VIETNAM ACADEMY OF SCIENCE AND TECHNOLOGY

UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

BACHELOR THESIS

Online chess web game

Major: Information and Communication Technology

Member: Ly Tran Gia Minh – 22BI13288

Supervisor: Mr. Huynh Vinh Nam

July 2025

Contents

DECLARATION

I declare that the thesis is entirely my own under the guidance of my supervisor, Mr. Huynh Vinh Nam. I certify that the work and the results are totally honest and unprecedented previously published in the same or any similar form. In addition, all assessments, comments, and statistics from other authors and organizations are indicated and have been cited accordingly. If any fraud is found, I will take full responsibility for the content of my thesis.

Hanoi, June 2025

Student

Ly Tran Gia Minh

ACKNOWLEDGEMENT

My thesis would not be completed without the support of many people I wish to express my thankfulness.

Firstly, I would like to sincerely thank my project supervisor, Mr. Huynh Vinh Nam, for his enthusiastic guidance. Not only that, but he is also the person who suggest the idea for my project – a web chess game. I would also be pleased to extend my deepest gratitude to all the professors, staff, and friends that supported me in the past three years. Finally, I want to express my thanks to all my family, who gave me the opportunity to join USTH and learn precious knowledge.

List of Abbreviations

PVP Play vs Player
URL Uniform Resource Locator
Vs versus
AI Artificiel Intelligence
ID Identification

List of Figures and Tables

Chapter 1

Introduction

1.1 Context

Nowadays, people may find many ways to entertain. For example, there are many activities such as watching movies, playing sports, or playing board games. We know chess as the popular two-player board game of strategy with 64 squares arranged in an 8x8 grid. The goal of the game is to checkmate the opponent's king, which means threatening it with an inescapable capture.

Inspired by Mr. Huynh Vinh Nam, I could make up my mind with stimulating a web chess game. My aim is to make a system based on chess.com, a famous online chess platform and focus on matching systems, storing user data.

1.2 Motivation

Firstly, my motivation is to make a kind of entertainment for people. Through joining in a chess game, people can put away their stress and focus on the game.

Secondly, I hope to popularize the trend of this game. Moreover, the more people find interest in chess, the more opportunities for them to participate in outdoor events such as chess tournaments or chess

clubs. This helps people connect to others and improve social connectivity.

Thirdly, this project provides me with an opportunity to apply the knowledge I have studied during my time in USTH. The game system integrates various fields of knowledge, including databases, software engineering, and computer networks. The system is implemented using the JavaScript programming language.

Therefore, the goal of this is to develop a chess.com based system that helps people improve chess skills and interest; moreover, creates a space for entertainment.

1.3 Report Structure

This thesis presents the comprehensive development process of a web chess game system. Firstly, **Chapter 1: Introduction** outlines the context and significance of the trend of chess game, as well as the motivation for implementing the based system. Secondly, **Chapter 2: Objectives** define the goals of the project, identify the system requirements, and provide an overview of the key functionalities alongside the anticipated outcomes. Thirdly, **Chapter 3: Requirements Analysis** elaborates on the functional and non-functional requirements of the system, presenting relevant use cases and possible user interaction scenarios. Next, **Chapter 4: Methodology** describes the selected tools, technologies, and development strategies, along with the reason behind their adoption. Then, **Chapter 5: Result and Discussion** presents the implemented features, evaluates system performance, and discusses the limitations and challenges encountered during development. Finally, **Chapter 6:**

Conclusion and Future Work summarizes the findings of the thesis and proposes potential directions for enhancing and extending the system in future research.

Chapter 2

OBJECTIVE

1. Project Objective

This project aims to develop my software engineering skills, from analyses the problem to analyses the problem, implement system. So, I decided to design and implement a web chess based system that facilitates gameplay and match results for players. The system is expected to improve the efficiency of matchmaking, reduce latency and record the winner and loser of each match. The main objectives include:

1. Develop a room forming with a room ID for another player to join
2. Implement core features such as:

- User authentication and security: Account creation, login, password reset.
- Data management: Tools for managing user information, room ID, and match records.
- Gameplay operations: Supports the movements of chess and activities of the users.

3. Provide match record system: Match History and Leaderboard help players to see the past matches and ranking due to number of winning matches.
4. Provide settings for client: Players can switch language, theme and board size.

2. Desired Feature:

The online chess web game is designed for only one role in the system: Player. The role has access to specific functions on the platform as described below:

1. Authentication Features
    a. Register: Users who have never joined in the system have to create a new account.
    b. Login: System allows users to access.
    c. Forgot password: In case users forget the password, they can reset it.
2. Gameplay Features
    a. Play with Player mode: Player will go matchmaking with another random player who does the same.
    b. Play with Friend mode: Player can send a room ID after creating a room to who they want to competitive.
    c. Play vs AI mode: Player can play against AI to practice.
3. Match Record Features
    a. Match History: Players can see their past matches result.
    b. Leaderboard: Where system ranking players based on the number of winning matches.

3. Expected Outcome

The game system helps players have the best experience with the following key outcomes:

- A multi-mode selection gameplay
- Secure user authentication and data management
- A web system with very low latency
- Settings functionality to support player with convenience.

Chapter 3

REQUIREMENT ANALYSIS

In this section, we will provide a quick overview of the project's features as well as use cases and scenarios for the system. This section contains the full functional specification, including navigation paths showing the sequence of screens.

1. Overall System Requirements

In general, the chess game system must meet the following requirements:

- Authentication login system.
- User authorization and function system:
  + For administrators: Initialize data and manage the system database.
  + For players: Play different modes, view match history and leaderboard.

2. Functional Requirements

The chess game system is designed to serve 2 mains user roles: Administrators and Player. Each role has access to different features depending on their rights in the system.

2.1 Administrator Role

Function	Requirement Description
Data Initialization	The admin can manage master data such as account information, roomID and result.
Account Deletion	The admin can choose account to delete

Table 1: Functional Requirements of Administrator Role

2.2 Player Role

Functional	Requirement Description
Play vs Player	Players will play against a random player
Play vs Friend	Players can play with whom they want by sending roomID
Play vs AI	Players can practice with bot to practice their skills
View Match History	Players can see their past matches with other players
View Leaderboard	Players can see their ranking in the server

Table 2: Functional Requirements of Player Role

2.3 Server Functionalities

Functional	Requirement Description
User Management	The system must send and receive data through secured APIs
Game Logic	The system must enforce all the rules of chess
Matchmaking and Game Session Management	The system must handle all the functions such as creating room, automatic matchmaking, starting match when two players are paired and saving match result
Data Persistence and History	The system must save completed game to the server

Table 3: Functional Requirements of Server Functionalities

3. Non-functional Requirements

3.1. Performance Requirements

Non-Functional	Requirement Description
Latency	The application must respond to user actions within 2 seconds under normal conditions
Handle possibility	The game system must be able to handle a maximum of 100 concurrent players online at the same time

Table 4: Non-functional requirements of Performance Requirements

3.2. Availability and Reliability

Non-functional	Requirement Description
Connection	The game system must work with an internet connection, can be Wi-Fi or data cellular.

Table 5: Non-functional requirements of Availability and Reliability

3.3. Security Requirements

Non-functional	Requirement Description
Secure Password Storage	The password is stored as bcrypt hash, a very secure method.
Prevent Unauthorized Actions	Ensure a user is only performing actions they are allowed to

Table 6: Non-functional requirements of Security Requirements

3.4. Usability Requirements

Non-functional	Requirement Description
The system UI	Must be intuitive and easy to navigate, requiring no more than 3 minutes of training for first-time users
Error message	Must be clear, visible

Table 7: Non-functional requirements of Usability Requirements

3.5. Compatibility

Non-functional	Requirement Description
Device requirement	The desktop application must run on Win 11 and later

Table 8: Non-functional Requirement of compatibility

4. Use Case

4.1. Use Case Diagram

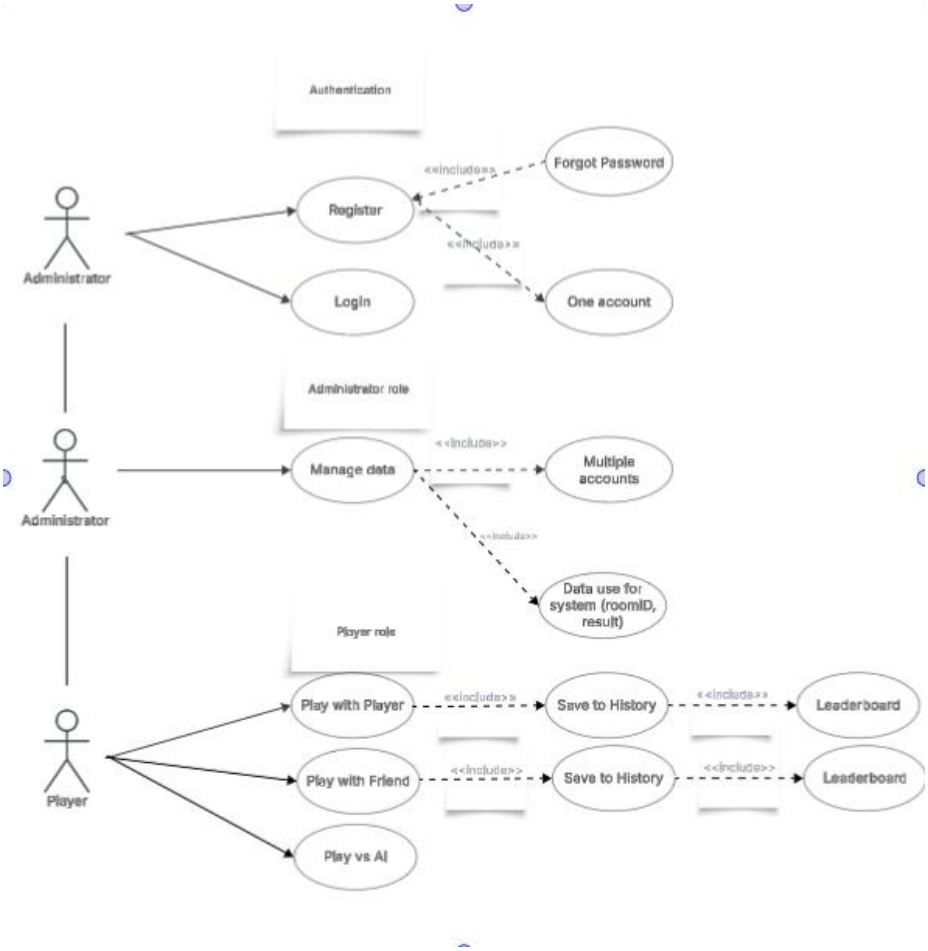

Figure 1: Use case diagram

Figure 1 shows the Use Case diagram that consists of all features of the system.

