Task Manager

@author: Leeroy Liu
CVWO Project Assignment

1. Introduction

This application is a simple CRUD (Create, Read, Update, Delete) Task Manager to keep track of tasks that the user has. The application has a simple categorising system to allow categories for various tasks, for easier searching and organising of the Task Manager.

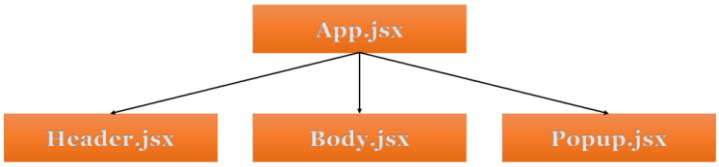
2. Design Analysis

2.1 React Frontend using Semantic UI with Rails API

Semantic UI library was chosen for the React frontend because of its easy compatibility with the Object-Oriented paradigm in React, which I found was easier to read due to encapsulation principles of components and with states as objects, rather than React hooks and more functional paradigm of React introduced in 2019.

Due to more familiarity with React and to allow the application to be a single-page application, a benefit of using React, I have decided to make the application more frontend-heavy and use Ruby on Rails as a RESTful (Representational State Transfer) API for React JavaScript X components to fetch JSON data from. This is to have a client-server architecture and clearly separate the frontend from the backend.

2.2 React Components Structure



To reduce complications in state changes and props transfers, App.jsx is used as the main parent component of other components, and it is where states and methods are stored. Like the redux framework, the App.jsx works somewhat similar as a single source of truth, especially for interactions between child components, as props are passed from the App.jsx to its child components. Redux was not used as I felt that the application was still simple enough that it does not require redux, but future scaling of the project may require redux to be used.

The concept behind the child components is to split them into their various User Interface components: the header, the body, and the popup (modal). The methods associated with them are usually handlers for button clicks, but the main logic behind the methods are in the parent component, App.jsx. Some methods in App.jsx are reused, for example the handlePopup method is used to open different types of modals in Popup.jsx, namely add, update, show, and delete popups.

2.3 User Interface

The user interface was made to adapt to different window sizes, and be formatted to stack if mobile version / small window size was used. The tasks are organised such that the last updated task will be pushed up to the top, although sorting algorithms may be implemented in future. The date on the left of the task title shows the date when the task was last updated. A minor feature of having the edit and delete button both on the

"show" modal and in the main body, was to allow easy access for users to edit and delete tasks.

3. Accomplishments

Personally, I felt that the project itself was an accomplishment as it was my first time using rails and hosting it. Initially, I faced the problem of asynchronous vs synchronous calls, which was solved by revisiting the React Component lifecycle, and updating state once asynchronous calls are done. Furthermore, there were very few online guides on using React together with Rails, and many use React routes and rails routing, while what I wanted to create was a single-page application without the use of routes. As React was not very compatible with Rails itself, gems such as react-rails or react_on_rails needs to be imported as well, for easier handling of components. There were also many obstacles with regards to the use of Semantic UI, one major one being that there was a warning for memory leak when closing modals by clicking outside of the modal, which is likely due to the component being unmounted when the outside of the modal is clicked. To fix this, clicking outside the modal to close it was disabled, and a close button was added instead on the top right corner.

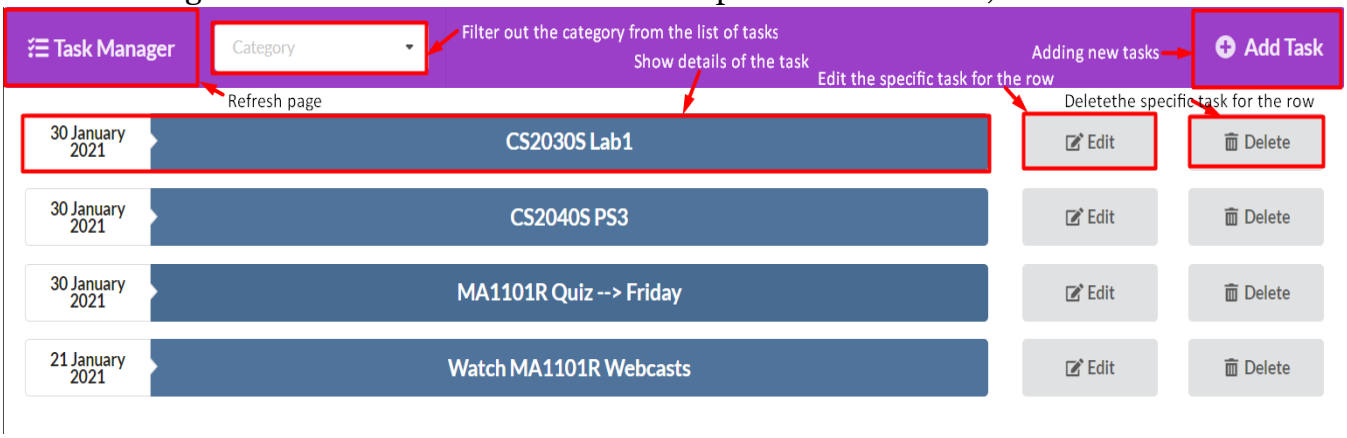
Postgresql was also difficult to setup initially due to the password configurations and pg_hba.conf file, but after much debugging and trying it on foreman to mimic production (when deploying to Heroku), it managed to work.

