QuickSPLASH

SENG 513: WEB-BASED SYSTEMS

Group 16

Mona Agh

Hamzah Umar

Masroor Syed

Niroojen Thambimuthu

Artemiss Rahim

Ahmed Hasan

SENG513 Final Project Report

April 2019

Table of Contents

1 Introduction

When a large group of people get together in social gatherings, party game can be used as an icebreaker and provide entertainment. Due to technology improvement, there are various online party games that can be played by people with various skill levels since party games are intended to be easy for new players, and no one in social gatherings is supposed to be eliminated from the game. Therefore, we decided to implement a web-based and mobile friendly online party game which is called Quicksplash for everyone at a lower cost.

1.1 Background

As it is inferred from the name, party game is supposed to be played when a large group of people get together, therefore it needs that every player be present physically in a same place. This can be a shortage for games since people might not be able to meet each other due to various reasons such as lack of time and money or living far away. However, Quicksplash is designed in a way that provides an opportunity for everyone to play the game from their comfort place and feels the vibe of a party at a lower cost.

The user of our system is every people with different ages who knows how to use a computer or mobile at a minimum level and are interested in playing games. Since party games are supposed to be easy for new players, we provide a game with simple rules that it does not need any experience in gaming. People can use our game when they come together or use it as a place to meet virtually.

1.2 Project Goals

The goal of our project is providing the users with a free gaming platform that targets everyone who looks for a party game to use in social gathering or connecting with their friends. Also, we allows players to personalize their games based on their preferences.

1.3 Project Accomplishments

The current implementation of QuickSplash needs its users to make an account in order to play the game. After logging in, players are able to create a lobby and invite their friends by sending them the lobby code, or join a lobby that has been created before. Players who host a game, can specify game rules such as number of rounds and lobby size. After starting the game, each player is provided with some questions with no real answer which they should try to respond with clever and funny answers at a specific time. Then, the game changes to voting phase which everyone should vote on other player's response to their questions. The players cannot vote on the same question that they answered. At the end of the

round, the top three players who got the most votes are presented in a leaderboard, and the winner will be announced. Also, each player has a profile page which track of their informations and gaming statistics.

2 Project Description

2.1 Home

Upon opening QuickSplash, a new user will be presented with the following page. From this point a user could navigate to login, create a lobby, join lobby, or how to play pages. To create or join a game, a user must be logged in. Attempting to create or join a game will reroute a user to the login page, thus, from this point, they must either visit the login page or the how to play page. This page demonstrates sprites we have created for the game, well as animations and sounds we used on top of them.

Figure 1: QuickSplash appearance upon page load

In the case that a user selects “How to Play”, they will be taken to the following page. This page outlines the rules that manage QuickSplash.

