Technology Feasibility Analysis

Date: 11/8/2017

Project sponsor: Barbara Jenkins

Team faculty mentor: Ana Paula C. Steinmacher

BRAINSTIM STUDIOS

Vincent Messenger (Lead)
Anderson Moyers
Nathan Franklin

Table of contents

1. Introduction

Alzheimer's Disease, or AD, is a progressive form of dementia which gradually destroys mental functions and memory. It often manifests as short term memory loss in early stages, progressing to pervasive, long-term memory loss. Cognitive function declines until bodily functions are impaired, ultimately leading to death: Alzheimer's is the sixth leading cause of death in the U.S. and as of 2015, there were an estimated 29.8 million people suffering worldwide from AD. Research has shown that engaging in intellectual activities may reduce your risk of symptoms associated with AD.

Brain stimulation games are a method of Alzheimer's Disease prevention that has gained attention in the last 5-10 years. Current solutions such as Lumosity make use of this concept by presenting users with stimulating brain challenges. Lumosity is an online site that uses scientific research about the brain to create games that enhance cognition in different aspects. 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. There is one aspect however that many brain game platforms, including Lumosity, do not address: **social interaction**.

Research has shown that regular stimulating interactions can also reduce the risk of Alzheimer's Disease. 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. Words will be constructed in a grid-like fashion, which allows words to overlap. Once a game is finished, the user's score is calculated and they can compare their score and words with other users 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 everyday 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.

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.

In this technology feasibility analysis document, we discuss our primary needs to implement a web application version of WordScuffle, the possible technologies which could fulfill these needs, and our ultimate choices and conclusions for specific technologies. The team needs to investigate all possible technologies for each aspect of the project, because changing a technology after development has begun will result in wasted time and effort. Throughout this paper, we will analyze our technology options for user authentication, server language, database, word validation, and Javascript framework.

2. Technological Challenges

Our team has identified four main challenges we must address before we begin developing our web application: a.) establish an authentication system and database, b.) create our own server, c.) determine a method of word validation and d.) decide on a Javascript framework.

We will need a secure method for authenticating users, as well as a database to store user data. Both of these are necessary to maintain history and contextual data for players; this data is vital for our web application not only because it enables players to participate in communities and track their scores but because it supports the broader purpose of our web application: providing social and cognitive stimulation to support brain function and reduce the symptoms of Alzheimer's disease.

To facilitate communication between the frontend and the authentication and database services, we will also need to create our own server. Creating a server is necessary for gameplay integrity because it allows us to abstract data and perform tile set generation remotely instead of on a player's device. Using a server will also better support general data storage for our implementation because it is a web application, not a standalone application.

Word validation is necessary because we seek to reduce word misspellings in our implementation. Since this web application will be a timed game, we need a quick and reliable way to validate words that users create.

Lastly, we must find a suitable Javascript framework to communicate with the server and to provide tools for UI manipulation. We decided on Javascript because we require a robust modern client-side framework to create a fast-paced user experience that involves frequent object manipulations from users. To simplify development and maintain feasibility, we need a framework that has good compatibility with the other technologies we choose and we must find this framework very early in the development process.

3. Technology Analysis

In this section, we give in-depth analysis for each of the requirements that we proposed earlier in this document. For each requirement, we compare our top three technology choices, and justify our final decision.

3.1 Authentication

In order to keep track of individual player data, we will need to give users the ability to create an account and authenticate into the account at later times. Because user accounts may contain personal information related to profiles and payment information for monetization, security is vital. When a user visits our web application through a browser, they will have to login in order to see any of their user data. Once logged in, they will also have the option to sign out of the system.

When researching various web application authentication methods, we were looking for options that would be easy to setup and maintain, provide robust authentication functionality, and that were secure. We found a couple highly recommended methods, which were Firebase and Passport.js. Based on our research, we believe that both of these methods would work for our web application. However, there is also the option of creating our own authentication that should be explored.

3.1.1 Firebase

Firebase is a service that is backed by Google. That means that it is built on Google's infrastructure and provides automatic scalability. It has an entire suite of products that developers can mix and match to provide robust functionality to users. One of these features is user authentication. It's also important to note that Firebase is cloud based. It also provides workflows for resetting user passwords and other admin-like requirements. The main drawbacks of using Firebase relate to future support: Firebase currently offers (for free)

authentication services which are highly customizable and powerful. The options and costs may change with time and require additional developer support to stay abreast of these changes.

