



Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 10

Wykład dla kierunku: **Informatyka Stosowana II rok**

Rok akademicki: **2015/2016 - semestr zimowy**

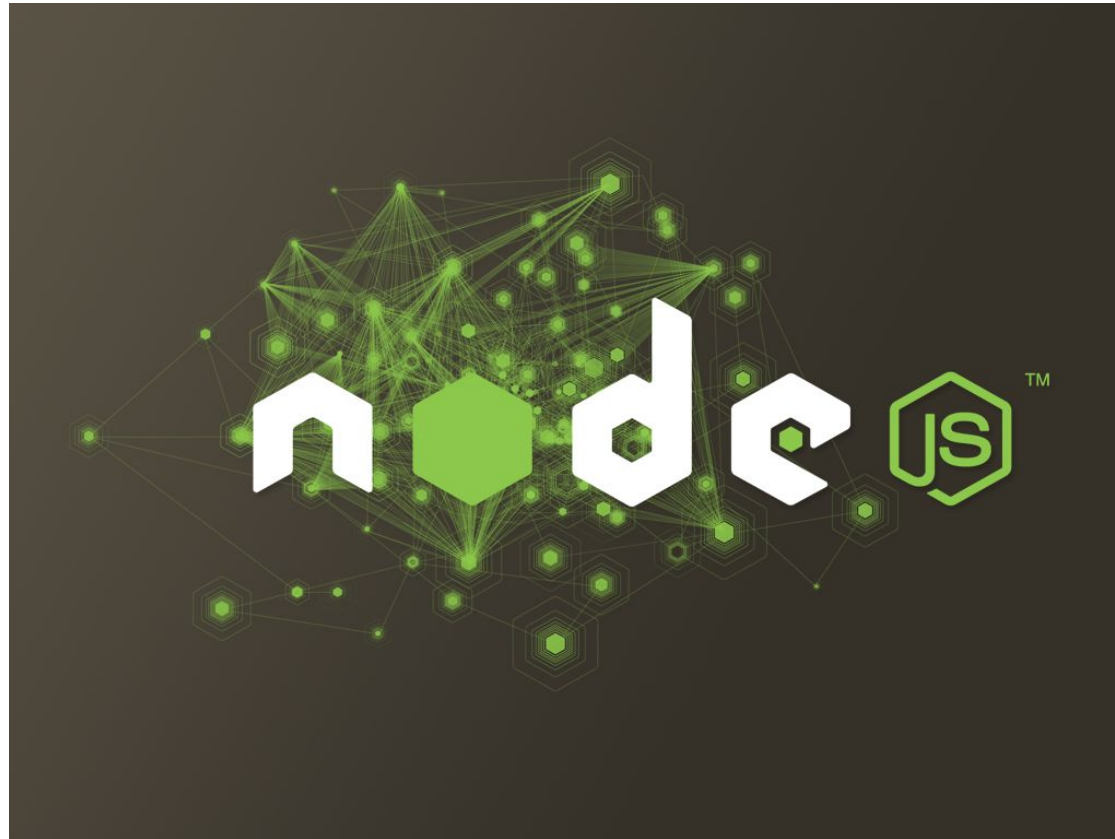
Przypomnienie z poprzedniego wykładu

Wprowadzenie do środowiska Node.js

Repozytorium NPM

Wzorzec projektowy MVC (Model - Widok - Kontroler)

Node.js



<http://nodejs.org/>

Obsługa żądań

MODEL ZDARZENIOWY

KLIENCI

Żądanie HTTP

Żądanie HTTP

Żądanie HTTP

■ ■ ■

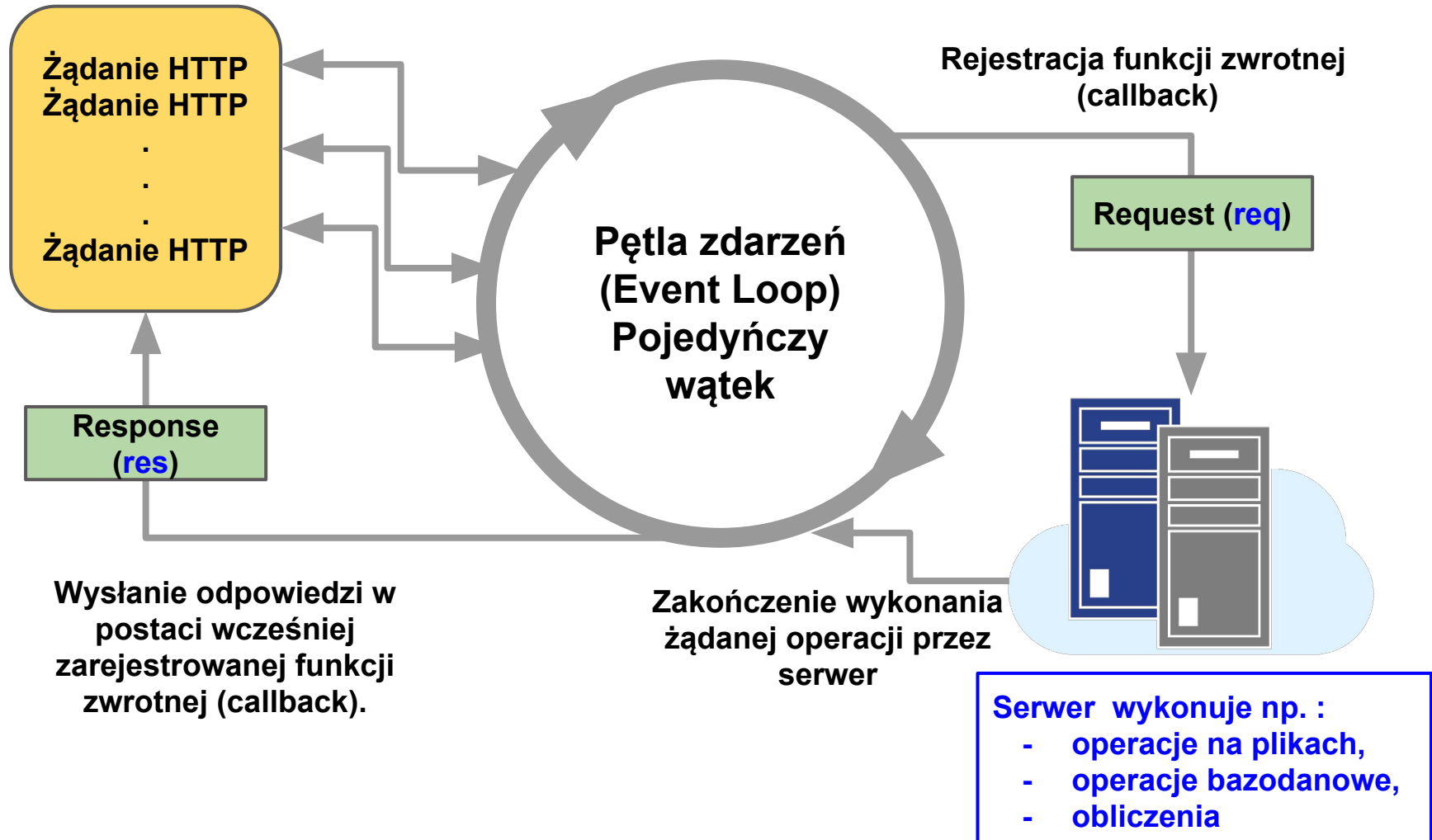
Żądanie HTTP

SERWER

Wątek

W modelu zdarzeniowym Node.js wykorzystuje tylko jeden wątek do obsługi wielu żądań, oraz “pętlę zdarzeń” co powoduje że aplikacja taka jest bardzo wydajna i skalowalna. W praktyce przy żądaniach które nie wymagają złożonych operacji obliczeniowych można obsłużyć nawet do **1 miliona żądań** jednocześnie.

