Capstone Final Report

Date: 5/10/2018

Version 2.0

Project sponsor: Barbara Jenkins

Team faculty mentor: Ana Paula C. Steinmacher

BRAINSTIM STUDIOS

Vincent Messenger (Lead)
Anderson Moyers
Andy Salazar
Nathan Franklin

Table of contents

1. Introduction

Alzheimer’s Disease, or AD, is a progressive form of dementia which gradually destroys mental function and memory. It often manifests as short term memory loss like forgetting minor details in early stages and it progresses to pervasive, long-term memory loss like forgetting essential functional tasks and loved ones. In the last stages of AD, cognitive function declines until bodily functions are impaired, ultimately leading to death. As of 2015, there were an estimated 29.8 million people suffering worldwide from AD. It is the sixth leading cause of death in the U.S., a new case is diagnosed every 66 seconds and more than 5 million Americans live with the disease at a cost of $259 billion per year. Without any treatment, those numbers are projected to explode to 16 million Americans with the disease, at a cost of over $1 trillion a year, by 2050.

Research indicates that fortunately, regular cognitively stimulating interactions can reduce the risk of Alzheimer’s Disease. This research has precipitated interest in playing brain stimulation games to keep the brain healthy and game companies have started researching and designing games for this purpose. Lumosity, an online site, is an example of a gaming platform that offers this type of cognitive gameplay.   Lumosity relates their game design to studies conducted on how humans learn: the idea is that by giving users a fun way to challenge their brains, users can keep their brains healthy and reduce the symptoms of degenerative brain diseases.

Barbara Jenkins, our sponsor, has created a fast-paced word game called WordScuffle that incorporates social gameplay in order to provide users with maximum potential for increasing their brain health. The game generates random letter sets with which the user will have three minutes to construct as many words as possible. Users, or “players”, construct words in a grid-like fashion, which allows words to intersect. Once a game is finished, a player’s score is calculated and they can compare their score and words with other players of the game.

There are different game modes that present users with different scoring systems. This forces users to adapt the way they think to successfully solve the challenge that is presented. The game will generate ten letter sets per game-type every day that each user can complete. Once a given letter set has been completed, users can then compare their scores with other users of the game. On top of this, the user will have the option to play unlimited practice games, where unique letter sets will be generated at the beginning of each game. These practice games are not eligible for community comparison.

WordScuffle currently takes place with a pencil and paper, time and scores are manually kept, and results are compared through email. It takes considerable time to tally up scores, scores and results are viewable by players before they may have finished their own tileset, and there is enough entropy in the game’s workflow that more time is spent with minute tasks of gameplay than playing the game. Because much of the gameplay requires “manual” human processing, there are numerous chances for error.

Our team BrainStim Studios is working with Barbara to realize this game as a web application and resolve these workflow problems to make the game more fun, more interactive, less tedious, and even more socially stimulating. Our web application will offer automatic, integrated word validation which will reduce misspelled words. A score calculator will also be updated as users construct words onto their board. Scores will be maintained in a database, where players can retrieve scores and results from other players. Word validation combined with more robust scorekeeping will eliminate human error and reduce entropy in the game. To boot, our scoring system will improve competitiveness because it prevents players from seeing results before they have finished their own set. To enhance social stimulation, we will provide players with a way to add other players, so they can filter high scores to just those they wish to see. Our web application improves on the pen-and-paper version of the game by:

- Automating scoring, allowing players to focus on word combination
- Improve social aspects by controlling tileset generations and high-score viewing, as well as implementing an ability for users to add friends
- Providing word validation to lower confusion and eliminate scoring mistakes

The purpose of the Final Report is to summarize our motivation behind the development of our web application, WordScuffle. This document will include the development process of coming up with the requirements, to the architecture overview, and the testing plan for our web application. There will be a project timeline discussed to elaborate on the deadlines we set in order to release the prototype on time for our sponsor and users. We will be discussing future possibilities that can be made to enhance our web application. Overall the Final Report is a document that will tell the story of our project from beginning to end.

2. Process Overview

Our web application, WordScuffle, was a year long project that took plenty of planning and scheduling in order to deliver the product to our sponsor on time. We had bi-weekly meetings with our sponsor to go over the development life cycle for our web application. After gaining the requirements, we came up with specific roles for each member of our group so we could divide tasks up weekly, to complete the web application on time. For WordScuffle, we had to discuss which technologies we would use to create a well developed web application. Our team used BitBucket to keep the versions of our web application up-to-date on all of our computers. The process overview was very important because it got our team organized in order to develop our web application, WordScuffle.

When our project started, we met with our sponsor, Barbara Jenkins, to discuss the development life cycle of our web application. This is where we went over the key requirements that needed to be implemented from the paper version of WordScuffle, to the web application of the game. After our first meeting, we suggested that we had bi-weekly meetings with Barbara to go over key dates that were important for the year. We came up with the date to release our prototype of WordScuffle by the second week of March. Once we released the prototype, we started our testing which included Unit Testing, Integration Testing, and Usability Testing. Our team then fixed any of the bugs that came up, and released the full web application to our sponsor by the second week of May. Figuring out the development life cycle was an important phase to gather key information to develop WordScuffle.