3.1.2 Passport.js

Passport.js can be easily added into web applications to provide user authentication. It currently provides 307 different strategies for authenticating uses, including social sign-on and numerous Open Authorization (OAuth) providers. However, based on our research, it seems that there is still some configuration setup required to get the authentication functioning. Also, the user data would live on the server that is serving the application, which could increase the cost to host the application.

3.1.3 Creating Own Authentication

Creating our own authentication is an option that is not practical for our project. We need a secure way to store our user data, which our team could definitely achieve. However, it would be unwise for our team to spend the time to create a robust authentication library that provides the features that are already provided by other established methods such as the ones listed above.

3.1.4 Conclusions

According to the table below, Firebase is the option that meets all three core requirements we seek in an authentication system: this option is the easiest to integrate into web applications, it provides numerous features that the team can use, and has excellent security. Overall, Firebase will take the least amount of time to implement and maintain, which is important to the team so we can focus on bigger pieces of the project.

	Ease of Setup/Maintenance	Robust Functionality	Security
Firebase	x	x	x
Passport.js	x		x
Create Own Auth			x

3.1.5 Proving Feasibility

Our basic plans for proving the feasibility of using Firebase are to simply implement user login and logout functionality in a web app. We have researched many tutorials of setting up Firebase in a web application, which includes the Firebase documentation, and we have

determined that Firebase will be quick and easy to implement using whichever Javascript framework we choose to use.

3.2 Server Language

For this project, the team will need a server for communicating with the authentication service, database, and web applications running on individual client devices. It will also be responsible for serving the web application front-end code.

There are a few things to consider when choosing a language with which to write a web application server. We need the server language to be easy to develop and maintain, we need available external libraries, and we need the language to be easily compatible with Javascript. Based on team experience and research, we have narrowed our server language options to Java and Javascript Node.js. There is also the option of creating a serverless web application that should be explored.

3.2.1 Java

Java is a strongly typed language that has been used to create application servers since its emergence in the late 1990s. It has a sizable development community, which means there are many external libraries that can be used. One benefit to using Java is that we could take advantage of multithreading. However, since our server logic will not be overly complex, utilizing this sort of functionality might be unnecessary for our web application. The downsides of using Java for our server are related to ease of use and compatibility. A Java-based server would have less inherent compatibility with our Javascript-based front-end than a Javascript-based server. Furthermore, although Javascript is very robust it is also more complex, would require classing for all functionality, and has numerous features we do not need based on the complexity of our server-side needs.

3.2.2 Node.js

Node.js is a server language written entirely with Javascript. It is extremely simple to write a server to serve a web application's frontend code in Node.js. Also, it can be achieved without the overhead of creating classes to communicate with each other, which is a requirement in Java. Also, because Node.js is written with Javascript, it is readily compatible with Javascript libraries and data structures. Another benefit to using Node.js is that we could easily import external libraries using Node Package Manager (npm). It is important to note that certain members of the team already have extensive experience with using Node.js to serve web application frontend code and data. The drawbacks of Node.js relate to robustness and volatility: Node.js does not support multi-threaded processing and its API frequently undergoes changes which may require developers to maintain code.

3.2.3 Serverless Web Application

A serverless web application is possible to achieve if there is little need for a server, but there are definite negative impacts due to this. For one, all logic would take place on the client's device, which could potentially lead to security risks. Since we are building a web game that incorporates community score features, it is a bad idea to allow communication with our database to be handled by only the client's device. This would also expose details about the administration section of the web application that should be kept hidden. By bypassing any type of server, we also place restrictions on functionality that can be explored with the web application.

3.2.4 Conclusions

The table below outlines ease of development/maintenance, library support and Javascript compatibility as our main requirements for a server technology. Node.js meets all three of these. Beyond attributes already discussed, Node.js has the benefit of being able to effortlessly use and serve Javascript Object Notation (JSON) data, and research has shown that Node.js is quicker at performing the sort of tasks we would be building into our server logic. Also, since both the client and server will be written in the same language, we will not have to rewrite logic in a different language if we decide to move code between our server and client.