4.2. Users Characteristics

There are 2 types of users that interact with then system: Administrator and Player. Each type has different roles and responsibilities, so they have distinct requirements:

- Administrator: A user with full administration, responsible for managing the database of the system and players' account.
- Player: A user who can access the system to play games and see their playing information.

5. Use Case and Scenario Description

5.1. Use Case: Register

This use case allows a player to create a new user account in the system. Players are the only users authorized to register accounts, which administrators can only manage or delete.

Flow event of register:

Actor Action	System Action	Data
1. Players navigate to the “Create Account” section	2. The system requests the Players to fill in the required information: Username, Password, URL Avatar (if needed)	
3. The Player fills in the required information and click “Create’	4. The system checks if the account is ready in the Database and then requests exists account, else the system creates and saves the information of the account to the Database	- User name - Password - URL Avatar (if needed)

Table 9: Register Flow Event

Preconditions:

- Player does not have an account
- Player has Internet connection

Postconditions:

If the use case is successful, a new account is created and stored in the system.

5.2. Use case: Login

This use case describes how an actor logs into the system when a client app like the Administrator or Player has Internet.

The actor can be an Administrator, a Player.

Flow event of Login:

Actor Action	System Action	Data
1. The actor enters the account (username, password)	2. The system validates the username and password. If the username or password is invalid, the system displays an error message “Incorrect username or password”	- Username - Password

Table 10: Login flow event

Preconditions:

- The user account must be existed
- Player must connect to the Internet

Postconditions:

If the use case is successful, the user logged into the system. Otherwise, the system state is unchanged.

5.3. Use Case: Forgot Password

This use case describes how an actor reset their password if they have forgotten it.

Flow event of forgot password:

Actor Action	System Action	Data
1. The user selects the “Forgot Password?” option in the Login screen	2. The system prompts the user to enter their registered email address	- Email
3. The user provides the requested information	4. The system validates the input and checks if the account exists, and then the system sends a reset password link to the user’s email	- Email
5. The user enters the reset password link	6. The system redirects user to the “change password” process	- Email - Reset Password Link

Table 11: Forgot Password flow

Preconditions:

- Player has a valid email address
- Player has an existing account in the server

Postconditions:

If the use case is successful, the user can change the password. Otherwise, no changes are made to the account.

5.4. Use Case: Play with Players Mode

This use case describes how an actor goes matchmaking and joins in a match with another one.

Flow event of Play with Players Mode:

Actor Action	System Action
1. Player selects mode “Play new game”	2. The system address to the selection mode of chess: “Bullet”, “Blitz”, “Rapid”, or “Daily”
3. After selecting mode, Player is in queue and waiting for the match	4. The system finds another player who goes in the same mode and finds match to make pairs

Table 12: Play with Player event flow

Preconditions:

- Player must connect to the Internet
- Player account must be existed

Postconditions:

If the use case is successful, player can compel with another player. Otherwise, they are queued till another player joins

5.5. Use Case: Play with Friend

This use case describes how an actor invites their friends to join in the match with them.

Flow event of Play with Friend:

Actor Action	System Action
1. The Player chooses “Play with Friends” mode	2. The system addresses the Player to the selection “Join room” or “Create New Room”
3. If Player chooses to create new room	4. The system redirects Player to a playroom with the roomID that they can send to the other player
5. If Player chooses to join a room with ID room	6. The system joins Player in a exist room

Table 13: Play with Friend Flow

Preconditions:

- Player must connect to the Internet
- Player account must be existed

Postconditions:

If the use case is successful, two players can go for the match.

5.6. Use Case: Leaderboard

The use case describes how players can see their ranking in the game server.

Flow event of Leaderboard:

Actor Action	System Action
1. Player clicks to the "Leaderboard"	2. The system redirects to the "Player Leaderboard"

Table 14: Leaderboard Flow

Preconditions:

- Player must connect to the Internet
- Player account must be existed

Postconditions:

If the use case is successful, the Player can see their ranking with other players.

5.7. Use case: Match History

This use case describes how an actor can see their past matches.

Flow event of match history:

Actor Action	System Action
1. The player navigates to match history section	2. The system redirects the player to match history section

Table 15: Match History Flow

Preconditions:

- Player must connect to the Internet
- Player account must be existed

Postconditions:

If the use case is successful, the Player can see their past matches.

Chapter 4

Methodology

In this section, we describe the tools and techniques employed in the project, explain the reasons for their selection, and demonstrate how they were applied in practice.

1. Tools and Techniques

1.1. MongoDB

The project requires choosing a suitable database system. MongoDB and MySQL were considered based on their compatibility with the project's requirements.

Criteria	MongoDB	MySQL
Data Model	Collections with BSON documents (JSON-like). Data is stored in a flexible, hierarchical format	Tables pre-defined rows and columns. Data is highly structured.
Schema	Flexible Schema (Dynamic): No strict schema is required. Documents in the same collection can have different structures. Ideal for evolving applications.	Rigid Schema (Pre-defined): The structure of tables (columns and data types) must be defined before inserting data. Data enforces consistency
Query Language	MQL (MongoDB Query Language): Uses JSON-like query objects. Very intuitive for developers working with JavaScript/Node.js.	SQL (Structured Query Language): The industry standard for relational databases. Extremely powerful for complex queries and reports.
Scalability	Horizontal Scaling (Sharding): Designed for easy, native horizontal scaling by distributing data across multiple servers. Excellent for very large datasets.	Vertical Scaling: Traditionally scaled by increasing the power (CPU, RAM) of a single server. Horizontal scaling is possible but is often more complex to implement and manage.

Performance	Extremely fast for read/write operations on single documents and for large, unstructured datasets. Ideal when your primary access pattern is to retrieve a whole document.	Very high performance for complex queries involving multiple JOINs and aggregations on well-structured data. Optimized for transactional consistency.

Table 16: MongoDB vs MySQL

Based on the criteria of schema, query language, scalability, and performance, MongoDB was selected as the database system for this project. In MongoDB, a single chess can be naturally represented as a single document. It is a self-contained collection of related information, and that is precisely what a MongoDB document is designed to store.

To sum up, MongoDB was the best choice because it is perfect for calculating player statistics, leaderboards, and winning rates directly in the database.

1.2. Node.js with Express.js

This is the server-side brain in my application. It handles requests, manages game state, and communicates with the database. Node.js is a natural fit for MongoDB backend because both use JavaScript and JSON-like data structures (BSON in MongoDB's case). This synergy simplifies development. Express.js in a minimalist and

flexible framework for building the API endpoints, such as POST api/games to create a game or POST /api/games/:id/move to submit a move. When a request hits a route, the underlying power of Node.js is non-blocking, event-driven architecture becomes crucial; it efficiently handles asynchronous operations like querying the MongoDB database for a game's state or writing an update, all without halting the server.

1.3. Chess.js

This is a crucial technique: Do not write your own chess rules engine from scratch. It's incredibly complex. Chess.js and python-chess are two chess logic engines for chess game system; however, chess.js is suitable for JavaScript and Node.js than python-chess. So, I decided to choose chess.js, which is lightweight, fast, and comprehensive.

When a Player submits a move to my API, my backend code will fetch the game document from MongoDB. Next, the system will load the current_fen string into a chess.js instance:

```
const game = new Chess(fenString);
```

Then, the system attempts to make the move:

```
const result = game.move({ from: 'e2', to: 'e4' });
```

If the result is null, the move was illegal, the system rejects the player's request move. Else if the move was legal, get the new FEN:

```
Const newFen = game.fen();
```

The system will update MongoDB document with the new FEN and add the move to the moves array. Finally, the server can check for game over: game.isOver(), game.isCheckmate(), etc.

1.4. Chessboard.jsx for React

For the development of the frontend user interface, I choose the chessboard.jsx library, a powerful and modern React component designed specifically for rendering interactive chessboards. This choice was strategic, as it allowed me to abstract away the significant complexities of board rendering, piece movement, and user input handling. By adopting a declarative approach, our application's state, primarily the FEN (Forsyth-Edwards Notation) string representing the current board position, serves as the single source of truth.
The chessboard.jsx component declaratively renders the board based on this FEN string. When a user makes a move via its intuitive drag-and-drop interface, the component triggers a callback function that communicates the move to our backend for validation. Upon receiving a state update—either from the user's own confirmed move or a real-time update from an opponent via WebSockets—our React application simply updates the FEN state, and chessboard.jsx efficiently re-renders the board to reflect the new position. This technique not only streamlined development significantly but also ensured a robust, performant, and responsive user experience that is tightly synchronized with our backend game logic.

2. System Architecture

The project is built upon a modern, decoupled three-tier architecture. This is a standard and highly effective pattern for web applications. The three primary tiers are:

1. The Frontend (Presentation tier): The user interface that runs in the user’s web browser.

2. The Backend (Logic tier): The server-side application that contains the core business logic and API.
3. The Database (Data tier): The persistent storage for all game and user data

This architecture is “decoupled” because the frontend and backend are separate applications that communicate over a network via a well-defined API. This separation allows them to be developed, deployed and scaled independently.

2.1. The Frontend (Presentation Tier)

The Frontend constitutes the user-facing portion of the system, developed as a Single Page Application (SPA) using the React.js framework. Its primary responsibility is to render a dynamic and interactive user interface that runs entirely within the user's web browser. It leverages specialized components like chessboard.jsx to display the game board, pieces, and move history. For user-initiated actions, such as submitting a move, it employs an HTTP client like Axios or the native Fetch API to communicate with the backend's RESTful API. Simultaneously, it maintains a persistent WebSocket connection via the Socket.IO client library to receive real-time updates, ensuring the UI reflects opponent moves and other game state changes instantly without requiring a page refresh.

2.2. The Backend (Logic tier)

The Backend, or Logic Tier, acts as the authoritative brain of the entire chess application, built on the Node.js runtime with the Express.js framework. It serves as the single source of truth,

processing all business logic and ensuring the integrity of the game state. Through its RESTful API, it exposes secure endpoints for the frontend to consume. A critical function of this tier is server-side validation; it never trusts client input. Upon receiving a move, it utilizes the chess.js library to verify its legality against the official game rules before proceeding. The backend is the sole component permitted to communicate directly with the database, using an Object-Document Mapper (ODM) like Mongoose to orchestrate all data reads and writes. Furthermore, it manages the real-time communication layer, using its Socket.IO server to broadcast game state updates to all relevant clients after a successful action, completing the live gameplay loop.

