**Full Unit – Interim Report**

GUESS MY DOODLE

ADVANCED WEB DEVELOPMENT

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

[Introduction 3](#_Toc152762075)

[Project Details & Motivations 3](#_Toc152762076)

[Aims & Goals of The Project 4](#_Toc152762077)

[Objectives - Milestones Summary 4](#_Toc152762078)

[Use Cases 5](#_Toc152762079)

[Planning, Timescale & Summary of Completed Work 7](#_Toc152762080)

[Web Frameworks 8](#_Toc152762081)

[States-of-the-art of Web Development 8](#_Toc152762082)

[Architectural Paradigms and Design Patterns 9](#_Toc152762083)

[Security 11](#_Toc152762084)

[Considerations 11](#_Toc152762085)

[Software Engineering 12](#_Toc152762086)

[Methodology 12](#_Toc152762087)

[Testing 12](#_Toc152762088)

[End System Development 14](#_Toc152762089)

[Running The Application 14](#_Toc152762090)

[Work Log 14](#_Toc152762091)

[Potential Future Enhancements 14](#_Toc152762092)

[Bibliography 14](#_Toc152762093)

* UML Sequence Diagram
* Modify UML to include Round class

# Introduction

## Project Details & Motivations

Websites are a key aspect of modern society. They have replaced parts of day-to-day lives by removing obstacles such as distance in order 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 and the possibility of adding your own custom words. 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.

When connecting to the website, the user is greeted with a page, allowing them to sign in or register an account. This is a requirement in order to proceed further into the website.

Upon signing in, they are then taken to the home page of the website. This will allow them to browse current public games that are being played. If they wish to join in on the session, they can simply click on the lobby and will be join it. In addition, on the home page, they are also provided the option of creating their own game, which can be either private or public. By default, all games are set to a private visibility.

When creating your own lobby, you are presented with settings to customise the gameplay, ranging from number of rounds to the amount of time each user should have to draw the word. Furthermore, users can join the lobby through distributing the specific URL provided, however they must be signed in to join it. Once they are ready to start, they can press “PLAY”, which will turn the lobby into a Game session.

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. 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 live while being drawn. If a user joins late to the game session, the drawing information is stored, and therefore the late user will be instantly caught up to date by displaying the current drawing on the canvas.

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 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.

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 to clear the canvas and an eraser tool to rub out aspects of the brush. Other potential implementations for added functionality may be a fill bucket and text tool, however, should they be implemented, will be placed in a shop section to be unlocked, providing an incentive for interacting with the website.

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

I was motivated to do this project as during lockdown, this was a very popular website to visit 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 liked 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 with friends.

Furthermore, when trying to join a public game, they do not provide the ability to select which lobby 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.

In addition, websites similar in nature to my project also lack capabilities such as making an account (e.g. skribbl.io make you write a name every time and select an avatar), 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.

## 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, or canvas-related drawings. 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.

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.

Finally, the last aim of project should be to allow the distribution of lobbies (or even joining game sessions late). This will be achieved through a custom URL link, containing the unique ID of the lobby they wish to join as a parameter, in order for 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.

## 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, and ensuring authentication to prevent unauthorised users from proceeding further into the website. 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) [16].

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 the login details, allowing them to proceed straight into interacting with the lobbies and 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 should be encrypted through salting and hashing, possible through the Passport package on Node.

Furthermore, through Axios, the front-end server 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.

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 allowing exclusive tools or changing the colour of their avatar card.

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 provided. As mentioned earlier, through Socket.io this will be transmitted in real-time.

## 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.

#### 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. Once they entered the necessary 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. |

#### 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 present a pop-up window for them, 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. |

#### 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 taken into it. |

#### 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 top 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. |

#### 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).> The URL will contain the custom lobbyID, only available to those who have received the link, making it a private game. |

#### Playing A Round

|  |  |
| --- | --- |
| Use Case | Playing A Round Within Lobby |
| Actor | 2x User with an account |
| Sequence | The first actor must create a private 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, the next person in the lobby then has to select a 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. |

#### 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. This will then display a confirmation dialog to confirm the user’s intent. Once confirmed, the points will decrement according to the sum of the purchase, and the user will have access to the item. |

## 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, which I will proceed to explain whilst evaluating my completed work.

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 as 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 in order 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 also 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.

