Zaawansowane Programowanie Webowe

Laboratorium nr 8

Celem laboratorium jest zapoznanie z możliwościami JavaScript po stronie serwera poprzez wykorzystanie platformy NodeJS. W ramach ćwiczenia stworzony zostanie prosty serwer http oraz zdefiniowane podstawowe usługi REST potrzebne do realizacji modelu CRUD na bazie danych MangoDB.

Tematyka laboratorium:

Na laboratorium wykorzystujemy aplikację sklepu sportowego tworzonego w ramach zajęć laboratoryjnych. Zgodni z harmonogramem powinniście mieć już Państwo stworzoną funkcjonalność Front-endu zarówno dla obsługi klienta jak i panel administracyjny. Po realizacji lab 7 Państwa aplikacja zasilana jest danymi pochodzącymi z dedykowanego środowiska realizowanego w modelu PaaS – FireBase. Jak pamiętamy jest to platforma umożliwiająca realizacje różnego rodzaju usług w tym obsługę funkcjonalności serwera backendowego. Korzysta on z wbudowanej baza danych typu MangoDB oraz graficznego interfejsu do definiowania schematu danych.

Dzisiejsze laboratorium spróbujemy napisać własną implementacje serwera aplikacyjnego, którego zadaniem będzie zarówno hostowanie aplikacji Frontendowej (tak aby przestać korzystać z wbudowanego w środowisko Angular CLI serwera) jak i obsługi zapytań REST odnoszących się modelu danych. Całość tej funkcjonalności zostanie zrealizowana z wykorzystaniem NodeJS – jako platformy programistycznej JS po stronie serwera, frameworku Express oraz biblioteki mongoose.

Materiały pomocnicze do laboratorium można znaleźć pod poniższymi adresami:

https://www.tutorialspoint.com/nodejs/nodejs_first_application.htm - opis poszczególnych elementów NodeJS
http://adrianmejia.com/blog/2014/10/01/creating-a-restful-api-tutorial-with-nodejs-and-mongodb

lub moje wykłady nr 5 i 6 znajdujące się na stronie.

Etap 1. **Instalacja Node.js**

Przejdź do witryny http://nodejs.org/ i pobierz pakiet instalacyjny Node.js dla Twojej platformy (dostępne są wersje dla systemów operacyjnych Windows, Linux i OS X) i zainstaluj go. Upewnij się o instalacji menedżera pakietów ***npm*** oraz o dodaniu katalogu instalacyjnego Node.js do ścieżki (zmiennej środowiskowej PATH).

Aby przetestować poprawność instalacji Node.js, przejdź do wiersza poleceń i wydaj polecenie node. Poczekaj na zmianę znaku zachęty, a następnie wprowadź poniższe polecenie (w jednym wierszu):

```
function testNode() {return "Node działa!"}; testNode();
```

W przypadku interaktywnego użycia Node.js dane wejściowe zostaną uznane za kod JavaScript. Jeżeli instalacja Node.js jest prawidłowa, to powinieneś otrzymać następujące dane wyjściowe:

```
'Node działa!'
```

Zadanie 1. Samodzielnie napisz serwera WWW w czystym NodeJS , który pozwoli na serwowanie dowolnych plików statycznych (w tym serwujące nasz frontend) oraz realizują podstawowe funkcje obsługi zapytań o dodatkowe dane. W przypadku zapytań o dane zwracaj randomowe dane zaszyte bezpośrednio w serwerze lub w pliku zewnętrznym.

Informacje pomocnicze.

Instalacja prostego serwera WWW

Korzystając z NodeJS można w bardzo prosty sposób stworzyć prosty serwer http serwujący pliki statyczne. W tym celu należy:
Z poziomu katalogu instalacyjnego Node.js wydaj poniższe polecenia:

```
npm install connect
npm install serve-static
```

Polecenie npm to menedżer pakietów Node.js, który pobierze pliki wymagane do działania modułu Connect wymaganego do realizacji prostego serwera http.
Następnie utwórz nowy plik o nazwie *server.js* (nadal pozostając w katalogu instalacyjnym Node.js) i umieść w nim poniższy kod.

```
var connect = require('connect');
var serveStatic = require('serve-static');
connect().use( serveStatic("../angular")).listen(5000);
```

Kod umieszczony w pliku server.js powoduje utworzenie prostego serwera WWW odpowiadającego na żądania kierowane do portu 5000 i obsługującego pliki w katalogu angular, który na dysku znajduje się na tym samym poziomie co katalog instalacyjny Node.js.