Once we went over the development life cycle, we discussed the key roles for each of our team members. Vincent Messenger took the team lead, which involves direct communication with our sponsor, Barbara Jenkins. The team lead coordinates task assignments and ensures work is progressing, runs meetings, and makes initial efforts to resolve conflicts. Vince also was the release manager of our project which he coordinated the project versioning and branching. Vince was the lead for the backend development for WordScuffle. Anderson Moyers was the architect for team BrainStim which made him responsible for ensuring the core architectural decisions are followed during implementation. Anderson took the role of front end developer which he worked on the User Interface to enhance the user experience. This role corresponded to Anderson being the release manager for any front end updates to the project. Nathan Franklin took the responsibility of the recorder, which included maintaining detailed meeting minutes, records weekly tasks for all team members. Nathan contributed to starting the deliverables that were due for the Capstone class which he added information about the papers before the team started working on them. Andy Salazar was the website manager which included keeping the team website uptodate. Andy was also involved with backend development for WordScuffle, which was coordinated with Vince when working on tasks. Having team roles was important in order to divide the tasks separately without one team member handling most of the project by themselves.

In order to develop WordScuffle the best way possible, we came up with the technologies to create a well developed web application. We first started off by using BitBucket for version control of our project. BitBucket works like GitHub, where there is one project and multiple group members can work on the same project together on different computers. For our IDE, we used Visual Studio Code as our code editor which includes debugging, task running, and embedded Git control. This was the last of the development life cycle before going into developing our project, WordScuffle.

In conclusion, the development life cycle was an important phase of our project in order to get the ball rolling on developing WordScuffle. We first had to meet with our sponsor, Barbara Jenkins, to gain the key requirements for our web application. Our team came up with specific roles for each team member so tasks were divided and many tasks could get done at once. We then chose tools to help us develop the project so we all could contribute at once without code conflicts. The process overview was a key process used to develop our project WordScuffle.

3. Requirements

To acquire project requirements for our implementation of WordScuffle, our team has been meeting bi-weekly with our client for approximately three months. During our first month of meetings we played WordScuffle to gain basic ideas about how the game is played and we discussed basic requirements for the implementation such as Mrs. Jenkins' desire for the game to be a Web 2.0 application and to be mobile friendly.

In our last month of meetings, we focused on honing domain-level requirements to reach main functional, non-functional and environmental requirements. We approached requirements acquisition by examining the way the game is currently played manually and then conceiving of a comparable workflow for our web application implementation of WordScuffle. We used the workflow of how a user will interact with the web application to determine our functional requirements. We were then able to extract non-functional and environmental requirements based on what we determined we must have functionally.

The main domain-level requirements determined by our project sponsor are the following. First and most importantly we should be able to play the pen and paper version of WordScuffle with our new implementation. Our implementation should secondly be a Web 2.0 Application which combines with our third requirement that it be playable on mobile devices, especially iPad. For our fourth domain-level requirement, we should include an administration console so that an administrator can change global settings for our WordScuffle once it is up and running, manage data from the game, and manage accounts and communities. Fifth and finally, we must design our web application with monetization in mind.

3.1 Functional Requirements

Playing a round of WordScuffle is our main category of functionality and it is comprised of four main functions: constructing words, keeping score, tracking time, and validating words. These are our first, second, third and fourth functional requirements. When a player begins a game, this cues these four functions concurrently: the player must have the ability to form words; while they form words, our web application will simultaneously keep their score, track time for the round, and perform word validation.

Remaining functionality consists of what a user can do in the web application beyond playing a game. Our fifth functional requirement is to view and compare scores, and our sixthis to allow users to add friends. In our implementation of WordScuffle, players can search by emails to look up their friends and add them. When a player adds a friend, they can then compare their scores, they should see how their scores rank with other members of their friends list. Our seventh functional requirement is the ability of a user to manage their account in order to upgrade/downgrade their account type or change their personal information. General requirements and requests of our project sponsor determine our eighth functional requirement, the ability to access an administration console to change application settings. Our project sponsor has requested the ability to log in to an administration area to manage variables and data associated with the game.

Finally, logging in or authenticating is our ninth main functional requirement because some of our other functional requirements, like the ability to access an admin console to change settings and the ability to track scores and add friends, are determined by our ability to identify users within our web application.

3.2 Non-functional Requirements

There are several non-functional requirements we have determined, some of which we feel are quantitative components of functional requirements while others are not and yet they are important for the overall success of our web application.

For instance, there are particular quantitative requirements we have determined necessary in order to make forming words a naturally-mapped experience for users on a wide variety of devices. The functional requirement is users must be able to form words, but we additionally require gameplay to feature drag and drop tiles. Furthermore, to maintain feedback in the game, lifting a tile should require no more than 250ms and afterwards, the location of the tile relative to the tip of the finger should be no more than 1-2mm from the original configuration of the fingertip when the user originally touched the tile.

Another non-functional, quantitative requirement relates to our functional requirement of word validation. We have determined that to successfully validate words in a manner that prevents players from becoming impatient with gameplay, word validation should complete and provide feedback within two seconds for each validation attempt. To follow industry standards, we have an additional accuracy requirement that our word validation appropriately reflect what is contained in the official Scrabble$_{TM}$ dictionary.