Overall, even though it was a simple app, it was an eye-opener to how difficult it could be to keep track of props and state changes, and that redux or the concept of "single source of truth" is very helpful in such situations. I am also glad that I can use some CS1101S knowledge of .map and .filter for the arrays, and some functional programming in my methods.

4. User Manual

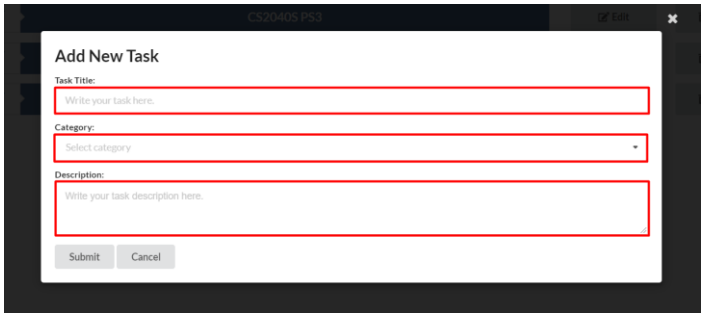
4.1 Navigating the application

The image below shows the different clickable parts of the website, and their functions.



4.2 Adding tasks

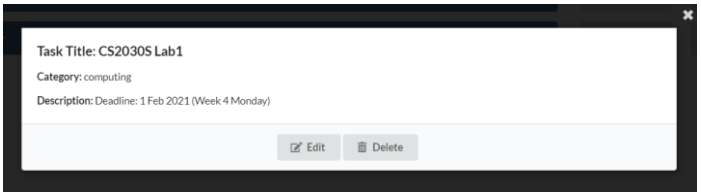
1. After clicking the "Add task" button, a form will pop up. Fill in the Task Title and Description, and choose the Category via the dropdown selection. You can also add a category by clicking on the field and manually adding your own category.



2. Once you are done, click on “Submit” to add the task. You can also cancel and close the form by clicking “Cancel” or clicking on the “X” icon on the top right of the popup.

4.2 Showing details of a particular task

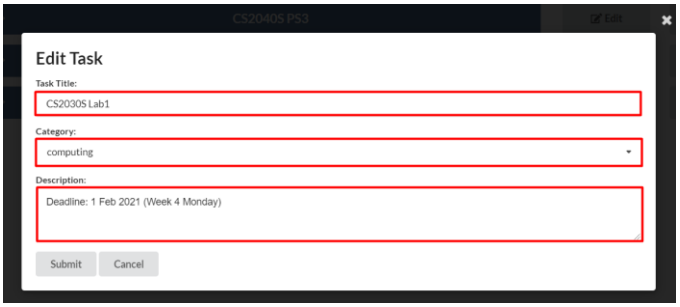
1. After clicking on the task, the following popup will show, revealing details of the task.



2. You can edit or delete the task by clicking on their respective buttons, or close the popup by clicking on the “X” icon on the top right of the popup.

4.3 Editing a particular task

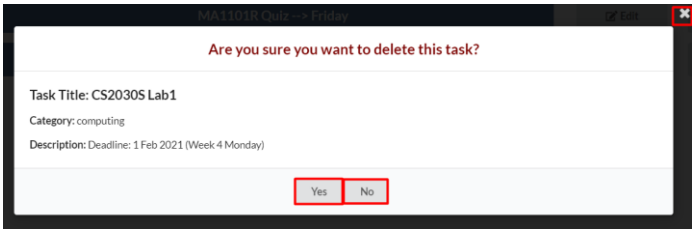
1. There are 2 ways to edit a task: from the main page, or from the popup showing details of the task. After which, the following popup will show with details filled in.



2. Edit the respective fills similar to adding of task, and click “Submit” to finalise. You can also cancel or close the form.

4.4 Delete a particular task

1. There are 2 ways to delete a task, which is the same as editing. A popup will appear:



2. Click “Yes” to finalise the deletion, and click “No” or close the popup to cancel.