Później będziemy mogli skonfigurować pole z nazwą użytkownika, aby wywoływało funkcję JavaScript w celu wykonania żądania do serwera.

Uwaga!

Używaj otwierającego i zamykającego znacznika <script>.

*Niektóre przeglądarki nie obsługują poprawnie samozamykającego się znacznika `<script>` — np. `<script/>`. Zawsze używaj **osobnego znacznika otwierającego i zamykającego dla znacznika** `<script>`.*

```
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Filmy Mike'a</title>
  <link rel="stylesheet" href="css/movies.css" />
  <script src="scripts/validation.js" type="text/javascript"></script>
</head>
```

Tak jak w poprzednim rozdziale plik validation.js będziemy uzupełniać stopniowo.

```
<html>
<script src="
... js" />
<img
src="siteLogo.
png" />
</html>
```

registration.html

Pobierz kod XHTML i CSS strony rejestracji.

Jeżeli jeszcze tego nie zrobiłeś, pobierz przykładowe pliki pod adresem *ftp://ftp.helion.pl/przyklady/hfajax.zip*. W folderze *r02* znajdziesz plik *registration.html*. Dopisz do niego pogrubiony wyżej znacznik `<script>`.

Wprowadź zmiany do pliku registration.html — strony rejestracji witryny Mike'a.

Nie istnieją głupie pytania

P: I w czym problem? Przecież to samo robiliśmy w poprzednim rozdziale w witrynie z rockandrollowymi pamiątkami, prawda?

O: Jak na razie tak. Ale większość aplikacji ajaksowych rozpoczyna się kilkoma znacznikami `<script>` i zewnętrznymi plikami z kodem JavaScript.

P: Ale i tak będziemy tylko przesyłać żądanie i uzyskiwać odpowiedź, nieprawdaż?

O: Pewnie. W zasadzie niemal wszystkie aplikacje ajaksowe można opisać tak prosto. Chociaż — jak się przekonasz, gdy dojdziemy do strony rejestracji — tak naprawdę możliwe są dwie interakcje z serwerem: jedna, którą budujemy do sprawdzenia nazwy użytkownika, oraz kliknięcie przycisku *Wyślij* po wypełnieniu formularza.

P: I co w tym takiego strasznego?

O: Jak sądzisz? Potrafisz dostrzec jakieś problemy z utworzeniem dwóch sposobów przesyłania dwóch różnych żądań do serwera?

Hej, ale zdaje się, że nie skończyliśmy jeszcze z kodem XHTML. Co z procedurą obsługi zdarzenia onblur dla pola z nazwą użytkownika? Chcemy uruchamiać jakiś kod po każdym wpisaniu nazwy użytkownika, prawda?

Oddzielaj zawartość strony od jej zachowania.

Moglibyśmy wywołać kod JavaScript bezpośrednio z kodu XHTML, np. umieszczając zdarzenie `onblur` w polu nazwy użytkownika. Ale byłoby to mieszanie zawartości strony z jej zachowaniem.

Kod XHTML opisuje ***zawartość*** i ***strukturę*** strony: dane, które są na stronie, takie jak nazwa użytkownika i opis witryny z recenzjami, oraz układ tych danych. Natomiast sposób, w jaki strona reaguje na czynności użytkownika, jest ***zachowaniem*** strony. Tym zwykle zajmuje się JavaScript. Z kolei kod CSS definiuje prezentację strony, czyli jej wygląd.

Oddzielenie zawartości, zachowania i prezentacji jest dobrym rozwiązaniem, nawet jeżeli samodzielnie tworzysz stosunkowo prostą stronę. A jeśli należysz do zespołu budującego złożone aplikacje, jest to jeden z najlepszych sposobów pozwalających uniknąć przypadkowego wmieszania się w czyjąś pracę.

Oddzielaj zawartość, zachowanie i prezentację strony.

Jeżeli tylko jest to możliwe, staraj się oddzielić od siebie zawartość strony (XHTML), jej zachowanie (JavaScript i procedury obsługi zdarzeń) oraz jej prezentację (CSS). Dzięki temu witryny będą elastyczniejsze i łatwiejsze w utrzymaniu i aktualizacji.

WYTĘŻ UMYSŁ

Jak sądzisz, dlaczego oddzielenie zawartości witryny od jej zachowania i prezentacji ułatwia wprowadzanie zmian?

Tę zasadę nazywa się czasem unobtrusive JavaScript (dyskretny JavaScript).

Gała prawda o procedurach obsługi zdarzeń

Wywiad tygodnia: Skąd naprawdę pochodzicie?

Head First: Cieszę się, że z nami jesteś. W tym tygodniu mamy bardzo ciekawe pytania.

Procedura obsługi: Naprawdę? Zawsze chętnie odpowiadam na pytania.

Head First: Jest jedno pytanie, które wszyscy zadają. Skąd dokładnie pochodzisz?

Procedura obsługi: Przybyłam z krainy ECMA, czyli...

Head First: Nie, nie, mam na myśli: skąd zostałaś ***wywołana***?

Procedura obsługi: Hm... Myślę, że ludzie z ECMA opowiedzieliby własną historię, ale skoro nalegasz... Zwykle jestem wywoływana z pola lub przycisku formularza XHTML. Czasem też z okien.

Head First: Czyli jesteś wywoływana ze stron XHTML?

Procedura obsługi: Tak, zazwyczaj.

Head First: Tak myślałem. To rozwiązuje spór. U nas usłyszeliście to jako pierwsi.

Procedura obsługi: Czekaj, czekaj. Jaki spór?

Head First: Dzwonił do nas JavaScript i zaklinał, że może cię wywołać. Mówił coś o zachowaniu wywołującym zachowanie... Kompletny bełkot.

Procedura obsługi: A, pewnie mówisz o przypisywaniu mnie programowo. Bardzo mądry ten JavaScript.

Head First: Programowo? Co to znaczy?

Procedura obsługi: Widzisz, w głębi duszy jestem po prostu właściwością...

Head First: Yyy, to ma związek z ECMA?

Procedura obsługi: ...którą można ustawić za pomocą JavaScriptu. Nie, słuchaj dalej. Wiesz, czym jest DOM, prawda?

Head First: Nie za bardzo... Tego dotyczy któryś z późniejszych rozdziałów.

Procedura obsługi: Trudno. Zrozum, każdy element na stronie WWW jest po prostu obiektem. Pola i przyciski są tylko obiektami mającymi określone właściwości.

Head First: Jasne, spotkaliśmy już kilka pól. Było całkiem miło. Ale Przycisk nigdy nie odpowiadał na wywołanie.

Procedura obsługi: Cóż, zdarzenia takie jak `onblur` i `onload` są związane ze mną za pośrednictwem tych właściwości.

Head First: Masz na myśli sytuację, gdy w XHTML-u piszemy np. `onblur="checkUsername()"` dla elementu przyjmującego dane?

Procedura obsługi: Dokładnie! To jest tylko właściwość pola. Ty jedynie wskazujesz przeglądarce, którą funkcję ma uruchomić oraz jak ma obsłużyć to zdarzenie.

Head First: Kompletnie się pogubiłem.

Procedura obsługi: Za pomocą JavaScriptu możesz przypisać wartość właściwości obiektu, prawda?

Head First: Mówisz, że procedur obsługi zdarzeń nie trzeba przypisywać w kodzie XHTML?

Procedura obsługi: Tak! Możesz to zrobić bezpośrednio w kodzie JavaScript... i oddzielić treść od zachowania.

Head First: To zaskakujące. Ale jak najpierw uruchomić JavaScript, aby przypisać procedurę obsługi zdarzenia?

Procedura obsługi: Jest pewien trik. Jakieś pomysły?

Head First: Nie jestem pewien. Spytajmy publiczność.

W jaki sposób możesz uruchomić inicjalizujący fragment JavaScriptu ***bez*** odwoływania się do funkcji na stronie XHTML?

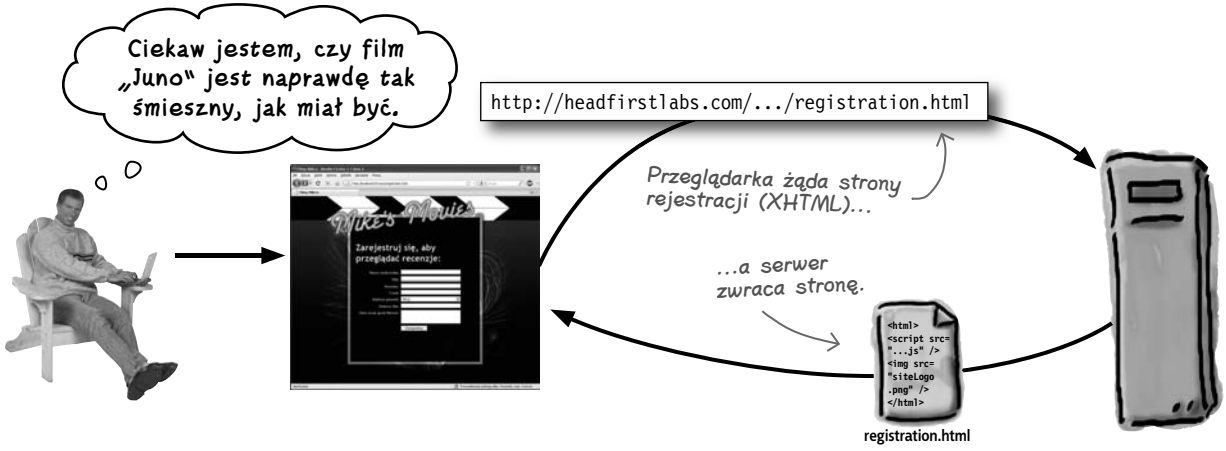
.....
.....
.....

PROGRAMOWA konfiguracja procedury obsługi zdarzenia window.onload

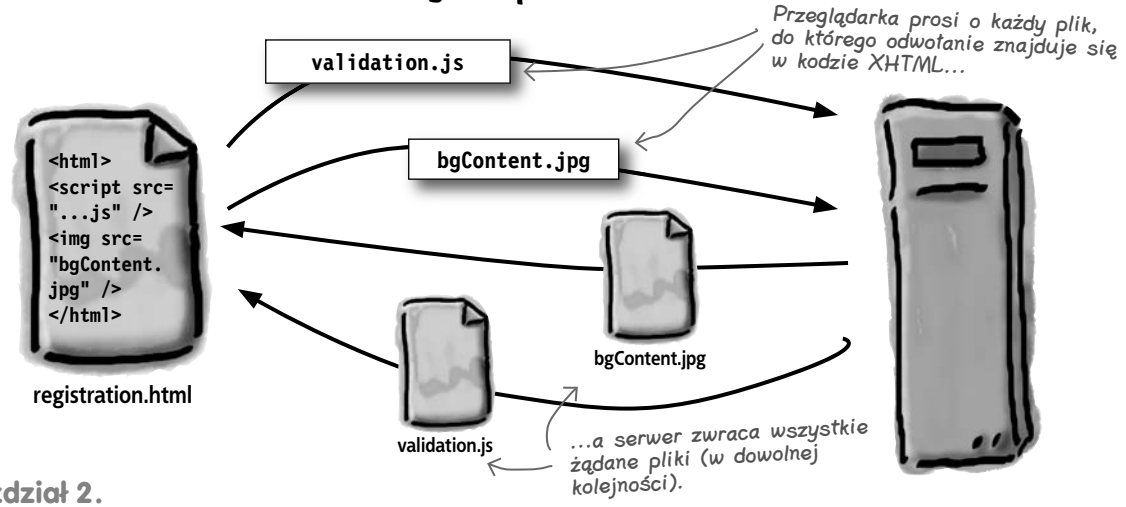
Chcemy, aby część kodu JavaScript była uruchamiana przy ładowaniu strony rejestracji, a to oznacza, że należy go dołączyć jako procedurę obsługi jednego z pierwszych zdarzeń — `window.onload`.

Możemy to zrobić programowo, ustawiając właściwość `onload` obiektu `window`. Jak? Przyjrzyjmy się, co dokładnie się dzieje podczas otwierania strony rejestracji przez użytkownika witryny Mike'a:

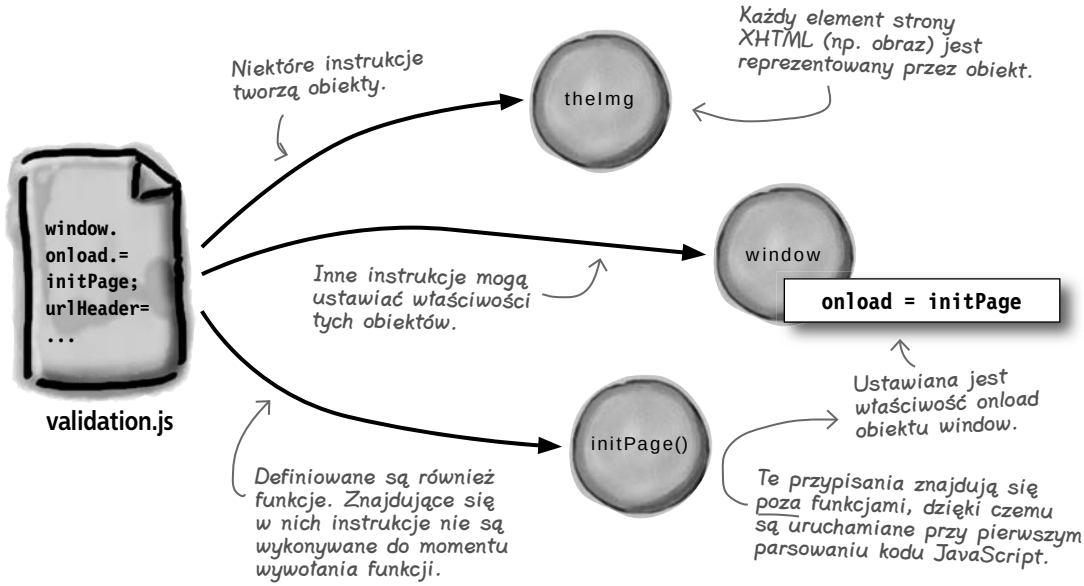
Na początku użytkownik otwiera stronę rejestracji w przeglądarce.



