Software Test Plan

Date: 4/10/2018

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., and a new case is diagnosed every 66 seconds.

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. 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.

Our team BrainStim Studios is working with Barbara to realize this game as a web application. 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. 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 view their friends' scores. Our web application improves on Barbara's current 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

We have implemented the game as a web application now. The next step towards having a completed product is to thoroughly test our app. The game is playable, and 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 now we have to make 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 makes sure the main components of our application work correctly in all conditions. Lastly, we explain our usability testing.

2. Unit Testing

For our Unit Testing plan, we have decided to split testing segments up according to our functional requirements. For each functional requirement, we have explained the main workflow that the requirement was designed to accomplish correctly, and the alternative flows that we have to make sure do not crash the game.

Test Case 1: Construct Words

Constructing words is one of the main functional requirements of playing a round of WordScuffle. In order to make WordScuffle playable on mobile devices, we will implement tile movement with drag-and-drop functionality. A user will receive a bank of tiles from which they can drag onto a grid to form words. The grid will be at least 13x13 but should be designed to accommodate spaces up to 16x16 (grid size will be statically defined by an administration setting and the grid will be generated accordingly). Both the grid and tiles will adjust automatically in size according to screen size and the size of both the grid and the tiles must correspond to one another at any given time. The user should be able to drag tiles to the grid and back to the tile tray. Words can be constructed vertically and horizontally.

With every tile change (either placed or removed from the grid), the board is sent to the server for word validation.

Main Flow

The main flow involves a tile being placed on the board, connected to other tiles and creating a valid word. The board is sent to the server for word validation. The server finds all words in the board and for each word. The server finds the word in a hash table containing all Scrabble™ tournament-legal words (that resides on our server). All words from the board were verified to be valid, so the server tells the client that the board is valid. The client changes all tiles on the board to be green so the user knows the word is valid.

Alternative Flow

The alternate flow is that an invalid word is created with the tiles. The server gets the board, finds the words on the board, doesn't find the word in the list of valid words, and tells the client that the board is invalid. The tiles in the board are changed to purple.

The other alternative flow is when a tile is picked up from the board and placed in the tile tray or placed somewhere else on the board. Any change causes the board to be re-validated by the server. Tiles can only be placed in the starting tile tray, or on the playgrid.

Test Case 2: Keep Score

Keeping score is another functional requirement of playing a round of WordScuffle. As words are formed on our gameplay grid each round, a user's active score for that round is always displayed. The score will be incremented for each correct word that is validated. If letters are dragged from the grid and this causes words to become invalid, the score will be decremented and displayed accordingly. When a round is complete, our web application will automatically save the user's score for that round in a cloud-based database. As a user plays more rounds of WordScuffle, their score history will continue to accrue in our database.

Main Flow

The main flow for scoring is that the board is changed and it is sent to the server. The server finds all words on the board. The words are all legal, and the server adds up the points for every word and sends this new score to the client. The client updates the displayed score.

Alternative Flow

The alternate flow is that the board is changed and it is sent to the server. The server finds all words on the board. Only one word has to be invalid when the words are checked for validity, and the server sends a score of zero to the client. The client displays the score of zero.

Test Case 3: Track Time

When a user begins a round of WordScuffle, our web application will begin timing the round and it will display the time throughout that round. The default time for each round, as determined by Ms. Jenkins, is three minutes. The global setting for the length of a round will be editable on the administration console. When time is up, the game is over and our application

will exit the wordplay screen. Time tracking will be implemented in a way that cannot be manipulated by a user during gameplay.

Main Flow

A game challenge is started and the time is set to the time specified in the admin console. The time reaches zero, and the board and tile tray are sent to the server.

Alternative Flow

A game challenge is started and before time runs out, the user closes or by redirect the webpage. The server knows when to expect the challenge to end, and when the board is not submitted on time, a score of zero is saved for the challenge.

Test Case 4: Validate Words

Word validation occurs throughout each round of WordScuffle. When a singular non-adjacent tile is placed on the board it will not be validated and will automatically be assumed to be an invalid word. This is because our implementation and our dictionary do not accept one-letter words. Word validation for a given array of tiles on the board will begin when a user completes the placement of a second adjacent tile and will continue to occur after each successive tile placement within a particular array. Word validation will also occur after a tile is removed from an array on the board, as long as the array remains at least two tiles long. Each time an array of tiles is changed, (whether it is one tile long or two or more tiles long), our gameplay grid will provide visual feedback to the user by changing the appearance of the entire tile array to indicate that it is either an invalid or valid word.

