

Final Year Project Report

Full Unit – Final Report

Guess My Doodle

Dilraaj Gill

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: DongGyun Han



Department of Computer Science
Royal Holloway, University of London

April 10, 2024

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged in references or comments in my code.

Word Count:

20,920

18,089– excluding appendix

16,658 – excluding appendix and bibliography

Student Name: Dilraaj Singh Gill

Date of Submission: 10/04/2023

Signature:

Dilraaj Singh Gill

Chapter 1: Introduction.....	4
1.1 Project Details & Motivations	4
1.2 Aims & Goals of the Project.....	6
1.3 Objectives – Milestones Summary	6
1.4 Planning, Timescale & Summary of Completed Work.....	7
1.5 Survey of Related Literature	11
1.6 Justifying Key Decisions	12
Chapter 2: Web Frameworks	13
2.1 States-Of-The-Art Web Development.....	13
2.2 Architectural Paradigms & Design Patterns	14
Chapter 3: Software Engineering	17
3.1 Methodology	17
3.2 Testing	17
3.3 UML Sequence Diagram	18
3.4 Use Cases	26
3.5 Algorithms.....	28
Chapter 4: End System Development	30
4.1 Installation Manual.....	30
4.2 Work Log.....	31
4.3 Documentation.....	39
Chapter 5: Assessment.....	40
5.1 Professional Considerations for this Project.....	40
5.2 Self-Evaluation.....	42
5.3 Potential Future Enhancements	43
Chapter 6: Bibliography.....	44
Chapter 7: Appendix	48
7.1 User Manual.....	48
7.2 Code Snippets	61

Chapter 1: Introduction

1.1 Project Details & Motivations

Websites are a key aspect of modern society. They have replaced parts of our day-to-day lives by removing obstacles such as distance, to connect us all together and create memories from all over the world. I aim to create an engaging website based upon the objectives of drawing a word that has been assigned to you by the server, from a collection of pre-defined word with the possibility of adding your own custom words as well. Whilst the user is drawing, the rest of the members in the lobby will have to guess the word being drawn, similar to the likes of skribbl.io, gartic.io or Pictionary.

When a user first connects to the website, they are greeted with a page that combines the login and registering components to either sign in if they already own an account or register their details. This is a requirement to proceed further into the website. To accomplish this task, it is possible through either manually inputting the fields required or signing in with Google. If opting to Sign in With Google, the user is then required to input their username afterwards. All users will begin with a profile picture selected for them by the server, however, will have the option of later changing it. If a user is registering an account, a form appears to demonstrate the validity of their inputs (e.g. unique email, unique username and meeting the password criteria). The form will not be processed without ensuring this criterion is met. Cookies are then stored on their respective browser when signing in, however this can be removed by signing out.

Upon signing in, they are then taken to the home page of the website. From the home page, it showcases the username and profile picture of the respective user signed in, containing a clickable dropdown consisting of profile customisation and signing out. In addition, the top of the container rests a wide variety of features – user's stored points, button to navigate to the store and refresh the home page. Refreshing the home page does not reload the entire page, but rather the list of lobbies available to join, allowing users to browse current public games that are available to join. If they wish to join in on the session, they can simply click on the respective button and will join the session. The lobby card displayed will showcase the number of players in the game, the profile picture and username of the host and the unique ID of the lobby. All public lobbies that have not ended are available to join. In addition, on the home page, they are also provided the option of creating their own game. which can be either private or public.

When creating your own lobby, you are presented with the list of players active in the session alongside settings to customise the gameplay, capable of changing the number of rounds, how many minutes a user will have to draw, visibility of the game session and custom words to be included. All settings and their modifications are updated in real-time to all present users. By default, all games are set to a private visibility. Furthermore, users can join the lobby through distributing the specific URL provided, however they must be signed in to join it. All buttons and interactions for the settings can only be done by the host. If someone else were to attempt to interact, it will not register or show any indication that the request has been completed. Once they are ready to start, the host can press "Play Game", which will turn the lobby into a Game session. From the list of players, the host can also manage who is able to join, by selecting a user and removing them from the game session. A minimum of 2 users are required to start.

After starting, the first user will be presented with a selection of generated words to draw from a pre-defined list with the option of also adding their own custom words from the settings page. Once they have selected their choice, the user will then draw it on the canvas, where it is then transmitted to all other devices in the sessions real-time while being drawn. On the other hand, if the drawing user does not decide upon a word within 10 seconds, a word is automatically selected for them. On the chance a user joins late to the game session, the drawing information, round quantity, and timer is

stored, therefore the late user will be instantly caught up. Whilst drawing, the other users will only be able to see the length of the word provided as a hint (for the user drawing, it will display the entire word). A timer is placed in the top-bar, to highlight to all users, the remaining time accordingly.

A chat box is provided to allow users to communicate between each other and guess the word. Communication functionality is restricted to only guessers within the same lobby. If the user guesses the word within the allocated time, both the drawer and the guesser will receive points. The drawer will receive the sum of the guesser's points in that turn divided by 2, placing an emphasis on good drawing quality. The rest of the lobby will be able to see who has guessed the word correctly, and who is still guessing. The next user will be able to draw either once everyone has guessed the word, or at the end of the allocated time. In between each user's turn, a message will be displayed over the canvas component to highlight the word that was being drawn, alongside each user's point summary (if they gained points).

To provide added benefits, the canvas will allow for different supported functionality, such as adjusting the line thickness or modifying the colour, to accurately represent the word being drawn. In addition, there will also be shortcuts and buttons to clear the canvas, an eraser tool and undo component that are provided to each user by default. These accelerators aim to simplify and better the user experience. The user can also opt to select the button instead by utilising the toolbar located below the canvas

At the end of each round (or if a player leaves early / kicked), the points are added onto their user profile. This is to ensure that progress is not lost if a player disconnects. Players can view the points of others on at the side of the canvas in the player-list, where it will showcase active players in the lobby and their respective points sum.

Finally, at the end of the game session, the top 3 users of the session will be displayed, showcasing who the winner is, alongside a message explaining where you have finished. To avoid repetition of creating a lobby again, the host will have the option of restarting the lobby, directing everyone who present back to the settings page, loaded with the previous settings to ensure smooth continuity.

From the points stored on a user's profile, they can then be spent in the item shop. This contains a wide variety of purchasable features, ranging from a "Fill Tool" to new profile pictures. The fill tool is an additional feature that can be bought to quickly fill areas on the canvas with a solid colour. Without purchasing the item, it is hidden from the list of available tools in the game session. In addition, by purchasing profile pictures, it allows the user to customise their profile and change according to their preference.

I was motivated to do this project as during secondary school and the COVID lockdown, this was a very enjoyable game to enjoy with friends and family, however a lack of certain features inhibited the overall experience for us.

For starters, although you could accumulate points, they served no functionality outside of the lobby, which felt like a missed opportunity to provide incentives to the users, as despite all the time I interacted with the website, the benefits were restricted to only the interaction and joy with friends. By not storing a user profile, it allowed for repetition in the experience, which provided to be an inconvenience.

When trying to join a public game, websites such as skribbl.io do not provide the ability to select which game session you want to join, instead it is automatically assigned to you, restricting the amount of choice with who we interact with and the amount of freedom the player has. The only opportunity the end-user gets to choose their game session, is if another user were to send them the lobby link. This led me to join lobbies on other websites that were inactive.

In addition, websites similar in nature to my project also lack capabilities such as making an account. For example, when interacting with skribbl.io, they require you to input your name and avatar every

time, which led to repetition in the interactivity of the website when trying to join or create a lobby, as I wished the information would save. My project will thus aim to provide solutions for these points.

1.2 Aims & Goals of the Project

The first aim of my project is to ensure real-time transmission of information related to lobbies and gameplay. This can be either communicating chat messages, canvas-related drawings, and lobby changes. This will allow my website to provide the best experience, as users will not suffer from slow communication when interacting with each other, providing the best gameplay possible.

The next aim of my project is to reduce the need of users having to enter their details repeatedly. This is because it is a flaw I highlighted from my experience with alternative websites such as Skribbl.io.

Furthermore, my website should also provide an incentive for the user to interact with it. The benefit of providing an incentive is that the user will retain interest with the website, repeatedly accessing it and using it to interact with friends and family. This is an aim of my project as it is a short-coming I had noticed of other alternatives and hindered the overall enjoyability of them.

For the gameplay of the website, it should aim to have only one person drawing at any given time, whilst everyone else guesses the word. Users will take turns drawing the word they have selected, and if a user guesses correctly, not only will they receive points, but so will the drawer. As mentioned in the first aim, these should be transmitted in real-time. At the end of the user's turn, to provide clarity to the others that were unable to guess the correct word, all devices will display the message on the screen, highlighting the word that was being drawn and how everyone performed. Information relevant to the lobby such as the users, messages in the chat, and drawings will remain private and limited to be viewed by only those in the session.

Whilst drawing, the items currently selected (e.g. drawing tool, colour, line thickness) should be made clear to the user, so that they understand what they have chosen and how their drawing should look like. This creates an easy experience, as rather than seeming confused by the system state, it is clearly visible for them.

In addition, the status of the game session should always be visible too. If the game is currently being modified from its settings, it should be made visible to all users in the lobby. Likewise, when the drawing user is selecting a word, this should be communicated to all devices, and upon choosing the word, the hints should appear to highlight the location of letters and spaces in the respective word.

Finally, the last aim of the project should be to allow the distribution of lobbies (or even joining game sessions). This will be achieved through a custom URL link, containing the unique ID of the lobby they wish to join as a parameter, enabling the back-end server to identify it. It should only be possible to join lobbies that have been created by clicking the "Create Lobby" button from the home page, therefore any random lobbyID that has not been instantiated through this way will display an "Invalid session" message.

1.3 Objectives – Milestones Summary

The first objective for my project is to use Express to create a back-end server. On this server, I can utilise HTTP routes and Socket.io [3] to transmit information. HTTP routes will be used to meet the aims of creating an account, signing in, ensuring authentication to prevent unauthorised users from proceeding further into the website and purchasing items. On the other hand, socket.io will be used to achieve real-time transmission where required in scenarios specified within the aim (lobby and gameplay specifically as they require quick transmission) [15].

To also avoid repetition, a key issue highlighted, I will utilise cookies to store the session on the browser, reducing the need for the end-user to repeatedly enter login details, allowing them to proceed straight into interacting with the game sessions. Signing in is achievable through either manually inputting details or using their Google account via OAuth 2.0, meaning only the information required is shared, meeting the aim of keeping the user data secure [2]. Manually inputted information will be stored by encryption through salting and hashing, possible through the Passport package on Node.

Furthermore, through Axios, the front-end will communicate with the back-end server through defined routes when trying to join a lobby, where the correct checks are in place to ensure the lobby can be identified and exists [9]. This will be achieved by forwarding this request to the Dispatcher design pattern, to check if a lobby correlating to this ID exists or not. All gameplay functionality by the socket will be handled by the GameDispatcher, to then direct the request if required to the necessary Game object and bring about a response. This also allows for quick and easy validation and authentication if the user is allowed to perform this action.

In addition, to meet the aims of providing an added incentive, I will create additional functionality through a store. All tallied points related to the gameplay are stored alongside the user's information, where it can then be spent in the store to unlock cool features, either exclusive drawing items (fill tool) or profile pictures. Once unlocked, the points will be deducted from their account, however the purchase will be blocked if they do not have the correct amount stored.

With regards to milestones in the gameplay, it should be possible for everyone to view the drawing, yet only one person is able to draw at any given time. Users will take turns to draw out the word they have chosen from the Server, whilst everyone else aims to guess it through the chatbox area provided. As mentioned earlier, through Socket.io this will be transmitted in real-time. Validations will be performed on the messages to ensure that the user drawing is not permitted to send messages to users in the lobby, avoiding scenarios of cheating.

Also, when the user is interacting with the canvas, the various selected items must be made evident. This will be achieved through styling affects, highlighting the colour and current tool selected. This is important in a system to show the current properties and the system status, otherwise it leads to further confusion and a poor experience. The necessity to highlight the system state wherever possible is further reinforced by displaying the word being drawn and the performance of all users in that turn.

When a user joins a game session, all users should see the same components on their screen, regardless of if one of them joined late or not. This is achieved through storing the "state" of the game session on the respective Game object. As a result, upon connection when joining the game, the Game class can initialise the new socket connection with the required information. On the front-end, once receiving the state of the game session, the required information mentioned earlier is received, catching up late users so all screens are synchronised. This includes current round information, drawings, points etc.

The points earned from interacting with lobbies must remain persistent. If a user were to be kicked or leave a game early, the points should be updated on the termination of their connection or at the end of the game. This is to avoid misleading users to say they earned points when this was not the case, highlighting inconsistency. Likewise, it also helps to avoid hosts of a lobby abusing their power and not allowing anyone else to gain points by kicking them out.

1.4 Planning, Timescale & Summary of Completed Work

Since creation of the project timeline (referring to the project plan), I have had to make some modifications to the timeline for term 1, which I will proceed to explain whilst evaluating my completed work. My plan to implement this project was to focus on functionality, as this is the most

important aspect of this website, to then prioritise design in the second term after completing all required functionality.

Starting from week 2, I was successfully able to implement a login and registering system, after setting up a MongoDB installation on my local machine (as a temporary solution). Through using Passport, I was able to use it to store the details of the user onto the local database, whilst storing session cookies on the client's browser [8]. One thing I have not yet implemented is oAuth 2.0. My reasoning for delaying its implementation is that it is not a necessity for my functionality. Although it hosts a range of security benefits, at the start of the project, it seemed more logical to focus on functionality, and setting up oAuth could prove to be a tedious task, delaying the rest of the project.

In week 3, I was able to successfully create my Canvas component, using mouse event triggers to draw onto it. Furthermore, colour selector and line thickness modifier components had also been implemented. I chose to neglect the eraser tool temporarily as this was also deemed added functionality. When creating the timeline, I failed to consider the necessity of components like an eraser. A temporary substitute for the eraser is to change the colour to white and draw over the line on the canvas.

For weeks 4, I had enabled drawings to be submitted over socket communication, whilst restricting usage to one drawer at a time, however elected to skip the fill tool, for the same reasoning as the eraser.

Week 5 was completed successfully, however when planning the timeline, I had failed to consider that to display the public lobbies on the home page, I needed to create functionality for the lobby, which was scheduled to occur on week 7-8, which hindered my ability to display the lobbies onto the home page.

Week 6 and 7, I was able to implement the intended functionality in its entirety, however, I implemented modifications to my original plan. Initially, I intended on having a Game class which would use Round classes for the interactions within that round. Each Game object would be stored in an array on the server component to communicate with and speak to the objects. This initial idea would result in my Server code becoming cluttered handling multiple functionalities and socket communications, hence I opted to create a new class called GameDispatcher. Rather than handling both functionality and communication, this will allow me to dispatch functionality requests to the GameDispatcher (which will now store an array of all the Game objects rather than the server), allowing the Server to focus on socket communication. The benefit of this is that the code becomes more organised and structured as it follows design patterns, clearly demonstrating the transportation from the socket to the back-end Game objects. A consequence of this is that it disrupted the timeline I was originally intending to follow, resulting in not all the week 8 functionality to be completed.

For week 8, I have successfully implemented the timer according to the lobby settings and display on the front-end as to whose turn it is to draw. At the end of the timer, the turn will then proceed to the next user. Functionality I was not able to complete due to timing constraints were the distribution of points and selecting a word, as I elected to focus on the interim report and presentation, as opposed to delaying these two critical items. Upon completion of it, I will then revisit this content. Furthermore, as I had skipped certain content as they were not a necessity to basic functionality, I intend on implementing them during the term break.

Over the term break (between term 1 and term 2), I successfully implemented methods for managing updating points for users between each round and at the end of the game. Alongside this, I also created a LobbyCard component using TDD, to display the information of each public lobby and its respective ID. In addition, as highlighted earlier, functionality such as undo, fill and clear canvas were not incorporated initially due to being of lesser priority, therefore after completing the basic implementations of the canvas, over the term break, functionality was created for these tools. Furthermore, I also implemented the HTML skeleton of the store and the route of the web page. The

initial item added to the store was the Fill Tool, as I wanted this to be an exclusive reward for those who earned the necessary points. This allowed me to catch up to the original timeline intended.

Within term 2, a large emphasis was placed on improving the user experience. To do this, it required learning how to utilise CSS and Bootstrap within my website to ensure a fluid and responsive web design. In comparison to term 1, where the target was to ensure functionality for the components, this term required me to plan and design how I wanted the website to be structured. From the original project plan, adjustments had to be made. The following section will discuss the progress made weekly and any adjustments from the plan.

In week 2, after reading the applications of colour theory to entice users [5], I began reformatting my current React components, starting from the login page. I began by utilising React-Bootstrap [21] to modify the login page and register page. Initially, I wanted to have distinct routes for logging in and registering, however, for ease of access, I combined the two into one menu. This results in the user easily able to change whether they want to sign in or register with a new account. Furthermore, I took advantage of bootstrap icons to display a Google logo for those wishing to take advantage of OAuth 2.0. This allowed me to complete week 2 successfully according to the project plan.

Within week 3, I refactored the home page with a temporary design. This was achieved through Bootstrap buttons and designing “Lobby Cards” to display each respective lobby. In addition, I then went ahead of schedule and began configuring the lobby interactions too. To clean up the design of the game, I elected to use a popover, whereby pressing the respective button will load either the colour picker or the line thickness tool. This ensured a clean aesthetic was created, which is vital towards having a good user experience, as by cluttering the layout, it can create a chaotic outlook [5].

From week 4, deviations had to be made from the original timeline. As per the project plan, it was intended to begin styling the “public lobbies page” however this was a component of the Home page that had already been refactored. Furthermore, user avatars had not been implemented, making it impossible to “Display user avatar alongside lobby”. The implementation of user avatars was delayed significantly due to licensing issues. When trying to select images for my background, avatar, etc, it was important for me to follow ethical standards. This meant ensuring the correct licenses were in place before I could utilise them on my website. To proceed working, I began refactoring the main component of the website, the Lobby. This was achieved by fixing the player-list component to become scrollable, turning the canvas buttons into a toolbar that rests below and lastly using Bootstrap columns to align the list of players to the left, the canvas and toolbar in the middle, alongside the chatbox to the right-hand side. Further adjustments were also made to visibly demonstrate the system state, by displaying a timer regarding how long is left, the current round being played, and icons representing the host and drawing user respectively.

Similarly to the previous week, the timeline for week 5 was also unable to be followed directly, due to missing features because of the professional issues encountered. The original project plan timeline described week 5 to mainly consist of modifying the player-card component to display the user’s avatar beside the gameplay, which could not be completed as the current database schema did not incorporate an avatar field. As a result, I continued from week 4 to focus on the usability of the gameplay. This entailed creating a function for copying the URL of lobbies directly to the clipboard, a background consistent on all pages, CSS styling for the chatbox, a flashing red countdown on the timer when approaching below 10 seconds, a hints component to display the length (and spaces) of the word above the canvas and completed TDD steps to implement a StoreCard component. From the original plan, barring the instructions that required an avatar, the rest were successfully implemented.

For week 6, as I was currently a week ahead of schedule, I was able to capitalise upon this time and correct the plan by modifying the database schema to include a profile picture and assign a default picture to all new users, after resolving the professional issue at hand. This allowed me to then revisit week 4 and 5 and implement the functionalities for displaying the profile picture on the home

page, within the player-list, and the host's avatar on the lobby card showcasing all available games that can be joined. In addition to this, I also implemented new profile pictures in the store that could be purchased and instantly assigned to your user's avatar (with no capability of selecting from your list of purchased icons).

Week 7 built upon the progress made in the previous week. In order to store the list of purchased avatars, the schema was changed again to store an array of all available icons. Through the TDD process, a new component for a modal was created to showcase all icons the user owns and the current selected one, allowing for customisability and selection for the end user.

Week 8 was originally meant to allow for a Store to be created, however this was done over the term break, therefore during this week, it was primarily for testing (which week 7 was intended to be), where I used the website to understand any missing features and weaknesses to the gameplay. From this, I highlighted there was no sign out button and the home page became slightly cluttered with all the buttons present. Therefore, the home page was refactored once again, moving the profile selector and a new sign out button to a dropdown from the user's profile.

After this point, all the components from the project plan had been incorporated into the website. This gave me time to reflect upon the project as a whole and identify areas that could be improved to create an overall more enjoyable and fluid experience.

From week 9, I began to delve further in my design to improve the usability, a key concept for the website. This was achieved through displaying the point sum and word between changing users, a podium at the end of the game highlighting the top 3 players and where the signed in user finished, a restyled toolbar actively highlighting the selected tools, colours etc, improved responsiveness of the gameplay by scaling the website to the new size of the window, and registration validation for password criteria, valid usernames and displaying this status for the end-user. Another key improvement made during week 9 was the "Play Again" button. Related websites incorporated lots of repetition to the end-user, something my project aims to reduce. By allowing a new play again button, I can improve the gameplay for all users in the current session.

Further changes were made in Week 10, where I incorporated the ability of kicking players out of a lobby (if you were the host) and update their points. In addition, I then used this opportunity to ensure my unit tests were still behaving correctly, especially with the new Bootstrap components being utilised over the standard HTML, resulting in different behaviours for the front-end tests. To allow for easier testing, I removed all validation and authentication from these front-end components and refactored them into individual and reusable components. Furthermore, I implemented key binds, allowing for quick and easy changing of functionality such as brush, fill, eraser, and undo. This allows for simple accessibility and usability for users. Also, when joining a new game, my code is now more resilient to edge cases, being able to handle a user re-joining from another device, a user leaving mid-game etc.

Overall, I was able to follow the project plan timeline with correctness and ensured that the functionality and steps being made flowed naturally. Barring the initial mistake in the timeline where it instructed to display the user's avatar before creating support for it, the timeline outlined in term 2 was followed to completion.

1.5 Survey of Related Literature

1.5.1 Skribbl.io

The first related concept to my project is Skribbl.io. This approach involves an experience that does not require the user to register an account, resulting in information such as name and avatar to be inputted every time they wish to interact with the website; an issue I had highlighted with this system and aim to correct. My solution is to implement user accounts and session storing cookies; therefore, the user will only have to sign in once to have their information saved.

