

Luke Welling
Laura Thomson



PHP i MySQL

Tworzenie stron WWW
Vademecum profesjonalisty

Wydanie V

Rozwiązanie dla niezawodnych, dynamicznych witryn!



Helion

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.
Copyright © 2017 by Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education Inc.
Polish language edition published by HELION S.A. Copyright © 2017.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

	O autorach	19
	O współautorach	19
	Wprowadzenie	21
Część I	Stosowanie PHP	31
Rozdział 1.	Podstawowy kurs PHP	33
	Zastosowanie PHP	34
	Tworzenie przykładowej aplikacji: „Części samochodowe Janka”	34
	Formularz zamówienia	34
	Przetwarzanie formularza	36
	Osadzanie PHP w HTML	36
	Zastosowanie znaczników PHP	37
	Instrukcje PHP	38
	Odstępy	38
	Komentarze	39
	Dodawanie zawartości dynamicznej	39
	Wywoływanie funkcji	40
	Używanie funkcji date()	40
	Dostęp do zmiennych formularza	41
	Zmienne formularza	41
	Łączenie łańcuchów znaków	43
	Zmienne i łańcuchy znaków	43
	Identyfikatory	44
	Typy zmiennych	44
	Typy danych w PHP	45
	Siła typu	45
	Rzutowanie typu	46
	Zmienne zmiennych	46
	Deklarowanie i używanie stałych	46
	Zasięg zmiennych	47
	Używanie operatorów	48
	Operatory arytmetyczne	48
	Operatory łańcuchowe	49
	Operatory przypisania	49
	Operatory porównań	52
	Operatory logiczne	52
	Operatory bitowe	53
	Pozostałe operatory	53

Obliczanie sum w formularzu	56
Pierwszeństwo i kolejność	57
Funkcje zarządzania zmiennymi	59
Sprawdzanie i ustawianie typów zmiennych	59
Sprawdzanie stanu zmiennej	60
Reinterpretacja zmiennych	61
Podejmowanie decyzji za pomocą instrukcji warunkowych	61
Instrukcja if	61
Bloki kodu	62
Instrukcja else	62
Instrukcja elseif	63
Instrukcja switch	63
Porównanie różnych instrukcji warunkowych	65
Powtarzanie działań przy użyciu iteracji	65
Pętle while	66
Pętle for i foreach	67
Pętle do..while	68
Wyłamywanie się ze struktury skryptu	69
Używanie alternatywnych składni struktur sterujących	69
Używanie struktury declare	70
W następnym rozdziale	70
Rozdział 2. Przechowywanie i wyszukiwanie danych	71
Zapisywanie danych do późniejszego użycia	71
Przechowywanie i wyszukiwanie zamówień Janka	72
Przetwarzanie plików	72
Otwieranie pliku	73
Tryby otwarcia pliku	73
Stosowanie funkcji fopen() do otwarcia pliku	73
Otwieranie pliku przez protokół FTP lub HTTP	75
Problemy z otwieraniem plików	76
Zapisywanie danych w pliku	77
Parametry funkcji fwrite()	78
Formaty plików	79
Zamykanie pliku	79
Odczyt z pliku	81
Otwieranie pliku w celu odczytu — fopen()	82
Wiedzieć, kiedy przestać — feof()	82
Odczytywanie pliku wiersz po wierszu — fgets(), fgets() i fgetsv()	83
Odczyt całego pliku — readfile(), fpassthru(), file() i file_get_contents()	83
Odczyt pojedynczego znaku — fgetc()	84
Odczytywanie zadanej długości — fread()	85
Inne funkcje plikowe	85
Sprawdzanie istnienia pliku — file_exists()	85
Określanie wielkości pliku — filesize()	85
Kasowanie pliku — unlink()	86
Poruszanie się wewnątrz pliku — rewind(), fseek() i ftell()	86
Blokowanie pliku	87
Lepszy sposób obróbki danych — bazy danych	88
Problemy związane ze stosowaniem plików jednorodnych	88
Jak RDBMS rozwiązują powyższe problemy?	89
Propozycje dalszych lektur	89
W następnym rozdziale	89

Rozdział 3.	Stosowanie tablic	91
	Czym są tablice?	91
	Tablice indeksowane numerycznie	92
	Inicjowanie tablic indeksowanych numerycznie	92
	Dostęp do zawartości tablicy	93
	Dostęp do tablic przy zastosowaniu pętli	94
	Tablice z innymi indeksami	94
	Inicjowanie tablicy	95
	Dostęp do elementów tablicy	95
	Stosowanie pętli	95
	Operatory tablicowe	96
	Tablice wielowymiarowe	97
	Sortowanie tablic	100
	Stosowanie funkcji sort()	100
	Stosowanie funkcji asort() i ksort() do porządkowania tablic	101
	Sortowanie odwrotne	102
	Sortowanie tablic wielowymiarowych	102
	Zastosowanie funkcji array_multisort()	102
	Typy sortowań definiowane przez użytkownika	103
	Odwrotne sortowanie zdefiniowane przez użytkownika	104
	Zmiany kolejności elementów w tablicach	105
	Stosowanie funkcji shuffle()	105
	Odwracanie kolejności elementów w tablicy	106
	Wczytywanie tablic z plików	107
	Wykonywanie innych działań na tablicach	110
	Poruszanie się wewnątrz tablicy	
	— funkcje each(), current(), reset(), end(), next(), pos() i prev()	110
	Dołączanie dowolnej funkcji do każdego elementu tablicy	
	— funkcja array_walk()	111
	Liczenie elementów tablicy: count(), sizeof() i array_count_values()	112
	Konwersja tablic na zmienne skalarne — funkcja extract()	112
	Propozycje dalszych lektur	114
	W następnym rozdziale	114
Rozdział 4.	Manipulowanie łańcuchami znaków i wyrażenia regularne	115
	Przykładowa aplikacja — Inteligentny Formularz Pocztowy	115
	Formatowanie łańcuchów znaków	117
	Przycinanie łańcuchów — funkcje chop(), ltrim() i trim()	118
	Formatowanie wyjściowych łańcuchów znaków	118
	Łączenie i rozdzielanie łańcuchów znaków za pomocą funkcji łańcuchowych	125
	Stosowanie funkcji explode(), implode() i join()	125
	Stosowanie funkcji strtok()	126
	Stosowanie funkcji substr()	127
	Porównywanie łańcuchów znaków	127
	Porządkowanie łańcuchów znaków	
	— funkcje strcmp(), strcasecmp() i strnatcmp()	128
	Sprawdzanie długości łańcucha znaków za pomocą funkcji strlen()	128
	Dopasowywanie i zamiana łańcuchów znaków za pomocą funkcji łańcuchowych	129
	Znajdowanie fragmentów w łańcuchach znaków	
	— funkcje strpos(), strchr(), strrchr() i strstr()	129
	Odnajdywanie pozycji fragmentu łańcucha — funkcje strpos() i strrpos()	130
	Zamiana fragmentów łańcucha znaków — funkcje str_replace() i substr_replace()	131
	Wprowadzenie do wyrażeń regularnych	132
	Podstawy	132
	Ograniczniki	132

