
Nowoczesne aplikacje internetowe
Mongo, Express, AngularJS, Node.js
POZNAJ NOWE PODEJŚCIE DO APLIKACJI INTERNETOWYCH!
Jeff Dickey
Helion

Tytuł oryginału: Write Modern Web Apps with the MEAN Stack: Mongo, Express, AngularJS, and Node.js

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-1758-1

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)

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

Printed in Poland.

SPIS TREŚCI

PRZEDMOWA 9

WPROWADZENIE 10

ROZDZIAŁ 1 JAK ZMIENIA SIĘ NOWOCZESNA ARCHITEKTURA SIECI? 12
Powstanie aplikacji statycznych 14
Poznaj grubego klienta 17

ROZDZIAŁ 2 DLACZEGO JAVASCRIPT JEST DOBRYM WYBOREM DLA NOWOCZESNYCH APLIKACJI? 20
Co to jest AngularJS? 22
Co to jest Node.js? 25
Co to jest Express? 33
Co to jest MongoDB? 35

ROZDZIAŁ 3 ROZPOCZYNAMY PRACĘ NAD PROJEKTEM APLIKACJI SPOŁECZNOŚCIOWEJ 40
Utworzenie statycznej imitacji strony wyświetlającej najnowsze posty 42
Użycie AngularJS w aplikacji 43
Dodawanie nowych postów 46
Co dalej? 49

ROZDZIAŁ 4 UTWORZENIE API NODE.JS 50
Punkt wyjścia 52
Utworzenie postów za pomocą API 54
Obsługa modeli MongoDB za pomocą Mongoose 55
Użycie modeli Mongoose wraz z API 56
Co dalej? 59

ROZDZIAŁ 5 INTEGRACJA NODE.JS I ANGULARJS 60
Komponent $http 62
Użycie $http do odczytu postów z API 63
Udostępnianie pliku posts.html za pomocą Node.js 64
Użycie $http do zapisu postów w API 65
Zmiana kolejności wyświetlania postów 66
Uporządkowanie pliku server.js 67
Uporządkowanie kodu AngularJS 71
Co dalej? 75

ROZDZIAŁ 6 **AUTOMATYZACJA ZA POMOCĄ GULP** **76**
Wprowadzenie do Grunt i Gulp 78
„Witaj, świecie" w Gulp 79
Przygotowanie kodu JavaScript za pomocą Gulp 80
Kompilacja CSS za pomocą Gulp 87
Zadanie dev w Gulp 89
Inne wtyczki Gulp 91
Co dalej? 92

ROZDZIAŁ 7 **UTWORZENIE UWIERZYTELNIANIA W NODE.JS** **94**
Wprowadzenie uwierzytelniania na podstawie tokenu 96
JSON Web Token (JWT) 97
Użycie BCrypt 101
Uwierzytelnianie z użyciem MongoDB 103
Co dalej? 106

ROZDZIAŁ 8 **DODANIE ROUTINGU I UWIERZYTELNIANIA KLIENTA** **108**
Routing 110
Utworzenie formularza logowania 113
Uwierzytelnianie Express 116
Zdarzenia AngularJS 119
Uwierzytelnianie postów aplikacji społecznościowej 121
HTML5 pushstate 123
Rejestracja 124
Wylogowanie 125
Zapamiętaj mnie 126
Klucz zewnętrzny użytkownika 127
Co dalej? 128

ROZDZIAŁ 9 **OBSŁUGA POWIADOMIEŃ ZA POMOCĄ WEBSOCKET** **130**
Wprowadzenie do WebSocket 132
Jak działa WebSocket? 133
Do czego można wykorzystać WebSocket? 134
WebSocket w budowanej aplikacji społecznościowej 135
WebSocket w AngularJS 139
Architektura WebSocket 141
Dynamiczna nazwa hosta WebSocket 146
Co dalej? 147

ROZDZIAŁ 10 **WYKONYWANIE TESTÓW E2E** **148**
Konfiguracja narzędzia Protractor 150
Frameworki testowania w JavaScript 151
Utworzenie prostego testu Protractor 152
Przygotowanie w narzędziu Protractor definicji oczekiwanego zachowania 162
Wtyczka chai-as-promised 164
Kiedy należy wykonywać testy typu E2E? 165
Co dalej? 166