In terms of the interaction with the website, the concepts are similar to my project. Skribbl.io presents a unique and colourful interface. Despite having a theme of blue, the project is ripe with vibrant colours through its multi-coloured logo and avatar customisations. Furthermore, there are minor animations that are not too noticeable; helping to provide that sketching or drawing sort of feeling. It clearly demonstrates who is drawing at any given moment and each user's points. Players are listed by who has the most points. At the end of the game session, it misses functionality to clearly demonstrate the winner (e.g. like a leader board demonstrating who has won).

With regards to customisability, the website is full of unique features. It allows the host to define the maximum number of players, spoken language, drawing time, number of rounds, word mode, word count (number of options of words to be presented), hints and lastly, custom words. The wide range of options allow the user to customise their experience to fit their needs.

As previously discussed, issues with this website arises from missing functionality and repetition when joining lobbies.

1.5.2 Gartic.io

The next related website to my project is Gartic.io. This website provides a similar system to mine, as it allows for people to register an account or also sign in with their Google account, however registering an account is not a compulsory action as users can also proceed as a Guest. Although this feature is useful for one-time users, it would go against the implementation of my project, as there is no way of tracking a guest's tallied points.

When accessing the website, you are allowed to browse rooms that are currently active, displaying the number of people, language being used to communicate etc. There even exists functionality for filtering lobbies according to a certain criterion.

Once within a room, the functionality remains consistent with that of Skribbl.io. Limitations of this website are that users have a separate chat to place their guesses in and another chat to communicate with one another (that is restricted to only those signed in). This is a difference compared to my project, as rather than having two clashing chat boxes, I intend on having one to provide multi-purpose. Another aspect where my project differs from the Gartic.io implementation is that it will have a store, allowing further customisation and user freedom, to earn and unlock cool features.

1.5.3 Drawasaurus

Drawasaurus is another similar concept. The user interface is incredibly simple and effective. It accurately employs colour theory through having a main colour (purple) and an accent colour of green to be used in smaller details such as in the timer or displaying scores. Compared to the previous concepts, it lacks transitions, which would enable a smoother and enjoyable experience.

As for the interactions with the website, it employs similar behaviour as mine, providing tools such as brush, eraser, and fill. The interactivity seems rough in certain scenarios, and when drawing, it seems abrupt and not as smooth compared to the likes of my project, skribbl.io and Gartic.io. There is no selection for avatar cards and customisation. There exists one chat box, to provide the ability to guess and communicate with the rest of the room.

In terms of customisability, it also allows custom settings when creating a room, similar to my project, allowing the possibility of custom words, defining the number of rounds per game, and drawing time (in seconds).

1.6 Justifying Key Decisions

The first key decision made was to not **entirely** support mobile devices. When navigating on the website, aspects like signing in, customising your avatar, and exploring the store have been designed to support every single device and screen resolution. The issue with interacting from a device with a smaller screen size, is when playing in the lobby, there are a lot of necessary components that would have to be adjusted in order to accommodate for the smaller size. For instance, it would need to display the canvas, chat box, player list and when drawing, the tools would also need to be visible for selection as well. Likewise, a custom keyboard is required to be created to avoid scenarios whereby the native mobile keyboard will push the display up, forcing it to be hidden. To display all these components on a smaller screen creates a cluttered outlook, especially on a smaller screen. The primary reason for excluding entire mobile device support is that this screen type is not the primary targeted user. The website is intended and optimised to be playable on screens of larger size to enjoy the game as intended. With more time and resources, it may be possible to design and re-structure the screen to support mobiles, however this does not remain a priority, as it is more enjoyable and targeted for those with larger screens to interact with.

Another key decision was to not implement a text tool when interacting with the canvas. Whilst originally under consideration and natively supported by the canvas, when researching how to implement this functionality into my project, it led me to question whether this was taking away from the challenge of trying to draw the word and the added difficulty of having to do this by a paint brush. By implementing a text tool, it also did not fit the artistic style of how I wanted the drawings to look, and the overall user experience would be over-complicated, hence I decided to not proceed with implementing this feature. The contrasting styles of text alongside brush strokes would heavily contrast against each other and led me to believe it would become more a hinderance and a gimmick than a useful tool.

Similarly to the text tool, I did not implement a functionality for drawing shapes by a button. Despite that the text and shapes tool were not explicitly involved or mentioned in the project plan, it was an idea that I did consider implementing as an additional feature if ahead of schedule and the timing permits it. The reason for not creating this was due to the fact it was not deemed a priority, alongside the added complexity required within the limited time of the project. To design a button that would allow for dynamically sized shapes, this would require two canvases to be present on the web page. The first canvas stores “temporary drawings” that would be populated when the drawing is changing dynamically. An example of this is when the user presses their mouse down to create a shape, they must then drag their cursor out to define the size, with the shape adjusting accordingly. Whilst the mouse is pressed, the shape is dynamic, hence stored on the first canvas. The second canvas stores all concrete drawings that were on the dynamic canvas, but once lifting the mouse up, are transported to the second permanent canvas. The complexity to integrate two canvases together and support the shape functionality would be increasingly difficult to accomplish in the limited time available, and as this was not mentioned in the timeline nor a requirement for my concept, I therefore had chosen to ensure the current features and web design were implemented to a high standard instead.

Finally, when deciding to create profile pictures, I opted against allowing users to upload their own images that could be displayed on their profile. Although the concept would allow for users to feel more connected with their character through the added customisation, malicious users could take advantage of this component to display indecent images. The only way to prevent users from manipulating this system in it’s current form was to avoid implementing this functionality entirely.

Chapter 2: Web Frameworks

2.1 States-Of-The-Art Web Development

To create my project, I am utilising React and Express.js frameworks for the back end and front-end respectively. In addition, I am also using MongoDB to store details for login & registering alongside the user's points, which is earned through using the website.

I chose to use Express.js due to the simplicity the framework provides to Node.js. It provides key functionalities to my project, such as middleware, to handle server requests in the form of HTTP requests, such as logging in or registering. Furthermore, Express also provides excellent performance, which makes it suitable for the project I am creating as it will be involved in lots of communication between various clients. Another benefit of using Express, is that it has lots of support, providing various packages such as Passport, Mongoose, Axios and Socket.io, which are all detrimental to my project. Passport will allow me to authenticate users, either by storing session-cookies or through their login details. In addition, the use of salting and hashing passwords will provide a layer of added security for users of the website. Mongoose will allow me to interact with my MongoDB NoSQL database efficiently, whilst also interacting with passport to assign sessions to users [7].

Socket.io is vital to ensure my project can communicate with other client sessions in the same lobby. By using sockets, it will be used within the canvas and drawing related features to provide real-time communication between all devices. This is more efficient than using HTTP requests as socket.io allows for bidirectional communication, with low latency connections, which is detrimental for my interactive website.

Also, I chose to use React due to its component-based architecture [4]. This allows me to easily organise my front-end into smaller structured components, providing simplicity and easily understand the transfer of data as the user interacts with the website. The use of components makes it easy to develop re-usable elements for my website. Functionality such as React states are useful to monitor user interaction or update values that are displayed on the website with ease. Furthermore, React Router will be used to handle all routing for the website, which is useful when trying to navigate through the different areas of the website and for using route parameters for storing information such as lobbyID etc.

Through the react-router, I am controlling my navigation with buttons, re-directing users as required. An example of this is the store button on the home page, navigating users to the /store route when clicked. Within my gameplay functionality, when a user is kicked, they are forcibly navigated away from the website. This lets me store a state when navigating away, and checking for the presence of this state can allow me to bring about a response e.g. a modal.

Finally, I will be using MongoDB to store information as required on the NoSQL database structure [12]. I selected MongoDB due to its high-performance capabilities and the way it stores data similar to a JavaScript object, allowing it easy to incorporate within my project. In addition, querying is kept simple and interacts easily with my passport package to access and create user accounts.

All these frameworks combined form a MERN stack, which is the perfect implementation for my project [1]. Advantages of utilising the MERN stack is that they all use one language (JavaScript) for all ends of the website. As a result, it will reduce the overall complexity of the development process. Further advantages of the MERN stack will be discussed under architectural paradigms and design patterns.

To style my front-end components, I will be making use of React-Bootstrap and custom CSS. React-Bootstrap provides a set of UI components that will allow me to create a responsive design in my

project, adapting the website to various screen sizes and devices. This allows my website to operate on smaller devices such as tablets. In addition, React Bootstrap integrates directly into my React front-end framework without any issues or added complexity, as the Bootstrap items are treated as React components that are being rendered.

By using custom CSS, I can modify the pre-built components from React-Bootstrap to fit the aesthetic of my project. This is vital to allow my design to become more appealing, as the general Bootstrap components are rather basic and barebones.

2.2 Architectural Paradigms & Design Patterns

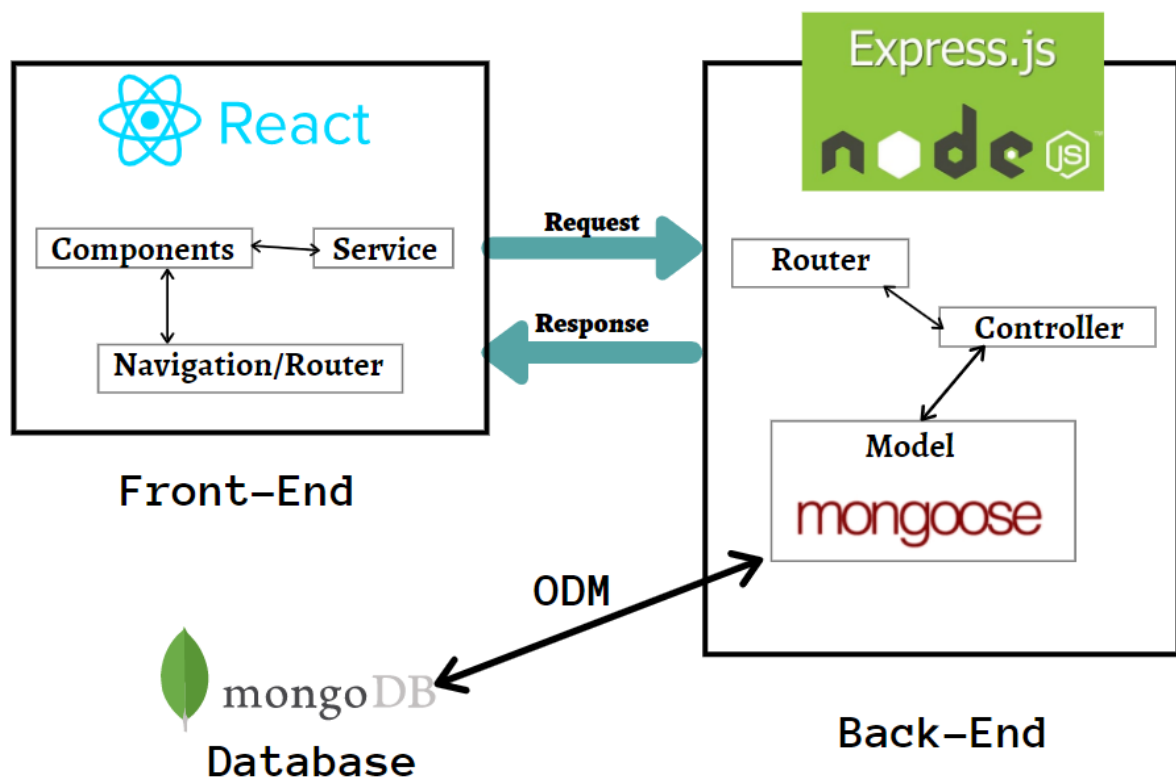


Figure 1. Diagram showing interaction between MERN Stack [17]

The users will interact with the web server hosted on a port (e.g. localhost:3000). This port will allow them to view the website through client-side rendering. This means that the browser will oversee rendering of the website and React components for the user. The benefits of this are fast response times as they do not have to wait for the server to respond, allowing for reduced server load. This is important as the back-end server will be detrimental to functionality of the system, hence any reduction in load that is possible should be done.

The website will then be used to communicate with the server, which is ran on a different port (e.g. localhost:3001). The back-end server (using Express) will handle requests such as logging in or socket communication, and take the necessary response required. Furthermore, a database server will be used, that remains only in connection with the back-end server for security purposes, to avoid users trying to gain unauthorised access. For the client to log in, the request is sent from the webserver to the back-end server. From here, the request is processed, and the necessary information is collected from the database server to approach a final decision.

For architectural paradigms, I am using RESTful Architecture to create my web-based back-end server. The benefit of using this architectural paradigm is that it will allow me to create a client-

server structure to the website. Furthermore, it allows me to use methods to control responses to HTTP requests such as GET, POST, PUT, DELETE etc. To control interaction, this is handled by requests to the specific route, with the ability to transmit JSON format data.

Alongside RESTful Architecture, I am also using event-driven architecture. This is used for socket communication, as they work according to event handlers. Event-driven architecture is used for real-time features, which is applicable to the project as to handle canvas interactions, it must be provided in real-time. Events can be triggered by multiple interactions, such as chat box functionality, or drawing on the canvas. The benefit of this architecture is that it will allow for quick processing of the event and the transmission of data to the server, providing rapid responsiveness. This is achieved through defining event handlers according to the socket message, specifying what to do with the submitted data.

Furthermore, through using React, I am employing component-based architecture. This is because React breaks down pages into components, which can be treated as HTML elements, to construct the full page. Through this architecture, it allows for re-usability, as components can easily be reproduced to make more HTML elements on the page. In addition, information can be transmitted between components to adjust the respective elements via conditional rendering, which can be used to display the information onto the page as intended or for calculations.

Design patterns are vital in web development, providing a template of structure to re-occurring problems [13]. The benefit of utilising design patterns is that they will typically bring performance and execution benefits. Within my work, I have actively considered the implementation of design patterns in order to solve communication issues and develop structure and organisation, specifically in my back-end server.

From usage of the MERN stack, it employs MVC architecture (Model-View-Controller) [16]. The model of this architecture is the MongoDB database, which will be responsible managing the data of the website. For the back end to interact with the database, I will deploy Mongoose to communicate and store information. The view component of this architecture is determined by React. This will provide a platform for the user interaction, by displaying the website on the browser and forwarding requests and information to the controller. Lastly, the back end Express server is the controller of the MVC architecture. It will receive requests from the front-end React view component. After processing it, the server will then bring about the required response, which may involve interacting with the database (model) to update or retrieve information.

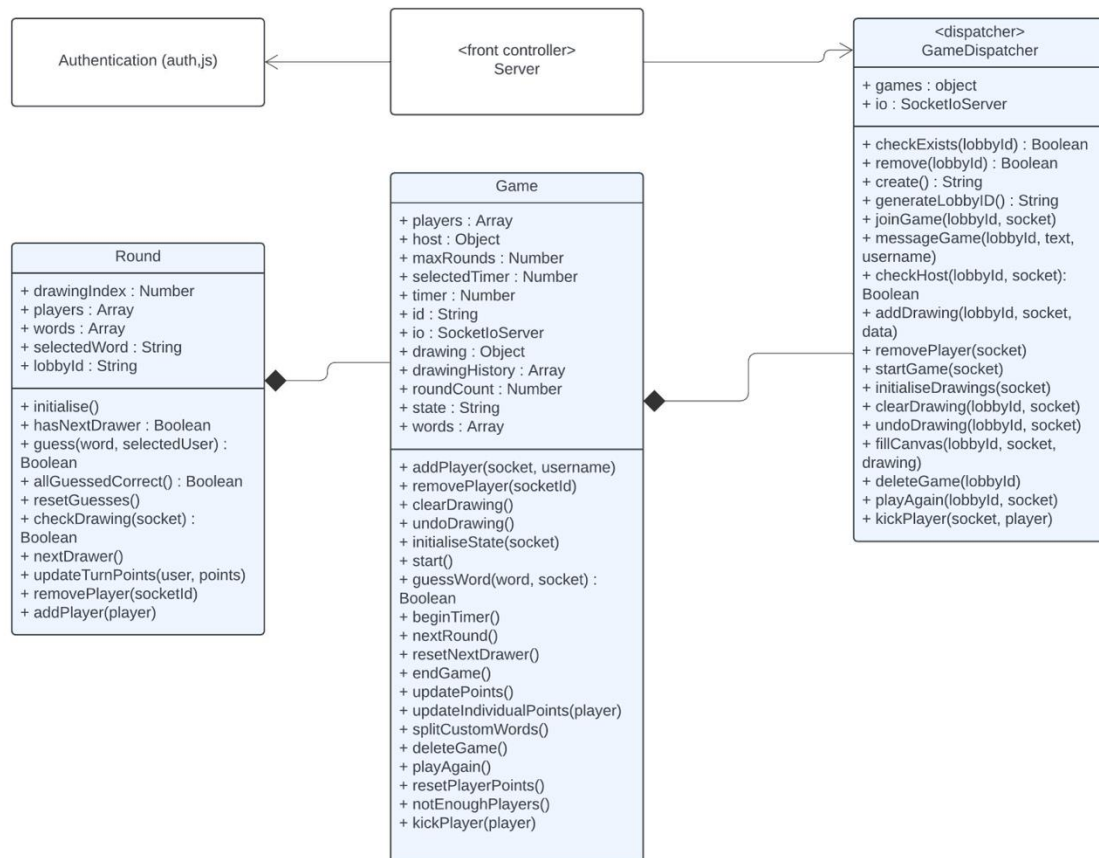


Figure 2. (Simplified) Back-end UML Class Diagram

A design pattern I will be using in my project is a front controller pattern. When attempting to join a lobby, the server will first authenticate you before dispatching the request to a GameDispatcher object. The GameDispatcher class stores all the games that are currently running. When any game related socket transmission has occurred, the information is passed onto the GameDispatcher to make the necessary updates to the appropriate Game object. As a result, the server is the front-controller, as it will handle all requests made, and then the GameDispatcher is a dispatcher design pattern, as the server will use this object to dispatch the request to the appropriate Game.

In addition, on the back-end server, I have used a Decorator design pattern, to add functionality to previous objects without altering its behaviour. This is shown through adding middleware to HTTP requests, such as my auth.js file, or a bodyparser. The use of auth.js is to process the request before it is sent to the response method. This allowed me to create a separate JavaScript file to process authentication for accessing the website or entering registration details. Furthermore, through using `express.urlencoded()`, it will allow me to handle information from the incoming HTTP data sent by forms. In addition, by using `express.json()`, it will allow the server to access the body of incoming JSON data.

Chapter 3: Software Engineering

3.1 Methodology

To create my project, I will use TDD (test-driven development) to design the components of my code. TDD aims to create tests before the development of the code. This allows me to outline key functionality and create functions that meets the requirements set out by the tests.

The TDD process begins with creating a failed test. This test case should define the expected functionality of what is required to be implemented. It will fail the test as no code has been created to pass it.

In the next step of the TDD process, I will begin to create code to pass the failed test. This is aimed to be kept incredibly simple and possibly even faked, just to return the expected outcome of the function. Upon passing the test, the next process is to refactor what is written. This will involve improving the code to either develop structure, remove redundant lines of code, or improve efficiency / readability.

By using TDD, the code I create will have a higher quality, due to it being tested against pre-defined test cases, and therefore less likely to suffer from bugs. Furthermore, through refactoring, my code will be simple to understand and well-maintained, whilst simultaneously ensuring all requirements in terms of functionality are achieved.

Furthermore, I am using Git as my version control system. Git allows me to create feature and release branches to focus on features independently of other aspects of the project, which can then be merged back into the main branch to complete the overall functionality. This allows me to focus on one feature at a time, alongside creating designated testing branches, so features can be tested and coded simultaneously, despite not affecting one another. If one branch requires aspects of a different feature to be completed, the two branches can be merged to continue development. Furthermore, a release branch will allow me to create a new version of the project where it is at a stable point in development. I can then carry out extensive testing on the release branch to ensure stability and that high quality has been achieved.

In addition, Git is particularly useful in ensuring any actions to modify the code can also be undone too. It allows for mistakes written in the code or files to be rectified and can be reverted to previous versions to solve any problems. The commit-based system allows for me to select what files I wish to apply in a commit and provide a message to explain changes.

3.2 Testing

I will be implementing unit tests for the back-end express server and the majority of front-end components, where appropriate. To structure my tests, I will begin by creating the necessary file. If creating a back-end test, I will create .test.js file, whereas for my front-end test, I will create a .test.jsx file. At the top of each file, I will begin by importing necessary libraries, before describing tests that must be made to check for every aspect of functionality.

To create the back-end unit tests, I will be using the “SuperTest” library [10]. SuperTest will allow me test vital functionality for my server such as registering, logging in, authentication checks, etc, and will play a key role in completing my TDD tests. This will work by passing the server application object as a parameter, allowing me to check for the different HTTP requests / socket messages being communicated through event triggers between client-server communication. A requirement for this is to export my back-end server to allow SuperTest to operate. Furthermore, for my back-end tests,

the test file is treated like a client to the server, sending requests to the back end and using SuperTest to ensure the server is receiving the messages and performing the necessary steps for a response.

To create my front-end unit tests, I will be utilising React's testing library and Jest [11]. Through React's testing library, I can manually trigger events (fireEvent) and render components. After rendering a component, I can then use the screen object from the testing-library to hook onto specific HTML objects through their role (screen.getByRole) or text (screen.getByText). This will let me test if the rendering of my components works as intended, but also ensure the functionality is executing correctly. The "jest" library will allow to mock other libraries being utilised for communication such as Axios, or sockets [6]. Through mocking, I can replace functions with stub code, to ensure methods are being called when required and how they interact with expected returned values.

In addition, for my automated end-to-end tests for my project, I will utilise Cypress to simulate interactions with my front-end website to test how the back end will respond. Furthermore, by using Cypress, I can also ensure that the response is made clear to the front-end users. With Cypress, it can help to simulate real actions [14] such as filling out forms, entering text to ensure the correct changes (either front-end or back-end response) is made.

To create my Cypress end-to-end tests, I defined "specs" that will store relevant tests such as home, lobby etc. The specs will be of the file type "file.cy.js". For my project, when creating these tests, I ran into an issue. When testing the game functionality, my project requires two user actors, however this is not possible through Cypress. This remained a limitation of Cypress when trying to test the entire functionality of my project. As a result, all the other functionalities could be tested through automated test cases, however the majority of game features itself had to be manually tested. As a result, the greater part of the end-to-end testing remained automated, but gameplay functionality had to be manually tested upon every modification / update to the system.

