Connect 4

GameHub

Product Owner: Benito Reyes

Scrum Master: BranDon Brown/Mekhi Green

Developers: Benito Reyes,BranDon Brown,Mekhi Green, Ramsay Burls

Sprint Goal

Deliver a seamless gameplay experience by implementing interactive game logic and server synchronization, while introducing a secure login screen to support future multi-user functionality.

Stories Delivered

01

As a user, I want to click a “play again” button, so that I can quickly start a rematch.

Goal: Functional “Play Again” button that clears the board and starts a new match while keeping scores.

02

As a user, I want to see a message when someone wins or the game is a draw, so that I understand the outcome.

Goal: Display end-game messages (e.g., “Player 1 Wins!”, “Draw!”) near the board.

03

As a user, I want to log into my account so that I can securely access my games. And have my data be stored securely

Goal: The user can successfully enter valid credentials on the login page and be redirected to the game page, confirming secure authentication and session access.

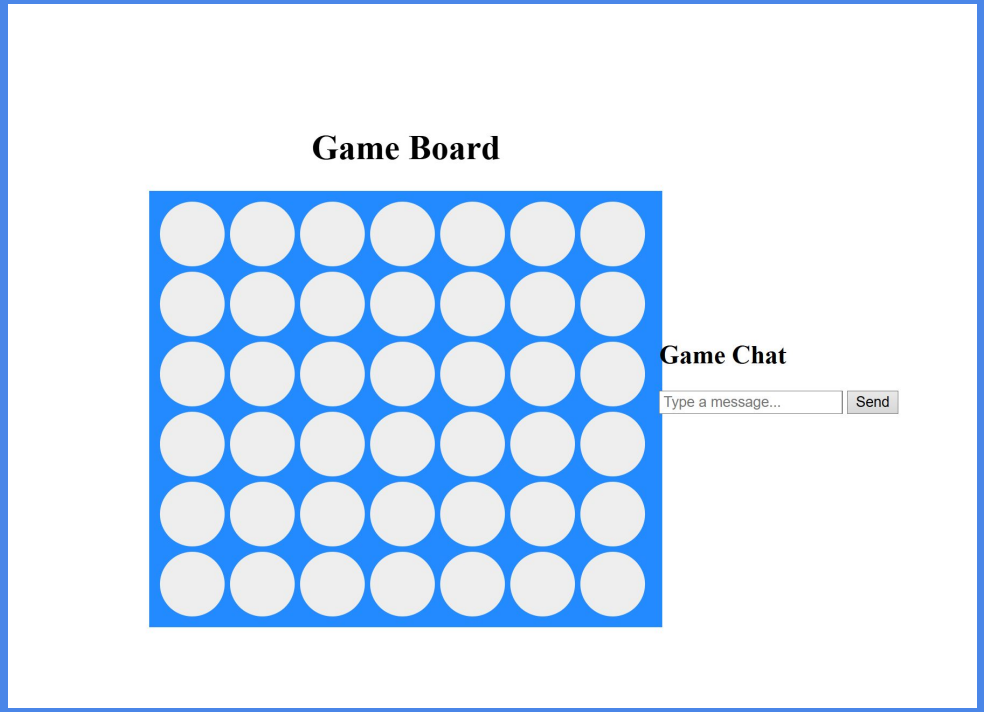
04

As a user, I want to send and receive live chat messages during a match so that I can communicate with my opponent in real time.

