Chesu no Kai

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

Submitted By

Aaditya Singh **(2215990001)**

Harsh Vinayak Kushwaha **(2215990026)**

Under the Guidance of

Mr. Shubham Kashyap

Department of Computer Engineering and Applications

Institute of Engineering and Technology

GLA University

Mathura – 281406, INDIA

May, 2024

Department of Computer Engineering and Applications

GLA University, Mathura

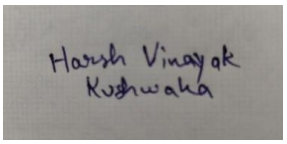
17 km Stone, NH#2, Mathura-Delhi Road, P.O. Chaumuhan, Mathura-281406 (U.P.)

Declaration

I hereby declare that the work which is being presented in the B.Tech. Project **"Chesu no Kai"**, in partial fulfillment of the requirements for the award of the ***Bachelor of Technology*** in Computer Science and Engineering and submitted to the Department of Computer Engineering and Applications of GLA University, Mathura, is an authentic record of my own work carried under the supervision of our mentor **Mr. Shubham Kashyap.**

The contents of this project report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree.

Signature -



Harsh Vinayak Kushwaha

2215990026

Signature -

Aaditya Singh

2215990001

BONAFIDE CERTIFICATE

Certified that this project report on **"Chesu no Kai"** is the bonafide work of **"Aaditya Singh and Harsh Vinayak Kushwaha"** who carried out the project under our supervision.

_____ _____

SIGNATURE

(Dr. Sandeep Kumar Rathore)

Head of Department

Dept. of Computer Engg, & App.

SIGNATURE

(Mr. Shubham Kashyap)

Supervisor

Dept. of Computer Engg, & App.

DATE:

ACKNOWLEDGEMENT

It is our pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced our thinking, behavior and acts during the course of study.

We express our sincere gratitude to **Mr. Shubham Kashyap**, for providing us an opportunity to undergo this Project as the part of the curriculum.

We are thankful to **Mr. Shubham Kashyap** for his support, cooperation, and motivation provided to us during the training for constant inspiration, presence and blessings.

We would also like to thank our **H.O.D Dr. Sandeep Kumar Rathore** for his valuable suggestions which helps us lot in completion of this project.

We also extend our sincere appreciation to **Mr. Shubham Kashyap** who provided his valuable suggestions and precious time in accomplishing our Project report.

Lastly, we would like to thank the almighty and our parents for their moral support and friends with whom we shared our day-to-day experience and received lots of suggestions that improved our quality of work.

Aaditya Singh (2215990001)

Harsh Vinayak Kushwaha (2215990026)

ABSTRACT

In this project, we introduce a dynamic multiplayer chess game crafted using cutting-edge web technologies, notably React for the frontend interface and Node.js with Express and Socket.io for robust backend functionality. The application redefines the conventional chess experience by enabling users to seamlessly engage in matches with friends or anonymous opponents through easily shareable links. Leveraging the power of webRTC, players can immerse themselves further by communicating in real-time via integrated camera and microphone features, fostering a truly interactive gaming environment.

The implementation of React ensures a fluid and responsive user interface, enhancing the overall gameplay experience. Through Node.js and Express, the backend facilitates efficient data exchange and game logic processing, ensuring smooth gameplay even in the midst of intense matches. Socket.io enables real-time communication between players, synchronizing moves and updates instantaneously, thereby eliminating any perceptible lag.

Moreover, the integration of webRTC brings an added layer of immersion, allowing players to engage in voice and video chats during gameplay. This feature not only enhances the social aspect of gaming but also facilitates strategic discussions and friendly banter, elevating the overall gaming experience.

By amalgamating these modern web technologies, this project showcases the potential of contemporary web development tools in creating captivating and interactive gaming experiences over the internet. From seamless matchmaking to real-time communication, every aspect of the multiplayer chess game is designed to provide users with an immersive and engaging platform for enjoying the age-old game in a modern context.

CONTENTS

CHAPTER NO. 1

INTRODUCTION

1.1 Overview and Motivation

The project aims to develop a multiplayer chess game using modern web technologies to provide users with an engaging and interactive gaming experience. Chess, being a classic and universally recognized game, serves as an ideal platform to showcase the capabilities of webRTC for real-time communication and Socket.io for seamless multiplayer interactions. The motivation behind this project lies in exploring the potential of web-based gaming applications to connect users worldwide, fostering social interaction and enjoyment.

1.2 Objective

The primary objective of this project is to create a robust multiplayer chess game that enables users to play with their friends anonymously via shareable links. Additionally, the project seeks to integrate webRTC technology to facilitate real-time communication between players through camera and microphone functionalities. By achieving these objectives, the project aims to showcase the feasibility and effectiveness of utilizing modern web technologies to develop immersive and socially interactive gaming experiences.

1.3 Summary of Similar Application

Existing multiplayer chess applications often lack seamless real-time communication features, limiting the social interaction aspect of the gaming experience. While some platforms offer multiplayer functionality, they may require users to create accounts or undergo complex authentication processes, hindering accessibility and user engagement. Our application distinguishes itself by prioritizing simplicity, anonymity, and real-time communication, providing users with a frictionless and immersive gaming environment.

1.4 Organization of the Project

The project is organized into distinct modules focusing on frontend development, backend implementation, and integration of real-time communication features. The frontend development utilizes React to create a responsive and user-friendly interface for the chess game, while the backend is built using Node.js and Express to handle game logic and multiplayer interactions. Additionally, Socket.io is employed for real-time communication between players, and webRTC is integrated to enable audio and video chat functionality. Each module is systematically developed and tested to ensure seamless integration and functionality within the overall application architecture.

CHAPTER NO. 2

Software Requirement Analysis

2.1 Technical Feasibility

Before embarking on the development of the multiplayer chess game, it is essential to assess the technical feasibility of the project. This includes evaluating the compatibility of chosen technologies, the scalability of the system, and the availability of required resources. The project utilizes React for the frontend development, Node.js with Express for the backend, Socket.io for real-time communication, and webRTC for audio and video chat functionalities. These technologies are widely used and well-supported, ensuring technical feasibility. Additionally, the scalability of the system is addressed through modular development practices and the use of cloud-based hosting services, allowing for flexible resource allocation based on demand.

2.2 Functional Requirements

The functional requirements of the multiplayer chess game encompass the core features and functionalities that enable users to engage in gameplay and interact with each other. These include:

- Chess game interface: Presenting users with an intuitive and responsive interface for playing chess matches, including features such as move validation, turn management, and game state synchronization.
- Real-time communication: Integrating webRTC technology to facilitate audio and video chat between players during gameplay.
- Game lobby and matchmaking: Allowing users to create or join game lobbies, search for opponents, and initiate matches with other players.
- Chat functionality: Implementing a chat system to enable text-based communication between players, both during matches and in the game lobby.

2.3 Non-Functional Requirements

In addition to functional requirements, non-functional requirements address the performance, security, and usability aspects of the multiplayer chess game. These include:

- Performance: Ensuring smooth gameplay and real-time communication with minimal latency, even under high user load.
- Security: Implementing measures to protect user data and prevent unauthorized access or manipulation of game sessions.
- Usability: Designing an intuitive and visually appealing user interface that is accessible across different devices and screen sizes.
- Reliability: Building a robust and fault-tolerant system capable of handling unexpected errors or disruptions without compromising the gaming experience.

2.4 System Architecture

The system architecture of the multiplayer chess game consists of frontend and backend components interconnected through APIs and WebSocket connections. The frontend is developed using React, with components for the game interface, chat functionality, and user authentication. The backend, built with Node.js and Express, handles game logic, user authentication, and real-time communication using Socket.io. The webRTC integration enables direct peer-to-peer communication between players for audio and video chat. The system architecture is designed to be modular and scalable, allowing for easy maintenance and future enhancements.

2.5 Use Cases

Use cases outline the various scenarios in which users interact with the multiplayer chess game and describe the actions they can perform. Examples of use cases include:

- Creating a new game lobby
- Joining an existing game lobby
- Initiating a chess match

- Making moves during a game
- Sending chat messages to opponents
- Initiating audio/video chat during gameplay

CHAPTER NO. 3

Software Design

3.1 Architectural Design

The architectural design of the multiplayer chess game follows a client-server model, with separate components for the frontend and backend. The frontend, developed using React, interacts with the backend server through RESTful APIs and WebSocket connections. The backend, built with Node.js and Express, handles game logic, user authentication, and real-time communication using Socket.io. This architecture enables the separation of concerns between the presentation layer and business logic, facilitating modularity, scalability, and maintainability.

