**Full Unit – Interim Report**

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](#_Toc962935995)

[The Problem 4](#_Toc1620832079)

[Aims & Goals Of The Project 4](#_Toc882683784)

[Survey of Related Literature 4](#_Toc231810397)

[Objectives - Milestones Summary 4](#_Toc86973239)

[Rationale 4](#_Toc1554579619)

[Web Frameworks 4](#_Toc1982774264)

[States-of-the-art of Web Development 5](#_Toc853867746)

[Architectural Paradigms and Design Patterns 5](#_Toc730127822)

[Security 6](#_Toc161287909)

[Considerations 7](#_Toc1430441437)

[Software Engineering 7](#_Toc1238322364)

[Methodology 7](#_Toc1476584651)

[Testing 8](#_Toc1446497703)

[End System Development 8](#_Toc2016509498)

[Architecture of End System 9](#_Toc603523012)

[Features of the End System 9](#_Toc1515644802)

[Running The Application 9](#_Toc386202859)

[Work Log 9](#_Toc853951646)

[Potential Future Enhancements 9](#_Toc46338867)

#### Early Deliverables

1. A report describing the state-of-the-art of Web development, including technologies/frameworks/platforms and their comparison as described above. The report will also explain the concrete choices made and the respective rationale/justification from a software engineering perspective (i.e. a justification merely based on a skills learning perspective will not suffice)
2. A report on Web development architectural paradigms and applicable design patterns, including an initial discussion of security, privacy and key-operational aspects and considerations (e.g. cloud deployments and DevOps aspects)
3. A report describing the system to be implemented, including a comprehensive set of well-defined use cases (or user stories).
4. A functional prototype implementation of the system, clearly demonstrating sound design principles and implementation practices (e.g. on DB design, on UI design and interaction).

Cited from the Project List

The (first term) written reports will be expected to contain the following contents:

* State of art web development
  + React
    - Front end
    - Render components and for navigation and router
  + Express
    - Back end
    - Create server to handle HTTP requests
    - Also use server for socket.io communication
    - Socket.io communication is used for the canvas and communicating lobby functionality to the rest of the game
  + Axios
    - Used to handle HTTP requests from the front-end to transmit it to the back-end
  + Socket.io
    - Used to handle quick communication between hosts and server
    - Transmit lobby information and update game
* aims, objectives and literature survey;
  + Aims & Goals
    - Aim focuses on purpose and “why” of object
  + Objectives – Milestones Summary
    - Emphasis actions and “what” to accomplish
* Architectural pattern
  + Client rendering
* Design patterns
  + Can I call server a singleton?
  + Factory
    - Client creates room
    - Send information to server
    - Server creates Game
* Security
  + Socket only transmits to people that have joined the lobby
    - Will require people to sign in
  + Storing accounts will be with salting and hashing if they register with email and password
  + Saving user state with cookies
  + Option to sign in with oAuth 2.0, therefore their details will be safe
* Considerations
  + Deploying online through Heroku
* planning and time-scale;
* summary of completed work;
  + Home Page allows for creating a game
    - Does not have functionality for browsing public games yet
  + Login Page & Registration completed (bar oAuth 2.0)
    - Encrypted through salting and hashing
  + Canvas can draw and be transmitted to other users
    - Line thickness & Colour too
  + Lobby can make settings
  + Chatbox to communicate with other users
  + No game implementation yet with timer
* Use Cases
  + https://www.usability.gov/how-to-and-tools/methods/use-cases.html
  + Registering In
  + Logging In
  + Browsing and Joining Public Games
  + Creating A Private Game
  + Joining A Private Game
  + Playing A Round
* bibliography and citations;
* End System Development
  + Features of End System [Proof of Concept / Milestone]
  + Running the Application
    - Npm init -y
    - Nodemon server
    - Npm start
  + Work Log
    - Diary??
  + Potential Future Enhancements
    - Display error messages as a modal
* Software Engineering
  + Methodology
    - TDD
  + Testing
    - Create unit tests and then make code to pass tests
    - Refactor

# Introduction

## Project Details

## Aims Of The Project

## Objectives - Milestones Summary

## Use Cases

#### 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 it’s 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 |
| 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 in to 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 in to 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 ran 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. |

# 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. **MERN STACK?** 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.

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. 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. 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.

**MAYBE TALK ABOUT HOW THEY WILL INTERACT HERE?**

## Architectural Paradigms and Design Patterns

Figure 1: 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 be in charge of 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 thee back-end server. From here, the request is processed, and the necessary information is collected from the database server to approach a final decision.

**UML DESIGN PATTERN?**

A design pattern I will be using in my project is a Factory creational pattern. This will be used when the client is creating a new game. Once the user has sent the request to the back-end server, the Server will process this request and create a new “Game” class. This new class will now handle the user’s socket communication, all lobby functionality, and update the gameplay as required.

## 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 looked into deployment applications such as Heroku to upload my project onto and host the website. 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.

# 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. **Possible reference to SuperTest documentation here**. 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. 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 work 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 **POSSIBLE REFERENCE TO JEST DOCUMENTATION**. 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. 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

#### Diary.MD Copy

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

## 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 opened up to allow for further services such as Apple, Facebook, X 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.