Project Idea:

ChessSup is an innovative online chess platform where players can engage in real-time multiplayer matches, bringing the timeless game of chess to life in the digital world. Through a user-friendly interface, powered by cutting-edge technologies like Node.js and SocketIO for instant communication, MongoDB for efficient data storage, and React.js for seamless frontend interaction, ChessSup offers an immersive gaming experience accessible to players of all skill levels. Players can challenge opponents from around the globe, make moves, and see them instantly reflected on their opponent's board, bringing up an atmosphere of excitement and competition. In addition to multiplayer matches, ChessSup features chat functionality for player interaction, a replay mode with undo/redo options for reviewing past games, and a vibrant community page where players can connect, share strategies, and build friendships. Whether you're a seasoned chess grandmaster or a novice eager to learn, ChessSup provides a dynamic platform for enjoying the strategic depth and timeless appeal of chess in a modern, accessible format.

Requirements:

- Real-time online multiplayer chess gameplay
- Session-based authentication
- Efficient network bandwidth usage
- Chess game logic implementation
- Replay mode with undo/redo functionalities
- Chat feature for players and spectators
- Community page for interaction
- Personalization features

Technical Stack:

- Backend
    - Node.js
    - Express
    - Session-based authentication
        - Express-Session
    - SocketIO
    - Nginx
    - MongoDB
        - Mongoose
- Frontend
    - Single Page Application (SPA)
    - React.js
    - Redux
        - Thunk
        - Saga

Architecture:

Based on the provided information, here's an architecture flow for the ChessSup project:

1. **Client Side (Frontend) Flow:**
    - The client-side is implemented as a Single Page Application (SPA) using React.js.
    - Redux is used for state management.
    - Thunk and Saga are middleware libraries for Redux, handling asynchronous actions such as API calls and side effects.
    - The UI is updated in real-time using SocketIO for receiving updates from the server.

- Multiple gameplay from multiple tabs in the browser is supported, with shared authentication among tabs.

2. **Server Side (Backend) Flow:**
    - Backend is implemented using Node.js and Express.
    - Session-based authentication is used for user authentication, managed by Express-Session.
    - MongoDB is used as the database, with Mongoose as the ODM.
    - SocketIO is used for real-time communication between the server and clients.
    - Nginx is used as a reverse proxy server.
    - AWS EC2 is used for hosting the application.
3. **Gameplay Flow:**
    - The game supports multiplayer, requiring at least two players.
    - Players interact with the game through HTTP calls.
    - SocketIO is used for real-time updates, enabling players to see each other's moves in real-time.
    - Game rules are enforced by algorithms implemented on the server, covering movements like check, checkmate, stalemate, promotion, etc.
    - All logic related to winning or failing is managed by the server and reflected in the game record after server validation.
    - Players can make requests such as surrender, draw, or rollback during the game.
4. **Optimization:**
    - To conserve network bandwidth, SocketIO's namespace area is divided on every page.
    - Only the movement of the piece is transferred over the network when updating the chessboard, minimizing data transfer.
    - Partial updates to the view are made to minimize React reconciliation.

5. **Additional Features:**
    - Utility functions include flipping the chessboard, requesting surrender, draw, or rollback.
    - Replay functionality allows players to redo, undo, fast redo, and fast undo moves.
    - Ranking page shows real-time rankings, and recently played games are displayed on the main page.
    - Community page is designed as a REST API for basic bulletin board functionality.

Design:

The backend utilizes Node.js with Express for handling HTTP requests and SocketIO for real-time communication between clients and the server. MongoDB with Mongoose is used for data storage, and AWS EC2 instances host the application. Nginx is employed as a reverse proxy for efficient load balancing.