ROZDZIAŁ 11 **TESTOWANIE SERWERA NODE.JS** **168**
To nie całkiem są testy jednostkowe 170
Framework Mocha dla Node.js 171
Kontroler Post 173
SuperTest 174
Router bazowy 175
Użycie routera bazowego wraz z SuperTest 176
Modele w testach kontrolerów 177
Testowanie kontrolera z uwierzytelnieniem 179
Pokrycie kodu 181
Polecenie npm test 183
JSHint 184
Co dalej? 185

ROZDZIAŁ 12 **TESTOWANIE KODU ANGULARJS** **186**
Użycie narzędzia Karma 188
Bower 189
Konfiguracja narzędzia Karma 191
Podstawowy test usługi 193
Testowanie HTTP za pomocą narzędzia Karma 194
Użycie narzędzia Karma do przetestowania kontrolera 196
Testowanie za pomocą komponentów spy 200
Co dalej? 202

ROZDZIAŁ 13 **WDROŻENIE W HEROKU** **204**
Platforma jako usługa 206
Jak działa Heroku? 207
12 czynników w aplikacji 208
Wdrożenie aplikacji w Heroku 209

MongoDB w Heroku ... 211
Redis w Heroku ... 212
Kompilacja zasobów ... 214
Klaster Node.js ... 216
Co dalej? ... 217

ROZDZIAŁ 14 **WDROŻENIE W DIGITAL OCEAN ... 218**
Co to jest Digital Ocean? ... 220
Architektura jeden kontra wiele serwerów ... 221
Fedora 22 ... 222
Utworzenie serwera ... 223
Instalacja Node.js ... 225
Instalacja MongoDB ... 226
Instalacja Redis ... 227
Uruchomienie aplikacji społecznościowej ... 229
Uruchomienie aplikacji społecznościowej za pomocą systemd ... 230
Wdrożenie bez przestoju serwera ... 231
Migracja do architektury wieloserwerowej ... 235
Co dalej? ... 237
Podsumowanie ... 238

SKOROWIDZ ... 239

ROZDZIAŁ 3.

Rozpoczynamy pracę nad projektem aplikacji społecznościowej

Skoro poznałeś już komponenty tworzące stos MEAN, w tym rozdziale wykorzystamy je do zbudowania praktycznej aplikacji.

Wspomniana aplikacja będzie podobna do serwisu Twitter. Na głównej stronie będą wyświetlane najnowsze posty. Ponadto każdy użytkownik otrzyma stronę profilu, na której zostaną wyświetlone utworzone przez niego posty. Gdy będziesz poznawać różne elementy aplikacji, będę omawiać jej kolejne funkcje, takie jak uaktualnianie na żywo za pomocą WebSocket.

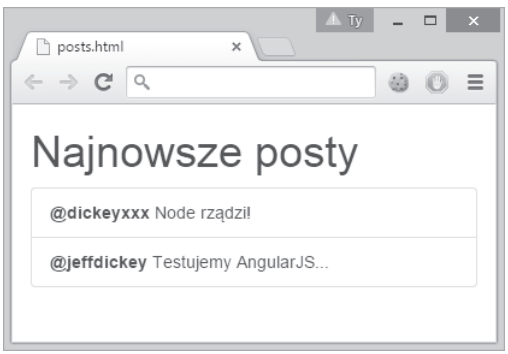
Pracę rozpoczniemy od przygotowania strony wyświetlającej najnowsze posty. Posty te pozostaną anonimowe aż do chwili zaimplementowania pewnego mechanizmu uwierzytelniania.

UTWORZENIE STATYCZNEJ IMITACJI STRONY WYŚWIETLAJĄCEJ NAJNOWSZE POSTY

Pierwszym krokiem jest utworzenie za pomocą HTML i Bootstrap (*http://getbootstrap.com/*) statycznej imitacji strony. Utwórz więc nowy katalog dla aplikacji, a następnie umieść w nim plik o nazwie *posts.html* zawierający przedstawiony poniżej kod HTML.

