CA2

Connect 4 (Multiplayer Game with Socket IO)

A multiplayer Connect4 game built with React, Node and Socket IO

Running live on : https://connectquattros.netlify.app/

Backend Repo : https://github.com/thisisanish/Connect4-BackEnd

Frontend Repo : https://github.com/thisisanish/Connect4-FrontEnd

Anish Agarwal

11707587

Introduction

This Project is built to demonstrate the use of Socket IO (A web socket implementation) in areas apart from chat apps. This project is a popular game of Connect 4/ 4-in-a-line/etc where the aim is to connect the 4 pieces in a line like tic tac toe but is a multiplayer variant !. The catch is that unlike tic-tac-toe, the chips/coins/piece fall to the bottom of the stack.

This simple project uses :-

Node : Environment to Run JS

Express: A backend Framework

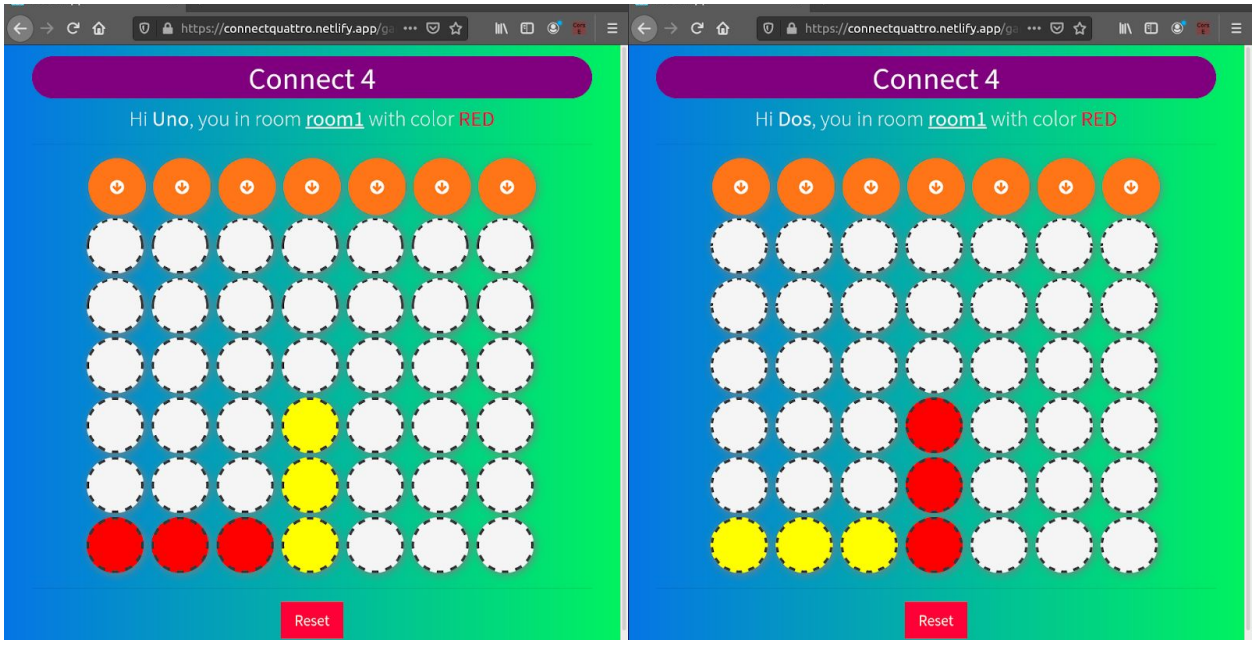
React: A Frontend framework

Socket Io: A web socket implementation (For both, server and client) [The secret to multiplayer]

