math_2024_052

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Факультет математики та інформатики
Кафедра математичного моделювання

Розробка онлайн гри “Шахи” з використанням фреймоворку Spring

Кваліфікаційна робота
Рівень вищої освіти – перший (бакалаврський)

Виконав:
студент 4 курсу, 407 групи
Волинський Денис Володимирович

Керівник:
доктор фіз.-мат. наук, професор
Черевко І.М.

До захисту допущено
на засіданні кафедри
протокол № 17 від 6 червня 2024 р.
Зав. кафедрою _____ проф. Черевко І.М.

Чернівці – 2024

Анотація

Кваліфікаційна робота спрямована на вивчення та практичне застосування фреймворку Spring для розробки Веб застосунку для створення гри “Шахи”. В роботі реалізовано створення бази даних із користувачами, взаємодію бази даних із користувачами на стороні сервера за допомогою фреймворку Spring, реєстрацію та валідацію користувачів, здійснено програмування поведінки шахових фігур на стороні сервера та їх відображенні на стороні клієнта у веб-браузері, а також програмування ігрових подій.

Для реалізації сервісів використовувались технології Spring (модулі: SpringWeb, Spring Validation, Spring Boot, Web Socket, Thymeleaf), MongoDB, JQuery, SockJS, STOMP.

Результатом проведеної роботи є веб застосунок «шахи» із можливістю гри користувачами по мережі.

Ключові слова: Spring Framework, Веб-застосунок, Гра "Шахи", Взаємодія сервер-клієнт, Програмування шахових фігур, Серверне програмування, Клієнтська візуалізація

Annotation

The qualification work is aimed at the study and practical application of the Spring framework for the development of a Web application for creating the game "Chess". In the work, the creation of a database with users, database interaction with users on the server side using the Spring framework, user registration and validation, programming of the behavior of chess pieces on the server side and their display on the client side in a web browser, as well as programming of game events

The services were implemented using Spring technologies (modules: SpringWeb, Spring Validation, Spring Boot, Web Socket, Thymeleaf), MongoDB, JQuery, SockJS, STOMP.

The result of the work is a web application of "chess" with the possibility of playing by users over the network.

Keywords: Spring Framework, Web Application, Chess Game, Server-Client Interaction, Chess Pieces Programming, Server Programming, Client Visualization

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів наукових досліджень інших авторів мають посилання на відповідне джерело.

_____ Д. А. Пантюхін
(підпис)

Зміст

ВСТУП

Spring Framework - це потужний і всеосяжний фреймворк з відкритим кодом для розробки додатків на мові Java. Створений Родом Джонсоном у 2003 році під керівництвом Роберта Харропа з компанії Interface21 (пізніше Pivotal Software).

Род Джонсон розробив Spring як альтернативу моделі розробки Enterprise JavaBeans (EJB), яка була складною і важкою для використання. На відміну від EJB, Spring базувався на простіших концепціях, таких як прямий POJO-програмування (Plain Old Java Objects), впровадження залежностей (DI) та аспектно-орієнтоване програмування (AOP).

Перша версія Spring 1.0 була випущена в березні 2004 року під ліцензією Apache 2.0. Вона містила ядро контейнера DI, модуль AOP та абстракцію для доступу до даних. З часом Spring еволюціонував, додаючи підтримку веб-розробки з MVC, безпеки, тестування та багато іншого.

Spring активно розвивається спільнотою, спонсорується Pivotal Software і випускається під ліцензією Apache 2.0. Екосистема Spring постійно зростає, пропонуючи інструменти для хмарних обчислень, реактивного програмування, обробки великих даних та інтернету речей.

Завдяки своїй модульній структурі, чистому коду та активній підтримці, Spring Framework залишається однією з найпопулярніших технологій для розробки корпоративних додатків на Java. Він широко використовується у великих технологічних компаніях, фінансових установах та урядових організаціях по всьому світу.

У процесі виконання кваліфікаційної роботи здійснено дослідження можливостей фреймворку Spring для створення гри “шахи”. Розглянуто основні поняття, модулі та методи застосувань, роботу із базами даних, валідація даних, розробка сайту. Здійснено написання логіки гри в шахи мовою java та взаємодії веб частини застосунку і серверної частини, динамічну обробку ігрових подій на стороні клієнта.

РОЗДІЛ І. Фреймворк Spring

1.1. Вступ. Основні поняття [1]

Spring - це популярний Java-фреймворк, який допомагає швидше і простіше створювати веб-застосунки. Він був створений у 2003 році і з тих пір став одним з найпопулярніших інструментів для Java-розробки. Spring робить код чистішим, легшим для тестування і дозволяє зосередитися на створенні функціональної частини додатку, а не на конфігурації середовища.

Основні поняття Spring Framework:

1. Біни (Beans): У Spring, всі головні частини програми (об’єкти) називаються "бінами".
2. Впровадження залежностей (Dependency Injection, DI): Spring автоматично надає бінам для створення те, що їм потрібно. Це зменшує кількість коду і робить його простішим.
3. Інверсія управління (Inversion of Control, IoC): Біни створюються і керуються Spring IoC контейнером, а не безпосередньо розробником.
4. Анотації: Це спеціальні позначки в коді, які кажуть Spring, як поводитися з класом. Для прикладу анотація @Controller каже, що клас обробляє веб-запити, @Service - що він містить бізнес-логіку.
5. Model-View-Controller (MVC): Це шаблон для веб-додатків. Model - дані, View - те, що бачить користувач, Controller - обробляє дії користувача та налаштовує взаємодію із сервером та клієнтом.
6. Spring Boot: Це надбудова над Spring, яка робить налаштування ще простішим. Багато речей працюють "з коробки", без додаткових налаштувань.
7. Робота з базами даних: Spring полегшує та стандартизує збереження даних у базу даних. Він має інструменти, які зменшують кількість коду, та зменшує шанс розробникам писати баги.
8. Безпека: Spring Security допомагає захистити вебзастосунок. Він легко налаштовується для роботи з логінами, паролями та правами доступу.
9. Шаблони (Templates): Spring працює з такими інструментами як Thymeleaf, які дозволяють легко створювати динамічні веб-сторінки.
10. Тестування: Spring має вбудовані функції, які полегшують тестування кожної частини вебзастосунку. Це допомагає переконатися, що все працює правильно.

1.2 Особливості фреймфорку Spring [2-3]

Фреймворк має кілька особливостей, які роблять його зручним та популярним для розробки вебдодатків на мікросервісній архітектурі, RESTful веб-сервісів для десктопних, мобільних та веб додатків. Деякі з цих особливостей включають:

1. Легкість у використанні:
    - Проста конфігурація через анотації та Java-коди, без надмірного використання XML.
    - Конвенція над конфігурацією, зменшуючи потребу в шаблонному-коді.
    - Чіткі абстракції та прості моделі програмування (POJO).
2. Контейнер IoC (Inversion of Control):
    - Впровадження залежностей (DI) без плутаних залежностей.
    - Підтримка декількох областей видимості для управління життєвим циклом бінів (сінглтони, прототипи, HTTP-запити, сесії тощо).
3. Аспектно-орієнтоване програмування (AOP):
    - Розділення навколишньої функціональності (логування, безпека, управління транзакціями) від основної бізнес-логіки.
    - Підтримка декларативного програмування (анотації) та інтроспекції на рівні байт-коду.
4. Абстракції для баз даних:
    - Spring JDBC: Спрощений доступ до JDBC.
    - Spring ORM: Інтеграція з Hibernate, JPA.
    - Spring Data: Абстракція для різних NoSQL сховищ (MongoDB, Cassandra, Redis тощо).
5. Веб-модуль Spring MVC:
    - Потужна реалізація патерну MVC для веб-розробки.
    - Сучасне асинхронне програмування, RESTful веб-сервіси.
    - Підтримка різних видів повернення: @ResponseBody, View Resolvers, Content Negotiation.
6. Безпека Spring Security:
    - Потужний фреймворк безпеки для аутентифікації та авторизації.
    - Підтримка різних стандартів та протоколів (LDAP, OAuth, SAML, JWT).
7. Тестування:
    - Підтримка Mock об'єктів через бібліотеки як Mockito.
    - Просте юніт-тестування з використанням мокінгу та контейнера IoC.
    - Спеціальні анотації для інтеграційного тестування (@SpringBootTest, @WebMvcTest).

8. Інтеграція:
    - Широка інтеграція з багатьма бібліотеками та фреймворками (JMS, JavaMail, AMQP, Quartz тощо).
    - Підтримка віддаленого доступу через Spring Remoting (RMI, HTTP Invoker, Hessian, Burlap, AMQP).
9. Реактивне програмування:
    - Підтримка реактивного стилю програмування через Project Reactor.
    - Реактивні веб-програми на основі Spring WebFlux.

11. Spring Boot:
    - Швидке створення автономних виробничих додатків.
    - Вбудований веб-сервер Tomcat, автоматична конфігурація, метрики тощо.
    - Велика екосистема "стартерів" для різної функціональності.
12. Модульність:
    - Spring складається з більш ніж 20 модулів, що можна використовувати незалежно.

1.3. Набір Модулів Spring [1-3]

Spring Framework складається з набору модулів, кожен з яких забезпечує певний функціонал. Ось основні модулі Spring:

1. **Spring Core:**
    - Ядро контейнера Spring
    - Впровадження залежностей (Dependency Injection)
    - Подієва модель (Event System)
    - Ресурси (Resource Abstraction)
    - Валідація (Validation)
    - Колекції (Collections)
2. **Spring Context:**
    - Інтернаціоналізація (I18n)
    - Контекст додатку (Application Context Events)
    - Доступ до ресурсів (Resource Loading)
    - Прозоре створення контексту (Transparent Creation)
3. **Spring AOP:**
    - Аспектно-орієнтоване програмування (AOP)
    - Введення в Spring AOP

4. **Spring Web:**
    - Веб-додатки
    - Веб-клієнти
    - Тестування веб-контексту
5. **Spring WebMVC:**
    - Веб-модель-вигляд-контролер (Web MVC)
6. **Spring WebFlux:**
    - Реактивна веб-розробка
    - Реактивні веб-додатки (Reactive Web Applications)
7. **Spring WebSocket:**
    - Двосторонні веб-сокети (Bidirectional WebSockets)
8. **Spring Data:**
    - Інтеграція з JDBC (DB Access with JDBC)
    - Інтеграція з ORM (ORM with JPA/Hibernate)
    - Підтримка NoSQL сховищ даних (Support for NoSQL Data Stores)
9. **Spring Security:**
    - Безпека веб-додатків (Web Application Security)
    - Аутентифікація (Authentication)
    - Авторизація (Authorization)
10. **Spring Integration:**
    - Інтеграція додатків (Application Integration)
    - Обробка повідомлень (Messaging)
11. **Spring Batch:**
    - Пакетна обробка (Batch Processing)
    - Робота з великими обсягами записів (Bulk Operations)
12. **Spring Cloud:**
    - Розробка мікросервісів (Microservices Development)
    - Сервісне виявлення (Service Discovery)
    - Конфігураційний сервер (Config Server)

Ці модулі можна використовувати окремо або в комбінації, в залежності від потреб проекту. Наприклад, для створення веб-додатку можна використовувати Spring Core, Spring Web, Spring WebMVC та Spring Security. Для мікросервісної архітектури можуть знадобитися Spring Core, Spring Cloud та Spring Data.

