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.

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.

Spis treści

27

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 wczytywane, 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.

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 nasłuchiwania 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 uwzględnia łańcuch funkcji obsługujących aktualizowanie zagnieżdżonych komentarzy. Przebieg procesu dodawania nowych komentarzy jest następujący:

1. Klient wysyła 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 ładuje 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

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>Komentarze</title>
  <link rel="stylesheet" type="text/css"
      href="/static/css/comment_styles.css" />
</head>
<body>
  <h2>Przykład komentarzy</h2>
  <div ng-controller="photoController">
    <div id="photosContainer">
      <div class="photoItem" ng-repeat="photo in photos">
        <img class="listPhoto" ng-click="setPhoto(photo._id)"
             ng-src="../images/{{photo.filename}}" />
      </div>
    </div>
    <div>
      <div id="photoContainer">
        <p class="imageTitle">{{photo.title}}</p>
        <img class="mainPhoto"
             ng-src="../images/{{photo.filename}}" />
      </div>
      <div id="photoComments"
          ng-init="addComment=false;replySubject='';replyBody=''">
        <div class="comment"
             ng-include="'/static/comment_thread.html'"></div>
      </div>
    </div>
  </div>
  <div ng-controller="pageController">
   <div id="pageComments"
          ng-init="addComment=false;replySubject='';replyBody=''">
     <div class="comment"
          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">
    <img class="listPhoto" ng-click="setPhoto(photo._id)"
         ng-src="../images/{{photo.filename}}" />
  </div>
</div>
```

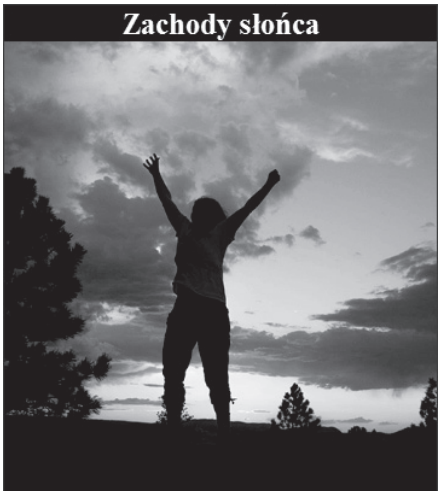
Zauważ, że w celu ustawienia bieżącego zdjęcia w kodzie kontrolera dla dyrektywy `ng-click` ustawiono metodę `setPhoto()`. Na rysunku 27.2 pokazano element wyrenderowany na stronach.

Rysunek 27.2. Zdjęcia renderowane w przeglądarce jako elementy <img>

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 `<img>`:

```
<div id="photoContainer">
  <p class="imageTitle">{{photo.title}}</p>
  <img class="mainPhoto"
       ng-src="../images/{{photo.filename}}" />
</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.

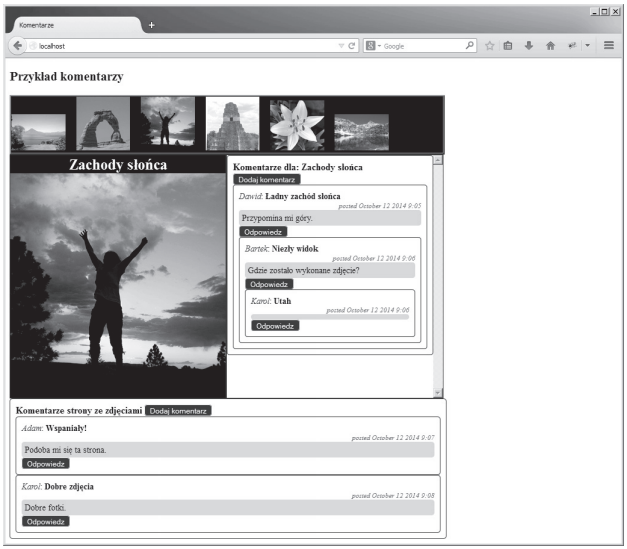
```
<div id="photoComments"
    ng-init="addComment=false;replySubject='';replyBody=''">
   <div class="comment"
        ng-include="'/static/comment_thread.html'"></div>
</div>
```

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.



