

Express.js

Tworzenie aplikacji
sieciowych w Node.js

Azat Mardan

Tytuł oryginału: Pro Express.js: Master Express.js: The Node.js Framework For Your Web Development

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-1664-5

Original edition copyright © 2014 by Azat Mardan.
All rights reserved.

Polish edition copyright © 2016 by HELION SA.
All rights reserved.

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 the Publisher.

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

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

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

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

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/expres>
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	11
O recenzentach technicznych	13
Wstęp	15
Wprowadzenie	17
Dlaczego napisałem tę książkę?	17
Dla kogo jest przeznaczona ta książka?	18
O czym jest ta książka?	18
O czym nie jest ta książka?	18
Przykłady	19
Errata i dane kontaktowe	19
Podziękowania	21
Część I	
Rozpoczęcie pracy	23
Rozdział 1. Rozpoczęcie pracy z Express.js	25
Jak działa Express.js?	26
Zależności firm trzecich	27
Utworzenie egzemplarza	27
Nawiązanie połączenia z bazą danych	27
Konfiguracja ustawień aplikacji Express.js	28
Zdefiniowanie oprogramowania pośredniczącego	28
Zdefiniowanie tras	28
Uruchomienie aplikacji	28
Instalacja Express.js	29
Instalacja Express.js Generator	32
Podsumowanie	34

Rozdział 2. Witaj, świecie	35
Rozpoczęcie pracy	35
Użycie procedur obsługi żądań	36
Wyświetlanie komunikatów w powłoce	37
Usprawnienie aplikacji	38
Polecenia generatora	39
Wygenerowanie szkieletu aplikacji Express.js	40
Przegląd struktury aplikacji	41
Plik app.js	41
Moduły i architektura MVC	45
Monitorowanie pod kątem zmian w plikach	46
Podsumowanie	47
Część II Dokładne omówienie API	49
Rozdział 3. Konfiguracja, ustawienia i środowiska	51
Konfiguracja	51
app.set() i app.get()	52
app.enable() i app.disable()	52
app.enabled() i app.disabled()	53
Ustawienia	53
env	54
view cache	54
view engine	55
views	55
trust proxy	56
jsonp callback name	56
json replacer i json spaces	57
case sensitive routing	58
strict routing	59
x-powered-by	60
etag	60
query parser	61
subdomain offset	62
Środowiska	62
Podsumowanie	65
Rozdział 4. Praca z oprogramowaniem pośredniczącym	67
Zastosowanie oprogramowania pośredniczącego	68
Najważniejsze oprogramowanie pośredniczące	71
compression	72
morgan	74
body-parser	75
urlencoded()	76
cookie-parser	77
express-session	77
csurf	78

express.static()	79
connect-timeout	80
errorhandler	82
method-override	82
response-time	83
serve-favicon	84
serve-index	85
vhost	87
connect-busboy	87
Inne oprogramowanie pośredniczące	88
Podsumowanie	89
Rozdział 5. Silniki szablonów i Consolidate.js	91
Jak używać silników szablonów?	92
app.engine()	93
Rzadziej stosowane biblioteki	94
Dostępne silniki szablonów	97
Jade	97
Haml.js	97
EJS	97
Handlebars.js	97
Adaptory Hogan.js	97
Combyne.js	98
Swig	98
Whiskers	98
Blade	98
Haml-Coffee	98
Webfiller	98
Consolidate.js	98
Podsumowanie	100
Rozdział 6. Parametry i routing	101
Parametry	101
app.param()	105
Routing	108
app.NAZWA()	108
app.all()	111
Ukośniki na końcu	111
Klasa Router	111
router.route(path)	111
Procedury obsługi żądania	113
Podsumowanie	114
Rozdział 7. Obiekt request w Express.js	117
request.query	118
request.params	120
request.body	121
request.route	122

request.cookies	123
request.signedCookies	124
request.header() i request.get()	125
Inne atrybuty i metody	125
Podsumowanie	129
Rozdział 8. Obiekt response w Express.js	131
response.render()	132
response.locals	134
response.set()	135
response.status()	137
response.send()	138
response.json()	141
response.jsonp()	142
response.redirect()	144
Inne właściwości i metody odpowiedzi	144
Strumienie	148
Podsumowanie	150
Rozdział 9. Obsługa błędów i uruchamianie aplikacji	151
Obsługa błędów	151
Uruchomienie aplikacji	155
app.locals	155
app.render()	155
app.mountpath	156
app.on('mount', funkcja(nadrzędna){...})	157
app.path()	157
app.listen()	157
Podsumowanie	161
Część III Rozwiązywanie najczęściej pojawiających się problemów	163
Rozdział 10. Abstrakcja	165
Oprogramowanie pośredniczące	165
Trasy	166
Połączenie oprogramowania pośredniczącego i tras	168
Podsumowanie	170
Rozdział 11. Wskazówki dotyczące baz danych, kluczy i strumieni	171
Użycie baz danych w modułach	171
Klucze i hasła	173
Plik JSON	173
Zmienne środowiskowe	175
Strumienie	175
Podsumowanie	179

