Online Chess Game using SpringBoot

Team A15:
Ayush Praveen - PES1UG20CS900
Aneesh Ravishankar - PES1UG20CS051
Amogh Skanda Suresh - PES1UG20CS036
Ananya Menon - PES1UG20CS043

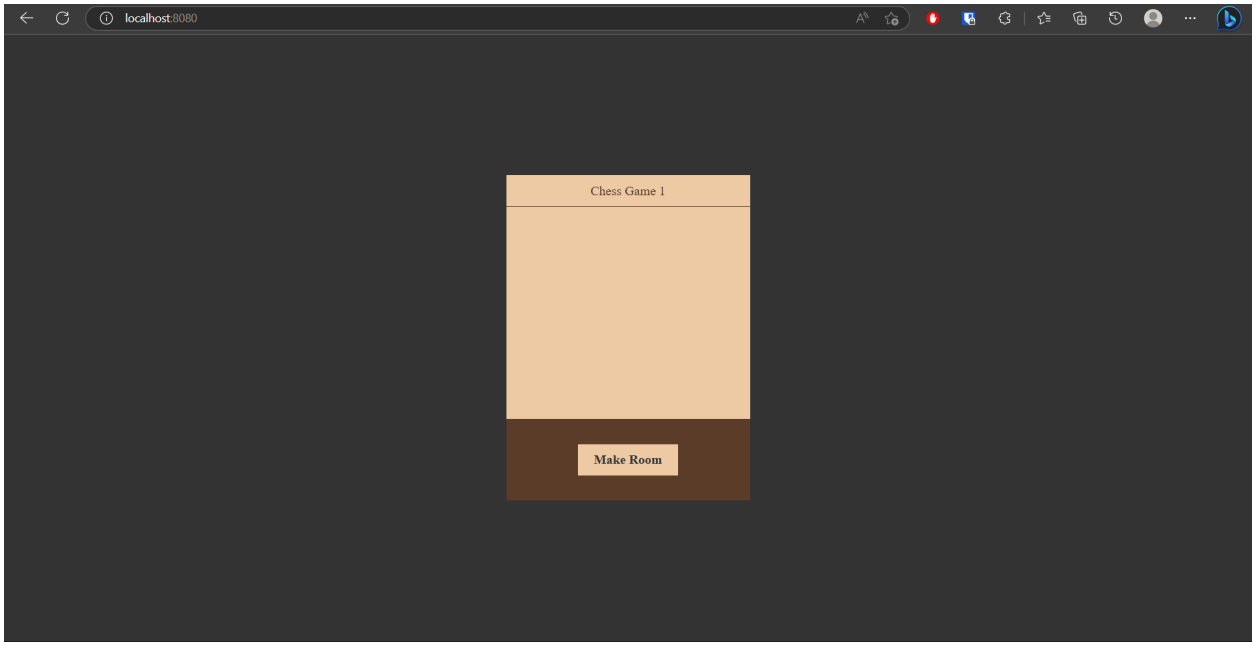
Synopsis:

