

PAT Project Report

On

Real-time Chess with Socket.IO

Done by:

Shamshadh Unissa

20951A05H1(Batch-5)

Under the Guidance

of

Dr. Ch. V.R Padmaja

Project Chief Mentor

and

G. Indu

Supporting Faculty



Department of Computer Science and Engineering

Institute of Aeronautical Engineering

(Autonomous)

Dundigal-Hyderabad, 500043, Telangana

Outline

1. Title

2. Abstract

a. Brief Introduction

b. Existing System

c. Proposed System

3. Introduction

4. Literature Survey

5. Proposed System/ Methodology

6. Results

7. Conclusion

8. References

Real-time Chess with Socket.IO

Abstract:

The Real-Time Chess with Socket.io project is a web-based chess application that enables multiple players to engage in real-time chess matches over the internet. The project utilizes a combination of HTML, CSS, JavaScript, Socket.io, Node.js, and Express.js to provide a dynamic and interactive chess experience. The application establishes a bidirectional communication channel between the server and the clients using Socket.io, allowing real-time updates and seamless synchronization of chess moves. Users can join game rooms, make moves, and see the changes reflected on the chessboard instantaneously. The Real-Time Chess with Socket.io project showcases the power of real-time communication and demonstrates the seamless integration of web technologies to create a multiplayer chess platform. It provides an engaging and immersive chess experience for players, allowing them to compete against each other in real-time from anywhere in the world.

a. Brief Introduction:

To develop a real-time multiplayer chess game using Socket.IO, leveraging the power of JavaScript and web technologies. By implementing Socket.IO, players will experience synchronized gameplay, allowing multiple participants to make moves and see the actions of others in real time. This project provides an opportunity to gain expertise in web development, Socket.IO, real-time communication, and implementing complex game logic. The game will enforce the rules of chess, offer an intuitive user interface, and include additional features such as chat functionality and a timer. By building this project, I will develop practical skills in JavaScript, Socket.IO, game development, and creating immersive multiplayer experiences.

b. Existing System:

- Existing chess games or platforms lack real-time multiplayer functionality, synchronized gameplay, and features like chat or a timer, limiting the immersive multiplayer experience for chess enthusiasts.
- Currently, most chess games follow a turn-based gameplay model.

- Online chess platforms also lack real-time interaction and require manual refreshing for updates.
- These limitations hinder the dynamic and immersive experience that real-time gaming offers.
- Online chess platforms offer the convenience of playing against opponents remotely, but they still operate on a turn-based structure.
- Players take turns making moves, and the game state is updated only when the opponent's move is received or when the player manually refreshes the page.

c. Proposed System:

- Proposed system is a real-time chess game that revolutionizes the traditional turn-based gameplay model.
- The core technology used in the project is Socket.io, a powerful JavaScript library that enables real-time bidirectional communication between the server and the client.
- Players will have the ability to create or join game rooms, where they can engage in real-time gameplay with opponents from anywhere in the world.
- The game will feature a user-friendly interface that allows players to make their moves seamlessly and view the opponent's moves in real-time.
- The system will incorporate features like move validation, game state management, and special move handling, adhering to the standard rules and logic of chess.
- User authentication will be implemented to ensure secure access to the game rooms, track player statistics, and manage user accounts.
- Additionally, the system may include an AI opponent with adjustable difficulty levels, providing players with the option to play against the computer at their desired skill level.
- The real-time nature of the game will add excitement and immersion, creating a captivating gaming experience for chess enthusiasts.
- Through the proposed system, we aim to provide a platform that offers a seamless real-time chess experience, fostering competitiveness, and enhancing player engagement.

Introduction:

The Real-Time Chess with Socket.io project is a web-based application that brings the classic game of chess into the digital realm, allowing multiple players to engage in real-time chess matches over the internet. By leveraging the power of modern web technologies such as HTML, CSS, JavaScript, Socket.io, Node.js, and Express.js, this project aims to provide a dynamic and interactive chess experience that transcends physical boundaries.

Chess is a strategic board game that has been enjoyed by millions of players worldwide for centuries. Traditionally played on a physical chessboard, the game offers a challenging mental exercise and an opportunity for players to test their strategic thinking and decision-making skills. With the advent of technology, the digital transformation of chess has opened up new possibilities for players to connect and compete with opponents from anywhere in the world.

The Real-Time Chess with Socket.io project embraces the concept of real-time communication and enables players to engage in chess matches with seamless synchronization and instant updates. Through the utilization of Socket.io, a JavaScript library for real-time bidirectional communication, players can make moves, see the changes reflected on the chessboard, and receive live updates in real-time. This real-time aspect of the project adds a layer of excitement and engagement, making it an immersive and thrilling chess experience.

The server-side implementation of the project relies on Node.js and Express.js, popular technologies for building scalable web applications. The Express.js framework provides a robust foundation for handling incoming connections, managing game rooms, validating moves, and broadcasting updates to all connected clients. Socket.io acts as the bridge between the server and the clients, facilitating the real-time communication and event handling required for the multiplayer chess functionality.

On the client-side, the project utilizes HTML, CSS, and JavaScript to create a visually appealing and user-friendly interface. The chessboard UI is rendered using HTML and enhanced with CSS for a polished and intuitive design. JavaScript is responsible for handling user interactions, communicating with the server via Socket.io, and updating the UI based on received events. This client-server architecture ensures a smooth and responsive chess-playing experience for the players.

The Real-Time Chess with Socket.io project showcases the seamless integration of various web technologies to create a multiplayer chess platform that transcends physical limitations. It provides an engaging and immersive chess experience, allowing players to compete against each other in real-time from the comfort of their own devices. Whether it's a casual game between friends or a competitive match between chess enthusiasts, this project offers a platform for players to test their skills, strategize, and experience the thrill of real-time chess.

Literature Survey:

"Real-Time Multiplayer Chess Game Using Web Sockets" by W. Tarantini et al. (2016):

This research paper explores the implementation of a real-time multiplayer chess game using Web Sockets, a technology similar to Socket.io. It discusses the challenges of handling concurrent connections, synchronization of game state, and real-time updates. The paper presents a detailed architectural design and evaluation of the system's performance.

"Real-Time Web Applications with Socket.io" by J. Echavarria (2014):

This book serves as a comprehensive guide to Socket.io, covering the fundamentals of real-time web applications and exploring various use cases. While not specific to chess, it provides valuable insights into the features and functionalities of Socket.io, including broadcasting

messages, handling events, and scaling applications. The book can serve as a reference for implementing real-time features in your chess project.

"Web Sockets: Modern HTML5 Socket.IO" by A. Wong (2013):

This tutorial-style article focuses on the use of Web Sockets, the underlying technology of Socket.io, to build real-time web applications. It covers the basics of WebSocket communication, including establishing connections, sending and receiving messages, and handling events. Although not chess-specific, the article provides a solid understanding of the underlying technology used in your project.

"Chessboard.js: A JavaScript Chess Board Library" by C. Hardwick (2012):

Chessboard.js is a popular JavaScript library for rendering chessboards on web pages. This library provides functionality for displaying chess pieces, highlighting legal moves, and handling user interactions. Understanding and utilizing this library can enhance the visual appeal and user experience of your real-time chess project.