Since I was not able to complete week 8 entirely, I elected to focus on the interim report and presentation, rather than delaying these two critical items. Upon completion of it, I will then revisit the content to be implemented in week 8. 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, allowing me to follow the intended timeline for term 2.

## Summary of Related Literature

#### 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.

#### 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.

#### 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 rash 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).

# Web Frameworks

## States-of-the-art of Web Development

In order to create my project, I am utilising React and Express.js for the back end and front-end frameworks 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 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, provide 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, such as a header or footer. 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.

Finally, I will be using MongoDB to store information as required on the NoSQL database structure [13]. I selected MongoDB due it’s 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.

## Architectural Paradigms and Design Patterns

Figure : Example Diagram

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 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 communication 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, 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 [14]. 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) [17]. 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.

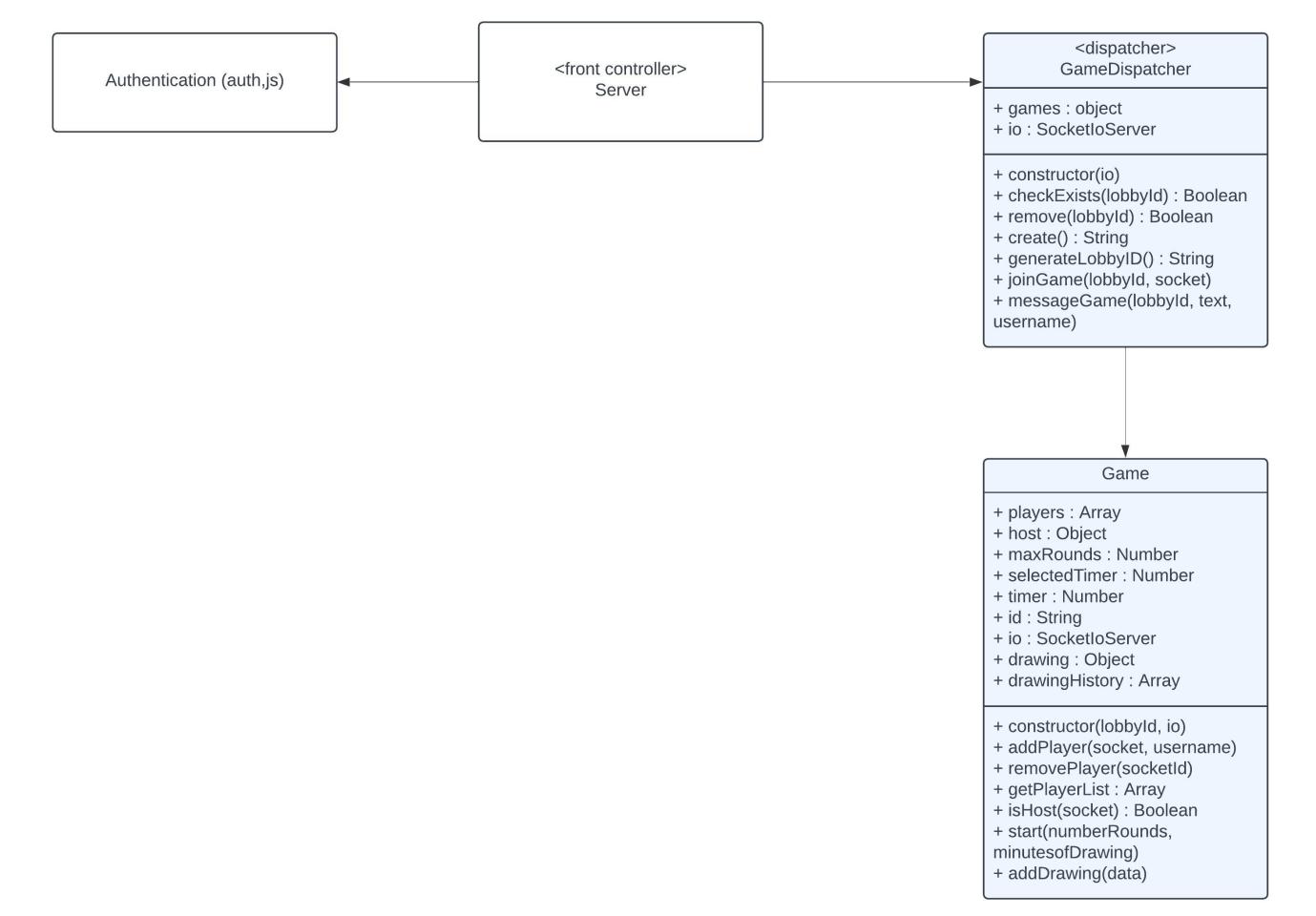


Image : Back-end UML Diagram

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 GameManager object. The GameManager class stores all the games that are currently running. When any game related socket transmission has occurred, the information is passed onto the GameManager 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 GameManager is a dispatcher, 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 it’s 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.