3.3 UML Sequence Diagram

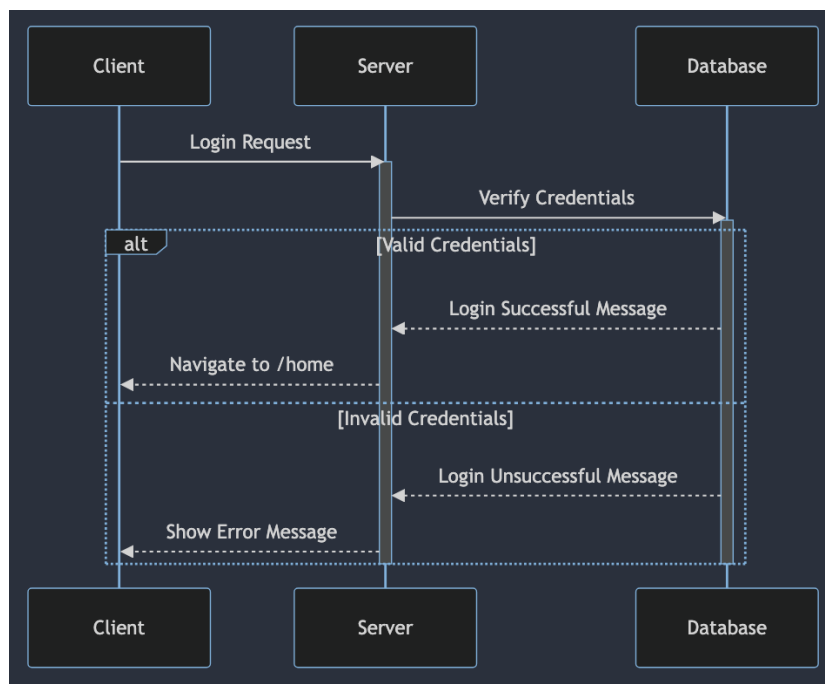


Figure 3. Sequence For Logging In

The first sequence created is signing in. To sign in, the user sends their login request to the server, which forwards it to the database. If the credentials are valid, the database will return a successful

status to the server, which will prompt it to redirect the user to '/home'. On the other hand, if the credentials are invalid, the user is displayed an error message.

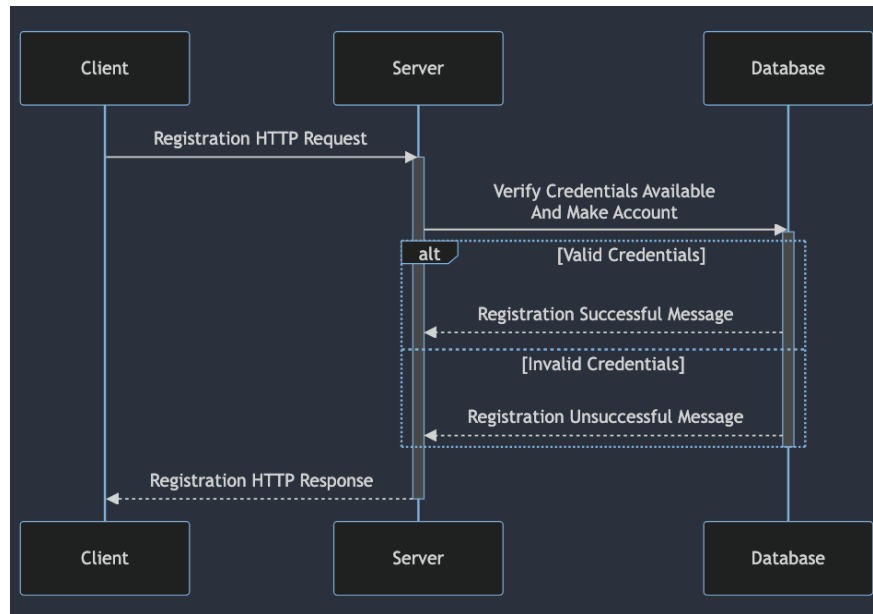


Figure 4. Sequence For Registration

The next sequence is registration. For this sequence, the user will send their registration request, where the back-end server will communicate to the MongoDB database to ensure that the credentials are valid. No matter if they are valid or invalid, the response will be emitted back to the client, so they are aware of the status of their registration.

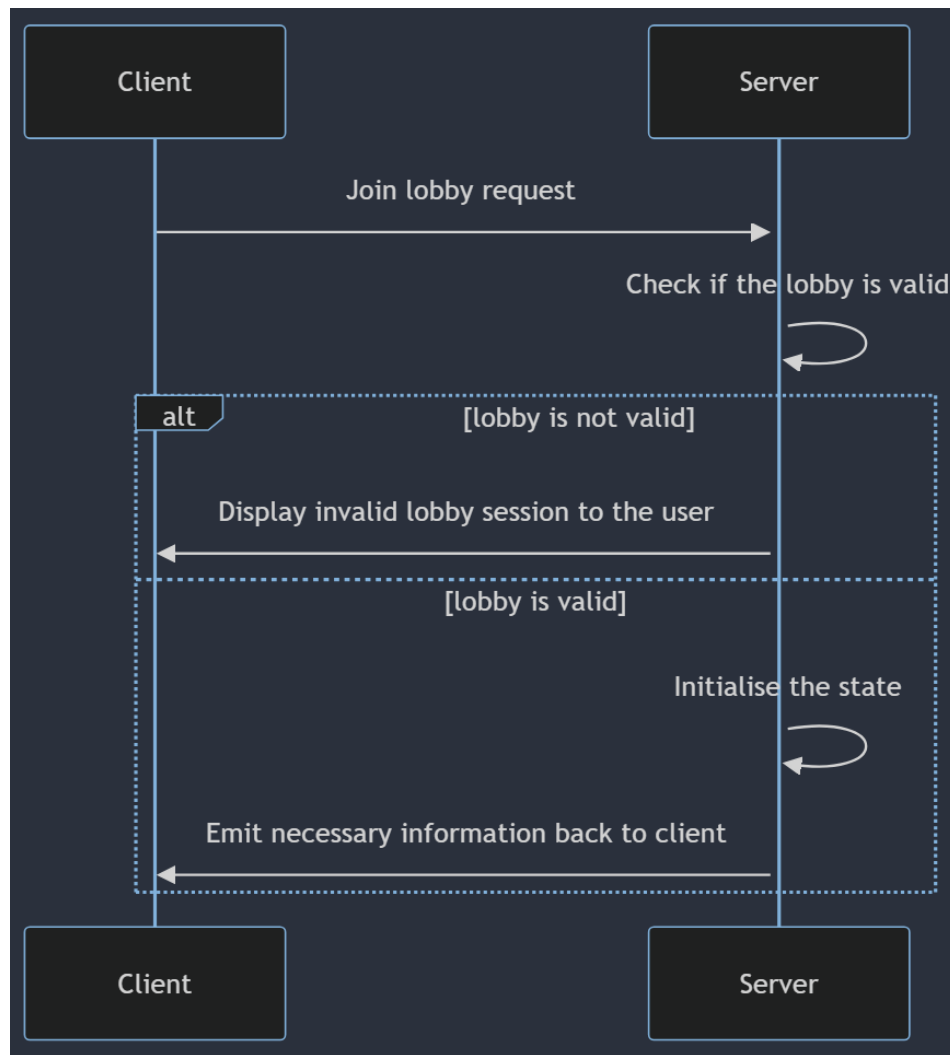


Figure 5. Sequence for Joining Lobby

The sequence in Figure 5 is when initially joining a lobby. A request is made to the server from the client highlighting the lobby they wish to join. The server will then authenticate this request to check if the lobby is valid. The validity of it depends on if the lobby exists or not. If invalid, a message is presented to the client, whereas if the lobby is valid, the state is initialised, and the necessary components and information is obtained and sent to the client.

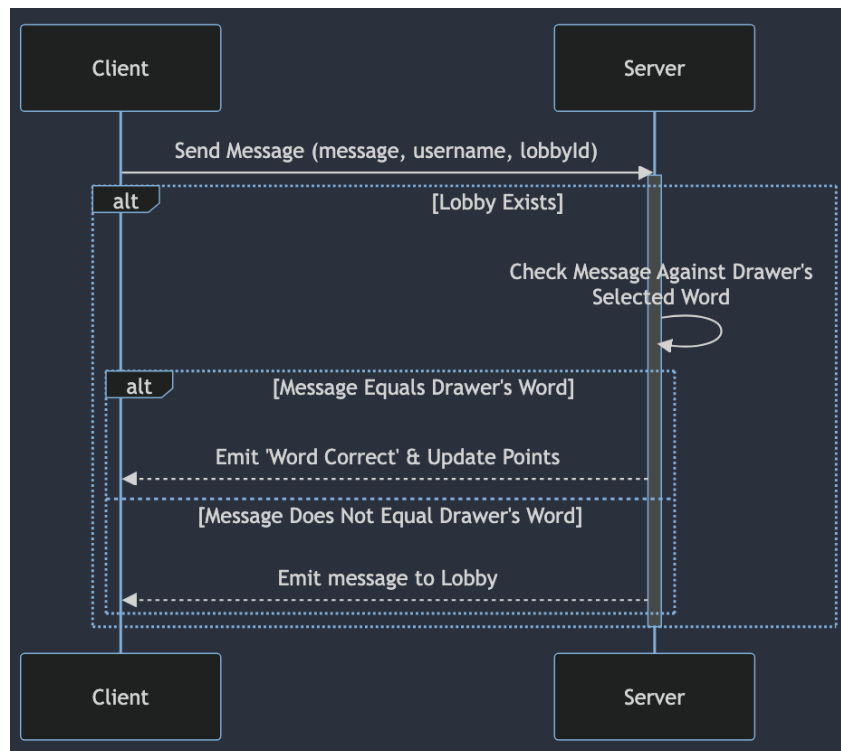


Figure 6. Sequence For Messaging / Guessing

This sequence above depicts how messaging and communication is handled by the back-end server. Upon receiving a message, it will check if it is equal to the selected word being drawn. This is done to prevent other users from viewing the correct word. If the word is equal to the drawer's word, then the user will receive a correct word message and update the points accordingly. On the other hand, if it does not equal the word, then it is emitted for everyone to see, allowing the chatbox to be used as a communication feature.

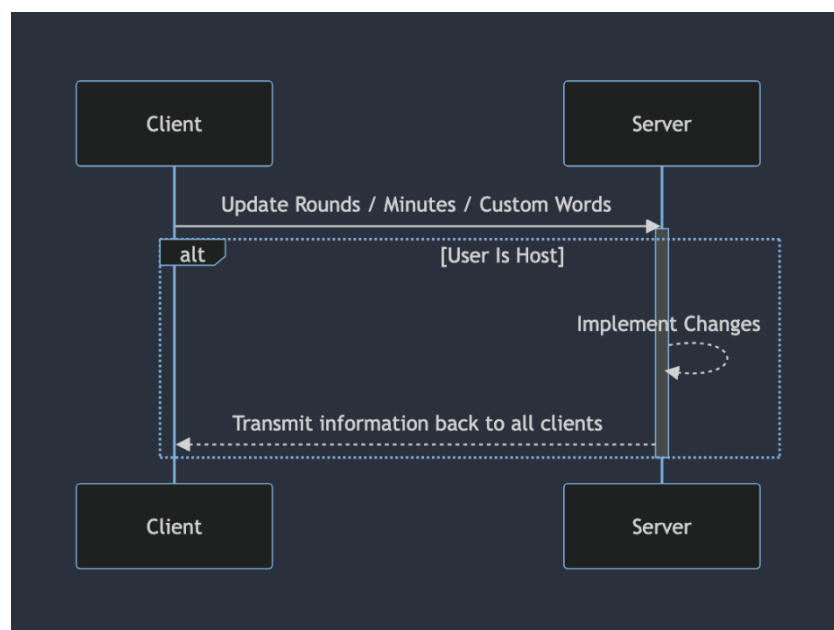


Figure 7. Sequence For Customising Settings

Figure 7 shows the sequence for interacting with the settings of the lobby. This feature is restricted to only the host of the lobby. When a user is trying to update the values, it is communicated to the back-end server, where it will perform the correct checks to ensure that only if the user has host privileges, will the values be updated. If true, it is transmitted back to show the new values to all clients in the lobby.

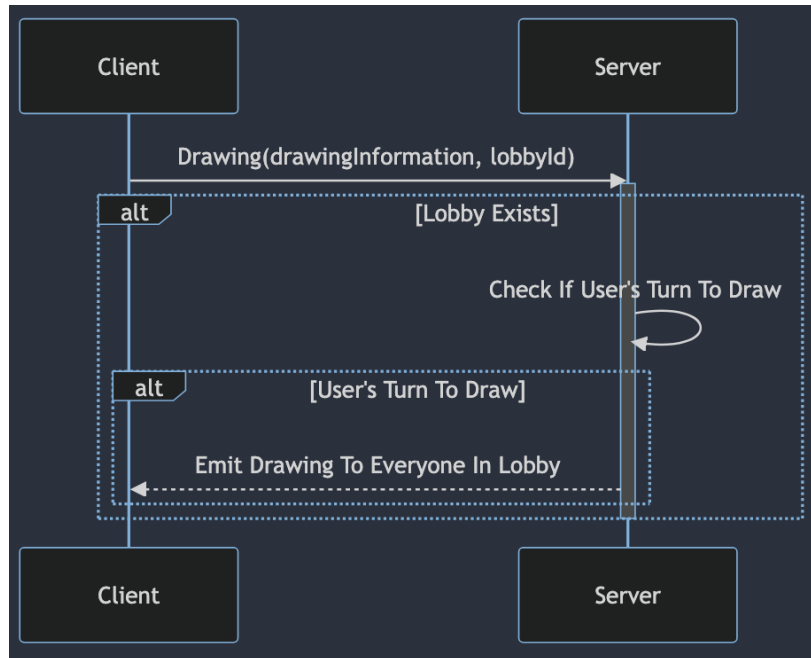


Figure 8. Sequence For Validating Drawing

The next sequence demonstrates the process of trying to draw on the canvas. Everyone can interact with the canvas; however, processing of that information is performed by the back-end server. Upon receiving information that someone is trying to draw, it will perform a check to see if the lobby exists, to avoid any error. If true, then it will check if the user is meant to be drawing; if that returns true as well, then the drawing is sent to everyone in the lobby.

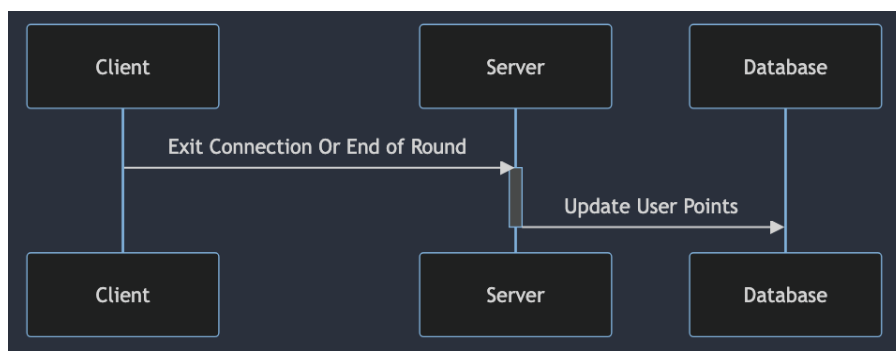


Figure 9. Sequence For Saving Points

Figure 9 is an important sequence for my project, which is updating points. This is performed at the end of each round, or if the user exits before then. Upon termination of the connection, the server will receive notice that the client has left, causing it to process the points they may have earned. To do this, it will interact with the database to adjust the user's points accordingly.

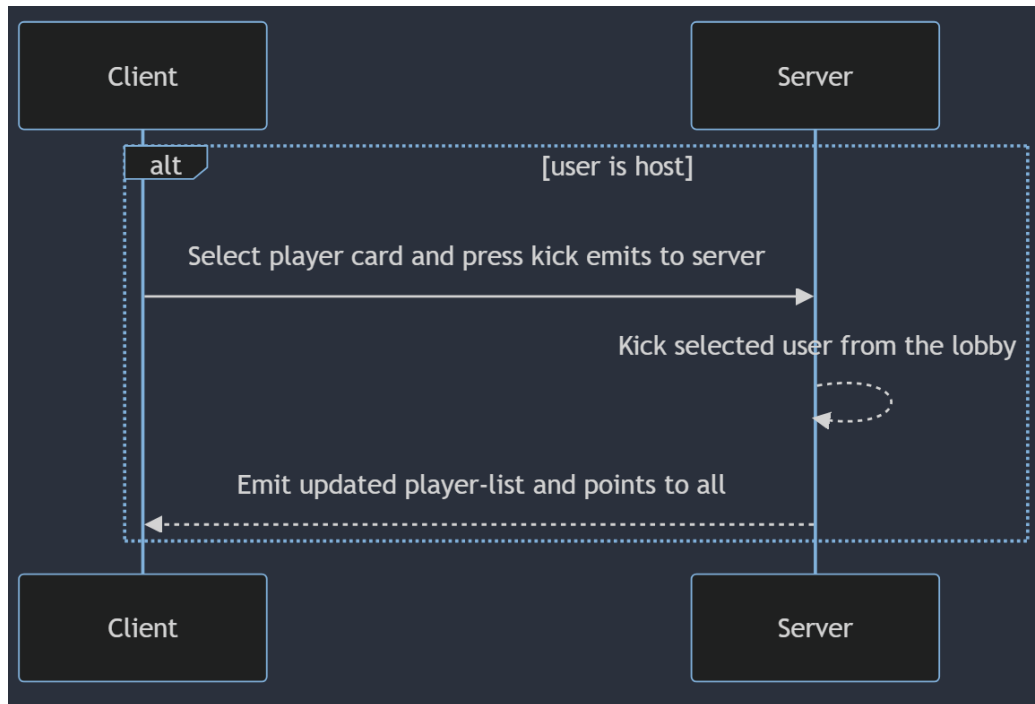


Figure 10. Sequence for Kicking User

Figure 10 demonstrates the process in kicking a user out of the current session. This provides a simplified version of the general process, however, is explained in further detail during the Code Snippets section of the report. When the host selects a player card, it will then confirm they wish to kick them. This emits a message to the back-end server, where it will process the request and remove the highlighted user from the lobby. The lobby will then proceed by emitting the new player list, which no longer contains the kicked user

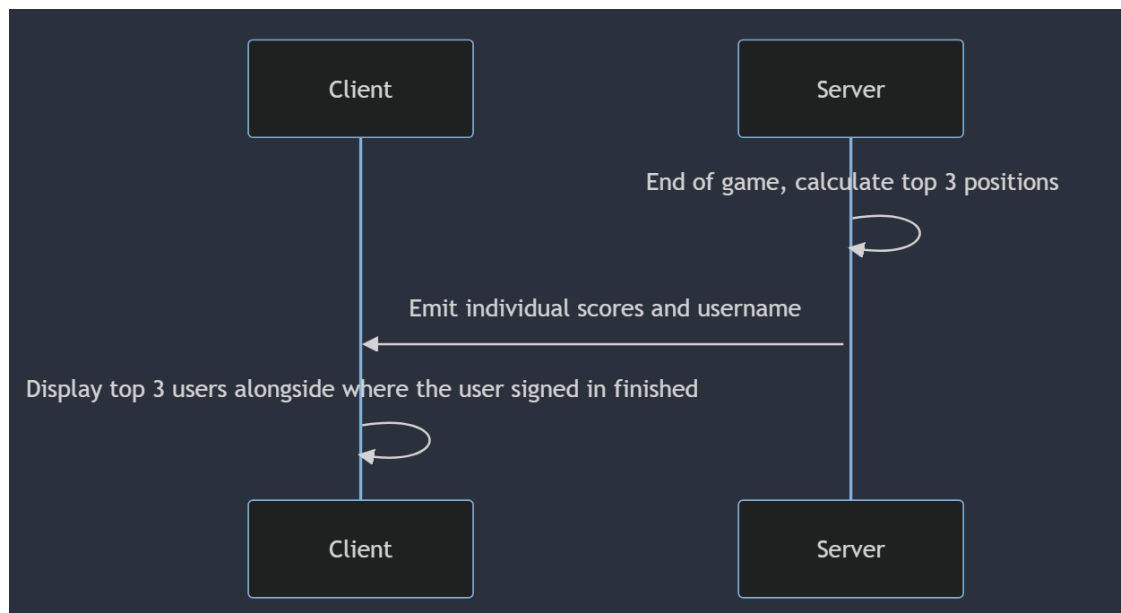


Figure 11. Sequence for Displaying Podium

Figure 11 highlights the process of displaying the podium at the end of the game. The server will calculate the positions of all the members and will return this sorted list to the client. The podium

users will then be displayed for all the users to see, alongside the signed in user's respective individual performance. This signifies to the lobby that the game has been finished.

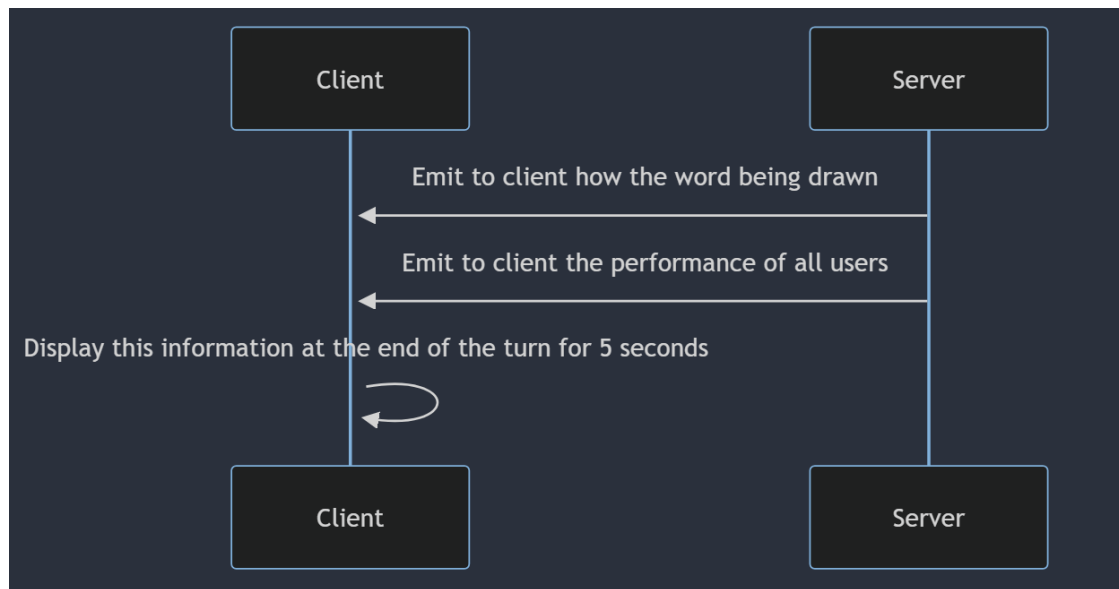


Figure 12. Sequence for Displaying End of Turn Results

Figure 12 is a key component that activates when transferring from one user drawing to another. This is used as a transitional mechanism, signifying the end of the original user's turn drawing. When this activates, it will display the word that was being drawn, alongside the respective names and the points gained of each user who guessed the word correctly (and also the person drawing). This information is visible for 5 seconds, before then transitioning to the next user.