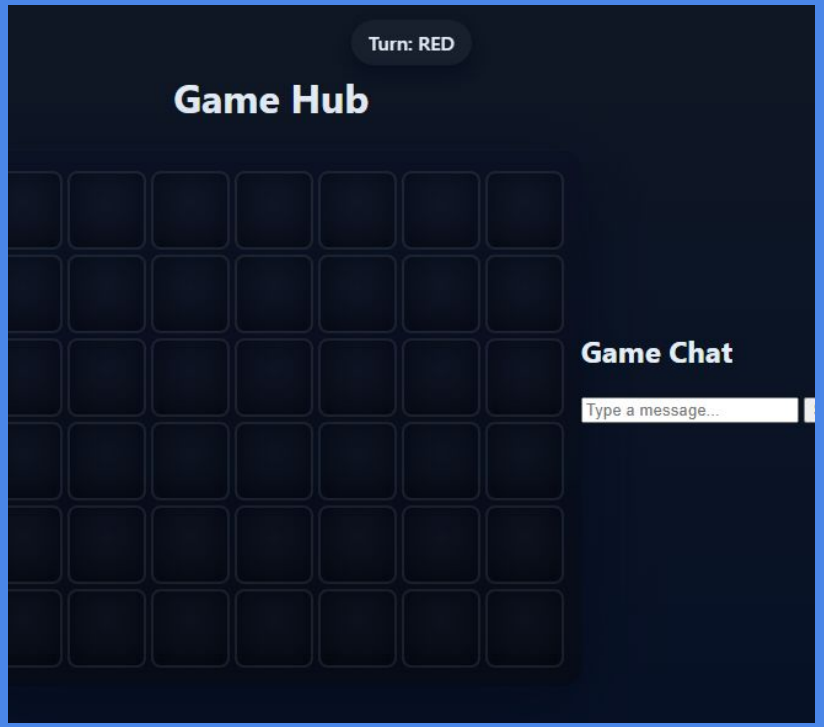
Goal: Players can exchange live messages in real time within the game interface, and messages appear without refreshing.

Updated Design Artifacts

Sprint 1



Sprint 2



Architecture Overview

Game Flow

1. Player connects via Socket.IO.
2. Server assigns role (`red` or `yellow`) and tracks players.
3. Players take turns dropping pieces on the board.
4. Moves are broadcast to the opponent via Socket.IO.
5. Win condition is checked locally and announced.

Chat Flow

1. Backend generates StreamChat token using `socket.id` as `userId`.
2. Token and userId are sent to frontend via `chat-auth` event.
3. Frontend connects to StreamChat using `connectUser()`.
4. A `gaming` channel is created or joined.
5. Players can send and receive messages in real-time.

Security & Environment

- `.env` file stores API secrets (never committed)
- StreamChat token is generated server-side only
- Frontend receives only the token and userId through requesting get from backend

Dependencies

- `express` — initial HTTP server
- `socket.io` — real-time game communication
- `stream-chat` — chat SDK (client + server)
- `dotenv` — environment variable management
- 'Node' — another server for browser testing

Testing & CI

- Manual testing via browser

[Database ERD.pdf](https://github.com/user-attachments/files/22752355/Database.ERD.pdf)

3. Players take turns dropping pieces on the board.
4. Moves are broadcast to the opponent via Socket.IO.
5. Win condition is checked locally and announced.

Chat Flow

1. Backend generates StreamChat token using `socket.id` as `userId`.
2. Token and userId are sent to frontend via `chat-auth` event.
3. Frontend connects to StreamChat using `connectUser()`.
4. A `gaming` channel is created or joined.
5. Players can send and receive messages in real-time.