1.4. Початок розробки вебзастосунку [2-4]

Spring Boot значно спрощує процес розробки, надаючи автоматичне налаштування, вбудований сервер та швидкий старт розробки. Завдяки цьому можна зосередитися на написанні основної бізнес-логіки веб-застосунку.

Для початку розробки веб-застосунку на Spring Framework, потрібно виконати кілька основних кроків:

1. **Встановити середовище розробки**
   - Встановити Java Development Kit (JDK) - середовище виконання Java. Наприклад JDK 18.
   - Встановити інтегроване середовище розробки (IDE), наприклад IntelliJ IDEA.
2. **Створити новий проект Spring Boot**
   - Створити проєкт за допомогою Spring Initializr (https://start.spring.io).
   - Вибрати потрібні залежності для веб-розробки, наприклад, "Spring Web", "Thymeleaf" (для шаблонів) та "Spring Data JPA" (для роботи з резяційними базами даних).
   - Завантажити згенерований проект і імпортувати його в IDE.
3. **Налаштувати Spring Boot**
   - Створити головний клас зі анотацією @SpringBootApplication - точка входу для додатку.
   - Створити файл application.properties (або application.yml) для налаштування властивостей додатку, наприклад порт сервера та налаштування бази даних.
4. **Розробити модель даних**
   - Створити класи-сутності (entities) зі анотаціями @Entity для зберігання даних у базі даних.
   - Створити репозиторії зі анотацією @Repository для доступу до даних (використовуючи Spring Data JPA).
5. **Створити веб-контролери**
   - Визначити класи контролерів зі анотацією @Controller або @RestController для обробки HTTP-запитів.
   - Прив’язати HTTP-методи (@GetMapping, @PostMapping тощо) з методами контролера.
   - Повертати дані у вигляді модельних об'єктів або шаблонів для відображення.
6. **Налаштувати шляхи та шаблони (View)**
   - Створити HTML-шаблони (Thymeleaf) в каталозі src/main/resources/templates.

- Використовувати синтаксис Thymeleaf (th:*) для динамічного відображення даних з контролерів.

7. **Запустити додаток**
   - Запустити Spring Boot додаток з головного класу зі анотацією @SpringBootApplication.
   - Перевірити функціонал у браузері (http://localhost:8080).

РОЗДІЛ ІІ. Теоритечні відомості про роботу

2.1. Постановка завдання

1. Ознайомитись із способами реалізізації вебзастосунка «Шахи»
2. Ознайомитись із фреймворком Spring і його можливостями
3. Вибір архітектури (MVC)
4. Створення та первинне налаштування Spring
5. Створення сайту та бази даних
6. Переписання ігрової логіки із курсової роботи із мови Python на Java
7. Тестування і виправлення

2.2. Ігрова логіка гри «Шахмати»[5]

Шахи - це стратегічна настільна гра для двох гравців, в якій гравці переміщують фігури з метою захопити короля противника. Гравці ходять по черзі кожен хід в переміщаючи свою фігуру на порожню клітку або на клітку, зайняту фігурою супротивника, яку можна захопити. Мета гри в шахи полягає в тому, щоб захопити короля супротивника. Якщо король опиняється під ударом і може бути захищений, це називається "шахом" . Якщо король перебуваючи під шахом, не може захиститися, це називається "матом", що означає кінець гри.

Кожна фігура у шахах має свої особливості та правила ходу. Ось короткий опис, як кожна фігура ходить:

- **Пішак** - найслабша шахова фігура. Ходить на одну клітинку вперед, за винятком першого ходу, коли вона може піти на дві клітинки вперед. Вона б'є по діагоналі на одну клітку вперед ліворуч і праворуч.

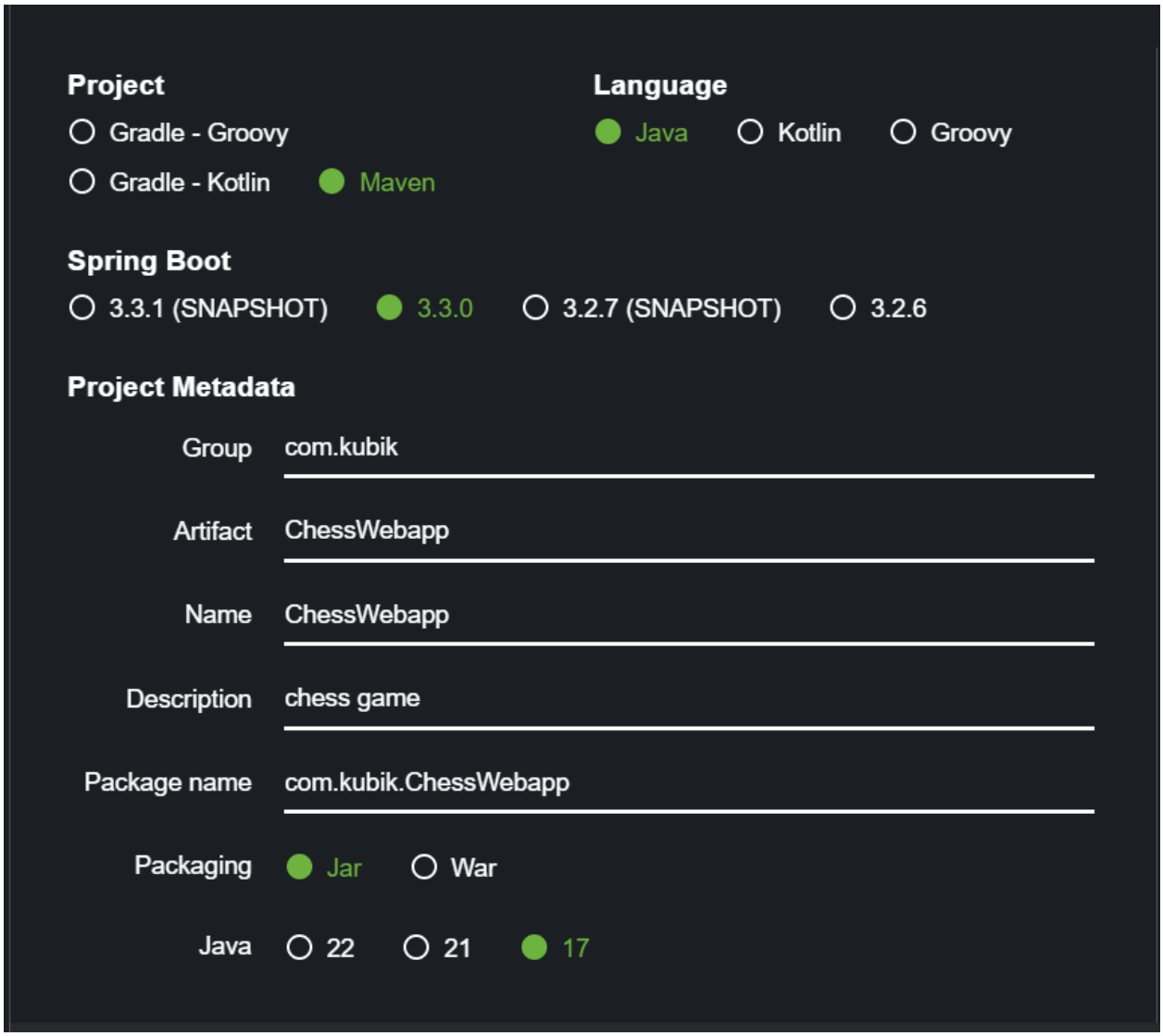
- **Тура** - друга за силою фігура на шахівниці. Ходить по горизонталі та вертикалі на будь-яку кількість клітин.
- **Кінь** - третя за силою фігура на шахівниці. Ходить буквою "Г": дві клітинки по вертикалі або горизонталі, і потім одну клітинку убік, перестрибуючи через інші фігури.
- **Слон** – третя за силою фігура на шахівниці. Ходить по діагоналі на будь-яку кількість клітин.
- **Королева** - найсильніша фігура на шахівниці. Ходить як тура і слон, по горизонталі, вертикалі та діагоналі на будь-яку кількість клітин.
- **Король** – найважливіша фігура на шахівниці, адже метою гри є його захист від атак фігур суперника задля попередження мату. Ходить на одну клітку у будь-якому напрямку.

Також існує кілька додаткових правил, наприклад, " рокіро́вка" - хід, в якому король і тура змінюються місцями, і "взяття на проході" - можливість для пішака атакувати пішака супротивника, який щойно зробив подвійний хід.