2.3. The Database (Data tier)

The Data Tier provides the system's persistent storage layer and is implemented using **MongoDB Atlas**, a fully managed, cloud-hosted NoSQL database service. This tier is responsible for storing all application data, including users and games, in collections of flexible BSON documents. A key architectural decision enabled by MongoDB is the modeling of each chess game as a **single, self-contained document**. This structure is highly efficient, as it allows the complete history and state of a game to be retrieved in a single, fast database operation without requiring complex joints. By leveraging a managed service like Atlas, the architecture offloads critical infrastructure concerns such as scalability, backups, and security, allowing the application logic to remain focused on its core functionality.

3. Database Design

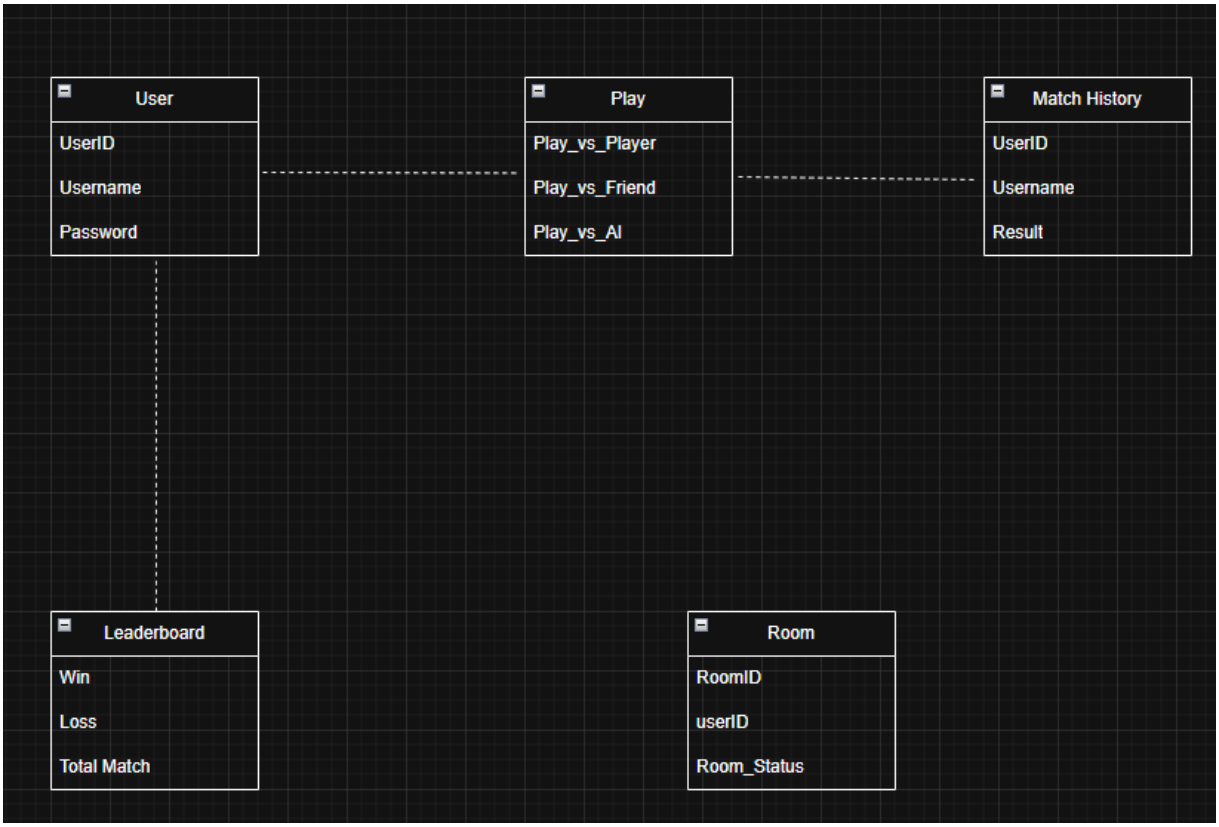

Figure 2: Overall Database Design

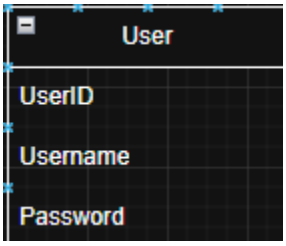

Table 17: User Table

The User Table stores information of Player. It includes UserID, Username, and Password.

Table 18: Play Function Table

The Play Table stores information about the Play Function. It includes 3 modes: Play vs Player, play vs Friend and play vs AI.

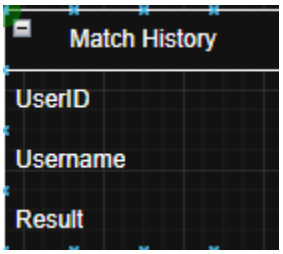

Table 19: Match History Table

The Match History Table stores information about Match History. This sections shows the UserID, Username and results of past matches.

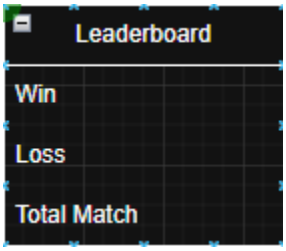

Table 20: Leaderboard Table

The Leaderboard Table stores information about Leaderboard. This function supports the winning matches, losing matches and total matches of players in the server.

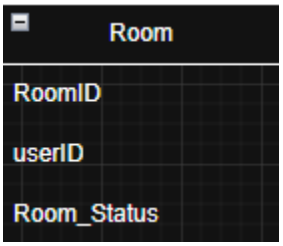

Table 21: Room Forming Table

The Room Forming Table stores information about Room Forming. It includes RoomID, UserId joining in the room and Room Status.

4. Use Case Implementation

4.1. Register

This use case describes how a user creates their accounts.

When the user access to the web, the server requests them to login. In case that they don’t have an account, they have to create a new one. The server requires the user to provide the username, password and URL Avatar (if necessary).

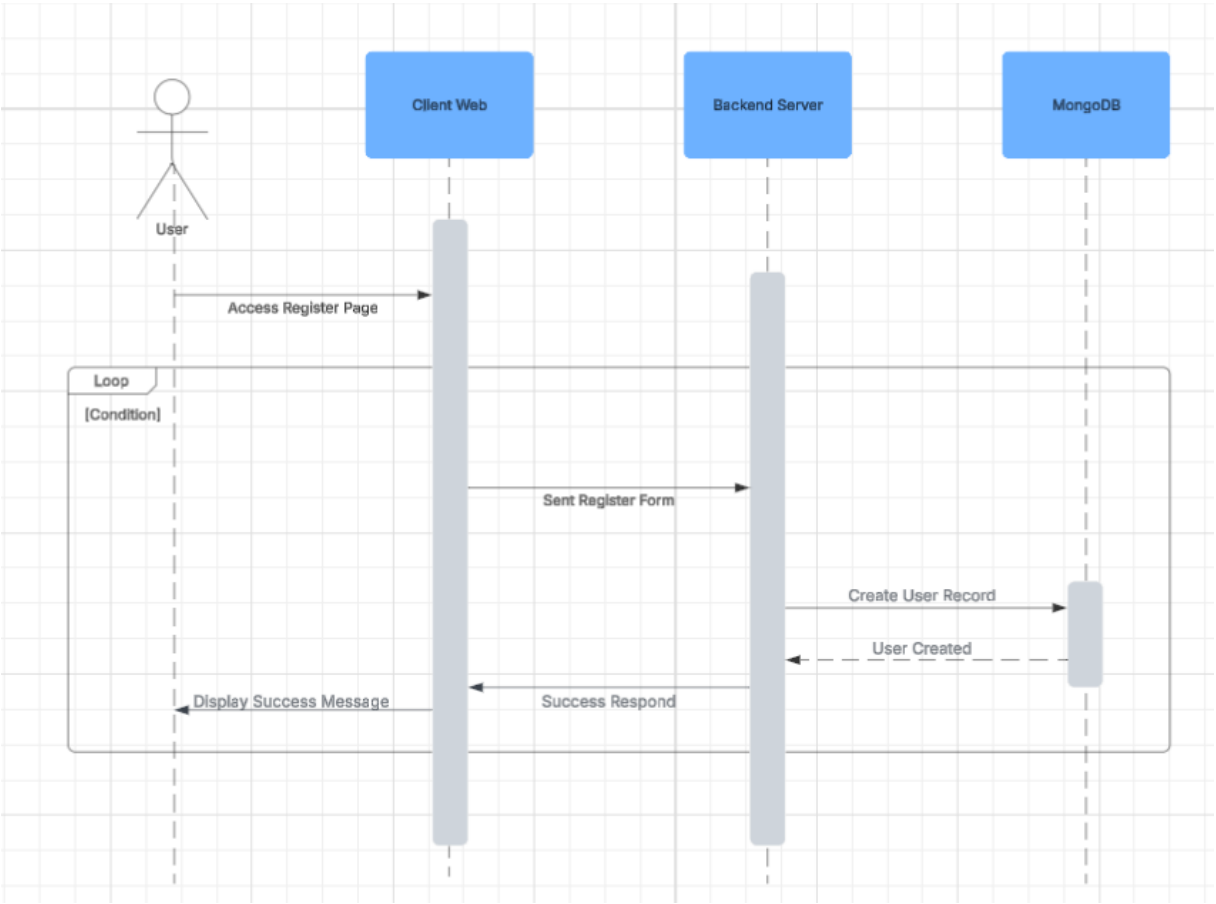

Table 22: Register Sequence Diagram

4.2. Login

This use case describes how a user logins to the server.