We include tile set generation here as a non-functional requirement because a user action does not precipitate generation; this process will occur on a normal basis on our server to create the data which our users will access during gameplay. Tile set generation is an extremely important part of playing WordScuffle: each day, our server must randomly generate ten tile sets (these are the default settings for WordScuffle) for gameplay: all players will receive the same tileset for each round and there are ten total rounds.

Similarly, device compatibility and reliability of server are two more requirements which are important for our web application but which are not specific functions triggered by a user. For device compatibility: based on project requirements defined by our project sponsor, our implementation of WordScuffle should be both playable on mobile devices and a Web 2.0 application. This means that our implementation must be a web application that is fully scalable and playable on everything from mobile devices with small screens to desktop browsers and it must render the same across a variety of browsers. For browsers, we expect to support about 95% of the general American population.

For reliability of server: because our web application will be served, we have a non-functional requirement to provide reliability and a contingency plan in case of failure. Given a failure of our server, we must automate rebooting and automate notifications with server status for quick follow up from a support person or developer. Providing redundancy to restore settings from a catastrophic failure is vital. If such a failure occurs, users should be given feedback within several seconds that the application's backend is out of service. Whatever gameplay has occurred client-side should be saved until the server is back up to avoid user frustration.

4. Architecture and Implementation

Figure 1 (below) shows the high-level detail of how we built our web application for our sponsor's game, WordScuffle. The implementation is broken into the Server, Frontend Client, and Hosting components described below. We also will discuss the drag-and-drop library we used called Dragula.

Server

Our teams has chosen to use Node.js to create a server to perform tasks related to user management and authentication. As shown in Figure 1, the server will communicate with Firebase to allow users to create accounts and login to their accounts. The server will also use Firebase's Realtime Database to store gameplay data such as user data, community scores, and gameplay configuration.

This server will perform tasks related to gameplay. The server will be responsible for generating and serving random tilesets that players will use to construct their word grids. It will also perform the word validation as players construct their words as well as scoring for each word that is validated. Once a user has completed a challenge, the server saves the word grid and score into the Firebase Database.

Frontend Client

The team will be using Angular 4 to construct a highly responsive user interface that will allow users to use the various features offered by WordScuffle. The server will serve random generated tilesets to the frontend client, which players will use to construct words onto a play grid. Once the game time limit has passed, the frontend client will send the constructed grid and score to the server to be saved.

The frontend client will also allow the user to manage their account data and participate in community gameplay functionality. From the frontend client, the user will be able to change the personal information associated with their account. They will also be able to form communities with other users in order to compare scores and constructed word grids.

Because Angular is a web framework, it is compatible with all mobile and web browsers. Furthermore, users will be able to access our web application on mobile devices and any browser on a computer. Angular will allow our web application to resize to fit any device.

Dragula

As shown in Figure 1, we are using ng2-Dragula that is an implementation of Dragula that is optimized to work with Angular. We are using Dragula to create the locations on the page for users to drag and drop tiles to and from on the webpage. Dragula is used in our web application as a container element in our templates that help organize the tilesets when being dropped into the game grid. The tilesets are dragged over a container and once let go, they drop into place for the user trying to configure a word to validate.

Hosting

The team will be hosting the web-application on DigitalOcean. DigitalOcean is a cloud computing platform that will help manage infrastructure easily. DigitalOcean's servers use high-performance Solid State Disks that will directly benefit the performance of our hosted web-application, WordScuffle.

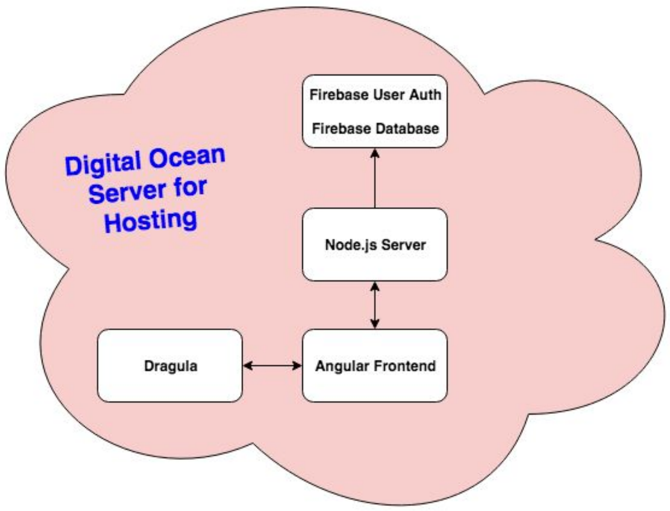


Figure 1: Architectural Design of our WordScuffle Web Application.

Figure 2 (below) is a diagram that describes the Architectural Overview of the Data Flow of our web application. Firebase will be used as our Database which will be communicating to the server back and forth. The server will then communicate data to the services such as User, Game, and Friend services. These services will then communicate back and forth with the various front end app components of our web application.

Our team chose to use Firebase Database to authenticate users who are using the web application. Firebase Database stores realtime gameplay, scores, daily challenges, and communities being defined by users on the Node.js server. Data is stored as JSON and synchronized in real time to every connected client. With the realtime database, our client will automatically receive updates with the newest data.

