homepage

Table of Contents

Project Description

People generally go to their web browser to seek the same type of information from the same sources on a daily basis. Many people use the default start page of their browser as their homepage yet this rarely offers quick, adequate access to the information of these sources. Homepage is an alternative start page where people can have a highly customizable dashboard featuring widgets, news feeds, quick access bookmarks, and a simple search bar. Homepage has the potential to greatly improve the everyday web browsing experience.

Process

We followed the XP process this semester for developing homepage. We worked in pairs on different sections of the code for each iteration. At the beginning of an iteration, we held a team meeting where we planned what each group would accomplish and estimated hours needed for each portion. We did have some issues with following the iterative process, mainly with estimating the time needed to finish some use cases. Early in the project when team members were learning Angular, integration into the site took much longer than originally anticipated. This meant that some use cases had to be moved to the next iteration, while others got dropped. Since homepage is a website and most of the code is in html/css/javascript/php, there was not very much object-oriented type refactoring that could be done. Much of the javascript relies on asynchronous function calls with nested callbacks which is really hard to refactor. We had no issue in refactoring the php portion of the code however since php is an object oriented programming language. Each group handled refactoring by reviewing their previous iterations' code and ensuring it followed the proper structure and format. The testing was done in phpunit, Selenium, and qunit. Each group wrote back end tests for all php classes in phpunit, while front end functionality was tested with Selenium. The few javascript files were tested with qunit. We had a few issues initially with testing in trying to get the libraries working on everyone's setup as well as picking the right framework. With regards to group collaboration, we scheduled an in-person meeting once a week to discuss progress and report any issues for each iteration. We also set up a group chat via Slack where we could talk about the project outside of meetings. The main issue with our collaborative development was choosing the right time to work as a pair or to finding additional time outside of our weekly meeting to work as a team.

Requirements & Specifications

Listed below in Table 1 are the use cases and user stories implemented on homepage.

Use Case	**User Story**
Page Layout and Styles	As a user, I want to see my widgets, news feed, and bookmarks graphically from the dashboard.
User Creation System	As a user, I want to be able to create an account and view a customized dashboard.
Login System	As a user, I want my dashboard configuration and settings to be saved from session to session.
Logout System	As a user, I want to be able to logout from homepage and return to the default page.
Editable Widgets	As a user, I want to be able to choose which widgets appear on my dashboard, and save this configuration across logins.
Weather Widget	As a user, I want to be able to see the current, high, and low temperatures at my current location for the day as well as the current time in one widget.
Calculator Widget	As a user, I want to have a calculator widget that can perform basic algebraic math.
Search	As a user, I want easy access to internet search when starting my browsing experience.
Twitter Feed	As a user, I want to have the ability to login and see my home Twitter feed on my dashboard.
Quickbar	As a user, I want to be able to have quick access to frequently visited sites on my dashboard with the ability to add/delete sites.
Instagram Feed	As a user, I want to have the ability to login and see my Instagram feed on my dashboard.
Stock Ticker Widget	As a user, I want to be able to view stocks in one of my widgets on my dashboard.
Gmail Widget	As a user, I want to be able to view my gmail inbox information in one of my widgets on my dashboard.
Bookmarks Bar	As a user, I want to be able to view, add, and delete the

	bookmarks in the folders of my bookmarks bar on my dashboard.
Todo-List Widget	As a user, I want to be able to maintain a todo list on my dashboard, saving my tasks across logins.
Random Background Images	As a user, I want my dashboard to have a random background image.
Generic Widgets	As a user, I want access to more generic widgets such as Spotify and a calendar utility to provide more options when selecting widgets.
Default View	As a user, I want to have default setting for widgets, bookmarks, the quickbar, and the feed to display when I am not logged in or when I first create my account.

Table 1: All implemented use cases and user stories for homepage.

Architecture & Design

The structure for homepage is an AngularJS front-end combined with a PHP back-end that communicates with a relational database via MySQL.