Zbiory i klasy znaków	133
Powtarzalność	134
Podwyrażenia	135
Podwyrażenia policzalne	135
Kotwiczenie na początku lub na końcu łańcucha znaków	135
Rozgałęzianie	135
Dopasowywanie specjalnych znaków literowych	136
Podsumowanie metaznaków	136
Sekwencje specjalne	137
Odwołania wsteczne	138
Asercje	138
Wykorzystanie wszystkich zdobytych informacji — inteligentny formularz	139
Odnajdywanie fragmentów łańcuchów za pomocą wyrażeń regularnych	140
Zamiana fragmentów łańcuchów za pomocą wyrażeń regularnych	141
Rozdzielanie łańcuchów za pomocą wyrażeń regularnych	141
Propozycje dalszych lektur	142
W następnym rozdziale	142
Rozdział 5. Ponowne wykorzystanie kodu i tworzenie funkcji	143
Zalety ponownego stosowania kodu	143
Koszt	144
Niezawodność	144
Spójność	144
Stosowanie funkcji require() i include()	144
Stosowanie funkcji require() do dołączania kodu	145
Stosowanie require() w szablonach stron WWW	146
Stosowanie opcji auto_prepend_file i auto_append_file	150
Stosowanie funkcji w PHP	151
Wywoływanie funkcji	151
Wywołanie niezdefiniowanej funkcji	153
Wielkość liter a nazwy funkcji	154
Definiowanie własnych funkcji	154
Podstawowa struktura funkcji	154
Nadawanie nazwy funkcji	155
Parametry	156
Zasięg	158
Przekazanie przez referencję czy przekazanie przez wartość?	160
Stosowanie słowa kluczowego return	161
Zwracanie wartości przez funkcje	162
Implementacja rekurencji	163
Implementacja funkcji anonimowych (lub domknięć)	165
Propozycje dalszych lektur	166
W następnym rozdziale	166
Rozdział 6. Obiektowy PHP	167
Koncepcje programowania obiektowego	167
Klasy i obiekty	168
Polimorfizm	169
Dziedziczenie	169
Tworzenie klas, atrybutów i operacji w PHP	170
Struktura klasy	170
Konstruktory	170
Destruktry	171
Tworzenie egzemplarzy	171
Stosowanie atrybutów klasy	172
Wywoływanie operacji klas	172

Kontrola dostępu przy użyciu modyfikatorów <code>private</code> i <code>public</code>	173
Pisanie funkcji dostępowych	174
Implementacja dziedziczenia w PHP	175
Kontrolowanie widoczności w trakcie dziedziczenia	
przy użyciu <code>private</code> i <code>protected</code>	176
Przesłanianie	177
Zapobieganie dziedziczeniu i przesłanianiu przy użyciu słowa kluczowego <code>final</code>	178
Wielokrotne dziedziczenie	179
Implementowanie interfejsów	180
Cechy	180
Projektowanie klas	182
Tworzenie kodu dla własnej klasy	183
Zaawansowane mechanizmy obiektowe w PHP	189
Używanie stałych klasowych	189
Implementowanie metod statycznych	190
Sprawdzanie typu klasy i wskazywanie typu	190
Późne wiązania statyczne	191
Klonowanie obiektów	191
Używanie klas abstrakcyjnych	192
Przeciążanie metod przy użyciu <code>__call()</code>	192
Używanie metody <code>__autoload()</code>	193
Implementowanie iteratorów i iteracji	194
Generatory	195
Przekształcanie klas w łańcuchy znaków	197
Używanie API Reflection	197
Przestrzenie nazw	198
Stosowanie podprzestrzeni nazw	200
Prezentacja globalnej przestrzeni nazw	200
Importowanie przestrzeni nazw oraz określanie ich nazw zastępczych	200
W następnym rozdziale	201
Rozdział 7. Obsługa błędów i wyjątków	203
Koncepcja obsługi wyjątków	203
Klasa <code>Exception</code>	205
Wyjątki definiowane przez użytkownika	206
Wyjątki w Częściach samochodowych Janka	207
Wyjątki i inne mechanizmy obsługi błędów w PHP	210
Propozycje dalszych lektur	211
W następnym rozdziale	211
Część II Stosowanie MySQL	213
Rozdział 8. Projektowanie internetowej bazy danych	215
Koncepcje relacyjnych baz danych	216
Tabele	216
Kolumny	216
Wiersze	216
Wartości	217
Klucze	217
Schematy	218
Relacje	218
Jak zaprojektować internetową bazę danych?	219
Określ obiekty świata realnego, których model chcesz wykonać	219
Unikaj przechowywania redundantnych danych	220
Zapisuj atomowe wartości kolumn	221

	Dobierz właściwe klucze	222
	Pomyśl o zapytaniach, które zadasz bazie	222
	Unikaj tworzenia tabel z wieloma pustymi polami	222
	Typy tabel — podsumowanie	223
	Architektura internetowej bazy danych	224
	Propozycje dalszych lektur	225
	W następnym rozdziale	225
Rozdział 9.	Tworzenie internetowej bazy danych	227
	Użytkowanie monitora MySQL	228
	Logowanie się do serwera MySQL	229
	Tworzenie baz i rejestrowanie użytkowników	230
	Definiowanie użytkowników i przywilejów	230
	Wprowadzenie do systemu przywilejów MySQL	231
	Zasada najmniejszego przywileju	231
	Rejestrowanie użytkowników: polecenia CREATE USER oraz GRANT	231
	Typy i poziomy przywilejów	233
	Polecenie REVOKE	236
	Przykłady użycia poleceń GRANT i REVOKE	236
	Rejestrowanie użytkownika łączącego się z internetu	237
	Używanie odpowiedniej bazy danych	238
	Tworzenie tabel bazy danych	238
	Znaczenie dodatkowych atrybutów kolumn	240
	Typy kolumn	241
	Rzut oka na bazę danych — polecenia SHOW i DESCRIBE	243
	Tworzenie indeksów	243
	Identyfikatory MySQL	244
	Wybór typów danych w kolumnach	246
	Typy liczbowe	246
	Propozycje dalszych lektur	251
	W następnym rozdziale	251
Rozdział 10.	Praca z bazą danych MySQL	253
	Czym jest SQL?	253
	Zapisywanie danych do bazy	254
	Wyszukiwanie danych w bazie	256
	Wyszukiwanie danych spełniających określone kryteria	257
	Wyszukiwanie danych w wielu tabelach	259
	Szeregowanie danych w określonym porządku	264
	Grupowanie i agregowanie danych	264
	Wskazanie wierszy, które mają być wyświetlone	266
	Używanie podzapytań	267
	Dokonywanie zmian rekordów w bazie danych	269
	Zmiana struktury istniejących tabel	269
	Usuwanie rekordów z bazy danych	271
	Usuwanie tabel	272
	Usuwanie całych baz danych	272
	Propozycje dalszych lektur	272
	W następnym rozdziale	272
Rozdział 11.	Łączenie się z bazą MySQL za pomocą PHP	273
	Jak działa internetowa baza danych?	273
	Wykonywanie zapytań do bazy danych z poziomu strony WWW	276
	Sprawdzenie poprawności wpisanych danych	277
	Ustanawianie połączenia z bazą danych	278
	Wybór używanej bazy danych	279