Main Flow

The main flow for validation of words occurs when a user builds a contiguous word grid of two or more tiles. In this situation, the frontend client sends off a request to the server to validate the built word grid. The server then recursively parses all words from the word grid, validates them, and sends the results back the frontend client. If all parsed words are valid, the user's word grid will be colored green. There score will also be updated. If not all parsed words are valid, the user's word grid will be colored purple and their score will be set to zero.

Alternative Flow

The first alternative flow for word validation occurs if a user builds a word grid that has 2 or more tiles, but is not contiguous. In this situation, a request is sent off to the server to validate the grid. The server then parses all words from the grid and discovers that it is not a contiguous word grid. When this happens, the server sends the frontend client a response that indicates

that the grid is not valid, which triggers the user's board to be colored purple and their score to be updated to zero.

The second alternative flow for word validation occurs if a user's device loses network connectivity. In this situation, the frontend sends off requests to the server, but the server does not receive the requests until the user's device regains connection to the network. When this happens, all word validation completely stops. However, because we created a web application, one of our environmental requirements is that a user has a device that is connected to the internet. Because of this, the team does not have a solution to an interrupted network connection.

Test Case 5: View/Compare Results

An instance of a gameplay result includes a user's final score and word configuration for a round; this data is saved after each round of WordScuffle. Users will have access to view results independently of playing a game, available at any point from the main menu after they are logged in. A user will also have the opportunity to view results after finishing a round of WordScuffle. When a user views their gameplay results they will be given these results in relation to their community members' score data (if they are participating in a community). Viewable gameplay results include historical results occurring previously to the current day as well as a ranking with results for the current day relative to other members of the community.

Main Flow

The main flow for viewing and comparing challenge occurs when the user clicks on a challenge that they have completed. In this situation, the frontend directs the user to the View Solution component where their submitted solution is shown. At this time, the frontend sends off a request to the server to build a challenge leaderboard object based off the user's friends list. This list is based on the user's friends that have also completed the challenge. Once the server has built this leaderboard object, it is sent back to the frontend client where it is displayed for the user. When the user clicks on a name in the leaderboard display, the displayed solution is changed to the solution of whichever name they clicked on in the leaderboard.

Alternative Flow

The only alternative workflow for viewing and comparing challenge results occurs when the user clicks on a challenge that is not completed. When this happens, if the challenge is unlocked, the user is directed the the Game Component where they can then play the challenge. If the user clicks on a challenge that is not completed and is locked, nothing happens. In this case, the application stays the same so the user cannot view challenge results before they have completed the challenge.

Test Case 6: Manage Friends

After a user logs in, they will have access to a link in our applications main menu for accessing community information. This link will take them to a main section that gives them the option to manage their friends list. A user will also be able to view basic information like their friends list and pending friend requests. Any functionality outside of viewing the community in their user profile will redirect the user to the main community management page.

Main Flow

The main flow for managing friends occurs when a user navigates to the friends page and searches for another user by email address. When the user has found someone they wish to add, they can hit the plus button to send them a friend request. This places a pending friend request object in both users' data.

Another main flow occurs when a user wants to remove a friend. The user does this by clicking on the 'X' on a friend on the friend page. When a user does this, this sends a request to the server to remove the corresponding friend object from both users' data.

Test Case 7: Manage User Profile

When a user signs up for WordScuffle, they will be prompted to create a basic user profile with first and last name, email address, and a password. Further customization is not required to participate in communities or gameplay, but we will provide options for further personalization. Similar to managing communities and viewing gameplay results, a user will be able to access their profile for viewing or editing from the main menu at any point after logging in. From the user profile page, they will have the options to create an avatar, view personal information, manage their membership type (we are designing our web application with the possibility of monetization in future implementations), distinguish which information is viewable by others, and view basic information about the community they have joined.

Main Flow

The main flow for managing the User Profile occurs when the user clicks on their username and a drop down appears. The user clicks on "Account" where they are sent to the User Settings Component. This is where the user is able to update their username and clicking the "Update Username" button which saves the name for the user and is then displayed when the page is refreshed on the top left corner of the webpage. The user can also reset their password by clicking the "Reset Password" button which then displays to the user that an email

has been sent to the email registered with the user to reset their password. The user can click the “x” to close the message to reset their password.

Alternative Flow

Another alternative flow is when the user tries to enter a blank username. This is not allowed so the front end will tell the user to enter a username that is valid. The front end will not let the username be saved and it will display an error message when the user clicks on “Update Username”.

