UE22CS352B - Object Oriented Analysis & Design

Mini Project Report

ChessMate:A Smart Multiplayer Chess Application

Submitted by:

Name : PES2UG22CS294 Maitri Maheshkumar Shekhda
PES2UG22CS303 Mansa Pandey

Semester:6 Section:E

Dr. Kamatchi Priya L

January - May 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem Statement:

Chess is a globally popular strategy game, and while many digital implementations exist, building a simplified, modular version provides a strong foundation for learning object-oriented design and software architecture. Most beginner-level chess applications either lack clean architecture or fail to demonstrate good software engineering practices.This project serves both as a functional multiplayer chess platform and as an educational blueprint for applying OOP principles, UML design, and best practices in Java and Spring Boot.
This project addresses the need for a basic, real-time multiplayer chess game that focuses on:

- Enabling two players to play against each other (Player vs Player).
- Demonstrating object-oriented design principles and patterns.
- Providing a clear separation of concerns using a layered architecture.
- Building a foundational backend system that can be extended in the future.

Key Features:

Our project is a comprehensive, secure, and scalable Online Multiplayer Chess Platform developed with an architectural mindset inspired by Object-Oriented Analysis and Design (OOAD). Below is an elaboration of the major features that showcase the technical depth and breadth of our implementation:

1. Robust Authentication System

- User Registration & Login: We implemented a secure and seamless authentication module where users can register, log in, and manage their credentials. All user inputs are validated with strict form validations to enhance security and user experience.
- Password Encryption: To ensure confidentiality, passwords are encrypted using industry-standard hashing algorithms (e.g., BCrypt), handled by the `EncryptionService.java`. This safeguards sensitive data and complies with security best practices.
- Password Reset & Validation: We have integrated a password reset module that includes email-based validation (for future extension). Incorrect username or password attempts are caught, logged, and managed efficiently to prevent brute force attacks.

2. Dynamic Chess Game Engine

- Fully Functional Chessboard: A responsive, interactive chess board supports classic black and white gameplay. Chess pieces are visually intuitive and follow traditional movement rules.
- Move Highlighting: The game dynamically highlights all valid moves for a selected piece using move validation logic. This enhances player experience, especially for beginners.
- Move Validation: Implemented in `MoveValidator.java`, this module checks the legality of each move according to chess rules, ensuring accurate and fair gameplay.
- Check and Checkmate Detection: The engine includes logic to detect check and

checkmate scenarios. Players are notified visually, and the game transitions smoothly between turns.

3. Modular Architecture & OOAD Approach

- All services, models, controllers, and utilities follow the Single Responsibility Principle, ensuring each class is maintainable, testable, and easy to debug.
- The design is extendable; new features like user chat, AI opponent integration, or tournament mode can be added with minimal impact to the core logic.

4. Interactive User Experience

- Intuitive UI: A clean frontend using JavaFX or web-based rendering allows intuitive drag-and-drop or click-based chess movements.
- Real-Time Game State: Real-time updates are managed efficiently, allowing turn-based interaction without latency or inconsistencies.

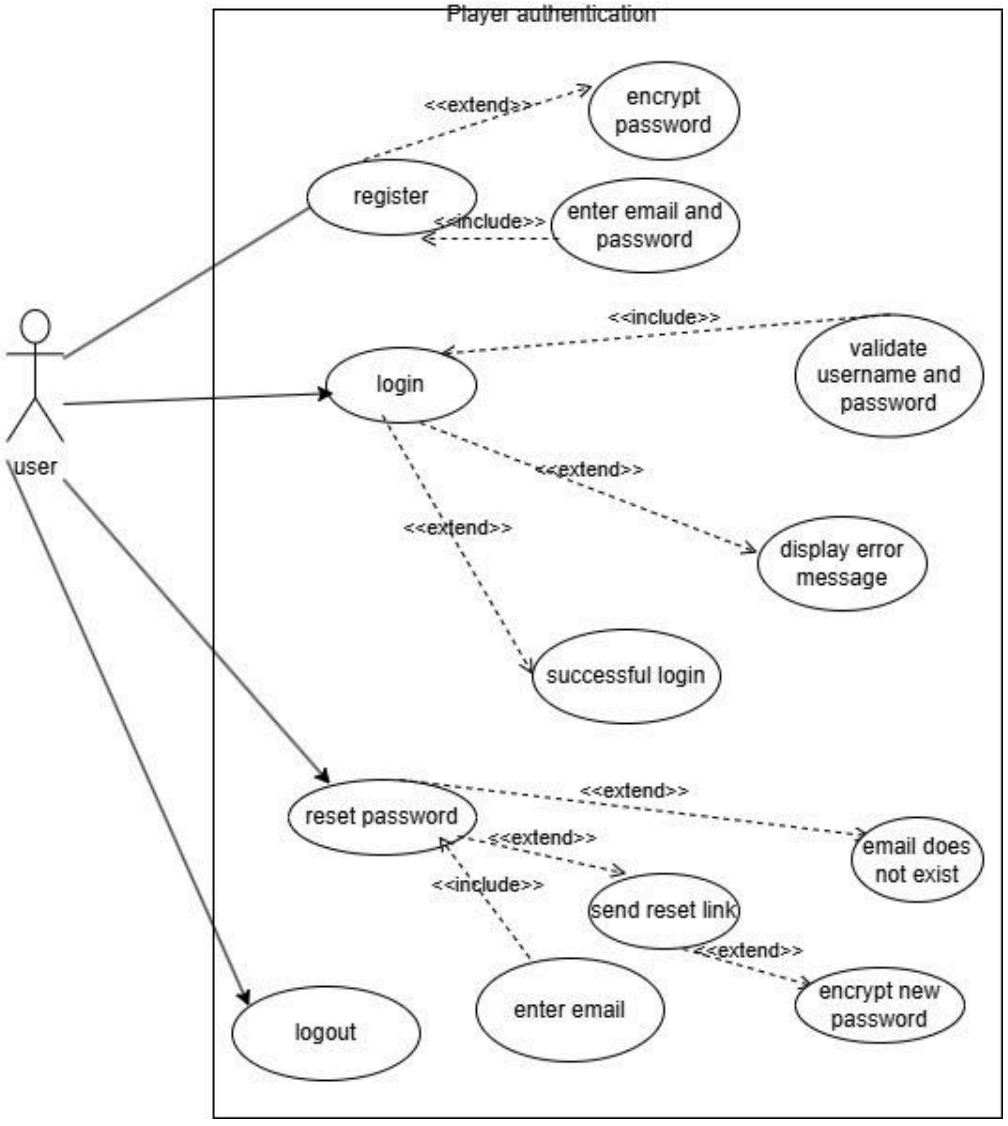
5. Scalability & Extensibility

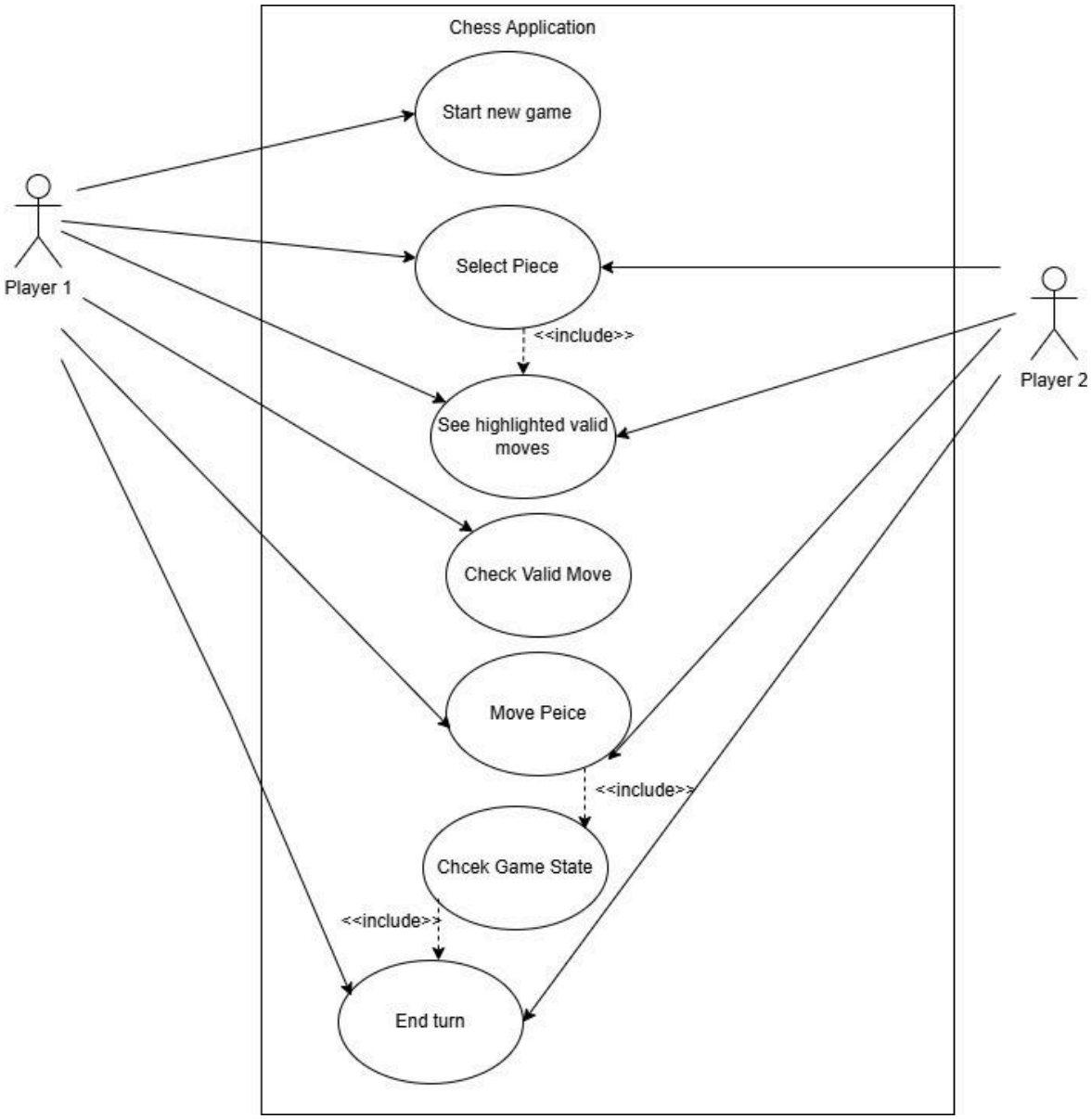
- The codebase is scalable by design. Through design patterns like Strategy and Singleton, our architecture supports future enhancements without touching stable code.
- Support for multiplayer, player profile statistics, and future AI integration is already scaffolded.g.