Następnie przeglądarka zaczyna przetwarzać stronę i prosi o kolejne pliki, do których napotka odwołania.

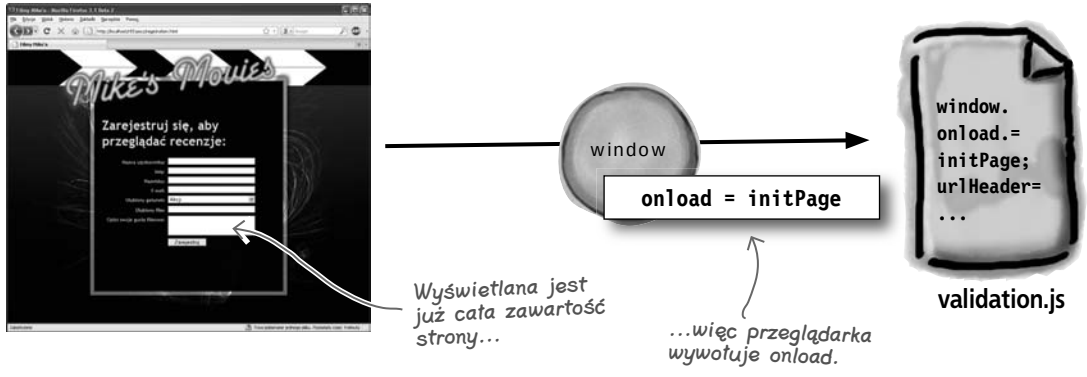


Jeżeli plik jest skryptem, przeglądarka parsuje go, tworzy obiekty i wykonuje wszystkie instrukcje nieznajdujące się w funkcjach.



W końcu, po załadowaniu i przetworzeniu wszystkich plików, przeglądarka wywołuje zdarzenie window.onload i funkcję zarejestrowaną do obsługi tego zdarzenia.

To wszystko dzieje się, jeszcze zanim można korzystać ze strony... a więc błyskawicznie!



Kod JavaScript znajdujący się poza funkcjami jest wykonywany podczas odczytu skryptu

validation.js

Chcemy, aby nasza procedura obsługi zdarzenia była uruchamiana wraz z wyświetleniem strony. Dlatego musimy przypisać funkcję do właściwości `onload` obiektu `window`.

Aby mieć pewność, że procedura ta zostanie przypisana wraz z załadowaniem strony, umieścimy kod przypisujący poza funkcjami w pliku *validation.js*. Dzięki temu przypisanie zostanie wykonane, zanim użytkownik będzie mógł cokolwiek zrobić na stronie.

Ten kod nie jest funkcją... Jest on wykonywany podczas odczytu skryptu przez przeglądarkę.

Ten wiersz nakazuje przeglądarce wywołanie funkcji initPage() tuż po załadowaniu elementów strony.

```
window.onload = initPage;

function initPage() {
  document.getElementById("username").onblur =
    checkUsername;
}

function checkUsername() {
  // uzyskaj obiekt żądania
  // i wyślij go do serwera
}

function showUsernameStatus() {
  // aktualizuj stronę, aby wskazać,
  // czy nazwa użytkownika została przyjęta
}
```

To nakazuje przeglądarce wywołać funkcję checkUsername(), gdy użytkownik opuści pole nazwy użytkownika w formularzu.

W rozdziałach 5. i 6. dokładnie omówimy metodę getElementById. Na razie musisz jedynie wiedzieć, że zwraca ona element XHTML o określonym identyfikatorze.

Oto kolejny przypadek programowego przypisania procedury obsługi zdarzenia.

To jest funkcja, która będzie tworzyła i wysyłała obiekt żądania. Napiszemy ją troszeczkę później.

To zaktualizuje stronę po otrzymaniu odpowiedzi z serwera.

Napisz początkową wersję *validation.js*.

W edytorze tekstu utwórz nowy plik, *validation.js*, i napisz w nim przedstawione wyżej deklaracje funkcji. Pamiętaj, aby przypisać funkcję `initPage()` do właściwości `onload` obiektu `window`!

Co kiedy się wydarza?

Na tym etapie wiele się dzieje. Omówmy to dokładnie, aby mieć pewność, że wszystko odbywa się dokładnie wtedy, gdy tego chcemy.

Na początku...

Gdy przeglądarka ładuje plik XHTML, znacznik `<script>` nakazuje jej załadować plik JavaScript. Cały kod, który znajduje się na zewnątrz funkcji, w tym pliku zostanie wykonany *natychmiast*, a interpreter JavaScriptu w przeglądarce utworzy funkcje, choć kod znajdujący się w nich nie zostanie jeszcze wykonany.

Następnie...

Instrukcja `window.onload` nie znajduje się w funkcji, więc zostanie uruchomiona tuż po załadowaniu pliku skryptowego *validation.js* przez przeglądarkę.

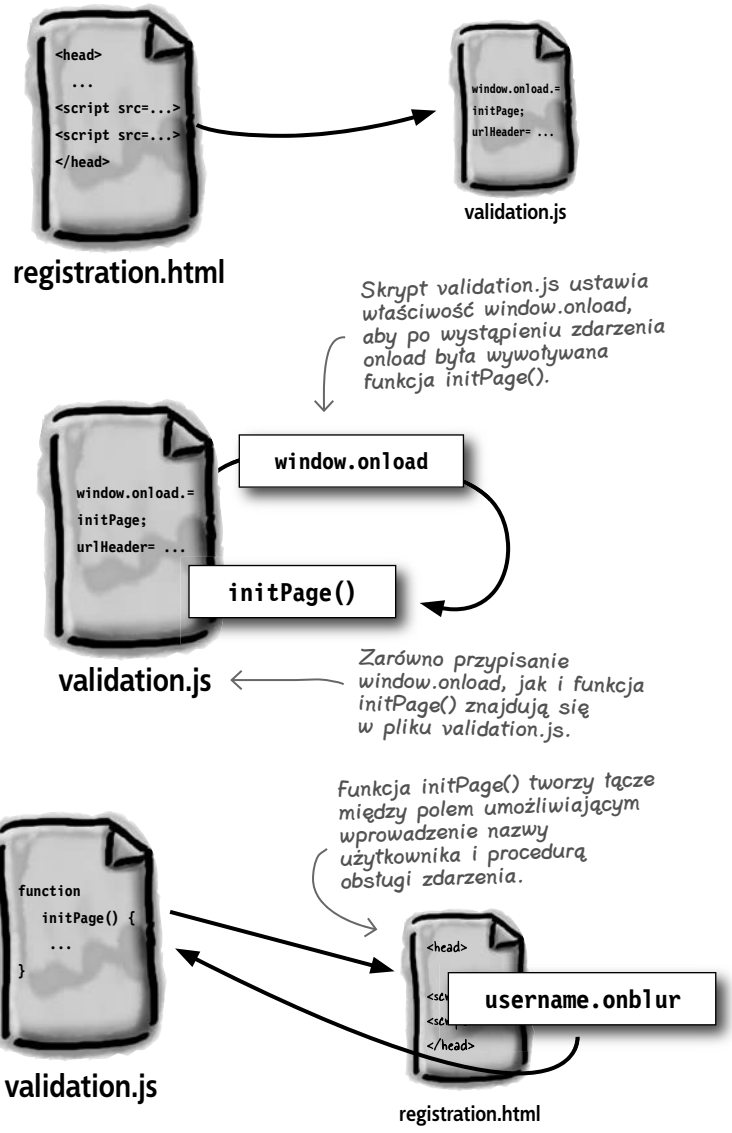
Instrukcja `window.onload` przypisuje funkcję `initPage()` jako procedurę obsługi zdarzenia. Funkcja ta zostanie wywołana, gdy tylko wszystkie pliki, do których znajdują się dowołania w kodzie XHTML, zostaną załadowane, ale zanim użytkownik będzie mógł skorzystać ze strony WWW.

Mimo że odbywa się to po kolei, WSZYSTKO dzieje się, zanim użytkownicy będą mogli wykonać jakiekolwiek czynności na stronie.

I w końcu...

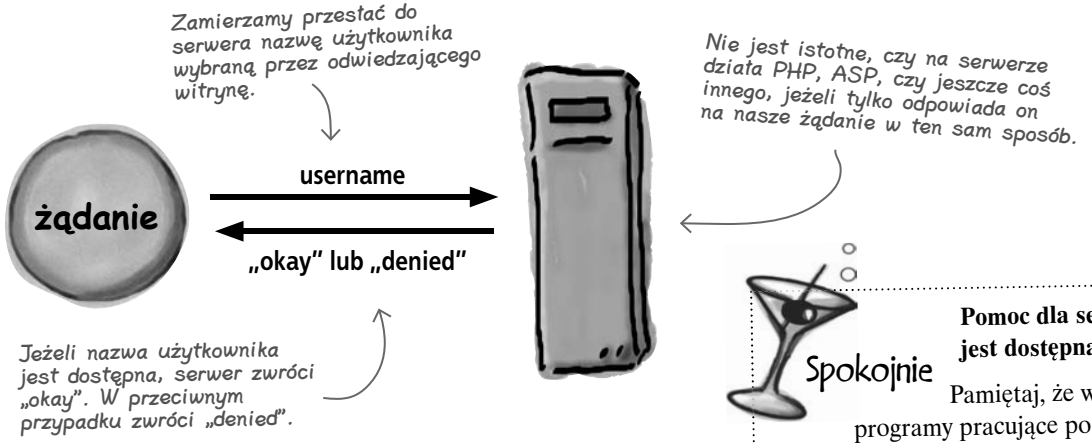
Funkcja `initPage()` zostaje uruchomiona. Wyszukuje ona pole o identyfikatorze `username`. Następnie przypisuje funkcję `checkUsername()` do zdarzenia `onblur` tego pola.

Efekt jest taki sam jak po wpisaniu `onblur="checkUsername()"` w kodzie XHTML. Jednak nasz sposób jest bardziej czytelny, ponieważ oddziela kod (funkcję JavaScript) od struktury i zawartości (XHTML).



A na serwerze...

Zanim będziemy mogli przetestować efekty naszej pracy nad stroną rejestracji, musimy sprawdzić serwer. Co serwer powinien dostać od nas w żądaniu? Czego możemy oczekiwać od serwera?



Pomoc dla serwera jest dostępna online.

Pamiętaj, że wybrane programy pracujące po stronie serwera są dostępne wśród przykładów do tej książki pod adresem *ftp://ftp.helion.pl/przyklady/hfajax.zip*.

Nie istnieją głupie pytania

P: Jeszcze raz, czym jest ten obiekt window?

O: Obiekt `window` reprezentuje okno przeglądarki użytkownika.

P: A więc window.onload jest wykonywane, gdy tylko użytkownik zażąda strony?

O: Nie tak szybko. Przeglądarka zaczyna od parsowania kodu XHTML i wszystkich plików, do których znajdują się odwołania w kodzie XHTML — np. CSS i JavaScript. Zatem kod znajdujący się w skryptach poza funkcjami jest wykonywany przed funkcją określoną w zdarzeniu `window.onload`.

P: Dlatego mogę w pliku ze skryptem przypisać funkcję do window.onload?

O: Dokładnie tak. Wszystkie skrypty, do których znajdują się odwołania w kodzie XHTML, są czytane *przed* wyzwoleniem zdarzenia `onload`. Następnie, po wyzwoleniu `onload`, użytkownicy faktycznie mogą zacząć korzystać ze strony.

P: Myślałem, że aby uruchomić kod JavaScript, trzeba go wywołać. Jak to jest?

O: Dobre pytanie. Przez wywołanie musisz uruchamiać kod JavaScript znajdujący się *w* funkcjach, a kod znajdujący się *poza* funkcją jest wykonywany przy parsowaniu danego wiersza kodu przez przeglądarkę.

P: Ale powinniśmy to chyba przetestować, aby upewnić się, że wszystko działa poprawnie?

O: Fakt. Zawsze przetestuj projekt aplikacji, zanim uznasz, że działa.

P: W tym kodzie nic się nie dzieje. Jak mam go przetestować?

O: To jest kolejne dobre pytanie. Jeżeli masz kod, który nie daje widocznych rezultatów, możesz poratować się wierną funkcją `alert()`.

Jazda próbna

Zabierz nową stronę rejestracji na przejażdżkę.