Rozdział 12. Redis i wzorce uwierzytelniania	181
Redis	181
Wzorce uwierzytelniania	184
Podsumowanie	185
Rozdział 13. Wielowątkowość z użyciem klastrów	187
Przykład wielowątkowości	187
Podsumowanie	190
Rozdział 14. Stosowanie bibliotek Stylus, Less i Sass	191
Stylus	191
Less	192
Sass	193
Podsumowanie	193
Rozdział 15. Zapewnienie bezpieczeństwa	195
Cross-Site Request Forgery	195
Przetwarzanie uprawnień	197
Nagłówki zabezpieczeń w HTTP	198
Weryfikacja danych wejściowych	199
Podsumowanie	201
Rozdział 16. Socket.IO i Express.js	203
Użycie Socket.IO	203
Uruchomienie aplikacji	207
Podsumowanie	208
Rozdział 17. Domeny i Express.js	209
Zdefiniowanie problemu	209
Prosty przykład oparty na domenie	210
Utworzenie aplikacji opartej na domenie	211
Podsumowanie	214
Rozdział 18. Sails.js, DerbyJS, LoopBack i inne frameworki	215
Sails.js	215
DerbyJS	217
LoopBack	219
Inne frameworki	222
Podsumowanie	222
Część IV Przykłady	223
Rozdział 19. Galeria Instagram	225
Zaczynamy pracę nad galerią Instagram	225
Wyświetlanie galerii	229
Podsumowanie	230
Rozdział 20. Aplikacja Todo	231
Ogólne omówienie projektu	232
Konfiguracja	235
Plik app.js	237

Trasy	242
Jade	246
Less	250
Podsumowanie	251
Rozdział 21. API REST	253
Podstawy API RESTful	254
Wprowadzenie do testów	255
Zależności	259
Implementacja serwera	260
Podsumowanie	265
Rozdział 22. Aplikacja HackHall	267
Co to jest HackHall?	267
Uruchomienie HackHall	268
Struktura aplikacji	274
Plik package.json	275
Aplikacja Express.js	276
Trasy	280
Plik index.js	280
Plik auth.js	280
Plik main.js	284
Plik users.js	287
Plik application.js	292
Plik posts.js	295
Modele Mongoose	301
Testy Mocha	307
Podsumowanie	312
Dodatki	313
Dodatek A Dalsza lektura i zasoby	315
Inne frameworki Node.js	315
Książki poświęcone Node.js	316
Klasyka JavaScript	318
Kursy	318
Dodatek B Migracja Express.js 3.x do 4.x: oprogramowanie pośredniczące, trasy i inne zmiany	319
Wprowadzenie do oprogramowania pośredniczącego, które nie zostało dołączone do wydania Express.js 4	319
Usunięcie z aplikacji Express.js 4 metod uznanych za przestarzałe	321
app.configure()	321
app.router()	321
res.on('header')	322
res.charset	322
res.headerSent	322
req.accepted()	322

Inne zmiany wprowadzone w Express.js 4	322
app.use()	322
res.location()	323
app.route()	323
json spaces	323
req.params	323
res.locals	323
req.is	323
Działający w powłoce generator Express.js	323
Poznanie nowego egzemplarza tras w Express.js 4 oraz sposoby jego łączenia z innymi	323
Kolejne zasoby dotyczące migracji do Express.js 4	325
Dodatek C Ściąga z Express.js 4	327
Instalacja	328
Generator	328
Podstawy	329
Trasy i metody HTTP	329
Żądanie	329
Skróty nagłówek żądania	330
Odpowiedź	330
Sygnatury procedury obsługi	330
Stylus i Jade	330
Body	331
Static	331
Oprogramowanie pośredniczące Connect	331
Inne popularne oprogramowanie pośredniczące	331
Dodatek D ExpressWorks	333
Instalacja	333
Sposób użycia	334
Zerowanie	334
Zadania	334
Witaj, świecie	334
Jade	335
Stary dobry formularz	335
Static	335
Style CSS	335
Param pam pam	335
Co znajduje się w zapytaniu?	336
Dane JSON	336
Podsumowanie	336
Skorowidz	337



Galeria Instagram

Jeżeli kolejne rozdziały książki czytasz po kolei, to poznałeś już ważne, choć suche szczegóły API oraz miałeś styczność z jedynie abstrakcyjnymi rozwiązaniami. Część IV powinna okazać się najbardziej ekscytująca, ponieważ cztery znajdujące się tutaj rozdziały zostały poświęcone programowaniu i przykładom.

