CT30A3203 Web Applications – Project work Documentation

Niklas Kumpulainen

Niklas.kumpulainen@student.lut.fi

Student Number: 0567737

LUT

17.12.2021

Introduction

This project was done as a proof of study and a demonstration of acquired skills for the course: “CT30A3203 Web Applications “. The code should have summary of functionality for every module or function as comments to make it readable. As a note the code can be crude at places even though I have tried to minimalize it to the best of my abilities taken account of the time restraints. Header information can be seen also in the repository readme and commented in the project’s client’s “App.js” file.

Technology choices

The project was created as a “full stack” project with the client and server being accessible from the same directory. There were several different technologies utilized in the project from which the most defining ones will be covered as well as some noteworthy dependencies.

The front end was done with React framework. This seemed as an obvious choice for me because the framework was practiced during the exercises and the lecture videos are really informative on the basics. Also, the framework itself is quite easy to get around and it can offer a great workflow boost with the creation of components with combination of JavaScript and JSX.

The server-side backend was created with Nodejs and Express framework. The usage of the before mentioned technologies were mostly justified because they were the most familiar for me. Also, the extensive amount of external information as well as the information from course lectures were more than enough to meet the requirements for this project assignment.

As a database and database system, the project utilizes the functionalities of “mongoDB” and its JavaScript framework “mongoose”. Mongoose is easy to use in the server backend and there has been a lot of exercises as well as lecture videos on the subject matter which is why they have been used for handling the leg work needed for database usage.

Installation

The application has been tested in development locally so should the running of the application be done as well (This was not specified in the description of the project work so I am assuming this is acceptable).

First step for installation should be to download the file from the GitHub repository (where this word document is also in). You should extract the zip to a folder where the “Fullstack-project” file is the root of the application.

After this you should open the project in windows PowerShell so that the current folder open is “Fullstack-project”. Then you should run command: “**npm install**”.

The commands are defined in “package.json” file in the root folder but ultimately this will install all the required dependencies. This should be enough to run the project, at least according to my testing.

Once you have installed that, **you will have to create a file named “.env”** in the folder named server (full path: “Fullstack-project/server/”). **Otherwise, the login validation will not work and the application won't work either**. The “.env” file should contain one string and one string only: “SECRET=[random letters and characters of your choosing]” just like in the lecture video 14.

(*This problem came up during submission for which I haven’t found a solution to. GitHub repository (for a good reason) hides or doesn’t allow “.env” file that was originally in the files. I'm terribly sorry for this and hope the project can be still graded)*

Running the project in the root folder works by typing “**npm run dev**” in PowerShell.

Installation was done according to course exercise instructions and was tested to work with systems specifications as follows:

- Windows 10
- node 12

Guidelines/user manual

This section covers the questions: “how to reach a mentioned feature” and “what does the system have to offer”. Main features that might need guiding towards will be covered here. It needs to be noted that the application is set to be reachable from the port 3000 so to get to the index page, one should go to the following address: http://localhost:3000/ . That being said, the guidelines might simplify or leave out some prefixes so keep the link in mind and use it for reference. It needs to be noted that if some functionality seems to do nothing, try logging out and logging in again. The expiration of Json web token is set to 20 minutes but sometimes this might have not been taken into account.

Home/index

- As mentioned before the home or the index page is reachable from the URL address: http://localhost:3000/ as well as the hyperlink named “home” at the top of the page header.

Login/Register:

- Login and register features can be found at the top of the page header through namely the hyperlinks (light blue/navy) background. They are also reachable from the URL: (localhost prefix) + “/login”, and “/register”

Log out:

- User can log out ultimately ending the session and removing authorization token
- User can log out by clicking the “Log Out” hyper link at the top of the page in the header.

