Software Design

Date: 2/20/2018

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 create communities with 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 form communities
- Providing word validation to lower confusion and eliminate scoring mistakes

The purpose of this software design document is to work like a ‘blueprint’ to show how we intend to implement WordScuffle. With the given idea of WordScuffle, we were able to create some requirements that would need to be met to successfully create what Barbara imagined. We have nine main functional requirements: constructing words, keeping score, tracking time, validating words, allowing users to view and compare scores, allowing users to join or create a community, allowing users to manage their account, easy to use/access admin console, and user authentication in the form of logging into accounts. These functional requirements are coupled with the non-functional requirements of tileset generation, tile set attributes, word validation attributes, device compatibility, and server reliability. The environmental requirements are that the game will require the users have internet access, the game will only run on devices with certain hardware specifications, and the reliability of our chosen servers is out of our control.

2. Implementation Overview

WordScuffle is a fast-paced word game that was created by our sponsor, Barbara Jenkins. Currently the only way to play the game is with pencil and paper, recording scores down, and comparing your scores with friends through email. Our team BrainStim will be developing a highly responsive 2.0 web-based application and mobile-friendly game. To solve this we will be using many technologies to implement WordScuffle onto a server and having it accessible to many users around the world.

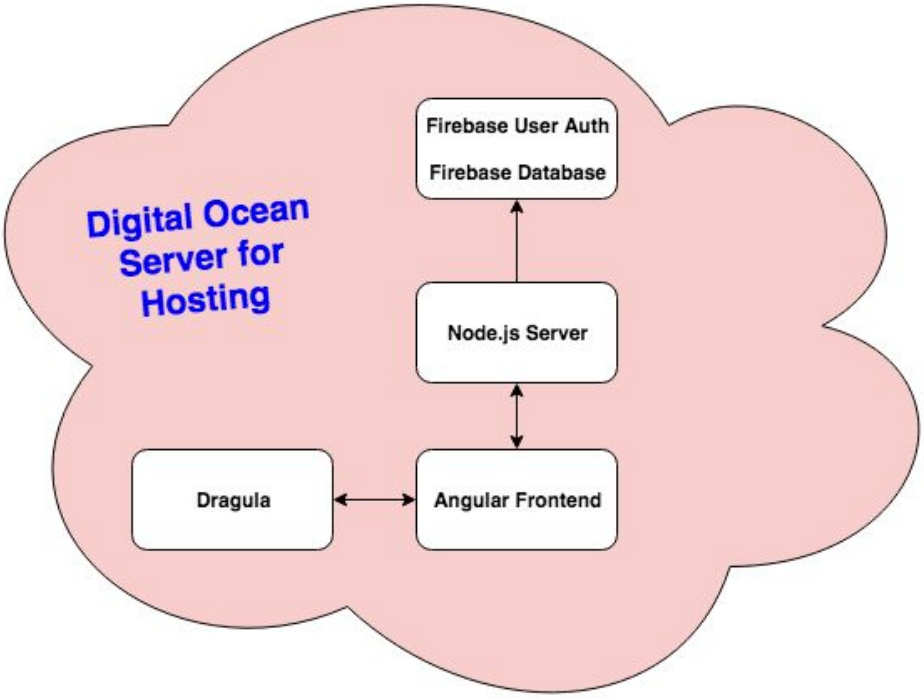


Figure 1: Architectural Design of our WordScuffle Web Application.

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.

3. Architectural Overview

Figure 1 shows the high-level detail of how we built our web application for our sponsor's game, WordScuffle. Below in Figure 2, we have 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 as shown in Figure 2 below.