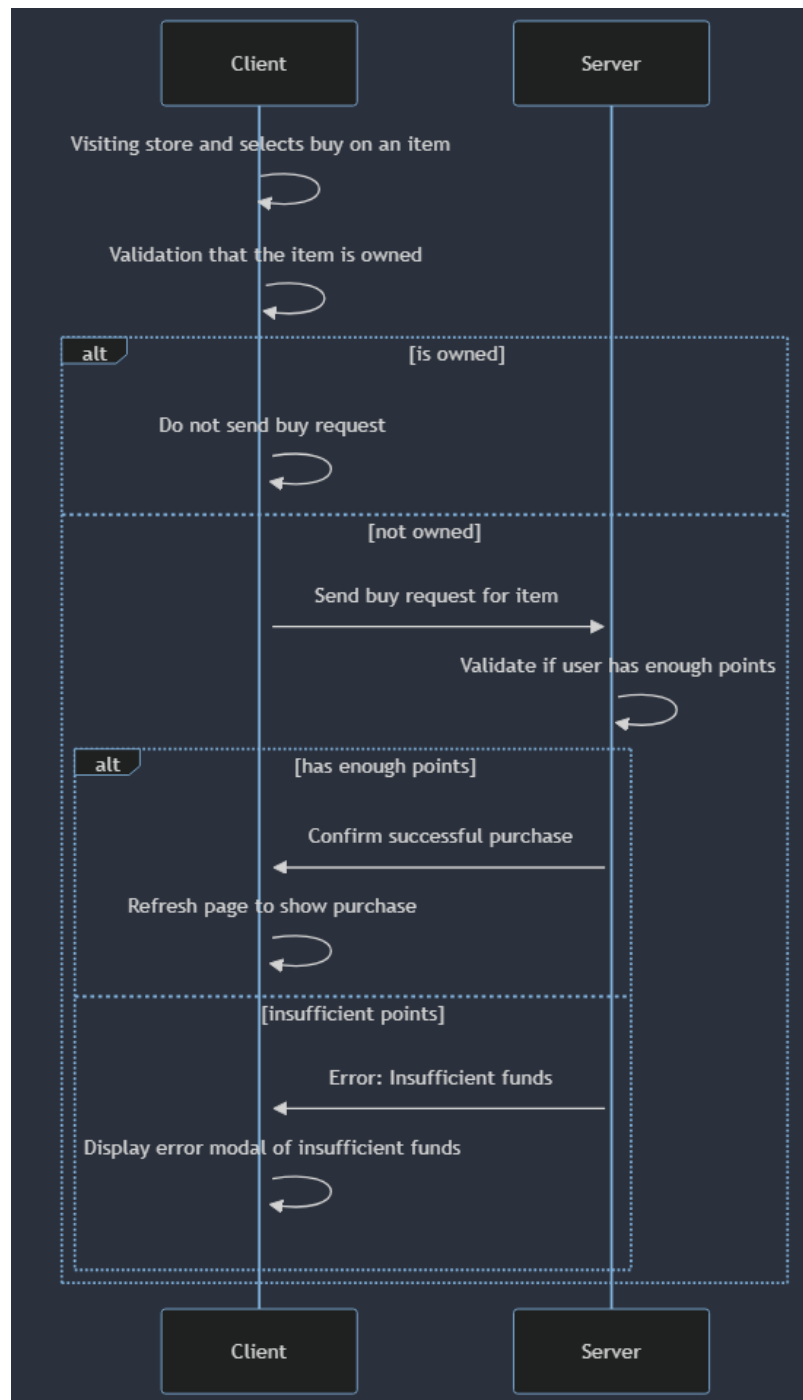


Figure 13. Sequence for Purchasing Item

Figure 13 highlights the full complexity when purchasing from the store. When on the /store route, it is only possible to purchase an item that is unowned, hence validation will take place on both front and back-end servers. If the item is unpurchased, only then will the request be sent to the back-end server, where it is then checked if the user has enough points to purchase this. If lacking funds, an error should be displayed for the user to understand the status of the purchase. On the other hand, if the purchase was successful, then the page will highlight that the new item is owned.

3.4 Use Cases

I have designed the following use cases to demonstrate the functionality of the website, walking you through from registering the account all the way to interacting with others.

Figure 1. Registering

Use Case	Registering Account
Actor	User wanting to make an account
Sequence	The user will visit the “/register” route. From here, they will enter their choice of username, email address and password. The information will be validated to ensure they meet the criterion specified. Failure to meet this will be displayed to the user. Once they entered the necessary and valid information, they can proceed with submit. The information is now stored in the database (and the password in its encrypted form), and the user can navigate through the website
Alternative Sequence	The user will visit the “/register” route. From here, they will select “Sign In With Google”. This will create a pop-up to allow the user to select the Google account they wish to use. Once selected, they can review the permissions required to use the website, and then proceed to create the account. This will re-direct them to complete their login by entering a chosen, unique, username.

Figure 2. Logging In

Use Case	Logging In
Actor	User with an account
Sequence	The user will visit the “/login” route. From here, they will be prompted with a form to input their username and password. After inputting their password, they will submit the form to the back-end server, where the request is authenticated, and they are allowed to progress if the user exists.
Alternative Sequence	The user will visit the “/login” route. From here, they will select the “Sign In With Google” option. This will re-direct to the Google website window, showing the google accounts they have currently signed-in on their browser and available to sign in with on the website. Once selected an account, the service will check if the account has signed up to the website and allow them to progress if so.

Figure 3. Browsing & Joining Public Lobby

Use Case	Browsing Public Lobbies
Actor	User with an account
Sequence	The user will begin by signing into the website. Once they have signed in, they are taken to the home page, which will allow them to scroll through the different public lobbies available to join. From here, by selecting a lobby, they will be able to join the session.

Figure 4. Creating A Private Game

Use Case	Creating A Private Game
Actor	User with an account
Sequence	The user will begin by signing into the website. Once they have signed in, on the home page, they are presented with the list of public lobbies and a button at the bottom of the screen allowing them to create their own game. Upon pressing this button, they are taken to a lobby customisation page, where they can change the time people have to draw, number of rounds to play etc. In addition, they can distribute the lobby link to invite others. The user that has created the lobby is deemed to be the host of it.

Figure 5. Joining A Private Game

Use Case	Joining A Private Game
Actor	2x User with an account
Sequence	The first actor must create a private game (as seen in the previous use case) to start the lobby. Once the lobby has been initiated, to invite others to join, they must distribute the custom URL link (e.g. http://localhost:3000/lobby/privateLobbyID). This can also be obtained from the button below the settings, which when clicked will load the URL into your clipboard. The URL will contain the custom lobbyID, only available to those who have received the link, making it a private game.

Figure 6. Playing A Round

Use Case	Playing A Round Within Lobby
Actor	2x User with an account
Sequence	The first actor must create a game and invite the second actor to join it. Once the host (first actor) has customised the lobby settings as required, they must click “START” to begin the lobby gameplay. From here, the host will select a word to draw. The rest of the players will have to guess

	it in the chat box. If all the players have guessed the word (except the person drawing) or the timer has run out, it will then go to the next person in the lobby to choose their word and draw it. This process is done until all the players in the lobby have had one go. Depending on the number of rounds, it may be repeated according to the lobby settings.
--	--

Figure 7. Purchasing from Shop

Use Case	Purchasing from the shop
Actor	User with an account
Sequence	To begin, the user must be signed in and on the home page of the website. From here, they will select the store button to browse the items available from the shop. To purchase something from the shop, the user will select the “BUY” button located underneath the item. Once clicked, the points will decrement according to the sum of the purchase, and the user will have access to the item.
Alternative Sequence	If the user has already purchased the item they are looking at, the button is already disabled. This is to avoid repeated transactions where the customer may have accidentally clicked it. In addition, the text of the button will also be different, displaying “Already Own”.

3.5 Algorithms

In my project, part of the timeline involved researching different algorithms and tools that can be utilised to create new functionality. This led me to discover the flood fill algorithm, which allowed me to create my fill component for the canvas. A quick summary of the algorithm is that it has a starting point from the user’s mouse pointer. All neighbouring pixels that are of the same colour as the starting point will then have their colour replaced.

This algorithm is run when the user has selected (and unlocked) the fill tool. When they go to use the component on the canvas, the starting point will be when they trigger the mouseDown event. The algorithm will then scan neighbouring pixels and if they meet the criteria (same colour as the starting pixel), the colour is then replaced by the new desired colour. Flood fill terminates when the pixels no longer “have the particular attribute” [22]. With regards to parameters, it takes a target colour and the new colour that should override the current.

I had opted to use this algorithm over boundary fill, as my canvas is required to respond to various boundary colours, however using the boundary fill algorithm would restrict the canvas to stop at boundaries of the specified colours, potentially ruining the design the user has created. Within the flood fill algorithm, there exist different variants. I had chosen to use 4-connected, therefore when analysing a pixel’s neighbour, it will only check the pixels directly touching it. On the other hand, 8-connected will also expand to the diagonally adjacent pixels, which could cause the colour to expand beyond the boundary, hence I decided against using this version.

The disadvantage of this algorithm is that it typically can be quite slow. Originally, I had opted to implement the algorithm myself, utilising recursion to go through the neighbouring pixels and terminate once it reaches the end of the starting colour, however this led to severe performance issues

for larger operations. As a result, I made use of the more optimised and refined q-floodfill package from npm, which allowed for a significant performance boost for my project [23].

Chapter 4: End System Development

4.1 Installation Manual

To run the application, the user must have Node.js downloaded. The installer can be found on the following website:

<https://nodejs.org/en/download>

Once you have Node installed on the device, clone the Git repository to your local computer or download the files. Open a terminal up within the repository and run the command

npm i

from the back-end folder and the front-end folder. This will install the node modules required for both servers.

To deploy the system, I have created a custom npm script following [18] that will run both front-end and back-end servers at the same time. In the terminal located in the root of the repository, run the command

npm run start

to deploy both servers. If there is an issue involving concurrently, whereby it says it is not installed on the system you can either install it using **npm i concurrently** on the back-end, or manually run the servers. This can be achieved through opening a terminal in the root directory and run

node server.js and open another terminal in the front-end directory and run **npm start**

This will run the website and the back-end server to allow for interaction between the two. My system uses a local MongoDB database on the local device, therefore anyone attempting to deploy the website will require MongoDB on their machine. To use the project with a database (which a requirement from deployment), you must install mongodb on your device which can be found in the following links:

<https://www.mongodb.com/docs/manual/installation/>

or through mongosh (after installing the server) via:

https://www.mongodb.com/docs/mongodb-shell/?_ga=2.80922007.1681306570.1701989949-391924372.1690298159

Another difficulty when running the application will be trying to use OAuth 2.0. This uses a .env file, as the information within is sensitive and relevant to my Google account, hence is unable to be shared publicly. As a result, OAuth 2.0 will not work without these credentials.

4.2 Work Log

Copy Of Diary.md

12/10/23

- Created front-end react app
- Created back-end Express.js directory
- The next steps will be to begin creating the Express server

13/10/23

- Initialised server and listen on port 3001
- Removed React-initialised files
- Displayed welcome message to ensure page outputs
- Created base template for Register and Login components
- Established react routing to display different components

16/10/23

- Added React states to store LoginPage details
- Added React states to store RegisterPage details
- Function to handle POST request to server created

17/10/23

- Installed passport and mongoose dependencies
- Establishing authorisation file to handle registration requests
- Created MongoDB database
- Registered details are saved to the database

18/10/23

- Added React state to store login status
- Login successfully implemented into Server
- Home page added to be re-routed to after login

19/10/23

- Canvas implemented to provide drawing capabilities
- Fixed bug preventing drawing from beginning automatically

23/10/23

- Implemented input to adjust line thickness variable
- Updated canvas line to adjust to line thickness input

25/10/23

- Added tests to ensure registration handles errors appropriately

- Improved auth.js error handling to send error messages back to client

26/10/23

- Added tests to ensure login handle errors appropriately
- Login tests are all passed

27/10/23

- Created canvas tests to ensure states update on mouseEvent

30/10/23

- Created test file for managing and updating canvas to all

31/10/23

- Pass tests designed to ensure connection with socket.io

01/11/23

- Create tests to handle transmitting drawing data
- Added code to pass tests
- Added validation to ensure correct transmitted data sent

07/11/23

- Create socket manager for front end socket to connect to back-end
- Bug fix to stop drawing from not displaying correctly
- All drawings are sent to server, before sent back to client (including client drawing)
- Modified and added new tests through TDD process

08/11/23

- Added new tests to track first connection
- Refactored code to pass tests
- Refactored code to allow only first connection to draw
- Added states and context to manage login status
- Imported into HomePage

09/11/23

- Built tests for HomePage to ensure those logged in can only view
- Created code to pass front-end tests
- Store user details in React context
- Built back-end tests to authenticate users
- Created code to pass back-end tests
- Unauthenticated users sent straight to login page 10/11/23
- Attempted to create front-end TDD tests for ChatBox
- Refactored to create front-end code for Chatbox
- Created back-end TDD tests for ChatBox

- Refactored to create back-end code for ChatBox
- Fix issue with compiling front-end code This is done by researching Jest mocking to learn how to mock responses from Axios

15/11/23

- Created interim report
- Populated interim report with a plan

16/11/23

- Updated interim report
- Committed gitignore file to avoid committing node_modules folder
- Removed node_modules folder from the repository

17/11/23

- Refactored ChatBox code to pass tests
- Exported ChatBox function to test handling messages
- Added ChatBox to canvas component Next Steps: Add the context into the canvas and use that username, rather than hard-coded value Create Lobby.jsx to store components rather than adding to Canvas

19/11/23

- Created Lobby.test.jsx to begin creating lobby functionality through TDD
- Designed first test and written code to pass test
- Updated interim report
- Reconfigured registering tests and page to add email field

20/11/23

- Create Game class tests for TDD process
- Created Game class to manage interactions of lobby
- Integrated Game class into server file
- Create generating lobby back-end tests
- Created code to pass tests
- Ensure only created lobbies can be joined (no random lobbies) Next Steps: Make sure drawings from one lobby do not affect another lobby If a user joins late, display the history of drawings

21/11/23

- Created GameDispatcher class to dispatch requests to appropriate games
- Unit tests created to make GameDispatcher functionality
- Passed unit tests
- Moved server code interacting with game to GameDispatcher methods
- Use methods within server to interact
- Completed next steps planned on 20/11/23 Next Steps: Add lobby customisation component

22/11/23

- Added lobby customisation component
- Created new methods in GameDispatcher and Game classes to support customisations Next Steps: Create Round class to handle each turn

26/11/23

- Refactored test code
- Removed unused test files
- Added test code to previous files to ensure testing all functionality

5/12/23

- Refactored front end test code
- Created Round class
- Created Round class test file
- Developed tests for Round class
- Through TDD process, created Round class functionality

6/12/23

- Created timer to change who is drawing at the end of their elapsed timer Next Steps: At the end of the round, remove components (conditional rendering) Distribute a word at the start of user's turn

7/12/23

- Create tests to display who is drawing
- Developed code to pass the tests
- Integrate new code into Lobby

13/12/23

- Updated tests to provide displaying words functionality
- Create new tests for displaying words
- Developed code to pass new tests
- Refactored code to integrate word selection Next Steps At the end of gameplay, remove components (Still needs doing) Update points on back-end and display to front-end

14/12/23

- Refactored authentication to import User model
- Update points on back-end Next Steps At the end of gameplay, remove components

20/12/23

- Added text to end of gameplay
- Defined tests for further settings (public and private lobbies)
- Added further settings for public and private lobbies

24/12/23

- Define TDD tests for creating lobby card
- Create code to pass TDD tests
- Modify game dispatcher to return list of public games
- Display front-end lobby card according to dispatcher return

27/12/23

- Add functionality for undoing the most recent path
- Add functionality for clearing the canvas Next Steps: Research ways of creating a fill component and determine if it should be implemented Research ways of creating a text component and determine if it should be implemented Begin creating route for store

28/12/23

- Utilised floodfill to create fill component Next Steps: Research ways of creating a text component and determine if it should be implemented Begin creating route for store

01/01/24

- Adjust schema for googleId
- Create route for Google sign in

03/01/24

- Create front-end route for completing profile (as username required)
- Add login with Google button to select account

04/01/24

- Modify back-end serialisation of User schema
- Add back-end route to let OAuth users input their username
- Began TDD process for adding store
- Add conditional rendering to only display "fill" button if unlocked Next Steps: Conditional rendering if item is already bought Discover other tools to implement

09/01/24

- Conditional rendering displays item is already bought Next Steps Begin styling the website

10/01/24 - 17/01/24

- Imported Bootstrap into the project to allow styling customisations
- Adjusted login and registration components to use Bootstrap components and custom CSS styling
- Created a new page component for Login and Register

- Utilise conditional rendering to display form displayed according to the pill selected
- Imported Bootstrap-icons to display icons accordingly
- Re-designed lobby cards on Home Page using Bootstrap
- Re-designed buttons on Home Page

18/01/24

- Utilise Google SVG from bootstrap-icon to improve appearance
- Added functionality for registering a Google account into the website
- Converted HTML skeleton of the lobby customisation into a Bootstrap form using components
- Configured the design of the lobby customisation page to display the player list to the left of the customisations

19/01/24 - 24/01/24

- Re-configured lobby componenets to use Bootstrap Button functionality
- Re-configured PlayerCard to use the Bootstrap Card component
- Designed new components using Popover to display components allowing a cleaner user experience when changing colour or adjusting thickness of brush
- Replaced button text to use Bootstrap icons

25/01/24

- Adjusted the player list component of the website to make it scrollable. This will ensure that the website will stay in the correct aspect ratio and shape; especially useful with populated lobbies
- Moved the buttons into one "toolbar" that is presented below the canvas. The benefit of doing this is that then the screen is more organised and structured, enabling a better user experience.
- Employed custom CSS to highlight the canvas, therefore making it clear to the user that they are drawing
- Restructured the website, to present the information in a logical order (players on the left, canvas and toolbar in the middle, chatbox on the right)

29/01/24

- Added new feature to display how long the user has left on their turn when drawing, to make it clear when it will swap to the next person
- Added new feature to display the current round being played. This will help new users understand how much has been played so far and how many rounds are remaining
- Display two new icons to replace text, highlighting on the player card the user that is the host and the user that is currently drawing respectively
- Modified back-end code to support new socket transmission to implement these features

30/01/24 - 31/01/24

- Created a new component from unit tests for easier use when a user has created a new lobby and they would like to distribute the URL, simply clicking on it will put the link in the user's clipboard
- Added a bootstrap toast notification so that it makes clear to the user it has been copied
- Fixed issue with the canvas where after applying a background, it would be transparent, by adding CSS styling to create a white background on the class
- Implement new background obtained from Adobe Stock license
- New ChatBox implementation to highlight messages more clearly and structure them more appropriately relative to the other components in the lobby
- Next Steps: Finalise design of the lobby by adding a word "hint" and fix the issue that exists with timer (can be done in candidate branch) Reconfigure the home page take up more the space available as it is quite poor in style currently

01/02/24 - 08/02/24

- Implemented a bug fix for an issue whereby the time would not decrement by one due to function running repeatedly
- Implemented another bug fix where the game state did not update - new players would be able to draw yet the game had ended
- Games are only displayed on the home page if their game state is not finished
- Flashing red countdown for when the timer reaches 10 seconds
- Added a hint to be displayed as "_" to demonstrate the length of the word
- Home Page now displayed more uniformly to take up more amounts of space and improve the overall aesthetic design
- Create TDD tests for the new store card component and implemented necessary functionality and styling
- Point updates are now floored to avoid long decimals in their sum
- Next Steps: Add a default profile picture and with the option of buying new profile pictures from the store

13/02/24 - 16/02/24

- Modified the database schema and model to include a profile picture value, which will store the file name of the associated user's picture, so that it can be displayed with ease
- Define a default profile picture to be used when creating an account (not possible to modify)
- Display the profile picture on the home page and lobby card to show other player's your avatar and also the host's avatar when looking for a lobby to join
- Add new profile picture components to the store allowing them to purchase new ones with the points they have earned and saved up
- Implemented functionality to clear the canvas at the end of the user's turn
- Next Steps: Implement functionality to allow user's to select their profile picture from the settings

20/02/24 - 28/02/24

- Modify the user schema to store the list of all purchased profile pictures, allowing them to modify their avatar as they please to the various owned pictures they have purchased

- Modify components to support the changes implemented to the User model
- Created TDD tests for PictureSelector component and implemented code to pass these tests (only show the purchased profile pictures and the currently selected picture)
- Change the design of the PictureSelector, so rather being it's own route, it will appear as a modal on the home page

29/02/24 - 06/03/24

- Implement styling fixes to clearly display to the user the Username, as before it was blocked by similar colours
- Changed profile picture button is now a dropdown that is toggled from selecting the user's profile picture / username
- Dropdown functionality also includes a sign out button to reset the current session and navigate to the /login page

07/03/24 - 18/03/24

- Display the points total and the word at the end of each round to improve the user experience. This helps to show the progress each user is making at the end of each turn through a 5 second gap at the end of the timer
- Styled new points total and word using CSS and React-Bootstrap components
- Re-styled the toolbar at the bottom of the canvas to help show active tools
- Improved the responsiveness of the website; the canvas can now scale and maintain aspect ratio to support varying screen resolutions and devices
- Improved the components for LineThickness and ColourSelector to create a better sense of simplicity and drawing
- At the end of the gameplay session, a play again button will appear that only the host can interact with to start the session again
- Improved registering and logging in to check if the username and email address is valid and hasn't been taken and display this dynamically to the text fields
- Implemented server side validation for confirming if password and username are valid fields and re-direct to the home page after registering

19/03/24 - 27/03/24

- Display the word to the user drawing as a measure in case they either forget or were not present for the selection
- Usability improvements by implementing front-end timer to automatically select the word after 10 seconds to avoid users not selecting anything
- Fixed issue of only one user in lobby by removing user from an active game session if they are the only ones remaining
- Bug fix to re-distribute the "host" role if the host were to leave the game
- Restrict starting the lobby unless 2 or more users are present
- Efficiency improvements to avoid frequent fetching of user information from the database
- Calculate points for the drawing user
- Utilise a public repository to fetch words that can be drawn
- Further usability improvements by allowing users to re-connect from another device without any issues
- The old device will be kicked from the lobby session and will continue to run

- Bug fix where host could kick themselves out
- Bug fix to leave the socket room upon disconnection
- Reformatted unit test cases after adjusting code from standard HTML to Bootstrap components
- Usability improvements by implementing keybinds for all users to access

28/03/24 - 05/04/24

- Server side validation that a user has selected a word after 10 seconds
- Finalised automated end-to-end tests using Cypress
- Display responsiveness improvements to adjust according to screen size
- Custom command to run front and back end servers simultaneously
- Updating final report

06/04/24 - 09/04/24

- Refactored the Expired Session modal out of the lobby and into it's own individual component with TDD unit test
- Bug fix from testing to fix issue when clearing and kicking user
- Refactored canvas toolbar outside of Lobby into it's own individual component with TDD unit test
- Completed documentation testing to ensure the documentation is of high standard
- Refactored RevealWord into new component alongside TDD unit test
- Refactored Podium into new component alongside TDD unit test
- Usability enhancements by allowing messages to flash green if guessing word correctly
- Bug fix to avoid issue where user triggers keybinds typing in custom words

4.3 Documentation

Documentation for both the front-end and back-end servers and their components were created through the use of JSDoc. In addition, a user manual and instructional manual for deployment are also available in this report for guidance on deploying and interacting with the website and its various features.