Models:

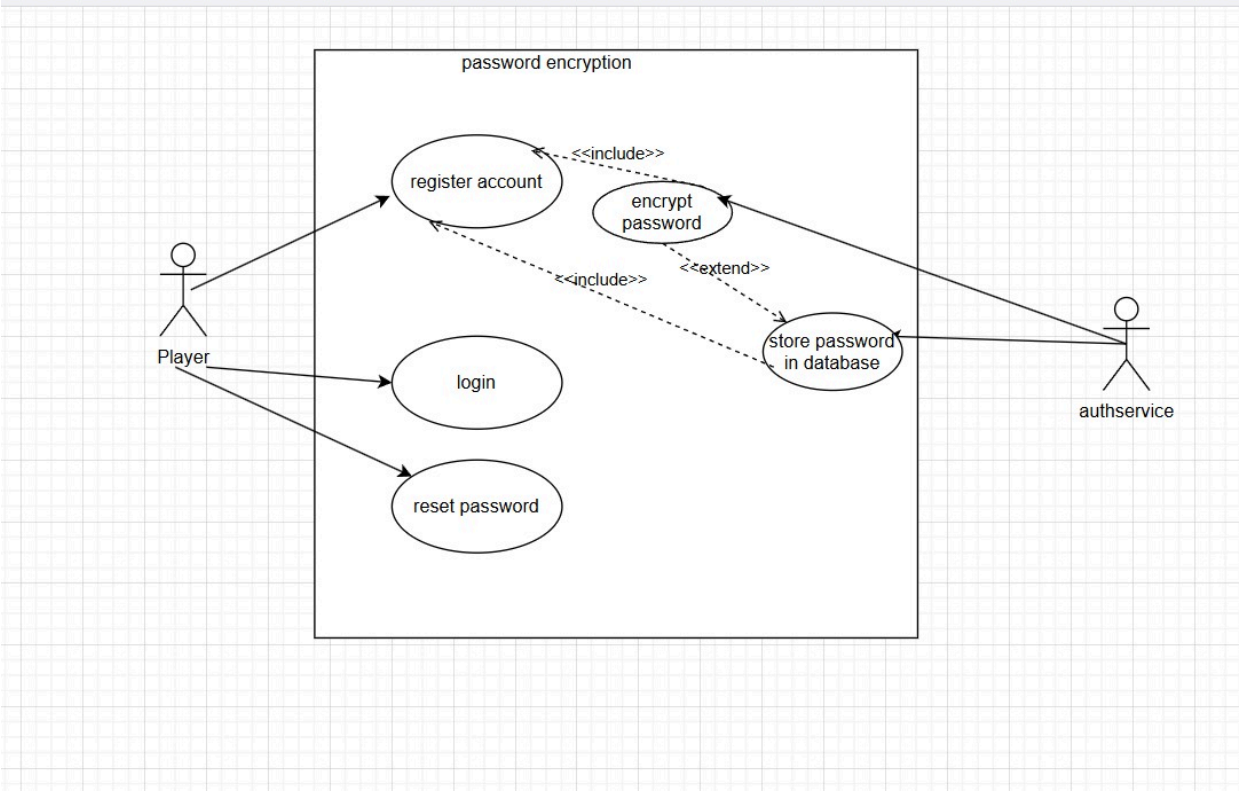
Use Case Diagram:

Major Use Case:

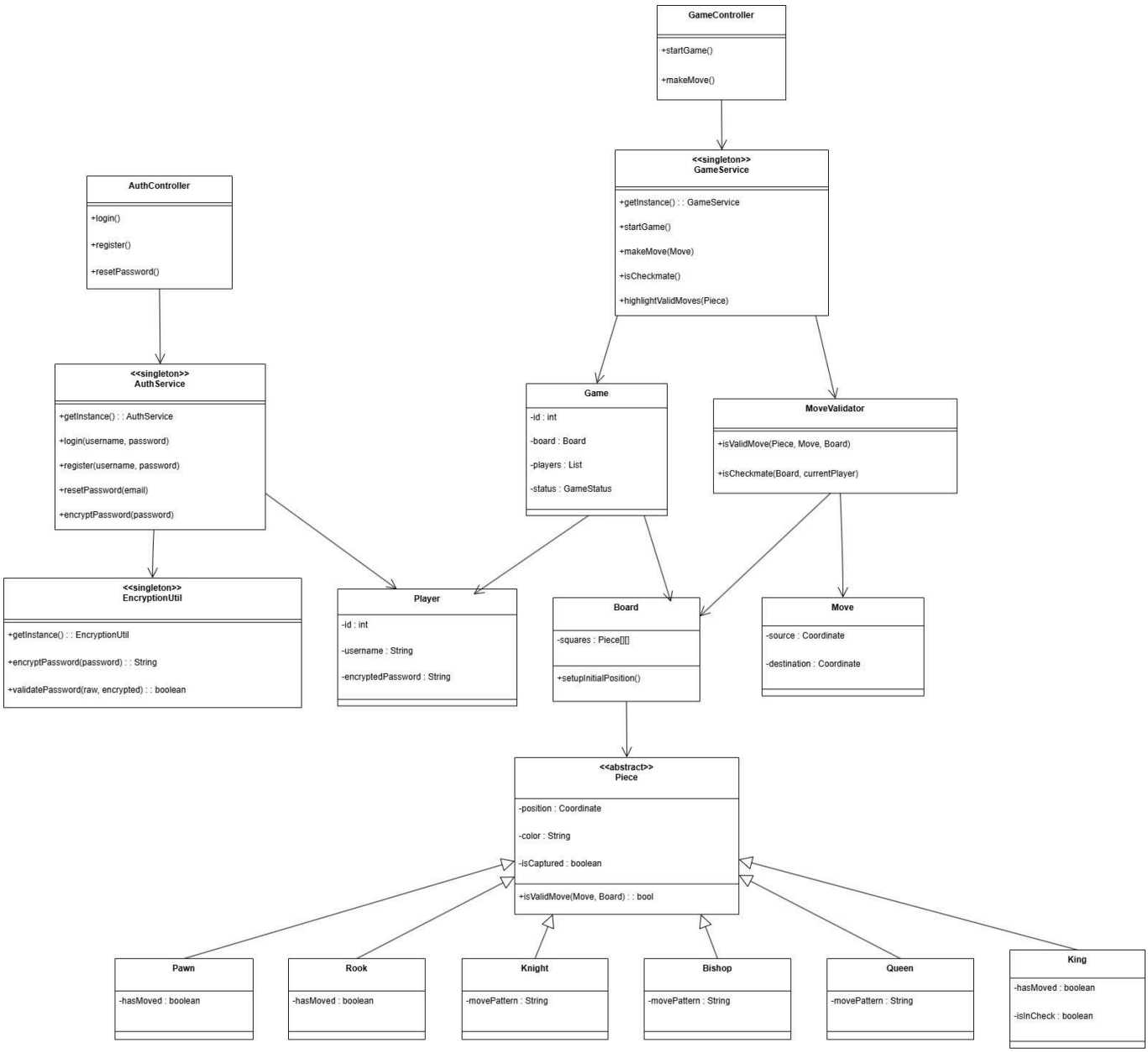



Chess Application
Start new game
Select Piece
<<include>>
See highlighted valid moves
Check Valid Move
Move Peice
<<include>>
Chcek Game State
<<include>>
End turn
Player 1
Player 2

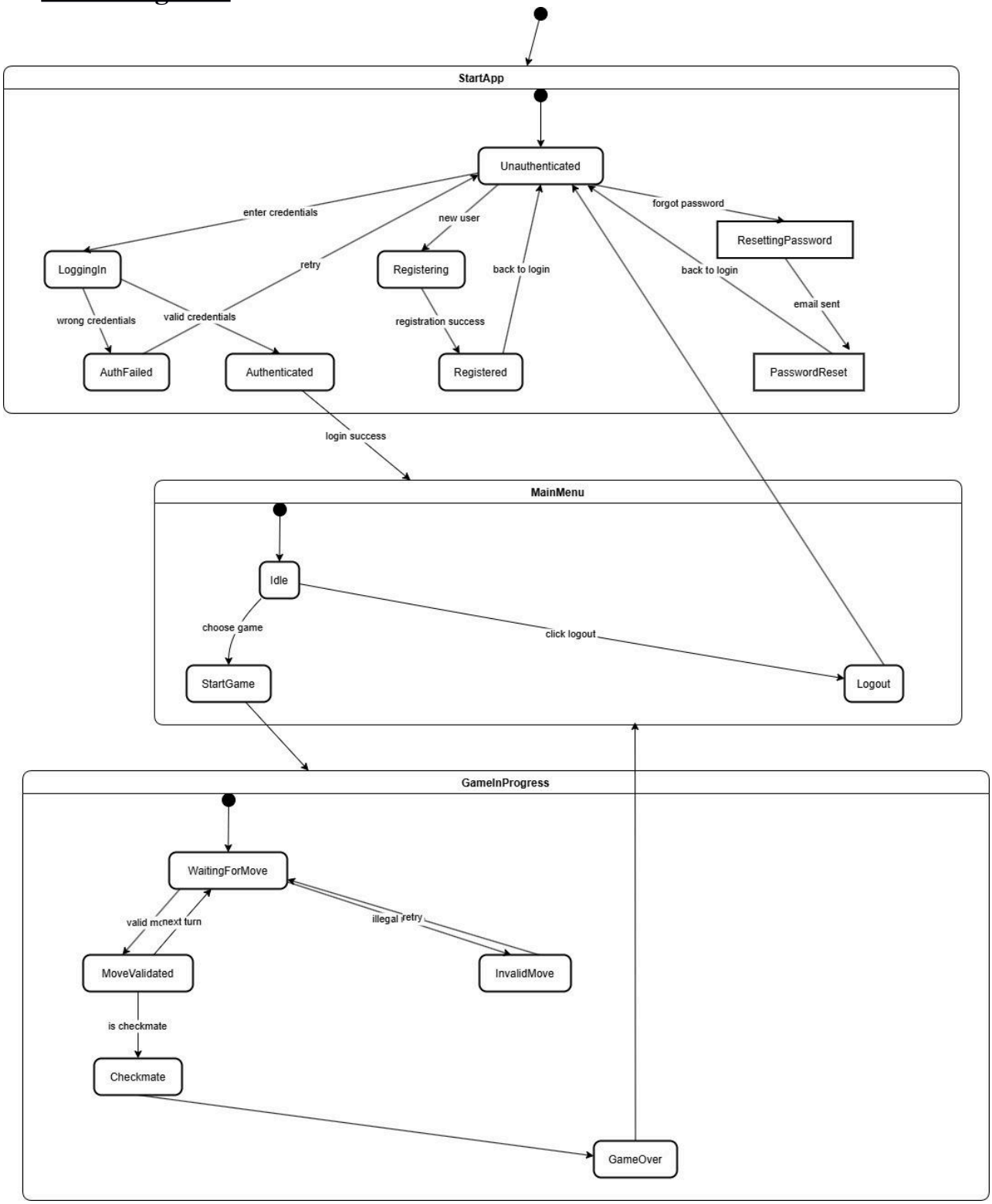
Minor Use Case:



Class Diagram:

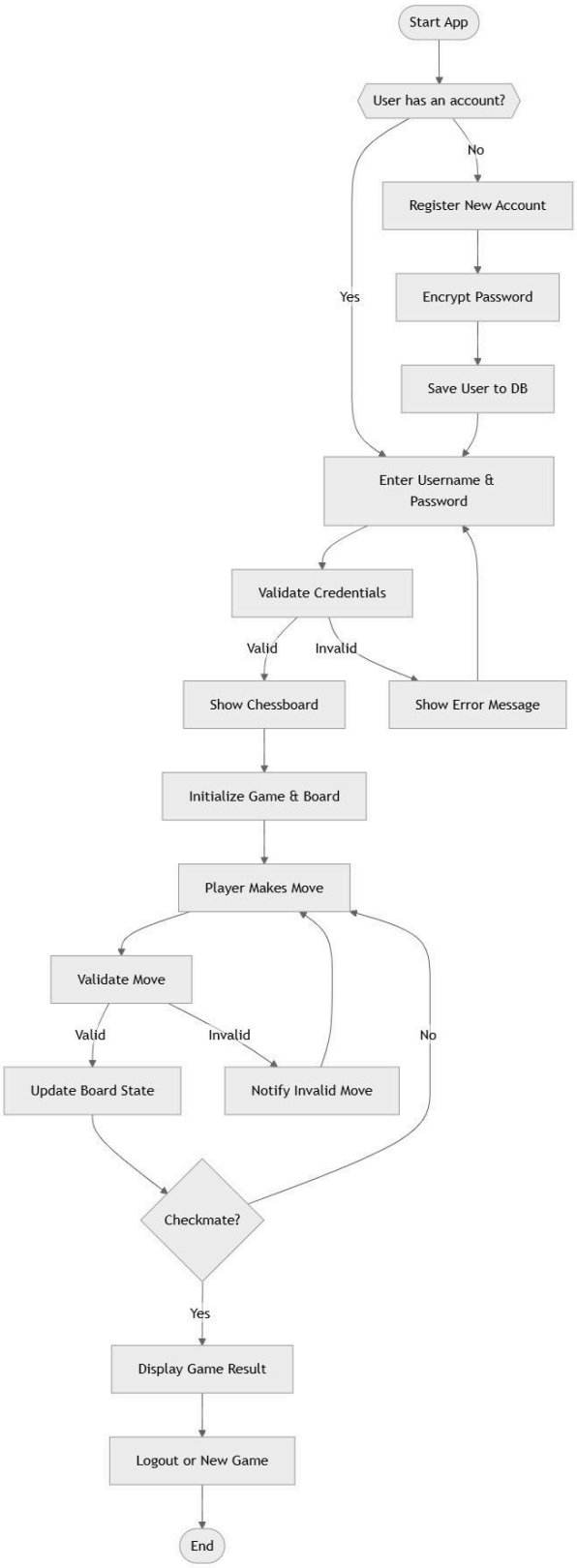


State Diagram:

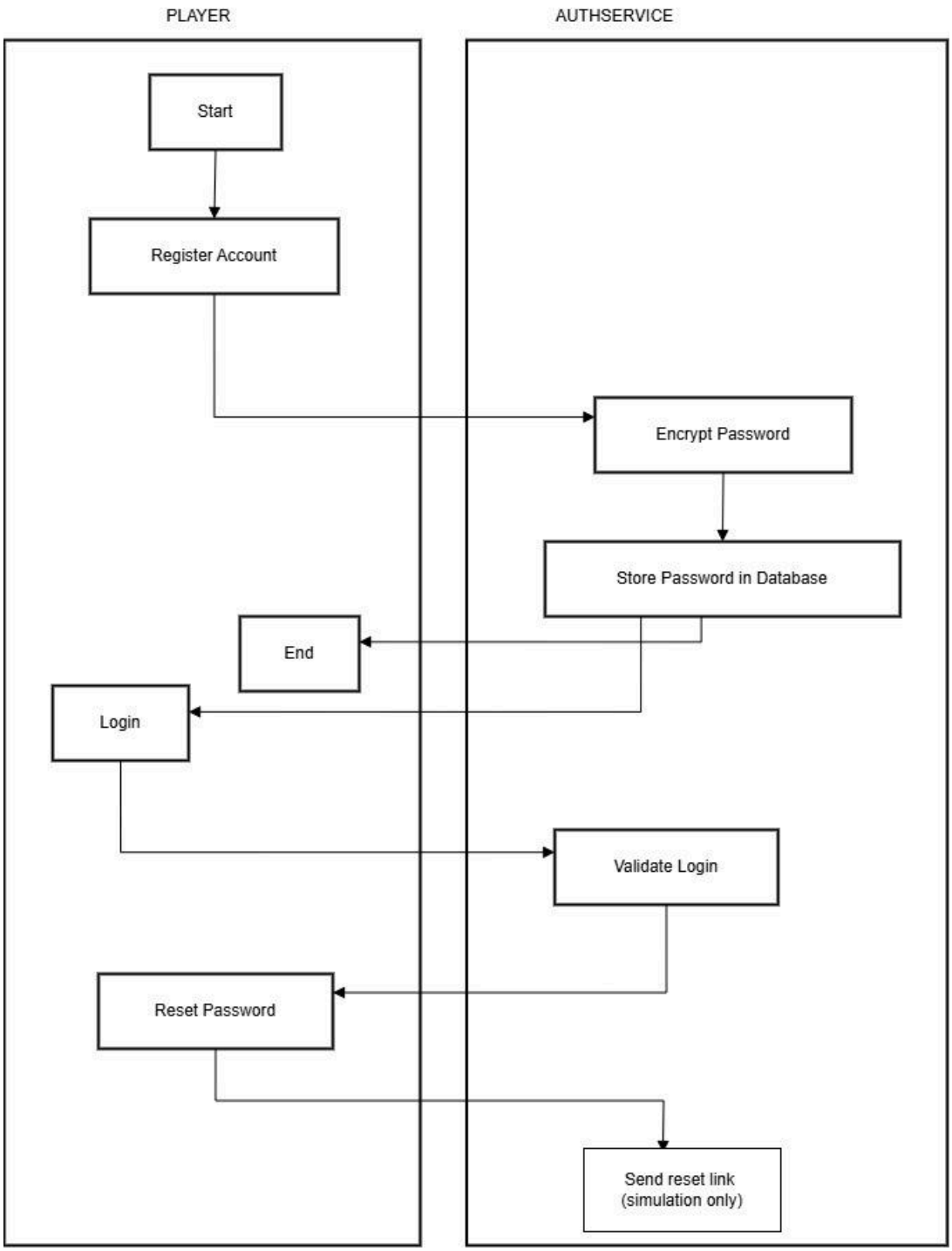


Activity Diagrams:

1. Major Usecase



2. Minor Use case



Architecture Patterns, Design Principles, and Design Patterns:

Architecture Patterns

Model – View – Controller Pattern (MVC)

- Model: This includes the core game logic and data structures — such as Game, Player, Board, and Piece. These classes define how chess operates, including move rules, game state, and player data.

- View: As there is no graphical interface or API layer, the view is simulated through terminal/console output or controlled test cases. This "view" component reflects the game state or move outcomes during development, testing, or demonstrations.we have implemented it using ui using javafx and swing Jpanel.

- Controller: This layer coordinates between the user inputs (e.g., move commands or method calls in a main driver class) and the model logic. It orchestrates how the Game progresses, who moves next, and what happens after each action.

This architecture supports separation of concerns:

- Model handles business logic,

- Controller handles input and flow control,

- View (in this case, console/text-based) reflects the state/output.

Design Principles

A. Single Responsibility Principle (SRP)

Definition: Each class in a system should have one, and only one, reason to change. That means it should have a single, well-defined responsibility.

Where It's Implemented:

1. **AuthService.java**
   - Handles authentication operations such as login, registration, and password reset.
   - By isolating all auth-related logic, changes in authentication (e.g., adding 2FA) won't affect unrelated components like game logic.
2. **GameService.java**
   - Manages chess game logic—player turns, board updates, and endgame conditions.
   - Keeps the game logic decoupled from UI or player details.

3. **PlayerService.java**
   - Manages player data: fetching profiles, updating stats.
   - Ensures separation from authentication and gameplay logic.
4. **MoveValidator.java**
   - Checks move validity based on the piece type and current board state.
   - Ensures validation is separated from actual movement execution.

It's Easier to maintain and scale, as each module has a focused responsibility.

B. Open/Closed Principle (OCP)

Definition: Software entities should be open for extension but closed for modification.

Where It's Implemented:

1. **Piece.java**
   - Abstract base class defining the interface for all chess pieces.
   - Method `isValidMove(Move move, Board board)` is overridden in subclasses.
2. **Subclasses (e.g., Pawn.java, Rook.java, Knight.java)**
   - Each implements unique movement logic via `isValidMove()`.