Post listing:

- This feature shows all the created posts and their respective data summarized.
- Not required to log in
- Can be found in the top of the page header through hyper link named “Posts”. Feature is also reachable from the URL: "localhost:3000/posts”
- Note: as the database is empty when starting, post listing view will be empty as well. Get started by clicking “Create Post” button after you have registered and logged in

Create a post:

- By clicking the “Create Post” button or by going to the URL: “localhost:3000/posts/add”

Post View:

- This feature is shows expanded details of a single post element from the post listing.
- Post View is reachable BY CLICKING POST LISTING ELEMENT. This should be emphasised because it might not be obvious.
- In Post view an authenticated user can:
    - Comment
        - Write in the text area below “Comment:”
        - press “Comment” button to send the comment
    - Up vote/ down vote
        - Press the “Up vote” or “Down vote” buttons
        - previously pressed button will be underlined for clarity
    - Edit or remove own comment
        - By clicking the comment’s text, which turns the text into a text area, one can edit their own comment
        - By clicking the delete button (showing only on own comments) one can delete their own comment
    - Edit or remove own post
        - By pressing the “Edit” Button below your post in Post View you can enter the “Edit mode” to edit your post’s content
        - In “Edit mode”, pressing the “Cancel” button will cancel the editing done to the post
        - In “Edit mode”, pressing the “Save” button will save the editing done to the post
        - In “Edit mode”, pressing the “Delete” button will delete the post from the database
- In Post view a non-authenticated user can:

    - See post content
    - See comments
    - See upvotes

Admin account:

- An admin account is created when entering the login screen as the user account's table in the database is empty. You can for demonstration purposes log into the account by using username: "admin" and password: "12345". This is of course not smart to do in production of any kind.

Features and grading

In this section the mandatory and all the additional requirements and features are listed with explanations and possible grade requests. Explanations are there to point out details how the feature manifests or just reasoning for the point amount requested.

Mandatory features

Feature	Explanation
Login and Register	The system offers login and registration possibilities for the user. The user's data is saved into database securely. Feature can be reached from link in the website header, or from "/register" or "/login"
Utilization of database	Database is utilized for handling required models such as user accounts and posts on the website.
Implementation of backend with Node.js	The project uses Node.js specifically with Express framework to handle backend API calls.
Authentication	Several of the critical features require authentication from backend validation (and frontend conditional rendering solutions for less critical features). Authorization is verified with JWT (jsonwebtoken) library in the back end for all the crucial operations like login, addition and removal of database items. -Only authenticated users can post new code snippets, comment on existing posts and vote -Non authenticated users can see posts, comments and vote counts
Creation, editing and the Listing of database elements	All the posts inside the database are listed in "posts" view or "/posts" which can be clicked to show all the information and to vote and comment. Users and admin can edit and delete their own posts and comments.
Responsive design	The page layout is designed to always scale to the width of the screen with the maximum of 1000 pixels there are no static

	elements and every needed element is accessible on smaller screens

Additional features

Feature	Grade request	Explanation
User can edit their own comments/posts	4/4	Users can edit their posts in the "single post view" or "/posts/:id" by pressing the Edit button. Users can also edit their own comments in a post by clicking the comment text, this switches the text into a text area to type in.
Utilization of a frontside framework	5/5	The project work was made as a "full stack" project with React working and utilized fully in the frontend and nodejs with Express in the backend.
Admin account with rights to edit all the posts...	3/3	Admin account is created into the database if empty for demonstration purposes on the load of login page. Admin accounts username is "admin" and password is "12345". Admin account can edit and remove any post or comment. Comments are an array of objects inside the Post model, so comments are deleted when a post is deleted.
Vote (up or down) posts and comments (only one vote per user)	1/3	Users can vote for posts and only once. The reason for requesting only 1 point is as follows: - Comments cannot be upvoted or downvoted - rollbacking votes works but the communication of the system to user is lacking.
Last edited timestamp is stored and shown with posts/comments	1/2	Requesting 1 out of 2 points. Timestamps can be seen in posts list view and single post view alongside with the headers. Timestamps are stored when creating or editing posts and comments, but the timestamps do not communicate that they were edited.

Grading

As the mandatory features are in my opinion fulfilled which would yield the minimum 25 points and the sum of the additional features implemented is 14, I would propose that the project work is worth at least 34 (25+14) points out of 50. For the effort put in to the project and the time spent I only feel comfortable asking this amount.