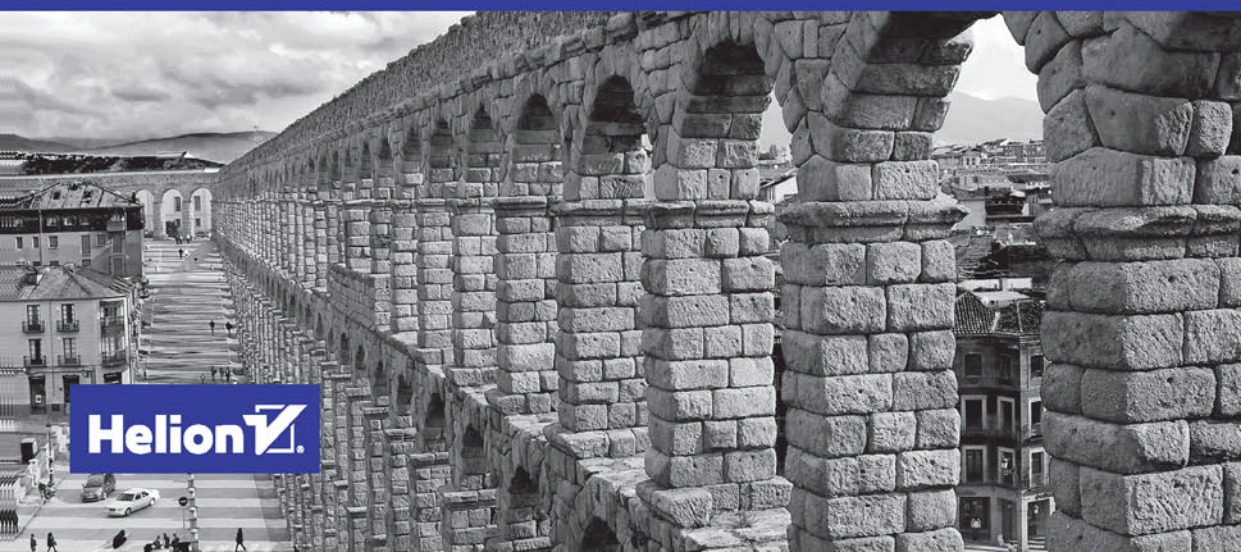
Luke Welling
Laura Thomson

PHP i MySQL

Tworzenie stron WWW
Vademecum profesjonalisty

Wydanie V

Rozwiązanie dla niezawodnych, dynamicznych witryn!



Tytuł oryginału: PHP and MySQL Web Development (5th Edition)

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-3256-0

Authorized translation from the English language edition, entitled: PHP AND MYSQL WEB DEVELOPMENT, Fifth Edition, ISBN 0321833899; by Luke Welling; and by Laura Thomson; published by Pearson Education, Inc, publishing as Addison Wesley.

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/phmsv5.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/phmsv5
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Rozdział 23.

Integracja JavaScriptu i PHP

W tym rozdziale przedstawione zostaną sposoby stosowania języka JavaScript do interakcji ze skryptami PHP wykonywanymi na serwerze w celu realizacji akcji, które nie wymagają pełnego odświeżania strony w przeglądarce.

Oto kluczowe zagadnienia, które zostały opisane w tym rozdziale:

- Prezentacja frameworka jQuery.
- Podstawowe pojęcia oraz techniki związane ze stosowaniem jQuery.
- Integracja jQuery i PHP.
- Tworzenie aplikacji do pogawędek przy użyciu jQuery i PHP.

Przedstawienie technologii AJAX

Asynchroniczne żądania wykonywane przez przeglądarki WWW są powszechnie nazywane żądaniami AJAX, przy czym „AJAX" to akronim pochodzący od słów *Asynchronous JavaScript and XML*, który powstał około 2003 roku. Mimo że w nazwie pojawia się słowo „XML", w tym rozdziale nie będzie mowy o tym języku, gdyż w nowoczesnych rozwiązaniach AJAX zazwyczaj operuje bądź to na kodzie HTML, bądź też na danych w formacie JSON (JavaScript Object Notation).

Powodem, dla którego z punktu widzenia twórców aplikacji internetowych AJAX jest interesującą i potężną technologią, jest słowo odpowiadające pierwszej literze akronimu: technologia AJAX pozwala na wykonywanie żądań *asynchronicznych*. W praktyce oznacza to, że istnieje możliwość przesyłania na serwer, na którym są wykonywane skrypty PHP, żądań generowanych przez skrypty JavaScript, i to bez konieczności odświeżania całych stron wyświetlanych w przeglądarce. Proces ten pozwala na tworzenie aplikacji internetowych zapewniających użytkownikom doskonałe wrażenia i bardzo przypominających klasyczne aplikacje komputerowe, a jednocześnie pozwala na implementację interfejsu użytkownika w sposób modularny, którego nie można byłoby uzyskać w przypadku stosowania zwyczajnych, pełnych żądań, sprawiających, że strona za każdym razem jest pobierana i odświeżana w całości.

Termin „AJAX" stał się popularny w roku 2003, kiedy to implementacje języka JavaScript w większości nowoczesnych przeglądarek zaczęły obsługiwać możliwości generowania żądań asynchronicznych, realizowanych przy użyciu klasy `XMLHttpRequest` (czasami określanej jako XHR). Niemniej jednak obecnie, w nowoczesnej erze aplikacji internetowych, zamiast tych niskopoziomowych API powszechnie stosowane są wszechstronne i działające we wszystkich przeglądarkach frameworki JavaScript. Na potrzeby tego rozdziału do przedstawienia sposobów wykorzystania technologii AJAX do komunikacji z serwerem WWW zostanie użyty bardzo popularny framework JavaScript — jQuery.

Krótka prezentacja jQuery

jQuery jest niezwykle popularnym frameworkiem JavaScript. Rozwiązania takie jak jQuery odgrywają obecnie bardzo ważną rolę, tworząc jednolite API pozwalające na budowanie w języku JavaScript oprogramowania, które będzie działać niezależnie od przeglądarki wykorzystywanej przez użytkownika. Bez frameworków takich jak jQuery podczas pisania aplikacji ich twórcy musieliby samodzielnie radzić sobie z osobliwościami poszczególnych przeglądarek oraz ich różnych wersji i rozbieżnościami w ich działaniu. Frameworki rozwiązują te problemy, pozwalając programistom skoncentrować się na logice pisanych aplikacji, a nie na wszelkich możliwych przeglądarkach, z których potencjalnie mogą korzystać użytkownicy.

Framework jQuery nie tylko sam oferuje ogromne możliwości, lecz jest także bardzo elastyczny i rozszerzalny, gdyż udostępnia całą kolekcję wysokiej klasy wtyczek. Dzięki tym wtyczkom dostępna jest większość możliwości funkcjonalnych, których programista może potrzebować w aplikacji. W tym rozdziale wykorzystywane będą jedynie podstawowe możliwości samego frameworka jQuery (określane jako jQuery Core), a w szczególności te związane z technologią AJAX.

Stosowanie jQuery w aplikacjach internetowych

Zastosowanie jQuery jako jednego z elementów z przybornika narzędzi do tworzenia aplikacji internetowych jest wyjątkowo proste. Ponieważ framework jQuery jest zwyczajną biblioteką JavaScript, wystarczy go w standardowy sposób dołączyć do dokumentu HTML, używając znacznika `<script>`.

Bibliotekę jQuery można dołączyć do strony na dwa sposoby:

- Pobrać ją i zainstalować jako jeden z elementów tworzonej aplikacji internetowej, a następnie odwoływać się do jej pliku w znaczniku `<script>`.
- Skorzystać z CDN jQuery do pobierania biblioteki jQuery, dzięki czemu nie trzeba będzie dodawać żadnych plików do lokalnego projektu. W tym przypadku znacznik `<script>` będzie się odwoływał do zewnętrznego adresu URL.

W celu zapewnienia jak największej przenaszalności kodu w tej książce wykorzystane zostanie to drugie rozwiązanie.

A zatem zapewnienie możliwości użycia jQuery w aplikacji internetowej sprowadza się do dołączenia tej biblioteki do dokumentu HTML przy wykorzystaniu poniższego znacznika `<script>`, odwołującego się do jej najnowszej wersji:

```
<script scr="//code.jquery.com/jquery-3.1.1.min.js" />
```

Warto zwrócić uwagę na to, że w powyższym adresie URL został pominięty używany protokół (na przykład http://). Jest to rozwiązanie celowe, informujące przeglądarkę, że zasób, do którego adres URL się odwołuje, należy pobrać przy użyciu protokołu zdefiniowanego przez dokument nadrzędny. A zatem jeśli strona została pobrana przy wykorzystaniu protokołu https://, to zostanie on także użyty do pobrania biblioteki. Dzięki zastosowaniu takiego rozwiązania można uniknąć ewentualnych komunikatów o zagrożeniach, które przeglądarka mogłaby wyświetlać na przykład w przypadku pobierania niebezpiecznego zasobu na stronie pobranej przy użyciu bezpiecznego protokołu.