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="http://netdna.bootstrapcdn.com/bootstrap/3.1.1/
  ↪css/bootstrap.min.css">
</head>
<body>
  <div class='container'>
    <h1>Najnowsze posty</h1>
    <ul class='list-group'>
      <li class='list-group-item'>
        <strong>@dickeyxxx</strong>
        <span>Node rządzi!</span>
      </li>
      <li class='list-group-item'>
        <strong>@jeffdickey</strong>
        <span>Testujemy AngularJS...</span>
      </li>
    </ul>
  </div>
</body>
</html>
```

Teraz wyświetl ten dokument HTML w przeglądarce internetowej. Powinieneś otrzymać wynik pokazany na rysunku 3.1. Lubię rozpoczynać budowę wszystkich aplikacji AngularJS, stosując powyższe podejście — najpierw umieszczam na stronie statyczny kod HTML, a następnie zastępuję go dynamicznym kodem AngularJS.



RYSUNEK 3.1. *Strona wyświetlająca najnowsze posty*

UŻYCIE ANGULARJS W APLIKACJI

Framework AngularJS pozwala na dołączanie obiektów JavaScript do modelu DOM (czyli po prostu elementów HTML). W kontrolerze przeprowadzasz połączenie obiektów z tak zwanym `$scope` oraz definiujesz sposób jego wyświetlenia w kodzie HTML.

Przed znacznikiem zamykającym `</body>` umieść polecenie dołączające framework AngularJS:

```
<script src='https://ajax.googleapis.com/ajax/libs/angularjs/1.2.18/angular.js'></script>
```

Teraz możesz zadeklarować element `<body>` jako aplikację AngularJS:

```
<body ng-app='app'>
```

Pozostało już tylko zdefiniowanie dopasowanego modułu. Operację tę przeprowadzamy w znaczniku `<script>` umieszczonym po znaczniku importującym framework AngularJS:

```
<script>
  angular.module('app', [])
</script>
```

W tej części książki będziemy tworzyć kod wewnątrz znacznika `<script>` znajdującego się w dokumencie HTML, zamiast odwoływać się do zewnętrznych plików JavaScript. Takie rozwiązanie może wydawać się niechlujne, ale to tylko stan przejściowy, który ulegnie zmianie, gdy wykorzystamy Node.js do obsługi kodu HTML.

Moduły pozwalają na rozdzielenie kodu AngularJS. Powyżej mamy deklarację modułu. Pierwszy argument to nazwa modułu, natomiast drugi to tablica wymieniająca moduły, od których zależy działanie modułu właśnie tworzonego. Moduły wykorzystujemy w celu uzyskania dostępu do komponentów, takich jak `ngAnimate` (narzędzia animacji), `ngRoute` (działający po stronie klienta router dla aplikacji w postaci pojedynczej strony), lub własnych modułów. Do modułów jeszcze wrócimy. Teraz zapamiętaj, że tworzona aplikacja musi być zadeklarowana jako moduł.

W celu zadeklarowania kontrolera wywołujemy metodę `.controller()` w egzemplarzu modułu. Metoda ta pobiera dwa argumenty: nazwę i funkcję używaną do zbudowania egzemplarza kontrolera. We wszystkich komponentach AngularJS stosowany jest ten sam wzorzec.

Zmodyfikuj znacznik `<script>`, aby przedstawiał się następująco:

```
<script>
  var app = angular.module('app', [])
  app.controller('PostsCtrl', function ($scope) {
    $scope.posts = [
      {
        username: 'dickeyxxx',
        body: 'Node rządzi!'
      },
      {
        username: 'jeffdickey',
        body: 'Testujemy AngularJS...'
      }
    ]
  })
</script>
```

Moduł aplikacji jest przechowywany w zmiennej `app`. Utworzyliśmy w nim kontroler o nazwie `PostsCtrl`, który zawiera tablicę przedstawiającą posty wyświetlone na stronie HTML.

Podczas zadeklarowania kontrolera wykonaliśmy operację, która w świecie AngularJS jest określana mianem *wstrzyknięcia zależności $scope*.

Utworzenie kontrolera (lub innego komponentu AngularJS) wymaga zadeklarowania funkcji, podobnie jak to zrobiliśmy wcześniej. Kiedy deklarujesz argumenty, AngularJS wyszukuje je na podstawie podanych nazw. W przypadku zmiany `$scope` na `$foobar` może się pojawić komunikat o błędzie (choć jeszcze nie w utworzonym dotąd kodzie, ponieważ kontroler nie został wczytany). Według mnie wstrzykiwanie zależności to jedna z najważniejszych możliwości oferowanych przez AngularJS.

Czym jest `$scope`? Jest to po prostu obiekt, do którego masz dostęp zarówno w kodzie HTML, jak i kontrolerze.

Więcej przykładów zależności poznasz w dalszej części książki. W tym momencie zawartość tablicy `$scope.posts` po prostu wyświetlimy na stronie. Przedstawiony poniżej kod HTML umieść na stronie (istniejący statyczny kod HTML pozostaw w celach porównawczych):