Inside our back-end, various objects such as a `User` or a `Quickbar` are stored in the database with tables names “Users” or “Quickbars”, respectively. If one object contains another object, such as a `User` having a `Quickbar`, then the ID of the `Quickbar` row in the Quickbars table will be referenced by the corresponding `User` in the Users table who owns that `Quickbar`. When a user boots up homepage in their browser and they are not logged in, a default `User` is created for them in the PHP back-end containing a default `Quickbar`, `Bookmarks`, `Widgets`, and `Feed`. However, if they are logged in, a new `Session` is created and their preferences are loaded into a `User` object by PHP that will contain all the user’s referenced objects. When the `User` model and all the referenced objects have been created, the `User` model is loaded into a JSON file for Angular to parse and manipulate.

Inside of our front-end is a JSON file that contains all of the information for the current user. This data is echoed in by PHP as stated above. AngularJS then reads the JSON file of the current user and loads in the correct widgets and bookmarks (bookmarks bar and quickbar) in the corresponding areas of the site. If a user is logged into Twitter or Instagram, the feed will display their customized feeds, otherwise a default is displayed by AngularJS. The customized homepage is now visible to the user with all of their saved preferences. The AngularJS framework will handle the user interface from this point on. When the user modifies their preferences, such as creating a new bookmark or selecting a new widget, AngularJS will send off the new information as a form to a PHP controller script on the back-end that will update the values in the database, sometimes done asynchronously through AJAX and otherwise just done synchronously.

The choice to use PHP and AngularJS for our application was a very effective choice because of the smooth handoff ability between the two. PHP handles the form processing and database interactions with ease, then places the necessary information into a JSON file for AngularJS to immediately read in the changes and efficiently manipulate the DOM for the user’s user interface. If the user needs to change a preference, log in, or log out, AngularJS can seamlessly

hand off the information to PHP for back-end processing. The two work very well together and allow the homepage project to work elegantly and efficiently.

Shown below are UML diagrams for the various parts of the system. Figure 1 contains the structure for the entire system, while Figures 2 through 5 show each individual component. homepage contains four main components: the feed, the quickbar, the widgets, and the login system.