Bootstrap: For fast development of components and making it responsive

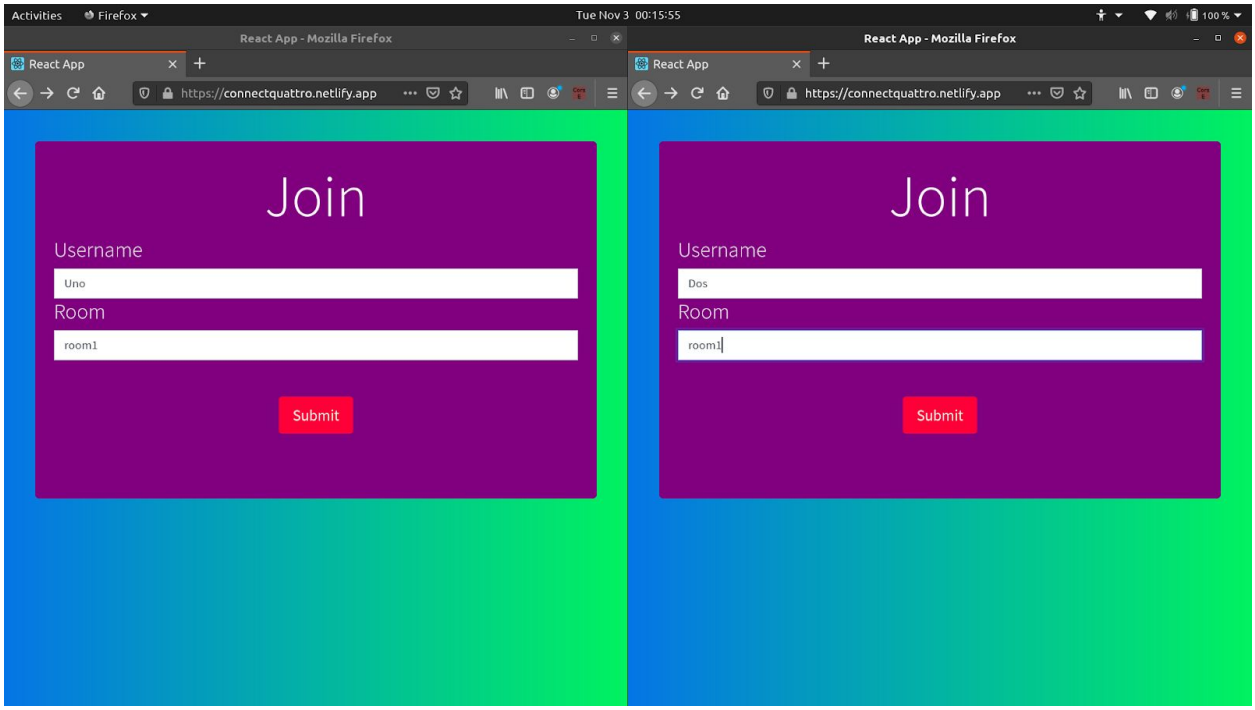
Netlify: To host React App

Heroku : To host the Backend



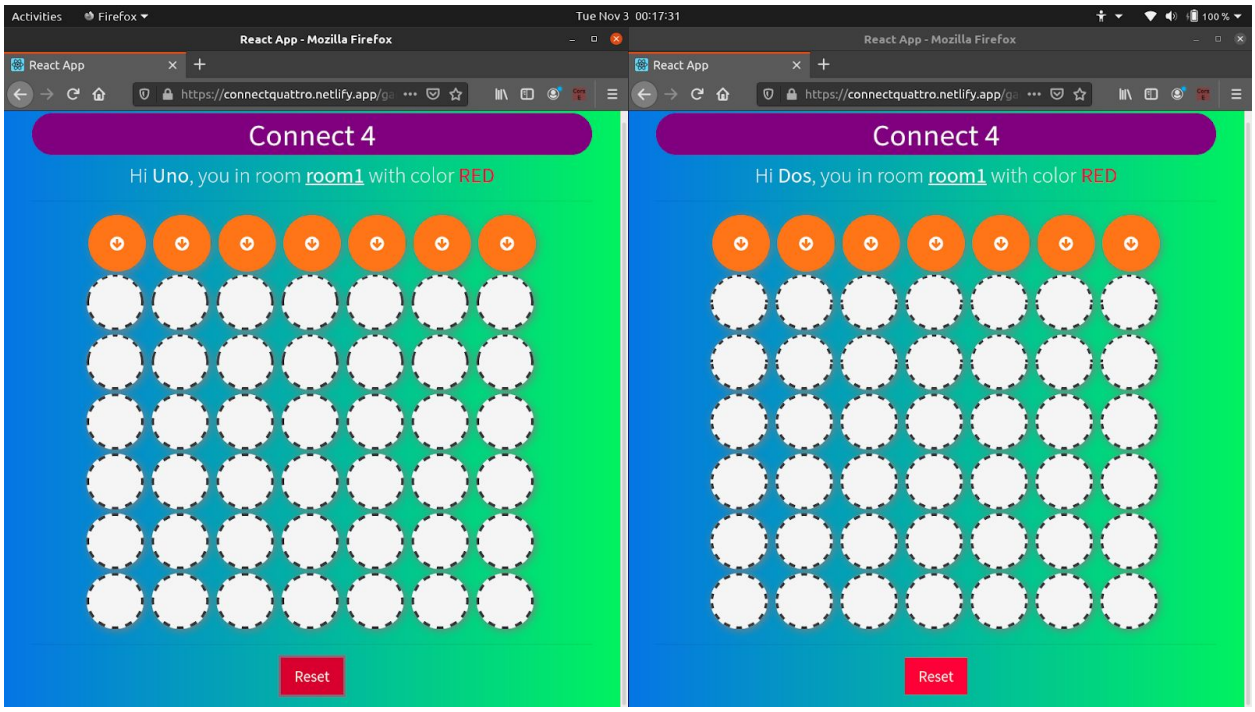
The Flow of the Program from the User's POV

1) Two Users Enter the game by entering their preferred username and common room they want to join.