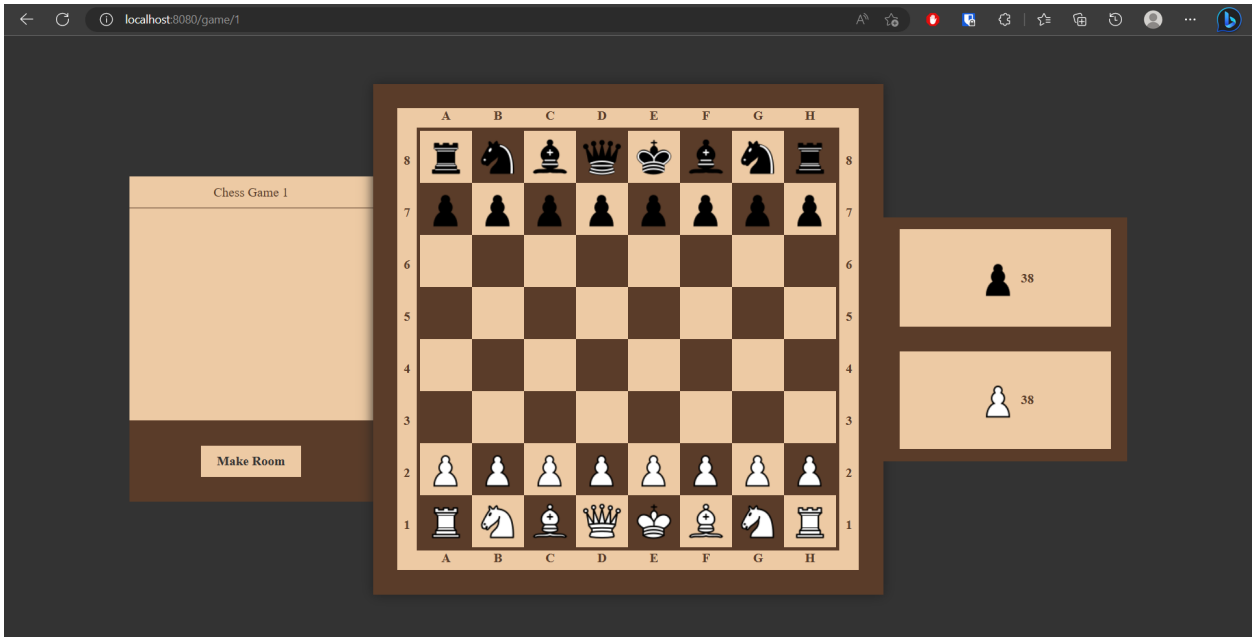
The online chess game is a web-based multiplayer game that allows players to play chess with each other in real-time. The game is implemented as a web application using modern web technologies such as Handlebars, CSS, JavaScript, and server-side frameworks like Spring Boot.

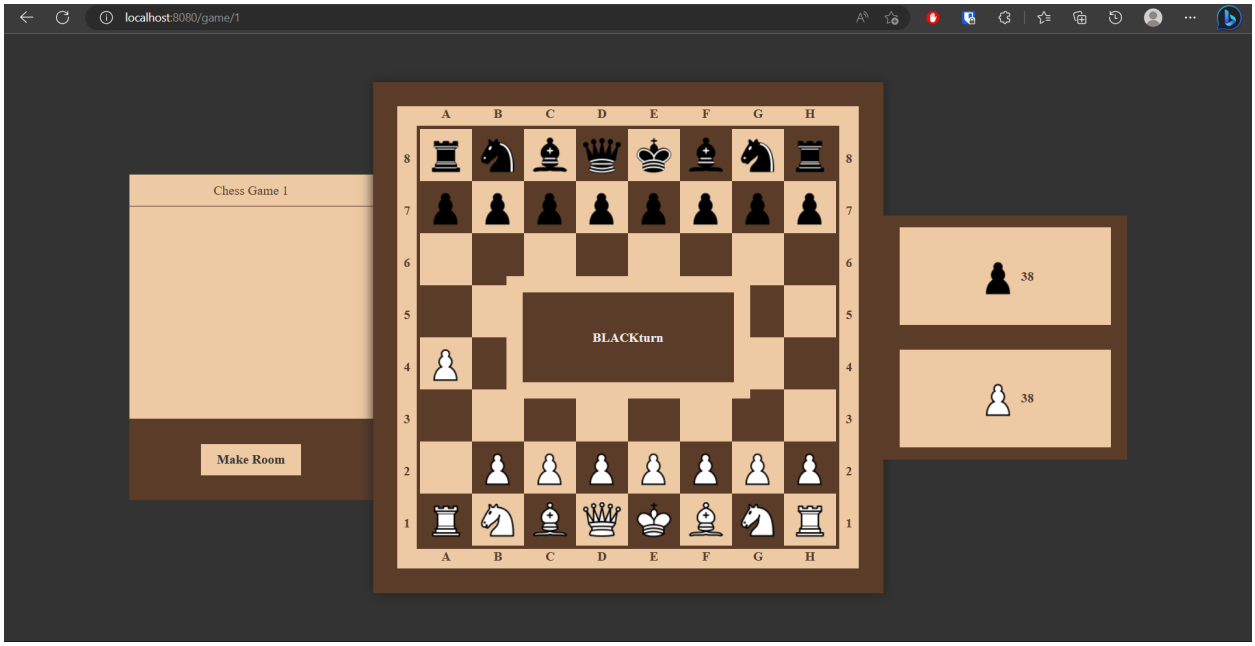
The game is designed to be easy to use and accessible to players of all levels of experience. It features a clean and intuitive user interface that allows players to create a room and start playing games quickly and easily.