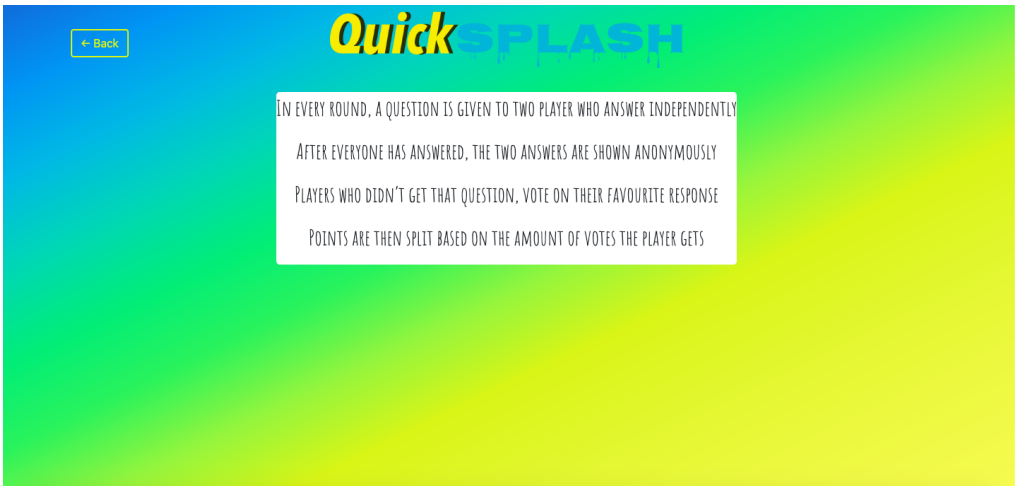

Figure 2: How to play page

2.2 Login

If the user wishes to login they will navigate to the following page. From here a returning user will be able to login to their account, whereas a new user may sign up for an account to play with. Any incorrect validation information or improper sign-up credentials (e.g. weak passwords or taken usernames) will be informed to the user via a message on the screen.