```
<div ng-controller='PostsCtrl'>
  {{ posts }}
</div>
```

Gdy dodasz powyższy kod, na stronie powinieneś otrzymać dane w formacie JSON przedstawiające zawartość tablicy `$scope.posts` zdefiniowanej w kontrolerze. Wewnątrz nawiasów klamrowych można umieścić kod JavaScript, więc jeśli chcesz wyświetlić jedynie treść pierwszego postu, to zmodyfikuj kod w następujący sposób:

```
<div ng-controller='PostsCtrl'>
  {{ posts[0].body }}
</div>
```

Pamiętaj, że podczas współdzielenia danych kontrolera z widokiem używamy `$scope`. Z kolei podczas odwoływania się z poziomu widoku do danych kontrolera *opuszczamy* `$scope`.

Skoro dotarliśmy tak daleko, możemy zakończyć „angularyzację” strony, przynajmniej jej statycznej wersji. Oto kod źródłowy ukończonej strony:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="http://netdna.bootstrapcdn.com/bootstrap/3.1.1/css/
  ↪bootstrap.min.css">
</head>
<body ng-app='app'>
  <div ng-controller='PostsCtrl' class='container'>
    <h1>Najnowsze posty</h1>
    <ul class='list-group'>
      <li ng-repeat='post in posts' class='list-group-item'>
        <strong>@{{ post.username }}</strong>
        <span>{{ post.body }}</span>
      </li>
    </ul>
```

```
  </div>
  <script src='https://ajax.googleapis.com/ajax/libs/angularjs/1.2.18/angular.js'></script>
  <script>
    var app = angular.module('app', [])
    app.controller('PostsCtrl', function ($scope) {
      $scope.posts = [
        {
          username: 'dickeyxxx',
          body: 'Node rządzi!'
        },
        {
          username: 'jeffdickey',
          body: 'Testujemy AngularJS...'
        }
      ]
    })
  </script>
</body>
</html>
```

DODAWANIE NOWYCH POSTÓW

Na obecnym etapie pracy połączyliśmy kod z AngularJS w taki sposób, aby kontroler mógł dostarczać dane, które następnie będą wyświetlone przez widok. Jednak teraz dane pozostają dokładnie takie same jak w przygotowanej na początku statycznej wersji strony.

Potrzebny jest nam element na stronie pozwalający użytkownikom dodawać nowe posty.

Jeżeli nowy element zostanie umieszczony w tablicy `$scope.posts`, AngularJS automatycznie uaktualni widok w celu wyświetlenia tego elementu. Dlatego też gdy umieścimy na stronie element `<input>` i przycisk pozwalający na dodanie nowego postu, wprowadzone przez użytkownika dane trzeba dodać do tablicy.

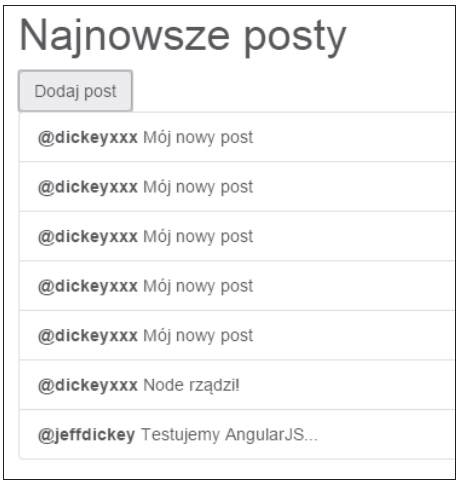
Aby zilustrować tę koncepcję, dodamy przycisk na stronie. Umieść go przed znacznikiem `<ul>` o klasie `list-group`:

```
<button ng-click='addPost()' class='btn btn-default'>Dodaj post</button>
```

Dyrektywa `ng-click` jest zwykle używana do wywoływania funkcji w `$scope`. W tym momencie kliknięcie przycisku nie powoduje żadnej reakcji, ale po zdefiniowaniu funkcji będzie można dodać nowe posty do już istniejących. W kontrolerze umieść następujący kod:

```
$scope.addPost = function () {
  $scope.posts.unshift({
    username: 'dickeyxxx',
    body: 'Treść nowego postu!'
  })
}
```

Wykorzystana metoda `unshift()` to metoda JavaScript, która powoduje umieszczenie nowego elementu na początku tablicy. Po wprowadzonych zmianach kliknięcie przycisku *Dodaj post* spowoduje dodanie elementów do listy (patrz rysunek 3.2).



