Kierunki rozwoju Oprogramowania

MEAN Stack

Odtworzenie projektu

Jeżeli skopiowałes swój projekt z ostatnich zajęć, aby zainstalować brakujące paczki wydaj polecenie

```
npm install
```

Po chwili powinieneś posiadać wszystkie zależności jakie zostały dodane na poprzednim laboratorium.

REST API

W tym momencie powinieneś posiadać najważniejsze endpointy do operowania na postach, jednak bez ich implementacji.

Do testowania **API** wtyczki takiej jak RESTClient przy wysyłaniu danych metodą **POST** albo **PUT** sformatuj je w następujący sposób:

```
{"created_by": "Jarek", "text": "Hello World"}
```

Pamiętaj także o dodaniu odpowiedniego nagłówka, żeby typ danych był rozpoznany jako JSON:

```
Content-Type: application/json; charset=UTF-8
```

Żeby poprawnie dekodować informacje po stronie serwera w pliku aplikacji dodamy moduł **body-parser**, który ułatwi nam operowanie na requestach i ich zawartości.

```
// Załączanie modułów
var bodyParser = require('body-parser');
```

```
// Sekcja middleware
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
```

Pamiętaj, że dyrektywa **use** musi znaleźć się przed pierwszym użyciem parsera (w tym przypadku **api**). W przeciwnym przypadku nie otrzymasz błędu a jedynie puste ciało zapytania lub w przypadku dostępu do pól **undefined** jako wartość.

Teraz przeniesiemy się do modułu obsługującego API. Do zapisu nowego posta w bazie użyjemy funkcji **save** która jako argument przyjmuje **callback** z 2 argumentami - pierwszym z nich jest ewentualny błąd, drugim obiekt dodany do bazy (w razie powodzenia).

```
post = new Post()
post.message = req.body.created_by;
post.text = req.body.text;
post.save(function(err, post){
  if(err)
    res.send(err);
  res.json(post);
});
```

Teraz dodaj kilka postów za pomocą wtyczki RESTClient bądź narzędzia **curl**

```
curl -X POST -H 'Content-Type: application/json; \
  charset=UTF-8' -i 'http://127.0.0.1:3000/api/posts' \
  --data '{"created_by": "Jarek", "text": "hello"}'
```

Sprawdź czy posty dodały się do bazy odwiedzając stronę **api/posts**. Jako odpowiedź powinieneś otrzymać tablicę postów z bazy danych. Jeżeli jeszcze nie stworzyłeś odpowiedniej metody dla tego endpointu poniżej znajduje się implementacja.

```
Post.find({}, function(err, posts){
  if(err)
    return res.send(err);
  return res.json({'posts' : posts});
  });
```

Zaimplementuj analogiczne metody (**GET**, **PUT**, **DELETE**) dla endpointu **api/posts/:id**. Wartość id pobierzesz z **req.params.id**. Do pobrania pojedynczego postu użyj funkcji **findById**.

```
Post.findById(req.params.id, function(err, post){})
```

Przy usuwaniu posta użyj funkcji **remove**. Pierwszym argumentem w tym przypadku będzie słownik identyfikujący obiekty do ususnięcia

```
Post.remove({_id: req.params.id}, function(err){})
```

Authentication

Dodamy teraz możliwość uwierzytelniania użytkowników.
W tym celu musimy dodać dodatkowe moduły - express-session, passport, passport-local i bcrypt-nodejs

Implementacja modułu do uwierzytelniania została przygotowana wcześniej i znajduje się w materiałach do laboratorium. Wypakuj pliki do odpowiednich katalogów.

W pliku głównym serwera importujemy wymagane moduły.

```
var session = require('express-session');
var passport = require('passport');
var auth = require('./routes/auth')(passport);
```

Ostatnie polecenie spowoduje, że do endpointów z modułu **auth** został podczepiony **passport**. Jest to middleware, który pomoże nam w uwierzytelnianiu. Żeby poprawnie działał należy na początku listy middleware (wszystkie polecenia **app.use(…)**) zainicjalizować sesje i passport oraz dodać informację o endpointach w module **auth**

```
app.use(session({secret: 'secret token'}));
app.use(passport.initialize());
app.use(passport.session());

var initPassport = require('./config/passport-init');
initPassport(passport);

app.use('/auth', auth);
```

Pamiętaj, że schema **Users** musi być już dostępna w momencie kożystania z modułu **passport**.

Możesz przetestować logowanie dowolnym narzędziem (najpierw utwórz użytkownika, potem możesz się zalogować). Przy wylogowywaniu możesz otrzymać błąd 404 ze względu, że jesteś przekierowywany na ścieżkę */*. Dodaj ścieżkę obsługująca ten endpoint. Będzie ona wyświetlała stronę główną aplikacji (plik dostępny w materiałach).