Upewnij się, że wprowadziłeś wszystkie zmiany do plików *registration.html* i *validation.js*, a następnie załaduj stronę rejestracji w przeglądarce. Niewiele się zmieniło, prawda?

Funkcja `initPage()` nie przynosi żadnych widocznych rezultatów, a funkcja `checkUsername()` jeszcze w ogóle nic nie robi... Musimy jednak sprawdzić, czy funkcja `checkUsername()` jest wywoływana, gdy użytkownik wpisze nazwę użytkownika i przejdzie do wypełniania innego pola.

To trochę toporne, ale dodajmy kilka instrukcji `alert()`, aby upewnić się, że funkcje, które napisaliśmy, są faktycznie wywoływane.

```
window.onload = initPage;

function initPage() {
  document.getElementById("username").onblur = checkUsername;
  alert("Wewnątrz funkcji initPage().");
}

function checkUsername() {
  // uzyskaj obiekt żądania i wyślij go do serwera
  alert("Wewnątrz funkcji checkUsername().");
}

function showUsernameStatus() {
  // aktualizuj stronę, aby wskazać, czy nazwa użytkownika została przyjęta
}
```

validation.js

Sprawdź, czy wszystko działa.

Funkcja alert() daje nam wizualną informację zwrotną. Wiemy, że funkcja initPage() jest wywoływana...

...podobnie jak checkUsername(), która jest wywoływana po wpisaniu nazwy użytkownika i opuszczeniu tego pola formularza przez internautę.

Niektóre części projektów ajaksowych będą takie same... zawsze

Metodę `window.onload` i funkcję `initPage()` wykorzystaliśmy już dwukrotnie — raz w rockandrollowym sklepie Roba, a drugi raz na stronie rejestracji Mike'a. W kolejce czeka tworzenie obiektu żądania działającego na stronie rejestracji tak samo jak w witrynie Roberta.

Dobrzy projektanci aplikacji szukają podobieństw i starają się ponownie wykorzystywać kod z innych projektów i aplikacji.

Wiele elementów w aplikacjach ajaksowych się powtarza. Część twojej pracy polega na tworzeniu kodu w taki sam sposób, abyś nie musiał nieustannie pisać identycznych fragmentów. Spójrzmy, jak w witrynie Mike'a wygląda tworzenie i używanie obiektu żądania:

Większość tych szczegółów zmienia się w różnych aplikacjach zależnie od ich funkcjonalności, układu, stylu itd.

1 Strona ładuje się oraz wykonuje zadania i inicjalizację specyficzne dla aplikacji.

2 Wywoływany jest kod JavaScript specyficzny dla aplikacji, który musi wysłać żądanie do serwera.

3 Tworzony jest nowy obiekt żądania.

```
request = createRequest();
```

validation.js

żądanie

```
createRequest() {...}
```

Ta część — czyli tworzenie obiektu żądania — jest taka sama we wszystkich aplikacjach w metodologii Ajax.

żądanie

username

"okay"

"denied"

Serwer zwraca „okay" lub „denied".

4 Obiekt żądania jest konfigurowany z użyciem danych aplikacji i wysyłany do serwera.

5 Serwer zwraca odpowiedź do przeglądarki, używając obiektu żądania.

serwer WWW

Funkcja createRequest() jest zawsze taka sama

Niemal w każdej aplikacji ajaksowej potrzebujemy funkcji tworzącej obiekt żądania... i taką już mamy. To funkcja `createRequest()`, którą widzieliśmy w rozdziale 1. Przyjrzyjmy się dokładniej, jak tworzy ona obiekt żądania w różnych sytuacjach we wszystkich typach przeglądarek.

IE5 na komputerach Macintosh wciąż nie działa, nawet z tym kodem niezależnym od rodzaju przeglądarki.

Aby ta funkcja nadawała się do wielokrotnego użytku, nie może ona zależeć od konkretnej przeglądarki ani od szczegółów konkretnej aplikacji.

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

  return request;
}
```

To jest obsługiwane przez wiele przeglądarek, a tym samym przez wielu użytkowników.

Pamiętaj, że musimy próbować, dopóki nie znajdziemy składni zrozumiałej dla każdej przeglądarki.

Ten wiersz zwraca żądanie do kodu wywołującego.

Nie istnieją głupie pytania

P: A więc jak naprawdę nazywa się ten obiekt żądania?

O: Większość osób nazywa go `XMLHttpRequest`, ale to jest naprawdę trudne do wymówienia. Oprócz tego w niektórych przeglądarkach nazywa się go inaczej, np. `XMLHTTP`. Zdecydowanie łatwiej jest nazywać go po prostu obiektem żądania, unikając tym samym przywiązania do konkretnej przeglądarki. I tak większość osób myśli o tym w ten sposób — jak o żądaniu.

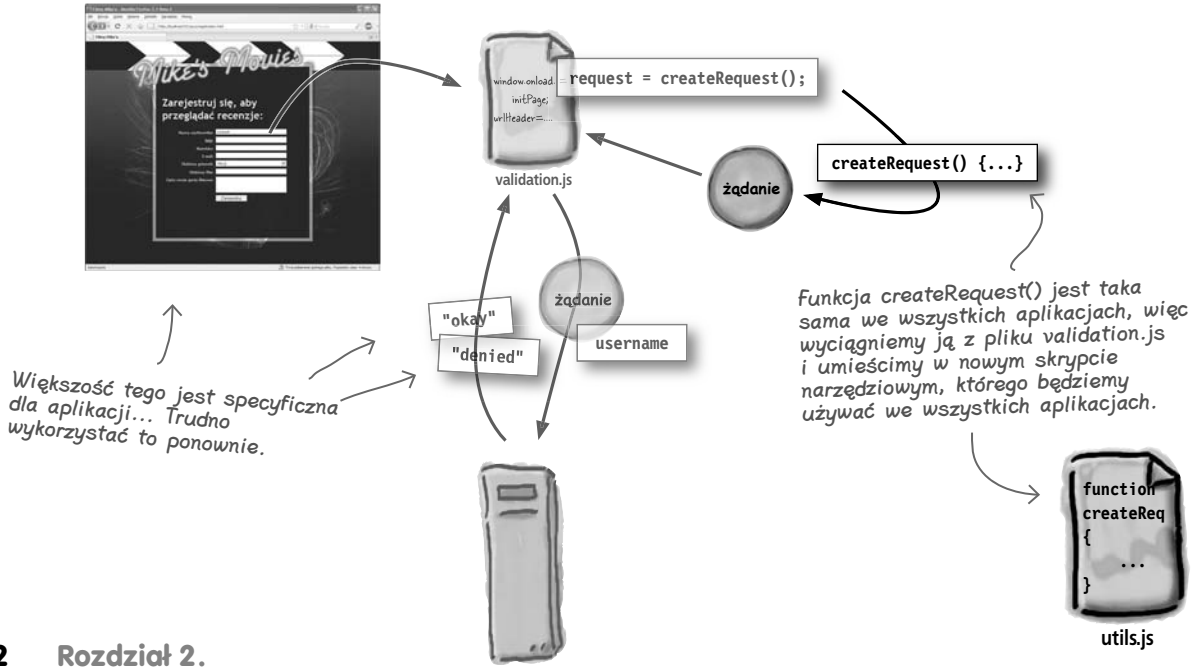
Poczekaj... Jeżeli jest to dokładnie taki sam kod, jak wcześniej, dlaczego go po prostu nie skopiujemy i nie wkleimy?

Kopiowanie i wklejanie nie jest dobrym pomysłem na wielokrotne wykorzystanie kodu.

Funkcja `createRequest()` dla witryny Mike'a jest dokładnie taka sama, jak funkcja `createRequest()` w witrynie Roba, nad którą pracowaliśmy w rozdziale 1. Skopiowanie kodu, który napisaliśmy w rozdziale 1., i wklejenie go do pliku *validation.js* jest jednak złym pomysłem. Jeżeli się okaże, że trzeba wprowadzić jakąś zmianę, będziesz musiał to zrobić w dwóch miejscach. A jak sądzisz, co będzie się działo podczas pracy nad dziesięcioma lub dwudziestoma aplikacjami?

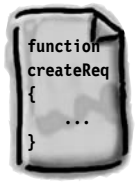
Jeżeli trafisz na kod wspólny dla kilku aplikacji, wydziel go ze skryptów specyficznych dla aplikacji i umieść w skrypcie narzędziowym wielokrotnego użytku. A więc w przypadku `createRequest()` możemy usunąć tę funkcję z pliku *validation.js* witryny z recenzjami i utworzyć nowy skrypt. Nazwiemy go *utils.js* i będziemy umieszczać w nim kod wspólny dla różnych aplikacji.

Później każda nowa aplikacja, którą stworzymy, będzie mogła odwoływać się zarówno do *utils.js*, jak i skryptu zawierającego kod JavaScript specyficzny dla aplikacji.



☐ Utwórz nowy plik o nazwie *utils.js*. Napisz w nim funkcję `createRequest()`, której kod znajdziesz w poprzednim rozdziale lub na stronie 91, i zapisz zmiany.

Wprowadź każdą z tych zmian do własnego kodu, odhaczając kolejne punkty.



utils.js

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

  return request;
}
```

Zazwyczaj warto umieścić kod narzędziowy jako pierwszy, a kod specyficzny dla aplikacji jako drugi. Jeśli wyrobisz sobie takie nawyki, kod, który napiszesz, będzie postrzegany jako spójny i uporządkowany.

☐ Otwórz plik *registration.html* i dopisz znacznik `<script>` z odwołaniem do nowego skryptu *utils.js*.

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Filmy Mike'a</title>
    <link href="movies.css" rel="stylesheet" type="text/css" />
    <script src="scripts/utils.js" type="text/javascript"></script>
    <script src="scripts/validation.js" type="text/javascript"></script>
</head>
```

registration.html

☐ Jeżeli dopisałeś już funkcję `createRequest()` do pliku *validation.js*, usuń ją. Funkcja ta powinna znajdować się teraz tylko w skrypcie *utils.js*.

Nie istnieją głupie pytania

P: Dlaczego umieściliśmy odwołanie do utils.js przed odwołaniem do validation.js?

O: Często kod specyficzny dla aplikacji będzie wywoływał kod narzędziowy. Dlatego przeglądarka powinna zinterpretować kod narzędziowy, zanim zacznie parsować kod, który mógłby wywołać kod narzędziowy. Poza tym jest to ładny sposób porządkowania kodu — narzędzia jako pierwsze, kod specyficzny dla aplikacji jako drugi.

P: Ale wciąż nie rozumiem, jak działa funkcja createRequest()...

O: No tak. Określiliśmy `createRequest()` jako funkcję wielorazowego użytku i przenieśliśmy ją do skryptu narzędziowego. To dobrze, ale wciąż musimy dowiedzieć się, co ten kod tak naprawdę robi.

Oddziel to, co jest identyczne w różnych aplikacjach, i przekształć ten kod w zestaw funkcji wielorazowego użytku.

Twórz obiekt żądania... w wielu przeglądarkach

utils.js

Czas zająć się JavaScriptem i dowiedzieć się, co się dzieje. Omówmy dokładnie, krok po kroku, co robi każdy fragment funkcji `createRequest()`.

1 Tworzenie funkcji.

Rozpoczynamy od utworzenia funkcji, którą może wywołać każdy inny kod, gdy będzie potrzebował obiektu żądania.

Ta funkcja może być wywołana z dowolnego miejsca w aplikacji.

```
function createRequest() {
  // utwórz zmienną o nazwie „request”
}
```

Niezależnie od użytej składni nowy egzemplarz obiektu żądania będzie zachowywał się tak samo.

To oddziela kod wywołujący od fragmentów związanych z kompatybilnością przeglądarek.

2 Próba utworzenia XMLHttpRequest — przeglądarki inne niż microsoftowe.

Definiujemy zmienną o nazwie `request` i próbujemy przypisać jej nowy egzemplarz obiektu `XMLHttpRequest`. To działa niemal we wszystkich przeglądarkach, oprócz Microsoft Internet Explorer.

XMLHttpRequest działa w Safari, Firefoksie, Mozilli, Operze i większości przeglądarek poza tworzonymi przez Microsoft.

```
function createRequest(){
  try {
    request = new XMLHttpRequest();
  } catch (tryMS) {
    // nie zadziałało, więc spróbujemy czegoś innego
  }
}
```

3 Próba utworzenia ActiveXObject dla przeglądarek Microsoftu.

W bloku **`catch`** próbujemy utworzyć obiekt żądania, korzystając ze składni specyficznej dla przeglądarek Microsoftu. Istnieją jednak dwie *różne* wersje obiektowych bibliotek Microsoftu, więc musimy wypróbować obydwie.

Większość wersji IE obsługuje tę składnię...

...ale niektóre wymagają innej biblioteki.

Cały ten kod... wstawiamy tutaj.