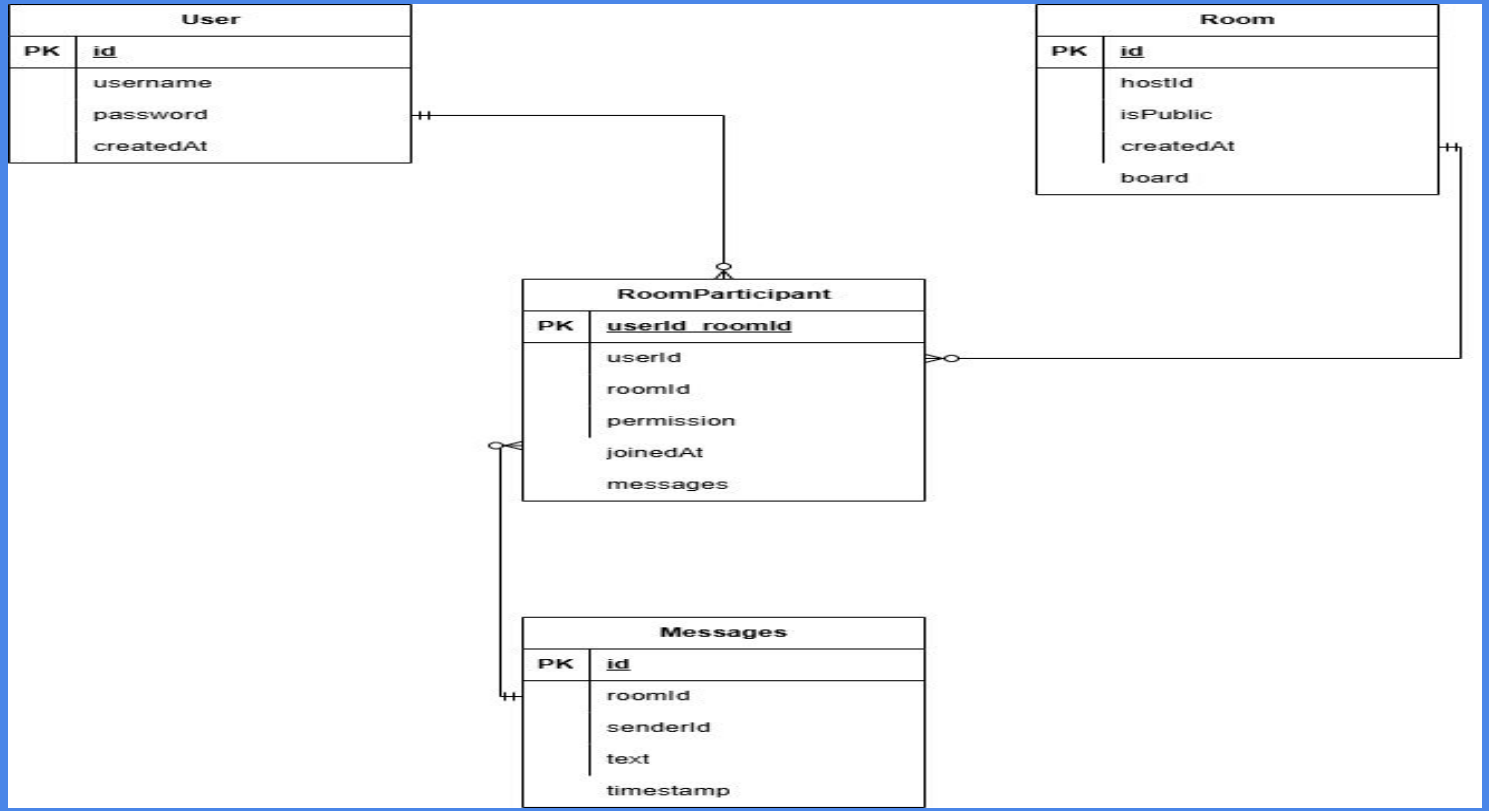
Security & Environment

- `.env` file stores API secrets (never committed)
- StreamChat token is generated server-side only
- Frontend receives only the token and userId through requesting get from backend
- passwords are encrypted

Dependencies

- `express` — initial HTTP server
- `socket.io` — real-time game communication
- `stream-chat` — chat SDK (client + server)
- `dotenv` — environment variable management
- 'Node' — another server for browser testing
- 'bcrypt' — used to encrypt password
- 'cookie' — used to set userId and token cookies
- 'neon' — the postgresql database being used
- 'uuid' used to generate random userId's and roomId's
- 'vite' — used to package streamchat bundle for browser/html usage
- 'prisma' — used to query the database more easily

ERD


User
PK
id
username
password
createdAt
Room
PK
id
hostId
isPublic
createdAt
board
RoomParticipant
PK
userId roomId
userId
roomId
permission
joinedAt
messages
Messages
PK
id
roomId
senderId
text
timestamp

Demo Link