W tym rozdziale dowiesz się, jak używać Express.js wraz z usługami zewnętrznymi opracowanymi przez firmy trzecie (tutaj to API Storify). Celem budowanej aplikacji jest pobranie zdjęć Instagram ze Storify, a następnie wyświetlenie ich w galerii. Poza frameworkiem Express.js wykorzystamy jeszcze następujące moduły:

- `superagent` (<https://www.npmjs.com/package/superagent>);
- `consolidate` (<https://www.npmjs.com/package/consolidate>);
- `handlebars` (<https://www.npmjs.com/package/handlebars>).

Zdecydowałem się na wymienione moduły, ponieważ są one popularne w pewnych kręgach programistycznych Node.js. Istnieje więc duże prawdopodobieństwo, że je napotkasz lub będziesz z nich korzystał w przyszłości.

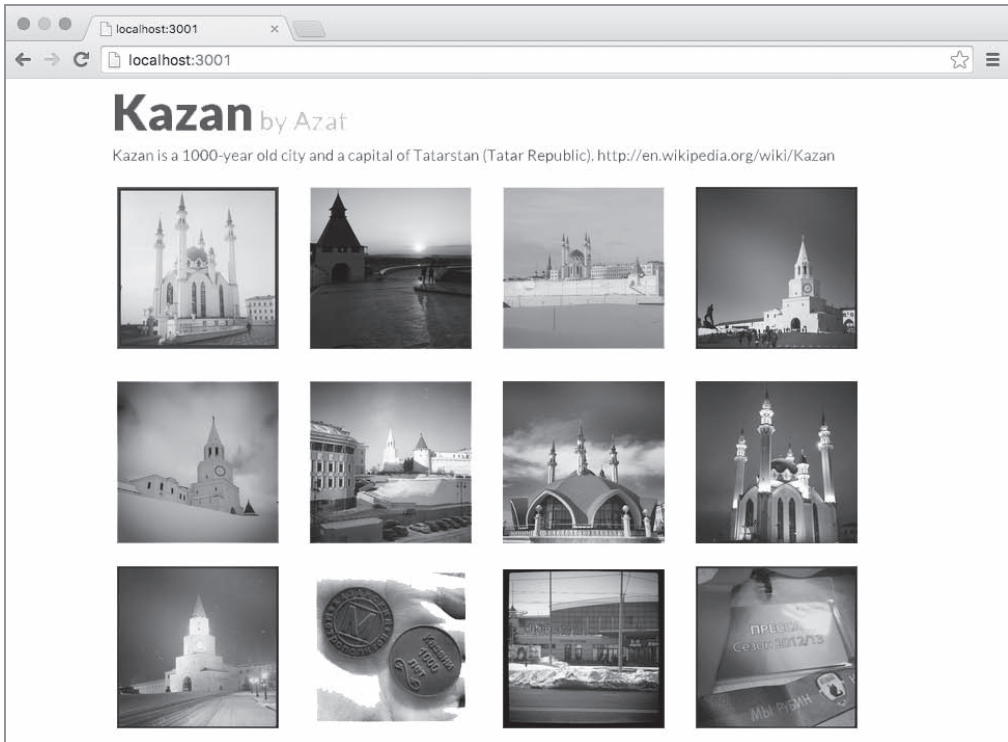
■ **Uwaga** Pełny kod źródłowy przykładu omawianego w rozdziale znajdziesz pod adresem: <ftp://ftp.helion.pl/przyklady/expres.zip>.

Storify (<http://storify.com/>) działa w Node.js (<https://nodejs.org/>) i Express.js (<http://expressjs.com/>). Dlatego też wymienione technologie można wykorzystać do utworzenia aplikacji pokazującej, jak budować rozwiązania opierające się na żądaniach HTTP i API firm trzecich.

Zaczynamy pracę nad galerią Instagram

Galeria Instagram będzie pobierać obiekt, a następnie wyświetlać jego tytuł, opis i elementy/obrazy, podobnie jak pokazałem na rysunku 19.1.

■ **Uwaga** Jeżeli zastanawiasz się, co oznacza słowo Kazan (Kazań), wyjaśniam, że to ponadtysiącletnia stolica Republiki Tatarstanu.