Bardziej zaawansowany serwer http zbudowany w NodeJS wyglądałby mniej więcej tak:

```
 var http = require('http');
var fs = require('fs');
var url = require('url');
http.createServer( function (request, response) {
        // Parsuje żądanie zawierające nazwę pliku
    var pathname = url.parse(request.url).pathname;
        // Wyświetlanie nazwy pliku, którego dotyczyło żądanie
    console.log("Request for " + pathname + " received.");
    fs.readFile(pathname.substr(1), function (err, data) {
    if (err) {
        console.log(err);
        response.writeHead(404, {'Content-Type': 'text/html'});
    } else {
        response.writeHead(200, {'Content-Type': 'text/html'});
        response.write(data.toString()); // zwracanie treści wybranego pliku
    }
    response.end(); // wysyłanie odpowiedzi
});
}).listen(5000);
```

Uruchomienie serwera WWW

Aby uruchomić serwer WWW, z poziomu katalogu instalacyjnego Node.js wydaj poniższe polecenie:

node server.js

W ten sposób nastąpi wczytanie pliku *server.js* i rozpoczęcie nasłuchiwania żądań HTTP na porcie 5000.

Etap 2. **Instalacja wymaganych pakietów potrzebnych do wystawienia REST API**

Aplikacja Sklep składa się z dwóch modułów: sklepu prezentującego dostępne artykułu klientowi, który może je zamawiać oraz modułu administracyjnego do zarządzania

paletą produktów. Oba moduły wymagają dostępu do danych znajdujących się na serwerze Backendu w bazie MangoDB poprzez wystawione usługi REST API.

Framework Express

Express to szybki, minimalistyczny szablon aplikacji internetowych i mobilnych dla Node.js. Pozwala na szybkie budowanie aplikacji bazujących na Node.js poprzez utworzenie warstw pośredniczących odpowiadających na żądania http oraz dostarczających mechanizmów routing będącego podstawą REST API - różne akcje dla różnych metod i adresów URL.

Instalacja:

npm install express

Utworzenie szkieletu aplikacji

Najprostszy serwer plików statycznych w NodeJS z wykorzystaniem Express wyglądałby tak:

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
    res.send('działa ale na razie statycznie')
})
app.listen(3000);
```

Następnie tworzymy plik server.js którego szkielet mógłby wyglądać tak:

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

app.get('/', function (req, res) {
    console.log("Otrzymano żądanie GET dla strony głównej");
    res.send('Hello GET');
})
app.post('/', function (req, res) {
    console.log("Otrzymano żądanie POST dla strony głównej");
```

```
        res.send('Hello POST');
})
app.delete('/usun', function (req, res) {
        console.log("Otrzymano żądanie DELETE dla strony /usun");
        res.send('Hello DELETE');
})
app.put('/user_list', function (req, res) {
        console.log("Otrzymano żądanie PUT dla strony /user_list");
        res.send('Lista użytkowników');
})
app.get('/ab*cd', function(req, res) { // wzorzec strony: abcd, abxcd, ab123cd, ...
        console.log("Otrzymano żądanie GET dla strony /ab*cd");
        res.send('Wzorzec strony dopasowany');
})
var server = app.listen(5000, function () {
        var host = server.address().address
        var port = server.address().port
        console.log("Przykładowa aplikacja nasłuchuje na http://%s:%s", host, port)
});
```

Zadanie 2. Przygotuj plik product.json zawierający przykładowe dane – struktura pliku product – taka sama jak we wcześniejszych lab. Następnie zaimplementuj REST API pozwalające na odczyt, modyfikacje, usuwanie i dodawanie nowych pozycji w pliku.

Przykładowy kawałek kodu serwera odczytujący i zwracający listę produktów mógłby wyglądać tak:

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

var app = express();

var fs = require("fs");          <-  moduł do obsługi plików

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

        fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
```

```
        console.log( data );

        res.end( data );

});

})

app.listen(5000);
```

Przetestuj funkcjonowanie napisanego przez Ciebie Serwera z aplikacja Sklep.

Biblioteka mongoose

Mongoose to moduł Node, dzięki któremu używanie bazy danych MongoDB jest bardzo łatwe. Model Mongoose (zgodnie z architekturą model-widok-kontroler) zapewnia interfejs dla kolekcji MongoDB, a także dodatkowe, użyteczne funkcje, takie jak hierarchie schematów, warstwę pośrednią i weryfikację. Dokumentacje biblioteki znajdziesz pod adresem *http://mongoosejs.com/docs/*

Aby nie trzeba było instalować lokalnie bazy MangoDB proponuje skorzystać z usługi mLab udostępniającej dostęp do wirtualnych instancji bazy w chmurze zewnętrznej. https://mlab.com
Zapoznaj się z dokumentacją na znajdująca się na stronie i stwórz swoją własną bazę.

Instalacja:

npm install mongoose

Dołączamy mongoose do projektu i łączymy się z bazą danych test (zostanie utworzona jeśli takiej nie było):

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');
```