## Security

Security is an aspect of my project that is taken seriously, as it will store client sensitive information such as email addresses and passwords. To ensure client data is kept secure, when storing within the MongoDB database, I will utilise salting and hashing to encrypt the password. This will enhance the security of stored data, as it will provide an added layer of defence from attacks, specifically attackers trying to reverse-engineer passwords from hashed values. In the event of a breach, the plaintext passwords are thus still protected.

Furthermore, when registering, the user can also opt to use oAuth 2.0, using their Google account to sign in and sharing the necessary information. This allows users to avoid handing out details like passwords, as the oAuth service will provide the information required. This means that in the event of a breach, sensitive information is not accessible to the hackers.

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.

## Considerations

Aspects of my project that I am still considering are deployment for the website. Upon further research, I have investigated deployment applications such as Heroku to upload my project onto and host the website [10]. This will ensure the website is constantly accessible all the time. Alongside hosting the website, I would also need to ensure that the MongoDB database is accessible too, which can be achieved through MongoDB Atlas, a cloud solution to host the MongoDB database to deploy and run.

In addition, I must also consider aspects of colour theory within my website [5]. The colours chosen will be specifically elected to create a fun and exciting environment. This will allow me to target my specific demographic. To do so, they will be kept simple, as having understood the application of colours in websites, too many contrasting colours will create a chaotic outlook.

# Software Engineering

## 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, the user 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 improve structure, remove redundant lines of code, or improve efficiency / readability. Refactoring will enable me to product high-quality code that is simple to understand and tailored towards the test case created.

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 back 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.

## 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 important 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 [11]. 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. Currently, in my work I have used this to create register, logging in, checking authentication status, and managing socket communication for the canvas.

To create my front-end unit tests, I will be utilising React’s testing library and Jest [12]. 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 the HTML5 Canvas [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. Currently, through these libraries, I have created HomePage, Canvas and ChatBox front-end unit tests.

In addition, I will create end-to-end tests for my project, as this will test for how my front-end will interact with my back-end server. To do so, I will use Cypress to simulate interaction with my front-end website’s user-interface [15]. Although I have not currently implemented any end-to-end tests, I do intend on adding this once the functionality of creating individual lobbies and entire games has been created, as otherwise refactoring multiple times will add new functionality, requiring me to add new steps for the end-to-end testing.

# End System Development

## Running The Application

In order 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. Open a terminal up within the git 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 run the back-end server, simply type into the back-end folder terminal

**Nodemon server.js**

During development, Nodemon is incredibly useful to restart the server if there have been any files changes within the directory of the back-end server, therefore any modifications to the server code will automatically restart it.

To run the front-end server, simply type into the front-end folder terminal

**Npm start**

This will run the website and the back-end server to allow for interaction between the two. Currently, I am using the MongoDB on my local device as a temporary solution, therefore anyone attempting to deploy the website will struggle to login / register (unless they have it installed on their local machine), however once I move the database over to the cloud solution, it should provide a fix this issue.

## Work Log

#### Copy of Diary.MD

Diary

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

- Initalised 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

- Commited gitignore file to avoid commiting 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

## Potential Future Enhancements

At this stage in my development, 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 different options available to the user and reducing the need to store sensitive information on the server’s database.

# 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=qcsoLHusAFsC&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 my 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] Heroku. (2022) Heroku Documentation <https://devcenter.heroku.com/>

Heroku is valuable to my project as it will allow me to deploy my website onto their services. As a result, it will allow me to focus on developing good quality code whilst deploying it online for everyone to use. This will create increased accessibility, ensuring everyone who wishes to join, can do so. The documentation will allow me to view instructions on how to move my website to their services.

[11] 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.

[12] 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.

[13] 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.

[14] 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 work through a Dispatcher pattern, rather than using an array of items.

[15] 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.

[16] 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 is not applicable in solutions that require real-time response. It’s bidirectional communication 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.

[17] 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.

[18]