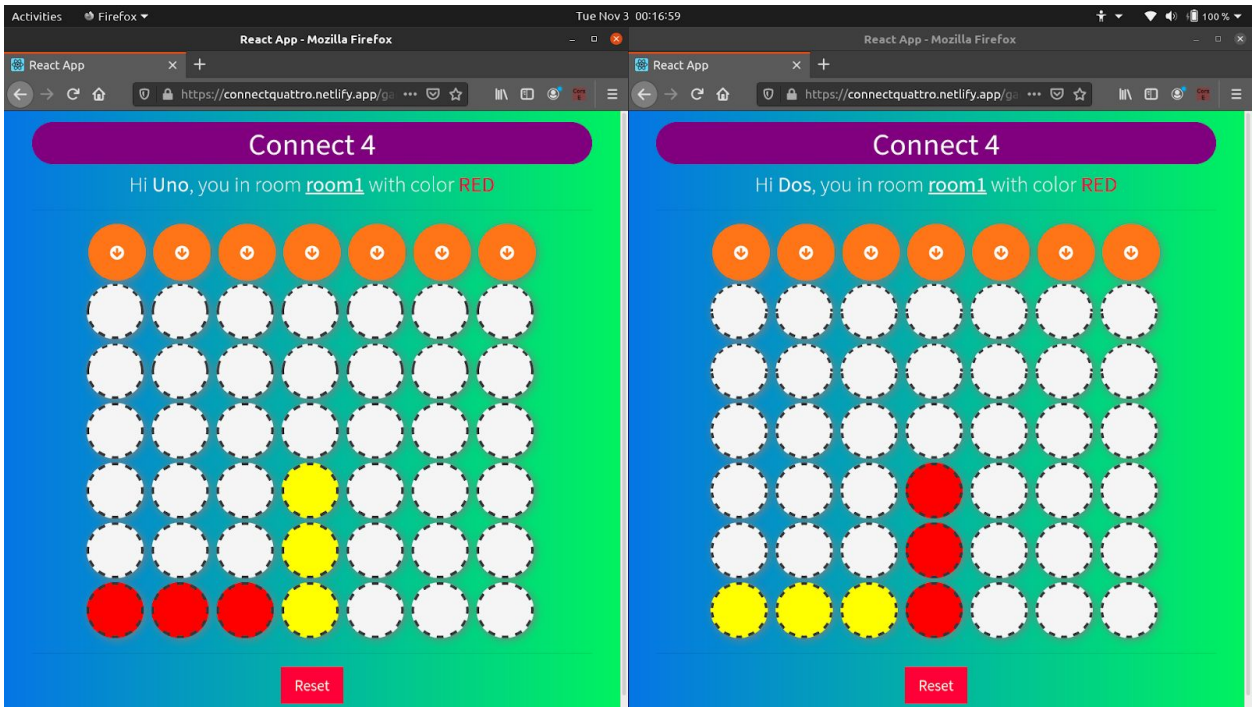
This is running on a webserver and the users can use any browser to join the game and they could play with each other as long as they both enter the same room.

2) Once they enter, A blank canvas appears with crucial information.



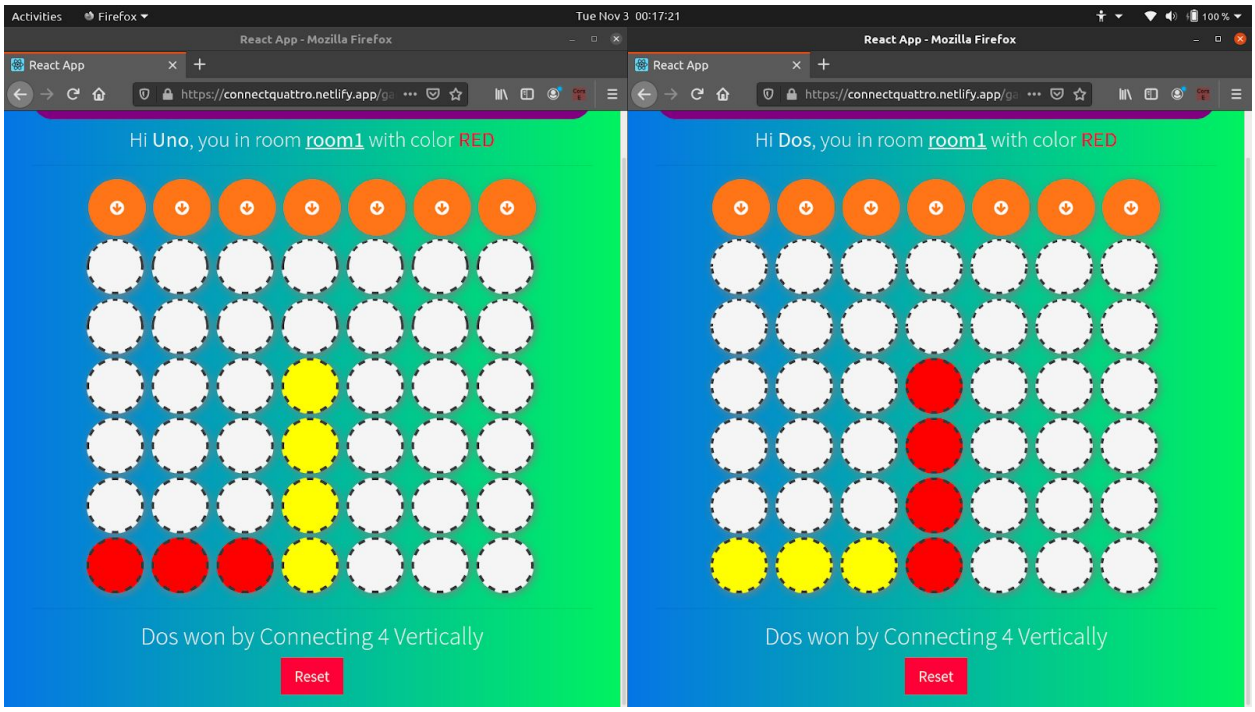
The both have same color, to easy the complexity for the programming. Despite having the same color (RED) the opponent would have a different color from each of their respect.