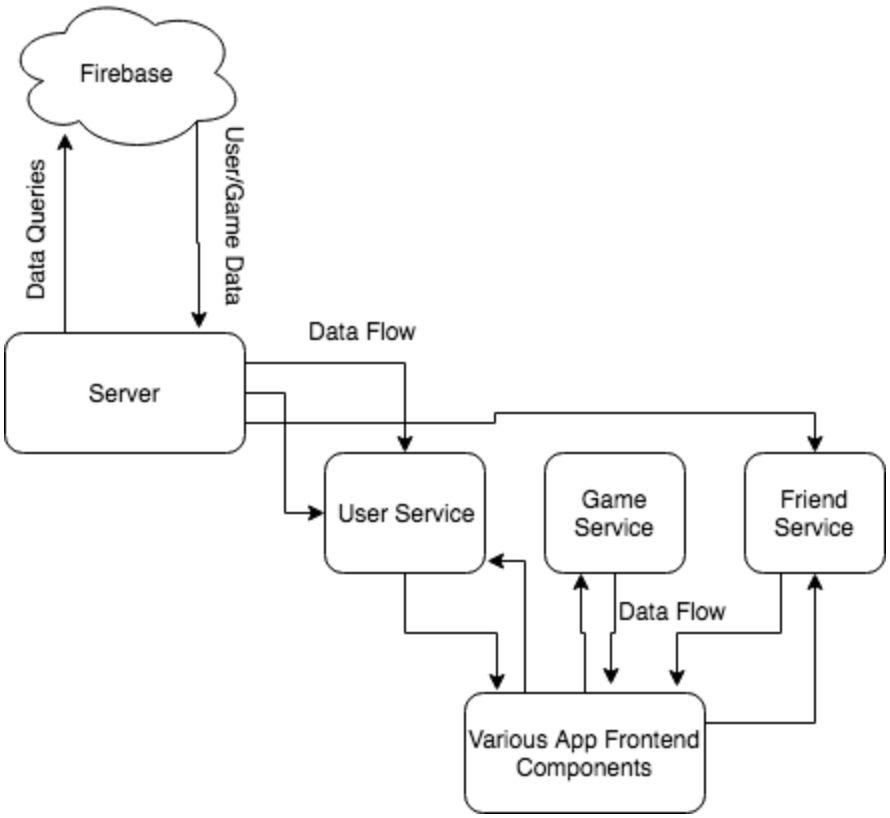


Figure 2: Architectural Overview of Components Data Flow

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.

4. Module and Interface Descriptions

Our web application is broken up into several modules and components. This was done to organize logic and implementation into separate areas to ensure that the various components do not interfere with each other. The sections below describe the various modules and components that make up our application.

App Component

The AppComponent is the main overarching component of our web application from which every other component is routed. It essentially bootstraps the front-end of our web app and handles global logic and settings. When a user visits the web app in a browser, they are first routed to AppComponent which then navigates the user to the default component for the web application. Using router settings in the app module (app.module.ts), AppComponent defines URLs throughout the site to navigate the browser to their associated components. To simplify login and signup, login/signup modals are included as subcomponents of the AppComponent, which makes them available across the entire application. The main ways that a user will interact with the AppComponent is by clicking on different navigational links and buttons. User's also have the ability to interact with the AppComponent's router through workflow based routing.

