



Brad Dayley

Node.js, MongoDB, AngularJS

Kompendium wiedzy

Poznaj potencjał platformy Node.js!



Helion 

Tytuł oryginału: Node.js, MongoDB, and AngularJS Web Development

Tłumaczenie: Piotr Pilch

ISBN: 978-83-283-0111-5

Authorized translation from the English language edition, entitled: NODE.JS, MONGODB, AND ANGULARJS WEB DEVELOPMENT; ISBN 0321995783; by Brad Dayley; published by Pearson Education, Inc, publishing as Addison Wesley.
Copyright © 2014 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 © 2015.

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 bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi 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/nodekw.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/nodekw>
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 autorze	13
Część I	Wprowadzenie	15
	Wprowadzenie	17
	Kto powinien przeczytać tę książkę?	17
	Dlaczego należy przeczytać tę książkę?	17
	Czego się dowiesz z tej książki?	18
	Czym jest środowisko Node.js?	19
	Czym jest komponent MongoDB?	20
	Czym jest komponent AngularJS?	20
	Jaka jest struktura tej książki?	21
	Uzyskiwanie przykładowych kodów	22
	Uwagi końcowe	22
Rozdział 1	Podstawowe informacje o stosie Node.js-AngularJS	23
	Podstawowe środowisko projektowania aplikacji internetowych	23
	Komponenty stosu Node.js-AngularJS	27
	Podsumowanie	31
	W następnym rozdziale	32
Rozdział 2	Wprowadzenie do języka JavaScript	33
	Definiowanie zmiennych	33
	Typy danych języka JavaScript	34
	Użycie operatorów	35
	Implementowanie pętli	39
	Tworzenie funkcji	42
	Zasięg zmiennych	44
	Użycie obiektów języka JavaScript	45
	Przetwarzanie łańcuchów	48
	Korzystanie z tablic	51
	Dodawanie obsługi błędów	53
	Podsumowanie	56
	W następnym rozdziale	56

Część II Środowisko Node.js	57
Rozdział 3 Wprowadzenie do środowiska Node.js	59
Środowisko Node.js	59
Instalowanie środowiska Node.js	60
Praca z pakietami Node.js	62
Tworzenie aplikacji Node.js	68
Zapisywanie danych do konsoli	72
Podsumowanie	74
W następnym rozdziale	74
Rozdział 4 Użycie zdarzeń, procesów nasłuchiwania, liczników i wywołań zwrotnych w środowisku Node.js	75
Model zdarzeń środowiska Node.js	75
Dodawanie zadań do kolejki zdarzeń	79
Implementowanie wywołań zwrotnych	88
Podsumowanie	92
W następnym rozdziale	93
Rozdział 5 Obsługa danych wejścia-wyjścia w środowisku Node.js	95
Praca z danymi JSON	95
Użycie modułu Buffer do buforowania danych	97
Użycie modułu Stream do strumieniowania danych	104
Podsumowanie	118
W następnym rozdziale	118
Rozdział 6 Uzyskiwanie dostępu do systemu plików ze środowiska Node.js	119
Porównanie synchronicznych i asynchronicznych wywołań systemu plików	119
Otwieranie i zamykanie plików	120
Zapisywanie plików	122
Odczytywanie plików	127
Inne zadania związane z systemem plików	133
Podsumowanie	140
W następnym rozdziale	140
Rozdział 7 Implementowanie usług HTTP w środowisku Node.js	141
Przetwarzanie adresów URL	141
Przetwarzanie łańcuchów zapytania i parametrów formularza	144
Obiekty żądania, odpowiedzi i serwera	145
Implementowanie klientów i serwerów HTTP w środowisku Node.js	153
Implementowanie serwerów i klientów HTTPS	162
Podsumowanie	165
W następnym rozdziale	166

Rozdział 8	Implementowanie usług gniazd w środowisku Node.js	167
	Gniazda sieciowe	167
	Obiekty Server i Socket protokołu TCP	168
	Implementowanie klientów i serwerów gniazd TCP	176
	Implementowanie serwerów i klientów TLS	181
	Podsumowanie	186
	W następnym rozdziale	186
Rozdział 9	Skalowanie aplikacji przy użyciu wielu procesorów w środowisku Node.js	187
	Moduł process	187
	Implementowanie procesów podrzędnych	192
	Implementowanie klastrów procesów	203
	Podsumowanie	208
	W następnym rozdziale	209
Rozdział 10	Użycie dodatkowych modułów środowiska Node.js	211
	Użycie modułu os	211
	Użycie modułu util	213
	Podsumowanie	219
	W następnym rozdziale	220
Część III	Komponent MongoDB	221
Rozdział 11	Baza danych NoSQL i komponent MongoDB	223
	Dlaczego baza danych NoSQL?	223
	Komponent MongoDB	224
	Typy danych bazy danych MongoDB	226
	Planowanie modelu danych	227
	Podsumowanie	233
	W następnym rozdziale	234
Rozdział 12	Wprowadzenie do korzystania z komponentu MongoDB	235
	Tworzenie środowiska komponentu MongoDB	235
	Administrowanie kontami użytkowników	241
	Konfigurowanie kontroli dostępu	245
	Administrowanie bazami danych	247
	Zarządzanie kolekcjami	250
	Podsumowanie	256
	W następnym rozdziale	256
Rozdział 13	Wprowadzenie do współpracy komponentu MongoDB i środowiska Node.js	257
	Dodawanie sterownika komponentu MongoDB do środowiska Node.js	257
	Nawiązywanie połączenia z komponentem MongoDB ze środowiska Node.js	258

Obiekty używane w sterowniku środowiska Node.js dla komponentu MongoDB	265
Uzyskiwanie dostępu do baz danych i modyfikowanie ich	273
Uzyskiwanie dostępu do kolekcji i modyfikowanie ich	276
Podsumowanie	280
W następnym rozdziale	280
Rozdział 14 Modyfikowanie dokumentów bazy danych MongoDB w środowisku Node.js	281
Opcje wprowadzania zmian w bazach danych	281
Operatory aktualizowania baz danych	283
Dodawanie dokumentów do kolekcji	283
Uzyskiwanie dokumentów z kolekcji	286
Aktualizowanie dokumentów w kolekcji	287
Niepodzielne modyfikowanie dokumentów w kolekcji	290
Zapisywanie dokumentów w kolekcji	291
Użycie opcji upsert do wstawiania dokumentów do kolekcji	293
Usuwanie dokumentów z kolekcji	294
Usuwanie pojedynczego dokumentu z kolekcji	296
Podsumowanie	297
W następnym rozdziale	297
Rozdział 15 Uzyskiwanie dostępu do dokumentów bazy danych MongoDB w środowisku Node.js	299
Podstawowe informacje na temat zestawu danych	299
Obiekty query	300
Obiekty options zapytań	302
Znajdowanie konkretnych zestawów dokumentów	304
Określanie liczby dokumentów	306
Ograniczanie zestawów wynikowych	308
Sortowanie zestawów wynikowych	313
Znajdowanie różnych wartości pola	315
Grupowanie wyników	316
Korzystanie ze środowiska MapReduce podczas agregowania wyników	319
Podsumowanie	324
W następnym rozdziale	324
Rozdział 16 Użycie biblioteki Mongoose dla schematu ze strukturą i do sprawdzania poprawności	325
Biblioteka Mongoose	326
Nawiązywanie połączenia z bazą danych MongoDB za pomocą biblioteki Mongoose	327
Definiowanie schematu	328
Kompilowanie modelu	332

Obiekt Query	333
Obiekt Document	339
Znajdowanie dokumentów za pomocą biblioteki Mongoose	340
Dodawanie dokumentów za pomocą biblioteki Mongoose	343
Aktualizowanie dokumentów za pomocą biblioteki Mongoose	344
Usuwanie dokumentów za pomocą biblioteki Mongoose	349
Agregowanie dokumentów za pomocą biblioteki Mongoose	351
Użycie środowiska sprawdzania poprawności	354
Implementowanie funkcji pośrednich	356
Podsumowanie	359
W następnym rozdziale	359
Rozdział 17 Zaawansowane zagadnienia związane z bazą danych MongoDB	361
Dodawanie indeksów	361
Użycie ograniczonych kolekcji	364
Stosowanie replikacji	365
Implementowanie tworzenia segmentów danych	369
Implementowanie magazynu GridFS	377
Naprawianie bazy danych MongoDB	383
Tworzenie kopii zapasowej bazy danych MongoDB	384
Podsumowanie	385
W następnym rozdziale	385
Część IV Użycie modułu Express do usprawnienia pracy	387
Rozdział 18 Implementacja komponentu Express w środowisku Node.js	389
Wprowadzenie do komponentu Express	389
Konfigurowanie tras	392
Użycie obiektów Request	397
Użycie obiektów Response	398
Implementowanie mechanizmu szablonów	406
Podsumowanie	411
W następnym rozdziale	411
Rozdział 19 Implementacja funkcji pośrednich komponentu Express	413
Funkcje pośrednie	413
Użycie funkcji pośredniej query	416
Udostępnianie plików statycznych	416
Obsługa danych treści żądania POST	418
Wysyłanie i odbieranie informacji cookie	419
Implementowanie sesji	421
Stosowanie podstawowego uwierzytelniania HTTP	423

Implementowanie uwierzytelniania sesji	424
Tworzenie niestandardowych funkcji pośrednich	427
Podsumowanie	428
W następnym rozdziale	428
Część V Komponent AngularJS	429
Rozdział 20 Wprowadzenie do komponentu AngularJS	431
Dlaczego komponent AngularJS?	431
Komponent AngularJS	433
Przegląd cyklu życia aplikacji AngularJS	436
Integrowanie komponentu AngularJS z istniejącym kodem JavaScript i jQuery	437
Dodawanie komponentu AngularJS do środowiska Node.js	438
Inicjowanie komponentu AngularJS w dokumencie HTML	438
Użycie globalnych interfejsów API	439
Tworzenie prostej aplikacji AngularJS	441
Podsumowanie	445
W następnym rozdziale	445
Rozdział 21 Moduły komponentu AngularJS i wstrzykiwanie zależności	447
Przegląd modułów i wstrzykiwania zależności	447
Definiowanie modułów AngularJS	448
Implementowanie wstrzykiwania zależności	452
Podsumowanie	454
W następnym rozdziale	454
Rozdział 22 Implementowanie zasięgu jako modelu danych	455
Zasięgi	455
Implementowanie hierarchii zasięgów	461
Emitowanie i rozgłaszanie zdarzeń	463
Podsumowanie	467
W następnym rozdziale	468
Rozdział 23 Tworzenie widoków za pomocą szablonów środowiska AngularJS	469
Szablony	469
Użycie wyrażeń	470
Użycie filtrów	473
Tworzenie filtrów niestandardowych	479
Podsumowanie	482
W następnym rozdziale	483

Rozdział 24 Implementowanie dyrektyw w widokach aplikacji AngularJS	485
Dyrektywy	485
Użycie wbudowanych dyrektyw	486
Tworzenie własnych dyrektyw do rozszerzania kodu HTML	500
Podsumowanie	509
W następnym rozdziale	509
Rozdział 25 Implementowanie usług środowiska AngularJS w aplikacjach internetowych	511
Usługi środowiska AngularJS	511
Użycie wbudowanych usług	512
Tworzenie usług niestandardowych	527
Podsumowanie	529
W następnym rozdziale	529
Część VI Tworzenie praktycznych komponentów aplikacji internetowych	531
Rozdział 26 Dodawanie kont użytkowników do witryny internetowej	533
Używane biblioteki	533
Struktura katalogowa projektu	534
Definiowanie modelu użytkowników	535
Tworzenie serwera	535
Implementowanie tras	537
Implementowanie tras kontrolera użytkowników	538
Implementowanie widoków użytkownika i uwierzytelniania	543
Implementowanie modułu i kontrolera środowiska AngularJS	549
Użycie kont społecznościowych jako źródeł uwierzytelniania	549
Podsumowanie	555
W następnym rozdziale	556
Rozdział 27 Dodawanie wątków komentarzy do stron	557
Używane biblioteki	557
Struktura katalogowa projektu	558
Definiowanie modeli komentarzy, odpowiedzi, zdjęć i stron	559
Tworzenie serwera komentarzy	561
Implementowanie tras do obsługi wyświetlania i dodawania komentarzy	563
Implementowanie tras kontrolerów opartych na modelu	563
Implementowanie widoków komentarzy i zdjęć	569
Implementowanie modułu i kontrolerów środowiska AngularJS do obsługi widoków komentarzy	576
Inicjowanie aplikacji	581
Podsumowanie	583
W następnym rozdziale	583

Rozdział 28 Tworzenie własnego koszyka zakupów	585
Opis projektu	585
Używane biblioteki	586
Struktura katalogowa projektu	586
Definiowanie modeli klientów, produktów i zamówień	588
Tworzenie serwera koszyka zakupów	592
Implementowanie tras do obsługi żądań dotyczących produktów, koszyka i zamówień	593
Implementowanie tras kontrolera opartego na modelu	594
Implementowanie widoków kasy i koszyka zakupów	598
Implementowanie modułu i kontrolera środowiska AngularJS do obsługi widoków koszyka zakupów	611
Inicjowanie aplikacji	619
Podsumowanie	621
W następnym rozdziale	621
 Rozdział 29 Tworzenie interaktywnych komponentów aplikacji Web 2.0	 623
Opis projektu	623
Używane biblioteki	624
Struktura katalogowa projektu	624
Definiowanie modelu projektu	626
Tworzenie serwera aplikacji	626
Implementowanie tras do obsługi widoków	627
Implementowanie widoku z kartami	628
Implementowanie widoku usługi pogodowej	633
Implementowanie elementów, które można przeciągać	637
Implementowanie dostępu do danych dynamicznych	641
Inicjowanie aplikacji	646
Podsumowanie	648
 Skorowidz	 651

Dodawanie wątków komentarzy do stron

W tym rozdziale zamieszczono praktyczny przykład implementowania pełnego stosu Node.js-MongoDB-AngularJS. Dowiesz się, jak umożliwić użytkownikom dodawanie zagnieżdżonych komentarzy do jednego lub większej liczby obszarów strony internetowej. Opcja dodawania komentarzy stanowi typowy element wielu witryn internetowych. Wiąże się z tym kilka trudności.

Przykład zaprezentowany w tym rozdziale prezentuje pojedynczą stronę internetową, która umożliwia komentowanie całej strony, a także poszczególnych zdjęć ładowanych na stronie. Kod jest prosty i łatwy do prześledzenia.

Aby było Ci jeszcze łatwiej go prześledzić, nie uwzględniono w nim zarządzania użytkownikami, uwierzytelnianiem i sesjami. Jeśli chcesz przypomnieć sobie te zagadnienia, wróć do rozdziału 26. Przykład prezentowany w tym rozdziale zawiera niewielką funkcję, która symuluje użycie nazwy użytkownika w sesji w celu zezwolenia na wyświetlenie komentarzy wielu użytkowników.

Używane biblioteki

W projekcie omawianym w tym rozdziale używane są dodatkowe moduły NPM środowiska Node.js. Aby przejść dalej, musisz je zainstalować w katalogu projektu. Oto one:

- `express`. Używany jako główny serwer WWW projektu.
- `body-parser`. Zapewnia obsługę treści JSON na potrzeby żądań POST.
- `ejs`. Używany do renderowania szablonów HTML.
- `mongodb`. Używany do uzyskiwania dostępu do bazy danych MongoDB.
- `mongoose`. Używany do zapewniania modelu danych strukturyzowanych.

Kod w tym rozdziale wymaga również dostępności biblioteki AngularJS.

Struktura katalogowa projektu

Projekt zorganizowano w ramach następującej struktury katalogowej:

- `./`. Zawiera podstawowe pliki aplikacji i foldery pomocnicze. Jest to główny katalog projektu.
- `./node_modules`. Jest tworzony po zainstalowaniu w systemie wyszczególnionych wcześniej modułów NPM.
- `./controllers`. Zawiera kontrolery tras komponentu Express, które zapewniają interakcję między trasami i zmianami w bazie danych MongoDB.
- `./models`. Zawiera definicje modelu biblioteki Mongoose dla obiektów w bazie danych.
- `./static`. Zawiera wszystkie pliki statyczne, które należy wysłać (np. kod CSS i kod aplikacji AngularJS).
- `./views`. Zawiera szablony HTML, które będą renderowane przez mechanizm EJS.

Uwaga

Jest to tylko jedna z metod organizowania kodu. Choć nie musisz korzystać z takiej struktury katalogowej, pamiętaj o tym, że struktura powinna stanowić część ogólnej koncepcji projektów, aby łatwiejsze było znalezienie szukanego kodu.

