Online Multiplayer Tic-Tac-Toe Game

Submitted By:

Name of Student	*University Roll No.*
Soumyadeep Basak	*12022002016018*
Shubham Sahu	*12022002016047*

Prof. Deepsubhra Guha Roy

Academic Year: 2024-25

REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE MACHINE LEARNING) OF MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE MACHINE LEARNING)

INSTITUTE OF ENGINEERING AND MANAGEMENT
KOLKATA

CERTIFICATE OF RECOMMENDATION

We hereby recommend that the thesis prepared under our supervision by **Soumyadeep Basak** and **Shubham Sahu** entitled **Online Multiplayer Tic-Tac-Toe Game** be accepted in partial fulfillment of the requirements for the degree of BACHELOR OF TECHNOLOGY IN "COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE MACHINE LEARNING)".

_____

_____

Head, CSE(AIML)Department ***Project Guide***

Institute of Engineering and Management, Kolkata

Introduction

The **"Online Multiplayer Tic-Tac-Toe Game"** is a web-based application designed to offer an engaging and interactive experience for users to play Tic-Tac-Toe in real-time with friends or other online players. This project combines classic Tic-Tac-Toe gameplay with the convenience and connectivity of the web, allowing players to log in, create or join game rooms, and compete in live matches from any location using a desktop or mobile device. In addition to game rooms, the platform provides features like in-game chat, an achievement system, a leaderboard, and user profiles, creating a competitive and social gaming experience for players of all ages.

The primary aim of the **Online Multiplayer Tic-Tac-Toe Game** is to bring a simple yet iconic game into a multiplayer, web-based format that is both intuitive and efficient. Built using **Flask** for the backend, **Socket.io** for real-time communication, and **MySQL** for data management, the application ensures a seamless user experience and reliable game mechanics. The real-time functionality facilitates quick moves and updates, while the database securely manages user data, game records, and leaderboards, enhancing both user engagement and operational efficiency.

With a user-friendly interface and a focus on multiplayer interaction, this online Tic-Tac-Toe game transforms traditional gameplay into a competitive, social experience. Whether a user wishes to challenge friends or find new opponents, the application provides a fun and accessible platform for everyone. The design prioritizes scalability and security, ensuring future expansion to accommodate more features, such as ranked gameplay and AI opponents. This project not only offers entertainment but also serves as an educational platform to showcase the implementation of real-time web applications and user-driven content management in a gaming context.

Abstract

The **Online Multiplayer Tic-Tac-Toe Game** is a web-based application built using **Flask**, **MySQL**, and **Socket.io** that enables users to play the classic Tic-Tac-Toe game in real time with other players. Designed to offer an engaging and competitive gaming environment, this platform provides key features including user authentication, room creation for multiplayer matches, an in-game chat, and live leaderboard tracking. Players can initiate new games, invite friends, or join public rooms, while the database keeps track of user information, game summaries, and win-loss records.

This project aims to build a fully functional online multiplayer experience by integrating **Socket.io** for real-time gameplay interactions, allowing instant updates to each player's game board and enabling smooth, synchronized gameplay. The backend, developed in Flask, handles user authentication, room management, and game logic, while **MySQL** is used to efficiently manage persistent data such as user profiles, game histories, and rankings. The application uses modular code, with separate routes for game management, user operations, and leaderboard updates, ensuring a clean, maintainable structure.

The system’s real-time capabilities are further enhanced by implementing socket-based communication that updates player moves instantly and enables in-game chat, fostering a dynamic and interactive experience. By designing the game with a focus on multiplayer functionality and data accuracy, the project overcomes the limitations of single-player Tic-Tac-Toe, transforming it into a social, competitive platform. The use of a robust backend and structured database ensures a reliable experience, enhancing user satisfaction and engagement in the gameplay.

Project Aim

The aim of this project is to develop an interactive, online Tic-Tac-Toe game that offers a robust multiplayer experience with real-time gameplay, leaderboard rankings, user authentication, and comprehensive administration capabilities. The project combines Flask, Socket.IO, and MySQL to deliver a scalable, web-based solution for engaging and competitive gameplay with live updates.

Main Goal

The main objective is to develop a comprehensive online Tic-Tac-Toe game platform that allows users to register for an account, securely log in, and participate in engaging, real-time multiplayer games. The platform will offer seamless matchmaking to connect players, enabling them to join games instantly or invite specific friends for a match.

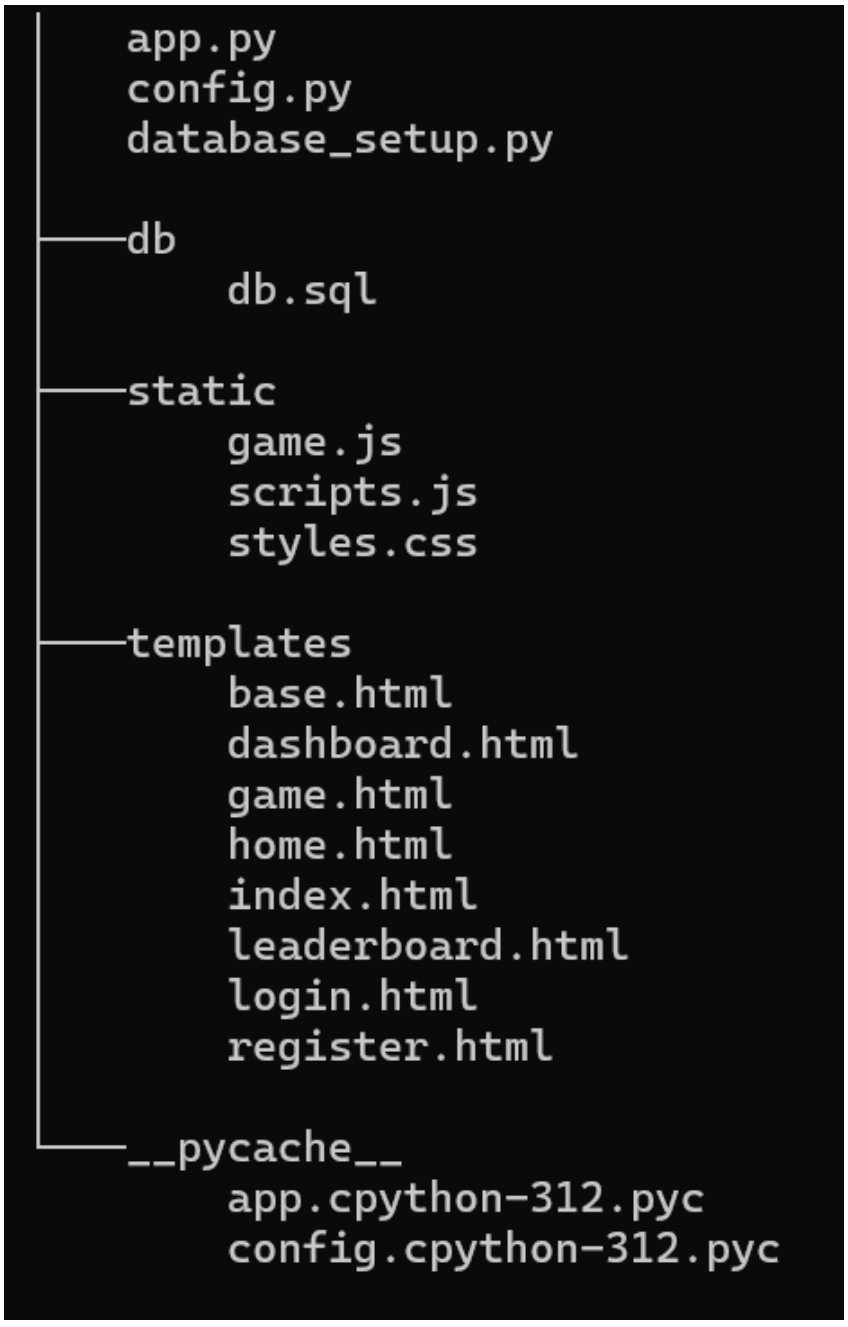
An integral feature is the live leaderboard, which not only displays players' rankings but also tracks detailed game statistics, such as wins, losses, draws, and overall performance scores. This leaderboard provides players with a competitive edge and motivation to improve their skills.