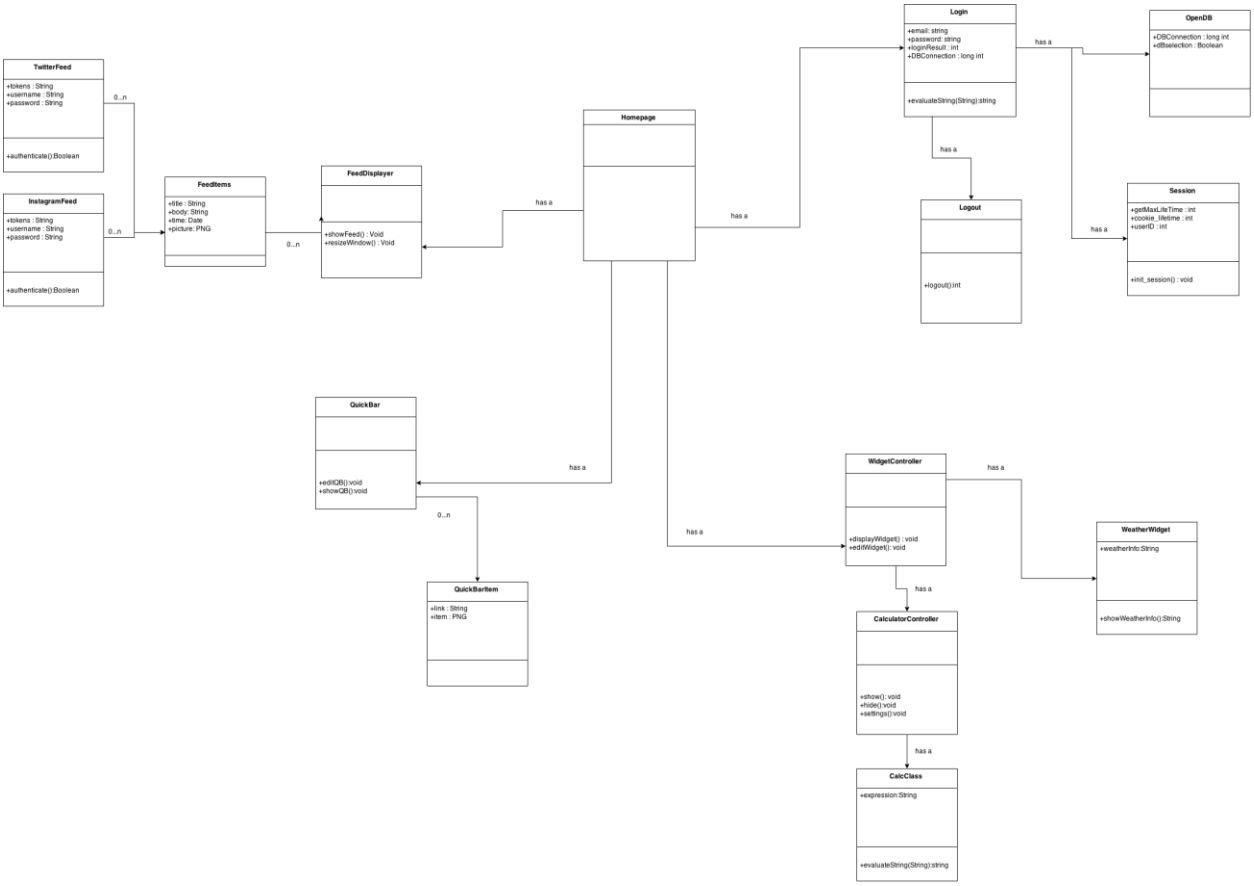

Figure 1: Total system architecture consisting of four main components: the feed, quickbar, widgets, and login. Figures 2-5 are larger images of each component.

The feed contains a `FeedDisplayer` which displays generic feed items on the dashboard. These `FeedItems` are generated by either the `TwitterFeed` or `InstagramFeed`. This structure is shown below in Figure 2.

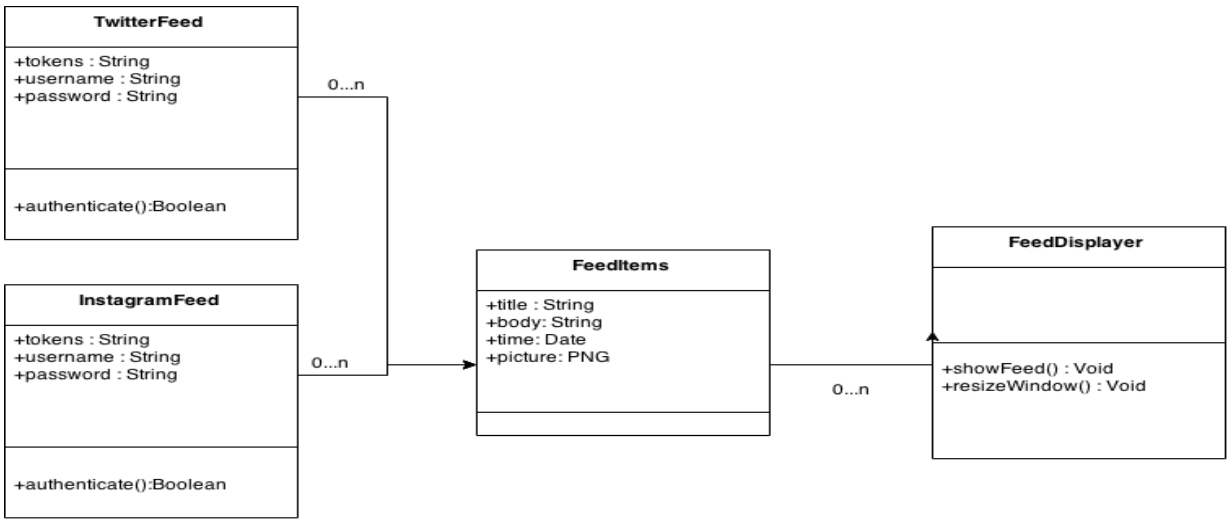

Figure 2: Architecture for the feed implementation. `FeedItems` interface is populated by the individual `TwitterFeed` or `InstagramFeed`.

The quickbar is populated via Angular on the dashboard. The backend code simply consists of a `Quickbar`, which itself contains the various `QuickbarItems`, as shown in Figure 3.

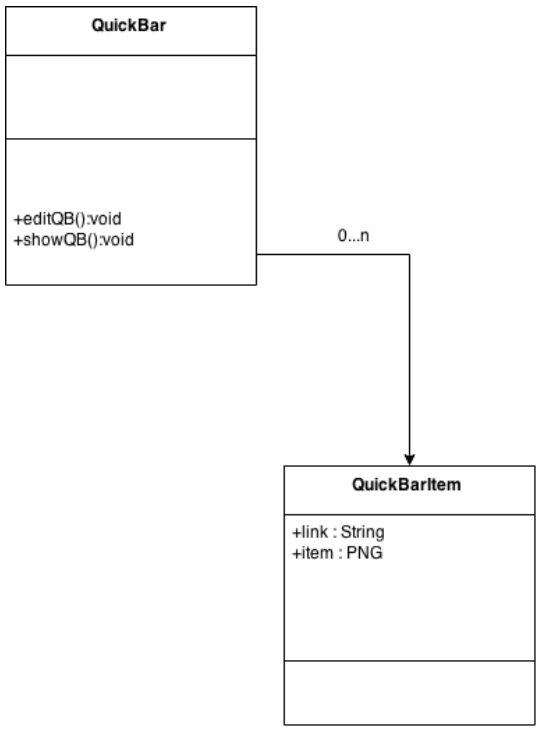

Figure 3: Architecture for the quickbar. The `Quickbar` for any user consists of `QuickBarItems` for each entry.

The widget structure is shown in Figure 4. The calculator widget is an example of any of the selectable widgets. The weather widget on the other hand always is present on the site.