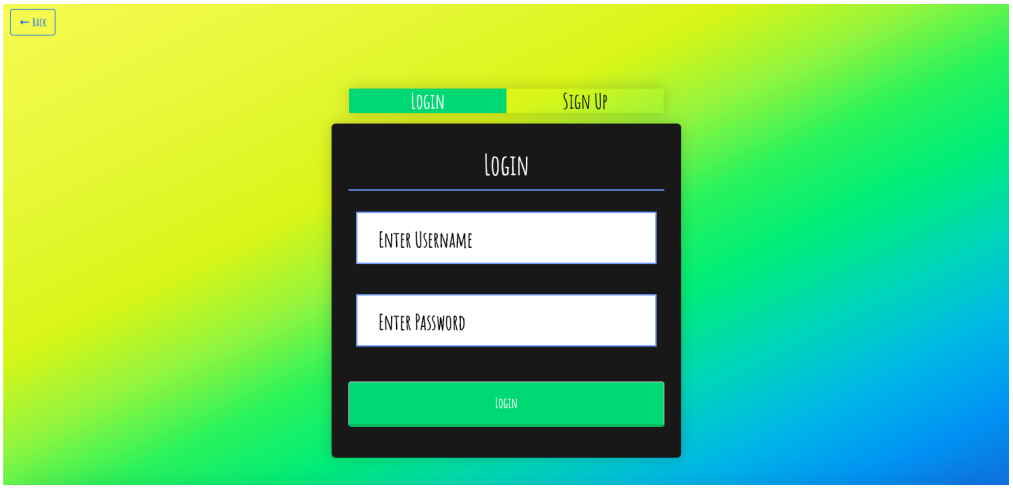

Figure 3: Login page upon navigation

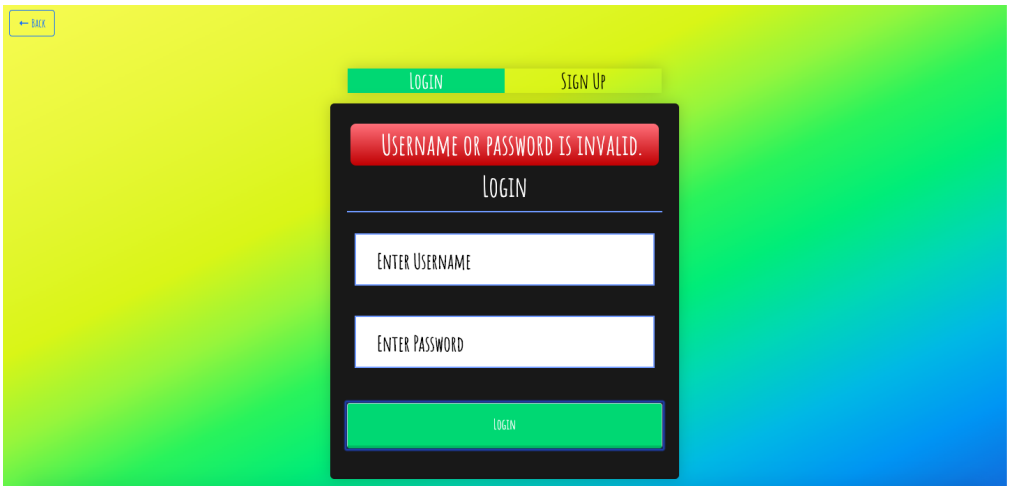

Figure 4: Message pop-up on incorrect validation information

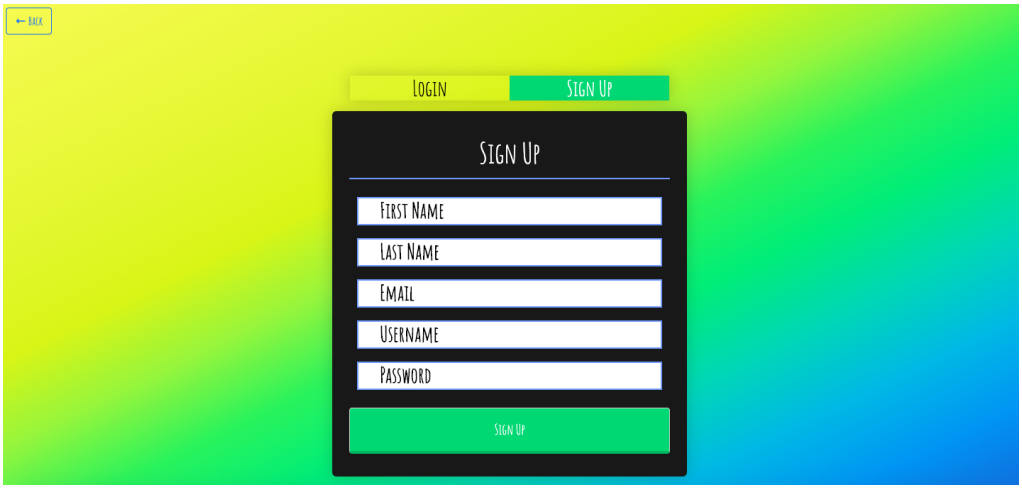

Figure 5: Sign up tab of login page

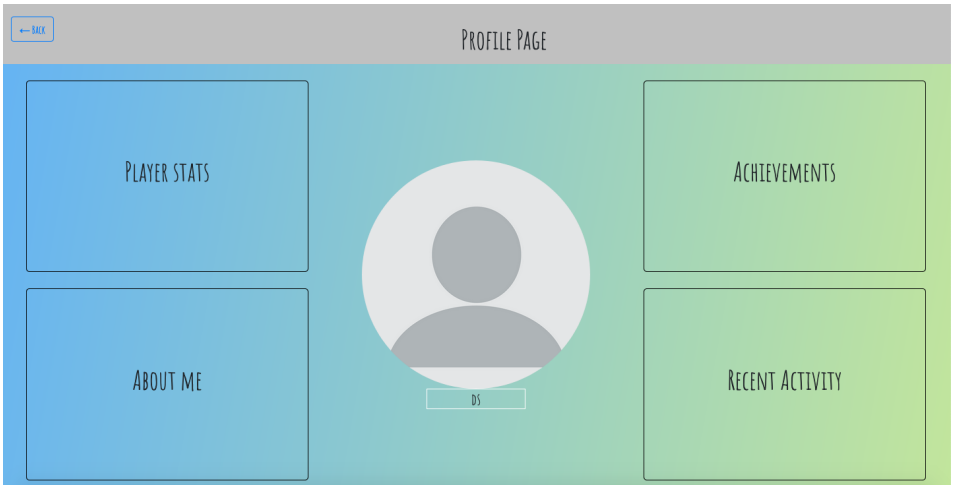

Figure 6: The profile page

Figure 7: Home page for logged in user

2.3 Lobbies

From the create a lobby page they may create a lobby according to their preferences. The options include time per round (range = 20 seconds-80 seconds), lobby size (range = 3-8 people), number of rounds (range = 1-5), and AFK timeout (1 minute-5 minutes). Once a user has selected their preferred settings, they can select the 'Create' button, which will take them to the next page.