```
try {
  request = new ActiveXObject("Msxml2.XMLHTTP");
} catch (otherMS) {
  try {
    request = new ActiveXObject("Microsoft.XMLHTTP");
  } catch (failed) {
    // to również nie zadziałało — po prostu nie możemy uzyskać obiektu żądania
  }
}
```

4 Jeżeli nic nie zadziała, zwróć null.

Wypróbowaliśmy trzy różne sposoby uzyskania obiektu żądania. Jeżeli parser dotrze do tego bloku, będzie to oznaczało, że żaden z nich nie zadziałał. Dlatego deklarujemy `request` jako `null` i pozwalamy kodowi wywołującemu zdecydować, co z tym zrobić. Pamiętaj, że `null` jest obiektem, który otrzymujesz, gdy nie masz egzemplarza obiektu.

To umieścimy w ostatnim bloku catch.

```
request = null;
```

Zwrócenie null oznacza, że pateczkę przejmuje kod wywołujący, który może zdecydować, jak zgłosić błąd.

5 Składamy wszystko i zwracamy obiekt żądania.

Pozostaje nam tylko zwrócić zmienną `request`. Jeżeli wszystko się udało, `request` wskazuje na obiekt żądania. W przeciwnym wypadku wskazuje na `null`:

```
function createRequest() {
  try {
    request = new XMLHttpRequest();
  } catch (tryMS) {
    try {
      request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (otherMS) {
      try {
        request = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (failed) {
        request = null;
      }
    }
  }
  return request;
}
```

Dla przeglądarek innych niż microsoftowe.

Dla fanów Internet Explorera.

Mimo wszystko zawsze coś zostanie zwrócone, choćby tylko wartość null.

Moglibyśmy wygenerować tutaj komunikat o błędzie, ale pozwolimy kodowi wywołującemu zdecydować, co zrobić, gdy nie uzyska obiektu żądania.

CELNE SPOSTRZEŻENIA

- W różnych przeglądarkach do uzyskania obiektu żądania używana jest odmienna składnia. Kod powinien uwzględniać każdy rodzaj składni, aby aplikacja działała w różnych przeglądarkach.
- Niezależnie od składni użytej do uzyskania egzemplarza obiektu żądania, sam obiekt zachowuje się zawsze w ten sam sposób.
- Zwrócenie null, gdy uzyskanie egzemplarza obiektu żądania nie było możliwe, pozwala kodowi wywołującemu zadecydować co dalej. Jest to rozwiązanie elastyczniejsze niż wygenerowanie komunikatu o błędzie.

Projekt aplikacji ajaksowej obejmuje zarówno stronę WWW, jak i program po stronie serwera

Chociaż formularz dla strony rejestracji już istnieje, musimy nad nim popracować, aby uzyskać nazwę użytkownika, a później wyświetlić komunikat o błędzie, jeśli nazwa jest zajęta.

Nawet jeżeli ktoś inny musi zajmować się pisaniem kodu wykonywanego po stronie serwera, musimy wiedzieć, ***co*** przesłać do tego kodu i ***jak*** przesłać te informacje.

Spójrz na etapy sprawdzania poprawności nazwy użytkownika. Większość z nich wiąże się z interakcją z formularzem WWW lub z programem po stronie serwera.

Tym zajmuje się funkcja createRequest().

Pamiętaj, że funkcja createRequest() nie obsługuje błędów, więc musimy to zrobić samodzielnie.

1. **Spróbuj uzyskać obiekt żądania.**
2. **Wyświetl komunikat, jeżeli przeglądarka nie może utworzyć żądania.**
3. **Pobierz nazwę użytkownika wpisaną do formularza.**

To współpracuje z formularzem.

4. **Upewnij się, że nazwa użytkownika nie zawiera znaków, które stanowiłyby problem w żądaniu HTTP.**

Tutaj przygotowujemy się do wysłania nazwy użytkownika do serwera.

5. **Dołącz nazwę użytkownika do URL serwera.**
6. **Poinformuj przeglądarkę, którą funkcję powinna wywołać, gdy serwer odpowie na żądanie.**

To jest funkcja zwrotna, którą wkrótce napiszemy.

7. **Poinformuj przeglądarkę, jak przesłać żądanie do serwera.**
8. **Wyślij obiekt żądania.**

Mamy wolne aż do powrotu obiektu żądania, który przeglądarka przekaże do funkcji zwrotnej.

Tutaj mamy jeszcze więcej interakcji z serwerem.

Dobry projekt ajaksowy bazuje głównie na interakcji. Musisz wchodzić w interakcje z użytkownikami za pośrednictwem strony WWW i z logiką biznesową za pośrednictwem programów pracujących po stronie serwera.

Magnesiki z kodem

Większość kodu funkcji `checkUsername()` porozklejano na lodówce. Czy potrafisz poskładać go z powrotem? Nawiasy klamrowe spadły na podłogę i są zbyt małe, aby je znaleźć i podnieść. Możesz ich więc dopisać tyle, ile będziesz potrzebował.

```
function checkUsername() {

}
```

```
window.onload = initPage;
```

validation.js

```
request.send(null);
```

```
alert("Nie można utworzyć żądania");
```

```
var theName = document.getElementById("username").value;
```

```
if (request == null)
```

```
} else {
```

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

```
request = createRequest();
```

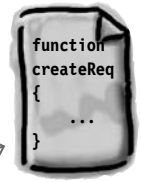
```
request.onreadystatechange = showUsernameStatus;
```

```
var username = escape(theName);
```

```
var url = "checkName.php?username=" + username;
```

Magnesiki z kodem: Rozwiązanie

Większość kodu funkcji `checkUsername()` porozklejano na lodówce. Twoje zadanie polegało na poukładaniu kodu w działającą funkcję.



utils.js

```
function checkUsername() {
  request = createRequest();
  if (request == null)
    alert("Nie można utworzyć żądania")
  } else {
    var theName = document.getElementById("username").value;
    var username = escape(theName);
    var url = "checkName.php?username=" + username;
    request.onreadystatechange = showUsernameStatus;
    request.open("GET", url, true);
    request.send(null);
  }
}
```

Zaczynamy od wywołania naszej funkcji narzędziowej z pliku utils.js w celu uzyskania obiektu żądania.

Jeżeli zostaje zwrócone null, oznacza to, że działanie funkcji zakończyło się niepowodzeniem...

...więc informujemy o tym użytkownika.

Metoda getElementById wyszukuje pole formularza o identyfikatorze username.

value to wartość wpisana przez użytkownika

Funkcja escape() w JavaScripcie oczyszcza dane wpisane przez użytkownika na wypadek, gdyby w tekście znalazły się spacje czy też znaki zapytania.

Nazwę użytkownika dołączamy do adresu URL.

To jest funkcja zwrotna, do której przeglądarka prześle obiekt żądania, gdy serwer odpowie na żądanie.

To informuje przeglądarkę, jak wysłać żądanie. Używamy metody GET i przesyłamy je do URL określonego przez zmienną url. Parametr true oznacza, że ma to być żądanie asynchroniczne, czyli użytkownik będzie mógł wypełniać formularz podczas sprawdzania nazwy użytkownika przez serwer.

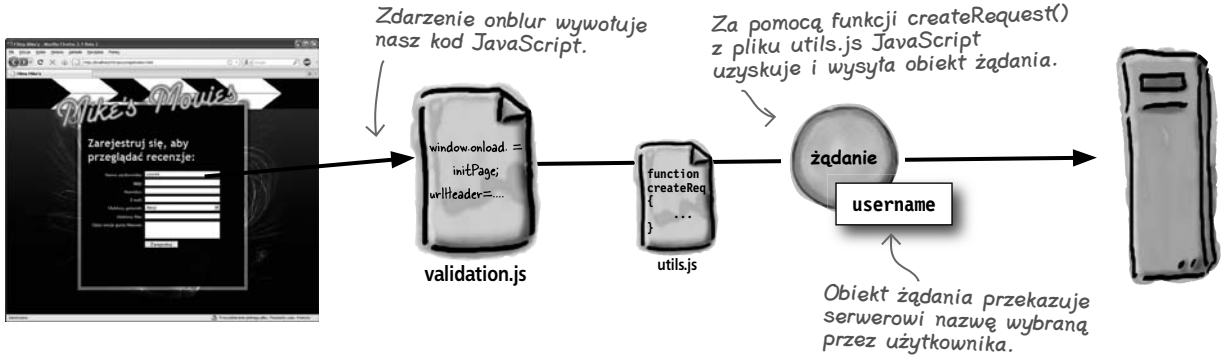
Metoda send() wysyła obiekt żądania do serwera. Parametr null oznacza, że nie przesyłamy żadnych dodatkowych danych.

Cały ten kod znajduje się w pliku validation.js.

validation.js

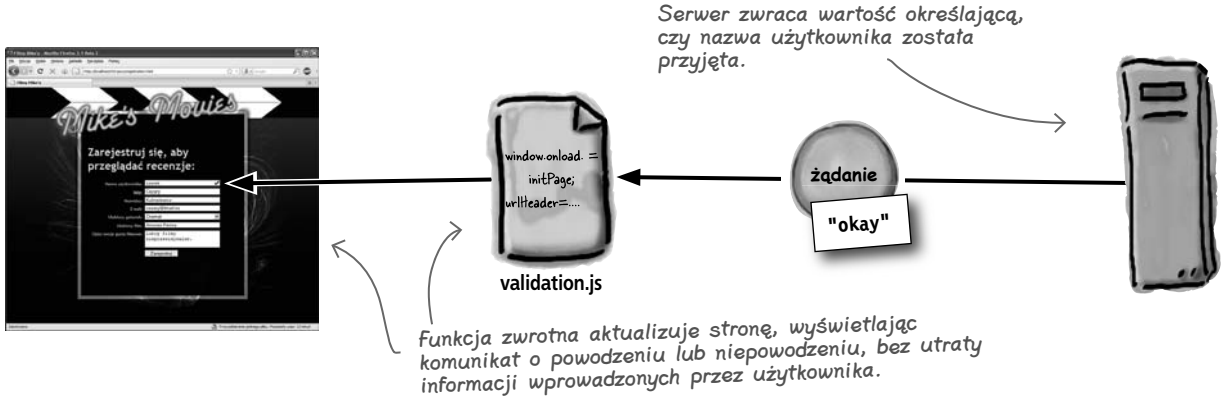
Co już zrobiliśmy...

Przygotowaliśmy się do tego, aby żądanie zostało przesłane do serwera, kiedy tylko internauta wpisze nową nazwę użytkownika.



Co jeszcze musimy zrobić...

Jesteśmy prawie gotowi do uzyskania odpowiedzi na żądanie:



Nie istnieją głupie pytania

P: Co właściwie robi getElementById()?

O: Metodę `getElementById()` omówimy *bardzo* dokładnie w rozdziałach 5. i 6., gdy będziemy zajmować się DOM-em. Na razie musisz wiedzieć, że zwraca ona obiekt JavaScript reprezentujący element XHTML na stronie.

P: A czym jest value?

O: Funkcja `getElementById()` zwraca obiekt JavaScript reprezentujący element XHTML. Tak jak wszystkie obiekty, obiekt zwracany przez tę funkcję posiada metody i właściwości. Właściwość `value` zawiera tekst znajdujący się w elemencie — w tym przypadku to, co użytkownik wpisał w polu nazwy użytkownika.

Jazda próbna

Zanim przejdziemy dalej, upewnijmy się, że wszystko działa...

Nasz kod JavaScript wciąż nie aktualizuje strony w żaden sposób. Korzystając z kilku dodatkowych funkcji `alert()`, możemy jednak sprawdzić, czy funkcja `checkUsername()` działa zgodnie z naszymi oczekiwaniami.

W edytorze tekstu otwórz plik *validate.js* i dopisz do funkcji `checkUsername()` kod przedstawiony poniżej. Jest to ten sam kod, nad którym pracowałeś w ćwiczeniu z magnesikami, ale znajduje się w nim kilka alertów pozwalających śledzić, co robi przeglądarka.

Po dodaniu kodu zapisz plik i załaduj stronę w przeglądarce. Wpisz cokolwiek do pola przeznaczonego na nazwę użytkownika. Powinny zostać wyświetlone wszystkie komunikaty.

validation.js