"Chess Algorithms" by M. Schmidt (2013):

This book provides an in-depth exploration of algorithms used in chess programming. It covers algorithms for move generation, move ordering, board representation, evaluation functions, and search algorithms like minimax or alpha-beta pruning. Understanding these algorithms can enhance the gameplay experience and allow you to implement more advanced features, such as AI opponents or analysis tools, in your real-time chess project.

Objectives:

Develop a Real-Time Chess Platform: The main objective of the project is to create a web-based platform that enables multiple players to play chess in real-time over the internet. The platform should provide a seamless and immersive chess experience with synchronized game state and instant updates.

Implement Real-Time Communication: Enable bidirectional real-time communication between the server and clients using Socket.io. Implement features such as live updates, move broadcasting, and chat functionality to facilitate real-time interactions among players.

Create an Interactive Chessboard: Develop an interactive and visually appealing chessboard UI using HTML, CSS, and JavaScript. The chessboard should allow players to make moves by dragging and dropping chess pieces and provide visual feedback for legal moves, check, and checkmate.

Support Multiplayer Functionality: Allow players to join game rooms or create their own rooms to play against friends or random opponents. Implement the ability to handle multiple concurrent games, track players' scores, and provide a leaderboard for competitive gameplay.

Implement Move Validation and Game Logic: Develop algorithms and logic to validate chess moves, enforce the rules of chess, and update the game state accordingly. Ensure that illegal moves are rejected, check and checkmate conditions are detected, and the game ends appropriately.

Implement User Authentication and Security: Integrate user authentication and registration functionality to allow players to create accounts, login, and track their game history. Ensure the security of user data and implement measures to prevent unauthorized access or cheating.

Methodology:

Server-Side Implementation:

- Node.js as the server-side runtime environment.
- Utilize Express.js, a web application framework, to handle incoming HTTP requests and serve static files.
- Implement Socket.io, a real-time communication library, to establish bidirectional communication between the server and clients.
- Set up routes and handlers to manage game rooms, validate moves, and handle authentication.

Client-Side Implementation:

- HTML, CSS, and JavaScript to build the client-side interface.
- Design the chessboard UI using HTML and style it with CSS to create an interactive and visually appealing layout.
- Utilize JavaScript to handle user interactions, such as drag-and-drop functionality for moving chess pieces.
- Implement Socket.io on the client-side to establish a connection with the server and receive real-time updates.

Game Logic:

- Algorithms and functions to handle the game logic, such as move validation, board representation, and detecting checkmate or stalemate conditions.
- Implement functions for legal move generation, piece movement, capturing, and promoting pawns to higher-ranked pieces.
- Keep track of the game state, including the positions of chess pieces, current player turns, and check/checkmate status.

Multiplayer Functionality:

- Allow players to create or join game rooms, each with a unique identifier.

- Implement functionality to match players with opponents based on their preferences, such as random matchmaking or friend invitations.
- Facilitate communication between players within a game room, allowing them to exchange moves, chat messages, and game updates in real-time.

User Authentication:

- To provide user registration and login functionality to allow players to create accounts and track their game history.
- Implement authentication measures to secure user data and protect against unauthorized access.
- Utilize secure storage mechanisms, such as password hashing and encryption, to store user credentials.

Deployment and Scaling:

- Deploy the application on a web server, such as using cloud platforms like Heroku or AWS.
- Consider load balancing and scaling techniques to handle a large number of concurrent players.
- Optimize server-side code and database queries for performance and scalability.

Documentation and Testing:

- Provide comprehensive documentation, including installation instructions, usage guidelines, and API documentation.
- Conduct unit testing to ensure the correctness of critical functionalities, such as move validation and game logic.
- Perform integration testing to verify the interactions between the client and server components.

HTML: HTML (Hypertext Markup Language) is the standard markup language for creating the structure and content of web pages.

CSS: CSS (Cascading Style Sheets) is used to style and enhance the appearance of HTML elements on the web page.

JavaScript: JavaScript is a programming language that enables dynamic interactions on the client-side. It allows you to handle user input, update the UI, and communicate with the server.

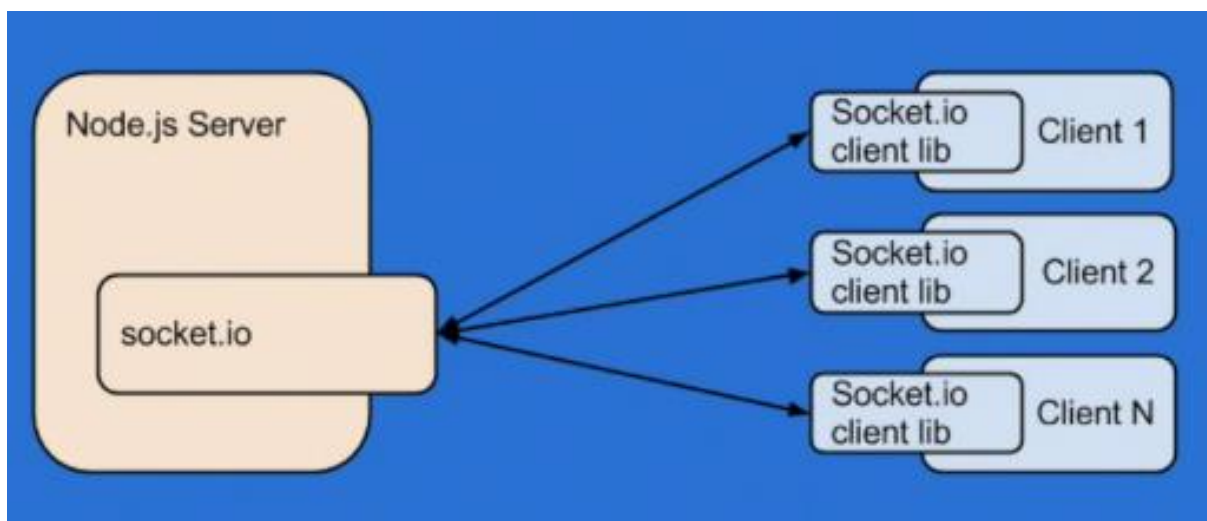
Socket.io: Socket.io is a JavaScript library that enables real-time bidirectional communication between the client and the server. It simplifies the implementation of real-time features, such as live updates, chat, and multiplayer functionality.

Node.js: Node.js is a runtime environment that allows you to run JavaScript on the server-side. It provides a rich ecosystem of libraries and packages, making it a popular choice for building scalable web applications.

Express.js: Express.js is a web framework for Node.js that provides a set of tools and features for building web applications. It simplifies routing, middleware management, and handling HTTP requests and responses.

By combining these technologies, a dynamic and interactive chess application that allows multiple players to play in real-time, make moves, and see the updates reflected across all connected clients.

