

# OTHELLO

**Secure Software Design and Programming**

**SWE 681**

Programming Project Report

By

Nishith Vadlamudi(G01373775)

Sathwik Vemulapally(G01380860)

(Under Professor Dr David A. Wheeler)

May 12, 2023



# Introduction

## Game Overview

Othello, a classic strategy board game that has been enjoyed for centuries. In this game, players take turns placing pieces on a 8x8 board, with the goal of flipping as many of their opponent's pieces as possible. The objective of the game is to have the majority of discs of your color at the end of the game. The game is won by the player with the most pieces of their color on the board when the game ends. [Wikipedia 2023]

This documentation will provide you with all the information you need to download, install, and play Othello, as well as troubleshoot any issues that may arise. We'll also explain the architecture of the game, including the technologies used and how the backend and frontend work together. Whether you're a seasoned Othello player or just discovering the game for the first time, this documentation will help you get the most out of your experience. **So, let's get started!**

## Rules Of the Game

### Overview

1. The game starts with an empty board. The first player (who usually plays with the black pieces) places a piece on any of the empty squares on the board. Then, the other player (who usually plays with the white pieces) places a piece on another empty square. Players take turns until the end of the game. [Hunter 2023]
2. A player can place a piece only if it surrounds at least one of the opponent's pieces. Surrounding means placing a piece in a way that there is an unbroken line of the player's pieces in a row, column, or diagonal direction, and the line ends with an opponent's piece.
3. Once a player places a piece in a way that it surrounds an opponent's piece(s), all the surrounded opponent's pieces are flipped over, changing their color to the player's color.
4. A player must make a move that flips at least one of the opponent's pieces. If a player cannot make such a move, they must pass their turn.
5. If a player cannot make a move that flips at least one of the opponent's pieces, they must pass their turn. If both players pass their turn consecutively, the game ends.
6. The game ends when neither player can make any more moves. This happens when the board is filled with pieces or when there are no more empty squares that can be surrounded by the opponent's pieces.
7. After the game ends, the player who has the most pieces on the board (of their color) wins. If both players have the same number of pieces, the game is a tie.

### How to Place and Flip Pieces

1. To place a piece, a player clicks on an empty square on the board. The square must be surrounded by the opponent's pieces in such a way that at least one of them can be flipped over.
2. After a player places a piece, the game checks if the placement is legal. If it is, the game flips all the necessary opponent's pieces.
3. To flip an opponent's piece, a player must place their own piece in a way that it surrounds an opponent's piece. All the opponent's pieces that are surrounded in this way will be flipped over and become the player's pieces.
4. Once a piece is flipped, it cannot be flipped back to its original color. It becomes a permanent part of the player's pieces.

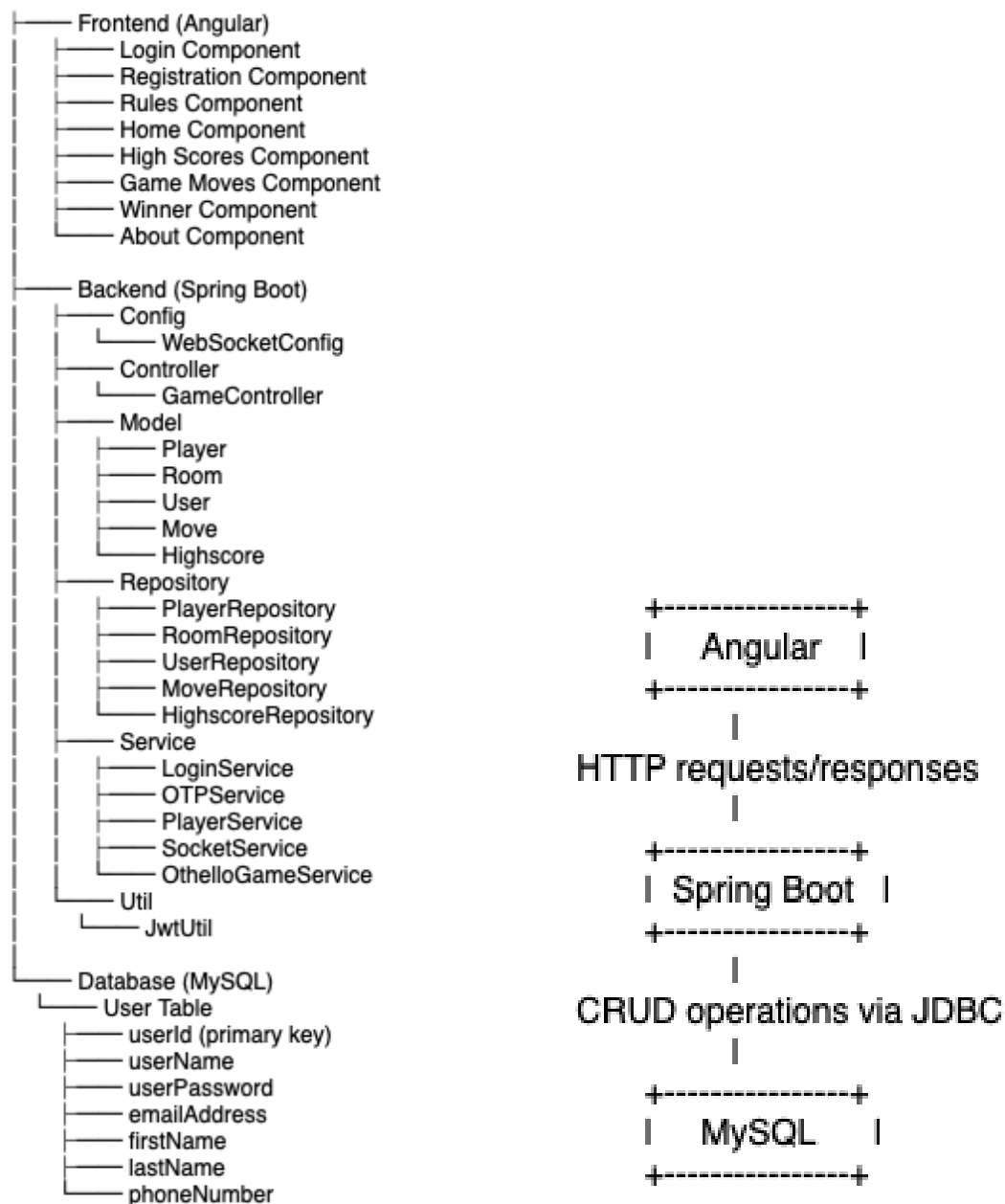
### Determining the Winner

1. After the game ends, the player who has the most pieces of their color on the board wins.
2. If both players have the same number of pieces of their color then the game is a tie.

