ADVANCED WEB APPLICATIONS

Project

Atte Hiltunen

LUT-university

1. Guide to start up the servers.

1. Download the repository from GitHub https://github.com/AtteHil/AWA_project
2. Navigate to folders /client and /server with two different git bash windows.
3. Write command “npm install” on both and press enter.
4. Make file “.env” to your /server folder and input this “SECRET=OwnSecretKey”
5. Make sure you have MongoDB open and running in port mongodb://127.0.0.1:27017
6. After downloading required packages, you can write command “npm run dev” in /client side bash
7. Run command “npm run start-dev” on /server side bash.
8. Server should be now up and running at http://localhost:5173/

I made the project with react and TypeScript. I wanted to learn more of these technologies which were not so familiar to me before. Wanted to learn new and improve my skills.

You can create example users with test, but these users do not like anyone on default. You have to create yourself at least 2 users and like both ways to be able to see and use chats.

2. Tests

I set some tests to /test folder which can be ran by firstly installing packages in /test folder. Backend needs to be running for the tests front doesn’t. there are only 5 tests + populating database with 10 sample users’ function. (therefore, I don’t take 5 points from tests). Test are done with plain JavaScript.

run bash commands.

“cd /test”

“npm install”

“node test.js”

This will return print array of results 0 is pass and if -1 appears file will tell what test failed.

3. Backend

(/server folder which has API endpoints and MongoDB interface)

Whole backend is using purely TypeScript and MongoDB is used as database for storing data. Backend utilizes RESTful API that is known from course materials and weekly tasks.

Bcrypt is used for hashing the password and hash is stored into the database.

Mongoose is used to handle data between backend and MongoDB. Simple find functions (find, findByID etc.) are commonly used in the code to find different data from database. Mongoose schemas I use can be found from models folder under/server folder.

For server side I chose to use Express. This was the most familiar choice and easiest to use for the hosting.

Authenticating the user passport jwt is used. This is familiar from courses weekly tasks and was the best option. Token is set to be alive for 30 minutes. Then it will not work anymore, and user must log in again.

As email and password validation I chose to use express-validator for its simplicity. Email is checked to be email form and password is set to be at least 8 characters long, at least 1 number, one symbol and one upper- and lowercase characters.

Api endpoints speak for themselves the endpoint address describes what is it used for. For example, /Login is normally for login purposes and /message is for pushing new message to database.

Updating user, you must fill every field there is (old data except password is automatically filled). Update function also updates matches arrays usernames so your matches see new username also.

4. Frontend

Sites front is done with react entirely. Most of the responsiveness of the site is made with css, but header responsiveness is done in header component (because it is made with app bar styles).

Different components are in components folder, and they have their own purpose on different pages. In this project I decided to go with component per page style.

Swipe action is done with react-tinder-card (https://www.npmjs.com/package/react-tinder-card) so user can swipe to left or right whether they dislike or like the person.

Header which is done with material-ui app bar is set on every page(https://mui.com/material-ui/react-app-bar/). Also, background image is set in header which was done with image creator AI by Microsoft.

Chat component uses react-chat-elements to display the messages that are fetched from the database (https://www.npmjs.com/package/react-chat-elements). Chat is also set with buttons that appear for each match and they change the index of which cha is currently visible on the component. There is a refresh button to fetch new messages from database.

Changing language is made with i18n library. This is changing the language from app bar buttons. This is also familiar from weekly tasks and in the public/locales are different language packets. Bio is not translated on purpose because it is also meaning same thing in Finnish. Also, alert messages take language property from i18n and based on the current language alert the correct messages.

Registration password requirements are shown when mouse enters and toggled of when mouse leaves (On mobile works when password field is pressed requirements come visible). Help to mouse hover is looked from this website: https://plainenglish.io/blog/how-to-handle-mouse-hover-events-in-react

5. Features and Points

Feature	**Points (I think I deserve)**
Basic features	25
Utilization of a frontside framework, such as React.	5
Project made with TypeScript (Had to learn a lot during project so I think it deserves some points)	3
One can swipe to left or right to dislike or like the profile (works with mouse but better with touchscreen)	2
If match is being found the UI gives option to start chat immediately (alert with current language is shown to user)	2
Last edited timestamp is stored and shown within chat (can be seen on every message)	2
Translation of the whole UI in two or more languages (English and Finnish)	2
Created unit tests (5 + populating database with sample users)	3
Using chat elements and making buttons for each chat (buttons have users name and they update index upon pressed. They are visible on the chat page)	2
Using password and email validation (this was not required but wanted to make that users use correct credentials. Made with express-validator to backend)	2
Password requirements show on mouse hover	1
CSS and styling of the site (site is nicer to use when it looks good and is themed. Each react component has own css)	2
Total	**51**