System Design and Architecture:



<code>io.emit()</code>	send a message to all connected clients
<code>socket.broadcast()</code>	send a message to everyone but me
<code>socket.emit()</code>	send a message to the connected socket
<code>socket.on()</code>	callback to handle messages in client or server

The system design for the real-time chess project involves various components:

User Interface (UI):

- Design an intuitive and responsive user interface to display the chessboard, chess pieces, and game-related information.
- Implement drag-and-drop functionality for moving chess pieces on the board.
- Provide options for player interactions, such as initiating moves, offering draws, or resigning the game.
- Include a chat interface to enable players to communicate during the game.

Game Management:

- Manage game rooms: Allow players to create or join game rooms with unique identifiers.
- Track game state: Maintain the state of each game, including the positions of chess pieces, current turn, and game status (ongoing, checkmate, draw, etc.).
- Handle moves: Validate moves, update the game state, and enforce chess rules.
- Handle time controls: Implement timers to track players' time and enforce time limits for moves.

Real-Time Communication:

- Enable real-time communication between players using Socket.io.
- Broadcast moves and game updates to all players within a game room in real-time.

- Implement a chat system to allow players to communicate with each other during the game.

Data Storage:

- Store game moves, timestamps, player statistics, and other game-related information for future reference.

Deployment and Scaling:

- Deploy the application on a web server, considering scalability and performance requirements.
- Cloud platforms like Azure or GitHub for deployment, ensuring horizontal scaling and load balancing as the number of concurrent users increases.

Documentation and Testing:

- Provide comprehensive documentation, including system architecture, installation instructions, and API documentation.
- Conduct thorough testing, including unit testing, integration testing, and performance testing, to ensure the system functions correctly and meets the performance requirements.

Evaluation / Discussion:

Technologies play a crucial role in different aspects of your real-time chess project. HTML, CSS, and JavaScript handle the client-side user interface and interactivity, while Node.js, Express.js, and Socket.io enable server-side communication and real-time updates, AI algorithms enhance the gameplay experience by providing an intelligent opponent.

Specifications:

Real-Time Gameplay: The project aims to provide a real-time gameplay experience where players can see live updates and interact with each other in real-time.

Multiplayer Functionality: The project includes the ability for players to create or join game rooms and play against each other.

User Authentication: User authentication ensures secure access to the game rooms and tracks individual player statistics and game history.

Game Logic:

Move Validation: The game logic includes algorithms to validate the legality of moves. Each chess piece has specific movement rules, and the game logic and rules. It will check for legal moves, handle capturing of opponent pieces, and consider special moves like castling, en passant, and pawn promotion.

Game State Management: The game logic keeps track of the state of the chess game. This includes the positions of the chess pieces, the current turn, whether a player is in check, and conditions like checkmate, stalemate, or a draw.

Turn-based System: The game logic enforces a turn-based system, ensuring that players take turns making moves. It validates that the correct player is making a move and prevent unauthorized moves during the opponent's turn.

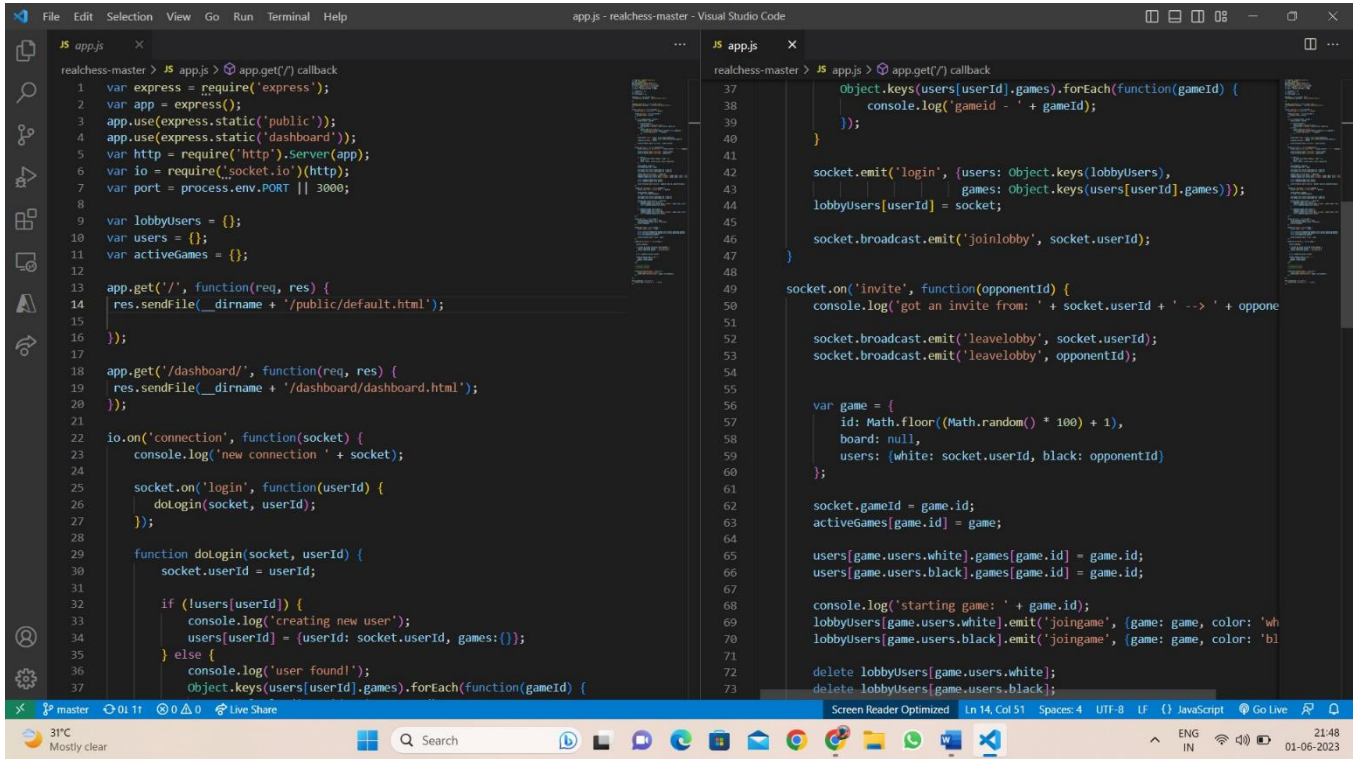
Check and Checkmate Detection: The game logic detects when a player's king is in check and determine if the check leads to checkmate, ending the game.

Draw Conditions: The game logic handle draw conditions such as stalemate (when a player has no legal moves but is not in check), threefold repetition, and the 50-move rule.