Oprócz struktury katalogowej do projektu dołączane są poniższe pliki z kodem. Następująca lista ma na celu ułatwienie Ci zorientowania się w funkcjonalności każdego pliku:

- `./comment_init.js`. Zapewnia autonomiczny kod inicjalizacji. W celu przeprowadzenia inicjalizacji projektu kod dodaje do bazy danych MongoDB obiekt strony początkowej i kilka zdjęć.
- `./comment_server.js`. Ładuje niezbędne biblioteki, tworzy połączenie z bazą danych MongoDB i uruchamia serwer Express. Jest to główny plik aplikacji.
- `./comment_routes.js`. Definiuje trasy dla serwera Express. Plik ten obsługuje funkcje, które nie dotyczą bazy danych.
- `./controllers/comments_controller.js`. Definiuje funkcje dla tras, które wymagają interakcji z bazą danych MongoDB w celu pobierania i aktualizowania komentarzy.
- `./controllers/pages_controller.js`. Definiuje funkcje dla tras, które wymagają interakcji z bazą danych MongoDB w celu pobierania obiektów strony.
- `./controllers/photos_controller.js`. Definiuje funkcje dla tras, które wymagają interakcji z bazą danych MongoDB w celu pobierania jednego lub wszystkich obiektów zdjęć.
- `./models/comments_model.js`. Definiuje model obiektów komentarzy.
- `./models/page_model.js`. Definiuje model stron komentarzy.
- `./models/photo_model.js`. Definiuje model zdjęć komentarzy.

- *./views/photos.html*. Zapewnia dla aplikacji główną stronę zdjęć, która umożliwi użytkownikom wybranie zdjęć i dodanie komentarzy do strony lub wybranego zdjęcia.
- *./static/comment.html*. Umożliwia wyświetlenie zagnieżdżonych komentarzy przy użyciu dyrektywy `ng-repeat`. Jest to częściowy szablon środowiska AngularJS.
- *./static/comment_thread.html*. Udostępnia środowisko do dodawania wątku komentarzy w różnych miejscach na stronie internetowej. Jest to częściowy szablon środowiska AngularJS.
- *./static/js/comment_app.js*. Zapewnia definicje kontrolerów i modułów środowiska AngularJS w celu obsługi pobierania obiektów stron, zdjęć i komentarzy z serwerów, a także zapisywania nowych komentarzy.
- *./static/css/comment_styles.css*. Udostępnia style CSS dla stron HTML środowiska AngularJS.
- *./images*. Zawiera obrazy używane w przykładzie. Odpowiada folderowi dla tego rozdziału, który wchodzi w skład archiwum kodów książki.
- *./lib*. Zawiera niezbędne pliki biblioteki AngularJS, które są używane w omawianym przykładzie. Odpowiada folderowi dla tego rozdziału, który wchodzi w skład archiwum kodów książki.

Definiowanie modeli komentarzy, odpowiedzi, zdjęć i stron

Pierwszym krokiem procesu implementowania aplikacji zawsze powinno być przyjrzenie się wymaganiom modelu obiektów. W omawianym przykładzie celem jest umożliwienie użytkownikom dodawania komentarzy do stron internetowych. Aby to zrobić, musisz najpierw zidentyfikować obiekty, do których użytkownicy mogą dodawać komentarze. Przykład prezentuje stronę internetową zawierającą kilka zdjęć. By zezwolić użytkownikom na dołączanie komentarzy zarówno do witryny internetowej, jak i do poszczególnych zdjęć, wykorzystasz możliwość wielokrotnego użycia. Aby zapewnić obiekty, do których zostaną dołączone komentarze, niezbędny jest model dla stron i zdjęć. W dalszej części rozdziału omówiono projekty schematów modeli.

Definiowanie modelu Page

Kod z listingu 27.1 implementuje schemat modelu Page dla aplikacji. Schemat jest bardzo prosty. Zawiera jedynie pola `name` i `commentId`. Zauważ, że pole `name` skonfigurowano jako unikalne, co jest niezbędne, by było możliwe wyszukiwanie strony według nazwy. Głównym zadaniem modelu Page jest udostępnienie miejsca, w którym będzie można powiązać stronę internetową z wątkiem komentarzy.

Listing 27.1. Plik `page_model.js`: implementowanie modelu `Page` dla biblioteki `Mongoose`

```

01 var mongoose = require('mongoose'),
02     Schema = mongoose.Schema;
03 var PageSchema = new Schema({
04     name: {type: String, unique: true},
05     commentId: Schema.ObjectId
06 });
07 mongoose.model('Page', PageSchema);

```

Definiowanie modelu `Photo`

Kod z listingu 27.2 implementuje schemat modelu `Photo` dla aplikacji. Ten schemat również jest bardzo prosty. Zawiera pola `title`, `filename`, `timestamp` i `commentId`. Pole `commentId` ma takie samo przeznaczenie jak pole `commentId` w modelu `Page`. To wielokrotnie używane pole ułatwia późniejsze dodawanie komentarzy do dowolnych dodatkowych modeli stron lub obiektów, które mają zostać dołączone do aplikacji.

Listing 27.2. Plik `photo_model.js`: implementowanie modelu `Photo` dla biblioteki `Mongoose`

```

01 var mongoose = require('mongoose'),
02     Schema = mongoose.Schema;
03 var PhotoSchema = new Schema({
04     title: String,
05     filename: String,
06     timestamp: { type: Date, default: Date.now },
07     commentId: Schema.ObjectId
08 });
09 mongoose.model('Photo', PhotoSchema);

```

Definiowanie modelu `CommentThread`

Model komentarzy ma za zadanie obsługiwanie zagnieżdżonych komentarzy. W komponencie `MongoDB` dostępne są dwie różne metody, które to umożliwiają. Możesz zapisać każdy komentarz jako pojedynczy dokument i zapewnić odsyłacz do obiektów nadrzędnych, ale możesz też zapamiętać wszystkie komentarze wątku w jednym dokumencie. Obie metody działają, wiążą się z nimi jednak bardzo odmienne implementacje i konsekwencje.

Przechowywanie drzewa komentarzy w jednym dokumencie ułatwia i przyspiesza ładowanie danych z bazy `MongoDB`. Mankamentem jest to, że z powodu braku ustalonej głębokości dla komentarzy po dodaniu nowych komentarzy wymagane jest zapisanie całej struktury odpowiedzi. Ponadto musisz pobrać cały dokument, aby uzyskać dowolny z komentarzy.

Przechowywanie komentarzy jako pojedynczych dokumentów oznacza łatwość wyszukiwania wybranego dokumentu, a także zapisywania dowolnych dodatkowych komentarzy/odpowiedzi, gdyż są one odrębnymi dokumentami. Aby jednak załadować

całe drzewo komentarzy, musisz zaimplementować złożoną funkcję agregacji lub przeprowadzić wiele wyszukiwań w celu zbudowania drzewa. Oba rozwiązania są znacznie mniej efektywne niż załadowanie pojedynczego dokumentu.

Ze względu na to, że komentarze są bardzo często czytane, lecz rzadko tworzone, dobrą opcją jest zapisanie struktury zagnieżdżonych komentarzy/odpowiedzi w pojedynczym dokumencie bazy danych MongoDB, a następnie zaktualizowanie całej struktury po dodaniu nowego komentarza.

Kod z listingu 27.3 implementuje schematy modeli `CommentThread` i `Reply`. Schemat modelu `CommentThread` pełni rolę głównego dokumentu dla komentarzy dotyczących danego obiektu. Schemat modelu `Reply` służy do przechowywania komentarzy/odpowiedzi dodanych do strony lub zdjęcia. Zauważ, że pole `replies` obiektów `Reply` i `CommentThread` to tablica obiektów schematu odpowiedzi:

```
replies:[ReplySchema]
```

Listing 27.3. Plik `comments_model.js`: implementowanie modelu `CommentThread` dla biblioteki `Mongoose`

```
01 var mongoose = require('mongoose'),
02     Schema = mongoose.Schema;
03 var ReplySchema = new Schema({
04   username: String,
05   subject: String,
06   timestamp: { type: Date, default: Date.now },
07   body: String,
08   replies:[ReplySchema]
09 }, { _id: true });
10 var CommentThreadSchema = new Schema({
11   title: String,
12   replies:[ReplySchema]
13 });
14 mongoose.model('Reply', ReplySchema);
15 mongoose.model('CommentThread', CommentThreadSchema);
```

Taka konfiguracja umożliwia wzajemne zagnieżdżanie komentarzy wewnątrz siebie i tworzenie na tym samym poziomie wielu komentarzy. Na rysunku 27.1 pokazano zagnieżdżoną strukturę w rzeczywistym dokumencie przechowywanym w bazie danych MongoDB.

Tworzenie serwera komentarzy

Po zdefiniowaniu modelu możesz rozpocząć implementowanie serwera komentarzy. Kod z listingu 27.4 implementuje serwer Express dla aplikacji obsługującej komentarze. Kod powinien wyglądać znajomo. Zawiera on biblioteki `express` i `mongoose`, a ponadto nawiązuje połączenie z bazą danych MongoDB za pośrednictwem biblioteki `Mongoose`.

```

C:\Windows\system32\cmd.exe - mongo
> db.commentthreads.findOne(<title:'Komentarze dla: Zachody słońca')
{
  "_id" : ObjectId("5438ef4896d351bc08668746"),
  "title" : "Komentarze dla: Zachody słońca",
  "replies" : [
    {
      "_id" : ObjectId("543a2822aa4eb5c809f741c5"),
      "body" : "Przypomina mi góry.",
      "subject" : "ładny zachód słońca",
      "username" : "Dawid",
      "replies" : [
        {
          "timestamp" : ISODate("2014-10-12T07:06:05.373Z"),
          "replies" : [
            {
              "timestamp" : ISODate("2014-10-12T07:06:22.790Z"),
              "replies" : [ ],
              "_id" : ObjectId("543a286aa4eb5c809f741c7"),
              "body" : "",
              "subject" : "Utah",
              "username" : "Karol"
            }
          ]
        },
        {
          "_id" : ObjectId("543a285daa4eb5c809f741c6"),
          "body" : "Gdzie zostało wykonane zdjęcie",
          "subject" : "Niezły widok",
          "username" : "Bartek"
        }
      ]
    },
    {
      "timestamp" : ISODate("2014-10-12T07:05:06.594Z")
    }
  ]
}
>
> =

```

Rysunek 27.1. Obiekt CommentThread z zagnieżdżonymi odpowiedziami przechowywanymi w bazie danych MongoDB

Listing 27.4. Plik comments_server.js: implementowanie serwera aplikacji komentarzy za pomocą serwera Express i przez nawiązywanie połączenia z bazą danych MongoDB

```

01 var express = require('express');
02 var bodyParser = require('body-parser');
03 var mongoose = require('mongoose');
04 var db = mongoose.connect('mongodb://localhost/comments');
05 require('./models/comments_model.js');
06 require('./models/photo_model.js');
07 require('./models/page_model.js');
08 var app = express();
09 app.engine('html', require('ejs').__express);
10 app.set('views', __dirname + '/views');
11 app.set('view engine', 'html');
12 app.use(bodyParser());
13 require('./comment_routes')(app);
14 app.listen(80);

```

Zauważ, że dla każdej definicji modelu występuje instrukcja `require`. Służy ona do tworzenia obiektu `Schema` w obrębie biblioteki `Mongoose`. Ponadto dołączono plik `./comment_routes` w celu zainicjowania tras dla serwera przed rozpoczęciem nasłuchiwanie na porcie 80.

Implementowanie tras do obsługi wyświetlania i dodawania komentarzy

W ramach konfiguracji serwera Express ładowany jest plik `./comment_routes.js` (listing 27.4). Kod z listingu 27.5 zapewnia trasy niezbędne do pobrania strony, zdjęcia i danych komentarza, a także dodania komentarzy do zdjęć lub stron.

Listing 27.5. Plik `comments_routes.js`: implementowanie tras serwera aplikacji komentarzy dla serwera Express

```
01 var express = require('express');
02 module.exports = function(app) {
03   var photos = require('./controllers/photos_controller');
04   var comments = require('./controllers/comments_controller');
05   var pages = require('./controllers/pages_controller');
06   app.use('/static', express.static( './static')).
07     use('/images', express.static( '../images')).
08     use('/lib', express.static( '../lib')
09   );
10   app.get('/', function(req, res){
11     res.render('photos');
12   });
13   app.get('/photos', photos.getPhotos);
14   app.get('/photo', photos.getPhoto);
15   app.get('/page', pages.getPage);
16   app.get('/comments/get', comments.getComment);
17   app.post('/comments/add', comments.addComment);
18 }
```

W wierszach od 6. do 9. implementowane są trasy statyczne do obsługi pobierania kodu CSS, kodu JavaScript, obrazów i częściowych szablonów środowiska AngularJS, które są wykorzystywane w omawianym przykładzie. Obrazy i foldery *lib* środowiska AngularJS znajdują się w głównym katalogu projektów. Inne pliki statyczne są umieszczone w folderze `./static` w obrębie projektu.

Zauważ, że gdy użytkownik uzyska dostęp do głównej lokalizacji witryny, w wierszu 9. renderowany jest szablon `photos.html`. Wszystkie trasy w wierszach od 13. do 17. prowadzą interakcję z bazą danych MongoDB, a ponadto są obsługiwane w procedurach obsługi kontrolera opisanych w następnym podrozdziale.

Implementowanie tras kontrolerów opartych na modelu

Oprócz implementowania standardowej obsługi tras wymagane jest również zastosowanie obsługi tras do interakcji z bazą danych. Aby zachować przejrzystość kodu i zapewnić odpowiedni podział zakresu odpowiedzialności, takie procedury obsługi tras zostaną wydzielone ze standardowego pliku `comment_route.js` do ich własnych plików (dla każdego modelu).

W kolejnych punktach omówiono implementację kontrolerów specyficznych dla modeli Page, Photo i CommentThread.

Implementowanie kontrolera modelu Page

Kod z listingu 27.6 implementuje kod obsługi tras dla modelu Page. Występuje tu tylko jedna trasa getPage(), która znajduje stronę na podstawie pola name i zwraca błąd lub postać łańcuchową JSON obiektu. Nazwa strony do znalezienia trafia do parametru pageName w łańcuchu zapytania GET.

Listing 27.6. Plik pages_controller.js: implementowanie trasy getPage dla serwera Express

```
01 var mongoose = require('mongoose'),
02     Page = mongoose.model('Page');
03 exports.getPage = function(req, res) {
04   Page.findOne({ name: req.query.pageName })
05     .exec(function(err, page) {
06       if (!page){
07         res.json(404, {msg: 'Nie znaleziono strony.'});
08       } else {
09         res.json(page);
10       }
11     });
12 };
```

Implementowanie kontrolera modelu Photo

Kod z listingu 27.7 implementuje kod obsługi tras dla modelu Photo. Obsługiwane są dwie trasy. Procedura obsługi trasy getPhoto() wyszukuje pojedynczy dokument Photo na podstawie pola _id przekazanego jako parametr photoId w łańcuchu zapytania GET. Procedura obsługi trasy getPhotos() pobiera wszystkie dokumenty Photo. Obie procedury zwracają postać łańcuchową JSON wyników.

Listing 27.7. Plik photos_controller.js: implementowanie tras getPhoto i getPhotos na serwerze Express w celu pobierania zdjęć

```
01 var mongoose = require('mongoose'),
02     Photo = mongoose.model('Photo');
03 exports.getPhoto = function(req, res) {
04   Photo.findOne({ _id: req.query.photoId })
05     .exec(function(err, photo) {
06       if (!photo){
07         res.json(404, {msg: 'Nie znaleziono zdjęcia.'});
08       } else {
09         res.json(photo);
10       }
11     });
12 };
13 exports.getPhotos = function(req, res) {
14   Photo.find()
```

```
15 .exec(function(err, photos) {
16   if (!photos){
17     res.json(404, {msg: 'Nie znaleziono zdjęć.'});
18   } else {
19     res.json(photos);
20   }
21 });
22 };
```

Implementowanie kontrolera modelu CommentThread

Kod z listingu 27.8 implementuje kod obsługi tras dla modelu CommentThread. Obsługiwane są dwie trasy: `getComment()` i `addComment()`. Ze względu na wielkość pliku w kolejnych punktach rozdzielono kod dla tych dwóch tras.