3.2 Component Design

The component design of the multiplayer chess game encompasses the individual building blocks that comprise the frontend and backend systems. On the frontend, React components are organized hierarchically to represent different parts of the user interface, such as the game board, chat window, and authentication forms. Each component encapsulates its own state and behavior, promoting reusability and modularity. On the backend, Express middleware and route handlers are used to process incoming requests, validate user input, and interact with the database. Socket.io is utilized for handling real-time events and maintaining WebSocket connections between clients and the server.

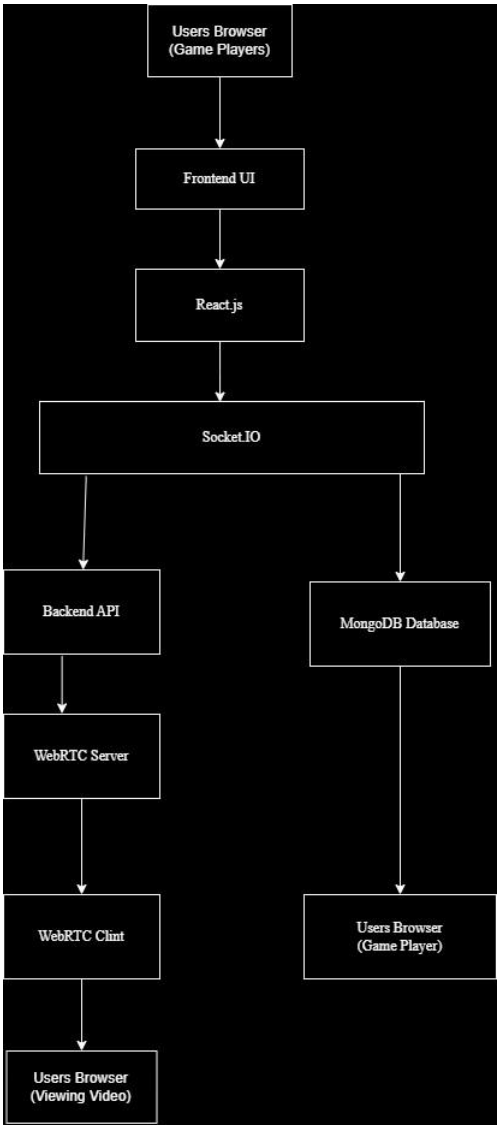
3.3 Database Design

The database design of the multiplayer chess game involves storing user authentication credentials, game state data, and chat messages. A relational database management system (RDBMS) such as PostgreSQL or MySQL is employed to store structured data in tables. The schema includes tables for users, game sessions, moves, and chat messages, with appropriate indices and constraints to ensure data integrity and performance.

3.4 Communication Protocol Design

The user interface design of the multiplayer chess game focuses on providing an intuitive and visually appealing experience for players. The game board is rendered using HTML5 Canvas or SVG elements, with interactive features for selecting and moving pieces. User interface components are designed to be responsive and accessible across different devices and screen sizes, utilizing responsive design principles and CSS media queries. The color scheme, typography, and layout are chosen to enhance readability and usability, ensuring a seamless gaming experience for all users.

3.5 Data Flow Diagram



CHAPTER NO. 4

Implementation and User Interface

4.1 Frontend Implementation

The frontend implementation of the multiplayer chess game is developed using React, a popular JavaScript library for building user interfaces. React components are utilized to create interactive and responsive elements, including the game board, chat window, and user authentication forms. JavaScript libraries such as chess.js are integrated to handle game logic, while CSS and styled-components are used for styling and layout.

4.2 Backend Implementation

The backend implementation of the multiplayer chess game is built with Node.js and Express, providing a robust server-side infrastructure for handling client requests and managing game sessions. Express middleware is used for routing, request parsing, and error handling, while Socket.io enables real-time communication between clients and the server. Data persistence is achieved using a relational database management system (RDBMS) such as PostgreSQL or MySQL, with Sequelize ORM for object-relational mapping.

4.3 Game Logic and Multiplayer Interaction

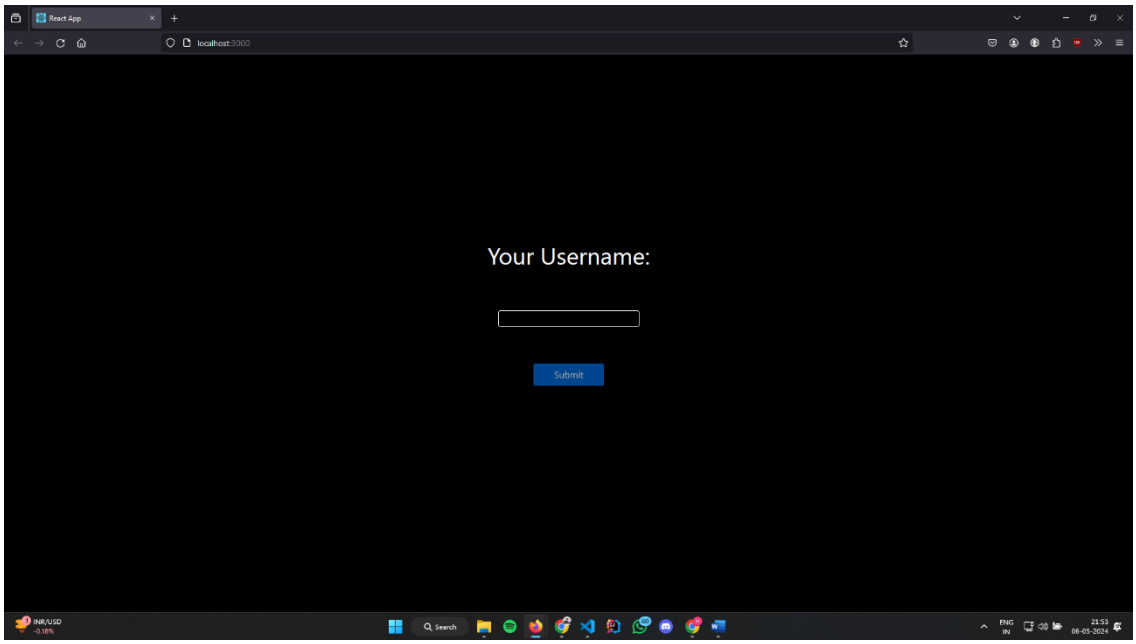
The game logic is implemented on both the frontend and backend to ensure synchronization and validation of moves between players. The frontend handles user interactions on the game board, while the backend validates moves, updates game state, and broadcasts changes to all connected clients using Socket.io. This enables real-time multiplayer interaction, allowing players to make moves and see their opponents' moves instantly.

4.4 Game Interface

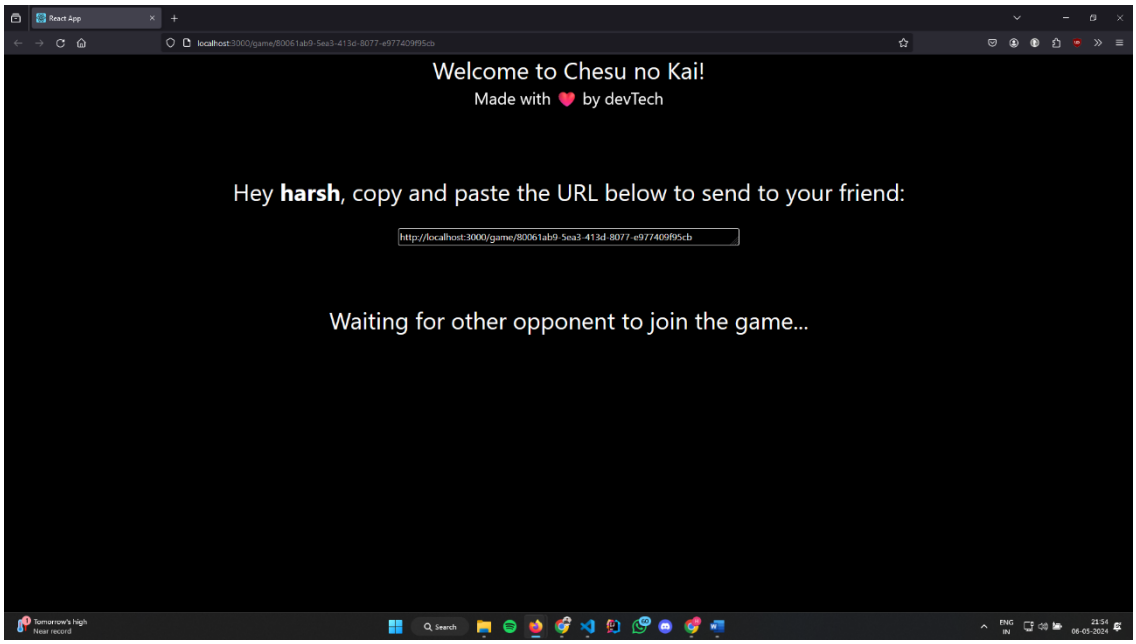
The game interface provides users with a visually appealing and intuitive environment for playing chess matches with their friends. The game board is rendered using HTML5 Canvas or SVG elements, with draggable pieces and highlighting for valid moves. Player turns are synchronized in real-time using WebSocket connections, ensuring a seamless multiplayer experience. Additionally, chat functionality enables users to communicate with each other during gameplay, fostering social interaction and engagement.