Additionally, the platform will include an admin dashboard designed for efficient user and game management. Through this dashboard, administrators can monitor player activity, view individual and overall game outcomes, and manage user accounts to ensure a secure and fair gaming environment. The dashboard will also offer tools for reviewing and moderating game results to prevent abuse or exploitation.

With user authentication, game history, and player achievements being key aspects, this platform aims to deliver a dynamic and secure gaming

experience while fostering a sense of community among Tic-Tac-Toe enthusiasts.

File Tree Structure Of The Project



Methods

Administration Module

1. **User Management**: The admin dashboard allows the administrator to manage users by viewing registered players, excluding the admin account, and reviewing user details.
2. **Game Results Tracking**: Admins can access a detailed games list displaying match results, including player emails instead of ID numbers for readability.
3. **Leaderboard Management**: The admin dashboard includes a dedicated leaderboard view, showcasing player stats (wins, losses, and scores) sorted by score to highlight top performers.
4. **Game Monitoring and Logging**: Admins can view real-time log entries and monitor errors or exceptions using logging tools to enhance platform reliability.

User Module

1. **User Registration and Login**: Users can securely register with hashed passwords, log in, and access game features with session-based authentication.
2. **Real-time Game Interaction**: Users join and play Tic-Tac-Toe matches in real-time using Socket.IO for dynamic room updates and move validations.
3. **Statistics and Leaderboard Access**: Each user's game statistics (wins, losses, and score) are updated after each match. A public leaderboard provides competitive motivation by displaying user rankings.
4. **Score Calculation and Update**: The leaderboard score is dynamically updated after every match based on wins, losses, and total games played, ensuring an accurate representation of player performance.

Components for the Online Multiplayer Tic-Tac-Toe Game

1. Frontend (Client-side)

- **Game Board UI:** The grid interface with clickable cells that users interact with to make moves.
- **User Authentication:** Login and registration forms for user authentication.
- **Room Management:** UI for creating or joining game rooms with options for rematch, player invite, etc.
- **Leaderboard Display:** Displays top players based on wins, losses, and scores.
- **Notifications & Alerts:** Inform players of events such as successful moves, invalid moves, game completion, or a win/loss/draw outcome.
- **Real-time Updates:** Frontend receives real-time updates of the game state via WebSocket, updating the game board and scoreboard accordingly.

2. Backend (Server-side)

- **Flask Application Server:** Main application server handling HTTP requests and sessions.
- **SocketIO Server:** Real-time communication server using WebSockets for handling game moves, room updates, and chat messages.
- **RESTful API:** For operations like user login, registration, leaderboard fetching, etc.
- **User Management:** Handles user authentication, registration, password hashing, and session management.
- **Game Logic:** Server-side logic for:
    - Checking valid moves.

    - Determining win/loss/draw conditions.
    - Updating player turns.
- **Room Management:** Logic to create and manage game rooms, join/leave rooms, and reset rooms for rematches.
- **Leaderboard Management:** Tracks game results, stores user stats (wins, losses, scores), and calculates rankings.
- **Admin Dashboard:** Separate dashboard with tools for moderating games and managing users.

3. Database

- **User Table:** Stores user credentials, email, and other account details.
- **Game Table:** Records each game instance, winner, loser, and timestamp for historical data.
- **Leaderboard Table:** Tracks wins, losses, and calculated scores for each player, updating with each game result.

4. Security & Middleware

- **Authentication Middleware:** Ensures users are authenticated before accessing game rooms or game moves.
- **Access Control:** Protects admin-only routes and separates admin functionality from standard user functionality.
- **CORS Configuration:** Ensures secure cross-origin requests for clients hosted on different domains.
- **Input Validation:** Validates all incoming data for moves, room actions, and user input to prevent injection attacks.

5. Logging & Monitoring

- **Logging System:** Captures login attempts, room creation, move validation, and win/loss events.

- **Error Handling:** Custom error messages for various situations (e.g., invalid moves, failed login).
- **Performance Monitoring:** For server performance and uptime, especially useful for tracking real-time game states.

These components together provide a cohesive environment for an engaging, secure, and fully functional online multiplayer Tic-Tac-Toe game.

Database Design

1. Users Table

This table stores information about the players using the platform, which includes unique identification, login credentials, and contact information.

Attribute	Type	Description
`id`	`int`	Unique identifier for each user (primary key). This ID links to other tables to associate game statistics and results with the user.
`email`	`varchar(255)`	User's email address, used as a unique identifier for login and communication.
`password`	`varchar(255)`	Encrypted password for user authentication. It's stored securely to protect user accounts.

- **Role**: This table enables account management and user authentication, linking users to their game outcomes and leaderboard records.

2. GameResults Table

This table records the outcome of each game, logging the players involved and who won or lost. This is essential for tracking player statistics and leaderboard rankings.

Attribute	Type	Description
`id`	`int`	Unique identifier for each game result (primary key).
`game_id`	`int`	Unique ID for each game session, used to track game history.
`winner_id`	`int`	User ID of the game winner (foreign key to `Users` table), allowing the platform to credit the winner's record.
`loser_id`	`int`	User ID of the game loser (foreign key to `Users` table), tracking losses for leaderboard calculations.

- **Role**: This table keeps a historical record of game outcomes, allowing the leaderboard to display accurate win/loss records and player performance over time.

3. Leaderboard Table

This table summarizes each player's game performance. It keeps cumulative statistics for quick leaderboard retrieval and updates, making it easier to display player rankings without recalculating each game result.

Attribute	Type	Description
`id`	`int`	Unique identifier for each leaderboard entry (primary key).
`user_id`	`int`	User ID associated with this leaderboard record (foreign key to `Users` table).
`games_won`	`int`	Cumulative count of games won by the user, updated after each game win.
`games_lost`	`int`	Cumulative count of games lost by the user, updated after each game loss.

`total_score`	`decimal(10,2)`	Calculated score representing the user's overall performance; can be a formula based on wins, losses, and other metrics.
`games_drawn`	`int`	Count of games that ended in a draw for this user.

- **Role**: This table is crucial for leaderboard functionality, as it provides a snapshot of each player's gaming record, enabling efficient display of ranks based on wins, losses, and performance scores.

Summary of How These Tables Work Together:

1. **Users Table**: Manages player information and is the core for authentication and account management.
2. **GameResults Table**: Logs each game's outcome, linking to users by storing their IDs as winners or losers. It supports tracking of individual game history and ensures the leaderboard reflects up-to-date performance.
3. **Leaderboard Table**: Holds cumulative statistics for each player, providing quick and efficient access to ranking data based on game outcomes from `GameResults`.

Together, these tables ensure that users can log in, join games, and track their progress, while admins can monitor platform activity and leaderboard accuracy.

Source Code:

1.Frontend:

game.js

```
document.addEventListener("DOMContentLoaded", function() {
    const room = prompt("Enter room name:");
    const socket = io.connect(window.location.origin);
    const playerSymbol = {};

    // Join the room
    socket.emit('join_game', { room: room });

    socket.on('update_room', (data) => {
        document.getElementById("players").innerHTML = "Players in
room: " + data.players.join(", ");
    });

    socket.on('game_start', (data) => {
        const [player1, player2] = data.players;
        playerSymbol[player1] = 'X';
        playerSymbol[player2] = 'O';
        document.getElementById("status").innerHTML = "Game started!
Your symbol: " + playerSymbol[sessionEmail];
    });

    socket.on('move_made', (data) => {
        const cell = document.getElementById(`cell-${data.index}`);
        cell.innerText = data.symbol;
    });

   // Assuming you have SocketIO connected
socket.on('game_won', function(data) {
    const winner = data.winner;

    // Display the winner message
    alert(`Game Over! The winner is ${winner}`);
```

```
    // Optional: Reload the leaderboard after 2 seconds
    setTimeout(function() {
        window.location.href = "/leaderboard"; // Or make an AJAX
call to refresh the leaderboard
    }, 2000);
});


    // Emit move
    function makeMove(cellIndex) {
        socket.emit('make_move', { room: room, move: cellIndex });
    }

    // Attach click listeners to cells
    for (let i = 0; i < 9; i++) {

document.getElementById(`cell-${i}`).addEventListener("click",
function() {
            makeMove(i);
        });
    }
});
```

