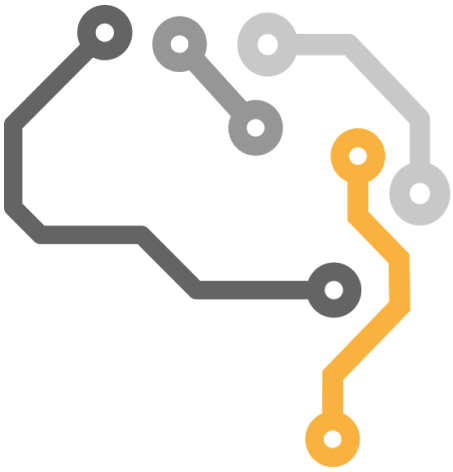
Requirements Specification

Date: 11/22/2017

Project sponsor: Barbara Jenkins

Team faculty mentor: Ana Paula C. Steinmacher



BRAINSTIM STUDIOS

Vincent Messenger (Lead)
Anderson Moyers
Nathan Franklin

Accepted as baseline requirements for the project:	
Client:	Team Lead:

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 overlap. 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 the challenge that is presented. The game will generate ten letter sets per gametype 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. However, these practice games are not eligible for community comparison.

In this requirements specifications document, we go into great detail about the requirements that we have discussed with our sponsor that will need to be satisfied in our finished product. The technologies we will be using to produce the finished product were carefully picked out so that the project requirements can be met. These requirements have been

classified as functional, non-functional, or environmental. With a clearly defined set of requirements, we will be able to use this document as a contract with the client for completing the project. If our software meets all the agreed upon functional requirements and performance requirements, the client should be happy with the finished product we supply them with in May 2017. The requirements will allow us to stick to a plan for the development of the software, and the client should not be surprised by the software we present to them. By thoroughly discussing with the client what they want us to produce, we will have a set of requirements that need not change as the software is constructed.

2. Problem Statement

Although there are existing platforms which offer cognitive gameplay, these platforms do not adequately address the social components of cognition in preventing the onset of AD. Lumosity's games for example offer no wider social context: gameplay is solo, players have little chance for interaction and players cannot learn from the results of others and compete.

WordScuffle offers a chance to have fun with wordplay while specifically addressing the social brainflexing that other platforms lack; the game however is currently played with 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 Ms. Jenkins to 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.

In the following sections we outline the overall picture of how we will translate the workflow of Ms. Jenkins' current implementation to software, the key requirements we must fulfill in order to successfully complete our implementation, and how the technologies we have chosen will meet our needs in this process. We begin with our Solution Vision in the next section.

3. Solution Vision

In order to solve the outlined issues, our team is working with Ms. Jenkins to realize WordScuffle as a mobile-friendly and highly responsive web application. The team will develop a server to perform tasks related to user management and gameplay. The team will also create a frontend client that will allow users to play games, manage their user data, and participate in community aspects of the game.

3.1 Node.js Server

The team will be using Node.js to create a server to perform tasks related to user management and authentication. This 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.

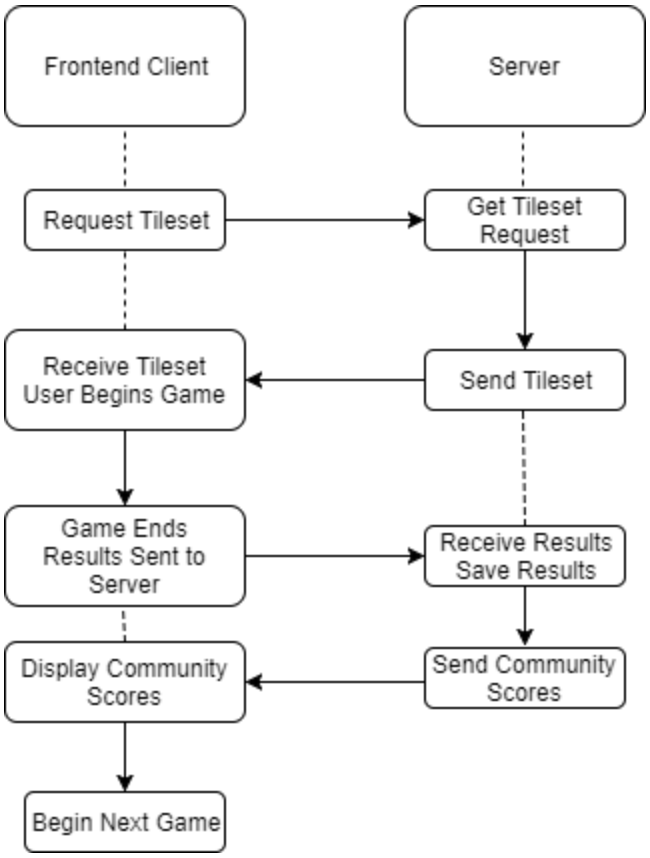
This server will also perform tasks related to gameplay. The server will be responsible for generating and serving random tilesets that players will use to construct their word grids. It will also perform the word validation as players construct their words, as well as scoring for each word that is validated.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