Test Case 8: Administration Console

Ms. Jenkins has requested an administration console which enables an administrator to manage variables about the game. We will provide authentication access to the console through the same login portal created for all users of the web application. An administrative user will have an additional option (beyond what a non-administrative user will have) to access the console from the main menu. The administration console will include options to remove or “ban” users from the web application in the case of use violation. It will include options to change global gameplay variables like the number of tiles generated per round (this affects the creation and layout of the game-grid as well as tile set generation), the number of rounds playable per day (this affects the number of tile sets generated per day), the time allotted per round and the score weighting. There will be separate global variables specific to the two different game modes.

Main Flow

The main flow for the administrative console occurs when a user is an administrative user. In this situation, when the user clicks on their name in the menu bar, they will see a dropdown option for the admin console. When they click on this link, a request is sent to the server to check if the user is an administrative user. If this check passes, the user is allowed into the admin console. At this time, the user has the option to modify game settings or manage the administrative user list. When a user modifies settings and clicks the save button, the frontend sends a request to the server. At this time, the server checks if the user that sent the request is an administrative user. If this check passes, the setting modifications are saved into the database.

Alternative Flow

The only alternative flow occurs when a user is not an administrative user. At any point within the main flow of this requirement a user is found not to be an administrative user, they will not be allowed to do anything. This means that they cannot navigate to the admin console.

However, if they navigate to the admin console and are subsequently inactivated as an admin user, all further requests they try to make regarding the admin console will return errors specifying that they are not an admin user.

Test Case 9: User Authentication

Logging in or authenticating is vital to our implementation because many of our other functional requirements require user authentication. For instance, providing an administration console requires our web app to differentiate between users to give administrative access to users only when appropriate. Participating in communities, score tracking, and comparing scores also requires the ability to remember a user's settings and previous history which is not possible without authentication. Creating and managing a user profile is also directly dependent on the ability to authenticate a user.

Main Flow

The main flow for accessing the User Authentication is when the user is at the home page for our web application. The user must click on the "Sign Up" button where a modal form is displayed by the front end to fill out. The user has to provide valid input fields for all inputs in the modal. Once the user clicks on "Sign Up", if the user has inputted valid inputs, the user can continue onto the web application and play WordScuffle.

Another main flow for accessing the User Authentication is when the user is already signed up to WordScuffle, the user can click on the "Login" button where a modal form is shown for the user to input the valid email address and password. Once entered the valid inputs, the user can click "Login" to be directed to the Daily Challenges Component and start playing WordScuffle. If the user has forgotten their password, the user can click "Forgot Password?" where the frontend displays a Password Reset modal. The user can than enter their email address and click "Reset", which then sends an email to the email of the user to reset their password.

Alternative Flow

One alternative flow can occur in multiple instances when a user is trying to sign up to WordScuffle. An error can occur when a user enters an email that is already taken by another user on Wordscuffle. The front end will display an error message to prompt the user to enter an email that has not been used yet. Another error than can occur is when the user enters a password into the password fields and the two passwords do not match. The front end will then display an alert to the user to input matching passwords. Another alternative flow for the User Authentication occurs when the user is trying to sign up, but has not filled out all the required fields. If this has occurred, the front end will send an alert to the user to fill out the required fields

that were missing. Once completed the user can then continue onto the web application and play WordScuffle.

Another alternative flow can occur on a couple instances when a user is logging into WordScuffle. An error can occur when the user enters the wrong email address or password, which the front end will then send an alert to the user, displaying which field was not correct. If the user enters an email that is not valid, the alert will display to the user that the email address does not exist in the WordScuffle database.

Test Case 10: Game Rules

Ms. Jenkins has supplied us with specific rules for the WordScuffle gameplay. There are general rules that apply to all gameplay, like every user getting the same generated tileset because users will compare scores for the specific tilesets, and there are specific rules that make the two game types unique.The player will have the ability to view the game rules for the different game modes that are offered.

Main Flow

The main flow for accessing the game rules occurs when a user is viewing the Daily Challenges Component and clicks on the “Question Mark” symbol. This is when the frontend displays a pop out on how to play the two game modes available. The scoring for each of the game modes is available for user to see as well. The user can access the game rules while playing each of the game modes on the Challenge Component following the process described above.

3. Integration Testing

Introduction

To aid understanding of how our app disseminates data, these are the main layers of our app:

Layer 3 (top) : Angular Components

These are the interfaces with which users directly interact. Components display and retrieve user input, but do not perform any direct communication with backend layers (they rely on Angular Services for this).

