Computing for Voluntary Welfare Organisations (CVWO) AY2021/22 Riding on Rails: Final Writeup

Student Number: A0229816W
Name: Huzaifa Raghav

Summary: My submission does not implement user authentication, so the API and front-end only support CRUD on tasks and categories (tags). However, I learned to implement the rest of my desired features with some small changes. I also hosted the front-end and back-end separately on Netlify and Heroku respectively.

Changes from mid-assignment plan

Backend: I moved some of the nested routes under /categories/:id/tasks to /tasks as I found this fit better with state in Redux Toolkit.

I did not implement user authentication and user-segregated tasks, as I wanted to prioritise learning to implement the core features first. I also included a "priority" field in the Task schema but did not implement related functionality in the front-end.

Frontend: I wanted there to be a default category (e.g Inbox in Todoist) if no other categories were created. However, I decided against this as I felt a real user would be confused by this. Instead, the site displays a dynamic welcome message based on the number of categories. I also added more sort options for newest/oldest tasks first (in terms of update order).

Learning points

Ruby on Rails: Since I needed a CRUD RESTful API for this assignment, a lot of the work on the API came from writing migrations to define the schema for each model and then configuring allowed routes in routes.rb, with much of the controller code being pre-generated. It was surprising how purpose-built Rails is for CRUD tasks, and it helped me to learn about relational databases.

TypeScript: I tried to use TypeScript types for most functions and React component props. However, I avoided using it for some custom hooks and props when the type was too complicated (e.g a dynamic Formik object), and also did not use React.FC to type components[1].

[1]https://github.com/facebook/create-react-app/pull/8177

Redux and Redux Toolkit: The main application state for tasks, categories, and loading status was implemented with Redux and Redux Toolkit, which is the recommended way to write Redux logic[2]. Using the React-Redux Hooks API is also recommended[2] so all components were written as functional components.

Libraries: I used Bootstrap and MUI for the UI, Formik and Yup for forms, axios to interact with the back-end API and date-fns to handle date parsing and formatting.

Separation, hosting of back-end and front-end: I separated the front-end and back-end code into two completely different repos instead of using the react-rails library. In the front-end, I wrote async functions (APIService.ts) to interact with the API that return Promises for use in Redux async thunks. The front-end is hosted on Netlify and the back-end API is hosted on Heroku. In the future, it could be good to learn how to host both parts separately, but on the same server, to reduce latency.

Future code improvements

Modal and form re-use: All forms in the application are inside modals triggered by buttons. However, I also wanted the form to reset whenever the modal is closed. This leads to an issue where the modal depends on a Formik resetForm function, but the form is a child component of the modal. I tried to implement custom hooks and components to enable modal and form re-use, but this cyclic dependency was an obstacle to doing so. I hope to learn to resolve this in the future.

Processing form submission: In the current implementation, form values are formatted to be called with dispatch on an action creator, which then calls the correct function from APIService to interact with the API. However, when trying to change the API endpoints later on I realised this made changes more difficult as validation has to be updated in two places. Instead, passing form values directly into the action creator would mean formatting only needs to happen once.

User manual

Thank you for using the Task Manager. When you visit the site you will see a welcome message once loading is complete. The dark sidebar on the left is used for navigation.

The first two buttons are to see all tasks across categories (**‘All tasks’**), or to see them grouped by due date (**‘Upcoming’**). A dividing line separates these buttons from the category names which when clicked take you to the page for the respective category where its tasks are shown.

[2]https://redux.js.org/style-guide/style-guide

When using forms, if fields have issues appropriate messages are displayed which must be resolved before submitting.

Categories - Creating, renaming, deleting

Create: Click the '**Add Category'** button in the sidebar. Type the new category's name in the form displayed, then click '**Add category '** below the form. You will be redirected to the new category's page.

Rename: Go to the page for the category. Click the '**Rename'** button at the top of the page. Type the new category name in the form displayed, then click the '**Rename category'** button.

Delete: Go to the page for the category. Click '**Delete'** at the top of the page. Click the '**Delete category'** button on the warning modal that is shown. This will delete the category and its associated tasks, and redirect you to the welcome page.

Tasks - Add, edit, delete, sorting, search

Add: Go to the page for the category the task should be in. Click '**Add Task'** at the top of the page. Fill in the fields as needed, then click '**Add task'** below the form.

Edit: Click the '**Edit'** button inside the task's box. Fill in the fields as needed - you can use the '**Category'** dropdown to switch a task's category. Click '**Edit task'** below the form when done.

Delete: Click the square checkbox next to the task's name.

Sort: Click the nearest '**Sort by'** button, depending on the page, to see sort options and click the desired option. The order of tasks in the list below will update.

Search: Click the nearest search field, which has the text **'Search for task'**. You can search by name or description, case insensitive, and the task list below the search field will update accordingly or display **'No tasks to show'** if no tasks matched your search.