## Game Architecture

The game is built using a client-server architecture, where the frontend is the client and the backend is the server. The client is responsible for rendering the game UI and handling user inputs, while the server is responsible for maintaining the game state, implementing the game logic, and storing player information in a database. [Alejandro 2023]

### Outline



## Frontend (Angular)

The frontend of the game is built using Angular, which is a popular front-end framework for building web applications. Angular provides a robust set of tools for building complex UIs and handling user interactions. In the case of the Othello game, Angular is responsible for rendering the game board, pieces, and other UI elements, as well as handling user inputs such as mouse clicks and touches. [Karolina 2021]

## Components

- **Login Component:** This component is responsible for rendering the login page where the user can enter their login credentials to access their account.
- **Registration Component:** This component is responsible for rendering the registration page where the user can create a new account by providing their username and password.
- **Rules Component:** This component is responsible for rendering the rules of the game to the user.
- **Home Component:** This component is responsible for rendering the home page of the game, which includes options to start a new game, view high scores, and view game moves.
- **High Scores Component:** This component is responsible for rendering the high scores of the game to the user.
- **Game Moves Component:** This component is responsible for rendering the moves made during a game to the user.
- **Winner Component:** This component is responsible for rendering the winner of the game to the user.

## Backend (Spring Boot)

The backend of the game is built using Spring Boot, which is a popular Java-based framework for building web applications. Spring Boot provides a set of tools and libraries for building robust and scalable web services. In the case of the Othello game, Spring Boot is responsible for maintaining the game state, implementing the game logic, and providing APIs for the frontend to communicate with the backend. [Mahipal 2023]

The communication between the frontend and the backend is implemented using HTTP requests and responses. When a user performs an action in the game, such as placing a piece on the board, the frontend sends an HTTP request to the backend, which updates the game state and sends a response back to the frontend with the updated game state. The frontend then updates the UI based on the new game state received from the backend.

## Config

- **GameController:** This class is responsible for handling incoming HTTP requests related to the game, such as starting a new game, making a move, and retrieving game history.

## Model

- **Player:** Contains information about a player, such as username, password, email, and score.
- **Room:** Contains information about a game room, such as room ID, number of players, and whether the game has started.
- **User:** Contains information about a user, such as username, password, and email.
- **Move:** Contains information about a move in the game, such as the player who made the move and the position on the board.

- Highscore: Contains information about the high scores of players.

## Repository

- PlayerRepository: Provides methods for accessing and manipulating player data in the MySQL database.
- RoomRepository: Provides methods for accessing and manipulating room data in the MySQL database.
- UserRepository: Provides methods for accessing and manipulating user data in the MySQL database.
- MoveRepository: Provides methods for accessing and manipulating move data in the MySQL database.
- HighscoreRepository: Provides methods for accessing and manipulating high score data in the MySQL database.

## Controller

- GameController: It includes various dependencies and repositories for user, game, and score data. There are methods for user registration, login, and OTP verification, as well as creating and joining game rooms. There is also a method for retrieving high scores

## Service

- LoginService: Provides methods for user authentication and login.
- OTPService: Provides methods for generating and verifying one-time passwords for user authentication.
- PlayerService: Provides methods for creating, updating, and retrieving player data.
- SocketService: Provides methods for handling real-time communication between players.
- OthelloGameService: Provides methods for managing Othello game logic, such as placing pieces and determining the winner.
- OthelloMoveRequest: Contains information about a move request in an Othello game, such as the player who made the move and the position on the board.

## Util

- JwtUtil: This class is responsible for generating and validating JSON Web Tokens (JWTs), which are used for user authentication and authorization.

## Database (MySQL)

The MySQL database is used to store player information such as usernames, passwords, and game scores. When a player creates an account, their information is stored in the database. When a player logs in to the game, the backend retrieves their information from the database and uses it to authenticate the player and retrieve their game history and scores.

## Installation Instructions

### Clone the repo

- Open the command prompt or terminal on your local machine.
- Change to the directory where you want to clone the repository.
- Run the following command to clone the repository:  
`git clone https://github.com/sathwikbittu/Othello.git`
- You now have a local copy of the Othello game project on your machine.

### Install MySQL for database and MySQL Workbench (Optional)

- Download the MySQL Community Server from the official MySQL website: <https://dev.mysql.com/downloads/mysql/>.
- Select your operating system and download the appropriate installer.
- Run the installer and follow the on-screen instructions to complete the installation.
- During the installation process, you will be prompted to set a root password for your MySQL Server. Make sure to remember this password as you will need it later.
- After the installation is complete, download and install MySQL Workbench from the official MySQL website: <https://dev.mysql.com/downloads/workbench/>.
- Launch MySQL Workbench and click on the "Local instance MySQL" option to connect to the MySQL Server.
- Enter the root password you set during the MySQL Server installation process.
- Once connected, you can create a new database and tables by running SQL queries or using the visual tools in MySQL Workbench.

### Install Spring boot for backend and Packages required

- Go to the Oracle Java SE Downloads page: <https://www.oracle.com/java/technologies/javase-downloads.html> and download java JDK and Verify that Java is installed by opening a command prompt or terminal and entering the command "java -version". This should display the version of Java you just installed.
- Go to the official Eclipse download page: <https://www.eclipse.org/downloads/> and install the eclipse IDE for best experience.

### Install spring boot extension

- Open Eclipse IDE.
- Go to the "Help" menu and select "Eclipse Marketplace."
- In the search box, type "Spring Tools" and press Enter.
- Click "Install" next to "Spring Tools 4 - for Spring Boot."
- Follow the prompts to complete the installation.
- Once the installation is complete, restart Eclipse.
- Click on "File" > "Import".
- Select "Existing Maven Projects" and click "Next".
- Browse to the directory where you cloned the Spring Boot project.

- Select the root directory of the project and click "Finish".
- Open the "application.properties" file located in the "src/main/resources" folder.
- Change the database connection properties (username and password) to match your local MySQL installation.
- Right-click on the project and select "Run As" > "Spring Boot App".
- Wait for the application to start up.

## Install Angular for frontend and packages required for the project to run

- Open a command prompt or terminal window.
- Make sure you have Node.js and npm installed on your machine. You can download and install Node.js from the official website: <https://nodejs.org/en/download/>
- Install the Angular CLI (Command Line Interface) globally by running the following command in the terminal:

*npm install -g @angular/cli*

- Navigate to the root directory of the project using the terminal.
- Install the project dependencies by running the following command:

*npm install*

- Once the dependencies are installed, run the command:
- ng s -o*
- Wait for the application to compile and the browser to open. If everything is set up correctly, you should see the login page of the Othello game.

