Frontend:

Before the first submission, I made the basic frontend features like the modal, forms using a mix of React material UI and JSX. I thought of using TypeScript to try to understand how each component worked but it was more difficult than I expected. There were many types that I needed to take note of and generics were needed when I was passing props down through components. I also learnt a bit of destructuring of props to make the code and functions more readable for users. TypeScript taught me the specific events being passed around.

In addition, I also learnt about React Router. While making the taskbar in material UI, I needed to provide users a way to move between tags. React Router helped me to link buttons on my navigation to the different priority tags as well as home page.

Using asynchronous functions was also quite interesting as it brought bugs into the handlers. Without event.preventDefault() to prevent the component from reloading after the changes(namely the form), the updated data would not have been sent to the database as the data is still being processed and instead an empty form would be sent.

Improvements:

I think I could have planned out the components better. A lot of the Handlers were inside the components instead of being in redux or context. In addition, I didn't manage to use my loading and error states properly.

I learnt useContext in the react courses I browsed through during the winter but I didn't end up using it because of time constraints from debugging handlers in my frontend. I feel that I could have cut down the repetition of my code and wouldn't require so many props if I had utilized my context. But hindsight is 20/20.

UI/UX. I only discovered most of the bugs after asking my family members and friends to try to use the app and break it. I had a rough idea of how I wanted the app to work and look but wireframing it in figma might help quicken my frontend development.

Validation of forms. I read some articles later about formik and yup which could have helped my edit and creating of tasks by ensuring proper inputs before submitting to my database. Authentication of users would have been another rabbit hole

Backend:

It was quite fun learning GO syntax but it had a lot of peculiarities that other languages didn't have. Using & and * to reference and dereference pointers was something I was not used to. Functions also needed to be captialised to be exported. Thankfully there were articles and videos which outlined how to make API calls to mongoDB. I had to also learn what was Postman and how it worked through video tutorials.

I was ecstatic when I used Postman to test my API calls and it worked. But this joy was soon overwhelmed by dread when I received multiple CORS errors when performing my fetch requests inside the frontend. I found some articles which outlined how to enable CORS within the backend so that the calls will go through in the frontend.

Deploying:

In order to ensure that the code works locally and in production, I had to make sure that there was environments with different API endpoints. Following this, I had to set up an instance of AWS EC2 and open ports 80(HTTP), 443(HTTPS) and 8000 (local host). I had to try and learn how to perform reverse proxy using Eginx through numerous guides, but I'm still not entirely sure whether it's secure. I also learnt a bit more about systemd in linux in order to keep my frontend and backend running.

User Manual:

1. Git clone project to local directory
2. Database
    a. Create mongoDB Cluster
    b. Click Connect -> connect your application -> Set Driver as Go and version 1.6
    c. Copy connection String and change password accordingly
    d. Create a file app.env in the same directory in as main.go and paste your database link:
    DB_SOURCE=mongodb+srv://evan:<password>@task-manager.ttttk.mongodb.net/myFirstDatabase?retryWrites=true&w=majority

3. Backend(One terminal)
    a. Test backend by running ```go run ./``` when cd inside backend, it should print out Connected to MongoDB if it works, else the program will terminate with error

4. Frontend(Another terminal)
    a. Run npm install when cd inside frontend
    b. Run npm run start:build
    c. An instance of the app should open in the web browser after a while
5. To add tasks, press the + button on the task bar,
    a. Add title
    b. Add description
    c. Press submit
6. To edit tasks, press the edit button on the task
    a. Add title
    b. Add description
    c. Confirm to update tasks, press delete to delete tasks

7. Navigation is done through the menu button on task bar