45209723 Bradley Kenny

COMP3120 *Individual Web Project*

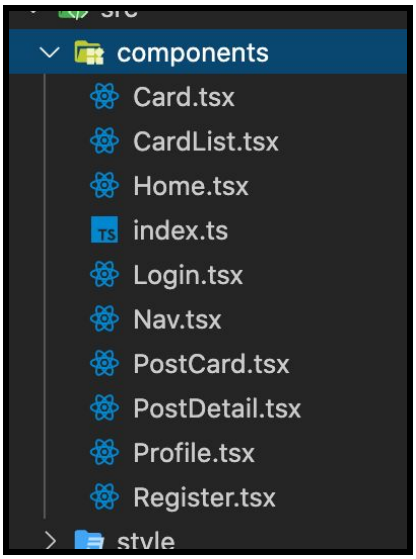
Link: https://mighty-anchorage-18501.herokuapp.com/

This project was based on creating a Twitter-like clone that is capable of supporting users, and having those users post messages on the application. The application had to support a lot of different features and have authentication to ensure that people were able to login and have their own profile.

For both parts of the project, I decided to implement them in TypeScript. I believe that overall this was beneficial to the development of the application as the static type checking enabled me to find bugs much quicker and prohibited the use of types where I shouldn't have been using them (JavaScript doesn't check this statically).

Frontend

The frontend of my application was built using React. By utilising this framework, I was able to quickly build user interface elements that were logicaly separated into components. These components were created to represent different parts of the user interface and heavily utilised React features such as state and props to correctly render information to the screen. The components were created in TypeScript which meant I was able to specify props using types. A screenshot of my components is below.



Furthermore, I was able to implement frontend routing to ensure that we could correctly render information to the screen as required. The frontend routing tool, `react-router` allows us to specify paths and then the application is able to determine that when we are at that path, we should render a specified component.

The frontend utilises JWTs to ensure that the user was authenticated correctly. Furthermore, we were able to use these to only show some components when the user was authenticated. This was also backed up by some backend checks but ultimately allowed us to hide UI elements to avoid user confusion.

By clicking on the ‘more’ part of each post, we are able to see a single page representing that post. This view would be useful for people that want to send a link to a particular post. In the future, we could add more advanced things to this page to enable them to edit, delete, or do other things to the post.

The frontend styling is based on the Bootstrap library. I used the tool `react-bootstrap` to be able to implement the Bootstrap elements and then used my own styling to give it a non-standard Boostrap look. This meant I was able to utilise the benefits of Bootstrap such as its grid system but also maintain my own look to the application.

Backend

The web application backend has been built off of Express and MongoDB. Express is the web application server that enables us to write backend code in JavaScript whereas MongoDB is the database instance that enables us to store data as documents online. For this part of the application, I continued to use TypeScript to enable the use of static typing in my development environment. I believe this enabled me to speed up development time and identify bugs a lot quicker in the backend development. However, as discussed in more detail in my *deployment* section, this did mean that I had to compile my application before being able to run it. This part was a bit tedious at times. However, I do believe it was worth having the static type checking support in the application.

The backend has been modularised to ensure that it is split logically for developers to understand where the logic for the application resides. The `db/` directory contains all the MongoDB models and the `controllers/` directory contains the routers for the Express server instance.

Within the db/ directory, I have specified my MongoDB models based on the information in the JSON file provided to us. I have used the Mongoose library and its documentation to ensure that I have properly structured the schema and have it in a format that meant I could easily use it in my routes.

Within the controllers/ are my Express routers. These are used to specify how each route will react to the user visiting it. For the backend, they are all prefixed by the /api term and some take in tokens to ensure that the user sending the request is authenticated. The main Express file then imports these routes to ensure they are able to be used by the Express server.

