Core dumped exception.

Technology choices.

Backend.

For the backend I used nodejs engine to run javascript on the backend, in addition to nodejs I selected express framework as it is the framework that I am most familiar with after this course. For simplicity I decided to use javascript in stead of typescript.

I decided to choose mangodb to hold my database as it is the simplest and most effective way I know.

Front-end.

For the front end I had different options, I discarded using vanilla javascript with a view engine as I wanted to create a spa. Therefore I had two main options React or Angular, In which I decided to use react as it is the newest technology and the technology I found more intuitive to use. I kept using javascript, using jsx in stead of tsx files.

Installation guidelines.

Clone or download the repository in your own machine, to run this project it is required to have installed in your machine node and node packet manager (npm), the application was built using node 16.19.0 and npm 8.19.3 so those are good. Once you have installed both node and npm, the next step is to go into the project folder, which has both the server and client folders, once you are there use npm I to install the necessary dependencies, after that, go into server folder and execute again npm i, finally go back to the main folder and go into the client folder and again execute npm i, after these steps you have successfully installed the project. Remember also to download mongodb for the database management.

Once you have installed everything go to the root folder (the one that has server and client), there you can now execute the command npm run dev which will start the development server in your own machine, client is held in port 3000, go to your browser and go to http://localhost:3000 , server is held in port 5000.

User manual.

First time you open the application everything will be empty, first thing that should do is create a new profile, that can be done with the register option in the navbar and fill the form and submit, after that you can Login and then create a new post using the ASK option. Afterwards you can comment in your own post, and after logging out the post will remain there. If you create a different account you will see that you can comment on others posts and see others profile, however you can only edit your own profile by adding a bio. From the profile you can see posts and comments and go to the post page or to the post page where the comment is located.

Features.

- Authenticated users can:
    - Post new code snippets
    - Comment on existing posts

- Non-authenticated users can see posts, comments (and vote counts)
- There is some page listing all the post, after opening one post, comments are also listed

- Responsive design
    - The app is usable with mobile devices and desktop browsers, this was achieved by adding a sidenavbar that will appear on smaller devices.
- All of above features gives 25 points.
- Utilization of a frontside framework, such as React, but you can also use Angular, Vue or some other (5)
- Use of a pager when there is more than 10 posts available (2)
- User can click username and see user profile page where name, register date, (user picture) and user bio is listed (2)
- Last edited timestamp is stored and shown with posts/comments, here not last edited but created on timestamp, as I did not implement the edition, therefore I think I could get one (1) point from this feature.

In total I expect 35 points.

Front-end development.

Components.

As the application is developed in react, I created several components to help myself while creating the application.

App component.

This is the main component, it always displays a responsive navbar (Nav component), I used react router to implement the pages.

```
<Routes>
  <Route path="/login" element={<Login setJwt={setJwt} setUser={setUser} jwt={jwt}/>} />
  <Route path="/register" element={<Register />} />
  <Route path="/" element={<Home user={user} jwt={jwt}/>} />
  <Route path="/post/ask" element={<Ask user={user} jwt={jwt}/>} />
  <Route path="/logout" element={<LogOut/>} />
  <Route path="/post/:id" element={<Post user={user} jwt={jwt}/>}/>
  <Route path="/profile/:username" element={<Profile />}/>
</Routes>
```

Those are the routes the Front-end side will use.

Ask component.

This component is formed by a form which will send a request to post a new code snippet.

Comment component.

This component will be used in the post component to allow users to post comments into posts.

Comments component.

Auxiliar component to list components in the home page and in the profile

Home component.

Component where the preview of the posts will be shown.

Login component.
Form to log in the app.

Logout component.
Logout of the app, deletes from the localstorage both the jwt and the user.