Layer 2 : Angular Services

Services that reside below components inside the Angular Framework. These services contain methods which directly write to or retrieve from Firebase (for user authentication related tasks) or pass requests down through Node.js’s server endpoints for all other tasks.

Layer 1: Node.js Server

Contains endpoints with which Angular Services interact. The server receives POST requests from Angular Services. For requests which require computation, the server computes them and responds to the Angular Service. For requests which require communication with Firebase, Node.js performs these read/write requests with Firebase and returns a response to the Angular Services layer.

Layer 0: Firebase

Our app uses two Firebase databases, one for user authentication and the other for all other game data. For any user authentication related requests, Firebase receives these directly from the Angular Services layer. For all other requests, it receives them from the Node.js Server layer.

Integration Test Cases

We have included test cases below for requirements which require the communication of more than one layer in our app.

ITC4: Validate Words

Main Flow

1.) A non-authenticated user has navigated to the `Demo Component` or an authenticated user has clicked on a Challenge on the `Challenges Component`.
2.) A user drags a tile onto the gameboard.
    a.) The `Game Component` updates a 2D array with configurations of where a tile is stored and sends this to the `Game Service`. The `Game Service` relays this 2D array to our Node.js server through an HTTP POST request. Our Node.js server recursively parses the 2D array and separates individual words which are then verified against a server-side hash table that holds all words listed in the Tournament Word List.
    b.) The Node.js server sends an HTTP response to the `Game Service` with a list of parsed words and whether each is valid or invalid. The `Game Service` relays this to the `Game Component`.

Expected Outcome

If data is valid

- If all words are valid, the Node.js server will locate all words in the TWL hash table.
- The Node.js server's HTTP response to the `Game Service` entailing parsed words will contain only entries that say words are valid.
- The `Game Component` will display the entire game board as valid (green).

If data is invalid

- If words are invalid, the Node.js server will not locate them in the TWL hash table.
- The Node.js server's HTTP response to the `Game Service` entailing the parsed words will contain an entry that says a word is invalid.
- The `Game Component` will display the entire game board as invalid (purple).

ITC5: View/Compare Results

Main Flow

1.) User has previously authenticated.
2.) When a user is logged in they are automatically redirected to the `Challenges Component` or they navigate there manually by clicking on *Challenges* on the main menu of the application.
    a.) At load, `Challenges Component` calls the `Game Service` which sends an HTTP POST request to our Node.js server to retrieve challenges from Firebase for the day by the current user's ID. This request includes information about all of the daily challenges as well as what challenge is the "next" available to be played. Data is filtered back up through the layers until it reaches the component, at which time it's displayed according to what's been played vs. what hasn't.
3.) User clicks on a past challenge
    a.) At load, the `View Solution Component` calls makes a call that filters through the same layers in a similar way to 2a above, except that it retrieves a single challenge by current user id and selected challenge. In this case Firebase returns jsonified data that expresses tiles that were placed on the board and their locations in a 2d-array format as well as the score for the challenge. This data returns up through the layers in a similar way and is then parsed back out onto the gameboard once it reaches the `View Solution Component`.
4.) User clicks on friends dropdown and selects a friends' score
    a.) Same process occurs as with 3a above, except the main retrieval relationship pertains to the clicked friend's user id instead of the current user's id.

Expected Outcome

If data is valid

- The view of which challenges have been played and their respective scores on the `Challenges Component` will be representative of what is stored in Firebase.
- Viewing a past challenge will deliver the same view of the game board as the last view the user saw when playing, including purple or green coloring if the board was invalid or valid respectively.
- Viewing another friend's solution will display a board with the same specifications as the point above, except for that user.

If data is invalid

- The `Challenges Component` relies on the number of objects returned from Firebase to generate challenges on the component so it will display a new challenge for whatever number of objects is returned. If no data is returned or if it returns something not parsable as an integer, the page will display no components.
- If a user selects a past challenge on their `Challenges Component` or views a past solution of a Friend and there is no data a truncated (smaller) game grid will show up and there will be no tiles present.