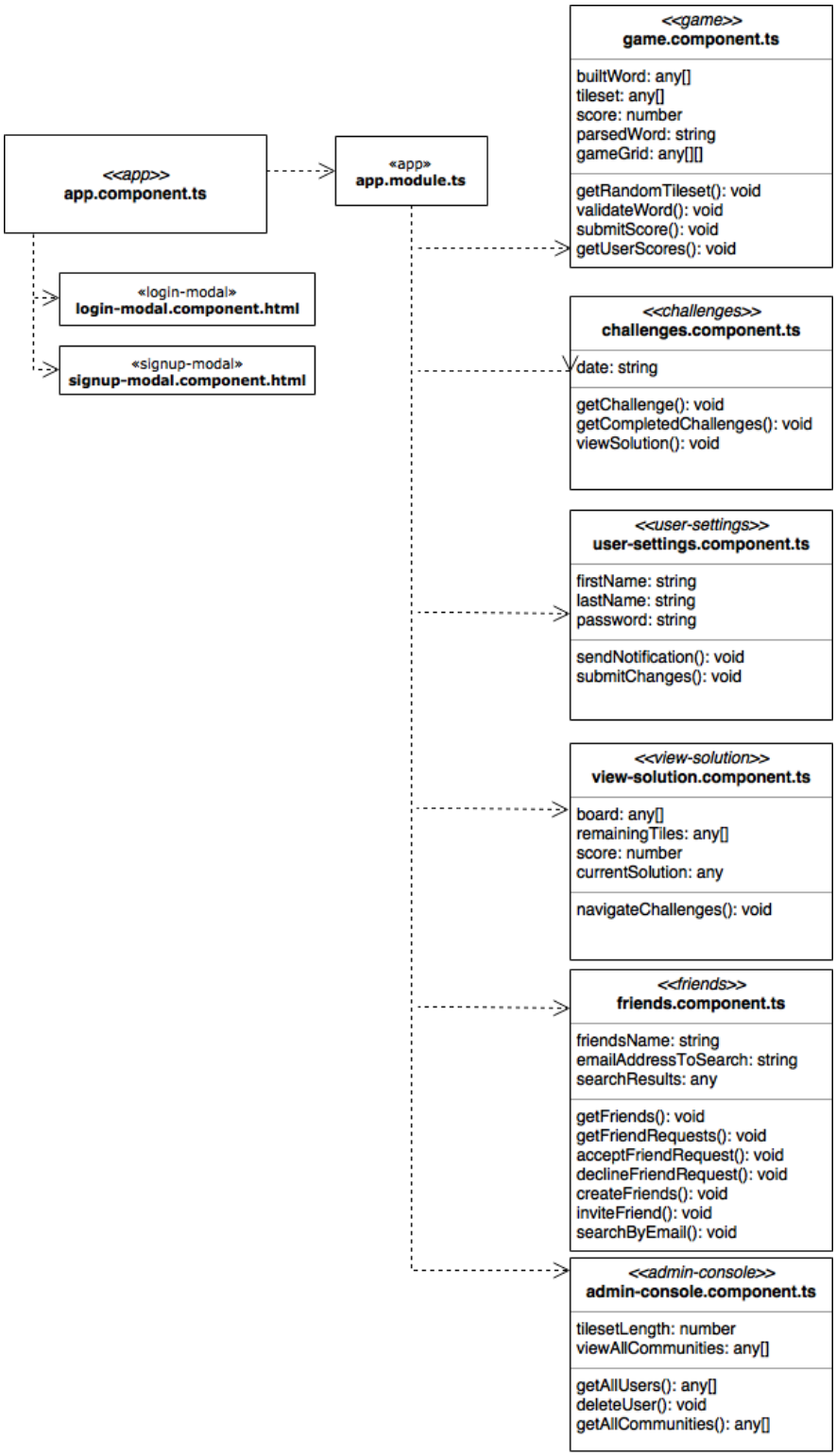


Figure 3: App Component Architecture

Game Component

GameComponent is navigationally accessible from within the ChallengesComponent. GameComponent is loaded when a user clicks on a new challenge listed in ChallengesComponent. The GameComponent is the main WordScuffle gameplay area where a player receives their letter tiles and places them on a grid while formed words are validated. GameComponent sends and receives data about the current letter tileset and word validation from game service while it retrieves and associates the current game and score to the current user using the user service. The main way that users will interact with the GameComponent is through the drag-and-drop interface provided by Dragula. Dragula is installed as a package and available to use throughout the different application components. Users will drag tiles onto the game grid, which builds the game grid data structure behind the scenes with each tile drop. User's will also be able to interact with the components public method of submitting a score if they wish to submit their challenge before the time runs out. This will be exposed through a button on the page.