RYSUNEK 3.2. *Oparta na AngularJS strona wyświetlająca najnowsze posty*

Ponieważ nie utworzyliśmy żadnego kodu odpowiedzialnego za uaktualnienie listy po zmianie tablicy `$scope.posts`, zastosowane podejście niewątpliwie pozwala na wyraźne oddzielenie logiki uaktualniającej dane od logiki widoku. Jeżeli posty zostaną uaktualnione przez operację usunięcia postu, wywołanie Ajax lub informacje WebSocket, to zajdzie potrzeba uaktualnienia `$scope.posts`, a AngularJS samodzielnie zajmie się uaktualnieniem widoku.

Na stronie musimy jeszcze umieścić znacznik `<input>` pozwalający użytkownikowi na dodanie treści nowego postu. W tym celu dodajemy znacznik `<input>`, dołączamy jego dane do obiektu `$scope`, który następnie powiązujemy z funkcją `addPost()`. Przed przyciskiem *Dodaj post* umieść następujący kod HTML:

```
<input ng-model='postBody' class='form-control' />
```

Dyrektywa `ng-model` powoduje powiązanie danego elementu z podaną właściwością w `$scope`. Jeżeli teraz w kontrolerze uzyskasz dostęp do `$scope.postBody`, to otrzymasz dane wprowadzone w nowo dodanym polu tekstowym.

Aby wykorzystać dane, które zostały wprowadzone przez użytkownika, zamiast danych na stałe zdefiniowanych wcześniej w kodzie, w następujący sposób uaktualnij funkcję `addPost()`:

```
$scope.addPost = function () {
  $scope.posts.unshift({
    username: 'dickeyxxx',
    body: $scope.postBody
  })
}
```

Dane wprowadzone przez użytkownika w nowo dodanym znaczniku `<input>` będą stanowiły treść postu.

Niedogodność przygotowanego rozwiązania wiąże się z faktem, że zawartość pola tekstowego nie jest zerowana po dodaniu postu. Można to łatwo naprawić przez przypisanie w funkcji `addPost()` wartości `null` właściwości `$scope.postBody`:

```
$scope.postBody = null
```

Ponieważ dołączanie danych jest dwukierunkowe, przypisanie dowolnej wartości do `$scope.postBody` spowoduje uaktualnienie elementu `<input>`, i na odwrót.

Jedyny problem zastosowanego podejścia polega na tym, że użytkownik może tworzyć puste posty. Rozwiązaniem tego problemu jest opakowanie wywołania funkcji `addPost()` następującym wyrażeniem warunkowym:

```
if ($scope.postBody) {
```

Poniżej przedstawiam pełny kod źródłowy aplikacji wraz z dodanym kodem Bootstrap w celu nadania stylu stronie:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="http://netdna.bootstrapcdn.com/bootstrap/3.1.1/css/
  ↪bootstrap.min.css">
</head>
<body ng-app='app'>
  <div ng-controller='PostsCtrl' class='container'>
    <h1>Najnowsze posty</h1>
```