Wysyłanie zapytań do bazy danych	279
Stosowanie poleceń przygotowanych	280
Odczytywanie rezultatów zapytań	281
Zamykanie połączenia z bazą danych	282
Wstawianie nowych danych do bazy	283
Używanie innych interfejsów bazodanowych PHP	286
Stosowanie ogólnego interfejsu dostępu do baz danych: PDO	286
Propozycje dalszych lektur	289
W następnym rozdziale	289
Rozdział 12. Administrowanie MySQL dla zaawansowanych	291
Szczegóły systemu przywilejów	291
Tabela user	293
Tabela db	295
Tabele tables_priv, columns_priv iprocs_priv	296
Kontrola dostępu: w jaki sposób MySQL używa tabel przywilejów	297
Zmiana przywilejów: kiedy zmiany zostaną uwzględnione?	298
Ochrona bazy danych	298
MySQL z perspektywy systemu operacyjnego	298
Hasła	299
Przywileje użytkowników	299
MySQL i internet	300
Uzyskiwanie szczegółowych informacji o bazie danych	301
Uzyskiwanie informacji poleceniem SHOW	301
Uzyskiwanie informacji o kolumnach za pomocą polecenia DESCRIBE	303
Jak wykonywane są zapytania: polecenie EXPLAIN	303
Optymalizowanie bazy danych	308
Optymalizacja projektu bazy danych	308
Przywileje	308
Optymalizacja tabel	308
Stosowanie indeksów	308
Używanie wartości domyślnych	309
Więcej wskazówek	309
Tworzenie kopii zapasowej bazy danych MySQL	309
Przywracanie bazy danych MySQL	310
Implementowanie replikacji	310
Konfigurowanie serwera nadrzędnego	311
Transfer danych początkowych	311
Konfigurowanie odbiorcy lub odbiorców	312
Propozycje dalszych lektur	312
W następnym rozdziale	312
Rozdział 13. Zaawansowane programowanie w MySQL	313
Instrukcja LOAD DATA INFILE	313
Mechanizmy składowania danych	314
Transakcje	315
Definicje dotyczące transakcji	315
Użycie transakcji w InnoDB	316
Klucze obce	317
Procedury składowane	318
Prosty przykład	318
Zmienne lokalne	321
Kursory i struktury sterujące	321
Wyzwalacze	324
Propozycje dalszych lektur	326
W następnym rozdziale	326

Część III	E-commerce i bezpieczeństwo	327
Rozdział 14.	Zagrożenia bezpieczeństwa aplikacji internetowych	329
	Identyfikacja zagrożeń	329
	Dostęp do wrażliwych danych	329
	Modyfikacje danych	332
	Utrata lub zniszczenie danych	332
	Blokada usługi	333
	Wstrzykiwanie złośliwego kodu	335
	Złamanie zabezpieczeń dostępu do serwera	336
	Zaprzeczenie korzystania z usługi	336
	Identyfikacja użytkowników	337
	Napastnicy i crackerzy	337
	Nieświadomi użytkownicy zainfekowanych komputerów	338
	Rozczarowani pracownicy	338
	Złodzieje sprzętu komputerowego	338
	My sami	338
	W następnym rozdziale	339
Rozdział 15.	Tworzenie bezpiecznych aplikacji internetowych	341
	Strategie zapewniania bezpieczeństwa	341
	Planowanie z wyprzedzeniem	342
	Równowaga między bezpieczeństwem i użytecznością	342
	Monitorowanie bezpieczeństwa	343
	Ogólne podejście do bezpieczeństwa	343
	Zabezpieczanie kodu źródłowego	343
	Filtrowanie danych pochodzących od użytkowników	343
	Unieważnianie danych wynikowych	347
	Organizacja kodu źródłowego	349
	Zawartość kodu źródłowego	350
	Zagadnienia dotyczące systemu plików	351
	Stabilność kodu i błędy	352
	Wykonywanie poleceń	352
	Zabezpieczanie serwera WWW oraz PHP	354
	Regularne uaktualnianie oprogramowania	354
	Analiza ustawień w pliku php.ini	355
	Konfiguracja serwera WWW	355
	Aplikacje internetowe	
	działające na współużytkowanych serwerach hostingowych	356
	Bezpieczeństwo serwera bazy danych	357
	Użytkownicy i system uprawnień	357
	Wysyłanie danych do serwera	358
	Łączenie się z serwerem	359
	Praca serwera	359
	Zabezpieczanie sieci	359
	Zapory sieciowe	360
	Wykorzystanie strefy zdemilitaryzowanej	360
	Przygotowanie na ataki DoS i DDoS	361
	Bezpieczeństwo komputerów i systemów operacyjnych	361
	Uaktualnianie systemu operacyjnego	361
	Udostępnianie tylko niezbędnych usług	362
	Fizyczne zabezpieczenie serwera	362
	Planowanie działań na wypadek awarii	362
	W następnym rozdziale	364

Rozdział 16. Implementacja metod uwierzytelniania przy użyciu PHP	365
Identyfikacja użytkowników	365
Implementacja kontroli dostępu	366
Przechowywanie haseł dostępu	368
Zabezpieczanie haseł	369
Zabezpieczanie więcej niż jednej strony	370
Podstawowa metoda uwierzytelniania	371
Wykorzystanie podstawowej metody uwierzytelniania w PHP	371
Wykorzystanie podstawowej metody uwierzytelniania na serwerze Apache przy użyciu plików .htaccess	373
Implementacja własnej metody uwierzytelniania	376
Propozycje dalszych lektur	376
W następnym rozdziale	376
 Część IV Zaawansowane techniki PHP	377
Rozdział 17. Interakcja z systemem plików i serwerem	379
Wprowadzenie do wysyłania plików	379
Kod HTML służący do wysyłania plików	380
Tworzenie kodu PHP obsługującego plik	381
Śledzenie postępów przesyłania plików	385
Najczęściej spotykane problemy	387
Stosowanie funkcji katalogowych	388
Odczyt z katalogów	388
Trzymywanie informacji na temat aktualnego katalogu	391
Tworzenie i usuwanie katalogów	391
Interakcja z systemem plików	392
Pobieranie informacji o pliku	392
Zmiana właściwości pliku	394
Tworzenie, usuwanie i przenoszenie plików	395
Stosowanie funkcji uruchamiających programy	395
Interakcja ze środowiskiem: funkcje getenv() i putenv()	398
Propozycje dalszych lektur	398
W następnym rozdziale	398
 Rozdział 18. Stosowanie funkcji sieci i protokołu	399
Przegląd protokołów	399
Wysyłanie i odczytywanie poczty elektronicznej	400
Korzystanie z danych z innych witryn WWW	400
Stosowanie funkcji połączeń sieciowych	403
Tworzenie kopii bezpieczeństwa lub kopii lustrzanej pliku	407
Stosowanie FTP w celu utworzenia kopii bezpieczeństwa lub kopii lustrzanej pliku	407
Wysyłanie plików	413
Unikanie przekroczenia dopuszczalnego czasu	413
Stosowanie innych funkcji FTP	414
Propozycje dalszych lektur	414
W następnym rozdziale	414
 Rozdział 19. Zarządzanie datą i czasem	415
Uzyskiwanie informacji o dacie i czasie w PHP	415
Strefy czasowe	415
Stosowanie funkcji date()	416
Obsługa znaczników czasu Uniksa	417
Stosowanie funkcji getdate()	419

