Kierunki rozwoju Oprogramowania

MEAN Stack

Edytor

Do pracy przy projekcie w supełności wystarczy edytor tekstowy taki jak SublimeText. Powinien on być już ainstalowany w systemie. W przeciwnym przypadku instrukcja jego instalacji znajduje się na stronie prowadzącego.

Logika aplikacji

Pierwszą czynnością którą należy wykonać jest zazwyczaj stworzenie aplikacji Angular (plik javascript), która będzie zawierała funkcjonalność po stronie front endu. Plik taki umieszczamy w katalogu **public** aby wyraźnie odseparować go od logiki serwera.

W katalogu `public/javascripts` tworzymy moduł **cwierkaczApp.js**, a następnie dodajemy do niego kontroler, który zajmie się podstawową funkcjonalnością aplikacji (wyświetlanie, dodawanie postów).

Następnie tworzymy przykładowe widoki `main.html`, `login.html`, i `register.html` (na razie będą to puste pliki), które umieszczamy w katalogu **public**.

```
//cwierkaczApp.js
var app = angular.module('cwierkaczApp', []);
app.controller('mainController', function(){
});
```

HTML

Naszym bazowym plikiem widoku będzie `main.html` umieszczony w katalogu `public`. Zawiera formularz do tworzenia nowych postów jak i listę ostatnich postów. W celu uproszczenia kodu na tym etapie nasza aplikacja będzie posiadała osobne pole do wprowadzania loginu (autora posta) za każdym razem - nie będzie go w żaden sposób weryfikować.

```
<!--main.html-->
<!doctype html>
<html>
```

```
  <head>
    <meta charset="utf-8">
    <title>Ćwierkacz</title>
  </head>
  <body>
    <div id='main'>
      <form>
        <input type="text" placeholder="Nick" />
        <textarea required maxlength="200" rows="3" placeholder="Powiedź coś">
        </textarea>
        <input class="button" type="submit" value="Ćwierk!" />
      </form>
      <div id="post-stream">
        <h4>Ostatnie Ćwierki</h4>
              <!--Zostanie uzupełnione przez logikę aplikacji-->
      </div>
    </div>
  </body>
</html>
```

Teraz pozostało jedynie podpięcie widoku do aplikacji.

Najpierw należy dołączyć pliki frameworka (angular.js) i aplikacji (część frontendowa - cwierkaczApp.js) do widoku.

```
<!--main.html-->
<head>
  <meta charset="utf-8">
  <title>Ćwierkacz</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js">
  </script>
  <script src="javascripts/cwierkaczApp.js"></script>
</head>
```

Następnie dodajemy dyrektywę `ng-app` do jednego z elementów drzewa DOM naszego widoku - będzie to body. Po tym należy jeszcze dodać dyrektywę `ng-controller` do elementu zawartego w pliku HTML (wewnątrz `ng-app`) i jako argument przekazać nazwę kontrolera. Będzie to element do którego kontroler będzie miał dostęp modyfikując zmienną **$scope**.

```
<!--main.html-->
<body ng-app="cwierkaczApp">
  <div id='main' ng-controller="mainController">
      ...
  </div>
</body>
```

Łączymy model i widok

Teraz możemy tworzyć obiekty w kontrolerze, które będą dowiązane do widoku. Dodajmy tablicę `posts` jako pole zmiennej `$scope` oraz słownik `newPost` do przechowywania informacji o aktualnie tworzonym wpisie. Stworzymy funkcję `post`, która doda zawartość naszego posta do tablicy za każdym razem kiedy użytkownik kliknie odpowiedni przycisk w widoku (wyślij).

```
//cwierkaczApp.js
var app = angular.module('cwierkaczApp', []);

app.controller('mainController', function($scope){
  $scope.posts = []; // wszystkie wpisy
  $scope.newPost = {created_by: '', text: '', created_at: ''}; // dane formularza

  $scope.post = function(){
    $scope.newPost.created_at = Date.now(); // data utworzenia
    $scope.posts.push($scope.newPost); // dodaj wpis do widoku
    $scope.newPost = {created_by: '', text: '', created_at: ''}; // czyść formularz
  };
});
```

Dowiązanie danych, filtrowanie

Aby Angular wiedział skąd wziąć zawartość do utworzenia nowego posta, w pliku `main.html` skorzystamy z dyrektywy `ng-model` wskazującej na nazwę pola w obiekcie **$scope**. Aby wyświetlić zawartość tablicy z postami użyjemy dyrektywy `ng-repeat`, która przeiteruje po dostarczonej zmiennej z modelu (musi to więc być lista, słownik lub dowolny element, który pozwala na iterowanie). Posty zostaną posortowane po dacie utworzenia. Zauważ że dla elementów parzystych/nieparzystych ustawiamy różne klasy, żeby ustawić im inne klasy - nie będą zlewać się w całość przy wyświetlaniu.

```
<!--main.html-->
<div id='main' ng-controller="mainController">
  <form ng-submit="post()">
    <input required type="text" placeholder="Nick" ng-model="newPost.created_by" />
    <textarea required maxlength="200" rows="3" placeholder="Powiedź coś"></textarea>
    <input class="button" type="submit" value="Ćwierk!" />
  </form>
  <div id="post-stream">
    <h4>Ostatnie Ćwierki</h4>
        <div class='post' ng-repeat="post in posts | orderBy:'created_at':true"
            ng-class-odd="'odd'" ng-class-even="'even'">
        <p>{{post.created_by}} says {{post.text}} at {{post.created_at}}</p>
        </div>
```

```
    </div>
  </div>
```