```
function checkUsername() {
      request = createRequest();
      if (request == null)
        alert("Nie można utworzyć żądania")
      else
      {
        alert("Mam obiekt żądania");
        var theName = document.getElementById("username").value;
        alert("Pierwotna wartość nazwy: " + theName);
        var username = escape(theName);
        alert("Wartość nazwy po czyszczeniu: " + username);
        var url= "checkName.php?username=" + username;
        alert("URL: " + url);

        request.onreadystatechange = showUsernameStatus;
        request.open("GET", url, true);
        request.send(null);
      }
    }
```

Te alerty są czymś w rodzaju komunikatów o stanie albo informacjami z debuggera... Pozwalają nam się dowiedzieć, co się dzieje „za kulisami".

Powinny zostać wyświetlone komunikaty o utworzeniu, skonfigurowaniu i wysłaniu żądania.

Aplikacje asynchroniczne zachowują się inaczej niż tradycyjne aplikacje internetowe, więc trzeba brać to pod uwagę podczas debugowania.

Aplikacje asynchroniczne nie zmuszają do oczekiwania na odpowiedź serwera, więc nie otrzymujesz z serwera całej strony. Większość interakcji między stroną WWW i serwerem jest zupełnie niewidoczna dla użytkownika. Jeżeli przeglądarka napotka problem podczas wykonywania kodu JavaScript, prawdopodobnie po prostu się zatrzyma i nie będziesz miał pojęcia, co się stało.

Komunikaty są dobrym sposobem śledzenia problemów, o których nie informuje przeglądarka. Pokazują one to, co widzi *przeglądarka*. Pozwalają dowiedzieć się, co dzieje się w tle, gdy użytkownicy beztrosko wpisują swoje dane.

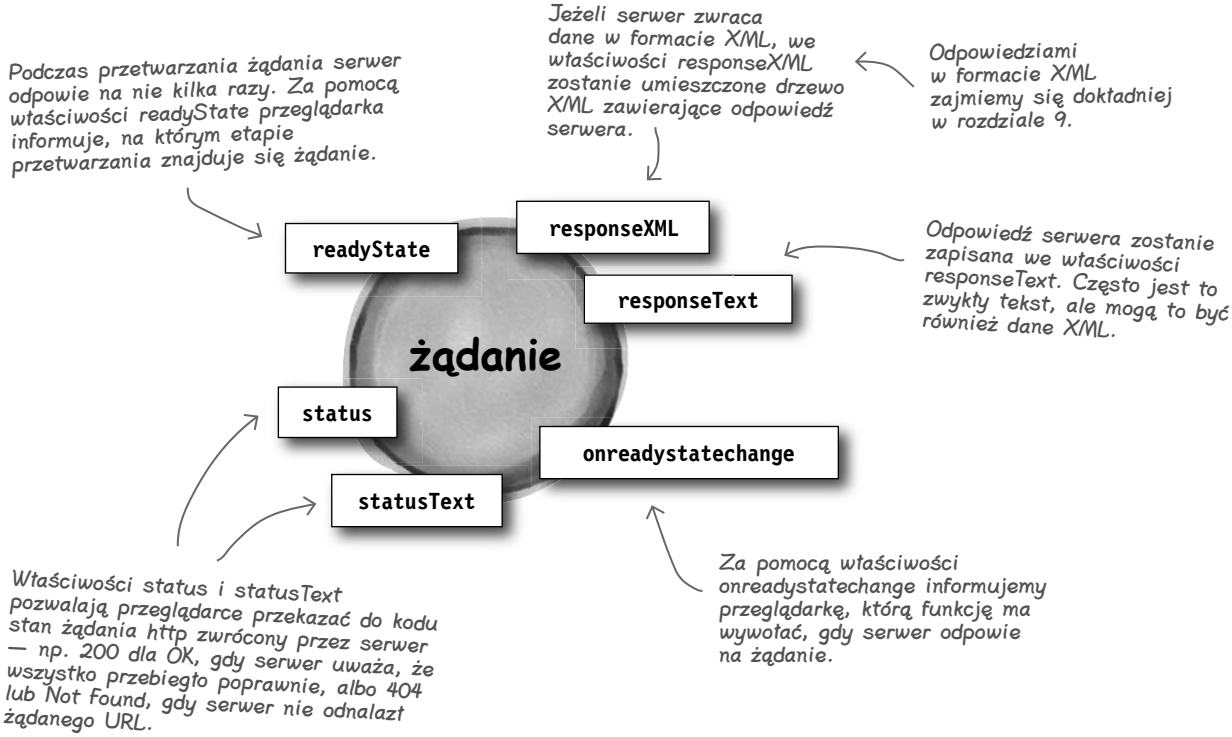
Po prześledzeniu rozwiązania wszystkich problemów usuniesz wszystkie instrukcje alert().

W aplikacjach asynchronicznych nie możesz liczyć na to, że serwer poinformuje cię o wystąpieniu problemu. To TY musisz określić, że wystąpił problem, i zareagować na niego w odpowiedni sposób.

Obiekt żądania łączy twój kod z przeglądarką

Musimy jeszcze napisać kod, który przeglądarka wywoła po uzyskaniu odpowiedzi na żądanie. Tutaj przyda nam się obiekt żądania. Dzięki niemu można poinformować przeglądarkę o tym, co ma zrobić. Za jego pomocą nakażemy więc przeglądarce wykonanie żądania do serwera i uzyskanie wyników.

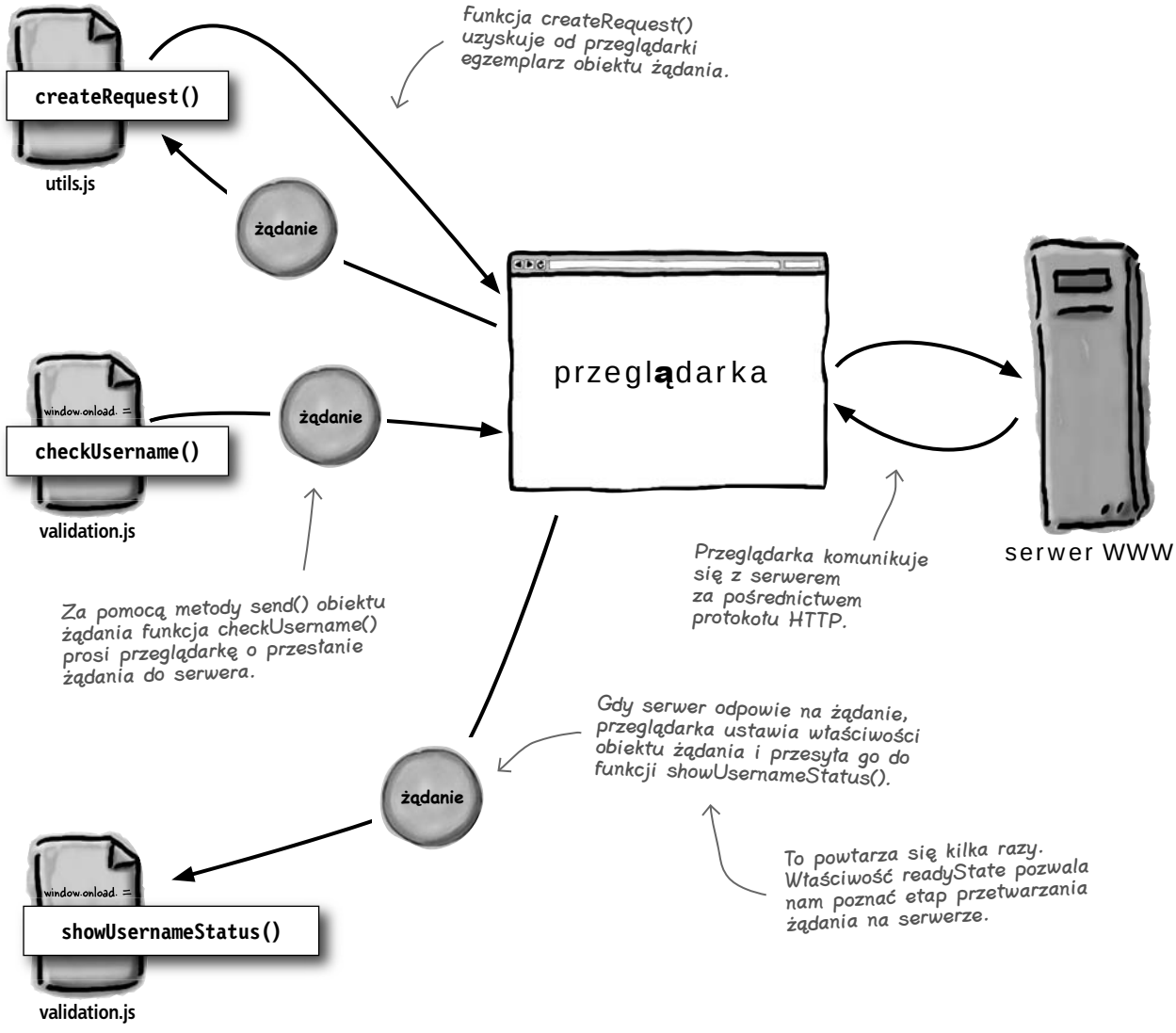
Ale jak to wszystko się dzieje? Pamiętaj, że ***obiekt żądania jest zwykłym obiektem w JavaScripcie***. Dlatego może on mieć właściwości, a te z kolei mogą mieć wartości. Istnieje kilka bardzo przydatnych właściwości. Jak sądzisz, które będą nam potrzebne w funkcji zwrotnej?



Przeglądarka daje kodowi dostęp do odpowiedzi serwera za pośrednictwem właściwości obiektu żądania.

Porozumiewasz się z przeglądarką, a nie z serwerem

Choć łatwo mówi się o kodzie „wysyłającym obiekt żądania do serwera”, nie jest to stwierdzenie do końca poprawne. W rzeczywistości ty porozumiewasz się z przeglądarką, a dopiero przeglądarka porozumiewa się z serwerem. **Przeglądarka wysyła obiekt żądania do serwera** i **przeglądarka tłumaczy odpowiedź serwera**, zanim przekaże dane z odpowiedzi do strony WWW.



Zbliżenie: Stany gotowości

Za pomocą właściwości `readyState` obiektu żądania przeglądarka informuje funkcję zwrotną, na jakim etapie przetwarzania znajduje się żądanie. Sprawdźmy, co to oznacza.

To jest stan gotowości żądania zapisany we właściwości readyState.

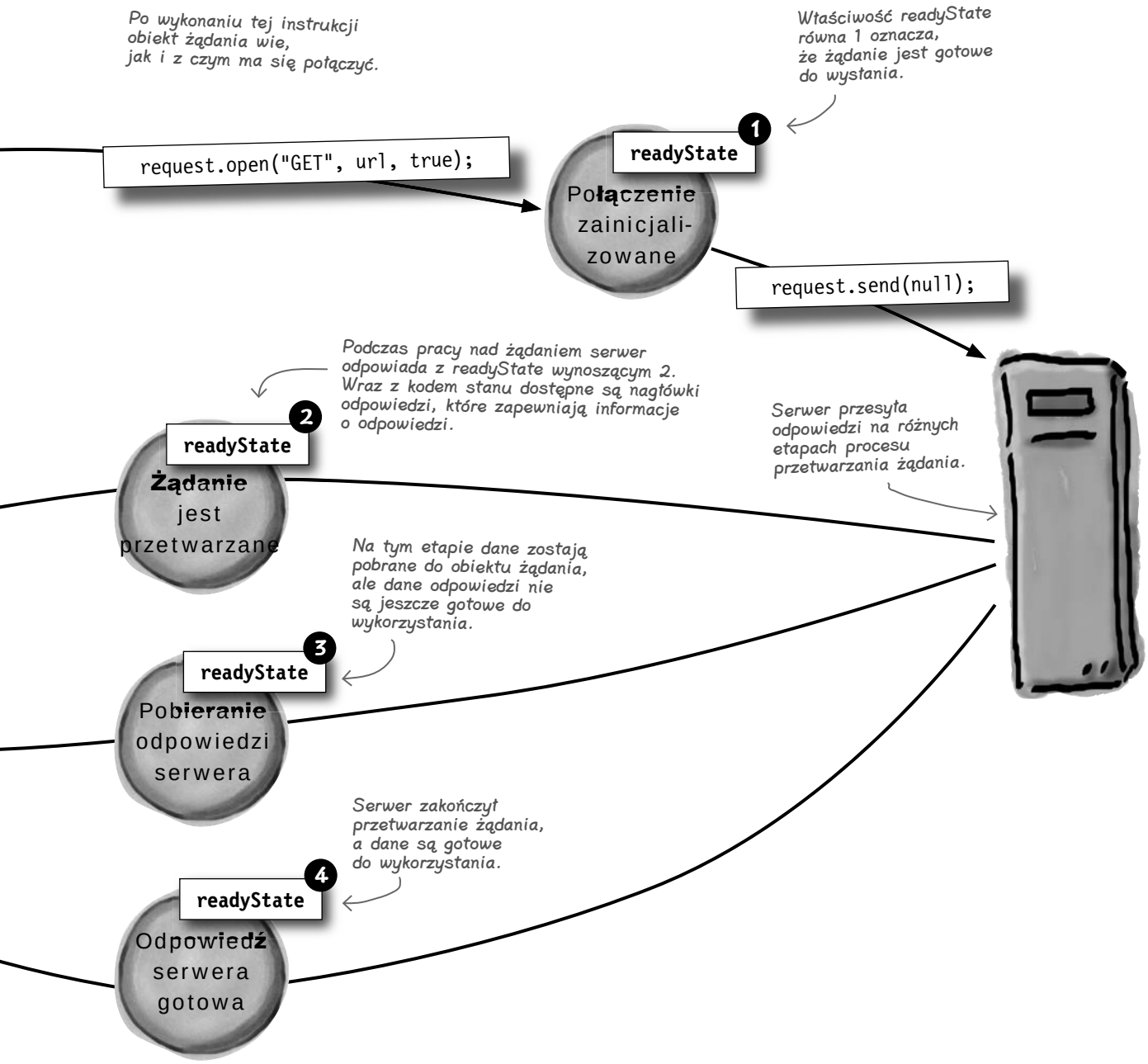
readyState 0

Połączenie niezainicjalizowane

Gdy użytkownik opuszcza pole nazwy użytkownika, funkcja createRequest() tworzy obiekt żądania.