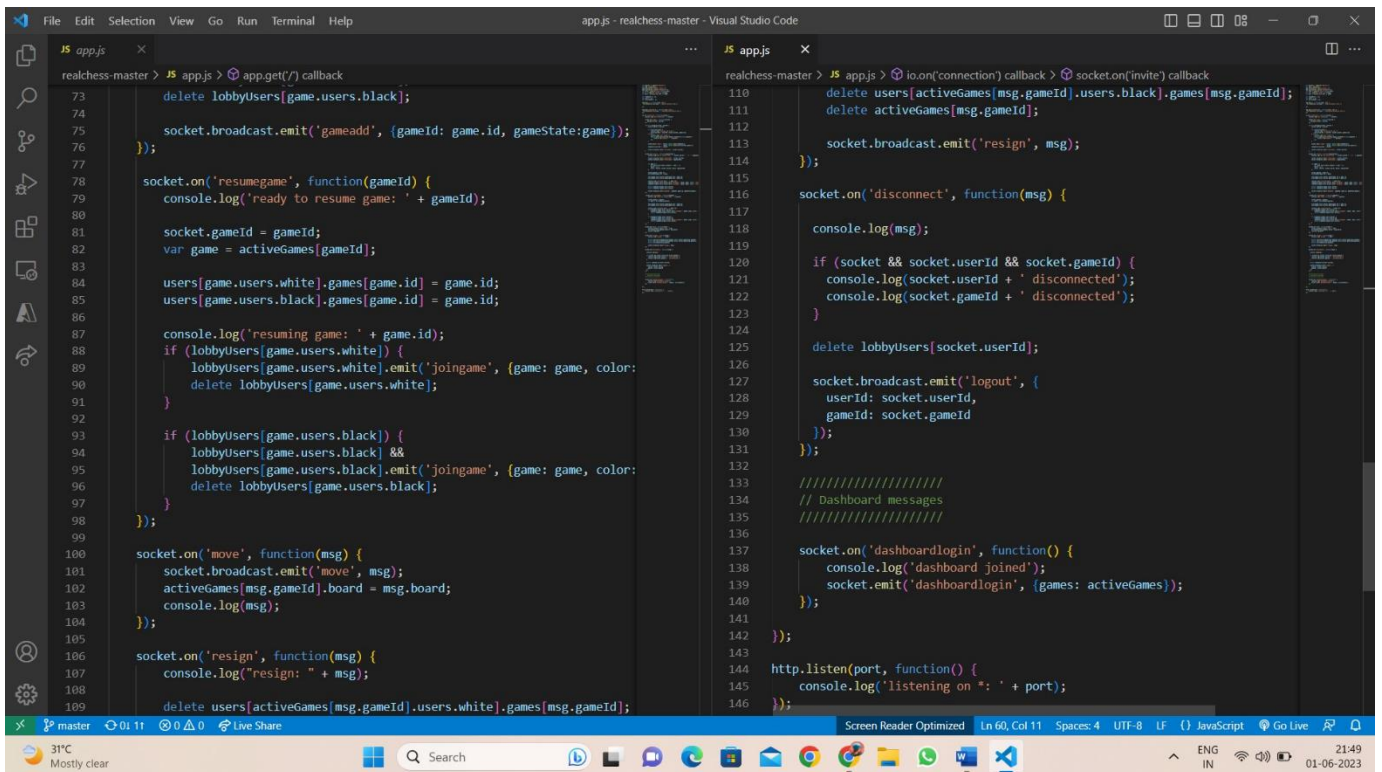
Time Controls: The game logic incorporates time controls, allowing players a limited amount of time for each move. It enforces time limits and handle scenarios like time expiry or timeout.

Implementing the game logic requires a solid understanding of chess rules and algorithms to validate moves, track game state, and determine game outcomes. It's crucial to thoroughly test the game logic to ensure accuracy, handle edge cases, and provide a fair and enjoyable gameplay experience for the users.

Result: App.js

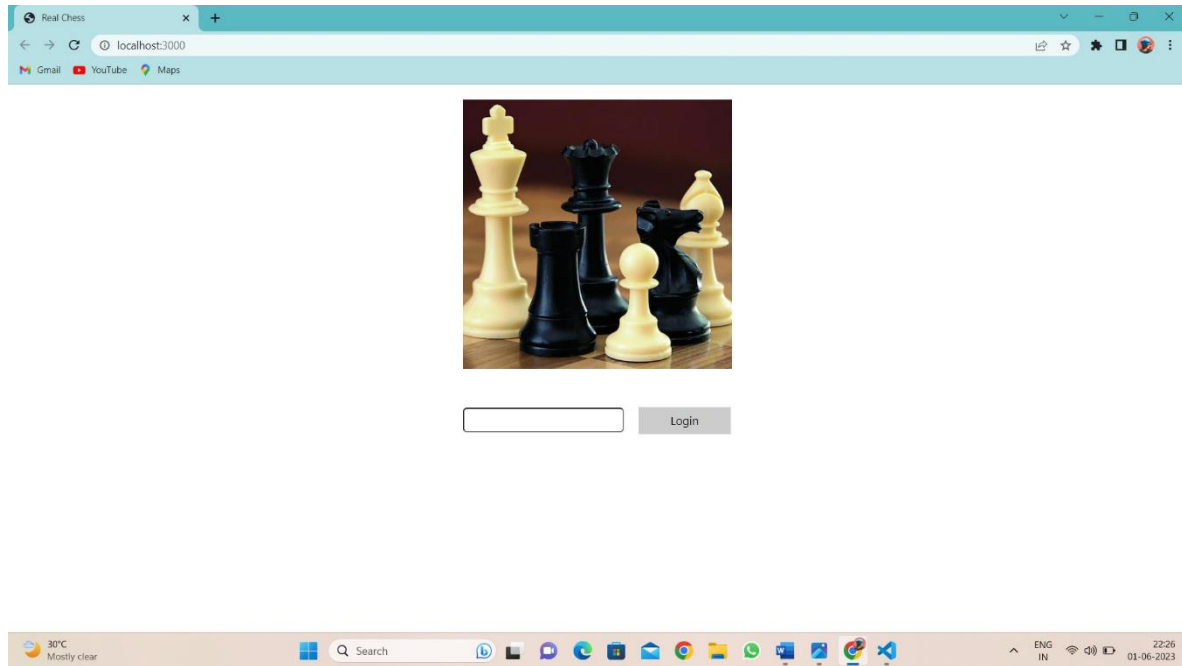


```
1 var express = require('express');
2 var app = express();
3 app.use(express.static('public'));
4 app.use(express.static('dashboard'));
5 var http = require('http').Server(app);
6 var io = require('socket.io')(http);
7 var port = process.env.PORT || 3000;
8
9 var lobbyUsers = {};
10 var users = {};
11 var activeGames = {};
12
13 app.get('/', function(req, res) {
14   res.sendFile(__dirname + '/public/default.html');
15 });
16
17 app.get('/dashboard/', function(req, res) {
18   res.sendFile(__dirname + '/dashboard/dashboard.html');
19 });
20
21 io.on('connection', function(socket) {
22   console.log('new connection ' + socket);
23
24   socket.on('login', function(userId) {
25     doLogin(socket, userId);
26   });
27
28   function doLogin(socket, userId) {
29     socket.userId = userId;
30
31     if (users[userId]) {
32       console.log('creating new user');
33       users[userId] = {userId: socket.userId, games: {}};
34     } else {
35       console.log('user found!');
36       Object.keys(users[userId].games).forEach(function(gameId) {
37
```