## Operating Instructions

- Open the Othello application.
- You will be taken to the login page. Enter your username and password and click on the "Login" button and if you are not a registered user click on register and proceed.
- After successful login, you will be directed to the main dashboard.
- After logging in, you can click on the "Rules" section to view the rules of the Othello game.
- You can also click on the "High Scores" section to view the top scores of players who have played the game before.
- To view the moves made by both players during a game, you can go to the "Game Moves" section. Here, you will be able to see a list of completed games, along with the moves made by both players in each game.
- To create a new room, click on the "Create a Room" button.
- Your room will be created, Once all players have joined the room, the game will start automatically.
- Each player takes turns placing their pieces on the board by clicking on an empty square.
- If a player places a piece adjacent to an opponent's piece and forms a straight line of their pieces and the opponent's pieces, the opponent's pieces will be flipped to the player's color.
- The game continues until no more moves can be made or until a player resigns.
- The winner is the player with the most pieces of their color on the board at the end of the game.



# Assurance Case

## SSL/TLS IMPLEMENTATION

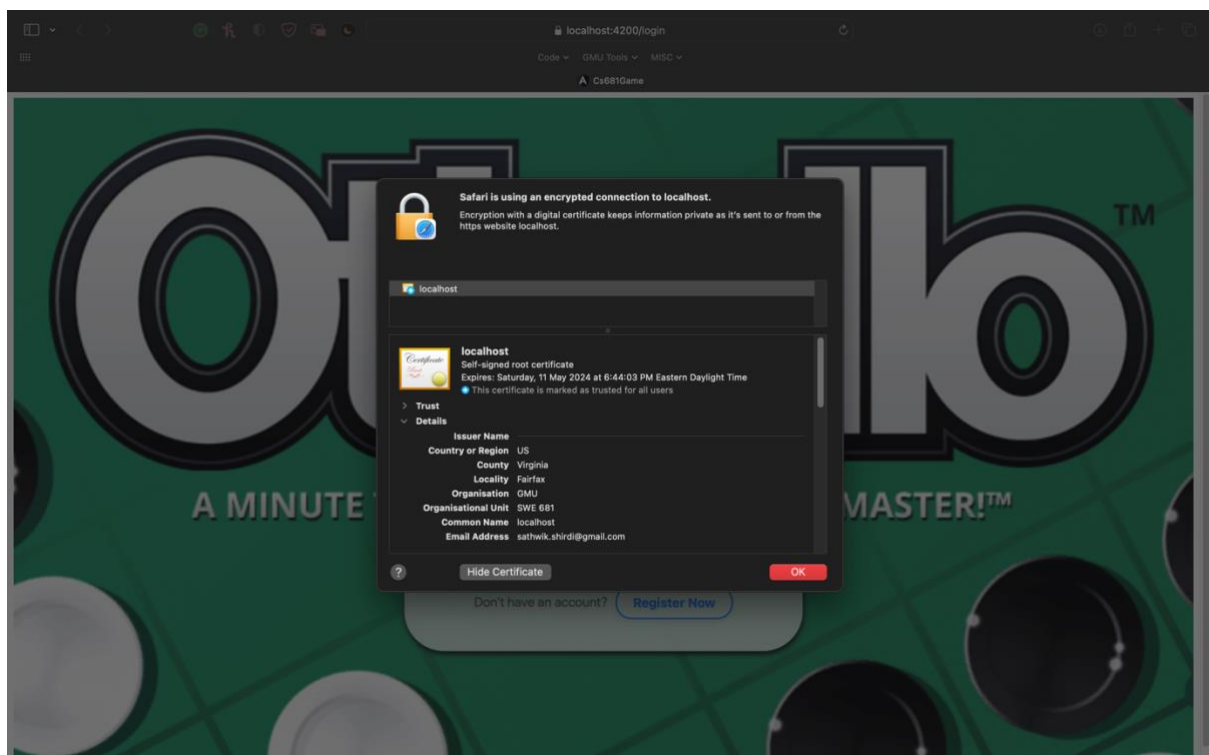
### CLAIM

By enabling a TLS (Transport Layer Security) connection in our game, we have provided a secure encryption channel that ensures the communication between the client and server is completely encrypted. TLS is a cryptographic protocol that provides data integrity and communication privacy, and is the successor to SSL (Secure Sockets Layer). With TLS, we ensure that sensitive data such as user names and passwords are protected from exposure and cannot be intercepted by third parties. This enhances the security of our game and provides a safe and secure experience for our players.

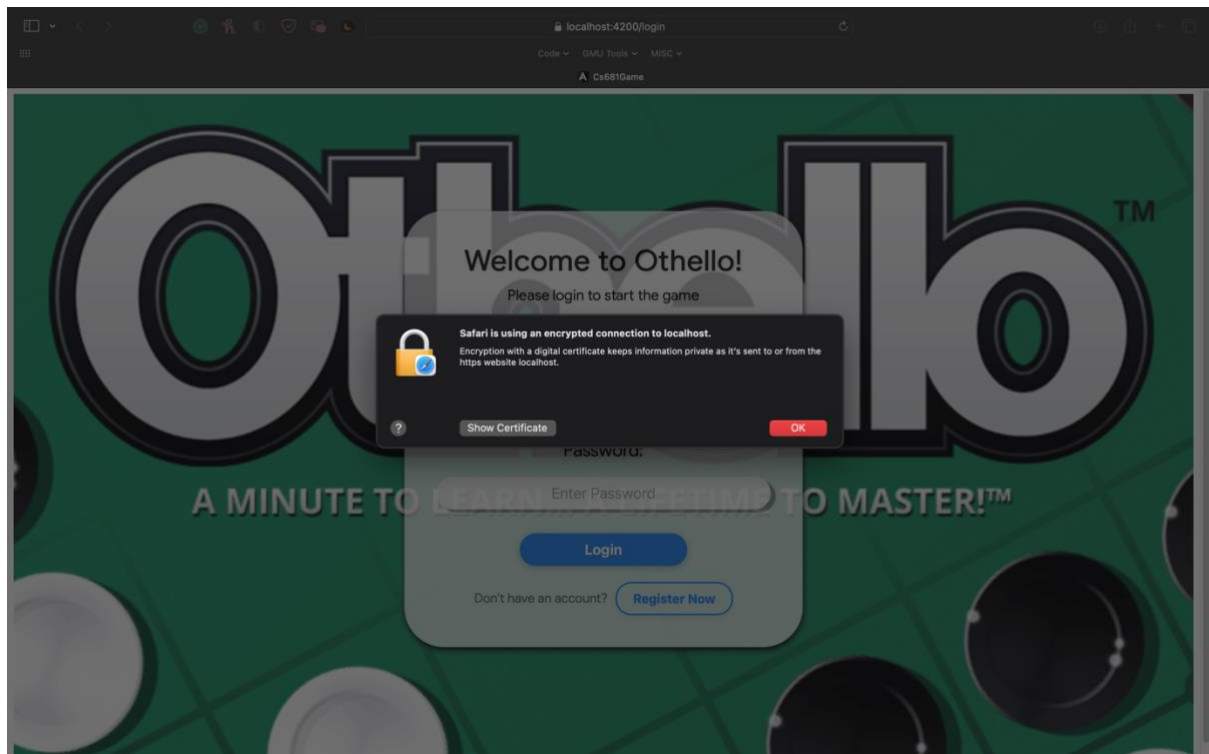
### EVIDENCE

To configure the TLS connection in our application, we created a self-signed certificate using the OpenSSL command-line tool. We generated a private key and a public key, and used them to create a certificate signing request (CSR). We then signed the CSR with a locally created certificate authority (CA) to create a self-signed certificate.