3.2 Angular 4 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 randomly 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.

This 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.

3.3 Workflow Diagram



The above diagram shows an example of a potential workflow for our solution. In this example, the frontend client requests a tileset from the server. The server responds with a randomly generated tileset, which the frontend client uses to start a new game for the user. Once the user finishes the game, the frontend client sends the results to the server to be saved. After the save has happened, the server then responds with the community results for that particular letter set. Once the community results have been displayed, the user can begin another game.

4. Project 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.

4.1 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 sixth is to join a community. In our implementation of WordScuffle, players can participate relative to communities they choose to join. When a player compares their scores, they should see how their scores rank with other members of their community. 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 join communities, are determined by our ability to identify users within our web application.

4.2 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.

4.3 Environmental Requirements

Our two main environmental requirements relate to a.) a low cost solution and b.) development for cross-compatibility.

Because we are completing this project with as low a budget as possible, we must choose technologies that include only nominal fees or are free. This requirement affects for instance affects our implementation of word validation. We must choose a word validation technique that is free yet fulfills our non-functional requirement of performing validation and providing result within two seconds.

Cross-compatibility for a wide variety of browsers and screen configurations is another considerable environmental requirement because it affects the scope of our UI design. Providing a gameplay experience in a browser versus creating a dedicated mobile application per platform means that we will need to design the UI more generally, with a widely applicable interface and a gaming experience that is cohesive no matter where a user logs in to play it. This means for instance that we must design for input types relative to the platform on which a user accesses the game through a browser. If a user is using a keyboard-enabled device they should be able to use this as well as drag and drop touch-based mechanisms. The design should be cohesive enough across different devices such that a user can find their needed task within 10 seconds on any device even if they haven't used that device before to access the application.

5. Potential Risks

Throughout the requirements acquisition phase, the team has identified and considered two main risks. The first risk comes from the fact that our chosen method for word validation is an API with a 5000 request limit for free plans. The second risk relates to server security. The team needs to come up with a way to ensure that secured areas within the application remain unbreached by unauthorized users.

5.1 API Request Usage

After conducting thorough research and consideration, our team decided to make use of the WordGameDictionary.com to power our applications word validation functionality. This API is free, reliable, and fast, so it fits in perfectly with the rest of our system architecture. However,

there is the fact that this API has a 5000 request limit for free developer accounts. Because of our environmental requirements to keep our solution costs low, this could be an issue for us. It is important to note that the team doesn't suspect we will ever hit our API request limit throughout the scope of this project, but if the project ever scales to a greater user base, there could be increased cost associated with hosting.

The team plans to solve the issue of our 5000 request limit by using some implementation of server-side caching. The team doesn't have all the details of this caching algorithm determined at this point, however we have determined that caching of requests could reduce the number of requests that leave our server every day. An example of such functionality can be seen in the following example.

Suppose two users make a request to the server to validate the same word. When the server gets the request from the first user, the server will check its local cache to see if that request has been made before. Since this is the first time a request for this particular word has come in, the server routes the request to the word validation API. Once it receives the response from the API, the server saves the results of the request in its local cache and sends the results to the requesting user. Now when the second user makes a request to validate this same word, the server will check its local cache, determine that it already has the validation results from this particular word, and send the results of this request back to the user. In this case, only one request was sent to satisfy two validation requests. This is just one example of a caching algorithm that prevent the team from ever hitting our 5000 request limit.