	Sprawdzanie poprawności dat przy użyciu funkcji <code>checkdate()</code>	420
	Formatowanie znaczników czasu	420
	Konwersja pomiędzy formatami daty PHP i MySQL	422
	Obliczanie dat w PHP	424
	Obliczanie dat w MySQL	425
	Stosowanie mikrosekund	426
	Stosowanie funkcji kalendarzowych	426
	Propozycje dalszych lektur	427
	W następnym rozdziale	427
Rozdział 20.	Umiejdzynarodawianie i lokalizowanie	429
	Lokalizacja to nie tylko tłumaczenie	429
	Zbiory znaków	430
	Zbiory znaków i ich związki z bezpieczeństwem	431
	Stosowanie wielobajtowych funkcji łańcuchowych w PHP	432
	Tworzenie struktury strony przystosowanej do lokalizacji	432
	Zastosowanie funkcji <code>gettext()</code> w umiejdzynarodowionej aplikacji	435
	Konfiguracja systemu w celu wykorzystania funkcji <code>gettext()</code>	436
	Tworzenie plików z tłumaczeniami	437
	Implementacja zlokalizowanych treści w PHP z użyciem funkcji <code>gettext()</code>	438
	Propozycje dalszej lektury	439
	W następnym rozdziale	440
Rozdział 21.	Generowanie obrazków	441
	Konfigurowanie obsługi obrazków w PHP	441
	Formaty obrazków	442
	JPEG	442
	PNG	442
	GIF	443
	Tworzenie obrazków	443
	Tworzenie kadru obrazka	444
	Rysowanie lub umieszczanie tekstu w obrazku	444
	Wyświetlanie ostatecznej grafiki	446
	Końcowe czynności porządkujące	447
	Stosowanie automatycznie generowanych obrazków na innych stronach	447
	Stosowanie tekstu i czcionek do tworzenia obrazków	448
	Konfiguracja podstawowego kadru	451
	Dopasowanie tekstu do przycisku	452
	Nadawanie tekstowi odpowiedniej pozycji	454
	Wpisywanie tekstu do przycisku	455
	Etap końcowy	455
	Rysowanie figur i wykresów danych	455
	Inne funkcje obrazków	462
	W następnym rozdziale	462
Rozdział 22.	Stosowanie kontroli sesji w PHP	463
	Czym jest kontrola sesji?	463
	Podstawowa zasada działania sesji	463
	Czym jest cookie?	464
	Konfiguracja cookies w PHP	464
	Stosowanie cookies w sesji	465
	Przechowywanie identyfikatora sesji	465
	Implementacja prostych sesji	466
	Rozpoczynanie sesji	466
	Zgłaszanie zmiennych sesyjnych	466

	Stosowanie zmiennych sesyjnych	467
	Usuwanie zmiennych i niszczenie sesji	467
	Przykład prostej sesji	467
	Konfiguracja kontroli sesji	469
	Implementacja uwierzytelniania w kontroli sesji	470
	W następnym rozdziale	476
Rozdział 23.	Integracja JavaScriptu i PHP	477
	Przedstawienie technologii AJAX	477
	Krótką prezentacją jQuery	478
	Stosowanie jQuery w aplikacjach internetowych	478
	Podstawowe pojęcia i techniki związane ze stosowaniem jQuery	479
	Stosowanie selektorów jQuery	479
	Stosowanie jQuery, technologii AJAX i skryptów PHP	487
	Ajaksowe pogawędki — skrypt serwera	487
	Metody jQuery służące do korzystania z technologii AJAX	490
	Kliencka część aplikacji do prowadzenia pogawędek	493
	Propozycje dalszej lektury	498
	W następnym rozdziale	498
Rozdział 24.	Inne przydatne własności	499
	Przetwarzanie łańcuchów znaków — funkcja eval()	499
	Zakończenie wykonania — die i exit	500
	Serializacja zmiennych i obiektów	500
	Pobieranie informacji na temat środowiska PHP	501
	Uzyskiwanie informacji na temat załadowanych rozszerzeń	502
	Identyfikacja właściciela skryptu	502
	Uzyskiwanie informacji na temat daty modyfikacji skryptu	502
	Czasowa zmiana środowiska wykonawczego	503
	Podświetlanie źródeł	504
	Używanie PHP w wierszu poleceń	505
	W następnej części	506
Część V	Tworzenie praktycznych projektów PHP i MySQL	507
Rozdział 25.	Stosowanie PHP i MySQL w dużych projektach	509
	Zastosowanie inżynierii oprogramowania w tworzeniu aplikacji WWW	510
	Planowanie i prowadzenie projektu aplikacji WWW	510
	Ponowne stosowanie kodu	511
	Tworzenie kodu łatwego w utrzymaniu	512
	Standardy kodowania	512
	Dzielenie kodu	515
	Stosowanie standardowej struktury katalogów	516
	Dokumentacja i dzielenie wewnętrznych funkcji	516
	Implementacja kontroli wersji	516
	Wybór środowiska programistycznego	517
	Dokumentacja projektów	517
	Prototypowanie	518
	Oddzielanie logiki i zawartości	519
	Optymalizacja kodu	519
	Stosowanie prostych optymalizacji	520
	Testowanie	520
	Propozycje dalszych lektur	521
	W następnym rozdziale	521

Rozdział 26. Usuwanie i rejestracja błędów	523
Błędy programistyczne	523
Błędy składni	523
Błędy wykonania	524
Błędy logiczne	529
Pomoc w usuwaniu błędów w zmiennych	530
Pozioomy zgłaszania błędów	532
Zmiana ustawień zgłaszania błędów	534
Wyzwalanie własnych błędów	535
Eleganckie rejestrowanie błędów	536
Rejestrowanie błędów w pliku dziennika	538
W następnym rozdziale	538
Rozdział 27. Tworzenie uwierzytelniania użytkowników i personalizacji	539
Składniki rozwiązania	539
Identyfikacja użytkownika i personalizacja	540
Przechowywanie zakładek	540
Rekomendowanie zakładek	541
Przegląd rozwiązania	541
Implementacja bazy danych	543
Implementacja podstawowej witryny	544
Implementacja uwierzytelniania użytkowników	546
Rejestracja użytkowników	546
Logowanie	551
Wylogowanie	554
Zmiana hasła	555
Ustawianie zapomnianych haseł	557
Implementacja przechowywania i odczytywania zakładek	561
Dodawanie zakładek	561
Wyświetlanie zakładek	563
Usuwanie zakładek	564
Implementacja rekomendacji	566
Rozwijanie projektu i możliwe rozszerzenia	568
Rozdział 28. Tworzenie internetowego klienta poczty elektronicznej z użyciem Laravela	571
Prezentacja frameworka Laravel 5	571
Tworzenie nowego projektu Laravel	571
Struktura aplikacji Laravel	572
Cykl obsługi żądań i wzorzec MVC Laravela	574
Klasy modelu, widok i kontroler frameworka Laravel	575
Rozdział 29. Tworzenie internetowego klienta poczty elektronicznej z użyciem Laravela — część 2.	591
Tworzenie prostego klienta IMAP przy użyciu Laravela	591
Funkcje IMAP udostępniane przez PHP	591
Opakowywanie funkcji IMAP na potrzeby aplikacji Laravel	599
Łączenie wszystkich elementów	
w celu implementacji internetowego klienta poczty elektronicznej	615
Implementacja klasy ImapServiceProvider	616
Strona uwierzytelniania aplikacji klienckiej	617
Implementacja głównego widoku aplikacji	621
Implementacja usuwania i wysyłania wiadomości	629
Wnioski	634