Listing 27.8. Plik `comments_controller.js` (`getComment`): implementowanie trasy do uzyskania wątków komentarzy dla serwera Express

```
04 exports.getComment = function(req, res) {
05   CommentThread.findOne({ _id: req.query.commentId })
06   .exec(function(err, comment) {
07     if (!comment){
08       res.json(404, {msg: 'Nie znaleziono modelu CommentThread.'});
09     } else {
10       res.json(comment);
11     }
12   });
13 };
```

Implementowanie procedury obsługi trasy `getComment()`

Procedura obsługi trasy `getComment()` wyszukuje pojedynczy dokument CommentThread na podstawie pola `_id` przekazanego jako parametr `commentId` w łańcuchu zapytania żądania GET.

Implementowanie procedury obsługi trasy `addComment()`

Procedura obsługi trasy `addComment()`, której kod widać na listingu 27.9, jest trochę bardziej złożona niż procedura `getComment()`, a ponadto uwzględniła łańcuch funkcji obsługujących aktualizowanie zagnieżdżonych komentarzy. Przebieg procesu dodawania nowych komentarzy jest następujący:

1. Klient wysłał żądanie zawierające identyfikator CommentThread, identyfikator komentarza nadrzędnego, do którego ma zostać dodany nowy komentarz, oraz obiekt JSON reprezentujący nowy komentarz.
2. Procedura obsługi trasy `/comment/add` wywołuje procedurę `addComment()`.
3. Obiekt CommentThread jest lokalizowany przy użyciu wartości `req.body.rootCommentId` z danych żądania POST.

4. Nowy obiekt Reply o nazwie newComment jest tworzony przy użyciu wartości req.body.newComment z danych żądania POST.
5. Procedura obsługi addComment() jest uruchamiana rekurencyjnie w celu przeszukiwania zagnieżdżonych komentarzy do momentu znalezienia komentarza, który jest zgodny z wartością req.body.parentCommentId przekazaną z treści żądania POST.
6. Nowy komentarz jest umieszczany w tablicy replies komentarza nadrzędnego.
7. Metoda updateCommentThread() używa następującej operacji update(), aby zaktualizować pole replies w dokumencie CommentThread za pomocą pola, które zawiera teraz zaktualizowany komentarz:


```
CommentThread.update({ _id: commentThread.id },
                      {$set: {replies: commentThread.replies}}).exec();
```
8. Odpowiedź informująca o powodzeniu lub niepowodzeniu jest odsyłana do klienta.

Listing 27.9. Plik comments_controller.js: implementowanie procedury obsługi trasy addComment() dla serwera Express

```

14 exports.addComment = function(req, res) {
15   CommentThread.findOne({ _id: req.body.rootCommentId })
16   .exec(function(err, commentThread) {
17     if (!commentThread){
18       res.json(404, {msg: 'Nie znaleziono modelu CommentThread.'});
19     } else {
20       var newComment = Reply(req.body.newComment);
21       newComment.username = generateRandomUsername();
22       addComment(req, res, commentThread, commentThread,
23                 req.body.parentCommentId, newComment);
24     }
25   });
26 };
27 function addComment(req, res, commentThread, currentComment,
28                    parentId, newComment){
29   if (commentThread.id == parentId){
30     commentThread.replies.push(newComment);
31     updateCommentThread(req, res, commentThread);
32   } else {
33     for(var i=0; i< currentComment.replies.length; i++){
34       var c = currentComment.replies[i];
35       if (c._id == parentId){
36         c.replies.push(newComment);
37         var replyThread = commentThread.replies.toObject();
38         updateCommentThread(req, res, commentThread);
39         break;
40       } else {
41         addComment(req, res, commentThread, c,
42                   parentId, newComment);
43       }
44     }

```

```
45 }
46 };
47 function updateCommentThread(req, res, commentThread){
48   CommentThread.update({ _id: commentThread.id },
49     {$set:{replies:commentThread.replies}})
50   .exec(function(err, savedComment){
51     if (err){
52       res.json(404, {msg: 'Nie zaktualizowano modelu CommentThread.'});
53     } else {
54       res.json({msg: "Powodzenie"});
55     }
56   });
57 }
```

Analiza procedury obsługi tras modelu Comment

Listing 27.10 prezentuje pełną implementację pliku *comments_controller.js*, który zapewnia obsługę tras dotyczącą modelu Comment. Kod tej procedury ładuje schemat dla modeli CommentThread i Reply. Model CommentThread zapewnia możliwość wyszukiwania dokumentów, a także aktualizowania ich po dodaniu nowych komentarzy. Model Reply tworzy obiekt Reply w momencie dodania nowego komentarza.

Listing 27.10. Plik *comments_controller.js*: pełna implementacja procedur obsługi tras modelu Comment dla serwera Express

```
01 var mongoose = require('mongoose'),
02     CommentThread = mongoose.model('CommentThread'),
03     Reply = mongoose.model('Reply');
04 exports.getComment = function(req, res) {
05   CommentThread.findOne({ _id: req.query.commentId })
06   .exec(function(err, comment) {
07     if (!comment){
08       res.json(404, {msg: 'Nie znaleziono modelu CommentThread.'});
09     } else {
10       res.json(comment);
11     }
12   });
13 };
14 exports.addComment = function(req, res) {
15   CommentThread.findOne({ _id: req.body.rootCommentId })
16   .exec(function(err, commentThread) {
17     if (!commentThread){
18       res.json(404, {msg: 'Nie znaleziono modelu CommentThread.'});
19     } else {
20       var newComment = Reply(req.body.newComment);
21       newComment.username = generateRandomUsername();
22       addComment(req, res, commentThread, commentThread,
23         req.body.parentCommentId, newComment);
24     }
25   });
26 };
```

```

27 function addComment(req, res, commentThread, currentComment,
28                     parentId, newComment){
29   if (commentThread.id == parentId){
30     commentThread.replies.push(newComment);
31     updateCommentThread(req, res, commentThread);
32   } else {
33     for(var i=0; i< currentComment.replies.length; i++){
34       var c = currentComment.replies[i];
35       if (c._id == parentId){
36         c.replies.push(newComment);
37         var replyThread = commentThread.replies.toObject();
38         updateCommentThread(req, res, commentThread);
39         break;
40       } else {
41         addComment(req, res, commentThread, c,
42                   parentId, newComment);
43       }
44     }
45   }
46 };
47 function updateCommentThread(req, res, commentThread){
48   CommentThread.update({ _id: commentThread.id },
49     {$set:{replies:commentThread.replies}})
50   .exec(function(err, savedComment){
51     if (err){
52       res.json(404, {msg: 'Nie zaktualizowano modelu CommentThread.'});
53     } else {
54       res.json({msg: "Powodzenie"});
55     }
56   });
57 }
58 function generateRandomUsername(){
59   // nazwa użytkownika będzie zwykle pochodzić z uwierzytelnionej sesji
60   var users=['Dawid', 'Karol', 'Bartek', 'Janek', 'Adam', 'Tomek'];
61   return users[Math.floor((Math.random()*5))];
62 }

```

Kod kontrolera komentarzy udostępnia funkcję, która losowo generuje nazwę użytkowników do celów testowych:

```

function generateRandomUsername(){
  //nazwa użytkownika będzie zwykle pochodzić z uwierzytelnionej sesji
  var users=['Dawid', 'Karol', 'Bartek', 'Janek', 'Adam', 'Tomek'];
  return users[Math.floor((Math.random()*6))];
}

```

Choć nazwa użytkownika będzie zwykle pochodzić z sesji w obiekcie Request, w tym miejscu pominięto kod sesji, aby ułatwić Ci przeanalizowanie przykładu.

Implementowanie widoków komentarzy i zdjęć

Po utworzeniu i skonfigurowaniu tras możesz rozpocząć implementowanie widoków renderowanych przez trasy i szablony AngularJS. W kolejnych punktach omówiono widok *photos.html* renderowany przez mechanizm EJS, a także częściowe widoki *comment.html* i *comment_thread.html*, które są udostępniane statycznie.

Implementowanie widoku zdjęć

Widok zdjęć zaprezentowany na listingu 27.11 to główny widok aplikacji w omawianym przykładzie. Widok jest ładowany w głównej trasie /. Nagłówek widoku rejestruje element `<html>` w aplikacji AngularJS i łączy plik *comment_styles.css*. Element `<body>` jest rozdzielany na dwie podstawowe sekcje.

Listing 27.11. Plik *photos.html*: implementowanie kodu głównego szablonu AngularJS strony ze zdjęciami

```
01 <!doctype html>
02 <html ng-app="myApp">
03 <head>
04   <title>Komentarze</title>
05   <link rel="stylesheet" type="text/css"
06     href="/static/css/comment_styles.css" />
07 </head>
08 <body>
09   <h2>Przykład komentarzy</h2>
10   <div ng-controller="photoController">
11     <div id="photosContainer">
12       <div class="photoItem" ng-repeat="photo in photos">
13         
15       </div>
16     </div>
17     <div>
18       <div id="photoContainer">
19         <p class="imageTitle">{{photo.title}}</p>
20         
22       </div>
23       <div id="photoComments"
24         ng-init="addComment=false;replySubject='';replyBody=''">
25         <div class="comment"
26           ng-include="'/static/comment_thread.html'"></div>
27       </div>
28     </div>
29   </div>
30   <div ng-controller="pageController">
31     <div id="pageComments"
32       ng-init="addComment=false;replySubject='';replyBody=''">
33       <div class="comment"
34         ng-include="'/static/comment_thread.html'"></div>
```

```

35 </div>
36 </div>
37 <script src="http://code.angularjs.org/1.2.9/angular.min.js"></script>
38 <script src="/static/js/comment_app.js"></script>
39 </body>
40 </html>

```

W górnej sekcji inicjowany jest kontroler `ng-controller="photoController"` środowiska AngularJS w celu zapewnienia funkcji umożliwiających interakcję ze zdjęciami z serwera WWW, w tym z dotyczącymi ich komentarzami. W dolnej sekcji inicjowany jest kontroler `ng-controller="pageController"`, aby udostępnić funkcje pozwalające na interakcję z komentarzami na stronie internetowej. Oba kontrolery służą do izolowania zasięgu, ponieważ implementują sekcje komentarzy.

Następujący kod definiuje szablon AngularJS, który renderuje tablicę dokumentów Photo przechowywaną w obiekcie `$scope.photos` modelu przy użyciu dyrektywy `ng-repeat`:

```

<div id="photosContainer">
  <div class="photoItem" ng-repeat="photo in photos">
    
  </div>
</div>

```

Zauważ, że w celu ustawienia bieżącego zdjęcia w kodzie kontrolera dla dyrektywy `ng-click` ustawiono metodę `setPhoto()`. Na rysunku 27.2 pokazano element wyrenderowany na stronach.



Rysunek 27.2. Zdjęcia renderowane w przeglądarce jako elementy ``

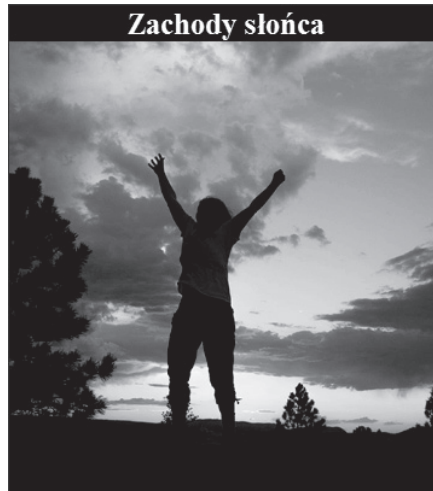
Następujący kod szablonu wyświetla na ekranie bieżący obiekt `$scope.photo` przy użyciu właściwości `photo.title`. Ponadto za pomocą właściwości `photo.filename` kod ustawia dla atrybutu `src` element ``:

```

<div id="photoContainer">
  <p class="imageTitle">{{photo.title}}</p>
  
</div>

```

Na rysunku 27.3 pokazano wyświetloną część strony ze zdjęciem.



Rysunek 27.3. Większy widok zdjęcia i tytułu wyświetlony w przeglądarce

Część strony internetowej z komentarzami dodajesz za pomocą poniższego kodu, który używa dyrektywy `ng-init` do inicjowania wartości `addComment`, `replySubject` i `replyBody` w bieżącym zasięgu, a następnie korzysta z dyrektywy `ng-include`, aby dołączyć kod częściowego szablonu `comment_thread.html` opisanego w następnym punkcie.

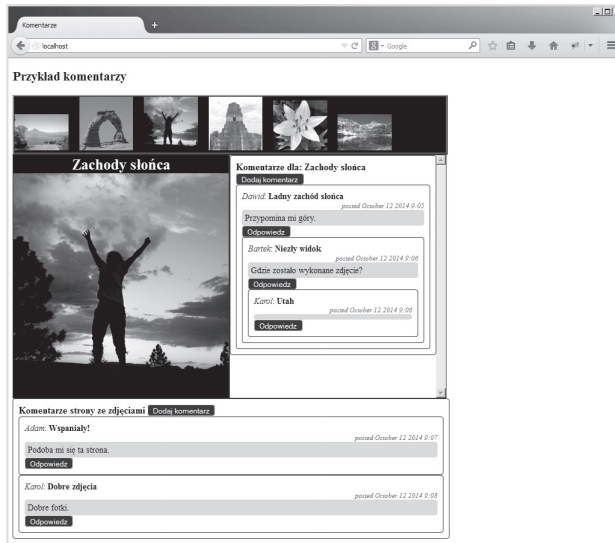
```
<div id="photoComments"
  ng-init="addComment=false;replySubject='';replyBody=''">
  <div class="comment"
    ng-include="/static/comment_thread.html"></div>
</div>
```

Sekcja podobna do powyższej jest dodawana do części strony ze zdjęciami, a także na dole strony. Umożliwia to wstawianie komentarzy w obu miejscach. Wartość `addComment` jest używana w szablonie `comment_thread.html` do określenia, czy komentarze mają zostać włączone, czy wyłączone.

Na rysunku 27.4 pokazano pełną aplikację z komentarzami w sekcji zdjęć oraz na dole strony.

Implementowanie widoku wątku komentarzy

Jak wspomniano w poprzednim punkcie, sekcja zdjęć i sekcja głównego obrazu strony ze zdjęciami korzystają z dyrektywy `ng-include`, aby dołączyć szablon częściowy `comment_thread.html`, który został zaprezentowany na listingu 27.12. Szablon ten został wydzielony, aby łatwiej było go w późniejszym czasie dołączyć do innych stron, a nawet dodatkowych sekcji tej samej strony.



Rysunek 27.4. Wyświetlona strona zdjęć z komentarzami w obszarze zdjęć i na dole strony

Listing 27.12. Plik `comment_thread.html`: implementowanie częściowego szablonu wątku komentarzy

```

01 <span class="commentTitle">{{commentThread.title}}</span>
02 <input type="button" ng-show="addComment==false"
03     value="Dodaj komentarz" ng-click="addComment=true"></input>
04 <form ng-show="addComment==true">
05   <label>Temat</label>
06   <input type="text" ng-model="replySubject"></input>
07   <label>Komentarz</label>
08   <textarea ng-model="replyBody"></textarea>
09   <input type="button" value="Wyślij"
10     ng-click=
11       "addComment=false; addReply(commentThread._id,replySubject,replyBody)"
12   ></input>
13 </form>
14 <input type="button" ng-show="addComment==true"
15     value="Anuluj" ng-click="addComment=false;"></input>
16 <div ng-repeat="comment in commentThread.replies"
17     ng-init="reply=false;replySubject='';replyBody=''">
18   <div class="comment" ng-include="'/static/comment.html'"></div>
19 </div>

```

Szablon częściowy uwzględnia tytuł `CommentThread` przy użyciu kodu `{{commentThread.title}}`. Przycisk elementu `<input>` przełącza atrybut `addComment` na wartość `true`, umożliwiając wyświetlenie formularza odpowiedzi (aktualnie ukryto go za pomocą dyrektywy `ng-show`). W obrębie tego formularza definiowany jest element `<input>` wraz z treścią komentarza `<textarea>` (rysunek 27.5). Przycisk *Dodaj komentarz*

powoduje wykonanie metody `addReply()` w zasięgu i przekazanie kontrolerowi wartości `commentThread._id`, `replySubject` i `replyBody` w celu wysłania do serwera żądania dodania komentarza.

Rysunek 27.5. Obszar wątku komentarzy z przyciskiem Dodaj komentarz i formularz służący do dodania komentarza

Na rysunku 27.5 pokazano widok wątku komentarzy wyrenderowany przed kliknięciem przycisku *Dodaj komentarz* oraz formularz odpowiedzi, który zostanie otwarty po kliknięciu tego przycisku.