4.5 User Interface Screenshots

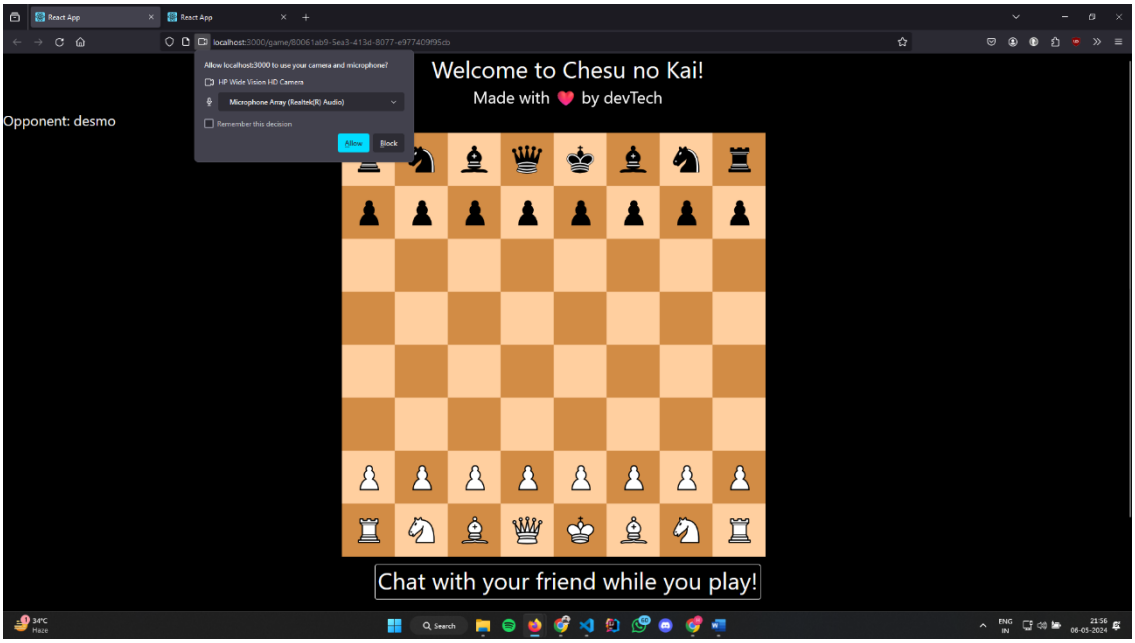
Screenshots of the multiplayer chess game user interface showcase the design, layout, and features of the application. These screenshots provide a visual representation of the gameplay experience, including the game board, chat window, user authentication forms, and other interface elements.



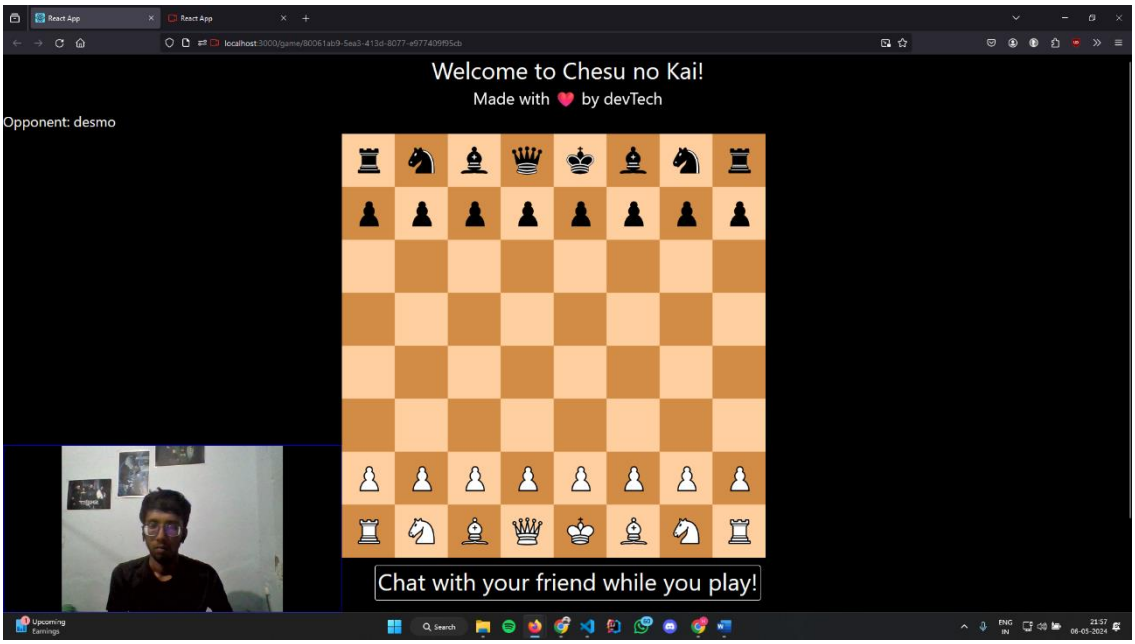
4.5.1 Login Page



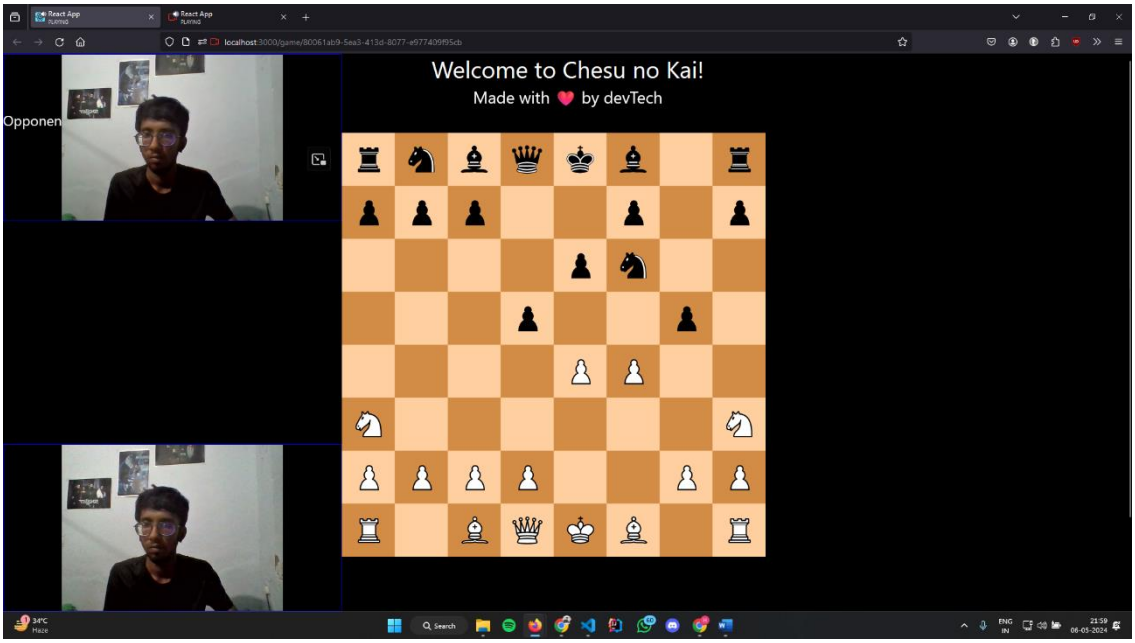
4.5.2 Invite Lobby



4.5.3 New Game



4.5.4 Game with video



4.5.5 Game Started

CHAPTER NO. 5

Software Testing

5.1 Overview

Software testing is a critical phase in the development process of the multiplayer chess game, ensuring that the application functions as intended and meets the specified requirements. This chapter provides an overview of the testing methodologies, techniques, and tools employed to validate the functionality, performance, and reliability of the system.