	Ease of Development and Maintenance	External Libraries	Ease of Javascript Compatibility
Java	x	x	
Node.js	x	x	x
Serverless App			x

3.2.5 Proving Feasibility

To prove the feasibility of using Node.js as our server language, the team plans to develop a server that will serve the frontend code for our web application. We will also add routes for user account creation, user login and logout, and saving user data to our database.

3.3 Database

User scores will be stored in a database every day, which will allow users to compare their solutions with other game players. High scores can be viewed on a worldwide scale, or by user-defined communities of users.

We researched different database technologies with a few key features in mind. Compatibility with Javascript means it will be easy to implement with the rest of the parts of the project, because Javascript is what we are using for the other main components of the project. We mainly focused on open source database technologies for cost reasons. The team is also looking for options that are easy and cheap to host. After comparing database technologies, we narrowed the decision to one of the top-rated SQL databases, one of the top-rated non-SQL databases, and Firebase.

3.3.1 MySQL

MySQL is the top-rated open source SQL database. It uses a relational database system which is great for complex queries. One key factor for including this in our evaluation is that each member of the team has experience with this technology. Because of this knowledge of MySQL, creating a database and constructing queries would be easier than learning a new database structure. However, since our user data is not overly complicated, using a SQL database could be unnecessary and lead to over complicating our application logic.

3.3.2 MongoDB

MongoDB is a database that is structured using Javascript Object Notation (JSON), so inherently it is compatible with Javascript. It is a NoSQL database, which means that SQL does not need to be used to retrieve stored data. Instead, the database is accessed using a Javascript-like syntax. Due to the absence of a structured query language, there are restrictions in terms of reporting that can be done, because it becomes more complicated to report from NoSQL databases. However, due to our specific requirements, our user data will not be overly complicated, and therefore we should have no issues retrieving data in any way we need.

3.3.3 Firebase

Firebase provides a real-time database that can be used with or without its authentication functionality. This real-time database is a NoSQL database that stores its data in

JSON. Since Firebase is provided by Google and is hosted as a cloud service, Firebase's real-time database is reliable. This also means the team would not have to find a place to host the database. It is important to note that Firebase provides free storage to a certain extent. When the project scales, the team would most likely need to eventually pay for additional storage. However, for the purposes of this project, we should be able to develop the web application without incurring any such costs.

3.3.4 Conclusions

Based on the database criteria in the table below, our database choice is Firebase. Because the rest of the project will use Javascript, it is important for the database to work well with Javascript. Since Firebase offers free storage, we will be able to avoid incurring costs until the project scales. With MySQL and MongoDB we would have to find a solution for hosting the database, but Firebase hosts their own databases and includes pre-existing functionality that we would otherwise have to implement on our own server. We have already determined that Firebase fits our needs for authentication and so we can simplify our database and authentication usage by combining them with one service.

	Includes Host Server	Free to Use During Prototyping Phase	Compatible Javascript
MySQL		x	x
MongoDB		x	x
Firebase	x	x	x

3.3.5 Proving Feasibility

To prove the feasibility of using Firebase's real-time database, the team will setup endpoints to facilitate communication between the Javascript framework and the database. We will make it possible to save and edit certain pieces of user data from the frontend client.

3.4 Word Validation

Ongoing word validation is a core functionality of our web application: with every tile placed on the game grid, any combinations of two or more tiles in a row will be checked for validity using a word list similar to the dictionary Scrabble™ uses.

Features to look at are how fast this validation happens; validation should happen quick enough to give the appearance of instant validation. This application has to run on tablets and smartphones, so available client-side resources will be limited compared to the computers we will be testing this application on. There are not a lot of available tools for word validation, but the tools we narrowed our options to are WordGameDictionary.com Validation API and Word-List. However, we will also be exploring the potential of creating our own word validation library.

3.4.1 WordGameDictionary.com API

WordGameDictionary.com has an API that developers can use to validate words. The free version is for non-commercial use and allows up to 5000 requests per day. Because validation happens on a server and transfers tiny amounts of data, the validation can be almost instant and is not dependent on client device resources. We have deduced that the three-minute, thirteen tile game will average thirty word validation calls. There are ten available games per game mode each day, so we will need to be careful how we set our application up to send requests to this server. We need to plan our solution to prevent sending 5000 requests a day, so we can avoid incurring extra costs.