Rozdział 30.	Integracja z mediami społecznościowymi	
	— udostępnianie i uwierzytelnianie	635
	OAuth — internetowa usługa uwierzytelniająca	635
	Przydziały typu kod autoryzacji	637
	Przydziały niejawne	638
	Implementacja internetowego klienta Instagrama	639
	Oznaczanie zdjęć jako lubianych	646
	Wniosek	647
Rozdział 31.	Tworzenie koszyka na zakupy	649
	Składniki rozwiązania	649
	Tworzenie katalogu online	650
	Śledzenie zakupów użytkownika podczas przeglądania	650
	Implementacja systemu płatności	650
	Interfejs administratora	651
	Przegląd rozwiązania	651
	Implementacja bazy danych	654
	Implementacja katalogu online	656
	Przedstawianie kategorii	658
	Wyświetlanie książek danej kategorii	660
	Przedstawianie szczegółowych danych książki	661
	Implementacja koszyka na zakupy	662
	Stosowanie skryptu pokaz_kosz.php	663
	Podgląd koszyka	665
	Dodawanie produktów do koszyka	667
	Zapisywanie uaktualnionego koszyka	669
	Wyświetlanie podsumowania w pasku nagłówka	669
	Pobyty w kasie	670
	Implementacja płatności	675
	Implementacja interfejsu administratora	676
	Rozwijanie projektu	682
	Dodatki	685
Dodatek A	Instalacja Apache, PHP i MySQL	687
	Instalacja Apache, PHP i MySQL w systemie UNIX	688
	Instalacja przy użyciu binariów	688
	Instalacja przy użyciu kodów źródłowych	689
	Podstawowe zmiany w konfiguracji serwera Apache	695
	Czy obsługa PHP działa poprawnie?	696
	Czy SSL działa poprawnie?	697
	Instalacja Apache, PHP i MySQL w systemie Windows	698
	Instalowanie PEAR	700
	Instalowanie PHP z innymi serwerami	700
	Skorowidz	701

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 żadaniami 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 modułarny, 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.

Krótką 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 src="//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` (określonej 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 tę 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 ciałoDokumentu = $('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 założmy, 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 klikniętym przycisku, można wykonać w następujący sposób:

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

    // Wykonanie czynności na klikniętym 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. <i>input focus</i>), 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 <div> 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:

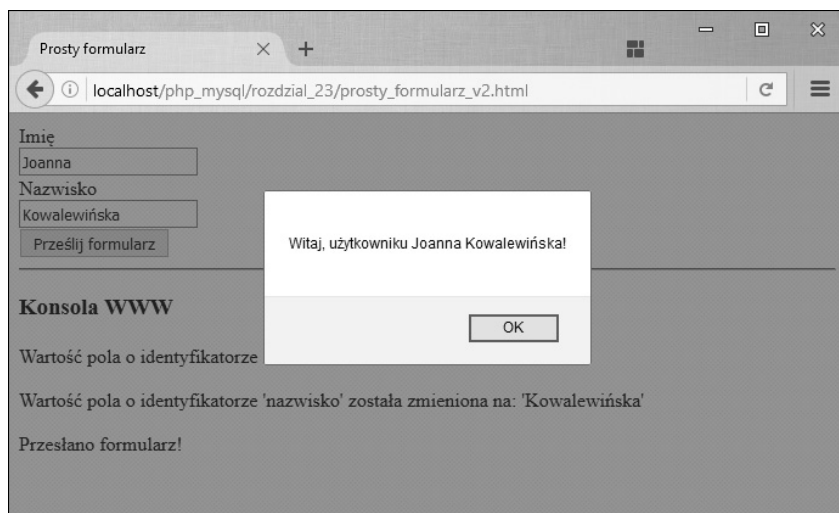
```

$('#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 obsługiwanych 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 stworzona 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 tabeli 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') ?
        'wyslaj' : '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 `application/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 `bład`, 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 przejrzaniu 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.jquery.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(dane);
  }
});
```