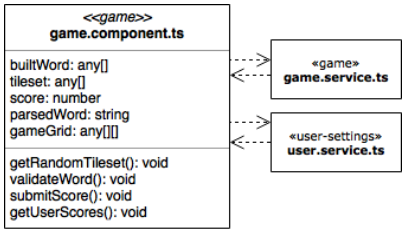


Figure 4: Game Component Architecture

Challenges Component

ChallengesComponent is available navigationally from any page in the web app after a user is signed in. It displays a list of challenges available to a user for that day. ChallengeComponent receives challenge data from the user service. The data it receives determines which challenges are available, which have been completed, and which can be played next. When a user clicks on the next playable challenge, they are routed to GameComponent. If the user clicks on a previously completed challenge, they are routed to ViewSolutionComponent. Challenges are generated every day and served to the users as they play through their challenges. However, the application also has the ability for users to play practice games where scores are not tracked. Because of this, the game component also has a method to ask the server for a freshly generated random tileset.

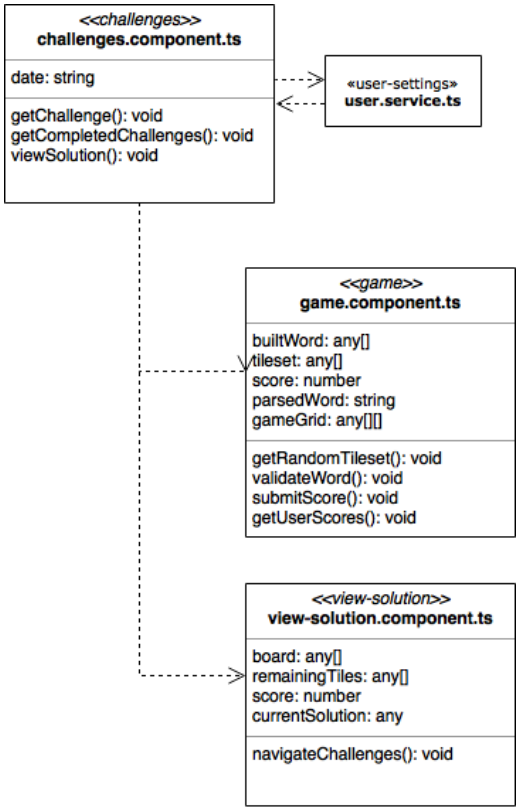


Figure 5: Challenges Component Architecture

Friends Component

FriendsComponent is available navigationally from any page in the web app after a user is signed in. This component gives the user the ability to manage their friends list. When users become friends, they then gain the ability to view each other's scores for the various daily challenges. The page will display the logged in user's current friends list, as well as any pending friend requests that they can respond to. FriendsComponent receives and sends information between user service (user.service.ts) to appropriately display information to a user and update information in the app's firebase database when data is updated from the front-end. One way the user will interact with this component is by typing a user's email address into the search bar in order to send friend requests. Also, the user will be presented with the option to remove any current friends from their friend list.

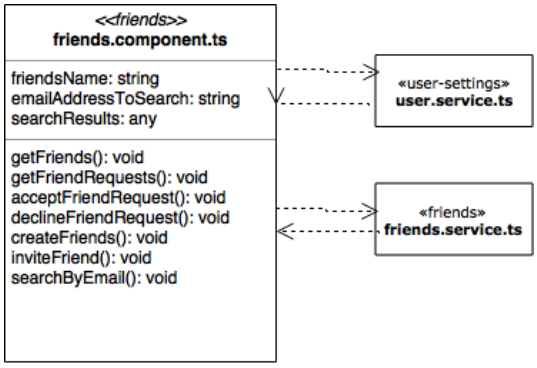


Figure 6: Friends Component Architecture

Admin Console Component

The AdminConsoleComponent is available navigationally from any page in the web app after a user is signed in if they are flagged as a site-wide administrator. The AdminConsoleComponent displays fields where an administrator can update global game-related and sitewide variables as well as perform some administrative tasks related to managing others' accounts. The user will most communicate with this component through input fields that they can modify with associated buttons they can press to update the respective values in the database.

