CVWO Final-Assignment Submission Emily Ong Hui Qi (A0240634M)

Reflections

Embarking on this assignment has been very fulfilling and enriching. What I had originally thought was a simple task evolved to be something more complex than I had expected. In fact, completing the assignment seemed almost an overstatement at this point. At the start, I was very ambitious and had many plans to integrate complex functionalities and tools into the application, as what I had set out in the Mid-Assignment submission. I wanted to set up a complete continuous-integration and deployment pipeline, use Cron jobs, and go beyond the scope of the assignment to enable sharing and configuration of user roles, just like Google Drive. Although some of these tasks were completed, they were not quite as ideal as I would expect. Nevertheless, I had many great takeaways from this project!

Planning and managing expectations.

When I started out this project, I have a rough skeletal conception of how the application would look and behave. I wrote down the features that I wanted, designed the database diagrams and researched on possible tools and libraries that I would use. However, software development is an incremental developmental process and I only realized late to build things incrementally, rather than in one-go. Finally, it was also vital to manage expectations- each project needs its own unique-selling point, instead of being jam-packed with various features (that no one might even know how to use!). It is important to prioritize core functionalities, which was why I decided to spend some good amount of days refactoring code, before moving on to solidify existing functionalities, making them better and better.

Atomic Web Design

One of my takeaways was the idea of atomic web design, where components are built on top of each other much alike the atoms, molecules and organisms in biological systems. This made very intuitive sense for me, as I had commonly seen React projects structured “component-wise” where components used to make component XXX are parked under the same folder as component XXX, or other similar strategies, which seemed too convoluted. Throughout the project, I tried to adhere to the idea of atomic web design, building components that are reusable and abstractable (by prop designs), ensuring that components at the lowest level are purely presentational.

Learning Golang and React

Another takeaway was the learning of Golang and React, both I was previously unfamiliar with, and it turned out to be an awesome experience despite a tumultuous start! Initially, I relied on blog articles and documentations to set things up, such as configuring JSON Web Tokens for user authentication, or how to use Redux to manage state, but as I slowly became more comfortable with development, I was better able to accomplish my goals and things began to pick up pace!

Database Design

Furthermore, I learnt a lot on database design and how to model relationships with primary and foreign keys. One of the pain points during development was that the SQL queries were quite slow. As I used Gorm as an ORM to interact with the

database, it was harder to investigate the cause as I had to add debug statements to print out what the resultant queries were. And thus, the next milestone of this project is to fully investigate slow queries, since this affects user experience greatly. Some ideas to approach this that I can think of is to connect a performance monitoring dashboard first. This allows us to know which queries are the slowest, and to start fixing from that query. To understand the problem better, I would try writing raw SQL statements and compare that with using Gorm prepared statements, and to also look further into Gorm's documentation on other speedups or optimization tricks.

Coding Quality

Furthermore, I am glad to subject my project under strong coding quality standards right from the start: I set up linters, used Typescript and tried to organize my code and comments in a structured manner. This proved to have pay off later as the project scales. It became easier in a way to build the project as many components and functionalities are reusable, and it was also easier to debug any problems with the application. It was also a fun challenge for me to set up Makefile and scripts to handle common shell commands, such as running or deploying the application.

Deployment & Tests

One of the things that I am satisfied with was also being able to deploy my application and configuring a domain (https://tuskmanager.rocks/) and subdomain (https://app.tuskmanager.rocks/). In deploying the application, I chose to deploy the backend on Heroku (https://tusk-manager-backend.herokuapp.com/), as Heroku provides many out-of-the-box addons, such as connection to a Postgres service, and strong command-line interface support. I have also picked up Docker and set up Github Actions to build a Docker image and push to Heroku directly whenever a commit is pushed to the main branch. For the frontend, I used Vercel and configured the domain name system settings to point the domain to the Vercel site, which was quite a frustrating but worthwhile experience!