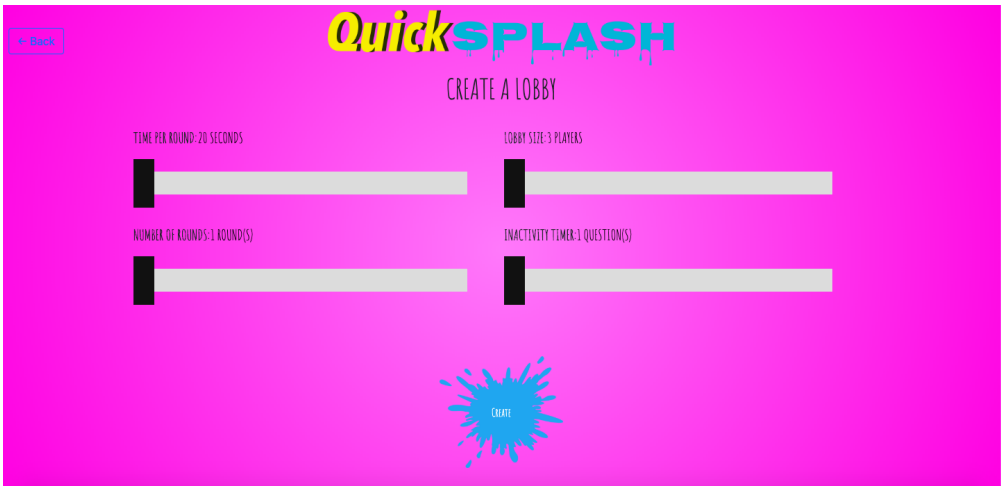

Figure 8: Create a lobby page

The following page is the join lobby page. In the case the user navigates here from the home page, both join code and nickname input fields will be blank. In the case the user navigates here from the create a lobby page, the join code input field will contain the code for the lobby they have just created. After the fields contain the appropriate information, the user may use the 'Join' button to join their lobby. In the case the join code is incorrect, an alert will inform the user to try again.

Figure 9: Join a lobby page with a preset join code

2.4 Game

The creator of the lobby will be taken to a lobby containing their sprite matched with their username, the lobby code, and the a button to start the game. The game music will begin playing once the user has entered this page. In the case that the number of users is not the lobby size and the host selects the 'Start' button, an alert will play, informing the user they need a different number of users in their lobby to play the game.

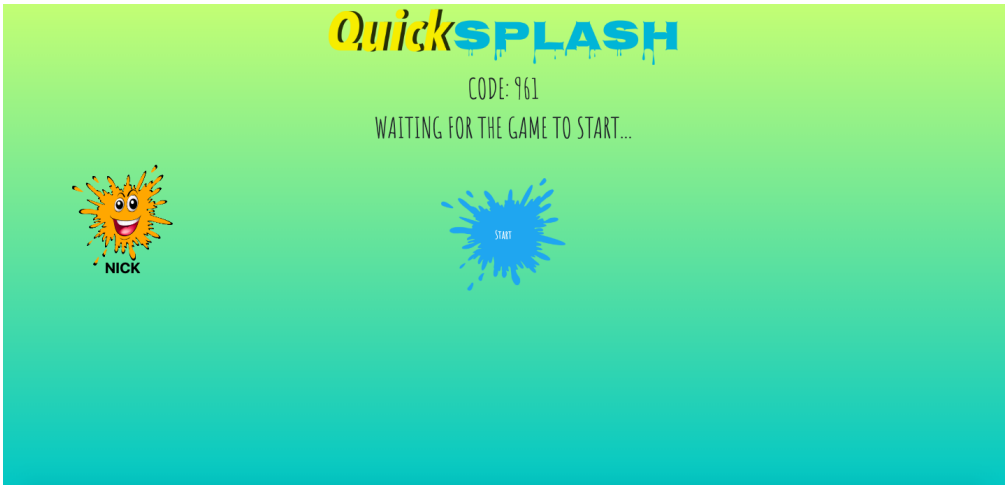

Figure 10: Waiting screen for lobby host

A player which is not the lobby host, upon joining will be taken to a similar waiting screen, however, excluding the start button. Once the number of users is appropriate, the lobby host may start the game.