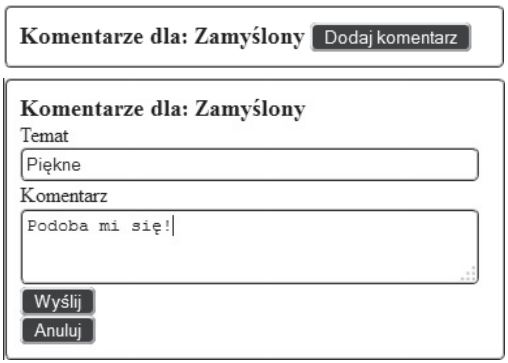
Rysunek 27.4. Wyświetlona strona zdjęć z komentarzami w obszarze zdjęć i na dole strony

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 zagnieżdżać 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`.

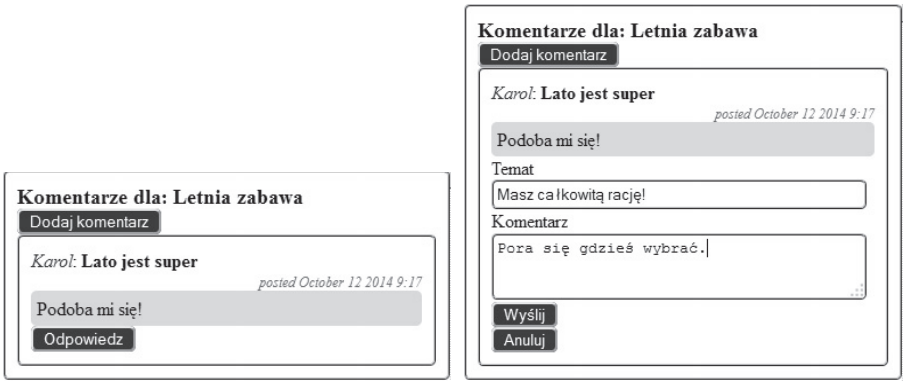
```
<div ng-repeat="comment in comment.replies"
     ng-init="reply=false;replySubject='';replyBody=''">
  <div class="comment" ng-include="'/static/comment.html'"></div>
</div>
```

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 })
```

```
    .success(function(data, status, headers, config) {
      callback(null, data);
    })
    .error(function(data, status, headers, config) {
    });
  };
}
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

```
app.controller('photoController', ['$scope', '$http', 'commentSrv',
                                   function($scope, $http, commentSrv) {
    $http.get('/photos')
     .success(function(data, status, headers, config) {
       $scope.photos = data;
       $scope.photo = $scope.photos[0];
       $scope.loadComments();
     })
     .error(function(data, status, headers, config) {
       $scope.photos = [];
     });
    $scope.loadComments = function(){
      commentSrv.getComment($scope.photo.commentId,
                            function(err, comment){
        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

```
var app = angular.module('myApp', []);
function CommentObj($http) {
  this.getComment = function(commentId, callback){
    $http.get('/comments/get', {params: {commentId: commentId}})
      .success(function(data, status, headers, config) {
        callback(null, data);
      })
      .error(function(data, status, headers, config) {
        callback(data, {});
      });
  };
  this.addComment = function(rootCommentId, parentId,
                             newComment, callback){
    $http.post('/comments/add', { rootCommentId: rootCommentId,
                                  parentCommentId: parentId,
                                  newComment: newComment })
    .success(function(data, status, headers, config) {
      callback(null, data);
    })
    .error(function(data, status, headers, config) {
    });
  };
}
app.service('commentSrv', ['$http', CommentObj]);
app.controller('photoController', ['$scope', '$http', 'commentSrv',
                                   function($scope, $http, commentSrv) {
    $http.get('/photos')
     .success(function(data, status, headers, config) {
        $scope.photos = data;
        $scope.photo = $scope.photos[0];
        $scope.loadComments();
      })
      .error(function(data, status, headers, config) {
        $scope.photos = [];
      });
    $scope.loadComments   = function(){
      commentSrv.getComment($scope.photo.commentId,
                            function(err, comment){
        if(err){
          $srope.commentThread = {};
        } else {
          $scope.commentThread = comment;
        }
      });
    };
    $scope.addReply = function(parentCommentId, subject, body){
      var newComment = {subject:subject, body:body};
      commentSrv.addComment($scope.commentThread._id,
                            parentCommentId,
                            newComment, function(err, comment){
          $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             $srope.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

C

D

E

F

J

K

L

Ł

M

N

O

P

Ś

T

Z

Ź

Ż

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 przejdziesz 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 stworz aplikacje 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.

Helion

30033 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

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

sięgnij po WIĘCEJ

KOD KORZYŚCI

ISBN 978-83-283-0111-5

9 788328 301115

Informatyka w najlepszym wydaniu

cena: 99,00 zł

Addison Wesley

ALWAYS LEARNING

PEARSON