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(dane);
  },
  '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(dane, status, jqXHR) {
    console.log(dane);
  }
});
```

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 `$.get()` w jej wywołaniu przekazywane są adres URL żądanego zasobu, wszelkie parametry żądania (w formie zwyczajnego obiektu JavaScript) oraz funkcja zwrotna, którą należy wykonać po pomyślnym zakończeniu żądania. Jeśli podczas obsługi żądania wystąpią jakies błędy, metoda `$.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 zdefiniowanych 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(dane, status, jqXHR) {
  console.log(dane.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>
<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 zależała 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 upływie 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 jQuery, 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

A

- agregowanie danych, 264
- AJAX, 477, 487
- aliasy, 262
- analiza ustawień, 355
- Apache, 373, 687
- Apache HTTP Server, 355
- API Reflection, 197
- aplikacje WWW, 510
- architektura internetowej bazy danych, 224
- asercje, 138
- asynchroniczne żądania, 477
- atak
 - DDoS, 361
 - DoS, 361
- atomowe wartości kolumn, 221
- atomowość, 315
- atrybuty, 170
 - klasy, 172
 - kolumn, 240
- automatyczne generowanie obrazków, 447
- autoryzacja, 635
 - OAuth, 642
- awaria, 362

B

- baza danych, 88, 215
 - MySQL, 253
- bezpieczeństwo, 327
 - aplikacji, 329
 - komputerów, 361
 - serwera bazy danych, 357
 - serwera WWW, 354
 - sieci, 359
 - systemów operacyjnych, 361
 - aplikacji, 341
- biblioteka
 - GD, 442
 - jQuery, 478, 492

- blokada
 - usługi, 333
 - pliku, 87
- błędy, 203, 352, 523
 - logiczne, 529
 - programistyczne, 523
 - składni, 523
 - w zmiennych, 530
 - wykonania, 524

C

- cechy, 180
- cookie, 464
- CSS, Cascading Style Sheets, 519
- czcionki, 448

D

- dane wrażliwe, 329
- data i czas, 415
- DDL, Data Definition Language, 254
- definiowanie
 - użytkowników, 230
 - własnych funkcji, 154
- dekrementacja, 51
- destrukторы, 171
- DMZ, demilitarized zone, 360
- dodawanie
 - zakładek, 561
 - zawartości dynamicznej, 39
- dokumentacja projektów, 517
- dołączanie kodu, 145
- domknięcia, 165
- dopasowanie łańcuchów znaków, 129, 452
- dostawcy usług, 574
- dostęp
 - do danych żądania, 580
 - do elementów tablicy, 95
 - do serwera, 336
 - do tablic, 94
 - do zawartości tablicy, 93
 - do zmiennych formularza, 41

działanie bazy danych, 273
dziedziczenie, 169, 175
dzielenie kodu, 515

E

e-commerce, 327
Eloquent, 588
e-mail, 611

F

filtrowanie
 danych, 343
 łańcuchów znaków, 118
 wartości, 346
fizyczne zabezpieczenie serwera, 362
format
 GIF, 443
 JPEG, 442
 PNG, 442
formatowanie
 HTML, 121
 łańcuchów znaków, 117, 122
formaty
 daty, 422
 obrazków, 442
 plików, 79
formularz
 komentarzy, 116
 obliczanie sum, 56
 zamówienia, 34
framework
 jQuery, 478
 Laravel 5, 571
funkcja
 array_count_values(), 112
 array_multisort(), 102
 array_reverse(), 106
 array_walk(), 111
 asort(), 101
 checkdate(), 420
 chop(), 118
 count(), 112
 current(), 110
 date(), 40, 416
 DATE_FORMAT(), 422
 die(), 500
 each(), 110
 end(), 110
 eval(), 499
 exit(), 500
 explode(), 125
 extract(), 112
 feof(), 82

fgetc(), 84
file(), 84
file_exists(), 85
file_get_contents(), 84
filesize(), 85
flock(), 87
fopen(), 73, 210, 412
fpassthru(), 84
fread(), 85
fseek(), 86
ftell(), 86
fwrite(), 78
get_loaded_extensions(), 502
getdate(), 419
getenv(), 398
getlastmod(), 502
gettext(), 435, 438
highlight_string(), 504
htmlspecialchars(), 119, 120
imagecolorallocate(), 445
imagefill(), 445
imagepng(), 447
imap_body(), 598
imap_etchstructure(), 608
imap_fetch_overview(), 596, 598
imap_list(), 593
implode(), 125
include(), 144, 150
ini_get(), 504
join(), 125
ksort(), 101
list(), 402
ltrim(), 118
mail(), 400
MySQL DATE_FORMAT(), 423
next(), 110
nl2br(), 121
pos(), 110
prev(), 110
print(), 122
putenv(), 398
range(), 106
readfile(), 84
require(), 144, 145, 146
reset(), 110
rewind(), 86
serialize(), 501
show_source(), 505
shuffle(), 105
sizeof(), 112
sort(), 100
str_replace(), 131
strchr(), 129
stristr(), 129
strlen(), 128

- strpos(), 130
- strchr(), 129
- strrpos(), 130
- strstr(), 129
- strtok(), 126
- substr(), 127
- substr_replace(), 131
- textdomain(), 439
- trim(), 118
- unlink(), 86
- usort(), 103
- funkcje
 - anonimowe, 165
 - dostępowe, 174
 - FTP, 414
 - IMAP, 591
 - kalendarzowe, 426
 - katalogowe, 388
 - łańcuchowe, 129, 432
 - obrazków, 462
 - nazwa, 154
 - parametry, 156
 - plikowe, 85
 - połączeń sieciowych, 403
 - sieci, 399
 - struktura, 154
 - uruchamiające programy, 395
 - własne, 154
 - wywołanie, 153
 - zarządzania zmiennymi, 59
 - zwracanie wartości, 162
- G**
- generatory, 195
- globalna przestrzeń nazw, 200
- grupowanie danych, 264
- grupy tras, 577
- H**
- hasła, 299, 368, 555
- hasło zapomniane, 557
- HTML
 - osadzanie PHP, 36
- I**
- IDE, Integrated Development Environment, 517
- identyfikacja
 - użytkowników, 337, 365, 540
 - właściciela skryptu, 502
- identyfikatory, 44
 - MySQL, 244
 - sesji, 465
- IMAP, 591
- implementacja
 - bazy danych, 543, 654
 - dziedziczenia, 175
 - funkcji anonimowych, 165
 - interfejsów, 180
 - interfejsu administratora, 676
 - iteratorów, 194
 - katalogu online, 656
 - klasy ImapServiceProvider, 616
 - klienta Instagrama, 639
 - kontroli dostępu, 366
 - kontroli wersji, 516
 - koszyka na zakupy, 662
 - metod statycznych, 190
 - metody uwierzytelniania, 376
 - OAuth, 637
 - płatności, 675
 - podstawowej witryny, 544
 - prostych sesji, 466
 - przestrzeni nazw, 200
 - rekomendacji, 566
 - rekurencji, 163
 - replikacji, 310
 - systemu płatności, 650
 - usuwania wiadomości, 629
 - uwierzytelniania, 470
 - uwierzytelniania użytkowników, 546
 - widoku aplikacji, 621
 - wysyłania wiadomości, 629
 - wyświetlania wiadomości, 626
- indeksy, 243
- informacje
 - o bazie danych, 301
 - o dacie i czasie, 415
 - o katalogu, 391
 - o pliku, 392
 - o środowisku PHP, 501
- inicjowanie tablic, 92, 95
- InnoDB
 - transakcje, 316
- instalacja
 - Apache, 688, 692, 698
 - MySQL, 688, 698, 690
 - PEAR, 700
 - PHP, 688, 698, 692
- instrukcja
 - else, 62
 - elseif, 63
 - if, 61
 - switch, 63
- instrukcje PHP, 38
- interfejs, 180
 - administratora, 651, 676
 - MessageInterface, 606

internetowa baza danych, 227
iteracja, 65, 194
izolacja, 316

J

jądra aplikacji, 574
język
 DDL, 254
 JavaScript, 477
