CVWO Assignment 3: Mid-Assignment

Shen Yichen

This assignment will be about making a simple TODO manager with Ruby on Rails. The basic functionality should allow users to create, modify, delete and label their TODOs.

Use Cases

List

The main and default interface of the app should be the TODO list screen. Here, the TODO tasks are listed with the closest due date on top. A user may perform actions on their TODOs using icon buttons next to the individual TODOs.

Checkboxes will be present next to each list item. Checking the checkbox will label the TODO as done, and will shift it out of view. A switch or collapsible element will be present on the list screen, which will allow the users to show or hide TODOs that have been marked as done.

Labels

A list of labels should appear in a side bar. After clicking on the labels, the user should be brought to a page containing a list that is visually and functionally similar to the main list, but filtered according to the label he/she has selected.

Labels will also have a administration page, where users can add or remove them.

Creation, Edit, and Delete

Users should be able to create new TODOs via a simple form. The form allows the user to enter a short summary (as the title), elaboration, due date, as well as specify the labels.

A quick input box might be present in the list view too, which allows users to quickly add TODOs with a simple title, without any other details.

Editing a TODO takes place on the same template as the creation page. A user may also assign labels or remove them post-creation.

Posts can be deleted with a icon button next to them on the main list.

Implementation

Backend

The backend for the basic features will be straight forward. Using Rails, 2 resources will be created — likely with `'generate scaffold'` — for each of TODO and label. They will then be linked with a many to many relation.

The when dealing with labels, the filtered version of the TODO list can be shown with the "show" route for a single label.

For completed TODOs, they will be marked with a property done, and will be excluded from a normal query, but included if the user specifies it.

Frontend

The frontend will be done with Materialize CSS (http://materializecss.com/). Each page will be templated with a navbar at the top and a sidebar for navigation. The different labels will be present in a collapsible element in the side bar, each of which can be clicked to view the TODOs under that label.

At the most basic level, forms and UI elements will simply be done with the Rails erb templates, using the Rails form API. Interaction will then be based on a traditional page based flow. This simple implementation can be replaced with a more complex one — such as one using react.js — if time is to permit. This possibility will be explained below.

Possible Extensions

This section will cover the possible extensions to the above use case and implementation.

Users

It may be possible to introduce users to the app, and have the TODOs belong to a certain user, effectively allowing the app to be used by multiple users simultaneously.

The first way is to simply implement a traditional login system. Then both TODOs and labels will be related to a single user, effectively creating a seperate environment for each user. The implementation will follow https://www.railstutorial.org/book/log_in_log_out roughly.

The second way is to implement a cookie/session based list. A user is assigned a session key on login, and all TODOs and labels will be tagged to this session key. The user remembers the URL for his session, and can access the session by entering the key as part of the URL. The result should be roughly like how NUSMods work, though the implementation might prove troublesome.

react.js

react.js maybe considered for parts of the frontend. The most obvious location for this addition would be the main TODO list. Instead of getting rails to render a full page, we could get react to make a AJAX call to obtain JSON on the TODO information. As a result, marking as done or deleting TODOs can be handled by react, eliminating the need to render a new page every time an action is performed.

Deployment

Phusion Passenger might be considered for web deployment. I currently have a server hooked onto NUSNET, and have always been curious about how the deployment side of Rails was handled. If time permits, I would look into setting up the app on my own server, which can then be accessed via the NUS network.