After creating the account, the system requests the user to login. Once the user submits the login form including the username and password, the web client redirects the user to the main menu

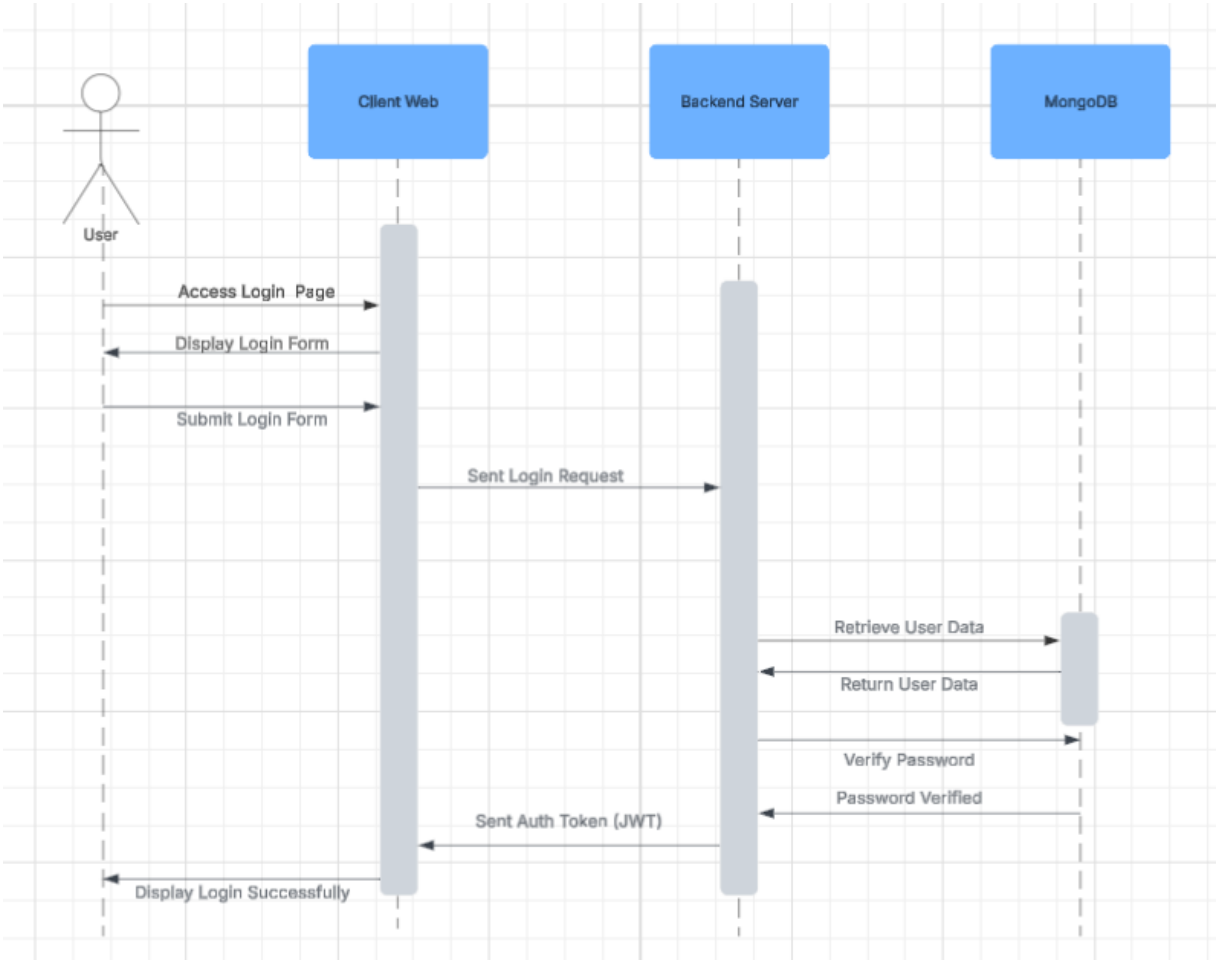

Table 23: Login Sequence Diagram

4.3. Forgot Password

This use case describes how a user resets their password after selecting “Forgot Password” option.

The user clicks “Forgot Password” on the login screen. The web client displays a form where the user enters their email address. This information is sent to the server in order that the server gives back to the user a reset password link.

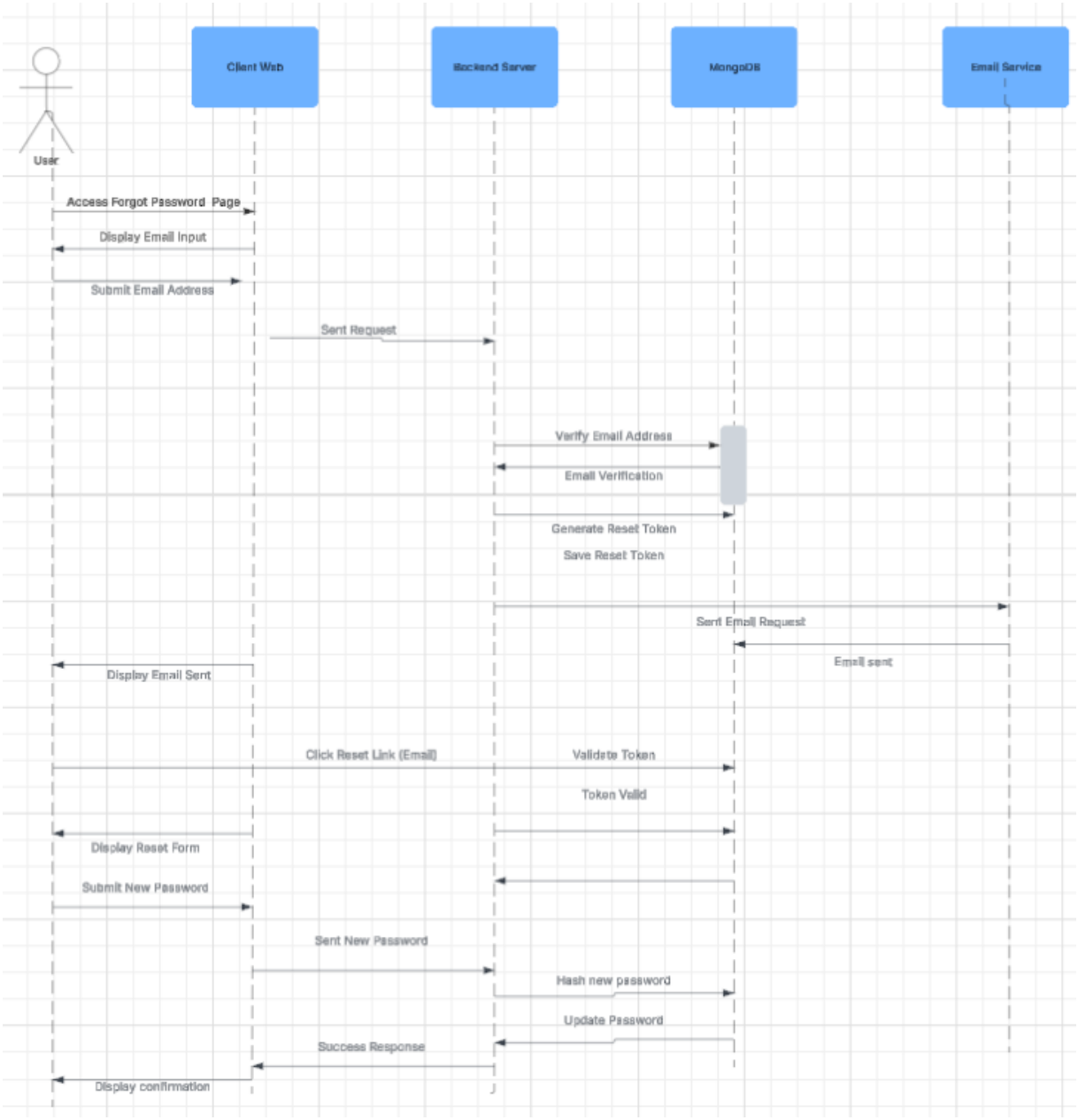

Table 24: Forgot Password Sequence Diagram

4.4. Play with Player

This use case describes how a user joins in the play with players mode

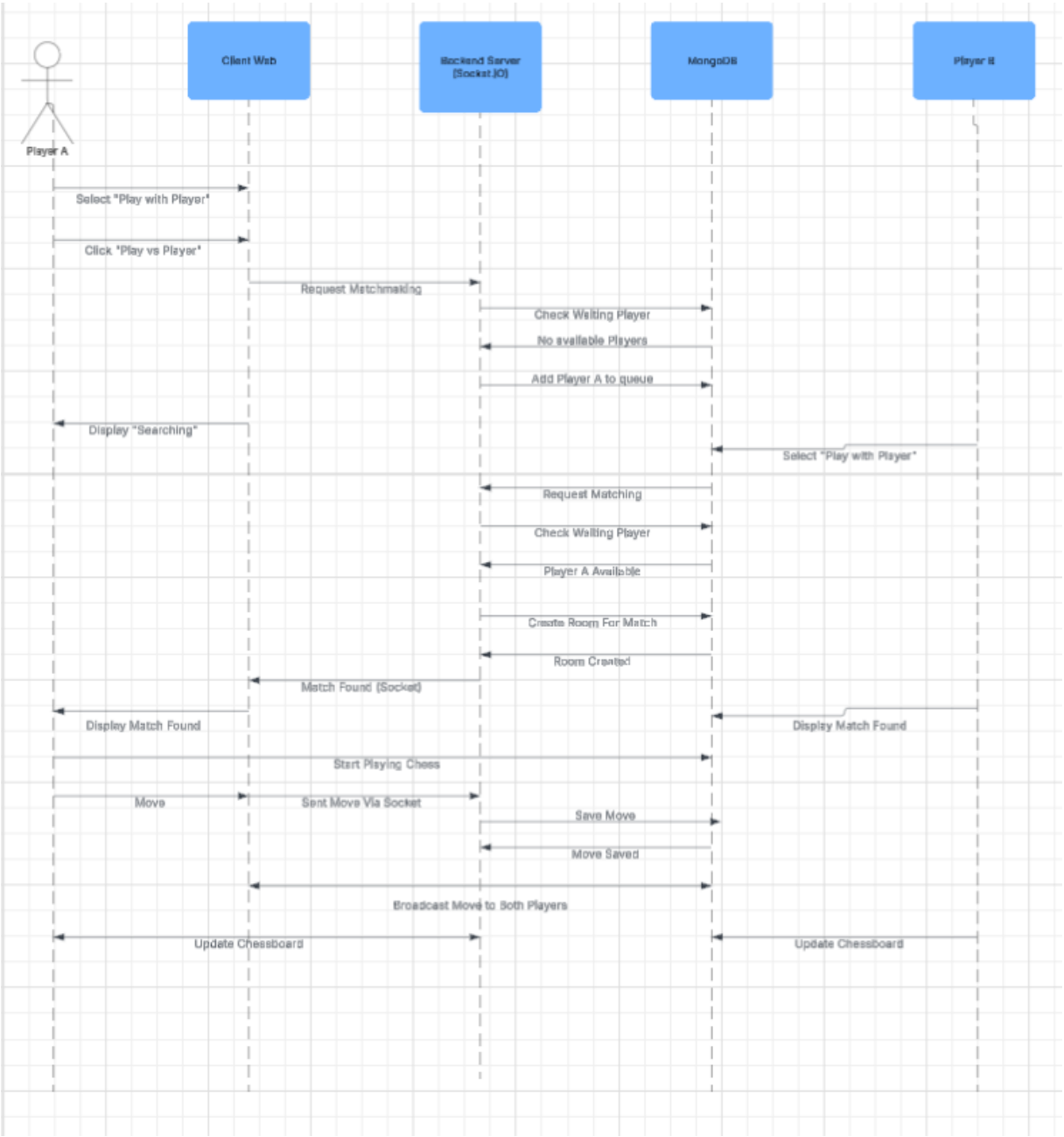

Table 25: Play with Player Sequence Diagram

4.5. Play with Friend

This use case describes how a user invites their friend to compel with.