scripts.js

```
// Wait for the DOM content to be fully loaded
document.addEventListener("DOMContentLoaded", function() {
    // Check if Socket.IO library is available and the socket is not
already initialized
    if (typeof io !== 'undefined' && typeof socket === 'undefined') {
        // Initialize the socket connection
        var socket = io.connect(window.location.origin);

        // Socket connection successful
        socket.on('connect', function() {
            console.log('Connected to server');
```

```
        });

        // Listen for game updates from the server
        socket.on('game_update', function(data) {
            console.log('Game update received:', data);
            updateBoard(data.boardState);  // Assuming data contains
board state
        });

        // Listen for game winner event
        socket.on('game_winner', function(data) {
            alert('Winner: ' + data.winner);
        });

        // Attach socket to window object to avoid re-initializing
        window.socket = socket;
    }
});

// Function to emit a move event
function makeMove(cellIndex) {
    if (window.socket) {
        console.log('Player clicked on cell ' + cellIndex);
        window.socket.emit('make_move', { cell: cellIndex });

        // Disable the clicked cell or update UI
        document.getElementById('cell-' +
cellIndex).classList.add('disabled');
    } else {
        console.error("Socket connection not initialized.");
    }
}

// Update the game board visually based on board state
function updateBoard(boardState) {
    for (let i = 0; i < 9; i++) {
        const cell = document.getElementById('cell-' + i);
        cell.textContent = boardState[i] || '';
        cell.classList.remove('disabled');
```

```
    }
}

// Reset the game on server and clear the board
function resetGame() {
    if (window.socket) {
        window.socket.emit('reset_game');

        document.querySelectorAll('.cell').forEach(cell => {
            cell.textContent = '';
            cell.classList.remove('disabled');
        });
    } else {
        console.error("Socket connection not initialized.");
    }
}
```

styles.css

```
body {
    font-family: 'Roboto', sans-serif;
    background: linear-gradient(135deg, #f3f5f7 0%, #e2e6ea 100%);
    color: #2d2d2d;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    line-height: 1.6;
    transition: all 0.3s ease;
}

h1, h2 {
    color: #333;
    text-align: center;
    margin-top: 1rem;
    font-weight: 700;
    text-transform: uppercase;
}
```

```css
.container {
    max-width: 800px;
    margin: 2rem auto;
    padding: 2.5rem;
    background-color: #ffffff;
    border-radius: 15px;
    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);
    transition: transform 0.3s ease;
}

.container:hover {
    transform: scale(1.01);
}

button {
    background: linear-gradient(45deg, #007bff, #0056b3);
    color: white;
    padding: 12px 35px;
    border: none;
    border-radius: 30px;
    font-size: 1rem;
    font-weight: bold;
    cursor: pointer;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
    transition: background 0.3s, transform 0.2s ease;
}

button:hover {
    background: linear-gradient(45deg, #0056b3, #007bff);
    transform: translateY(-3px);
}

button:active {
    transform: translateY(1px);
}

.dashboard, .leaderboard {
    display: flex;
```

```
    flex-direction: column;
    align-items: center;
}

.table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 1.5rem;
    overflow: hidden;
    border-radius: 8px;
}

.table th, .table td {
    padding: 16px;
    border-bottom: 1px solid #ddd;
    text-align: left;
    font-size: 1rem;
}

.table th {
    background-color: #007bff;
    color: white;
    font-weight: 700;
}

.table tr:nth-child(even) {
    background-color: #f8fafc;
}

.table tr:hover {
    background-color: #e7f0fa;
}

.alert {
    color: #ff5a5a;
    font-weight: bold;
    margin: 1.5rem 0;
    font-size: 1.1rem;
}
```

```css
.game-container {
    margin: 20px auto;
    max-width: 420px;
    padding: 2rem;
    border-radius: 15px;
    background-color: #ffffff;
    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);
    transition: transform 0.3s ease;
}

.game-container:hover {
    transform: scale(1.02);
}

#game-board {
    display: grid;
    grid-template-columns: repeat(3, 100px);
    grid-template-rows: repeat(3, 100px);
    gap: 15px;
    margin-top: 1.5rem;
}

.cell {
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 2.5rem;
    width: 100px;
    height: 100px;
    background-color: #e9ecef;
    border: 2px solid #007bff;
    border-radius: 12px;
    cursor: pointer;
    transition: background-color 0.3s, transform 0.2s;
}

.cell:hover {
    background-color: #d8e2ec;
```

```
    transform: scale(1.05);
}

.cell.X {
    color: #28a745;
    font-weight: bold;
}

.cell.O {
    color: #dc3545;
    font-weight: bold;
}

.winner {
    background-color: #ffc107 !important;
    color: #2d2d2d !important;
}

.notification {
    margin-top: 20px;
    color: #555;
    font-weight: bold;
    font-size: 1.2rem;
}

nav {
    background-color: #3a3f47;
    padding: 15px;
    border-bottom: 3px solid #007bff;
    width: 100%;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
}

.nav-links {
    display: flex;
    justify-content: center;
    align-items: center;
}
```

```css
.nav-links a {
    color: #f8f9fa;
    text-decoration: none;
    padding: 0 20px;
    font-size: 18px;
    font-weight: 500;
    transition: color 0.3s;
}

.nav-links a:hover {
    color: #adb5bd;
}

footer {
    background-color: #3a3f47;
    color: #adb5bd;
    text-align: center;
    padding: 1.5rem;
    margin-top: 25px;
    font-size: 0.9rem;
}

footer p {
    margin: 0;
}

.nav-links a + a::before {
    content: " | ";
    padding: 0 10px;
    color: #adb5bd;
}

.nav-links a:first-child::before {
    content: "";
}

@media (max-width: 768px) {
    .container, .game-container {
        padding: 1.5rem;
```

```
    }

    #game-board {
        grid-template-columns: repeat(3, 1fr);
        gap: 10px;
    }

    .cell {
        font-size: 2rem;
        width: 80px;
        height: 80px;
    }

    .nav-links {
        flex-direction: column;
    }

    .nav-links a {
        padding: 10px 0;
    }
}
/* General Link Styling */
a {
    color: #007bff; /* Default link color */
    text-decoration: none;
    font-weight: 600;
    padding: 8px 16px;
    border-radius: 6px;
    transition: color 0.3s, background-color 0.3s;
}

a:hover {
    color: #ffffff;
    background-color: #007bff; /* Background color on hover */
    box-shadow: 0 4px 12px rgba(0, 123, 255, 0.3); /* Add shadow on
hover */
}

a:active {
```

```
    transform: translateY(1px); /* Slight movement on click */
}

a:focus {
    outline: 2px dashed #007bff; /* Outline for accessibility */
    outline-offset: 3px;
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Leaderboard</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstra
p.min.css">
</head>
<body>
    <h1 class="text-center my-4">Leaderboard</h1>
    <div class="container">
        <table class="table table-bordered table-striped">
            <thead class="thead-dark">
                <tr>
                    <th>Email</th>
                    <th>Games Won</th>
                    <th>Games Lost</th>
                    <th>Total Score</th>
                </tr>
            </thead>
            <tbody>
                {% for entry in leaderboard %}
                <tr>
                    <td>{{ entry.email }}</td>
                    <td>{{ entry.games_won }}</td>
                    <td>{{ entry.games_lost }}</td>
```

```
                    <td>{{ entry.total_score }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
</body>
</html>
```

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Tic-Tac-Toe Game</title>
    <!-- Add any common CSS or JS files -->
    <link rel="stylesheet" href="{{ url_for('static',
filename='styles.css') }}">
    <script src="{{ url_for('static', filename='scripts.js') }}"
defer></script>

</head>
<body>
    <header>
        <nav>
            <div class="nav-links">
                <a href="{{ url_for('home') }}">Home</a>
                {% if 'loggedin' in session %}
                    <a href="{{ url_for('dashboard')
}}">Dashboard</a> |
                    <a href="{{ url_for('game') }}">Play Game</a> |
                    <a href="{{ url_for('leaderboard')
}}">Leaderboard</a> |
                    <a href="{{ url_for('logout') }}">Logout</a>
                {% else %}
                    <a href="{{ url_for('login') }}">Login</a> |
```