The Node.js Server can create an HTTP server that listens to server ports and gives a response back to the client. The server will communicate with the Firebase Database to authenticate users, as well as ensuring real time data is being displayed properly on the front end of the web application. Once a user has generated a random tileset of unique words, the server will validate the word, then send the validation to the front end of the web Application, which we will be using Angular 4.

Angular 4 is being used to create components on our frontend that will be communicating with our client to display the functions of our web application. Our web application is using Angular 4 to route from different components to one another, as well as keeping a nice interface for different devices, such as computers, mobile devices, and tablets.

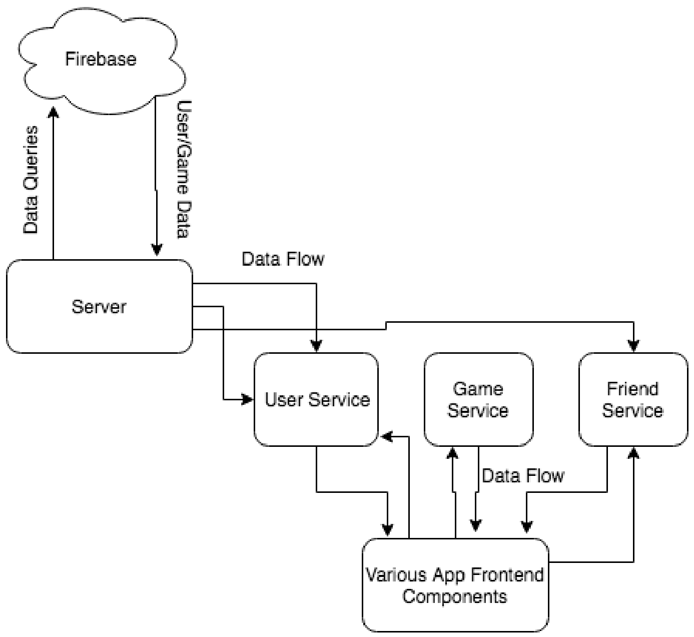


Figure 2: Architectural Overview of Components Data Flow

Because we use Javascript Angular.js framework, our application uses the model-view-controller architecture.

5. Testing

We only pushed changes to the server that appeared to function properly, but a big part of testing is to make sure the product is fail-safe. All features were implemented from our requirements document and work how they should, but testing makes sure the features behave correctly for all possible workflows. The most important work flows are the unexpected workflows that a user may do either unintentionally or maliciously. The software must handle these alternative workflows properly. We have split our testing into three types. The functional requirements from our requirements document have been used to create our *unit tests*. Our integration testing made sure the main components of our application work correctly in all conditions. Lastly, we did usability testing to make sure users could easily use the website.

For our Unit Testing plan, we wrote tests to check each function for proper output for given inputs that covered the whole range of possible workflows. For example, our unit test for the scoring function would pass it a board and we would expect a particular output. Scoring checks for non-contiguous tiles, tiles in the tile tray, and validity of all words on the board, giving us a minimum of four tests to make sure all workflows are properly working for the scoring function.

Our integration tests made sure app functionality that spread over multiple components worked properly. This required us to check variables throughout the process. An example of this would be viewing scores:

- The user clicks a completed challenge score
- Send the request to our server
- Server gets scores from firebase database
- Server sends scores to user
- Scores populate the dropdown menu in the score view page.

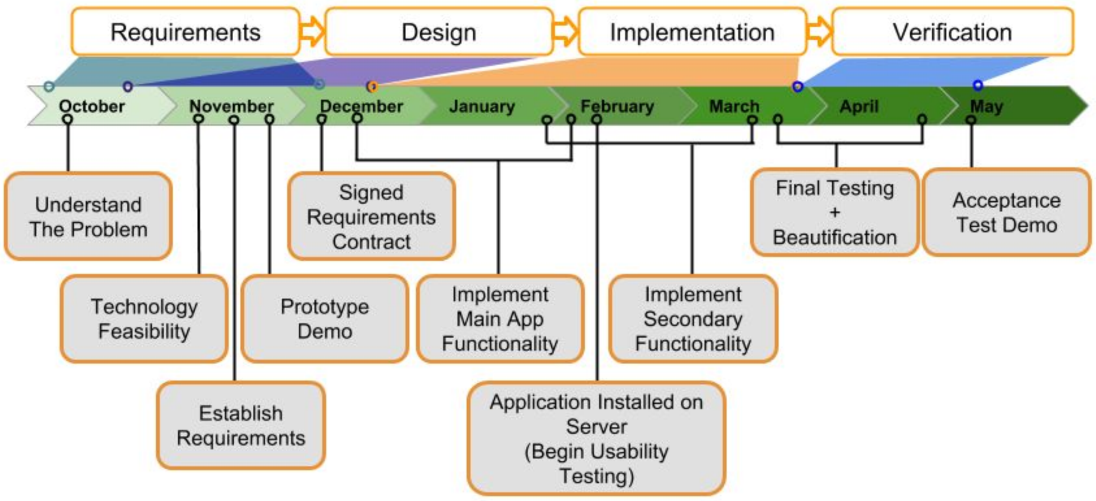
For testing these, we used debugging code to print to the server console, and viewed the firebase database to find the expected results.