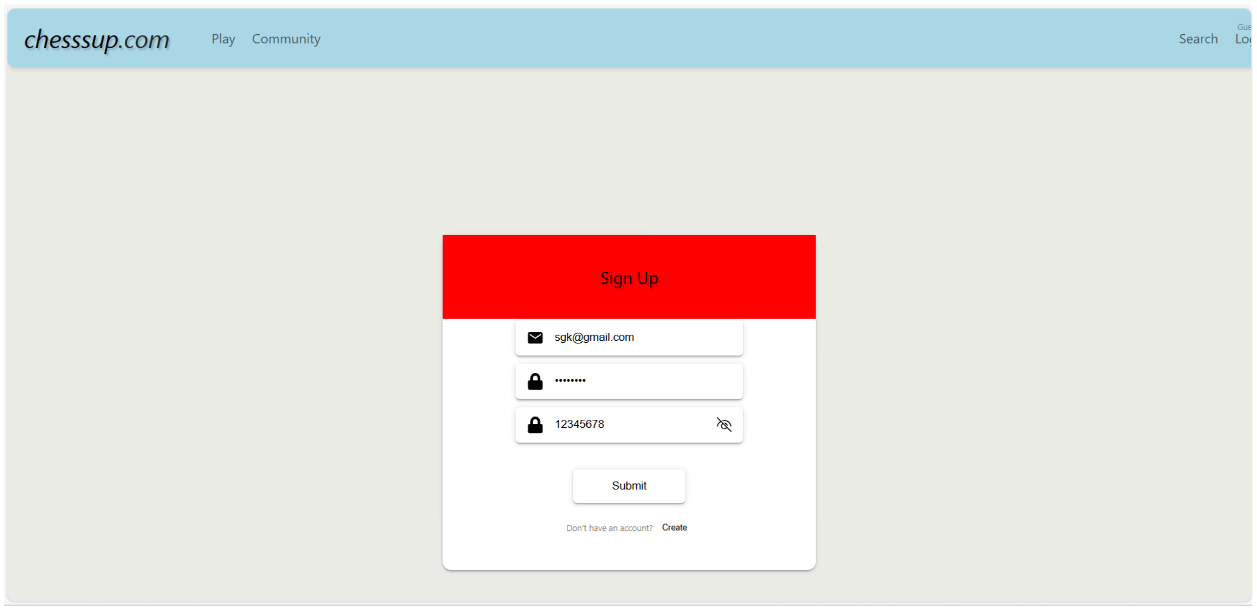
The frontend is built as a React.js SPA, ensuring a responsive and interactive user experience. Redux manages the application state, while middleware like Thunk and Saga handle asynchronous actions. The design focuses on scalability, performance, and real-time updates to provide a seamless gaming experience.

- How this design works
    - How to update View
        - All requests from client side are made through http call
        - All response from server side are made through SocketIo
        - This allows multiple gameplay from multiple tabs in the browser
        - The authentication is shared among the tabs of the browser, but the service is provided individually for each tab
    - Optimization
        - To conserve network bandwidth, socketio's namespace area is divided on every page

        - When updating the chessboard, Only the movement of the piece is transferred to the network
        - By not receiving the entire board, you can partially update the view and minimize the reconciliation of React
- Game Rules
    - Process
        - Currently, the service only supports multiplayer. Therefore, at least two players are required to play the game
        - The game has a default time and a recharge time that is filled when you turn over
        - In-game users are divided into players (black and white) and spectators
        - You can play chess only during your turn, and spectators can only watch and chat
    - Algorithm
        - Game rules are driven by algorithms that i create
        - All movement of chess such as check, checkmate, stalemate, promotion, etc. are provided
        - All logic related to winning or failing will be reflected in the record after it is finally approved by the server
- Functions
    - Utility
        - It is possible to flip the chess board (At first, the chess piece you grab is facing down)
        - Request for surrender, request to draw, or request to roll back is now available
    - Replay
        - Redo, Undo, FastRedo, FastUndo is now available
        - It is also possible to view a specific board state in replay mode by pressing a notation
- ETC
    - Main Page
        - You can check the ranking page in real time, and you can see recently played games next to it
    - Community Page

- It is designed as a REST API and is a community in the form of a basic bulletin board.