```
request = createRequest();
```

Gdy właściwość readyState obiektu żądania ma wartość 4, funkcja showUsernameStatus() używa odpowiedzi serwera do zaktualizowania strony.


Po wykonaniu tej instrukcji obiekt żądania wie, jak i z czym ma się połączyć.
Właściwość readyState równa 1 oznacza, że żądanie jest gotowe do wysłania.
request.open("GET", url, true);
1
readyState
Połączenie zainicjalizowane
request.send(null);
Podczas pracy nad żądaniem serwer odpowiada z readyState wynoszącym 2. Wraz z kodem stanu dostępne są nagłówki odpowiedzi, które zapewniają informacje o odpowiedzi.
2
readyState
Żądanie jest przetwarzane
Serwer przesyła odpowiedzi na różnych etapach procesu przetwarzania żądania.
Na tym etapie dane zostają pobrane do obiektu żądania, ale dane odpowiedzi nie są jeszcze gotowe do wykorzystania.
3
readyState
Pobieranie odpowiedzi serwera
Serwer zakończył przetwarzanie żądania, a dane są gotowe do wykorzystania.
4
readyState
Odpowiedź serwera gotowa

Przeglądarka wywołuje kod zwrotnie i przekazuje do niego odpowiedź serwera

Zawsze, gdy zmienia się wartość właściwości `readyState` obiektu żądania, przeglądarka musi coś zrobić. I co robi? Wywołuje funkcję przypisaną do właściwości `onreadystatechange` obiektu żądania:

Gdy zmienia się stan gotowości odpowiedzi, czyli za każdym razem, gdy serwer przesyła do przeglądarki informację o przetwarzanym żądaniu, przeglądarka wywołuje tę funkcję.

```
function checkUsername() {
       request = createRequest();
       ...
       request.onreadystatechange = showUsernameStatus;
       ...
}
```

validation.js

Funkcja zwrotna musi się upewnić, że odpowiedź jest już gotowa do użytku; może sprawdzić właściwość `readyState` i status serwera, a następnie, na podstawie odpowiedzi serwera, podjąć jakieś działania:

To jest nazwa funkcji, której użyliśmy we właściwości onreadystatechange. Jeżeli nazwy nie będą zgodne, funkcja nie zostanie wywołana.

```
function showUsernameStatus() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      if (request.responseText == "okay") {
        // jeżeli jest to „okay”, nie trzeba wyświetlać komunikatu o błędzie
      }
      else {
        // jeżeli wystąpił problem, informujemy użytkownika
        alert("Przykro nam, ale nazwa użytkownika jest zajęta.");
      }
    }
  }
}
```

Instrukcja if daje pewność, że dalsza część kodu nie zostanie wykonana, jeżeli readyState ma wartość różną od 4, która oznacza, iż serwer zakończył pracę nad żądaniem.

Jeśli nie wystąpiły żadne błędy, serwer wysyła status „200”.

W responseText jest przechowywana wartość tekstowa zwracana przez serwer. Jeżeli jest nią „okay”, oznacza to, że nazwa użytkownika jest wolna.

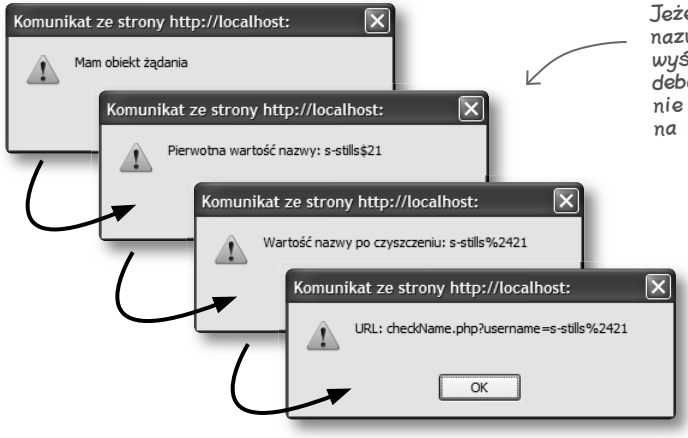
Ten kod również zamieszczamy w validation.js.

validation.js

Jazda próbna

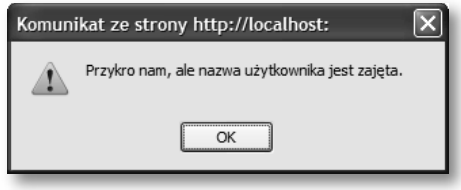
Dopisz funkcję `showUsernameStatus()` do pliku *validation.js* i załaduj stronę w przeglądarce.

Spróbuj wpisać jakąkolwiek nazwę użytkownika, oprócz `stefan` i `natalia`. Przeglądarka powinna wyświetlić wszystkie alerty, które napisaliśmy w celu przetestowania funkcji `initPage()` i `checkUsername()`.



Jeżeli wpiszesz poprawną nazwę użytkownika, zostaną wyświetlone alerty z kodu debugującego, ale żaden nie będzie wskazywał na wystąpienie błędu.

Teraz spróbuj wpisać `stefan` lub `natalia` jako nazwę użytkownika. Powinien wyświetlić się komunikat o błędzie umieszczony w funkcji `showUsernameStatus()`.



Ten komunikat powinien zostać wyświetlony po wpisaniu „stefan" lub „natalia" do pola nazwy użytkownika i opuszczeniu tego pola. Ktoś już się zarejestrował, używając tej nazwy użytkownika.

Upewniwszy się, że wszystko działa poprawnie, usuń z funkcji `checkUsername()` dodane wcześniej alerty. Powinieneś jedynie pozostawić komunikat o niepowodzeniu przy tworzeniu żądania (funkcja `checkUsername()`) oraz o niedostępności nazwy użytkownika (funkcja `showUsernameStatus()`).

Gdy masz pewność, że interakcja między kodem i serwerem przebiega poprawnie, nie potrzebujesz już debugujących instrukcji `alert()`.

Pokaż Mike'owi ajaksową stronę rejestracji

Wszystko działa. Jeżeli jednak przekażesz kod Mike'owi, a on umieści go w witrynie, wciąż będą występowały pewne problemy:

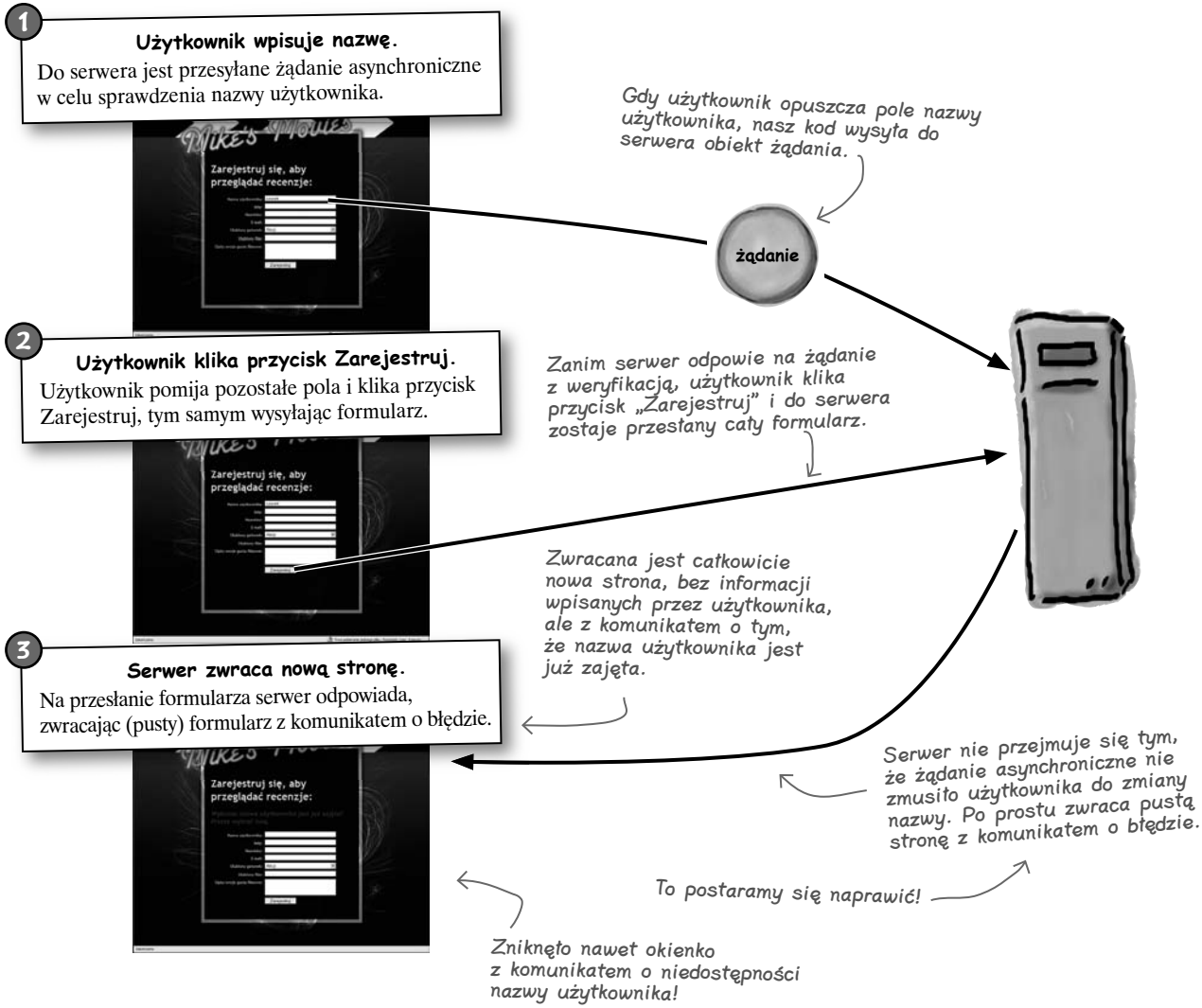
Co się stało? Czy cały wysiłek włożony w udoskonalenie strony rejestracji poszedł na marne? Był niepotrzebny?

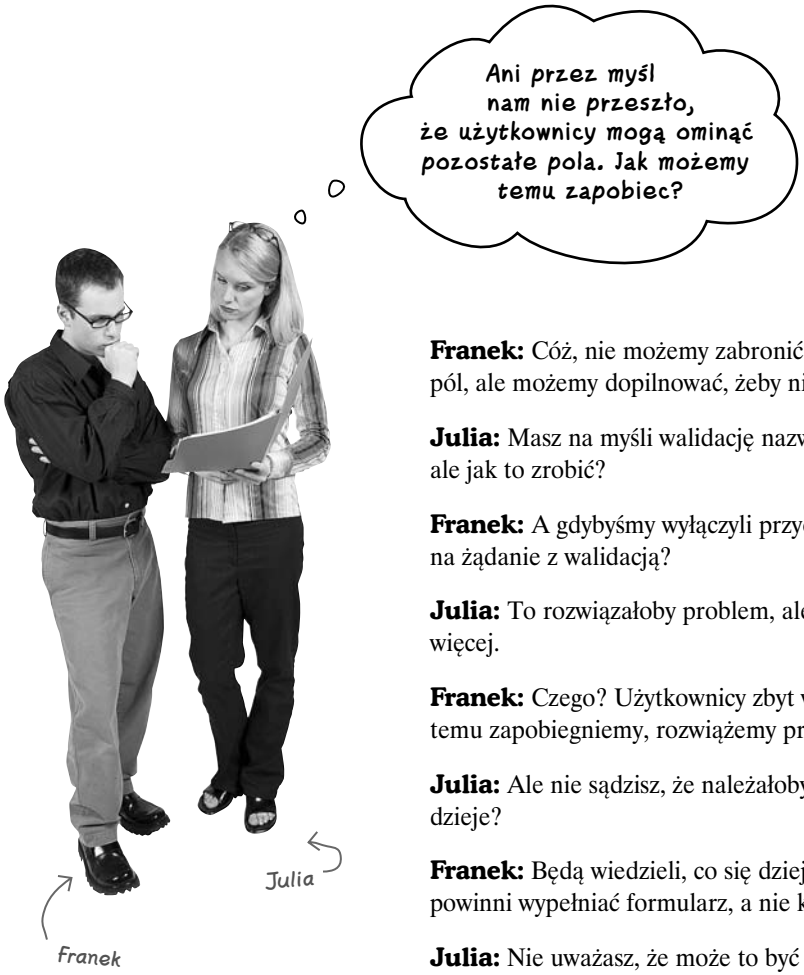
Co TY o tym sądzisz?

Teraz formularz może przesyłać żądania do serwera na dwa sposoby

Załóżmy, że użytkownik robi dokładnie to, czego oczekujemy: wpisuje nazwę użytkownika i — podczas gdy żądanie asynchroniczne jest przesyłane do serwera i obsługiwane przez przeglądarkę (wywołującą funkcję zwrotną) — wypełnia pozostałe pola formularza. Wszystko działa świetnie, tak jak się tego spodziewamy.

A teraz załóżmy, że użytkownik tak bardzo chce przeczytać recenzję filmu *Iron Man*, iż wpisuje nazwę użytkownika, pomija pozostałe części formularza i klika przycisk *Zarejestruj*. Co się wtedy stanie?





Nie możesz zakładać, że użytkownicy zrobią wszystko dokładnie tak jak ty... Przygotuj plan na KAŻDĄ sytuację!

Franek: Cóż, nie możemy zabronić użytkownikom omijania poszczególnych pól, ale możemy dopilnować, żeby nie wyprzedzali naszego żądania.

Julia: Masz na myśli walidację nazwy użytkownika? Tak, to brzmi świetnie, ale jak to zrobić?

Franek: A gdybyśmy wyłączyli przycisk *Zarejestruj* do czasu, aż serwer odpowie na żądanie z walidacją?

Julia: To rozwiązałoby problem, ale wydaje mi się, że potrzebujemy czegoś więcej.

Franek: Czego? Użytkownicy zbyt wcześnie wysyłają formularz, więc jeżeli temu zapobiegniemy, rozwiążemy problem.

Julia: Ale nie sądzisz, że należałoby jakoś poinformować użytkowników, co się dzieje?

Franek: Będą wiedzieli, co się dzieje, gdy włączymy przycisk. Do tego czasu powinni wypełniać formularz, a nie klikać przycisk *Zarejestruj*.

Julia: Nie uważasz, że może to być niejasne? Jeżeli użytkownik skończy wypełniać formularz lub w ogóle nie będzie chciał tego zrobić, utknie na stronie, nawet nie wiedząc dlaczego.

Franek: Musimy go w takim razie poinformować, że aplikacja coś robi. Może wyświetlimy komunikat?

Julia: Kolejny alert? To będzie równie denerwujące. Może obrazek? Moglibyśmy wyświetlić obrazek w trakcie przesyłania żądania do serwera...

Franek: ...a po zweryfikowaniu nazwy użytkownika wyświetlić inny.

Julia: A jeżeli użylibyśmy obrazków wskazujących, czy nazwa została zaakceptowana, czy też nie, moglibyśmy pozbyć się alertu, gdy wystąpi z nią problem.

Franek: Idealnie! Wizualna informacja zwrotna ***bez*** wyskakujących okienek, które wszystkich irytują. Podoba mi się!

☐ **Wyświetlaj grafikę „w trakcie" podczas przesyłania żądania z weryfikacją.**

Przesyłając do serwera żądanie w celu zweryfikowania nazwy użytkownika, obok pola z nazwą użytkownika wyświetlimy grafikę informującą użytkowników, co się dzieje. Dzięki temu, wypełniając formularz, będą oni dokładnie zorientowani w sytuacji.

Prawdopodobnie metoda getElementById zaczyna już wyglądać znajomo. Daje ona dostęp do elementu na stronie XHTML.

Wprowadź te zmiany do swojego kodu i zaznacz pola.