```
                    <a href="{{ url_for('register') }}">Register</a>
                {% endif %}
            </div>
        </nav>
    </header>

    <main>
        <!-- The content specific to each page will be injected here
-->
        {% block content %}
        {% endblock %}
    </main>

    <footer>
        <p>&copy; 2024 Tic-Tac-Toe Game. All rights reserved.</p>
    </footer>

    <script
src="https://cdn.jsdelivr.net/npm/socket.io@4.6.0/dist/socket.io.min.
js"></script>
    <script src="{{ url_for('static', filename='scripts.js')
}}"></script>
</body>
</html>
```

register.html

```
{% extends 'base.html' %}

{% block content %}
    <h1>Register</h1>
    <form method="POST">
        <div>
            <label for="email">Email</label>
            <input type="email" name="email" id="email" required>
        </div>
        <div>
            <label for="password">Password</label>
            <input type="password" name="password" id="password"
```

```
required>
        </div>
        <button type="submit">Register</button>
    </form>
    <p>Already have an account? <a href="{{ url_for('login')
}}">Login here</a></p>
{% endblock %}
```

login.html

```
{% extends 'base.html' %}

{% block content %}
    <h1>Login</h1>
    <form method="POST">
        <div>
            <label for="email">Email</label>
            <input type="email" name="email" id="email" required>
        </div>
        <div>
            <label for="password">Password</label>
            <input type="password" name="password" id="password"
required>
        </div>
        <button type="submit">Login</button>
    </form>
    <p>Don't have an account? <a href="{{ url_for('register')
}}">Register here</a></p>
{% endblock %}
```

user.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
```

```
    <title>User List</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='styles.css') }}">
</head>
<body>
    <h2>User List</h2>

    <!-- Display the results in a table -->
    <table border="1">
        <thead>
            <tr>
                <th>User ID</th>
                <th>Email</th>
            </tr>
        </thead>
        <tbody>
            {% for user in users %}
                <tr>
                    <td>{{ user['id'] }}</td>
                    <td>{{ user['email'] }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
</html>
```

dashboard.html

```
{% extends 'base.html' %}
{% block content %}
    <div class="dashboard-container">
        <h1>Welcome to your Dashboard, {{ email }}</h1> <!--
Displaying the logged-in user's email -->
        <!-- Inside dashboard.html -->
        <div class="dashboard-options">
            <h3>Select one of the options below:</h3>
            <ul>
                <li><a href="{{ url_for('game') }}"
```

```
class="dashboard-link">Play Game</a></li>
                <li><a href="{{ url_for('leaderboard') }}"
class="dashboard-link">View Leaderboard</a></li>
                <li><a href="{{ url_for('logout') }}"
class="dashboard-link">Logout</a></li>
            </ul>
        </div>
    </div>
    <!-- Include the Socket.io library from CDN -->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.6.0/socket.io
.min.js" defer></script>
<!-- Include your own custom scripts.js -->
<script src="{{ url_for('static', filename='scripts.js')
}}"></script>

{% endblock %}
```

game.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Game Room</title>

    <link rel="stylesheet" href="{{ url_for('static',
filename='styles.css') }}">
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.1/socket.io
.min.js"></script>
    <script>
        // Pass session email to game.js for setting player symbol
        const sessionEmail = "{{ session['email'] }}";
    </script>
</head>
<body>
```

```
    <div class="game-container">
        <h1>Tic-Tac-Toe</h1>

        <!-- Display the room and players -->
        <div id="room-info">
            <h3>Room: <span id="room-name">{{
request.args.get('room') }}</span></h3>
            <div id="players">Players in room:</div>
        </div>

        <!-- Display game status -->
        <div id="status">Waiting for players to join...</div>

        <!-- Game board -->
        <div id="game-board">
            <div class="row">
                <div id="cell-0" class="cell"></div>
                <div id="cell-1" class="cell"></div>
                <div id="cell-2" class="cell"></div>
            </div>
            <div class="row">
                <div id="cell-3" class="cell"></div>
                <div id="cell-4" class="cell"></div>
                <div id="cell-5" class="cell"></div>
            </div>
            <div class="row">
                <div id="cell-6" class="cell"></div>
                <div id="cell-7" class="cell"></div>
                <div id="cell-8" class="cell"></div>
            </div>
        </div>
    </div>
    <!-- JavaScript for game logic -->
    <script src="{{ url_for('static', filename='game.js')
}}"></script>
</body>
</html>
```

games.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Game Results</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='styles.css') }}">
</head>
<body>
    <h2>Game Results</h2>
    <!-- Display the results in a table -->
    <table border="1">
        <thead>
            <tr>
                <th>Game ID</th>
                <th>Winner</th>
                <th>Loser</th>
            </tr>
        </thead>
        <tbody>
            {% for row in game_results %}
                <tr>
                    <td>{{ row['id'] }}</td>
                    <td>{{ row['winner_email'] }}</td>
                    <td>{{ row['loser_email'] }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
</html>
```

home.html

```
{% extends 'base.html' %}
{% block content %}
    <h1>Welcome to the Tic-Tac-Toe Game!</h1>
    <p>This is the homepage. You can log in, register, or play the
game.</p>
    <a href="{{ url_for('login') }}">Log in</a> | <a href="{{
url_for('register') }}">Register</a>
{% endblock %}
```

leaderboard.html

```
{% extends 'base.html' %}
{% block content %}
    <h1 class="text-center my-4">Leaderboard</h1>
    <div class="container">
        <table class="table table-bordered table-striped">
            <thead class="thead-dark">
                <tr>
                    <th>Email</th>
                    <th>Games Won</th>
                    <th>Games Lost</th>
                    <th>Total Score</th>
                </tr>
            </thead>
            <tbody>
                {% for entry in leaderboard %}
                <tr>
                    <td>{{ entry.email }}</td>
                    <td>{{ entry.games_won }}</td>
                    <td>{{ entry.games_lost  }}</td>
<td>{{ entry.total_score }}</td></tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
{% endblock %}
```

admin_dashboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Admin Dashboard</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='styles.css') }}">
</head>
<body>
    <!-- Header Section -->
    <header>
        <h1>Admin Dashboard</h1>
        <nav>
            <ul>
                <li><a href="{{ url_for('admin_leaderboard')
}}">Leaderboard</a></li>
                <li><a href="{{ url_for('gameslist')
}}">Games</a></li>
                <li><a href="{{ url_for('users') }}">User
List</a></li>
            </ul>
            <a href="/logout" class="logout">Logout</a>
        </nav>
    </header>

    <!-- Main Content Section -->
    <main>
        <section>
            <h2>Welcome, Admin</h2>
            <p>Use the navigation above to access different sections
of the dashboard.</p>
        </section>
    </main>
    <!-- Footer Section -->
    <footer>
```

```
        <p>&copy; 2024 Admin Dashboard. All rights reserved.</p>
    </footer>
    <!-- Scripts -->
    <script
src="https://cdn.jsdelivr.net/npm/socket.io@4.6.0/dist/socket.io.min.
js"></script>
    <script src="{{ url_for('static', filename='scripts.js')
}}"></script>
</body>
</html>
```

admin_leaderboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Leaderboard</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstra
p.min.css">
</head>
<body>
    <h1 class="text-center my-4">Leaderboard</h1>
    <div class="container">
        <table class="table table-bordered table-striped">
            <thead class="thead-dark">
                <tr>
                    <th>Email</th>
                    <th>Games Won</th>
                    <th>Games Lost</th>
                    <th>Total Score</th>
                </tr>
            </thead>
            <tbody>
                {% for entry in leaderboard %}
                <tr>
```

```
                    <td>{{ entry.email }}</td>
                    <td>{{ entry.games_won }}</td>
                    <td>{{ entry.games_lost }}</td>
                    <td>{{ entry.total_score }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
</body>
</html>
```