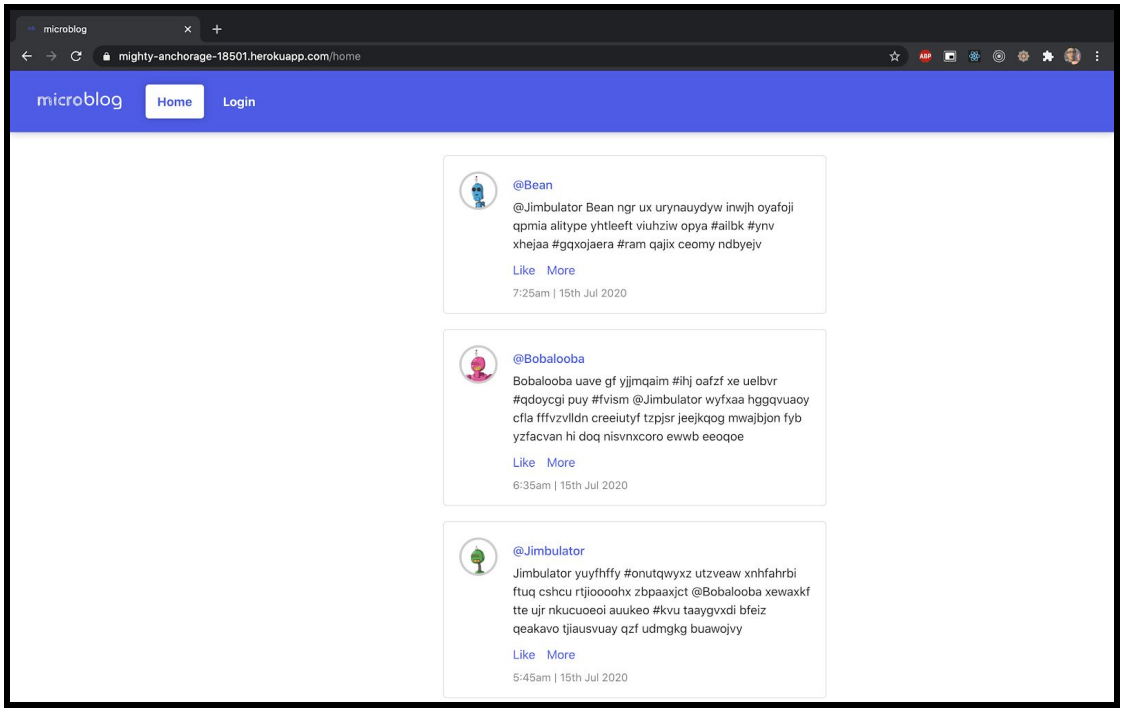
All passwords were encrypted using the `bcrypt` library to ensure we were not holding unencrypted user passwords at any time. Having the passwords encrypted means that we have no way of reading these without knowing the actual password as we use the library to compare the two hashes given the users plain text account.

User Interface

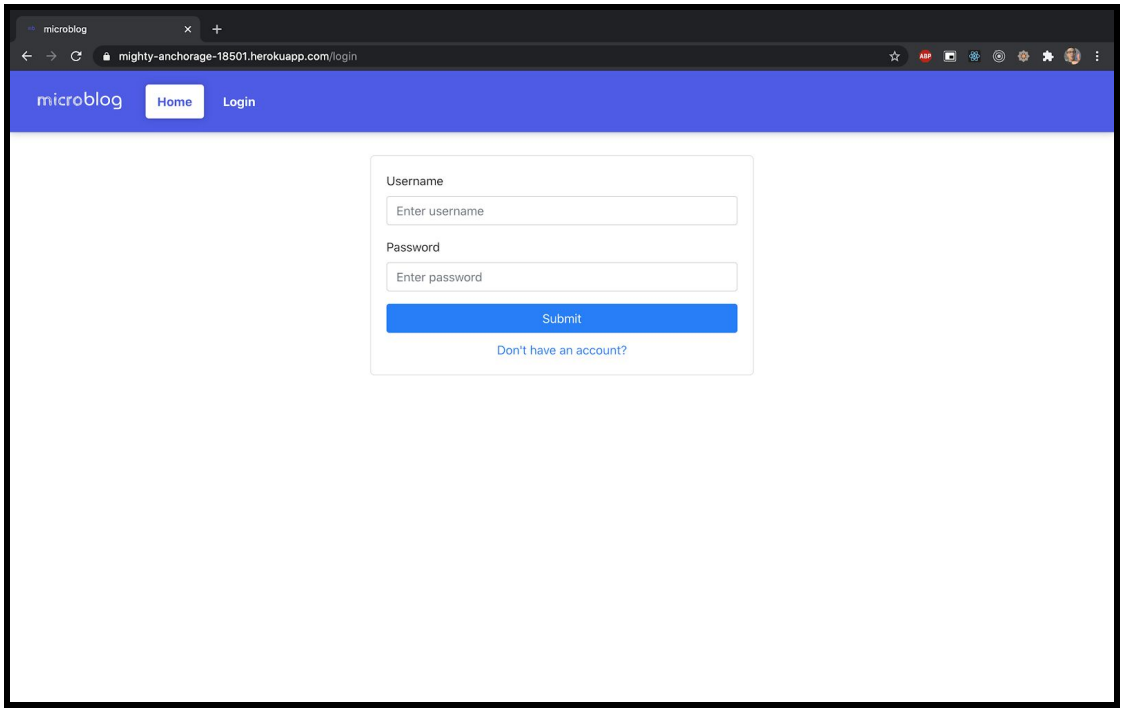
In this section, I will provide some description of the user interface as well as the pages that the user would expect to see when navigating my application.