```
    <form role='form'>
      <div class='form-group'>
        <div class='input-group'>
          <input ng-model='postBody' class='form-control' />
          <span class='input-group-btn'>
            <button ng-click='addPost()' class='btn btn-default'>
            → Dodaj post</button>
          </span>
        </div>
      </div>
    </form>
    <ul class='list-group'>
      <li ng-repeat='post in posts' class='list-group-item'>
        <strong>@{{ post.username }}</strong>
        <span>{{ post.body }}</span>
      </li>
    </ul>
  </div>
  <script src='https://ajax.googleapis.com/ajax/libs/angularjs/1.2.18/angular.js'></script>
  <script>
    // Utworzenie modułu app.
    var app = angular.module('app', [])
    // Utworzenie modułu PostsCtrl.
    // Wstrzyknięcie zależności $scope.
    app.controller('PostsCtrl', function ($scope) {
      // Wykonanie funkcji nastąpi po kliknięciu przycisku "Dodaj post".
      $scope.addPost = function () {
        // Dodany będzie jedynie post zawierający treść.
        if ($scope.postBody) {
          // Umieszczenie nowego postu na początku tablicy $scope.posts.
          $scope.posts.unshift({
            username: 'dickeyxxx',
            body: $scope.postBody // Treścią postu będzie tekst wprowadzony przez użytkownika.
          })
          // Usunięcie zawartości pola.
          $scope.postBody = null
        }
      }
      // Dane początkowe.
      $scope.posts = [
        {
          username: 'dickeyxxx',
          body: 'Node rządzi!'
        },
        {
          username: 'jeffdickey',
          body: 'Testujemy AngularJS...'
        }
      ]
    })
  </script>
</body>
</html>
```

CO DALEJ?

W tym momencie masz w pełni funkcjonującą aplikację przeznaczoną do publikowania pewnych informacji. Na pewno masz pełną świadomość największego problemu utworzonej aplikacji, czyli braku możliwości zapisania danych. Posty zostaną trwale utracone po zamknięciu przeglądarki internetowej przez użytkownika. Wprawdzie posty można zapisać w lokalnym magazynie danych przeglądarki, ale wówczas nie będzie można współdzielić danych z innymi użytkownikami. (Współdzielenie jest niezwykle ważnym aspektem w aplikacjach społecznościowych).

W następnym rozdziale opracujemy więc API, które pozwoli aplikacji na zapisywanie danych. Z kolei w rozdziale 5. zintegrujemy zbudowaną aplikację AngularJS z API Node.js, które również przygotujemy.

SKOROWIDZ

A

Ajax, 14
algorytm
  BCrypt, 101
  algorytm hash, 101
AMQP, 143
AngularJS, 22
  testowanie kodu, 186
  WebSocket, 139
API, 16
API Node.js, 50
aplikacje
  MEAN, 9
  statyczne, 14
architektura
  oparta na zdarzeniach, 28
  sieciowa, 12
    infrastruktura, 17
    modularność, 18
    prototypowanie, 18
    ustandaryzowane narzędzia, 17
    wydajność, 17
  WebSocket, 141
  wieloserwerowa, 235
automatyczny sharding, 37
automatyzacja, 76

B

baza danych MongoDB, 37
bazy danych, 34
  centralne, 236
  oparte na dokumentach, 35
BCrypt, 101
BDD, 171
błąd o kodzie 404, 115
Bootstrap, 42
Bower, 189

C

centralne bazy danych, 236
chmura, 220
ciągła integracja, continuous integration, 183
CommonJe, 29
CSS, 87

D

debuger Protractor, 159
demon mongod, 38
Digital Ocean, 218, 220
  tworzenie konta, 223
dodawanie postów, 46
dokumenty, 35
dołączanie narzędzia Uglifier, 83
DOM, 22
droplet, 224
dynamiczna nazwa hosta, 146
dyrektywa
  ng-click, 46
  ng-controller, 120
  ng-model, 47
  ng-repeat, 23
działanie
  architektury sieciowej, 17
  Heroku, 207

E

element
  <body>, 43
  <input>, 47
  , 23
  , 23
Express, 33

F

Fedora 22, 222
filtr beforeEach, 197
format
  BSON, 36
  JSON, 24
  JWT, 97
formularz logowania, 113, 159
framework
  AngularJS, 22
  Express, 52
  Jasmine, 151
  jQuery, 22
  Mocha, 151, 171
  QUnit, 151