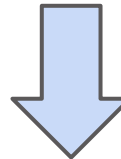
Pętla zdarzeń (Event loop)



Przykład prostego kodu (z strony nodejs.org)

```
var http = require('http');  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('<html><head></head><body>Test</body></html>');  
}).listen(1337, '127.0.0.1');  
console.log('Server running at http://127.0.0.1:1337/');
```

We wcześniejszym przykładzie serwer zwracał w odpowiedzi zwykły tekst, natomiast teraz zwraca dokument hipertekstowy ze statyczną stroną internetową.



Stworzyliśmy serwer obsługujący żądania HTTP !!!

Instalacja dla Linux

Przechodzimy do kartoteki bin gdzie wykonujemy polecenie:

```
MacBook-Pro-Marcin-2:bin marcin$ ./node -v  
v0.10.33
```

Program odpowiada
numerem wersji

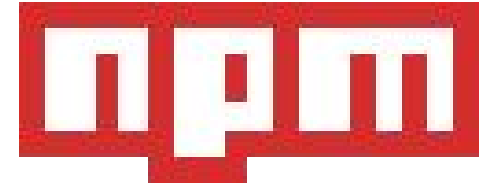
Zaleca się zainstalowanie globalnej wersji node.js dostępnej dla wszystkich użytkowników oraz możliwie do uruchomienie a każdego miejsca w strukturze kartotek.

Uruchomienie skryptu polega na wywołaniu programu “node” z parametrem określającym nazwę skryptu (gdy node.js jest zainstalowany lokalnie):

```
> ./node  serwer.js
```

Repozytorium NPM

Integralną częścią środowiska NODE.JS, jest bogate repozytorium modułów (bibliotek), dzięki któremu mamy dostęp do wielu gotowych funkcji.



<http://npmjs.org/>

Na stronie internetowej projektu można przeglądać i wyszukiwać pakiety, jednak ich instalacja odbywa się w poziomym wierszu poleceń:

```
| > ./npm install nazwa-pakietu |
```

Pakiety instalują się w kartotece `lib/node_modules`.

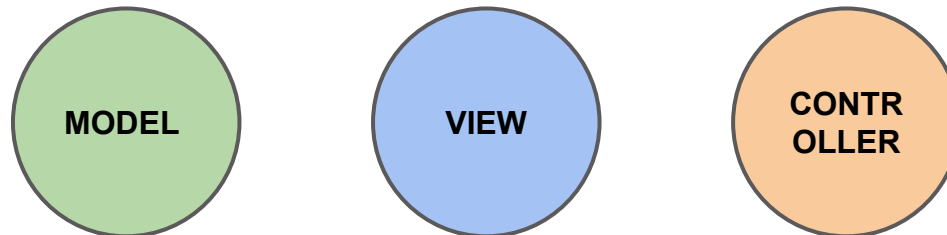
Do globalnej instalacji pakietów NPM należy posiadać prawa administratora i wydać polecenie:

```
| > ./npm -g install nazwa-pakietu |
```


Model-View-Controller

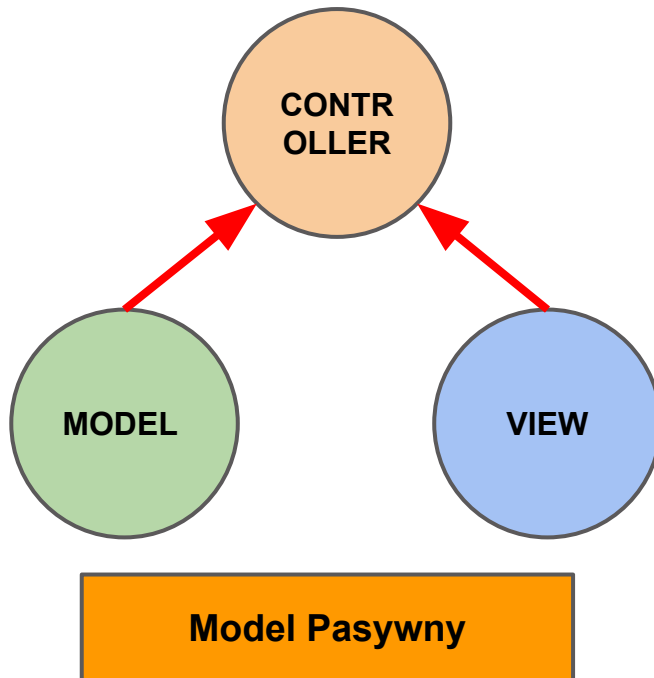
Model-View-Controller (MVC) [*Model-Widok-Kontroler*] - jest to wzorzec projektowy (podejście które jest bazą w oparciu o którą tworzymy aplikację), dzielący projektowaną aplikację na trzy warstwy:

- Model (dane / logika)
- Widok (prezentacja danych)
- Kontroler (interakcja z użytkownikiem + sterowanie aplikacją)

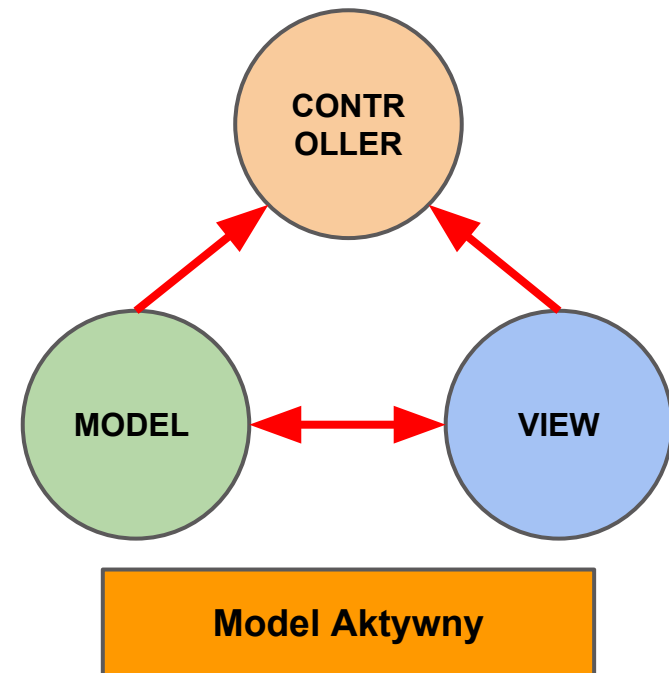


Można go zaimplementować bez użycia bibliotek czy specjalistycznych platform programistycznych, stosując jasne reguły podziału na konkretne komponenty w kodzie źródłowym. W ten sposób każdy komponent aplikacji można niezależnie od siebie rozwijać, implementować i testować.

Model-View-Controller



W modelu “pasywny” nie ma wymiany danych pomiędzy modelem a widokiem z pominięciem kontrolera, wszystkie akcje są wywoływane i sterowane przez kontroler.

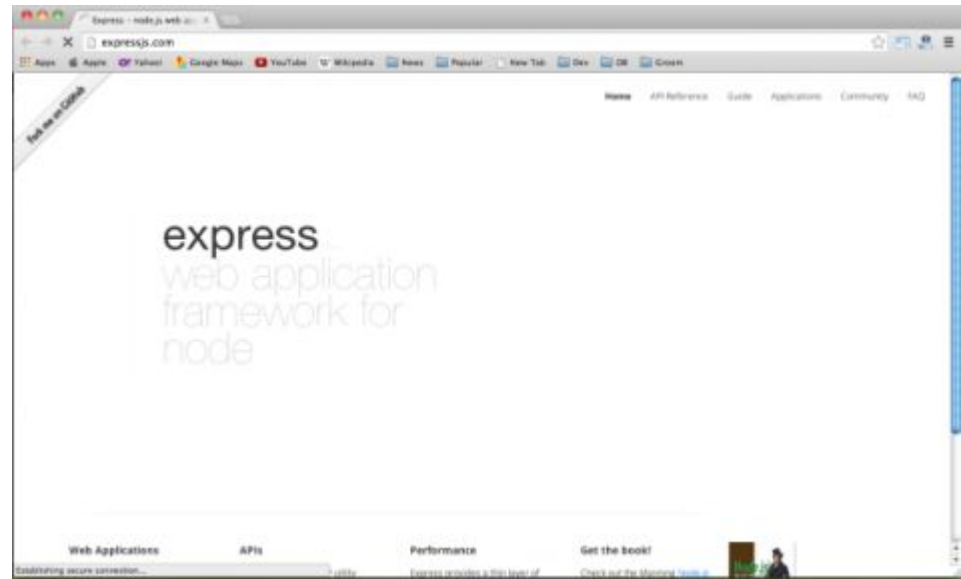


W model aktywnym model może bezpośrednio przekazywać dane do widoku z pominięciem kontrolera.