The core gameplay mechanics of the game follow the standard rules of chess, with players taking turns moving their pieces on a virtual chessboard. The game includes features such as move validation, legal move highlighting, and game state tracking to ensure that the game is fair and accurate.

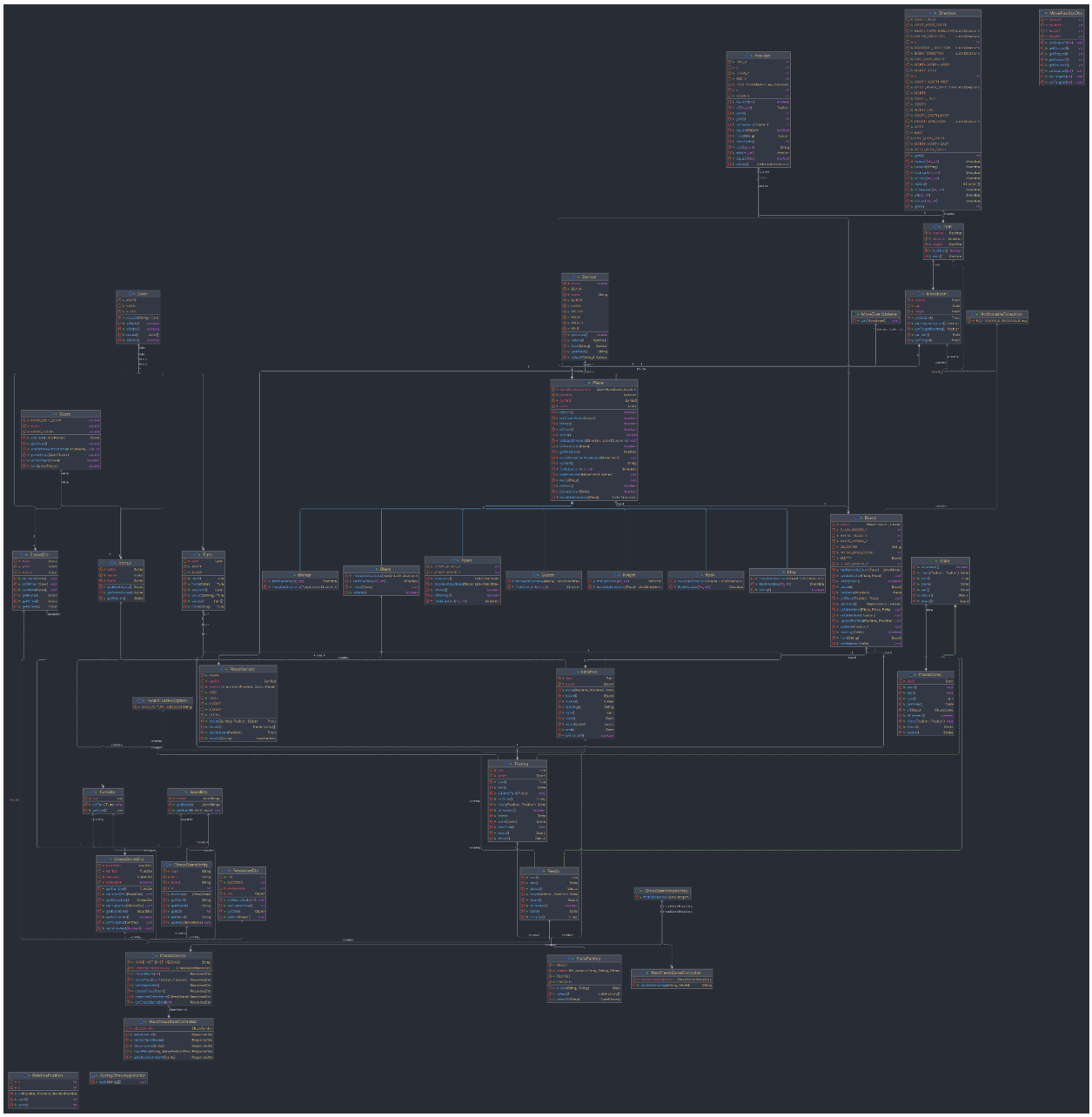
Overall, the online chess game provides a fun and engaging platform for players to enjoy the classic game of chess with their friends.