Na dole szablonu wątku komentarzy w poniższych wierszach kodu używana jest dyrektywa `ng-repeat` do załadowania szablonu częściowego `comment.html`, który renderuje komentarz za pomocą dyrektywy `ng-include`. Zauważ też, że inicjowane są wartości `reply`, `replySubject` i `replyBody`. Są one używane w szablonie komentarzy do wyświetlania/ukrywania formularza odpowiedzi, a także do przekazywania danych do funkcji `addReply()`:

```
<div ng-repeat="comment in commentThread.replies"
  ng-init="reply=false;replySubject='';replyBody=''">
  <div class="comment" ng-include="'/static/comment.html'"></div>
</div>
```

Implementowanie widoku komentarzy

Jak wspomniano w poprzednim punkcie, sekcja wątku komentarzy strony używa dyrektywy `ng-include` do dołączenia szablonu częściowego `comment.html` (jego kod został zaprezentowany na listingu 27.13) dla każdej odpowiedzi w tablicy `replies` obiektu komentarzy.

Listing 27.13. Plik `comment_thread.html`: implementowanie częściowego szablonu komentarzy

```
01 <span class="username">{{comment.username}}</span>:
02 <span class="subject">{{comment.subject}}</span>
03 <p class="timestamp"
04 >posted {{comment.timestamp|date:"MMMM d yyyy H:mm"}}</p>
```

```

05 <p class="commentBody">{{comment.body}}</p>
06 <input type="button" ng-show="reply==false"
07     value="Odpowiedz" ng-click="reply=true"></input>
08 <form ng-if="reply">
09   <label>Temat</label>
10   <input type="text" ng-model="replySubject"></input>
11   <label>Komentarz</label>
12   <textarea ng-model="replyBody"></textarea>
13   <input type="button" value="Wyślij"
14     ng-click="addReply(comment._id,replySubject,replyBody)" />
15 </form>
16 <input type="button" ng-show="reply==true"
17     value="Anuluj" ng-click="reply=false;"></input>
18 <div ng-repeat="comment in comment.replies"
19     ng-init="reply=false;replySubject='';replyBody=''">
20   <div class="comment" ng-include="'/static/comment.html'"></div>
21 </div>

```

Korzystając na tym etapie z szablonu częściowego, możesz zagnieźdzać odpowiedzi w innych odpowiedziach po prostu przez wczytanie tego samego bloku (zostało to pokazane poniżej) do przeprowadzenia iteracji odpowiedzi za pomocą dyrektywy `ng-repeat`. Do dołączenia tego samego formularza stosowana jest też dyrektywa `ng-include`.

```

<div ng-repeat="comment in comment.replies"
    ng-init="reply=false;replySubject='';replyBody=''">
  <div class="comment" ng-include="'/static/comment.html'"></div>
</div>

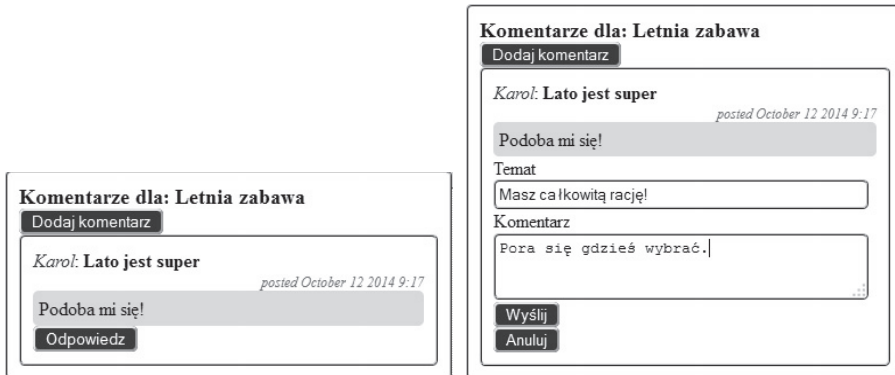
```

Szablon częściowy *comment.html* zawiera nazwę użytkownika, temat, znacznik czasu i treść komentarza. Przycisk elementu `<input>` przełącza atrybut `reply` na wartość `true`, umożliwiając wyświetlenie formularza odpowiedzi (aktualnie ukryto go za pomocą dyrektywy `ng-show`). Formularz odpowiedzi widoczny na rysunku 27.6 przypomina formularz służący do dodania pierwszego komentarza. Przycisk *Odpowiedz* powoduje wykonanie metody `addReply()` w zasięgu i przekazanie kontrolerowi wartości `comment._id`, `replySubject` i `replyBody` w celu wysłania do serwera żądania dodania komentarza.

Na rysunku 27.6 pokazano widok wątku odpowiedzi wyrenderowany przed kliknięciem przycisku *Odpowiedz* oraz formularz odpowiedzi, który zostanie otwarty po kliknięciu tego przycisku.

Dodawanie kodu CSS w celu określenia stylów widoków

Listing 27.14 prezentuje kod CSS używany do określenia stylów dla elementów widoku zdjęć, a także szablonów częściowych AngularJS.



Rysunek 27.6. Udzielanie odpowiedzi na komentarz znajdujący się w wątku

Listing 27.14. Plik comment_styles.css: implementowanie stylów CSS dla plików HTML widoków

```

01 div#photosContainer{
02   background:black; color: white;
03   border: 4px ridge blue; width:800px;
04 }
05 div.photoItem{ display:inline-block; width:120px; }
06 div#photoContainer,
07 div#photoComments{
08   background:black; color: white; width: 400px; height:450px;
09   display: inline-block; float:left; border: 2px ridge blue;
10 }
11 div#photoComments{ background:white; color: black;
12   max-height:500px; overflow-y:scroll;
13 }
14 div#pageComments{ margin:0px; clear:both; width:810px; }
15 div.comment{ border: 2px ridge blue; padding:10px;
16   border-radius: 5px;
17 }
18 img.listPhoto{ width:100px; }
19 img.mainPhoto{ width:400px; }
20 .commentBody{ background-color:lightgray;
21   border-radius: 5px; padding: 5px; margin:0;
22 }
23 .imageTitle{ font-size: 28px; font-weight: bold;
24   text-align: center; margin:0
25 }
26 .commentTitle{ font-size: 18px; font-weight: bold;}
27 .username{ font-style: italic; color:blue; }
28 .subject{ font-weight: bold;}
29 .timestamp{ color:#555555; font-style: italic;
30   font-size: 12px; text-align: right; margin:0;
31 }
32 input[type="text"],
33 textarea{ border: 2px ridge blue; border-radius: 5px;
34   padding:3px; width:95%;

```

```

35 }
36 input[type="button"]{ background-color:blue;color:white;
37 border-radius: 5px;
38 }

```

Implementowanie modułu i kontrolerów środowiska AngularJS do obsługi widoków komentarzy

Po zakończeniu tworzenia widoków niezbędne jest zaimplementowanie kodu kontrolera środowiska AngularJS do ich obsługi. Widoki wymagają możliwości uzyskania strony, zdjęć i komentarzy, a ponadto dodania nowych komentarzy.

Należy udostępnić dwa kontrolery: po jednym dla komentarzy dotyczących zdjęć i komentarzy na pełnej stronie. Ponieważ istnieje wiele kontrolerów, które wymagają interakcji z serwerem w celu pobierania i dodawania komentarzy, dobrym pomysłem jest też utworzenie usługi obsługującej te dwie operacje. W dalszej części rozdziału opisano proces tworzenia kompletnego kodu aplikacji AngularJS, który został zaprezentowany na listingu 27.18.

Tworzenie usługi komentarzy

Ze względu na to, że do obsługi pobierania i dodawania komentarzy niezbędna będzie usługa komentarzy, w pierwszej kolejności musisz ją dodać. Kod z listingu 27.15 implementuje usługę komentarzy, która korzysta z metody usługi. Konieczne jest utworzenie definicji funkcji o nazwie `CommentObj`. Jako jedyny parametr funkcja konstruktora akceptuje usługę `$http`.

Listing 27.15. Plik `comment_app.js` (`commentSrv`): implementowanie usługi AngularJS, która zapewnia funkcje wielokrotnego użycia służące do pobierania komentarzy z serwera oraz dodawania na nim nowych komentarzy

```

02 function CommentObj($http) {
03   this.getComment = function(commentId, callback){
04     $http.get('/comments/get', {params: {commentId: commentId}})
05       .success(function(data, status, headers, config) {
06         callback(null, data);
07       })
08       .error(function(data, status, headers, config) {
09         callback(data, {});
10       });
11   };
12   this.addComment = function(rootCommentId, parentId,
13                             newComment, callback){
14     $http.post('/comments/add', { rootCommentId: rootCommentId,
15                               parentCommentId: parentId,
16                               newComment: newComment })

```

```
17     .success(function(data, status, headers, config) {
18         callback(null, data);
19     })
20     .error(function(data, status, headers, config) {
21     });
22 };
23 }
24 app.service('commentSrv', ['$http', CommentObj]);
```

Definiowana jest funkcja `getComment()`, która akceptuje parametr `commentId`, a następnie kieruje żądanie GET do trasy `/comment/get` w celu pobrania danych całego komentarza. Zauważ, że funkcja akceptuje też funkcję wywołania zwrotnego. Zwraca ona błąd (jeśli zostanie zgłoszony przez usługę `$http`) lub dane komentarza z serwera.

Funkcja `addComment()` akceptuje parametr `rootCommentId` dla wątku komentarzy, parametr `parentId` dla komentarza nadrzędnego i parametr `newComment`, który jest obiektem JavaScript z właściwościami `subject` i `body`. Funkcja kieruje żądanie POST do trasy `/comment/add` w celu dodania nowego komentarza do bazy danych MongoDB. Podobnie jak w przypadku funkcji `getComment()`, wykonywana jest funkcja wywołania zwrotnego i zwracane są wyniki. Przy użyciu następującego wiersza tworzona jest następnie usługa `commentSrv` w celu wstrzyknięcia usługi `$http`:

```
app.service('commentSrv', ['$http', CommentObj]);
```

Możliwe jest teraz wstrzyknięcie usługi `commentSrv` do używanych kontrolerów.

Implementowanie kontrolera zdjęć

W dalszej kolejności konieczne jest zaimplementowanie kontrolera do obsługi części strony powiązanej ze zdjęciami. Listing 27.16 prezentuje kod kontrolera `photoController`. Zauważ, że do kontrolera jest wstrzykiwana usługa `commentSrv` wraz z usługą `$http`.

Listing 27.16. Plik `comment_app.js` (`photoController`): implementowanie kontrolera środowiska AngularJS, który obsługuje części widoku powiązane ze zdjęciami i dotyczącymi ich komentarzami

```
25 app.controller('photoController', ['$scope', '$http', 'commentSrv',
26     function($scope, $http, commentSrv) {
27     $http.get('/photos')
28     .success(function(data, status, headers, config) {
29         $scope.photos = data;
30         $scope.photo = $scope.photos[0];
31         $scope.loadComments();
32     })
33     .error(function(data, status, headers, config) {
34         $scope.photos = [];
35     });
36     $scope.loadComments = function(){
37         commentSrv.getComment($scope.photo.commentId,
38             function(err, comment){
39             if(err){
```

```

40     $scope.commentThread = {};
41   } else {
42     $scope.commentThread = comment;
43   }
44 });
45 };
46 $scope.addReply = function(parentCommentId, subject, body){
47   var newComment = {subject:subject, body:body};
48   commentSrv.addComment($scope.commentThread._id,
49                         parentCommentId,
50                         newComment, function(err, comment){
51     $scope.loadComments();
52   });
53 };
54 $scope.setPhoto = function(photoId){
55   $http.get('/photo', {params: {photoId: photoId}})
56     .success(function(data, status, headers, config) {
57     $scope.photo = data;
58     $scope.loadComments();
59   })
60     .error(function(data, status, headers, config) {
61     $scope.photo = {};
62   });
63 };
64 }]);

```

Kontroler inicjuje obiekty `$scope.photos` i `$scope.photo` przy użyciu żądania GET usługi `$http` kierowanego do trasy `/photos`. Funkcja `$scope.loadComments()` używa funkcji `commentSrv.getComment()` do pobrania komentarzy dla bieżącego zdjęcia przez przekazanie parametru `$scope.photo.commentId`. Funkcja wywołania zwrotnego po prostu ustawia wartość `$commentThread` używaną w widoku.

Metoda `$scope.addReply()` jest wywoływana po kliknięciu przez użytkownika przycisku *Wyślij* dla odpowiedzi do komentarza w szablonie. Zauważ, że metoda generuje najpierw obiekt `newComment` z tematem i treścią przekazywanymi z widoku, a następnie korzysta z metody `commentSrv.addComment()`, aby wysłać aktualizację do serwera.

Funkcja `$setPhoto()` akceptuje wartość `photoID` z interfejsu użytkownika, po czym kieruje żądanie GET usługi `$http` do trasy `/photo` w celu pobrania obiektu zdjęcia. Obiekt `$scope.photo` jest aktualizowany przy użyciu danych, a następnie wywoływana jest funkcja `$scope.loadComments()` do załadowania komentarzy dla nowego folderu.

Implementowanie kontrolera środowiska AngularJS dla strony

Następnym wymaganym krokiem jest implementacja kontrolera do obsługi komentarzy na stronie internetowej. Kod z listingu 27.17 zawiera kod kontrolera `pageController`. Zauważ, że do tego kontrolera również jest wstrzykiwana usługa `commentSrv` wraz z usługą `$http`.

Listing 27.17. Plik `comment_app.js` (`pageController`): implementowanie kontrolera środowiska AngularJS, który obsługuje część widoku związaną z komentarzami na stronie

```
65 app.controller('pageController', ['$scope', '$http', 'commentSrv',
66                               function($scope, $http, commentSrv) {
67     $http.get('/page', {params:{pageName:"Strona ze zdjęciami"}})
68     .success(function(data, status, headers, config) {
69       $scope.page = data;
70       $scope.loadComments();
71     })
72     .error(function(data, status, headers, config) {
73       $scope.Page = {};
74     });
75     $scope.addReply = function(parentCommentId, subject, body){
76       var newComment = {subject:subject, body:body};
77       commentSrv.addComment($scope.commentThread._id,
78                             parentCommentId,
79                             newComment, function(err, comment){
80         $scope.loadComments();
81       });
82     };
83     $scope.loadComments = function(){
84       commentSrv.getComment($scope.page.commentId,
85                             function(err, comment){
86         if(err){
87           $scope.commentThread = {};
88         } else {
89           $scope.commentThread = comment;
90         }
91       });
92     };
93   }]);
```

Kontroler inicjuje wartość `$scope.page`, tworzy żądanie GET usługi `$http` kierowane do trasy `/pages` i przekazuje jako parametr umieszczoną na stałe w kodzie nazwę "Strona ze zdjęciami". Funkcja `$scope.loadComments()` wywołuje funkcję `commentSrv.getComment()`, aby pobrać komentarze dla strony przy użyciu wartości `$scope.page.commentId`. Funkcja wywołania zwrotnego ustawia po prostu wartość `$commentThread` używaną w widoku.

Metoda `$scope.addReply()` jest wywoływana po kliknięciu przez użytkownika przycisku *Wyślij* w szablonie komentarzy na stronie. Zauważ, że metoda generuje najpierw obiekt `newComment` z tematem i treścią przekazanymi z widoku, a następnie używa metody `commentSrv.addComment()` do wysłania aktualizacji do serwera.

Kompletna aplikacja AngularJS

Listing 27.18 prezentuje w całości kod kompletnej aplikacji AngularJS. Zauważ, że w wierszu 1. definiowana jest aplikacja, a następnie w dalszych wierszach kodu dodawana jest usługa `commentSrv` oraz kontrolery `photoController` i `pageController`.

