CVWO Final-Assignment Submission Ng Xing Yu (A0234386Y)

Having no prior experience with Ruby on Rails or the React framework, I have learnt a lot from the process of completing this assignment and managed to create a working product with many of the requirements I specified in Level 1. The UI/UX of the frontend is still rather minimal and only achieves the basic use cases that I had in my plan. Due to time constraints and unfamiliarity with the React framework, I was also did not properly implement unit / integration testing in my frontend codebase for this assignment. I attempted to implement abstraction barriers between the Data, Domain and Presentation Layers with the intent of making it easier to implement tests for various parts of the frontend, but it doesn't seem very useful when not implementing any tests.

I also discovered the importance of thinking about the user experience and interfaces from the beginning even when designing the backend, since that is the only portion visible to customers, and having a feature rich backend API without a decent frontend interface to allow the users to interact with the backend isn't very useful. While I did plan out all the use cases at the start of the project, I only started implementing the frontend UI after the bulk of the backend was implemented, and the presentation layer of the frontend was one of the last things I looked at. In retrospect, it might have been easier to implement the entire frontend UI statically (while using mocks for API responses) before looking at other aspects of the frontend like the redux store or interfacing with the API so I would have a better idea of the endpoints I would need while also reducing the development complexity when working on the other frontend layers or the backend.

I initially attempted to mock my backend REST API using Miragejs and managed to handle simpler endpoints, however I found myself rewriting much of the logic from my backend code in JavaScript just for the mock API, and since I had access to the backend code, I chose to spin up a development backend instance locally using docker and used that to work on my frontend code.

When working on the backend, I realised how useful the entity-relationship diagram was in helping me to plan the structure of the database. My approach to properly test all my backend code and endpoints also saved me a lot of time and effort, since it was able to help me catch various broken functionality and approach development in a more systematic manner. This also gave me greater confidence when trying to implement certain unfamiliar concepts such as the self-referential table for tasks.

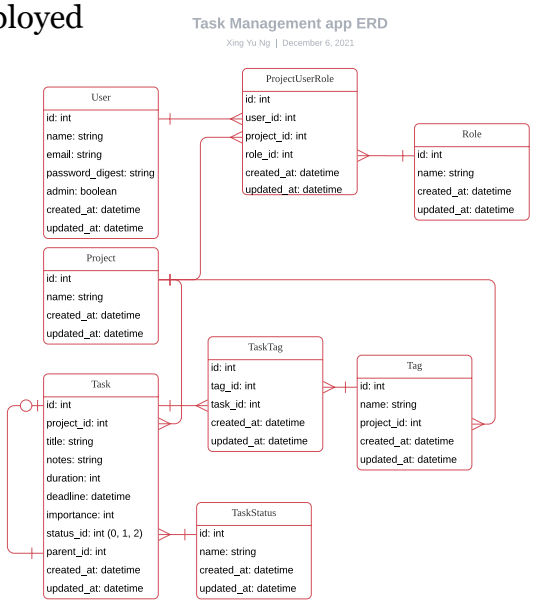
Implementation details: (Refer to mid-assignment submission for use cases and plan)

- Rails backend with React frontend deployed with Github actions
- Deployed backend on Heroku using Docker at https://taskmanager-ngxingyu.herokuapp.com/. Refer to taskmanager-backend/test.http for sample api commands.

- Frontend implemented with react / redux, deployed at https://taskmanager-ngxingyu.netlify.app/

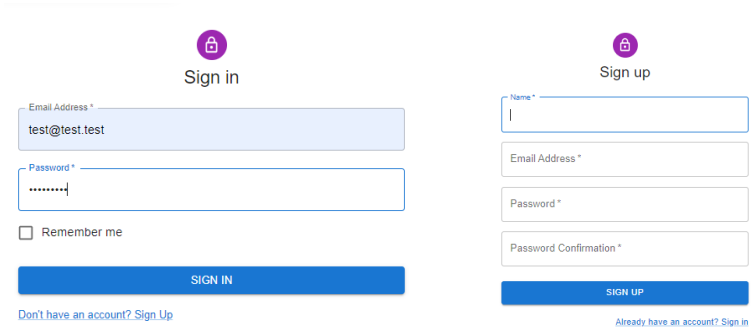
Data

- User table for user accounts
- Project table for projects
- Tasks table for tasks/subtasks etc.

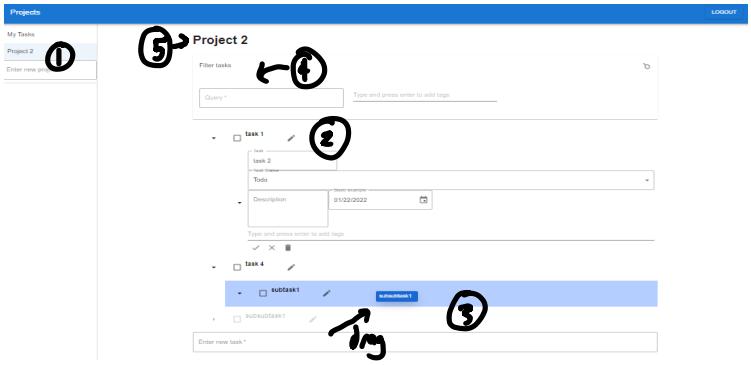


User guide:

- Login / Signup page:



- Project tasks page

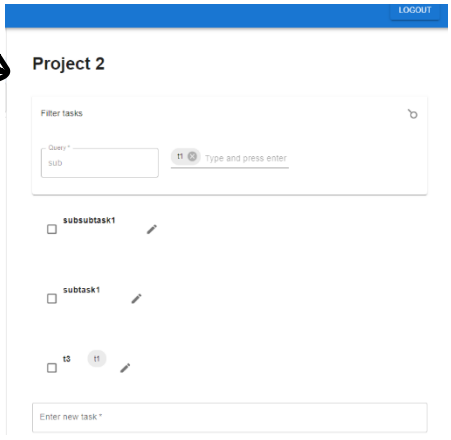


1) Select the projects in the left pane to view tasks, or enter a project title in the textbox to create new project

2) Click on the edit icon to edit the details of a task, or select the checkbox to toggle its completion status

3) Drag a task (A) onto another task (B) to make task B the parent of task A

4) Click on “Filter tasks” and enter some string or tags to filter by either



5) Click on project title to modify details (below)

- Change project title / sharing of project with other users with varying restrictions (implemented in backend)