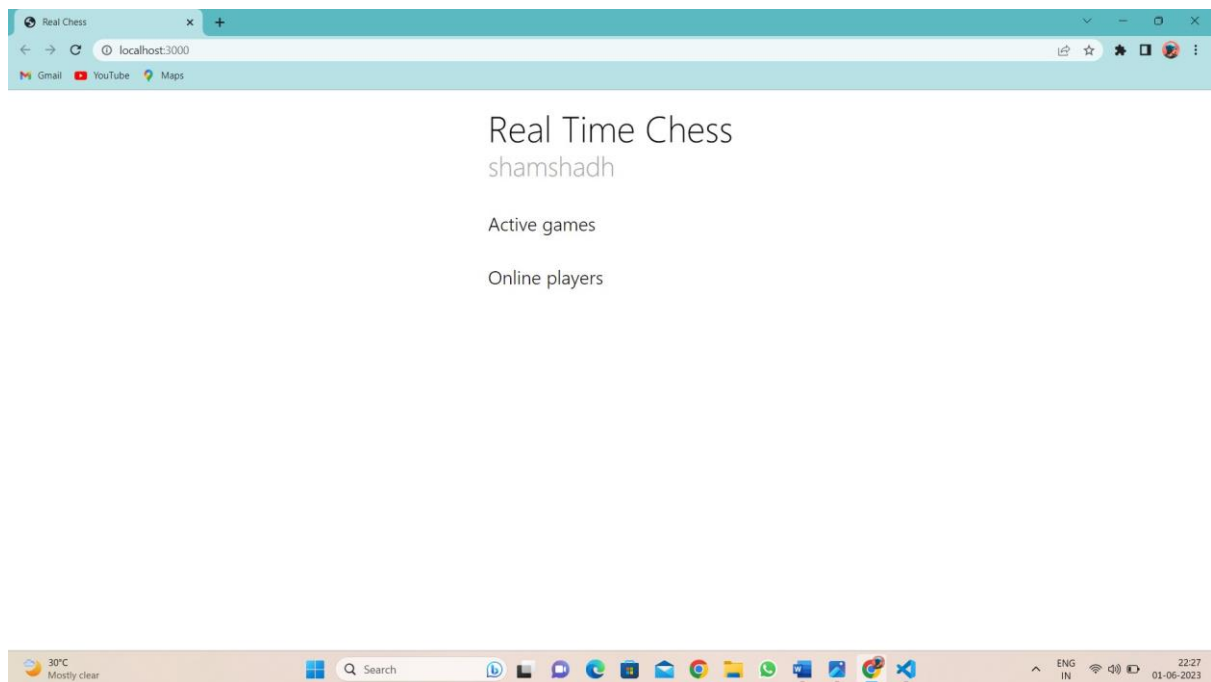


```
73   delete lobbyUsers[game.users.black];
74   socket.broadcast.emit('gameadd', {gameId: game.id, gameState: game});
75 });
76
77 socket.on('resumeGame', function(gameId) {
78   console.log('ready to resume game: ' + gameId);
79
80   socket.gameId = gameId;
81   var game = activeGames[gameId];
82
83   users[game.users.white].games[game.id] = game.id;
84   users[game.users.black].games[game.id] = game.id;
85
86   console.log('resuming game: ' + game.id);
87   if (lobbyUsers[game.users.white]) {
88     lobbyUsers[game.users.white].emit('joingame', {game: game, color:
89     delete lobbyUsers[game.users.white];
90   }
91
92   if (lobbyUsers[game.users.black]) {
93     lobbyUsers[game.users.black] &&
94     lobbyUsers[game.users.black].emit('joingame', {game: game, color:
95     delete lobbyUsers[game.users.black];
96   }
97
98 });
99
100 socket.on('move', function(msg) {
101   socket.broadcast.emit('move', msg);
102   activeGames[msg.gameId].board = msg.board;
103   console.log(msg);
104 });
105
106 socket.on('resign', function(msg) {
107   console.log('resign: ' + msg);
108
109   delete users[activeGames[msg.gameId].users.white].games[msg.gameId];
110
111   delete users[activeGames[msg.gameId].users.black].games[msg.gameId];
112   delete activeGames[msg.gameId];
113
114   socket.broadcast.emit('resign', msg);
115 });
116
117 socket.on('disconnect', function(msg) {
118   console.log(msg);
119
120   if (socket && socket.userId && socket.gameId) {
121     console.log(socket.userId + ' disconnected');
122     console.log(socket.gameId + ' disconnected');
123   }
124
125   delete lobbyUsers[socket.userId];
126
127   socket.broadcast.emit('logout', {
128     userId: socket.userId,
129     gameId: socket.gameId
130   });
131
132   ////////////////
133   // Dashboard messages
134   ////////////////
135
136   socket.on('dashboardlogin', function() {
137     console.log('dashboard joined');
138     socket.emit('dashboardlogin', {games: activeGames});
139   });
140
141 });
142
143 http.listen(port, function() {
144   console.log('listening on *: ' + port);
145 });
146
```

