CVWO AY19/20 - Mid-Assignment

Sebastian Toh Shi Jian (A0196545R)
https://github.com/sebastiantoh/todo-app

Basic Use Cases

1. Encapsulate data into a Task.
    a. Compulsory Fields:
        i. Title (represented in database as **:string** field with minimum length 1, and maximum length of 50 characters)
        ii. Description (represented in database as **:text** field with minimum length 1)
    b. Optional Fields:
        i. Tags (represented in database as **:tag_list** – an array of tags)
            - Details of how tags are implemented are abstracted away with the use of the ActsAsTaggableOn gem
        ii. Due Date (represented in database as **:datetime** field)
    c. Default Fields:
        i. Task Creation Time (represented in database as **:datetime** field)
            - By default, tasks are rendered according to the task creation time in ascending order
        ii. Task Updated Time (represented in database as **:datetime** field)
        iii. Completed (represented in database as **:boolean** field, which is initialised to **false** upon task creation).
2. CRUD operations for tasks and tags
3. Backend API Endpoint
    a. Tasks: **/api/v1/tasks**
    b. Tags: **/api/v1/tags**
4. Ability to tag tasks
    a. Add dropdown for existing tags, showing their frequency. Sort tags based on decreasing frequency, then alphabetically in ascending order.
5. Mark tasks as completed, along with option to hide tasks that are completed
6. Tasks filtering (search bar for filtering tasks based on title, description, and tags)
7. Custom tasks sorting (sort based on title, due date, number of tags)
8. Indication for tasks which are over-due
9. Allow users to undo Update and Delete operations to tasks
10. Timestamps to indicate task creation date and last updated date

Execution Plan

Tasks and Tags CRUD Operations	
Backend: Create Task and Tag (use ActsAsTaggableOn gem) model	✓
Backend: Create REST API endpoints using controllers so that frontend and backend can communicate via JSON	✓
Frontend: Render tasks and tags for end-users to read.	✓
Frontend: Render forms and buttons for end-users to create, update and delete tasks and tags	✓
Tasks	
Input Validation: both frontend and backend	✓
Add notification upon successful task creation/update/deletion	✓
Ability to mark tasks as complete	✓
Add due dates	
Add indication if task is overdue	
Tags	
Make tags case insensitive, store tags as lowercase in backend, and render them in lowercase	✓
Prevent storing of duplicate tags	✓
Add dropdown for existing tags and show their frequency	✓
Sorting	
Sort tasks based on task creation time (default)	✓
Add option to sort tasks based on title, due date, or number of tags	
Sort tags within dropdown menu based on frequency (descending) and alphabetically (ascending)	✓
Filtering	
Filter tasks based on title, description, or tags	✓
Add some indication if no tasks match filter requirement	✓
Add ability to hide tasks that are completed	
Optimization	
Mobile Optimization	
Cross-Browser Compatibility (Firefox, Chrome, IE10 and above)– fix all console warnings (if any)	
Refactor code – make components leaner, abstract out duplicate code	
Make sure code is properly documented	
Add typechecking with React PropTypes	
Hosting and Deployment	
Seed database with tasks	
Host on Heroku	
Other miscellaneous: (if time permits)	
Improve website accessibility and UX	
Add ability to undo Update and Delete operations to tasks	
Add timestamps to indicate task creation date and last updated date	
Add ability to archive tasks	
Create new view for archived tasks	

Current Challenges

1. Testing
    a. Often, when I make a git commit thinking that I've successfully implemented a feature, I will chance upon a bug when implementing the next feature. Currently, I test each feature manually, and while I try to be systematic in testing each feature, my tests are often not comprehensive enough, resulting in bugs after I've committed my code.
    b. I hope to adopt Test-Driven Development, to solve the above problem. However, I'm unfamiliar with writing tests in both React and Rails, and it may take some time for me to be familiar with doing so.
2. Unfamiliarity with Rails (and Ruby in general)
    a. Because of my unfamiliarity with Rails, I tend to write most of my code in Javascript. Thus, most of the rendering is done client-side, instead of server-side.
3. Uncertain of best practices concerning UI, UX
4. Over-reliance on stateful components, when certain components can be written as stateless components