WWW Programming 2019 – Coursework

Discussion forum

Topi Nieminen (424727)
topi.nieminen@tuni.fi
https://github.com/toniemin/www-coursework

This is a short document describing how the coursework requirements were met. Full list of requirements can be found on page 2. This coursework was originally going to be an extended 10 ECTS coursework, but was cut to the 5 ECTS version in the end. Page 3 still has 10 ECTS requirements listed. The coursework fails the 10 ECTS requirements by not having a single page application made using React and Redux.

Planned for the 5 ECTS version as well, the project was going to have a discussion forum built-in, but was cut. The server-side contains some functionality for the discussion functionality (database can store discussions and comments), but no actual UI exists.

The database can be initiated by running 'node databaseInitializor" in the root directory of the project.

The permissions are implemented with objects called 'actions' that contain 3 things: a) path that is allowed e.g. 'api/users', b) the HTTP verb that is allowed on that path e.g. 'GET', and c) attributes (read document fields) the user can access. E.g. the basic user can only access the 'username' fields of the users in the database. The action objects are heavily inspired by the REST API. The permissions are objects that contain a list of these actions. Each role has a permission object associated with it. The full list of current permissions can be found in 'permissions.json'. The permissions file is given to the database initializor script to be saved to the database. The actual user-access-control is done using DYI access-control middleware with json web tokens.

Requirements (5 ECTS):

- The coursework must work with the virtual machine that is created with the provided Vagrantfile **OK, works with the vagrant file (downloaded at the start of the course).**
- A single coherent application (not a set of completely separate functionalities) written with Node.js and Express **OK, all works by running "node app".**
- Your application must store its data in a database and use an ODM/ORM (e.g. Mongoose or Sequelize) to access it:
    - Mongo DB or MySQL (this is so that the application will work with the virtual machine that is created with the provided Vagrantfile) **OK, using Mongo DB with Mongoose.**
- You will need to have user accounts and roles for users **OK, there are 5 roles: admin, moderator, member (membership fee payed), member (fee not payed) and unregistered.**
    - Each user should be able to register to the system and remove themselves from the system **OK, users can register and delete themselves from the user settings.**
    - Each registered user should be able to modify their own data **OK, users can change their password and email through the user settings. They can also "pay their membership fee" to use the system.**
    - Each registered user should have a role or multiple roles which represent a certain set of permissions **OK, all roles have a collection of actions they can take**
        - The minimum is that there is a user role and administrator role **OK**
        - These are set by the administrator (e.g. an admin can make a user an admin) **OK**
        - Each user starts with a certain basic role **OK**
    - An administrator can modify/remove the data of any user, and also remove users **OK**
- Use an MVC style structure **OK**
- The application must be safe (Take care of at least XSS, CSRF, SQL/NoSQL Injections) **OK? Using only helmet-middleware.**
- Data input must be validated **OK, mongoose schemas and the controllers contain validation**
- Use HTML verbs correctly **OK**
- Return correct status codes **OK**
- You must use a template engine (e.g. handlebars, but others are also accepted) **OK, using 'express-handlebars'.**
- The UI can be very simple **OK, very simple indeed**
- Use comments in your code **OK, I tried to comment what I thought as important.**

Requirements (10 ECTS):

- The **full** requirements of the 5 ECTS coursework. **OK**
    - Read the requirements and description from the 5 ECTS Coursework page.
- You should store passwords using bcrypt **OK**
- Add a JSON REST API to your server., **OK, minimal JSON rest api implemented**
- Create a single page application with React that has at least the same functionality as your server side rendered application. **FAIL, no React application implemented**
    - Your server should serve this application as static content. **FAIL, no application**
        - You may use Create React App or have the files use a CDN
    - Use JSON web tokens for handling authentication **OK, uses custom middleware**
    - Your React application should use Redux to handle the state (the coursework can be accepted even if this point is not implemented, but it will effect grading!) **FAIL**
    - You can use libraries etc. However, if they automate things too much, it will be taken into account when grading. **OK, not many libraries were used**
    - Your single page application uses AJAX calls to communicate with the server. **FAIL, no application**
    - The system should be usable through both:
        - The single page application (created for 10 ECTS only). **FAIL**
        - The view generated by a template engine (created for 5 ECTS). **OK**