Documentation

CT30A3204 Advanced Web Applications - Final Project

Moraga Hernández, Marta Xiaoyang

Technology choices

- Back-end: Express
- Front-end: React
- Database: MongoDB
- Responsive design: MUI components
- Authentication: session based, using Passport.js

Installation guidelines

- After downloading the repository it is necessary to download the dependencies, from the root folder run the commands:
  npm run preinstall
  npm run install
- Build the react application (can be skipped if the intention is to run on development options) from root folder:
  npm run build
- On MongoDB **establish the connection to mongodb://localhost:27017**
  - The application, when used, will create a database called finalproject and three collections: comments, posts and users.
  - If it does not, they can be created manually.
- (Optional) In the project folder there is a folder called collections, with three files: comments.json, posts.json and users.json. They can be imported into mongoDB, so that it is easier to test the application. The accounts have easy passwords that do not follow normal restrictions:
  username: alwaysFirst, password: 1
  username: claramente, password: 123456aA
  username: htmlIsDifficult, password: 1
  username: secondBest, password: 1
  username: Karl1980, password: 123456aA
- If react application was built, start the application: (it will be on http://localhost:1234/)
  npm start
- If the intention is to run on dev options, it is necessary to set NODE_ENV to development and run on dev: (server will be on http://localhost:1234/)
  NODE_ENV=development npm run dev:server
  - And, in another command window: (client will be on http://localhost:3000/)
    npm run dev:client

User manual

Where is it? Once it has been started is on http://localhost:1234/
(or {client: http://localhost:3000/ server: http://localhost:1234/} if the variable NODE_ENV === "development")

What can it do?

Explanations are commented on the code, this is a just a summary

Client

Components' files and App.js are all commented

Routes:

- ➔ */* - Index page where one can see the posts, each post title leads to their detail's page
- ➔ */login* - Login page
- ➔ */register* - Register page, password should be strong (min 8 characters, and include at least: 1 Uppercase, 1 Lowercase, 1 Number, 1 Symbol. Example: 123456aA.)
- ➔ */post/post_id* - Page that show the details of certain post, if logged in, an option to write a comment appears
- ➔ */write* - Page to write a post

The Header appears on all pages.

If logged in: Posts, Write Post, Logout buttons appear.
If not logged in: Posts, Login, Register appear.

Server

app.js, passport-config.js and the models's files (Comment.js, Post.js, User.js) are all commented

API routes:

- ➔ */api/user/register* - POST To register an user, sends msg if user invalid with reason (adds to database)
- ➔ */api/user/login* - POST To login
- ➔ */api/user/logout* - POST To logout, cannot logout if not logged in
- ➔ */api/user/logged_in* - GET To let the server check if the session is valid
- ➔ */api/post/create* - POST To create a post (adds to database)
- ➔ */api/post/:post_id/comment/create* - POST To write a comment to a post (adds to database)
- ➔ */api/post/:post_id/details* - GET To fetch the details of a post (info and comments)
- ➔ */api/posts* - GET To fetch all the post and their basic info (comments not included)

Features implemented

The features implemented are the basic ones:

- Backend with Node.js (Express)
- Utilisation of a database (MongoDB)
- Authentication (using Passport.js)
- Everyone can see post and comments

- Only authenticated users can post and comment
- Responsive design (MUI components)

Plus:

- Utilisation of frontside framework (React) (5 mas points)

The basic features (25 max points) and the extra one (5 max points) are correctly implemented. There are not a lot of features, but the ones implemented work, therefore the points I expect are 30. Unless there is something that does not work correctly.

Some screenshots of how it should look: