

Cougs in Space

Design Document

10/16/2017

Version <1.0>



SSS

Jared Strand

Luke Seo

Rahul Singal

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. INTRODUCTION	
2	
II. ARCHITECTURE DESIGN	2-
3	
II.1. OVERVIEW	2-
3	
III. DESIGN DETAILS	3-
5	
III.1. SUBSYSTEM DESIGN	3-
4	
III.1.1. HTML User Interface	4
III.1.2. JavaScript Interface	4
III.1.3. Python Database Server	4
III.2. DATA DESIGN	5
III.3. USER INTERFACE DESIGN	5
IV. TESTING PLAN	5-
6	
V. REFERENCES	
6	

I. Introduction

The purpose of providing this design document is to illustrate and outline the main component and subsystem diagrams that will be used as a guide to serve as a visual aid for our web development project. The goal of this project is to build a website for the WSU Cougs In Space Club. We want to build this website so that the site administrator will be able to post updates about events and happenings from within the club. The website will mainly serve to keep current and potential members informed with the current projects of the club and progress within each team. It will also serve as a platform to attract donors and keep past donors informed with where and how their money is being utilized.

Section II includes Architecture Design

Section II.1 includes the overview of the software system

Section III includes Design Details

Section III.1.1 includes HTML User

Interface Section III.1.2 includes

JavaScript Interface Section III.1.3

includes Python Database Server Section

III.1 includes Subsystem Design

Document Revision History

Rev 1.0 10-16-2017 Initial Version

Rev 2.0 11-6-2017 Iteration 2 Version

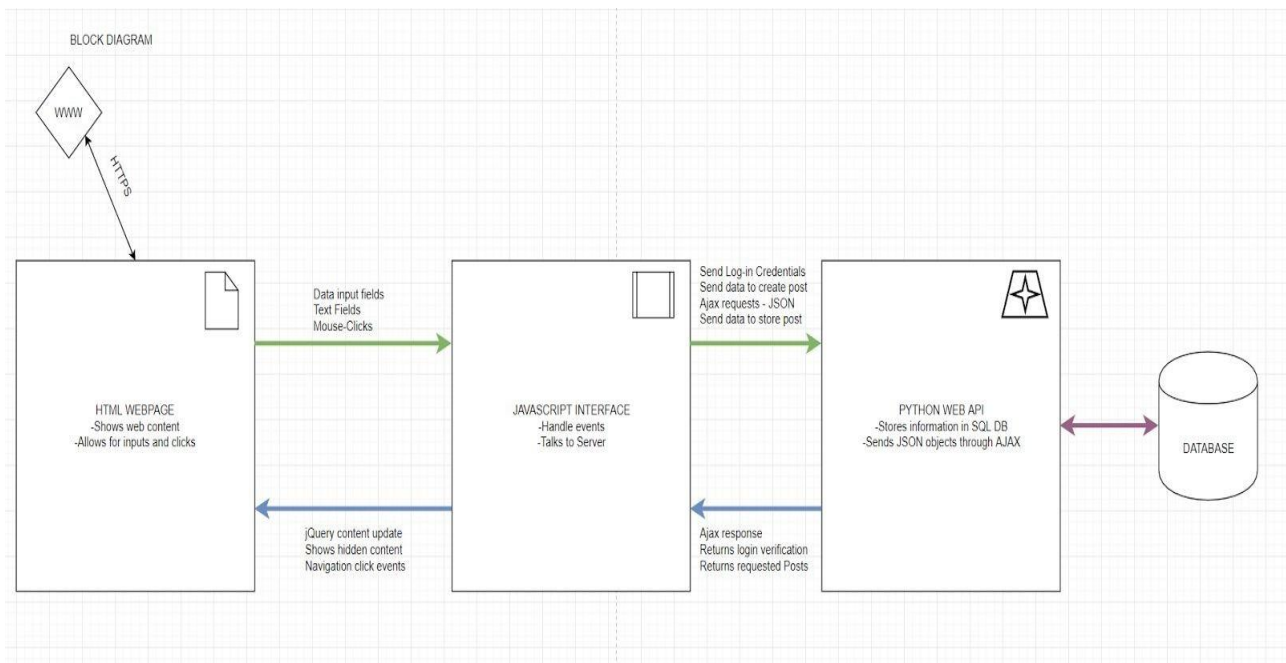
Rev 3.0 12-6-2017 Iteration 3 Version (Highlighted)

II. Architecture Design

II.1. Overview

Our software system will be a website designed to handle basic navigation and content post updates per team page that is implemented. We have decided that the best way to serve up a solid website to our end user is to build three main components, or subsystems that will work cohesively within the entire website system structure. The first of these subsystems is going to be our user interface. This user interface is going to be written in HTML with the help of the Bootstrap libraries and grid layout CSS styling. Our user interface will be designed in such a way that will allow it to have many interactive buttons and action events that users may trigger with mouse clicks and keyboard input. Our second sub-system will be the JavaScript database interface and event handling software. This JavaScript sub-system will allow the user to modify certain input data fields and handle many mouse button clicks on certain pages. In this subsystem we will also make extensive use of the jQuery libraries to dynamically update our HTML front end content. Our third and final sub-system interface is the python database server interface. This API will be designed to work with the JavaScript and jQuery commands from the second interface to store and withdraw data from an SQL

database. For our software system we have decided to implement a Model View Controller architectural pattern. The rationale behind choosing this pattern was the fact that our website will need to run interactive commands on the client side and be able to make requests to a database structure on the server side. With the Model View Controller architecture pattern, we are able to process and handle many user interactions and interfaces on the client side and minimize the amount of server-side computation that must be handled.



III. Design Details

III.1 Subsystem Design

There are three subsystems within our architecture which includes the front-end UI with HTML/CSS/BootStrap, the JavaScript database interface and the Python flask backend server. The front-end UI is implemented using HTML and BootStrap. We utilize the BootStrap grid system to implement the navigation bar and the sidebar so the user can easily switch between various pages through the website. In the Requirements Document we explain that the users the website will target are admins/team leaders, potential donors, WSU student and K-12 students. In the navigation bar we have links that are aimed the respective users such as a Get Involved page for WSU students which informs them on how to become a part of the club.

The second subsystem in our architecture is the JavaScript database interface which handles various calls between the client and server. Administrators and team leaders will have special privileges which results in more client/server interactions. Team leaders are able to create/delete/update posts within their respective teams, while the

website administrator will be allowed to do the same throughout all team pages along with modifying additional content throughout the website and linking their Google calendar to the sidebar to insert upcoming events/meetings for the club. We describe these privileges briefly in section 2.1 of the Requirements Document and in use cases 4/5. Most these create/delete/update requests to the server will be initiated by a button, administrators/team leaders will have a special view of the web pages since they'll be allowed to edit. The common user will also make requests to the server when they load a team page, the javascript will call to the server for posts from the team page and sort them by a specified parameter, the server will return a JSON object with posts that contain an id, team, title, update (body text), like count and a created date.

The final subsystem in our architecture is the Python flask backend server which handles all the calls from the front-end UI which is handled by the JavaScript code. The main responsibility for the backend server is to store the data the website will use to display on various pages. There are two database tables that the server will contain which includes the Posts and User table. The Posts table will contain posts created by either the Team Leader or Administrator since they'll be the only users allowed to create/edit/delete content. As mentioned before the posts contain several attributes and each post may contain up to 2048 words. The other datatable table that the backend will handle is the User database which keeps track of all users that will be able to login and create/edit/delete content. When deleting a team's post history the server will return a 200 status along with a JSON object that should contain a field with "status: 1", otherwise it will return a 500 error. Creating a post will also return a similar JSON object, except with all the current posts for that team along with a 200 status, it will also return 500 if there's an error.

III.1.1 [HTML User Interface]

We have the HTML pages set up. The HTML pages will show the content of the pages (right now it is blank, until we get more information from the club). The only input we have set up at this point is the redirect to another HTML page via the navigation bar. Right now there is a separate HTML file for each page, there is a lot of redundant content and we will try to minimize the unnecessary extra HTML files in future iterations. Eventually we will make use of show and hide for webpages. This will show content specific to certain pages and hide content that is not relevant to that page.

III.1.2 [JavaScript Interface]

We have the file structure to use the javascript interface setup. We have not implemented any requests or server interactions yet. Basically the javascript interface connects a request from the HTML to the Server. In future iterations, when a user wants to log in, the javascript

will allow the data to be sent to the server and will receive the server's response. The response will be sent back to the HTML side where it will perform its specified task. The JavaScript will also handle showing and hiding posts for a specific team since it'll all be handled on a single HTML page, similar to how the Smile project hides/shows content.

III.1.3 [Python Database Server]

Currently the Python database server we'll be using isn't running, but we have a general outline for how the Flask database will be implemented and how it should operate. The backend database server will consist of two main database tables, a post and user table. The post table will be similar to the Smile projects table since it will contain a unique id for each post, team name, title of the post, update which is the body of the post, like count and a created at attribute. Then the user table will be used to handle various users that will be logging into the site to create/update/delete content. The backend also handles various routes for the site, one feature of the website is that there'll be a team page that will handle displaying posts for all the various teams in Cougs in Space. When a user clicks on a specific team the JS will make a call to the backend depending on the team they choose and the backend will respond by sending back a JSON object with the posts for that team in order by the specified order by parameter. An alternative to grabbing posts for a team is requesting it from the backend with a specified ID which corresponds to a team's ID. The backend will also handle other requests such as creating a post for a specific team so that it's displayed when browsing that teams page.

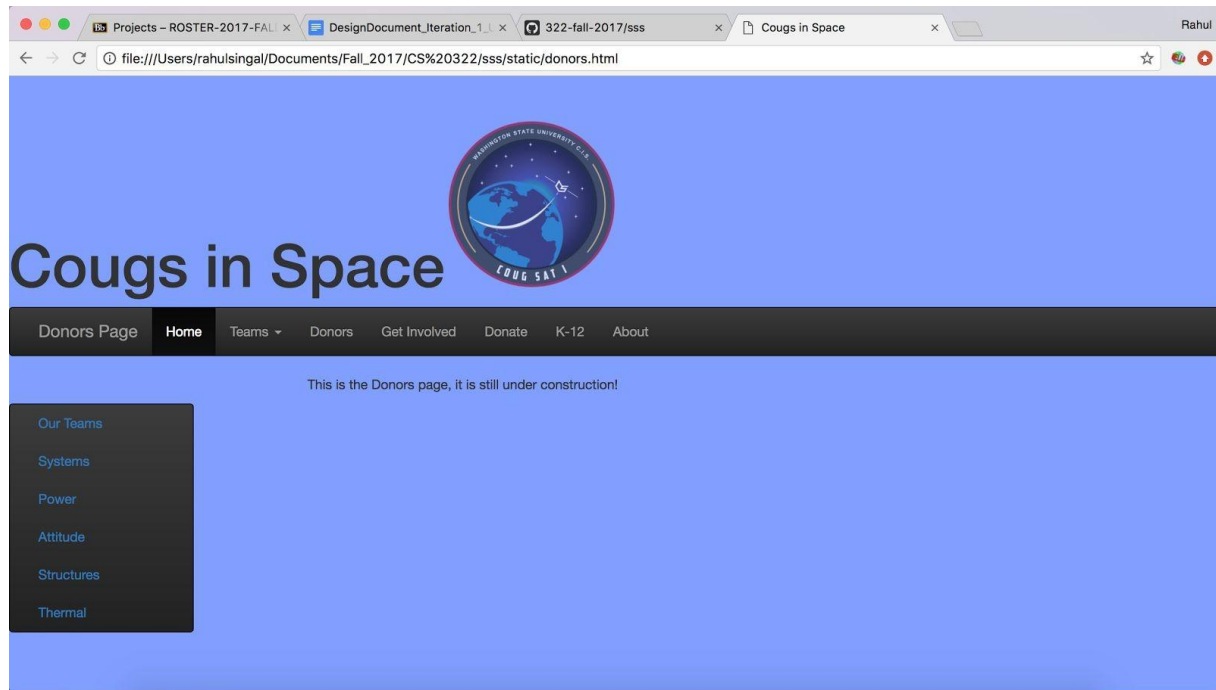
III.2 Data design

For our project we are basing the database tables similar to the Smile project by allowing administrators to post on various pages. This requires a Post database tables that has attributes such as id, team, title, update, like count and created at. This will allow us to utilize the various attributes when it comes to adding, deleting or updating posts by a certain trait. We also include a User database table that will be used to manage users for the website that will be allowed to update/post/delete content from the website. The User database table has only username/password attributes and they'll be used to handle the login of users.

III.3 User Interface Design

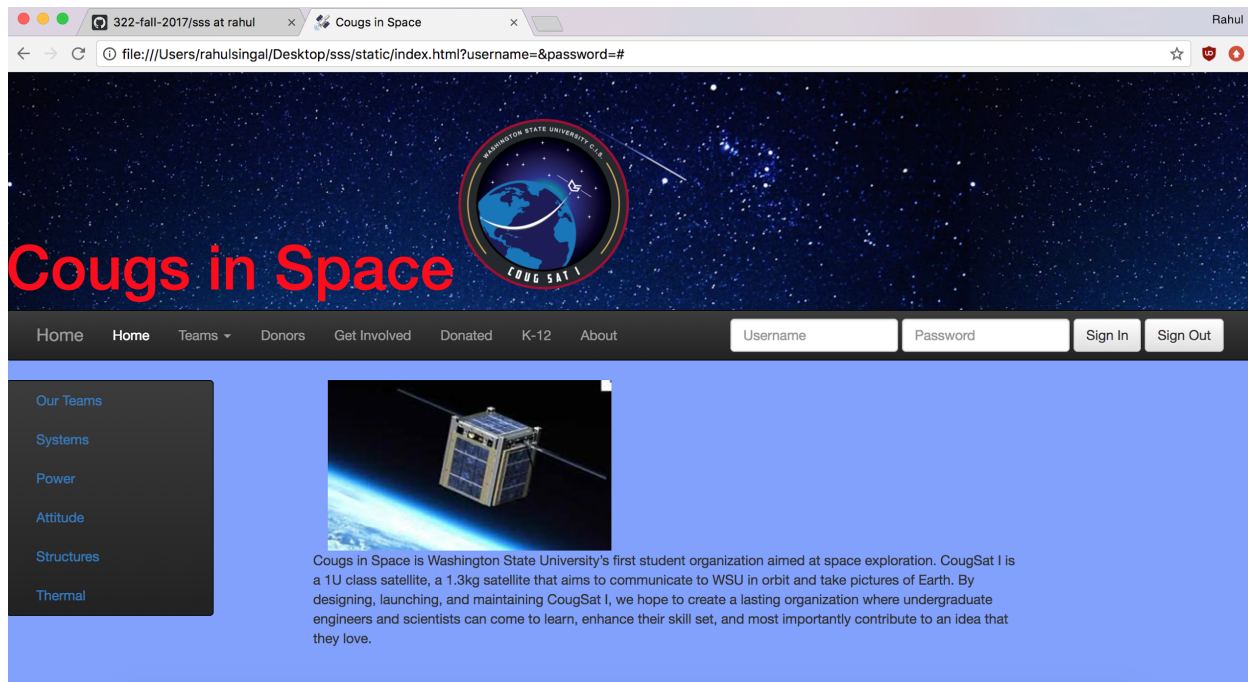
So far for iteration 1 our user interface consists of the following: The home html page has the name of the club alongside the logo. It has a horizontal navigation bar spanning the entire width of the page (even when resized). It has a vertical navigation bar to the teams within the club. By hovering and clicking on each navigation bar option the homepage is redirected to the page specified by the click. Those pages: about, k-12, donate, donors, teams, get involved. Each of those pages have the same navigation bar setup for now. Eventually there will be more content specific to each page once we get the information

from the club. New features will be added as planned out in the use cases. Here is a screenshot of the main page.

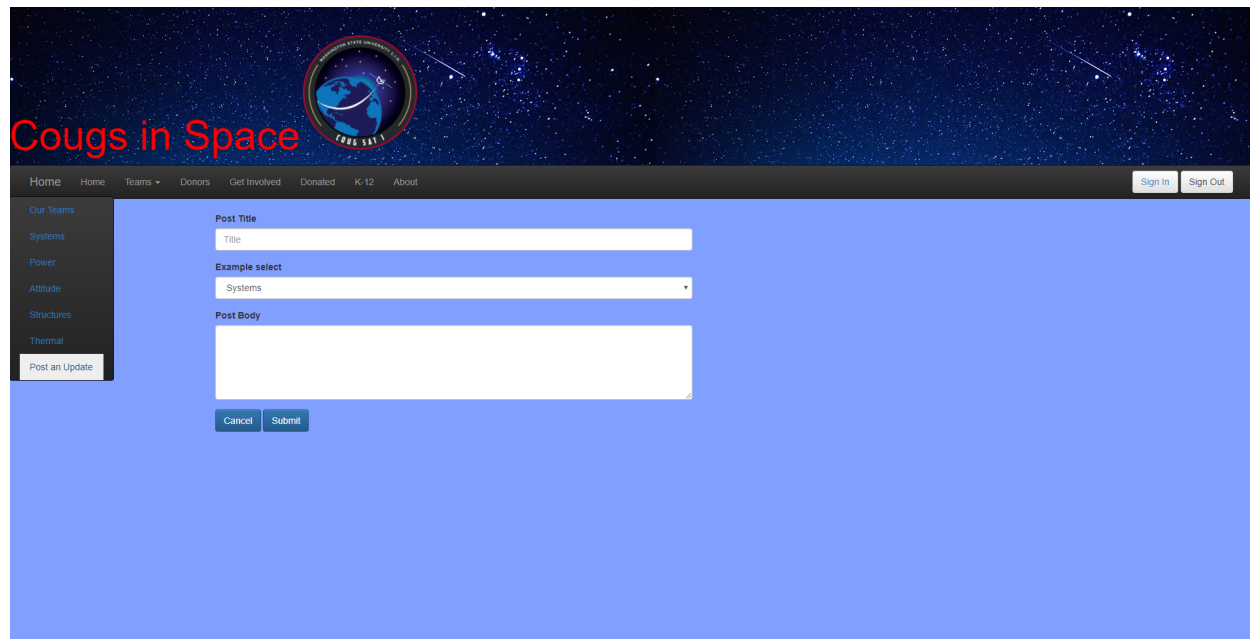


ITERATION 3

At the end of iteration 3 our home page looks as follows:

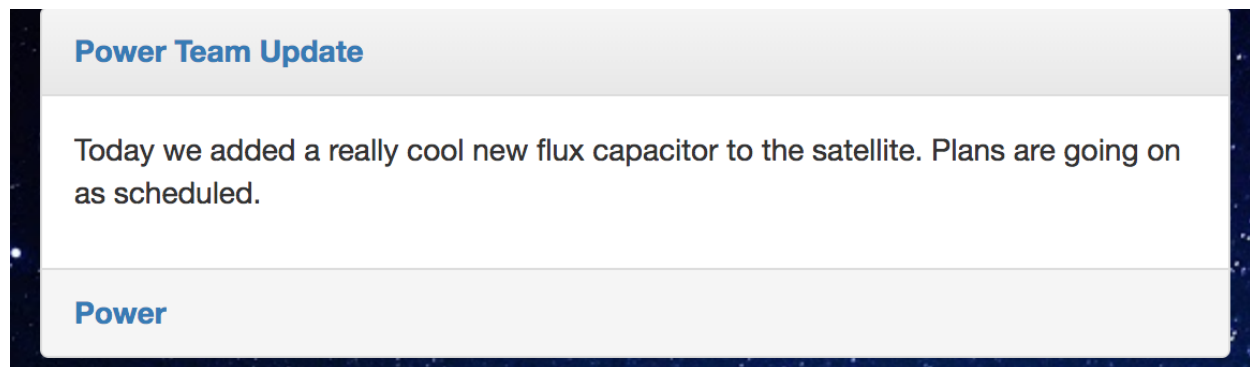


As you can see there is a big difference in the look of the home page from the first iteration to the third one. We have added new stylistic features. We cleaned it up, since there were some UI design issues previously that was causing some issues. The major issues we encountered was when we added new content, the format of the information got affected. For some reason, the picture of the cubesat caused the text to overlap with the sidebar, so we had to do a lot of manual testing to figure out why this issue was coming about.



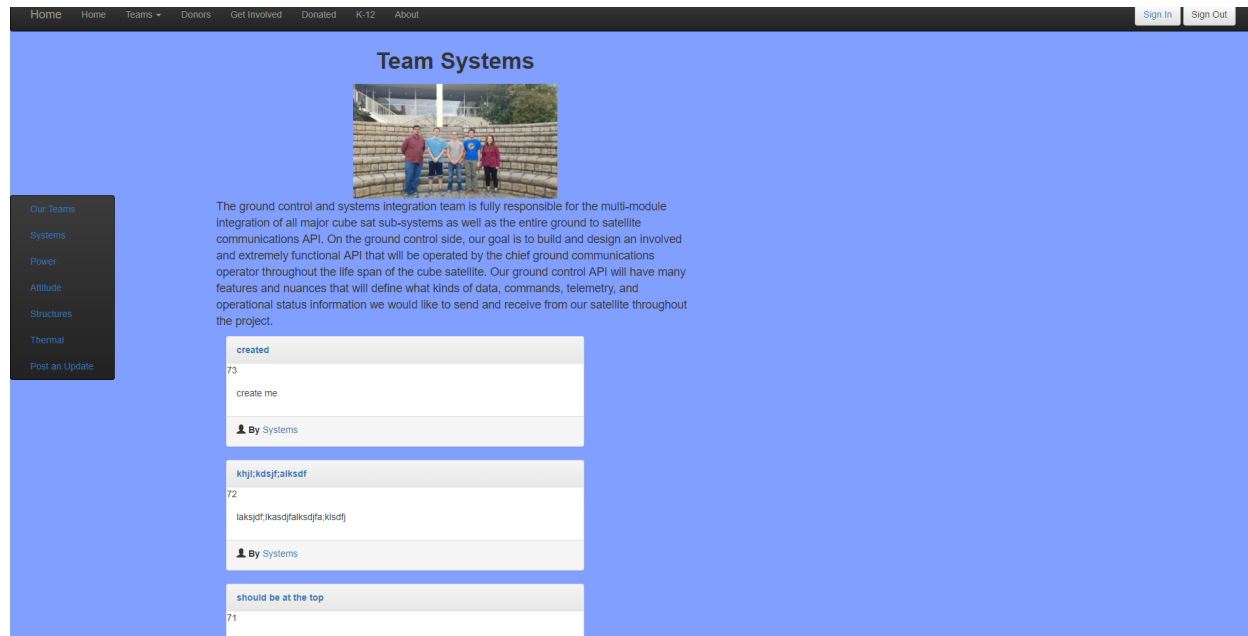
The screenshot shows the 'Cougs in Space' website interface. At the top is a dark blue header with a starry background and a circular logo featuring a globe and the text 'COUG SAT-1'. Below the header is a navigation bar with links: Home, Home, Teams, Donors, Get Involved, Donated, K-12, and About. On the right side of the navigation bar are 'Sign In' and 'Sign Out' buttons. A sidebar on the left lists categories: Our Teams, Systems, Power, Altitude, Structures, Thermal, and a 'Post an Update' button. The main content area is a light blue form with the following fields: 'Post Title' with a 'Title' input field, 'Example select' with a dropdown menu showing 'Systems', and 'Post Body' with a large text area. At the bottom of the form are 'Cancel' and 'Submit' buttons.

The form for the post title and post body is how the team can put an update about the club. They simply enter the information and submit it. Our back end takes care of the rest and will put the information into a form shown below.



The screenshot shows a form titled 'Power Team Update' with a light gray background. The form contains a text area with the message: 'Today we added a really cool new flux capacitor to the satellite. Plans are going on as scheduled.' Below the text area is a section titled 'Power'.

This is the form that the information will be put into. Once it is here, the page will refresh and collect all the posts for the team and list them on their page. The ordering of posts is by time by default.



Here is an example of how the team pages should look. In this example, we are on the Systems Team page. It has a picture of the team and describes what the team is doing and what they are looking for. Basically, the team can decide what to put here. It also has multiple posts!

IV. Testing Plan

Based on the current state of our system our testing plan will consist of using some unit testing features and many functional tests. Since we have not yet deployed our code on an application server service, we are still testing on our localhost machines. Once we have assessed our features in the HTML document and ensured that our content and features have satisfied the necessary requirements from our product backlog we can move forward with more in-depth testing of our software. When we begin to implement our functional testing framework for our frontend and backend API's we will be using Postman to aid us in uncovering bugs in our routes to our server and our data processing scripts. We will also be using user-written test cases to test that our ajax requests and our python routes are executed as expected and yield the proper results. We plan to use ajax requests to our python server to handle the GET/POST/DELETE requests from our front-end. Using postman we will be able to determine that our URLs and JSON files are appropriated and handled properly. Some of the routes that our API will service consist of handling the POST request from the front end that wishes to store a new post for a specific team in a section of the database. Another example of a route that will need to be tested is our GET request. The GET request will depend on a few environmental variables such as, where the user is requesting from, or which team and how many of a post a user is requesting. The main modules that we plan to collect test data from are the server, and the event handlers in the JavaScript files. One way that we plan to test the user interface is to allow test candidates to come in and give us feedback about certain features that they noticed and usability issues they might have encountered. We plan to take the results of these use cases and use the data to help us develop our user interface for our final iteration.

ITERATION 3:

For the Front End:

The User Interface Design required a lot of testing and debugging to get it looking as it does in the image. No tests were written, we basically had to determine why errors in the UI were occurring by trial and error. One major issue was the alignment of the text information. When we added images to the pages, the text would overlap with the image. We assumed that simply applying “overflow: hidden”, the problem would be solved. This was not the case and through debugging we realized that we did not assign the image with “col-md-12”. This is bootstrap specific which indicates that the image needs to take all 12 grids, since the bootstrap utilizes a 12-grid system as explained in their documentation.

For Back End:

The back end was the most challenging aspect of this project. Testing was no exception. We used postman to test our get and post requests. We had a lot of syntax errors when calling our functions as we were working on different machines. Overall, we ran the server and postman while inputting information into the posts and kept debugging till we figured the issues out. We were able to successfully configure the server routes. This allowed us to properly access the correct information in the database based on the query. We needed to use the reversed query for the post collection procedure in order to send back the most recent posts on top. This required testing and troubleshooting the database and optimizing the JavaScript. During the development of the tests in the JavaScript we were able to use the inspector function in the chrome browser to analyze our faults in the code as well as the HTML injection that we performed using jQuery. The utilization of this automated tool helped us to cut down on our JavaScript debugging time. Through the use of various testing tools and automated tests we were able to find bugs quickly and issue the correct remediation's across all integrated software.

V. References

<https://getbootstrap.com/docs/4.0/getting-started/introduction/>