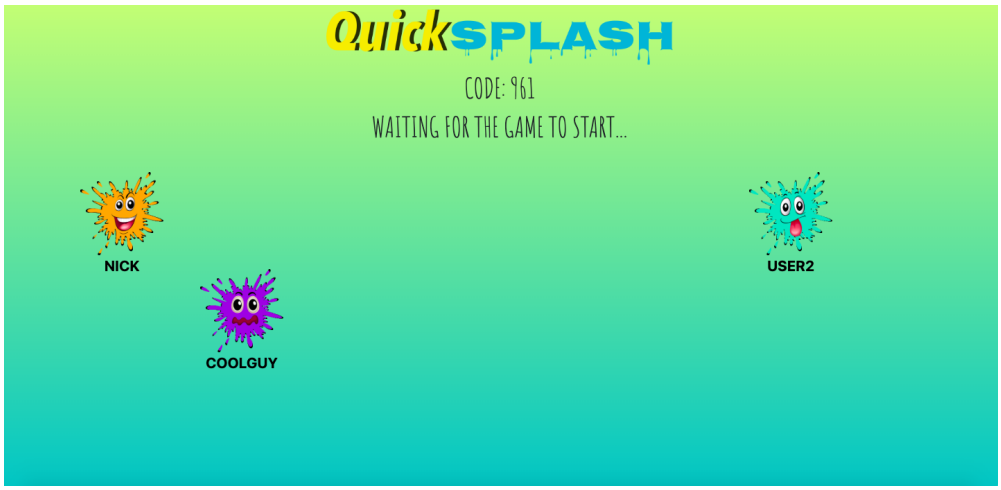

Figure 11: Waiting screen for a non-host player

A round transition will play informing the user of the upcoming round. The text informs individuals on the instructions, being to answer two questions.

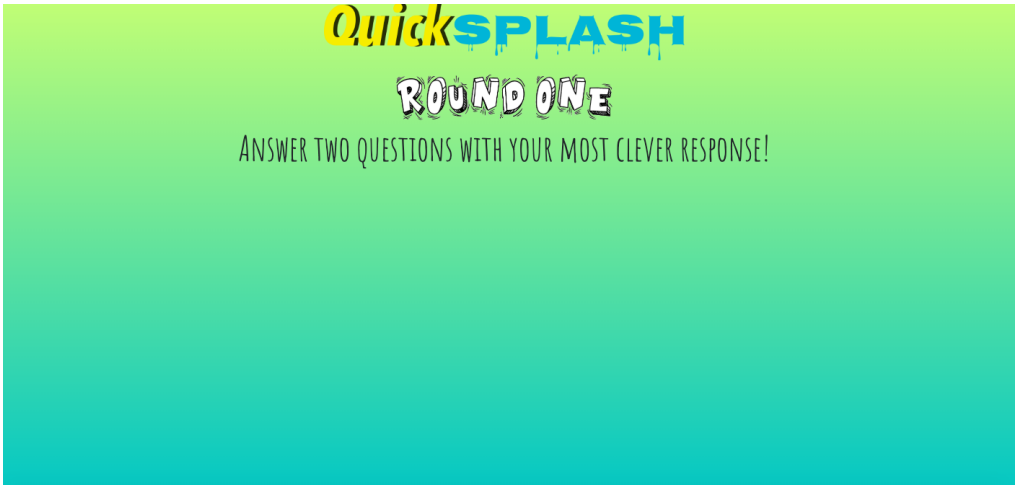

Figure 12: Round transition screen

Following the transition, the user will be prompted with a two questions, to which they must provide a clever response to. The question will also be assigned to another, random player in the lobby, with the answers being pitted against one another upon the voting phase. A timer will be present in the upper left corner informing the player of their remaining time in seconds. The timer represents the time remaining for both questions, rather than each individual question.