ITC6: Manage Friends

Main Flow

1. A user has previously authenticated with Firebase.
2. They click on the *Friends* link in the main navigation menu of our app.
    a. At load, the `Friends Component` communicates with the `Friends Service` to retrieve current friends. This service sends an HTTP POST request to the Node.js server with the same request -- The Node.js server sends a read request to Firebase to retrieve a list of current userIDs (friends) associated with the current userID. This data is filtered back up through these layers.
3. A list of friends is displayed on the `Friends Component.` Any pending requests (as recipient or sender) are also displayed on the friends list.
4. A user searches by email and adds a friend by pressing the *add* button next to a returned result -or- they accept/deny a pending friend request by clicking *accept* or *deny* next to a listed friend -or- they delete an existing friend by clicking *delete* next to an existing friend.
    a. The `Friends Component` sends a corresponding request (addFriend(), acceptInvite(), denyInvite(), deleteFriend()) that is filtered down through the same layers in a similar way as mentioned in 2a.
5. Any actions mentioned in step 4 are updated immediately in the component and viewable by the user without refreshing the entire page.

Expected Outcome

If data is valid

- The list of friends will accurately reflect what is in Firebase at any given time.
- A “friend” block will show up for each person that is a friend, or is pending in any way.
- Searching for a friend will show a result if that user is in Firebase, otherwise it will not.

If data is invalid

- If no friend objects are returned from Firebase, no friends objects will be displayed on the page.
- If the endpoint to the server is not working (lack of internet connection), there will be no status refresh to indicate that an action occurred because there will be no postback indicating a change. This means that nothing will visibly happen on the page.

ITC7: Manage User Profile

Main Flow

1. User is previously authenticated with Firebase.
2. They click on the *Account* link in the main navigation menu of our app.
    a. At load, the `User Settings Component` queries the `User Service` to find out the current variable for username. This is stored in current session for the app and so the `User Service` does not have to send any external requests when it is displayed on this component. The username field is displayed in the component view.
3. The user fills in a new username and clicks *Change username* or they click another button to *reset password.*
    a. The `User Settings Component` sends a corresponding request to the `User Service` which sends a write request directly to Firebase (no intermediate layers needed). Firebase responds with a success or failure.

Expected Outcome

If data is valid

- Firebase responds with no error message.
- A success message is displayed in the component view.

If data is invalid

- Firebase returns a descriptive error message.
- An error message is displayed in the component view.

ITC8: Administration Console

Main Flow

1.) A user has previously authenticated with Firebase and the current user settings within `User Service` have been updated to admin = true if the user is an admin or admin = false otherwise.
2.) The user sees the *Admin* option under the main dropdown menu for the app if admin = true. Otherwise, they do not see the link.

3.) User visits the Admin link if visible.
    a.) The `Admin Service` communicates with a Node.js endpoint through a POST request to retrieve current admin settings from Firebase. Once our Node.js server receives the POST request from the `Admin Service`, Node.js sends a formatted request to Firebase and returns results from Firebase once it receives them. The `Admin Console Component` then displays data received from the Node.js server's POST response.
4.) Once on the page, user can change game mode settings or administrator settings.
5.) If user wants to change game mode settings
    a.) user clicks on accordion section for game mode
    b.) user clicks on dropdown in particular setting and chooses new value
    c.) to submit changes, they click *Save Admin Console* button.
6.) If user wants to change administrator settings
    a.) user types an email address in the search box and submits the search
    b.) if a user account corresponding to the email shows up, they can add the user to the list of administrators by clicking the *add* button.
    c.) user wants to demote an administrator (leave them on list but remove their privileges, they click the *subtract* button.
    d.) user wants to remove an administrator entirely from list, they click the *x* button.
7.) For steps 6 and 7 and their subcomponents, a data interaction takes place when the user clicks any of the buttons listed. This triggers:
    a.) The `Admin Console Component` sends data which correspond to the currently viewed configuration of the administration console to the `Admin Service`. The `Admin Service` opens a POST request through an endpoint with our Node.js server. The Node.js server then sends a write request to Firebase to update currently stored admin settings to those included in the request.

Expected Outcome

If data is valid

- Admin Console displays data currently stored in Firebase.
- If data is changed, Firebase is updated to reflect what's viewed in admin console.
- Administrator changes take place immediately and gameplay changes take place the next day.

If data is invalid

- Our app prevents invalid data by providing dropdown buttons with specific values. Any changes to values of settings must come from a predefined list which our app supports.
- Hackish attempts to change data values are prevented by requiring that a user be authenticated as an administrator in association with changing values.
- Our app prevents non administrator access to the `Admin Console Component` by both removing links but also implementing an Auth Guard (this allows any form of direct access to this component by a non-admin user).

ITC9: User Authentication

A user must be authenticated to view the following functionalities:

ITC5: View/Compare Results
ITC6: Manage Friends
ITC7: Manage User Profile
ITC8: Access Administration Console

Main Flow

User has a previously created account, but the website is not logged in under a currently authenticated token.