 PHP, 22
 SQL, 253
jQuery, 478, 487

K

kadr obrazka, 444
katalogi, 391
klasa, 168, 170
 Attachment, 614
 Exception, 205
 ImapServiceProvider, 616
 klienta IMAP, 602
 Message, 606
klasy
 abstrakcyjne, 192
 modelu, 575
 znaków, 133
klauzula *Patrz* polecenie
klient
 Instagrama, 639
 poczty elektronicznej, 571, 591, 615
klonowanie obiektów, 191
klucz, 93, 217, 222
klucze obce, 317
kolejność operatorów, 57
kolumny, 216, 241
komentarze, 39, 116
komentowanie kodu, 514
konfiguracja
 cookies, 464
 kontrolni sesji, 469
 odbiorcy, 312
 podstawowego kadru, 451
 serwera nadrzędnego, 311
 serwera Apache, 695
 serwera WWW, 355
 strefy zdemilitaryzowanej, 360
konstruktory, 170
kontrola
 dostępu, 366
 sesji, 463, 469
 wersji, 516
kontroler frameworka Laravel, 575
kontrolery, 578
konwencje nazewnicze, 512

kopia
 bezpieczeństwa, 407
 lustrzana pliku, 407
 zapasowa bazy, 309
koszyk na zakupy, 649
 dane klienta, 670
 dodawanie produktów, 667
 implementacja, 662
 implementacja bazy danych, 654
 implementacja katalogu online, 656
 implementacja płatności, 675
 interfejs administratora, 676
 kasa, 670
 kategorie, 658
 pliki aplikacji, 653
 podgląd, 665
 skrypt kontrolujący, 664
 wyświetlanie podsumowania, 669
zapisywanie, 669

L

Laravel 5, 571
logowanie, 551
 OAuth, 640
lokalizacja, 429, 432

Ł

łańcuchy znaków, 43, 115, 197
 dopasowywanie, 129
 formatowanie, 117
 łączenie, 125
 porównywanie, 127
 rozdzielanie, 125
 zamiana, 129
łączenie
 łańcuchów znaków, 43, 125
 tabel, 259, 260
łączone operatory przypisania, 50

M

mechanizm Eloquent, 588
media społecznościowe, 635
metaznaki, 136
metoda
 \$.ajax(), 490
 \$.get(), 492
 \$.getJSON(), 493
 __autoload(), 193
 __call(), 192
 fetch(), 608
 on(), 483

metody

- jQuery, 490
 - statyczne, 190
- mikrosekundy, 426
- modele, 584
- modyfikacje danych, 332
- modyfikator
 - private, 173
 - protected, 173
 - public, 173
- monitor MySQL, 228
- monitorowanie bezpieczeństwa, 343
- MySQL, 23, 213
 - agregowanie danych, 264
 - identyfikatory, 244
 - kod źródłowy, 28
 - kopia zapasowa, 309
 - koszt, 27
 - obliczanie dat, 425
 - ochrona, 298
 - optymalizacja, 308
 - procedury składowane, 318
 - przenośność, 27
 - przywileje, 233
 - przywracanie bazy, 310
 - replikacja, 310
 - składowania danych, 314
 - struktury sterujące, 321
 - szeregowanie danych, 264
 - tworzenie bazy danych, 227
 - typy połączeń, 263
 - usuwanie rekordów, 271
 - usuwanie tabel, 272
 - wsparcie, 28
 - wstawianie danych, 283
 - wydajność, 27
 - wyszukiwanie danych, 256, 259, 261
 - wyzwalacze, 324
 - zaawansowane programowanie, 313
 - zapisywanie danych, 254
 - zmiana rekordów, 269
 - zmiana struktury tabel, 269
- MySQL 5.x, 28

N

nazwy

- funkcji, 154, 155
- tabel, 262



- OAuth, 635
- obiekty, 168
- obliczanie
 - dat, 424
 - sum, 56
- obrazki, 441
- obsługa
 - błędów i wyjątków, 203
 - obrazków, 441
 - zadań, 574
 - zadań AJAX, 492
- ochrona bazy danych, 298
- ODBC, 286
- odczyt
 - z katalogów, 388
 - z pliku, 81
- odczytywanie zakładek, 561
- odstępy, 38
- odwołania wsteczne, 138
- ograniczanie ryzyka, 334
- opakowywanie funkcji IMAP, 599
- opcja
 - auto_append_file, 150
 - auto_prepend_file, 150
- opcje konfiguracyjne sesji, 470
- operacje, 170
 - CRUD, 588
- operator, 48
 - równości, 52
 - tłumienia błędów, 54
 - trójkowy, 54
 - typu, 55
 - wykonania, 54
 - wykonawczy, 352
- operatory
 - arytmetyczne, 48
 - bitowe, 53
 - łańcuchowe, 49
 - podzapytań, 268
 - porównania, 258
 - porównań, 52
 - przypisania, 49
 - referencji, 51
 - tablicowe, 55, 96
- optymalizacja
 - kodu, 519, 520
 - bazy danych, 308
- organizacja kodu źródłowego, 349
- osadzanie usług, 400
- otwieranie pliku, 73
- oznaczanie zdjęć, 646

P

- parametry, 156
- PDO, 286
- PEAR, 700
- personalizacja, 539, 540
- pętla
 - do..while, 68
 - for, 67, 94
 - foreach, 67
 - while, 66
- PHP, 22
 - API Reflection, 197
 - atrybuty, 170
 - data i czas, 415
 - dokumentacja, 26
 - dziedziczenie, 175
 - formatowanie łańcuchów znaków, 117
 - funkcje kalendarzowe, 426
 - funkcje łańcuchowe, 432
 - funkcje sieci, 399
 - generowanie obrazków, 441
 - identyfikatory, 44
 - integracja z bazami danych, 25
 - klasy, 170
 - kod źródłowy, 26
 - komentarze, 39
 - kontrola sesji, 463, 469
 - łańcuchy znaków, 43, 115
 - metody uwierzytelniania, 365, 371
 - metodyki programowania, 26
 - obliczanie dat, 424
 - obsługa błędów i wyjątków, 203
 - obsługa mechanizmów obiektowych, 25
 - odstępy, 38
 - operacje, 170
 - operatory, 48
 - podświetlanie składni, 504
 - połączenie z bazą, 273
 - programowanie obiektowe, 167
 - przechowywanie danych, 71
 - przenośność, 25
 - przestrzenie nazw, 198
 - przetwarzanie plików, 72
 - skalowalność, 24
 - stałe, 46
 - stałe klasowe, 189
 - stosowanie funkcji, 151
 - tablice, 91
 - typy zmiennych, 44
 - w HTML, 36
 - wbudowane biblioteki, 25
 - wiersz poleceń, 505
 - wsparcie, 26
 - wydajność, 24
 - wyrażenia regularne, 132
 - wysyłanie plików, 379
 - wyszukiwanie danych, 71
 - wywoływanie funkcji, 40
 - zasięg zmiennych, 47
 - zawartość dynamiczna, 39
 - zmienne, 43
 - znaczniki, 37
- PHP 7, 26
- pierwsza aplikacja, 34
- planowanie
 - projektu, 510
 - z wyprzedzeniem, 342
- plik
 - .htaccess, 373
 - dziennika, 538
 - php.ini, 355, 503
- pliki
 - blokowanie, 87
 - formaty, 79
 - jednorodne, 88
 - odczyt, 81
 - problemy z otwieraniem, 76
 - przenoszenie, 395
 - tłumaczeń, 437
 - tryby otwarcia, 73
 - tworzenie, 395
 - usuwanie, 395
 - wczytywanie tablic, 107
 - zamykanie, 79
 - zapisywanie danych, 77
 - zmiana właściwości, 394
- pobieranie
 - listy wiadomości, 594
 - pliku, 412
 - wiadomości pocztowych, 598
- poczta elektroniczna, 400
- podświetlanie składni, 504
- podzapytania
 - podstawowe, 267
 - skorelowane, 268
 - w charakterze tabeli tymczasowej, 269
 - wierszowe, 268
- pogawędki, 496
- polecenie
 - ALTER TABLE, 270, 271
 - CREATE USER, 231
 - DESCRIBE, 243, 303
 - EXPLAIN, 303–307
 - GRANT, 231, 236, 291
 - GROUP BY, 265
 - INNER JOIN, 259
 - INSERT, 283
 - LIMIT, 266
 - LOAD DATA INFILE, 313
 - ORDER BY, 264

REVOKE, 236
 SELECT, 266
 SHOW, 243, 301
 UPDATE, 269
 polimorfizm, 169
 połączenie
 z bazą, 273, 278
 z serwerem, 359
 z serwerem FTP, 410
 ponowne stosowanie kodu, 143, 511
 porównywanie łańcuchów znaków, 127
 postinkrementacja, 51
 poziomy przywilejów, 233
 późne wiązania statyczne, 191
 preinkrementacja, 51
 priorytet operatorów, 57
 procedury składowane, 318
 programowanie zorientowane obiektowo, 167
 projektowanie
 klas, 182
 tabel, 219
 protokół, 399
 FTP, 75