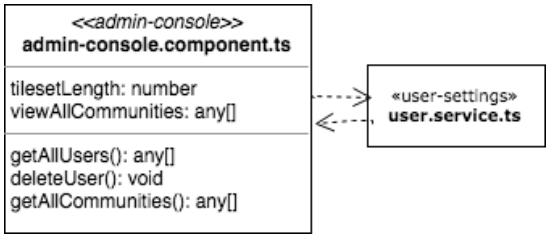


Figure 7: Admin Console Component Architecture

User Settings Component

UserSettingsComponent is available navigationally from any page in the web app after a user is signed in. The page supports the ability to change basic settings associated with a person's user account. It automatically subscribes to certain data fields within the user service (user.service.ts) and updates these fields if a user submits changes on the page. The user will interact with this component through input fields where they can change their modifiable data.

They will also be presented with a button that will send an email with a link to change their password.

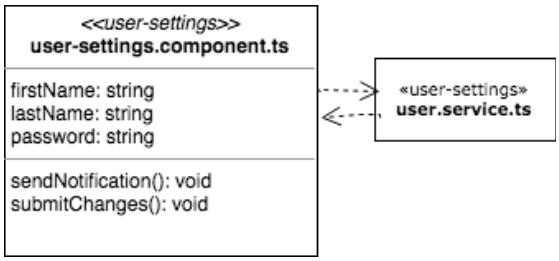


Figure 8: User-Settings Component Architecture

View Solution Component

ViewSolutionComponent is accessible navigationally from within the ChallengesComponent. A user is routed to this component when a user clicks on a previously completed challenge on ChallengesComponent. Data loaded inside this component is static, or read-only. ViewSolutionComponent receives data about the particular challenge instance using the user service (user.service.ts) and displays it identically to what it looked like when they originally played and submitted it in GameComponent. Because of this functionality, the user's interaction with this component is limited. They only way they can interact with this component is by clicking a button to route them back to the daily challenges page.

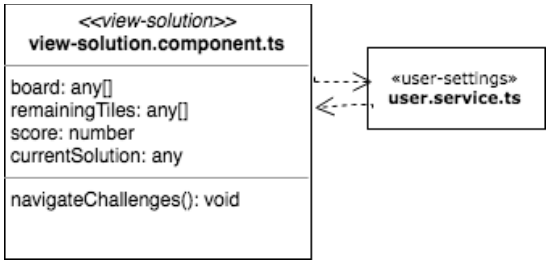


Figure 9: View Solution Component Architecture

5. Implementation Plan

We started the Spring 2018 semester with a working prototype and a signed requirements specifications document. The prototype was used as a proof of concept to verify that our chosen technologies were going to work with our plan. The requirements specifications document explicitly listed what the app would have to do, and was signed by Barbara so it would act as a contract. At the end of this semester, the app will be evaluated by how well the requirements specifications were met.