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

Należy jeszcze umieścić informację o lokalizacji plików statycznych (katalog public)

```
var path = require('path');
app.use(express.static(path.join(__dirname, 'public')));
```

Integracja klienta i serwera

Zacznijmy od dodania w globalnym scopie funkcji do wylogowywania użytkownika

```
var app = angular.module('cwierkaczApp').
run(function($rootScope, $http) {
    $rootScope.authenticated = false;
    $rootScope.current_user = '';

    $rootScope.logout = function(){
        $http.get('auth/signout');
        $rootScope.authenticated = false;
        $rootScope.current_user = '';
    };
});
```

Zwróć uwagę na obiekt **$http**. Posłuży nam on za chwilę do wysyłania zapytań do serwera.

Następnie dodamy pozostałe funkcjonalności (login, register, post). Zaczniemy od logowania i rejestracji, które zostaną zdefiniowane w **authController**, który zostanie dodany na te potrzeby w pliku **cwierkaczApp.js**

```
app.controller('authController',
function($scope, $http, $rootScope, $location){
  $scope.user = {username: '', password: ''};
  $scope.error_message = '';
});
```

Definicje funkcji odpowiadającej za logowanie wygląda następująco (umieść ją wewnątrz kontrolera to autentykacji):

```
$scope.login = function(){
    $http.post('/auth/login', $scope.user)
    .then(function(response){
      if(response.data.state == 'success'){
        $rootScope.authenticated = true;
        $rootScope.current_user = response.data.user.username;
        $location.path('/');
      }
      else{
        $scope.error_message = response.data.message;
      }
    });
  };
```

Dodaj definicje funkcji do rejestracji użytkownika i umieść ją w tym samym miejscu co funkcja do logowania.

Factory

Skorzystamy teraz z bardzo wygodnego mechanizmu, który zadziała dzięki temu, że dostęp do zasobów (Posty) został zrealizowany poprzez REST API.

W tym celu musimy wstrzyknąć do aplikacji dodatkową zależność.

```
var app = angular.module('cwierkaczApp',
  ['ngRoute', 'ngResource'])
```

Pierwsza z nich pozwoli nam na dynamiczne modyfikowanie zawartości strony na podstawie szablonów natomiast druga pozwala na korzystanie z zasobów.

Aby móc korzystać w naszej aplikacji z Postów jako zasobów użyjemy fabryki.

```
app.factory('postService', function($resource){
    return $resource('/api/posts/:id');
});
```

Powyższy fragment kodu umieść przed definicją głównego kontrolera. Wewnątrz kontrolera musimy jeszcze zdefiniować funkcję do tworzenia nowego posta oraz zainicjalizować listę postów tymi prawdziwymi.

```
app.controller('mainController',
function(postService, $scope, $rootScope){
  $scope.posts = postService.query();
  ...
});
```

Gdzieś w miejsce trzykropka wstawimy metodą do tworzenia postów (należy ją podpiąć pod przycisk do wysyłania posta w pliku HTML).

```
$scope.post = function() {
    $scope.newPost.created_by = $rootScope.current_user;
    $scope.newPost.created_at = Date.now();
    postService.save($scope.newPost, function(){
        $scope.posts = postService.query();
        $scope.newPost = {created_by: '', text: '', created_at: ''};
    });
};
```

Widoki

Ostatnim etapem będzie połączenie wszystkiego w działającą pełnoprawną aplikację. Plik który znajduje się w materiałach do laboratorium w folderze **public** jest bazowym szablonem z podpiętymi wszystkimi akcjami.

Miejsce oznaczone dyrektywą **ng-view** będzie uzupełniane odpowiednio przygotowanymi widokami (podmieniane).
To czym będzie podmienianie zdefiniujemy w pliku **cwierkaczApp.js**

```
app.config(function($routeProvider){
    $routeProvider
        .when('/', {
            templateUrl: 'main.html', controller: 'mainController'
        })
        .when('/login', {
            templateUrl: 'login.html', controller: 'authController'
        })
        .when('/register', {
            templateUrl: 'register.html', controller: 'authController'
        });
```

```
});
```

Powyższy kod należy umieścić poza definicją kontrolerów.
Zauważ, że dla każdego widoku możemy (i to robimy) zdefiniować inny kontroler, który będzie zajmował się realizacją pewnej logiki. Drugim przekazywanym argumentem jest plik z widokiem, który będzie wstrzykiwany.
Pliki te możesz stworzyć w oparciu o wyniki z poprzedniej części laboratorium (po prostu wytnij odpowiedni fragment kodu)