```
function checkUsername() {
  document.getElementById("status").src = "images/inProcess.png";
  request = createRequest();
  …
}
```

Poprzez wyświetlenie tego obrazka poinformujesz użytkownika, że coś się dzieje.

validation.js

☐ **Po weryfikacji wyświetl komunikat o stanie.**

Po powrocie obiektu żądania możemy wyświetlić inny obrazek, korzystając z funkcji zwrotnej. Jeżeli nazwa użytkownika zostanie zaakceptowana, wyświetlimy odpowiedni obrazek. W przeciwnym wypadku wyświetlimy ikonę oznaczającą błąd.

Możemy zastąpić wyskakujące okienko przyjemniejszą ikonką.

```
function showUsernameStatus() {
  …
    if (request.responseText == "okay") {
      document.getElementById("status").src = "images/okay.png";
    }
    else {
      alert("Przykro nam, ale nazwa użytkownika jest zajęta.");
      document.getElementById("status").src = "images/inUse.png";
      …
    }
  …
}
```

Ta grafika jest wyświetlana, gdy serwer zaakceptuje nazwę użytkownika.

Ta grafika zostaje wyświetlona, jeżeli nazwa użytkownika jest zajęta.

validation.js

WYSIL SZARE KOMÓRKI

Co sądzisz o tym rozwiązaniu? Czy jest ono zgodne z zasadą rozdzielania prezentacji i treści? Zmieniłbyś tutaj coś?

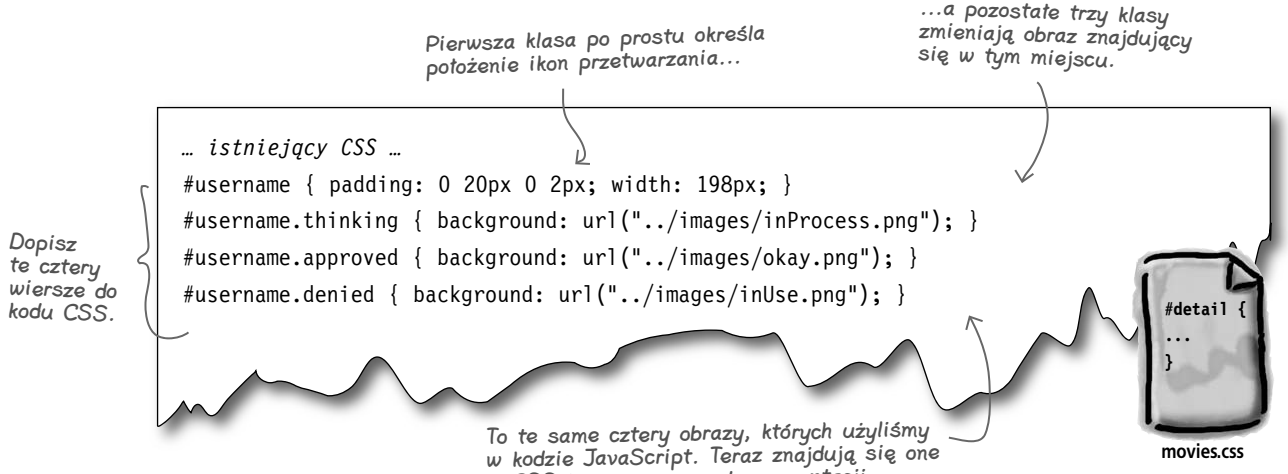
Staraj się trzymać prezentację w CSS-ie, a zachowanie w JavaScripcie.

XHTML określa strukturę i treść. CSS powinien zajmować się prezentacją, obrazami, kolorami i czcionkami. Z kolei JavaScript powinien opisywać, co robi strona, czyli jej zachowanie. Pomieszanie tych obszarów sprawi, że projektant nie będzie mógł zmienić obrazu, ponieważ będzie on określony w kodzie — a wtedy programista będzie musiał zmieniać strukturę strony. To nigdy nie prowadzi do niczego dobrego.

Choć nie jest to zawsze możliwe, powinieneś starać się utrzymywać prezentację w CSS-ie i używać JavaScriptu do interakcji z CSS-em, a nie do bezpośredniej zmiany prezentacji strony.

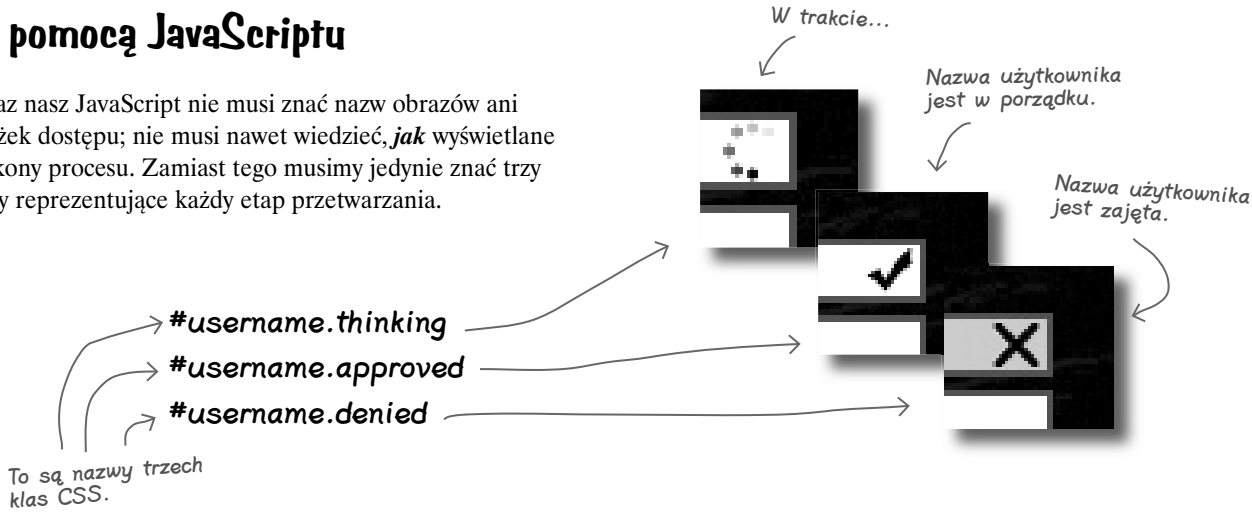
Utwórzmy klasy CSS dla każdego stanu przetwarzania żądania...

Zamiast bezpośrednio zmieniać obraz, umieścimy wszystkie informacje o obrazie w CSS-ie. Otwórz plik *movies.css* i dopisz poniższe selektory CSS:



...i zmieńmy klasę CSS za pomocą JavaScriptu

Teraz nasz JavaScript nie musi znać nazw obrazów ani ścieżek dostępu; nie musi nawet wiedzieć, ***jak*** wyświetlane są ikony procesu. Zamiast tego musimy jedynie znać trzy klasy reprezentujące każdy etap przetwarzania.



Teraz możemy (ponownie) wprowadzić zmiany do JavaScriptu. Tym razem będziemy zmieniali klasę CSS, zamiast bezpośrednio zmieniać obraz.

```
function checkUsername() {
  document.getElementById("status").src = "images/inProcess.png");
  document.getElementById("username").className = "thinking";
  request = createRequest();
  ...
```

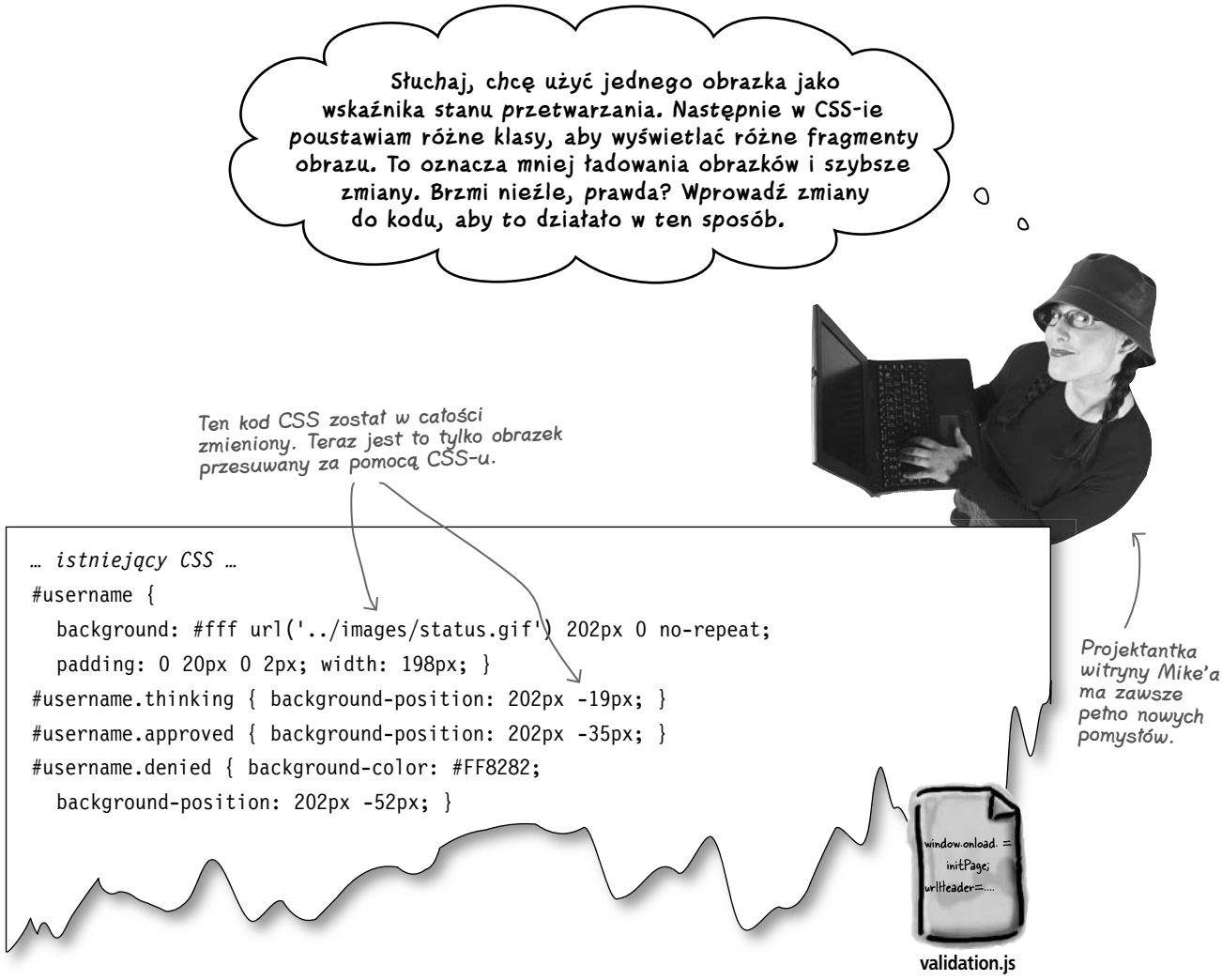
validation.js

```
window.onload = initPage;
urlHeader=....
```

Za pomocą właściwości className elementu możesz zmienić klasę CSS.

Pamiętaj, aby usunąć wiersze, które bezpośrednio zmieniały obrazek.

