CVWO Midterm Submission

Use Cases

1.Management of Todos
Each todo item should have a title which serves as a short description of the task, and also have a body for more details if needed. The user should be able to indicate if a task has been completed and also indicate a due date. Users should be able to hide or show completed todos. Each todo item should also have the capability of having sub tasks, since it is a natural way to group together related tasks and indicate dependency between them.

2.Categorisation of Todos
The user should be able to group the todos into separate lists (e.g ""Work", "Home," etc...). Within each list, the user should be able to further categorise the todo by adding tags to the todos. Search functionality needs to be implemented for users to filter based on these tags. CRUD operations need to be implemented for the todo lists and the tags.

3.Authentication
JWT token based authentication will be used to authenticate and authorise users to see and perform actions on only their tasks. Users should automatically login to the application if they have a valid token, and automatically be prompted to re-login once their token expires.

Execution Plan

I was rather new to both React and Rails and web development in general. So I took some time to understand basic web development concepts and familiarise myself with both these frameworks.

After some research on using React with Rails, I decided to set up the Rails application as an API and consume that API using React on the frontend. I felt that decoupling the backend and frontend, would make development easier as I could just focus on thing at a time. I also wanted to learn more about creating and consuming APIs in general and I felt this would be a good way to do so.

Current Progress

1. CRUD Operations for TodoLists and TodoItems have been completed

So on the backend in Rails, models for TodoLists and TodoItems has been created and a many-to-one association has been set up between them(Each TodoList has many TodoItems). Routes and Controllers for these two models have been configured so the frontend can call the API to retrieve and update the data as needed.

On the frontend, the react application is able to make API calls to the backend, and the appropriate UI has been set up so that the user is able to perform the CRUD operations. Material UI was used on the frontend, as they have an extensive list of UI components with styling, which helped speed up the development.

2. Basic Authentication has been completed
On the backend, models for Users has been created and a many-to-one association has been set up between Users and TodoLists. Routes for creation of users, and getting JWT tokens back have been set up. Routes for CRUD operations on TodoLists and TodoItems have also been protected and need a valid JWT token to work. The Knock gem was used here to help with the generation of the JWT tokens.

On the frontend a basic UI for signing in/up and logging out has been implemented. Upon signing up/in the JWT token is stored in localStorage. When the app mounts, localStorage is checked for a token so that the user doesn't have to login again.

Next Steps

1.Creation of tags and filtering todos based on the tags
On the backend, I intend to create a Tags Model and set up a many-to-many association between TodoItems and Tags. The filtering logic will also be put in the backend, with the filter parameters being passed through the API by the frontend.

On the frontend, I need to add a search bar and other appropriate UI to facilitate CRUD operations on tags by the user.

2.Ability to have subtasks for each todo item
I am still exploring the best way to implement this functionality in the backend and to display it nicely on the frontend. On the backend side, it should involve adding a parent attribute to each todo item which holds the id of the parent todo item and also a type attribute, which indicates if the todo item is a parent or a subtask.

3.Proper Error Handling
Currently any errors from the API are not properly handled, and this is bad for user experience, because either the app might crash with no backup view or certain buttons may just seem unresponsive to the user(when in fact there was some error in the response). So any errors need to handled properly in the front end.

3.User Interface and Styling
Material UI is very helpful since they have many common components and inbuilt styling. But this also makes the application have a very "default" look at the moment. So more focus needs to put into the styling and also designing the user interface to make things feel intuitive for the user.

Main Challenges/Further Exploration

1.State Management.
Even though this is a small app, and there's not too many moving parts, I felt my state management getting quite messy and I found myself passing down props from one component to another. I read about contexts in React, to avoid having to repeatedly pass down props, so currently my code uses them, although I am not entirely sure if I have made the best use of them. So I will be further exploring the best practices for state management in React applications and also the use of Redux.

2.API Design

Currently my API is rather basic in nature, with the responses just containing the data needed. I understand there are specifications and good practices as to how the API should be designed and the JSON responses should be structured, so I will be looking into those and seeing if I can make any improvements.