Node.js + MVC

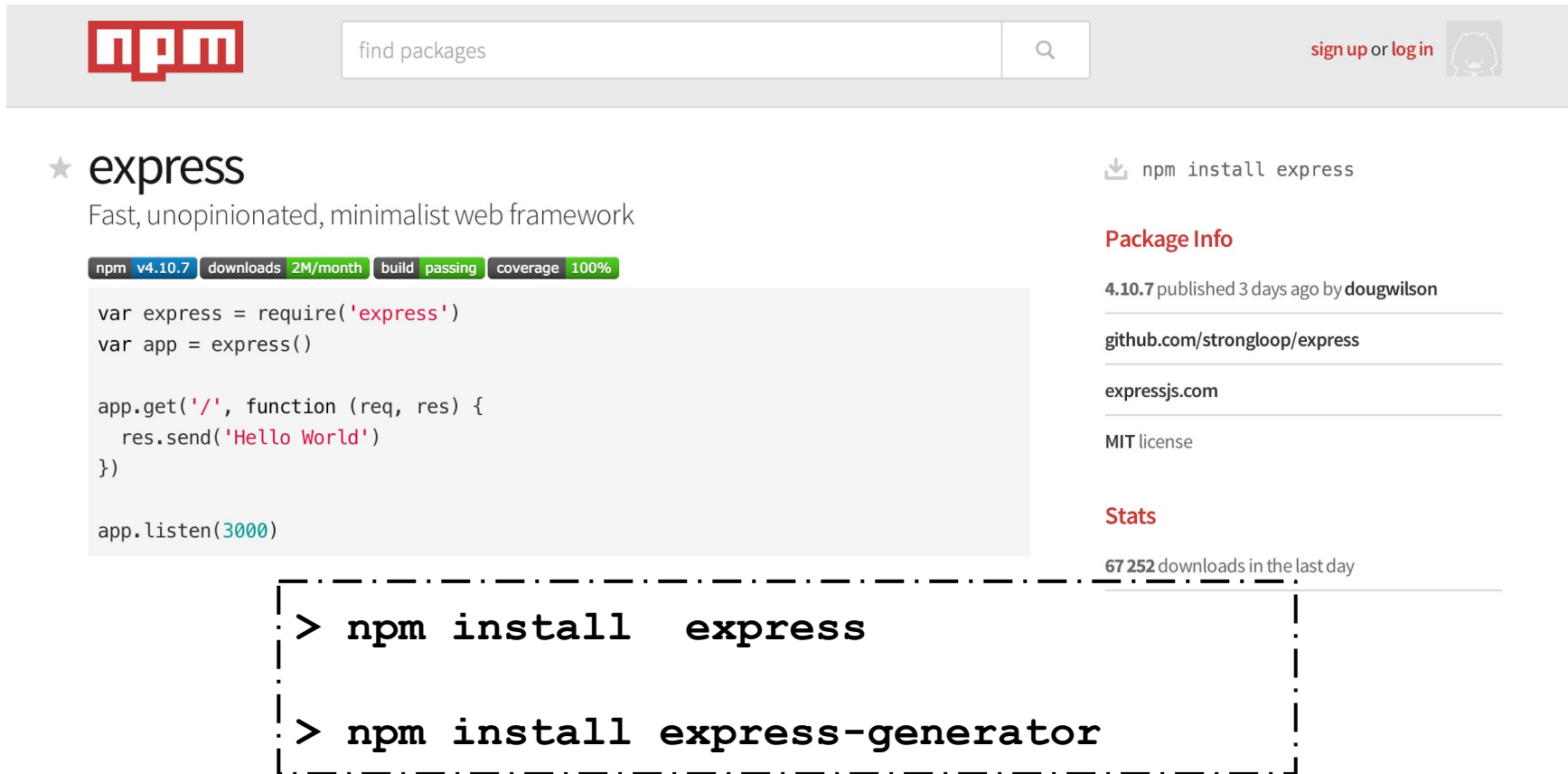
Istnieje kilka środowisk ułatwiających tworzenie aplikacji według wzorca projektowego MVC w systemie NODE.JS:

<http://expressjs.com/>



Express JS - instalacja

Instalacja Express.js odbywa się za pomocą menadżera pakietów NPM.



The screenshot shows the npm website interface. At the top is the npm logo and a search bar with the text "find packages". To the right of the search bar are links for "sign up or log in" and a user profile icon. Below the search bar, the "express" package is featured with a star icon. The package description reads "Fast, unopinionated, minimalist web framework". Below the description are several status badges: "npm v4.10.7", "downloads 2M/month", "build passing", and "coverage 100%". A code block displays a sample Express.js application setup. To the right of the code block, there is a section for "Package Info" which includes the version "4.10.7 published 3 days ago by dougwilson", the GitHub repository "github.com/strongloop/express", the website "expressjs.com", and the "MIT license". Below the package info is a "Stats" section showing "67 252 downloads in the last day". At the bottom of the screenshot, a dashed box contains two terminal commands: `> npm install express` and `> npm install express-generator`.

npm

find packages

sign up or log in

★ express

Fast, unopinionated, minimalist web framework

npm v4.10.7 downloads 2M/month build passing coverage 100%

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

npm install express

Package Info

4.10.7 published 3 days ago by dougwilson

github.com/strongloop/express

expressjs.com

MIT license

Stats

67 252 downloads in the last day

```
> npm install express
> npm install express-generator
```

Express JS - projekt “0”

Po zainstalowaniu środowiska express.js wraz z generatorem dostajemy nowe narzędzie służące do tworzenia aplikacji według wzorca projektowego MVC. Fizycznie jest to program wykonywalny który służy do generowania projektów “0” oraz obsługi całej aplikacji:

Express JS - projekt “0”

Po zainstalowaniu środowiska express.js wraz z generatorem dostajemy nowe narzędzie służące do tworzenia aplikacji według wzorca projektowego MVC. Fizycznie jest to program wykonywalny który służy do generowania projektów “0” oraz obsługi całej aplikacji:

- 1) W pierwszym kroku stworzymy nowy folder w którym będziemy przechowywać pliki projektu:

```
> mkdir test
```

Express JS - projekt “0”

Po zainstalowaniu środowiska express.js wraz z generatorem dostajemy nowe narzędzie służące do tworzenia aplikacji według wzorca projektowego MVC. Fizycznie jest to program wykonywalny który służy do generowania projektów “0” oraz obsługi całej aplikacji:

- 1) W pierwszym kroku tworzymy nowy folder w którym będziemy przechowywać pliki projektu:

```
> mkdir test
```

- 2) W drugim kroku generujemy projekt “0”

```
> express test
```

Express JS - projekt “0”