Extension Example:

- Add a new class like `Amazon.java` (Queen + Knight moves) without modifying existing pieces.

Benefit: New behaviors can be introduced with minimal risk to existing stable code.

Design Patterns

A. Strategy Pattern

Definition: Defines a family of algorithms, encapsulates each one, and makes them interchangeable.

Where It's Implemented:

- **Piece.java** acts as the strategy interface with the method `isValidMove()`.
- Subclasses like `Knight.java`, `Bishop.java`, `Queen.java` implement distinct movement logic.

Example:

- `Knight.java` implements L-shaped movement.
- `Rook.java` implements horizontal/vertical movement.

Use Case:

- System calls the specific piece's `isValidMove()` method without needing to know the internal logic.

Advantage: Behavior changes dynamically based on the piece, promoting cleaner, modular code.

B. Singleton Pattern

Definition: Ensures a class has only one instance and provides a global access point.

Where It's Implemented:

- **EncryptionService.java**: Handles encryption/decryption of passwords.
- **GameService/AuthService**: Using Spring Boot, these are singletons by default via `@Service` annotation.

Example:

- `EncryptionService` uses consistent hashing logic across all services (Auth, Reset, etc.).

Use Case:

- Promotes memory efficiency and consistency in encryption logic.

Advantage: Centralized behavior, low resource usage, and reliable global access.

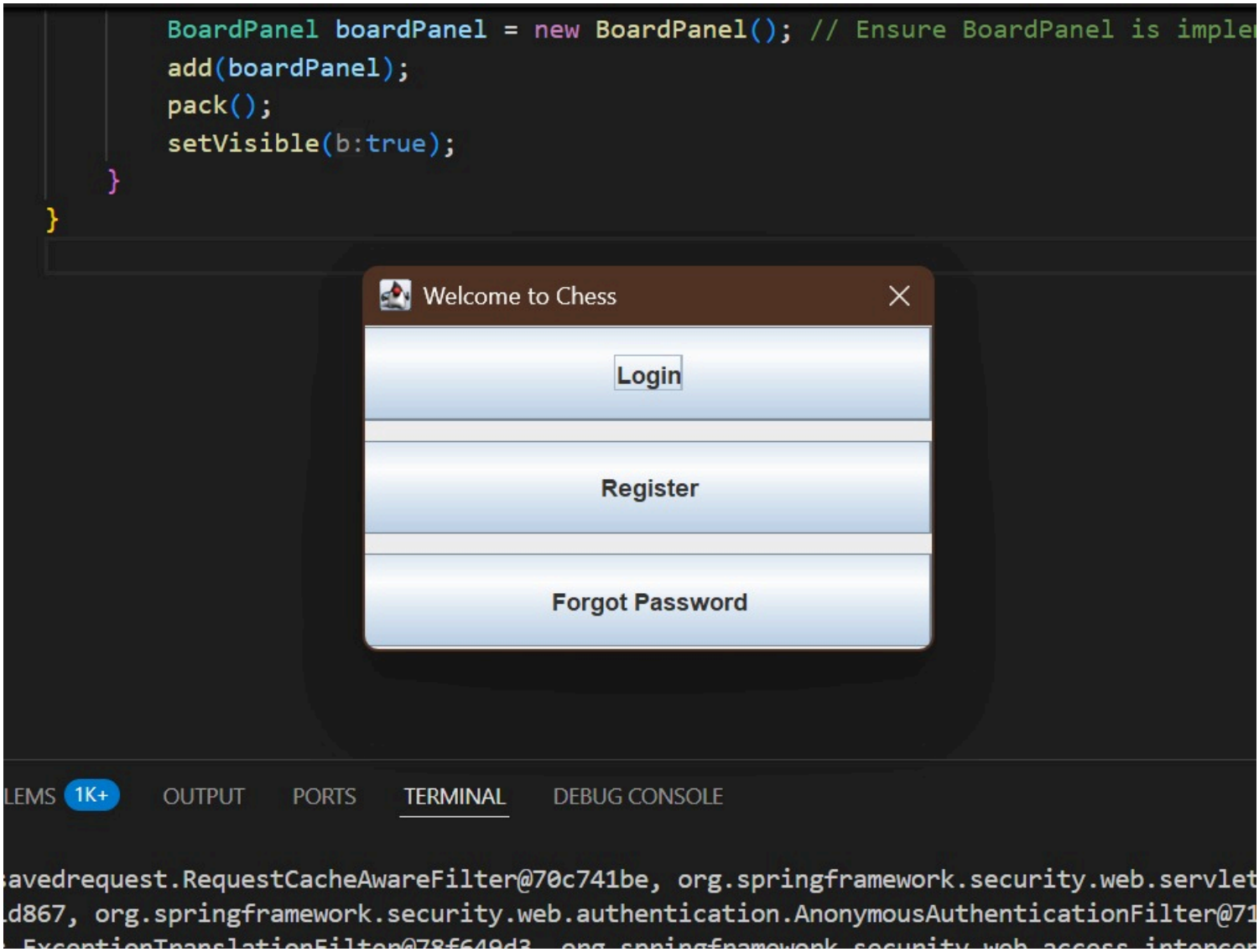
Github link to the Codebase:

https://github.com/Maitri-Shekhda/OOAD-Chess-Application

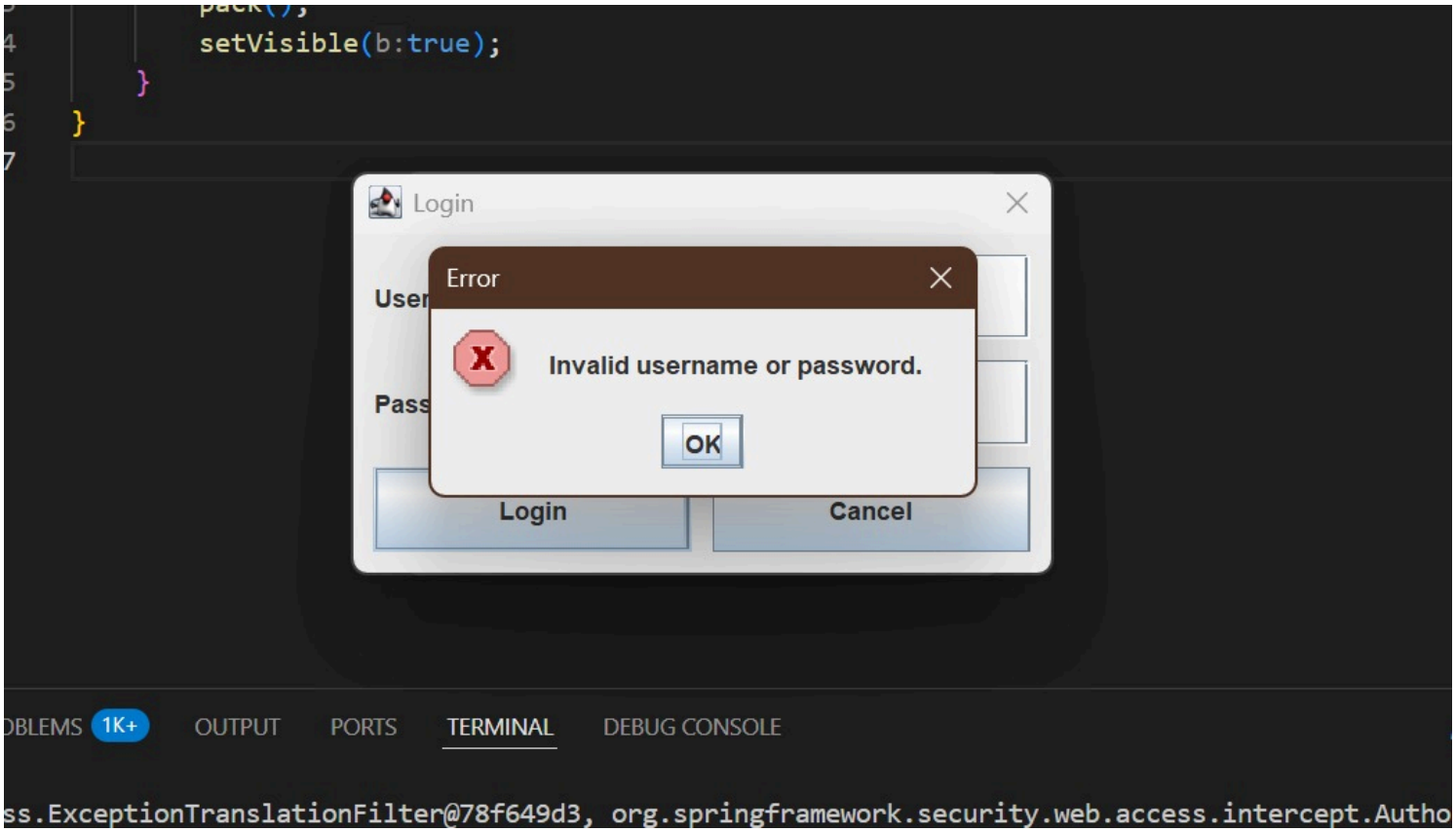
Screenshots

UI:

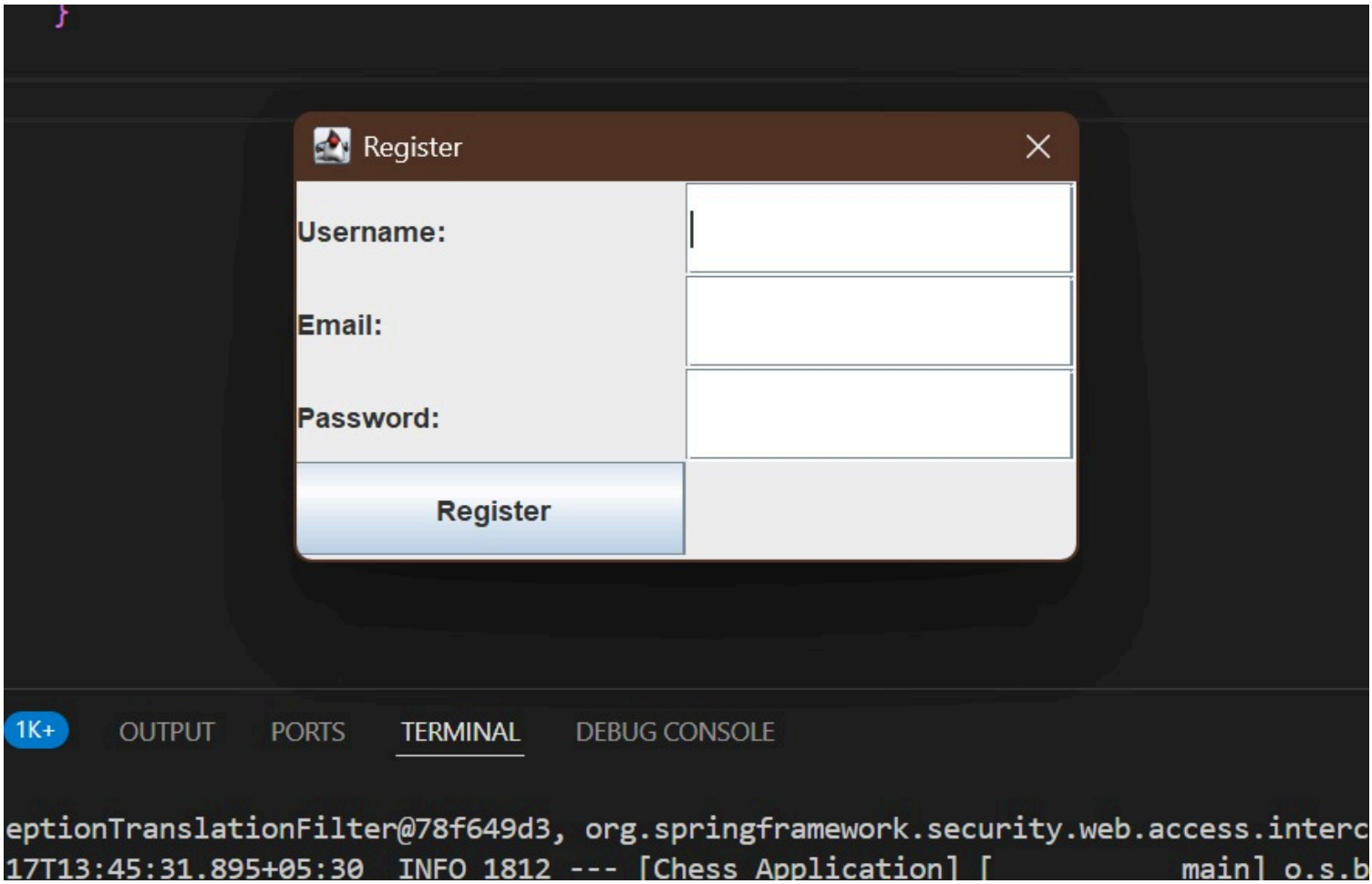
The following option appear after the code successfully run.



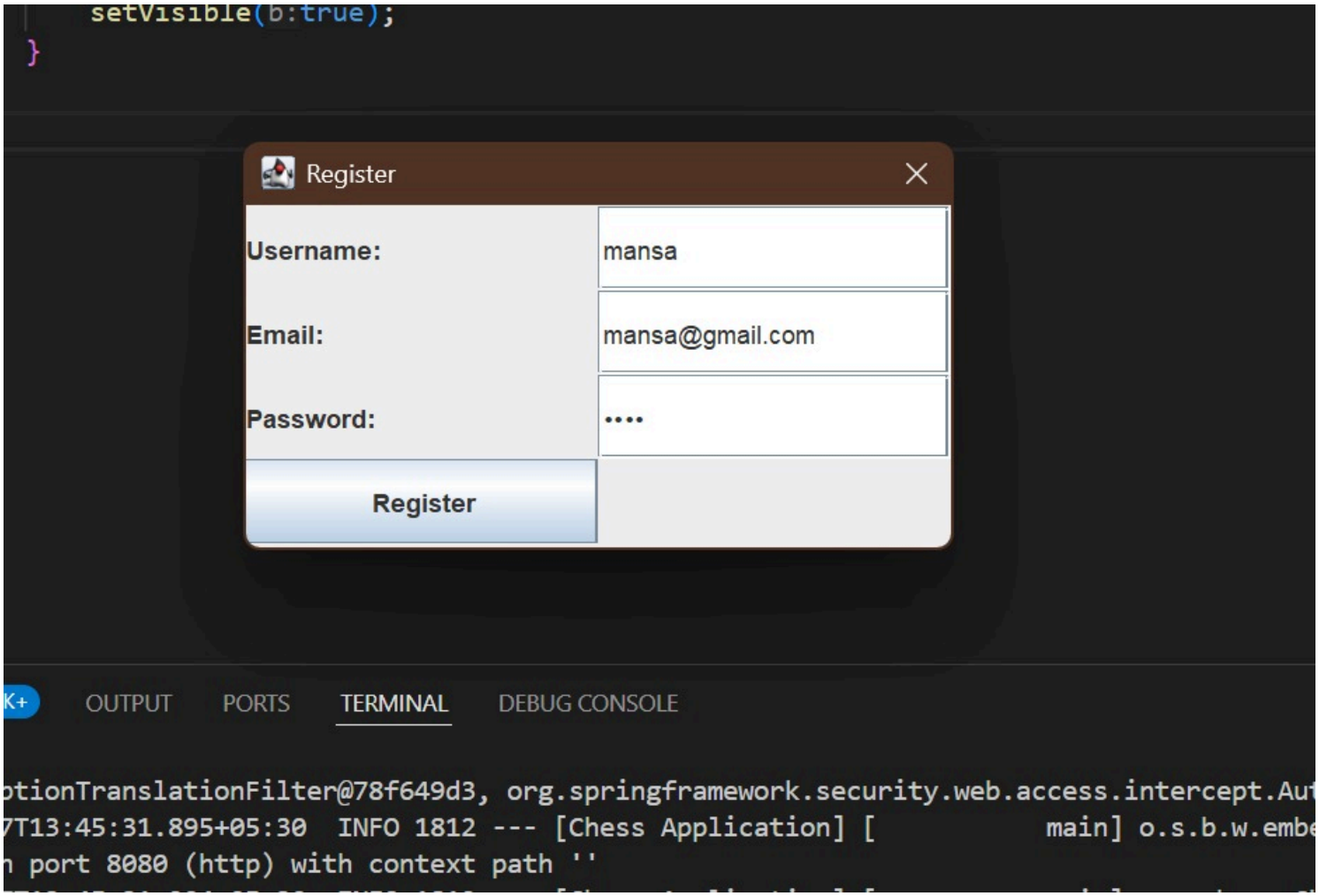
If invalid username or password is entered:


setVisible(b:true);
}
}
Login
Error
Invalid username or password.
OK
Login
Cancel
OUTPUT
PORTS
TERMINAL
DEBUG CONSOLE
ss.ExceptionTranslationFilter@78f649d3, org.springframework.security.web.access.intercept.Autho

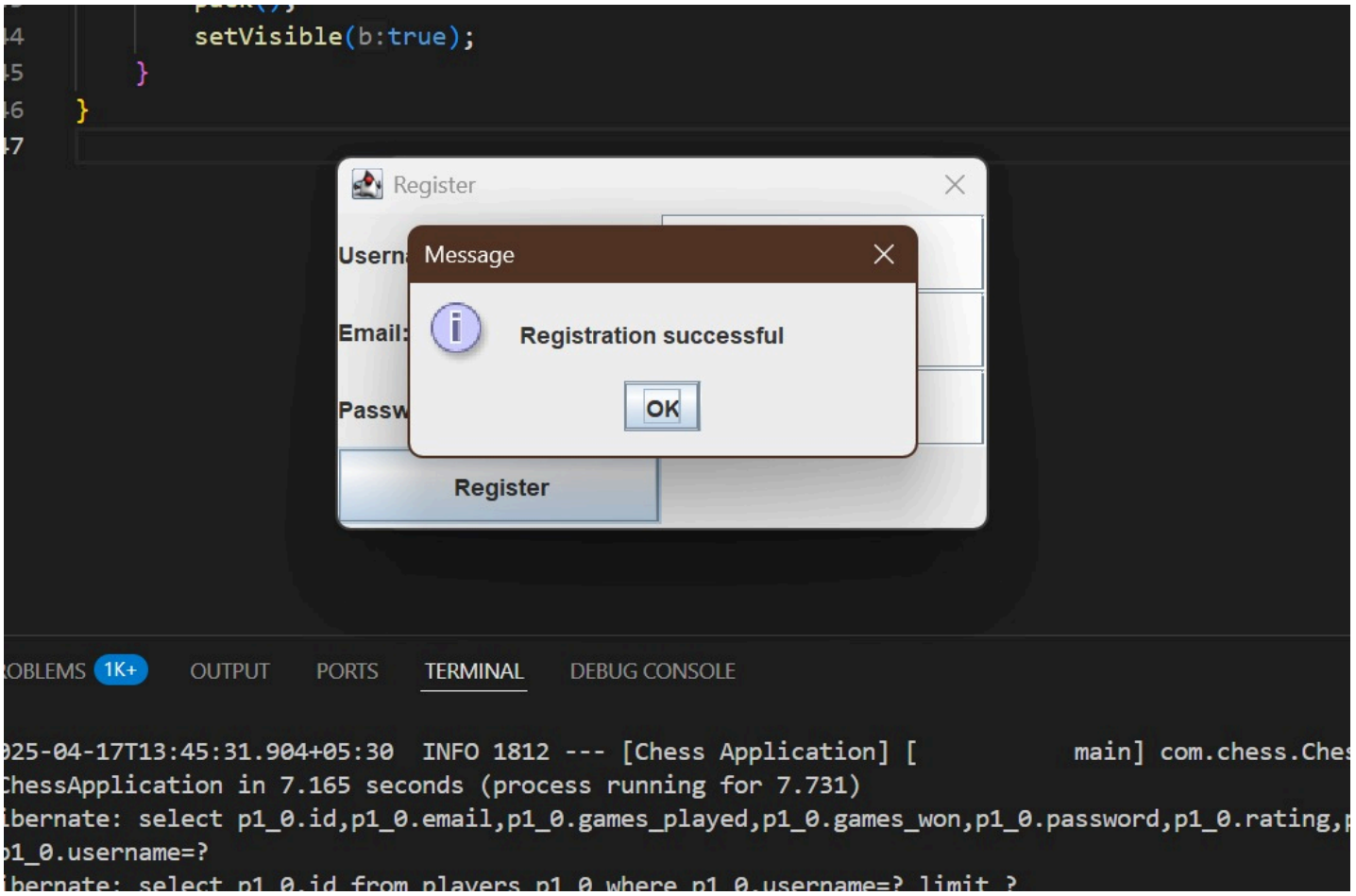
The following table appears after clicking on the registration option