Rysunek 19.1. Galeria Instagram

Struktura plików aplikacji przedstawia się następująco:

- index.js
- package.json
- views/index.html
- css/bootstrap-responsive.min.css
- css/flatly-bootstrap.min.css

Pliki CSS pochodzą z biblioteki Bootstrap (<http://getbootstrap.com/>) i motywu Flatly (<http://bootswatch.com/flatly/>). Z kolei *index.js* to nasz główny plik Node.js zawierający większość logiki, natomiast *index.html* to szablon Handlebars. Aplikacja używa zwykłych arkuszy stylów CSS zdefiniowanych w dwóch plikach znajdujących się w katalogu *css*.

Poniżej wymieniałem zależności aplikacji:

- express w wersji 4.8.1 — framework Express.js;
- superagent w wersji 0.18.2 — do wykonywania żądań HTTP(S);
- consolidate w wersji 0.10.0 — w celu użycia silnika szablonów Handlebars w Express.js;
- handlebars w wersji 2.0.0-beta.1 — w celu użycia silnika szablonów Handlebars.

Zawartość pliku *package.json* przedstawia się następująco:

```
{
  "name": "sfy-gallery",
  "version": "0.2.0",
  "description": "Galeria Instagram: oparty na Node.js przykład użycia API Storify",
  "main": "index.js",
```

```
"scripts": {
  "test": "echo \"Błąd: nie podano testu.\" && exit 1"
},
"dependencies": {
  "consolidate": "0.10.0",
  "express": "4.8.1",
  "handlebars": "2.0.0-beta.1",
  "superagent": "0.18.2"
},
"repository": "https://github.com/storify/sfy-gallery",
"author": "Azat Mardan",
"license": "BSD"
}
```

Aby zainstalować niezbędne moduły, wydaj poniższe polecenie:

```
$ npm install
```

Teraz utwórz plik *index.js*. Na początku wymienionego pliku powinny znaleźć się następujące zależności:

```
var express = require('express');
var superagent = require('superagent');
var consolidate = require('consolidate');

var app = express();
```

Kolejnym krokiem jest konfiguracja silnika szablonów:

```
app.engine('html', consolidate.handlebars);
app.set('view engine', 'html');
app.set('views', __dirname + '/views');
```

Dalej przechodzimy do przygotowania katalogu statycznego wraz z oprogramowaniem pośredniczącym:

```
app.use(express.static(__dirname + '/public'));
```

Jeżeli chcesz wykorzystać inną galerię, możesz to zrobić. Potrzebujesz jedynie nazwy użytkownika autora oraz tzw. *story slug*. W przypadku mojej galerii o Kazaniu (stolicy Tatarstanu) podaj następujące dane:

```
var user = 'azat_co';
var story_slug = 'kazan';
```

Następnie umieść swoje wartości: klucz API Storify, nazwę użytkownika i token, jeśli go masz. W chwili pisania książki API Storify pozostało publiczne, co oznacza *brak konieczności przeprowadzania uwierzytelniania* (nie trzeba używać klucza). Jeżeli w przyszłości sytuacja ulegnie zmianie, żądanie klucza API można będzie złożyć na stronie: <http://dev.storify.com/request>. Zawsze też możesz zajrzeć do oficjalnej dokumentacji dostępnej na stronie: <http://dev.storify.com/api/summary>.

```
var api_key = "";
var username = "";
var _token = "";
```

Kolejnym krokiem jest zdefiniowanie trasy głównej (/):

```
app.get('/', function(req, res){
```

Elementy z API Storify pobieramy w wywołaniu zwrotnym trasy za pomocą metody `superagent.get()`:

```
superagent.get("http://api.storify.com/v1/stories/"
+ user + "/" + story_slug)
```

Punktem końcowym API Storify jest "http://api.storify.com/v1/stories/" + user + "/" + story_slug, czyli w omawianym przykładzie to: <https://api.storify.com/v1/stories/azat/kazan>. Jedną z zalet metody superagent jest możliwość łączenia metod. Na przykład metoda query() powoduje wysłanie danych w ciągu tekstowym zapytania:

```
.query({api_key: api_key,
        username: username,
        _token: _token})
```

Metoda set() pozwala na określenie nagłówków żądania:

```
.set({Accept: 'application/json'})
```

Z kolei metoda end() pobiera wywołanie zwrótne przeznaczone do wykonania po otrzymaniu odpowiedzi:

```
.end(function(e, storifyResponse){
  if (e) return next(e);
```

Aby wygenerować szablon z obiektem wskazywanym przez właściwość content odpowiedzi HTTP, możemy użyć poniższego fragmentu kodu:

```
    return res.render('index', storifyResponse.body.content);
  })
})
```

```
app.listen(3001);
```

