CVWO Project Final Write-up

Hao Sitong A0238908R

What I learned

Even though I already had some experience in web development, this is the first time I have built an entire full-stack project on my own. Building a simple to-do list app is more complicated than I expected, and doing it all by myself allowed me to learn more than I could have imagined.

I first implemented the frontend, using React with Typescript. I was somewhat familiar with React, but I learned more about the lifecycle of React, how it re-renders every time the state is updated. I also learnt to use the useEffect hook to do something (in this case, update loggedIn state) after the component had been rendered.

I also learned a lot about Typescript and how it helps to prevent errors by catching them at compile time, especially in the case of type errors. However, I did find using Typescript to be a little counterproductive in my project. I still enjoyed learning Typescript and appreciate its effectiveness for larger scale projects.

I modularised my app in different nested components, such as Card, List, and ListItem. However, this soon proved to be cumbersome for transfer of data, such as states and callback functions. I researched a bit about this and I realised some components could have been put under the same file and been nested there, such as ListItem in List. Since ListItem only appears in List, it makes no sense to separate them. I think this could also be solved with Redux.

The maintainability and scalability of my code can be greatly improved using Redux and using the MVC structure, which I both have experience with, but I believe they cause unnecessary hassle and complication for such a small project so I did not use them. I really hope to be able to use and witness the power of these tools if I can work on a CVWO project.

For the backend, I decided to challenge myself by using Golang, something I have never touched before but always wanted to use due to its known effectiveness and performance for server programming. I really enjoyed using the language. I love how it is statically typed but still has a simple, elegant syntax. I considered using libraries such as gin to implement my RESTful API, but I decided not to as I want to learn the inner workings of it, and the standard golang http library seemed to be enough.

Using golang proved to be a real challenge as there is much less online resources/tutorials on it. The first problem I encountered was how to use middlewares to authenticate requests, a pattern that I'm familiar with from experience in Express.js. Eventually I managed to do it using a custom ServeMux, something similar to a Router, and implementing middlewares by using a chain of function applications.

As I used Heroku and Netlify to host the backend and frontend separately, another problem I encountered (that cost me quite a bit of sleep) was cross origin resource sharing (CORS), which by default prevents cross origin requests and cookies. It took me a while to realise that the browser sends a preflight request of the type OPTIONS, for which I have to handle manually. I also needed to configure access control options for it to work. Eventually I made it work but I am unable to make it work on Safari which prevents cross origin cookies by default (a change that was introduced recently), and I was quite surprised by the sheer number of friends I have that use Safari. I learned that I could use nginx to proxy the requests, but I don't have enough time to work on that. In the meantime, I will just have to apologise to my Safari-using friends.

I also learned about and used bcrypt hashing for passwords and JWTs for user authentication.

While setting up the database, I learned more about SQL, and about foreign key constraints and ids. I took a shortcut by setting up the database on TablePlus in the interest of time but I'm sure there's a more proper way to do it. I hope I can learn more about databases and how to handle migrations in the future.

What I will improve on next time

Even though my code works well, I still hope to learn to make my code as efficient and simple as possible. I hope to gain more experience and learn better programming practices, either frontend or backend, to improve the reliability and performance of my code. I believe a good way to do that is to learn existing codebases or work with more experienced developers.

If I (hopefully) get to work on a CVWO project, I will acquire a more thorough understanding of the tools and technologies I use beforehand. For example, I want to gain a deeper understanding of Golang and its advanced topics such as goroutines. I also hope to understand better how React and its virtual DOM works, and how to fully harness the power of react, such as using Refs, Portals etc. to build more complex projects dependant on reliability and performance.

I will also need to implement more secure authentication practices, such as better JWT validation.

The lack of standardisation of naming and types across frontend and backend proved to be a pain during my development process. Next time, I will surely standardise them more and perhaps have a more thorough plan before embarking on the project.

With a much better understanding of web development across the stack, I will definitely be more efficient for future web development projects. I will be able to focus my effort on more advanced topics and less trivial things, which I'm looking forward to.

Conclusion

This project allowed me to learn a lot about the web, which I find very fascinating. Nonetheless, there is still so much to learn about web development and the many technologies/tools. After this project, I'm inspired and motivated to learn more and be a better developer. I'm looking forward to utilising what I learned in this project to build more powerful and reliable systems that people can depend on.