2.Backend:

db.sql

```sql
CREATE DATABASE IF NOT EXISTS game_db;

USE game_db;

DROP TABLE IF EXISTS Users;

DROP TABLE IF EXISTS Leaderboard;

DROP TABLE IF EXISTS GameResults;

CREATE TABLE Users (

    id INT AUTO_INCREMENT PRIMARY KEY,

    email VARCHAR(255) NOT NULL UNIQUE,

    password VARCHAR(255) NOT NULL

);


CREATE TABLE GameResults (
```

```
    id INT AUTO_INCREMENT PRIMARY KEY,
    game_id INT NOT NULL,
    winner_id INT,
    loser_id INT,
    FOREIGN KEY (winner_id) REFERENCES Users(id) ON DELETE SET NULL,
    FOREIGN KEY (loser_id) REFERENCES Users(id) ON DELETE SET NULL
);
CREATE TABLE Leaderboard (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    games_won INT DEFAULT 0,
    games_lost INT DEFAULT 0,
    games_drawn INT DEFAULT 0,
    total_score DECIMAL(10, 2) DEFAULT 0.00,
    FOREIGN KEY (user_id) REFERENCES Users(id) ON DELETE CASCADE
);
```

database_setup.py

```
import mysql.connector
from mysql.connector import Error
def create_database_connection():
    """Establishes a connection to MySQL."""
```

```
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="your_username",
            password="your_password"
        )
        if connection.is_connected():
            print("Connected to MySQL server")
        return connection
    except Error as e:
        print(f"Error: {e}")
        return None
def execute_script(connection, script_file):
    """Executes SQL script from a file."""
    try:
        cursor = connection.cursor()
        with open(script_file, 'r') as file:
            sql_script = file.read()
        for statement in sql_script.split(';'):
            if statement.strip():
                cursor.execute(statement)
        connection.commit()
        print("Database and tables created successfully.")
```

```
    except Error as e:
        print(f"Error executing script: {e}")


def main():
    connection = create_database_connection()
    if connection:
        execute_script(connection, 'db.sql')
        connection.close()
        print("Connection closed.")
if __name__ == "__main__":
    main()
```

app.py

```
from flask import Flask, render_template, request, redirect, url_for, flash,
session
from flask_mysqldb import MySQL
from werkzeug.security import generate_password_hash, check_password_hash
import MySQLdb.cursors
from functools import wraps
from flask_socketio import SocketIO, emit, join_room, leave_room
import logging
from flask_cors import CORS
import pandas as pd
# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app = Flask(__name__)
CORS(app, resources={r"/*": {"origins": "*"}})
app.config.from_object('config.Config')
mysql = MySQL(app)
app.secret_key = app.config['SECRET_KEY']
socketio = SocketIO(app, cors_allowed_origins="*")
```

```
admin_email='basaksoumyadeep04@gmail.com'
games = {}

# Login-required decorator
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'loggedin' not in session:
            flash('Please log in to access this page.')
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        hashed_password = generate_password_hash(password,
method='pbkdf2:sha256')

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        try:
            cursor.execute('INSERT INTO Users (email, password) VALUES (%s,
%s)', (email, hashed_password))
            mysql.connection.commit()

            cursor.execute('SELECT id FROM Users WHERE email = %s', (email,))
            user = cursor.fetchone()
            cursor.execute('INSERT INTO Leaderboard (user_id, games_won,
games_lost, games_drawn, total_score) VALUES (%s, 0, 0, 0, 0)', (user['id'],))
            mysql.connection.commit()

            flash('Registration successful! Please log in.')
            return redirect(url_for('login'))
        except MySQLdb.IntegrityError:
            flash('Email already registered. Please log in.')
            return redirect(url_for('login'))
        finally:
            cursor.close()
    return render_template('register.html')
```

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        try:
            cursor.execute('SELECT * FROM Users WHERE email = %s', (email,))
            user = cursor.fetchone()

            if user and check_password_hash(user['password'], password):
                session['loggedin'] = True
                session['email'] = user['email']
                session['user_id'] = user['id']
                flash('Login successful!', 'success')
                if(email==admin_email):
                    return redirect(url_for('admin_dashboard'))
                else:
                    return redirect(url_for('dashboard'))
            else:
                flash('Incorrect email or password', 'danger')
        finally:
            cursor.close()
    return render_template('login.html')

@app.route('/dashboard')
@login_required
def dashboard():
    return render_template('dashboard.html', email=session['email'])

@app.route('/admin_dashboard')
def admin_dashboard():
    return render_template('admin_dashboard.html')

@app.route('/users')
def users():
    try:
        # Connect to the database
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        # Exclude the admin email and fetch other users
          # Replace with actual admin email
        query = """
```

```
        SELECT id, email
        FROM Users
        WHERE email != %s
        """
        cursor.execute(query, (admin_email,))
        users = cursor.fetchall()

        # Close the cursor
        cursor.close()

        # Render the HTML template with user data
        return render_template('users.html', users=users)

    except Exception as e:
        print(f"An error occurred: {e}")
        return f"An error occurred while fetching user list: {e}", 500

@app.route('/gameslist')
def gameslist():
    try:
        # Connect to the database
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        # Fetch data with email instead of IDs for winner and loser
        query = """
        SELECT
            gr.id,
            u1.email AS winner_email,
            u2.email AS loser_email
        FROM
            GameResults gr
        JOIN
            Users u1 ON gr.winner_id = u1.id
        JOIN
            Users u2 ON gr.loser_id = u2.id
        WHERE
            u1.email != %s AND u2.email != %s
        """
        cursor.execute(query, (admin_email,admin_email,))
        results = cursor.fetchall()
        # Close the cursor
        cursor.close()

        # Pass the results to the HTML template
        return render_template('games.html', game_results=results)
    except Exception as e:
```

```
        # Print and return the specific error message
        print(f"An error occurred: {e}")
        return f"An error occurred while fetching game results: {e}", 500
@app.route('/room')
@login_required
def room():
    return render_template('room.html', email=session['email'])
@app.route('/game')
@login_required
def game():
    return render_template('game.html')
@socketio.on('join_game')
@login_required
def on_join(data):
    room = data['room']
    username = session.get('email')
    if not username:
        return
    join_room(room)
    if room not in games:
        games[room] = {'players': [], 'turn': 0, 'board': [''] * 9,
'completed': False}
    game = games[room]
    if username not in game['players']:
        game['players'].append(username)
    emit('update_room', {'players': game['players']}, room=room)
    if len(game['players']) == 2:
        emit('game_start', {'players': game['players']}, room=room)
@socketio.on('make_move')
def on_move(data):
    room = data['room']
    move_index = data['move']
    player = session.get('email')
    game = games.get(room)
    # Ensure game exists, isn't completed, and it's the player's turn
    if not game:
        emit('error', {'message': 'Game not found.'})
        return
    if game['completed']:
        emit('error', {'message': 'Game is already completed.'})
        return
    # Check if the move is from the correct player
    if player != game['players'][game['turn']]:
        emit('error', {'message': 'Not your turn.'}, room=player)
        return
```

```
    # Check if the cell is already occupied
    if game['board'][move_index] != '':
        emit('invalid_move', {'message': 'Cell already taken.'}, room=player)
        return
    # Make the move
    symbol = 'X' if game['turn'] == 0 else 'O'
    game['board'][move_index] = symbol
    emit('move_made', {'index': move_index, 'symbol': symbol}, room=room)
    # Check for a winner after the move
    result = check_winner(game['board'], game['players'][0],
game['players'][1],room)
    if result:
        game['completed'] = True
        emit('game_won', {'winner': result['winner']}, room=room)
        update_winner(result['winner'])
        update_loser(result['loser'])
        del games[room]  # Clean up completed game
    else:
        game['turn'] = (game['turn'] + 1) % 2  # Switch turns
def check_winner(board, player1_email, player2_email, room):
    winning_combinations = [
        (0, 1, 2), (3, 4, 5), (6, 7, 8),
        (0, 3, 6), (1, 4, 7), (2, 5, 8),
        (0, 4, 8), (2, 4, 6)
    ]
    for a, b, c in winning_combinations:
        if board[a] == board[b] == board[c] and board[a] != '_'and
board[a]!='':
            # Determine the winner and loser based on the board state
            winner_email = player1_email if board[a] == 'X' else player2_email
            loser_email = player2_email if board[a] == 'X' else player1_email