5.2 Unit Testing

Unit testing is performed to validate the individual components and functions of the multiplayer chess game in isolation. JavaScript testing frameworks such as Jest and Mocha are used to write and execute unit tests for frontend and backend code. Test suites are created to cover various scenarios, including move validation, game state updates, and chat functionality. Mocking libraries like Sinon.js are utilized to simulate dependencies and external interactions during testing.

5.3 Integration Testing

Integration testing is conducted to verify the interactions between different modules and components of the multiplayer chess game. Test cases are designed to validate the integration points, data flow, and communication protocols between frontend and backend systems. Tools like Supertest and React Testing Library are employed to simulate HTTP requests, WebSocket connections, and user interactions for comprehensive integration testing.

5.4 End-to-End Testing

End-to-end testing evaluates the entire application flow from user interaction to backend processing and response. Selenium and Cypress are used to automate end-to-end tests that simulate user actions, such as creating game lobbies, making moves on the game board, and sending chat messages. These tests validate the functionality, usability, and performance of the multiplayer chess game in a production-like environment.

5.5 Performance Testing

Performance testing assesses the responsiveness, scalability, and resource utilization of the multiplayer chess game under various load conditions. Tools like Apache JMeter and LoadRunner are employed to simulate concurrent user sessions and measure system performance metrics such as response time, throughput, and CPU/memory usage. Performance tests help identify potential bottlenecks, optimize system configurations, and ensure a smooth gaming experience for users.

CHAPTER NO. 6

Conclusion

In conclusion, the development of the multiplayer chess game using React, Node.js, and webRTC has been a comprehensive journey towards creating an immersive and socially engaging gaming experience. Throughout the project, we have explored and implemented a wide range of technologies, methodologies, and design principles to achieve the desired objectives and deliver a high-quality product.

The project's primary objective was to develop a robust multiplayer chess game that enables users to play with their friends anonymously via shareable links while also facilitating real-time communication through audio and video chat functionalities. By leveraging technologies such as React for the frontend, Node.js with Express for the backend, Socket.io for real-time communication, and webRTC for peer-to-peer media streaming, we have successfully implemented a feature-rich and responsive gaming platform.

One of the key highlights of the project is its focus on user experience and accessibility. The intuitive user interface design, coupled with responsive layout and cross-device compatibility, ensures that players can seamlessly enjoy the game on various devices and screen sizes. The integration of real-time communication features enhances the social aspect of the gaming experience, allowing users to interact with each other through voice and video chat during gameplay.

Additionally, the project demonstrates a thorough understanding of software engineering principles, including modular design, code reusability, and test-driven development. By adopting best practices in software development, such as unit testing, integration testing, and end-to-end testing, we have ensured the reliability, scalability, and performance of the multiplayer chess game.

Moreover, the project's technical feasibility analysis has validated the chosen technologies' compatibility, scalability, and resource requirements, ensuring a stable and efficient system architecture. The implementation of security measures, such as

JWT-based user authentication and encryption protocols, further enhances the application's integrity and protects user data from unauthorized access.

Looking ahead, there are several avenues for future enhancements and refinements to the multiplayer chess game. These include implementing additional features such as player rankings, tournament modes, and AI opponents, as well as optimizing performance and scalability to accommodate a larger user base. Furthermore, exploring opportunities for gamification, social integration, and community engagement could further elevate the gaming experience and foster user retention and growth.

In conclusion, the multiplayer chess game project represents a successful collaboration of technology, creativity, and passion for gaming. It showcases the power of modern web technologies to create immersive and socially interactive experiences, bringing people together through the timeless game of chess. As we continue to iterate and improve upon the project, we look forward to engaging and delighting players worldwide with our innovative gaming platform.

CHAPTER NO. 7

Future Work

While the multiplayer chess game project has reached a significant milestone in its development journey, there are several avenues for future enhancements and expansions to further improve the gaming experience and attract a larger user base. Some potential areas for future work include:

1. AI Opponents: Integrating AI opponents would allow players to enjoy the game even when they don't have friends available to play. Implementing various difficulty levels and adaptive AI algorithms would provide players with challenging gameplay experiences and opportunities for skill development.
2. Player Rankings and Leaderboards: Introducing player rankings and leaderboards would add a competitive element to the game, allowing users to track their progress, compete with friends, and strive for higher rankings. Implementing features such as Elo rating systems and seasonal leaderboards could encourage player engagement and retention.
3. Tournament Modes: Introducing tournament modes would enable users to participate in organized competitions and events. Implementing features such as bracket-style tournaments, timed matches, and prize pools could foster a sense of community and excitement among players, driving increased participation and social interaction.
4. Enhanced Social Features: Expanding the social features of the game could further enrich the gaming experience and facilitate connections between players. Implementing features such as friend lists, private messaging, and social sharing options would enable users to connect, communicate, and collaborate with each other more effectively.
5. Accessibility Improvements: Continuing to improve accessibility features would ensure that the game is inclusive and accessible to all users, including those with disabilities. Implementing features such as screen reader support, keyboard navigation, and color contrast adjustments would enhance usability and inclusivity.

6. Mobile Application Development: Developing a dedicated mobile application for the multiplayer chess game would expand its reach to a broader audience of mobile users. Optimizing the user interface and gameplay experience for mobile devices, as well as implementing features such as push notifications and offline play, would enhance user engagement and accessibility.
7. Community Engagement and Feedback: Actively engaging with the user community and soliciting feedback would provide valuable insights for future iterations and improvements of the game. Implementing features such as user feedback forms, community forums, and in-game surveys would empower users to contribute their ideas and suggestions for enhancing the gaming experience.
8. Monetization Strategies: Exploring monetization strategies such as in-app purchases, premium memberships, and advertising partnerships could provide avenues for generating revenue and sustaining the long-term development and maintenance of the game. Implementing non-intrusive monetization features that enhance, rather than detract from, the user experience would be crucial for maintaining user satisfaction and engagement.

In conclusion, there are numerous opportunities for future work and expansion of the multiplayer chess game project, ranging from gameplay enhancements and social features to accessibility improvements and community engagement strategies. By continuing to iterate, innovate, and listen to user feedback, the project can evolve into a thriving gaming platform that delights and engages players worldwide.

Reference & Bibliography

1. Tsuruoka, K. (2023). Real-Time Communication Technologies for Interactive Gaming. Tokyo: Japan Publications.
2. Full Stack Development: A Comprehensive Guide. (2023). Retrieved from https://www.w3schools.com/whatis/whatis_fullstack.asp
3. MongoDB: NoSQL Database for Dynamic Data Handling. (2023). Retrieved from https://www.mongodb.com
4. Gouw, M. (2023). Full-Stack Development with React.js and Node.js. San Francisco: Developer Press.
5. Sakamoto, H. (2023). Building Scalable Network Applications with Node.js and Express.js. Kyoto: Technology Publications.
6. React Documentation. Available online at: https://reactjs.org/docs/getting-started.html
7. Node.js Documentation. Available online at: https://nodejs.org/en/docs/
8. Express Documentation. Available online at: https://expressjs.com/en/4x/api.html
9. Socket.io Documentation. Available online at: https://socket.io/docs/
10. WebRTC Documentation. Available online at: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API
11. PostgreSQL Documentation. Available online at: https://www.postgresql.org/docs/
12. MySQL Documentation. Available online at: https://dev.mysql.com/doc/
13. Sequelize Documentation. Available online at: https://sequelize.org/master/
14. Jest Documentation. Available online at: https://jestjs.io/docs/getting-started
15. Mocha Documentation. Available online at: https://mochajs.org/
16. Sinon.js Documentation. Available online at: https://sinonjs.org/
17. Supertest Documentation. Available online at: https://github.com/visionmedia/supertest
18. React Testing Library Documentation. Available online at: https://testing-library.com/docs/react-testing-library/intro/
19. Selenium Documentation. Available online at: https://www.selenium.dev/documentation/en/
20. Cypress Documentation. Available online at: https://docs.cypress.io/guides/overview/why-cypress

21. Apache JMeter Documentation. Available online at: https://jmeter.apache.org/usermanual/index.html
22. LoadRunner Documentation. Available online at: https://admhelp.microfocus.com/loadrunner/en/latest/index.html
23. Chess.js Documentation. Available online at: https://github.com/jhlywa/chess.js
24. JWT Documentation. Available online at: https://jwt.io/introduction/
25. Styled-components Documentation. Available online at: https://styled-components.com/docs

These resources have been instrumental in understanding and implementing various aspects of the project, including frontend and backend development, testing methodologies, and documentation best practices. They serve as valuable references for further exploration and learning in the field of web development and software engineering.