Nav component.
Navbar component, used for navigation, the responsiveness is achieved thank to materialize and i also got help from a video on youtube with link: https://www.youtube.com/watch?v=AhioxFWkYRg&ab_channel=WebZone and to implement it in react from https://medium.com/@elhamza90/react-materialize-sidenav-in-4-steps-7365f6176b09

Pagination component.
Got help doing this functionality from here https://www.youtube.com/watch?v=IYCa1F-OWmk&ab_channel=TraversyMedia

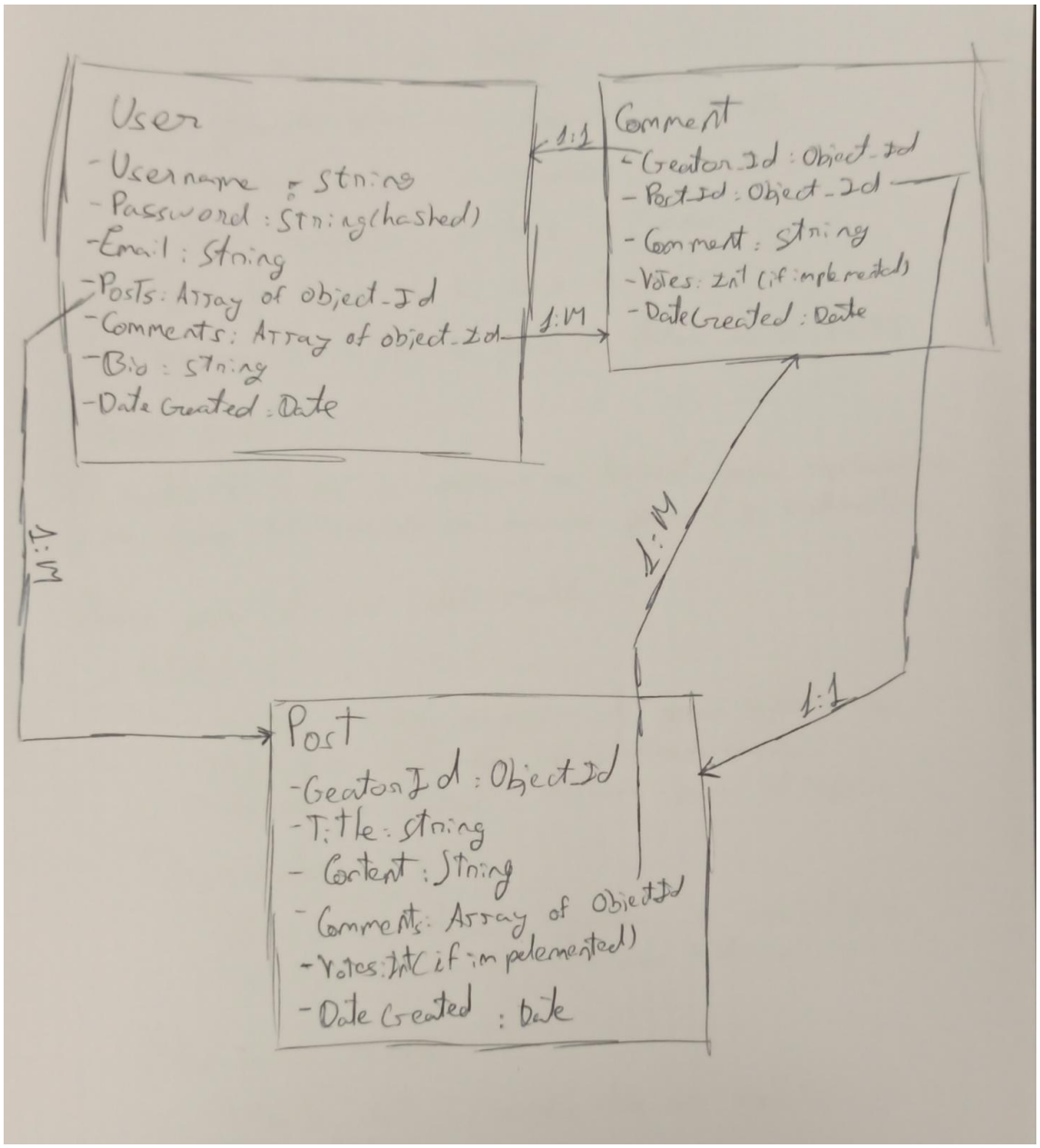
Post component.
Component that will display a full post with its comments.

PostPreview component.
Preview of the post that will be shown in home page

Profile component.
Profile of the user, will be able to submit a new bio. It also displays posts and comments of the user.

Register component.
Component that a user will use to register a new account.

Database.
This is the schema I followed for the database, each entity is a mongoDB collection:



Here we can see links between collections so we have an easier access.

Backend.

Every route used is commented properly in the backend.