CVWO Assignment

User Manual

To install set up the forum on your local machine, follow these steps:

Option 1 (without Docker)

1. Clone the repository.

   `git clone https://github.com/minreiseah/cvwo-assignment.git`

2. Install the dependencies: `cd web && npm install` (frontend) and `go get` (backend).
3. Set up the PostgreSQL database. Migration files are found under `db/migration`.
4. Set up the environment variables:
   1. `web/sample.env` (frontend)
   2. `sample.env` (backend)
5. Start the development servers: `cd web && npm start` (frontend) and `make start` (backend).

Option 2 (with Docker)

1. Clone the repository.

   `git clone https://github.com/minreiseah/cvwo-assignment.git`

2. Build the Docker images.

   `docker compose build`

3. Set up the environment variables:
   1. `web/sample.env` (frontend)
   2. `sample.env` (backend)
4. Start the Docker containers.

   `docker compose up`

Reflections

Diving in

I picked Go as my choice of backend for three reasons. One, Ruby seemed rather outdated and opinionated. Two, Go allows for more flexbility in building web servers. Three, I wanted to take on the challenge.

After going through this two month process of web development, I can gladly say that I thoroughly enjoy Go and tolerate React. Go was surprisingly beginner-friendly with a strong standard library, but offers a depth of functionality that I am sure to explore in my future projects.

Testing

This project involved tests on both the React and Go sides, of which I am much happier about the latter.

I began my project by building up the frontend. As there was no API to interface with, my React tests involved mocking axios and simulating API calls with Jest and the React Testing Library. It was extremely painful as the tests were not very useful to actually getting a product out nor were they easy to implement. I simply decided to write these tests because I wanted to learn how testing in React worked.

The backend, on the other hand, was a lot more enjoyable because I was actually testing where my SQL queries were functional. For instance, as categories and threads were tightly coupled via a association table, writing tests would ensure that any changes made to the category handlers would not break the thread handlers.

Furthermore, there was a lot more documentation available to writing Go tests as compared to the ever-changing world of frontend development. Even chatgpt was not kept up to date with the newest trends in React development.

If I were ever to be on a team that had to build a frontend independent of the backend, testing would definitely help to write more readable and bugfree code. However, for future *personal* projects, I will just build a backend first to avoid frontend tests. Well, I am still quite proud of my first attempt at writing unit tests. My next step would be to write E2E tests.

Challenges

Docker & Hosting

While local development was done with docker compose, it is *relatively* expensive to use in production. Building and deploying three separate services (frontend, backend, database) via `docker compose up` is not possible (for free) on services such as AWS because of a requirement to use 1. ECR to host the containers, 2. ECS to deploy the containers, and 3. RDS to host and manage a PostgreSQL database.

Furthermore, the above would just allow for the web application to run on a VPS. However, networking and reverse-proxying would also have to be done so as to allow end-users to access the application. This is something I will look into for future projects that require scalability.

Hence, I have resorted to using Render as a *free* cloud hosting provider for now. For future projects, AWS is definitely something I would aim to implement.

Networking

In deciding between hosting all my services on the same network or on different networks, I went with the latter option for two reasons:

One, simplicity; it is easier to host my services on three different networks and access them by their external URL. In this way, the PAAS I used does all the routing.

Two, hosting all services on the same network will take up a lot of time due to my lack of proficiency with networking.

What I would have liked to add

On documentation; as a novice to frontend development, using React docgen would have been helpful if not for the constant refactoring and movement of components. This would have made the project rather difficult to maintain, requiring more effort to document than to build the actual product.

After 'finishing' the project, I had many API routes written that went unused. Some features I would have liked to add to the frontend would be:

- member profile page (view threads and comments)
- inline updating of threads/comments

On security, I want to learn how to project my backend API with authentication. This allows only authorised users to access protected resources on the server.

Concluding Remarks

All in all, this project has been a great learning experience. More than just web development, I gained a sense of appreciation for building products from a client's point-of-view. I hope to be given the opportunity to apply my technical skills on projects that make a real difference in the lives of others.