Chapter 5: Assessment

5.1 Professional Considerations for this Project

5.1.1 Security

Security is a necessity in modern society. It is paramount to ensure the correct measures are in place when storing information, regardless of its sensitivity. This is required to prevent unauthorised access or modification, and safeguard information. From an ethical standpoint, when users are inputting their information to my website, it is then expected that they will be stored securely, hence I have an obligation to implement security measures. Furthermore, from a practical view, in the event of a breach, this can lead to significant repercussions, that could be either of financial or legal origin, therefore security and privacy is also of importance for not only the users signing up, but also as the owner of the project, it is therefore in my best interests. For my project, I have implemented plenty of security measures to ensure that information has been safeguarded.

As the world becomes more digital, the value of information increases, and hence the impact of breaches can lead to more drastic effects. In the event of a compromised system, this impacts all entities involved. Around the world, regulations are in place, such as GDPR, to highlight the standards required of securing information. Failure to comply with the regulations specific to the region can lead to significant damages. Areas of concern within my project that I have security measures implemented to mitigate against are storage and encryption of passwords and sensitive information, XSS attacks, and unauthorised access and distribution of information.

Within my project, I am storing sensitive information such as the user's password. When implementing this functionality, I had to consider the best ways of storing this information to avoid security breaches and then potential access to this sensitive information. As a result, when storing these values, they are encrypted through salting and hashing, hence in the event of a breach, there remains no unencrypted sensitive information. For a malicious actor to decrypt this, it will require plenty of computational resources and time. Also, to ensure strong privacy for users, this also means to block any weak passwords. As a result, my project enforces a strict password policy, therefore increasing the resistance to brute force attacks. This helps to increase the complexity of passwords, therefore harder to be compromised and decrypted by malicious actors.

Furthermore, by allowing OAuth 2.0 for users to sign in with their Google accounts, this not only proves to be more convenient, but also more secure. OAuth 2.0 lets a user access the service, but without providing sensitive information such as a password, achieved through relying on Google to authenticate the user and provide any required information the database may need. This minimises the risk of exposed account information.

When interacting with the website, it is important to parse information that is being displayed. This allows my website to be resistant to XSS attacks. By using the React framework, it escapes any content rendered inside JSX expressions naturally. As a result, any XSS attacks are rendered as text, rather than executing the malicious JavaScript code. Furthermore, SQL injection attacks will not work on my project, due to the fact I am utilising MongoDB (a NoSQL database) therefore the query syntax is significantly different.

Security is also considered for the lobby of the websites, as it will require clients to sign in to join a lobby or access the home page. This is to authorise all users utilising the website and chatting to other users. For socket communication, nothing will be broadcasted to all users, rather the sockets will join a "lobbyID" and then information is transmitted to participants of that lobby. Once a lobby has finished, it will be deleted from the server, therefore unauthorised participants cannot access lobby information such as users, chat history etc.

If inadequate or no security measures have been implemented, this can lead to large-scale breaches. An example of this from the public domain is the 2017 Equifax data breach. This breach affected 147.9 million Americans, 15.2 million British citizens and 19,000 Canadian citizens [19], resulting in significant damages for Equifax, such as legal actions, distrust amongst customers, significant drop in share price and large fines by regulatory bodies. It was highlighted from this breach that Equifax had failed to safeguard information due to the fact credentials were left unencrypted, and attackers could easily access them and exfiltrate without detection. The attack originated from poor patch management, ultimately leaving the company vulnerable to exploitation from a critical vulnerability. The company was unable to implement the update on all systems [19], and attackers were then able to monitor databases for months without detection. This example highlights the importance of using encryption, as poor security measures allowed sensitive information to be accessed with ease. By storing sensitive data in cipher text, even in the event of a breach, it is then needed to be de-coded which depending on the strength of the information (e.g. for my project, the password strength), and how the information had been encoded, this can take up significant computational resources and time.

5.1.2 Licensing

For advanced web development, another professional issue I was concerned with is licensing. When creating my website, it was a struggle finding the images for my vision whilst adhering to copyright laws and following ethical standards. Imagery is an essential component of my project. It is used throughout the entirety of my website and allows for text to be broken up to create visual appeal and enhance the overall user experience, as it simplifies complex information for the user's benefit. Furthermore, this allows for better engagement amongst users as well.

Whilst selecting images for the website, a key concern of mine was to ensure the correct licensing and permissions were in place to avoid copyright infringement. This proved to be a challenging task, as I had a set vision in mind but the images available would require payment and licenses. My initial idea was to make use of generative AI and provide a prompt that best describes what I require to be made. By using generative AI, it would have allowed me to create my own images, rather than buying licenses for it, however it lacked the capabilities to provide the image I described in its full extent, hence I had to look elsewhere. To find the best images that would fit the aesthetic I wish to have, I researched many different sites for stock images and their costs, allowing me to identify the best selection of imagery and comparing the cost for its implementation. This led me to purchase an Adobe Stock license, providing access to a variety of images I could utilise. This helped me to ensure that the images were licensed correctly and obtained legally, therefore copyright infringement was no longer a concern. Failing to purchase the license but using the images would go against ethical standards and disrespecting the artist. From purchasing this license, I've adhered to a high level of ethical standards by supporting the artist who created the images I've utilised.

Furthermore, despite wanting a similar aesthetic to competitor websites, I chose not to directly copy images from them, highlighting the ethical standards of my project. This allowed me to create my own identity and version that differs from the direct competitors, as the key part of this project was applying the vision and features that I wanted for it, allowing me to not follow suit and steal other website's identity. The use of Adobe Stock allowed me to create my own visual style and appeal, whilst simultaneously supporting the group of creators I bought the images from.

Ensuring the correct licenses are in place is detrimental for a website, as otherwise they may face both financial and legal repercussions. One example from the public domain where these issues were not properly addressed was with Getty Images. Getty Images provide stock images, however, have taken legal action against various organisations that operate using their images despite lacking the correct licensing for it. This further extends to the AI field, as Getty Images was concerned that Stability AI utilised their images without consent [20] to build a competing business. They accused the company of "copyright infringement, trademark infringement, unfair competition, trademark dilution, DCMA violations and related state law claims" [20]. Their claims are further reinforced by the fact that their watermark is also present on the output images from the AI, hence the company would be breaking "the DMCA by trying to remove it" [20]. This remains ongoing, highlighting the

complexity of these cases and licensing rights, hence it is critical to ensure the correct rights to images used.

5.2 Self-Evaluation

Over the duration of this project, some of my achievements were being able to implement all of the functionalities highlighted in the project plan, whilst being able to expand upon the original project scope to include additional features, such as the ability to kick a player, changing sockets if re-joining from another location and simultaneously kicking the other device out, accessibility improvements through keybinds when drawing on the canvas and more. Furthermore, my code can successfully manage and navigate around edge cases, therefore less prone to bugs and exploits. This was able to be achieved through effective time-management. The structure of the project plan allowed me to flow naturally from the different components in my project and therefore integrating them together proved to be of lesser difficulty than initially expected. A major reason for being able to handle edge cases so well is by following the TDD process, because when designing unit tests, you must consider designing tests that fail, succeed, but also challenge the code you have produced to be resistant to glitches and bugs.

In addition, I was able to have a consistent and well-designed structure to my website. The components integrate well together to provide the overall functionality required. The use of React-Bootstrap provided me with a template of components I could customise to my wanting, alongside ensuring their responsiveness to different screen sizes – despite not being a priority, it is incorporated in all of the React components besides the gameplay itself.

Some difficulties I faced on the project was refactoring unit tests in the TDD process. By following this process, after implementing verification checks to ensure the user is signed in, this led to additional complexity when trying to test individual components. This stemmed from the fact these components would try to check if the unit test is authorised by validating if it has either stored cookies or if the react context value for authentication is true. To mitigate the increasing complexity of these tests by validating authentication, I refactored the components to store the checks into an authentication wrapper attached to the necessary routes. This enabled me to resume testing as normal.

Furthermore, as highlighted earlier when discussing the benefits and disadvantages of Cypress, the gameplay component of my functionality relies on 2 user actors. Utilising Cypress, it proved to have a limitation whereby all, but the gameplay aspect of the project could be tested through automation. As a result, when modifying the gameplay, this had to be manually tested to ensure the components are behaving as expected from them, which proved to be incredibly time-consuming. Testing proved to be the difficult aspect of the project due to the lack of experience I have with creating unit and end-to-end tests for web development, therefore a lot of documentation and research had to be explored to understand these tools and implement them effectively to provide the template of my project. Although testing proved difficult to grasp initially, by following through and using TDD and E2E tests, my code is less susceptible to bugs and able to handle edge cases well, therefore despite a harder learning curve, it provided more benefits than issues.

To conclude the self-evaluation, the project was a great success. Despite modifications being made to the timeline drafted in the project plan (as discussed in section 1.4), I was able to successfully implement all the functionalities and goals specified within the allocated time, and able to create new functionalities beyond the original scope of the project because of the effective time-management and flow from one component to another. Although issues arose (primarily from testing), they were mitigated through research into the documentation to gather a better understanding of how they can be utilised in my project.

5.3 Potential Future Enhancements

Within the current stage of my project, I have only considered using OAuth 2.0 to allow for Google accounts to sign-in, however in the future, this could be adjusted to allow for further services such as Apple, Facebook, X (Twitter) etc. The advantage of this is that it will provide a variety of services that can be utilised as the user wishes, providing more options for them.

Furthermore, within the game session there could also be more managing functionality for the host. This could range from the ability to permanently ban users from that game session, or if somebody is typing abusively in chat, a mute function to prevent their messages from being seen by other users. This would help to ensure an enjoyable experience for all of those taking part.

In addition, as highlighted earlier from the key justifications that were made during development, another potential enhancement could be to further optimise mobile devices. Currently, smaller mobile screens specifically, have not been optimised when drawing, whereas all other displays have. Whilst all the other components of the project support mobile devices, optimising the lobby gameplay requires additional complexity, and as this demographic does not fit the target market - was not considered a priority. This is due to the limitations and difficulties of trying to draw a word and interact with other users within a limited screen size, hence the gameplay would be inhibited and cause a terrible experience. To optimise the lobby component for mobile devices, it would need to completely re-structure the current design for these screens, and new components would be required too, such as a keyboard to avoid using the native mobile keyboard as it will adjust the screen by raising (and potentially hiding) the web content.

Lastly, sound effects could be utilised to help enhance user engagement and better their experience. This is because they create a more engaging and immersive experience, therefore utilising them could stimulate other senses beyond the user's vision. This could help to improve retention, as users will feel more engaged and encourage them to continue interacting with the website.

Chapter 6: Bibliography

- [1] Monika Mehra, Manish Kumar, Anjali Maurya, Charu Sharma, Shanu. (2021). MERN Stack Web Development. *Annals of the Romanian Society for Cell Biology*, 25(6), 11756–11761. Retrieved from <http://www.annalsofrscb.ro/index.php/journal/article/view/7719>

When researching different frameworks to create my project, this article helped to highlight the impact of a MERN stack, such as the performance benefits they can have and ease of implementation due to the fact they are all using JavaScript.

- [2] Boyd, R., 2012. *Getting started with OAuth 2.0*. " O'Reilly Media, Inc.". <https://books.google.co.uk/books?id=qcs0LHusAFsC&lpg=PR3&ots=kpILf0XncT&dq=oauth%202.0%20security%20benefits&lr&pg=PR3#v=onepage&q=oauth%202.0%20security%20benefits&f=false>

oAuth 2.0 is a term I knew little about, and this online book helped me to understand the importance and benefits of implementing this mechanism into my project. This will allow users to sign in with their existing accounts.

- [3] Socket.IO. (2023). Socket.IO Documentation. <https://socket.io/docs/v4/>

Socket.IO will allow real-time communication, a necessity for this project when controlling access to the canvas, relaying messages between the users and controlling guesses. The documentation will explain the functionality to me and allow me to implement it in my work, as this is a new aspect of web design for me.

- [4] Rawat, P., & Mahajan, A. N. (2020). ReactJS: A modern web development framework. *International Journal of Innovative Science and Research Technology*, 5(11), 698-702.

React.js will allow me to create the multiple different components individually, and then bring them all together to be displayed. This powerful framework is key for the front-end development and passing information between the different routes created.

- [5] Sik-Lanyi, C. (2012). Choosing effective colours for websites. In *Colour Design* Woodhead Publishing.

Colour theory is vital to website design. By ensuring that I pick a colour scheme that is compatible with the theme of my website, but also compatible with each other, it will allow me to create a pleasant user interface and thus experience.

- [6] Mozilla. (2023) Canvas API, MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

The Canvas API is the heart of the website. This will allow the users to communicate with each other and complete their objectives. The documentation will allow me to take the user input and convert this interaction into a drawing displayed on the canvas.

- [7] Mongoose. (2023) Mongoose Documentation <https://mongoosejs.com/docs/index.html>

Mongoose will allow the Express server to communicate with the MongoDB database. It provides a simpler procedure in accessing the database over the default MongoDB. This will allow the server to easily update the database. The documentation is required for me to query through different methods to best create the functionality I require.

- [8] Passport.js. (2023) Passport.js Documentation <https://www.passportjs.org/docs/>

Passport.js is required for my code to provide authentication, store sessions in the form of cookies to allow users to return with ease and allow implementation of OAuth 2.0 and salting / hashing the user's password into the database.

- [9] Axios. (2023) Axios Documentation <https://axios-http.com/docs/intro>

Axios is incredibly valuable to my project. To allow the front-end of my code to communicate with the back-end Express server, Axios is detrimental to this. All HTTP requests are done through this dependency. The documentation will allow me to see the structure I should be applying to my code.

- [10] Supertest. (2023) NPM <https://www.npmjs.com/package/supertest>

Supertest is an integral part of my project, specifically for testing and development. As my project is employing TDD to develop functionality, Supertest will allow me to host my back-end server, to execute unit tests and ensure the correct functionalities are working as intended.

- [11] Jest. (2023) Jest Documentation <https://jestjs.io/docs/getting-started>

Similar to Supertest, Jest is an important JavaScript testing framework. This is vital to mock objects, mostly to be used for my front-end React tests, as I will need to emulate communication with the back-end server to ensure my front-end works according to the expected response.

- [12] MongoDB. (2023) MongoDB Atlas Documentation
<https://www.mongodb.com/docs/atlas/>

MongoDB Atlas is a cloud database solution provided by MongoDB. This will allow me to move my local NoSQL MongoDB database to the cloud, ensuring for 24/7 availability as it is not suitable to be running it off a local machine. In addition, this will also provide a more secure implementation and security features.

- [13] Singh, M. H. (2023) Analyzing And Improving Web Application Quality Using Design Patterns
<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5bc96df3f498485ac409a9a9908df85d1e84fa98>

Analysing and improving web application quality using design patterns helped me to understand the importance of these patterns, especially for web development. When coding, I actively considered scenarios from where I could adjust my code to fit design patterns, which is what caused the timeline adjustment, as I opted to add more structure to my back-end through a Dispatcher pattern, rather than using an array of items.

- [14] Cypress. (2023) Cypress Documentation
<https://docs.cypress.io/guides/overview/why-cypress>

Cypress is another testing framework I will be employing in my work. This is vital for end-to-end testing. When creating my front-end and back-end, the tests are implemented without consideration for the other respective side. For example, my front-end test will not communicate with a back-end server, but rather just mock the object to emulate the expected response. Cypress will allow me to create end-to-end tests, therefore interacting directly with the front-end and back-end code. This will be vital to ensure communications and functionality are operating correctly in a live environment.

- [15] Q. Liu and X. Sun, "Research of Web Real-Time Communication Based on Web Socket," International Journal of Communications, Network and System Sciences, Vol. 5 No. 12, 2012, pp. 797-801. <https://www.scirp.org/journal/paperinformation?paperid=25428>

In order to transmit information related to gameplay and lobbies, they had to be handled in real-time. From reading this, I understand that HTTP communications are not applicable in solutions that require real-time response. Sockets employ bidirectional communication that utilises minimal overhead, providing an 'enormous reduction in network traffic and latency' proving sockets to be the ideal solution in transmitting information in real time.

- [16] Jha, Shubham (2021). "Understanding the MVC Architecture in the MERN Stack." Medium <https://shubhamjha25.medium.com/understanding-the-mvc-architecture-in-the-mern-stack-aff893abce50>.

From reading this, it highlights the individual components of the MVC architecture, their role, and what frameworks of the MERN stack apply to which components. The article helped to highlight how information should flow between each other to achieve a response.

- [17] Yeshwanthini S. (2021) "Illustration about MERN Stack" <https://medium.com/techiepedia/what-exactly-a-mern-stack-is-60c304bffe4>

From reading this article, it clearly covers the MERN stack and how the concepts are linked, especially through the useful image created. It highlighted the significance of each part of the MERN stack, and further useful libraries that could be utilised within them, such as Axios

- [18] Ravithamara. (2024) "How to run Frontend and Backend with one command?" <https://medium.com/@rwijayabandu/how-to-run-frontend-and-backend-with-one-command-55d5f2ce952c>

From reading this, it demonstrated clearly to me how I can utilise a custom script located in my package.json alongside concurrently to deploy both my front-end and back-end servers. This helped reduced the need for two open terminals when deploying the application and are now a part of my instruction manual for deploying the code.

- [19] Wikipedia contributors. (2024). 2017 Equifax data breach. In Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/2017_Equifax_data_breach

From reading this Wikipedia article, it showed the significance of data security, and why I need to consider how I store data. It highlighted the repercussions of inadequate security and poor management of patch updates, proving to be an excellent example of what can happen when neglecting security and not considering it to be an important aspect of a company, or in this case, project.

- [20] Meshi Law Firm. (2023) "Getty Images, Inc. v. Stability AI, Inc." <https://www.meshilaw.com/litigation-tracker/getty-v-stability-ai>

Getty Images vs Stability AI is a very recent lawsuit, alleging the misuse of licensed photos to train AI. This underscores the importance of licensing, especially when selecting images for a website, and proves to be a key example of the repercussions of this

- [21] React-Bootstrap Documentation. (2024) . "Introduction" <https://react-bootstrap.netlify.app/docs/getting-started/introduction>

React-Bootstrap is utilised in my front-end to take advantage of the pre-configured components and it's Rows and Cols to structure my content. The advantage of this is that I can provide a consistent design across all of my web pages and these components can be

responsive to adjust to different sizes.

- [22] Wikipedia contributors. (2023). “Flood fill”. In Wikipedia, The Free Encyclopedia https://en.wikipedia.org/wiki/Flood_fill

Floodfill is the algorithm utilised to help fill areas of the canvas. From this citation, I help to explain how the algorithm works in general, alongside it’s application within my project as well. Without research into this algorithm, it would not be possible to implement

- [23] Npm, Inc. (2020). “q-floodfill” <https://www.npmjs.com/package/q-floodfill?activeTab=readme>

This npm package provides the documentation of how to utilise and implement an efficient floodfill algorithm. This helps to ensure that my canvas does not provide a delayed response, which is a necessity, as when using the fill tool, it should help to look almost instantaneous.