API Storify zwraca dane w formacie JSON. Informacje o używanym formacie znajdziesz na stronie: https://api.storify.com/v1/stories/azat_co/kazan (przyjmując założenie, że API nadal będzie publiczne, jak w chwili pisania książki). Zwięźłą (tzn. niepokazującą wszystkich zagnieżdżonych obiektów) postać danych JSON pokazałem na rysunku 19.2.

```
{
  - content: {
    sid: "516d9496b41520c44701a7fd",
    title: "Kazan",
    slug: "kazan",
    status: "published",
    template: null,
    version: 2,
    permalink: "http://storify.com/azat_co/kazan",
    shortlink: "http://sfy.co/j1l1",
    description: "Kazan is a 1000-year old city and a capital of Tatarstan (Tatar Republic).
http://en.wikipedia.org/wiki/Kazan",
    thumbnail: "http://distilleryimage2.s3.amazonaws.com/6e863d009b9f1e2aea022000a9d0ee7_7.jpg",
    + date: [-],
    private: false,
    not_indexed: false,
    is_spam: false,
    topics: [ ],
    siteposts: [ ],
    + meta: [-],
    + stats: [-],
    modified: false,
    deleted: false,
    canEdit: false,
    + author: [-],
    comments: [ ],
    page: 1,
    per_page: 20,
    totalElements: 45,
    + elements: [-]
  },
  code: 200
}
```

Rysunek 19.2. Przykład zwięzłych danych wyjściowych dostarczanych przez API Storify dla encji albumu

Wyświetlanie galerii

Skoro przygotowaliśmy aplikację pobierającą dane ze Storify i wywołującą szablon `index` w celu ich wyświetlenia, warto spojrzeć na szablon Handlebars, który znajduje się w pliku `views/index.html`:

```
<!DOCTYPE html lang="en">
<html>
  <head>
    <link type="text/css"
      href="css/flatly-bootstrap.min.css"
      rel="stylesheet" />
    <link type="text/css"
      href="css/bootstrap-responsive.min.css"
      rel="stylesheet"/>
  </head>

  <body class="container">
    <div class="row">

      Teraz użyjemy {{title}} do wyświetlenia tytułu albumu Storify oraz {{author.name}}
do wyświetlenia autora:

      <h1>{{title}}<small> by {{author.name}}</small></h1>
      <p>{{description}}</p>
    </div>
    <div class="row">
      <ul class="thumbnails">
```

Kolejnym krokiem jest wykorzystanie wbudowanej w Handlebars konstrukcji do przeprowadzenia iteracji przez elementy tablicy. W trakcie każdej iteracji następuje wygenerowanie nowego znacznika ``:

```
      {{#each elements}}
      <li class="span3">
        <a class="thumbnail" href="{{permalink}}"
          target="_blank">
          
        </a>
      </li>
      {{/each}}
    </ul>
  </div>
</body>
</html>
```

Po uruchomieniu aplikacji za pomocą polecenia `node .` i przejściu pod adres: `http://localhost:3000` zobaczysz wyświetlone zdjęcia. Działanie aplikacji jest następujące: po przejściu na podaną stronę serwer lokalny wykonuje żądania do Storify i pobiera z galerii Instagram łączy z zdjęć.

Podsumowanie

Framework Express.js i moduł superagent pozwalają programistom na pobieranie danych dostarczanych przez usługi opracowane przez firmy trzecie, takie jak: Storify, Twitter i Facebook, za pomocą jedynie kilku wierszy kodu. Umożliwiają też zarządzanie tymi danymi. Przykład przedstawiony w rozdziale jest prosty, ponieważ nie wykorzystuje bazy danych. W kolejnym rozdziale przystąpimy do budowy aplikacji Todo, która zostanie oparta na bazie danych MongoDB.

-
- **Uwaga** W większości przypadków dostawcy usług (np.: Google, Facebook i Twitter) wymagają uwierzytelnienia (w chwili pisania książki nie było takiego wymogu dla API Storify). Aby wykonywać żądania: OAuth 1.0, OAuth 2.0 i OAuth Echo, rozważ użycie modułów: `oauth` (<https://www.npmjs.com/package/oauth>; GitHub: <https://github.com/ciaranj/node-oauth>), `everyauth` (<https://www.npmjs.com/package/everyauth>; GitHub: <https://github.com/bnoguchi/everyauth>) i/lub `passport` (<http://passportjs.org/>; GitHub: <https://github.com/jaredhanson/passport>).
-

Skorowidz

A

- abstrakcja, 165
 - kodu, 101
- adaptery Hogan.js, 97
- adres
 - e-mail, 285
 - URL, 104, 262
- API, 49
 - REST, 166, 253
 - REST JSON, 268
 - Storify, 227
- aplikacja
 - Backbone.js, 268, 274
 - Express.js, 276
 - HackHall, 267
 - członkostwo w społeczności, 295
 - modele Mongoose, 301
 - plik package.json, 275
 - strona postów, 273
 - strona użytkowników, 290, 292
 - struktura, 274
 - testy Mocha, 307
 - trasy, 280
 - Todo, 231
 - Jade, 246
 - konfiguracja, 235
 - Less, 250
 - plik app.js, 237
 - trasy, 242
- aplikacje
 - front-endu, 274
 - oparte na domenie, 211