Po wykonaniu powyższej operacji pusty do tej pory katalog został wypełniony plikami stanowiący w pełni działającą aplikację według wzorca MVC:

```
MacBook-Pro-Marcin-2:~ marcin$ express test
```

```
create : test
create : test/package.json
create : test/app.js
create : test/public
create : test/public/images
create : test/public/stylesheets
create : test/public/stylesheets/style.css
create : test/routes
create : test/routes/index.js
create : test/routes/users.js
create : test/views
create : test/views/index.jade
create : test/views/layout.jade
create : test/views/error.jade
create : test/bin
create : test/bin/www
create : test/public/javascripts
```

```
install dependencies:
$ cd test && npm install
```

```
run the app:
$ DEBUG=test ./bin/www
```


Express JS - projekt “0”

Po wykonaniu powyższej operacji pusty do tej pory katalog został wypełniony plikami stanowiący w pełni działającą aplikację według wzorca MVC:

```
MacBook-Pro-Marcin-2:~ marcin$ express test
```

```
create : test
create : test/package.json
create : test/app.js
create : test/public
create : test/public/images
create : test/public/stylesheets
create : test/public/stylesheets/style.css
create : test/routes
create : test/routes/index.js
create : test/routes/users.js
create : test/views
create : test/views/index.jade
create : test/views/layout.jade
create : test/views/error.jade
create : test/bin
create : test/bin/www
create : test/public/javascripts
```

```
install dependencies:
$ cd test && npm install
```

```
run the app:
$ DEBUG=test ./bin/www
```

Lista wygenerowanych plików
w kartotece “test”

Express JS - projekt “0”

Po wykonaniu powyższej operacji pusty do tej pory katalog został wypełniony plikami stanowiący w pełni działającą aplikację według wzorca MVC:

```
MacBook-Pro-Marcin-2:~ marcin$ express test
```

```
create : test
create : test/package.json
create : test/app.js
create : test/public
create : test/public/images
create : test/public/stylesheets
create : test/public/stylesheets/style.css
create : test/routes
create : test/routes/index.js
create : test/routes/users.js
create : test/views
create : test/views/index.jade
create : test/views/layout.jade
create : test/views/error.jade
create : test/bin
create : test/bin/www
create : test/public/javascripts
```
















```
install dependencies:
$ cd test && npm install
```

```
run the app:
$ DEBUG=test ./bin/www
```

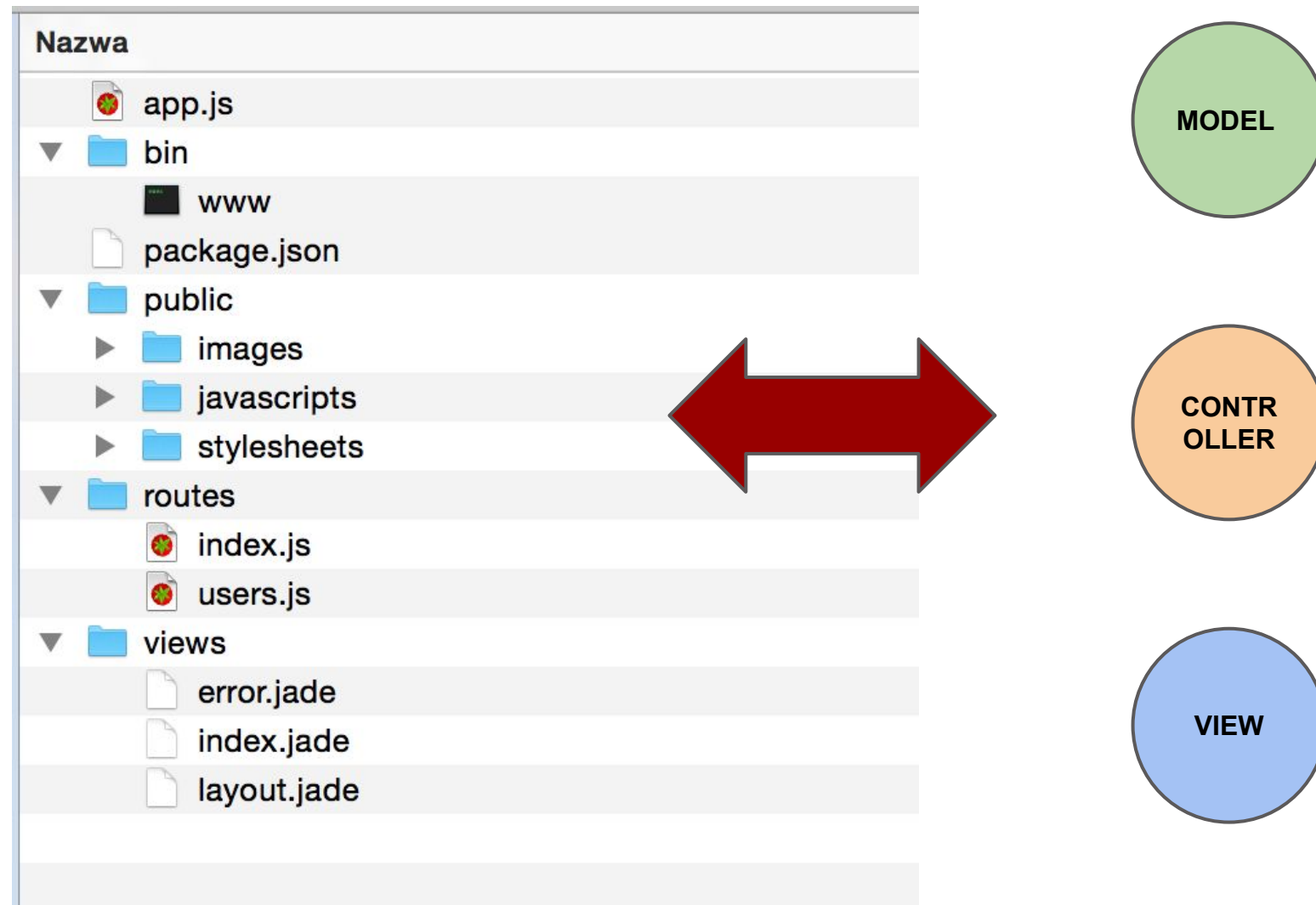
Lista wygenerowanych plików
w kartotece “test”

