WWW-Programming coursework 10 ECT

university of tampere

Oskari Niskanen

428246

2019

**Introduction**

The coursework is a full stack application, which functions as an administration system for a movie club. It features a backend server with server generated web page for front end, and JSON REST API which is accessed through the static version of the web page (Made with React using CDN). On the following chapters, I will introduce what requirements there were and how my application handles them. Coursework was done alone.

**Instructions for usage**

First off, you need to boot up your vagrant virtual server and head to the folder that I supplied with this coursework. There you need to run command **npm install**. After that you might have to boot up MongoDB server with the command **sudo service mongod start.** After you know that the database server is up, locate the main folder of the app and run command **mongoimport --jsonArray --db users --collection users --file users.json** and it should populate the MongoDB database with some users. After you’re done, you can run command **node app.js** and the server should boot up. Then head to <http://localhost:3000> and you have arrived at the server generated website. On the login screen there is a link to the statically served website or you can just visit <http://localhost:3000/static/index.html> to go there directly. I’ve prepared a test user for you, so you can login with that if you want or make a new account.

**Email**: admin@weto.com

**Password:** 123456

Any questions or problems can be sent to this email:

**Oskari.niskanen@tuni.fi**

**Requirements**

* **The coursework must work with the virtual machine that is created with the provided Vagrantfile**
  + The application runs perfectly inside the virtual machine
* **A single coherent application (not a set of completely separate functionalities) written with Node.js and Express** 
  + It is a coherent application and the parts work together to achieve the result.
* **Your application must store its data in a database and use an ODM/ORM (e.g. Mongoose or Sequelize) to access it.**
  + MongoDB/Mongoose combo is used to store objects in the database
* **You will need to have user accounts and roles for users**
  + There are currently 3 roles in the application, which are: Admin, Basic and unregistered user. Each of these have different permissions and functionalities. Admin can edit/delete everyone and set their roles/password/email/payment status.
  + Registered users can edit their own data, except Basic users cannot edit membership payment status nor their role. They can also unregister from the system.
  + Unregistered users are only allowed to see event list without locations and to register.
  + Each user starts as basic user, unless “Admin” role is chosen when registering from server generated web page. I left the option there to choose to register as an Admin, in case Admin deletes themselves.
* **Use a MVC style structure**
  + MVC style structure is being used as the application consist of Model, View and Controller files. It also has separate routing.
* **The application must be safe (Take care of at least XSS, CSRF, SQL/NoSQL Injections)**
  + XSS is being taken care of with the usage of Helmet middleware.
  + CSRF authentication is being taken care of with the csurf-middleware
  + NoSQL injections are taken care of with content-filter middleware, which checks for forbidden characters e.g $ or {
* **Data inputs must be validated**
  + Data inputs are validated with the express-validator/check – middleware. Empty or invalid inputs will get denied.
* **Use HTML verbs correctly**
  + **CRUD**  methods are being used in the proper way, except form data can only GET or POST, so in the server generated version POST is used instead of PATCH when updating user data.
* **Return correct status codes**
  + Correct status codes are being returned in case there are some errors. Error messages are somewhat scarce, because it provides better security when the application doesn’t tell what exactly went wrong. This is the case especially when logging in or trying to access unauthorized information.
* **You must use a template engine (e.g. handlebars, but others are also accepted)**
  + Handlebars is being used when generating the server-side webpage.
* **Use comments in your code**
  + Code is commented throughoutly.
* **You should store passwords using bcrypt**
  + Everyone’s passwords are safely encrypted with bcrypt.
* **Add a JSON REST API to your server.**
  + Server contains fully fledged JSON REST API, which can be accessed through Postman. Static web page communicates with the API to handle its functions.
* **Create a single page application with React that has at least the same functionality as your server side rendered application.**
  + The single page application is served as static content and contains the same functionalities as the server side rendered application. It is made with React using CDN.
* **Use JSON web tokens for handling authentication**
  + JSON web tokens are used to handle authentication with the REST API and only getting events and logging in is possible without the token. This token is sent with each request after receiving it and server verifies it before routing it the request to the controller.
* **Your React application should use Redux to handle the state**
  + I did not implement Redux into my application.
* **Your single page application uses AJAX calls to communicate with the server**
  + Communication between the static web page and the server is done by using fetch API.
* **The system should be usable through the server generated web page and the static SPA**
  + This works as intended and the users can access the site through both ways. To access server generated site the URL is http://localhost:3000 and when accessing the static web page, the URL is <http://localhost:3000/static/index.html>
* **Create some way to easily add some initial data**
  + A file named **insertion.js** can be run by using command “node insertion.js**”** in terminal and the data will be added to the database. After that, the server can be started with the command “node server.js”