SE31520: Enhancing the CS-Alumni Application

Due on Monday, December 10, 2012

Alexander Brown

Contents

Introduction

Server Architecture

Having worked in a Level 3 Service team, my attitude toward changing existing code is not to do so unless it is completely necessary and at that, to modify the code as little as possible.
The first change I made was to make a simple addition to the UserController which would echo the username of the currently logged-in user. This allowed me to *a*) confirm the user has been logged-in; *b*) check if the user is an administrator (currently this is defined by the username being equal to "admin").
This change involved adding a single method (and associated `GET` route) which only rendered JSON (i.e. it would only accept content-type JSON).
The only other change I made to add functionality to include the details of a user's image in the JSON file from the `GET` request to `/user/:id.json`.
Figure 1 shows the overall class diagram for the CSA Server.
As is apparent from figure 1, the architecture of the server hasn't changed at all. The only additions are those mentioned above (the addition of a single `GET` route, and some additional information in the User's JSON file). Other than this I felt no real need to change the architecture of the server, nor the way in which it performs the current functionality.
I did have to fix some small pieces of code in the way emails were produced and sent to make that code actually work. This was part of fixing the tests on the CSA server.

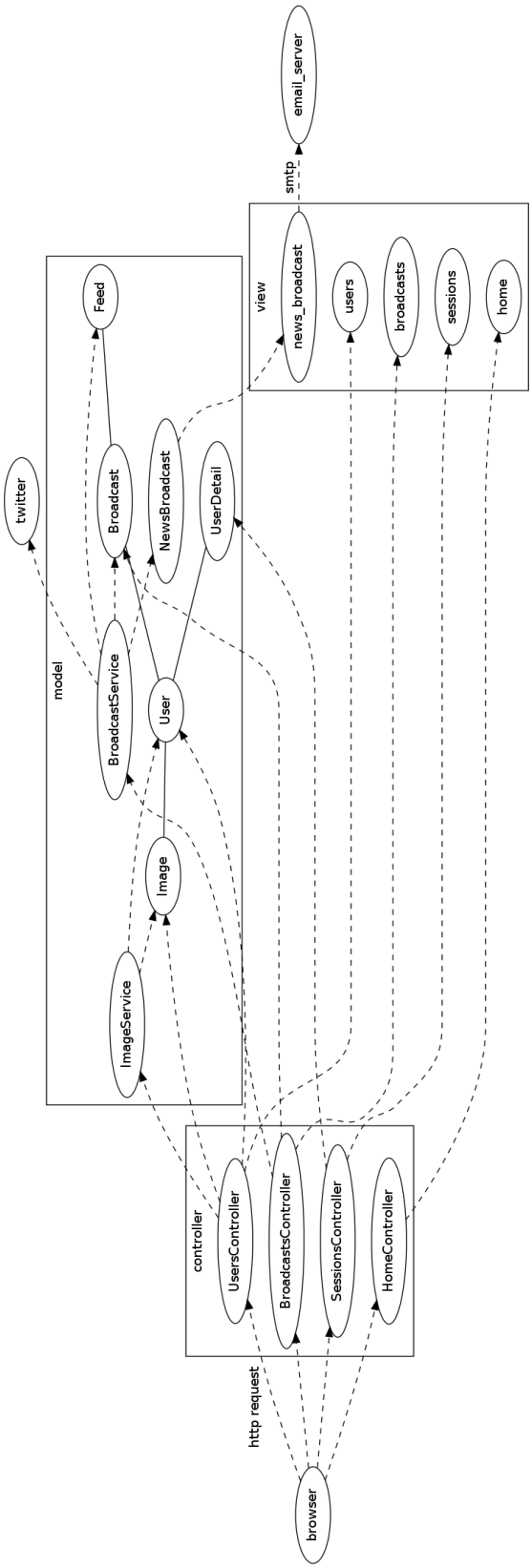

Figure 1: Class diagram for the CSA server

Client Architecture

As the client doesn't need to do a lot I decided to keep it fairly simple. I still tried to follow the Model, View, Controller (MVC) design pattern as best as I could. Separating out a lot of the REST interaction into a sole class.
This also means it would be relatively easy to swap out this method of interaction without too much effort. The views take up the majority of the class diagram as there are different ways of presenting the same information. However even these have been kept as simple as possible.
Figure 2 shows the UML Class diagram for the client.

Figure 2: UML Class diagram for the client

From this I then started to consider the choice of language. Ruby was the obvious choice to keep a bit of compatibility between client and server. I also found that Rails has a gem which provides client-based REST access to a Rails application; ActiveResource. This saved me a lot of time and effort during implementation, not having to work out the exact, correct, REST requests and the form the JSON files passed between the two take.
Ruby also has some nice graphics libraries, FXRuby is one which is cross-platform, with a lot of good documentation, based on the FOX toolkit for C++.

Test Strategy

The first part of the strategy was to fix all the failing tests on the CSA server; the most of these were caused by there not being a logged-in user when performing the actions. It was a simple matter of creating an entry in the `user_details` table in the test fixtures. I then got this entry and added it to a fake session Rails testing provides to assist with this scenario.[1]
With this the majority of the failing tests then passed. The only other broken tests concerned emailing out broadcasts. This was caused by a problem with the code to do this, as mentioned in the Server Architecture section.
With this complete I then turned my attention to testing the CSA client. However, when I started to work out what needed testing I found out there was very little. It seemed pointless to test the ActiveResource models in the client as they were provided by an external API which should be well tested. The only thing I could test is that the methods for logging-in and checking administrator users to make sure the GUI only enables the correct options.
The final step was to look back at the server. As all the client does is poll the server for information using ActiveResrouce it stands to reason that if the server is correctly tested, the client should work correctly with it. Of course in real life it's never quite that simple, but just running and using the client should be enough to verify that it's working correctly.

Evaluation

References

[1] G. V. Buren, "Passing sessions and referers in rails functional tests." http://garrickvanburen.com/archive/passing-sessions-and-referers-in-rails-functional-tests/, October 2007.