Usability testing focused on user interaction with the webpage. One requirement for the user interaction in our web application is giving the user a persistent menu that is available throughout the application. The web application is meant to be ‘appified’, which means navigating the website has to be possible without the URL bar or back button. For our gameplay, we focused on usability testing to ensure that users can navigate through the gameplay. To ensure easy navigation throughout our web application, we added buttons to navigate the user. As soon as the code hosted to a server, we begun receiving feedback from Barbara and new users.

In conclusion, our unit tests focused on smaller scale communication of data, while the integration testing focused on the organization and distribution of the data throughout the application. Our usability testing focused on providing a streamlined user experience. Successful results in these various tests will ensured that our application was ready to be made available to a larger number of users. The unit tests and integration tests all passed when we ran them after finishing the application development, and when showing our finished product to a new user, they were able to figure out how to navigate the website without needing help.

6. Project Timeline

Our project timeline was a guideline for our team to follow to get tasks done on time in order to release a fully functional product to our sponsor. We came up with important dates to have certain tasks completed by so we could move on to the next phase of our project. Some of the key phases represented in our timeline was to complete the automatic word validation, creating the challenges component, friends component, and the game component. Once these tasks were completed, our prototype would be released to our sponsor and beta testers. During our time programming, we had documents to write and presentations that would correspond to the progress made on the web application throughout the year. Having the project timeline was essential to give our team a guideline on completing tasks on time so there could be time for setbacks if needed.



During our implementation phase, we had important dates that were set to complete tasks on our application component. One of the main components that needed to be completed was the game component. This was one of the longer phases in our project because we had to have it fully functional before moving on to other tasks such as the word validation and game mode rules. Once we finished the game component, we implemented the friends component so users could search and add friends on our web application. This task would then lead to completing the view solution component, so users could then go back and view past solutions to compare results with their friends. One of the last components that needed to be implemented was the admin console component for our sponsor. The admin console component gives Barbara the option to edit the settings for each game mode such as changing the time per challenges, the amount of tiles given to the users, and how many daily challenges there are for the day. Once the implementation phase was completed before Spring Break, we hosted our web application to the Digital Ocean server and released our project to Barbara Jenkins and beta testers, which our focus shifted to the testing phase.

The testing phase was in progress while implementing the prototype. We had to test while implementing to ensure the functionality was working before releasing the prototype. Our testing after Spring Break involved beta testers to try out our game and give us feedback on anything confusing to the user. We were still meeting with our sponsor bi weekly to gain feedback from her, and any issues that she was having that needed to be fixed. We faced an interesting challenge early in April, where there was a new iOS update. This update affected our web application by altering our scrollable feature to keep the screen still while a user dragged and dropped a tile. We ended up researching and fixing this issue a week or so after, however with the testing phase, this allotted time for any bugs or errors that could occur. The testing phase was very crucial to fix any issues that our sponsor encountered, and we were very thankful to have time in the year just for testing because of any unexpecting issues such as the iOS update in April.

Another key phase over the project timeline were the documents and presentations. We had a handful of documents that were crucial to the project development. This took time away from implementing, so our group split the papers throughout the team in order to make progress on the implementation as well as having a final deliverable to hand in to our mentor. The key papers that were written were the Feasibility Document, Requirements Document, Design Document, and the Software Testing Plan. Our team also had three Design Review presentations that gave our class an update on the progress we made with our web application. The main presentation we had was the UGRADS symposium at the end of April. This is where we showed off our final product to the public, it is also the last presentation before our Acceptance Test Demo. Once we have completed this, we hand off our final product along with a user manual to our sponsor Barbara Jenkins, and we are done with the Capstone class.

This year had many deliverables and presentations that kept our team very busy. We had to set important due dates for our product so we could start implementation and have these tasks done on time. We had presentations that inform our class the progress we had made on our web application. During implementation, we started on the testing phase which lasted all the way into April. Our project timeline was very thorough with a lot of assignments and milestones that were completed along the way.

7. Future Work

For our web application WordScuffle, there have been many ideas that have been brought up for future updates. One of the main ideas brought up is the idea of monetization. Monetization would be added for users that would play the demo for free, however after playing the demo we would recommend the user to enroll in a one-time or monthly subscription. Another idea that was brought up was to incorporate definitions to the view-solution component. This would be for when the user has completed a challenge, they could view definitions of the words they had created in the word grid. A final idea that we have thought about adding in the future was to incorporate a third game mode for the users. These ideas have been discussed with our sponsor to include in our web application WordScuffle.

The idea of adding monetization to our web application was brought up at the beginning of our design process. WordScuffle is an addictive word game and to keep it running on the web requires some money that our sponsor is willing to pay. The cost of hosting it on a server, as well as keeping the website domain is pricey for our sponsor. Our team believes the users that have played WordScuffle are willing to help keep the web application going. So we have come up with a plan for the future that would include a one-time or monthly fee for a subscription to the site. This process would be included by offering the users to play the demo version of our web game, and once the demo time has ran out, the user would have an option to keep playing the demo or to subscribe to our web application. The demo version of our web game does not save the scores of the user, as well as offer the second game mode which the scoring emphasises to create longer words for more points. Monetization is a great idea for our web application because we know how addicting the word game is, and if there is no option besides playing the demo, users will be encouraged to create an account and subscribe to our web application.

