Adam Nelson

CS 360 Math Lab Queue (final project)

Report

12/17/13

- **Description**: Explain the purpose of the site and what functionality you implemented.
    - The purpose of the site is to allow students to sign up for assistance in the BYU math lab using a mobile device, including a laptop, tablet, phone, etc. Eventually, the app could be extended to generate reports for use by those in charge of the BYU math lab.
    - I did a little bit of everything, but I focused my work on the communication between admin users and the database through the UI. That is, I designed and set up the ability for admin users to edit user permissions. I had to learn a lot of new technologies to be able to do this. Using coffeeScript, I was able to set up a system wherein whenever a user clicks a checkbox the database is dynamically changed. This is done using ajax in my coffeeScript. Understanding the flow of data in this highly structured framework took me hours to understand and implement. I was pretty proud of myself and quite frankly a bit surprised at my ability to get this to work. This was probably the hardest thing I did for this project and most time consuming. To see this code, look in app/views/admin/edit_user_privileges.html.erb to see the view that calls the coffeescript and ajax in app/assets/javascripts/sessions.js.coffee, which then calls the appropriate action in the admin controller located in app/controllers/admin_controller.html.erb.
    - While I did not design the schema for the relational database we used for development, I did learn how to use the Rails Active Record Query Interface, which is the abstraction away from SQL code that is used in Rails. This was actually pretty difficult to do. Seeing as I was working in a brand new language in a brand new framework, I didn't have much of an idea of what I was doing. It took a lot of research to be able to effectively query, update, and delete from the database. Frankly, I think pure SQL is easier, but it was neat to see how the ARQI worked.
    - Further, I implemented the code that looks at the user's highest permission on login that determines the home page that the user is sent to, the admin homepage, the ta homepage, or the student homepage. Extending on this, I implemented the ability for users with various privileges to navigate between various user perspectives. That is, for a TA, I ensured that they could see the Student view if they wished, and not the Admin view. Likewise, I ensured that students could only ever see the Student view, and that

Admin users could choose any perspective they wanted to. This took a lot of work too, as it forced me to learn the intricacies of routing.

- After implementing these two things, I implemented the admin users ability to manage the courses in the system. That is, I created the manage_course view, which displays all the known courses in the system, and allows for an admin user to add to or remove from the database any course. I worked on the styling of the add/delete buttons using css and javascript. I've had a small amount of experience with both of these technologies. I enjoyed the small refresher I got in working on this project. Speaking of using new technologies, I also had to read up on HTML, as it's been a while since I've had to work with that. We used HTML quite a bit in the various views. For this feature I decided not to try to make the list of current courses update dynamically. Rather, I chose to write this code more quickly. Every time a new course is added, I simply pull the necessary data fields from the view and enter it into the relational database using an action in the controller. Next, the controller action renders the same page, but this time since the new course is in the database it is added to the list. The list is small enough that the change looks dynamic, so just to get a working model and to streamline our progress I didn't worry about using ajax. I did something pretty clever (I thought) as well. If you inspect the page's elements you'll notice that the table of courses has a hidden <td> item in front of every <tr>. This holds the id of the course in the database, which means nothing to the user, but is necessary for me to remove a given course from the database. There is a potential security flaw here, but given it's only accessible to admin users anyway, I wasn't too concerned with it. Please don't dock points because I admitted to using a hack! ☺

- I then went through the application and made sure that the design was all uniform. I had to change a lot of things in the student and ta view so that they would match up with the interface on the admin view. This included adding the home button to pages that were one step or more away from the users default home page, and making sure that the current view name (admin, ta, or student) is always visible on the top of the screen just below the page name. Some of the page names weren't being displayed in the dark blue bar at the top of the screen, so I changed those as well. After making all the pages uniform to the design that Ryan created and I added to, I moved on to working more with the ta view.

- Others had already done most of what was required for the ta view, but when the queue was empty I noticed a bug: the ta was still allowed to press the "help student" button, which didn't lead anywhere (there was a nil value that caused the program to crash). Therefore, I caused the "help student" button to disappear if the queue is empty, and I made the queue display "none" when it's empty.

- I also went around fixing the small bugs that I noticed along the way. For example, instead of allowing any user request to appear in the active help queue I added a filter to the query to the database that restricted the result set to those requests that were active (i.e. that had not already been dealt with). I made sure the user's position in the queue was accurately calculated and displayed as well using the same filter.

  - I fixed a bug where if a user tried to log in to the system and was denied permission an error message was displayed. Then, if the user gave valid credentials, the error message persisted. I made it so that the message disappeared when appropriate.
  - I made other minor changes here and there. I don't remember all of what I did, but those are the main highlights.

- **Database Schema**: Describe how data is stored in your database. If you used a relational database, show your entity-relationship diagram that includes all entities, attributes, and relationships, and explain what it contains. You should use the standard entity-relationship format as shown in class. Use a program to draw this diagram, such as lucidchart or dia. If you used a document databse, show your JSON document structure and explain what it contains.
  - See databaseDesign.pdf on github. Basically, there are the following tables:
    - Users – contains an id, username, creation date, and modified date
    - Courses – contains an id, creation date, and modified date, title, discipline, course number
    - Privileges – contains an id, privilege_type, section, course_id, creation date, modified date
    - Requests – contains an id, a Boolean active field, an owner user id, creation date, and modified date
    - Courses_users – contains a mapping from many courses to many users based on ids
    - Privileges_users – contains a mapping from many privileges to many users based on ids.
- **Contributions**: Describe in detail the contributions made by each member of the project. Each member is responsible for contributing a few paragraphs in this section.
  - For an in-detail description about what I did, see the first section.
- **Future Work**: Describe a roadmap for future development, with additional features you could add or changes to the interface.
  - Adding a confirmation message for when an admin user wants to delete a course from the database would be nice.

    - Allowing users to generate reports based on past help requests would be beneficial, and probably not too hard to do, but it would be time consuming.