Figure 13: The prompt screen

Once all users have input their answers, they will be taken to a voting screen, wherein users that have not answered a particular question will have the opportunity to vote on their favourite answer. Every vote will allocate 100 points to the user which wrote it. In the case that a user did answer the question, they will be presented with a screen informing them that they may not vote this round.

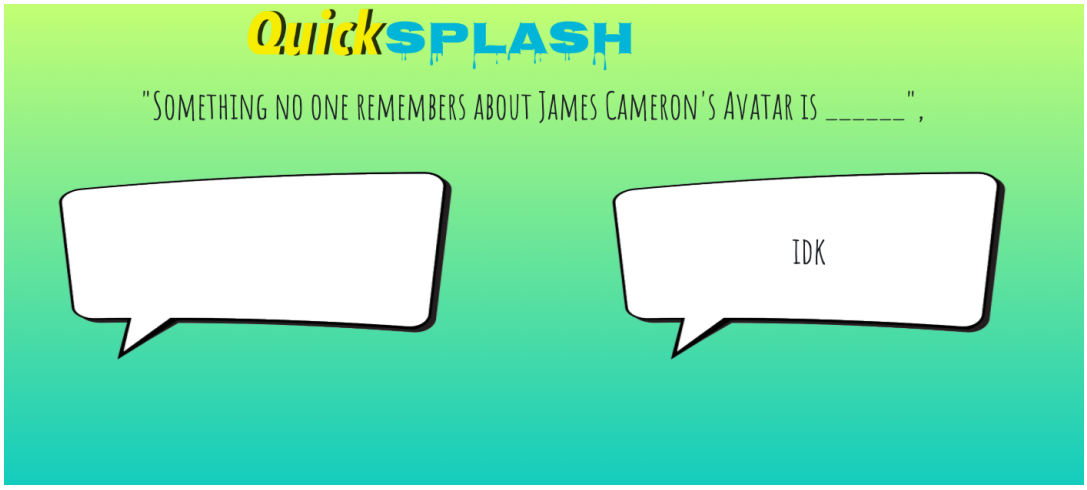

Figure 14: Voting screen for user which answered the question

Figure 15: Voting screen for user which did not answer the question

After the voting page, the user is taken to another waiting page, displaying all users which have finished voting. After this, all users are taken to the results screen. This displays the top three players along with their point count and relative ranking.

Figure 16: The post-vote waiting screen

Figure 17: The results screen

This process will repeat for the number of rounds the host had selected upon lobby creation, after which a final results page will be shown, showing the winner of all rounds, navigating back to the home screen after a short period. The users profile stats will be updated at this point with their game stats.

3 Proposal vs Final Project

In our project proposal, we identified 6 different features that we planned on implementing. They are listed in order of priority as identified in the project proposal.

1. A QuipLash style multiplayer game

   Status: Implemented

   This was the very core of our application, so implementing it was paramount to the successful completion of our project. Our final implementation includes a complete multi-player game experience that is based on the popular game QuipLash created by Jackbox game.

   We implemented a complete, real-time party game where users can submit answers, vote on others' answers, and view the results at the end of every round and at the end of the game itself.

2. In-Game chat between players in the same lobby

   Status: Not implemented

   We chose to not implement this feature for a couple of reasons:

- When creating the front-end for our application, we found that the layout looked rather cluttered when a chat was displayed simultaneously with the game. This was particularly true when the application was used on smaller screens

In hindsight, listing this feature as the second-most important feature for our game might have been a mistake. For example, Quiplash (the game from which we drew inspiration) does not feature an in-game chat either.

3. Account creation

   Status: Implemented

   The final implementation of our application allows users to create accounts with their own usernames. Additionally, the account created by users tracks a few important statistics from their gameplay. These include the number of games a user has played, the total number of points they have amassed across their games, and the total number of wins that have managed to get.