3) And the Game Starts



The Opponent is Always Yellow and the one playing is always Red. Aim is to click the Orange Button and drop a coin and try to connect 4 of the same color that belongs to you.

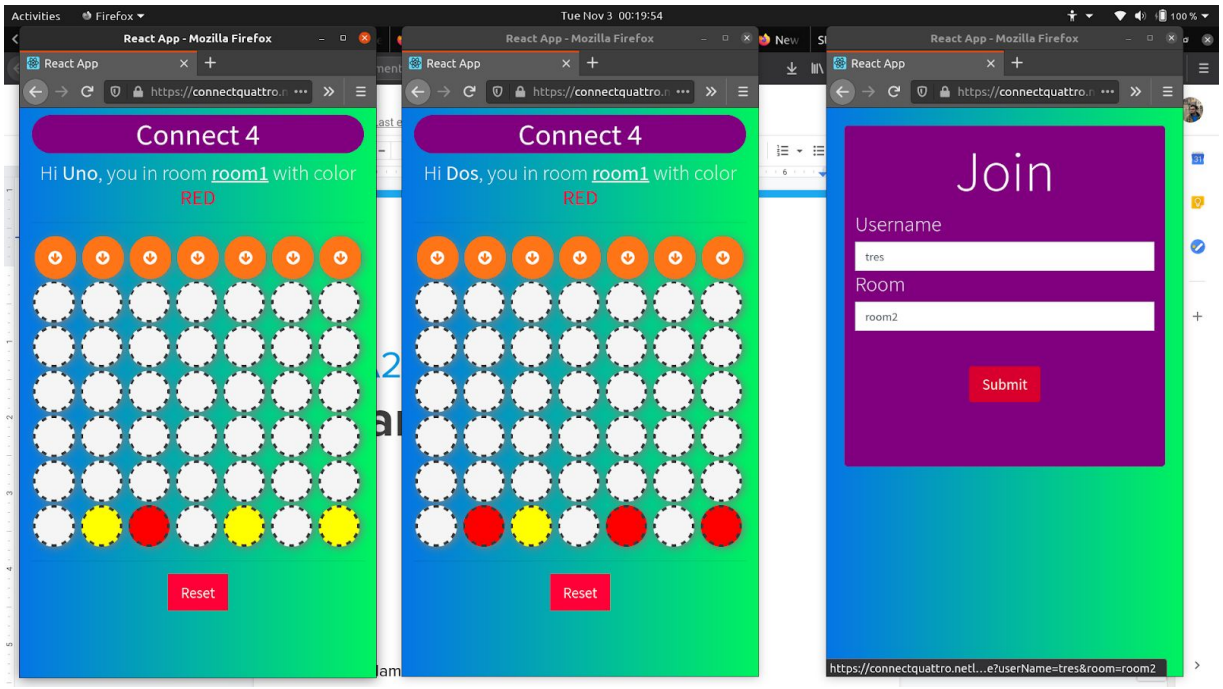
4) Connect 4 !