- architektura MVC, 45
- arkusze stylów
 - CSS, 191
 - stylów Less, 250
- atak
 - typu brute force, 271
 - typu CSRF, 78, 195
- atrybut
 - request.accepted, 126
 - request.acceptedCharsets, 127
 - request.acceptedLanguages, 126
 - request.fresh, 126
 - request.host, 126
 - request.ip, 126
 - request.ips, 126
 - request.originalUrl, 126
 - request.path, 126
 - request.protocol, 126
 - request.secure, 126
 - request.stale, 126
 - request.subdomains, 126
 - request.xhr, 126

B

- baza danych, 171
 - Redis, 181
- bezpieczeństwo, 195
- biblioteka
 - bcryptjs, 275
 - Bootstrap, 226
 - Consolidate.js, 98
 - Less, 192

biblioteka

- mongoose, 275
- Mongoskin, 253
- passport, 275
- Sass, 193
- sendgrid, 275
- Socket.IO, 203
- Stylus, 191

biblioteki wewnętrzne, 274

Body, 331

C

certyfikat SSL, 160

Consolidate.js, 98

CORS, cross-origin resource sharing, 56

CRUD, create, remove, update, delete, 253

CSRF, cross-site request forgery, 78

CSS, 191, 335

D

dane

- JSON, 336
- wejściowe, 208
- wyjściowe, 40

definiowanie trasy, 28

DerbyJS, 217

dołączanie biblioteki, 205

domeny, 209

dostęp do aplikacji, 283

działanie Express.js, 26

E

e-mail, 276, 289, 292

Express.js 4, 319

ExpressWorks, 333

F

Foreman, 268

format JSON, 139, 255

framework

- Compound, 222
- DerbyJS, 217
- Geddy, 222
- Hapi, 222
- LoopBack, 219

Sails.js, 215

Total.js, 222

frameworki Node.js, 315

funkcja

- app.get(), 36
- clientErrorHandler(), 277
- cookieParser(), 77
- del(), 298
- express.static(), 79
- findByIdAndRemove(), 289
- findOrCreate(), 290
- json(), 118
- next(), 109
- render(), 92
- require(), 169
- send(), 206
- updateById(), 262
- urlencoded(), 76

G

galeria Instagram, 225, 226

generator, 328

generowanie, 91

szkieletu aplikacji, 40

wartości hash, 285

GitHub, 267

H

hash, 285

hasła, 173

cookie, 269

Heroku, 269

I

implementacja

OAuth, 279

serwera, 260

informacje o użytkowniku, 270, 283

instalacja, 29, 328

Express.js Generator, 32

ExpressWorks, 333

J

Jade, 246, 330, 335

K

klasa Router, 111, 323
 klaster, 187, 189
 klient front-endu, 268, 277
 klucze, 173
 kody stanów HTTP, 153
 kompilacja szablonów, 91
 komunikaty błędów, 201
 konfiguracja, 51

- Heroku, 269
- ustawień aplikacji, 28
- zamiast konwencji, 25
- try-catch, 210

 konwencja zamiast konfiguracji, 25

L

Less, 192, 250
 LoopBack, 219

M

MDN, Mozilla Developer Network, 57
 menedżer npm, 66
 metoda

- _express(), 94
- add(), 288
- app.all(), 111
- app.configure(), 63, 321
- app.disable(), 52
- app.enable(), 52
- app.engine(), 93
- app.get(), 52
- app.listen(), 157, 159, 172
- app.param(), 105
- app.path(), 157
- app.proto.create(), 219
- app.render(), 155
- app.route(), 323
- app.router(), 321
- app.set(), 52
- app.use(), 322
- compare(), 285
- compression(), 72
- contain(), 257
- exports.angelList(), 281
- findByIdAndRemove(), 286
- findOrAddUser(), 290