4. Creation of private lobby by host

   Status: Implemented

   In our web application, a user can select the 'Create Lobby' button from the home page. Once a lobby is created, the server generates a random, unique code. Other players who want to join this lobby must enter the unique code to do so. This ensures the privacy of the lobbies as we had intended

5. Customizable rules for a private lobby

   Status: Implemented

   On the 'Create Lobby' page described above, there are 4 sliders that a user can change to customize the rules of the lobby. The 4 rules that can be changed are:

   - Time per round: This sets how much time each player has to answer their questions
   - This sets the maximum number of players allowed in a lobby
   - Number of rounds: This determines how many rounds a game runs for
   - Inactivity Timeout: This controls how many rounds of inactivity a player is allowed before they are removed from the lobby.

6. Moderation of the lobby by the host (e.g.: kick out players from the lobby)

   Status: Not implemented

This was another feature that we unfortunately did not implement. In the proposal, we identified this feature as having the lowest priority so we decided to not implement it due to the time constraints we faced. We felt this was a justified decision because it didn't affect any of the core functionality of the application.

4 Project Requirements

4.1 Client-side Requirements

For the client side implementations of this project, we used the open-source library known as React.js, for the creation of all our user interfaces. React uses the principles of single-page application (SPA), where the web application interacts with the user, by actively rewriting the current page instead of reloading another page from the server. This allows our application to run more like a tradition desktop app, alternatively to a web page. The front-end implementations was done mainly using HTML, CSS and JavaScript codings, along with other supportive libraries such as Bootstrap, CreateJS and so forth. Regarding browser testing, we vigorously tested our application on the Google Chrome browser, mainly using the Google Developer Tools for mobile and responsiveness testing. To conclude, our client sided requirements meets the criteria of having single page interface, the usage of HTML, CSS and JavaScript technologies and partially the criteria of browser testing.

4.2 Mobile Support Requirements

For our application, we followed a mobile friendly approach in designing our front-end. We prototyped our initial pages for desktop applications and then for the mobile layout, with the combining usages of media queries and viewport height/width settings for most of the pages. We used the Google Tools to replicate some mobile screens (iPhone 5-X, iPad, Galaxy and others), and tried loading it on a mobile device with limited success, due to having issues with the PaaS (Platform as a service) implementation with our nodejs server implementations on Heroku. We also had issues dealing with some different layout compatibility between Chrome and Firefox browsers. During an event of an unexpected disconnection from the network, such user would lose their game statistics and points and won't able to return to a game. However, the user will still be logged upon reloading, due to the cookies being effective. To summarize, our mobile support isn't at a completed stage.

4.3 Server Side Component Requirements

Our application is build using the MERN stack. All of our front-end is implemented with react. For our back-end we have an node js server that is build with the express framework and for our database

we decided to use mongoose which is a library for mongodb that manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB. Our react front-end doesn't include any business logic and it only emits messages using Socket.io to the server helping it perform the business logic, such as at the start of the game our front-end request our server for n question where n is the number of player in the game. Our server also uses Socket.io to communicate with the front-end, and it is in our server where we decide which questions to send which player and how to allocate the points, how to manage AFK timeouts, how to update the database and all the other necessary business logic needed for our game.

4.4 Multiple Users & Roles Requirements

Our game application supports multiple users and 2 types of roles during gameplay runtime. Our game will run multiple users, using different browsers with their respective private browsing mode. Our game can support up to 8 players, of which 7 are regular players and a single lobby creator player. A player is a user who can join a lobby and a lobby creator player is one who is a player and one who can make a lobby which includes set rules. A lobby creator player can make a lobby with 4 sets of rules, which are time per round, number of rounds, lobby size and away from keyboard(AFK) timeout in minutes. We implemented a very basic user authentication setup, which are stored in our MongoDb database, where a user would enter a username and password. It would initially check for the valid username then password afterwards, without letting the user know whichever was invalid. A user would need to sign up, prior to login in. To summarize, all users can choice to be either a player or lobby creator player, and all users have a user authentication to login, so our application meets the project requirements for multiple users and roles.