Chapter 7: Appendix

7.1 User Manual

7.1.1 Creating An Account

To create an account, visit the “/register” route (localhost:3000/register if hosted locally). Upon connecting to the website, you should see this display, presenting the option of registering with either Google or manually inputting your details

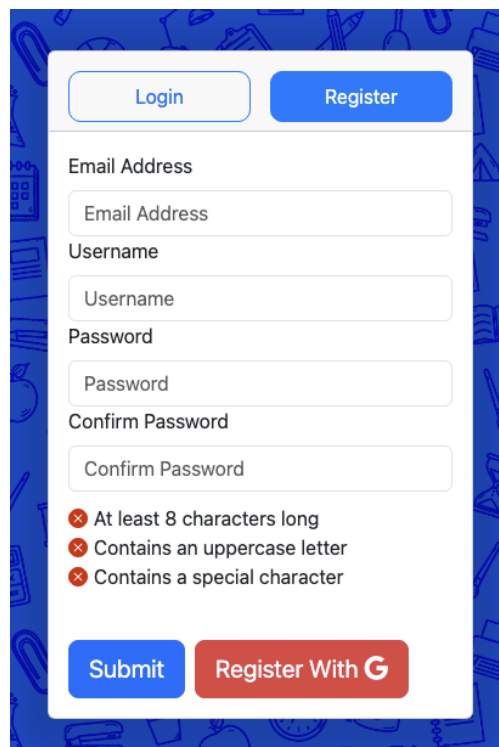
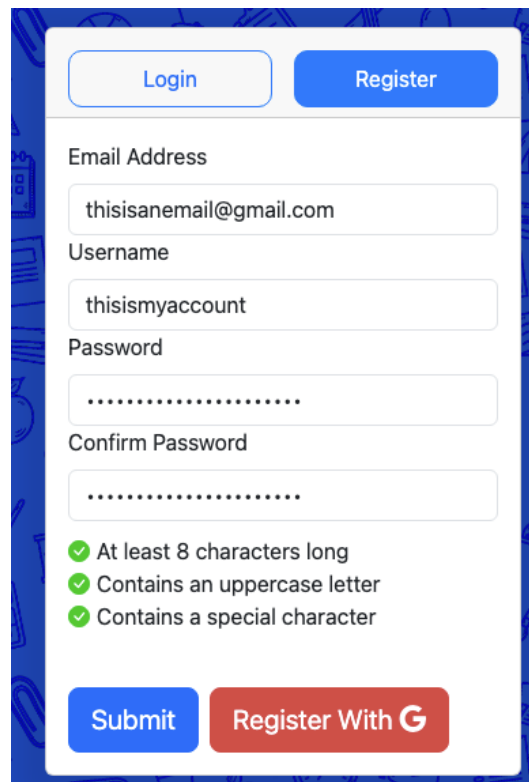
The image shows a registration form on a blue background with white icons. At the top, there are two buttons: 'Login' (white with blue border) and 'Register' (solid blue). Below these are four input fields: 'Email Address', 'Username', 'Password', and 'Confirm Password', each with a placeholder of the same name. Under the 'Confirm Password' field, there are three red error messages, each preceded by a red 'x' icon: 'At least 8 characters long', 'Contains an uppercase letter', and 'Contains a special character'. At the bottom, there are two buttons: 'Submit' (solid blue) and 'Register With G' (red with a white 'G' logo).

Figure 14. Image when registering an account

Proceed ahead with entering the details of your user account. This will be validated in real-time to ensure compliance such as a unique username, email address, and a password complying with the criteria shown. These are dynamically updated, highlighting any issues. It will not be possible to proceed without ensuring these details are valid.



Registration form fields and buttons:

- Login (button)
- Register (button)
- Email Address: thisisanemail@gmail.com
- Username: thisismyaccount
- Password:
- Confirm Password:
- Submit (button)
- Register With G (button)
- Validation messages:
 - ✓ At least 8 characters long
 - ✓ Contains an uppercase letter
 - ✓ Contains a special character

Figure 15. User details inputted into the form presented

After inputting the valid details, pressing submit will send the HTTP request to the server and create your user account. Upon creation, you will then be navigated to the /home route, where you will be presented with this screen

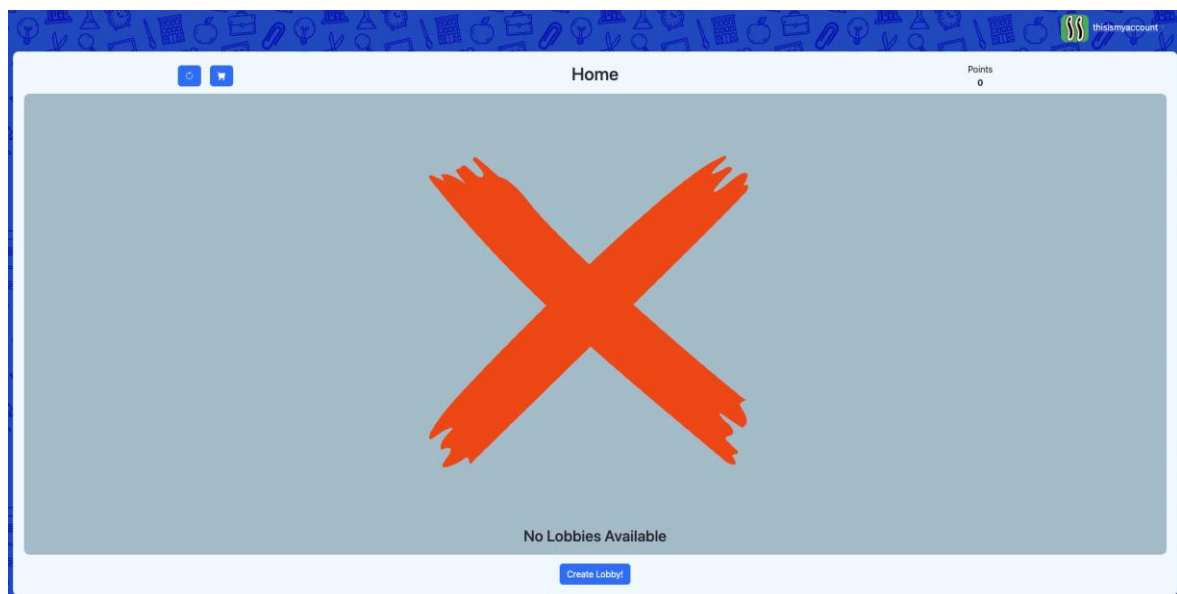


Figure 16. Home page after registering an account

If opting to instead register with your Google account, simply click the **Register With G** button shown on the register page (Figure 14). If you are signed in with only one Google account on the browser, you will be re-directed to a new form to select your Username. On the other hand, if your browser contains multiple Google accounts signed in, you have the option of selecting which account will be used.

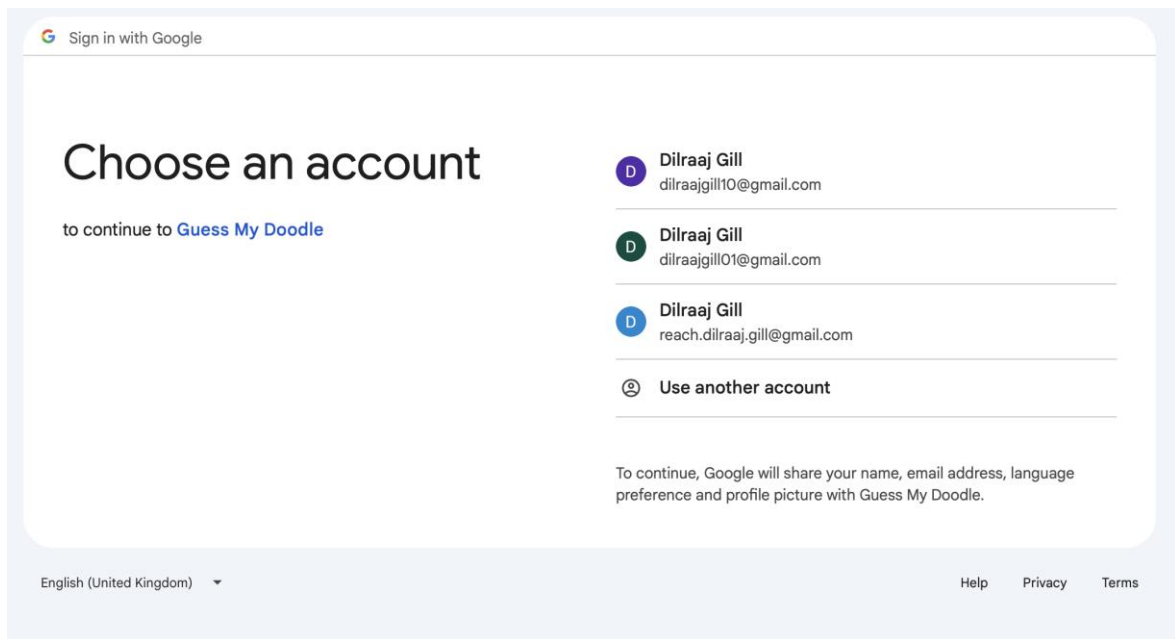


Figure 17. Selecting a Google account to register with

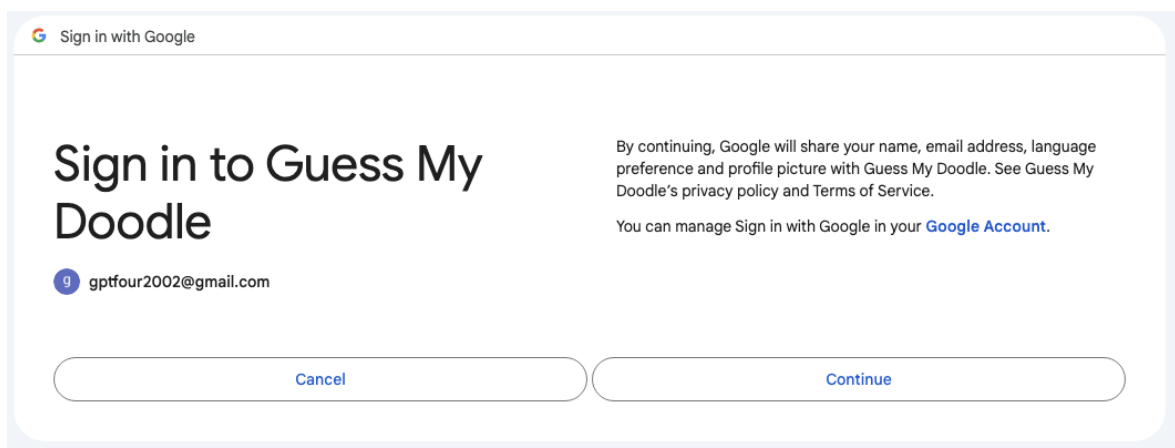


Figure 18. Confirmation of permissions

Upon confirmation of the account to sign up with, a new form will appear asking for the username you wish to use.

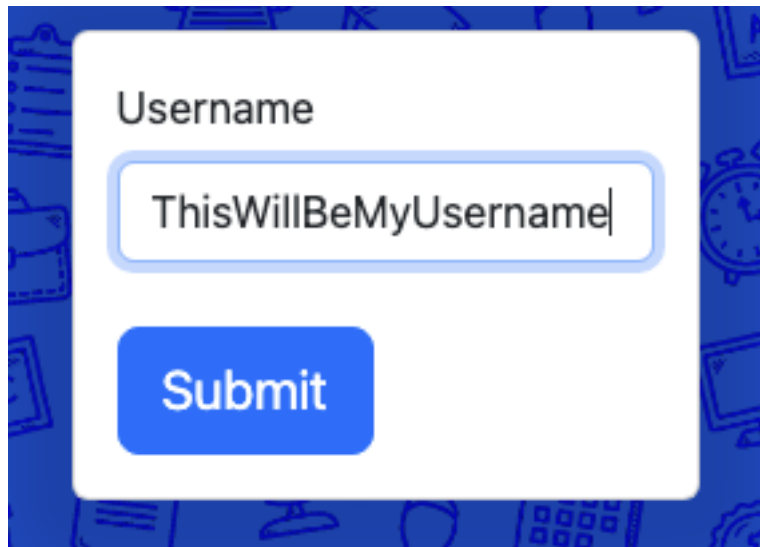
A login form with a white background and rounded corners, set against a blue patterned background. It features a 'Username' label, a text input field containing the text 'ThisWillBeMyUsername', and a blue 'Submit' button.

Figure 19. Username that the user wants to select

Pressing submit after inputting a valid username directs the user to the home page.

7.1.2 Signing In

To sign in with your user account, visit the “/login” route (localhost:3000/login if hosted locally). Upon connecting to the website, you should see this display, presenting the option of signing in with either Google or manually inputting your details

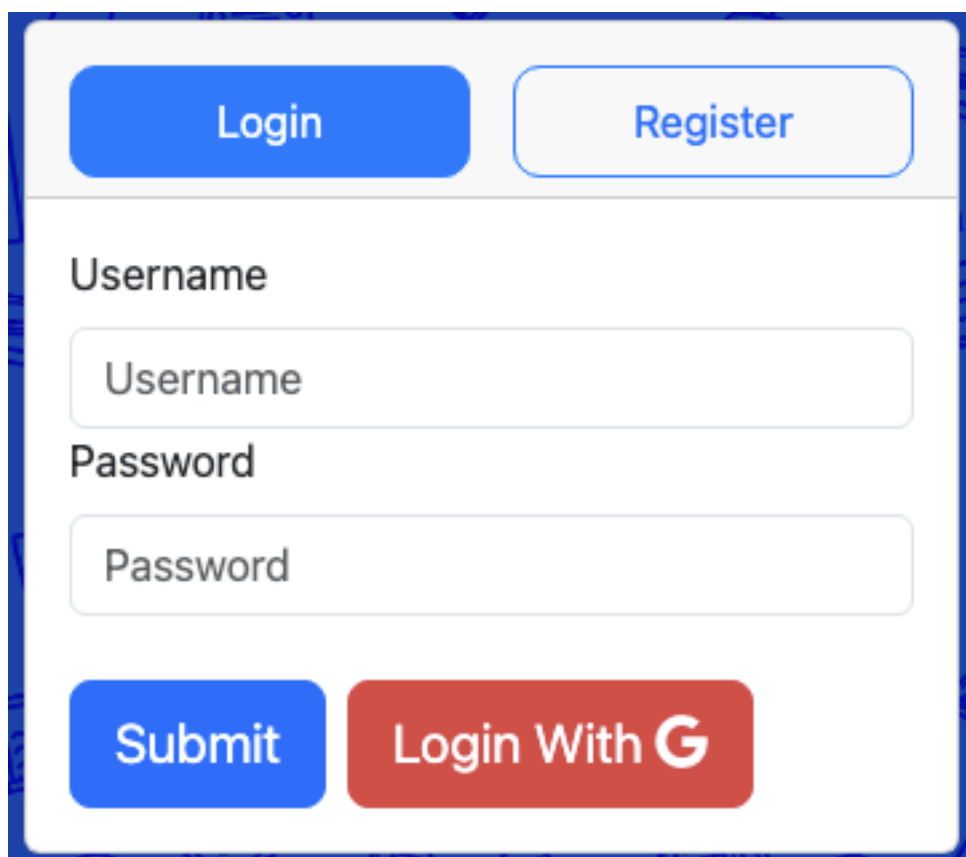
A login form with a white background and rounded corners, set against a blue patterned background. At the top are two buttons: a blue 'Login' button and a light blue 'Register' button. Below these are two input fields: 'Username' and 'Password'. At the bottom are two buttons: a blue 'Submit' button and a red 'Login With G' button.

Figure 20. Login form presenting details to enter

If you registered your account through Google, you must proceed by clicking the **Login With G** button, whereas if you opted to sign in by manually inputting your details, you should enter the details used into the form presented. Logging in with Google will present the following screen, where you will select the account registered (this may be automatically chosen if only one account is signed into the browser).

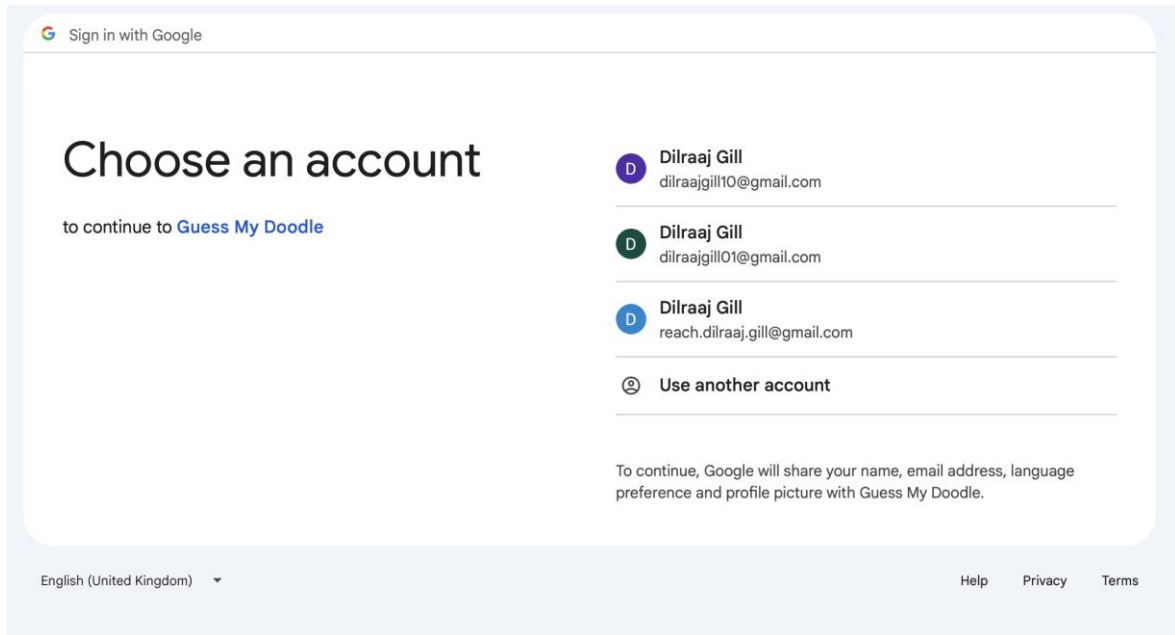


Figure 21. Selecting a Google account to sign in with

Entering the correct user details and pressing **Submit** or selecting the Google account will navigate you to the home page

7.1.3 Creating A Lobby

To create a lobby, the user must be signed in and on the home page.

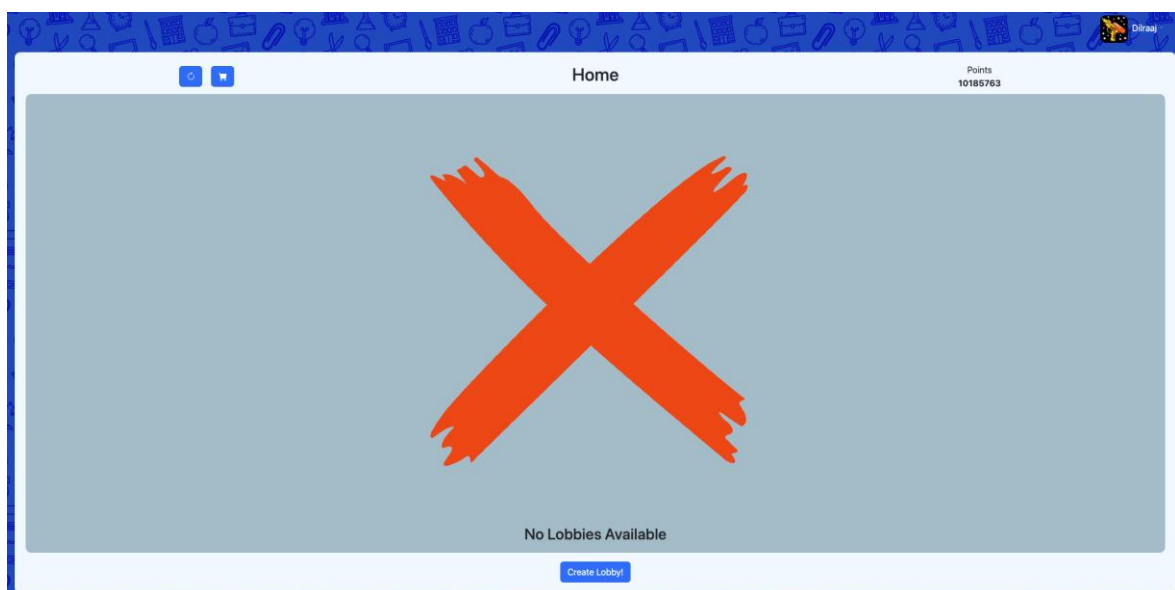


Figure 22. Home page when viewing the website

On the screen presented, a **Create Lobby** button is visible at the bottom. Pressing this will create a lobby generated of random ID.

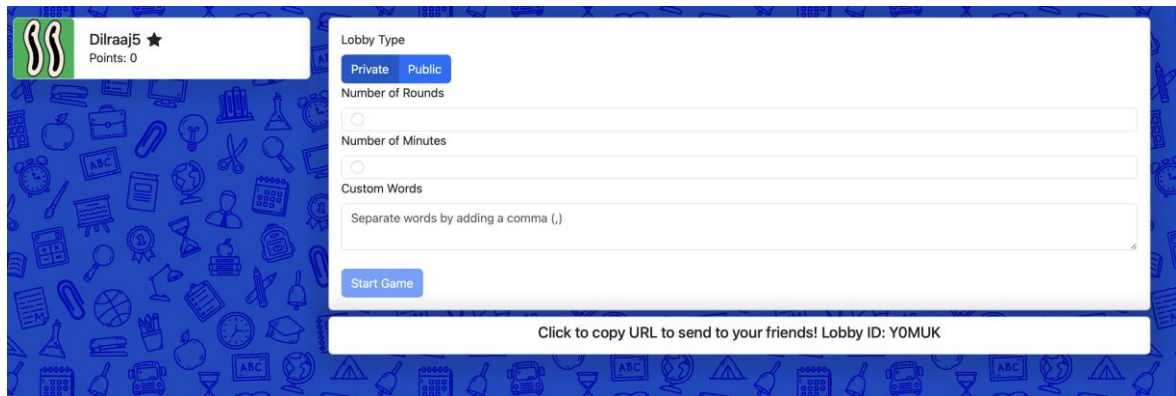


Figure 23. Lobby settings page to allow customisations

From here, the user can customise various lobby settings before starting the game. By changing the lobby type to public, any user on the website can join from the home page, however by keeping the lobby type to private, users can only join from the unique URL presented. It is possible to change the number of minutes, round and allowing for custom words (where they must be split by a comma) from this menu.

7.1.4 Joining A Lobby

To join a public lobby, these are presented on the home page when you visit the website.