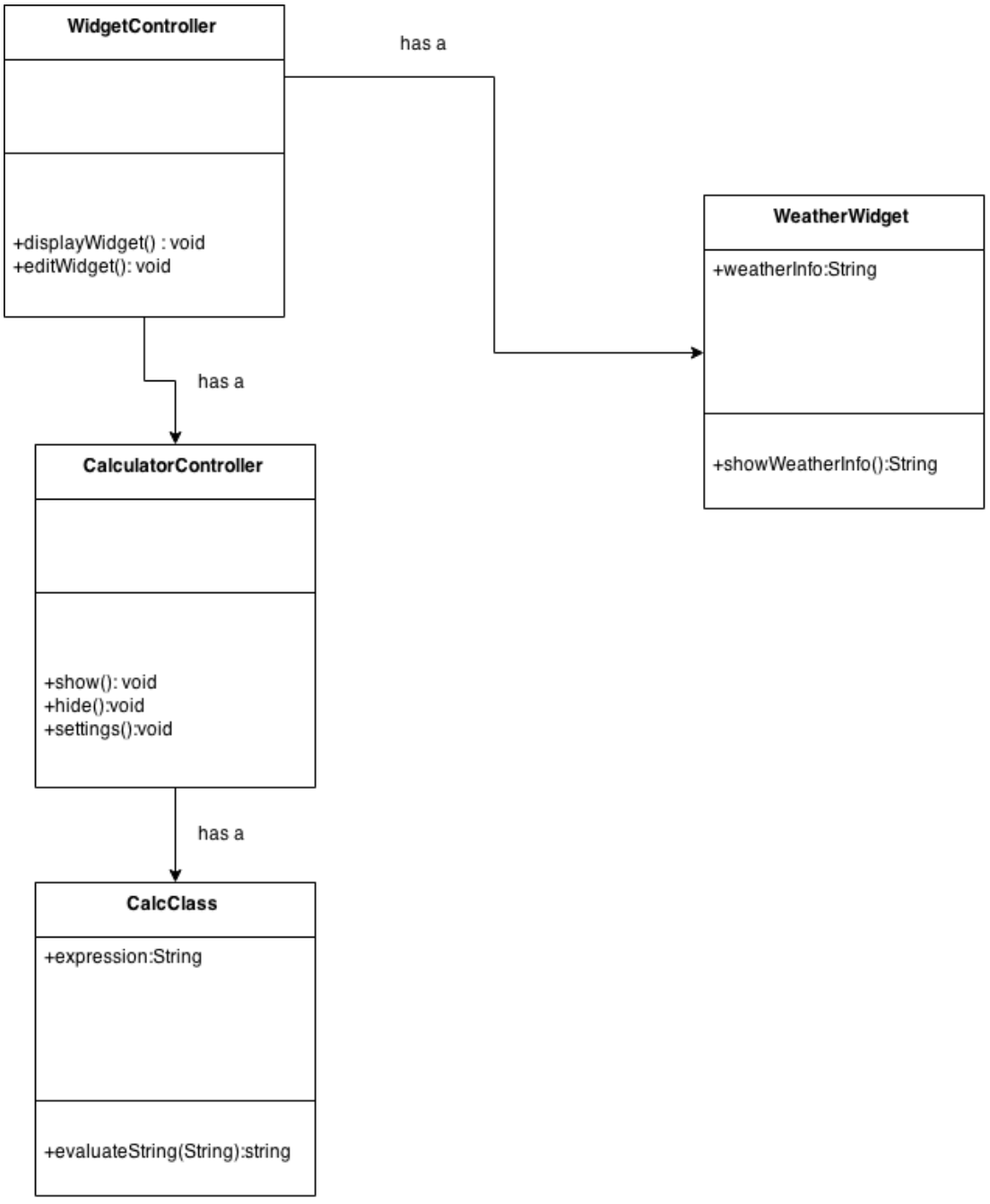

Figure 4: Architecture for the widget classes. The weather widget is always present, while the calculator classes are representative of any selectable widget.

Figure 5 shows the UML structure for the login system. A `Session` is created for each `User` to store their `userID` for future database accesses.