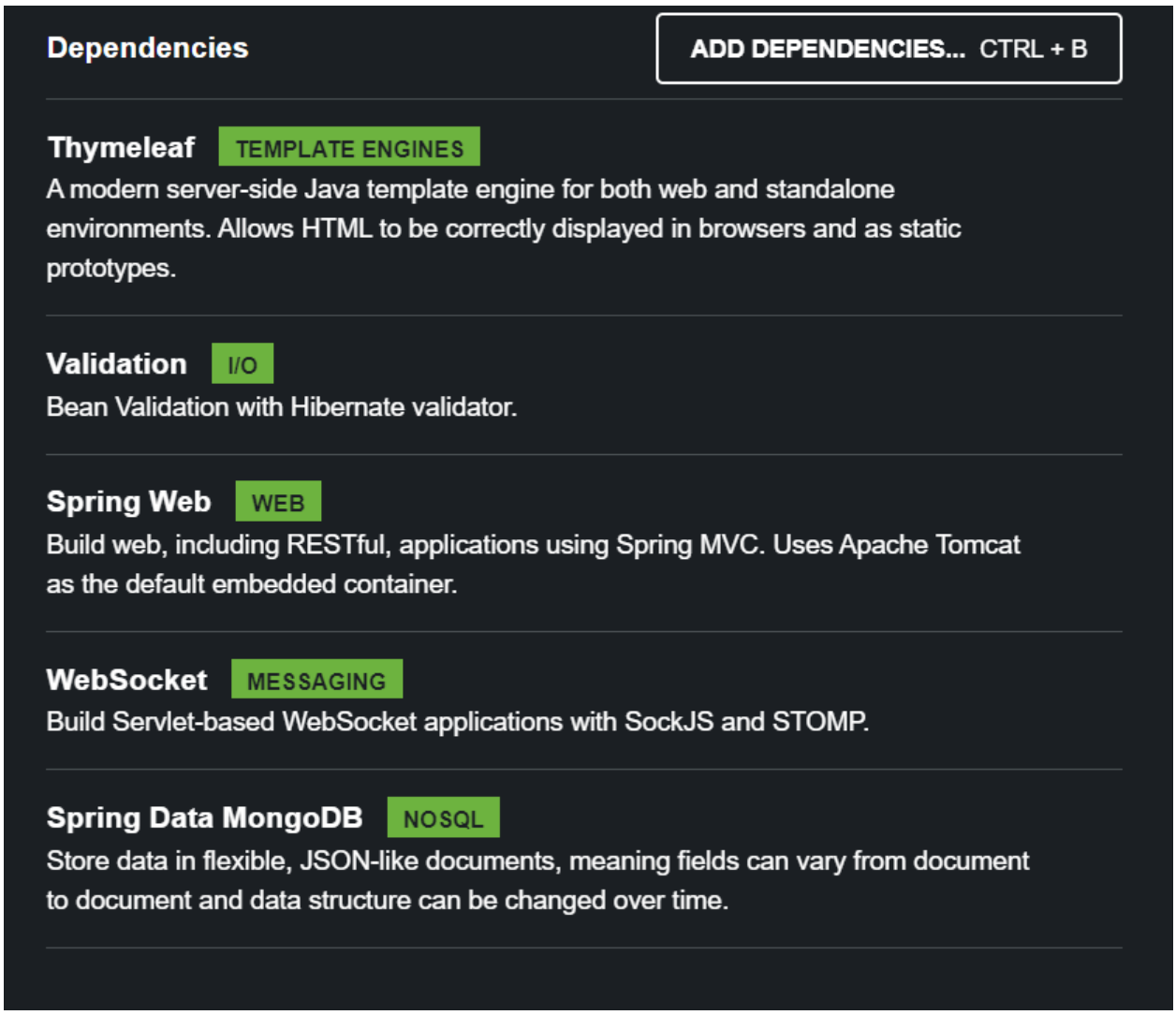
РОЗДІЛ ІІІ. ПРОГРАМНА ЧАСТИНА

3.1. Написання програми [3, 6-8]

Почнемо зі створення проєкту за допомогою Spring Initializr (https://start.spring.io). Обираємо мову java та версію 17, збірник проєктів maven, стабільну та новішу версію Spring Boot 3.3.0, назву проєкту ChessWebapp, Тип пакування jar:



Додаємо необхідні модулі: Spring Web, Spring Validation, Spring Boot, Spring Data MongoDB, WebSocket, Thymeleaf:



Скачуєм заготовку до проєкту та розархівовуємо в потрібний каталог. Відкриваємо проєкт в зручному середовищі розробки та бачимо головний клас ChessWebappApplication.java:

```
package com.kubik.ChessWebapp;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ChessWebappApplication {
    public static void main(String[] args) {
        SpringApplication.run(ChessWebappApplication.class,
args);
    }
}
```

Та файл конфігурації pom.xml із всіма залежностями (які ми вказували в Spring Initializr) та властивостями:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>3.3.0</version>
       <relativePath/>
    </parent>
    <groupId>com.kubik</groupId>
    <artifactId>ChessWebapp</artifactId>
    <version>1.0.0</version>
    <name>ChessWebapp</name>
    <description>chess game</description>
    <properties>
       <java.version>17</java.version>
    </properties>
    <dependencies>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-thymeleaf</artifactId>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-validation</artifactId>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-web</artifactId>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-websocket</artifactId>
       </dependency>
       <dependency>
          <groupId>org.projectlombok</groupId>
          <artifactId>lombok</artifactId>
          <optional>true</optional>
       </dependency>
       <dependency>
          <groupId>org.modelmapper</groupId>
          <artifactId>modelmapper</artifactId>
          <version>3.2.0</version>
       </dependency>
       <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

Додаємо властивості в application.properties щоб Spring міг підключитись до бази даних:

```
spring.application.name=ChessWebapp
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=chess
spring.mongodb.embedded.storage.oplogSize=1024
spring.main.allow-bean-definition-overriding=true
spring.autoconfigure.exclude=org.springframework.boot.autoconfig
ure.mongo.embedded.EmbeddedMongoAutoConfiguration
```

Пишемо об’єкт гравця який буде зв’язуватись із базою даних (мапитись):

```
@Data
@Document(collection = "chessUsers")
public class ChessUser {
    @Id
    private String id;
    private String email;
```

```
    private String nickname;
    private String password;
}
```

Напишемо DTO (Data Transfer Object) Для передачі об’єкту на веб:

```
@Data
public class ChessUserDto {
    private Long id;
        @Email(message = "Invalid email")
    private String email;
        @Size(min = 3, message = "Cannot be less that 3")
    private String nickname;
        @Size(min = 3, message = "Cannot be less that 3")
    private String password;
    private String confirmPassword;
}
```

Опишемо методи для роботи з базою даних юзерів:

```
public interface ChessUserRepository extends
MongoRepository<ChessUser, Long> {
    ChessUser findByNickname(String nickname);
}
```

Опишемо сервісний клас у якому буде відбуватися основна бізнес-логіка входу на реєстрації користувачів у вебзастосунку:

```
@Service
@RequiredArgsConstructor
public class ChessUserService {
    private final ChessUserRepository chessUserRepository;
    private final ModelMapper modelMapper;

    public String createUser(ChessUserDto chessUserDto) {
        return
chessUserRepository.save(modelMapper.map(chessUserDto,
ChessUser.class)).getId();
    }

    public ChessUser getUserByNickname(String nickname) {
        return chessUserRepository.findByNickname(nickname);
    }

    public List<ChessUser> getAllUsers() {
        return (List<ChessUser>) chessUserRepository.findAll();
    }
}
```

Опишемо також сервіс у якому буде відбуватися основна логіка гри:

```
@Service
@RequiredArgsConstructor
public class ChessGameService {
    private final BoardRepository boardRepository;

    public Board createGame(ChessUser user) {
        Board board = new Board();
        board.setFirstChessPlayer(user);
        boardRepository.save(board);
        return board;
    }

    public Board connectToRandom(ChessUser user) {
        Optional<Board> optionalGame =
boardRepository.findFirstBySecondChessPlayerIsNullAndBoardStatus
(BoardStatus.NEW);
        optionalGame.orElseThrow(() -> new RuntimeException("All
games are occupied"));
        Board board = optionalGame.get();
        board.setSecondChessPlayer(user);
        board.setBoardStatus(BoardStatus.IN_PROGRESS);
        boardRepository.save(board);
        return board;
    }

    public Board connectToSpecific(ChessUser user, String
gameId) {
        Optional<Board> optionalGame =
boardRepository.findById(gameId);
        optionalGame.orElseThrow(() -> new
RuntimeException("There is no game with provided id"));
        Board board = optionalGame.get();
        board.setSecondChessPlayer(user);
        board.setBoardStatus(BoardStatus.IN_PROGRESS);
        boardRepository.save(board);
        return board;
    }

    public Board getGame(String gameId) {
        Optional<Board> optionalGame =
boardRepository.findById(gameId);
        optionalGame.orElseThrow(() -> new RuntimeException("All
games are occupied"));
        return optionalGame.get();
    }

    public Board move(Move move) {
        Optional<Board> optionalGame =
boardRepository.findById(move.getGameId());
        optionalGame.orElseThrow(() -> new
RuntimeException("There is no game with provided id"));
```

```
        Board board = optionalGame.get();
        String squareIndex = move.getSquareIndex();

board.mouseClick(Character.getNumericValue(squareIndex.charAt(0)
), Character.getNumericValue(squareIndex.charAt(1)));
        GameResult gameResult = board.checkGameOver();
        if (gameResult != null) {
            board.setGameResult(gameResult);
            board.setBoardStatus(BoardStatus.FINISHED);
        }
        boardRepository.save(board);
        return board;
    }
}
```

Напишемо класи (контролери) для написання API до якого будуть звертатись користувачі із браузера.

- Контролер IndexController для редіректу із корневого URL на URL-адрес /login:

```
@Controller
public class IndexController {
    @GetMapping("/")
    public String index() {
        return "redirect:/login";
    }
}
```

- Контролер LoginController де відбувається перенаправлення на сторінку реєстрації, сторінку невдалого входу та головну сторінку у разу успішного входу:

```
@Controller
@RequiredArgsConstructor
public class LoginController {
    private final ChessUserService chessUserService;

    @GetMapping("/login")
    public String showWelcomePage(Model model) {
        return "login";
    }

    @GetMapping("/registration")
    public String showRegistration(ChessUserDto chessUser, Model
model) {
        return "registration";
    }

    @PostMapping("/main-page")
    public String logIn(ChessUserDto chessUser, BindingResult
bindingResult, Model model) {
```

```
        ChessUser userFromDB =
chessUserService.getUserByNickname(chessUser.getNickname());
        if (userFromDB == null ||
!userFromDB.getPassword().equals(chessUser.getPassword())) {
            bindingResult.rejectValue("password",
"error.password", "Invalid password");
            return "login-failed";
        }
        model.addAttribute("user", userFromDB);
        return "main-page";
    }
}
```

- Контролер RegistrationController потрібен у випадку якщо користувач буде реєструватися на сайті:

```
@Controller
@RequiredArgsConstructor
public class RegistrationController {
    private final ChessUserService chessUserService;

    @PostMapping("/registration-result")
    public String registerUser(@Valid ChessUserDto chessUser,
BindingResult bindingResult, Model model) {
        if (bindingResult.hasErrors()) {
            return "registration";
        }
        if
(!chessUser.getPassword().equals(chessUser.getConfirmPassword())
) {
            bindingResult.rejectValue("confirmPassword",
"error.user", "Passwords do not match");
            return "registration";
        }
        chessUserService.createUser(chessUser);
        model.addAttribute("user", chessUser);
        return "registration-successful";
    }
}
```

- Контролер GameController який керує шахівницею та взаємодію користувачів з нею:

```
@Controller
@AllArgsConstructor
public class GameController {
    private final ChessGameService chessGameService;
    private final ChessUserService chessUserService;
    private final SimpMessagingTemplate simpMessagingTemplate;

    @PostMapping("/board-create")
```

```
    public String createBoard(@RequestBody String gameId, Model
model) {
        model.addAttribute("board",
chessGameService.getGame(gameId));
        return "fragments/game-page :: game-board";
    }

    @PostMapping("/game-create-json")
    public ResponseEntity<Board> createGame(@RequestBody
ChessPlayerDto chessPlayer) {
        ChessUser userByNickname =
chessUserService.getUserByNickname(chessPlayer.getNickname());
        return
ResponseEntity.ok(chessGameService.createGame(userByNickname));
    }

    @PostMapping("/connect-random-json")
    public ResponseEntity<Board> connectRandom(@RequestBody
ChessPlayerDto chessPlayer) throws RuntimeException {
        ChessUser userByNickname =
chessUserService.getUserByNickname(chessPlayer.getNickname());
        return
ResponseEntity.ok(chessGameService.connectToRandom(userByNicknam
e));
    }

    @PostMapping("/connect-json")
    public ResponseEntity<Board> connectSpecific(@RequestBody
GameConnect connect) throws RuntimeException {
        ChessUser userByNickname =
chessUserService.getUserByNickname(connect.getPlayer().getNickna
me());
        return
ResponseEntity.ok(chessGameService.connectToSpecific(userByNickn
ame, connect.getGameId()));
    }


    @PostMapping("/game-move")
    public ResponseEntity<Board> move(@RequestBody Move move)
throws RuntimeException {
        Board board = chessGameService.move(move);
        simpMessagingTemplate.convertAndSend("/topic/game-
progress/" + board.getId(), board);
        return ResponseEntity.ok(board);
    }
}
```