Sprawdzamy czy połączenie się udało:

```
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'błąd połączenia...'));
db.once('open', function() {
        // połączenie udane!
});
```

Podczas zarządzania danymi za pomocą Mongoose konieczne jest zarejestrowanie schematu. Przedstawiony poniżej fragment kodu pokazuje, jak można zarejestrować schemat dla bazy danych przechowującej zadania. Odpowiada on pojęciu kolekcja w MangoDB.

```
var Schema = mongoose.Schema;
var Tasks = new Schema({
        project: String,
        description: String
});
```

Mając schemat, na jego bazie tworzymy model

```
mongoose.model('Task', Tasks);
```

DODANIE ZADANIA

Po zarejestrowaniu schematu można uzyskać do niego dostęp. Przedstawiony poniżej fragment kodu pokazuje, jak dodać zadanie za pomocą modelu:

```
var Task = mongoose.model('Task');
var task = new Task();
task.project = 'Malowanie';
task.description = 'Pomalować rower na czerwono.';
task.save(function(err) {
        if (err) throw err;
        console.log('Zadanie został o zapisane.');
});
```

WYSZUKIWANIE DOKUMENTU

Wyszukiwanie za pomocą Mongoose jest łatwe. Dostępna jest bogata składnia zapytań z MongoDB: find(), findById(), findOne() i where().
Metoda find() modelu Task pozwala na wyszukanie wszystkich dokumentów lub wskazanie konkretnych za pomocą obiektu JavaScript zawierającego kryteria filtrowania. Przedstawiony poniżej fragment kodu przeprowadza operację wyszukiwania zadań powiązanych z konkretnym projektem,
a następnie wyświetla unikatowy identyfikator i opis każdego znalezionego zadania:

```
var Task = mongoose.model('Task');
Task.find({'project': 'Malowanie'}, function(err, tasks) {
        for (var i = 0; i < tasks.length; i++) {
                console.log('ID:' + tasks[i]._id);
                console.log(tasks[i].description);
```

```
        }
});
```

UAKTUALNIANIE DOKUMENTU

Wprawdzie istnieje możliwość użycia metody find() modelu do wyszukania dokumentu, który następnie będzie zmieniony i zapisany, ale modele Mongoose oferują także metodę update() przeznaczoną do wymienionego celu. Przedstawiony poniżej fragment kodu pokazuje, jak można uaktualnić dokument za pomocą Mongoose:

```
var Task = mongoose.model('Task');
Task.update( {_id: '4e65b793d0cf5ca508000001'},
        {description: 'Pomalować rower na zielono.'},
        {multi: false},
        function(err, rows_updated) {
                if (err) throw err;
                .log('Uaktualniono.');
        }
);
```

USUWANIE DOKUMENTU

Po pobraniu dokumentu bardzo łatwo można go usunąć. Pobieranie i usuwanie dokumentu odbywa się za pomocą wewnętrznego identyfikatora (lub innego kryterium, jeśli zamiast metody findById() używana jest find()) i kodu podobnego do poniższego:

```
var Task = mongoose.model('Task');
Task.findById('4e65b3dce1592f7d08000001', function(err, task) {
        task.remove();
});
```

Inne metody do usuwania:
remove(warunki, [funZw]) - usuwa wszystkie dokumenty spełniające podane warunki
findByIdAndRemove(id, [op cje], [funZw]) - usuwa dokument o podanym id, zwraca usuwany dokument przez funZw
findOneAndRemove(warunki, [opcje], [funZw]) - usuwa pierwszy z dokumentów spełniających podane warunki, zwraca usuwany dokument przez funZw

Etap 3. **Implementacja serwera http w NodeJS w oparciu o dane znajdujące się w MangoDB**

Zadanie 3. W oparciu o wcześniejszy opis zaimplementuj serwer http obsługujący REST API oraz model danych wykorzystywany w aplikacji Sklep. -> dotyczy zarówno modułu klienta jak i administratora. Do przechowywania danych wykorzystaj bazę danych MangoDB.

Szkielet naszej aplikacji – serwer WWW wygląda następująco:

```
var express = require('express');
var mongoose = require('mongoose');
var bodyParser = require('body-parser');
var app = express();

// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }))
app.use(bodyParser.json()); // parse application/json
// ... schemat i model
// ... obsługa API
app.get('/', function(req, res) {
res.end("Witam Was serdecznie na mojej stronie :)");
})
app.listen(5000);
```