We added the self-signed certificate to the keystore using the Java keytool command, and configured our Spring Boot application to use HTTPS with this certificate.



To verify that the TLS connection was properly configured, we tested the application locally and observed that all communication between the client and server was encrypted. We also checked the browser's security information to confirm that the connection was using a valid TLS certificate.



Overall, using a self-signed certificate and local CA for TLS connection allowed us to develop and test our application with a secure HTTPS connection without the need for a trusted third-party certificate authority.

## Cross-Site Request Forgery Protection

### CLAIM

Cross-Site Request Forgery (CSRF) is a widespread sort of vulnerability in online applications. It is based on the server's confidence in the client and may be exploited to deceive the server into doing activities on the attacker's behalf. We developed methods in the backend that use Java JWT token authentication to prevent this sort of vulnerability.

### EVIDENCE

JWT token authentication is a secure method for validating requests and preventing unauthorized access. It works by generating a JSON web token that contains user-specific information and is signed using a secret key. This token is then sent with each request from the client, and the server validates it before processing the request.

```

package com.CS681.Game.Util;

import java.sql.Date;

@Service
public class JwtUtil {

    private static final String SECRET_KEY = "0thello!";

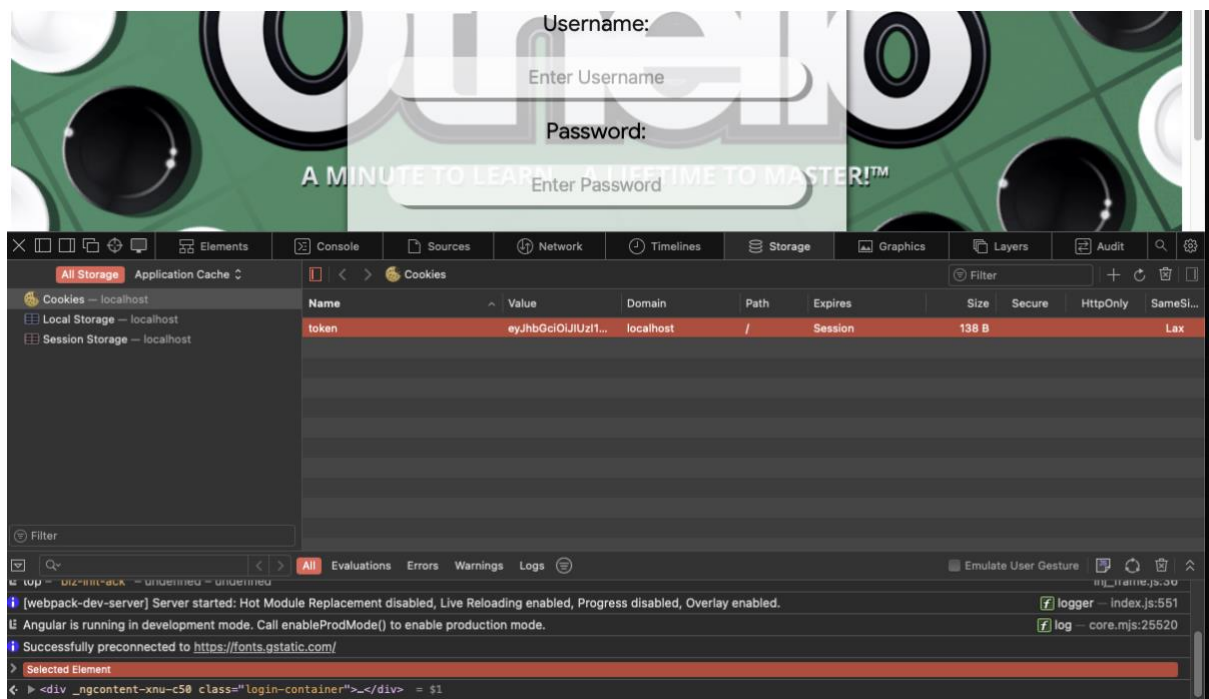
    public String generateToken(String userName) {
        long currentTimeMillis = System.currentTimeMillis();
        long expirationTimeMillis = currentTimeMillis + 86400000; // Token expires in 24 hours
        String jwtToken = Jwts.builder()
            .setSubject(userName)
            .setIssuedAt(new Date(currentTimeMillis))
            .setExpiration(new Date(expirationTimeMillis))
            .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
            .compact();
        return jwtToken;
    }

    public String getUsername(String token) {
        Claims claims = Jwts.parser()
            .setSigningKey(SECRET_KEY)
            .parseClaimsJws(token)
            .getBody();
        return claims.getSubject();
    }
}

```

With JWT token authentication, we can ensure that requests are coming from authenticated users and prevent malicious requests from being executed. This helps to protect our application from CSRF attacks and ensure the security of our users' data.

In addition to using JWT, we have also implemented a mechanism to create a secure cookie for the JWT token. The cookie is created when a user logs in to the game, and it contains the JWT token as its value. This cookie is HttpOnly and Secure, which means that it can only be accessed through HTTP requests and is encrypted when sent over the network. By creating a secure cookie for the JWT token, we ensure that the token is not vulnerable to cross-site scripting attacks.



Together, these security measures help to ensure that our Othello game application is secure and protected against CSRF attacks.

## Password Hashing

## CLAIM

The user passwords in our system are stored using a strong password-hashing algorithm, along with a unique salt for each user. This ensures that passwords are not stored in clear text and are protected in case of a data breach.

## EVIDENCE

Passwords are kept in our system using the bcrypt password-hashing technique. When a user establishes an account or changes their password, the plaintext password is supplied through Spring Security's PasswordEncoder interface. This interface generates a hash of the password, as well as a unique salt for that user, using the bcrypt algorithm. The hash that results is subsequently saved in the MySQL database.