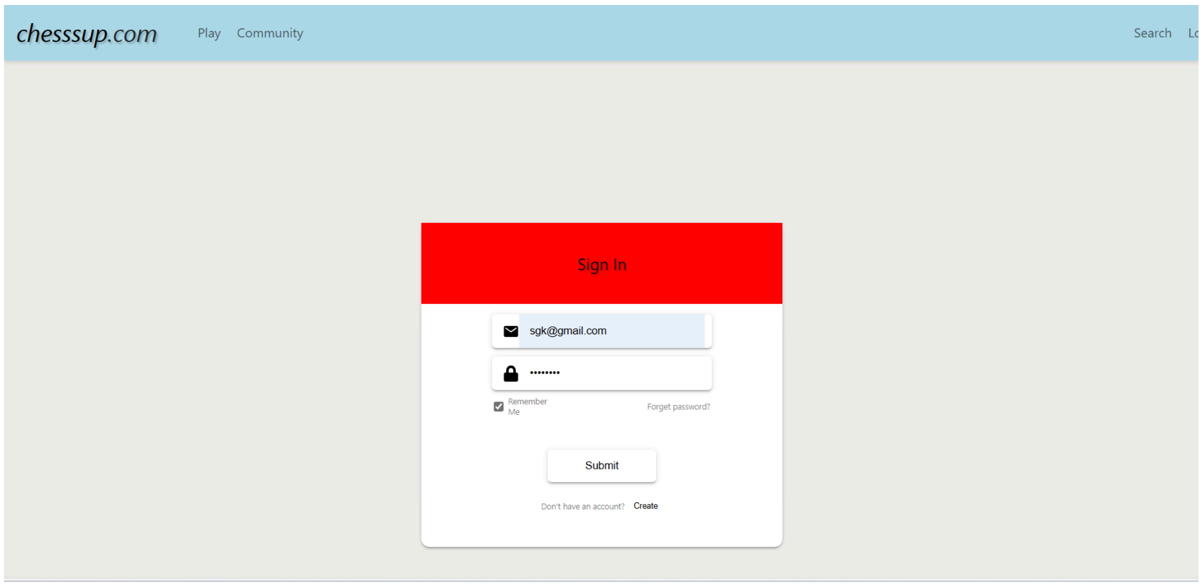
Screenshots:



sgk@gmail.com

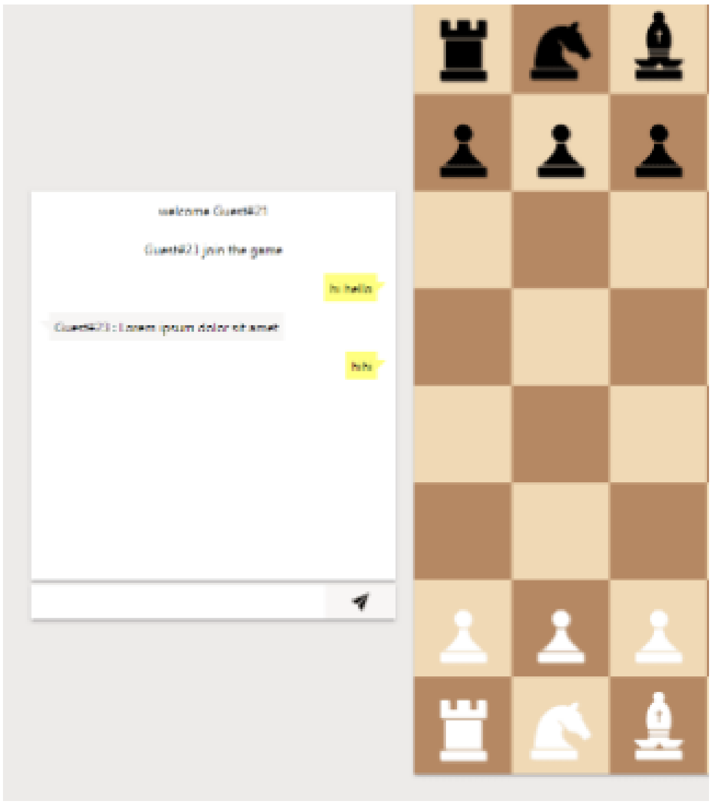
••••••••

12345678



sgk@gmail.com

••••••••


welcome Guest#21

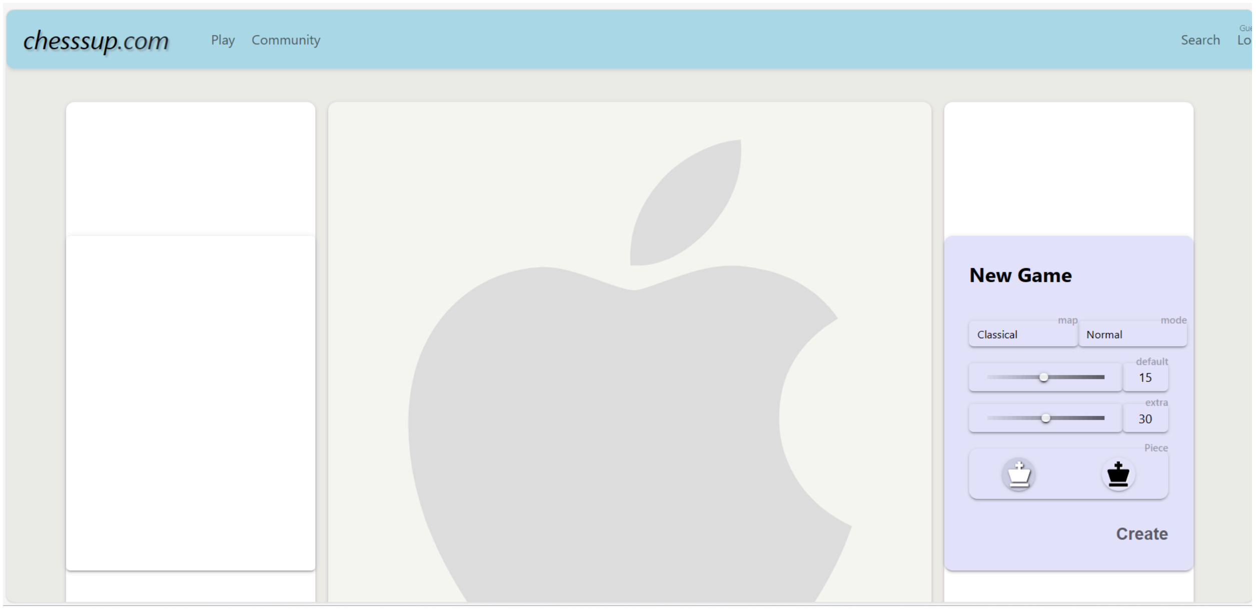
Guest#23 join the game

hi hello

Guest#23 : Lorem ipsum dolor sit amet

hihi

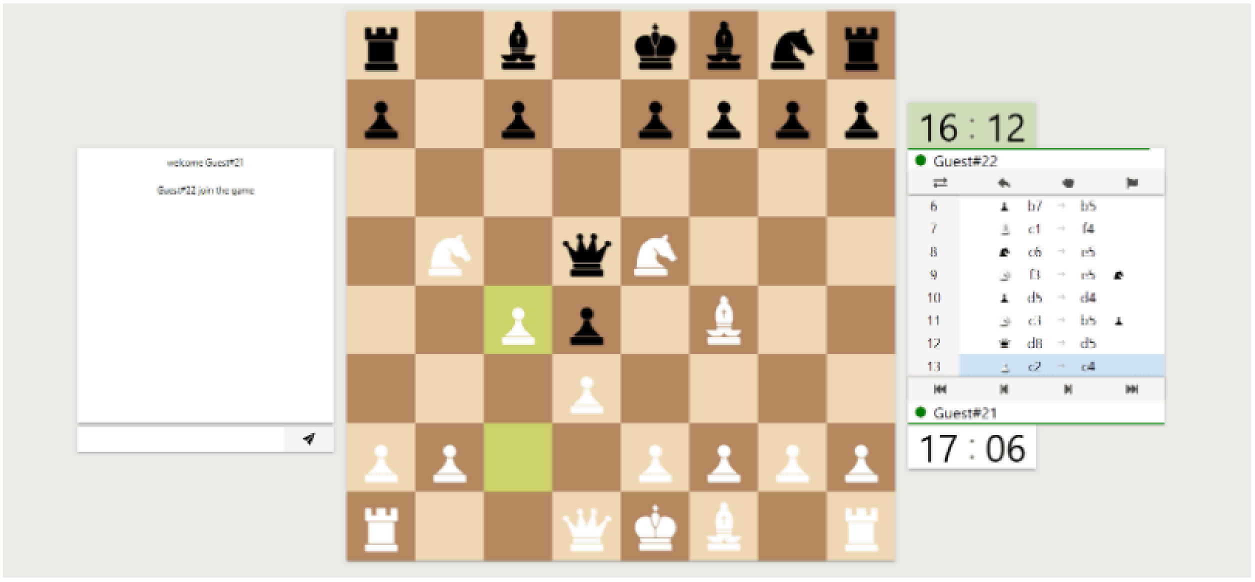
(chat)