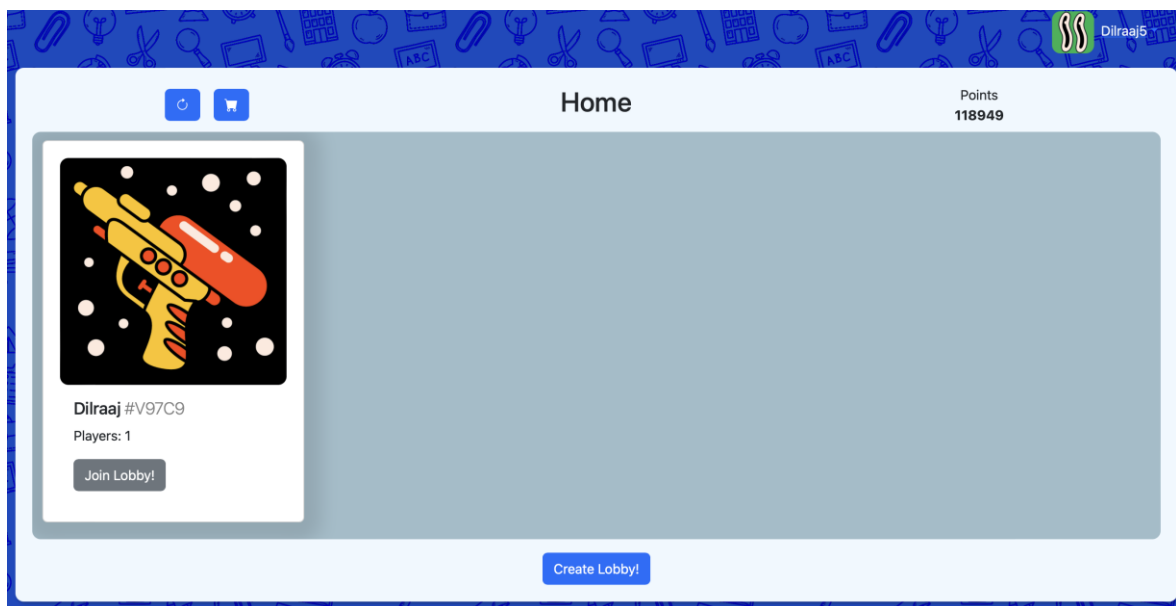


Figure 24. Joining a lobby from the home page

By selecting join lobby, you can join the session. Lobbies displayed here are either on the settings page or already started. Lobbies which have ended are filtered out of the display. By selecting **Join Lobby**, the user is now in the active session.

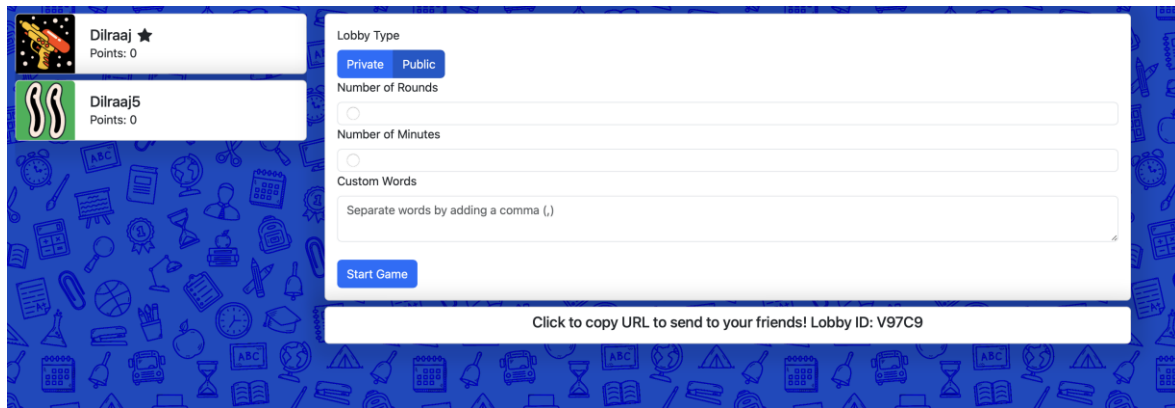


Figure 25. Updated lobby page once a new user joins in

Any updates on the settings are updated for all users as soon as they are processed by the server.

On the other hand, to join a private lobby, the user in the game session (either host or anyone else within) can click the button below settings that will automatically place the custom URL into the clipboard and distribute this accordingly. Any user that receives the URL can then navigate to it and join the session. They are required to be signed in.

The player list on the left-hand side showcases icons. The star is for the host, and the pen indicates who is drawing (not visible when deciding settings as no drawing user). To begin playing the game, there must be a minimum of 2 players in the lobby, otherwise the **Start Game** button is deactivated. Once ready, clicking the button will transform the lobby into an active game session. Only the host can start it.

7.1.5 Drawing Word

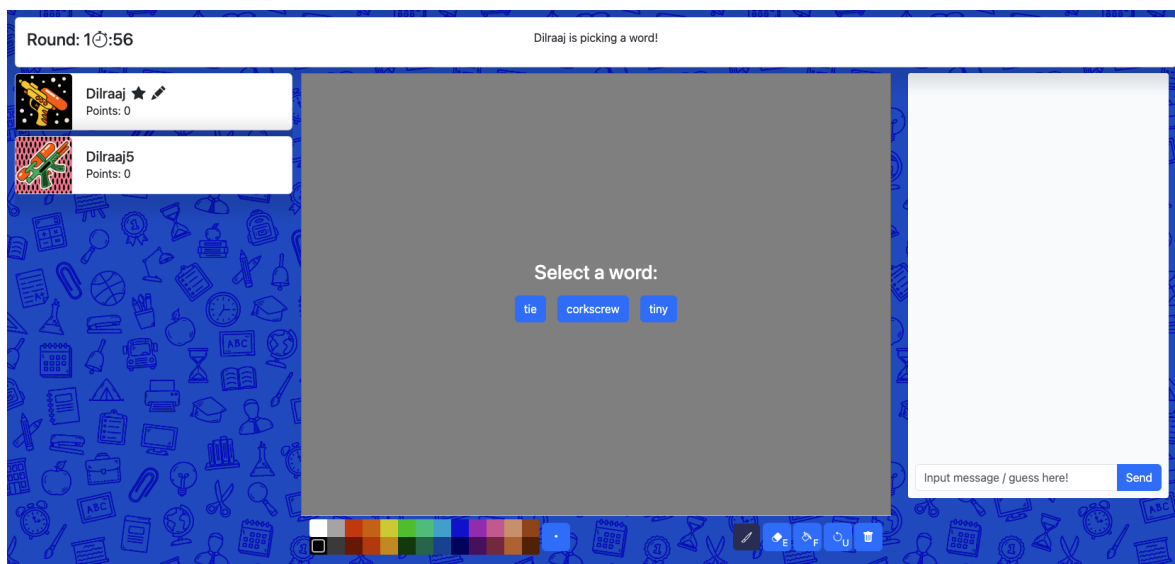


Figure 26. Prompt given to the user when selecting a word

When presented with a list of options, you have **10 seconds** to select your choice, otherwise it will be made automatically for you.

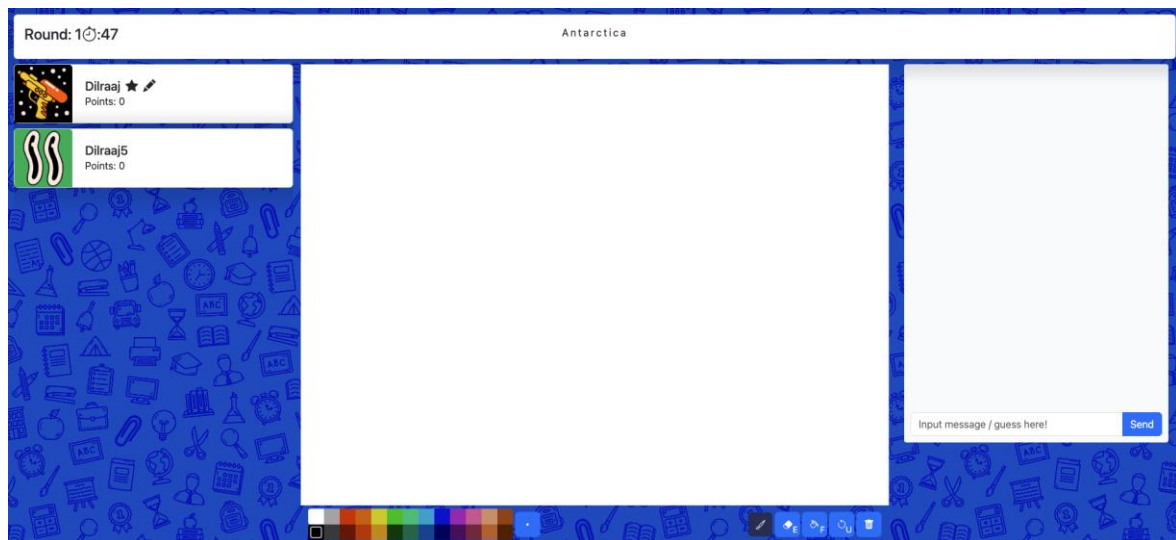


Figure 27. Canvas visible for the user to begin their drawing

After the word has been chosen, you can now make use of the tools available to draw. Furthermore, the top left highlights what round it is and the time remaining. The top-middle of the screen displays to you the word selected. This remains hidden from those who are guessing, and only serves as a reminder. Colours at the bottom can be utilised to create realistic drawings with colour accuracy. The button to the right of it allows the user to modify the thickness of the brush / eraser tool. To the lower right of the canvas is a brush tool, eraser, fill tool (**if unlocked from store**), undo and clear. The eraser removes the brush stroke, and the fill tool will fill an area with the selected colour. Undo will clear the most recent action and delete will clear the entire canvas. The chatbox remains restricted to only those guessing.

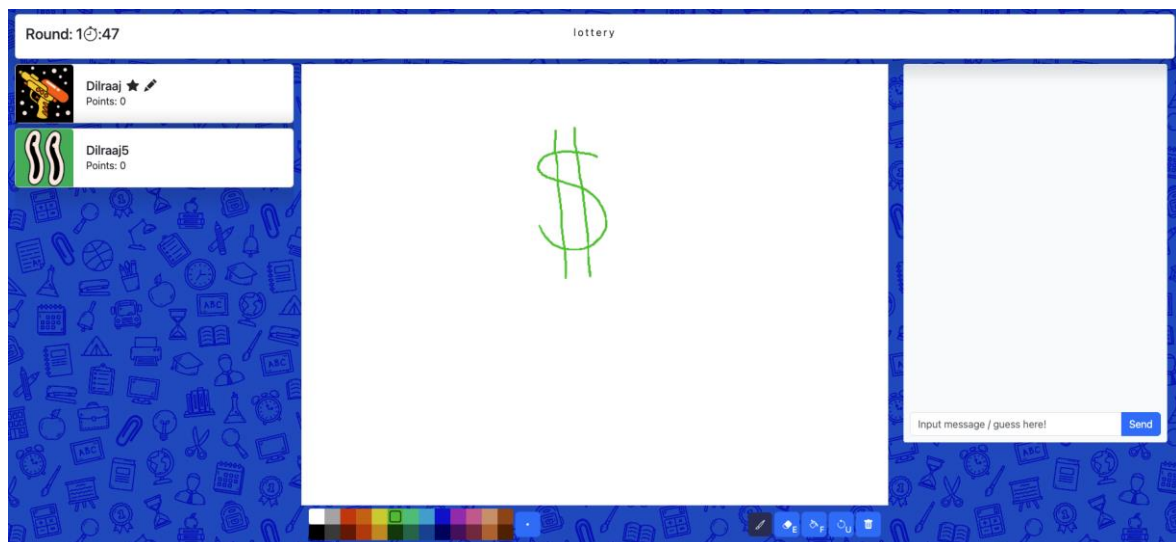


Figure 28. Outcome after utilising tools to draw word selected

7.1.6 Guessing Word

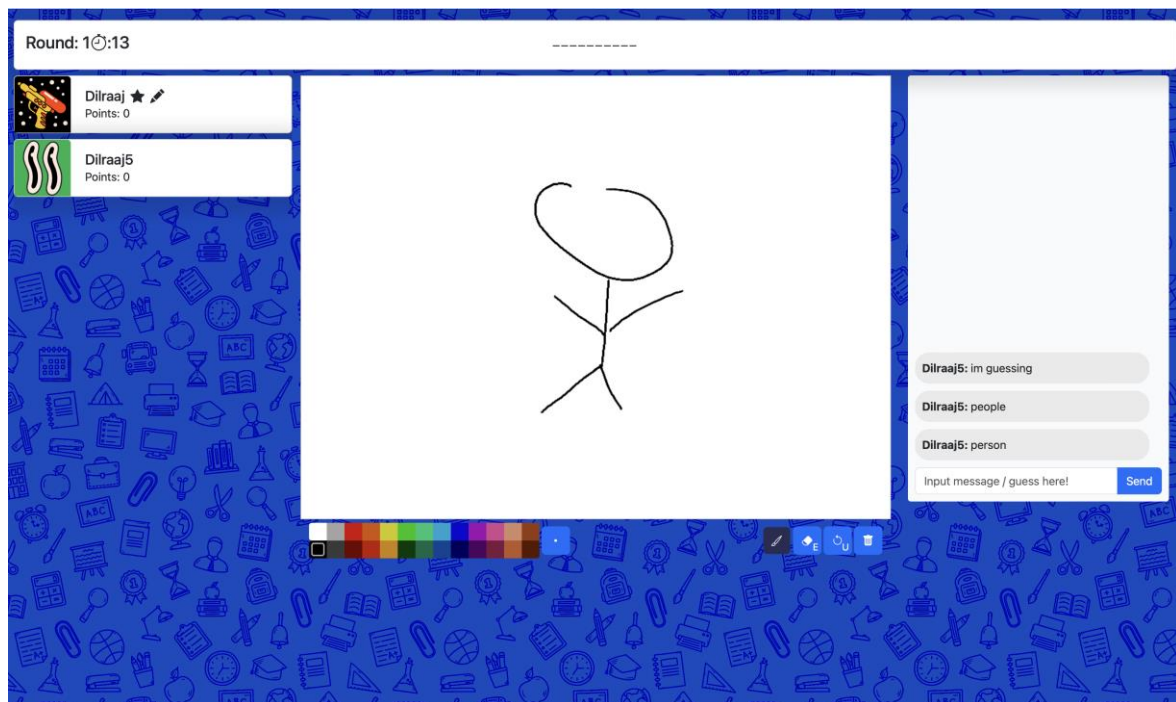


Figure 29. Perspective of user who is guessing the word being drawn

If you are guessing the word being drawn, you can make use of the chatbox provided. This serves for dual functionality – communicating to other users and placing your guesses. Correct guesses will not be shown users to see, as a means of avoiding cheating. Furthermore, hints are displayed at the top-middle to help give an indication of the length of the word selected. The drawing user's turn will finish either at the end of the timer, or if everybody in the lobby has guessed correctly – whichever come's first.

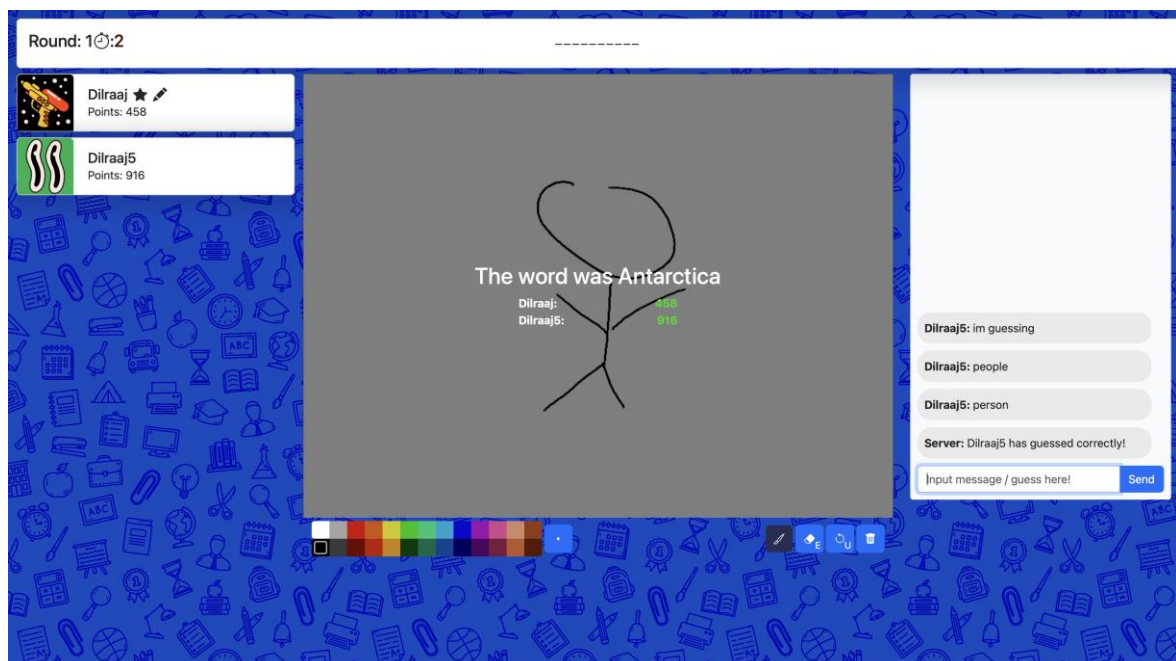


Figure 30. Summary of each user's turn at the end

At the end of the user's turn, a message is displayed, highlighting the word that was being drawn alongside everyone's performance. This can help to distinguish who performed well and who did not perform as well (*Figure 30*).

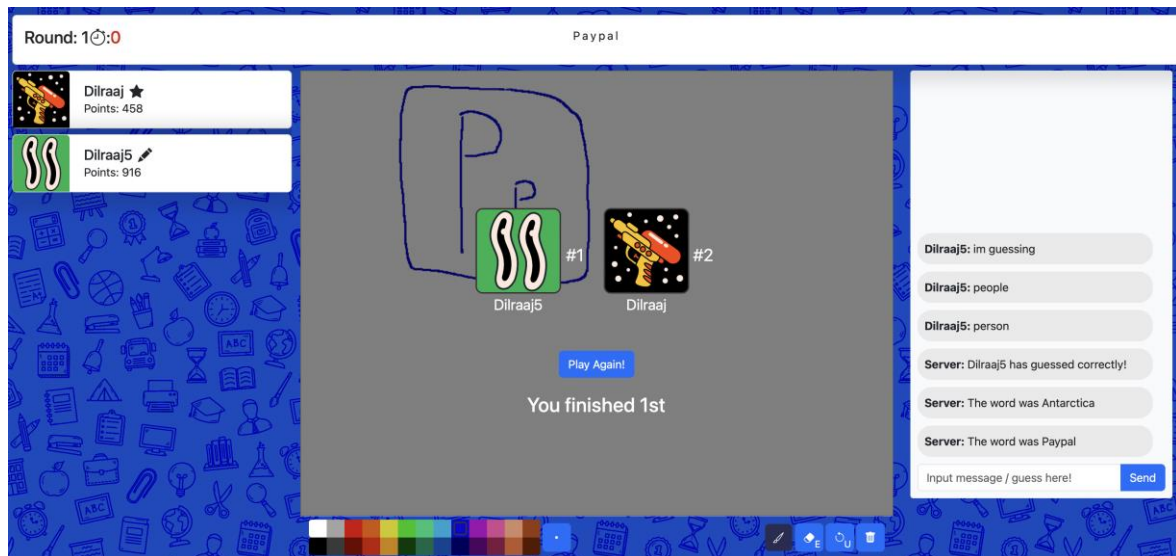


Figure 31. Podium positions and user's position at end of the game

At the end of the game, you will be able to see what position you came, alongside the users of the podium positions (top 3). From here, if the host wishes to play again, they can select "Play Again".

7.1.7 Store

To navigate to the store, either access the "/store" route or click the cart icon on the home page. This will direct you to the store page.

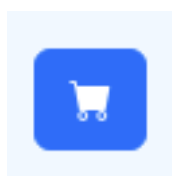


Figure 32. Store icon visible from the home page

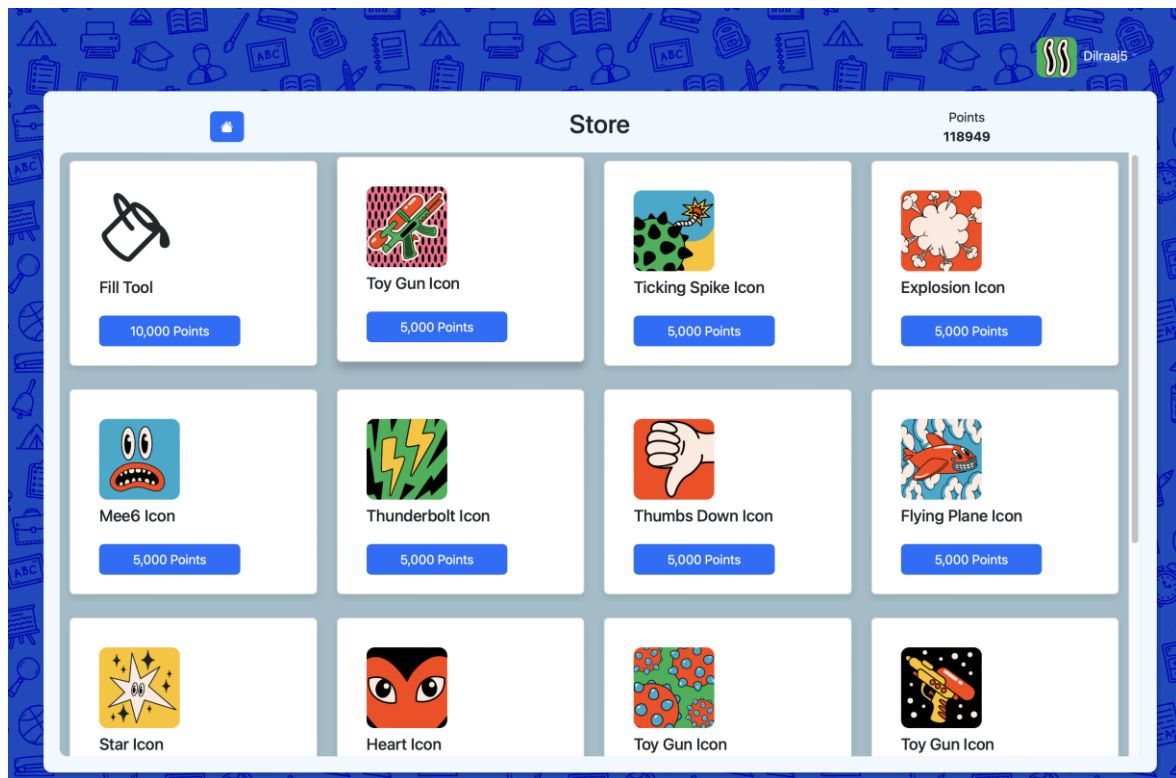


Figure 33. Each item in the store visible for purchase

From here, you can browse freely and purchase items as you wish. To earn currency, this can be achieved by playing games and guessing the word correctly or having other users guess your drawing correctly. With enough points saved, you can then purchase an item as you wish by clicking **Buy**.