localhost:8080
Chess Game 1
Make Room


localhost:8080/game/1
Chess Game 1
Make Room
A B C D E F G H
8 7 6 5 4 3 2 1
38
38


localhost:8080/game/1
Chess Game 1
Make Room
A B C D E F G H
8 7 6 5 4 3 2 1
BLACKturn
38
38

Class Diagram:



SOLID Design Principles

1. SRP and DIP is :

Followed by:

-BoardDto

BoardDto depends on the board class which is a low-level module however through the Board interface

BoardDto.java 1

```java
package ooadj.chessmp.dto;

import java.util.List;
import java.util.stream.Collectors;

import ooadj.chessmp.domain.game.Board;

public class BoardDto {
    private List<String> board;

    public BoardDto(Board board) {
        this.board = board.getBoard()
            .values()
            .stream()
            .map(piece -> piece.isBlack() ? piece.symbol().toUpperCase() : piece.symbol())
            .collect(Collectors.toList());
    }

    public List<String> getBoard() {
        return board;
    }

    public void setBoard(List<String> board) {
        this.board = board;
    }
}

```

SRP violations:
Board

```java
public class Board {
    public static final Board EMPTY = new Board();
    private static final int BLACK_PIECES_Y = 7;
    private static final int BLACK_PAWNS_Y = 6;
    private static final int WHITE_PIECES_Y = 0;
    private static final int WHITE_PAWNS_Y = 1;
    private static final int INITIAL_KING_COUNT = 2;
    private static final String DELIMITER = "";

    private Map<Position, Piece> board;

    private Board() {
        this.board = new TreeMap<>();
    }

    public static Board create() {
        Board board = new Board();

        List<Symbol> pieceSequence = Arrays.asList(
            Symbol.ROOK, Symbol.KNIGHT, Symbol.BISHOP, Symbol.QUEEN,
            Symbol.KING, Symbol.BISHOP, Symbol.KNIGHT, Symbol.ROOK
        );

        for (Position position : Position.values()) {
            board.setBlank(position);
        }

        for (int x = Position.BEGIN_X; x < Position.END_X; x++) {
            board.setPiece(Position.of(x, BLACK_PIECES_Y),
                PieceFactory.create(pieceSequence.get(x), Position.of(x, BLACK_PIECES_Y), Color.BLACK));
            board.setPiece(Position.of(x, BLACK_PAWNS_Y),
                PieceFactory.create(Symbol.PAWN, Position.of(x, BLACK_PAWNS_Y), Color.BLACK));
            board.setPiece(Position.of(x, WHITE_PAWNS_Y),
                PieceFactory.create(Symbol.PAWN, Position.of(x, WHITE_PAWNS_Y), Color.WHITE));
            board.setPiece(Position.of(x, WHITE_PIECES_Y),
                PieceFactory.create(pieceSequence.get(x), Position.of(x, WHITE_PIECES_Y), Color.WHITE));
        }
```