Listing 27.18. Plik comment_app.js: implementowanie aplikacji AngularJS obsługującej komentarze na stronach internetowych

```

01 var app = angular.module('myApp', []);
02 function CommentObj($http) {
03   this.getComment = function(commentId, callback){
04     $http.get('/comments/get', {params: {commentId: commentId}})
05       .success(function(data, status, headers, config) {
06         callback(null, data);
07       })
08       .error(function(data, status, headers, config) {
09         callback(data, {});
10       });
11   };
12   this.addComment = function(rootCommentId, parentId,
13                             newComment, callback){
14     $http.post('/comments/add', { rootCommentId: rootCommentId,
15                                parentCommentId: parentId,
16                                newComment: newComment })
17       .success(function(data, status, headers, config) {
18         callback(null, data);
19       })
20       .error(function(data, status, headers, config) {
21       });
22   };
23 }
24 app.service('commentSrv', ['$http', CommentObj]);
25 app.controller('photoController', ['$scope', '$http', 'commentSrv',
26                                function($scope, $http, commentSrv) {
27   $http.get('/photos')
28     .success(function(data, status, headers, config) {
29       $scope.photos = data;
30       $scope.photo = $scope.photos[0];
31       $scope.loadComments();
32     })
33     .error(function(data, status, headers, config) {
34       $scope.photos = [];
35     });
36   $scope.loadComments = function(){
37     commentSrv.getComment($scope.photo.commentId,
38                          function(err, comment){
39       if(err){
40         $scope.commentThread = {};
41       } else {
42         $scope.commentThread = comment;
43       }
44     });
45   };
46   $scope.addReply = function(parentCommentId, subject, body){
47     var newComment = {subject:subject, body:body};
48     commentSrv.addComment($scope.commentThread._id,
49                           parentCommentId,
50                           newComment, function(err, comment){
51       $scope.loadComments();

```

```

52     });
53   };
54   $scope.setPhoto = function(photoId){
55     $http.get('/photo', {params: {photoId: photoId}})
56       .success(function(data, status, headers, config) {
57         $scope.photo = data;
58         $scope.loadComments();
59       })
60       .error(function(data, status, headers, config) {
61         $scope.photo = {};
62       });
63   };
64 });
65 app.controller('pageController', ['$scope', '$http', 'commentSrv',
66   function($scope, $http, commentSrv) {
67     $http.get('/page', {params:{pageName:"Strona ze zdjęciami"}})
68       .success(function(data, status, headers, config) {
69         $scope.page = data;
70         $scope.loadComments();
71       })
72       .error(function(data, status, headers, config) {
73         $scope.Page = {};
74       });
75     $scope.addReply = function(parentCommentId, subject, body){
76       var newComment = {subject:subject, body:body};
77       commentSrv.addComment($scope.commentThread._id,
78         parentCommentId,
79         newComment, function(err, comment){
80         $scope.loadComments();
81       });
82     };
83     $scope.loadComments = function(){
84       commentSrv.getComment($scope.page.commentId,
85         function(err, comment){
86         if(err){
87           $scope.commentThread = {};
88         } else {
89           $scope.commentThread = comment;
90         }
91       });
92     };
93   });

```

Inicjowanie aplikacji

Po utworzeniu aplikacji konieczne jest też utworzenie dokumentów Page i Photo w bazie danych. Można to zrobić na kilka różnych sposobów. Na przykład możesz dodać taką opcję bezpośrednio do aplikacji. Jest to dobry wariant, jeśli te dokumenty okażą się niezbędne w późniejszym czasie. Innym rozwiązaniem jest użycie bezpośrednio w powłoce bazy danych MongoDB skryptu JavaScript, który zawiera polecenia służące do utworzenia

wymienionych dokumentów. Możliwe jest również napisanie prostego skryptu w środowisku Node.js lub za pomocą innego języka, który zapewnia dostęp do bazy danych MongoDB.

Listing 27.19 prezentuje podstawowy skrypt, który tworzy dokument Page i dodaje kilka dokumentów Photo. Każdy dokument Page i Photo zawiera też dokument CommentThread utworzony i powiązany z odwołaniem commentId.

Listing 27.19. Plik comment_init.js: implementowanie skryptu Node.js, który inicjuje dane dla aplikacji

```

01 var mongoose = require('mongoose');
02 var db = mongoose.connect('mongodb://localhost/comments');
03 require('./models/comments_model.js');
04 require('./models/photo_model.js');
05 require('./models/page_model.js');
06 var CommentThread = mongoose.model('CommentThread');
07 var Reply = mongoose.model('Reply');
08 var Photo = mongoose.model('Photo');
09 var Page = mongoose.model('Page');
10 function addPhoto(title, filename){
11   var comment = new CommentThread({title: "Komentarze dla: " + title});
12   comment.save(function(err, comment){
13     var photo = new Photo({title:title, filename:filename});
14     photo.commentId = comment.id;
15     photo.save(function(){
16       console.log("Zapisano: " + title);
17     });
18   });
19 }
20 CommentThread.remove().exec(function(){
21   Photo.remove().exec(function(){
22     Page.remove().exec(function(){
23       var comment = new CommentThread({title:"Komentarze strony ze zdjęciami"});
24       comment.save(function(err, comment){
25         var page = new Page({name:"Strona ze zdjęciami"});
26         page.commentId = comment.id;
27         page.save();
28       });
29       addPhoto('Wytrzymałość', 'arch.jpg');
30       addPhoto('Siła', 'pyramid.jpg');
31       addPhoto('Piękno', 'flower.jpg');
32       addPhoto('Zamyślony', 'boy.jpg');
33       addPhoto('Letnia zabawa', 'boy2.jpg');
34       addPhoto('Zachody słońca', 'jump.jpg');
35     });
36   });
37 });

```

Podsumowanie

W tym rozdziale wyjaśniono, jak zdefiniować model do przechowywania i pobierania zagnieżdżonych komentarzy. Opisany tu model przechowuje pełny wątek komentarzy jako pojedynczy dokument zamiast osobnych dokumentów dla każdego komentarza. Ułatwia to wczytanie całego wątku.

W rozdziale zaimplementowano też podstawowy model dla strony internetowej i zdjęć. Ponadto zintegrowano wiele modeli z procedurami obsługi tras serwera Express, szablonami AngularJS i aplikacją AngularJS.

W następnym rozdziale

W następnym rozdziale zajmiesz się kolejnym praktycznym przykładem dodawania produktów i koszyka zakupów do witryny internetowej.

Skorowidz

A

administrator
 baz danych, 247
 użytkowników, 245
adres URL, 25, 141
agregowanie
 danych, 353, 354
 dokumentów, 351
 wyników, 319
aktualizowanie
 dokumentów, 287, 289,
 344–348
 obiektów, 283
analiza
 danych JSON, 414
 procedury obsługi tras,
 567
 łańcucha adresu URL,
 143
AngularJS, 20, 31, 431
 cykl życia aplikacji, 436
 definiowanie modułów,
 448
 dodawanie do Node.js,
 438
 dodawanie dostawców,
 450
 dyrektywy, 485
 filtry, 477
 implementowanie
 dyrektyw, 485
 implementowanie usług,
 511

inicjowanie
 w dokumencie HTML,
 438
integrowanie
 komponentu, 437
kod wielokrotnego
 użycia, 21
konfiguracja modułu,
 449
kontrolery, 434
model danych, 433
moduły, 433, 447, 448
 przejrzystość, 21, 31
 rozszerzalność, 21, 31
 tworzenie aplikacji, 441
 tworzenie widoków, 469
uruchamianie modułu,
 450
usługi, 435
wiązanie danych, 21, 31,
 434
wielokrotne
 wykorzystanie kodu,
 31
wstrzykiwanie
 zależności, 435, 448,
 452
wyrażenia, 434
zakresy
 odpowiedzialności, 435
 zasięg, 433, 455
 zgodność, 21, 31
animacja, 522, 523
animacja jQuery, 525

API, 439
aplikacja
 AngularJS, 579
 myApp, 616
aplikacje RIA, 623
asynchroniczna metoda
 odczytu, 130
asynchroniczne
 strumieniowanie
 danych, 126, 131
 wywołania, 120
atak
 CSRF, 414
 typu man-in-the-middle,
 163, 181
atrybuty, 492
atrybuty obiektów Stats, 134

B

baza danych
 dodawanie indeksów,
 361
 kopia zapasowa, 384
 kopiowanie, 249
 MongoDB, 20, 361
 naprawianie, 383
 NoSQL, 223
 operatory
 aktualizowania, 283
 replikację, 365
 tworzenie, 248, 274
 usuwanie, 249, 274
 wprowadzanie zmian, 281

- bazowanie na dokumentach, 30
 - biblioteka
 - angular.js, 438, 439, 443
 - jQuery, 431, 524
 - Mongoose, 326, 341, 536, 560
 - agregowanie dokumentów, 351
 - aktualizowanie dokumentów, 344–347
 - dodawanie dokumentów, 343
 - funkcje pośrednie, 356
 - sprawdzanie poprawności, 354
 - usuwanie dokumentów, 349, 350
 - znajdowanie dokumentów, 341
 - OpenSSL, 182
 - blok
 - konfiguracji, 449
 - try/catch, 54
 - uruchamiania, 450
 - blokujące wejście-wyjście, 77
 - błędy, 53, 355
 - bufor
 - dzielenie na segmenty, 102
 - kopiowanie, 101
 - metody kodowania, 97
 - metody odczytywania, 100
 - metody zapisywania, 98
 - określanie długości, 101
 - złączanie, 104
 - buforowanie danych, 97
- C**
- CDN, Content Delivery Network, 438
 - certyfi­kat samopodpisany, 182
 - CSS, 523
 - cykl życia
 - aplikacji
 - faza inicjowania, 436
 - faza kompilacji, 436
 - faza wiązania danych, 437
 - danych, 232
 - zasięgu
 - faza mutacji modelu, 461
 - faza obserwowania mutacji, 461
 - faza rejestrowania elementu monitorującego, 460
 - faza tworzenia, 460
 - faza usuwania zasięgu, 461
- D**
- dane, 26
 - binarne, 97
 - buforowane, 97
 - HTML, 492
 - JSON, 95, 402
 - pogodowe, 635
 - zaplacza, 459
 - definiowanie
 - funkcji, 42
 - głównego elementu, 443
 - kontrolera usługi pogodowej, 634
 - mechanizmu szablonów, 406
 - modelu, 588
 - CommentThread, 560
 - komentarzy, 559
 - Page, 559
 - Photo, 560
 - projektu, 626
 - użytkowników, 535
 - modułów AngularJS, 448
 - operatorów zapytania, 337
 - schematu, 328, 331
 - adresów, 589
 - ilości, 590
 - klientów, 591
 - produktów, 590
 - rachunków, 589
 - zamówień, 590
 - szablonu
 - kart, 629
 - panelu, 628
 - widoku, 501
 - zmiennych, 33
 - dekompresowanie
 - buforów, 115
 - danych, 115
 - strumieni, 117
 - denormalizowanie danych, 229
 - deserializacja, 551
 - deskryptor pliku, 121
 - diagram komponentów, 24
 - długość
 - bufora, 101
 - łańcucha, 48
 - dodawanie
 - dokumentów, 283, 343
 - elementów do tablicy, 53
 - indeksów, 361
 - kodu CSS, 548, 574, 610
 - komentarzy, 563
 - kont użytkowników, 533
 - kontrolera do dyrektywy, 505
 - kontrolera do szablonu, 444
 - modułu passport, 550
 - obiektu locals, 407
 - obsługi błędów, 53
 - produktów do koszyka, 612
 - segmentów danych, 375

- serializacji
 - i deserializacji, 551
- strategii
 - uwierzytelniania, 550
 - wątków komentarzy, 557
 - zadań, 79
 - zdarzeń
 - niestandardowych, 85
- dokonywanie
 - platności, 614
 - zakupu, 616
- dokumenty, 224
- DOM, Document Object Model, 25
- domknięcie, 89
- dostawca
 - filtru, 474
 - obiektów komponentu
 - AngularJS, 451
 - usług, 451
- dostęp do
 - baz danych, 245, 273, 299, 459
 - danych dynamicznych, 641
 - kolekcji, 276
 - MongoDB, 238, 257
 - plików, 135
 - serwera WWW, 155
 - systemów zewnętrznych, 160
 - systemu plików, 119
 - trasy, 422
 - właściwości obiektu
 - Request, 398
 - zasięgu, 471
- drzewo, 25
- dynamiczny serwer WWW, 155
- dyrektywa, 433, 470, 485
 - a, 488
 - form, 489
 - input, 489
 - input.checkbox, 489
 - input.email, 489
 - input.number, 489
 - input.radio, 490
 - input.text, 490
 - input.url, 490
 - myDirective, 503
 - ngApp, 486
 - ngBind, 492
 - ngBindHtml, 493
 - ngBindTemplate, 493
 - ngBlur, 497
 - ngChange, 497
 - ngChecked, 497
 - ngClass, 493, 522
 - ngClassEven, 493
 - ngClassOdd, 493
 - ngClick, 497
 - ngCloak, 486
 - ngController, 486
 - ngCookies, 519
 - ngCopy, 497
 - ngCut, 497
 - ngDbclick, 497
 - ngDisabled, 493
 - ngFocus, 497
 - ngHide, 493, 522
 - ngHref, 486
 - ngIf, 494, 522
 - ngInclude, 487, 522
 - ngInit, 494
 - ngKeydown, 498
 - ngKeypress, 498
 - ngKeyUp, 498
 - ngList, 487
 - ngModel, 494
 - ngMouseDown, 498
 - ngMouseenter, 498
 - ngMouseleave, 498
 - ngMouseMove, 498
 - ngMouseover, 498
 - ngMouseup, 498
 - ngNonBindable, 487
 - ngOpen, 487
 - ngOptions, 490
 - ngPaste, 498
 - ngPluralize, 487
 - ngReadOnly, 487
 - ngRepeat, 494, 522
 - ngRequired, 487
 - ngSelected, 488
 - ngShow, 493, 522
 - ngSrc, 488
 - ngSrcset, 488
 - ngStyle, 495
 - ngSubmit, 498
 - ngSwipeLeft, 498
 - ngSwipeRight, 498
 - ngSwitch, 495, 522
 - ngTransclude, 488, 501
 - ngValue, 495
 - ngView, 488, 522
 - script, 488
 - select, 490
 - richTabs, 631
- dyrektywy
 - automatycznie usuwane, 523
 - niestandardowe, 506, 630
 - obsługujące funkcje szablonu, 486–488
 - połączone z danymi, 517
 - rozszerzające elementy formularza, 488
 - w package.json, 67
 - wbudowane, 486
 - wiązania danych, 495, 497
 - wiążące model z elementami, 491
 - wiążące zdarzenia z kontrolerami, 496
 - własne, 500
- dziedziczenie
 - funkcji, 216
 - prototypów, 216
- dzielenie buforów, 102

E

efekty przejścia, 526
 EJS, Embedded JavaScript, 406
 elementy

- łańcucha połączenia, 264
- monitorujące, 460

 emiter zdarzeń, 85, 87, 463
 Express, 30

- funkcje pośrednie, 413, 414
- implementowanie tras, 392
- informacje cookie, 30
- integracja, 30
- konfigurowanie
 - ustawień, 390
- obsługa błędów, 30
- sesje, 30
- uruchamianie serwera, 391
- zarządzanie pamięcią
 - podręczną, 30
- zarządzanie trasami, 30

F

filtr, 470, 473

- currency, 474, 476
- date, 475, 476
- filter, 474
- json, 476
- limitTo, 475, 476
- lowercase, 475
- number, 475
- orderBy, 475, 479
- uppercase, 470, 475

 filtrowanie

- elementów, 479
- elementów
 - dynamicznych, 478

 filtry

- niestandardowe, 479
- wbudowane, 474

format

- Big Endian, 97
- BSON, 226
- JSON, 95, 401
- Little Endian, 97
- XML, 95

 formatowanie łańcuchów, 213
 formularz, 144
 funkcja, 42

- .addListener(), 86
- .listeners(), 86
- .on(), 86
- .once(), 86
- .removeListener(), 86
- .setMaxListeners(), 86
- addCity(), 636
- addComment(), 577
- addToCart(), 601
- addValues(), 457, 458
- cartTotal(), 612
- changeName(), 466
- checkout(), 614
- combine(), 472
- deleteFromCart(), 602, 613
- emit(), 85
- exec(), 194
- execFile(), 195, 196
- fitlerText(), 481
- fork(), 200
- fs.openSync(), 122
- fs.write(), 125
- fs.writeSync(), 123
- getComment(), 577
- getSortObj(), 642
- getWeather(), 161, 635
- getWords(), 643
- GridFS, 361
- hasPW(), 426
- inspect(), 216
- link(), 505
- makePurchase(), 616
- mkdirSync(), 138
- nextTick(), 84

open(), 125, 131
 openSync(), 123, 131
 parseWeather(), 633
 pipe(), 114
 pośrednia

- basic-auth-connect, 423

 pośrednia body-parser, 418, 426
 pośrednia cookie-parser, 421
 pośrednia cookie-session, 421
 pośrednia query, 416
 pośrednia static, 416
 Query(), 76
 read(), 131
 readdirSync(), 136
 regenerate(), 426
 req.param(), 395
 rmdirSync(), 138
 Send(), 76
 setFilter(), 479
 setImmediate(), 83
 setInterval(), 81
 setProduct(), 600
 setShipping(), 615
 setSort(), 479
 setTimeout(), 80
 simpleTimeout(), 80
 spawn(), 197, 198
 truncateSync(), 137
 unlinkSync(), 136
 unref(), 83
 updateMessage(), 444
 verifyBilling(), 606, 615
 WalkDirs(), 136
 writeFruit(), 126
 funkcje

- anonimowe, 44, 89
- modułu console, 73, 74
- opakowujące, 90
- pomocnicze, 612
- pośrednie, 356, 357, 414
- pośrednie komponentu
 - Express, 413