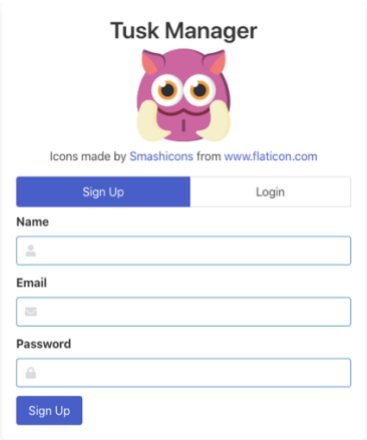
On the next step, I want to improve the pipeline and set up tests for both the frontend and backend applications, which I did not manage to accomplish during this iteration due to the lack of time. More so, I would like to learn how to write good unit tests and what it means to improve code coverage, as well as setting up a "Storybook" for the React frontend, which I believe is vital as the project scales, and more components are added to the application. Finally, I would like to explore the idea of setting up development, staging and production environments. Currently, a single Postgres instance is connected to both development and production environments, which makes data messy and harder to analyze in later stages of the project. Moreover, changes are pushed directly from development to production, which falls short of the aspect of quality assurance. However, such features are likely to have minimal impact as of now, since the project is maintained by one person and used on a small scale.

In conclusion, I am glad to have taken up this assignment. There were many aspects that I was prouder of, some less, but it was an overall awesome learning experience, that I never expected myself to have invested much time and effort into. It is probably the largest project that I have ever done, exploring both backend and frontend development, and being able to make decisions about the project direction and specifications provided me with a strong sense of ownership. I hope to continue this spirit on a larger scale and be able to learn how to better work with others on

real-world projects that are no longer prototypical, but rather, can make a genuine impact on someone else's life- and that, I believe is the value of software development.

User Guide

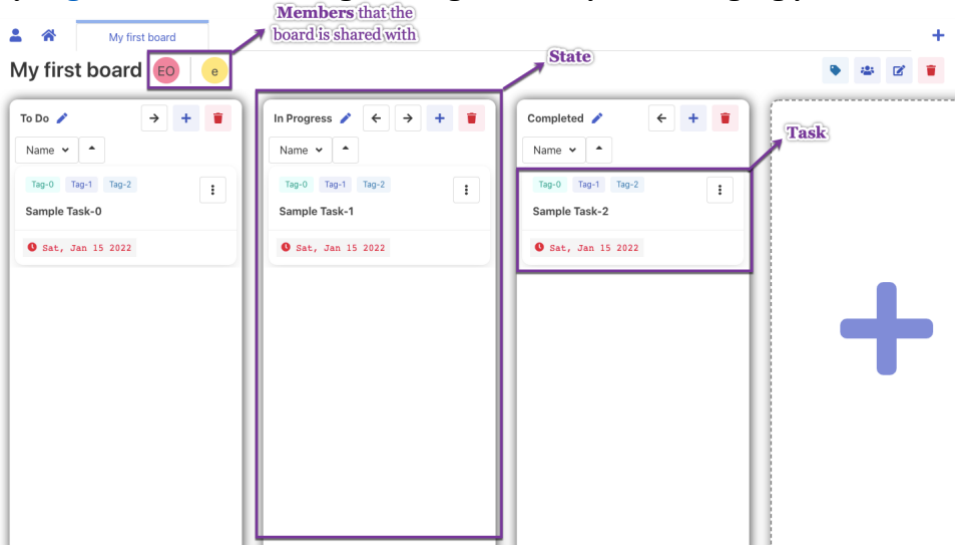
When you first enter the application, you will see this authentication page.



If you have an existing account, you may log in with your previous credentials (email and password). If you are new here, you may want to sign up for an account.

The password that you enter are hashed before being stored in the database and the authentication system is powered by the notion of JSON Web Tokens (JWT) authentication, which keeps you logged in for 24 hours.

A board is simply a visual interface that you can share with other members. Each board contains a collection of tasks and each task gets its own state. A state specifies the stage that the task is currently at (is the task under *To Do*, or *In Progress*?), and you can create new states depending on your workflow lifecycle. What's more, you can specify tags for each task for greater granularity in managing your work.



Read up more on the documentation site!