CT30A3204 Advanced Web Applications

Project Work Documentation

Joonas Liedes

# Technology Choices

For the backend I decided to implement a standard Node.js server using the Express framework, as this was covered in-depth on the course and is also widely used in the industry. For similar reasons I chose to use MongoDB as the database for this project. The backend is organised in a way that separates the API routes, API functionality, database schemas and the middleware to their respective folders for better readability and maintainability.

The app allows the user to register, login and post coding related questions and comment on them. In addition, non-authenticated users can read the posts and comments but are not allowed to post or comment. I decided to implement the authentication with the passport.js library and json web tokens due to its simplicity and ease of use. There are also several tutorials on them, which makes their use easily approachable. I then protected the routes related to posting with the tokens acquired from the authentication. When building the database model for the user, I opted to encapsulate the password encryption and verification inside the model to separate it from the API logic and thus make the code more robust.

Similar to the backend, technology choices for the frontend were influenced heavily by the course content and the industry standards and for that reason I chose to use React as the frontend framework. The modularity of Reacts component-based way of implementing frontends adapts very well to different use cases and makes building fast. Moreover, it’s a lightweight framework compared to Angular for example, and is therefore a suitable tool for this relatively small project. As is often the case, the front end comprises of React components and pages using these components and reside in their respective folders. In addition, I’ve separated the frontend logic to separate folders and files to improve readability.

I chose to use Axios for the API-calls in the frontend due to its popularity and for the fact that I’ve only used the Fetch-API previously. This proved to be a good decision, as it was ease to use and quick to implement. For state management I chose to learn and use Redux. For a small-scale project such as this, a robust library like Redux probably wouldn’t be necessary, as there is a lot of boiler plate code. However, because it’s widely used in the web development industry, I saw this as an opportunity to learn it for future use. In combination with the Redux browser extension, it proved to be an excellent tool. As the apps main functionality is to post coding related problems with code snippets, I used highlight.js library for syntax highlighting the posts. For styling the frontend I used custom CSS, because I believed that I needed the learning experience, as I’ve not written much of it from scratch. For future reference, libraries like Bootstrap or Materialize would be the way to go when it comes to responsive design.

In summary, I used the very popular MERN-stack for this project as its widely used and has been proven to be very robust in building full-stack web applications. Due to its popularity, There are loads of great tutorials and documentation on it, which make building with it efficient and easy. Design choices and guidelines for implementation of this project came mainly from the course materials and the content of Traversy Media Youtube-channel, especially the material covering the MERN-stack.

# Installation Guidelines

[See ReadMe at GitHub repository](https://github.com/j00lie/ct30a3204_project)

# User Manual

[See ReadMe at GitHub repository](https://github.com/j00lie/ct30a3204_project)

* Non-authenticated users can:
  + Read posts and comments
  + Register
    - By clicking on the “Register” button in the header
  + Login
    - By clicking on the “Login” button in the header
  + Logout
    - By clicking on the “Logout” button in the header
* In addition, an authenticated user can:
  + Post questions
    - By clicking on the “Ask a question” button on the index page
  + Comment
    - Comment on posts by selecting a post from the index page and then submitting a comment by clicking “Post Comment”

# Implemented Features

|  |  |
| --- | --- |
| **Feature** | **Max points** |
| Basic features (as stated in the previous chapter) with well written documentation | 25 |
| Utilization of a frontside framework, React. | 5 |
| Use some highlight library for the code snippets, for example https://highlightjs.org/ | 2 |
| Last edited timestamp is stored and shown with posts/comments | 2 |
| Your own feature: Implemented state management with Redux. I feel this is worth the points because of the steep learning curve and the hours that I had to put in to learn it and the Redux browser tools. | 4 |
| **Total** | **38** |

# Notes

I also implemented the update- and delete-routes in the backend for editing and removing the users posts but didn’t have the time to finish them on the frontend due to a hectic work schedule.