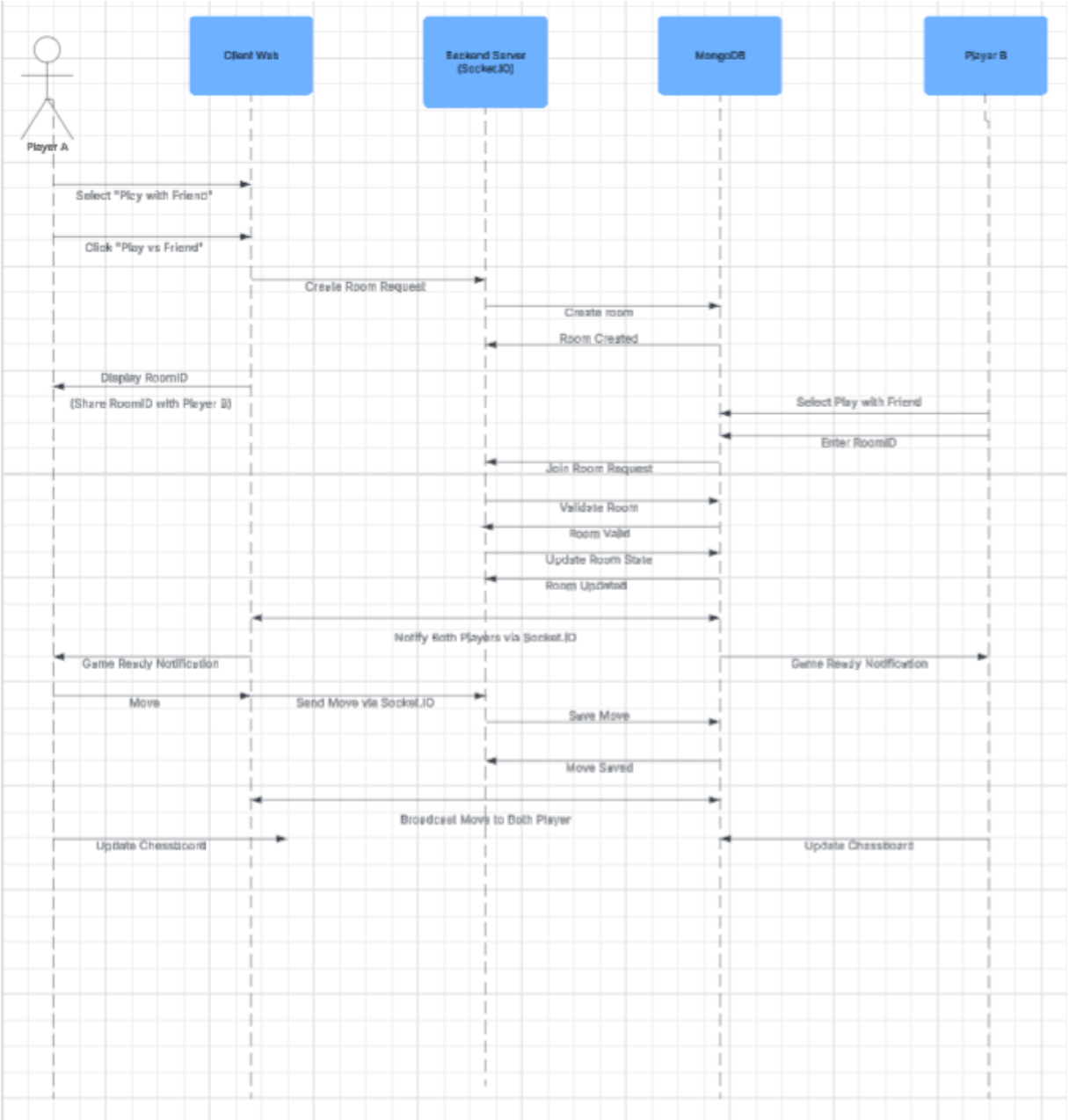

Table 26: Play with Friend Sequence Diagram

4.6. Play vs AI

This use case describes how a user selects and plays against AI.

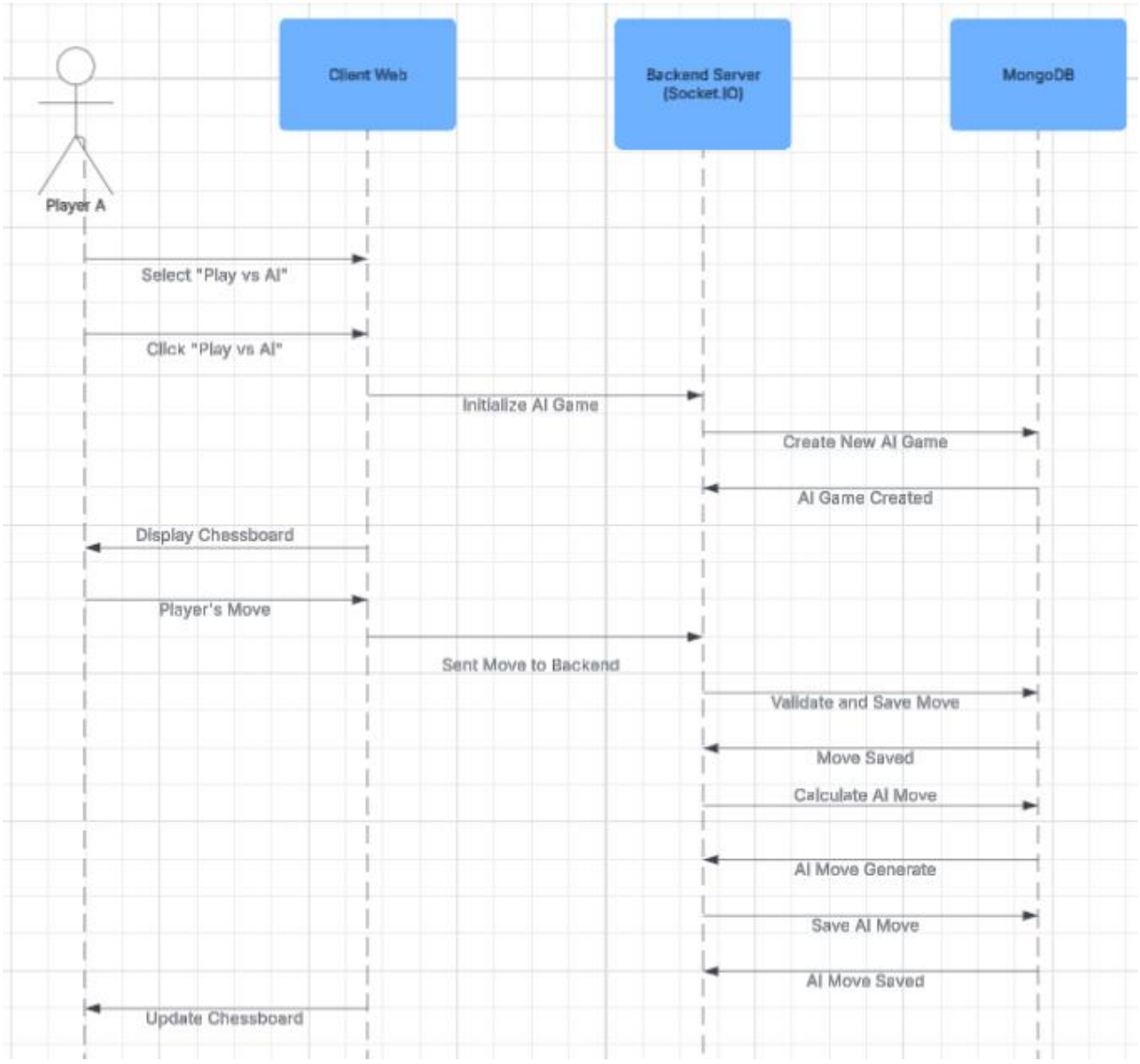

Table 27: Play with AI Sequence Diagram

Chapter 5

Result and Discussion

1. Result

In the web chess system, the main function have been successfully developed and implemented.

Authentication and Account Management:

- Users can login, change password, and recover forgotten passwords: Fully functional and stable.

Play function:

- Play vs Player: user matchmaking with a random player.
- Play vs Friend: user can invite friend to compel with
- Play vs AI: user practices against AI with three difficulties: easy, medium and hard.

Match Record:

- Match History: View player's past matches.
- Leaderboard: View ranking of players due to number of winning matches, losing matches, and total matches played.

The web chess game provides user with settings supporting language, theme, and board size changes. Moreover, the system has a small latency of delay, which makes player's experience are

uninterrupted. Beside, the system provides the surrender button for player in case they want to end the game earlier and they can send rematch request to the opponent.

2. Discussion

Despite being fully implemented, the system still has some trouble with the real-time update as the project was new to me. For example, the real-time communication seems to be unavailable or the fetch of leaderboard. However, everything else seems to be mostly perfect.

Chapter 6

Conclusion and Future Work

1. Conclusion

To sum up, the web chess online game system has successfully achieved the core objectives of the project. Although the system is not yet fully completed and still contains some pains, the essential components have been implemented and functional.

The system is built around a central server responsible for storing data and enjoying the game. I have developed a basic yet functional desktop application for players with simplified user interface to ensure accessibility while fulfilling the main goals of the project.

For the game system, there are some requirements for players to be able to join in the server. They have to connect to the Internet, which makes sure that they can connect to the server and other players.

Performance testing showed that the system can handle up to 500 simultaneous requests without causing system instability.

The main functions of the system are completed and operational:

- Players can create their account to join in the server and choose the game mode as they want.
- Administrator can manage data of players and matches.

2. Future Worl

To improve this chess game system, the following enhancements are recommended:

- Real-time updating and communication should be available.
- Advance the difficulties of AI which supports player in practice.
- More modes implemented to be more interesting, such as puzzle mode or spectating available matches.
- Tournament mode: 8 Players to find the best.
- Better background, which makes the game being more attractive.