In my application, when you go to the main URL, you are automatically re-routed to the `/home` directory. I believe this was a better representation of the page that the user was on. On this page, you are presented with a series of posts made by users. I decided to allow anyone to see the posts made by people

to ensure the platform was ‘open’. The screenshot below shows what the page looks like for someone not logged in.

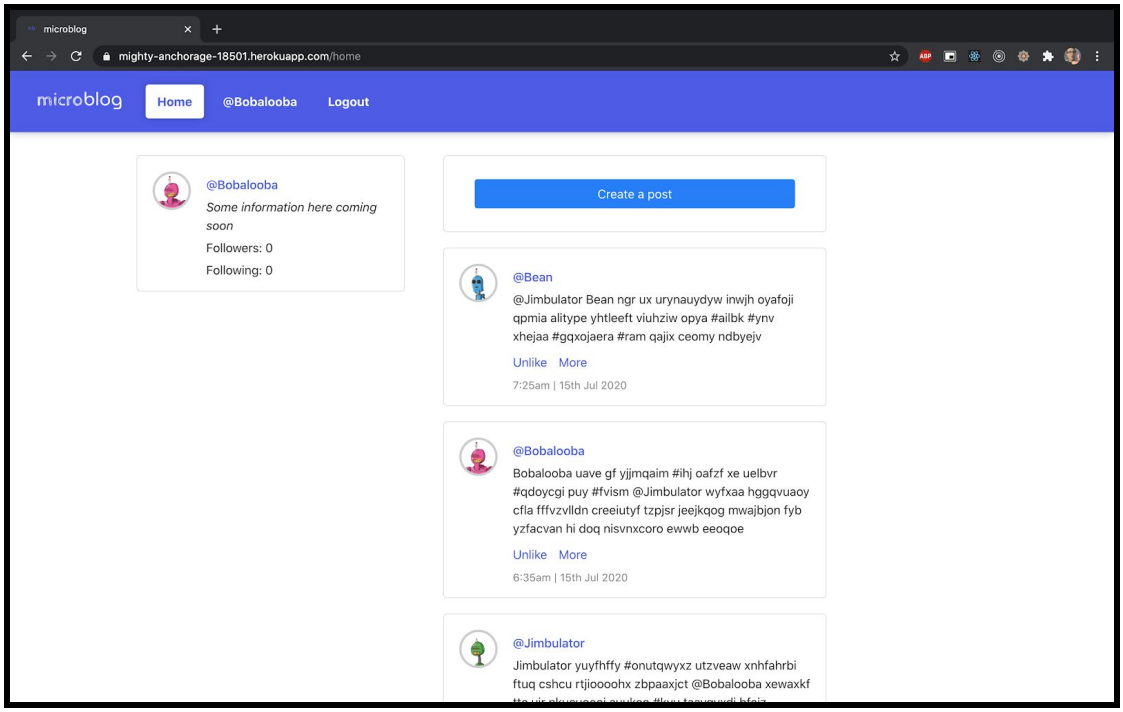


After being on this view, we would expect the user to want to log in next. By using the navigation bar, they are able to go to the login view where they are presented with a form to login.