frameworki testowania, 151
funkcja
  .login(), 114
  addPost(), 47
  create(), 200
  createUser(), 124
  encode(), 97
  fetch(), 200
  inject(), 193
  publish(), 143

G

gruby klient, thick client, 17
Grunt, 78
Gulp, 76, 78
  kompilacja CSS, 87
  przygotowanie kodu, 80
  wtyczki, 91
  zadanie dev, 89
gulp-nodemon, 89

H

hasło, 99
Heroku, 204
  12 czynników, 208
  MongoDB, 211
  panel administracyjny, 207
  Redis, 212
  wdrożenie aplikacji, 209
HTML, 42
HTML5 pushstate, 123

I

imitacja strony, 42
informacje
  o błędzie, 155
  o zalogowanym użytkowniku, 118
instalacja
  Bower, 189
  gulp-nodemon, 89
  Karmy, 191
  MongoDB, 38, 226
  Node.js, 30, 225
  pakietu Express, 33
  Redis, 227
integracja Node.js i AngularJS, 60

J

Jasmine, 151
język JavaScript, 14
jQuery, 22
JSHint, 184

K

klaster Node.js, 216
klient $http, 62
klucz zewnętrzny użytkownika, 127
kod Redis, 212
kolekcje, 35
kompilacja, 84
  CSS, 87
  zasobów, 214
komponent $http, 62
komponenty
  spy, 200
  stosu MEAN, 21
komunikat o błędzie, 83
konfiguracja narzędzia
  Karma, 191
  Protractor, 150, 153
kontroler
  ApplicationController, 126
  Post, 173
  PostsCtrl, 44

L

Linux, 222
logowanie, 113
lokalizator Protractor, 157

Ł

łącze logowania, 157

M

mapa źródła, 85
MEAN, 9
  AngularJS, 22
  Express, 33
  MongoDB, 36
  Node.js, 25
mechanizm cookies, 96
menedżer npm, 29
migracja do architektury wieloserwerowej, 235
mobilne API, 16
Mocha, 151
model DOM, 22
modele
  MongoDB, 55
  Mongoose, 56
  w testach kontrolerów, 177
MongoDB, 35
  baza danych, 38
  dokument, 38
  hierarchia danych, 38
  instalacja, 38
  kolekcja, 38
  uwierzytelnianie, 103
  wstawianie dokumentów, 39
  wykonywanie zapytań, 39
Mongoose, 55

N

narzędzie
  Heroku, 210
  JSHint, 184
  Karma, 188
  Protractor, 150
  SuperTest, 174
  Uglifier, 82
Node.js, 25
  bazy danych, 34
  framework Mocha, 171
  instalacja, 30
  integracja z AngularJS, 60
  serwer WWW, 31
  środowisko produkcyjne, 26
  testowanie serwera, 168
  tworzenie API, 50
  udostępnianie pliku, 64
  uwierzytelnianie, 94
  użycie modułów, 29
  wydajność, 27
nowoczesne aplikacje, 20

O

obiekt
  $scope, 44
  mockPostsSvc, 198
obietnica, 62
obsługa
  modeli MongoDB, 55
  powiadomień, 130
  pushstate, 123
  uwierzytelniania, 104
  wątków, 28
  żądań API, 54
  żądań GET, 179
  żądań POST, 179
odczyt postów, 63

P

PaaS, platform as a service, 206
pakiet
  blanket, 181
  body-parser, 52
  jwt-simple, 97
panel
  administracyjny, 207
  dropletu, 224
panoramowanie, 14
platforma jako usługa, PaaS, 206
plik
  .jshintrc, 184
  api.js, 176

plik
app.js, 80, 84
app-express.js, 33
config.js, 116
css.js, 88
foo.js, 29
gulpfile.js, 88
making_a_post.spec.js, 152, 158
module.js, 81
package.json, 52
post.js, 55
posts.html, 42, 64
posts.js, 173
posts.svc.js, 81
protractor.conf.js, 153
routes.js, 110
server.js, 55, 58, 67
user.js, 116
websockets.js, 136
pokrycie kodu, 181
polecenia systemd, 228
polecenie
curl, 54, 104
ensureIndex, 38
heroku logs, 210
mongo, 38
npm test, 183
ponowna kompilacja, 84
posty
kolejność wyświetlania, 66
odczyt, 63
powiadamianie o publikacji, 136
zapis, 65
preprocesor CSS
Less, 87
Sass, 87
Stylus, 87
programowanie w stylu BDD, 171
projekt aplikacji społecznościowej, 40
protokół WebSocket, 134
Protractor
konfiguracja, 150, 153
lokalizator, 157
oczekiwane zachowania, 162
tworzenie testu, 152
uruchomienie Node.js, 155
usunięcie zawartości bazy danych, 160
utworzenie postu, 160
przeniesienie kodu
AngularJS, 72
API, 68
przestrzenie nazw, 69
publikowanie zdarzeń, 145

Q

QUnit, 151

R

rejestracja, 124
repozytorium Git, 214
router bazowy, 175
routing, 108, 110
równoważenie obciążenia, 235

S

Sass, 87
serwer
chmury, 220
Redis, 212
sharding, 37
sieć prywatna, 236
skalowanie
pionowe, 37
poziome, 37