New Game

map: Classical
mode: Normal
default: 15
extra: 30
Piece

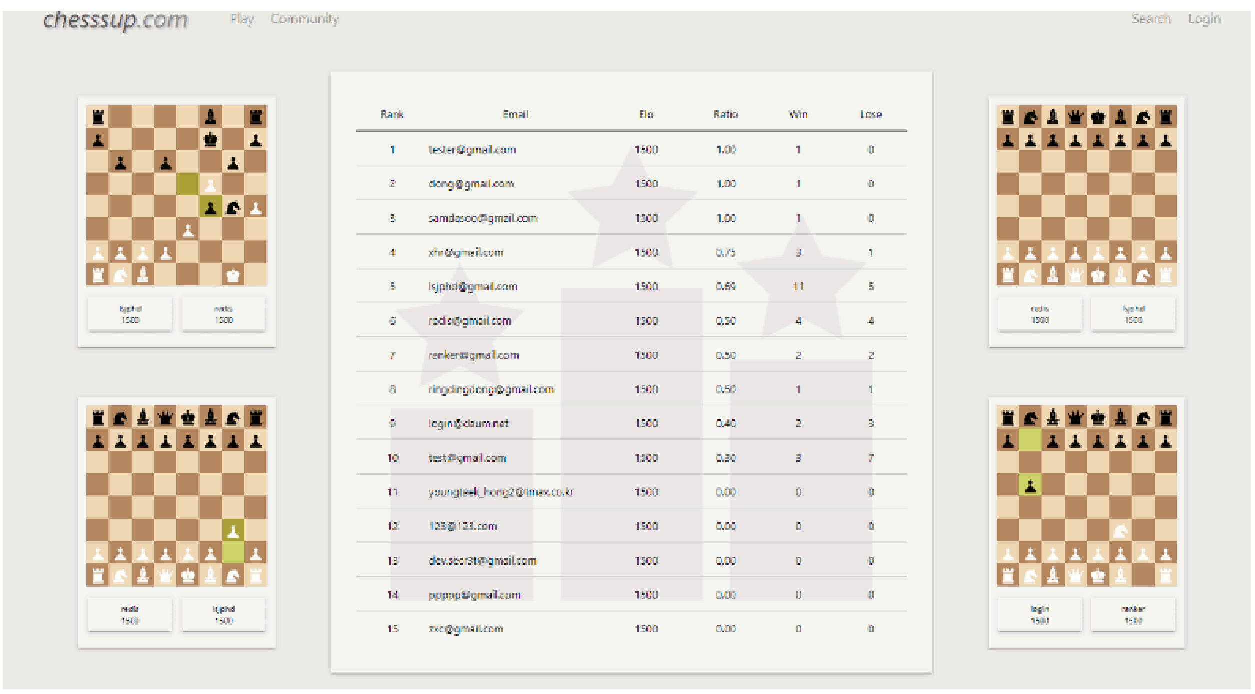
Create



welcome Guest#21

Guest#22 join the game

```
6   b7 → b5
7   c1 → f4
8   c6 → e5
9   f3 → e5
10  d5 → d4
11  c3 → b5
12  d8 → d5
13  c2 → c4
```



Rank	Email	Elo	Ratio	Win	Lose
1	tester@gmail.com	1500	1.00	1	0
2	dong@gmail.com	1500	1.00	1	0
3	samdasoo@gmail.com	1500	1.00	1	0
4	xhr@gmail.com	1500	0.75	3	1
5	lsjphd@gmail.com	1500	0.69	11	5
6	redis@gmail.com	1500	0.50	4	4
7	ranker@gmail.com	1500	0.50	2	2
8	ringdingdong@gmail.com	1500	0.50	1	1
9	login@daum.net	1500	0.40	2	3
10	test@gmail.com	1500	0.30	3	7
11	youngteek_hong2@tmax.co.kr	1500	0.00	0	0
12	123@123.com	1500	0.00	0	0
13	dev.secr3t@gmail.com	1500	0.00	0	0
14	ppppp@gmail.com	1500	0.00	0	0
15	zxc@gmail.com	1500	0.00	0	0