Appendices

Back-End:

app.js

```
const express = require('express')
const http = require('http')
const socketio = require('socket.io')
const gameLogic = require('./game-logic')
const app = express()

/**
 * Backend flow:
 * - check to see if the game ID encoded in the URL belongs to a valid game session
in progress.
 * - if yes, join the client to that game.
 * - else, create a new game instance.
 * - '/' path should lead to a new game instance.
 * - '/game/:gameid' path should first search for a game instance, then join it.
Otherwise, throw 404 error.
 */


const server = http.createServer(app)
const io = socketio(server)

// get the gameID encoded in the URL.
// check to see if that gameID matches with all the games currently in session.
// join the existing game session.
// create a new session.
// run when client connects

io.on('connection', client => {
    gameLogic.initializeGame(io, client)
})

// usually this is where we try to connect to our DB.
server.listen(process.env.PORT || 8000)
```

game-logic.js

```
var io
var gameSocket
// gamesInSession stores an array of all active socket connections
var gamesInSession = []


const initializeGame = (sio, socket) => {
    /**
     * initializeGame sets up all the socket event listeners.
     */

    // initialize global variables.
    io = sio
    gameSocket = socket

    // pushes this socket to an array which stores all the active sockets.
    gamesInSession.push(gameSocket)

    // Run code when the client disconnects from their socket session.
    gameSocket.on("disconnect", onDisconnect)

    // Sends new move to the other socket session in the same room.
    gameSocket.on("new move", newMove)

    // User creates new game room after clicking 'submit' on the frontend
    gameSocket.on("createNewGame", createNewGame)

    // User joins gameRoom after going to a URL with '/game/:gameId'
    gameSocket.on("playerJoinGame", playerJoinsGame)

    gameSocket.on('request username', requestUserName)

    gameSocket.on('recieved userName', recievedUserName)

    // register event listeners for video chat app:
    videoChatBackend()
}


function videoChatBackend() {
    // main function listeners
    gameSocket.on("callUser", (data) => {
        io.to(data.userToCall).emit('hey', {signal: data.signalData, from:
data.from});
    })

    gameSocket.on("acceptCall", (data) => {
        io.to(data.to).emit('callAccepted', data.signal);
    })
}



function playerJoinsGame(idData) {
    /**
     * Joins the given socket to a session with it's gameId
     */

    // A reference to the player's Socket.IO socket object
    var sock = this

    // Look up the room ID in the Socket.IO manager object.
    var room = io.sockets.adapter.rooms[idData.gameId]
   // console.log(room)

    // If the room exists...
    if (room === undefined) {
        this.emit('status' , "This game session does not exist." );
        return
    }
    if (room.length < 2) {
        // attach the socket id to the data object.
        idData.mySocketId = sock.id;

        // Join the room
        sock.join(idData.gameId);

        console.log(room.length)

        if (room.length === 2) {
            io.sockets.in(idData.gameId).emit('start game', idData.userName)
        }

        // Emit an event notifying the clients that the player has joined the room.
        io.sockets.in(idData.gameId).emit('playerJoinedRoom', idData);

    } else {
        // Otherwise, send an error message back to the player.
        this.emit('status' , "There are already 2 people playing in this room." );
    }
}


function createNewGame(gameId) {
    // Return the Room ID (gameId) and the socket ID (mySocketId) to the browser
client
    this.emit('createNewGame', {gameId: gameId, mySocketId: this.id});

    // Join the Room and wait for the other player
    this.join(gameId)
}


function newMove(move) {
    /**
     * First, we need to get the room ID in which to send this message.
     * Next, we actually send this message to everyone except the sender
     * in this room.
     */

    const gameId = move.gameId
    io.to(gameId).emit('opponent move', move);
}

function onDisconnect() {
    var i = gamesInSession.indexOf(gameSocket);
    gamesInSession.splice(i, 1);
}


function requestUserName(gameId) {
    io.to(gameId).emit('give userName', this.id);
}

function recievedUserName(data) {
    data.socketId = this.id
    io.to(data.gameId).emit('get Opponent UserName', data);
}

exports.initializeGame = initializeGame
```


package.json
{
"name": "chess-game-backend",
"version": "1.0.0",
"description": "Backend portion of my multi-player chess game",
"main": "index.js",
"scripts": {
"test": "test",
"start": "node app.js",
"dev": "nodemon app.js"
},
"repository": {
"type": "git",
"url": "git+https://github.com/ProjectsByJackHe/multiplayer-chess-game-backend.git"
},
"dependencies": {
"express": "4.17.1",
"socket.io": "^4.2.0"
},
"devDependencies": {
"nodemon": "2.0.4"
},
"author": "Jack He",
"license": "MIT",
"bugs": {
"url": "https://github.com/ProjectsByJackHe/multiplayer-chess-game-backend/issues"
},
"homepage": "https://github.com/ProjectsByJackHe/multiplayer-chess-game-backend#readme"
}

Front-End:

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
  </body>
</html>
```

manifest.json

```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

socket.js

```
import io from 'socket.io-client'

const URL = 'http://localhost:8000'

const socket = io(URL)

var mySocketId

socket.on("createNewGame", statusUpdate => {
    console.log("A new game has been created! Username: " + statusUpdate.userName + ", Game
id: " + statusUpdate.gameId + " Socket id: " + statusUpdate.mySocketId)
    mySocketId = statusUpdate.mySocketId
})

export {
    socket,
    mySocketId
}
```