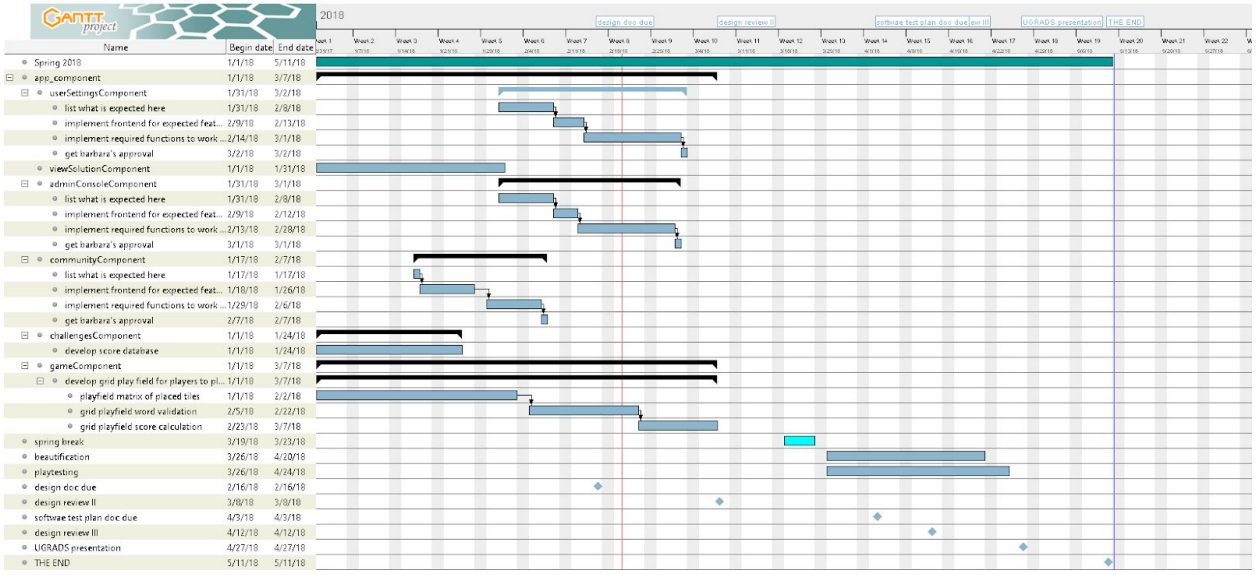
We were told to have a functioning application by Spring break. All tasks related to the components described in the section above are scheduled to be finished the week before Spring break. Beautification and playtesting should be the only parts left to do after spring break. Tasks are split up at our team meeting for the coming week. Tasks are split to match each person's programming skill-set.

Most of the components take place in the web application as individual webpages. Because of this separation, the less complex components one team member could be tasked with completing a component. Andy Salazar is working on the community component, Nathan Franklin is working on the userSettings, and admin Console, and the rest of the components are split between teammates based on subtask.

As seen in figure 10, development of these components starts with a clear understanding of what is expected. This allows us to find any overlooked problems before coding has begun. The front end of the website gets built first, then functions can be implemented to connect with the front end.

Anderson Moyers works front-end web development, and Vincent Messenger knows how to do everything else. Group coding sessions at least once a week have been a crucial part of completing the more complex components. The most complex and most important component is the game component. Word validation, score calculation, and game rule implementation (for both game types) are part of this component. At least one person always has a task related to this component.

Basic testing of components is done before new code is pushed to the git repository, so that the master will always be stable. We all test features as they are added to the master. Full testing will happen after Spring break. Unit tests and end-to-end tests have been automatically generated by the software we are using for app development, and will be ran after Spring break. Testing also consists of allowing Barbara to play the game



.

Figure 10: Gantt Chart of our Implementation Phase

The last six tasks in the gantt chart are the milestones for document final draft due dates and presentations dates. While the actual task breakdown and details for how work is split for these deliverables are not supposed to be in the project gantt chart, displaying these milestones in the chart remind us of other things that will need to be worked on alongside production of the app code. Document rough drafts and dry-run presentations are not displayed in the chart, because they would engulf the app development part of the chart. At all times of development, there will be at least one non-programming task being worked on.

6. Conclusion

In conclusion, our team BrainStim Studios will be developing a web application implementation of WordScuffle, a word game designed to incorporate cognitive and social gameplay to help reduce the symptoms of Alzheimer's Disease. Our web application will resolve several key workflow problems associated with the current pen and paper implementation of the game on which our web application is based. We have had many discussions with our sponsor Barbara Jenkins to create our set of requirements based off WordScuffle gameplay, expected website workflow, and website administration functions.

The main component, app component, will control the game, challenges, community, admin console, user setting, and view solution. It allows seamless navigation through the website, and controls what data gets sent to the client. We are confident that by following our design plan that has been thoroughly discussed in this document, we can create a web game that meets the project requirements and may help prevent the onset of Alzheimer's Disease.