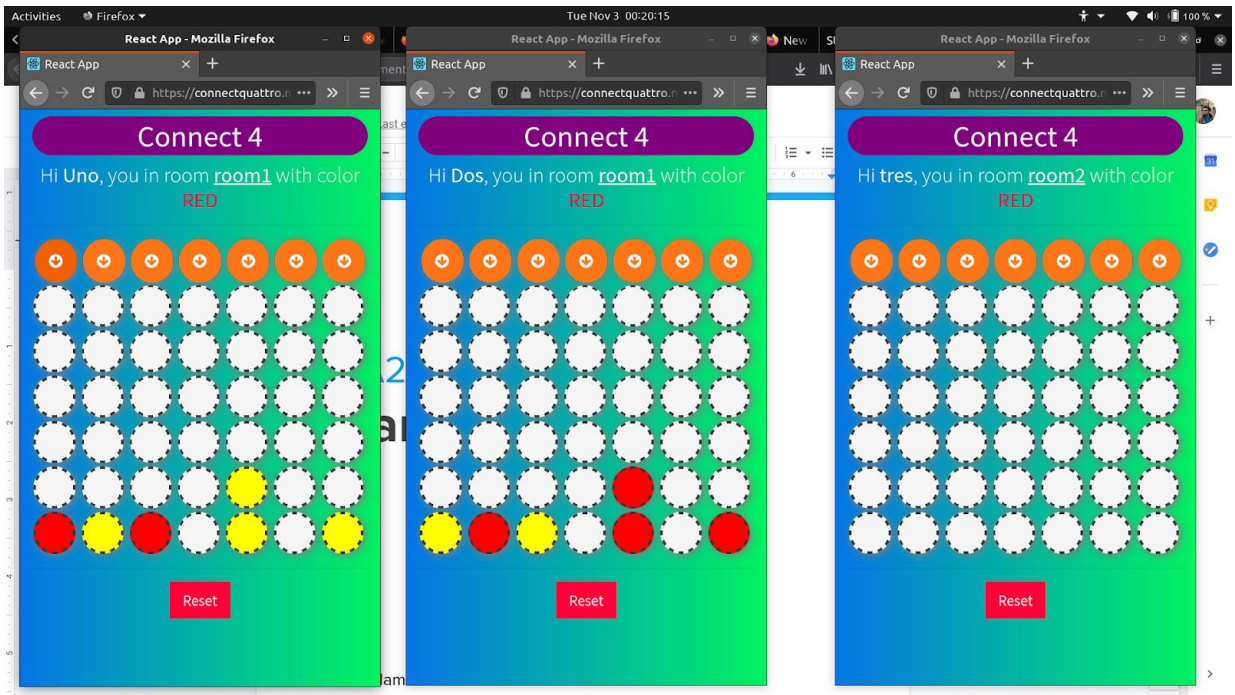


The Player **Dos** has connected 4 Vertically and won the game. The game would restart if someone plays another move or can be re-started by pressing Reset which resets the board.

5) Proof of Multiple sockets (rooms)

Another person can also play joining another room and would be greeted with a new board. He/she can invite his pals/buddies/friends and play along. All the rooms are independent of each other





Backend (Node and Socket)

The Backend Code is divided into 4 parts. Import s and middlewares, Express, Game Logic and Socket.

Imports and middlewares

```
const express = require('express'),
    app = express()
    server = require('http').createServer(app),
    port = process.env.PORT || 3001,
    io = require('socket.io')(server,{ origins: '*:*'}),
    cors = require('cors');
app.use(cors())
```

Sockets

```
io.on('connection',(socket)⇒{

    socket.on('join',({userName,room},cb)⇒{
        // socket.emit('yo',{yo:"yo"})
        console.log(socket.id,userName,room);
        socket.join(room)
        // console.log(gameArea[room]);
        loadData(userName,room)
        io.to(room).emit('game',gameArea[room])

        // console.log(gameArea);
        })

```

On ‘connection’ and on join ‘Listens the the events and registers the players in the game and create the board

```
socket.on('playPosition',({userName,room,position},cb)=>{

        console.log(userName, position, room);

        loadData(room)
        if(gameArea[room]){
            let game = playTurn(userName,position,room)
            // console.log(game);
            io.of('/').in(room).clients((err,clients)=>{
                game.playersInGame = clients
                //
            })
            // console.log(io.in(room).clients());
            // console.log(game);
            io.to(room).emit('game',game)
        }
            // console.log(gameArea);
            // console.log(gameArea[room]);
    })
```

on ‘playPosition’ the socket accepts 3 crucial arguments which it uses to play the turns of the player and accordingly emit the the board to that specific room

```
socket.on('resetBoard',(room)⇒{
        // console.log("Reset");
        resetBoard(room)
        loadData(room)
        io.to(room).emit('game',gameArea[room])

    })
```

In case a player/group need to reset the board for some reason, an event is listened and functions are called ad the resulting gameroom is emitted to that room

```
socket.on('disconnect',(reason)⇒{
        console.log(reason);
    })
```

Then a player is disconnected, the reason is logged to understand the reason and correct the error.

Game Logic

```
let gameArea = {}


let gameStatus  = [[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,]]
let playablePosition = [6,6,6,6,6,6,6]
let winnerPlayer = null
```

The data model the whole game works upon. gameStatus describes the board, playable position describes the valid number of positions, winnerPlayer is given a value when someone wins. gameArea holds all those data which a room as a key.

```
const loadData = (userName,room) =>{
    if(!gameArea[room]){
        let newRoomStarter = {
            gameStatus  : [[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,]],
            playablePosition : [6,6,6,6,6,6,6],
            winnerPlayer : null
        }
        gameArea[room] = newRoomStarter
    }
        return gameArea[room]
    }

```

loadData function loads/creates the board. It checks if the room exist in the game area, if it doesn't then it creates a blank board and returns it.

```
const playTurn = (player,position,room)⇒{
    if(gameArea[room].winnerPlayer){
        resetBoard(room)
        loadData(room)
    }
    // console.log(gameArea[room]);
    console.log(position);
    if(gameArea[room].playablePosition[position]==0){
        // console.log('Invalid move')
        return false
    }
    if(position > 6 | position <0 | position === null | position === undefined){
        // console.log(`position is ${position}`);
        // console.log('Wrong position')
        return false
    }

    updatePlayablePosition(position,room)

    let h = gameArea[room].playablePosition[position]
    let v = position
    // console.log(h,v)
    // console.log(gameArea);
    gameArea[room].gameStatus[h][v] = player
    // console.log(gameArea[room].gameStatus);
    checkWinVY(player,h,v,room)
    checkWinHX(player,h,v,room)
    return gameArea[room]

}
```