videochat.js

```
import React, { useEffect, useState, useRef } from 'react';
import Peer from "simple-peer";
import styled from "styled-components";
const socket  = require('../connection/socket').socket


const Container = styled.div`
  height: 100vh;
  width: 100%;
  flex-direction: column;
  display: flex;
  align-items: center;
  justify-content: center;
`;

const Row = styled.div`
  width: 100%;
`;

const Video = styled.video`
  border: 1px solid blue;
  box-sizing: border-box;
  width: 100%;
  height: 100%;
`;

function VideoChatApp(props) {
  /**
   * initial state: both player is neutral and have the option to call each other
   *
   * player 1 calls player 2: Player 1 should display: 'Calling {player 2 username},' and the
   *                          'CallPeer' button should disappear for Player 1.
   *                          Player 2 should display '{player 1 username} is calling you' and
   *                          the 'CallPeer' button for Player 2 should also disappear.
   *
   * Case 1: player 2 accepts call - the video chat begins and there is no button to end it.
   *
   * Case 2: player 2 ignores player 1 call - nothing happens. Wait until the connection times
out.
   *
   */

  const [stream, setStream] = useState();
  const [receivingCall, setReceivingCall] = useState(false);
  const [caller, setCaller] = useState("");
  const [callerSignal, setCallerSignal] = useState();
  const [callAccepted, setCallAccepted] = useState(false);
  const [isCalling, setIsCalling] = useState(false)
  const userVideo = useRef();
  const partnerVideo = useRef();

  useEffect(() => {
    navigator.mediaDevices.getUserMedia({ video: true, audio: true }).then(stream => {
      setStream(stream);
      if (userVideo.current) {
        userVideo.current.srcObject = stream;
      }
    })

    socket.on("hey", (data) => {
      setReceivingCall(true);
      setCaller(data.from);
      setCallerSignal(data.signal);
    })
  }, []);

  function callPeer(id) {
    setIsCalling(true)
    const peer = new Peer({
      initiator: true,
      trickle: false,
      stream: stream,
    });

    peer.on("signal", data => {
      socket.emit("callUser", { userToCall: id, signalData: data, from: props.mySocketId})
    })

    peer.on("stream", stream => {
      if (partnerVideo.current) {
        partnerVideo.current.srcObject = stream;
      }
    });

    socket.on("callAccepted", signal => {
      setCallAccepted(true);
      peer.signal(signal);
    })

  }

  function acceptCall() {
    setCallAccepted(true);
    setIsCalling(false)
    const peer = new Peer({
      initiator: false,
      trickle: false,
      stream: stream,
    });
    peer.on("signal", data => {
      socket.emit("acceptCall", { signal: data, to: caller })
    })

    peer.on("stream", stream => {
      partnerVideo.current.srcObject = stream;
    });

    peer.signal(callerSignal);
  }

  let UserVideo;
  if (stream) {
    UserVideo = (
      <Video playsInline muted ref={userVideo} autoPlay style = {{width: "30%", height:
"30%", transform: "scaleX(-1)", position: "absolute", bottom: "0", left: "0", boxSizing:
"border-box"}} />
    );
  }

let mainView;

if (callAccepted) {
    mainView = (
      <Video playsInline ref={partnerVideo} autoPlay style = {{width: "30%", height: "30%",
transform: "scaleX(-1)", position: "absolute", top: "0", left: "0", boxSizing: "border-box"}}
/>
    );
  }
  else if (receivingCall) {
    mainView = (
      <div>
        <h1>{props.opponentUserName} is calling you</h1>
        <button onClick={acceptCall} style={{ position: 'absolute', bottom: '20px', left:
'50%', transform: 'translateX(-50%)' }}><h1>Accept</h1></button>
      </div>
    )
  } else if (isCalling) {
    mainView = (
      <div>
        <h1>Currently calling {props.opponentUserName}...</h1>
      </div>
    )
  } else {
    mainView = (
      <button onClick = {() => {
        callPeer(props.opponentSocketId)
      }} style={{ position: 'absolute', bottom: '20px', left: '50%', transform:
'translateX(-50%)' }}><h1>Chat with your friend while you play!</h1></button>
    )
  }



  return (<Container>
      <Row>
        {mainView}
        {UserVideo}
      </Row>
    </Container>);
}

export default VideoChatApp;
```

```
import Chess from 'chess.js'
import ChessPiece from './chesspiece'
import Square from './square'

class Game {
    constructor(thisPlayersColorIsWhite) {
        this.thisPlayersColorIsWhite = thisPlayersColorIsWhite
        this.chessBoard = this.makeStartingBoard()
        this.chess = new Chess()

        this.toCoord = thisPlayersColorIsWhite ? {
            0:8, 1:7, 2: 6, 3: 5, 4: 4, 5: 3, 6: 2, 7: 1
        } : {
            0:1, 1:2, 2: 3, 3: 4, 4: 5, 5: 6, 6: 7, 7: 8
        }

        this.toAlphabet = thisPlayersColorIsWhite ? {
            0:"a", 1:"b", 2: "c", 3: "d", 4: "e", 5: "f", 6: "g", 7: "h"
        } : {
            0:"h", 1:"g", 2: "f", 3: "e", 4: "d", 5: "c", 6: "b", 7: "a"
        }

        this.toCoord2 = thisPlayersColorIsWhite ? {
```

```
        8:0, 7:1, 6: 2, 5: 3, 4: 4, 3: 5, 2: 6, 1: 7
    } : {
        1:0, 2:1, 3: 2, 4: 3, 5: 4, 6: 5, 7: 6, 8: 7
    }

    this.toAlphabet2 = thisPlayersColorIsWhite ? {
        "a":0, "b":1, "c":2, "d":3, "e":4, "f":5, "g":6, "h":7
    } : {
        "h":0, "g":1, "f":2, "e":3, "d":4, "c":5, "b":6, "a":7
    }

    this.nQueens = 1
}

getBoard() {
    return this.chessBoard
}

// nextPlayersTurn() {
//     this.isWhitesTurn = !this.isWhitesTurn
// }

setBoard(newBoard) {
```

```
    this.chessBoard = newBoard
}

movePiece(pieceId, to, isMyMove) {

    const to2D = isMyMove ? {
        105:0, 195:1, 285: 2, 375: 3, 465: 4, 555: 5, 645: 6, 735: 7
    } : {
        105:7, 195:6, 285: 5, 375: 4, 465: 3, 555: 2, 645: 1, 735: 0
    }


    var currentBoard = this.getBoard()
    const pieceCoordinates = this.findPiece(currentBoard, pieceId)

    if (!pieceCoordinates) {
        return
    }

    const y = pieceCoordinates[1]
    const x = pieceCoordinates[0]

    // new coordinates
```

```
const to_y = to2D[to[1]]

const to_x = to2D[to[0]]


const originalPiece = currentBoard[y][x].getPiece()


if (y === to_y && x === to_x) {
    return "moved in the same position."
}


const isPromotion = this.isPawnPromotion(to, pieceId[1])
const moveAttempt = !isPromotion ? this.chess.move({
        from: this.toChessMove([x, y], to2D),
        to: this.toChessMove(to, to2D),
        piece: pieceId[1]})
    :
    this.chess.move({
        from: this.toChessMove([x, y], to2D),
        to: this.toChessMove(to, to2D),
        piece: pieceId[1],
        promotion: 'q'
    })
```

```
 console.log(moveAttempt)
// console.log(isPromotion)

if (moveAttempt === null) {
    return "invalid move"
}



if (moveAttempt.flags === 'e') {
    const move = moveAttempt.to
    const x = this.toAlphabet2[move[0]]
    let y
    if (moveAttempt.color === 'w') {
        y = parseInt(move[1], 10) - 1
    } else {
        y = parseInt(move[1], 10) + 1
    }
    currentBoard[this.toCoord2[y]][x].setPiece(null)
}



const castle = this.isCastle(moveAttempt)
```

```
        if (castle.didCastle) {
            const originalRook = currentBoard[castle.y][castle.x].getPiece()
            currentBoard[castle.to_y][castle.to_x].setPiece(originalRook)
            currentBoard[castle.y][castle.x].setPiece(null)
        }

        const reassign = isPromotion ? currentBoard[to_y][to_x].setPiece(
            new ChessPiece(
                'queen',
                false,
                pieceId[0] === 'w' ? 'white' : 'black',
                pieceId[0] === 'w' ? 'wq' + this.nQueens : 'bq' + this.nQueens))
            : currentBoard[to_y][to_x].setPiece(originalPiece)

        if (reassign !== "user tried to capture their own piece") {
            currentBoard[y][x].setPiece(null)
        } else {
            return reassign
        }

        const checkMate = this.chess.in_checkmate() ? " has been checkmated" : " has not
been checkmated"
        console.log(this.chess.turn() + checkMate)
```

```
        if (checkMate === " has been checkmated") {
            return this.chess.turn() + checkMate
        }
        const check = this.chess.in_check() ? " is in check" : " is not in check"
        console.log(this.chess.turn() + check)
        if (check === " is in check") {
            return this.chess.turn() + check
        }

        console.log(currentBoard)
        this.setBoard(currentBoard)
    }



    isCastle(moveAttempt) {
        const piece = moveAttempt.piece
        const move = {from: moveAttempt.from, to: moveAttempt.to}

        const isBlackCastle = ((move.from === 'e1' && move.to === 'g1') || (move.from
=== 'e1' && move.to === 'c1'))
        const isWhiteCastle = (move.from === 'e8' && move.to === 'g8') || (move.from
=== 'e8' && move.to === 'c8')
```

```
if (!(isWhiteCastle || isBlackCastle) || piece !== 'k') {
    return {
        didCastle: false
    }
}

let originalPositionOfRook
let newPositionOfRook

if ((move.from === 'e1' && move.to === 'g1')) {
    originalPositionOfRook = 'h1'
    newPositionOfRook = 'f1'
} else if ((move.from === 'e1' && move.to === 'c1')) {
    originalPositionOfRook = 'a1'
    newPositionOfRook = 'd1'
} else if ((move.from === 'e8' && move.to === 'g8')) {
    originalPositionOfRook = 'h8'
    newPositionOfRook = 'f8'