Chapter 7

Appendix

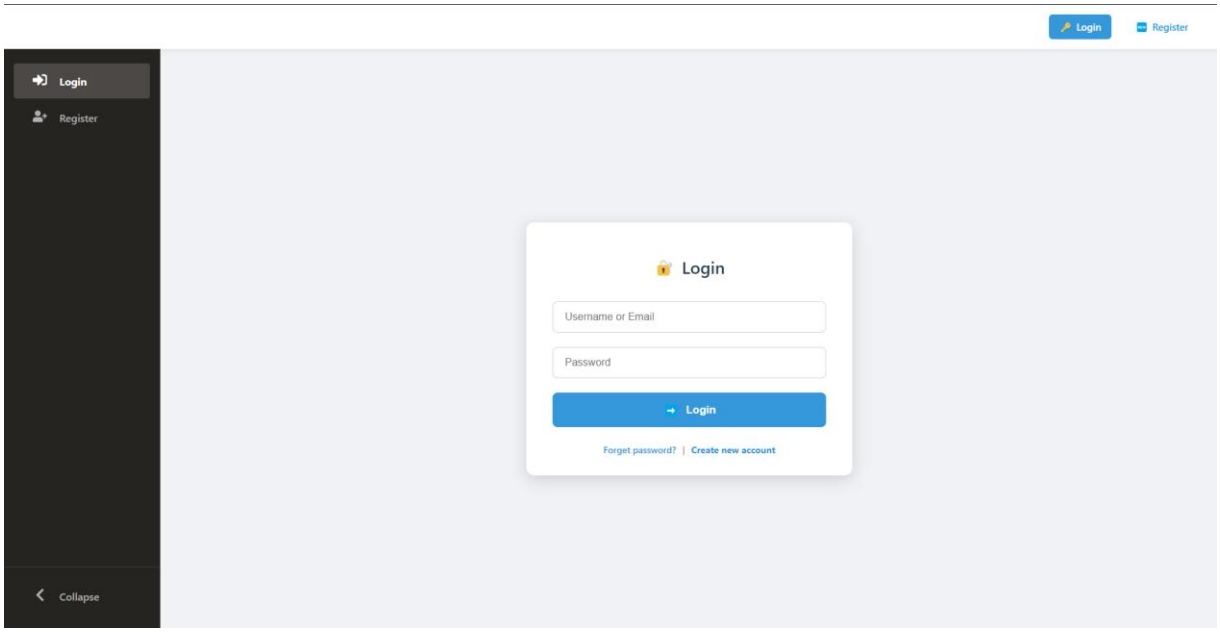

Figure 3: Form Login of the web client

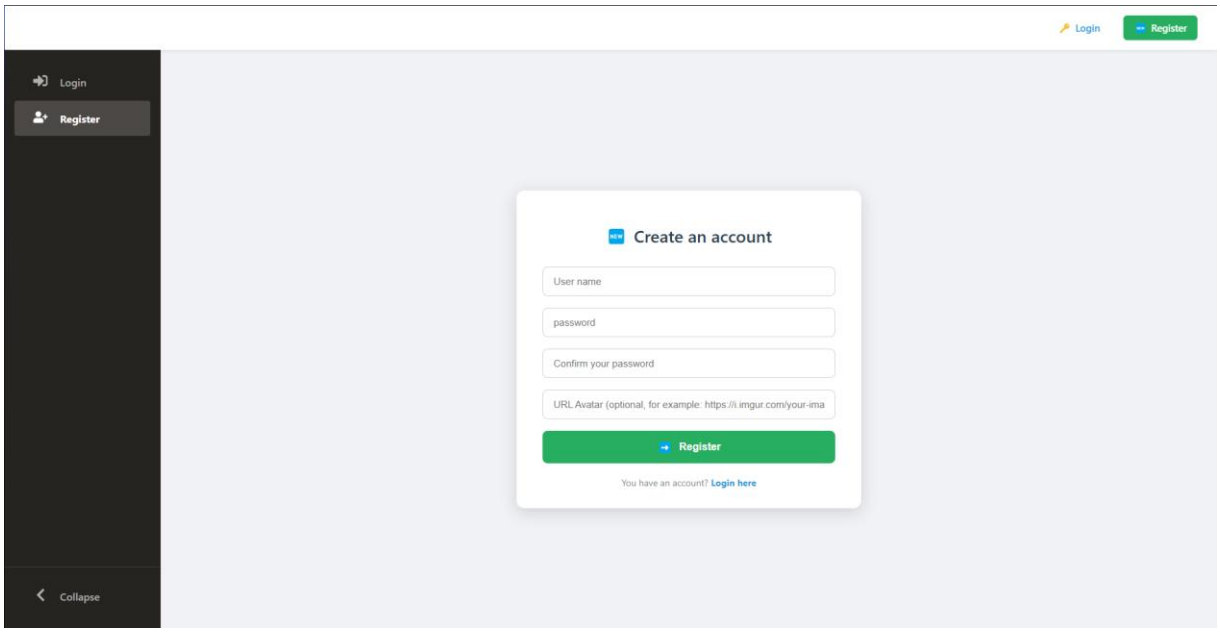

Figure 4: Register Form of the web client

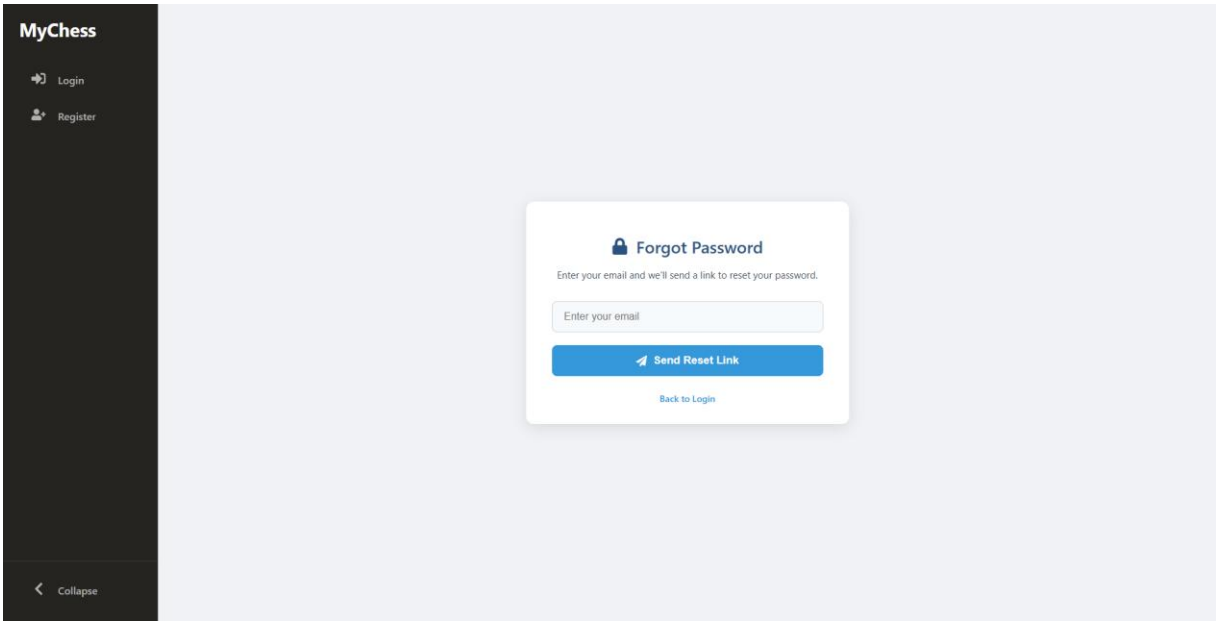

Figure 5: Forgot Password

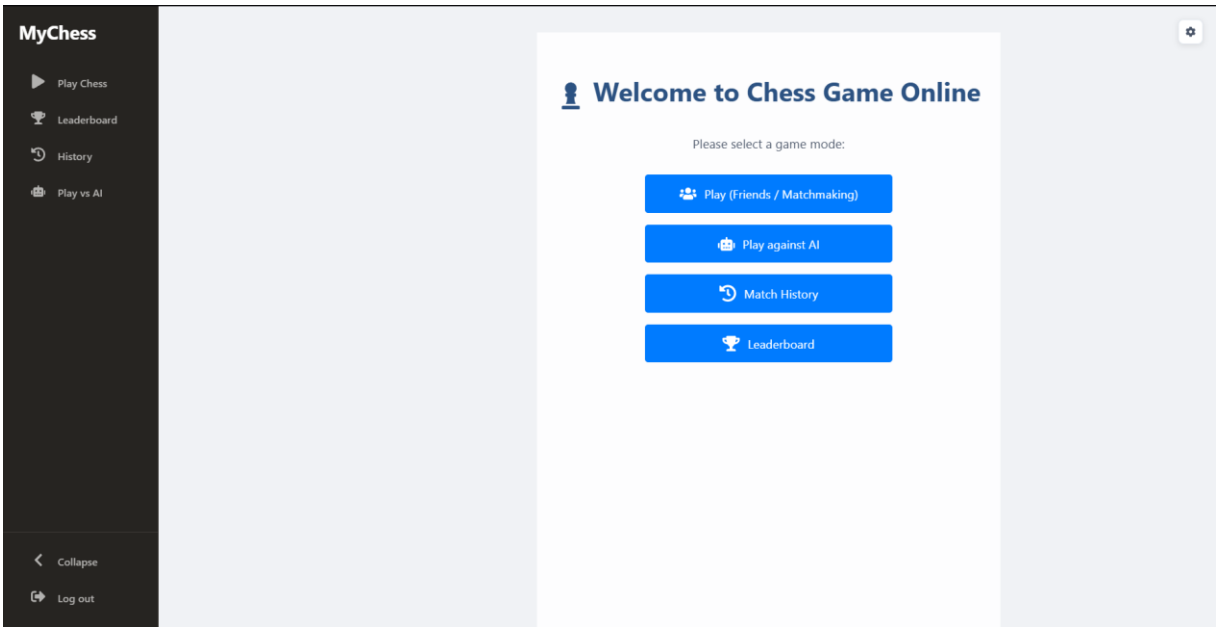

Figure 6: Main Menu of the web client