CSS

Ostatnią rzeczą o którą zadbamy w tej części będzie styl elementów. Dodaj do katalogu `stylesheets/` plik z materiałów.
Następnie w sekcji `head` dodaj wpis o arkuszach stylów, a następnie dodaj wyświetlanym elementom odpowiednie klasy:

```
<!--main.html-->
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ćwirkacz</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js">
    </script>
    <script src="javascripts/cwierkaczApp.js"></script>
    <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/bootstrap.min.css">
    <link rel="stylesheet" href="stylesheets/style.css">
  </head>
  <body ng-app="cwierkaczApp">
    <div id='main' class="container" ng-controller="mainController">
      <div class="col-md-offset-2 col-md-8">
        <div class="clearfix">
          <form ng-Submit="post()">
            <input required type="text" class="form-control" placeholder="Your name"
              ng-model="newPost.created_by" />
            <textarea required class="form-control" maxlength="200" rows="3"
              placeholder="Say something" ng-model="newPost.text"></textarea>
            <input class="btn submit-btn pull-right" type="submit" value="Ćwierk!" />
          </form>
          <div id="post-stream">
            <h4>Ostatnie Ćwierki</h4>
                <div class="post" ng-repeat="post in posts | orderBy:'created_at':true"
                ng-class-odd="'odd'" ng-class-even="'even'">
                  <p>{{post.text}}</p>
                <small>Posted by @{{post.created_by}}</small>
                <small
                class="pull-right">{{post.created_at | date:"h:mma 'on' MMM d, y"}}
                </small>
                </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

Register & login

Dodaj podobne pliki HTML pozwalające na logowanie się i rejestrację. Na razie nie będą one spełniały swojej funkcjonalności w 100%, jednak będą nam potrzebne później.

Server Node

Mamy już dostęp do podstawowego front endu, jednak jak można zauważyć informacje są tracone w momencie odświeżenia strony. Dodamy teraz część serwerową, która docelowo pozwoli nam na uwierzytelnianie, zachowywanie dodanych postów oraz zarządzanie widokami. Zaczniemy od zdefiniowania serwera i opracowania REST API, z którego będziemy korzystać.

Z materiałów dołączonych do laboratorium wypakuj katalog bin (jeżeli jeszcze tego nie zrobiłeś), a następnie w głównym katalogu utwórz plik **index.js** z następującą zawartością

```
var express = require('express');
var api = require('./routes/api');
var app = express();
app.use('/api', api);
module.exports = app;
```

W pierwszej linijce importujemy moduł express. Kolejna zaimportuje moduł, w którym za chwilę zdefiniujemy jak będzie zachowywać się API. Kolejna tworzy instancję aplikacji, a następnie definiuje, który moduł ma się zająć obsługą żądań kierowanych na adres **/api**. Ostatnia linijka jest odpowiedzialna za dopisanie odpowiednich elementów do składowych eksportowanych przez moduł. Bez niej, moduł nie będzie mógł zostać wykorzystany przez serwer nodejs (możesz spróbować ją za chwilę zakomentować i zrestartować serwer).

Aby uruchomić serwer wydaj polecenie

```
DEBUG=cwierkacz ./bin/www
```

Routing

Reguły routingu będziemy umieszczać w osobnych modułach tak żeby łatwo było poruszać się po projekcie i utrzymać go w porządku. Będą one znajdowały się w folderze **routes**. Pierwszy moduł pozwoli nam na dostęp do REST'owego API.
Zaczniemy od importu wymaganych modułów

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

W kolejnych krokach zdefiniujemy jakie endpointy będzie definiował serwer i dla każdego z nich określimy jak ma reagować na każdą z metod nam potrzebnych (POST, GET, DELETE, itd).

```
router.route('/posts')
    .get(
        function(req,res){
          return res.json({'posts' : []});
        }
    );
module.exports = router;
```

Odwiedź teraz odpowiednią podstronę lub skorzystaj z jednej z wtyczek dostępnych do Twojej przeglądarki (np: Postman, RESTClient) aby sprawdzić działanie endpointa i zweryfikować poprawność odpowiedzi.

Dodaj pozostałe metody, które pozwolą Ci na interakcje z serwisem (CRUD). Pamiętaj, że metody możesz układać w łańcuch aby nie powielać kodu

```
router.route('/posts')
    .get(function(req,res){})
    .post(function(req,res){})
    .delete(function(req,res){})
    .put(function(req,res){});
```

Model

Ostatnią częścią, którą zrealizujemy w tej części laboratorium będzie stworzenie modeli pozwalających na dostęp do bazy danych.

Serwis udostępniający bazę danych uruchomisz poleceniem **mongod**. Zapoznaj się z komunikatem i dodaj potrzebną opcję aby pozbyć się błędów wynikających z braku potrzebnego miejsca lub wskaż na partycję z windowsem, jako miejsce przechowywania danych.

Jak dobrze wiesz MongoDB nie narzuca żadnej zorganizowanej struktury na kolekcje. W związku z tym musimy wykorzystać dodatkowy moduł, który pozwoli na modelowanie danych aplikacji.

```
var mongoose = require('mongoose');
mongoose.connect("mongodb://localhost:27017/dbname")
var Schema = mongoose.Schema;
var postSchema = new mongoose.Schema({
    created_by: String,
    created_at: {type: Date, default: Date.now},
    text: String
});
mongoose.model('Post', postSchema);
```

W trzeciej linijce definiujemy schemat według, którego tworzone będą dokumenty opisujące posty. W ostatniej linijce rejestrujemy schemat tak aby można go było wykorzystać w innych fragmentach aplikacji w następujący sposób.

```
var mongoose = require( 'mongoose' );
var Post = mongoose.model('Post');
Post.findById(id, function(err, post){})
```

Pełna dokumentacja oraz przykłady użycia API Mongoose znajdują się pod adresem https://mongoosejs.com/docs/api.html