CVWO Mid-Assignment Write-up

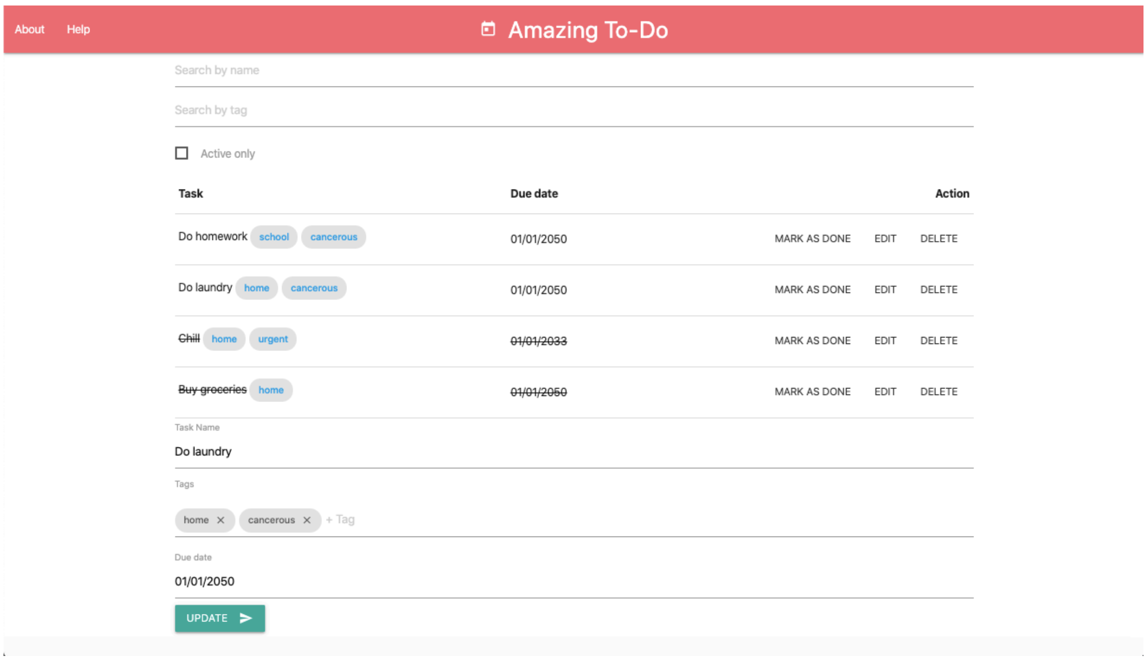
Ho Si Binh – A0201332A

I. Basic use cases

CRUD Operations: Users may view existing tasks, add new tasks, edit existing tasks and delete tasks.

Execution plan: I intend to develop my application as a single-page application using Rails as a REST JSON API. Basic CRUD operations are done via corresponding HTTP methods using fetch. At launch, only the most recent tasks are loaded.

Create and Update operations make use of similar input forms. React conditional rendering is used to provide them as needed. Input validation is currently done in the frontend, but I am going to add server-side validation also should users accidentally manage to overcome the frontend validation.



Search by name

Search by tag

☐ Active only

Task	Due date	Action
Do homework school cancerous	01/01/2050	MARK AS DONE EDIT DELETE
Do laundry home cancerous	01/01/2050	MARK AS DONE EDIT DELETE
~~Chill~~ home urgent	~~01/01/2033~~	MARK AS DONE EDIT DELETE
~~Buy groceries~~ home	~~01/01/2050~~	MARK AS DONE EDIT DELETE

Task Name
Do laundry

Tags
home ✕ cancerous ✕ + Tag

Due date
01/01/2050

UPDATE

At the moment: All tasks are loaded at launch.

Search through tasks: Users may search through tasks by name.

Execution plan: For simple searching through recent tasks (downloaded by the browser), handle in frontend so that the page does not have to reload. Provide an option to search through the entirety of tasks in the database.

At the moment: Handle in the frontend, similar to the technique in the Thinking in React tutorial.

Tagging of tasks: Users may label tasks with tags and search through them.

Execution plan: To create the current tagging frontend, I used Materialize CSS. Tags are stored as a string in a "category" attribute, using whitespaces as delimiters. This might be problematic if users have lots of tasks and tags due to regular usage. Moving on, I am going to create a tag model and use relational database concepts to improve lookup.

Similar to searching by name, searching by tags is currently handled in the frontend. Searching is currently limited to a single tag. I am going to figure out multi-tag search upon the addition of the tag model.

Mark as done: Although not a basic operation, I find this in most to-do lists. The app allows the user to mark tasks as done and strike them out, instead of just deleting them. This closer mimics the experience of a physical to-do list, and would, arguably, provide a greater sense of accomplishment on completing tasks.

Execution plan: Add a "done" attribute to the Task model. Render rows conditionally with strikethrough effect, similar to the technique in the Thinking in React tutorial.

II. Suggestions (Subject to changes)

Advanced searching and sorting features: More options to search through and sort tasks to serve users' needs, such as by due date, creation date, etc.

Export to calendar: Export tasks to common calendar formats so that users may add to their own calendar.

Save and clone previous task lists: For repeated tasks, it might be helpful to reuse previous task lists instead of adding tasks from scratch.

Notification: Notify users when tasks are about to be due through some channel.

User authentication: Necessary if intended for use by multiple users. Might learn to implement one or use the Devise gem.

III. Problems facing:

User interface: Currently, the task creation form is at the bottom of the page. Although I have little knowledge of user interface design, I find this is rather awkward when tasks build up as the user has to scroll down every time to create or edit tasks. I will find a more graceful way to handle it.

As I am new web development, initially, I spent lots of time learning the basic concepts and tools of modern web development, such as MVC; Ruby on Rails; relational databases; HTML, CSS and JavaScript; ajax and REST APIs; React; version control using git, etc… As such, I could only mostly work on the frontend thus far.

I created the react and rails apps separately. I learned of ways to set up react and rails as one app, which I might look into as they might be neater.