Instalacja standardowych
pakietów

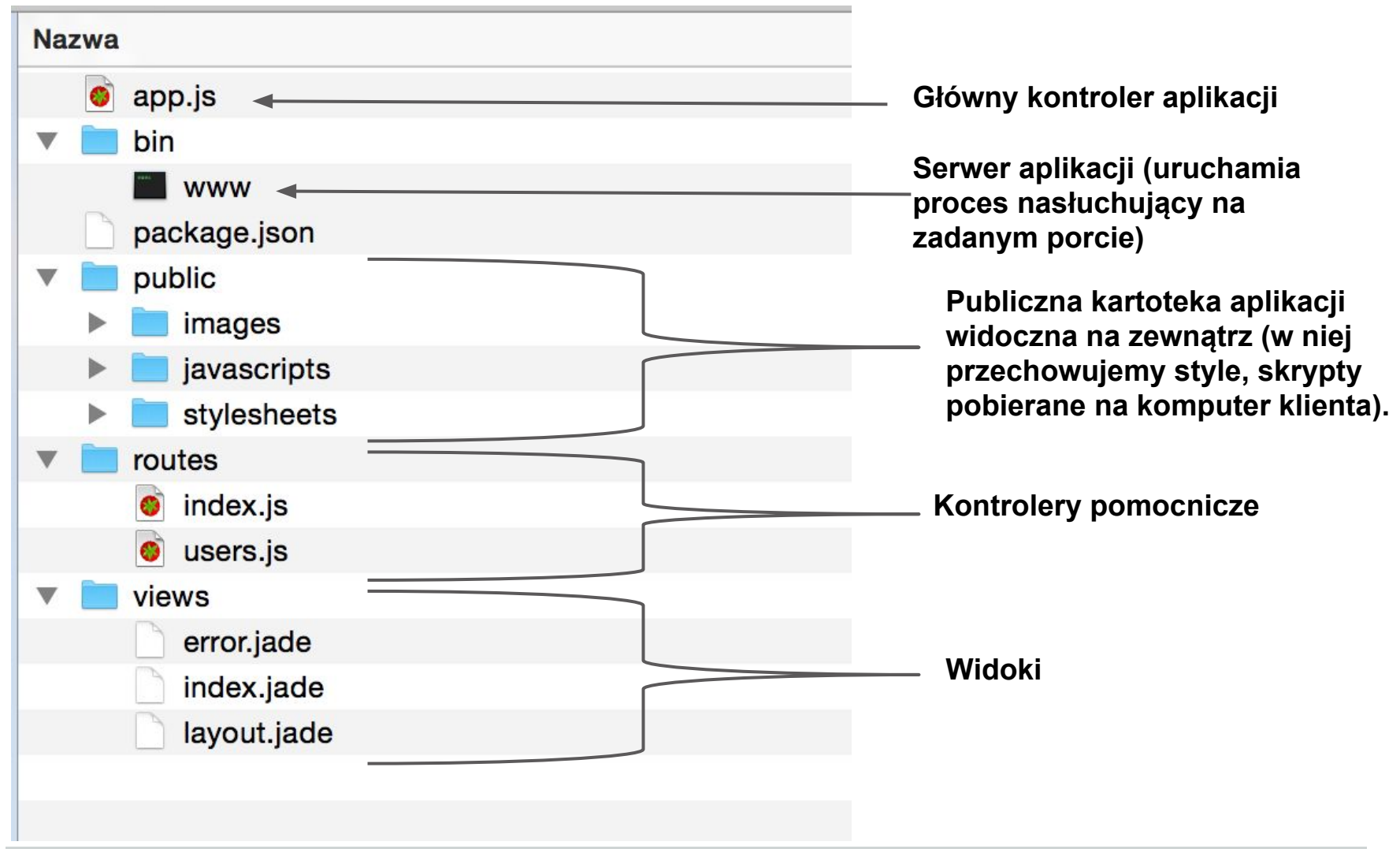
Express JS - struktura aplikacji

Nazwa	
	app.js
▼ 	bin
	www
	package.json
▼ 	public
▶ 	images
▶ 	javascripts
▶ 	stylesheets
▼ 	routes
	index.js
	users.js
▼ 	views
	error.jade
	index.jade
	layout.jade

Express JS - struktura aplikacji

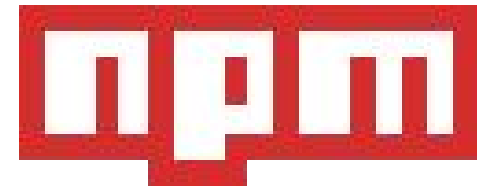


Express JS - struktura aplikacji



Repozytorium NPM

Integralną częścią środowiska NODE.JS, jest bogate repozytorium modułów (bibliotek), dzięki któremu mamy dostęp do wielu gotowych funkcji.



<http://npmjs.org/>

Standardowo w kartotece projektu powstaje plik package.json, który zawiera informacje o zainstalowanych pakietach w danym projekcie:

```
{
  "name": "hello",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.0.0",
    "express": "~4.2.0",
    "jade": "~1.3.0",
    "static-favicon": "~1.0.0"
  }
}
```

W projektach generowanych np. w Express plik ten powstaje automatycznie.

W innym przypadku należy taki plik wytworzyć samemu poleceniem:
> npm init

Aby przy instalacji dopisać informację o nowym pakiecie do tego pliku należy wykonać polecenie:
> npm install NAZWA_PAKIETU --save

Express JS - Uruchamianie

Po wygenerowaniu projektu “0” dostajemy w pełni działający wzorzec aplikacji który można uruchomić. Każdą aplikację przygotowaną za pomocą środowiska Express.js uruchamiamy w następujący sposób:

Express JS - Uruchamianie

Po wygenerowaniu projektu “0” dostajemy w pełni działający wzorzec aplikacji który można uruchomić. Każdą aplikację przygotowaną za pomocą środowiska Express.js uruchamiamy w następujący sposób:

```
~/test> npm start
```


Express JS - Uruchamianie

Po wygenerowaniu projektu “0” dostajemy w pełni działający wzorzec aplikacji który można uruchomić. Każdą aplikację przygotowaną za pomocą środowiska Express.js uruchamiamy w następujący sposób:

```
~/test> npm start
```

```
MacBook-Pro-Marcin-2:test marcin$ npm start
```

```
> test@0.0.0 start /Users/marcin/test
```

```
> node ./bin/www
```

Express JS - Uruchamianie

Po wygenerowaniu projektu “0” dostajemy w pełni działający wzorzec aplikacji który można uruchomić. Każdą aplikację przygotowaną za pomocą środowiska Express.js uruchamiamy w następujący sposób:

```
~/test> npm start
```

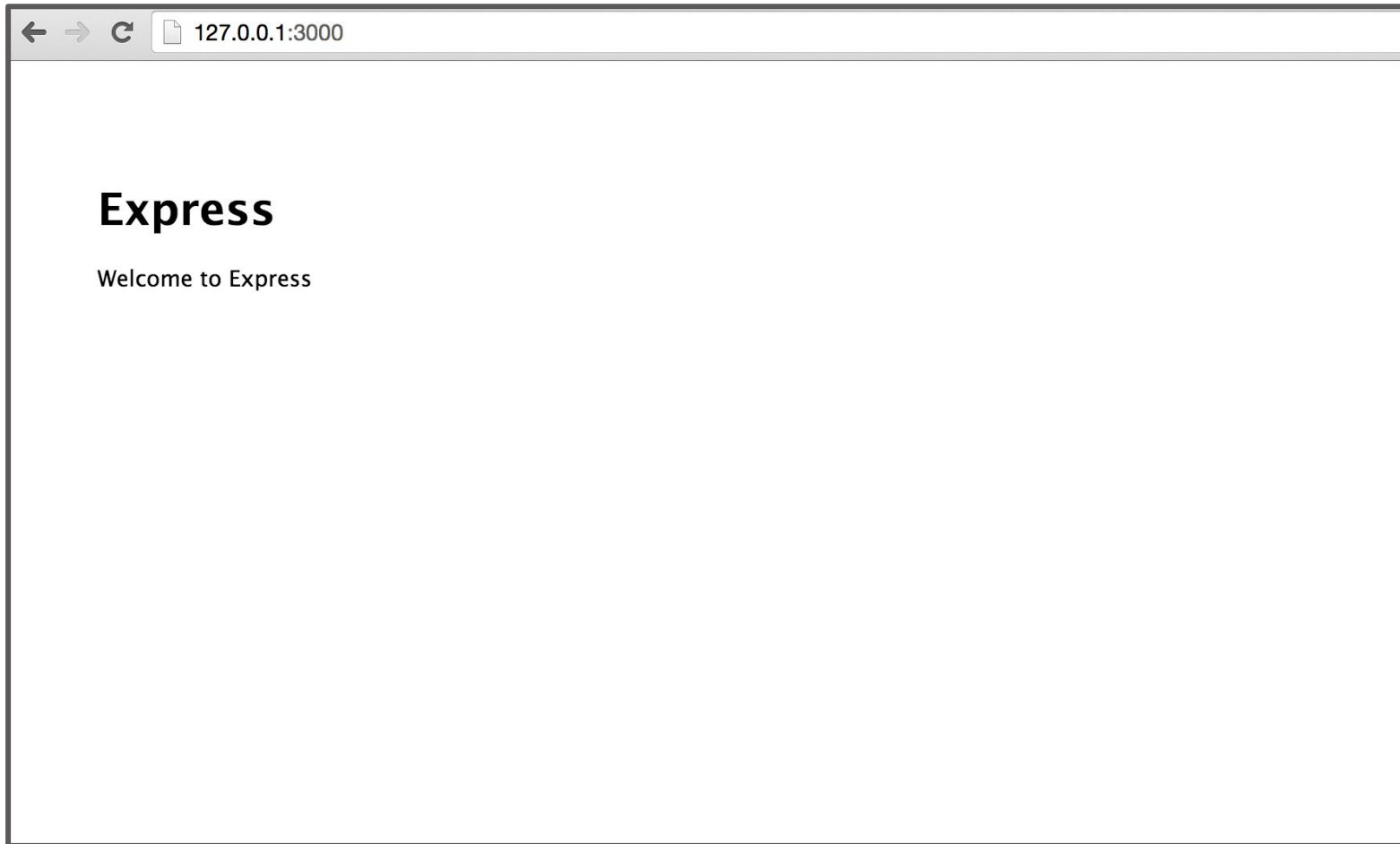
```
MacBook-Pro-Marcin-2:test marcin$ npm start
```

```
> test@0.0.0 start /Users/marcin/test
```

```
> node ./bin/www
```