3.4.2 Word-List

Word-list is a GitHub project with an MIT license and uses the same word list as the WordGameDictionary.com API. There is very little information about this project, and we would still need to manually implement validation if we use this technology. This package basically provides a word list that developers can then use to build their own validation.

3.4.3 Create Own Validation Library

Making our own validation would be time consuming, but would be specific for our needs. There is a tournament word list to start with, but searching through every word looking for the passed in word would be inefficient. This option is not our preference, because the team would like to spend more time on other aspects of the user experience.

3.4.4 Conclusions

According to the criteria listed in the table below, the WordGameDictionary.com API and implementing our own validation are our two best options. Ultimately, we choose the WordGameDictionary.com API because the provider already has the infrastructure infrastructure to implement validation. While implementing our own word validator would be possible, making it run efficiently enough to appear to validate in real time on any client device would take too much effort and time away from the overall development process.

	Easy to Implement	Free after Scaling	Not Reliant on Client Device Resources
WordGameDictionary API	x		x
Word-List		x	
Make our own		x	x

3.4.5 Proving Feasibility

To prove the feasibility of using the WordGameDictionary.com API to validate words for our web application, we will create proof-of-concept validation with basic words in an interactive prototype. Single words will be checked against the API to verify results. Because latency and number of queries is an important factor of feasibility, we will record validation times and use estimates of usage to determine how many queries will be submitted daily.

3.5 Javascript Framework

There are many advantages to using a Javascript framework in a web application. Javascript frameworks save developers large amounts of time and effort by providing well-structured pre-built patterns and functions. It is also much more secure to communicate with servers using Javascript frameworks due to provided functionality.

When researching Javascript frameworks, we sought options that would a.) work well with our technology decisions for authentication and database and b.) present the smallest learning curve for our team and c.) maximize external library support. We narrowed our options down to Angular 4, ReactJS, and Vue.js.

3.5.1 Angular 4

Angular 4 is the third version of AngularJS. It is an MVC-based Javascript library that was acquired by Google soon after its release in 2009. Angular provides ways to develop web applications supported across all platforms. One benefit is that since Angular and Firebase are both Google products, they have strong compatibility. Another benefit is Angular has two-way binding, meaning it provides ways to keep frontend data synchronized between the database and frontend clients. Angular also has a large development community, with external libraries being added every day. It is also important to note that as an added benefit, members of the team have extensive experience using Angular to create web applications. The drawbacks of using Angular are its learning curve and its maturity. Angular is verbose and can be complex to learn: although most of our team members have experience the complexity is increased for

team members without experience. Also, Angular has gone through significant paradigm changes between AngularJS (original version) and Angular 4. Although many developers are involved in the community, the structure of code and paradigm is relatively new.

3.5.2 ReactJS

ReactJS has grown in popularity in the development community in the past few years. Unlike Angular which is MVC based, ReactJS is more concerned with the task of creating user interfaces. ReactJS is compatible with both Firebase and Node.js, and the team has found that it is simple to add Firebase integration to a ReactJS application. Based on our research, it seems ReactJS has a relatively small learning curve, however since it is basically more of a view library, it could take time to get the team accustomed to best practices. It is also important to note that no members of the team have development experience using ReactJS.

3.5.3 Vue.js

Vue.js is a Javascript framework that in its core is only focussed on the view layer of the frontend client. However, it is advertised as incrementally adoptable, which means developers can integrate more robust Vue.js functionality as they find the need. This means that for simple web applications, Vue.js can be kept as a lightweight package, which is beneficial for speed and efficiency. Vue.js is compatible with both Firebase and Node.js, which would mean the team could use Vue.js with the other aspects of the web application. However, since Vue.js is a relatively new Javascript framework, there could be a bit of a learning curve for the team.

