NODE.JS i EXPRESS.JS

(08.01.2015 r.)

Użycie pakietu EXPRESS.JS do budowania sklaowalnych aplikacji internetowych

1) Node.JS umożliwa również tworzenie złożonych i zaawansowanych aplikacji “webowych” w architekturze MVC (Model-View-Controller). Dziś poznamy kompletne narzędzia i metody do tworzenia aplikacji w tej architekturze w środkowisku NODE.JS.

2) Do tworzenia aplikacji w architekturze MVC w środkowsku NODE.JS służy pakiet “Express” (strona projektu: https://github.com/visionmedia/express). Jest to bardzo rozbudowane narzędzie ułatwiające budowanie aplikacji webowych w NODE.JS. Aby z niego skorzystać należy przeprowadzić instalację pakietu “express” składającą się z kilku kroków:

0) Domyślnie używamy powłoki terminala “bash”. Jeśli nie jest to domyślna powłoka należy się do niej przełączyć wpisując w terminalu polecenie:

```
> bash
```

1) Definiujemy tworzymy następujące linki symboliczne w głownej kartotece nodejs:

```
> ln -s ./bin/node node
> ln -s ./bin/npm npm
```

2) W pliku .bashrc (jeśli nie istnieje tworzymy taki w sowjej głównej kartotece domowej) dodajemy następującą zmienną środowiskową:

export PATH=$PATH:/hom/username/nodejs/bin

gdzie username - ocznacza nazwę użytkownika
nodejs - oznacza nazwę kartoteki instalcji nodjs

3) Następnie w głównej kartotece domowej wykonujemy polecenie:

```
> source .bashrc
```

4) Za pomocą standardowej metody instalacji pakietówm należy zainstalować w środowisku node.js moduł “express” w głownej kartotece nodejs:

```
> ./npm install express
```

5) Dodatkowo musimy doinstalować pakiet generatora express:

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

Na tym etapie środowisko NODE.JS jest gotowe do tworzenie aplikacji w architekturze MVC.
Aby utworzyć nowy projekt należy wykonać następujące kroki:

1) Utworzyć w kartotece głównej NODE.JS nowy katalog np.:

```
> mkdir hello
```

2) Wygenerować strukturę aplikacji:

```
> express hello
```

3) Wejść do kartoteki hello:

```
> cd hello/.
```

4) Doinstalować starndardowe pakiety:

```
> npm install
```

5) Na tym etapie aplikacja jest gotowa do uruchomienia. Serwer uruchamiamy wydając polecenie:

```
> npm start
```

6) W przeglądarce, po wpisaniu adresu *"localhost:3000"* powinniśmy ujrzeć domyślną stronę projektu express.

7) Dodoatkowo w konsoli w której uruchomiliśmy serwer dostajemy "logi" np.:

```
GET / 200 607ms - 270b
GET /stylesheets/style.css 200 8ms - 110b
```

które mówią o czasach wykonywania przychodzących żadań do serwera i pobieranych plikach.

3) Zgodnie podziałem na trójwarstwową strukturę aplikacji (Model-View-Controller) poszczególne moduły aplikacji są rozproszone w różnych kartotekach. Podstawowa struktura kartotek jest następująca:

```
-rw-r--r--  05-26 10:52 app.js            // główny plik aplikacji
drwxr-xr-x  05-26 10:52 bin               // katalog z serwerem
drwxr-xr-x  05-26 10:52 node_modules      // dodatkowe moduły node.js
-rw-r--r--  05-26 10:52 package.json
drwxr-xr-x  05-26 10:52 public            // pliki publiczne np. css
drwxr-xr-x  05-26 10:52 routes            // kontroler
drwxr-xr-x  05-26 10:52 views             // widok
```

Kartoteka "models" nie jest standardowo tworzona.