- pośrednie
 - niestandardowe, 427
- przekazywanie
 - zmiennych, 43
- zasięgu, 457
- zwracanie wartości, 43

G

- globalne interfejsy API, 439
- gniazda
 - sieciowe, 167
 - TLS, 182
- grupowanie
 - dokumentów, 317, 324
 - wyników, 316

H

- hierarchia
 - kontrolerów, 463
 - zasięgów, 461, 466

I

- implementowanie
 - AngularJS, 438
 - agregacji, 323
 - alertów przeglądarki, 518
 - animacji, 522
 - animacji CSS, 524
 - częściowego szablonu, 572, 600
 - domknięcia, 89
 - dostępu do danych, 641
 - dyrektyw, 485
 - niestandardowych, 506, 630
 - zagnieżdżonych, 507
 - zdarzeń, 499, 508
 - emiterów zdarzeń, 85, 87
 - funkcji pomocniczych, 612
 - funkcji pośrednich, 356, 551

- funkcji rachunku, 615
- funkcji wysyłki, 615
- funkcji zakupu, 616
- hierarchii kontrolerów, 464
- hierarchii zasięgów, 461, 462
- klastra HTTP, 206
- klastrów procesów, 203
- klientów gniazd TCP, 176
- klientów i serwerów
 - HTTP, 153
- klientów TLS, 181
- komponentu Express, 389
- kontrolera, 457, 490
 - modelu
 - CommentThread, 565
 - modelu klientów, 596
 - modelu Page, 564
 - modelu Photo, 564
 - modelu produktów, 594
 - modelu zamówień, 595
- kontrolera zdjęć, 577
- liczników czasu, 80, 521
- łańcucha wywołań
 - zwrotnych, 136
- magazynu GridFS, 377
- mechanizmu szablonów, 406
- modelu
 - CommentThread, 561
 - obiektu User, 535
 - Page, 560
 - Photo, 560
 - zasięgu, 444
- modułu AngularJS, 453
- modułu i kontrolera, 549
- niestandardowej funkcji
 - pośredniej, 427

- obiektu
 - ClientRequest, 146
 - Grid, 377
 - GridStore, 379, 383
 - strumienia Duplex, 111
 - strumienia Readable, 107
 - strumienia Transform, 113
 - strumienia Writable, 109
- operatorów wyrażen
 - agregacji, 321
- parametrów trasy, 393, 397
- pętli, 39
- podrzędnych rozwidleń, 200
- procedur obsługi tras, 566, 567
- procesów nasłuchiwania
 - zdarzeń, 85, 87
- procesów podrzędnych, 192
- schematu, 331
- serwera
 - aplikacji, 627
 - arbitra, 366
 - Express, 515
 - gniazd TCP, 178
 - HTTP, 158
 - HTTPS, 162, 391
 - komentarzy, 561
 - koszyka zakupów, 592
 - TLS, 181
 - WWW, 153, 155
- serwerów żądań GET, 154
- serwerów żądań POST, 157
- sesji, 414, 421
- skryptu Node.js, 582
- strumienia Readable, 132

- implementowanie
 - strumienia Writable, 127
 - stylów CSS, 548, 575, 633, 637
 - szablonów, 410
 - szablonu koszyka, 602
 - szablonu wysyłek, 603
 - tras, 392, 537, 552, 563, 593, 627
 - getPage, 564
 - getPhoto, 564
 - kontrolera
 - użytkowników, 538
 - kontrolerów, 563
 - trasy
 - aktualizacji
 - użytkownika, 540
 - logowania
 - użytkowników, 539
 - rejestrowania
 - użytkowników, 538
 - usuwania
 - użytkownika, 541
 - uzyskiwania profilu
 - użytkownika, 540
 - tworzenia segmentów danych, 369
 - unikalnych pól, 330
 - usług, 511
 - usług gniazd, 167
 - usług HTTP, 141
 - usługi
 - \$animation, 526
 - \$cacheFactory, 518
 - \$cookieStore, 521
 - \$http, 517
 - Google, 553
 - internetowej, 160
 - pogodowej, 633
 - uwierzytelniania, 423–425, 550
 - widoków użytkownika, 543
 - widoku
 - index, 545
 - komentarzy, 573
 - koszyka, 602
 - login, 546
 - podsumowania, 607
 - produktów, 599
 - rachunku, 604
 - signup, 543
 - strony produktu, 601
 - user, 547
 - usługi pogodowej, 633
 - wątku komentarzy, 571
 - wysyłek, 603
 - z kartami, 628, 629
 - zakupów, 598
 - zamówień, 608
 - zdjęć, 569
 - wstrzykiwania
 - zależności, 452, 453
 - wywołań blokujących, 84
 - wywołań zwrotnych, 82, 88
 - zasięgu, 455
 - zestawu replik, 366
 - indeksowanie, 232
 - indeksowanie pól, 361
 - informacje
 - cookie, 30, 419, 519
 - o kolekcji, 279
 - o pliku, 134
 - o procesie, 190, 191
 - o repozytorium, 70
 - o usłudze Google, 555
 - o użytkownika, 408, 409
 - o wysyłce, 615
 - inicjowanie
 - aplikacji, 581, 619, 646
 - bazy danych, 646
 - instalowanie
 - modułów NPM, 64
 - MongoDB, 236
 - środowiska Node.js, 60
 - instrukcja
 - if, 38
 - switch, 39
 - throw, 55
 - integrowanie
 - komponentu AngularJS, 437
 - usług w module, 528
 - interakcja
 - z użytkownikiem, 26
 - z zewnętrznymi źródłami, 160
 - interfejs API, 439
 - interwał, 81
 - iteracja w obrębie tablic, 52
- ## J
- JavaScript, 33
 - JSON, JavaScript Object Notation, 95
 - JSONP, 403
- ## K
- katalog, 534, 558, 586, 624
 - node_modules, 61, 258
 - klasa
 - EventEmitter, 87
 - fadeClass, 524
 - fadeOut, 526
 - klaster, 374
 - klastery procesów, 203
 - klient
 - gniazd TCP, 176
 - HTTP, 153, 159
 - HTTPS, 162, 163
 - TLS, 181, 182
 - WWW, 154, 156
 - klucz
 - prywatny, 165
 - publiczny, 165
 - segmentu danych, 371
 - złożony, 371
 - kod
 - CSS, 548, 574, 631, 639
 - do wielokrotnego wykorzystania, 31

- JavaScript, 29
 - szablonu HTML, 458, 463, 466
 - kodowanie UTF8, 100
 - kolejka zdarzeń, 78, 79
 - kolekcja, 224, 250
 - db.system.users, 241
 - nebulae, 285
 - word_stats, 331
 - kolekcje
 - aktualizowanie dokumentów, 254, 287
 - dodawanie dokumentów, 253, 283
 - niepodzielne modyfikowanie dokumentów, 290
 - ograniczone, 229, 365
 - tworzenie, 250, 278
 - usuwanie, 251, 278
 - usuwanie dokumentów, 253, 294
 - usuwanie pojedynczego dokumentu, 296
 - uzyskiwanie dokumentów, 286
 - wstawianie dokumentów, 293
 - zapisywanie dokumentów, 291
 - znajdowanie dokumentów, 252
 - komentarze, 559
 - kompilator środowiska
 - AngularJS, 485
 - kompilowanie
 - modelu, 332
 - szablonu AngularJS, 470
 - komponent
 - AngularJS, 20, 429, 431
 - Express, 389
 - MongoDB, 20, 221, 223
 - komponenty
 - interaktywne, 623
 - podstawowe, 23
 - stosu Node.js-AngularJS, 27
 - kompresowanie
 - buforów, 115
 - danych, 115
 - strumieni, 117
 - komunikacja między przeglądarką i serwerem, 25
 - konfigurowanie
 - modułu AngularJS, 449
 - komponentu Express, 390
 - kontroli dostępu, 245
 - tras, 392
 - zasięgu dyrektywy, 503
 - konstruktor, 216
 - konto
 - administratora baz danych, 247
 - administratora użytkowników, 245
 - użytkownika, 241
 - kontroler, 434, 456
 - do obsługi komentarzy, 578
 - modelu
 - CommentThread, 565
 - klientów, 596
 - Page, 564
 - Photo, 564
 - produktów, 594
 - zamówień, 595
 - shoppingController, 611
 - SimpleTemplate, 458, 459
 - tableController, 642, 643
 - usługi pogodowej, 634
 - kontrolowanie procesu, 189
 - konwersacja, 78
 - kopiowanie
 - baz danych, 249
 - buforów, 101
 - koszyk zakupów, 585
- ## L
- liczba, 34
 - dokumentów, 306
 - kolekcji, 232
 - tras, 393
 - zestawów replik, 367
 - licznik czasu, 80
 - interwału, 81
 - natychmiastowy, 83
 - limit czasu, 80
 - lista
 - baz danych, 247, 273
 - kolekcji, 250, 277
 - użytkowników, 241
 - literal obiektowy, 35
- ## Ł
- ładowanie biblioteki, 443
 - łańcuch, 34, 48, 50, 213
 - JSON, 96
 - połączenia, 263
 - zapytania, 144, 393
 - łączenie
 - buforów, 104
 - łańcuchów, 50
 - strumieni, 114
 - tablic, 51
 - wywołań zwrotnych, 91, 92
- ## M
- magazyn GridFS, 377
 - mechanizm
 - EJS, 406, 407
 - Jade, 406, 409
 - metadane klastra, 374
 - metoda
 - \$broadcast(), 464
 - \$destroy(), 461
 - \$digest(), 461
 - \$emit(), 464
 - \$on(), 464

- metoda
 - \$scope.addReply(), 578
 - _flush(), 112
 - _transform(), 112
 - abort(), 147, 189
 - acceptsCharset(), 398
 - addObject(), 285
 - address(), 171, 175
 - addTrailers(), 149
 - addUser(), 243, 267, 269
 - admin(), 266
 - aggregate(), 319, 335
 - all(), 339
 - angular.module(), 449
 - animate(), 524
 - app.all(), 393
 - app.engine(), 407
 - app.listen(), 391
 - app.render(), 410, 411
 - append(), 352
 - arch(), 212
 - attachment(), 400
 - authenticate(), 267, 268
 - Buffer.byteLength(), 101
 - buffer.fill(), 98
 - buffer.toString(), 100
 - buffer.write(), 98
 - child_process.fork(), 175
 - chunkCollection(), 380
 - close(), 152, 175, 380
 - collection(), 266, 380
 - collectionInfo(), 266
 - collectionNames(), 266
 - collections(), 267
 - comment(), 337
 - concat(), 104
 - config(), 449
 - connect(), 263
 - controller(), 456
 - count(), 271, 307, 335
 - cpus(), 212
 - create(), 335, 343
 - createCollection(), 250, 267
 - createServer(), 152, 156
 - db(), 266
 - db.getSiblingDB(), 248
 - deserializeUser(), 552
 - destroy(), 171
 - directive(), 500
 - disconnect(), 194, 206
 - displayWords(), 304
 - distinct(), 271, 315, 335
 - doAsyn(), 357
 - drop(), 271
 - dropCollection(), 267
 - dropDatabase(), 249, 267
 - each(), 272, 287
 - elemMatch(), 339
 - end(), 109, 147, 149, 154, 158, 171
 - endianness(), 212
 - ensureIndex(), 364
 - eof(), 381
 - EOL, 212
 - error(), 515
 - exec(), 352
 - exists(), 338
 - exit(), 189
 - find(), 252, 286, 304, 335
 - findAndModify(), 271, 290
 - findAndRemove(), 271, 296
 - findOne(), 271, 286, 335
 - findOneAndRemove(), 335
 - findOneAndUpdate(), 335
 - freemem(), 212
 - fs.exists(), 133
 - fs.existsSync(), 133
 - fs.readFile(), 154
 - fsStatsSync(), 134
 - get(), 339, 398
 - getc(), 381
 - getConnections(), 174
 - getgid(), 191
 - getgroups(), 191
 - getHeader(), 149
 - getuid(), 191
 - GridStore.exist(), 381
 - GridStore.list(), 381
 - GridStore.read(), 381
 - GridStore.readlines(), 382
 - GridStore.unlink(), 382
 - group(), 316, 352
 - gt(), 337
 - gte(), 337
 - hint(), 337
 - hostname(), 212
 - hrtime(), 191
 - http.createServer(), 391
 - http.request(), 145, 164
 - https.createServer(), 164
 - https.request(), 164
 - in(), 338
 - index(), 364
 - inherits(), 110, 216
 - initgroups(), 191
 - initialize(), 551
 - insert(), 253, 269
 - invalidate(), 340
 - isArray(), 214
 - isClosed(), 273
 - isDate(), 214
 - isError(), 214
 - isInit(), 340
 - isModified(), 340
 - isRegExp(), 214
 - isSelected(), 340
 - json(), 402
 - JSON.parse(), 96
 - JSON.stringify(), 96
 - jsonp(), 402
 - kill(), 189, 194
 - limit(), 336, 352
 - listDatabases(), 268
 - listen(), 152, 174
 - loadavg(), 212
 - location(), 400
 - logout(), 267, 268
 - lookup(), 217
 - lt(), 338

- lte(), 338
- markModified(), 340
- match(), 352
- memoryUsage(), 190
- mod(), 338
- modifiedPaths(), 340
- mongoose.connect(), 332
- ne(), 338
- net.connect(), 168, 177
- net.createConnection(), 168
- net.createServer(), 175, 179
- networkInterfaces(), 212
- next(), 357
- nextObject(), 272
- nin(), 338
- open(), 266, 380
- parse(), 144
- passport.authenticate(), 552
- pause(), 106, 171
- ping(), 268
- pipe(), 106
- platform(), 212
- project(), 352
- put(), 378
- puts(), 381
- queryRemover(), 428
- querystring.parse(), 161
- read(), 106, 336, 352, 380
- readFile(), 128
- readInt16BE(), 100
- readInt16LE(), 100
- readInt8(), 100
- readlines(), 380
- readUInt32BE(), 100
- ref(), 171, 175
- regex(), 338
- release(), 212
- remove(), 253, 294, 349
- removeHeader(), 149
- removeUser(), 244, 267
- rename(), 270
- renameCollection(), 267
- req.isAuthenticated(), 553
- req.login(), 552
- req.logout(), 552
- require(), 536
- res.cookie(), 420
- res.download(), 405
- res.end(), 154
- res.redirect(), 405
- res.render(), 410
- res.send(), 394, 395
- resetDays(), 516
- resolve(), 218
- resume(), 106, 171
- rewind(), 272, 380
- run(), 450
- safe(), 336
- save(), 254, 270, 344
- seek(), 380
- select(), 336
- send(), 194, 205, 400
- sendfile(), 403
- serializeUser(), 552
- serverStatus(), 268, 276
- set(), 339, 400
- setEncoding(), 106, 170
- setgid(), 191
- setgroups(), 191
- setHeader(), 149
- setImmediate(), 84
- setKeepAlive(), 171
- setNoDelay(), 148, 171
- setOptions(), 336
- setSocketKeepAlive(), 148
- setTimeout(), 147, 171
- setuid(), 191
- sh.addShard(), 375
- skip(), 336, 352
- slice(), 102
- snapshot(), 336
- sort(), 273, 336, 353
- stats(), 271, 279
- status(), 399
- stream(), 381
- stringDecoder.write(), 100
- success(), 515
- tell(), 380
- tls.connect(), 183
- tls.createServer(), 184
- tmpdir(), 212
- toArray(), 272, 286
- toJSON(), 340
- toObject(), 340
- toString(), 99, 340
- totalmem(), 212
- type(), 212, 400
- unlink(), 380
- unpipe(), 106
- unref(), 171, 175
- unwind(), 353
- update(), 254, 270, 291, 335, 346
- uptime(), 191, 212
- url.format(), 142
- url.pars(), 154
- url.parse(), 142, 143
- url.resolve(), 143
- use(), 415
- util.format(), 214
- util.inherits(), 216
- util.inspect(), 215
- validate(), 340, 354
- where(), 337
- write(), 108, 147, 171, 381
- writeContinue(), 149
- writeFile(), 380
- writeHead(), 149
- writeInt16BE(), 98
- writeInt16LE(), 98
- writeInt8(), 98
- metody
 - kodowania, 97
 - kopiowania, 102
 - modułu cluster, 204
 - modułu dns, 217
 - modułu os, 212