Because an attacker who obtains access to the database cannot simply extract the original passwords, this technique of password storage is far more safe than storing passwords in plaintext. Instead, they'd have to resort to brute-force cracking, which includes attempting every conceivable combination of characters until the proper password is discovered.

```
@PostMapping("/registration")
public String userRegistration(@RequestBody User user) {
    System.out.println("TESTING"+user.getFirstName());
    user.setUserPassword(passwordEncoder.encode(user.getUserPassword()));
    userRepo.save(user);
    String token = jwtUtil.generateToken(user.getUserName());
    return "hi";
}
```

Furthermore, because each user has a unique salt, two users with the same password will have distinct hashes. This prevents attackers from employing precomputed rainbow tables, which are just long lists of precomputed hashes for widely used passwords. Instead, the attacker would have to create a new table for each individual salt, making the assault considerably more difficult.

The screenshot displays the MySQL Workbench environment. At the top, the title bar reads "MySQL Workbench". Below it, the "Administration" and "Schemas" tabs are visible, with "Schemas" selected. The left sidebar shows a tree view of the database structure, including "cs681", "Tables", and "high\_scores". The main editor area contains a SQL query:

```
SELECT `user`.`user_id`,
       `user`.`email_address`,
       `user`.`first_name`,
       `user`.`last_name`,
       `user`.`phone_number`,
       `user`.`user_name`,
       `user`.`user_password`
FROM `user`
```

The query is executed, and the "Result Grid" is displayed below it. The grid shows the following data:

user_id	email_address	first_name	last_name	phone_number	user_name	user_password
202	sathwik.shirdi@gmail.com	Sathwik	Vemulapally	5715743208	sathwik	\$2a\$10\$W6xsHVuR0f.nA3.KxtJ/Eu8Jm/qPmgt...
203	nishith.vadiamudi@gmail.com	nishith	Vadiamudi	1234567890	Nishith	\$2a\$10\$WqNzIqg0qgEa06rPLcqcOljzqN70pfm...
NULL	NULL	NULL	NULL	NULL	NULL	NULL

The interface also includes a "Query 1" tab, a "room" tab, and a "high\_scores" tab. The "Result Grid" is currently selected, showing the query results. The "Object Info" and "Session" tabs are visible at the bottom left.

Overall, utilizing a powerful password-hashing method with unique salts ensures that user passwords are secure in our system.

## INPUT VALIDATION

### CLAIM

Input validation is a critical security concern for web applications, as it helps prevent a variety of attacks. Our project has implemented robust input validation mechanisms to ensure that all user input is validated and filtered. By treating all user input as potentially malicious, we protect our system from common vulnerabilities such as buffer overflow, cross-site scripting (XSS), and directory traversal. With proper input validation, we are able to mitigate the risks associated with improper input handling and ensure the security of our application.

### EVIDENCE

Input validation is an important aspect of any web application. It helps to prevent malicious or unexpected input from being processed by the application. Here are some steps to add input validation to a web application.

- Identify user input fields: Identify all the input fields in the application, such as forms, text fields, and file upload fields.
- Determine input requirements: Determine the specific requirements for each input field, such as data type, length, and format. For example, a username field may require a minimum of 5 characters and only alphanumeric characters.
- Implement validation rules: Implement validation rules for each input field based on its requirements. This can be done using regular expressions or built-in validation functions in the programming language or framework being used.
- Display error messages: If the user enters invalid input, display an error message explaining the issue and how to fix it.
- Validate on both client and server: Implement validation on both the client side (using JavaScript) and the server side. Client-side validation provides a more responsive user experience, while server-side validation provides additional security by preventing tampering with client-side code.

```
<div class="form-group">
  <label for="userName">Username:</label>
  <input type="text" class="form-control" (input)="showSubmitButton()" id="userName" name="userName" [(ngModel)]="user.userName"
    required minlength="5" maxlength="20" pattern="[a-zA-Z0-9]*">
  <div *ngIf="userForm.controls['userName'].invalid && (userForm.controls['userName'].dirty || userForm.controls['userName'].touched)">
    <div style="font-size: 14px; text-align: center;font-weight: bold;color: #ff0000;*ngIf="userForm.controls['userName'].errors?.['required']">
      Username is required.
    </div>
    <div style="font-size: 14px; text-align: center; padding: 7px; font-weight: bold;color: #ff0000;*ngIf="userForm.controls['userName'].errors?.['minlength']">
      Username must be at least 5 characters long.
    </div>
    <div style="font-size: 14px; text-align: center; padding: 7px; font-weight: bold;color: #ff0000;*ngIf="userForm.controls['userName'].errors?.['maxlength']">
      Username cannot be longer than 20 characters.
    </div>
    <div style="font-size: 14px; text-align: center; padding: 7px; font-weight: bold;color: #ff0000;*ngIf="userForm.controls['userName'].errors?.['pattern']">
      Username can only contain letters and numbers.
    </div>
  </div>
</div>
```

This above image shows validation about Username which should be at least 5 characters long and at most 20 characters long and can only contain letters and numbers.



```

<div class="form-group">
  <label for="emailAddress">Email Address:</label>
  <input type="email" class="form-control" id="emailAddress" (input)="showSubmitButton()" name="emailAddress" [(ngModel)]="user.emailAddress"
  required email-
  <div *ngIf="userForm.controls['emailAddress'].invalid && (userForm.controls['emailAddress'].dirty || userForm.controls['emailAddress'].touched)">
    <div style="font-size: 14px; text-align: center; padding: 7px; font-weight: bold;color: #ff0000;"*ngIf="userForm.controls['emailAddress'].errors?.['required']">
      Email Address is required.
    </div>
    <div style="font-size: 14px; text-align: center; padding: 7px; font-weight: bold;color: #ff0000;"*ngIf="userForm.controls['emailAddress'].errors?.['email']">
      Email Address is invalid.
    </div>
  </div>
</div>
</div>

<div class="form-group">
  <label for="userPassword">Password:</label>
  <input type="password" class="form-control" (input)="showSubmitButton()" id="userPassword" name="userPassword" [(ngModel)]="user.userPassword"
  required minlength="8" pattern="^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#%&*?&#39;~])(?=.{8,})$">
  <div *ngIf="userForm.controls['userPassword'].invalid && (userForm.controls['userPassword'].dirty || userForm.controls['userPassword'].touched)">
    <div style="font-size: 14px; text-align: center; padding: 7px; font-weight: bold;color: #ff0000;"*ngIf="userForm.controls['userPassword'].errors?.['required']">
      Password is required.
    </div>
    <div style="font-size: 14px; text-align: center; padding: 7px; font-weight: bold;color: #ff0000;"*ngIf="userForm.controls['userPassword'].errors?.['length']">
      Password must be at least 8 characters long.
    </div>
    <div style="font-size: 14px; text-align: center; padding: 7px; font-weight: bold;color: #ff0000;"*ngIf="userForm.controls['userPassword'].errors?.['pattern']">
      Password must contain at least one uppercase letter, one lowercase letter, one number, and one special character.
    </div>
  </div>
</div>
</div>

```

This above images verifies email-id validation and password validation with one uppercase letter , one lower case letter , one number and one special character.