Another idea for the future of our web application is to include the definitions to the view-solution component. This was brought up in a meeting with our sponsor, when we were showing her the almost completed prototype, which she suggested it would be very intriguing to know what words meant that were created in WordScuffle. Users would go back to view their submitted word grids, which to the left of the word grid, would be a table that would display the validated words made, as well as the definitions to those words. Adding definitions to WordScuffle would enhance our web application to help teach users definitions of peculiar words.

Our team during the process of our web application, WordScuffle, came up with the idea of including a third game mode. This third game mode would include daily challenges, however each challenge would be either Game Mode 1 or Game Mode 2. The difference in game modes is the scoring in Game Mode 2 favors longer words over intersections. The score uses a base multiplier on the length of words created. So our Game Mode 3 the user would not know what game mode the challenge they were about to play was until they click on to play the challenge, which then the user will be shown if it is Game Mode 1 or Game Mode 2. Our team was really intrigued by adding this Game Mode 3 to our web application to enhance the playing features for our users.

Overall, our team has modularized our web application to add future aspects to WordScuffle. WordScuffle could be monetized to encourage users to subscribe to our web application to aid our sponsor in the costs of keeping WordScuffle up on the web. It is very hard to keep a web application going because there are payments to be made in order to keep the domain name and to host the web application on the server. Our team wished we were able to add definitions of the words created in the challenges that would help teach our users about the peculiar words that they created in their word grid. We also wished to have added a Game Mode 3 for our users which would include daily challenges that each challenges was randomized to be Game Mode 1 or Game Mode 2. Overall, we wished we had more time on this project, however we had certain requirements to make sure were functional before handing off our product to our sponsor, in which we came up with great additions for our web application in the future.