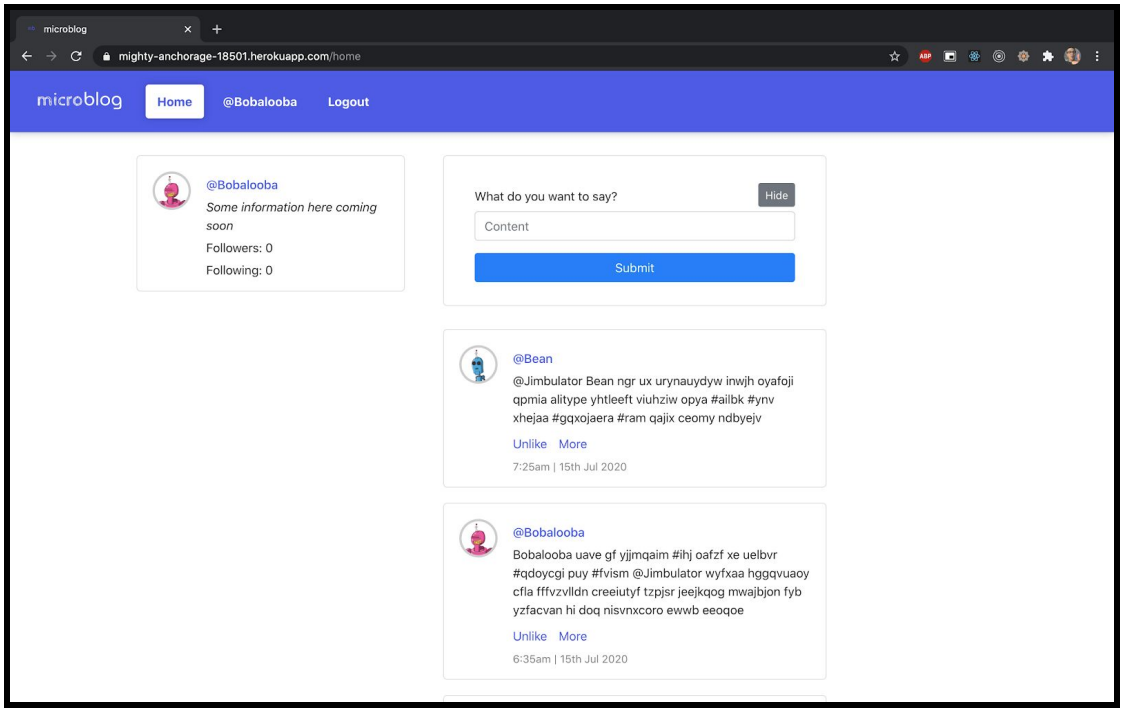


I also have a register view that is similar to this and will allow users to register an account. Both views present error messages when the username/password is incorrect or when a user doesn't enter the same password twice when trying to register.

Once logged in, users can see their profile card to the left of the feed cards. They can also see that they are logged in as their username is in the navigation bar and they have a 'logout' button present.