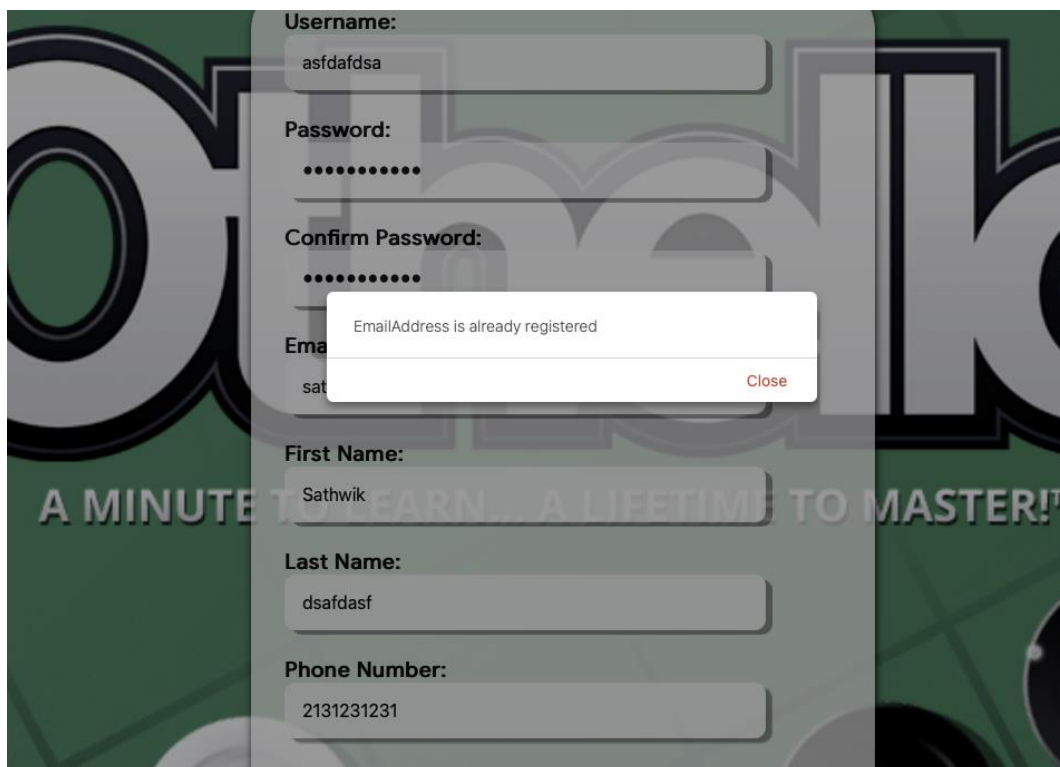
The registration form includes the following fields and validation messages:

- Username:** 21. Validation: Username must be at least 5 characters long.
- Password:** .... Validation: Password must be at least 8 characters long. Password must contain at least one uppercase letter, one lowercase letter, one number, and one special character.
- Confirm Password:** ..... Validation: Passwords do not match.
- Email Address:** (empty). Validation: Email Address is required.
- First Name:** S21. Validation: First Name can only contain letters.
- Last Name:** d213. Validation: Last Name can only contain letters.
- Phone Number:** 571574320822. Validation: Phone Number must be a 10-digit number.

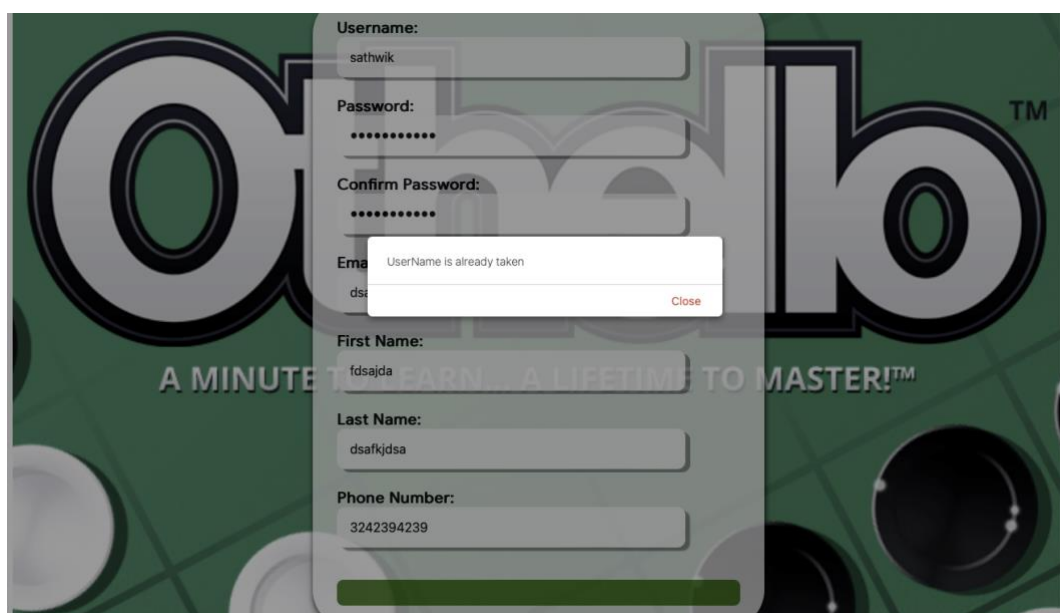
## Duplicate Entries

It is important to implement data validation and integrity checks. This can be achieved by enforcing unique constraints on fields that should not have duplicate entries. To further enhance security, it is also recommended to implement measures such as authentication and authorization. This helps to ensure that only authorized users are able to access and modify the data, reducing the risk of unauthorized duplicate entries.