2. Open close:
Followed by:
Chessgamerepository:

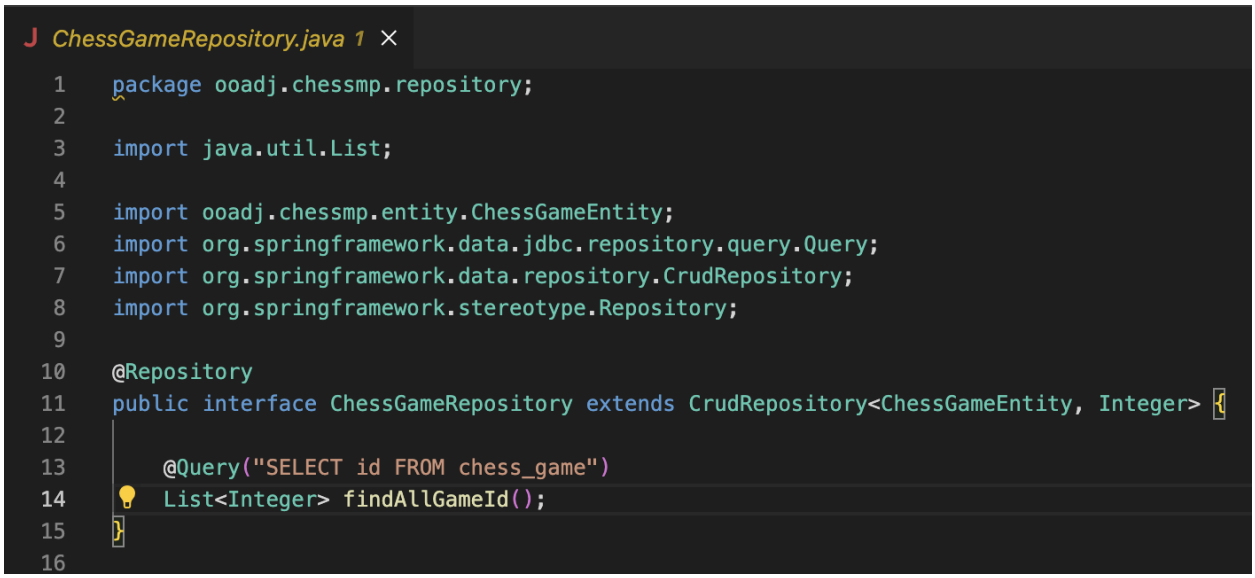
```java
import java.util.List;

import ooadj.chessmp.entity.ChessGameEntity;
import org.springframework.data.jdbc.repository.query.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

7 usages
@Repository
public interface ChessGameRepository extends CrudRepository<ChessGameEntity, Integer> {

    2 usages
    @Query("SELECT id FROM chess_game")
    List<Integer> findAllGameId();
}
```

3. LSP is implemented in



ChessGameRepository.java 1

```java
package ooadj.chessmp.repository;

import java.util.List;

import ooadj.chessmp.entity.ChessGameEntity;
import org.springframework.data.jdbc.repository.query.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ChessGameRepository extends CrudRepository<ChessGameEntity, Integer> {

    @Query("SELECT id FROM chess_game")
    List<Integer> findAllGameId();
}

```

DESIGN PATTERNS

Singleton pattern is used where a single instance of the class is created and a global point of access is given to that instance

Some code snippets :

Board:

```java

public class Board {
    2 usages
    public static final Board EMPTY = new Board();
    2 usages
    private static final int BLACK_PIECES_Y = 7;
    2 usages
    private static final int BLACK_PAWNS_Y = 6;
    2 usages
    private static final int WHITE_PIECES_Y = 0;
    2 usages
    private static final int WHITE_PAWNS_Y = 1;
    1 usage
    private static final int INITIAL_KING_COUNT = 2;
    1 usage
    private static final String DELIMITER = "";

    11 usages
    private Map<Position, Piece> board;
```