The main function that is evoked every time a player plays a turn. It validates and plays a position in a specific room and return the resultant game area.

```
const updatePlayablePosition=(position,room)⇒{
    // console.log(playablePosition[position]);
    // console.log(gameArea[room]);
    gameArea[room].playablePosition[position] = gameArea[room].playablePosition[position] - 1

}
```

This updates the data playablePosition

```
const checkWinVY = (turn,h,v,room)⇒{
    // console.log(h,v);
    let count = 0
    if(h<3){
        for(let i = 0; i<4;i++){
            // console.log(gameStatus[h+i][v]);
            if(gameArea[room].gameStatus[h+i][v]==turn){
                count = count+1
            }
        }
    }
    if(count ⩾ 4){
        winner(turn,`Connecting 4 Vertically`,room)
    }
}

const checkWinHX = (turn,h,v,room)⇒{
    let count = 0
    // console.log(h,v);

    gameArea[room].gameStatus[h].forEach(element ⇒ {
        // console.log(element);
        if(element == turn){
            // console.log(element);
            count = count+1
            // console.log(count);
            if(count ⩾ 4){
                winner(turn,`Connecting 4 Horizontally`,room)
            }
        }
        else{
            count = 0
        }
    });
    // console.log(count);
}
```

This keeps on checking the Vertical Match and horizontal match of 4 from the current position respectively.

```
const winner = (player,method,room)⇒{
    if(winnerPlayer){}
    gameArea[room].winnerPlayer = (`${player} won by ${method}`);
    // resetBoard(room)
}
```

Invoked after a match, it sends the winner details to the frontend

```
const resetBoard = (room)⇒{
    gameArea[room].gameStatus = gameArea[room].gameStatus = [[0,0,0,0,0,0,0,],[0,0,0,0,
0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,]]
    gameArea[room].playablePosition = [6,6,6,6,6,6,6]
    gameArea[room].winnerPlayer=null
}
```

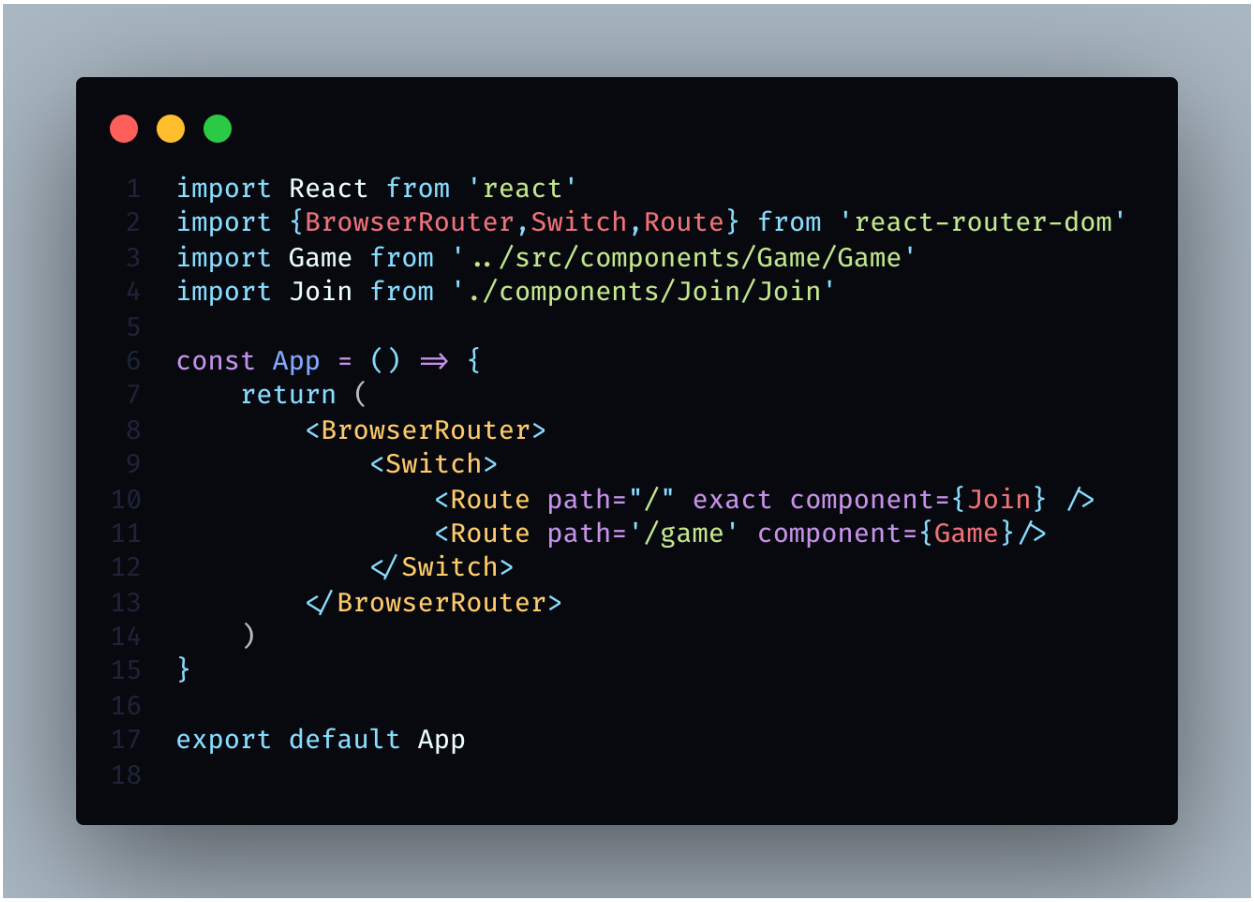
Resets the board when called

FrontEnd

Join - Handles the Joining of the players

Game - Is the Main Game Component that handles sockets from client side

GameBoard - Displays the data into a visually appealing way and allows clickables to function as needed.