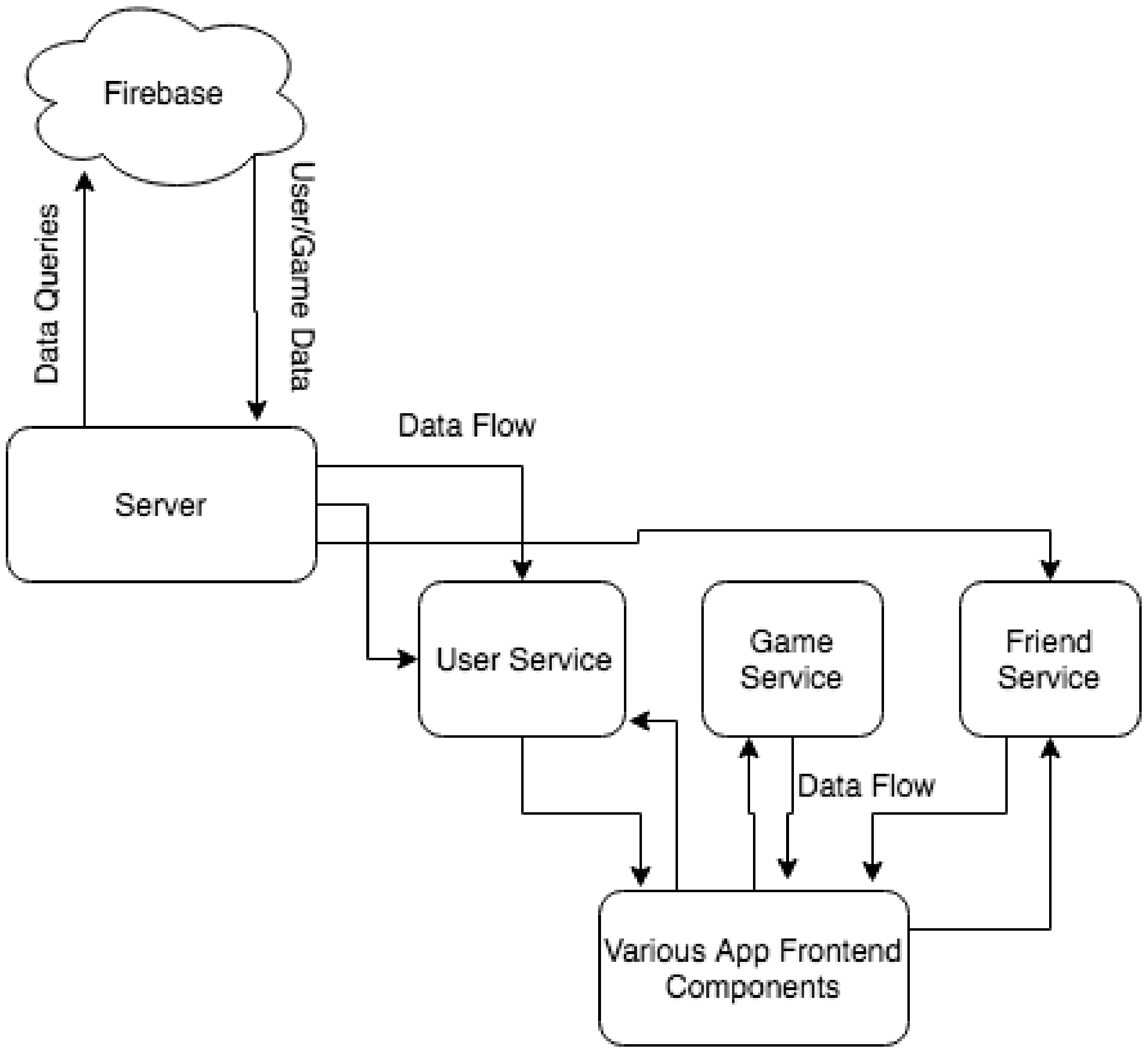
8. Conclusion

In conclusion, our team has developed a highly responsive mobile-friendly web application for our sponsor Barbara Jenkins. We have brought her game of WordScuffle, to the web, and it is playable from any device using a web browser. We have had meetings with Barbara to go over the key requirements that were needed to be met, in order for her to be happy with the product. Our team, Brainstim Studios, went over which process cycle was fit best in order to finish the project in a years time. We discussed the key roles each member of the team was assigned to so everyone was involved with the development of WordScuffle. Once we had roles, we gathered the key technologies that should be used for WordScuffle to create the best version of WordScuffle. Our project had a timeline of the key events that took place over the year, which was discussed how important these milestones were. Over the course of development there had been discussion with our team and sponsor about future ideas that if we had more time, would be developed and added to WordScuffle. Overall our year has been very busy with the development of WordScuffle, and we are proud of the job we did with the finished product.