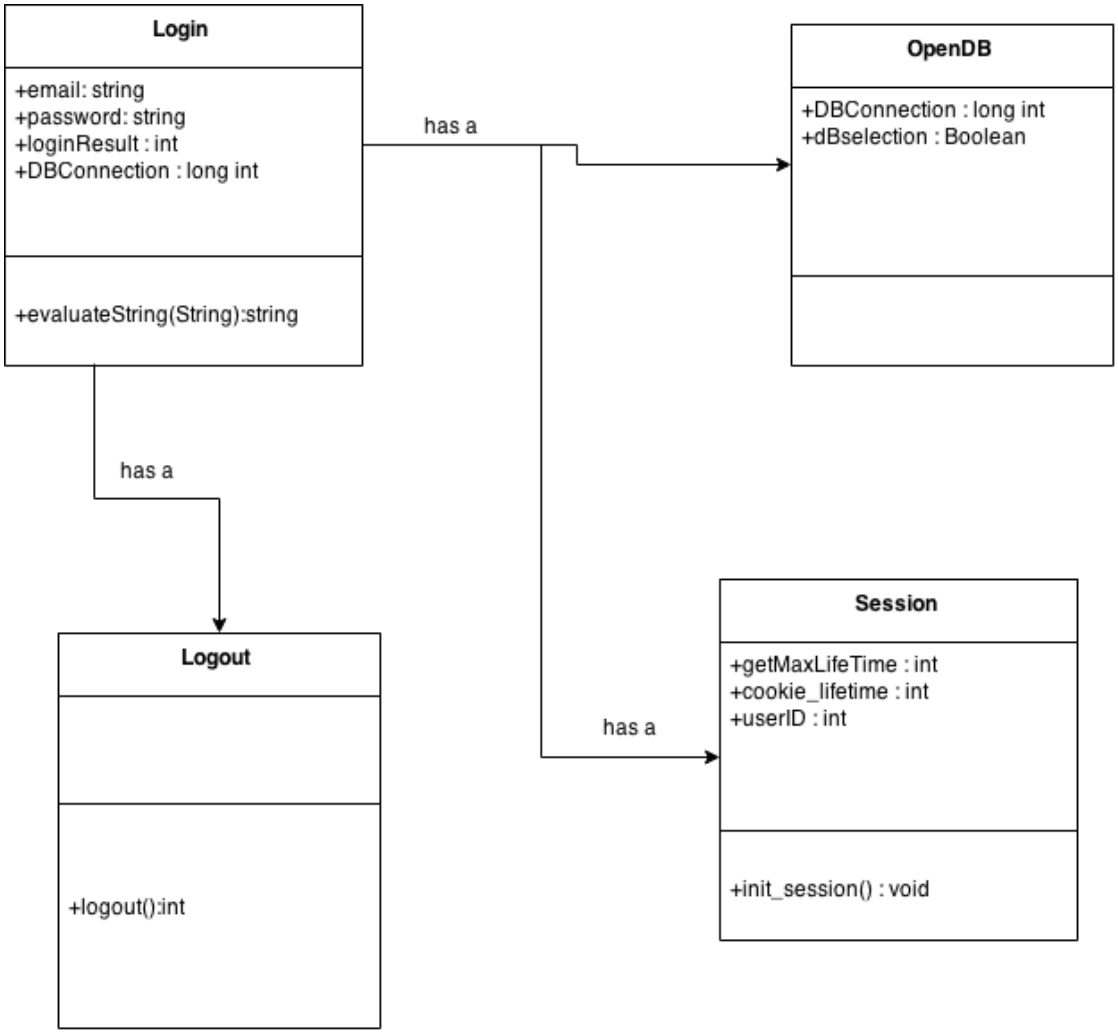

Figure 5: Architecture for the login system. A `User` is tied to a `Session` in this manner which is used in loading/saving their preferences.

Future plans

Nick

If we have more time, we would like to create more widgets as well as implement more social media feeds into the dashboard. We would like to implement a cache layer for the quickbar icons so we put less strain on the grabicon API. In addition, we would like to add daily themes for the background and color scheme for more customizability on the dashboard.

Brian

Reflecting on the process, one major thing that I would have changed was our usage of the Selenium Firefox plugin for front-end testing. In retrospect, it is not a very robust testing tool and has many timing issues that make writing good front-end tests almost impossible. The other thing I would change is our `User.php` class. This "god-like class" housing too many closely coupled classes. I would decouple all the associations of quickbar, todos, and bookmarks.

Adam

One major issue that should be addressed in the future of this project is changing how user preferences are loaded and saved. The `User.php` file is way too large and the load and save methods need to be broken up into the individual components. The quickbar, bookmarks, and wiki preferences should not all rely on this one method in User.php. This involves a large refactoring of the code but would make it more easily testable and reusable in the future. In general, more planning of the overall structure of the code would have been beneficial because every group working in parallel on different sections of the site made it challenging to integrate. Additionally, a different test suite should be used instead of Selenium. It relies too much on timings and is not as reliable as it should be across different machines and platforms. In terms of additional features for the future, incorporating other social media sites such as Facebook and creating more widget options should be high priorities.

John

In the future, I think we should add a more customizable UI. Although we have customization for types of widgets, we could further extend the customization and allow users to customize every single widget itself. We could improve the Gmail & Instagram widgets by allowing users to send or post content. These would be minor changes. One big thing we could implement is to let users save their customized desktops and share them with other users, creating a page of "Top Desktops" that would help new users get started with a great looking version of Homepage. There could even be categories for different types of users, for example, a "Science" Homepage which has a lot of Science/Engineering widgets. Or a "Music" Homepage that is based around Spotify, sharing what local bands/artists are performing. Or even a "Disney" Homepage that is catered toward kids. We could even create security settings for parents and let Homepage be a "safer" place for your kids to start browsing the internet.

Dion

If we have more time, I think we could develop more widgets and implement feeds from other social media as well. Reflecting on the development process, I think that the SVN process is good since it makes sure through testing and pair programming that our code runs well. I learned a lot about javascript this semester, previously I have near zero experience in creating websites.

Bill

If we have more time, I think we could refactor the individual widgets and feeds to be better integrated. Right now, the widgets are just templated in using Angular, with no persistence unless the creator of the widget chooses to hack it in. With more time, we could expose an interface for the widget creators to store data without being forced to directly interface with the database. With feeds, most of the feeds use OAuth, which means most of the feeds can share code for authenticating and storing tokens. However, right now each feed has its own set of code for authentication and storing tokens.

Reflecting back, I feel as if following the XP process was well worth the effort. However, while we were so absorbed with the need of the user through use cases, we did not consider the need of ourselves as developers. As a result, many of the issues as outlined above by me and others occurred. One of the things I learned through this class was how to balance the needs of the developers with the needs of the user, which sometimes do not line up perfectly.