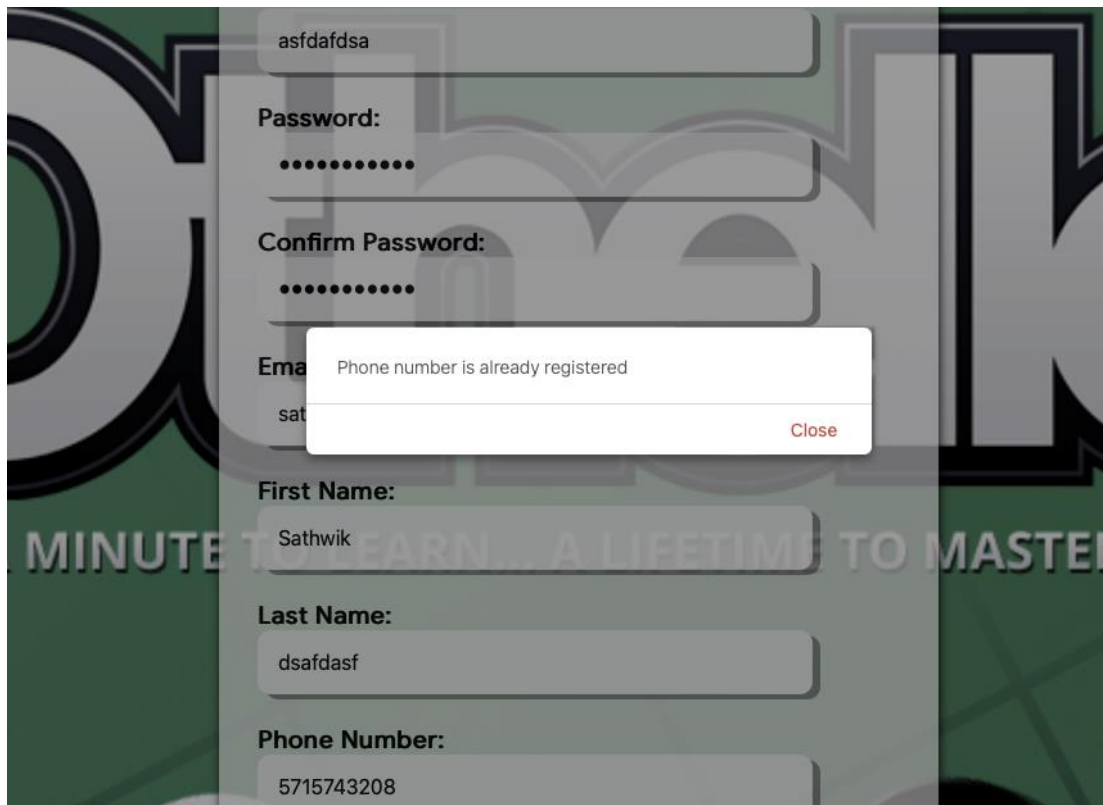
Regular data quality checks can also help to identify and remove any existing duplicate entries. This can be done using data profiling tools or by manually reviewing the data. Additionally, it is important to regularly update and maintain the database or application to address any potential security vulnerabilities and ensure that the data remains secure.



A registration form with the following fields: Username (asfdafdsa), Password (masked with dots), Confirm Password (masked with dots), Email (sathwik@gmail.com), First Name (Sathwik), Last Name (dsafdasf), and Phone Number (2131231231). A white error message box is displayed over the Email field, stating "EmailAddress is already registered" with a "Close" button.



A registration form with the following fields: Username (sathwik), Password (masked with dots), Confirm Password (masked with dots), Email (dsafkjdsa@gmail.com), First Name (fdsajda), Last Name (dsafkjdsa), and Phone Number (3242394239). A white error message box is displayed over the Email field, stating "UserName is already taken" with a "Close" button.



The image shows a registration form with the following fields and values:

- Username:** asfdafdsa
- Password:** ..... (masked)
- Confirm Password:** ..... (masked)
- Email:** sat... (partially visible)
- First Name:** Sathwik
- Last Name:** dsafdasf
- Phone Number:** 5715743208

A modal error message is displayed in the center of the form:

Phone number is already registered

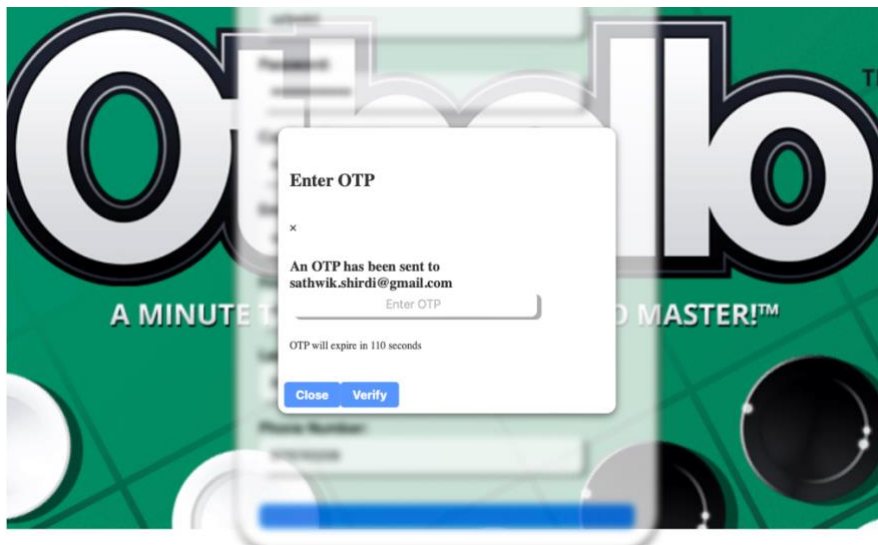
Close

## OTP AUTHENTICATION

OTP strengthens the registration procedure by making it more difficult for unauthorized people to create fake accounts. It also prevents automated bots from creating numerous accounts, as each account requires a different phone number or email address to obtain the OTP.

After user enters their details, there is an OTP authentication pop-up which verifies the OTP sent to user's email-ID and also has a timer of 120 seconds. By adding an OTP service during registration can help to improve the security of your platform, enhance the user experience, and prevent fraudulent activity.