} else {
    originalPositionOfRook = 'a8'
    newPositionOfRook = 'd8'
```

```
    }


    return {
        didCastle: true,
        x: this.toAlphabet2[originalPositionOfRook[0]],
        y: this.toCoord2[originalPositionOfRook[1]],
        to_x: this.toAlphabet2[newPositionOfRook[0]],
        to_y: this.toCoord2[newPositionOfRook[1]]
    }
}



isPawnPromotion(to, piece) {
    const res = piece === 'p' && (to[1] === 105 || to[1] === 735)
    if (res) {
        this.nQueens += 1
    }
    return res
}



toChessMove(finalPosition, to2D) {
```

```
        let move

        if (finalPosition[0] > 100) {
            move = this.toAlphabet[to2D[finalPosition[0]]] +
this.toCoord[to2D[finalPosition[1]]]
        } else {
            move = this.toAlphabet[finalPosition[0]] + this.toCoord[finalPosition[1]]
        }

       //  console.log("proposed move: " + move)
        return move
    }

    findPiece(board, pieceId) {
        for (var i = 0; i < 8; i++) {
            for (var j = 0; j < 8; j++) {
                if (board[i][j].getPieceIdOnThisSquare() === pieceId) {
                    return [j, i]
                }
            }
        }
    }
```

```
   makeStartingBoard() {
      const backRank = ["rook", "knight", "bishop", "queen", "king", "bishop", "knight",
"rook"]
      var startingChessBoard = []
      for (var i = 0; i < 8; i++) {
         startingChessBoard.push([])
         for (var j = 0; j < 8; j++) {
            // j is horizontal
            // i is vertical
            const coordinatesOnCanvas = [((j + 1) * 90 + 15), ((i + 1) * 90 + 15)]
            const emptySquare = new Square(j, i, null, coordinatesOnCanvas)

            startingChessBoard[i].push(emptySquare)
         }
      }
      const whiteBackRankId = ["wr1", "wn1", "wb1", "wq1", "wk1", "wb2", "wn2",
"wr2"]
      const blackBackRankId = ["br1", "bn1", "bb1", "bq1", "bk1", "bb2", "bn2", "br2"]
      for (var j = 0; j < 8; j += 7) {
         for (var i = 0; i < 8; i++) {
            if (j == 0) {
               // console.log(backRank[i])
```

```
                startingChessBoard[j][this.thisPlayersColorIsWhite ? i : 7 - i].setPiece(new
ChessPiece(backRank[i], false, this.thisPlayersColorIsWhite ? "black" : "white",
this.thisPlayersColorIsWhite ? blackBackRankId[i] : whiteBackRankId[i]))

                startingChessBoard[j + 1][this.thisPlayersColorIsWhite ? i : 7 -
i].setPiece(new ChessPiece("pawn", false, this.thisPlayersColorIsWhite ? "black" :
"white", this.thisPlayersColorIsWhite ? "bp" + i : "wp" + i))

            } else {

                startingChessBoard[j - 1][this.thisPlayersColorIsWhite ? i : 7 -
i].setPiece(new ChessPiece("pawn", false, this.thisPlayersColorIsWhite ? "white" :
"black", this.thisPlayersColorIsWhite ? "wp" + i : "bp" + i))

                startingChessBoard[j][this.thisPlayersColorIsWhite ? i : 7 - i].setPiece(new
ChessPiece(backRank[i], false, this.thisPlayersColorIsWhite ? "white" : "black",
this.thisPlayersColorIsWhite ? whiteBackRankId[i] : blackBackRankId[i]))

            }

          }

       }

       return startingChessBoard

   }

}


export default Game


class ChessPiece {

   constructor(name, isAttacked, color, id) {

       this.name = name
```

```
        this.isAttacked = isAttacked
        this.color = color
        this.id = id
    }

    setSquare(newSquare) {
        if (newSquare === undefined) {
            this.squareThisPieceIsOn = newSquare
            return
        }

        if (this.squareThisPieceIsOn === undefined) {
            this.squareThisPieceIsOn = newSquare
            newSquare.setPiece(this)
        }

        const isNewSquareDifferent = this.squareThisPieceIsOn.x != newSquare.x ||
this.squareThisPieceIsOn.y != newSquare.y

        if (isNewSquareDifferent) {
            // console.log("set")
            this.squareThisPieceIsOn = newSquare
            newSquare.setPiece(this)
```

```
        }
    }

    getSquare() {
        return this.squareThisPieceIsOn
    }
}


export default ChessPiece
class Square {
    constructor(x, y, pieceOnThisSquare, canvasCoord) {
        this.x = x // Int 0 < x < 7
        this.y = y // Int 0 < y < 7
        this.canvasCoord = canvasCoord
        this.pieceOnThisSquare = pieceOnThisSquare // ChessPiece || null
    }

    setPiece(newPiece) {
        if (newPiece === null && this.pieceOnThisSquare === null) {
            return
        } else if (newPiece === null) {
            // case where the function caller wants to remove the piece that is on this square.
```

```
            this.pieceOnThisSquare.setSquare(undefined)
            this.pieceOnThisSquare = null
        } else if (this.pieceOnThisSquare === null) {
            // case where the function caller wants assign a new piece on this square
            this.pieceOnThisSquare = newPiece
            newPiece.setSquare(this)
        } else if (this.getPieceIdOnThisSquare() != newPiece.id &&
this.pieceOnThisSquare.color != newPiece.color) {
            // case where the function caller wants to change the piece on this square. (only
different color allowed)
            console.log("capture!")
            this.pieceOnThisSquare = newPiece
            newPiece.setSquare(this)
        } else {
            return "user tried to capture their own piece"
        }
    }

    removePiece() {
        this.pieceOnThisSquare = null
    }

    getPiece() {
```

```
        return this.pieceOnThisSquare
    }

    getPieceIdOnThisSquare() {
        if (this.pieceOnThisSquare === null) {
            return "empty"
        }
        return this.pieceOnThisSquare.id
    }

    isOccupied() {
        return this.pieceOnThisSquare != null
    }

    getCoord() {
        return [this.x, this.y]
    }

    getCanvasCoord() {
        return this.canvasCoord
    }
}
```

```
export default Square

import React from 'react'

import Game from '../model/chess'

import Square from '../model/square'

import { Stage, Layer } from 'react-konva';

import Board from '../assets/chessBoard.png'

import useSound from 'use-sound'

import chessMove from '../assets/moveSoundEffect.mp3'

import Piece from './piece'

import piecemap from './piecemap'

import { useParams } from 'react-router-dom'

import { ColorContext } from '../../context/colorcontext'

import VideoChatApp from '../../connection/videochat'

const socket  = require('../../connection/socket').socket



class ChessGame extends React.Component {


    state = {

        gameState: new Game(this.props.color),

        draggedPieceTargetId: "",

        playerTurnToMoveIsWhite: true,

        whiteKingInCheck: false,
```

```
        blackKingInCheck: false
    }


    componentDidMount() {
        console.log(this.props.myUserName)
        console.log(this.props.opponentUserName)
        // register event listeners
        socket.on('opponent move', move => {
            // move == [pieceId, finalPosition]
            // console.log("opponenet's move: " + move.selectedId + ", " +
move.finalPosition)
            if (move.playerColorThatJustMovedIsWhite !== this.props.color) {
                this.movePiece(move.selectedId, move.finalPosition, this.state.gameState,
false)
                this.setState({
                    playerTurnToMoveIsWhite: !move.playerColorThatJustMovedIsWhite
                })
            }
        })
    }

    startDragging = (e) => {
```

```
    this.setState({
        draggedPieceTargetId: e.target.attrs.id
    })
}



movePiece = (selectedId, finalPosition, currentGame, isMyMove) => {
    var whiteKingInCheck = false
    var blackKingInCheck = false
    var blackCheckmated = false
    var whiteCheckmated = false
    const update = currentGame.movePiece(selectedId, finalPosition, isMyMove)

    if (update === "moved in the same position.") {
        this.revertToPreviousState(selectedId)
        return
    } else if (update === "user tried to capture their own piece") {
        this.revertToPreviousState(selectedId)
        return
    } else if (update === "b is in check" || update === "w is in check") {
        if (update[0] === "b") {
            blackKingInCheck = true
        } else {
```

```
            whiteKingInCheck = true
          }
        } else if (update === "b has been checkmated" || update === "w has been
checkmated") {
          if (update[0] === "b") {
            blackCheckmated = true
          } else {
            whiteCheckmated = true
          }
        } else if (update === "invalid move") {
          this.revertToPreviousState(selectedId)
          return
        }
        if (isMyMove) {
          socket.emit('new move', {
            nextPlayerColorToMove: !this.state.gameState.thisPlayersColorIsWhite,
            playerColorThatJustMovedIsWhite:
this.state.gameState.thisPlayersColorIsWhite,
            selectedId: selectedId,
            finalPosition: finalPosition,
            gameId: this.props.gameId
          })
        }
```

```
        this.props.playAudio()
        this.setState({
            draggedPieceTargetId: "",
            gameState: currentGame,
            playerTurnToMoveIsWhite: !this.props.color,
            whiteKingInCheck: whiteKingInCheck,
            blackKingInCheck: blackKingInCheck
        })

        if (blackCheckmated) {
            alert("WHITE WON BY CHECKMATE!")
        } else if (whiteCheckmated) {
