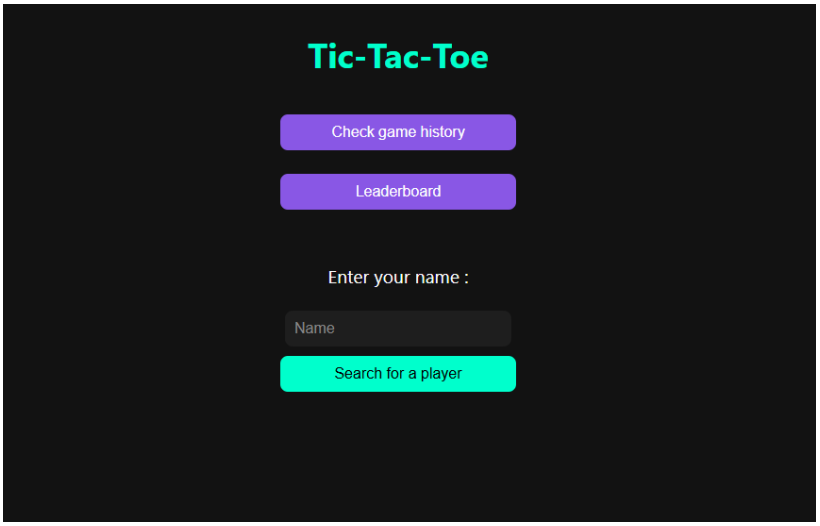
PPKS projekt

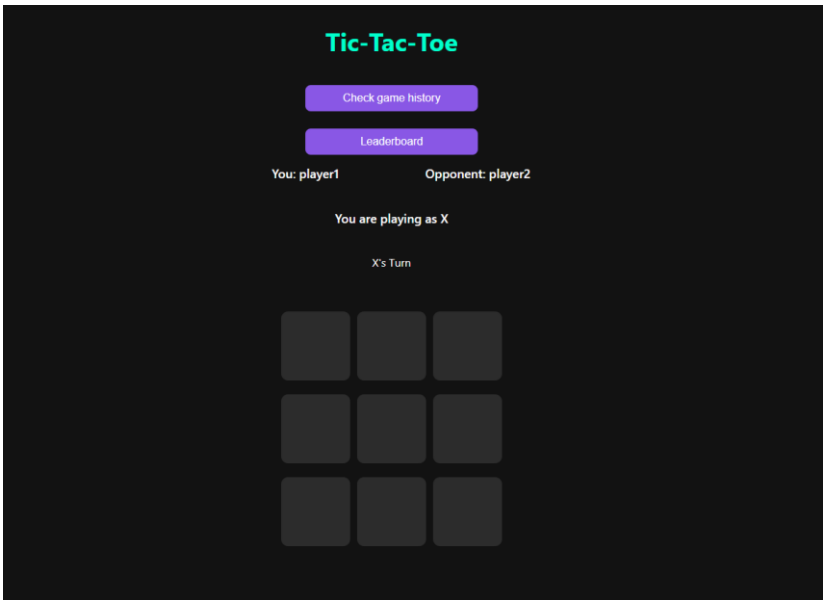
Student: Boris Boronjek

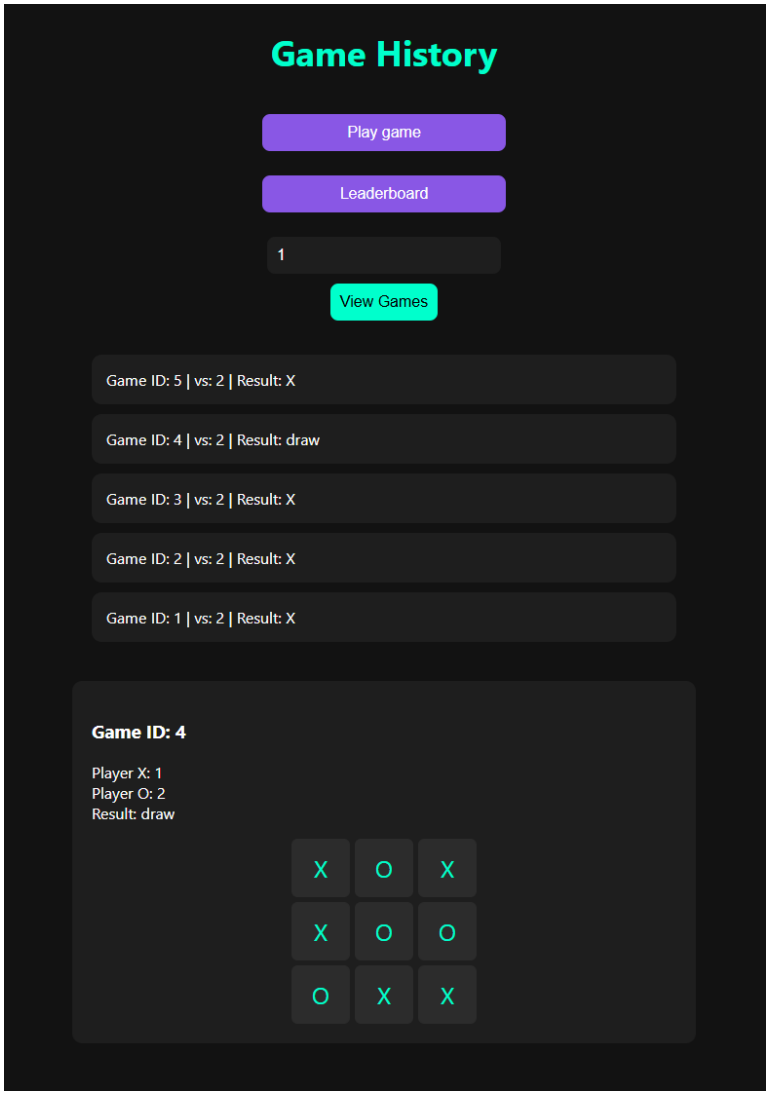
JMBAG: 0036531473

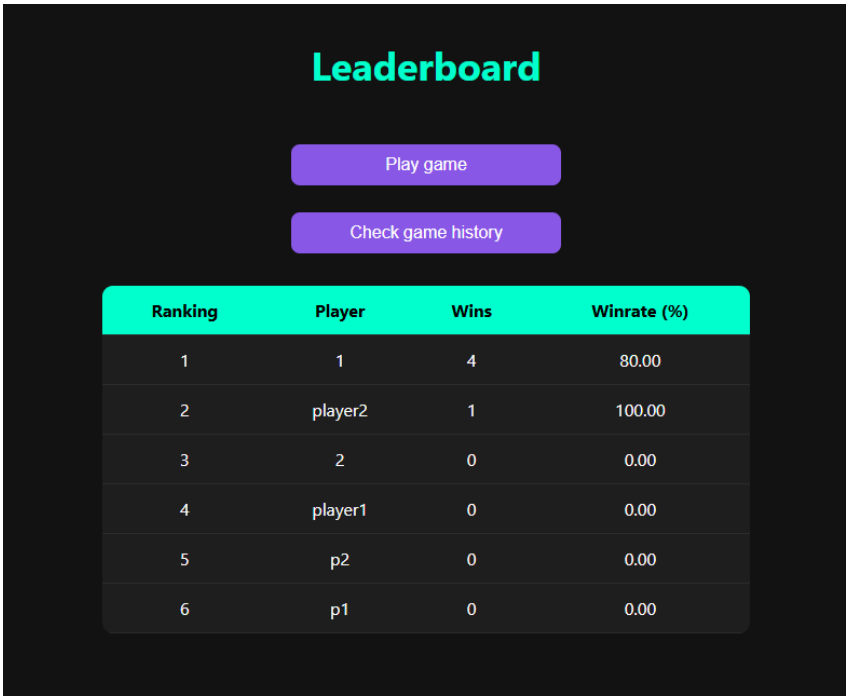
1. Uvod

Ova aplikacija je razvijena kao dio praktičnog zadatka iz kolegija programska potpora komunikacijskim sustavima. Riječ je o jednostavnoj aplikaciji za igru križić-kružić, proširenoj mogućnošću vođenja statistike, pregleda povijesti igara i leaderboarda. Fokus implementacije je bio na korištenju tehnologija koje se obrađuju na kolegiju: REST API, JavaScript, AJAX, CRUD operacije i rad s bazom podataka.






Game History
Play game
Leaderboard
1
View Games
Game ID: 5 | vs: 2 | Result: X
Game ID: 4 | vs: 2 | Result: draw
Game ID: 3 | vs: 2 | Result: X
Game ID: 2 | vs: 2 | Result: X
Game ID: 1 | vs: 2 | Result: X
Game ID: 4
Player X: 1
Player O: 2
Result: draw
X O X
X O O
O X X


Leaderboard
Play game
Check game history
Ranking Player Wins Winrate (%)
1 1 4 80.00
2 player2 1 100.00
3 2 0 0.00
4 player1 0 0.00
5 p2 0 0.00
6 p1 0 0.00

2. Tehnologije i alati

U razvoju aplikacije korišteni su sljedeći alati i tehnologije:

Frontend: HTML, CSS, JavaScript

Backend: Node.js (Express framework)

Baza podataka: postgreSQL

Komunikacija: REST API

Dohvat podataka: AJAX (fetch API)

Komunikacija između igrača: WebSocket

Verzijski sustav: GitHub

3. Opis funkcionalnosti

Igranje igre: na glavnoj stranici (index.html) korisnik može započeti novu igru i igrati kao igrač X ili O ovisno o tome koji igrač je prvi pokrenuo pretragu za igrom. Rezultat se nakon odigrane igre automatski pohranjuje u bazu podataka.

Povijest igara: korisnik može otvoriti game_history.html, pretražiti sve igre određenog igrača po imenu i pregledati stanje ploče za bilo koju njegovu igru.

Leaderboard: Na leaderboard.html prikazuje se ljestvica igrača prema broju pobjeda i postotku pobjeda.

4. Implementacija

REST API: backend koristi REST API endpointe za dohvat i spremanje podataka.

```
app.post("/saveGame", async (req, res) => {
  const { player_x, player_o, result, board_state } = req.body;

  try {
    const newGame = await pool.query(
      "INSERT INTO games (player_x, player_o, result, board_state) VALUES ($1, $2, $3, $4) RETURNING *",
      [player_x, player_o, result, board_state]
    );
    res.json(newGame.rows[0]);
  } catch (err) {
    console.error(err.message);
    res.status(500).send("Error during saving the game to the database");
  }
})

app.get("/games-by-player", async (req, res) => {
  const { name } = req.query;

  try {
    const games = await pool.query(
      "SELECT * FROM games WHERE player_x = $1 OR player_o = $1 ORDER BY id DESC",
      [name]
    );
    res.json(games.rows);
  } catch (err) {
    console.error(err.message);
    res.status(500).send("Error during retrieving the game from the database");
  }
});

app.get("/get-leaderboard", async (req, res) => {
  try {
    const leaderboardQuery = `
      SELECT player,
             COUNT(*) AS total_games,
             COUNT(*) FILTER (WHERE win = TRUE) AS wins,
             ROUND(COUNT(*) FILTER (WHERE win = TRUE) * 100.0 / COUNT(*), 2) AS winrate
      FROM (
        SELECT player_x AS player,
               (result = 'X')::boolean AS win
        FROM games
        UNION ALL
        SELECT player_o AS player,
               (result = 'O')::boolean AS win
        FROM games
      ) sub
      GROUP BY player
      ORDER BY wins DESC, winrate DESC;
    `;

    const { rows } = await pool.query(leaderboardQuery);
    res.json(rows);
  } catch (err) {
    console.error(err.message);
    res.status(500).send("Error during retrieving the leaderboard from the database");
  }
});
```

AJAX pozivi: ajax koristi fetch() za slanje i dohvat podataka bez reloadanja stranice

index.html

```
328     function saveGame(result) {
329         let boardState = {};
330         document.querySelectorAll(".btn").forEach(btn => {
331             boardState[btn.id] = btn.innerText;
332         });
333 
334         fetch("/saveGame", {
335             method: "POST",
336             headers: { "Content-Type": "application/json" },
337             body: JSON.stringify({
338                 player_x: document.getElementById("value").innerText === "X" ? name : opponentName,
339                 player_o: document.getElementById("value").innerText === "O" ? name : opponentName,
340                 result: result,
341                 board_state: JSON.stringify(boardState)
342             })
343         })
344             .then(res => res.json())
345             .then(data => console.log("Game saved:", data))
346             .catch(err => console.error("Save failed", err));
347     }
```

game_history.html

```
133     async function fetchGames() {
134       const name = document.getElementById("playerName").value;
135       if (!name) return alert("Enter a player name");
136 
137       document.getElementById("gameDetails").innerHTML = "";
138 
139       const res = await fetch(`/games-by-player?name=${encodeURIComponent(name)}`);
140       const games = await res.json();
141 
142       const container = document.getElementById("gamesList");
143       container.innerHTML = "";
144 
145       if (games.length === 0) {
146         container.innerHTML = "<p>No games found.</p>";
147         return;
148       }
```

leaderboard.html

```
 98     async function loadLeaderboard() {
 99       const res = await fetch("/get-leaderboard");
100       const data = await res.json();
101 
102       const tbody = document.getElementById("leaderboardBody");
103       tbody.innerHTML = "";
104 
105       data.forEach((player, index) => {
106         const tr = document.createElement("tr");
107         tr.innerHTML = `
108           <td>${index + 1}</td>
109           <td>${player.player}</td>
110           <td>${player.wins}</td>
111           <td>${player.winrate}</td>
112         `;
113         tbody.appendChild(tr);
114       });
115     }
```

Baza podataka: Aplikacija koristi postgreSQL s jednom tablicom koja omogućuje CRUD operacije nad podacima.

```sql
CREATE TABLE games (
    id SERIAL PRIMARY KEY,
    player_x VARCHAR(50) NOT NULL,
    player_o VARCHAR(50) NOT NULL,
    result VARCHAR(10) NOT NULL,
    board_state TEXT NOT NULL
);
```

	id [PK] integer	player_x character varying (50)	player_o character varying (50)	result character varying (10)	board_state text
1	1	1	2	X	{"btn1":"X","btn2":"","btn3":"O","btn4":"O","btn5":"O","btn6":"","btn7":"X","btn8":"X","btn9":"X"}
2	2	1	2	X	{"btn1":"X","btn2":"","btn3":"O","btn4":"O","btn5":"O","btn6":"","btn7":"X","btn8":"X","btn9":"X"}
3	3	1	2	X	{"btn1":"X","btn2":"","btn3":"O","btn4":"X","btn5":"O","btn6":"","btn7":"X","btn8":"O","btn9":"X"}
4	4	1	2	draw	{"btn1":"X","btn2":"O","btn3":"X","btn4":"X","btn5":"O","btn6":"O","btn7":"O","btn8":"X","btn9":"...
5	5	1	2	X	{"btn1":"X","btn2":"O","btn3":"X","btn4":"O","btn5":"X","btn6":"","btn7":"X","btn8":"","btn9":"O"}
6	6	p1	p2	draw	{"btn1":"X","btn2":"O","btn3":"O","btn4":"O","btn5":"X","btn6":"X","btn7":"X","btn8":"X","btn9":"...
7	7	player1	player2	O	{"btn1":"X","btn2":"","btn3":"O","btn4":"","btn5":"X","btn6":"O","btn7":"X","btn8":"","btn9":"O"}