Po uruchomieniu aplikacja uruchamia serwer HTTP nasłuchujący na adresie IP: 127.0.0.1 oraz standardowym porcie 3000

Express JS - Uruchamianie



Strona widoczna po uruchomieniu aplikacji “0”

Express JS - Uruchamianie



```
MacBook-Pro-Marcin-2:test marcin$ npm start
```

```
> test@0.0.0 start /Users/marcin/test  
> node ./bin/www
```

```
GET / 304 194.179 ms - -
```

```
GET /stylesheets/style.css 304 2.135 ms - -
```

```
GET /favicon.ico 404 31.420 ms - 1042
```

Struktura aplikacji

**Plik tworzący serwer i obsługujący nasłuchiwanie
żądań za pomocą protokołu http jest realizowane w
następujący sposób:**

Plik: `./bin/www`

Struktura aplikacji

Plik tworzący serwer i obsługujący nasłuchiwanie żądań za pomocą protokołu http jest realizowane w następujący sposób:

Plik: ./bin/www

```
/**
 * Get port from environment and store in Express.
 */

var port = parseInt(process.env.PORT, 10) || 3000;
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);
```

Struktura aplikacji

Główny kontroler aplikacji jest zlokalizowany w pliku “app.js” znajdujący się w głównej kartotece:

Plik: app.js

Struktura aplikacji

Główny kontroler aplikacji jest zlokalizowany w pliku “app.js” znajdujący się w głównej kartotece:

```
var routes = require('./routes/index');
var users = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

// uncomment after placing your favicon in /public
//app.use(favicon(__dirname + '/public/favicon.ico'));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', routes);
app.use('/users', users);
```

Plik: app.js

Struktura aplikacji

Funkcje kontrolera odpowiedzialne za realizację konkretnych funkcjonalności znajdują się w kartotece routes. W wersji “0” plik zawiera tylko funkcję obsługującą wyświetlanie strony głównej aplikacji:

Plik: routes/index.js

Struktura aplikacji

Funkcje kontrolera odpowiedzialne za realizację konkretnych funkcjonalności znajdują się w kartotece routes. W wersji “0” plik zawiera tylko funkcję obsługującą wyświetlanie strony głównej aplikacji:

Plik: routes/index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

Struktura aplikacji

Funkcje kontrolera odpowiedzialne za realizację konkretnych funkcjonalności znajdują się w kartotece routes. W wersji “0” plik zawiera tylko funkcję obsługującą wyświetlanie strony głównej aplikacji:

Plik: routes/index.js

Metoda która realizuje wyświetlanie odpowiedniego widoku

```
var express = require('express')
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

Funkcje routera

```
router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});
```

**Metoda GET, obsługująca żądania wyświetlania
zadanego adresu URI wpisanego w przeglądarce
przez użytkownika**

Funkcje routera

```
router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});
```

Metoda GET, obsługująca żądania wyświetlania
zadanego adresu URI wpisanego w przeglądarce
przez użytkownika

Możemy również obsługiwać żądania typu POST:

```
router.post();
```

Funkcje routera

```
router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});
```

Ścieżka (lub wyrażenie regularne), które mapuje adres wpisany w przeglądarce.

<http://127.0.0.1:3000/>

<http://127.0.0.1:3000/?name=test>

Funkcje routera

```
router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});
```

Wywołanie zwrotne (callback) mające zostać wykonanw w wyniku odpowiedzi na żdanie GET

Funkcje routera

```
router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});
```

Wywołanie zwrotne (callback) mające zostać wykonane w wyniku odpowiedzi na żądanie GET

req - obiekt przenoszący zapytanie klienta (żądanie)

res - Obiekt przenoszący odpowiedź aplikacji do klienta

Funkcje routera

```
router.get('/', function(req, res) {  
  res.render('index', { title: 'Express' });  
});
```

Metoda “renderująca” widok (ostateczny HTML)
który jest następnie przesyłany do klienta.

Metoda ta jako pierwszy argument przyjmuje nazwę widoku który ma zostać wyrenderowany (nazwa pliku z kartoteki views bez rozszerzenia) oraz opcjonalnie obiekt (JSON) z danymi które mają zostać przekazane do widoku.

W przykładzie powyżej renderujemy widok o nazwie “index” i przekazujemy obiekt zawierający jedną zmienną o nazwie title.

Funkcje routera

```
router.get('/', function(req, res) {  
  var file_name = __dirname + '/../public/' + 'file.pdf';  
  res.download(file);  
});
```

Metoda pobierająca pliki, a dokładniej mówiąc wskazująca na pliku które mają zostać przesłane klientowi po wywołaniu żądanego adresu URL.

Funkcje routera

```
router.get('/', function(req, res) {  
  var file_name = __dirname + '/../public/' + 'file.pdf';  
  res.download(file);  
});
```

Metoda pobierająca pliki, a dokładniej mówiąc wskazująca na pliku które mają zostać przesłane klientowi po wywołaniu żadanego adresu URL.

Jako argument funkcji download musimy podać bezwzględną (lub względną w stosunku do kartoteki w której znajduje się plik kontrolera) ścieżkę do pliku. Jako drugi argument funkcji download można przekazać string mówiący pod jaką nazwą plik zostanie zapisany na dysk użytkownika:

```
res.download(file, 'moja_nazwa_pliku.pdf');
```

Funkcje routera

```
router.get('/', function(req, res) {  
  var options = {  
    root: __dirname + '/../public/',  
    dotfiles: 'deny',  
    headers: {  
      'x-timestamp': Date.now(),  
      'x-sent': true  
    }  
  };  
  
  var fileName = 'ph.pdf'  
  res.sendFile(fileName, options, function (err) {  
    if (err) {  
      console.log(err);  
      res.status(err.status).end();  
    }  
    else {  
      console.log('Sent:', fileName);  
    }  
  });  
});
```

Funkcje routera

```
router.get('/', function(req, res) {  
  var options = {  
    root: __dirname + '/../public/',  
    dotfiles: 'deny',  
    headers: {  
      'x-timestamp': Date.now(),  
      'x-sent': true  
    }  
  };  
  
  var fileName = 'ph.pdf'  
  res.sendFile(fileName, options, function (err) {  
    if (err) {  
      console.log(err);  
      res.status(err.status).end();  
    }  
    else {  
      console.log('Sent:', fileName);  
    }  
  });  
});
```

Metoda “sendFile”
spełnia tę samą
funkcjonalność do
download. Jest
dostępna od wersji 4.8

Funkcje routera

```
router.get('/', function(req, res) {  
  res.send('<h1>Witaj</h1>');  
});
```

Metoda send, służy do przesyłania prostych wartości tekstowych do użytkownika. Domyślny format przesyłany za pomocą tej metody jest ustawiony na "text/html".

Funkcje routera

