Writeup

This project was naturally very fulfilling as I created it from the ground up (database, backend, frontend and finally deployment). At first, creating the database was quite simple as Heroku already provided a free tier PostgreSQL database for me.

However, the tough part came after that when I had to learn a new programming language, Golang. In my past experience, I've only worked with higher level interpreted languages such as Javascript and Python. It was an entirely new concept for me to use static typing, pointers and slices (and not just arrays). I had to face many difficulties such as understanding the file structure of Go projects, learning their syntax as well as their error handling. I've also learnt many other new things such as type casting, channels, runes, bit shifting, goroutines, interfaces, all of which I never applied in my project but was nonetheless fulfilling to learn. I'm especially fascinated by Go's ability to leverage concurrent processes, as it's a rather modern programming language that is more adapted to current multi-core computers.

The control flow of the code was very similar to other backend frameworks I've used before. There was a main code file which pumps up the server, connected to a router with their respective handlers (controllers) handling each route. Hence, it was quite manageable to set up the backend after learning the language.

After testing the API endpoints, I shifted my focus to the frontend. It was fun to create the components and stylesheets myself. There were definitely issues with the styling at many points in time, but the errors were manageable and easy to find. A slightly more challenging part was making sure the requests from the client won't crash the backend server. I could make the CRUD functionality on the frontend much faster, but I didn't dare to risk any potential issues that could crash the backend so I set a loading screen whenever I had to wait for any HTTP responses from the server.

A weird thing to note was that I was unable to test my phone when sharing the same wifi (hotspot, NUS wifi, etc) as my local computer. To my understanding, the devices should share the same network and can access each other via their private IPs. However, I was unable to connect to my computer's backend from my phone(which I suspect is due to CORS issues). The design of the app was very simple so I didn't worry about responsive design too much, but I did not feel too accomplished about that.

I also considered the use of the Redux store due to massive copy pasting along a chain of components, but I felt that the size of the boilerplate was too troublesome to justify its use.

The deployment of the application was the toughest part of the project. The main issue was the potential lengthy debugging time as I have to wait for the CICD pipeline to update the server every time I changed something. Thankfully, Heroku provides documentation for deploying Go projects which eased the process.

Regarding the errors, I had to add a buildpack (https://github.com/timanovsky/subdir-heroku-buildpack.git) for my project directory structure, so that Heroku could recognise my Golang application. I also mistakenly tried to run a build file generated on my local computer, in the Heroku dyno. This triggered an error as the binary in the file is catered to my local computer and could not be run in the dyno. I only found that out after accessing the dyno itself through the Heroku CLI and realising there was an executable file compiled

for me already. I also did not realise that Heroku provides a port number for the application as an environmental variable, which took me awhile to debug.

However, I'd say that the Heroku CLI's ability to access its configs, database add-ons, dynos and log files, as well as its CICD pipeline were all very helpful in the deployment process and also made me clearer as to what I'm doing during debugging.

Overall, it was very enjoyable to create and deploy a Golang project, a language which was rather unfamiliar to me. There are definitely still issues such as slow updates to the frontend, repetitive code structure, insufficient error handling, a rather fragile backend server, etc. But all in all, it was extremely satisfying to piece together the entire project together and form this application.

User Manual

To run the application, first clone the repository (https://github.com/kennethk-1201/cvwo.git).

As the repository was based on the latest push (which was during production), the following steps are needed to run on your local server:

1. Use a PostgreSQL database with the **Tasks** table schema below

Column	Type	Nullable
id (PRIMARY KEY)	integer	not null
title	character varying(100)	not null
body	character varying(500)	not null
deadline	timestamp without timezone	not null

2. In **backend/cmd/Main.go (line 25)**, change the port number to 8000 (or whatever unused port you have). Similarly, in **backend/internal/helper/helpers.go (line 16)**, change the **DATABASE_URL** argument to the database you're currently using. This is because the environmental variables were set by Heroku's configurations in production, but they do not exist in development.

3. NOTE: in **backend/cmd/Main.go (line 17)**, you can change the whitelist of origins for your CORS configuration.

4. Run **go mod download** in the **backend** directory to download all dependencies.

5. Run **go run cmd/main.go** to start the backend.

6. In **frontend/src/.env.default**, set **REACT_APP_BACKEND=http://localhost:8000** and change the name of the file to **.env**.

7. Run **npm install** in the **frontend** directory to install all dependencies.

8. Run **npm run start** to start the frontend.

Your server should now be running!