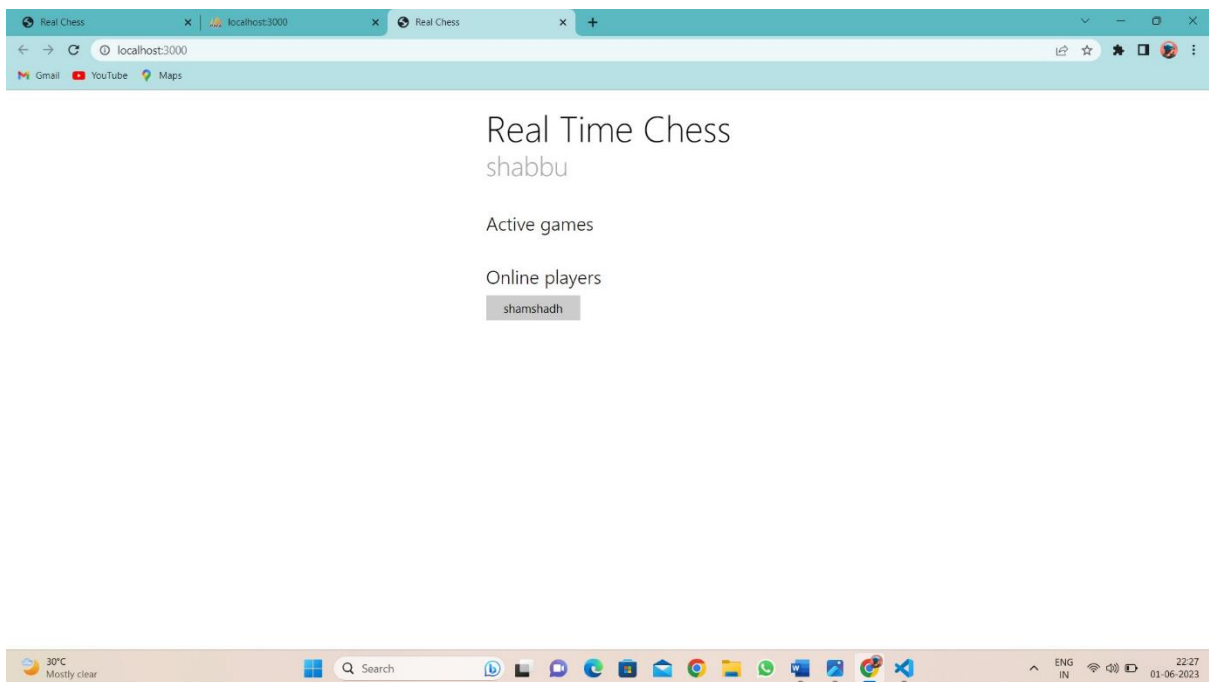
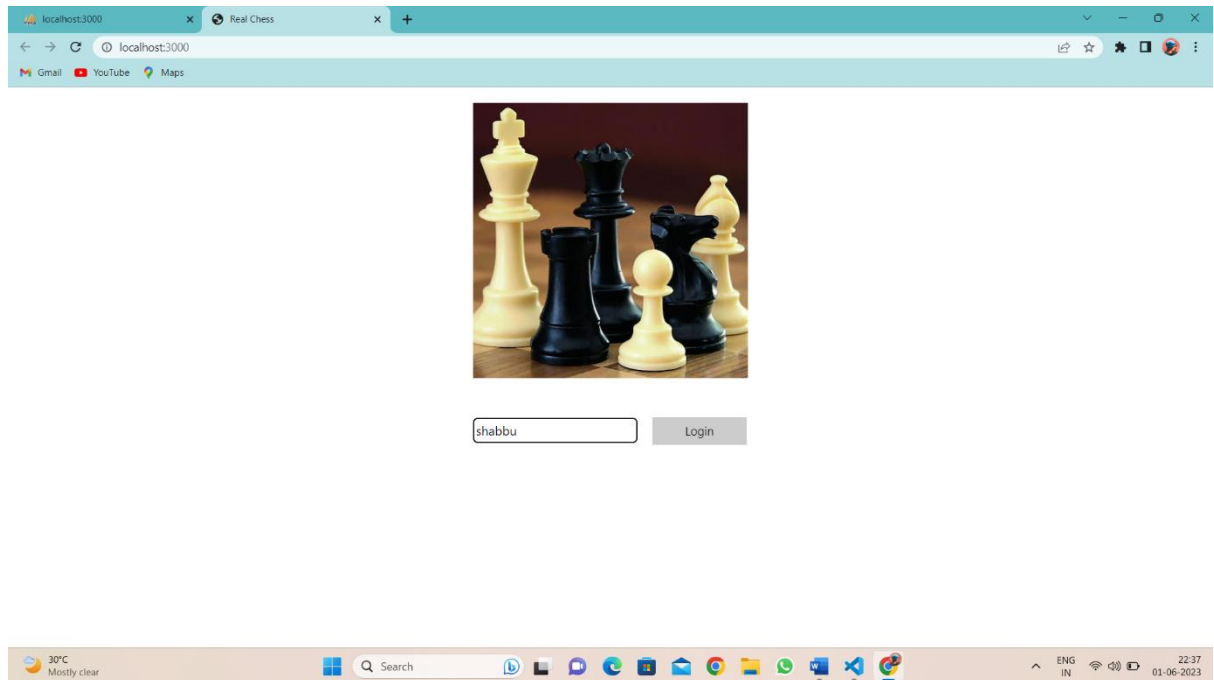

Login Page:

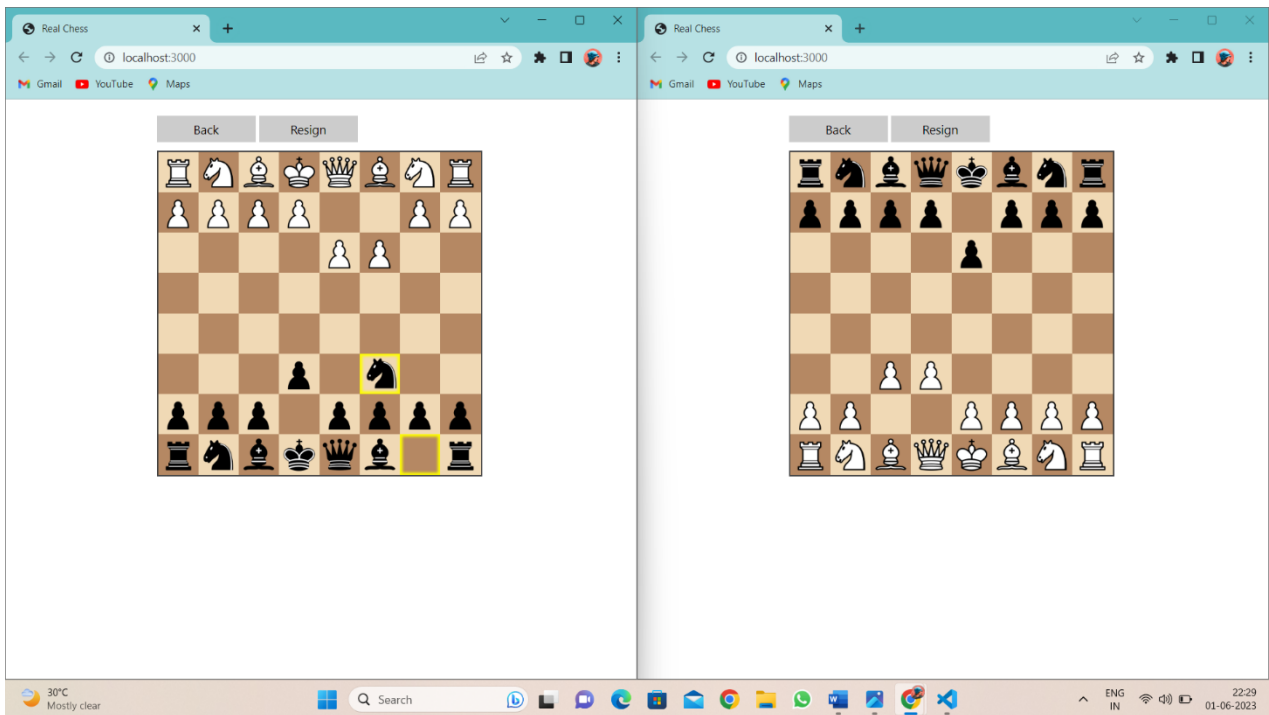
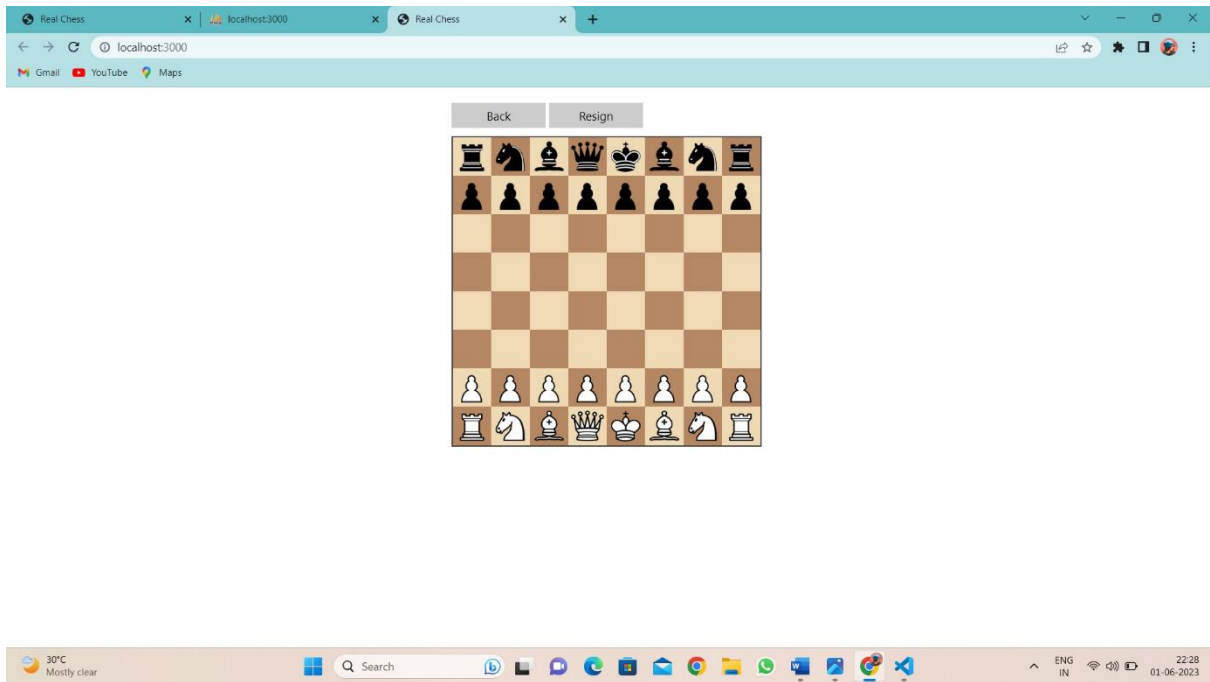