1.) Visit WordScuffle's website
2.) User clicks on Login Link
3.) User fills input fields for email address and password and they hit *enter* on the keyboard or press *submit* to submit a login request.
4.) The `Login Modal Component` submits the request to the `User Service` which transmits data directly to the user authentication database in Firebase.

Expected Outcome

If data is valid

- Firebase will not return an error message to to the `User Service`.
- The `User Service` will its variables to contain all of the Firebase object's relevant settings for the user that requested a login.
- Any components throughout the app that require user authentication to interface with data will request data about the current user from the `User Service`.

If data is not valid

- Firebase will return a descriptive error message related to the login request. If the user entered an incorrect email address, Firebase will return the message "The email address is badly formatted". If the email address is correct but the password is wrong, Firebase will return the message, "The password is invalid or the user does not have a password."
- The Login Modal component will display an alert that displays the exact message that Firebase returned.

4. Usability Testing

For usability testing, our group will be focusing on the design of our web application and the user interaction. One requirement for the user interaction in our web application is giving the user a persistent menu that is available throughout the application. For our gameplay, we focus 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. This will be a focus on our usability testing to ensure new users can navigate easily throughout our web application. These requirements will be the focus points to understand how real users interact with our web application.

UTC1: Navigability

For our web application, we provided our users with a persistent menu that is available throughout the application. The only page this is not available on is the home splash screen. From this menu, the user has access to every page around the application. The only pages that can't be reached through the menu are the Game Component and the View Solution Component, because these pages require the user to click on specific challenges on the page. For these reasons, the user is able to make it to any functionality of the application within three clicks.

To inform the user on how to navigate throughout our web application we incorporated buttons that will guide the user around the application. For each button we handpicked icons and verbiage to reflect the functionality that will be performed. An example of this is shown on the Friends Component where you are able to add and remove a friend. When adding a friend you can type in an email to search, once the email is typed and you have clicked to search, the user's name will appear and a green plus sign is available for the user to click on to send a request for the user to add as a friend. Once users are friends, they will appear in a list on the Friends Component. The user then has the option to remove the friend by clicking on the red 'x' button which the user that has been deleted will not be shown in the list. Another example that can be seen throughout the web application is the view instructions button. We used a 'question mark' symbol for the user to click, which will then explain how the user will play the game modes. In conclusion, we have provided buttons throughout the application with icons that reflect the functionality that will be performed. Doing such has allowed us to ensure users will be able to use our application without difficulty.

Measurement:

The user must be able to make it to any functionality of the application within three clicks.

UTC2: Gameplay Usability

Another aspect of usability testing we will focus on is gameplay. For our situation, we needed to be able to ensure that gameplay functionality is obvious enough for the users. Because we place an "instructions" component on the gameplay page, we believe that the user will be able to easily understand how to play the game. However, to assist the user, we have tried to make all functions related to gameplay intuitive. One way we do this is by changing the user's cursor to reflect the actions that are available to certain objects on the page. For example, when the user hovers over a tile, we change the cursor to a hand to reflect that they

can click and drag the tile. Another way we assist the user in understanding what is going on in the game is by coloring the game grid. When a user places a tile, the game grid is colored green if all parsed words are valid. If invalid words are encountered, we color the user's grid purple. This coloring happens within half a second of placing a tile, which provides the user with almost instantaneous feedback whether they word grid is valid or invalid.

Our application has been hosted online for over a month now, and we have been able to invite friends to play the game and give us feedback on usability during this time. Between reactions from the people playing during this 'closed beta' and Barbara's feedback, we have been able to make interface changes that create the interface that everyone is happy with as it is now.

Measurement:

The user must be able to make it throughout the gameplay in four clicks. While the user is playing a challenge, the user can drag and drop a tile within one click, and to submit the challenge the user can click once. To view previous solutions from the daily challenges component, the user can navigate to their solution in one click, and to view their friends scores they can navigate to it within two clicks once at the view solutions component.

5. Conclusion

In conclusion, our team BrainStim Studios has finished developing the Wordscuffle web application and the next big step towards a finished product is completing the tests described in this document. We had these tests in mind during the development phase, and now will be running hard-coded tests to assure the code works correctly and covers all possible workflows. Our unit tests focus on smaller scale communication of data, while the integration testing focuses on the organization and distribution of the data throughout the application. Finally, our usability testing focuses on providing a streamlined user experience. Successful results in these various tests will ensure that our application is ready to be made available to a larger number of users.