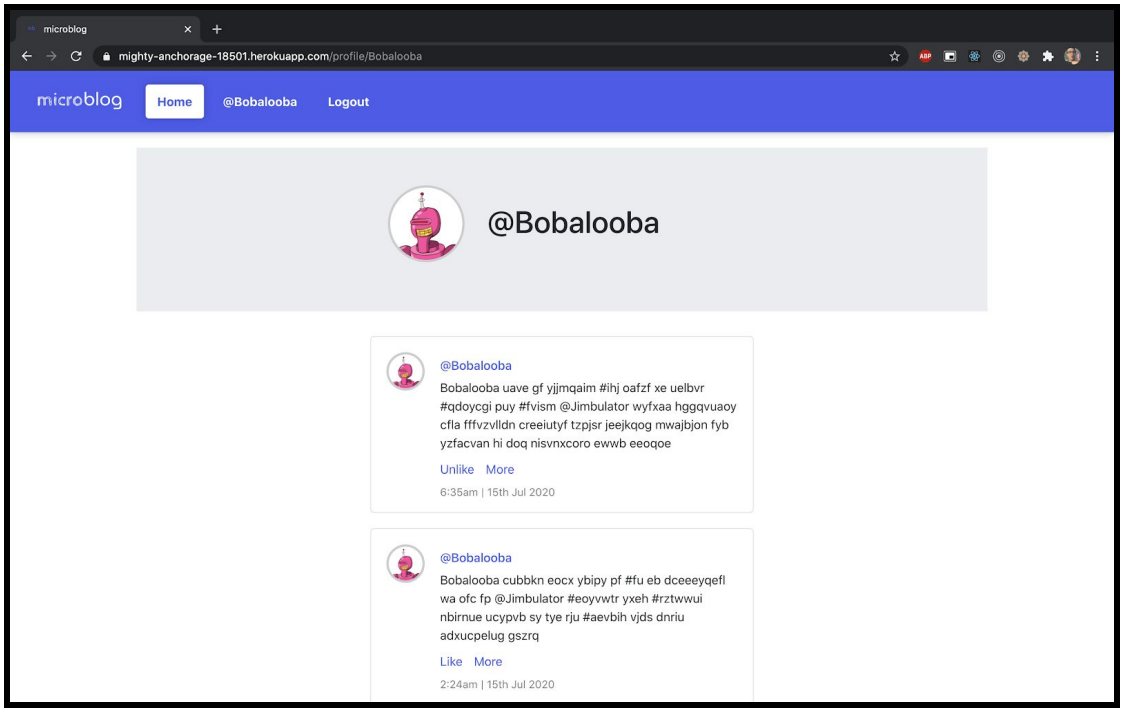


If the user wants to post something, they can press the "Create a post" button and will be presented with a form that enables them to create a post.



Users are also able to press the “like” button on each of the users' posts. The feed will come loaded with the button saying “Unlike” if they have already liked the post in the past.

The last main view is the profile. In this view, users are presented with a header that shows the username and avatar of the user’s profile you are viewing. Then, below that, they are shown a list of the cards that that user has posted.



45209723 Bradley Kenny

Deployment

To deploy the application, I used Heroku. After creating the Heroku application on my local machine, I was able to quickly learn that it wouldn't be as simple as the JavaScript examples used in class. Due to the nature of my application using TypeScript, this was a bit trickier than the standard ways we learnt in class. Heroku doensn't support compiling TypeScript. This meant that I needed to run the TypeScript compiler on my local machine and have the build tracked on my Git repository. This was quite confusing at first and took me a while to understand. It isn't as useful as having Heroku build for the files for us on each push but I was able to justify having to build and store the builds on my repository for using TypeScript.

Furthermore, I split my application into two parts `/server` and `/client`. This meant that the Heroku instance wasn't able to detect that my application was a Node.js application so I had to run the command

```
heroku buildpacks:set heroku/python
```

for it to understand that this was a Node application. Then, it was able to install my Node dependencies using the prefixes supplied (`/server` and `/client`). However, I had to build another package.json in my root directory that specified how to install the dependencies.

After completing this support, I was able to push the changes to my Heroku application and it was running.

Reflection

I believe that I did not get as much completed as I would have liked. I still have a lot of features not as polished as I liked and a lot of relatively easy features I could have implemented that would've made the application feel more 'whole'. Tasks like editing and deleting posts wouldn't have taken much work but I was unable to complete due to my own time management on the project. Furthermore, I would have liked to implement a feature like searching and advanced user information.

I believe that I have learnt a lot from this project and have a muhc better understanding of how all these components come together to build a web application using modern technologies. In particular, I believe I feel more comfortable using TypeScript for a project like this. I believe I will be able to take these findings into account and wokr with my team to build a much nicer and complete group project.