Wczytanie tej jednej biblioteki jQuery pozwala aplikacji internetowej na korzystanie z jej pełnych możliwości! W kilku kolejnych punktach rozdziału przedstawione zostaną podstawowe pojęcia związane z biblioteką jQuery oraz jej możliwości.

Podstawowe pojęcia i techniki związane ze stosowaniem jQuery

Na samym początku prezentacji sposobów stosowania jQuery opisane zostaną podstawowe pojęcia związane z tą biblioteką. Przede wszystkim możliwości jQuery są udostępniane programistom za pośrednictwem przestrzeni nazw funkcji jQuery, zawierającej pełne możliwości funkcjonalne tej biblioteki.

Ten uchwyt do przestrzeni nazw jQuery jest używany za każdym razem, kiedy chcemy skorzystać z możliwości biblioteki. Niemniej jednak wpisywanie `jQuery` za każdym razem byłoby dość niewygodne, dlatego też jQuery tworzy nazwę zastępczą (tak zwany alias), która pozwala odwoływać się do jQuery przy użyciu symbolu `$`. To właśnie ta nazwa zastępcza będzie stosowana w tym rozdziale, gdyż to ona jest przeważnie wykorzystywana podczas tworzenia aplikacji z użyciem biblioteki jQuery.

Trzeba jednak pamiętać, że w przypadku stosowania jQuery wraz z innymi frameworkami JavaScript, które także próbują korzystać z symbolu `$` jako *swojej* nazwy zastępczej, wciąż można używać jQuery, wykorzystując tryb „bez konfliktów”. W tym celu należy użyć wywołania `jQuery.noConflict()`; zwraca ono instancję, którą można przypisać dowolnej zmiennej lub dowolnie wybranej nazwie zastępczej:

```
var $nowajQuery = jQuery.noConflict();
```

Po tych wszystkich wyjaśnieniach możemy już przejść do przedstawienia dwóch podstawowych terminów związanych ze stosowaniem jQuery, którymi są „selektory” oraz „zdarzenia”.

Stosowanie selektorów jQuery

Selektory można sobie wyobrazić jako pewien rodzaj języka zapytań, pozwalający na identyfikację elementów dokumentów HTML na podstawie podanych kryteriów i wykonywanie na nich określonych operacji bądź też określanie logiki obsługi zdarzeń generowanych przez te elementy. Ten niby-język oferuje niezwykle duże możliwości, pozwalając na błyskawiczne odwoływanie się do elementów HTML stron na podstawie ich różnych atrybutów.

Aby lepiej wyjaśnić, jak działają selektory, w pierwszej kolejności przedstawimy (na listingu 23.1) prosty dokument HTML, który będzie podstawą do dalszych rozważań.

Listing 23.1. *prosty_formularz.html — prosty formularz używany do prezentacji selektorów*

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Prosty formularz</title>
</head>
<body>
  <form id="mojFormularz">
   <label for="imie">Imię</label><br/>
   <input type="text" name="personalia[imie]"
          id="imie" class="imieinazw"/><br/>
   <label for="nazwisko">Nazwisko</label><br/>
   <input type="text" name="personalia[nazwisko]"
          id="nazwisko" class="imieinazw"/><br/>
```

```
    <button type="submit">Prześlij formularz</button>
    </form>

   <hr/>

   <div id="konsolaW3">
     <h3>Konsola WWW</h3>
   </div>

   <script src="//code.jquery.com/jquery-3.1.1.min.js"></script>
 </body>
 </html>
```

W oparciu o przykładowy kod HTML przedstawiony na listingu 23.1 poniżej przeanalizowanych zostanie kilka różnych sposobów wykorzystania selektorów jQuery do odwołania się do rozmaitych elementów HTML. Jeśli trzeba wybrać jeden konkretny element, najlepszym rozwiązaniem jest skorzystanie z jego atrybutu `id`:

```
var nazwisko = $('#nazwisko');
```

To pierwsza z wielu dostępnych składni selektorów, korzystająca z operacji #, informującej, że łańcuch podany za tym znakiem określa wartość atrybutu `id` docelowego elementu HTML. W razie konieczności wybrania grupy elementów, na przykład obu pól tekstowych, można podać ich selektory, oddzielając je od siebie znakiem odstępu, jak pokazano w poniższym przykładzie:

```
var personaliaElem = $('#imie #nazwisko');
```

Powyższe wywołanie spowodowałoby zapisanie w zmiennej `personaliaElem` tablicy zawierającej dwa węzły — dwa elementy HTML, których identyfikatorami są, odpowiednio, `imie` i `nazwisko`.

Jednak w przypadku wybierania wielu elementów zazwyczaj nie określa się listy poszczególnych elementów przy użyciu operatora # i identyfikatorów. Znacznie częściej wybiera się grupę elementów należących do określonej klasy, niezależnie od ich identyfikatorów. Używa się do tego selektorów klasy, które mają następującą postać:

```
var personaliaElem = $('.imieinazw');
```

Ponieważ oba pola tekstowe w przedstawionym przykładowym dokumencie HTML należą do klasy `imieinazw` (określanej przy użyciu atrybutu `class` znacznika HTML), oba powyższe selektory dadzą te same wyniki. Selektory mogą także wybierać elementy na podstawie innych atrybutów HTML, a nie tylko atrybutów `id` i `class`; oto przykład:

```
var personaliaElem = $('input[type="text"]);
```

Ten przykład przedstawia nową składnię selektora, pozwalającą na odnajdywanie elementów HTML, których dowolnie wybrany argument będzie mieć dowolnie określoną wartość. W tym przypadku wybierane są wszystkie elementy `<input>` w dokumencie, w których atrybut `type` ma wartość `text`. Ponieważ w przykładowym dokumencie HTML są tylko dwa takie elementy, które jednocześnie mają tą samą wartość atrybutu `class`, a ich identyfikatorami są, odpowiednio, `imie` i `nazwisko`, wszystkie trzy powyższe przykłady dadzą taki sam wynik i zwrócą te same elementy.

Nie powinno być większym zaskoczeniem, że istnieje także możliwość wybierania elementów na podstawie ich typu. Na przykład gdyby konieczne było zwrócenie całej zawartości dokumentu HTML, można by ją pobrać przy użyciu następującego wywołania:

```
var cialoDokumentu = $('body');
```

Oprócz możliwości wybierania konkretnych elementów na podstawie wartości ich atrybutów lub nazwy elementu jQuery oferuje także możliwość stosowania zestawu pseudoselektorów pozwalających twórcom aplikacji na wybieranie elementów w sposób, który bardziej przypomina programowanie. W tym rozdziale nie zostaną przedstawione wszystkie dostępne pseudoselektory (ani nawet składnia wszystkich dostępnych rodzajów selektorów), niemniej jednak opiszemy kilka najbardziej użytecznych oraz najczęściej używanych rodzajów selektorów, a także ich składnię:

```
var pierwszePole = $('input:first');
```

Powyższe wywołanie zwraca pierwszy element <input> znaleziony w dokumencie. Gdyby konieczne było ograniczenie wyszukiwania do pierwszego elementu <input> znajdującego się w konkretnym formularzu, to można by to zrobić, łącząc ich selektory w następujący sposób:

```
var pierwszePole = $('#mojFormularz input:first');
```

Kolejnym przydatnym selektorem, zwłaszcza w przypadku operowania na tabelach HTML, jest selektor pozwalający na wybranie co drugiego wyniku zwróconego przez inny selektor. Na przykład wiadomo, że poniższy selektor zwróci każdy element <tr> w danym dokumencie HTML:

```
var wiersze = $('tr');
```

Dzięki zastosowaniu dodatkowego pseudoselektora :even lub :odd można wybrać każdy co drugi element ze zbioru zwracanego przez wcześniejszy selektor, przy czym mogą to być, odpowiednio, elementy „parzyste" i „nieparzyste":

```
var wierszeNP = $('tr:odd');
var wierszeP  = $('tr:even');
```

Selektorów można także używać do operowania na podstawowych obiektach JavaScript, takich jak obiekt dokumentu dostępny domyślnie na każdej stronie HTML (obiekt ten reprezentuje cały dokument). W takim przypadku wystarczy przekazać obiekt jako selektor:

```
var jQuerySelektorDoc = $(document);
```

I w końcu, choć z technicznego punktu widzenia nie ma to nic wspólnego z selektorami, w podobny sposób można tworzyć w pamięci zupełnie nowe elementy HTML, a następnie wykonywać na nich operacje i dodawać je do istniejącego dokumentu HTML, co właściwie odpowiada modyfikowaniu zawartości strony bez jej odświeżania. Na przykład załóżmy, że konieczne jest utworzenie nowego elementu <p>. Można to zrobić błyskawicznie w poniższy sposób:

```
var nowyAkapit = $('<p>');
```

Teoretycznie przy użyciu tej techniki można by tworzyć całe sekcje dokumentu HTML, a nawet całe dokumenty:

```
var nowyAkapit = $('<p>To jest <strong>bardzo ważny tekst</strong>.</p>');
```

Powyższe informacje to bardzo proste, skrócone wprowadzenie do zagadnień związanych z selektorami jQuery, niemniej jednak powinny one wystarczyć Czytelnikowi do zrozumienia kolejnych przykładów przedstawionych w tym rozdziale, prezentujących sposoby korzystania z technologii AJAX. Aby dokładniej poznać wszystkie dostępne selektory jQuery oraz ich składnię, należy zajrzeć do internetowej dokumentacji biblioteki, do jej sekcji poświęconej selektorom: *http://learn.jquery.com/using-jquery-core/selecting-elements/*.

Operowanie na zbiorach wynikowych selektorów

Skoro zostały już przedstawione podstawowe metody przeglądania dokumentów HTML służące do poszukiwania interesujących elementów, nadszedł czas, by zaprezentować różne sposoby operowania na tych tak zwanych zbiorach wynikowych selektorów. Koniecznie należy sobie

uświadomić, że zbiory wynikowe selektorów zwracane przez jQuery z założenia są traktowane jako wieloelementowe. Oznacza to, że selektor zwracający jeden element nie jest traktowany jako jednostkowy element, lecz jako zbiór składający się z tylko jednego elementu. To z kolei znaczy, że można wykonywać operacje na całym zbiorze, niezależnie od tego, czy znajduje się w nim tylko jeden element, czy też całe setki elementów.

W ramach przykładu przedstawiona zostanie metoda jQuery o nazwie `val()`, pozwalająca programiście na pobranie lub ustawienie atrybutu elementu wejściowego:

```
var myInput = $('#first_name');
console.log("Wartość elementu input o identyfikatorze #imie wynosi: '
  + myInput.val());
myInput.val('Jan');
console.log("Wartość pola o identyfikatorze #imie została zmieniona na: '
  + myInput.val());
```

W tym przykładzie wybierany jest tylko jeden element HTML o podanym identyfikatorze, o wartości `imie`. Niemniej jednak ponieważ selektor zawsze zwraca zbiór elementów, tej samej metody można by używać zawsze. Aby powyższy przykład był nieco bardziej praktyczny, przedstawiona zostanie druga metoda jQuery, `addClass()`; zgodnie z tym, co sugeruje jej nazwa, służy ona do określania klas, do których należą elementy HTML. Poniżej zaprezentowany został przykład użycia tej metody:

```
var polaPresonaliow = $('.imieinazw');
polaPresonaliow.addClass('kontrolki-formularza');
```

Ten przykładowy kod odnajduje wszystkie elementy HTML należące do klasy `imieinazw`, a następnie dodaje je także do klasy `kontrolki-formularza`.

W bardziej praktycznych zastosowaniach, w których elementy stron mogą, lecz nie muszą istnieć, bardzo dużego znaczenia nabiera sprawdzanie, czy zbiór wynikowy zwrócony na skutek wykonania selektora zwrócił jakieś elementy, czy nie. Ponieważ z technicznego punktu widzenia zbiór zawierający zero elementów wciąż jest zbiorem (a zatem jego sprawdzenie w języku JavaScript zwróciłoby wartość `true`), faktyczną wielkość zbioru należy odczytywać przy użyciu właściwości `length`:

```
var nameFields = $('.imieinazw');

if(nameFields.length > 0) {
  console.log("Znaleziono elementy należące do klasy 'imieinazw'.");
} else {
  console.log("Nie znaleziono elementów należących do klasy 'imieinazw'.");
}
```

Wprowadzenie do zdarzeń jQuery

Zdarzenia są jednym z kluczowych elementów języka JavaScript, a co za tym idzie, także i jQuery. Ponieważ sam JavaScript jest asynchronicznym językiem programowania (co oznacza, że logika programu nie zawsze jest wykonywana w tej samej kolejności), zdarzenia są niezbędne do tego, by zapewnić, że w przypadku zmian kolejności wykonywania nie zostanie utracone znaczenie aplikacji.

Programiści jQuery mają do dyspozycji dziesiątki różnych zdarzeń, reprezentujących przeróżne okoliczności. Niektóre z tych zdarzeń pochodzą z języka JavaScript; przykładem takiego zdarzenia może być `click`, emitowane za każdym razem, gdy użytkownik coś kliknie. Z kolei inne zdarzenia są konstrukcjami jQuery; przykładem może być `ready` — zdarzenie zgłaszane, kiedy wszystkie zasoby danego dokumentu HTML zostaną prawidłowo pobrane.

W hierarchii dokumentu HTML zdarzenia propagują z elementu źródłowego, poprzez jego elementy nadrzędne, aż w końcu są przekazywane przez cały dokument (co wyzwala akcje we wszystkich elementach), które danego zdarzenia nasłuchują. Podobnie jak w wielu innych systemach obsługi zdarzeń, procedury ich obsługi, nazywane także funkcjami nasłuchującymi, mogą zatrzymać propagację zdarzenia. W przypadku biblioteki jQuery podczas obsługi zdarzeń zazwyczaj w pierwszej kolejności jest używany selektor, który pozwala wybrać odpowiednie elementy, a następnie zostaje wywołana metoda `on()`, pozwalająca nasłuchiwać wybranego zdarzenia i wykonywać odpowiednią logikę, jeśli zostanie ono zgłoszone. Jednym z najprostszych możliwych przykładów obsługi zdarzeń jest nasłuchiwanie zdarzenia `ready`, zgłaszanego przez jQuery, kiedy cały dokument wraz ze wszystkimi zasobami zostanie prawidłowo pobrany:

```
$(document).on('ready', function(event) {
    // Kod do wykonania po zakończeniu pobierania dokumentu
});
```

Podobnie jak większość innych metod jQuery, także i metoda `on()` może być wywoływana na rzecz dowolnego selektora. Na przykład aby reagować na każde kliknięcie łącza, można by dołączyć funkcję nasłuchującą do każdego znacznika `<a>` z atrybutem `href` i nasłuchiwać zdarzeń `click`:

```
$('a').on('click', function(event) {
    // Czynności wykonywane za każdym razem po kliknięciu elementu <a> HTML.
});
```

Metoda `on()` jest uniwersalnym sposobem kojarzenia zdarzeń z procedurami ich obsługi, jednak zarówno dla wygody, jak i ze względów historycznych jQuery udostępnia także cały zestaw podobnych metod kojarzących funkcje nasłuchujące z konkretnymi zdarzeniami. Na przykład wywołania `$(document).on('ready', ...)` oraz `$(document).ready(...)` dają identyczne rezultaty.

W zależności od początkowego selektora może się zdarzyć, że będziemy chcieli stworzyć jedno zdarzenie dla wielu elementów HTML, lecz po jego zgłoszeniu operować wyłącznie na jednym elemencie, który je zgłosił. Jak można było zauważyć w dwóch ostatnich przykładach, domknięcie przekazane w celu obsługi zdarzenia posiadało jeden parametr: `event`. Parametr ten jest obiektem zdarzenia tworzonym w momencie jego zgłaszania; jego właściwość `target` zawiera odwołanie do konkretnego elementu strony, który zgłosił zdarzenie. A zatem wybraną operację, na przykład na kliknięty przycisku, można wykonać w następujący sposób:

```
$('button').on('click', function(event) {
    var przycisk = $(event.target);

    // Wykonanie czynności na kliknięty przycisku
});
```

I podobnie, w szczególności dla niektórych rodzajów zdarzeń, takich jak zdarzenie `click` elementu HTML `<a>`, domyślna funkcja nasłuchująca może wykonywać czynności, które nie powinny zostać wykonane. Oto przykład:

```
$('a').on('click', function(event) {
  var link = $(event.target).attr('href');
  console.log("Kliknięte łącze prowadziło do adresu URL: " + link);
});
```

Logicznie rzecz biorąc, powyższy fragment kodu powinien zapewniać możliwość nasłuchiwania na zdarzenia `click`, pobierania wartości atrybutu elementu źródłowego przy użyciu metody `attr()`, a następnie wyświetlanie wartości tego atrybutu w konsoli przeglądarki. I choć powinno tak być, to jednak powyższy kod nie będzie działał w taki sposób, gdyż istnieje domyślne działanie skojarzone z kliknięciem elementu (a konkretnie: zmiana strony wyświetlanej w przeglądarce na zasób o podanym adresie URL). Powyższy kod będzie działał właśnie w taki sposób, gdyż

niezależnie od tego, czy kod poprawnie nasłuchuje zdarzenia, będzie ono propagowane w górę dokumentu HTML, co w końcu doprowadzi do zastosowania domyślnego sposobu jego obsługi. Aby temu zaradzić, trzeba uniemożliwić propagację zdarzenia, używając do tego metody `preventDefault()`, dostępnej w każdym obiekcie zdarzenia. Poniższy przykład przedstawia fragment kodu korzystający z tej metody, który będzie działał zgodnie z oczekiwaniami:

```
$('a').on('click', function(event) {
  preventDefault();

  var link = $(event.target).attr('href');
  console.log("Kliknięte łącze prowadziło do adresu URL: " + link);
});
```

Jak wspomniano we wcześniejszej części rozdziału, istnieje wiele różnych zdarzeń, których można nasłuchiwać i które można obsługiwać — jest ich zbyt wiele, by można było je wszystkie szczegółowo opisać w tym rozdziale. Niemniej jednak tabela 23.1 zawiera listę kilku najczęściej używanych zdarzeń, które można obsługiwać przy wykorzystaniu frameworka jQuery.

Tabela 23.1. *Przydatne zdarzenia jQuery*

Zdarzenie	Typ	Opis
`change`	Zdarzenie formularza	Generowane w momencie zmiany wartości elementu formularza
`click`	Zdarzenie myszy	Generowane w momencie kliknięcia elementu
`dblclick`	Zdarzenie myszy	Generowane w momencie dwukrotnego kliknięcia elementu
`error`	Zdarzenie JavaScript	Generowane w momencie wystąpienia błędu JavaScript
`focusin`	Zdarzenie formularza	Generowane w momencie przenoszenia do elementu miejsca wprowadzania (ang. *input focus*), jednak jeszcze zanim to faktycznie nastąpi
`focus`	Zdarzenie formularza	Generowane, gdy element uzyska miejsce wprowadzania
`focusout`	Zdarzenie formularza	Generowane, gdy miejsce wprowadzania zostanie przeniesione do innego elementu
`hover`	Zdarzenie myszy	Generowane, gdy wskaźnik myszy będzie przesuwany w obszarze danego elementu
`keydown`	Zdarzenie klawiatury	Generowane po wciśnięciu klawisza
`keypress`	Zdarzenie klawiatury	Generowane po naciśnięciu klawisza, czyli jego wciśnięciu i zwolnieniu
`keyup`	Zdarzenie klawiatury	Generowane po zwolnieniu klawisza
`ready`	Zdarzenie dokumentu	Generowane po utworzeniu kompletnego obiektowego modelu dokumentu
`submit`	Zdarzenie formularza	Generowane po przesłaniu danego formularza

W poniższym przykładzie przedstawiona została zmodyfikowana wersja strony z listingu 23.1, łącząca wszystkie zaprezentowane do tej pory informacje dotyczące selektorów i zdarzeń, w celu wykonywania różnego rodzaju czynności. Nową wersję strony ukazuje listing 23.2.

Listing 23.2. *prosty_formularz_v2.html — przykład prostego formularza korzystającego z jQuery*

```
<!DOCTYPE html>
<html>
<head>
```

```
    <meta charset="utf-8" />
    <title>Prosty formularz</title>
  </head>
  <body>
    <form id="mojFormularz">
      <label for="imie">Imię</label><br/>
      <input type="text" name="personalia[imie]"
             id="imie" class="imieinazw"/><br/>
      <label for="nazwisko">Nazwisko</label><br/>
      <input type="text" name="personalia[nazwisko]"
             id="nazwisko" class="imieinazw"/><br/>
      <button type="submit">Prześlij formularz</button>
    </form>

    <hr/>

    <div id="konsolaW3">
      <h3>Konsola WWW</h3>
    </div>

    <script src="//code.jquery.com/jquery-3.1.1.min.js"></script>

    <script>
       var konsolaW3 = function(msg) {
          var konsola = $('#konsolaW3');
          var nowyKomunikat = $('<p>').text(msg);
          konsola.append(nowyKomunikat);
       }

       $(document).on('ready', function() {
          $('#imie').attr('placeholder', 'Jan');
          $('#nazwisko').attr('placeholder', 'Kowalski');
       });

       $('#mojFormularz').on('submit', function(event) {
          var imie = $('#imie').val();
          var nazwisko = $('#nazwisko').val();

          konsolaW3("Przesłano formularz!");
          alert("Witaj, użytkowniku " + imie + " " + nazwisko + "!");
        });

        $('.imieinazw').on('focusout', function(event) {
          var poleForm = $(event.target);
          konsolaW3("Wartość pola o identyfikatorze '" +
                    poleForm.attr('id') +
                    "' została zmieniona na: '" +
                    poleForm.val() +
                    "'");
          });
    </script>

  </body>
  </html>
```

Jak widać, w powyższym dokumencie HTML zostały wprowadzone znaczące zmiany — dodano do niego kilka procedur obsługi zdarzeń jQuery, których zadaniem jest ożywienie tego statycznego dokumentu. Pierwszą z tych funkcji jest `konsolaW3()`, której definicja została przedstawiona poniżej:

```
var konsolaW3 = function(msg) {
  var konsola = $('#konsolaW3');
  var nowyKomunikat = $('<p>').text(msg);
  konsola.append(nowyKomunikat);
}
```

Ta funkcja będzie używana w innych miejscach naszej aplikacji w celu wyświetlania na bieżąco informacji o realizacji skryptu. Jej działanie polega na dodawaniu nowego akapitu (elementu <p>) wewnątrz początkowo pustego elementu o identyfikatorze konsolaW3 za każdym razem, gdy konieczne jest wyświetlenie nowego komunikatu. To wspaniały przykład zastosowania jQuery do wyboru elementów, tworzenia nowej zawartości, a następnie modyfikowania wyświetlonego dokumentu HTML z poziomu kodu JavaScript.

Po przedstawieniu tej funkcji pomocniczej nadszedł czas, by zająć się opisem kluczowych możliwości funkcjonalnych prezentowanej aplikacji jQuery. Pierwszą z nich będzie funkcja wykonywana po zakończeniu wczytywania dokumentu:

```
$(document).on('ready', function() {
  $('#imie').attr('placeholder', 'Jan');
  $('#nazwisko').attr('placeholder', 'Kowalski');
});
```

Funkcja ta jest wywoływana po zakończeniu wczytywania dokumentu HTML, a jej działanie polega na dodaniu nowych atrybutów placeholder do pól tekstowych o identyfikatorach imie i nazwisko. Operacja ta jest wykonywana tak szybko, że użytkownik praktycznie nie jest w stanie jej zauważyć; nie ma znaczenia, czy atrybuty te zostaną wygenerowane przez skrypt, czy statycznie umieszczone w kodzie dokumentu HTML.

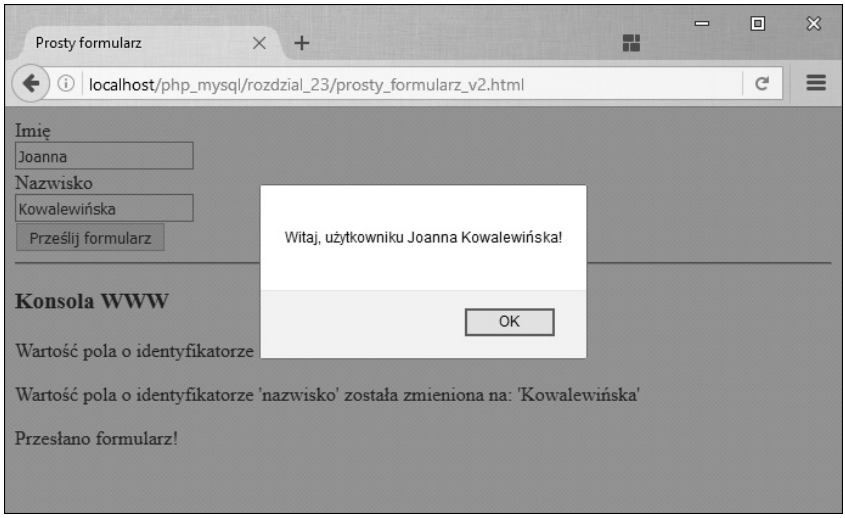
Reszta kodu aplikacji jest umieszczona w procedurach obsługi zdarzeń, które będą wykonywane asynchronicznie, a nie na skutek realizacji logicznej ścieżki działania aplikacji. Dlatego nie ma większego znaczenia, która z tych procedur zostanie opisana jako pierwsza. Poniżej przedstawiona została procedura obsługi zdarzeń focusout obu pól tekstowych formularza:

```
$('.imieinazw').on('focusout', function(event) {
  var poleForm = $(event.target);
  konsolaW3("Wartość pola o identyfikatorze '" +
            poleForm.attr('id') +
            "' została zmieniona na: '" +
            poleForm.val() +
            "'");
});
```

Po usunięciu miejsca wprowadzania z pola formularza (na przykład dlatego, że użytkownik chce wprowadzić dane w innym polu) zostanie wygenerowane zdarzenie focusout, co z kolei spowoduje wywołanie powyższej funkcji. Funkcja ta sprawdza (przy użyciu właściwości target przekazanego obiektu zdarzenia) element, który doprowadził do zgłoszenia zdarzenia, a następnie wyświetla komunikat, wywołując w tym celu przedstawioną wcześniej funkcję konsolaW3(). W rezultacie strona jest aktualizowana na bieżąco za każdym razem, gdy użytkownik zmieni zawartość dowolnego pola formularza. Efekty działania tej procedury obsługi zdarzeń zostały pokazane na rysunku 23.1.

Ostatnim zdarzeniem obsługiwanym w powyższej aplikacji jest zdarzenie submit, generowane w efekcie przesłania formularza. Zastosowany selektor określa, że funkcja nasłuchująca powinna zostać skojarzona wyłącznie z formularzem o identyfikatorze mojFormularz:

```
$('#mojFormularz').on('submit', function(event) {
  var imie = $('#imie').val();
  var nazwisko = $('#nazwisko').val();

  konsolaW3("Przesłano formularz!");
  alert("Witaj, użytkowniku " + imie + " " + nazwisko + "!");
});
```



Rysunek 23.1. *Wykonywanie czynności na formularzu obsługiwanym przez jQuery generuje rejestr operacji wyświetlany u dołu strony*

Ze względów demonstracyjnych także ta procedura obsługi zdarzeń jest bardzo prosta, a jej działanie ogranicza się do pobrania wartości z obu pól formularza i pokazania ich w okienku dialogowym przeglądarki wyświetlanym przy użyciu funkcji JavaScript `alert()`. Następnie funkcja aktualizuje samą stronę WWW, wyświetlając na niej informacje podane w formularzu.

Ten przykład kończy krótką prezentację frameworka jQuery. W żadnym razie nie można jej uznać za wyczerpującą, jednak pozwoli ona Czytelnikowi zrozumieć zagadnienia opisywane w dalszej części rozdziału, takie jak zastosowanie jQuery do komunikacji z serwerem WWW przy użyciu technologii AJAX.

Stosowanie jQuery, technologii AJAX i skryptów PHP

Oprócz wszystkich ogromnych możliwości związanych z manipulowaniem dokumentami HTML jQuery udostępnia także programistom cały zestaw możliwości funkcjonalnych przeznaczonych do prowadzenia asynchronicznej komunikacji z serwerami WWW. Możliwości te są wbudowane w implementację języka JavaScript w przeglądarkach WWW, jednak jQuery znacznie ułatwia ich wykorzystywanie, gdyż implementacje stosowane w poszczególnych przeglądarkach różnią się nieco od siebie, a jQuery wyodrębnia ich kluczowe cechy, udostępniając spójny interfejs API.

Aby zacząć korzystać z technologii AJAX przy użyciu jQuery, należy zapoznać się z przedstawionym w dalszej części rozdziału prostym przykładem aplikacji do obsługi internetowych pogawędek. Aplikacja ta będzie zapewniać wielu użytkownikom możliwość jednoczesnego prowadzenia pogawędek oraz odbierania wiadomości, i to bez konieczności odświeżania okna przeglądarki.

Ajaksowe pogawędki — skrypt serwera

Do obsługi internetowych pogawędek po stronie serwera niezbędny jest prosty skrypt PHP realizujący dwie podstawowe operacje: pobieranie przesyłanych wiadomości oraz zwracanie listy wiadomości, które jeszcze nie zostały wyświetlone danemu użytkownikowi. Ponieważ tworzona aplikacja ma korzystać z technologii AJAX, implementowany skrypt PHP będzie korzystał wyłącznie

z danych zapisanych w formacie JSON (*JavaScript Object Notation*). Co więcej, na potrzeby aplikacji zostanie utworzona tabela MySQL, gdyż aplikacja musi w trwały sposób przechowywać przesyłane wiadomości.

A zatem przed rozpoczęciem pisania skryptu PHP konieczne jest utworzenie tabeli MySQL. Poniżej przedstawione zostało polecenie `CREATE`, które tworzy bazę danych o nazwie `czat`, a w niej tabelę `wiadomosci_czatu`:

```
CREATE DATABASE czat;

USE czat;

CREATE TABLE wiadomosc_czatu (
  id INT(11) AUTO_INCREMENT PRIMARY KEY,
  wiadomosc TEXT,
  wyslana_przez VARCHAR(50),
  data_utworzenia INT(11)
);
```

Ta bardzo prosta baza tabela przechowuje podstawowe metadane o każdej wiadomości oraz jej treść. Tabela składa się z czterech kolumn: liczby, która w unikalny sposób identyfikuje każdy rekord tabeli, samej wiadomości, identyfikatora sesji użytkownika, który przesłał wiadomość, oraz liczby całkowitej reprezentującej uniksowy znacznik czasu określający, kiedy wiadomość została wysłana. Identyfikator sesji PHP jest ważny, gdyż na jego podstawie aplikacja będzie określać, czy dana wiadomość została wysłana przez użytkownika przeglądającego pogawędkę, czy też przez kogoś innego.

Listing 23.3 przedstawia kompletny kod skryptu służącego do tworzenia i wyświetlania pogawędki; poniżej został on dokładnie opisany i wyjaśniony.

Listing 23.3. *czat.php — wykonywany na serwerze skrypt tworzący i wyświetlający wiadomości*

```
<?php
session_start();
ob_start();
header("Content-type: application/json");

date_default_timezone_set('UTC');

// nawiązanie połączenia z bazą danych
$polaczenieBD = mysqli_connect('localhost', 'uzytkownik', 'haslo', 'czat');

if (mysqli_connect_errno()) {
   echo '<p>Błąd: Nie można nawiązać połączenia z bazą danych.<br />
   Proszę spróbować później.</p>';
   exit;
}

try {

    $aktualnyCzas = time();
    $idSesji = session_id();

    $czasSprawdzenia = isset($_SESSION['czas_sprawdzenia']) ?
                       $_SESSION['czas_sprawdzenia'] : $aktualnyCzas;

    $akcja = isset($_SERVER['REQUEST_METHOD']) &&
             ($_SERVER['REQUEST_METHOD'] == 'POST') ?
             'wyslij' : 'pobierz';
```

```
    switch($akcja) {
        case 'pobierz':

          //echo 'SELECT * FROM wiadomosci_czatu WHERE data_utworzenia >= '.
          //     $czasSprawdzenia. "  (". time() .") \n";

           $zapytanie = "SELECT * FROM wiadomosci_czatu WHERE data_utworzenia >= ?";

           $polecenie = $polaczenieBD->prepare($zapytanie);
           $polecenie->bind_param('s', $czasSprawdzenia);
           $polecenie->execute();
           $polecenie->bind_result($id, $wiadomosc, $idSesji, $date_created);
           $wynik = $polecenie->get_result();

           $noweWiadomosci = [];
           while($wiadomosc = $wynik->fetch_assoc()) {

               if($idSesji == $wiadomosc['wyslana_przez']) {
                  $wiadomosc['wyslana_przez'] = 'ja';
               } else {
                  $wiadomosc['wyslana_przez'] = 'inny';
               }

               $noweWiadomosci[] = $wiadomosc;
            }


           $_SESSION['czas_sprawdzenia'] = $aktualnyCzas;

           print json_encode([
               'sukces' => true,
               'wiadomosci' => $noweWiadomosci
           ]);
           exit;

        case 'wyslij':

            $wiadomosc = isset($_POST['wiadomosc']) ? $_POST['wiadomosc'] : '';
            $wiadomosc = strip_tags($wiadomosc);

            $zapytanie = "INSERT INTO wiadomosci_czatu (wiadomosc, wyslana_przez, data_utworzenia)
                          VALUES(?, ?, ?)";

            $polecenie = $polaczenieBD->prepare($zapytanie);
            $polecenie->bind_param('ssi', $wiadomosc, $idSesji, $aktualnyCzas);
            $polecenie->execute();

            print json_encode(['sukces' => true]);
            exit;
    }
} catch(\Exception $e) {
    print json_encode([
        'sukces' => false,
        'blad' => $e->getMessage()
    ]);

}
```

Powyższy prosty serwer pogawędek rozpoczyna się od włączenia sesji oraz buforowania poprzez wywołanie funkcji, odpowiednio, `session_start()` i `ob_start()`. Następnie ustawiony jest nagłówek odpowiedzi `Content-Type`, któremu zostaje przypisana wartość `applicatio/json`, dzięki czemu

klient będzie wiedział, że odpowiedź zawiera dokumenty JSON. Poza tym, aby wymusić stosowanie spójnych znaczników czasu, wywoływana jest funkcja `date_default_timezone_set()`.

Po wykonaniu tych podstawowych operacji zostaje utworzone połączenie z bazą danych MySQL, po czym skrypt sprawdza, czy zostało ono nawiązane prawidłowo. Jeśli wystąpiły jakieś problemy, to skrypt zostaje natychmiast zakończony, gdyż brak połączenia z bazą danych na tym etapie oznacza pogawędkę, w której nie będzie żadnych wiadomości.

Jeśli natomiast uda się prawidłowo nawiązać połączenie z bazą, to kolejną operacją, którą zostanie wykonana, będzie określenie sposobu obsługi bieżącego żądania. W przypadku żądań HTTP `GET` zostanie pobrana lista wszystkich wiadomości, które jeszcze nie zostały wyświetlone danemu użytkownikowi. Wiadomości te zostaną zwrócone i wyświetlone w przeglądarce. W przypadku żądań HTTP `POST` (czyli przesłania formularza) skrypt zapisze nową wiadomość, która następnie zostanie rozesłana do wszystkich innych użytkowników.

Bez względu na rodzaj żądania skrypt zawsze zwraca obiekt JSON, w którym jest dostępny klucz o nazwie `sukces`, zawierający wartość logiczną `true` lub `false`, zależnie od tego, czy operacja zakończyła się pomyślnie, czy nie. W razie niepowodzenia do obiektu zostanie także dodany klucz `blad`, zawierający komunikat błędu. W przypadku obsługi żądania HTTP `GET` zwracany obiekt będzie zawierał również klucz `wiadomosci` z listą wszystkich wiadomości, które powinny zostać wyświetlone w przeglądarce danego użytkownika.

Skrypt PHP przedstawiony na listingu 23.2 wykorzystuje kilka pojęć związanych z operacjami przeprowadzanymi na bazach danych (takich jak wstawianie i pobieranie informacji z bazy), które zostały przedstawione we wcześniejszej części książki. Niemniej jednak jego kluczowym aspektem jest sposób, w jaki jest on wykonywany. Prawidłowe działanie aplikacji wymaga tego, by przeglądarka wywoływała ten skrypt cyklicznie w celu aktualizacji interfejsu użytkownika i wyświetlania w nim nowych wiadomości. Dodatkowo strona WWW stanowiąca interfejs użytkownika aplikacji będzie udostępniała możliwość przesyłania wiadomości na serwer przy użyciu żądania AJAX, dzięki czemu będą one rozsyłane do wszystkich innych klientów biorących udział w pogawędce. W kolejnym punkcie rozdziału zostaną przedstawione kliencka część aplikacji oraz używane przez nią metody AJAX.

Metody jQuery służące do korzystania z technologii AJAX

Zanim rozpoczniemy tworzenie prostego interfejsu użytkownika prezentowanej tu aplikacji do obsługi pogawędek, przedstawimy różne metody biblioteki jQuery służące do wykonywania żądań AJAX. Warto zwrócić uwagę na to, że wszystkie metody zaprezentowane w następnej części rozdziału są w rzeczywistości uproszczonymi sposobami wywoływania jednej metody: `$.ajax()`.

Metoda jQuery $.ajax()

Oceniając metodę `$.ajax()` pod względem jej prototypu, można stwierdzić, że wygląda ona na stosunkowo prostą:

```
$.ajax(string url, object ustawienia);
```

Pierwszym parametrem tej metody jest adres URL, na jaki ma zostać wysłane asynchroniczne żądanie, a drugim — obiekt zawierający ustawienia żądania. Złożoność tej metody można określić dopiero po przejrzeniu listy wszystkich dostępnych ustawień służących do określenia sposobu jej działania, obsługi żądania oraz odpowiedzi. Ponieważ wszystkie te ustawienia są doskonale opisane w internetowej dokumentacji jQuery dostępnej na stronie *http://api.jqeury.com/jQuery.ajax/*,

nie będziemy powielać informacji o nich w tej książce. Zamiast tego w dalszej części tego podpunktu przedstawionych zostanie kilka najczęściej stosowanych przypadków użycia oraz sposobów ich realizacji z wykorzystaniem metody `$.ajax()`.

Pierwszy przykład wykonuje proste żądanie HTTP `GET`. Właściwość `success` określa funkcję, która będzie wywołana po prawidłowym wykonaniu żądania i do której zostaną przekazane dane pobrane z odpowiedzi, jej status tekstowy oraz obiekt jQuery.

```
// Wykonanie żądania HTTP GET
$.ajax('/example.php', {
    'method' : 'GET',
    'success' : function(dane, status, jqXHR) {
        console.log(data);
    }
});
```

Kolejny przykład przedstawia sposób wykonywania żądania HTTP `POST`, które przesyła na serwer jakieś dane. W przypadku prawidłowego obsłużenia żądania zostaje wywołana funkcja określona przez właściwość `success`, tak samo jak to było w przypadku generacji żądań `GET`. Jednak w tym przykładzie druga funkcja została określona we właściwości `error`. Ta funkcja zostanie wywołana, gdy wystąpi błąd (na przykład gdy serwer zwróci kod statusu `500`); informacje na jego temat mogą zostać wyświetlone w interfejsie użytkownika:

```
// Wykonywanie żądań HTTP POST z obsługą błędów
$.ajax('/example.php', {
    'method' : 'POST',
    'data' : {
        'myBoolean': true,
        'myString' : 'To są przykładowe dane.'
        },
    'success' : function(dane, status, jqXHR) {
        console.log(data);
    },
    'error' : function(jqXHR, status, zgloszonyBlad) {
        console.log("Wystąpił błąd: " + zgloszonyBlad);
    }
});
```

W przypadku chęci lub konieczności dodania do żądania jakichś nagłówków, takich jak wartości niezbędne do uwierzytelnienia żądania, można użyć klucza `headers`, któremu przypisywany jest obiekt zawierający pary nazwa-wartość, reprezentujące wszystkie potrzebne nagłówki:

```
// Dodawanie nagłówków do żądania GET
$.ajax('/example.php', {
    'method' : 'GET',
    'headers' : {
        'X-my-auth' : 'SomeAuthValue'
    }
    success: function(dana, status, jqXHR) {
        console.log(data);
    }
});
```

W przypadku generowania żądań AJAX wymagających skorzystania z protokołu uwierzytelniania HTTP w nowszych wersjach jQuery nie trzeba już samodzielnie ustawiać nagłówków uwierzytelniających HTTP przed wysłaniem żądania. W celu określenia informacji uwierzytelniających HTTP w obiekcie ustawień funkcji `$.ajax()` wystarczy teraz podać nazwę użytkownika (`username`) i hasło (`password`):

```
// Żądanie z użyciem uwierzytelniania HTTP
$.ajax('/example.php', {
    'method' : 'GET',
    'username' : 'mojanazwa',
    'password' : 'mojehaslo',
    'success' : function(data, textStatus, jqXHR) {
        console.log(data);
    }
});
```

W zależności od złożoności żądania AJAX oraz poziomu wymaganej kontroli nad nim może się okazać, że do jego wykonania będzie można użyć jednej z metod pomocniczych, eliminujących część złożoności związanych ze stosowaniem metody `$.ajax()` i wszystkich jej ustawień. Kolejny podpunkt tego rozdziału zawiera opis tych uproszczonych metod AJAX oraz sposobów ich stosowania w celu wykonywania żądań; po tej prezentacji zostanie przedstawiona następna część aplikacji do prowadzenia internetowych pogawędek.

Metody pomocnicze jQuery służące do obsługi żądań AJAX

W wielu sytuacjach elastyczność i złożoność zapewniane przez metodę `$.ajax()` nie będą potrzebne. Z tego powodu biblioteka jQuery udostępnia kilka pomocniczych metod AJAX wyodrębniających najczęściej występujące przypadki użycia. Łatwość ich stosowania ma jednak swoją cenę, gdyż czasami brakuje w nich użytecznych możliwości funkcjonalnych, takich jak obsługa błędów, które oferuje `$.ajax()`.

Poniżej przedstawiony został dość prosty sposób wykonywania żądań HTTP `GET` w celu pobrania zasobu z serwera WWW:

```
// Uproszczony sposób wykonywania żądań GET
$.get('/przyklad.php', {
    'parametrZapytania' : 'wartośćParametru'
}, function(dane, status, jqXHR) {
    console.log(dane);
});
```

W przypadku stosowania metody `$.get()` w jej wywołaniu przekazywane są adres URL żądanego zasobu, wszelkie parametry żądania (w formie zwyczajnego obiektu JavaScript) oraz funkcja zwrotna, którą należy wykonać po pomyślnym zakończeniu żądania. Jeśli podczas obsługi żądania wystąpią jakieś błędy, metoda `$.get()` zakończy się bez sygnalizowania jakichkolwiek nieprawidłowości.

Jak można się było spodziewać, istnieje także metoda `$.post()`, która działa w identyczny sposób (z tą różnicą, że zamiast żądań `GET` generuje żądania `POST`):

```
// Uproszczony sposób generowania żądań POST
$.post('/przyklad.php', {
    'parametrPost' : 'wartość parametru'
}, function(dane, status, jqXHR) {
    console.log(dane);
);
```

Istnieją również dwie dodatkowe metody pomocnicze, które mogą się okazać przydatne w pewnych okolicznościach. Pierwsza z nich, `$.getScript()`, pozwala na dynamiczne pobranie dokumentu JavaScript z serwera i wykonanie go, i to przy użyciu tylko jednego wiersza kodu:

```
$.getScript('/sciezka/do/skryptu/moj.js', function() {
    // Skrypt moj.js został wczytany i teraz można używać
    // wszelkich zdefiniowancyh w nim funkcji i obiektów.
});
```

W podobny sposób druga z metod, `$.getJSON()`, pozwala przesłać na podany adres URI żądanie HTTP `GET`, przetworzyć zwrócony dokumenty JSON i przekazać go do określonej funkcji zwrotnej:

```
// Wczytanie dokumentu JSON przy użyciu żądania HTTP GET
$.getJSON('/przyklad.php', {
    'parametrJSON' : 'wartośćParametru'
}, function(dame, status, jqXHR) {
    console.log(dame.status);
});
```

Po tym krótkim wstępie prezentującym możliwości obsługi żądań AJAX przy użyciu biblioteki jQuery możemy przejść do przedstawienia klienckiej części aplikacji do prowadzenia pogawędek internetowych.

Kliencka część aplikacji do prowadzenia pogawędek

We wcześniejszej części rozdziału zostały już zaprezentowany serwer pogawędek (skrypt *czat.php*, zamieszczony na listingu 23.3); pozostaje zatem jedynie przygotować sensowny interfejs użytkownika aplikacji, który będzie wykorzystywał bibliotekę jQuery zarówno do przesyłania na serwer nowych wiadomości, jak i do pobierania wiadomości z serwera i wyświetlania ich w przeglądarce użytkownika. W pierwszej kolejności zajmiemy się przygotowaniem prostego interfejsu HTML, którego układ określimy przy użyciu popularnego frameworka CSS Bootstrap i w którym zastosujemy specjalnie przygotowany arkusz stylów CSS do wyświetlania poszczególnych wiadomości w formie charakterystycznych „dymków" (patrz rysunek 23.4).

Wyszukane style CSS używane do wyświetlania wymyślnych dymków z wiadomościami zostały przygotowane przy pomocy doskonałego narzędzia o nazwie *Bubbler*, stworzonego przez Johna Clifforda i dostępnego na stronie *http://ilikepixels.co.uk/drop/bubbler/*.

Listing 23.4. *czat.html — interfejs użytkownika aplikacji do prowadzenia pogawędek*

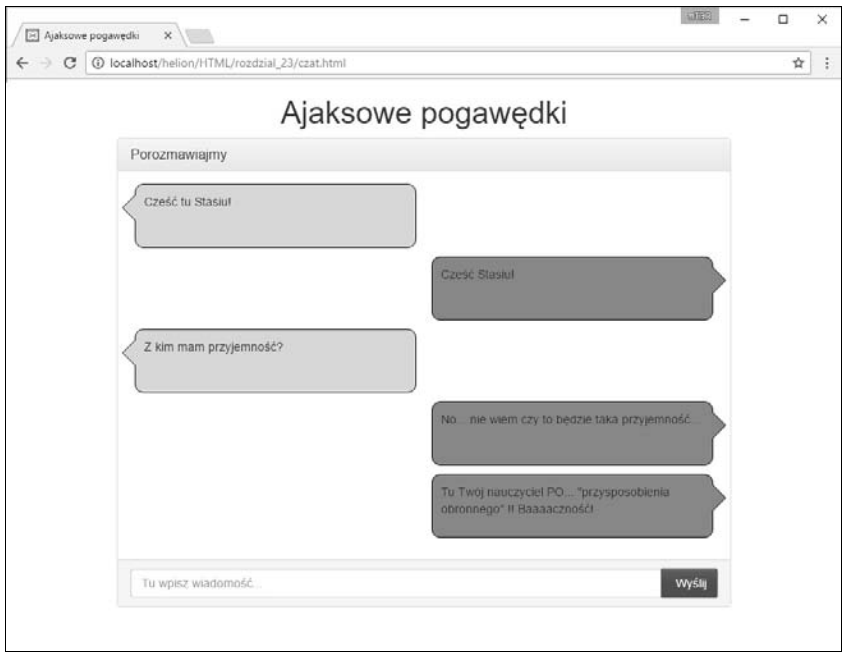
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Ajaksowe pogawędki</title>
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/
    ↪bootstrap.min.css">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/
    ↪bootstrap-theme.min.css">
    <style>
      .bubble-recv
      {
        position: relative;
        width: 330px;
        height: 75px;
        padding: 10px;
        background: #AEE5FF;
        -webkit-border-radius: 10px;
        -moz-border-radius: 10px;
        border-radius: 10px;
        border: #000000 solid 1px;
        margin-bottom: 10px;
      }

      .bubble-recv:after
      {
        content: '';
```

```
  position: absolute;
  border-style: solid;
  border-width: 15px 15px 15px 0;
  border-color: transparent #AEE5FF;
  display: block;
  width: 0;
  z-index: 1;
  left: -15px;
  top: 12px;
}

.bubble-recv:before
{
  content: '';
  position: absolute;
  border-style: solid;
  border-width: 15px 15px 15px 0;
  border-color: transparent #000000;
  display: block;
  width: 0;
  z-index: 0;
  left: -16px;
  top: 12px;
}

.bubble-sent
{
  position: relative;
  width: 330px;
  height: 75px;
  padding: 10px;
  background: #00E500;
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  border-radius: 10px;
  border: #000000 solid 1px;
  margin-bottom: 10px;
}

.bubble-sent:after
{
  content: '';
  position: absolute;
  border-style: solid;
  border-width: 15px 0 15px 15px;
  border-color: transparent #00E500;
  display: block;
  width: 0;
  z-index: 1;
  right: -15px;
  top: 12px;
}

.bubble-sent:before
{
  content: '';
  position: absolute;
  border-style: solid;
  border-width: 15px 0 15px 15px;
  border-color: transparent #000000;
  display: block;
  width: 0;
```

```
      z-index: 0;
      right: -16px;
      top: 12px;
    }

    .spinner {
      display: inline-block;
      opacity: 0;
      width: 0;

      -webkit-transition: opacity 0.25s, width 0.25s;
      -moz-transition: opacity 0.25s, width 0.25s;
      -o-transition: opacity 0.25s, width 0.25s;
      transition: opacity 0.25s, width 0.25s;
    }

    .has-spinner.active {
      cursor:progress;
    }

    .has-spinner.active .spinner {
      opacity: 1;
      width: auto;
    }

    .has-spinner.btn-mini.active .spinner {
      width: 10px;
    }

    .has-spinner.btn-small.active .spinner {
      width: 13px;
    }

    .has-spinner.btn.active .spinner {
      width: 16px;
    }

    .has-spinner.btn-large.active .spinner {
      width: 19px;
    }

    .panel-body {
      padding-right: 35px;
      padding-left: 35px;
    }

  </style>
</head>
<body>
<h1 style="text-align:center">Ajaksowe pogawędki</h1>
<div class="container">
  <div class="panel panel-default">
    <div class="panel-heading">
      <h2 class="panel-title">Porozmawiajmy</h2>
    </div>
    <div class="panel-body" id="panelCzatu">
    </div>
    <div class="panel-footer">
      <div class="input-group">
        <input type="text" class="form-control" id="trescWiadomosci" placeholder="Tu wpisz
        ↪wiadomość..."/>
        <span class="input-group-btn">
```

```
            <button id="wyslijWiadomoscBtn" class="btn btn-primary has-spinner" type="button">
              <span class="spinner"><i class="icon-spin icon-refresh"></i></span>
              Wyślij
            </button>
          </span>
        </div>
      </div>
    </div>
  </div>
  <script src="//code.jquery.com/jquery-3.1.1.min.js"></script>
  <script src="klient.js"></script>
  </body>
</html>
```

Kiedy strona ta zostanie wyświetlona w przeglądarce, utworzy prosty interfejs użytkownika, podobny do tego przedstawionego na rysunku 23.2; oczywiście jego konkretna postać będzie zależeć od prowadzonej konwersacji (warto zwrócić uwagę na to, że bezpośrednio po wyświetleniu strony nie będzie na niej żadnych dymków wiadomości).



Rysunek 23.2. *Ajaksowe pogawędki w akcji*

Aby ożywić statyczny dokument HTML wczytany i wyświetlony w przeglądarce, musimy zaimplementować w języku JavaScript kod, który nawiąże połączenie z serwerem PHP na serwerze i wyświetli przesłane przez niego wiadomości. Wszystkie te operacje są wykonywane przez skrypt JavaScript o nazwie *klient.js*, do którego odwołuje się przedstawiony wcześniej dokument HTML.

Aplikacja JavaScript odpowiada za odpytywanie skryptu PHP w regularnych odstępach czasu w celu pobierania wiadomości oraz za wyświetlanie każdej z nich w interfejsie użytkownika w formie dymku. Skrypt określa także funkcję, która będzie obsługiwać kliknięcia przycisku *Wyślij*, czyli pobierać wiadomość wpisaną w polu tekstowym i wysyłać ją na serwer w celu późniejszego wyświetlenia w interfejsie użytkownika.

