Jin Xunze
A0255453A

CVWO Final Submission Writeup

1. User manual

The web application done replicates that of Reddit, where users can log in, post, and comment on other's post. The project features CRUD functions for posts and comments and JWT authentication for authentication.

User	Posts	Comments
Create	Create	Create
Log in	Read	Read
Authenticate	Update	Delete
	Delete	

Tech stack used:

- PostgreSQL as database
- Ruby on Rails as backend
- ReactJS as frontend

The production-ready backend is hosted on https://cvwo-project.onrender.com/ and frontend is hosted on https://main--dreamy-dodol-f0dd1c.netlify.app/.

The dockerised version is also available. One can use the following commands to get the app running on their local machine, provided that ports 3000 and 3001 are available:

```
git clone https://github.com/jxunze/CVWO-project-dockerised.git
cd CVWO-project-dockerised
docker-compose build
docker-compose run web rails db:create db:migrate
docker-compose up
```

The backend will be running on http://localhost:3000 while the frontend will be available on http://localhost:3001.

2. Reflection

Prior to this project, I had a basic understanding of how a web app worked. I tried deploying a simple Flask app previously and had a rough idea of how the tech stack worked.

However, coming into this project, I was new to all the frameworks and languages mentioned. All the new concepts were daunting to me, and I found myself watching a lot of tutorials. The tutorials helped me get a gist of what was going on, but it was only when I dipped my feet into making the app was I able to truly learn the concepts. As the code became more complex, the concepts I have learned from watching the videos were challenged, and I often had to Google and read more on some of them.

As I spent more time coding and reading through documentation, I soon realized why Rails is such a popular framework. The MVC model in Rails helps simplify the development of the backend framework substantially, for example simplifying database connections and migrations, and providing controllers that bridged the gap between APIs and querying the database. Moreover, with a popular framework comes with a lot of online resources, which I can easily tap on. Every error that I have gotten had been asked by someone on the internet, making debugging much easier.

Learning and implementing ReactJS was manageable for me. Even as the application scaled and my code became more messy, the concept of many components coming together felt very intuitive for me, and I could easily pick up the framework and maintain understanding of my code. I was able to see why ReactJS is such a popular framework compared to vanilla HTML.

CSS was a big challenge for me. There were so much CSS resources online that it took me a long time to try and read the documentation for the different frameworks. I wanted to avoid learning about vanilla CSS and as I have heard many horror stories about it. I tried many frameworks, including Bootstrap and styled-components. Despite reading up on all the different options, I was still having trouble making the app look the way I wanted, especially when it came to arranging the components on the screen. With much reluctance, I decided that I needed to learn about vanilla CSS. I watched a few videos on vanilla CSS, and came across one from Fireship that recommended the concepts to learn, which included the box model and flexbox. I took my time to learn these concepts and were able to apply them in my app. Although I would say that I am still not proficient in CSS, I believe that my understanding has improved a lot more, and I am able to perform basic styling.

After I was satisfied with my application, I thought that the worst of app development was over and that deploying the app would be simple and fuss-free. Instead, I found myself spending the most time in trying to deploy the app. I read through reviews and documentations on different web services, but found that I could not deploy either my backend or frontend. I realized that the main issues were the different compatibility between versions of different software, and the environment variables that I must take note of. As the deadline approached, I was getting more frustrated and started considering my options. I decided to do a tried and tested method, which is to host both my backend and frontend on my Raspberry Pi, and leave it running in my dorm in school. This meant that only those within the NUS wifi will be able to access the website. I thus decided to dockerise my app so that I can run it on my Raspberry Pi easily. One day before the deadline, I talked to a friend about hosting my app, and decided to give Render and Netlify one last try. After much tinkering late into the night, the joy that I felt that I could finally log into my website was immeasurable. Through this process, I also understood the immense benefits that Docker brings, and why it has been rising in popularity in the past few years. It allows apps to be deployed easily and disregards downloading specific versions of packages by the user.

I spent much of my holidays thinking and coding up my project, and looking at my final product, I can finally say that the effort has been worth it. Although the app may seem simple and features lackluster, I learnt a lot along the way. From implementing a simple search bar to styling the posts the way I wanted, each detail of the app was deliberate and took time. I can appreciate and understand software engineers more, and it made me ponder the code and security behind some of the websites I have seen.

There are many future improvements to the app that I wish to make during my free time to make this into a better portfolio project. The first being adding likes and dislikes to each post and comments. I would also like to add more customizability to each profile, such as setting a profile picture and going into a profile to view all the user's posts. Lastly, I would also like to try penetration testing on the app to identify any security flaws or loopholes.

Throughout this holiday, I have learnt any skills and values pertaining to coding that I feel will not just apply to creating web apps. One of these skills is having the patience and capability to go through documentations. Much of the time I went to Stackoverflow to solve my queries, but I realized that my answers could have been easily answered by reading the documentations. Another of these takeaways is having the perseverance and tenacity to debug my code. I found many times when I gave up on a specific part of my app, I slept on it and came back the next day with new ideas to find it working again. This process took a long time, but was very rewarding when the bug disappeared, or sometimes frustrating when it was a small piece of code that I had constantly overlooked.