Games available:

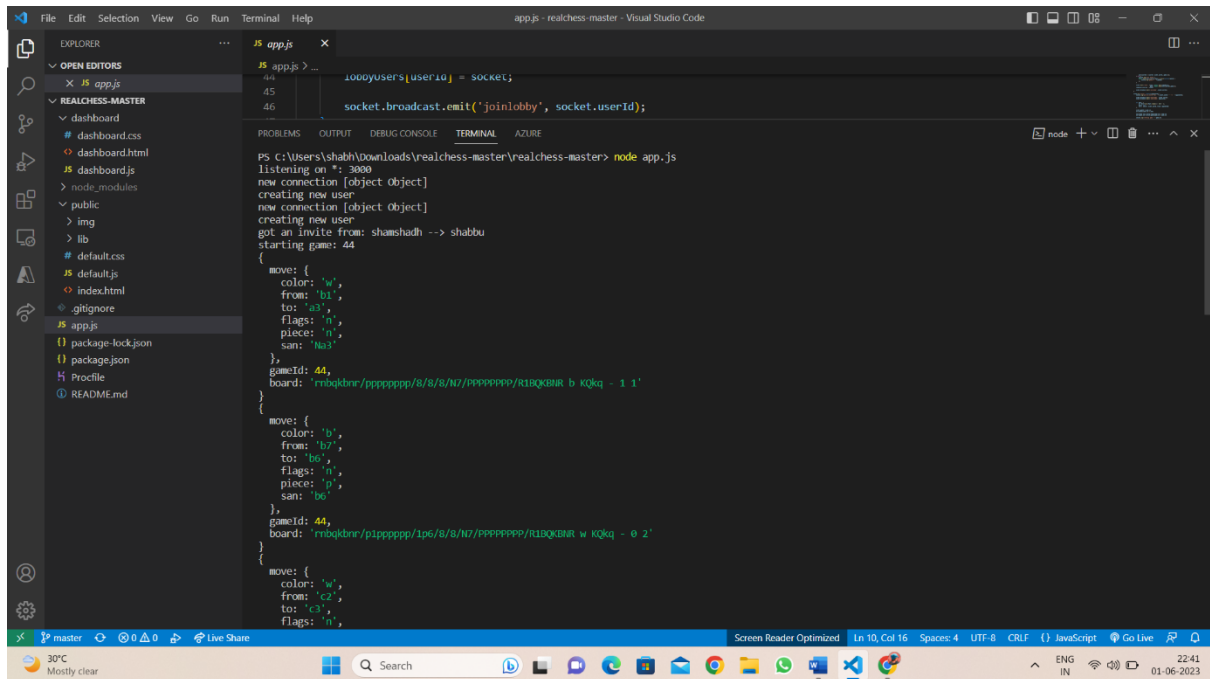


User login 2:

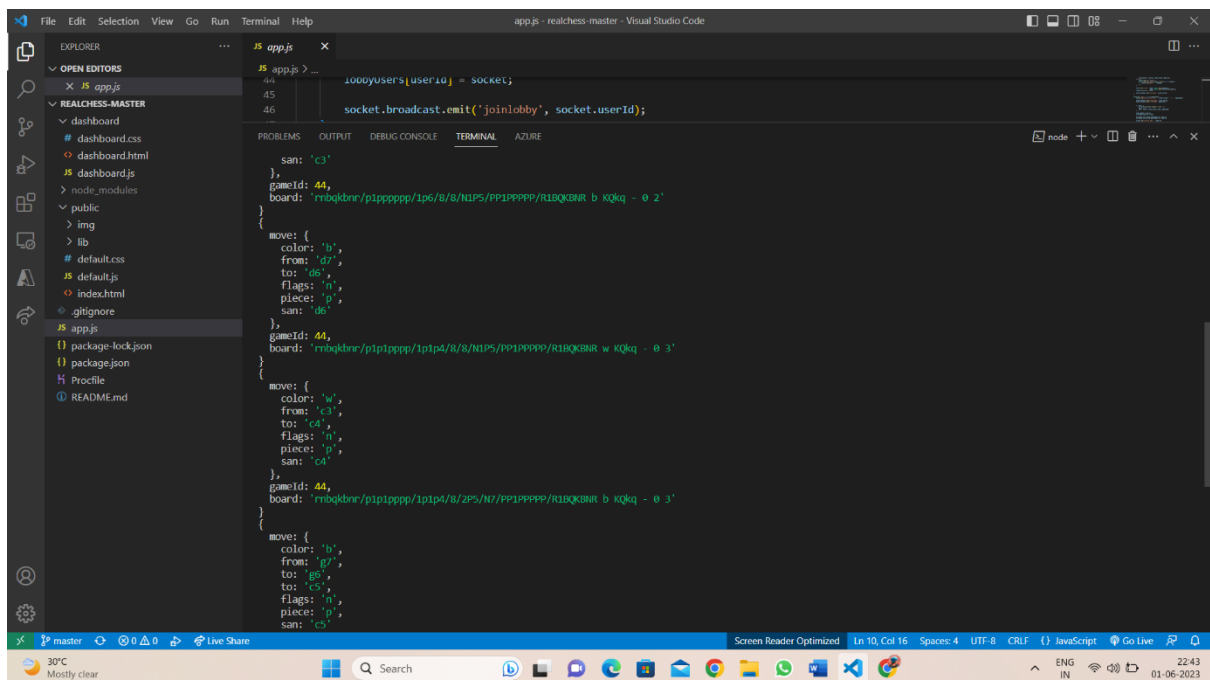




Terminal:

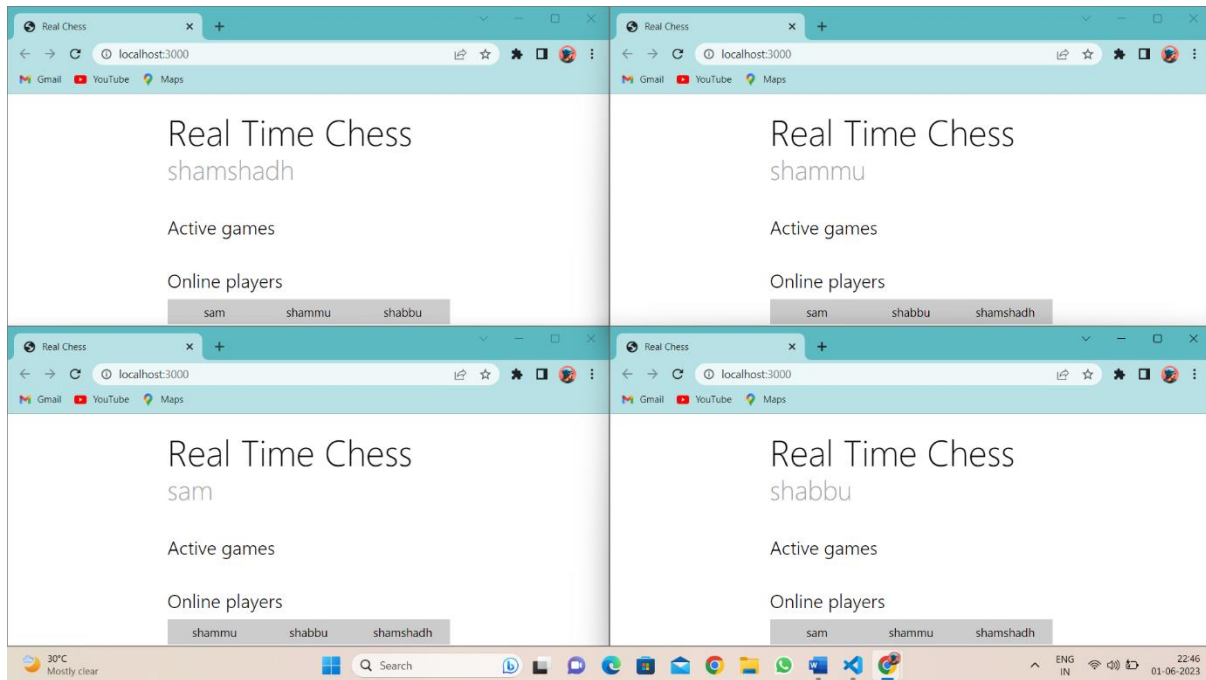


```
PS C:\Users\shabb\Downloads\realchess-master\realchess-master> node app.js
listening on *: 3000
new connection [object Object]
creating new user
new connection [object Object]
creating new user
got an invite from: shamshadh --> shabbu
starting game: 44
{
  move: {
    color: 'w',
    from: 'b1',
    to: 'a3',
    flags: 'n',
    piece: 'n',
    san: 'Na3'
  },
  gameId: 44,
  board: 'rnbqkbnr/pppppppp/8/8/N7/PPPPPPPP/R1BQKBNR b KQkq - 1 1'
}
{
  move: {
    color: 'b',
    from: 'h7',
    to: 'b6',
    flags: 'n',
    piece: 'p',
    san: 'b6'
  },
  gameId: 44,
  board: 'rnbqkbnr/p1pppppp/1p6/8/N7/PPPPPPPP/R1BQKBNR w KQkq - 0 2'
}
{
  move: {
    color: 'w',
    from: 'c2',
    to: 'e3',
    flags: 'n',
    piece: 'n',
    san: 'Nc3'
  },
  gameId: 44,
  board: 'rnbqkbnr/pp1ppppp/1p6/8/N7/PPPPPPPP/R1BQKBNR b KQkq - 0 3'
}
```



```
san: 'c3'
},
gameId: 44,
board: 'rnbqkbnr/p1pppppp/1p6/8/N7/PPPPPPPP/R1BQKBNR b KQkq - 0 2'
}
{
  move: {
    color: 'b',
    from: 'd7',
    to: 'd6',
    flags: 'n',
    piece: 'p',
    san: 'd6'
  },
  gameId: 44,
  board: 'rnbqkbnr/p1p1pppp/1p1p4/8/8/N7/PPPPPPPP/R1BQKBNR w KQkq - 0 3'
}
{
  move: {
    color: 'w',
    from: 'c3',
    to: 'c4',
    flags: 'n',
    piece: 'p',
    san: 'c4'
  },
  gameId: 44,
  board: 'rnbqkbnr/p1p1pppp/1p1p4/8/2P5/N7/PPPPPPPP/R1BQKBNR b KQkq - 0 3'
}
{
  move: {
    color: 'b',
    from: 'g7',
    to: 'g6',
    flags: 'n',
    piece: 'p',
    san: 'g6'
  },
  gameId: 44,
  board: 'rnbqkbnr/p1p1pppp/1p1p4/8/2P5/N7/PPPPPPPP/R1BQKBNR w KQkq - 0 3'
}
```

Multiplayers:



Conclusion:

The development of a real-time chess project utilizing technologies such as HTML, CSS, JavaScript, Node.js, Express.js, and Socket.io brings forth an immersive and interactive gaming experience. The project aims to provide real-time gameplay where players can engage in multiplayer functionality by creating or joining game rooms. The incorporation of user authentication ensures secure access to game rooms, tracking player statistics, and game history. Additionally, the inclusion of an AI opponent with adjustable difficulty levels adds a challenging element to the game. The implementation of game logic encompasses move validation,

game state management, turn-based mechanics, check and checkmate detection, and handling of draw conditions. Thorough testing and adherence to chess rules are essential to deliver accurate and enjoyable gameplay. By successfully implementing these specifications and game logic, the real-time chess project aims to create a captivating environment for chess enthusiasts, fostering competitiveness and enhancing the overall gaming experience.

References:

- ✓ <https://www.youtube.com/watch?v=Isfqigjo7fQ>
- ✓ "Node.js in Action" by Mike Cantelon, Marc Harter, T.J. Holowaychuk, and Nathan Rajlich
- ✓ "Express.js Guide: The Comprehensive Book on Express.js" by Azat Mardan
- ✓ "Learning Socket.IO" by Tyson Cadenhead
- ✓ Official documentation and tutorials for the technologies used, such as HTML, CSS, JavaScript, Node.js, Express.js, and Socket.io