3.5.4 Conclusions

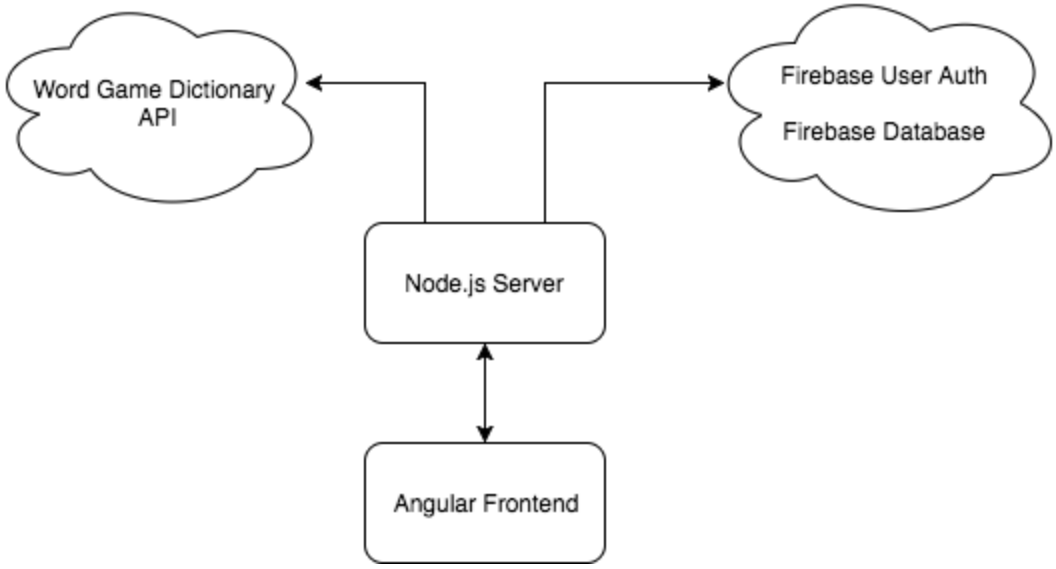
Our chosen option for a Javascript framework is Angular 4. Because Angular is an MVC-like Javascript framework, the team can take advantage of modularity and robust functionality for UI manipulation and communication with our various data services. The team can also make use of Angular's two-way binding to keep user data synchronized across devices. Because more members of the team have extensive experience developing web applications using Angular than those who do not, we feel that the overall lowered learning curve will be a really huge benefit for us in this project.

	Compatible with Server and Database	Small Learning Curve	External Libraries
Angular 4	x	x	x
ReactJS	x		x
Vue.js	x		

3.5.5 Proving Feasibility

Through previous experience we have determined that an Angular 4 application can be served by a Node.js server and that Angular can be updated and installed with the NPM package manager in Node.js. To demonstrate feasibility for this project, we will create an example Angular 4 front-end prototype and show that a.) this front-end communicates with the Node.js server and b.) it works with key graphical libraries such as Dragula (a drag and drop library) which we intend to use for our UI and gameplay. Through this prototype, we will prove UI library compatibility by allowing user to drag and drop letters to form a word which will then be submitted to the Node.js server. We will then return a result to the front-end which shows feedback about validation of the word.

4. Technology Integration



Above is a diagram of our planned prototyped system. Frontend code will first be served from the Node.js server to the users web browser. The Angular client that is served to the user's web browser makes requests to the Node.js server in order to communicate with Firebase user authentication service and the Firebase Real-time Database service. The Angular frontend will also make requests to the Node.js server in order to communicate with the Word Game Dictionary API to validate words as user construct them with tiles.

5. 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. Our web application will:

- expedite and simplify scoring, allowing players to focus on having fun
- improve game integrity by preventing players from viewing results before they submit their own
- provide word validation to lower confusion and reduce scoring mistakes

The table below gives an overview of our chosen technologies and our confidence level that we can successfully use these technologies in our project. We have chosen technologies that will expedite and simplify parts of our project that would be very complex if we implemented ourselves but for which there are pre-existing technologies and libraries. Using Firebase for our user authentication and database and using the WordGameDictionary.com API for our word validation are two such examples of this. This approach will allow us be as efficient as possible with our development process so that we can focus our attention on user interface and gameplay logic.

	Proposed Solution	**Confidence Level (1-5)** **1-strongly not confident** **3-neutral** **5-strongly confident**
User Authentication	Firebase	5
Server Language	Node.js	5
Database	Firebase	5
Word Validation	WordGameDictionary.com API	5
Javascript Framework	Angular 4	5

Now that we have our chosen technologies, we are excited to start prototyping WordScuffle. We are confident that we will be able to use these technologies to successfully create a web game to help prevent the onset of Alzheimer's Disease.