WebSocket: Aplikacija koristi WebSocket za ostvarivanje dvosmjerne komunikacije u stvarnom vremenu između igrača tijekom igranja igre križić-kružić

backend:

```
io.on("connection",(socket)=>{

    socket.on("find",(e)=>{

        if(e.name!=null){

            arr.push(e.name)

            if(arr.length>=2){
                let p1obj={
                    p1name:arr[0],
                    p1value:"X",
                    p1move:""
                }
                let p2obj={
                    p2name:arr[1],
                    p2value:"O",
                    p2move:""
                }

                let obj={
                    p1:p1obj,
                    p2:p2obj,
                    sum:1
                }
                playingArray.push(obj)

                arr.splice(0,2)

                io.emit("find",{allPlayers:playingArray})

            }

        }

    })

    socket.on("playing",(e)=>{
        if(e.value=="X"){
            let objToChange=playingArray.find(obj=>obj.p1.p1name===e.name)

            objToChange.p1.p1move=e.id
            objToChange.sum++
        }
        else if(e.value=="O"){
            let objToChange=playingArray.find(obj=>obj.p2.p2name===e.name)

            objToChange.p2.p2move=e.id
            objToChange.sum++
        }

        io.emit("playing",{allPlayers:playingArray})

    })

    socket.on("gameOver",(e)=>{
        playingArray=playingArray.filter(obj=>obj.p1.p1name!==e.name)
    })


})
```

frontend (index.html):

```
    socket.on("find", (e) => {
        let allPlayersArray = e.allPlayers

        if (name != '') {
            document.getElementById("userCont").style.display = "block"
            document.getElementById("oppNameCont").style.display = "block"
            document.getElementById("valueCont").style.display = "block"
            document.getElementById("loading").style.display = "none"
            document.getElementById("name").style.display = "none"
            document.getElementById("find").style.display = "none"
            document.getElementById("enterName").style.display = "none"
            document.getElementById("bigcont").style.display = "block"
            document.getElementById("whosTurn").style.display = "block"
            document.getElementById("whosTurn").innerText = "X's Turn"
        }

        let oppName
        let value

        const foundObject = allPlayersArray.find(obj => obj.p1.p1name == `${name}` || obj.p2.p2name == `${name}`);
        foundObject.p1.p1name == `${name}` ? oppName = foundObject.p2.p2name : oppName = foundObject.p1.p1name
        foundObject.p1.p1name == `${name}` ? value = foundObject.p1.p1value : value = foundObject.p2.p2value

        opponentName = oppName;
        document.getElementById("oppName").innerText = oppName
        document.getElementById("value").innerText = value

        if (value === "X") {
            myTurn = true;
        } else {
            myTurn = false;
        }
    })
```

```
    socket.on("playing", (e) => {
        const foundObject = (e.allPlayers).find(obj => obj.p1.p1name == `${name}` || obj.p2.p2name == `${name}`);

        let p1id = foundObject.p1.p1move
        let p2id = foundObject.p2.p2move

        if ((foundObject.sum) % 2 === 0) {
            document.getElementById("whosTurn").innerText = "O's Turn";
            myTurn = (document.getElementById("value").innerText === "O");
        } else {
            document.getElementById("whosTurn").innerText = "X's Turn";
            myTurn = (document.getElementById("value").innerText === "X");
        }

        if (p1id != '') {
            let btn = document.getElementById(`${p1id}`);
            btn.innerText = "X";
            btn.disabled = true;
            btn.style.color = "#00ffcc";
        }
        if (p2id != '') {
            let btn = document.getElementById(`${p2id}`);
            btn.innerText = "O";
            btn.disabled = true;
            btn.style.color = "#00ffcc";
        }

        check(name, foundObject.sum)
    })
```

5. Zaključak

Aplikacija demonstrira praktičnu upotrebu REST API-ja, AJAX komunikacije, manipulacije DOM-a, spremanja podataka u bazu i prikaza sučelja korisniku. Projekt je u potpunosti funkcionalan i može se dodatno proširivati. Uključuje osnovnu implementaciju WebSocketa za komunikaciju između igrača u stvarnom vremenu bez ponovnog učitavanja stranice.

6. GitHub

https://github.com/borisboronjek2/PPKS