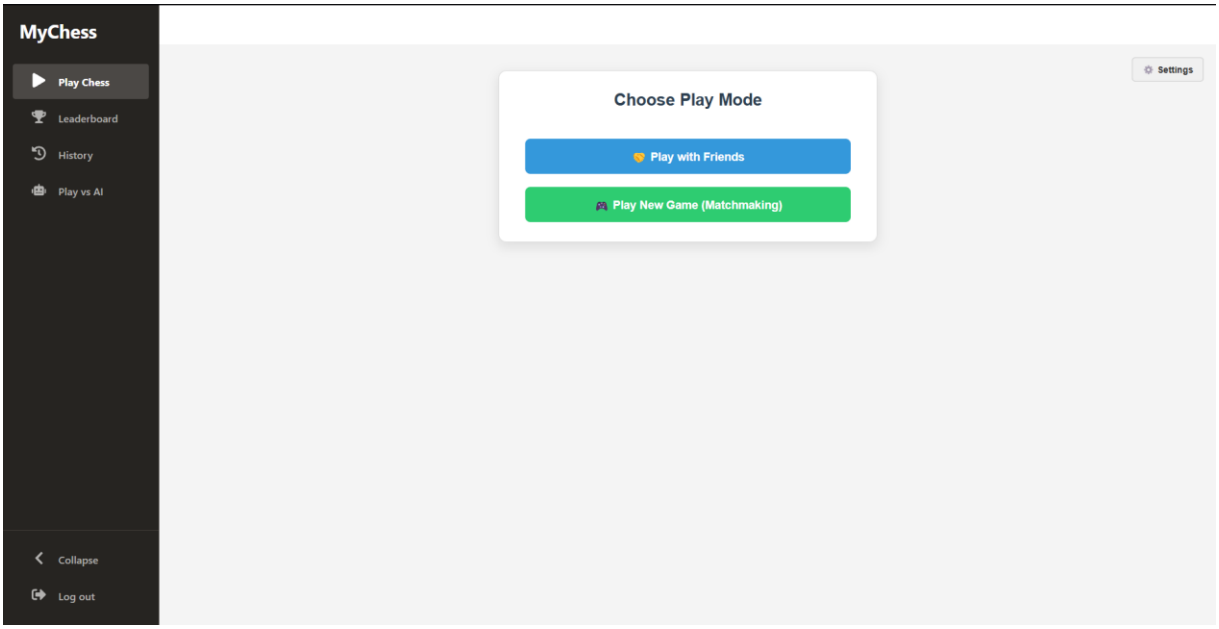

Figure 7: Play Page of the web client

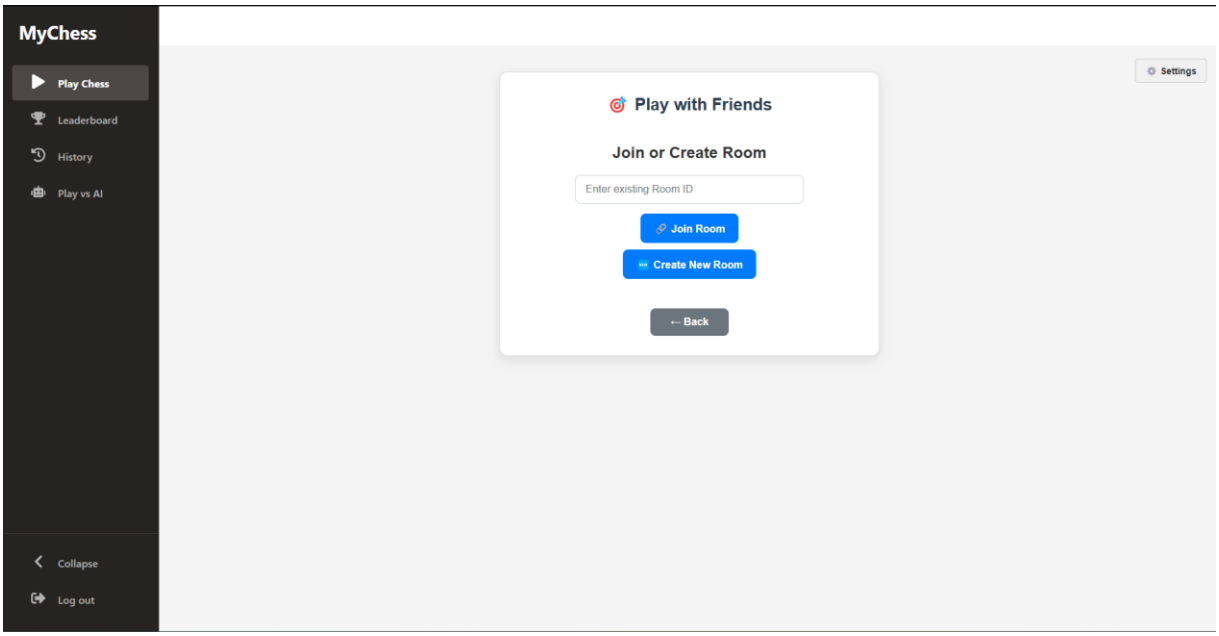

Figure 8: Play with Friend lobby

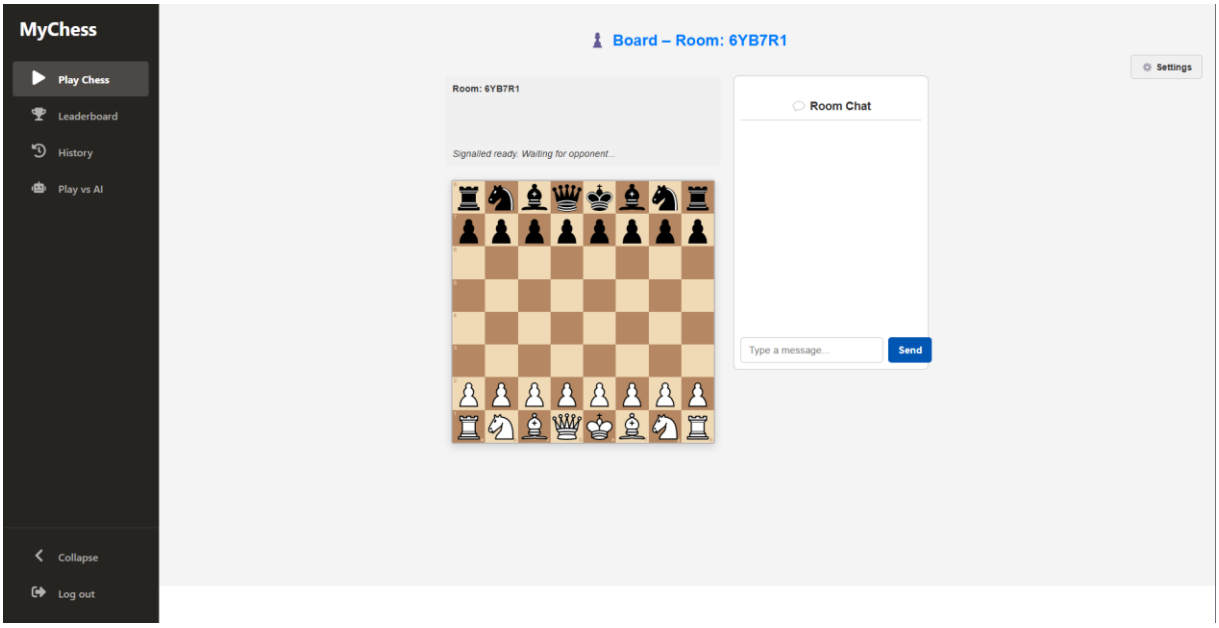

Figure 9: Play with Friend room form

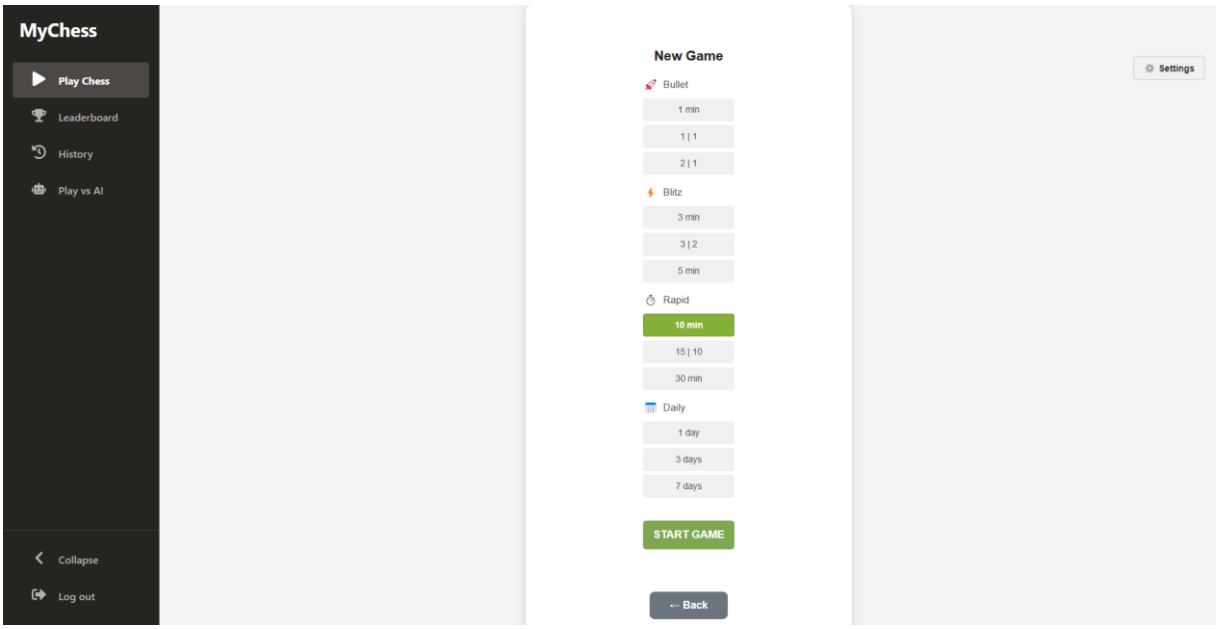

Figure 10 : Play vs Player mode selection

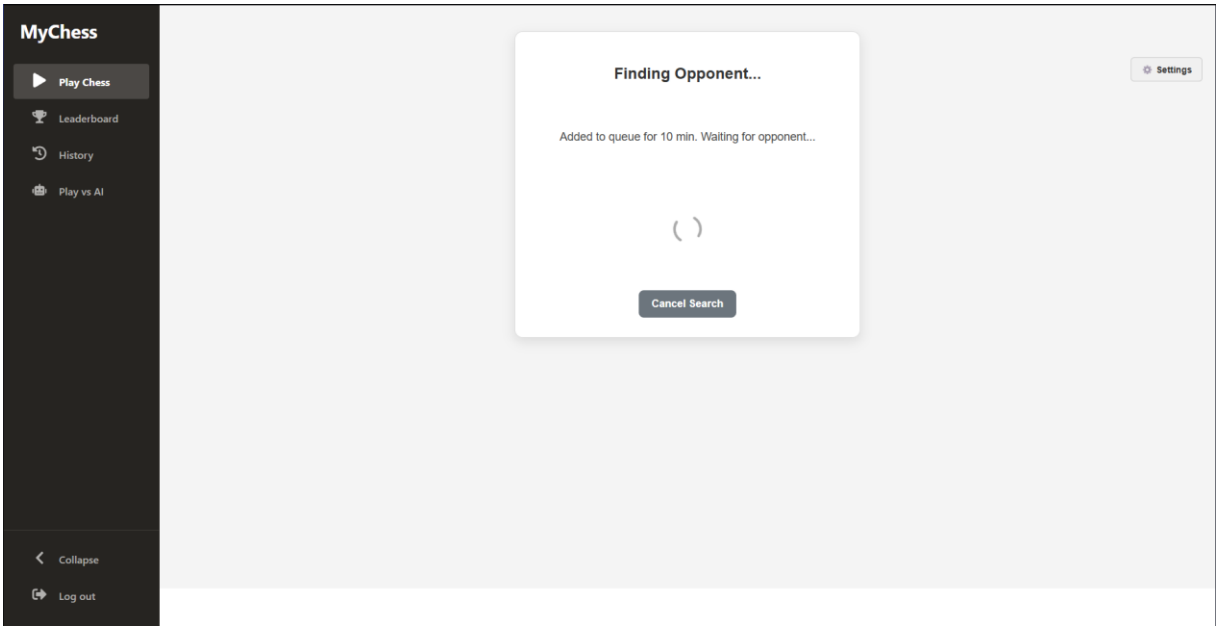

