CVWO Task Manager Final Write Up

User Manual

About:

This is a task manager with support for many users.

Types of users:

1. Normal User

Services provided by the app:

1. Make multiple lists to manage tasks
   Link: '/lists'

2. Create a task to be completed
   Link: '/tasks'

3. Add tags to a task
   Link: (tags can only be added/modified when creating a task)

4. Change attributes of task
   Link: (go to individual task)

   Details: Users can modify three attributes of their tasks: 'done/undone', 'starred/unstarred', 'archived/unarchived.

5. Search tasks
   Link: '/tasks/search'

   Details: Users can search through their tasks. They can add modifiers to the search such as tags, done/undone etc.

Accomplishments

1. Basic understanding of the MVC framework.

While working the with the LAMP stack, I was not well aware of a good way to organize my code. So often, I would be confused about what the control flow of my app was.

After using Rails, I have a much better understanding of the control flow in a web application. Rails separation of the model, view and controller also greatly helped me improve the efficiency of my application as I it became easier to locate where the mistakes/ bugs were hiding. (as opposed to LAMP stack where finding the source of the error was painful).

I also appreciated the idea of separating the backend from the front end which rails does by splitting the controller and the view. Through this assignment, I am better able to write more organized and human readable code.

2. Convention over configuration

Rails philosophy of favoring convention over configuration was very difficult to understand and appreciate at the start. I had much difficulty to create the first working prototype of my app because I could not internalize the fact that rails prefers and understands certain name patterns over others. Because of this, I had to spend a lot of time learning how to name variables and helpers in Rails.

But towards the end, as I began to use the API more comfortably, I kept thinking of how convention can actually be a better idea than configuration. This is because code is often read a lot more than it is written. Hence, enforcing a particular convention will make it much easier for other users to understand my code and work together with me.

This philosophy of rails helped reflect on the way I usually organize my code and how to do so to enable greater code readability.

3. Good database design

Rails made it easier to implement linked tables in the database with useful (has_many and belongs_to) relationships. Because of this ease to link tables, I was able to learn and implement foreign keys and joins more effectively. With this, I was able to minimize redundancy in storing data (unlike my weblog in LAMP). I was also able to manipulate data in other tables with more ease. Using foreign keys effectively also simplified the process of deleting dependent records when the parent record was deleted.

The main learning point for databases was that I should try to link my data better (with better hierarchy) such that I can manipulate the data more effectively and reduce redundancy.

4. Good code organization

Rails DRY philosophy enforced me to keep my code organized better so that I reduce duplication.

Often I found myself repeating code for my controllers and views. I did not realize this as a problem till later when I had to make changes to my view and controller design. Because I had repeated my code in several places, I had to fix each one separately and often missed out somewhere. Also, it was harder to find bugs because the same code was duplicated at many places.

As the problems grew in number, I decided to use helper functions for my controllers and layouts for my views. I realized that good abstraction of code can increase efficiency as bugs become easier to find and fix. Also the code becomes more easily maintainable and readable as well.