spaghetti JavaScript, 15
startup, 25
statyczna imitacja strony, 42
stos MEAN, 9, 21
styl strony, 47
Stylus, 87

Ś

środowisko
open source, 25
produkcyjne, 25
typu startup, 25

T

testowanie
  HTTP, 194
  kodu AngularJS, 186
  kontrolera z uwierzytelnieniem, 179
  kontrolerów, 196
  serwera Node.js, 168
  usługi, 193
testy
  E2E, 148, 152, 165
  funkcjonalne, 152
  jednostkowe, 170
tęczowa tablica, rainbow table, 101
token, 96, 97
  JWT, 97
trasy, 69
tworzenie
  API Node.js, 50
  dropletu, 224
  formularza logowania, 113
  postów, 54
  serwera, 223
  serwera WWW, 31
  statycznej imitacji strony, 42
  testu protractor, 152
  tokenów, 97
  uwierzytelniania, 94
typ ObjectId, 36

U

udostępnienie zasobów statycznych, 71
Uglifier, 82
układ graficzny, 70
uporządkowanie kodu, 71
uruchomienie aplikacji społecznościowej, 229, 230
usunięcie zawartości bazy danych, 160
uwierzytelnianie, 94
  klienta, 108
  mechanizm cookies, 96
  postów, 121
  token, 96
  z użyciem MongoDB, 103
użycie
  $http, 63, 65
  AngularJS, 43
  BCrypt, 101
  JWT, 98
  klastra Node.js, 216
  modeli Mongoose, 56
  MongoDB, 38
  narzędzia Karma, 188, 196
  routera bazowego, 176

W

wartość null, 197
wdrożenie aplikacji, 209
  bez przestoju, 231
  w Digital Ocean, 218
  w Heroku, 204
WebSocket, 130, 132
  architektura, 141
  dynamiczna nazwa hosta, 146
  nawiązanie połączenia, 135
  w AngularJS, 139
  zapewnienie bezpieczeństwa, 141
weryfikacja hasła, 99
wstrzyknięcie zależności $http, 63
wtyczka
  chai-as-promised, 164
  Redis To Go, 212
wtyczki Gulp, 91
wydajność w Node.js, 27
wylogowanie, 125
wyświetlanie postów, 66
wywołania zwrotne, 29

Z

zapis postów, 65
zarejestrowanie błędu, 85
zasoby statyczne, 71
zdarzenia, 28
  AngularJS, 119
ZeroMQ, 143
zmienna NODE_ENV, 116
znacznik, *Patrz* element
znak $, 62

Ż

żądanie
  GET, 68
  POST, 68

Aplikacje internetowe już nigdy nie będą takie same!

Na tradycyjne aplikacje składały się dwa elementy — ciężki back-end, odpowiadający za przetwarzanie danych, ich walidację i przechowywanie, oraz klient, którego zadaniem była prosta komunikacja z użytkownikiem. To rozwiązanie rodziło liczne kłopoty, przede wszystkim ze skalowaniem. Od nowoczesnych aplikacji użytkownicy wymagają najwyższej wydajności, niezawodności, możliwości działania w trybie offline oraz wygody użytkowania. Tradycyjne podejście do tworzenia aplikacji już się nie sprawdza. Potrzebujesz czegoś nowego!

Na ratunek przychodzą MongoDB, Express, AngularJS oraz Node.js. Dzięki temu zestawowi narzędzi będziesz w stanie stworzyć zaawansowaną aplikację z wygodnym interfejsem użytkownika oraz szerokimi możliwościami skalowania. Sięgnij po tę książkę i dowiedz się, czemu warto wybrać AngularJS, czym są bazy NoSQL i jakie są zalety Node.js. Ponadto nauczysz się budować wydajne API, integrować Node.js z AngularJS, automatyzować zadania za pomocą pakietu Gulp oraz korzystać z zaawansowanych mechanizmów autoryzacji. Na sam koniec zaznajomisz się z dobrymi praktykami — testami API i klienta.

DZIĘKI TEJ KSIĄŻCE:

- Poznasz możliwości AngularJS oraz Node.js
- Przekonasz się, dlaczego warto wybrać MongoDB
- Przygotujesz praktyczne API w Node.js
- Poznasz narzędzie Gulp
- Wykorzystasz JSON Web Token do uwierzytelniania
- Przetestujesz aplikację
- Nauczysz się tworzyć nowoczesne aplikacje internetowe!

JEFF DICKEY — wszechstronny programista sieciowy. Doświadczenie zdobywał, pracując z licznymi startupami z okolic San Francisco i Los Angeles. Obecnie prowadzi zespoły programistów i opracowuje projekty. Unika biurokracji, jest zaangażowany w serię spotkań Code for America Brigade w Los Angeles.

Helion
38360 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
ISBN 978-83-283-1758-1
9 788328 317581
Informatyka w najlepszym wydaniu
cena: 47,00 zł