When the game is finished/ends

finished.java

```java
public class Finished implements State {
    private Board board;
    private Turn turn;

    public Finished(Board board, Turn turn) {
        this.board = board;
        this.turn = turn;
    }

    private Score score(Color color) {
        return Score.calculate(board.findPiecesByColor(color));
    }

    @Override
    public State start() {
        throw new UnsupportedOperationException();
    }

    @Override
    public State end() {
        throw new UnsupportedOperationException();
    }

    @Override
    public State move(Position source, Position target) {
        throw new UnsupportedOperationException();
    }

    @Override
    public Board board() {
        return board;
    }

    @Override
    public Turn turn() {
        return turn;
    }
```

Board.java -

```java
public class Board {
    public static final Board EMPTY = new Board();
    private static final int BLACK_PIECES_Y = 7;
    private static final int BLACK_PAWNS_Y = 6;
    private static final int WHITE_PIECES_Y = 0;
    private static final int WHITE_PAWNS_Y = 1;
    private static final int INITIAL_KING_COUNT = 2;
    private static final String DELIMITER = "";

    private Map<Position, Piece> board;

    private Board() {
        this.board = new TreeMap<>();
    }

    public static Board create() {
        Board board = new Board();

        List<Symbol> pieceSequence = Arrays.asList(
            Symbol.ROOK, Symbol.KNIGHT, Symbol.BISHOP, Symbol.QUEEN,
            Symbol.KING, Symbol.BISHOP, Symbol.KNIGHT, Symbol.ROOK
        );

        for (Position position : Position.values()) {
            board.setBlank(position);
        }

        for (int x = Position.BEGIN_X; x < Position.END_X; x++) {
            board.setPiece(Position.of(x, BLACK_PIECES_Y),
                PieceFactory.create(pieceSequence.get(x), Position.of(x, BLACK_PIECES_Y), Color.BLACK));
            board.setPiece(Position.of(x, BLACK_PAWNS_Y),
                PieceFactory.create(Symbol.PAWN, Position.of(x, BLACK_PAWNS_Y), Color.BLACK));
            board.setPiece(Position.of(x, WHITE_PAWNS_Y),
                PieceFactory.create(Symbol.PAWN, Position.of(x, WHITE_PAWNS_Y), Color.WHITE));
            board.setPiece(Position.of(x, WHITE_PIECES_Y),
                PieceFactory.create(pieceSequence.get(x), Position.of(x, WHITE_PIECES_Y), Color.WHITE));
        }

        return board;
    }
```

Factory Method :

```java
public enum PieceFactory {
    PAWN(Symbol.PAWN, Pawn::new),
    ROOK(Symbol.ROOK, Rook::new),
    KNIGHT(Symbol.KNIGHT, Knight::new),
    BISHOP(Symbol.BISHOP, Bishop::new),
    QUEEN(Symbol.QUEEN, Queen::new),
    KING(Symbol.KING, King::new);

    private final Symbol symbol;
    private final BiFunction<Position, Color, Piece> creator;

    PieceFactory(Symbol symbol, BiFunction<Position, Color, Piece> creator) {
        this.symbol = symbol;
        this.creator = creator;
    }

    public static Piece create(Symbol symbol, Position position, Color color) {
        return Arrays.stream(values())
            .filter(pieceFactory -> pieceFactory.symbol == symbol)
            .findFirst()
            .orElseThrow(IllegalArgumentException::new)
            .creator.apply(position, color);
    }

    public static Piece createBlank(Position position) {
        return new Blank(position);
    }
}

```

Iterator Design Pattern

```java
package ooadj.chessmp.domain.piece;

import java.util.Iterator;

public class Path implements Iterator<Position> {
   private Position source;
   private Position target;
   private Direction direction;

   public Path(Position source, Position target, Direction direction) {
       this.source = source;
       this.target = target;
       this.direction = direction;
   }

   @Override
   public boolean hasNext() {
       return source.add(direction.getX(), direction.getY()) != target;
   }

   @Override
   public Position next() {
       return source = source.add(direction.getX(), direction.getY());
   }
}
```