```
function showUsernameStatus() {
  …
    if (request.responseText == "okay") {
      document.getElementById("status").src = "images/okay.png";
      document.getElementById("username").className = "approved";
    }
    else {
      alert("Przykro nam, ale nazwa użytkownika jest zajęta.");
      document.getElementById("status").src = "images/inUse.png");
      document.getElementById("username).className = "denied");
    }
  …
}
```



Zmiany? Nie potrzebujemy ich!

Projektantka witryny Mike'a wprowadziła wiele zmian, ale nie zmieniła nazw klas CSS dla każdego etapu przetwarzania. To oznacza, że ***cały kod JavaScript dalej działa***, bez aktualizacji! Gdy oddzielisz od siebie trzy aspekty aplikacji (treść, prezentację i zachowanie), wprowadzanie do niej zmian będzie ***znacznie*** łatwiejsze.

Kod CSS może być zmieniany w dowolnym momencie i nawet nie musimy o tym wiedzieć. Jeżeli tylko nie zmienią się nazwy klas CSS, nasz kod będzie dalej działał.

Odpowiednie oddzielenie treści, prezentacji i zachowania znacznie zwiększa elastyczność aplikacji.

Zezwalaj na rejestrację tylko wtedy, gdy wprowadzono odpowiednie dane

Po umieszczeniu wskaźników stanu przetwarzania pozostało nam jedynie wyłączyć przycisk *Zarejestruj* podczas ładowania strony i włączyć go, gdy nazwa użytkownika będzie w porządku.

Musimy wprowadzić tylko kilka zmian do pliku *validation.js*:

Wyłączenie przycisku Zarejestruj

Nazwa użytkownika nie jest sprawdzana zaraz po załadowaniu strony, dlatego możemy wyłączyć przycisk *Zarejestruj* w kodzie inicjalizującym.

Gdy przypiszemy właściwości disabled wartość true, użytkownicy będą mogli wypełniać pola, ale nie będą mogli kliknąć przycisku wysyłania.

```
function initPage() {
  document.getElementById("username").onblur = checkUsername;
  document.getElementById("register").disabled = true;
}
```

validation.js

Włączenie przycisku Zarejestruj

Jeżeli nazwa użytkownika jest w porządku, użytkownik może się zarejestrować, więc musimy włączyć przycisk *Zarejestruj*. Jeśli jednak występuje problem z nazwą, użytkownik musi wybrać inną; przycisk *Zarejestruj* powinien zatem być dalej wyłączony. Aby ułatwić użytkownikowi wprowadzenie zmian, kiedy nazwa zostanie odrzucona, skierujmy go z powrotem do pola z nazwą użytkownika:

Jeżeli nazwa użytkownika jest w porządku, włącz przycisk „Zarejestruj".

To kieruje użytkownika z powrotem do pola nazwy użytkownika.

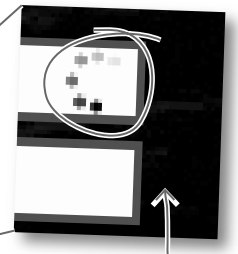
```
function showUsernameStatus() {
  …
    if (request.responseText == "okay") {
      document.getElementById("username").className = "approved";
      document.getElementById("register").disabled = false;
    }
    else {
      document.getElementById("username").className = "denied";
      document.getElementById("username").focus();
      document.getElementById("username").select();
      document.getElementById("register").disabled = true;
      …
    }
  …
}
```

Jeżeli nazwa użytkownika jest zajęta, przycisk „Zarejestruj" powinien być wyłączony.

validation.js

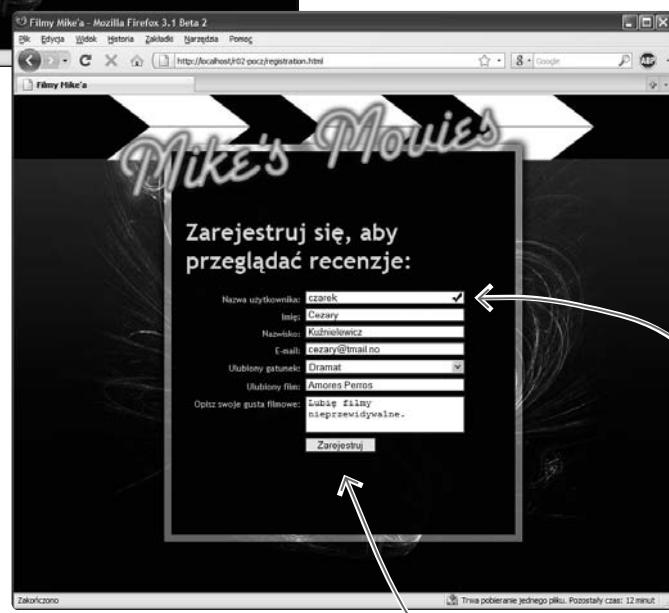
Jazda próbna

Upewnij się, że wprowadziłeś zmiany w plikach *validation.js* i *movies.css*. Załaduj stronę i sprawdź, czy wszystko działa zgodnie z oczekiwaniami.



Po wpisaniu nazwy użytkownika powinien zostać wyświetlony ten obrazek postępu.

Przycisk wysyłania jest wyłączony.



Ten obrazek informuje, że nazwa użytkownika jest w porządku.

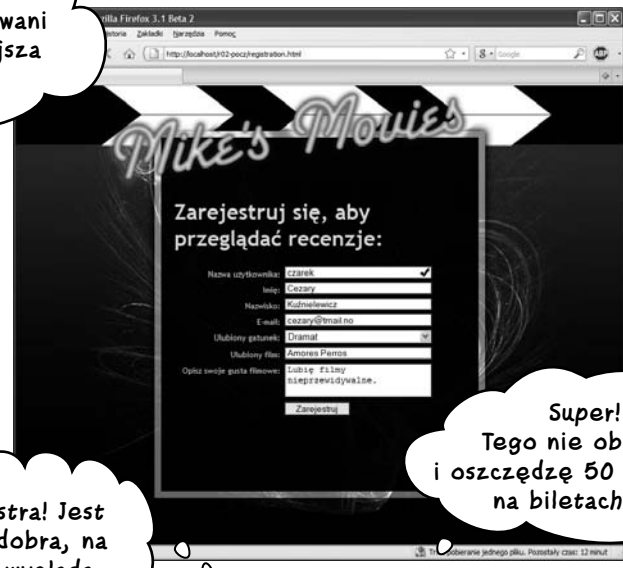
Teraz możesz przesłać stronę.

Uwaga!

Pliki z obrazami, do których odwołuje się kod CSS, znajdują się w pobranym folderze z przykładami do tego rozdziału.

Upewnij się, że masz kompletny folder z przykładami, który zawiera obrazy stanów przetwarzania.

Właśnie to miałem na myśli: usatysfakcjonowani użytkownicy i fajniejsza strona rejestracji.



Super! Tego nie obejrzę i oszczędzę 50 złotych na biletach.

Ekstra! Jest tak dobra, na jaką wygląda...

Mike jest szczęśliwy...

...a fani mogą się dostać do jego recenzji.

Teraz strona Mike'a...

- ...umożliwia użytkownikom pracę, podczas gdy nazwy użytkowników są weryfikowane przez serwer.
- ...zapobiega popełnianiu błędów, wyłączając przyciski, których użycie nie jest bezpieczne lub właściwe. Włącza je wtedy, gdy są przydatne.
- ...nie denerwuje użytkowników wyskakującymi okienkami, a mimo to przekazuje wizualną informację zwrotną.

W międzyczasie zacząłeś myśleć o projektowaniu aplikacji w zupełnie nowy sposób... Sposób, który wykracza poza tradycyjny model żądanie – oczekiwanie – odpowiedź.

Magnesiki z etykietami

Wszystkie etykiety opisujące, co się dzieje na nowej, poprawionej stronie rejestracji, spadły na podłogę. Czy potrafisz umieścić je w odpowiednich miejscach?


```
request = createRequest();
```

```
window.
    init
urlHeader=....
```

```
showUsernameStatus()
```

0 Niezainicjalizowane

2 Przygotowywanie odpowiedzi

3 Pobieranie odpowiedzi

4 Gotowa odpowiedź

```
request.send(null);
```

1 Zainicjalizowane

Gdy użytkownik opuszcza pole, funkcja obsługująca zdarzenie tworzy obiekt żądania.

Serwer wysyła odpowiedzi na kilku etapach procesu.

Podczas pracy nad problemem serwer odpowiada z readyState równym 2. Stan i nagłówki odpowiedzi są dostępne.

Przed wykonaniem tej instrukcji obiekt żądania wie, jak i gdzie się połączyć.

Na tym etapie dane są pobierane do obiektu żądania, ale nie są gotowe do użycia.

Obiekt żądania zostat utworzony, ale nie zawiera danych ani informacji w swoich właściwościach.

Gdy readyState wynosi 4, funkcja zwrotna używa odpowiedzi serwera do zaktualizowania strony.

Znajdź słowo

Poświęć chwilę i pozwól trochę popracować prawej półkuli. Oto standardowe wyszukiwanie słów. Wszystkie zostały wymienione w tym rozdziale.

X	A	R	S	M	O	K	E	J	U	D	H	E
A	C	T	I	V	E	X	O	B	J	E	C	T
C	V	I	O	R	S	M	A	L	T	R	S	V
Q	R	L	H	I	C	L	V	J	A	R	S	L
J	U	E	O	N	U	H	A	E	A	H	A	R
A	M	N	A	I	N	T	L	Y	H	E	R	A
Z	O	E	U	T	S	T	F	I	D	N	E	S
H	P	T	K	P	E	P	I	N	L	O	L	N
G	E	Y	C	A	E	R	L	O	X	L	B	R
A	N	I	A	G	R	E	E	A	L	D	G	R
O	U	N	B	E	D	Q	B	Q	L	E	A	K
I	N	G	L	F	A	U	R	L	U	N	S	A
N	D	C	L	R	I	E	F	R	N	E	D	Y
A	R	E	A	D	Y	S	T	A	T	E	S	D
J	E	R	C	I	C	T	H	R	I	Z	A	T

Lista słów

ActiveXObject

Ajax

Callback

CreateRequest

Initpage

Null

Open

Readystate

Send

URL

XMLHttpRequest

Magnesiki z etykietami: Rozwiązanie

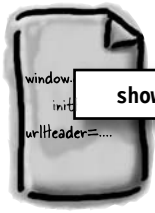
Wszystkie etykiety opisujące, co się dzieje na nowej, poprawionej stronie rejestracji, spadły na podłogę. Czy potrafisz umieścić je w odpowiednich miejscach?

Obiekt żądania został utworzony, ale nie zawiera danych ani informacji w swoich właściwościach.

Gdy użytkownik opuszcza pole, funkcja obsługująca zdarzenie tworzy obiekt żądania.


```
request = createRequest();
```

Podczas pracy nad problemem serwer odpowiada z readyState równym 2. Stan i nagłówki odpowiedzi są dostępne.




```
showUsernameStatus()
```

0 Niezainicjalizowane

Przed wykonaniem tej instrukcji obiekt żądania wie, jak i gdzie się połączyć.

2 Przygotowywanie odpowiedzi

3 Pobieranie odpowiedzi

```
request.send(null);
```

4 Gotowa odpowiedź

1 Zainicjalizowane

Na tym etapie dane są pobierane do obiektu żądania, ale nie są gotowe do użycia.

Gdy readyState wynosi 4, funkcja zwrotna używa odpowiedzi serwera do zaktualizowania strony.

Serwer wysyła odpowiedzi na kilku etapach procesu.

Znajdź słowo — Rozwiązanie

X	A	R	S	M	O	K	E	J	U	D	H	E
A	**C**	**T**	**I**	**V**	**E**	**X**	**O**	**B**	**J**	**E**	**C**	**T**
C	V	I	O	R	S	**M**	**A**	L	T	R	S	V
Q	R	L	H	**I**	C	**L**	V	**J**	A	R	S	L
J	U	E	O	**N**	U	**H**	A	E	**A**	H	A	R
A	M	N	A	**I**	N	**T**	L	Y	H	E	R	A
Z	**O**	E	U	**T**	S	**T**	F	I	**D**	**N**	**E**	**S**
H	**P**	T	**K**	**P**	E	**P**	I	N	L	O	L	N
G	**E**	Y	**C**	**A**	E	**R**	L	O	X	L	B	R
A	**N**	I	**A**	**G**	R	**E**	E	A	**L**	D	G	R
O	U	N	**B**	**E**	D	**Q**	B	Q	**L**	E	A	K
I	N	G	**L**	F	A	**U**	**R**	**L**	**U**	N	S	A
N	D	C	**L**	R	I	**E**	F	R	**N**	E	D	Y
A	**R**	**E**	**A**	**D**	**Y**	**S**	**T**	**A**	**T**	**E**	S	D
J	E	R	**C**	I	C	**T**	H	R	I	Z	A	T