CVWO Takeaways

Name: KIM Jungbae

Matriculation Number: A0194438U

Afterthoughts

As I went to serve for the Korean army after year 1 (and before then I was not even a CS student; I transferred my course just last semester), my knowledge of programming was very faint.

Through this assignment, I had to self-learn basic Git, Ruby, HTML / CSS, JavaScript, Rails, and React, which were extremely intensive since I only had around a month for winter break. I devoted six to nine hours every day, catching up with all the readings, and solidifying my understanding of the topics by doing little projects (The Odin Project helped significantly as a roadmap).

I could only start building for this project after the semester started (before then I was learning), which posed a great time constraint. Some of the functionalities I hoped to implement were not done as a result, but I have managed to implement **all the major features** for my app. So I feel very proud of my accomplishments.

Below are some more technical details outlining the things I have implemented, things I have not implemented (but provided a basic idea of how to go about implementing them), further areas that require more study, and a brief user manual.

What were achieved and not achieved

Based on the use cases explained in the mid-assignment

1) User creates a new to-do list. (DONE)
2) User views which tasks are set. (partially DONE)
    a) User can view tasks that are due today, and can also view tasks by category (including completed tasks), but not by date.
        i) Viewing by date would be very similar to viewing by category, with a little different search condition (on dates instead).
3) User searches for an existing task by its task description. (NOT DONE)
    a) Two ideas about implementing it.
        i) Handle search on React; retrieve all the tasks and then handle it through filter method.
        ii) Handle search on Rails; send necessary params and get back only relevant data.
            (1) Much preferable if the number of tasks is large.
4) User makes modifications to an existing task. (DONE)
5) User completes an existing task. (DONE)

    a) Custom route action had to be defined along with an appropriate controller action.
6) User de-completes an existing task. (NOT DONE)
    a) This, I believe, requires an advanced association on Category model, Task should have current_category_id and before_category_id as foreign keys.
7) User deletes an existing task. (DONE)
8) User deletes an existing category. (DONE)
    a) All tasks are moved to Default category, so the users will not lose their tasks by accidentally removing a category.

Areas for improvement

1. Devise forms implemented via Rails
    a. Was easier to use pre-defined helper methods that came with Devise gem.
    b. Still, can be done on React too! But need some work on password confirmation field.
2. Error handling from Rails validation needed in the future
    a. **.catch** after fetch call did not work.
    b. When creating a new category, restrictions were written on the page itself, but errors should still be shown for the convenience of the users.
    c. Includes receiving an error response and handling it on React.
3. Need more understanding on the use of CSRF token
    a. **protect_from_forgery with: :null_session** solved the problem, but still need more study on the use of that code (perhaps related to website security?).
4. Making Rails load associated records in advance as well
    a. This is particularly useful for categories column (with number of tasks indicated), I can simply count the length of the tasks for that category's tasks (instead of manual filter).
5. Handling static CSS files
    a. When deploying to Heroku, static CSS files were not loaded automatically, I had to write **stylesheet_link_tag** for all my CSS files as a result.
    b. I also found all my CSS files were applied to all my React components, even if I only imported necessary ones (perhaps a problem related to the use of Webpack?).
    c. As a consequence, I had to name my HTML elements very specifically, but later I found using CSS selectors I can still use general class names but still apply styles to specific HTML elements (e.g. by having an element with multiple classes).
6. React Router
    a. Router was not really necessary in my app as my app operates on a single page, but it would be nice to learn if I need to direct my web app to multiple other subdirectories.

User Manual

Welcome! My DaySharpener app is simple to use with very few instructions.

First, sign up for the website! You can use your username for login later.

On the website, you'll see the "Create Task" button at the upper right. Click it, tag your task to an appropriate category, type in your descriptions, and select the due date. Viola! Your task is ready to be sharpened.

You can also create a new category to tag more tasks! Click on the "Create new category" button on the left column, type in your category name, and that's it! You have a new category ready. Wish to delete a category but worried about losing all tasks under that category? Fret not, all your tasks will be moved to Default category automatically!

You can view your tasks via the main page, which shows tasks that are due today. Also, click on the category name you have created, this will present you all the tasks that belong to that category.

Made a mistake while creating a task? Don't worry! Simply locate the task and click the "Edit Task" button, you'll be able to edit it just like how you made it. Even more, you can simply delete the task by clicking the "Delete Task" button on the right.

Are you done with your task? Congratulations! Simply click the "Complete Task" button and the task is now considered complete! Try clicking the "View completed tasks" button, this will show you all the tasks you have completed so far. Be proud of your achievements!

And that's it! Now you have mastered my app. Continue to sharpen your days with DaySharpener...