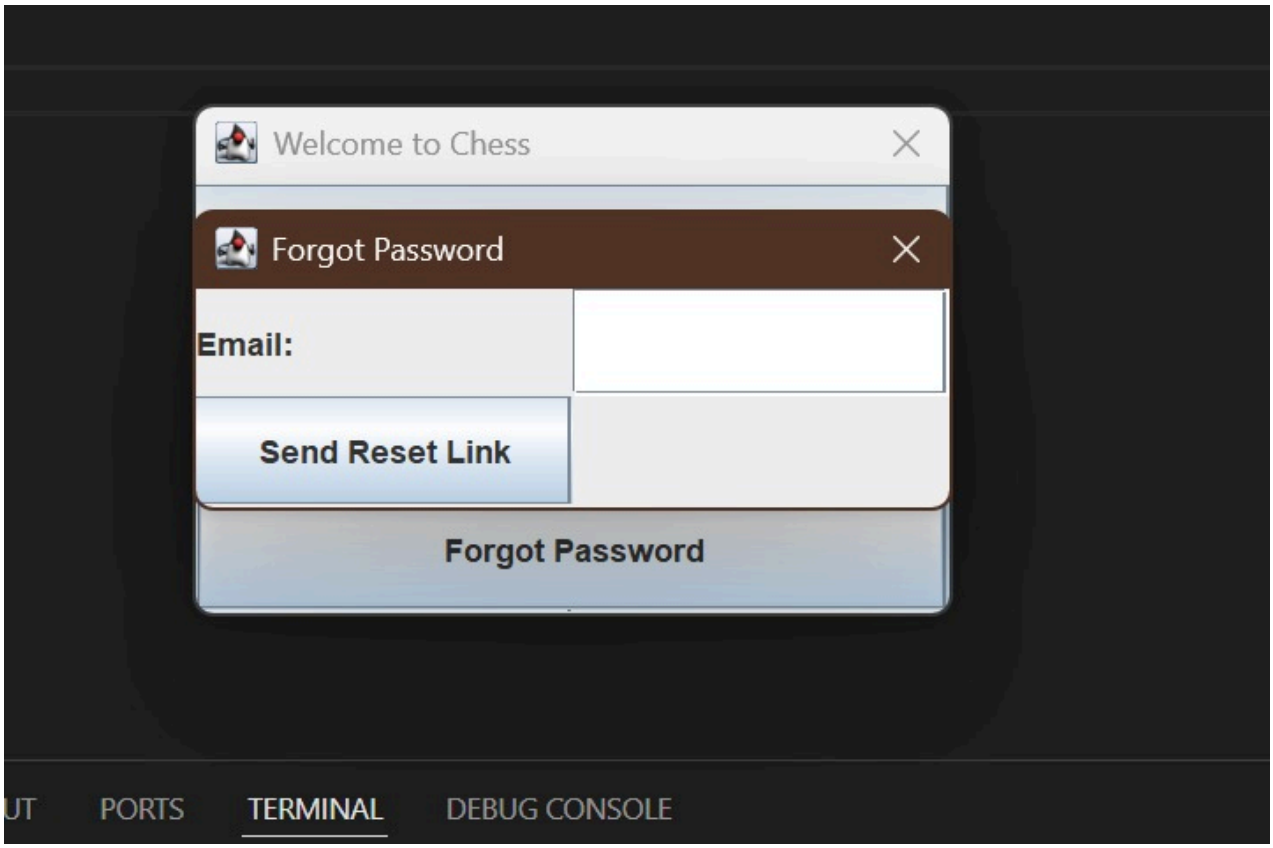
Enter details



after successful registration:

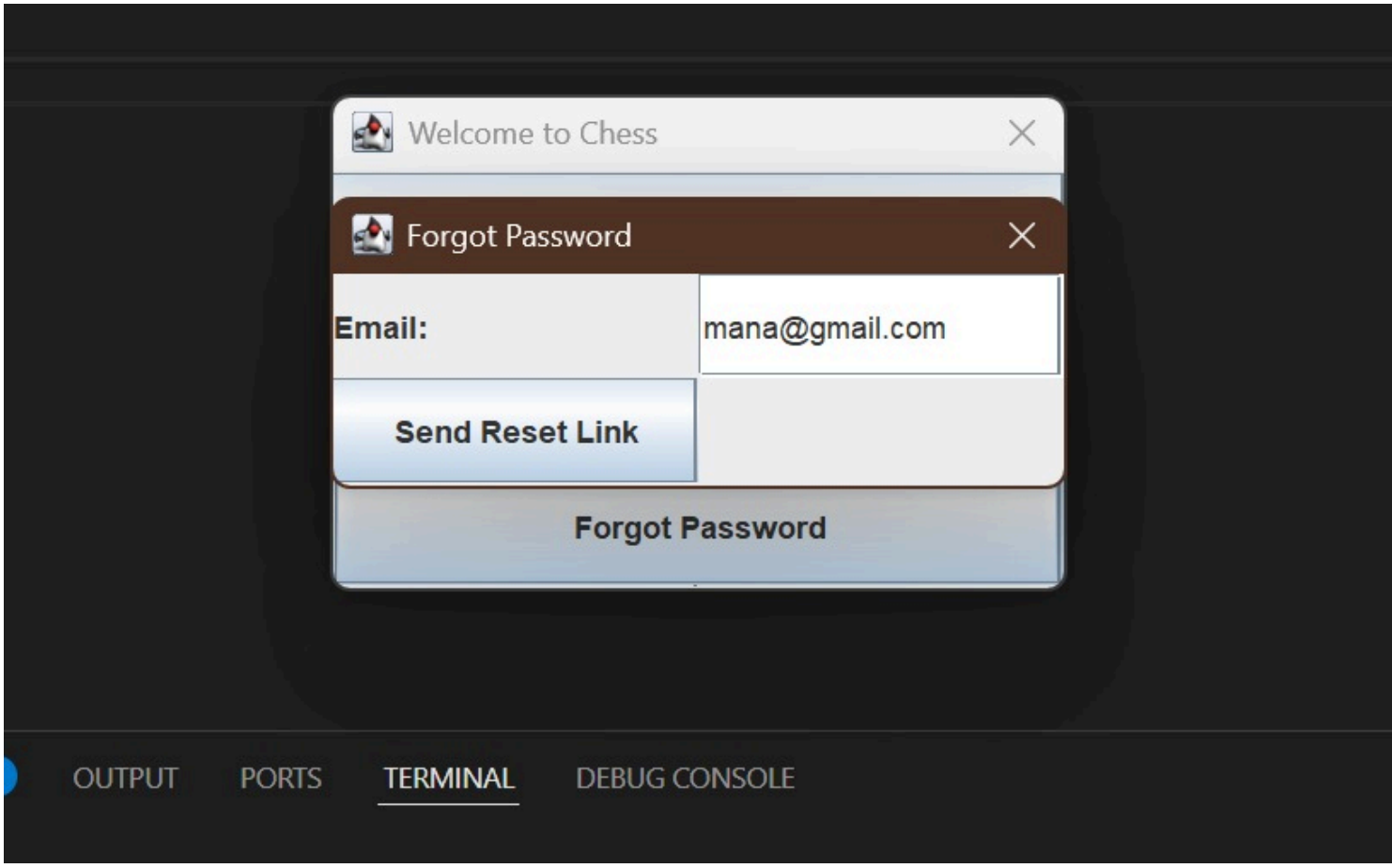


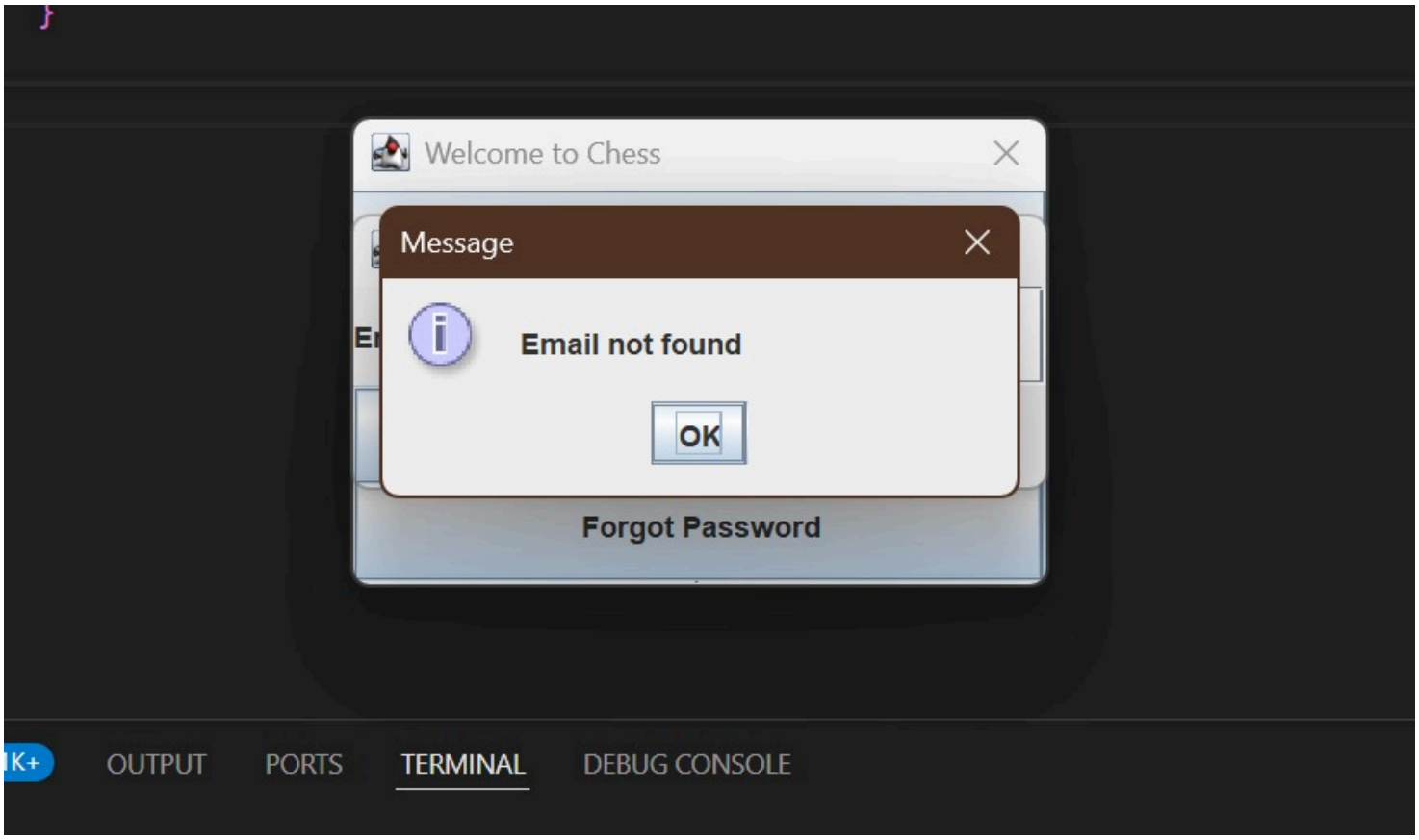
This page opens after clicking on the forgot password option:



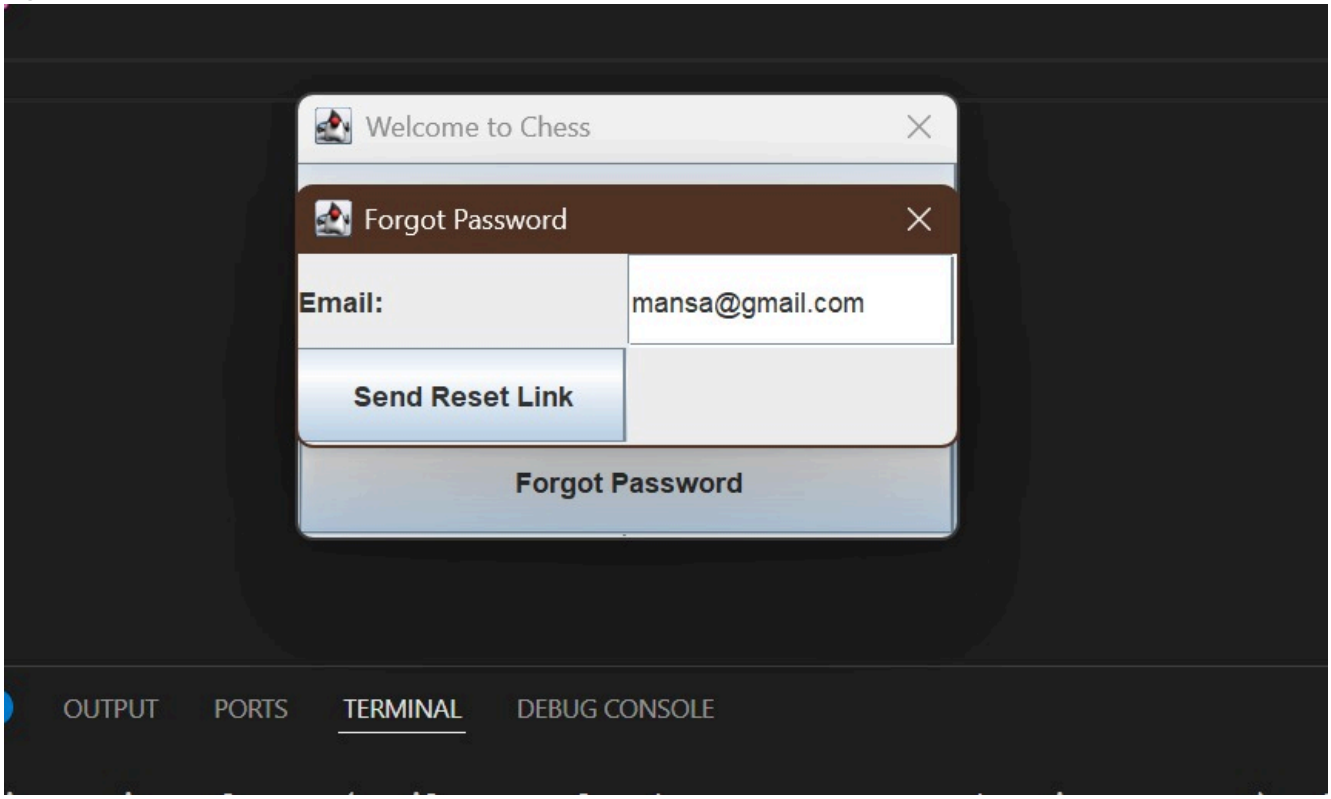
If the email entered in the appropriate section of the Forgot Password table does not exist in the database, an “Email not found” message is displayed as shown in