4.5 Persistence Requirements

For the persistence requirements of this project, we implemented a web cookie to deliver persistent capabilities. Each browser would have a single cookie, which lasts for 30 minutes, and includes 4 parameters of username, nickname inputted during game, an isauthenticated pointer and color data. Data is only stored locally during gaming runtime, and it will be updated to the database at the end of a successful gaming session. Thus the data during a runtime event will be lost, whenever users were to have a connection failure. Although during reconnection, they would still be logged in. This implies that previously saved data isn't lost, whenever the server side of the application is restarted or whenever users logs out then logs in again, due to the server reading off previously saved data directly from the database. Some updated or previous user activities can be seen in the profile page, such as number of games played, total points accumulated, number of wins and so forth. To summarize, our applications meets the project requirements of demonstrating persistence capabilities.

4.6 User Interaction Requirements

Being an interactive party game, we implemented many events where users are involved in a variety of user interactions. Examples of real-time user interactions can be seen during the waiting instances of the game lobbies, the turns of each player through the gaming process, the voting process of the game answers and so forth. When any user types joins a lobby waiting room, such user will have to wait for other users to join in before the lobby creator can start the game. In those moments, users will see other players join in a live environment, which is followed by the gaming processes. After all user submit their answer questions, all voting events occur one user at a time. Between those voting sessions, each users are dependent on the other user finishing up their voting turn, before having their own turn. Also at the end of the gaming process, all users will receive a results page which contains the final scores of all users. Thus to conclude, our application meets the project requirements of having user interactions.

5 Technologies Used

Our game was made using HTML, CSS, JavaScript, React, Socket.io,and CreateJs. The front-end was mostly done in ReactJs. The server side was implemented using Node.js. We combined the JavaScript with asynchronous communication via WebSockets to send and receive data from the server. From the suite of libraries from CreateJS, we used PreloadJS to load all our assets including questions, images, and sounds before the game begins. SoundJS was used for background music and we registered events handlers for certain events like clicks, mouse over, selected, etc. and played sounds for those. SoundJS gave us control over the sounds so we could change the volume on each stage. Bootstrap was used to make our pages responsive specifically for creating buttons. Cookies and local storage are used to store information such the player's nickname, player avatar colour etc. We used Express which is a web application framework for Node.js for our back-end. We are also using Socket.io from asynchronous communication between our server and our client. To store our questions, and user accounts, we used MongoDB. Different schemas were created for each new user when they signed up. During login authentication, we simply compared the username and password entered with a match in our database.

And finally form CreateJS we used EaselJS to create some of our graphics graphics on the canvas. In our game, the players do not have much control after answering the questions. Since the game is basically two players answering questions and then revealing both answers, we can use animations to display that. After joining a game, our application's functionality is accessible through a single page interface. So, that is why we used React.js to create a dynamic application where we can change data on the page without reloading it. Different components were creates such as player avatars, timer, answer boxes, questions,

forms for login and signup, etc. These different components were used in multiple different pages as they all had the same layout. Our game will have a lobby in which players can interact with each other. The interaction is you being able to vote on other player's answers. This is done through Socket.io, to implement communication between client and the server.

6 Deployment

In order to run our project, first it is needed to download the sources from the github and navigate to root folder and run "npm install" to install any dependencies. Then, using command "npm start", the game will run. Also, in order to connect to server, it is necessary to run server.js file using "node server.js" command.

`https://github.com/artrahim/QuickSplash`

7 Future Work

If we were given another semester to work on this project, first of all we would implement the feature to show the result for each questions and the number of votes each answer received. Also, we would enhance our animations which includes animations for assigning scores and for result stages, showing the changes in the top three players. Finally, we would implement the moderation of the lobby features by hosts which we included it is not in top priority.

8 Conclusions & Miscellaneous

To conclude, we are comfortable saying that the project was mostly successful. We implemented the majority of our functionality, including all the core functionality, and we learned a massive deal about many different web technologies and libraries.