```
router.get('/', function(req, res) {  
  res.send('<h1>Witaj</h1>');  
});
```

Metoda `send`, służy do przesyłania prostych wartości tekstowych do użytkownika. Domyślny format przesyłany za pomocą tej metody jest ustawiony na `"text/html"`.

Można również wysłać tablicę lub obiekt JSON, które zostaną następnie wyświetlone w formie preformatowanego html za pomocą znacznika `<pre>`:

```
res.send( [100, 200, 300] );  
res.send( {title: 'express'} );
```

Funkcje routera

```
router.get('/', function(req, res) {  
  res.sendStatus(200); // OK  
});
```

Metoda `sendStatus`, przesyła do przeglądarki tekstową postać odpowiedzi dla podanego jako argument kodu statusu serwera

Funkcje routera

```
router.get('/', function(req, res) {  
  res.sendStatus(200); // OK  
});
```

Metoda `sendStatus`, przesyła do przeglądarki tekstową postać odpowiedzi dla podanego jako argument kodu statusu serwera

```
res.sendStatus( 404 ); // Not Found  
res.sendStatus( 500 ); // Internal Server Error
```

```
> test@0.0.0 start /Users/marcin/test  
> node ./bin/www
```

```
GET /send 404 7.669 ms - 9  
GET /send 404 1.279 ms - 9  
GET /send 404 0.627 ms - 9  
GET /send 404 0.515 ms - 9
```

Funkcje routera

```
router.post('/formularz', function(req, res) {  
    res.render('index', { title: req.body.fname });  
});
```

W podobny sposób możemy korzystać z metody “POST” do przekazywania danych wpisanych w formularzu. Adres URI stanowiący pierwszy argument tej metody jest tzw. akcją która ma zostać wykonana po przesłaniu formularza.

Wszystkie dane zebrane z pól przesyłanego formularza są przekazane przez obiekt “req.body.NAZWA_POLA”. Aby pobrać dane posługujemy się notacją obiektową.

Funkcje routera

```
router.post('/formularz', function(req, res) {  
    res.render('index', { title: req.body.fname });  
});
```

W podobny sposób możemy korzystać z metody “POST” do przekazywania danych wpisanych w formularzu. Adres URI stanowiący pierwszy argument tej metody jest tzw. akcją która ma zostać wykonana po przesłaniu formularza.

Wszystkie dane zebrane z pól przesyłanego formularza są przekazane przez obiekt “req.body.NAZWA_POLA”. Aby pobrać dane posługujemy się notacją obiektową.

Formularz którego funkcja sterująca podana jest wyżej mógłby wyglądać następująco:

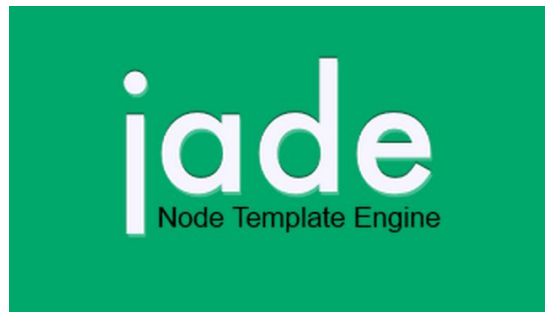
```
<form method='POST' action='/formularz'>  
    <input type='text' name='fname'>  
    <input type='submit' name='Wyslij'>  
</form>
```

JADE

W środowisku Express.js w celu uproszczenia formy oraz usprawnienia tworzenia widoków wprowadzono nowy język pisania szablonów tzw. JADE (node template engine). Jest bardzo blisko językowi html, jednak w JADE nie występują znaczniki, a jedynie nazwy określające dane znacznik.

JADE

W środowisku Express.js w celu uproszczenia formy oraz usprawnienia tworzenia widoków wprowadzono nowy język pisania szablonów tzw. JADE (node template engine). Jest bardzo blisko językowi html, jednak w JADE nie występują znaczniki, a jedynie nazwy określające dane znacznik.



<http://jade-lang.com/>

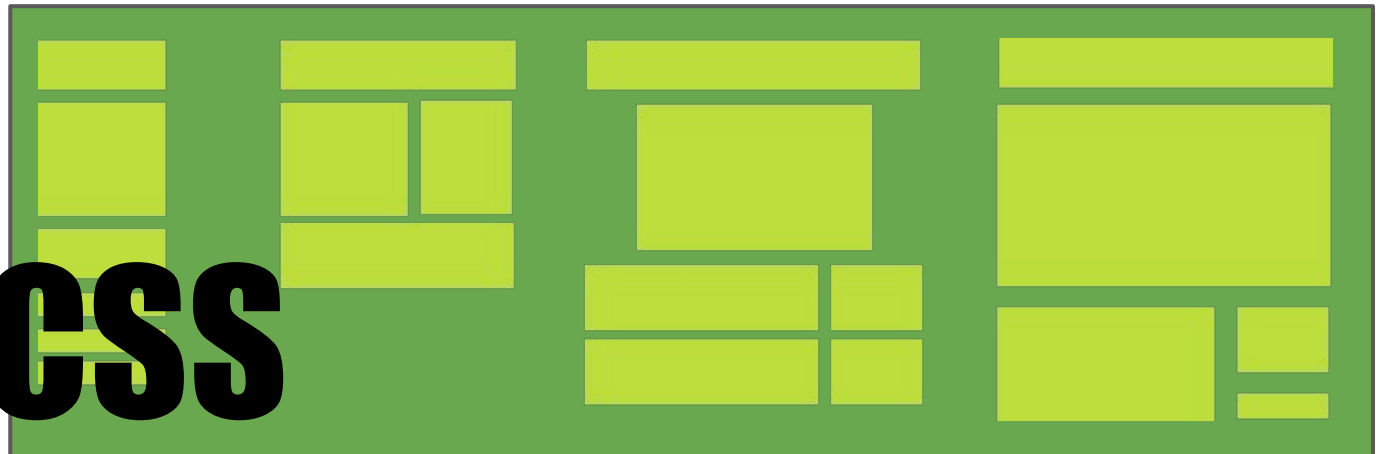
Struktura modułowa HTML



ZASADA:

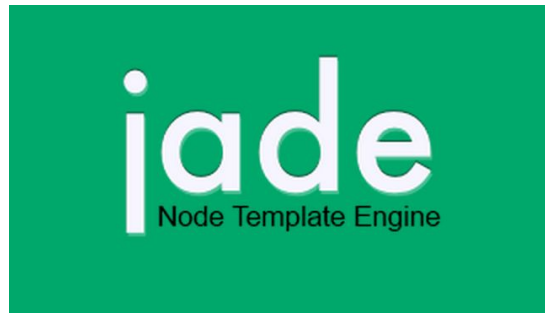
W ogólności zasada jaka powinna przyświecać tworzeniu stron zgodnie z metodologią RWD jest tworzenie jednego pliku HTML, a dla formatowania jego wyglądu wiele “layoutów” w CSS które będą zawierać odpowiednie reguły wyświetlania strony w zależności od rozdzielczości ekranu.

n x CSS



JADE

W środowisku Express.js w celu uproszczenia formy oraz usprawnienia tworzenia widoków wprowadzono nowy język pisania szablonów tzw. JADE (node template engine). Jest bardzo blisko językowi html, jednak w JADE nie występują znaczniki, a jedynie nazwy określające dane znacznik.



<http://jade-lang.com/>

```
<div>  
  Element blokowy  
</div>
```



```
div  
| Element blokowy
```

JADE

W języku JADE nie występują znaczniki otwierające oraz następujące po nich znaczniki zamykające. Przykład:

```
<body>
  <h1>Jade</h1>
  <div id="container">
    <p>You are amazing</p>
    <p>
      JADE
    </p>
  </div>
</body>
```


JADE

W języku JADE nie występują znaczniki otwierające oraz następujące po nich znaczniki zamykające. Przykład:

```
<body>
  <h1>Jade</h1>
  <div id="container">
    <p>You are amazing</p>
    <p>
      JADE
    </p>
  </div>
</body>
```



```
body
  h1 Jade
  #container
    p You are amazinng
    p.
      Jade
```

JADE

W języku JADE nie występują znaczniki otwierające oraz następujące po nich znaczniki zamykające. Przykład:

```
<body>
  <h1>Jade</h1>
  <div id="container">
    <p>You are amazing</p>
    <p>
      JADE
    </p>
  </div>
</body>
```



```
body
  h1 Jade
  #container
    p You are amazinng
    p.
      Jade
```

Zatem jak zachować hierarchię występowania i zagnieżdżania znaczników??

JADE

W języku JADE nie występują znaczniki otwierające oraz następujące po nich znaczniki zamykające. Przykład:

```
<body>  
  <h1>Jade</h1>  
  <div id="container">  
    <p>You are amazing</p>  
    <p>  
      JADE  
    </p>  
  </div>  
</body>
```



```
body  
  h1 Jade  
  #container  
    p You are amazinng  
    p.  
      Jade
```

Hierarchia znaczników w plikach JADE jest zaznaczana za pomocą
“wcięć” w tekście.

JADE

Hierarchia znaczników w plikach JADE jest zaznaczana za pomocą “wcięć” w tekście.

```
<html>
  <head>
    <title> Jade </title>
  </head>
  <body>
    <p>
      JADE
    </p>
  </body>
</html>
```



```
html
...head
.....title  Jade
...body
.....p
..... |  JADE
```

W przykładzie powyżej kropki oznaczają wcięcia zrobione za pomocą spacji. Alternatywnie można używać “tabulatora” (stałego odstępu). Należy jednak w pojedynczym pliku .jade używać spacji lub tabulacji (nigdy obu).

JADE

Hierarchia znaczników w plikach JADE jest zaznaczana za pomocą
“wciąć” w tekście.

```
<html>
  <head>
    <title> Jade </title>
  </head>
  <body>
    <p>
      JADE
    </p>
  </body>
</html>
```

JADE

Hierarchia znaczników w plikach JADE jest zaznaczana za pomocą “wcięć” w tekście.

```
<html>
  <head>
    <title> Jade </title>
  </head>
  <body>
    <p>
      JADE
    </p>
  </body>
</html>
```



```
html
...head
.....title  Jade
...body
.....p
..... |  JADE
```

W przykładzie powyżej kropki oznaczają wcięcia zrobione za pomocą spacji. Alternatywnie można używać “tabulatora” (stałego odstępu). Należy jednak w pojedynczym pliku .jade używać spacji lub tabulacji (nigdy obu).

JADE

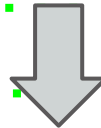
Zwykły tekst:

```
Plain text can include <strong>html</strong>  
<p>It must always be on its own line</p>
```

JADE

Zwykły tekst:

```
Plain text can include <strong>html</strong>  
<p>It must always be on its own line</p>
```

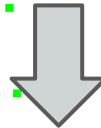


```
| Plain text can include <strong>html</strong>  
p  
  | It must always be on its own line
```


JADE

Zwykły tekst:

```
Plain text can include <strong>html</strong>  
<p>It must always be on its own line</p>
```



```
| Plain text can include <strong>html</strong>  
p  
| It must always be on its own line
```

Lista:

```
<ul>  
  <li>Item A</li>  
  <li>Item B</li>  
  <li>Item C</li>  
</ul>
```

```
ul  
  li Item A  
  li Item B  
  li Item C
```

JADE

JADE umożliwia wykonywanie bardzo prostych operacji wyliczeniowych:

```
<ul>
  <li>1</li>
  <li>2</li>
  <li>3</li>
  <li>4</li>
  <li>5</li>
</ul>
```

```
ul
  each val in [1, 2, 3, 4, 5]
    li= val
```

JADE

JADE umożliwia wykonywanie bardzo prostych operacji wyliczeniowych:

```
<ul>  
  <li>1</li>  
  <li>2</li>  
  <li>3</li>  
  <li>4</li>  
  <li>5</li>  
</ul>
```

```
ul  
  each val in [1, 2, 3, 4, 5]  
    li= val
```

oraz zadań które mają być wykonane w pętli:

```
<li>item</li>  
<li>item</li>  
<li>item</li>
```

```
- for (var x = 0; x < 3; x++)  
  li item
```

JADE

Instrukcja wyboru “case”:

```
- var friends = 10
case friends
  when 0
    p you have no friends
  when 1
    p you have a friend
  default
    p you have #{friends} friends
```

```
<p>you have 10 friends</p>
```

JADE

Zapis atrybutów w znacznikach:



```
<a href="google.com">Google</a>
```

```
<a href="google.com" class="button">Google</a>
```

```
a(href='google.com') Google
```

```
a(class='button', href='google.com') Google
```

JADE

Zapis atrybutów w znacznikach:



```
<a href="google.com">Google</a>  
<a href="google.com" class="button">Google</a>
```

```
a(href='google.com') Google  
a(class='button', href='google.com') Google
```




```
<a style="color:red;background:green"></a>
```

```
a(style={color: 'red', background: 'green'})
```

JADE

Zapis klas i identyfikatorów:



```
<a class="button"></a>  
<div class="content"></div>
```

```
a.button  
.content
```

JADE

Zapis klas i identyfikatorów:

```
<a class="button"></a>  
<div class="content"></div>
```

```
a.button  
.content
```

```
<a id="main-link"></a>  
<div id="content"></div>
```

```
a#main-link  
#content
```


JADE

Dziedziczenie widoków:

```

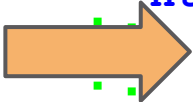
<!doctype html>
<html>
  <head>
    <title>Article
  </title>
</head>
<body>
  <h1>My Article</h1>
</body>
</html>

```

JADE

Dziedziczenie widoków:

```
<!doctype html>
<html>
  <head>
    <title>Article
    Title</title>
  </head>
  <body>
    <h1>My Article</h1>
  </body>
</html>
```



```
//- layout.jade
doctype html
html
  head
    block title
      title Default title
  body
    block content
```

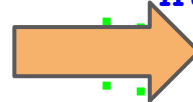


```
//- index.jade
extends ../layout.jade
block title
  title Article Title
block content
  h1 My Article
```

JADE

Dziedziczenie widoków:

```
<!doctype html>
<html>
  <head>
    <title>Article
    Title</title>
  </head>
  <body>
    <h1>My Article</h1>
  </body>
</html>
```



```
//- layout.jade
doctype html
html
  head
    block title
      title Default title
  body
    block content
```



```
//- index.jade
extends ./layout.jade
block title
  title Article Title
block content
  h1 My Article
```

Taka organizacja widoków
pozwała na stworzenie modularnej
i hierarchicznej struktury strony.

JADE

Przykład formularza na stronie które są przesyłane do serwera za pomocą funkcji “post”:

```
div
  form(method='POST' action='/formularz')
    input(type='text' name='imie')
    input(type='text' name='nazwisko')
    input(type=submit name='Wyslij')
```

JADE

Przykład formularza na stronie które są przesyłane do serwera za pomocą funkcji “post”:

```
div
  form(method=' POST'  action='/formularz' )
    input(type=' text'  name=' imie' )
    input(type=' text'  name=' nazwisko' )
    input(type=submit name=' Wyslij' )
```

Metoda kontrolera odbierająca dane z formularza:

```
router.post('/formularz', function(req, res) {
  var _imie = req.body.imie;
  var _nazwisko = req.body.nazwisko;
  var osoba = _imie + _nazwisko;
  res.render('index', { title: osoba });
});
```

KONIEC WYKŁADU 10