            # Update the game result in the database
            record_game_result(room, winner_email, loser_email)
            return {'winner': winner_email, 'loser': loser_email}
    return None  # No winner found

def record_game_result(room, winner_email, loser_email):
    cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
    try:
        # Fetch winner and loser IDs
        cursor.execute('SELECT id FROM Users WHERE email = %s',
(winner_email,))
        winner_id = cursor.fetchone()['id']
        cursor.execute('SELECT id FROM Users WHERE email = %s', (loser_email,))
```

```
        loser_id = cursor.fetchone()['id']

        # Insert game result into GameResults table
        cursor.execute(
            'INSERT INTO GameResults (game_id, winner_id, loser_id) VALUES (%s,
%s, %s)',
            (room, winner_id, loser_id)
        )
        mysql.connection.commit()
    except Exception as e:
        logger.error(f'Error recording game result: {e}')
    finally:
        cursor.close()

def update_winner(winner_email):
    cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

    # Fetch winner's user ID
    cursor.execute('SELECT id, games_won, games_lost FROM Users JOIN
Leaderboard ON Users.id = Leaderboard.user_id WHERE email = %s',
(winner_email,))
    winner = cursor.fetchone()

    # Increment games_won and calculate total_score
    games_won = winner['games_won'] + 1
    total_games = games_won + winner['games_lost']
    total_score = (games_won / total_games) * 100

    cursor.execute(
        'UPDATE Leaderboard SET games_won = %s, total_score = %s WHERE user_id
= %s',
        (games_won, total_score, winner['id'])
    )
    mysql.connection.commit()
    cursor.close()

def update_loser(loser_email):
    cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

    # Fetch loser's user ID
    cursor.execute('SELECT id, games_won, games_lost FROM Users JOIN
Leaderboard ON Users.id = Leaderboard.user_id WHERE email = %s',
(loser_email,))
    loser = cursor.fetchone()
    # Increment games_lost and calculate total_score
```

```
    games_lost = loser['games_lost'] + 1
    total_games = loser['games_won'] + games_lost
    total_score = (loser['games_won'] / total_games) * 100
    cursor.execute(
        'UPDATE Leaderboard SET games_lost = %s, total_score = %s WHERE user_id
= %s',
        (games_lost, total_score, loser['id'])
    )
    cursor.execute('UPDATE Leaderboard SET games_lost=games_lost-1 WHERE
user_id=%s',(loser['id']))
    mysql.connection.commit()
    cursor.close()

@app.route('/admin_leaderboard')
@login_required
def admin_leaderboard():
    update_leaderboard()
    # Connect to the database
    cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
    # SQL query to fetch leaderboard data, excluding emails in admin_dashboard
    query = """
    SELECT Users.email,
           Leaderboard.games_won,
           Leaderboard.games_lost,
           Leaderboard.total_score
    FROM Leaderboard
    JOIN Users ON Leaderboard.user_id = Users.id
    WHERE Users.email != %s
    ORDER BY Leaderboard.total_score DESC
    """
    cursor.execute(query, (admin_email,))
    # cursor.execute(query)
    leaderboard_data = cursor.fetchall()
    # Log the leaderboard data
    logger.info("Leaderboard Data: %s", leaderboard_data)
    # Close the cursor
    cursor.close()
    # Render the HTML template with leaderboard data
    return render_template('admin_leaderboard.html',
leaderboard=leaderboard_data)



@app.route('/leaderboard')
@login_required
```

```
def leaderboard():
    update_leaderboard()
    cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
    cursor.execute('SELECT Users.email, Leaderboard.games_won,
Leaderboard.games_lost, Leaderboard.total_score FROM Leaderboard JOIN Users ON
Leaderboard.user_id = Users.id ORDER BY Leaderboard.total_score DESC')
    leaderboard_data = cursor.fetchall()
    logger.info("Leaderboard Data: %s", leaderboard_data)
    cursor.close()
    return render_template('leaderboard.html', leaderboard=leaderboard_data)
def update_leaderboard():
    cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

    try:
        # Reset the Leaderboard
        # cursor.execute('UPDATE Leaderboard SET games_won = 0, games_lost = 0,
games_drawn = 0, total_score = 0.00')

        # Update wins
        cursor.execute(
            '''
            UPDATE Leaderboard AS lb
            JOIN (
                SELECT winner_id AS user_id, COUNT(*) AS games_won
                FROM GameResults
                GROUP BY winner_id
            ) AS win_counts
            ON lb.user_id = win_counts.user_id
            SET lb.games_won = win_counts.games_won
            '''
        )

        # Update losses
        cursor.execute(
            '''
            UPDATE Leaderboard AS lb
            JOIN (
                SELECT loser_id AS user_id, COUNT(*) AS games_lost
                FROM GameResults
                GROUP BY loser_id
            ) AS loss_counts
            ON lb.user_id = loss_counts.user_id
            SET lb.games_lost = loss_counts.games_lost''')
        # Calculate total games for each user to compute total_score
        cursor.execute(
```

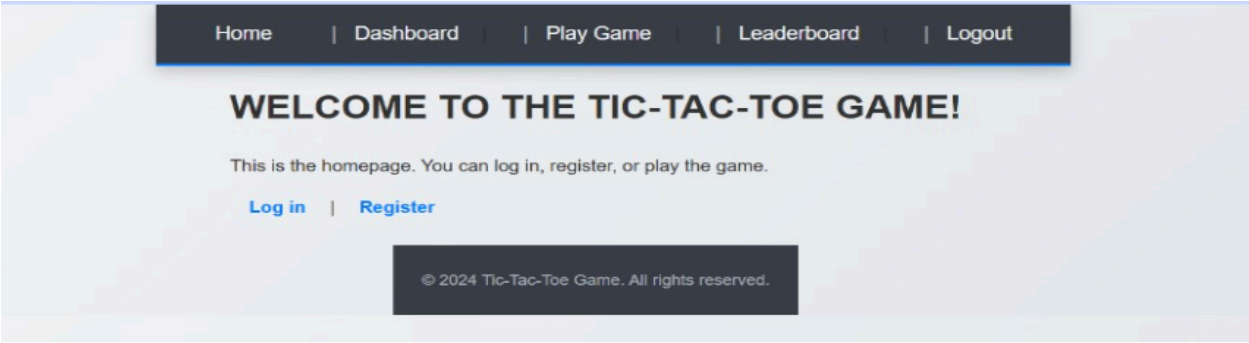
```
            '''
            UPDATE Leaderboard
            SET total_score = CASE
                WHEN games_won + games_lost + games_drawn > 0
                THEN (games_won / (games_won + games_lost + games_drawn)) * 100
                ELSE 0
            END
            ''')
        mysql.connection.commit()
    except Exception as e:
        logger.error(f'Error updating leaderboard: {e}')
        mysql.connection.rollback()
    finally:
        cursor.close()
@app.route('/logout')
@login_required
def logout():
    session.clear()
    flash('You have been logged out.')
    return redirect(url_for('home'))

if __name__ == "__main__":
    socketio.run(app, debug=True)
```

config.py

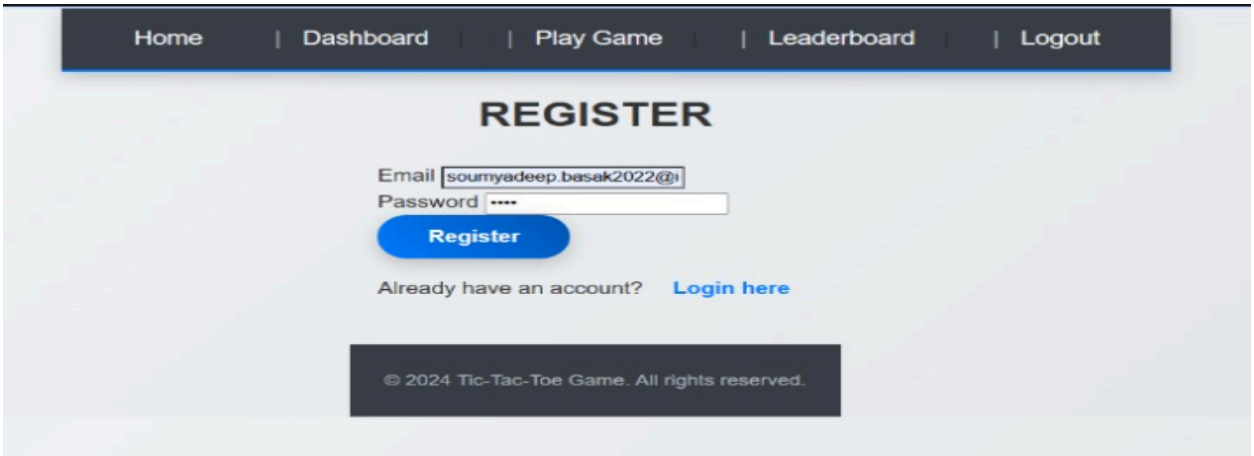
```
class Config:
    SECRET_KEY = 'my_super_secret_key_that_is_long_and_random_123456'

    # MySQL configurations for a single database
    MYSQL_HOST = 'your_host'
    MYSQL_USER = 'your_user'
    MYSQL_PASSWORD = 'your_mysql_password'
    MYSQL_DB = 'game_db'  # Single database for both authentication and stats
```

Screenshots:

Home Screen

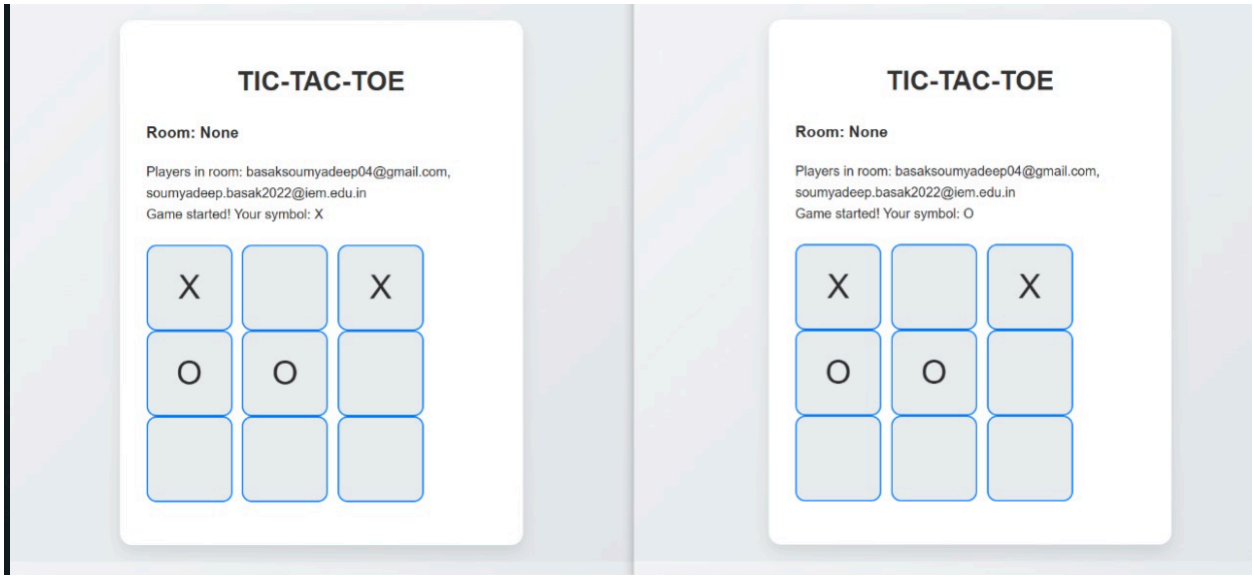


Register Screen

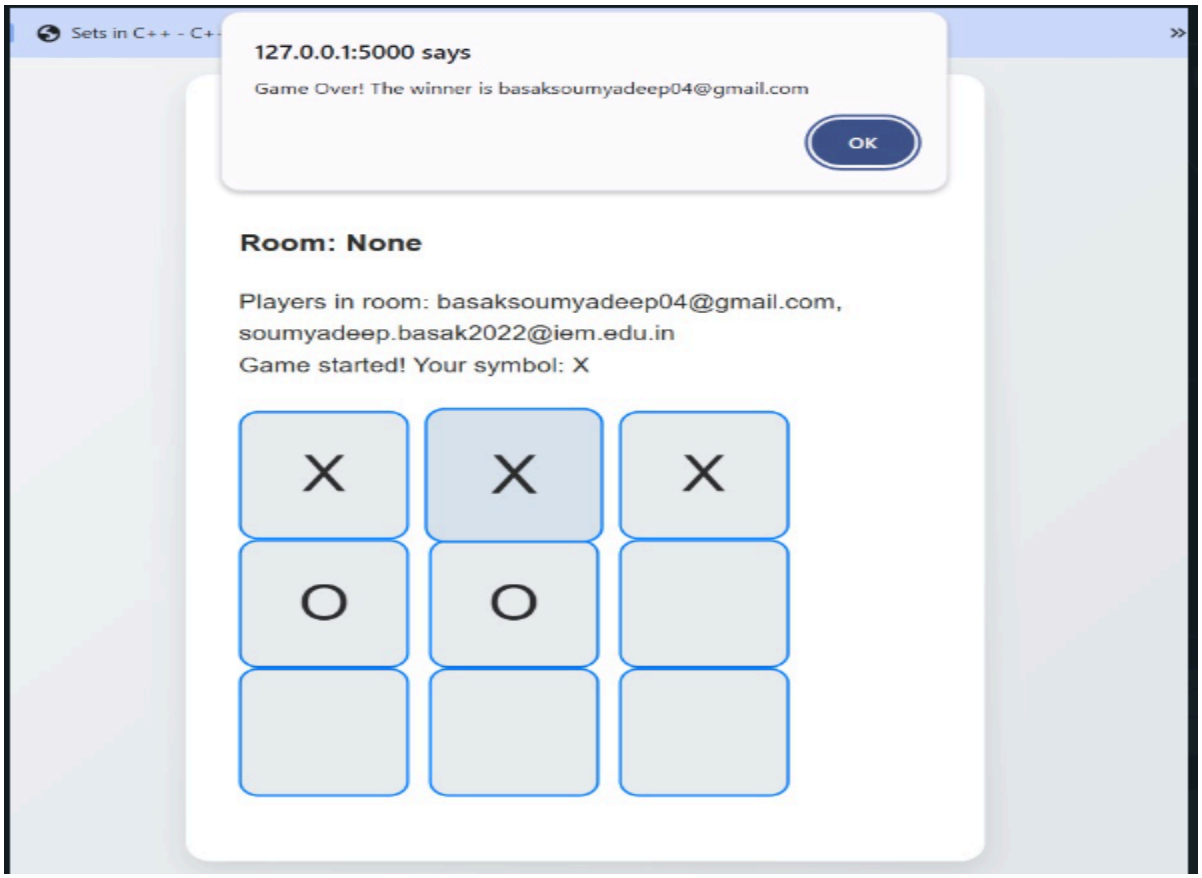


Login Screen

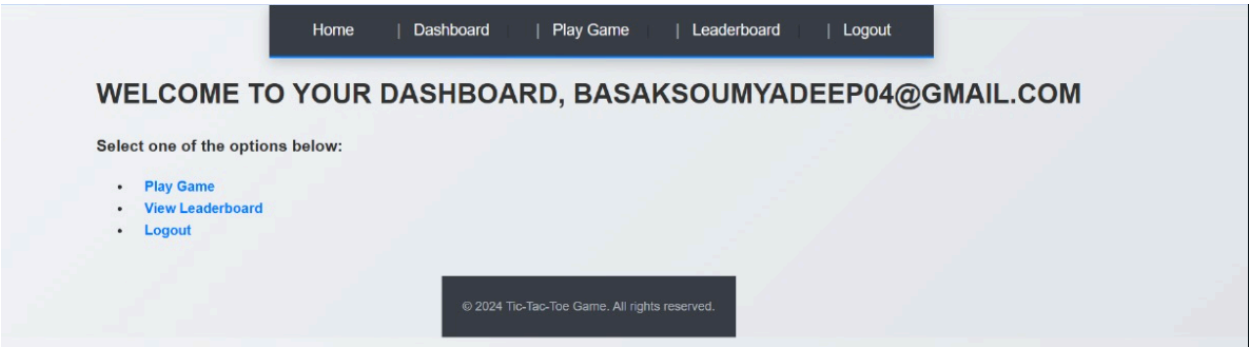
Game Screens:



Winner Screen:

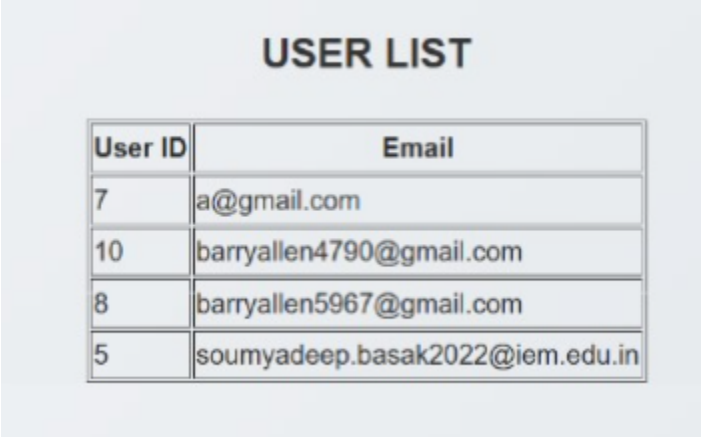


User Dashboard:

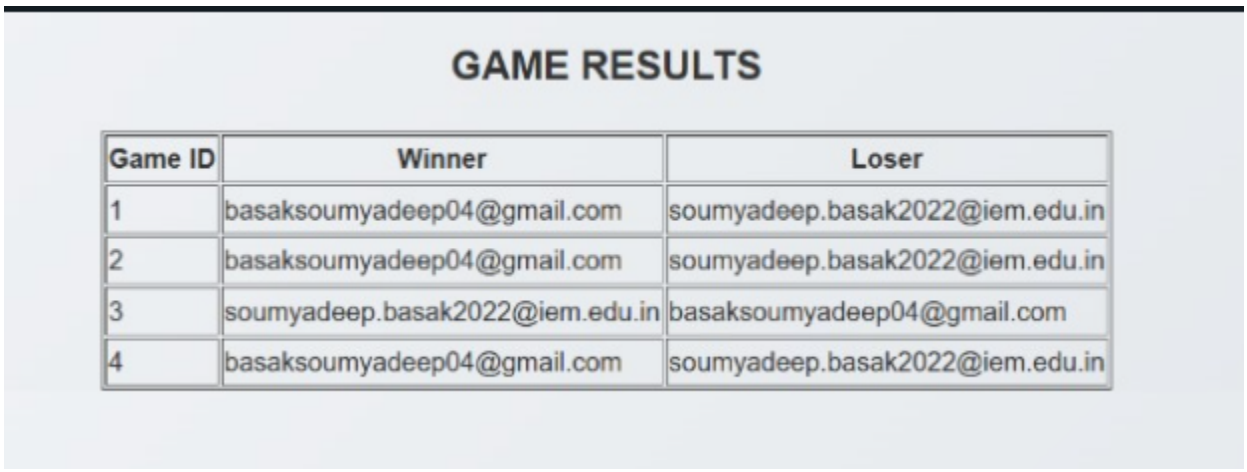


LeaderBoard:

User List(Admin Role):



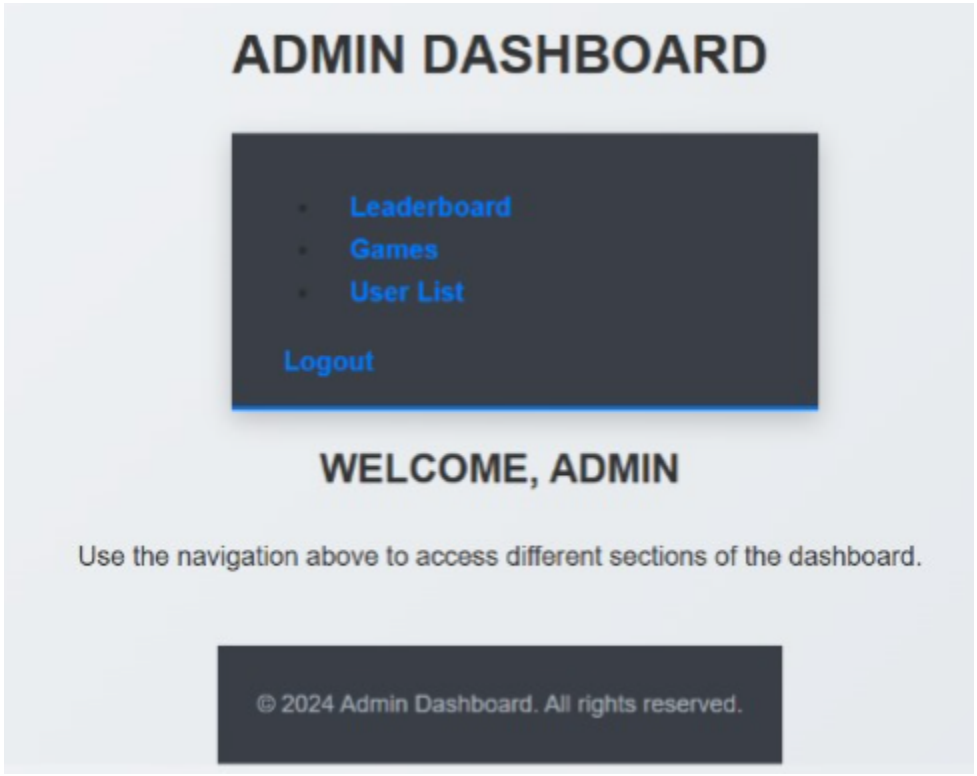
Game Results(Admin Role):



GAME RESULTS

Game ID	Winner	Loser
1	basaksoumyadeep04@gmail.com	soumyadeep.basak2022@iem.edu.in
2	basaksoumyadeep04@gmail.com	soumyadeep.basak2022@iem.edu.in
3	soumyadeep.basak2022@iem.edu.in	basaksoumyadeep04@gmail.com
4	basaksoumyadeep04@gmail.com	soumyadeep.basak2022@iem.edu.in

Admin Dashboard:



Future Look

Advanced Game Modes

- Tournaments: Create time-based or points-based tournaments where users can compete in multiple rounds. Winners of each game advance to the next round, with an overall tournament champion at the end.
- Customizable Game Rules: Allow users to play Tic-Tac-Toe with unique twists, such as larger grids (e.g., 4x4 or 5x5), variations like "3D Tic-Tac-Toe," or alternative winning conditions (e.g., a chain of four).

Enhanced User Profile & Stats

- Detailed User Profiles: Expand user profiles to include an avatar, bio, favorite game mode, and a history of recent games. Users could also earn badges based on milestones like consecutive wins or high ranks.
- Extended Game Statistics: Track and display more detailed statistics like average game duration, winning streaks, preferred opponents, and performance trends over time.