4) Architektura trójwarstwowa wymusza podział na akcje i widoki. Gdzie "akcje" (znajdujące się kontrolerze) są odpowiedzialne za wykonywanie operacji i obsługę żadań przychądzących do serwera, a widoki za wyświetlanie danych przychodzących z kontrolera. Standardowo "akcje" przechowywane są w krtotoece "routes", a widoki w kartotece "views".

Rozpocznijmy od kontrolera: w kartotece "routes" znajdują sie dwa plik jeden o nazwie "index.js" drugi "user.js". Zajmijmy się tym pierwszym, który odpowiada za obsługę strony głównej w naszej aplikacji. Jego struktura jest bardzo prosta i zawiera tylko jedną funckję obsługującą żądania GET przychodzące do serwera.

Dygresja:

GET -> nastepuje kiedy wchodzimy na stonę (wpisujemy adres w przeglądarce) i w odpowiedzi na żadanie tego typu serwer zwraca treść do wyświetlania.

POST -> następuje kiedy przesyłamy dane do serwera (np. wysyłając dane wpisane do formularza) które mają zostać przetworzone.

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

module.exports = router;
```

W tej funkcji obsługujemy żadanie GET przychodzące od klienta (z przeglądarki), wymuszące wyświetlenie strony głównej aplikacji. W odpowiedzi na przychodząse żadanie “req” formowana jest odpowiedz “res”, która renderuje stonę “index.html” na podstawie widoku znajdującego się w kartotece “views” (ale o tym za chwile). Przenalizujmy, jakie wartości przymuje metoda “render”.
Jako pierwszy argument podajemy nazwę pliku widoku który ma zostać wyświetlony w wyniku realizacji tego żadania, a drugi parametr to obiekt z danymi przekazywanymi do widoku które mają zostać wyświetlone ostatecznie na stronie. Można to porównać do funkcji która zwraca wartość (tu w postaci złożonego obiektu) do innej funkcji.
Przejdzmy teraz do widoku. W oparciu o architekturę MVC środowisko node.js wraz z modułem “express” wymusza wprowadzenie nowej formuły tworzenie widoków. Widoki poszczególnych strone przechowywane są w plikach “jade”. Pliki te nie mają postaci standardowego kodu html, pewnego rodzaju kodu “semi-html”. Strona zbudowana jest w oparciu o głowny widok przechowywany w pliku “layout.jade” oraz plików dodatkowych obsługujących poszczególne strony będące rozszerzeniem widoku głownego. W najprostszym ujęciu plik “layout” przechowuje część “head” strony, a pliki z konkretnymi podstronami zwierają tylko sekcję “body” która jest dynamicznie podmieniana w zależności od wysłanego żadania do serwera:

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

W klasycznym pliku kodzie html skłądnia powyższa składnia jade odpowiadałaby:

```
<!DOCTYPE html>
<html>
<head>
     <title> ….. </title>
     <link rel="stylesheet" href="/stylesheets/style.css" />
</head>
<body>
    …….
</body>
</html>
```

Jak widać składnia “semi-html” jade, jest pozbawiony znaków określających znaczniki oraz jest również samo uzupełniający się ponieważ nie ma znaczników zamykających!!!!
Przykład widoku wyświetlajacego stronę główną jest umieszczony w pliku “index.jade”:

```
extends layout

block content
  h1= title
  p Welcome to #{title}
```

Polecenie “extends layout” mówie serwerowi, że ten plik rozszerza funkcjonalność głównego widoku umieszczonego w pliku layout.

Dodatkowo w obu przypadkach widoku głównego, jak i widoku rozszerzającego widać również jak należy odbierać zmienne przekazywane przez w obiekcie funkcji render. Jak pamiętamy przekasywaliśmy w kontrolerze objekt:

```
{ title: ‘Express’}
```

Jest on teraz wyświetlany w dwóch miejscach w elemencie <h1> oraz <p>. W obu przypadkach dostęp do obiektu jest inny, jednak są to dwie równoważne metody. Z doświadczenie polecam metodę drugą czyli **`#{ nazwa zmiennej }`**.