Figure 11 : Play vs Player queue

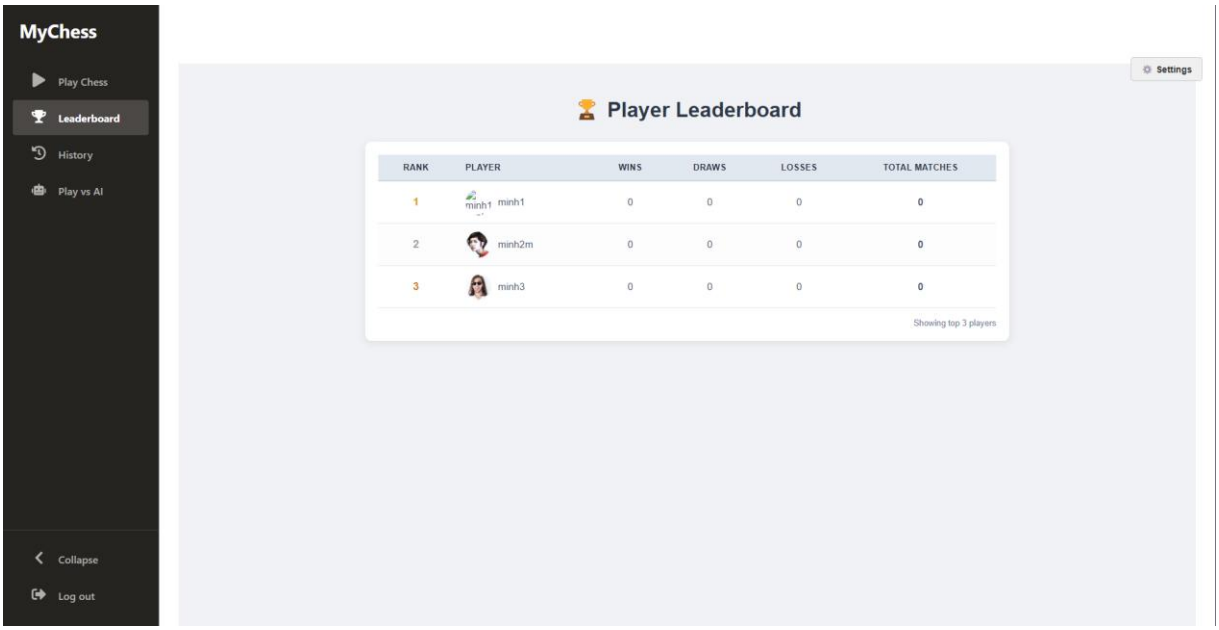

Figure 12 : Leaderboard UI

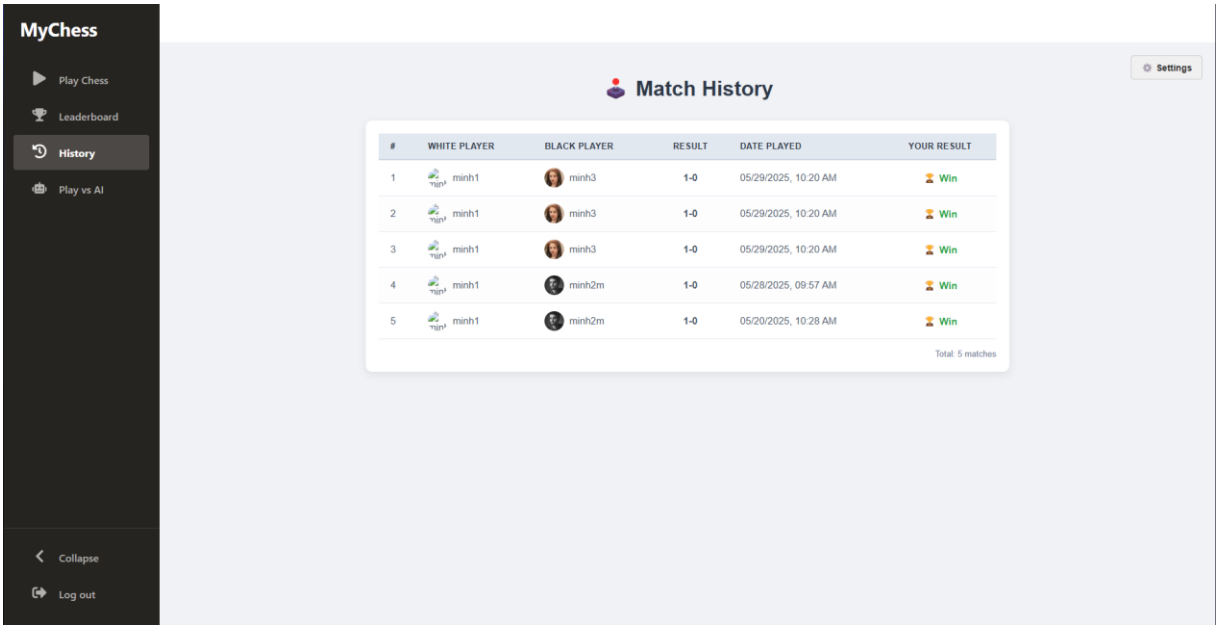

Figure 13 : Match History UI

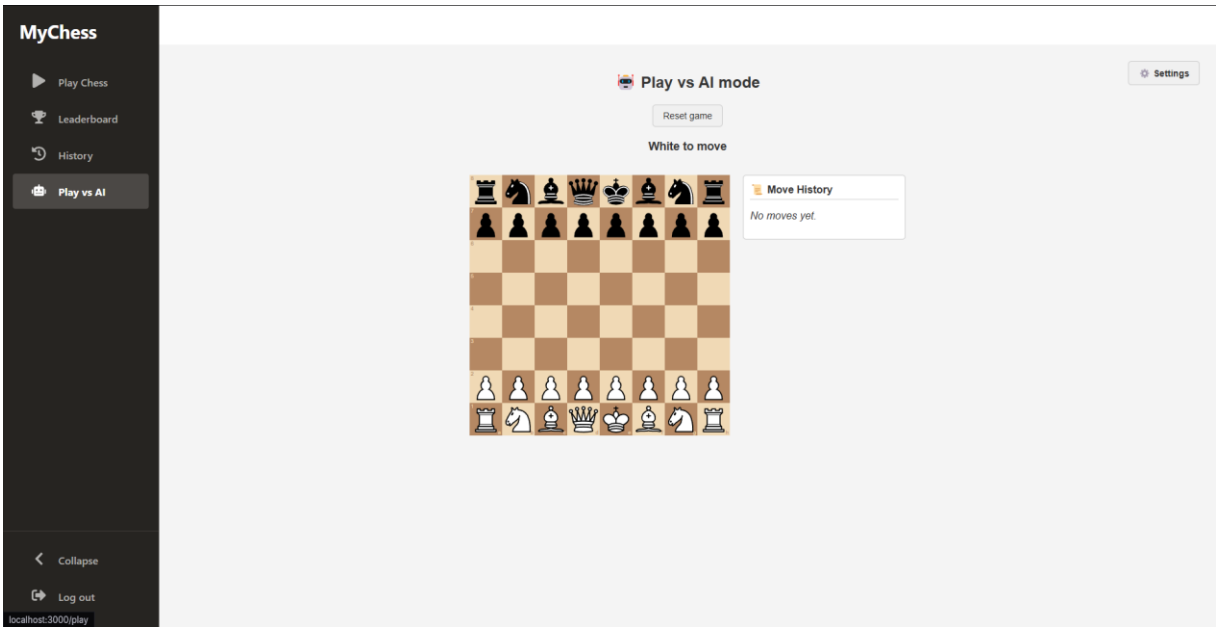

Figure 14 : Play vs AI page

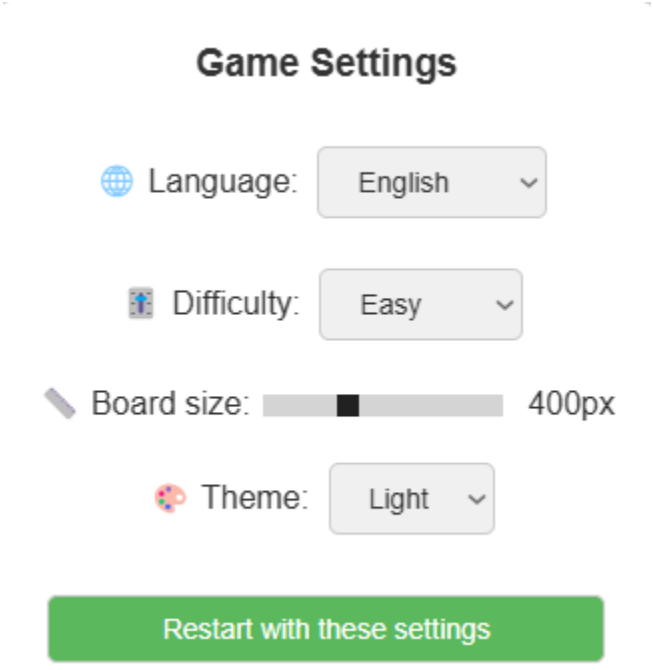

Figure 15 : Game Settings for Play vs AI page