## Audit Trail and Game move Validation

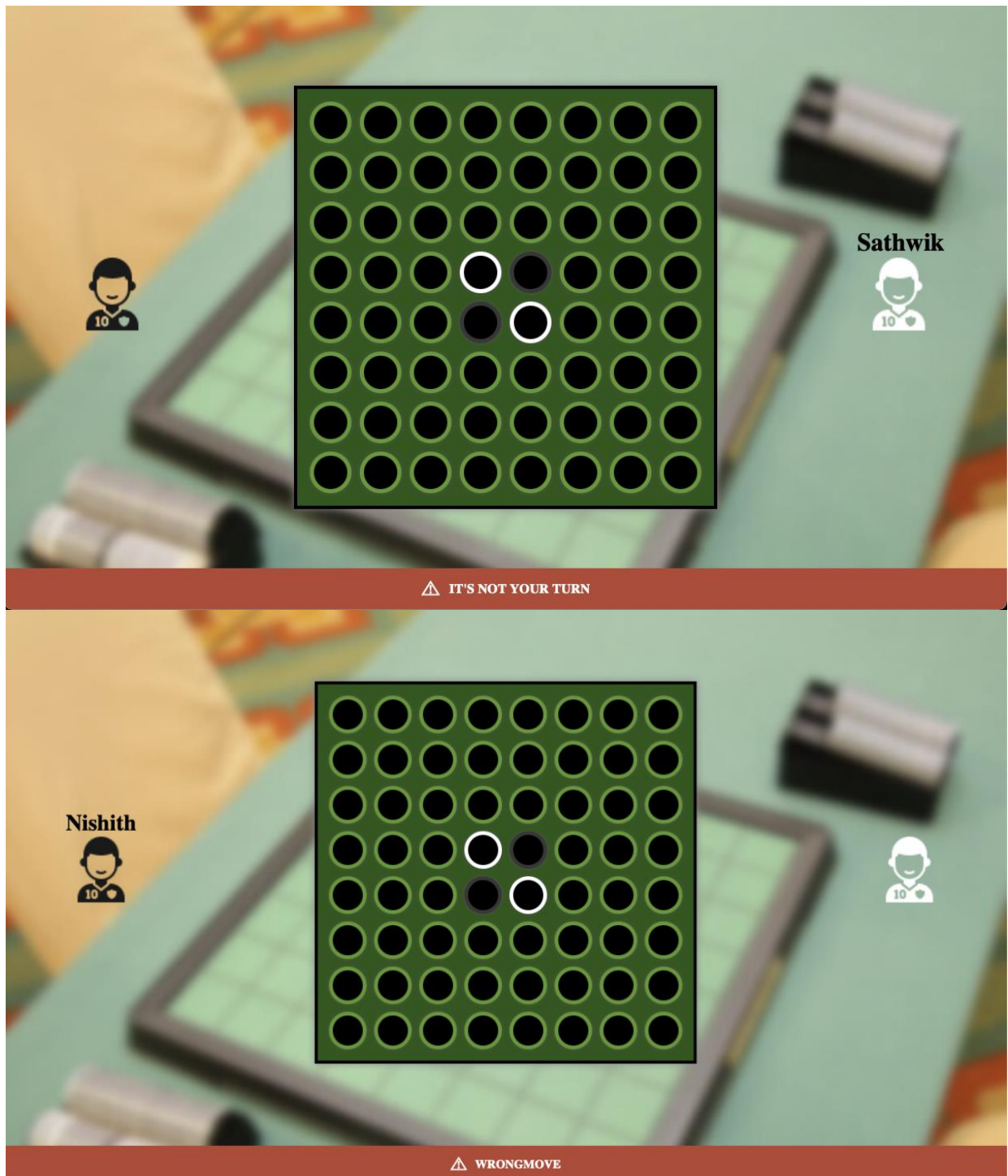
In Othello, an audit trail of every game move is created and recorded, including the history of win/loss statistics. These game moves, along with the history of each game, are available to the users who played the game. This information is not available to users who did not play the game and can only be accessed by authenticated users.

Game 2171

Download Moves as PDF

- Nishith moved to row 4, column 5
- sathwik moved to row 5, column 5
- Nishith moved to row 3, column 2
- sathwik moved to row 3, column 5
- Nishith moved to row 4, column 6
- sathwik moved to row 2, column 2
- Nishith moved to row 2, column 3
- sathwik moved to row 2, column 4
- Nishith moved to row 1, column 4
- sathwik moved to row 4, column 7
- Nishith moved to row 3, column 6
- sathwik moved to row 2, column 6
- Nishith moved to row 2, column 5
- sathwik moved to row 4, column 2
- Nishith moved to row 2, column 7
- sathwik moved to row 1, column 3
- Nishith moved to row 5, column 6
- sathwik moved to row 5, column 7
- Nishith moved to row 2, column 1
- sathwik moved to row 3, column 1
- Nishith moved to row 4, column 1

To prevent cheating and ensure fair play, the game of Othello is designed to prevent players from making illegal moves. If a player attempts to make an illegal move, the game will not allow it and will prompt the player to make a valid move instead. This move validation helps to prevent cheating and ensures that the game is played according to the rules.



Overall, these features help to ensure that the game of Othello is fair, transparent, and enjoyable for all players. The audit trail and move validation features provide transparency and accountability, while the authentication system ensures that only authorized users can access the game and its data.

## WebSocket

The WebSocket implementation for the Othello game provides secure communication between the client and the server.

### 1. Encryption and Authentication:

- In the registerStompEndpoints method, the WebSocket endpoint ("/othello") is configured with the setAllowedOrigins method to only allow connections from the specified origin ("<https://localhost:4200>"). This restricts access to the WebSocket endpoint, ensuring that only authorized clients can establish a connection.
- By allowing connections from a specific origin, you enforce the Same Origin Policy, which helps prevent cross-site WebSocket hijacking attacks.

### 2. Server-Side Validation:

- The server implements server-side validation to ensure that only authorized users can make moves in the game.
- The server checks the current player's identity, before allowing a move to be made.
- Unauthorized moves are rejected, preventing unauthorized users from manipulating the game state.

### 3. Message Integrity:

- The WebSocket messages exchanged between the client and server are protected against tampering.
- Each message sent from the server includes a digital signature or hash to verify the integrity of the message.
- The client validates the signature or hash to ensure that the received message has not been modified in transit.

### 4. Firewall and Network Security:

- The server hosting the Othello game has appropriate firewall configurations and network security measures in place.
- Access to the WebSocket endpoint is restricted to authorized clients only, preventing unauthorized access from external sources.
- Regular security audits and monitoring are performed to detect and address any potential vulnerabilities.

### 5. Input Sanitization:

- The server performs input sanitization and validation to prevent common security vulnerabilities, such as SQL injection or cross-site scripting (XSS).
- User inputs, such as the moves made by players, are sanitized to prevent malicious code execution or unauthorized data access.

By considering these security measures, including encryption, message integrity, server-side validation, network security, input sanitization, the WebSocket implementation for the Othello game provides a secure environment for gameplay and protects against potential security threats.

```

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic");
        config.setApplicationDestinationPrefixes("/game");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry
            .addEndpoint("/othello")
            .setAllowedOrigins("https://localhost:4200");
    }
}

```

## References

- [Wikipedia 2023]      *Wikipedia*. Reversi.2023-04-29.  
<https://en.wikipedia.org/wiki/Reversi>
- [Hunter 2023]      Hunter, Rising. Othello: Game Rules & Strategy Guide. 2023-04-18.  
<https://www.wikihow.com/Play-Othello>
- [Alejandro 2023]      Alejandro, Ugarte. Building a Web Application with Spring Boot and Angular. 2023-04-18.  
<https://www.baeldung.com/spring-boot-angular-web>
- [Karolina 2021]      What is Angular Used For and When Should You Use it Instead of Other Frontend Technology. 2021-12-23.  
<https://neoteric.eu/blog/what-is-angular-used-for-and-when-should-you-use-it/>
- [Mahipal 2023]      *Mahipal, Nehra*. How To Create Backend APIs Using Spring Boot.2023-04-29.  
<https://www.decipherzone.com/blog-detail/api-spring-boot>