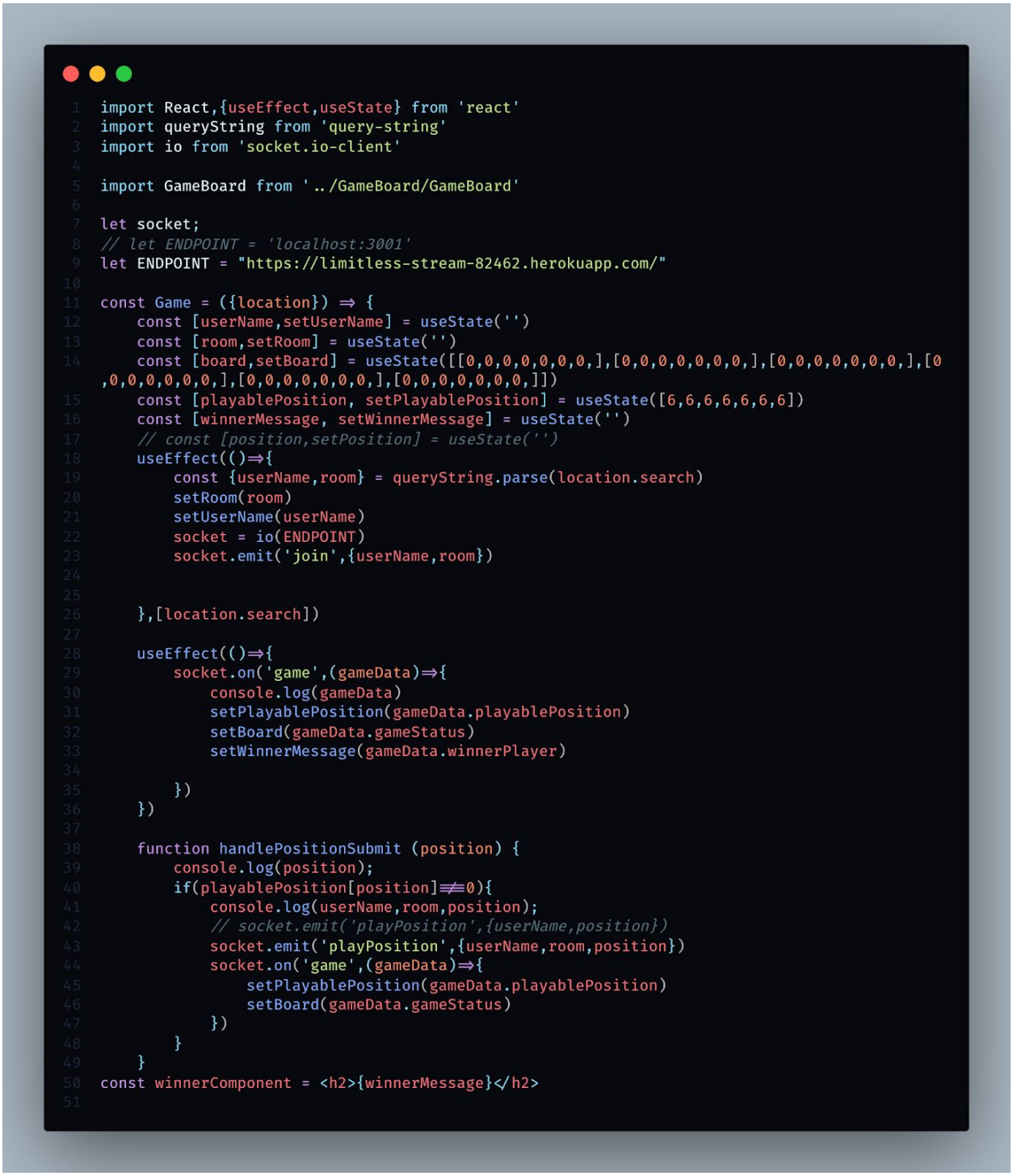


```
import React from 'react'
import {BrowserRouter,Switch,Route} from 'react-router-dom'
import Game from '../src/components/Game/Game'
import Join from './components/Join/Join'

const App = () => {
    return (
        <BrowserRouter>
            <Switch>
                <Route path="/" exact component={Join} />
                <Route path='/game' component={Game}/>
            </Switch>
        </BrowserRouter>
    )
}

export default App