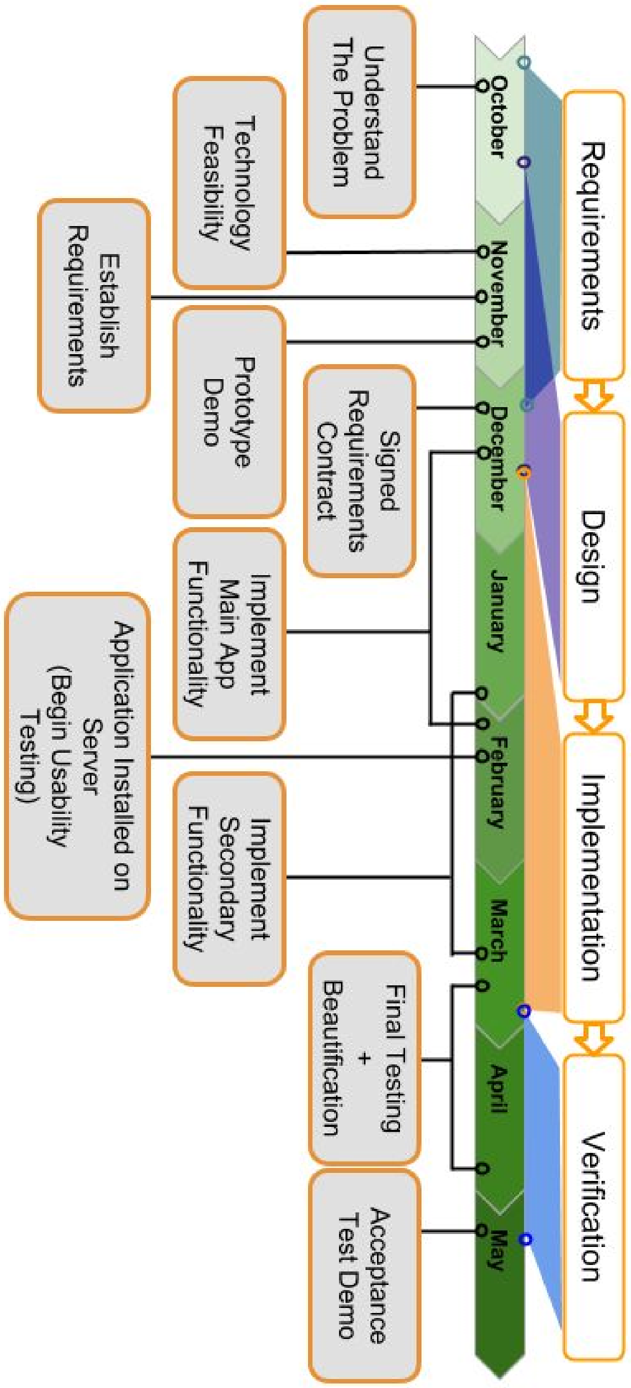
9. Appendix

Getting Workspace Setup

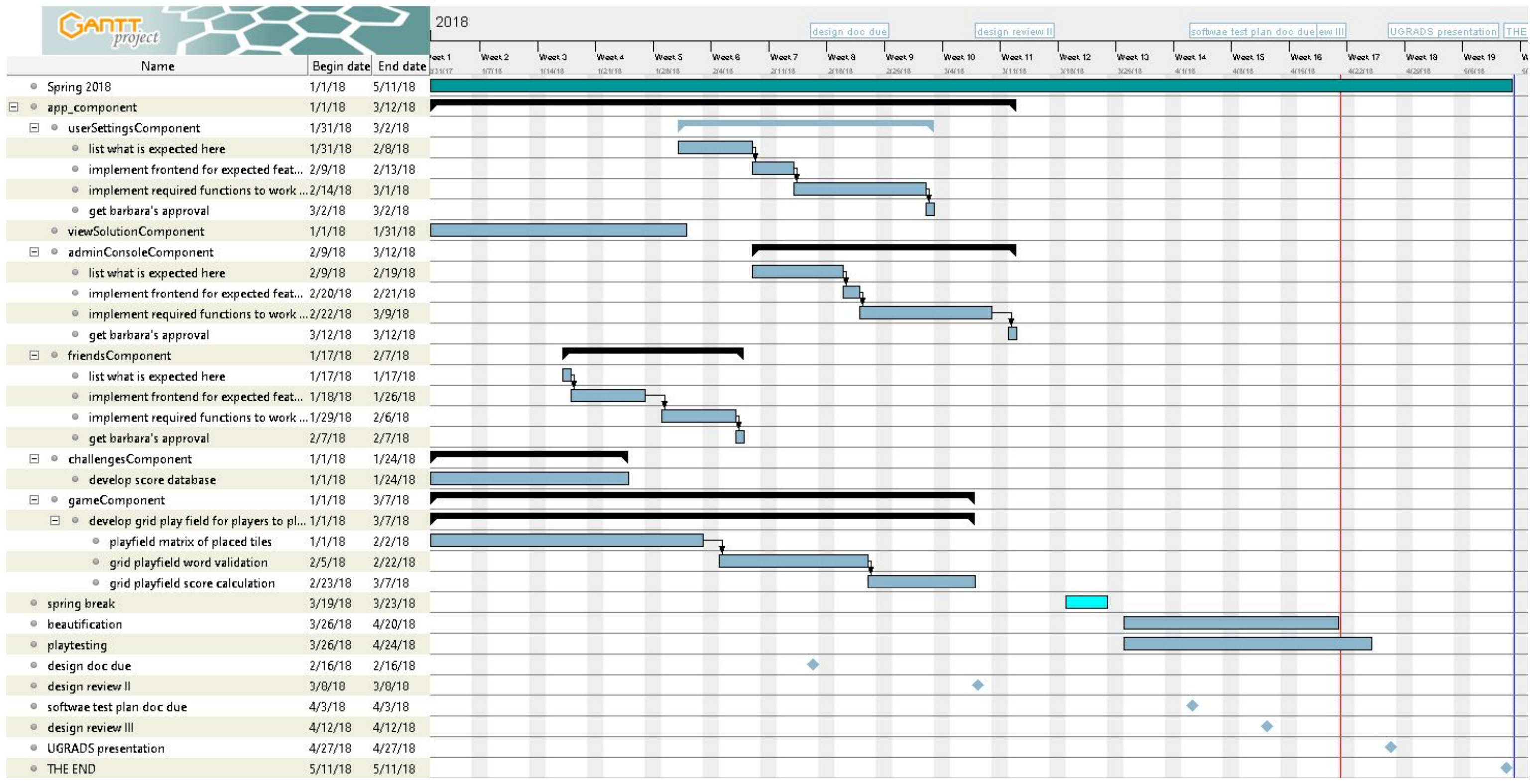
1. After getting access to the project repository, clone the repository.
2. Three protected config files need to be placed into the project config directory.
3. Install Node.js
4. Run the command: npm install
5. Run the command: ng build
6. To start the server, run the command: node server.js



Architectural Overview of the Data Flow of our web application. Firebase will be used as our Database which will be communicating to the server back and forth. The server will then communicate data to the services such as User, Game, and Friend services. These services will then communicate back and forth with the various front end app components of our web application.



Timeline for project.



Name	Begin date	End date
Spring 2018	1/1/18	5/11/18
app_component	1/1/18	3/12/18
userSettingsComponent	1/31/18	3/2/18
list what is expected here	1/31/18	2/8/18
implement frontend for expected feat...	2/9/18	2/13/18
implement required functions to work ...	2/14/18	3/1/18
get barbara's approval	3/2/18	3/2/18
viewSolutionComponent	1/1/18	1/31/18
adminConsoleComponent	2/9/18	3/12/18
list what is expected here	2/9/18	2/19/18
implement frontend for expected feat...	2/20/18	2/21/18
implement required functions to work ...	2/22/18	3/9/18
get barbara's approval	3/12/18	3/12/18
friendsComponent	1/17/18	2/7/18
list what is expected here	1/17/18	1/17/18
implement frontend for expected feat...	1/18/18	1/26/18
implement required functions to work ...	1/29/18	2/6/18
get barbara's approval	2/7/18	2/7/18
challengesComponent	1/1/18	1/24/18
develop score database	1/1/18	1/24/18
gameComponent	1/1/18	3/7/18
develop grid play field for players to pl...	1/1/18	3/7/18
playfield matrix of placed tiles	1/1/18	2/2/18
grid playfield word validation	2/5/18	2/22/18
grid playfield score calculation	2/23/18	3/7/18
spring break	3/19/18	3/23/18
beautification	3/26/18	4/20/18
playtesting	3/26/18	4/24/18
design doc due	2/16/18	2/16/18
design review II	3/8/18	3/8/18
softwae test plan doc due	4/3/18	4/3/18
design review III	4/12/18	4/12/18
UGRADS presentation	4/27/18	4/27/18
THE END	5/11/18	5/11/18

The gantt chart we created in January to keep us on schedule for the duration of the project.