Community & Social Features

- Friends & Invites: Enable players to add friends, see when they are online, and invite them directly to games or tournaments. This could also include a "matchmaking" feature to find players of similar skill levels.
- In-Game Chat: Allow in-game and private chat between players. To maintain a positive environment, implement chat monitoring or a reporting mechanism.
- Achievements & Rewards System: Introduce achievements for actions like completing the first 10 games, winning a set number of matches, or participating in tournaments. Players could earn virtual rewards or points redeemable for in-game benefits.

Leaderboard Enhancements

- Global, Regional, and Friends Leaderboards: Create different types of leaderboards—global to see top players worldwide, regional to encourage local competition, and friends-only for more personalized tracking.
- Seasonal Rankings: Introduce seasonal rankings where players compete within a limited time frame, and the leaderboard resets after each season, offering rewards for top players.

References:

MDN Web Docs – JavaScript Game Development Basics

- A guide on using JavaScript for building simple games. Offers practical tutorials on event handling, game loops, and dynamic rendering

Socket.io Documentation

- The official documentation covers everything needed to build real-time bidirectional communication for multiplayer games. Includes methods for handling rooms, events, and connections.

MySQL Documentation

- The official MySQL documentation is invaluable for understanding database schema creation, optimization, and queries specific to leaderboard and game tracking data.