```

Basic Routing functionality

```
import React,{useState} from 'react'
import {Link} from 'react-router-dom'
import './Join.css'

const Join = () => {
    const [userName,setUserName] = useState('')
    const [room,setRoom] = useState('')

    return (

            <div className="join">
                <h1 className="text-center text-white display-3">Join</h1>

                <form >
                    <h3 className="text-white">Username</h3
>
                    <input className="form-control text-lg" required type="text" name="
username" id="username" placeholder="username" onChange={(e)=>setUserName(e.target.value)}/>

                    <h3 className="text-white">Room</h3>
                    <input className="form-control text-lg"  required  type="text"  name="
gameroom" id="gameroom" placeholder="room" onChange={(e)=>setRoom(e.target.value)}/>
                    <br/>
                    <br/>
                    <div className="text-center">
                        <Link onClick={e=>(!room | !userName ?e.preventDefault():null )} to=
{`/game?userName=${userName}&room=${room}`}>
                            <button className="btn btn-lg btn-danger rounded" type="submit"
>Submit</button>
                        </Link>
                    </div>
                </form>
            </div>
    )
}

export default Join
```

Join Form which sends the user with queryparameter in the url based on the username and room



```
import React,{useEffect,useState} from 'react'
import queryString from 'query-string'
import io from 'socket.io-client'

import GameBoard from '../GameBoard/GameBoard'

let socket;
// let ENDPOINT = 'localhost:3001'
let ENDPOINT = "https://limitless-stream-82462.herokuapp.com/"

const Game = ({location}) => {
    const [userName,setUserName] = useState('')
    const [room,setRoom] = useState('')
    const [board,setBoard] = useState([[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,],[0,0,0,0,0,0,0,]])
    const [playablePosition, setPlayablePosition] = useState([6,6,6,6,6,6,6])
    const [winnerMessage, setWinnerMessage] = useState('')
    // const [position,setPosition] = useState('')
    useEffect(()=>{
        const {userName,room} = queryString.parse(location.search)
        setRoom(room)
        setUserName(userName)
        socket = io(ENDPOINT)
        socket.emit('join',{userName,room})


    },[location.search])

    useEffect(()=>{
        socket.on('game',(gameData)=>{
            console.log(gameData)
            setPlayablePosition(gameData.playablePosition)
            setBoard(gameData.gameStatus)
            setWinnerMessage(gameData.winnerPlayer)

        })
    })

    function handlePositionSubmit (position) {
        console.log(position);
        if(playablePosition[position]!==0){
            console.log(userName,room,position);
            // socket.emit('playPosition',{userName,room,position})
            socket.emit('playPosition',{userName,room,position})
            socket.on('game',(gameData)=>{
                setPlayablePosition(gameData.playablePosition)
                setBoard(gameData.gameStatus)
            })
        }
    }
const winnerComponent = <h2>{winnerMessage}</h2>

```

Used Hooks for maintaining state and useEffects for updating. This handles the socket connections and and the frontend game logic.

```
return (
        <div className="container game">

            <h1 className="text-center text-white"><strong>Connect 4</strong></h1
>
                <h3 className="text-center text-white">Hi <strong> {userName}</strong
>, you in room <strong><u>{room}</u></strong> with color <span className="text-danger"><
strong>RED</strong></span> </h3>

            <hr/>
            <div  style={{display:'flex',
            justifyContent:"center"}}>
                <GameBoard  handlePositionSubmit={handlePositionSubmit} playablePosition={
playablePosition} userName={userName}  board={board}/>
            </div>
            <hr/>
            <div className="text-center">
                <h3 className="text-center text-white">{winnerComponent}</h3>
                <button type="button" className="btn btn-lg btn-danger center" onClick={(e)=>
socket.emit('resetBoard',room)}>Reset</button>
                </div>
        </div>
    )
}
export default Game

```

And returns the basic details and the GameBoard Component and passes the props

```
import './GameBoard.css'
import React from 'react'


const GameBoard = ({board,userName, playablePosition,handlePositionSubmit}) => {
    // console.log(board);
    // console.log(userName);

    const handleDrop =(e)=>{
        // console.log(e);
        // console.log(e.currentTarget);
        handlePositionSubmit(e.currentTarget.name)

    }

    let TheBoard =()=> board.map((row) => <tr >{row.map((element=><td width="80rem"><div
classNameelement===userName?"cell col red":element===0?"cell col":"cell col yellow"}>  </
div></td>))}</tr>)

    let PlayablePosition = () =>{
        // console.log(playablePosition);
        return playablePosition.map((element,index)=>{
            console.log(element,index);
            return <td  align="center" cli><button  onClick={(e)=>handleDrop(e)} name={index
} className= {element!==0?"btn btnO rounded-pill btn-warning":"
btn btnO disabled rounded-pill btn-warning"}><i name={index} className="
fa fa-lg fa-arrow-circle-down"/></button></td>
        })
    }
    return (
        <table  className="center table-hover"  cellSpacing="0" >
            <tr>
                <PlayablePosition/>
            </tr>
            <TheBoard/>
        </table>
    )
}

export default GameBoard

```

This handles the display of the complete board and the submission of the position and manages the clickable/non clickable button.

*Css and other non essential code isn't included to preserve the size under 5mb