- findProfileById(), 305
- findStories(), 168
- getUser(), 288
- req.accepted(), 322
- request.accepts(), 126
- request.acceptsCharset(), 127
- request.acceptsLanguage(), 126
- request.get(), 125
- request.header(), 125
- request.is(), 126
- request.param(), 221
- res.location(), 323
- res.on(), 322
- res.json(), 57
- response.attachment(), 145
- response.clearCookie(), 145
- response.cookie(), 145
- response.download(), 145
- response.format(), 145
- response.get(), 145
- response.json(), 141
- response.jsonp(), 142
- response.links(), 145
- response.location(), 145
- response.redirect(), 144
- response.render(), 132, 133
- response.send(), 138, 139, 140
- response.sendFile(), 145
- response.set(), 135
- response.status(), 137
- response.type(), 145
- router.route(), 111
- socket.emit(), 205
- update(), 289

metody

- HTTP, 154, 254, 329
- idempotentne, 254
- nullipotentne, 255
- odpowiedzi, 144
- przestarzałe, 321

middleware, 67

modele Mongoose, 301

moduł, 45

- bcryptjs, 271, 285
- body-parser, 75
- cluster, 187
- connect-busboy, 87
- connect-timeout, 80
- cookie-parser, 71

moduł

- derby, 218
- domain, 209
- Mocha, 259
- oauth, 279
- okay, 214
- serve-static, 79
- zlib, 72

monitorowanie plików, 47

montowanie, 68, 157

N

nagłówek

- Content-Type, 136
- CORS, 56
- X-Powered-By, 61

nagłówki zabezpieczeń, 198

narzędzia monitorujące pliki, 47

narzędzie

- etag, 60
- ExpressWorks, 333
- Foreman, 268, 272, 309
- forever, 47
- node-dev, 47
- nodemon, 47
- supervisor, 47
- up, 47

O

obiekt

- app.locals, 155
- JSON, 286
- request, 117
- request.body, 121
- request.cookies, 123
- request.params, 120
- request.query, 118
- request.route, 122
- request.signedCookies, 124
- response, 131
- response.locals, 134

obsługa

- błędów, 44, 151, 277
- żądań, 36, 113
- żądań PUT, 262

odpowiedź, 330

opcja, *Patrz* ustawienie

operacje CRUD, 253

operator `||`, 277

oprogramowanie pośredniczące, 28, 67, 108, 165, 319

compression, 72

Connect, 331

cookie-parser, 77, 123

csrf, 79

errorhandler, 82

express.static(), 80

express-session, 77

inne, 88

method-override, 82

morgan, 74

najważniejsze, 71

popularne, 331

response-time, 83

serve-favicon, 84

serve-index, 85

static, 335

urlencoded, 76

vhost, 87

P

pakiet npm, 71

parametry, 101

pierwsza aplikacja, 35

plik

.env, 268, 274, 309

app.js, 41, 83, 237

application.js, 292

auth.js, 280

cluster.js, 190

favicon.ico, 88

hello.js, 38

index.jade, 133

index.js, 168, 280

lorem-ipsam.html, 96

main.js, 284

Makefile, 274, 308, 309

package.json, 32, 118, 274

posts.js, 295

profile, 274

readme.md, 274

seed-script.js, 269

server.js, 274

stream-express-req.js, 178

stream-http-req.js, 177

users.js, 287, 292

- pliki
 - *.styl, 192
 - CSS, 226
 - JSON, 173
- polecenia generatora, 39
- polecenie
 - node app, 81
 - node cluster, 189
 - npm init, 30
- połączenie z bazą danych, 27, 271
- potokowanie strumienia odpowiedzi, 176
- powiadomienia e-mail, 275
- problemy, 163
- procedury obsługi żądania, 113
- projekty typu open source, 267
- przetwarzanie uprawnień, 197

R

- Redis, 181
- renderowanie, 91
- repozytorium GitHub, 267
- REST, 110
- routing, 108
- rozwiązywanie problemów, 163

S

- Sails.js, 215
- Sass, 193
- SendGrid, 268, 269
- serwer
 - MongoDB, 272
 - REST, 253
- silnik szablonów, 73, 91, 99
 - Blade, 98
 - EJS, 97
 - Haml.js, 97
 - Haml-Coffee, 98
 - Handlebars.js, 97
 - Jade, 97
 - Swig, 98
 - Whiskers, 98
- skrót nagłówek żądania, 330
- skrypt seed-script.js, 269
- Socket.IO, 203
- Static, 331
- sterownik MongoDB, 171

- struktura
 - aplikacji, 41, 274
 - CRUD, 254
- strumienie, 148, 175
- strumieniowanie obrazu, 149
- style CSS, 335
- Stylus, 191, 330
- sygnatury procedury obsługi, 330
- szablon
 - index.jade, 246
 - layout.jade, 246
 - tasks.jade, 246
 - tasks_completed.jade, 246
- zwyfrowanie haseł, 275

Ś

- ścieżka dostępu, 69
- środowiska, 62

T

- TDD, test-driven development, 253
- testy, 253, 255
 - Mocha, 307
 - TDD, 307
- token, 281, 290
- trasy, 166, 242, 274, 279, 329
- tworzenie egzemplarza, 27
- typy danych Mongoose, 301