- metody
 - modułu process, 190
 - obiektu
 - Admin, 268, 269
 - Aggregate, 352
 - Array, 52, 54
 - ChildProcess, 194
 - ClientRequest, 147
 - Collection, 269, 271
 - Cursor, 272
 - Db, 266
 - Document, 339
 - GridStore, 380
 - HTTP Request, 397
 - Model, 335, 336
 - net.Server, 174
 - Query, 335–338
 - req.session, 425
 - Readable, 106
 - Response, 400
 - ServerResponse, 149, 150
 - Socket, 170
 - Stats, 134
 - String, 49
 - Worker, 205
 - Writable, 109
 - odczytu, 100
 - odczytywania obiektów
 - Buffer, 100
 - powłoki MongoDB, 239
 - tworzenia obiektów, 98
 - zapisywania danych, 98, 99
 - migawka systemu plików, 385
 - model
 - Comment, 567
 - CommentThread, 560
 - DOM, 25, 432
 - Page, 559
 - Photo, 560
 - wątkowości, 76
 - wywołań zwrotnych
 - zdarzeń, 76
 - zdarzeń, 75
 - moduł, 62
 - connect-mongo, 533
 - ejs, 533
 - express, 533
 - mongodb, 533
 - mongoose, 533
 - Buffer, 97
 - child_process, 192
 - cluster, 203, 206
 - connect, 414
 - console, 73, 74
 - dns, 211, 217
 - express, 65, 389
 - Express, 30, 387
 - fs, 119
 - http, 141
 - mongoose, 252, 258, 325, 341
 - net, 167
 - os, 211, 213
 - passport, 550, 552
 - process, 187, 189, 191
 - kontrolowanie procesu, 189
 - metody i właściwości, 190
 - potoki wejścia-wyjścia procesów, 187
 - sygnały procesów, 188
 - uzyskiwanie informacji, 190
 - querystring, 144
 - static, 416
 - Stream, 104
 - util, 213–216
 - zlib, 115–117
 - moduły NPM, 533
 - modyfikowanie
 - baz danych, 273, 281
 - kolekcji, 276
 - danych, 477
 - dokumentów, 290
 - modelu DOM, 504
 - MongoDB, 20, 29
 - administrowanie kontami, 241
 - bazowanie na dokumentach, 20, 30
 - dokumenty, 224
 - dostępność, 20, 30
 - kolekcje, 224
 - kontrola dostępu, 245
 - optymalizacja wydajności, 231
 - planowanie modelu danych, 227
 - skalowalność, 20, 30
 - skrypty dla powłoki, 240
 - środowisko GridFS, 377
 - środowisko komponentu, 235
 - środowisko MapReduce, 319
 - typy danych, 226
 - wydajność, 20, 30
 - monitorowanie pliku, 139
 - MVC, Model View Controller, 431
- ## N
- nagłówek Content-Type, 399
 - nagłówki HTTP, 26
 - naprawianie bazy danych, 383
 - narzędzia globalnych interfejsów API, 440, 441
 - nasłuchiwanie, 85
 - połączeń, 152
 - zdarzeń, 79, 86
 - nawiasy klamrowe, 458
 - niepodzielne operacje zapisu, 231
 - niezawodność zapisu, 258
 - Node.js, 19, 28, 57, 59
 - aplikacje, 68
 - dodawanie sterownika MongoDB, 257
 - dostęp do MongoDB, 257
 - dostęp do systemu plików, 119

- instalowanie, 60
 - instalowanie modułów, 64
 - konfiguracja, 29
 - lokalizacja instalacji, 61
 - model zdarzeń, 75
 - moduły, 62
 - moduły dodatkowe, 211
 - Node Package Registry, 62
 - obsługa danych
 - wejścia-wyjścia, 95
 - obsługa JavaScript, 19, 29
 - pliki wykonywalne, 61
 - połączenie z MongoDB, 258
 - publikowanie modułu, 69
 - rozszerzalność, 19, 29
 - skalowalność, 19, 29, 187
 - tworzenie modułu, 68
 - usługi gniazd, 167
 - usługi HTTP, 141
 - wybór IDE, 61
 - wyszukiwanie modułów, 63
 - normalizacja, 470
 - normalizowanie danych, 227
 - NoSQL, Not Only SQL, 223
- O**
- obcinanie plików, 137
 - obiekt, 45
 - Account, 87
 - Admin, 268, 275
 - Agent, 164
 - Array, 51
 - Buffer, 98
 - ChildProcess, 193
 - Collection, 269, 315
 - Connection, 327
 - Cursor, 272
 - Customer, 599
 - Db, 266, 364
 - Document, 326, 330, 339
 - error, 355, 378
 - Event, 464, 498
 - EventEmitter, 86, 216
 - Grid, 377
 - GridStore, 379
 - http.ClientRequest, 145
 - http.IncomingMessage, 149
 - http.ServerResponse, 148
 - locals, 407
 - Model, 333
 - Module, 449, 450
 - MongoClient, 260, 261
 - net.Server, 173
 - net.Socket, 168
 - options, 302
 - Query, 300, 333, 336
 - metody, 335, 338
 - opcje operacji bazodanowych, 336
 - operacje bazodanowe, 334
 - operatory, 337
 - Request, 395, 397
 - Response, 395, 398, 401
 - Schema, 326, 331, 332
 - Server, 150, 259
 - String, 48, 49
 - StringDecoder, 99
 - update, 283
 - URL, 142
 - User, 535
 - Worker, 205
 - obiekty niestandardowe, 46
 - obsługa
 - animacji, 522
 - błędów, 30, 53
 - danych wejścia-wyjścia, 95
 - dodawania produktów, 613
 - dyrektyw, 495
 - funkcji szablonu, 458
 - końca CSS3, 524
 - komentarzy, 578, 580
 - parametrów żądania POST, 418
 - sesji, 414, 421, 422
 - szablonu, 442
 - tras modelu Comment, 567
 - trasy addComment(), 565
 - trasy getComment(), 565
 - trasy /words, 641
 - usuwanie produktów, 613
 - widoków, 627
 - widoków komentarzy, 576
 - widoków koszyka zakupów, 611
 - wyświetlania, 563
 - zdarzeń, 499
 - żądań, 593
 - odczytywanie
 - asynchroniczne z pliku, 130
 - informacji cookie, 414
 - plików, 121, 128
 - synchroniczne pliku, 128
 - z buforów, 99
 - oddzielanie zakresów odpowiedzialności, 435
 - ODM, Object Document Model, 325
 - odporność na błędy, 367
 - ograniczenie
 - działania dyrektyw, 502
 - liczby dokumentów zwracanych, 310
 - liczby pól, 310
 - pól zwracanych, 309
 - wyników według wielkości, 308
 - zestawów wynikowych, 308
 - zestawu dokumentów, 309

- ograniczone kolekcje, 229, 364
- określanie
 - liczby dokumentów, 306, 307
 - stylów CSS, 548, 639
 - stylów widoków, 610
- opcja
 - process.nextTick, 80
 - upsert, 293
- opcje
 - funkcji
 - exec(), 195
 - fork(), 201
 - spawn(), 198, 199
 - metody
 - https.createServer(), 164
 - https.request(), 164
 - res.cookie(), 420
 - tls.connect(), 183
 - tls.createServer(), 184
 - modułu static, 416
 - obiektu
 - ClientRequest, 146
 - Module, 449
 - options, 303
 - query, 333
 - Schema, 329
 - Server, 173, 259
 - operacji bazodanowych, 336
 - polecenia npm, 64
 - połączenia z bazą danych, 261
- operator, 300
 - \$group, 320, 321
 - \$gt, 301
 - \$sort, 324
- operatory
 - agregacji, 319–321
 - aktualizowania baz danych, 283
 - arytmetyczne, 36, 322
 - łańcuchowe, 322

- obiektu Query, 301, 337
- obiektu update, 284
- porównania, 36, 37
- przypisania, 36
- warunkowe, 36
- opóźnianie pracy, 80
- optymalizacja wydajności bazy, 231
- otwieranie plików, 120

P

- pakiety NPM, 62
- parametry
 - formularza, 144
 - metody POST, 393
 - polecenia mongod, 237
 - w trasach, 393
 - zdefiniowane, 393, 395
- partycjonowanie, 372
- pętla, 39
 - do/while, 40
 - for, 40
 - for/in, 41
 - while, 39
 - zdarzeń, 83
- planowanie modelu danych, 227
- plik
 - angular_expressions.html, 472
 - angular_expressions.js, 472
 - angular_filter_custom.
 - ↳html, 481
 - angular_filter_customer.js, 481
 - angular_filter_sort.html, 479
 - angular_filter_sort.js, 478
 - angular_filters.html, 477
 - angular_filters.js, 476
 - animate.css, 526
 - auth_server.js, 535
 - billing.html, 605
- buffer_concat.js, 104
- buffer_copy.js, 102
- buffer_read.js, 100
- buffer_slice.js, 103
- buffer_write.js, 99
- callback_chain.js, 92
- callback_closure.js, 90
- callback_parameter.js, 89
- cart.html, 602
- cart_app.js, 611–617
- cart_init.js, 619
- cart_model.js, 589, 590, 591
- cart_routes.js, 593
- cart_server.js, 592
- cart_styles.css, 598, 610
- corsortext.js, 68
- chef.js, 202
- child_fork.js, 201
- child_process_exec.js, 195
- child_process_exec_file.
 - ↳js, 197
- child_process_spawn.js, 199
- cluster_client.js, 207
- cluster_server.js, 206
- cluster_worker.js, 207
- collection_create_list.
 - ↳delete.js, 278
- collection_stats.js, 279
- comment_app.js, 576–580
- comment_init.js, 582
- comment_styles.css, 575
- comment_thread.html, 572, 573
- comments_controller.js, 565–567
- comments_model.js, 561
- comments_routes.js, 563
- comments_server.js, 562
- customers_controller.js, 596

- db_connect_object.js, 262
- db_connect_string.js, 264
- db_create_list_delete.js, 274
- db_status.js, 276
- directive_bind.html, 495
- directive_bind.js, 495
- directive_custom.html, 507
- directive_custom.js, 506
- directive_event.html, 499
- directive_event.js, 499
- directive_form.html, 491
- directive_form.js, 490
- dns_lookup.js, 219
- doc_aggregate.js, 323
- doc_count.js, 307
- doc_delete.js, 295
- doc_delete_one.js, 296
- doc_distinct.js, 315
- doc_fields.js, 310
- doc_find.js, 287
- doc_group.js, 317
- doc_insert.js, 285
- doc_limit.js, 309
- doc_modify.js, 290
- doc_paging.js, 312
- doc_query.js, 305
- doc_save.js, 292
- doc_sort.js, 313
- doc_update.js, 289
- doc_upsert.js, 293
- draggable.html, 639
- draggable_styles.css, 640
- emitter_listener.js, 87
- express_auth.js, 423
- express_auth_one.js, 424
- express_auth_session.js, 425
- express_cookies.js, 420
- express_http_https.js, 391
- express_middleware.js, 427
- express_post.js, 418
- express_redirect.js, 405
- express_request.js, 398
- express_routes.js, 396
- express_send.js, 401
- express_send_file.js, 404
- express_send_json.js, 402
- express_session.js, 422
- express_static.js, 417
- express_templates.js, 410
- file_read.js, 128
- file_read_async.js, 130
- file_read_stream.js, 132
- file_read_sync.js, 129
- file_readdir.js, 136
- file_stats.js, 135
- file_write.js, 122
- file_write_async.js, 125
- file_write_stream.js, 127
- file_write_sync.js, 124
- first.html, 441
- first.js, 442
- google_auth.js, 553
- grid_fs.js, 378
- gridstore_fs.js, 382
- http_client_get.js, 156
- http_client_static.js, 154
- http_server_external.js, 160
- http_server_get.js, 155
- http_server_post.js, 158, 159
- http_server_static.js, 153
- index.html, 545
- info.html, 555
- injector.html, 453
- injector.js, 453
- login.html, 546, 554
- main_jade.jade, 409
- mongoose_aggregate.js, 353
- mongoose_connect.js, 327
- mongoose_create.js, 343
- mongoose_find.js, 341
- mongoose_middleware.js, 357
- mongoose_remove_many.js, 350
- mongoose_remove_one.js, 349
- mongoose_save.js, 345
- mongoose_update_many.js, 348
- mongoose_update_one.js, 346
- mongoose_validation.js, 355
- my_app.js, 549
- my_photos.html, 508
- nexttick.js, 84
- node, 61
- node_server.js, 441, 515
- orders.html, 609
- orders_controller.js, 595
- os_info.js, 213
- package.json, 66, 69, 70
- page_model.js, 560
- pages_controller.js, 564
- photo_model.js, 560
- photos.html, 569
- photos_controller.js, 564
- process_info.js, 191
- product.html, 601
- products.html, 600
- plik
 - products_controller.js, 594
- review.html, 607
- rich_pane.html, 628
- rich_tabs.html, 629
- rich_ui.html, 630
- rich_ui_app.js, 630, 634, 638, 642
- rich_ui_routes.js, 627
- rich_ui_server.js, 627
- rich_ui_styles.css, 633
- routes.js, 537
- scope_controller.js, 457
- scope_events.html, 466
- scope_events.js, 465

- scope_hierarchy.html, 463
- scope_hierarchy.js, 462
- scope_template.html, 458
- scope_template.js, 458
- service_animate.html, 525
- service_animate.js, 525
- service_cache.js, 518
- service_cookie.html, 520
- service_cookies.js, 520
- service_custom.js, 528
- service_http.html, 517
- service_http.js, 516
- shipping.html, 603
- shopping.html, 598
- signup.html, 544
- simple_interval.js, 82
- simple_timer.js, 81
- socket_client.js, 176
- socket_server.js, 178
- stream_duplex.js, 111
- stream_piped.js, 114
- stream_read.js, 107
- stream_transform.js, 113
- stream_write.js, 109
- styles.css, 548
- table_styles.css, 646
- tables.html, 644
- user.html, 547
- user_ejs.html, 408
- user_jade.jade, 409
- users_controller.js, 538–541
- users_model.js, 535
- util_inherit.js, 216
- weather.html, 635
- weather_controller.js, 633
- weather_styles.css, 637
- word_init.js, 646
- word_model.js, 626
- word_schema.js, 331
- words_controller.js, 641
- zlib_buffers.js, 116
- zlib_file.js, 117
- pliki
 - CSS, 26
 - HTML, 26
 - monitorowanie zmian, 139
 - multimedialne, 26
 - obcinanie, 137
 - odczytywanie, 127
 - statyczne, 153, 416, 441
 - tryby otwierania, 121
 - usuwanie, 136
 - uzyskiwanie informacji, 134
 - zapisywanie, 122
 - zmiana nazwy, 139
- pobieranie komentarzy, 576
- podział łańcucha, 50
- poła wymagane, 330
- polecenia powłoki
 - MongoDB, 239
- polecenie
 - db.repairDatabase, 383
 - mongod, 237, 375, 383
 - mongodump, 385
 - npm, 258
- połączenie z MongoDB
 - biblioteka Mongoose, 327
 - łańcuch połączenia, 263
 - Node.js, 258
 - obiekt MongoClient, 260
- porównanie kolekcji, 232
- porządkowanie elementów, 478–480
- potok, 114
- potoki wejścia-wyjścia
 - procesów, 187
- powłoka MongoDB, 238, 240
- praca
 - natychmiastowa, 83
 - okresowa, 81
- proces
 - główny, 206
 - mongod, 374
 - mongos, 374, 375
 - nadrzędny, 201
 - nasłuchiwanie, 85–87
 - podrzędny, 192, 201, 202
 - roboczy, 207
- program
 - Gzip, 414
 - Node Package Manager, 63
- protokół
 - HTTP, 25, 162
 - HTTPS, 25, 162
 - SSL, 165
 - TCP, 167
 - TLS, 181
- prototyp, 47
- przechowywanie
 - danych, 378
 - pliku, 382
- przeciąganie elementów, 637
- przełączarka, 25
- przekazywanie
 - parametrów, 88
 - zmiennych, 43
- przekierowywanie
 - odpowiedzi, 405
 - zadań, 405
- przekształcanie
 - danych JSON, 96
 - obiektów JavaScript, 96
 - obiektów w łańcuchy, 215
 - tablicy, 53
- przerywanie pętli, 41
- przetwarzanie
 - adresów URL, 141
 - łańcuchów, 48
 - obiektów Array, 52
 - obiektów String, 49
- przyciski przełącznika, 520
- przydatność danych, 233

przypisywanie funkcji
pośrednich, 415
publikowanie modułu, 69

R

rejestr Node Package
Registry, 62
relacja między zasięgami
i kontrolerami, 456
i szablonami, 457
i aplikacjami, 455
i danymi serwera, 459
renderowanie szablonów,
409, 411
replikacje, 232, 365
liczba serwerów, 367
liczba zestawów, 367
serwer arbitra, 365
serwer dodatkowy, 365
serwer główny, 365
RIA, Rich Internet
Applications, 623
rola, 242, 243
clusterAdmin, 244
dbAdmin, 244
dbAdminAnyDatabase,
244
read, 244
readAnyDatabase, 244
readWrite, 244
readWriteAnyDatabase,
244
userAdmin, 244
userAdminAnyDatabase
, 244
router zapytań, 370, 374
rozgłaszanie zdarzeń, 463
rozszerzalność, 29, 31
rozszerzanie kodu HTML,
500
rozwidlenia, 200