5.2 Server Security

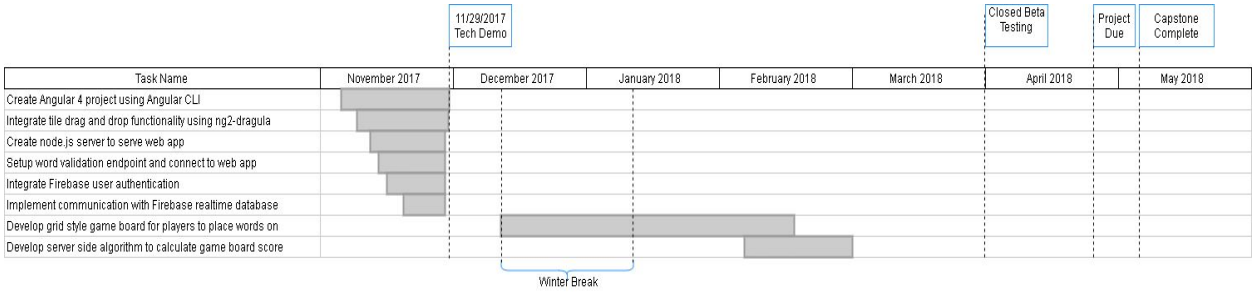
Because our solutions features secured areas such as an administrative console, our team needs to make sure that these areas remain secured to people that should not access them. For instance, we need to make sure that non-administrative users stay out of the administrative console.

By having the user client program running in the user's web browser send the unique user account token to the Node.js server when requesting web pages, the node.js server can verify the user's account type in the firebase database. Once the user account has been verified as having permissions to view the requested web page, the web page data can be sent to the user client. For administrative changes to WordScuffle rules or altering account information for other user accounts, the node.js server will act as a 'middleman' to verify the client has the authorized level of power to make such changes to the database.

6. Project Plan

Throughout November, we have started working with the different technologies that were chosen to implement the game of WordScuffle. Tasks were initiated in an order based on how

they interface with each other, starting with Angular.js as the main platform for the game.



When the class schedule is released next semester, we will be able to make a more detailed gantt chart for the rest of the project. At the moment, we have set a few major milestones for our project.

6.1 Tech Demo

To prepare for the tech demo, we plan on having all chosen technologies working together to give a general idea of gameplay and confirm the technologies will work together.

6.2 Winter Break

The exact level of productivity for this project over winter break is not certain. Technically, there will be more time to work on the project while classes are not in session, but we may have plans other than working on the project. We will stay in contact with each other over this time, but project productivity will be limited.

6.3 Closed Beta Testing

The plan is to get the whole game done by the end of March, so we can release it ‘private beta’ style to Ms. Jenkins and some of her friends. We will expect feedback from them, and will be able to put all programming efforts into any problems they have or changes they want that fit within our ‘contract’. It is important that some of the people in the closed beta testing are unfamiliar with the game, because players experienced with the game rules may have a bias for ease to learn the game and navigate the website.

6.4 Project Due

Near the end of April, the project will be due. Exact date is yet to be discovered. Depending on project progression and actual project due date, closed beta testing may happen sooner than noted.

6.5 Capstone Completion

At the end of the semester, when we have completed the project, we will consolidate a list of accounts and passwords linked to the technologies used for creating WordScuffle. We created a developer email to be used for all needed developer accounts used for creating WordScuffle, so that easy transfer of developer accounts to Barbara’s possession is possible. Instructions for

any important administration control not implemented into the front-end of the website will be written and given to Ms. Jenkins.

7. 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 also 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. 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

Our nine main functional requirements are as follows: 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 no tile dragging 'lag', accurate word validation (after every change to the tile playgrid) in less than two seconds, tilesets generated for the day's challenges at a set time, mobile device compatibility, and server reliability, as well as the environmental requirements to provide a low cost solution and address cross-compatibility to meet the needs of a variety of devices within a browser application.

To address our key requirements for our web implementation of WordScuffle, we will be combining Dragula (a Javascript-compatible drag-and-drop library) with a Node.js-served Angular front-end. We will supplement these technologies with a word validation API from WordGameDictionary.com (to get fast validation from the Scrabble$_{TM}$ dictionary) and the database and authentication services offered by Firebase, a mobile development platform backed by Google.