 HTTP, 75
 IMAP, 591
 IMAP4, 400
 OAuth, 635
 SMTP, 400
 prototypowanie, 518
 prowadzenie pogawędek, 493
 przechowywanie
 danych, 71
 haseł, 368
 przeciążanie metod, 192
 przekazanie
 przez referencję, 160
 przez wartość, 160
 przesłanie, 177
 przestrzenie nazw, 198
 przetwarzanie
 formularza, 36
 plików, 72
 przyciski, 452
 przypisywanie, 50
 przywileje, 230, 291, 297, 308
 przywracanie bazy danych, 310

R

RDBMS, 89
 redundantne dane, 220
 reinterpretacja zmiennych, 61
 rejestracja
 błędów, 523, 536, 538
 użytkowników, 237, 230, 546

rekomendacje, 566
 rekomendowanie zakładek, 541
 rekurencja, 163
 relacja klient-serwer, 224
 relacje, 218
 relacyjne bazy danych, 216
 replikacja, 310
 router frameworka Laravel, 575
 rozszerzenie
 Mbstring, 571
 OpenSSL, 571
 PDO, 286, 571
 Tokenizer, 571
 rozwijanie projektu, 568
 rysowanie figur, 455
 ryzyko, 330
 rzutowanie typu, 46

S

schematy, 218
 sekwencje specjalne, 137
 selektory, 479
 serializacja, 500
 serwer
 Apache, 373
 FTP, 410
 logowanie, 410
 modyfikacja pliku, 411
 pobieranie pliku, 412
 wysyłanie plików, 413
 zamykanie połączenia, 413
 IMAP, 592, 594
 MySQL, 229
 pogawędek, 493
 składowanie danych, 314
 skrzynki pocztowe, 593
 słowo kluczowe
 final, 178
 parent, 178
 return, 161
 sortowanie
 odwrotne, 102, 104
 tablic, 100
 własne użytkownika, 103
 spójność, 315
 sprawdzanie
 obecności kodu SQL, 347
 oczekiwanych wartości, 344
 typów zmiennych, 59
 typu klasy, 190
 obsługi PHP, 696
 poprawności danych, 277
 SQL, Structured Query Language, 253
 SSL, Secure Socjet Layer, 670, 697
 stabilność kodu, 352

- stała, 46
 - klasowa, 189
- stałe zgłaszania błędów, 533
- standardy kodowania, 512
- strefa zdemilitaryzowana, 360
- strefy czasowe, 415
- struktura
 - aplikacji Laravel, 572
 - declare, 70
 - dokumentu WWW, 519
 - funkcji, 154
 - katalogów, 516
 - klasy, 170
 - skryptu, 69
 - strony, 432
 - wiadomości e-mail, 611
- strumień wyjściowy błędów, 538
- styl
 - krótki, 37
 - XML, 37
- superużytkownik, 358
- system
 - plików, 351, 392
 - płatności, 650
 - przywilejów, 231
 - uprawnień, 357
- szablon Blade, 582
- szeregowanie danych, 264

Ś

- śledzenie zakupów, 650

T

- tabela, 216, 238
 - columns_priv, 296
 - db, 295
 - procs_priv, 296
 - tables_priv, 296
 - user, 293
- tablice, 91
 - indeksowane numerycznie, 92
 - wielowymiarowe, 97, 102
 - z innymi indeksami, 94
 - zmiana kolejności elementów, 105
- technologia AJAX, 477
- tekst przycisku, 455
- testowanie, 520
- transakcje, 315
- transfer danych początkowych, 311
- transmisja danych, 331
- trasy, 576
- trwałość, 316
- tworzenie
 - egzemplarzy, 171
 - funkcji, 143

- indeksów, 243
- internetowej bazy danych, 227
- katalogów, 391
- kopii bezpieczeństwa, 407
- obrazków, 443, 448
- tabel, 238
- typ danych
 - Array, 45
 - Boolean, 45
 - Float, 45
 - Integer, 45
 - Object, 45
 - String, 45
- typy
 - całkowitoliczbowe, 246, 247
 - danych w kolumnach, 246
 - daty i czasu, 248
 - kolumn, 241
 - liczbowe, 246
 - łańcuchowe, 249
 - łańcuchowe binarne, 250
 - o ustalonej precyzji, 248
 - przywilejów, 233
 - tabel, 223
 - ARCHIVE, 314
 - CSV, 314
 - InnoDB, 314
 - MEMORY, 314
 - MERGE, 314
 - MyISAM, 314
 - zmiennoprzecinkowe, 247
 - zmiennych, 44

U

- uaktualnianie
 - oprogramowania, 354
 - systemu operacyjnego, 361
- udostępnianie usług, 362
- uprawnienia, 357
- usługa uwierzytelniająca, 635
- ustawianie typów zmiennych, 59
- usuwanie
 - błędów, 523
 - baz danych, 272
 - katalogów, 391
 - rekordów, 271
 - tabel, 272
 - wiadomości, 629
 - zakładek, 564
- utrata danych, 332
- uwierzytelnianie, 365, 371, 470
 - aplikacji, 617
 - użytkowników, 539, 546
- użytkownicy, 357
- używanie bazy danych, 238

W

wartości, 217
 wcinanie, 514
 wdrażanie nowych wersji, 354
 widok, 575, 581
 administratora systemu, 652
 użytkownika systemu, 651
 widoki Blade, 582
 wielokrotne dziedziczenie, 179
 wiersz poleceń, 505
 wiersze, 216
 właściciel
 skryptu, 502
 pliku, 394
 wskazywanie typu, 190
 współużytkowane serwery hostingowe, 356
 wstrzykiwanie kodu, 335
 wybór
 bazy danych, 279
 środowiska programistycznego, 517
 wyjątki, 203
 użytkownika, 206
 wykresy, 455
 wylogowanie, 554
 wyrażenia regularne, 132
 asercje, 138
 dopasowywanie znaków specjalnych, 136
 klasy znaków, 133
 kotwiczenie, 135
 metaznaki, 136
 odnajdywanie fragmentów łańcuchów, 140
 odwołania wsteczne, 138
 ograniczniki, 132
 podwyrażenia, 135
 podwyrażenia policzalne, 135
 powtarzalność, 134
 rozdzielanie łańcuchów, 141
 rozgałęzianie, 135
 sekwencje specjalne, 137
 style składni, 132
 zamiana fragmentów łańcuchów, 141
 wysyłanie
 plików, 379, 380, 385, 413
 wiadomości, 629
 wyszukiwanie danych, 71, 256, 259, 261
 wyświetlanie
 grafiki, 446
 strumienia, 643
 zakładek, 563
 wywoływanie
 funkcji, 40, 151
 funkcji niezdefiniowanej, 153
 operacji klas, 172
 wyzwalacze, 324
 wyzwalanie własnych błędów, 535
 wzorzec MVC Laravela, 574

Z

zabezpieczanie
 hasła, 369
 kodu źródłowego, 343
 strony, 370
 zakładki, 540
 dodawanie, 561
 implementacja przechowywania, 561
 odczytywanie, 561
 usuwanie, 564
 wyświetlanie, 563
 zalety
 MySQL, 27
 PHP, 24
 zamiana
 fragmentów łańcuchów, 141
 łańcuchów znaków, 129
 zamykanie
 pliku, 79
 połączenia, 413
 połączenia z bazą, 282
 zapisywanie danych, 254
 zapory sieciowe, 360
 zapytania do bazy danych, 276, 279
 zarządzanie
 datą i czasem, 415
 zmiennymi, 59
 zasada
 działania sesji, 463
 najmniejszego przywileju, 231
 zasięg zmiennych, 47, 158
 zastosowanie
 PHP, 34
 znaczników PHP, 37
 zawartość kodu źródłowego, 350
 zbiory, 133
 wynikowe selektorów, 481
 znaków, 430
 zgłaszanie błędów, 532, 534
 zintegrowane środowisko programistyczne, 517
 zmiana
 hasła, 555
 przywilejów, 298
 struktury tabel, 269
 ustawień zgłaszania błędów, 534
 wielkości liter, 124
 właściwości pliku, 394
 zmiennie, 43
 formularza, 41
 lokalne, 321
 sesyjne, 466
 skalarne, 112
 zmiennych, 46
 znacznik czasu, 420
 Uniksa, 417
 znaczniki PHP, 37

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**



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 bywałcem konferencji poświęconych programowaniu aplikacji internetowych. Jakiś czas temu wykladał inżynierię oprogramowania na Uniwersytecie w Melbourne w Australii. W wolnym czasie zajmuje się propagowaniem zalet wolnego oprogramowania.



księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

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/nawosci>

ISBN 978-83-283-3256-0



9 788328 332560

cena: 109,00 zł



Addison
Wesley