the below screenshots

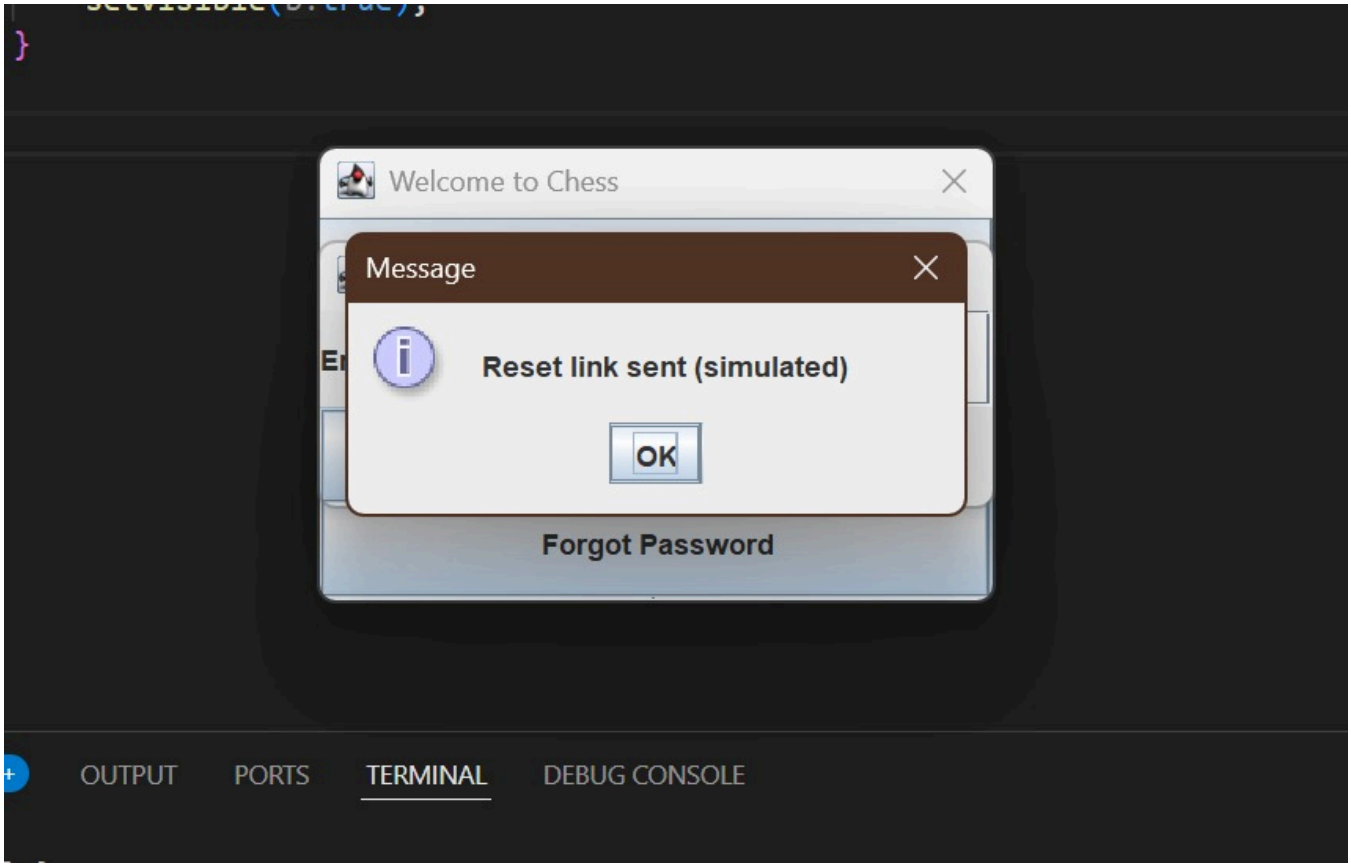




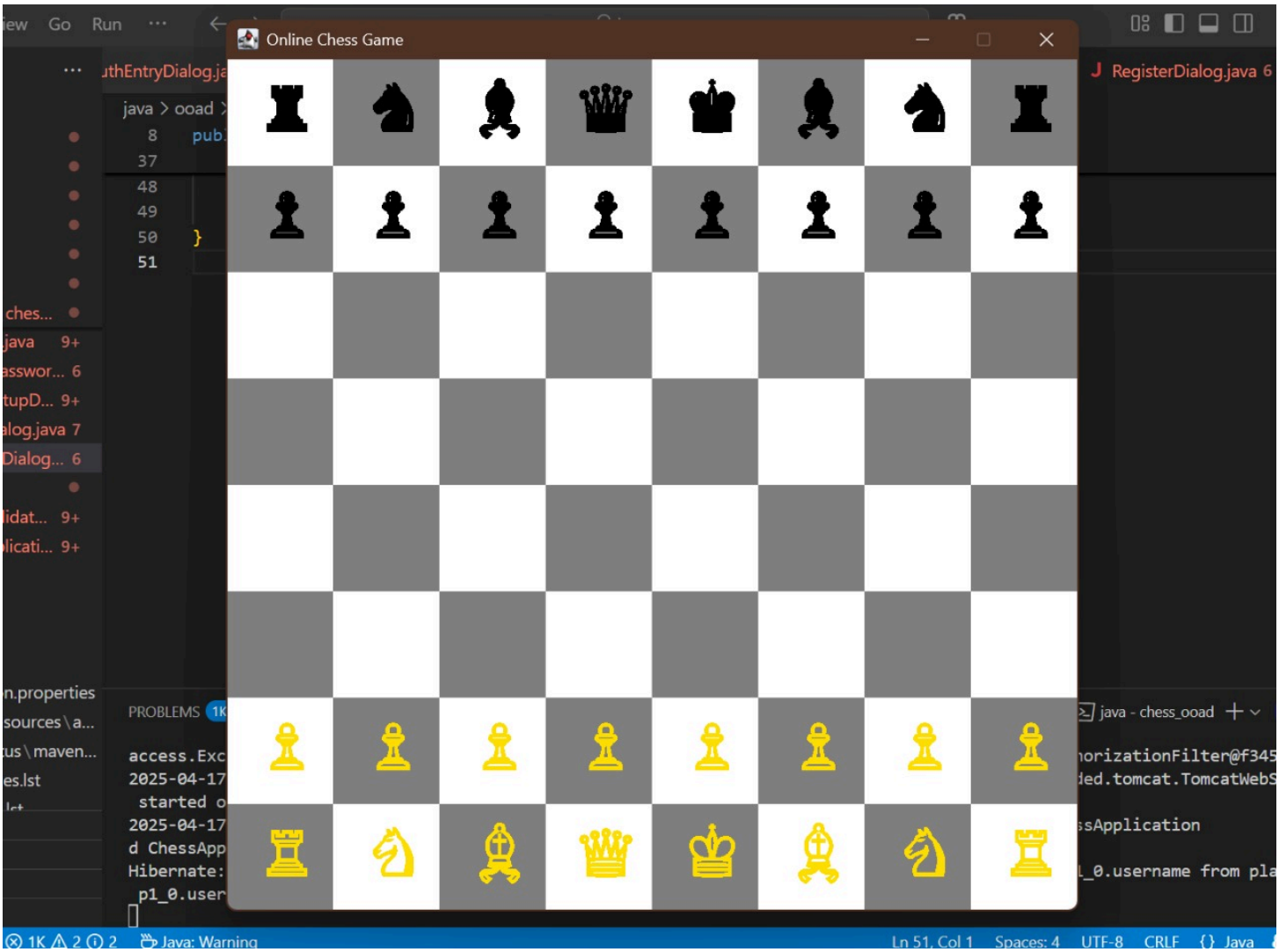
eg: This email exists in the database

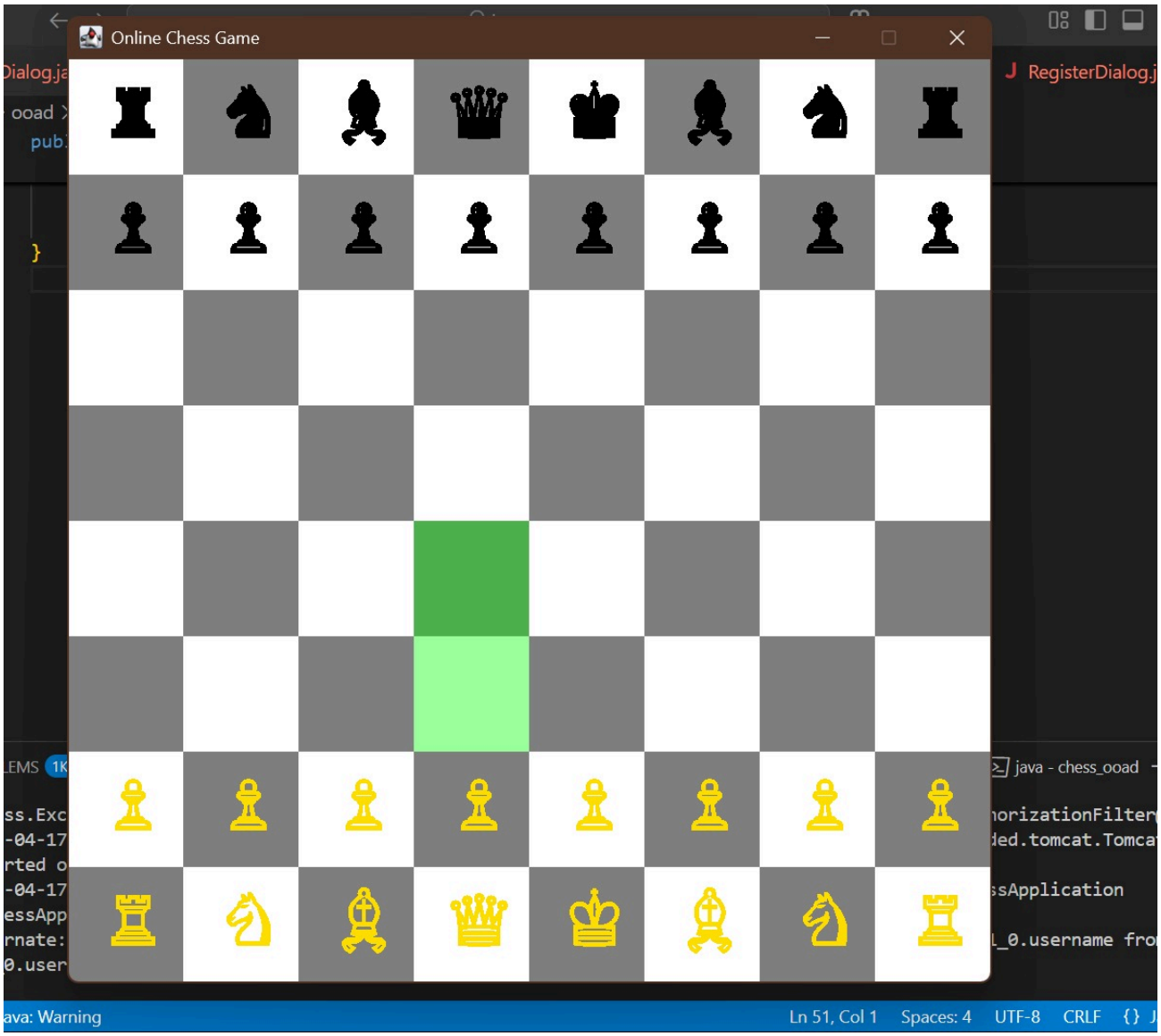


Since the entered email id exists in the database, the following image is displayed



The chessboard opens after a successful login as shown in the image

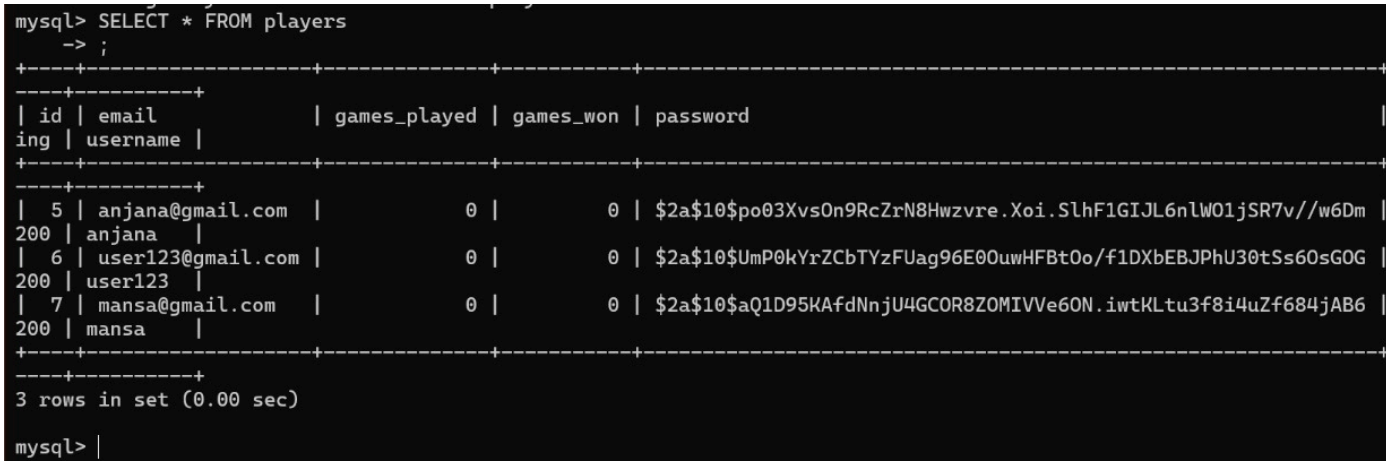

Online Chess Game
RegisterDialog.java 6
java > ooad
8 pub
37
48
49
50 }
51
PROBLEMS
java - chess_ooad
access.Exc
2025-04-17
started o
2025-04-17
d ChessApp
Hibernate:
p1_0.user
Java: Warning
Ln 51, Col 1
Spaces: 4
UTF-8
CRLF
Java




```
mysql> USE chess_db;
Database changed
mysql> SHOW TABLES;
+--------------------+
| Tables_in_chess_db |
+--------------------+
| games              |
| players            |
+--------------------+
2 rows in set (0.03 sec)

mysql>
```

The email ids and the encrypted passwords of the users are displayed in the database as shown below



```
mysql> SELECT * FROM players
    -> ;
+----+------------------+--------------+-----------+--------------------------------------------------------------+----------+
| id | email            | games_played | games_won | password                                                     | username |
+----+------------------+--------------+-----------+--------------------------------------------------------------+----------+
|  5 | anjana@gmail.com |            0 |         0 | $2a$10$po03XvsOn9RcZrN8Hwzvre.Xoi.SlhF1GIJL6nlWO1jSR7v//w6Dm | anjana   |
|  6 | user123@gmail.com |           0 |         0 | $2a$10$UmP0kYrZCbTYzFUag96E0OuwHFBtOo/f1DXbEBJPhU30tSs6OsGOG | user123  |
|  7 | mansa@gmail.com  |            0 |         0 | $2a$10$aQ1D95KAfdNnjU4GCOR8ZOMIVVe6ON.iwtKLtu3f8i4uZf684jAB6 | mansa    |
+----+------------------+--------------+-----------+--------------------------------------------------------------+----------+
3 rows in set (0.00 sec)

mysql>
```

Individual contributions of the team members:

Name	**Module worked on**
Maitri Maheshkumar Shekhda	chess board, chess pieces(black and white), checkmate conditions, highlighting the validity of the moves.
Mansa Pandey	Player authentication: register, login, forgot password, storing details in database, connecting to backend