If purchasing the fill tool, this item will automatically be added to your toolbar when drawing. Likewise, if purchasing a profile picture then it is also automatically applied, but you can go into the settings and select another picture as you wish.

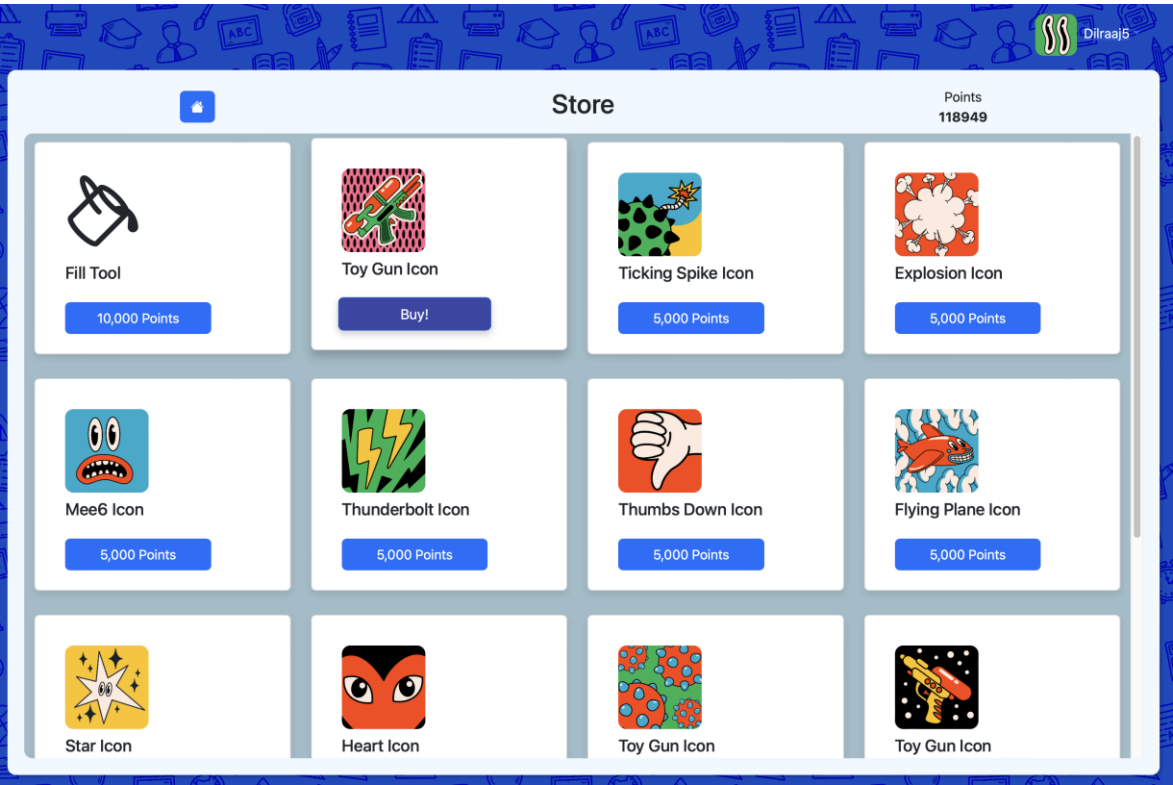


Figure 34. Pressing buy on the item to purchase

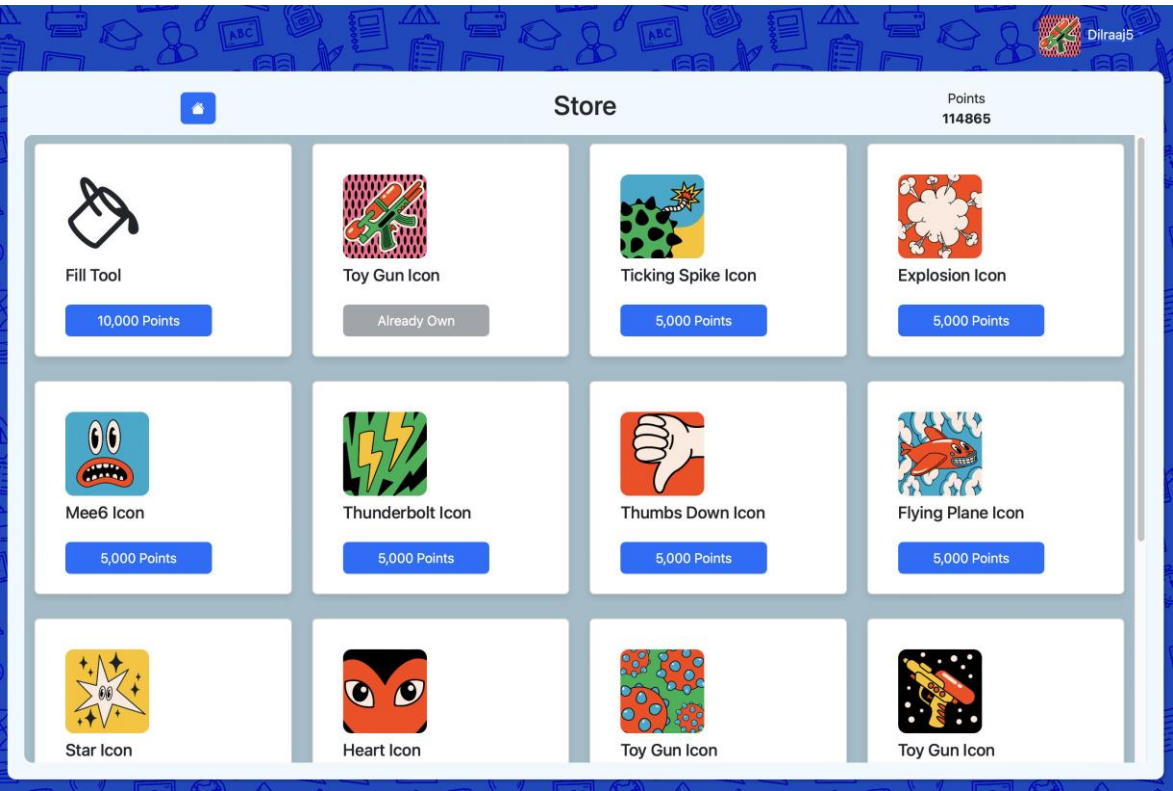


Figure 35. Update takes place instantly and button is deactivated

7.1.8 Changing Profile Picture

From either the store or home pages, it is possible to change your profile picture. Simply click the top right on your user profile avatar and username as shown below.



Figure 36. Dropdown clickable on the user's profile

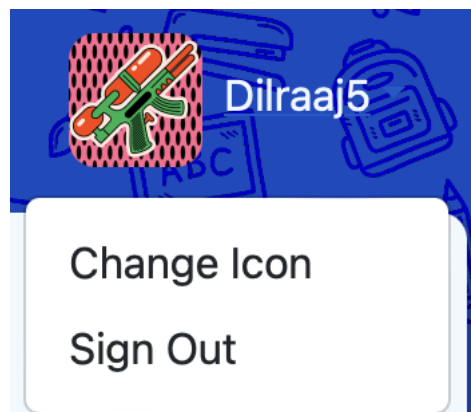


Figure 37. Clicking on the profile makes dropdown visible

This will bring up a menu of options, allowing you to sign out or change your selected icon. By selecting **change icon**, this will present a modal alongside your options.

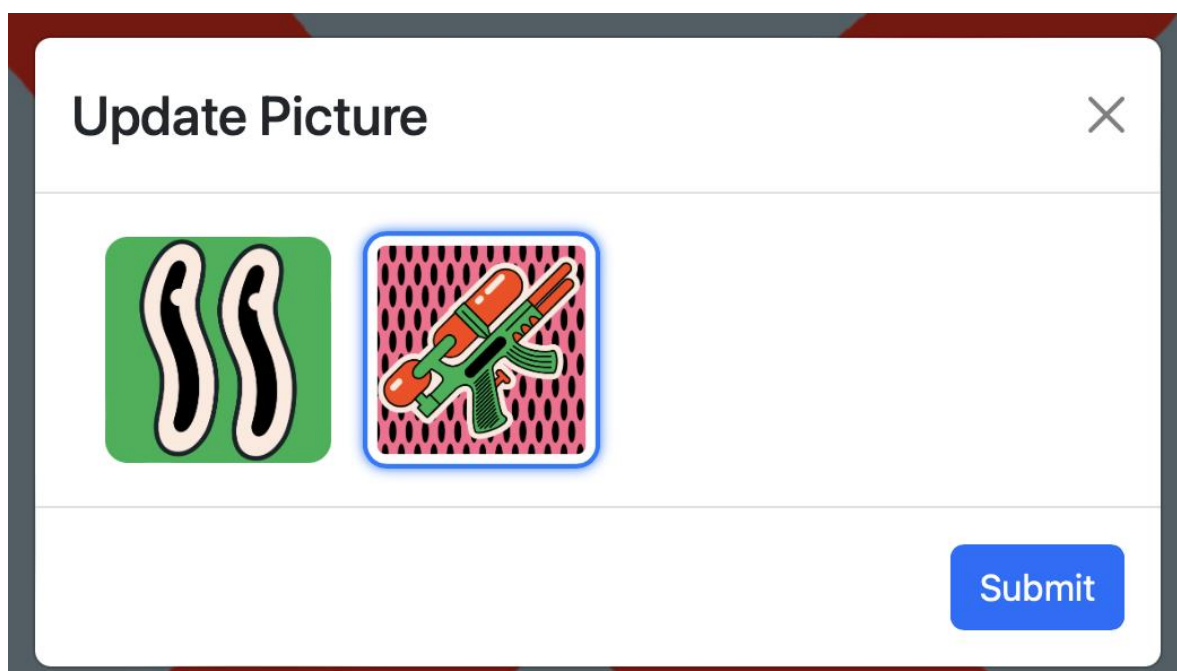


Figure 38. Profile pictures available for selection

Simply select your profile picture of choice and click submit to apply the changes, or you can close the modal from the top right to cancel any changes from taking place.

7.1.9 Signing Out

From either the store or home pages, it is possible to change your profile picture. Simply click the top right on your user profile avatar and username as shown in *Figure 36* & *Figure 37*.

This will bring up a menu of options, allowing you to sign out or change your selected icon. By selecting **sign out**, this will remove any cookies and remove your information, whilst navigating you to the /login route.

7.2 Code Snippets

7.2.1 Start Game

Start

```
start() {  
  // Start the lobby and adjust state  
  this.listCustom = this.splitCustomWords(this.customWords);  
  this.words = this.words.concat(this.listCustom);  
  this.state = "drawing";  
  this.io.to(this.id).emit("set-state", this.state);  
  // Set the timer and get the first person drawing  
  this.timer = this.selectedTimer * 60 + 5;  
  this.round = new Round(this.players, this.lobbyId, this.words, this.io);  
  this.io  
    .to(this.id)  
    .emit("currently-drawing", this.round.getCurrentDrawer().socket.username);  
  this.io.to(this.id).emit("new-round", this.roundCount + 1);  
  this.beginTimer();  
}
```

Figure 39. *start() Method: Game.js*

When starting the game, this action calls upon the start() method. This will split the custom words into an array. As shown in the demo video, the custom words are comprised of sentences separated by a comma, therefore this method will split the input wherever it locates a comma and then concatenate the split words alongside the original, default words. This will provide the selection of available words for the user.

Following this, the status of the lobby changes from “settings” to “drawing”, indicating that the lobby is now active, and this is emitted to the WebSocket of all the users. The timer is then initialised according to the selected duration, with 5 seconds added to display the word at the end of the user’s turn. A Round object is also created to begin the gameplay for the first round, and then the timer begins. The timer is handled in the Game class, and this is to control the interactions of the gameplay between users in the Round class.

Begin Timer

```
beginTimer() {
  // Start a timer for the user to draw within
  if (this.roundCount < this.maxRounds) {
    this.timerId = setInterval(() => {
      if (this.timer <= 5 && this.round && !this.revealWord) {
        // Emit word and display this alongside how each user performed
        this.io.to(this.id).emit("end-points", this.round.returnTurnPoints());
        this.io.to(this.id).emit("reveal-word", this.round.selectedWord);
        this.revealWord = true;
      }
      if (this.timer <= 0) {
        // Display chosen word in the chatbox
        clearInterval(this.timerId);
        this.revealWord = false;
        this.io.to(this.id).emit("receive-message", {
          text: `The word was ${this.round.selectedWord}`,
          username: "Server",
        });
        if (this.round) {
          if (this.round.hasNextDrawer()) {
            // If the timer is over and someone else is left to draw, then go to them
            this.resetNextDrawer();
          } else {
            // If nobody else is left to draw, delete the round and make a new Round object
            if (this.roundCount + 1 < this.maxRounds) {
              this.nextRound();
            } else {
              this.roundCount += 1;
              this.endGame();
            }
          }
        }
      }
    }, 1000);
  } else {
    if (this.round.allGuessedCorrect() && !this.revealWord) {
      // If everyone has guessed correctly, move to the next person
      this.timer = 6;
    }
    this.timer--;
  }
} else {
  this.endGame();
}
```

Figure 40. *beginTimer()* method: *Game.js*

The timer will begin by checking if the `roundCount` is smaller than the number of rounds to be played, which is determined by the settings. If smaller, the timer will then be initialised, repeating every second and decrementing the value of timer by 1 second every time. This is to allow for the Game to check if the timer has gone below 5 seconds, meaning that the drawing user's turn will be over, broadcasting the gain in points by each user and the word being drawn. If the timer reaches 0, it is then cleared, and the word is shown in the chat as well. After the end of a user's turn, it must then check if there is another available user to draw. If another user is eligible to draw (i.e. they have not drawn in this round as of yet), then the process will be repeated with them drawing. On the other hand, if there exists no such user, then a new Round must be created. To comply with the user's settings, it first must check that if by playing another round, it is not exceeding the maximum number

of rounds wished to be played. If it were to exceed the maximum number of rounds, then the game has ended. Another way of a user's turn ending is if all members of the lobby guessed the word correctly, then the timer is changed manually to trigger revealing the word and the performance of users.

7.2.2 Undo Most Recent Action

```
undoDrawing() {
  const index = this.lastMoveIndex();
  if (index !== -1) {
    this.drawingHistory = this.drawingHistory.slice(0, index);
    this.io.to(this.id).emit("undo-move", this.getDrawing());
  }
}

/**
 * Find last replaceable drawing action
 * @returns {number} Last time the user did an action that is undo-able
 */
lastMoveIndex() {
  for (let i = this.drawingHistory.length - 1; i >= 0; i--) {
    if (this.drawingHistory[i].type === "move") {
      return i;
    }
  }
}
```

Figure 41. `undoDrawing()` & `lastMoveIndex()`: *Game.js*

A key component of interacting with the canvas is the functionality that permits undoing the most recent action. When the user first calls the button, the command is sent to the `GameDispatcher`, where it will verify the user trying to undo is the active drawer. If true, it is then sent to the `Game` object.

When drawing on the canvas, the first `mousedown` event will emit a message to the server of drawing type `move`. This indicates that a new path has begun, allowing the server to differentiate between different paths a user is drawing. This also applies for when using the fill tool as well. As a result, the `Game` will attempt to find the most recent “move” index by traversing through the list from the end to the start. If discovered, it will remove everything from that index to the end of the array, thereby removing the most recent action. This is then emitted to all of the users, which will override the current drawing and re-render the new drawing history.

7.2.3 Remove Player

GameDispatcher

```
async removePlayer(socket) {  
  // Check if the lobby exists  
  if (this.checkExists(socket.lobbyId)) {  
    // Check if the user being removed is from the "active socket" to avoid duplicate usernames and sockets  
    if (this.games[socket.lobbyId].activePlayer(socket, socket.username)) {  
      // Remove user and update the lobby  
      await this.games[socket.lobbyId].removePlayer(socket.id);  
      socket.leave(socket.lobbyId);  
      const playerList = await this.games[  
        socket.lobbyId  
      ].getPlayerAndPoints();  
      this.io.to(socket.lobbyId).emit("set-players", playerList);  
      // Ensure there is still enough players  
      if (  
        (this.games[socket.lobbyId].players.length === 1 &&  
          this.games[socket.lobbyId].state === "drawing") ||  
        this.games[socket.lobbyId].players.length === 0  
      ) {  
        await this.games[socket.lobbyId].notEnoughPlayers();  
        this.deleteGame(socket.lobbyId);  
      }  
    }  
  }  
}
```

Figure 42. *removePlayer(): GameDispatcher*

The process of removing a player from a lobby stem across the various objects – from the GameDispatcher to the individual Round component. Within the GameDispatcher, it will first check if the game exists. Furthermore, another issue that once existed with my game was if a user is re-joining from another location, it would then kick the new location user rather than the old one. To mitigate this problem, my code will check if the user leaving is from an “active socket” whereby the socket is associated with a username, else it will not proceed. If the socket is associated with a username and that username is in the lobby, then it will remove the player. This is done by calling the Game removePlayer method, alongside updating the rest of the lobby to show that the user is now gone from the player list. By calling socket.leave(lobby id), this means that future socket messages will no longer be sent to this user, hence avoiding issues where the user joins another session.

Another concern when a player is leaving the lobby is to ensure there is enough users in the lobby, especially if the state is “drawing”. As a result, my code will then check for if there are 0 players, or if the state is “drawing” (hence active) and there is only 1 player in the game session. If true, then the game is stopped, as there is a minimum of 2 users required to proceed.

Game

```

async removePlayer(socketId) {
  // Remove player from the list of players
  const locatedPlayer = this.players.find(
    (user) => socketId === user.socket.id
  );
  // Update their points
  await this.updateIndividualPoints(locatedPlayer);

  if (this.round) {
    if (this.round.getCurrentDrawer().username === locatedPlayer.username) {
      this.timer = 5;
    }
    this.round.removePlayer(locatedPlayer.socket.id);
  }
  this.players = this.players.filter(
    (player) => player.socket.id !== socketId
  );
  // Check if the host needs to be re-assigned
  if (this.host.id === socketId && this.players.length > 0) {
    this.host = this.players[0].socket;
    this.icon = this.players[0].icon;
    this.io.to(this.id).emit("set-host", this.players[0].username);
  }
}
}

```

Figure 43. `removePlayer()`: Game

Within the Game object, using the socketID parameter, the algorithm will first search for a user located in the lobby of matching socketID. Once located, if the user was drawing, then it will remove them from the game session and go the next user's turn. Furthermore, if the user that left was the host, it will then re-assign those privileges to next user in the list (the one who joined immediately after the host).

Round

```

removePlayer(socketId) {
  const index = this.players.findIndex(
    (player) => player.socket.id === socketId
  );
  if (index >= 0) {
    const user = this.players[index];
    if (user && user.hasDrawn) {
      this.drawingIndex -= 1;
    }
    this.players = this.players.filter(
      (player) => player.socket.id !== socketId
    );
  } else {
    console.log("Cannot find user");
  }
}
}

```

Figure 44. removePlayer within Round

Within the Round functionality, an important aspect of this is to determine which users are available to draw and which are not. As a result, if the user that is leaving has drawn, then the variable tracking this information (drawingIndex) must be updated and decremented to show this change, before then removing the user from the list of players.

7.2.4 Joining A Game

GameDispatcher

```
async joinGame(lobbyId, socket, username) {  
  // Check if username exists  
  if (this.checkExists(lobbyId)) {  
    socket.username = username;  
    socket.lobbyId = lobbyId;  
    socket.points = 0;  
    socket.join(lobbyId);  
    // Add user & initialise state and values  
    await this.games[lobbyId].addPlayer(socket, username);  
    this.io  
      .to(lobbyId)  
      .emit("set-players", await this.games[lobbyId].getPlayerAndPoints());  
    this.games[lobbyId].initialiseState(socket);  
  } else {  
    socket.emit("invalid-game");  
  }  
}
```

Figure 45. joinGame(): GameDispatcher

When a user joins a game, this request is sent to the Dispatcher object (following the design pattern). The dispatcher will first check if this lobby exists. If the lobby does not exist, the WebSocket will receive a message, and through conditional rendering will highlight this to the user. On the other hand, if the lobby does exist, their socket will be updated with the required information such as their username, lobby, and points. The request is then sent to the necessary Game object, where the state of the session is then initialised.

Game.js

```

async addPlayer(socket, username) {
  try {
    // Check if user is already in the lobby
    const indexPlayer = this.players.findIndex(
      (player) => player.username === username
    );
    if (indexPlayer !== -1) {
      // Update their information if the user is rejoining from another device
      if (this.host && this.host.username === username) {
        this.host = socket;
      }
      if (this.round) {
        this.round.updateUserSocket(socket, username);
      }
      // Kick the old lobby device out
      this.players[indexPlayer].socket.emit(
        "kicked",
        "You have joined from another device!"
      );
      this.players[indexPlayer].socket = socket;
    } else {
      // Add the user's information into their Object
      const icon = await fetchUserProfilePicture(username);
      const info = { socket, username, points: 0, icon };
      this.players.push(info);
      if (this.round) {
        this.round.addPlayer(info);
      }
      if (!this.host) {
        this.host = socket;
        this.icon = icon;
      }
    }
  } catch (error) {
    console.error("Unable to add player to the lobby");
  }
}

```

Figure 46. addPlayer(): Game

From the Game object, the first check is ensuring this username is not already in the game session. If the user was already in the game from another device or tab, then their properties will be updated by updating their socket to the new one joining the session. If the user was drawing, they will now be able to draw from their new tab / device. Similarly, the host privileges will also be passed over as well. As for the other device, it will be kicked out of the lobby.

On the other hand, if the user is joining for the first time with no active account present already, their information will be fetched from the database before inserting them into the game and storing in the list of players.