            alert("BLACK WON BY CHECKMATE!")
        }
    }


    endDragging = (e) => {
        const currentGame = this.state.gameState
        const currentBoard = currentGame.getBoard()
```

```
        const finalPosition = this.inferCoord(e.target.x() + 90, e.target.y() + 90,
currentBoard)
        const selectedId = this.state.draggedPieceTargetId
        this.movePiece(selectedId, finalPosition, currentGame, true)
    }

    revertToPreviousState = (selectedId) => {
        const oldGS = this.state.gameState
        const oldBoard = oldGS.getBoard()
        const tmpGS = new Game(true)
        const tmpBoard = []

        for (var i = 0; i < 8; i++) {
            tmpBoard.push([])
            for (var j = 0; j < 8; j++) {
                if (oldBoard[i][j].getPieceIdOnThisSquare() === selectedId) {
                    tmpBoard[i].push(new Square(j, i, null, oldBoard[i][j].canvasCoord))
                } else {
                    tmpBoard[i].push(oldBoard[i][j])
                }
            }
        }
        tmpGS.setBoard(tmpBoard)
```

```
    this.setState({
        gameState: tmpGS,
        draggedPieceTargetId: "",
    })

    this.setState({
        gameState: oldGS,
    })
}


inferCoord = (x, y, chessBoard) => {
    var hashmap = {}
    var shortestDistance = Infinity
    for (var i = 0; i < 8; i++) {
        for (var j = 0; j < 8; j++) {
            const canvasCoord = chessBoard[i][j].getCanvasCoord()
            // calculate distance
            const delta_x = canvasCoord[0] - x
            const delta_y = canvasCoord[1] - y
            const newDistance = Math.sqrt(delta_x**2 + delta_y**2)
            hashmap[newDistance] = canvasCoord
```

```
            if (newDistance < shortestDistance) {
                shortestDistance = newDistance
            }
        }
    }

    return hashmap[shortestDistance]
}

render() {
    return (
    <React.Fragment>
    <div style = {{
        backgroundImage: `url(${Board})`,
        width: "720px",
        height: "720px",
        position: "relative",
        left: "30%",
        top: "50%", // Centered vertically
    }}
    >
        <Stage width = {720} height = {720}>
            <Layer>
```

```
{this.state.gameState.getBoard().map((row) => {
    return (<React.Fragment>
        {row.map((square) => {
            if (square.isOccupied()) {
                return (
                    <Piece
                        x = {square.getCanvasCoord()[0]}
                        y = {square.getCanvasCoord()[1]}
                        imgurls = {piecemap[square.getPiece().name]}
                        isWhite = {square.getPiece().color === "white"}
                        draggedPieceTargetId =
{this.state.draggedPieceTargetId}
                        onDragStart = {this.startDragging}
                        onDragEnd = {this.endDragging}
                        id = {square.getPieceIdOnThisSquare()}
                        thisPlayersColorIsWhite = {this.props.color}
                        playerTurnToMoveIsWhite =
{this.state.playerTurnToMoveIsWhite}
                        whiteKingInCheck = {this.state.whiteKingInCheck}
                        blackKingInCheck = {this.state.blackKingInCheck}
                        />)
            }
            return
```

```
                        })}
                    </React.Fragment>)
                })}
            </Layer>
        </Stage>
    </div>
    </React.Fragment>)
  }
}



const ChessGameWrapper = (props) => {
  const domainName = 'http://localhost:3000'
  const color = React.useContext(ColorContext)
  const { gameid } = useParams()
  const [play] = useSound(chessMove);
  const [opponentSocketId, setOpponentSocketId] = React.useState('')
  const [opponentDidJoinTheGame, didJoinGame] = React.useState(false)
  const [opponentUserName, setUserName] = React.useState('')
  const [gameSessionDoesNotExist, doesntExist] = React.useState(false)

  React.useEffect(() => {
```

```
        socket.on("playerJoinedRoom", statusUpdate => {
            console.log("A new player has joined the room! Username: " + statusUpdate.userName + ", Game id: " + statusUpdate.gameId + " Socket id: " + statusUpdate.mySocketId)
            if (socket.id !== statusUpdate.mySocketId) {
                setOpponentSocketId(statusUpdate.mySocketId)
            }
        })

        socket.on("status", statusUpdate => {
            console.log(statusUpdate)
            alert(statusUpdate)
            if (statusUpdate === 'This game session does not exist.' || statusUpdate === 'There are already 2 people playing in this room.') {
                doesntExist(true)
            }
        })

        socket.on('start game', (opponentUserName) => {
            console.log("START!")
            if (opponentUserName !== props.myUserName) {
                setUserName(opponentUserName)
                didJoinGame(true)
```

```
        } else {
            socket.emit('request username', gameid)
        }
    })


    socket.on('give userName', (socketId) => {
        if (socket.id !== socketId) {
            console.log("give userName stage: " + props.myUserName)
            socket.emit('recieved userName', {userName: props.myUserName, gameId:
gameid})
        }
    })

    socket.on('get Opponent UserName', (data) => {
        if (socket.id !== data.socketId) {
            setUserName(data.userName)
            console.log('data.socketId: data.socketId')
            setOpponentSocketId(data.socketId)
            didJoinGame(true)
        }
    })
  }, [])
```

```
return (
  <React.Fragment>
    {opponentDidJoinTheGame ? (
      <div>
        <h4> Opponent: {opponentUserName} </h4>
        <div style={{ display: "flex" }}>
          <ChessGame
            playAudio={play}
            gameId={gameid}
            color={color.didRedirect}
          />
          <VideoChatApp
            mySocketId={socket.id}
            opponentSocketId={opponentSocketId}
            myUserName={props.myUserName}
            opponentUserName={opponentUserName}
          />
        </div>
        <h4> You: {props.myUserName} </h4>
      </div>
    ) : gameSessionDoesNotExist ? (
```

```
        <div>
          <h1 style={{ textAlign: "center", marginTop: "200px" }}> :( </h1>
        </div>
      ) : (
        <div>
          <h1
            style={{
              textAlign: "center",
              marginTop: String(window.innerHeight / 8) + "px",
            }}
          >
            Hey <strong>{props.myUserName}</strong>, copy and paste the URL
            below to send to your friend:
          </h1>
          <textarea
            style={{ marginLeft: String((window.innerWidth / 2) - 290) + "px",
marginTop: "30" + "px", width: "580px", height: "30px"}}
            onFocus={(event) => {
               console.log('sd')
               event.target.select()
            }}
            value = {domainName + "/game/" + gameid}
            type = "text">
```

```
            </textarea>
          <br></br>

          <h1 style={{ textAlign: "center", marginTop: "100px" }}>
            {" "}
            Waiting for other opponent to join the game...{" "}
          </h1>
        </div>
      )}
    </React.Fragment>
  );
};

export default ChessGameWrapper
import React from 'react'
import { Image } from 'react-konva';
import useImage from 'use-image'

const Piece = (props) => {
  const choiceOfColor = props.isWhite ? 0 : 1
  const [image] = useImage(props.imgurls[choiceOfColor]);
  const isDragged = props.id === props.draggedPieceTargetId
```

```
    const canThisPieceEvenBeMovedByThisPlayer = props.isWhite ===
props.thisPlayersColorIsWhite

    const isItThatPlayersTurn = props.playerTurnToMoveIsWhite ===
props.thisPlayersColorIsWhite


    const thisWhiteKingInCheck = props.id === "wk1" && props.whiteKingInCheck

    const thisBlackKingInCheck = props.id === "bk1" && props.blackKingInCheck


    return <Image image={image}

        x = {props.x - 90}

        y = {props.y - 90}

        draggable = {canThisPieceEvenBeMovedByThisPlayer && isItThatPlayersTurn}

        width = {isDragged ? 75 : 60}

        height = {isDragged ? 75 : 60}

        onDragStart = {props.onDragStart}

        onDragEnd = {props.onDragEnd}

        fill = {(thisWhiteKingInCheck && "red") || (thisBlackKingInCheck && "red")}

        id = {props.id}

        />;

};


export default Piece

export default {
```

'pawn': ['https://upload.wikimedia.org/wikipedia/commons/0/04/Chess_plt60.png', 'https://upload.wikimedia.org/wikipedia/commons/c/cd/Chess_pdt60.png'],

'knight':['https://upload.wikimedia.org/wikipedia/commons/2/28/Chess_nlt60.png','https://upload.wikimedia.org/wikipedia/commons/f/f1/Chess_ndt60.png'],

'bishop':['https://upload.wikimedia.org/wikipedia/commons/9/9b/Chess_blt60.png','https://upload.wikimedia.org/wikipedia/commons/8/81/Chess_bdt60.png'],

'king':['https://upload.wikimedia.org/wikipedia/commons/3/3b/Chess_klt60.png','https://upload.wikimedia.org/wikipedia/commons/e/e3/Chess_kdt60.png'],

'queen':['https://upload.wikimedia.org/wikipedia/commons/4/49/Chess_qlt60.png','https://upload.wikimedia.org/wikipedia/commons/a/af/Chess_qdt60.png'],

'rook':['https://upload.wikimedia.org/wikipedia/commons/5/5c/Chess_rlt60.png','https://upload.wikimedia.org/wikipedia/commons/a/a0/Chess_rdt60.png']

}