U

- ukośniki na końcu, 111
- uruchamianie
 - aplikacji, 28, 155, 207
 - HackHall, 268
 - serwera MongoDB, 272
 - usług sieciowych, 197
- usprawnienie aplikacji, 38
- ustawienia niestandardowe, 53
- ustawienie
 - case sensitive routing, 58
 - jsonp callback name, 56
 - query parser, 61
 - strict routing, 59
 - subdomain offset, 62
 - view cache, 54

ustawienie
 view engine, 55
 views, 55
 x-powered-by, 60
 trust proxy, 56

uwierzytelnianie, 184
 OAuth, 279

użycie
 baz danych, 171
 biblioteki consolidate, 98
 domeny, 212
 klastrów, 187
 module.exports, 170
 silników szablonów, 92
 Socket.IO, 203

W

wersja generatora, 40
 weryfikacja danych wejściowych, 199
 wielowątkowość, 187
 właściwość
 app.mountpath, 156
 req.db.Post, 296
 wtyczka findOrCreate, 305
 wyrażenie regularne, 36

wyświetlanie
 galerii, 229
 komunikatów, 37
 wzorce uwierzytelniania, 184

Z

zadania, 334
 zależności, 259
 firm trzecich, 27
 zastosowanie oprogramowania pośredniczącego, 68
 zdarzenie receive, 205
 zerowanie zadań, 334
 zmienna
 __dirname, 169
 env, 54
 zmienna środowiskowa, 175
 process.env.PORT, 277

Ż

żądanie, 329
 DELETE, 255
 GET, 255
 POST, 255
 PUT, 255, 262

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

- 
1. ZAREJESTRUJ SIĘ
 2. PREZENTUJ KSIĄZKI
 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**

Express.js

Tworzenie aplikacji sieciowych w Node.js

Frameworku Express.js używa wiele znanych firm, takich jak MySpace i Storify, które dostrzegły ogromne korzyści płynące z wykorzystywania tej technologii, a także doceniły jej stabilność i bezpieczeństwo. Przed Express.js i Node.js otwiera się świetlana przyszłość. Oznacza to, że jako ekspert w zakresie Node.js, biegły posługujący się Express.js, staniesz się poszukiwanym specjalistą!

Ta książka jest kompleksowym podręcznikiem, który przedstawi Ci sposób działania Express.js w praktyce i przeprowadzi Cię przez poszczególne etapy budowy aplikacji. Autor jasno i precyzyjnie wyjaśnia wszystkie koncepcje, których zrozumienie jest niezbędne do programowania w Express.js. Znajdziesz tu omówienie zagadnień związanych m.in. z oprogramowaniem pośredniczącym, tworzeniem szkieletu aplikacji, generowaniem szablonów, przetwarzaniem danych, żądania i cookies, zarządzaniem uwierzytelnianiem i sesjami, obsługą błędów i przygotowaniu aplikacji do wdrożenia w środowisku produkcyjnym. Na pewno docenisz też liczne przykłady kodu źródłowego.

Jeśli — jako inżynier oprogramowania lub programista sieciowy — poszukujesz sposobu, aby bez wertowania setek stron dokumentacji programować z wykorzystaniem frameworku Express.js, a przy tym poszerzyć swoje kompetencje, ta książka jest właśnie dla Ciebie.

Dzięki tej książce:

- rozpoczniesz pracę z oprogramowaniem pośredniczącym
- będziesz sprawnie korzystać z silników szablonów
- dowiesz się, jak zapewnić aplikacji bezpieczeństwo
- nauczysz się tworzyć aplikacje oparte na domenie
- przeanalizujesz przykłady działających aplikacji, takich jak Instagram, HackHall i inne

Azat Mardan — od kilkunastu lat programuje aplikacje mobilne i sieciowe. Pracował nad wieloma aplikacjami o kluczowym znaczeniu, wykorzystywanymi przez agencje rządowe USA. Jest twórcą kilku projektów open source dla Node.js (m.in. ExpressWorks, mongoui, HackHall.com i NodeFramework.com), a także współtwórcą express, oauth, jade-browser i innych modułów npm. Tworzy programistyczne kursy online, publikuje specjalistyczne artykuły na blogu, a ponadto jest autorem książek poświęconych JavaScriptowi i Node.js.

Helion

38252 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>
 Katalogi najczęściej czytane:
 ● <http://helion.pl/bestsellery>
 Zamów informacje o nowościach:
 ● <http://helion.pl/novosci>

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

Apress

ISBN 978-83-283-1664-5



9 788328 316645

cena: 59,00 zł

ślęgnij po WIĘCEJ



KOD KORZYSCI