S

schemat, 328, 329
AddressSchema, 589
CustomerSchema, 591
OrderSchema, 590
ProductQuantitySchema,
590
ProductSchema, 590
schematy
dodawanie indeksów,
330
dodawanie metod, 331
segment, 102
segmentowanie danych, 370
dla kolekcji, 376
na podstawie wartości
mieszającej, 372
na podstawie zakresu,
372
w bazie danych, 375
serializacja, 551
serwer
Apache, 28
aplikacji, 626
arbitra, 365
dodatkowy, 365
Express, 391
główny, 365
gniazd TCP, 176, 178
HTTP, 153
HTTPS, 162, 165
komentarzy, 561
konfiguracji, 371, 374
koszyka zakupów, 592
routera zapytań, 374
TLS, 181, 183
WWW, 25, 27, 155, 441
skalowalność sterowana
zdarzeniami, 19, 29
składnia obiektów, 45
skrypt, 26
serwerowy, 27
styles.css, 548

słowo kluczowe
finally, 55
new, 98
sortowanie wyników, 313
sprawdzanie
poprawności, 354
poprawności
dokumentów, 355
typów obiektów, 214
SSL, Secure Sockets Layer,
181
status
HTTP, 399
serwera, 276
serwera MongoDB, 275
statyczny serwer WWW, 441
sterownik dla MongoDB,
257, 265, 276
stos Node.js-AngularJS, 23,
27
stosowanie
funkcji wywołania
zwrotnego, 395
parametrów trasy
łańcuchy zapytania,
394
wyrażenia regularne,
394, 470
zdefiniowane
parametry, 395
strategia
replikacji, 367
uwierzytelniania, 550
stronicowanie wyników,
311, 312, 643
struktura
drzewa, 25
katalogowa projektu,
534, 558, 586, 624
strumienie
Duplex, 110
Readable, 105
Transform, 112
Writable, 108

strumieniowanie
 danych, 104
 odczytu z pliku, 131
 zapisu do pliku, 126
 żądań GET, 414

style
 CSS, 467
 widoków, 574, 610
 widoku danych
 pogodowych, 637
 widoku tabel, 646
 widoku z kartami, 631

superkonstruktor, 216

sygnały procesów, 188

synchroniczne wywołania,
 120

synchroniczny zapis danych,
 215

system plików, 119

szablon, 406, 433, 457, 469
 AngularJS, 459, 472, 481,
 491, 495
 częściowy, 571–574
 dyrektyw, 502
 kart, 629
 mechanizmu EJS, 408
 mechanizmu Jade, 409
 panelu, 628
 widoku dyrektyw, 501

Ś

ścieżka, 133, 328
 bezwzględna, 138
 root, 404
 względna, 138

śledzenie żądań, 414

środowisko
 agregacji, 320
 IDE, 61
 MapReduce, 319
 MVC, 431
 Node.js, 19, 57, 59
 sprawdzania
 poprawności, 354

T

tablica, 35, 51–53
 \$scope.content, 611
 \$scope.words, 643
 customer.cart, 613

TCP, Transmission Control
 Protocol, 167

technologia AngularJS, 20

testowanie serwera, 207

TLS, Transport Layer
 Security, 181

transkluzja zasięgów
 zewnętrznych, 503

trasa, 392
 /images, 441
 /login, 426
 /logout, 426
 /remove/day, 516
 /reset/days, 515
 /static, 441
 /static/css, 441
 /static/js, 441
 /words, 641
 addComment(), 565
 addOrder(), 596
 deleteUser, 541
 getComment(), 565
 getUserProfile, 540
 login, 539
 signup, 538
 updateBilling(), 597
 updateUser, 540

trasy statyczne, 417

tryby otwierania pliku, 121

TTL, time-to-live, 363

tworzenie
 aplikacji, 581
 aplikacji AngularJS, 441
 aplikacji Node.js, 68
 baz danych, 248, 274,
 374
 buforów, 98
 certyfikatu, 182
 definicji schematu, 329
 dokumentów, 343, 344
 dyrektywy
 niestandardowej, 506

filtrów
 niestandardowych, 479

funkcji, 42
 funkcji anonimowej, 89
 funkcji opakowującej, 90
 instancji partycji, 232
 interaktywnych
 komponentów, 623

catalogów, 138

klastra, 203

klienta gniazd, 177

klienta gniazd TLS, 182

klienta HTTPS, 163

kolekcji, 250, 278

komponentów
 praktycznych, 531

kont użytkowników, 242

konta administratora
 baz danych, 247

konta administratora
 użytkowników, 245

kontrolera tras, 641

kopii zapasowej bazy,
 384

koszyka zakupów, 585

modułu, 68

niestandardowego
 obiektu, 87

niestandardowych
 funkcji pośrednich, 427

obiektów, 46
 Buffer, 98
 ClientRequest, 146
 Module, 449
 Server, 173, 259
 Socket, 169

potoku, 114

procesu, 197

procesu w procesie, 199

prototypów, 47

segmentów danych, 102,
 369

- klucz segmentu
 - danych, 371
 - typy serwerów, 370
 - serwera, 535
 - aplikacji, 626
 - gniazd TLS, 183
 - HTTPS, 165
 - komentarzy, 561
 - koszyka zakupów, 592
 - skryptów, 240
 - szablonów, 408
 - środowiska MongoDB, 235
 - usług, 512
 - usług niestandardowych, 527
 - usługi komentarzy, 576
 - usługi pogodowej, 633
 - widoków, 469
 - własnych dyrektyw, 500
 - własnych emiterów
 - zdarzeń, 80
 - wywołania, 79
 - zasięgu, 472, 476, 478
 - typy
 - danych, 34, 225, 328
 - filtrów, 477
 - funkcji pośredniej, 356
 - indeksów, 362, 363
- ## U
- udostępnianie plików
 - statycznych, 153, 416
 - URL, Uniform Resource Locator, 141
 - uruchamianie MongoDB, 236
 - usługa, 435
 - \$animate, 512, 522
 - \$animation, 525
 - \$cacheFactory, 512, 518
 - \$compile, 512
 - \$cookie, 519
 - \$cookies, 512
 - \$cookieStore, 519
 - \$document, 512
 - \$http, 512, 513
 - \$interval, 512, 521
 - \$locale, 512
 - \$location, 512
 - \$resource, 512
 - \$rootElement, 512
 - \$rootScope, 455, 512
 - \$route, 513
 - \$sce, 513
 - \$scope, 456
 - \$templateCache, 513
 - \$timeout, 513, 521
 - \$window, 513, 518
 - constant, 527
 - factory, 527
 - Google, 553–555
 - komentarzy, 576
 - pogodowa zaplecza, 633
 - service, 528
 - value, 527
 - usługi
 - HTTP, 141
 - niestandardowe, 527
 - środowiska AngularJS, 511
 - wbudowane, 512, 513
 - zaplecza, 27
 - ustawianie
 - nagłówków, 399
 - operacji bazodanowych, 334
 - statusu, 399
 - usuwanie
 - baz danych, 249, 274
 - dokumentów, 294–296, 349, 350
 - katalogów, 138
 - kolekcji, 251, 278
 - odniesień, 83
 - plików, 136
 - procesów nasłuchiwania, 86
 - produktów z koszyka, 613
 - użytkowników, 244
 - uwierzytelnianie, 246, 543
 - dzięki usłudze Google, 553
 - HTTP, 423
 - implementowanie tras, 552
 - poprzez konta społecznościowe, 549
 - strategie, 550
 - sesji, 424
 - uzyskiwanie dokumentów, 286
 - użycie
 - modułu w aplikacji, 71
 - obiektów, 45
 - operatorów, 35
 - użytkownicy, 24
- ## W
- wartość
 - boolowska, 34
 - elementu formularza, 491
 - null, 35
 - wątek pętli zdarzeń, 78
 - wczytywanie pliku łańcucha, 128
 - wdrażanie
 - klastra MongoDB, 373
 - zestawu replik, 368
 - Web 2.0, 623
 - weryfikowanie
 - istnienia ścieżki, 133
 - rachunku, 615
 - wiązanie danych, 31, 434
 - widoczność elementu, 492
 - widok, 433
 - danych pogodowych, 635–637
 - index, 545
 - koszyka, 602, 603
 - login, 546
 - podsumowania, 607, 608
 - produktów, 599, 600

widok

- rachunku, 604, 606
- signup, 543
- strony produktu, 601
- szablonu kart, 629
- szablonu panelu, 628
- tabel, 644, 645
- umożliwiający
 - przeciąganie, 639
- user, 547
- usługi pogodowej, 633
- wysyłek, 603, 604
- z kartami, 628, 629, 632
- zakupów, 598
- zamówień, 608, 609
- wielkość dokumentów, 231
- właściwości
 - modułu cluster, 204
 - modułu process, 190
 - obiektów ChildProcess, 194
 - obiektów Document, 339
 - obiektów
 - ServerResponse, 148, 150
 - obiektów Socket, 172
 - obiektów Worker, 205
 - obiektu event, 464
 - obiektu HTTP Request, 397
 - obiektu URL, 143
 - usługi \$http, 514
- włączanie
 - segmentowania danych, 375
 - uwierzytelniania, 246
- wstawianie dokumentów, 293
- wstrzykiwanie
 - kodu SQL, 20
 - zależności, 435, 447–452, 457

wybieranie metody

- partycjonowania, 372
- wyjątki, 55
- wymuszanie wymaganych pól, 330
- wyrażenia, 434, 469, 470
 - AngularJS, 471
 - aplikacji, 473
 - JavaScript, 471
 - regularne, 393
- wysyłanie
 - danych JSON, 402
 - danych JSONP, 403
 - odpowiedzi, 400, 405
 - odpowiedzi JSON, 401
 - plików, 403, 404
- wyszukiwanie
 - modułów NPM, 63
 - odwrotne, 219
 - podłańcucha, 50
- wyświetlanie
 - listy baz danych, 247, 273, 274
 - listy kolekcji, 250, 277
 - listy plików, 135
 - listy użytkowników, 241, 242
 - statusu serwera, 276
 - statystyk, 279
 - widoku, 25
- wywołania systemu plików
 - asynchroniczne, 120
 - synchroniczne, 120
- wywołania zwrotne, 88
 - implementowanie
 - domknięcia, 89
 - łączenie, 91
 - przekazywanie parametrów, 88
- wywołanie zwrotne
 - authenticate(), 262
 - checkGoal(), 88
- wzorzec obiektowy, 47

Z

- zakres znaczników segmentu danych, 376
- zamykanie plików, 120
- zapisywanie
 - asynchroniczne, 125
 - asynchroniczne w pliku, 122, 124
 - danych do konsoli, 72
 - dokumentów, 291, 345
 - łańcucha JSON, 122
 - pliku, 121
 - synchroniczne, 124, 215
 - synchroniczne w pliku, 123
 - w buforach, 98
- zarządzanie
 - kolekcjami, 250
 - pamięcią podręczną, 30
 - sesjami, 425
 - trasami, 30
- zasięg, 455
 - aplikacji do zakupów, 611
 - dyrektywy, 503, 504
 - główny, 455
 - nadrzędny, 462
 - zmiennych, 44
- zastępowanie
 - elementu szablonu, 502
 - słowa, 50
- zatrzymywanie działania MongoDB, 236
- zdarzenia
 - modułu cluster, 203
 - niestandardowe, 85
 - obiektów
 - ChildProcess, 193
 - ClientResponse, 147
 - Server, 151, 185
 - ServerResponse, 148, 150

- Socket, 170, 174
 - Worker, 205
 - wysyłane do procesów, 188
 - zdarzenie
 - \$broadcast(), 465
 - \$destroy, 505
 - animacji, 522
 - balanceChanged, 87
 - CharacterDeleted, 466
 - checkContinue, 151
 - clientError, 151, 185
 - close, 148, 151, 170, 193
 - connect, 147, 151, 170
 - connection, 151, 174, 180
 - continue, 147
 - data, 170
 - disconnect, 193, 204, 205
 - drain, 108, 170, 177
 - end, 170
 - error, 170, 174, 193, 205
 - exit, 193, 204, 205
 - finish, 108
 - fork, 203
 - listening, 174, 203
 - message, 193, 205
 - mousedown, 638
 - newSession, 185
 - online, 203
 - pipe, 108
 - request, 151
 - response, 147
 - resumeSession, 185
 - secureConnection, 185
 - setup, 204
 - SIGBREAK, 188
 - SIGHUP, 188
 - SIGINT, 188
 - SIGKILL, 189
 - SIGPIPE, 188
 - SIGSTOP, 189
 - SIGTERM, 188
 - SIGUSR1, 188
 - SIGWINCH, 188
 - socket, 147
 - timeout, 170
 - unpipe, 108
 - upgrade, 147, 151
 - zestaw
 - danych, 299, 300
 - dokumentów, 304
 - replik, 366
 - wynikowy, 308, 313
 - zgłaszanie własnych błędów, 55
 - złączanie buforów, 104
 - zmiana
 - bazy danych, 248
 - nazwy plików, 139
 - zmienna, 33
 - znacznik
 - <div>, 444
 - <html>, 438
 - , 570
 - <input>, 444
 - <script>, 438
 - <select>, 644
 - <tr>, 644
 - znajdowanie
 - dokumentów, 287, 341
 - różnych wartości pola, 315
 - zestawów dokumentów, 304
 - znak \$, 320
 - znaki specjalne, 48
 - zwracanie wartości, 43
- ## Ż
- źródła uwierzytelniania, 549
- ## Ż
- żądania HTTP, 144, 513
 - żądanie
 - AJAX, 25
 - Connect, 76
 - GET, 25, 154, 398, 579
 - GetData, 76
 - GetFile, 76
 - POST, 25, 158, 418, 516

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**



Stwórz skalowalną aplikację za pomocą najnowocześniejszych narzędzi!

W sieci króluje połączenie serwera Apache z bazą danych MySQL i językiem PHP. Do niedawna wydawało się, że pozycja tej trójcy na fotelu lidera jest niezagrożona. Jednak na horyzoncie pojawili się nowi gracze: Node.js, MongoDB oraz AngularJS! Node.js to platforma, która pozwoli Ci stworzyć niezwykle wydajną aplikację internetową przy użyciu języka JavaScript. AngularJS specjalizuje się w interfejsie użytkownika opartym na MVC. Natomiast MongoDB to jedna z najpopularniejszych baz danych NoSQL. Co wynika z połączenia tych trzech narzędzi? Przekonasz się, gdy sięgniesz po tę niezwykłą książkę!

W trakcie lektury poznasz składnię i niuanse języka JavaScript, a następnie zgłębisz tajemnice środowiska Node.js. Na kolejnych stronach znajdziesz informacje na temat korzystania ze zdarzeń, z procesów nasłuchiwania, wywołań zwrotnych oraz obsługi operacji wejścia-wyjścia. Ponadto przekonasz się, jak w Node.js zaimplementować usługi HTTP oraz skalować tworzoną aplikację. W dalszej części przejdiesz do poznawania tajników bazy MongoDB. Skonfigurujesz połączenie z bazą oraz poznasz dokumenty i ich kolekcje w MongoDB. Na sam koniec zobaczysz, jak przygotować klienta Twoich usług za pomocą AngularJS. Ta książka jest pasjonująca i prezentuje wszystkie warstwy nowoczesnej aplikacji internetowej – to lektura obowiązkowa każdego programisty!

Sięgnij po tę książkę i:

- odkryj niuanse języka JavaScript
- poznaj platformę Node.js oraz jej możliwości
- wykorzystaj potencjał platformy Node.js w połączeniu z bazą MongoDB
- poznaj AngularJS i stwórz aplikację z użyciem wzorca MVC

Brad Dayley – programista z wieloletnim doświadczeniem. Twórca licznych stron i aplikacji internetowych o różnym stopniu skomplikowania. Autor popularnych książek poświęconych bazom NoSQL, językowi JavaScript oraz innym narzędziom.



30033 numer katalogowy

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne

0 801 339900

0 601 339900

Informatyka w najlepszym wydaniu

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

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



ISBN 978-83-283-0111-5



9 788328 301115

cena: 99,00 zł


Addison
Wesley