Aby umożliwić odpytywanie skryptu PHP, należy skorzystać z mechanizmu JavaScript nazywanego licznikiem czasu. Mechanizm ten opóźnia wykonanie funkcji JavaScript o podany czas, a następnie wywołuje ją, kiedy ten czas upłynie. W przypadku prezentowanego skryptu funkcja ta nosi nazwę `odpytajSerwer` i jest zdefiniowana w następujący sposób:

```
var odpytajSerwer = function() {
  $.get('czat.php', function(wynik) {

    if(!wynik.sukces) {
      console.log("Błąd podczas pobierania nowych wiadomości z serwera!");
      return;
    }

    $.each(wynik.wiadomosci, function(idx) {

      var dymekCzatu;

      if(this.wyslana_przez == 'ja') {
        dymekCzatu = $('<div class="row bubble-sent pull-right">' +
                  this.wiadomosc +
                  '</div><div class="clearfix"></div>');
      } else {
        dymekCzatu = $('<div class="row bubble-recv">' +
                  this.wiadomosc +
                  '</div><div class="clearfix"></div>');
      }

      $('#panelCzatu').append(dymekCzatu);
    });

    setTimeout(odpytajSerwer, 5000);
  });
}
```

Funkcja `odpytajSerwer()` przeprowadza dwie podstawowe operacje: wykonuje asynchroniczne żądanie HTTP `GET` do skryptu na serwerze w celu pobrania nowych wiadomości, a następnie używa funkcji `setTimeout()`, by zaplanować wywołanie samej siebie po upłynięciu pięciu sekund. W trakcie realizacji tej funkcji zostaje wygenerowane i zakończone żądanie HTTP `GET`, jak również wykonane zostaje domknięcie przekazane w wywołaniu metody jQuery `$.get()`. Domknięcie to sprawdza wyniki zwrócone z serwera, a następnie w pętli wyświetla wszystkie komunikaty, dodając je do interfejsu użytkownika przy wykorzystaniu odpowiednich klas CSS.

Funkcja `odpytajSerwer()` musi zostać wywołana pierwszy raz w celu zapoczątkowania cyklu odpytywania serwera, a najlepszym momentem na zrobienie tego jest moment zakończenia wczytywania dokumentu. Z tego względu skrypt określa procedurę obsługi zdarzenia jQuery `ready` i w niej wywołuje funkcję `odpytajSerwer()`. W celu uatrakcyjnienia interfejsu użytkownika skrypt określa także procedurę obsługi zdarzeń `click` wszystkich przycisków na stronie, która odpowiednio dodaje do tych elementów lub usuwa z nich klasę `active`.

```
$(document).ready(function() {
  odpytajSerwer();

  $('button').click(function() {
    $(this).toggleClass('active');
  });
});
```

Ostatnim elementem skryptu *klient.js* jest przedstawiony poniżej fragment służący do przesyłania nowych wiadomości na serwer. Poniższy kod stanowi procedurę obsługi zdarzeń `click`, skojarzoną z przyciskiem umieszczonym na stronie. Funkcja ta odczytuje wpisany komunikat,

a następnie przesyła go na serwer przy użyciu żądania HTTP `POST`, umożliwiając odczytanie jej przez inne klienty biorące udział w pogawędce.

```
$('#wyslijWiadomoscBtn').on('click', function(zdarzenie) {
  zdarzenie.preventDefault();

  var wiadomosc = $('#trescWiadomosci').val();

  $.post('czat.php', {
    'wiadomosc' : wiadomosc
  }, function(wynik) {

    $('#wyslijWiadomoscBtn').toggleClass('active');

    if(!wynik.sukces) {
      alert("Błąd podczas wysyłania wiadomości!");
    } else {
      console.log("Wiadomość została wysłana!");
      $('#trescWiadomosci').val('');
    }
  });

});
```

Umieszczenie tych wszystkich stosunkowo prostych funkcji w jednym pliku JavaScript (w przypadku prezentowanej aplikacji jest to plik *klient.js*) oraz wczytanie go na stronie WWW to wszystko, czego potrzeba do zapewnienia poprawnego działania aplikacji czatu. Choć aplikacja ta nie zapewnia natychmiastowej aktualizacji interfejsu użytkownika (trzeba na nią czekać do pięciu sekund), to jednak pozwala na prowadzenie pogawędek z przyjaciółmi w czasie rzeczywistym, i to bez konieczności każdorazowego odświeżania całej strony.

Jeśli Czytelnik samodzielnie sprawdza prezentowane przykłady i używa przeglądarki Google Chrome, to bez trudu będzie mógł zweryfikować działanie przedstawionej tu aplikacji, otwierając dwa okna przeglądarki (jedno w trybie normalnym, a drugie w trybie anonimowym). W pogawędce może uczestniczyć dowolna liczba przeglądarek, o ile tylko każda z nich będzie używać unikalnego identyfikatora sesji PHP.

Propozycje dalszej lektury

W zamieszczonym w tym rozdziale wprowadzeniu dotyczącym technologii AJAX oraz sposobu korzystania z niej podczas tworzenia aplikacji internetowych jedynie bardzo pobieżnie przedstawiliśmy wszystkie dostępne możliwości tej technologii. Choć zaprezentowane informacje stanowią solidne podstawy do dalszej nauki, to jednak tematyka związana z biblioteką jQuery i jej stosowaniem jest znacznie bardziej obszerna — właściwie można jej poświęcić całą odrębną książkę. Niemniej lektura tego rozdziału powinna umożliwić Czytelnikowi implementację opisywanych technologii we własnych aplikacjach.

Jeśli Czytelnik jest zainteresowany dalszym poznawaniem biblioteki jQery, powinien przeczytać serię artykułów i poradników dostępnych na stronie *http://learn.jquery.com*.

W następnym rozdziale

To już niemal koniec tej części książki. Jednak zanim w ramach jej kolejnej części zajmiemy się projektami, najpierw przedstawimy kilka użytecznych informacji dodatkowych o możliwościach PHP, które nie znalazły się w poprzednich rozdziałach.

Skorowidz

E

F

G

H

I

N

O

P

R

S

Ś

T

U

W

Z

PHP i MySQL — rozwiązanie dla dynamicznych witryn!

Język PHP i serwer bazy danych MySQL to niezwykle popularne narzędzia o otwartym kodzie źródłowym. Wiele świetnych, dynamicznych stron WWW powstało właśnie dzięki połączeniu tych dwóch narzędzi. Ogromne możliwości PHP oraz elastyczność i szybkość MySQL, bezustanne rozwijanie tych technologii, a także niezawodna i chętna do pomocy społeczność sprawiają, że tworzenie profesjonalnych, interaktywnych witryn WWW z wykorzystaniem synergii PHP i MySQL jest pracą przyjemną, efektywną i satysfakcjonującą!

Trzymasz w ręku piąte wydanie poradnika dla projektantów WWW, po który twórcy stron sięgają wyjątkowo często. Książka została poprawiona, zaktualizowana o PHP 7 i rozszerzona o prezentację najnowszych możliwości MySQL. Zawiera przejrzysty opis podstaw PHP oraz konfiguracji i pracy z bazą danych MySQL. Przedstawiono w niej zasady poprawnego projektowania, pisania i wdrażania aplikacji. Uwzględniono też zagadnienia dotyczące bezpieczeństwa i uwierzytelniania użytkowników oraz implementacji takich rozwiązań w rzeczywistych witrynach WWW. Nie zabrakło tu również gruntownego wprowadzenia do zagadnienia integracji części klienckiej i serwerowej aplikacji internetowych za pomocą JavaScriptu.

Najważniejsze zagadnienia przedstawione w książce:

- podstawowe konstrukcje PHP i ich poprawne stosowanie
- projektowanie, tworzenie i utrzymywanie bazy danych SQL jako elementu aplikacji WWW
- zaawansowane techniki PHP i funkcje MySQL
- usuwanie błędów i ich rejestrowanie w pliku dziennika
- korzystanie z frameworka Laravel
- integracja aplikacji z mediami społecznościowymi

Laura Thomson — jest dyrektorem ds. technicznych w Mozilla Corporation. Wcześniej była prezesem firm OmniTI i Tangled Web Design. W wolnych chwilach jeździ konno lub dyskutuje o idei wolnego oprogramowania.

Luke Welling — jest architektem oprogramowania i bywalcem konferencji poświęconych programowaniu aplikacji internetowych. Jakiś czas temu wykładał inżynierię oprogramowania na Uniwersytecie w Melbourne w Australii. W wolnym czasie zajmuje się propagowaniem zalet wolnego oprogramowania.

Helion
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Informatyka w najlepszym wydaniu

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

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

ISBN 978-83-283-3256-0

cena: 109,00 zł

sięgnij po WIĘCEJ
KOD KORZYŚCI

Addison Wesley

ALWAYS LEARNING
PEARSON