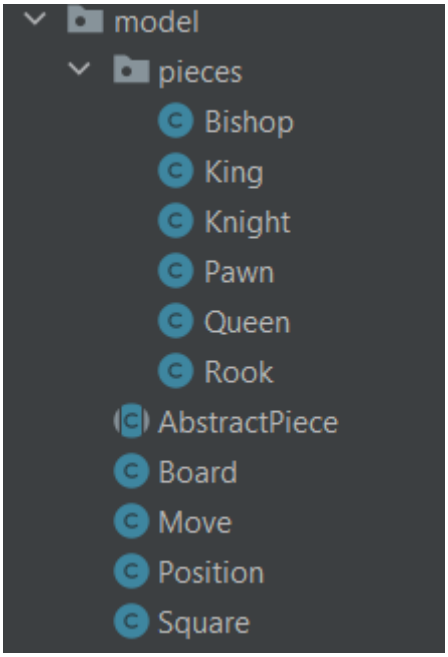
Для онлайн гри використаємо WebSocket, SockJS та STOMP, напишемо конфігурацію WebSocket:

```java
@Configuration
@EnableWebSocketMessageBroker
public class WebsocketConfig implements
WebSocketMessageBrokerConfigurer {
    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry)
{
        registry.addEndpoint("/move").withSockJS();

    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry)
{
        registry.enableSimpleBroker("/topic");
    }
}
```

Щоб реалізовувати логіку /move слід переписати логіку існуючої гри шахи (із кваліфікаційної роботи) з мови Python на Java:



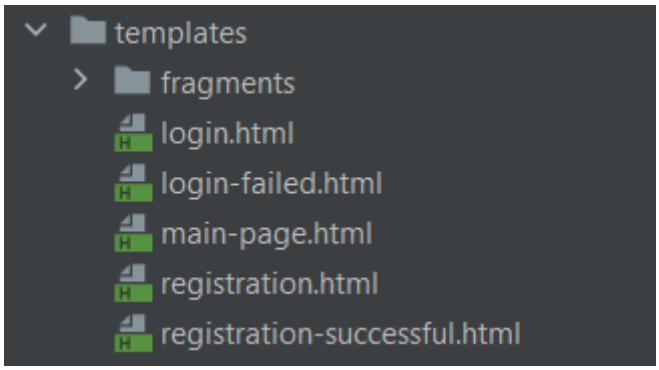
Напишемо JavaScript код із використанням Jquery, SockJS, та STOMP:

```javascript
function connectToSocket(gameId) {
    let socket = new SockJS(url + "/move");
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function (frame) {
        stompClient.subscribe("/topic/game-progress/" + gameId,
function (response) {
            let board = JSON.parse(response.body);
            console.log(board);
            refreshChessBoard(board);
        })
```

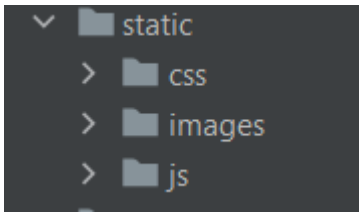
```
    })
}
```

В результаті за допомогою JQuery клієнт та сервер можуть зручно обмінюватися у вигляді JSON файлів а за допомгою SockJS, та STOMP це відбувається динамічно

Напишемо веб сторінки щоб перевірити чи наші контроллери працюють корректно:

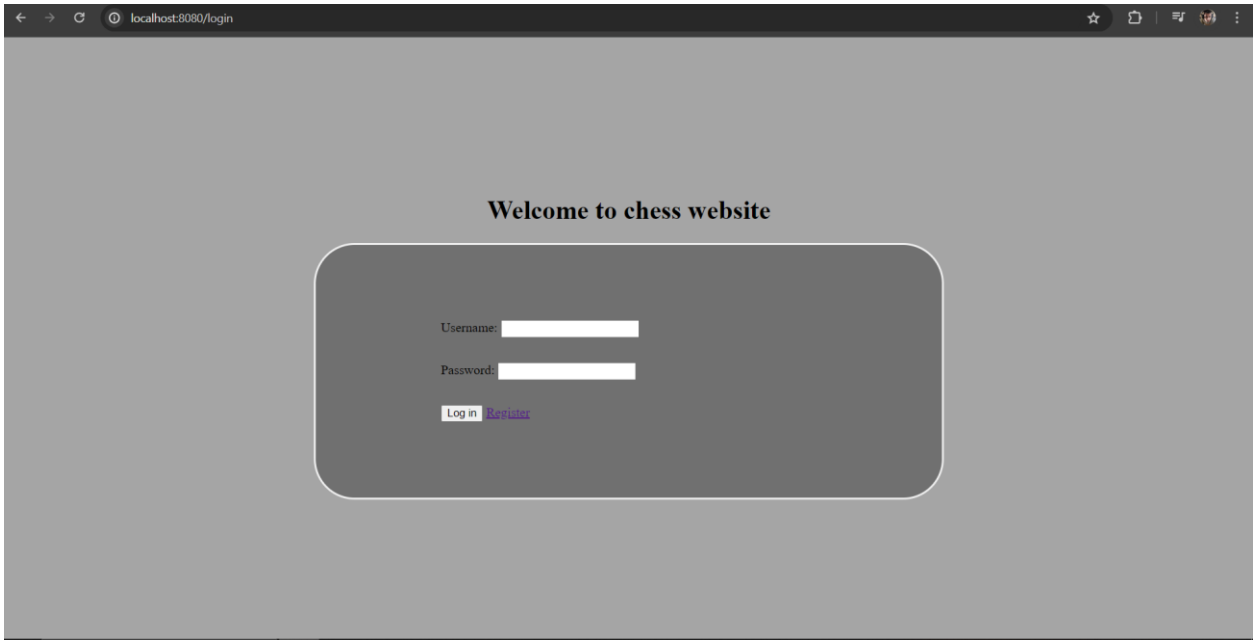


Також напишемо статичні файли типу стилів, картинок та скриптів:



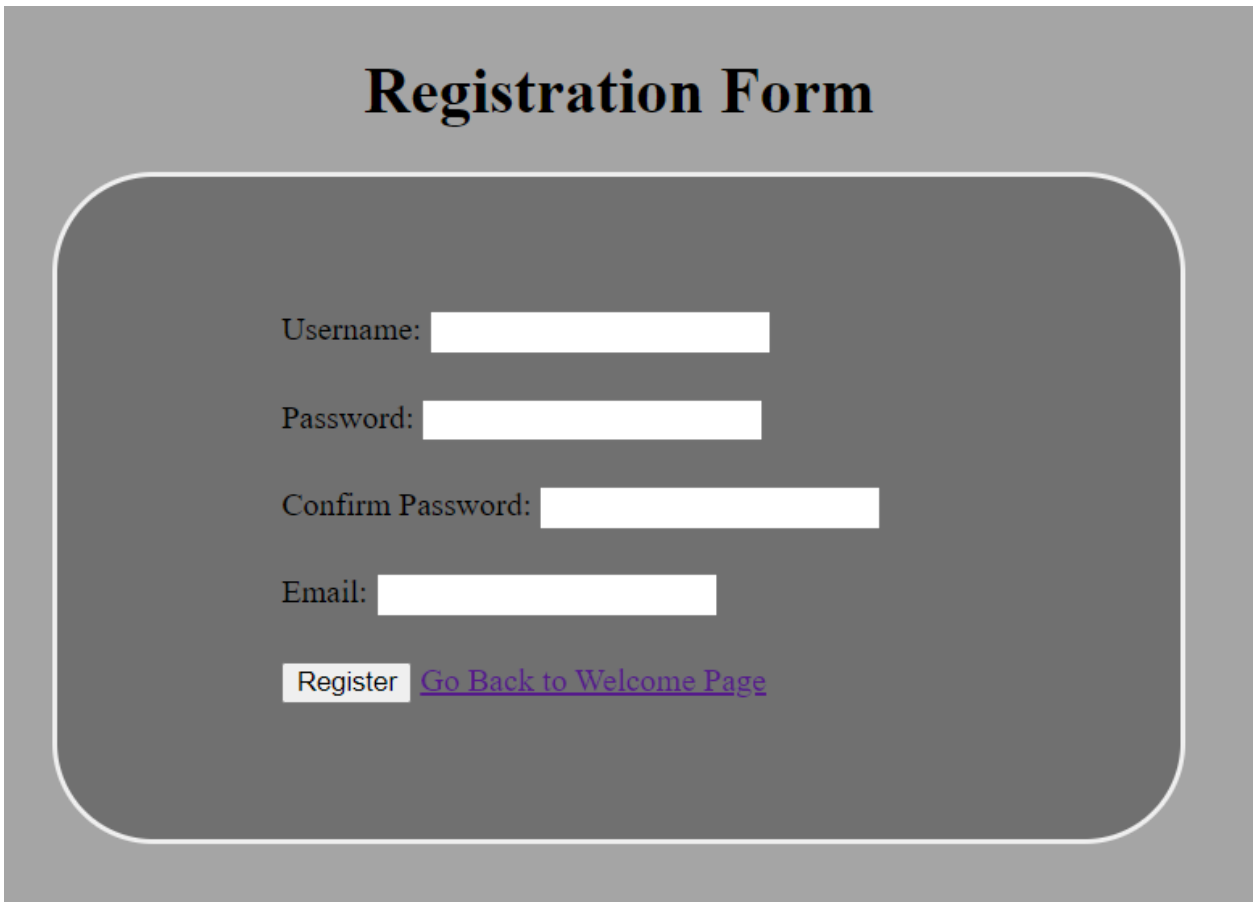
3.2. Демонстрація готового проєкту

Сторінка входу:

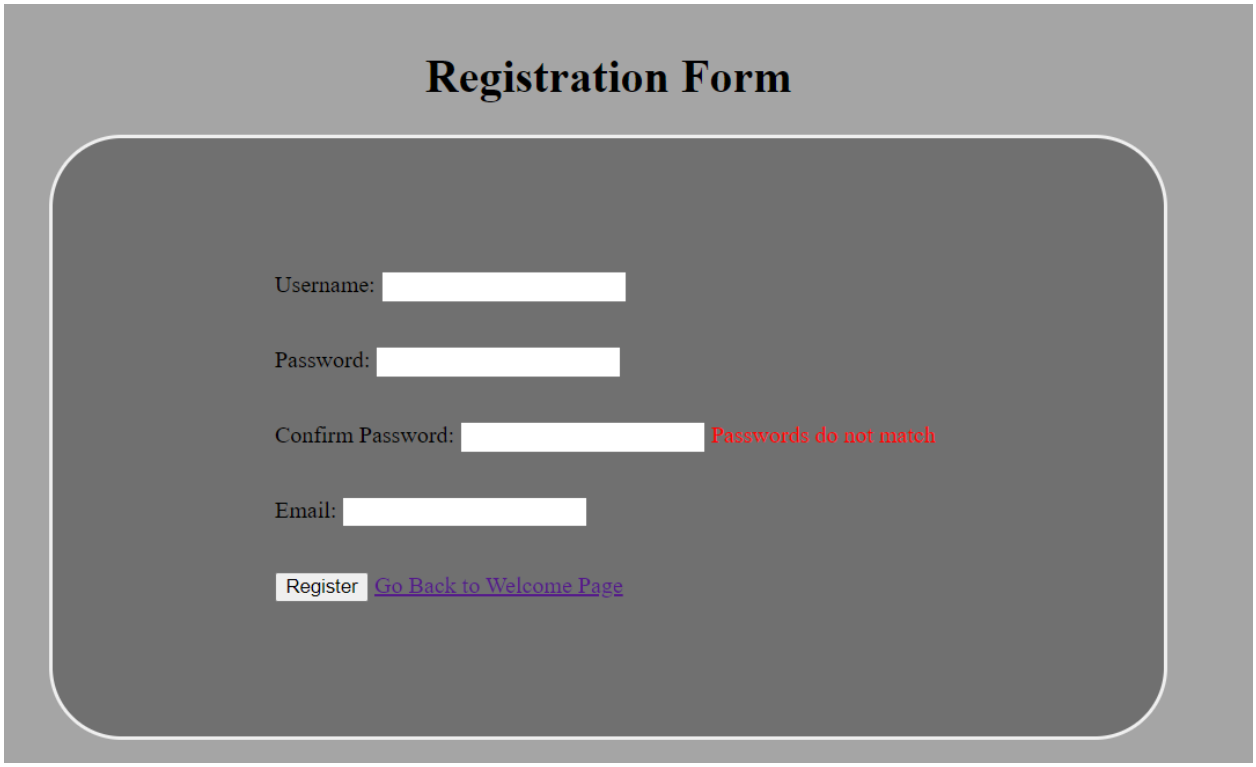


У разі незареєстрованого користувача отримуємо попередження:

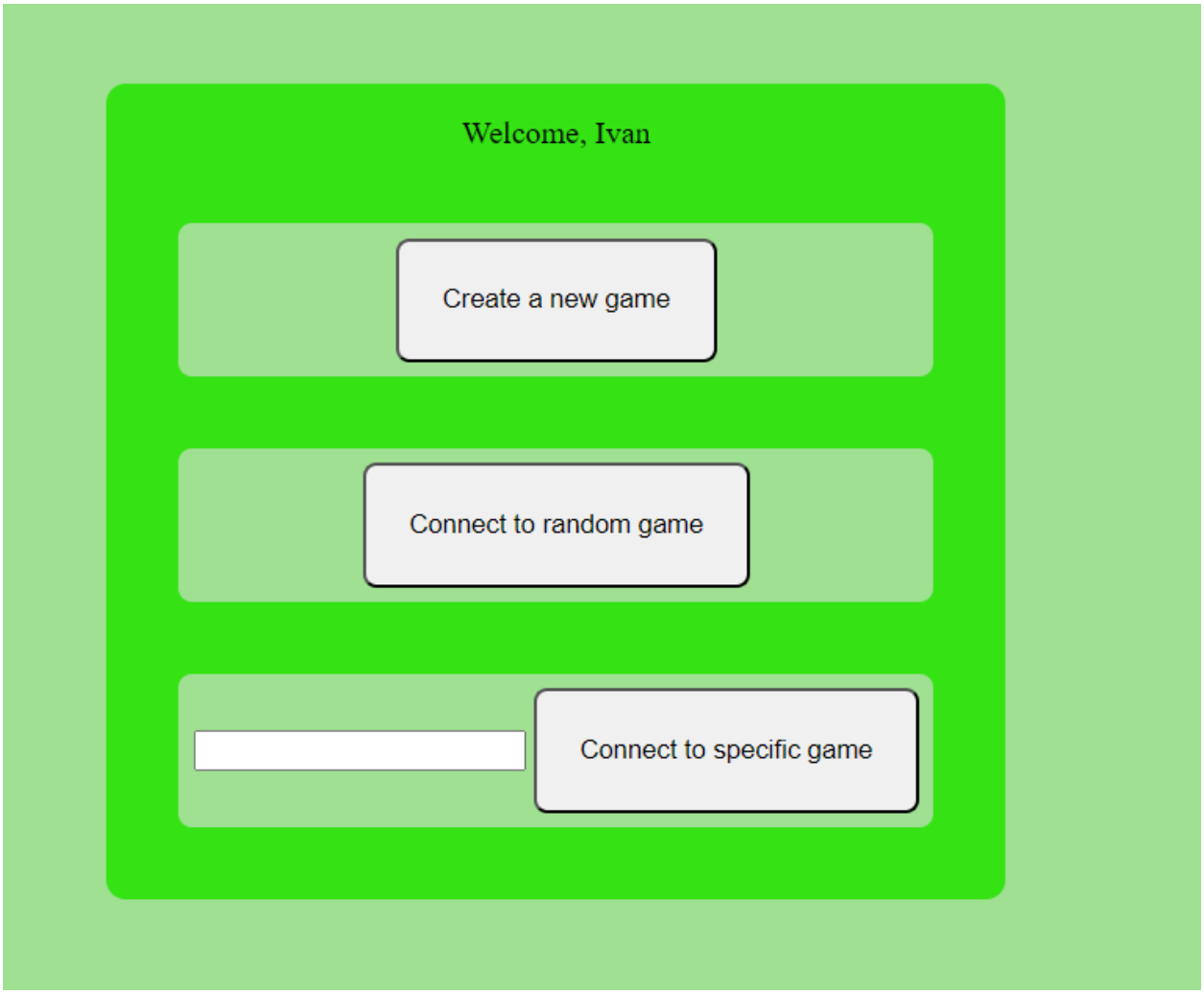
Сторінка реєстрації:



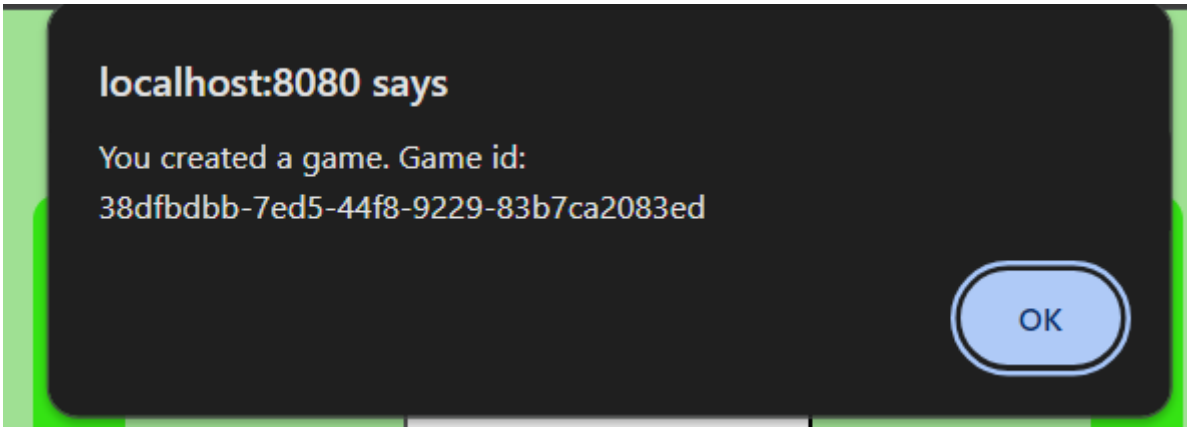
У разі реєстрації отримуємо попередження у випадку непроходження валідації:

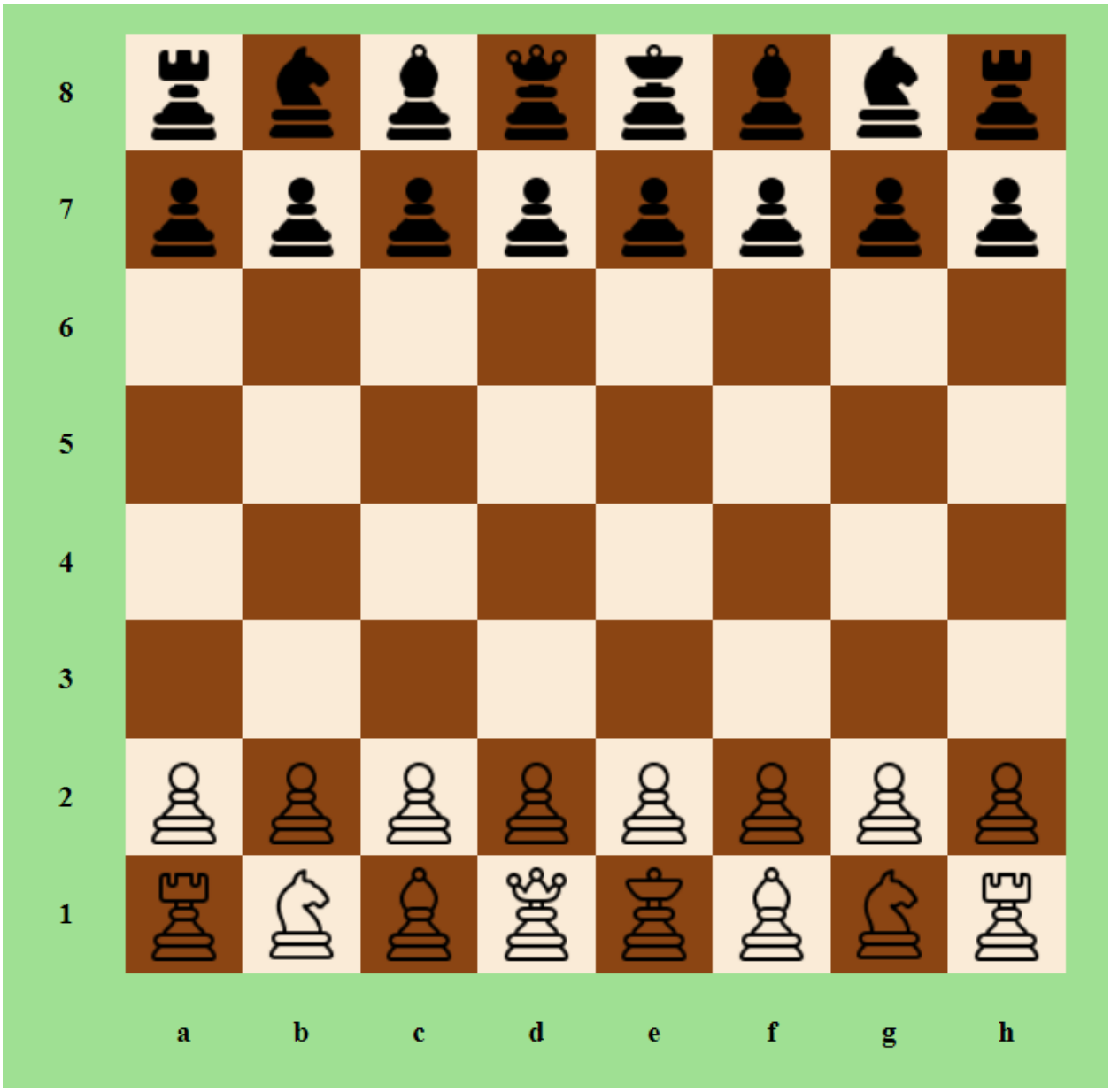


У разі успішного входу потрапляємо на головну сторінку де користувач може створити гру або під’єднатися до існуючої:

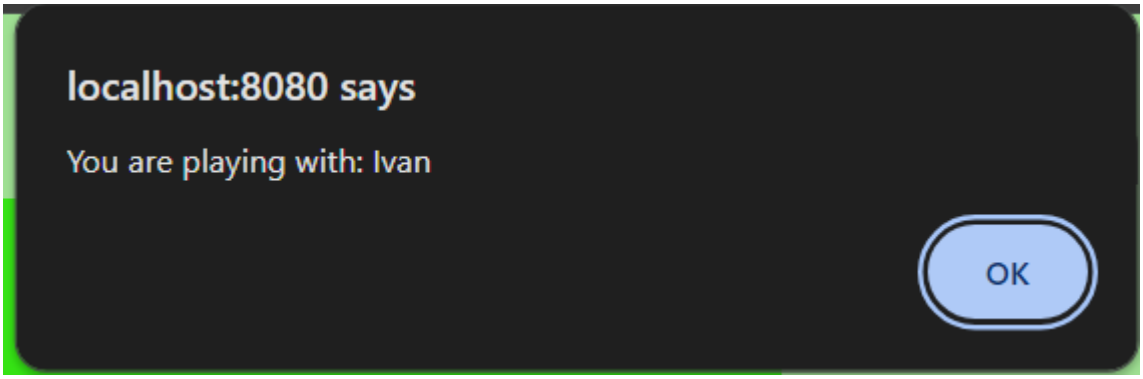


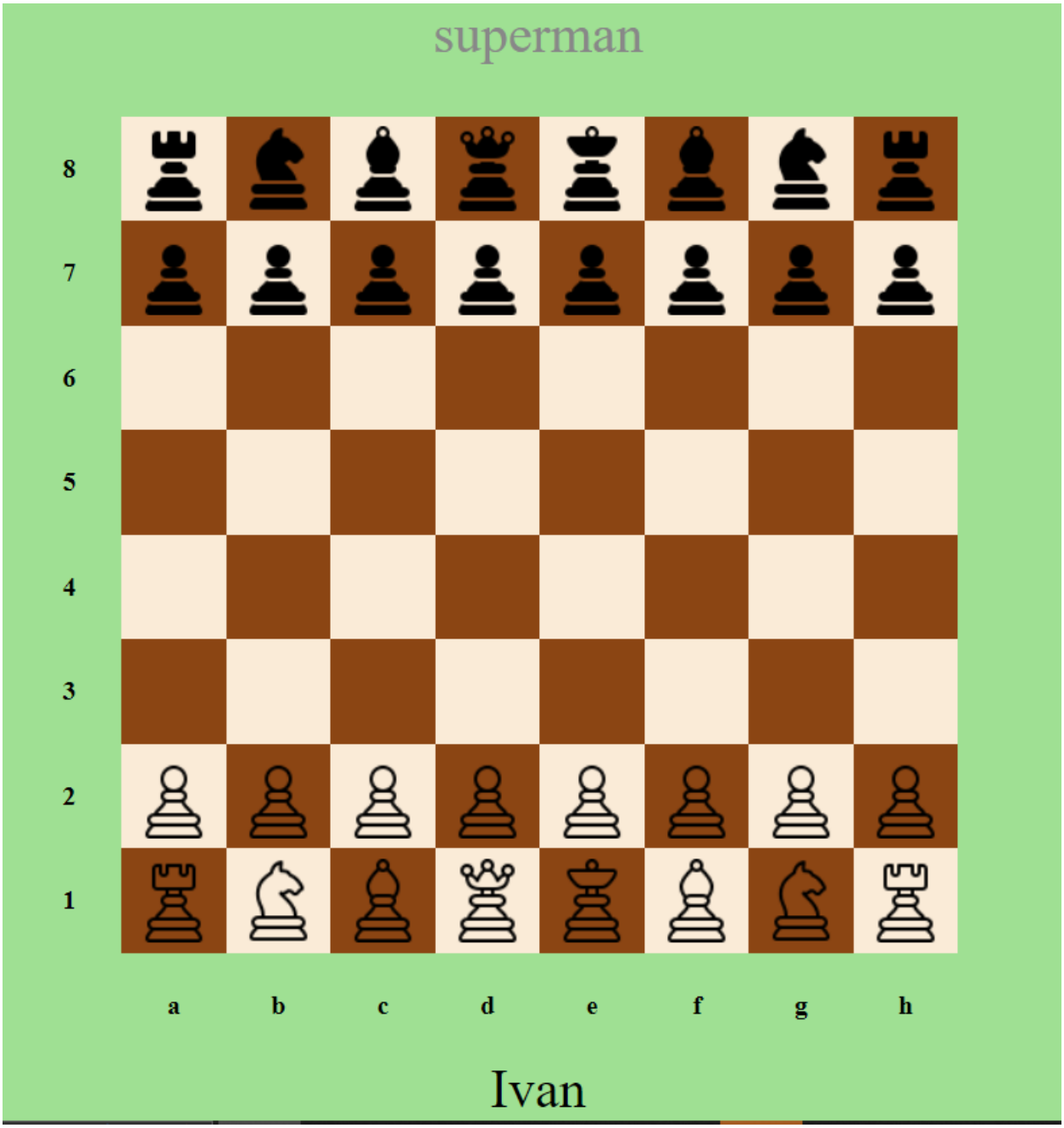
Створюємо гру, отримуємо унікальний ідентифікатор який можемо передати супернику для гри:



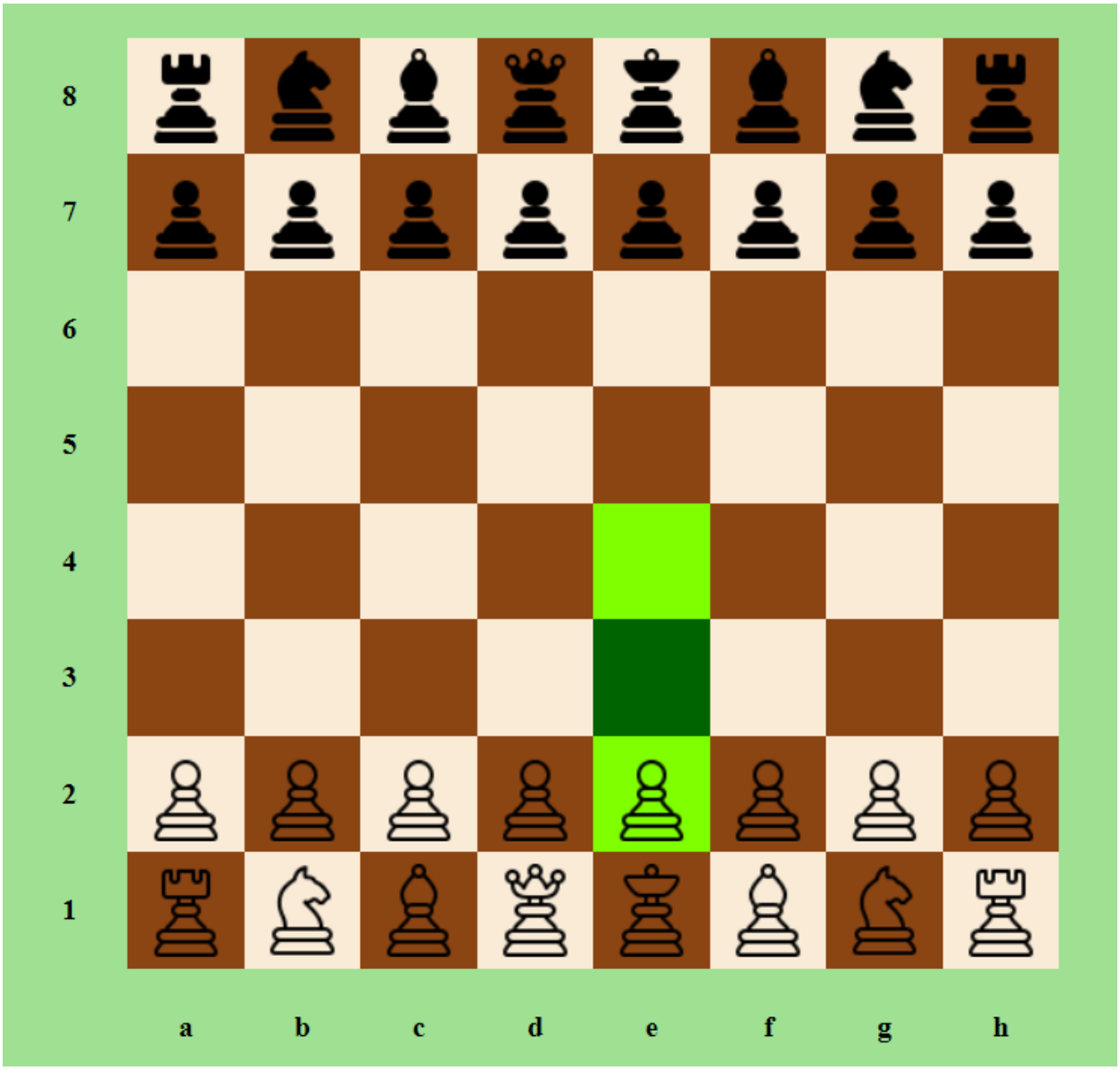


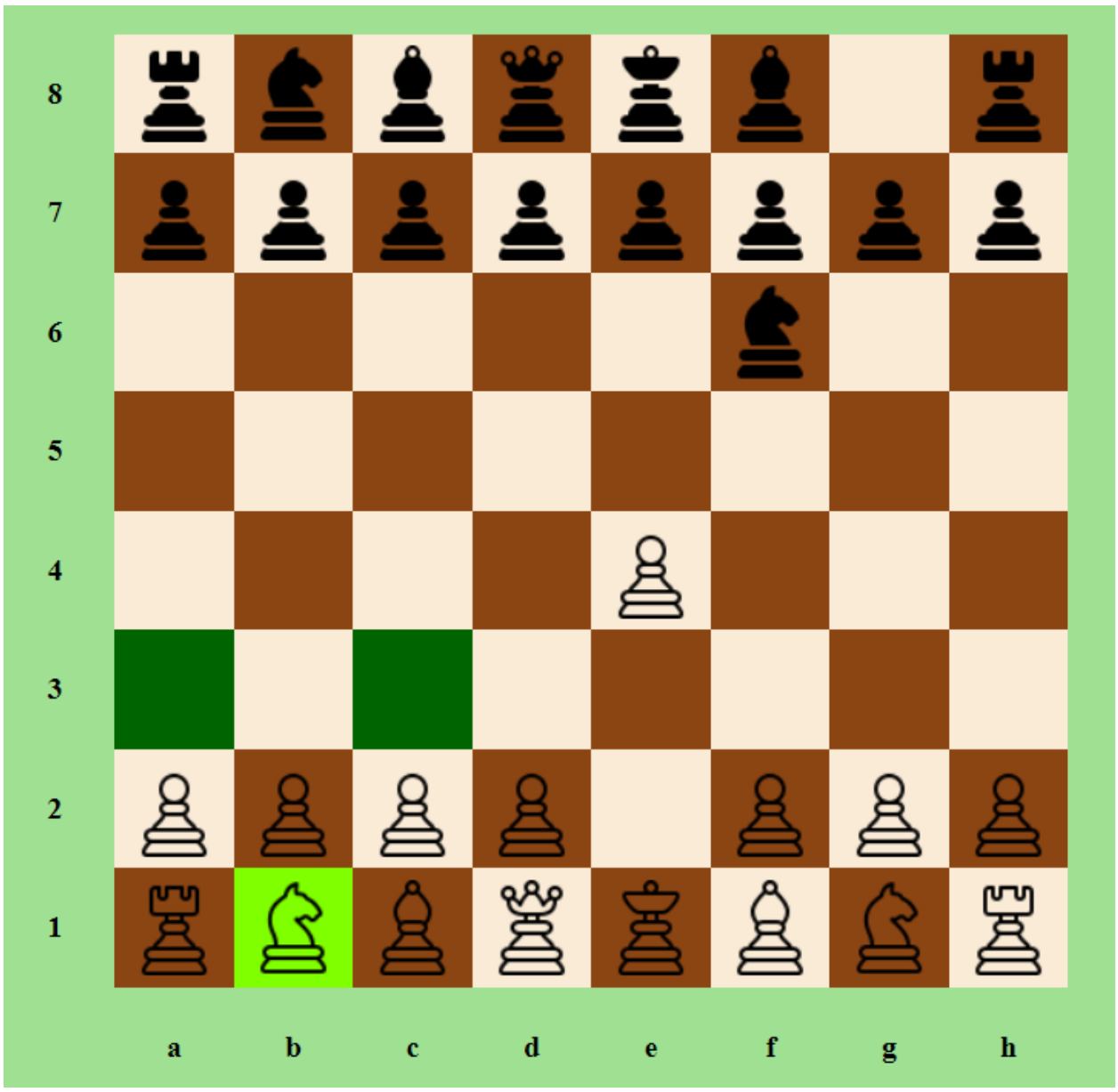
Соперник який введе код гри побачить наступне:

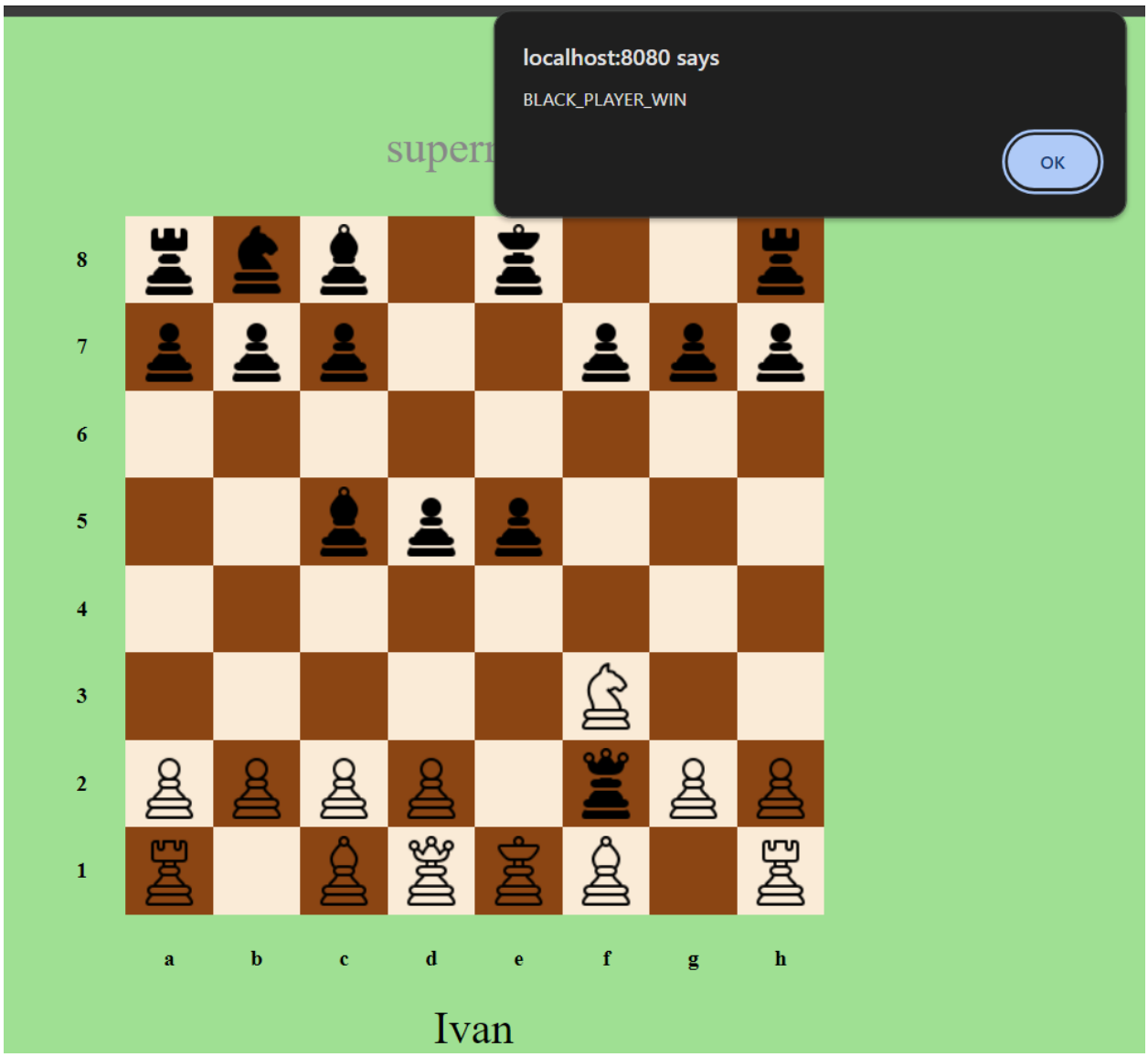




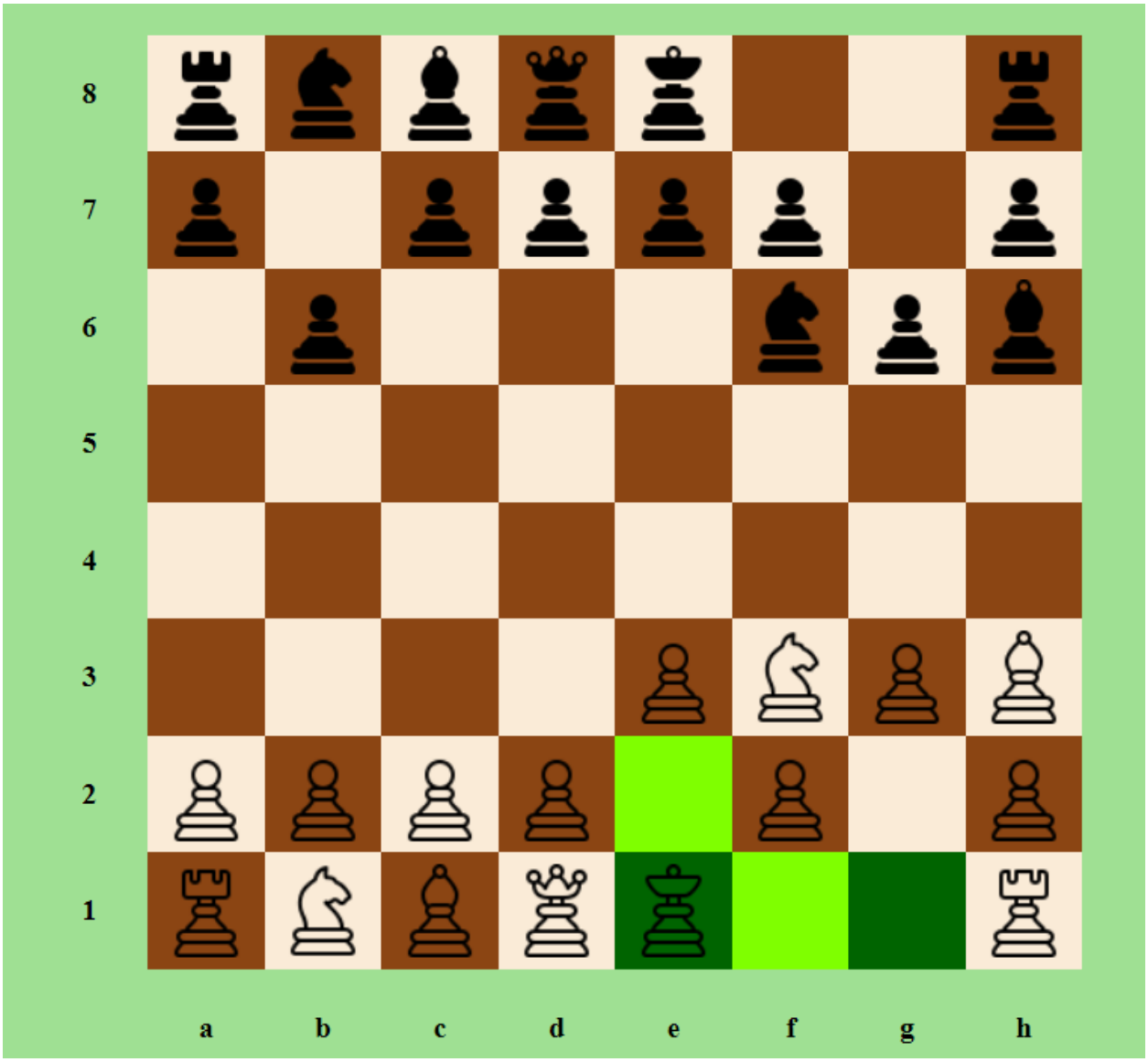
Шахівниця підчас гри:


8
7
6
5
4
3
2
1
a
b
c
d
e
f
g
h

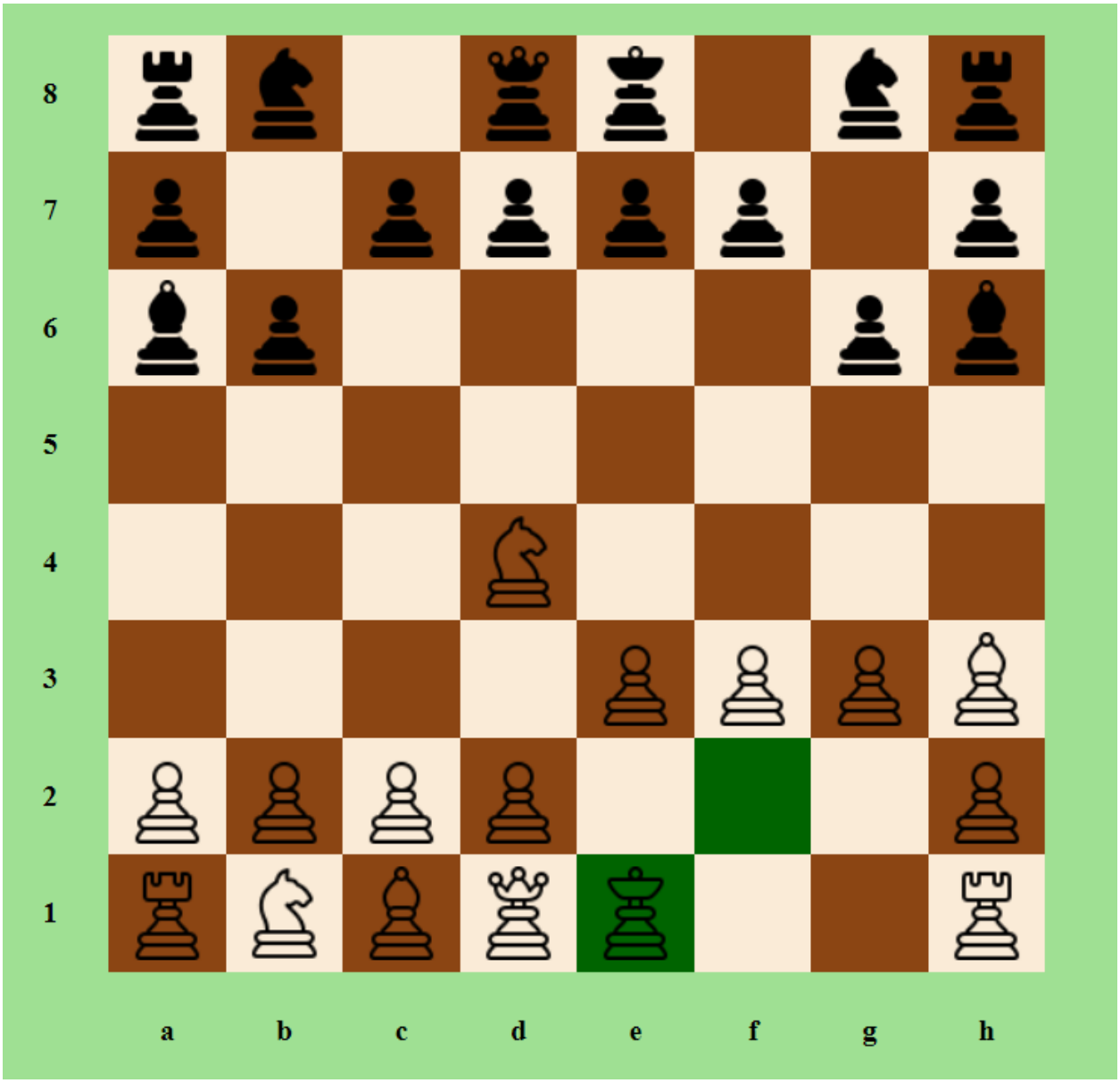

8
7
6
5
4
3
2
1
a
b
c
d
e
f
g
h



Можливість рокіровки:



Та запобігання короля від здійснення рокіровки:


8
7
6
5
4
3
2
1
a b c d e f g h

ВИСНОВКИ

- У процесі виконання кваліфікаційної роботи здійснено дослідження можливостей фреймворку Spring для створення гри “шахи”. Розглянуто основні поняття, модулі та методи застосувань, роботу із базами даних, валідація даних, написання сайту.
- Здійснено написання логіки гри в шахи мовою jav ата взаємодії веб частини застосунку і серверної частини, динамічну обробку ігрових подій на стороні клієнта.
- Для створення гри були використані модулі Spring: SpringWeb, SpringValidation, SpringBoot, WebSocket, Thymeleaf.
- У процесі розробки, набуто практичних навичок та знання можливостей фреймворку Spring та його використання для розробки веб застосунку.
- Результатом кваліфікаційної роботи є браузерна гра шахи

Перелік посилань

1. Java Tutorials [Електронний ресурс]. Режим доступу: https://javarush.com/ (дата звернення 15.10.2023);
2. Spring Documentation [Електронний ресурс]. Режим доступу: https://spring.io (дата звернення 23.12.2023);

3. Spring framework [Електронний ресурс]. Режим доступу: https://www.baeldung.com (дата звернення 17.01.2024);

4. Java Docs [Електронний ресурс]. Режим доступу: https://docs.oracle.com/javase/8/docs/api/ (дата звернення 5.11.2023);
5. Вивчення принципів гри “шахи” [Електронний ресурс]. Режим доступу до ресурсу: https://www.chess.com/uk (дата звернення: 15.10.2023);

6. MongoDB Documentation [Електронний ресурс]. Режим доступу: https://www.mongodb.com/docs (дата звернення 01.02.2024);

7. Using with SockJS [Електронний ресурс]. Режим доступу: https://stomp-js.github.io/ng2-stompjs/additional-documentation/sock-js.html (дата звернення 07.02.2024);

8. Using STOMP JS [Електронний ресурс]. Режим доступу: https://stomp-js.github.io/stomp-websocket/codo/extra/docs-src/Usage.md.html (дата звернення 07.02.2024);

Додатки

ChessConfig.java

```
package com.kubik.ChessWebapp.config;

import org.modelmapper.ModelMapper;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ChessConfig {
    @Bean
    public ModelMapper modelMapper() {
        return new ModelMapper();
    }
}
```

CorsConfig.java

```
package com.kubik.ChessWebapp.config;

import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.CorsRegistry;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigur
er;

@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("http://localhost:8080")
                .allowedMethods("*")
                .allowedHeaders("*");
    }
}
```

WebsocketConfig.java

```
package com.kubik.ChessWebapp.config;

import org.springframework.context.annotation.Configuration;
import
org.springframework.messaging.simp.config.MessageBrokerRegistry;
import
```

```
org.springframework.web.socket.config.annotation.EnableWebSocket
MessageBroker;
import
org.springframework.web.socket.config.annotation.StompEndpointRe
gistry;
import
org.springframework.web.socket.config.annotation.WebSocketMessag
eBrokerConfigurer;

@Configuration
@EnableWebSocketMessageBroker
public class WebsocketConfig implements
WebSocketMessageBrokerConfigurer {
    @Override
    public void registerStompEndpoints(StompEndpointRegistry
registry) {

registry.addEndpoint("/move").setAllowedOrigins("http://localhos
t:8080").withSockJS();

    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry
registry) {
        registry.enableSimpleBroker("/topic");
    }
}
```

ChessPlayerDto.java

```
package com.kubik.ChessWebapp.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class ChessPlayerDto {
    private String nickname;
}
```

ChessUserDto.java

```
package com.kubik.ChessWebapp.dto;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.Size;
```

```java
import lombok.Data;

@Data
public class ChessUserDto {
    private Long id;
        @Email(message = "Invalid email")
    private String email;
        @Size(min = 3, message = "Cannot be less that 3")
    private String nickname;
        @Size(min = 3, message = "Cannot be less that 3")
    private String password;
    private String confirmPassword;
}
```

GameConnect.java

```java
package com.kubik.ChessWebapp.dto;

import lombok.Data;

@Data
public class GameConnect {
    ChessPlayerDto player;
    String gameId;
}
```

ChessUser.java

```java
package com.kubik.ChessWebapp.entity;

import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Data
@Document(collection = "chessUsers")
public class ChessUser {
    @Id
    private String id;
    private String email;
    private String nickname;
    private String password;

}
```

Bishop.java

```
package com.kubik.ChessWebapp.model.pieces;

import com.kubik.ChessWebapp.model.AbstractPiece;
import com.kubik.ChessWebapp.model.Board;
import com.kubik.ChessWebapp.model.Position;
import com.kubik.ChessWebapp.model.Square;
import com.kubik.ChessWebapp.statics.PlayerColor;

import java.util.ArrayList;
import java.util.List;

public class Bishop extends AbstractPiece {
    public Bishop(Position position, PlayerColor color) {
        super(position, color);
        this.setImgPath("images/" +
color.toString().toLowerCase().charAt(0) + "_bishop.png");
        this.setName("B");
    }

    @Override
    public List<List<Square>> getPossibleMoves(Board board) {
        List<List<Square>> possibleMoves = new ArrayList<>();
        List<Square> direction = new ArrayList<>();
        for (int i = 1; i < 8; i++) {
            if (this.getPosition().getX() + i > 7 ||
this.getPosition().getY() - i < 0) {
                break;
            }
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX() + i,
this.getPosition().getY() - i)));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int i = 1; i < 8; i++) {
            if (this.getPosition().getX() + i > 7 ||
this.getPosition().getY() + i > 7) {
                break;
            }
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX() + i,
this.getPosition().getY() + i)));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int i = 1; i < 8; i++) {
            if (this.getPosition().getX() - i < 0 ||
this.getPosition().getY() + i > 7) {
                break;
            }
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX() - i,
this.getPosition().getY() + i)));
```

```
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int i = 1; i < 8; i++) {
            if (this.getPosition().getX() - i < 0 ||
this.getPosition().getY() - i < 0) {
                break;
            }
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX() - i,
this.getPosition().getY() - i)));
        }
        possibleMoves.add(new ArrayList<>(direction));
        return possibleMoves;
    }

    @Override
    public void specificMove(Square square, Board board, Square
prevSquare) {

    }
}
```

King.java

```
package com.kubik.ChessWebapp.model.pieces;

import com.kubik.ChessWebapp.model.AbstractPiece;
import com.kubik.ChessWebapp.model.Board;
import com.kubik.ChessWebapp.model.Position;
import com.kubik.ChessWebapp.model.Square;
import com.kubik.ChessWebapp.statics.PlayerColor;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class King extends AbstractPiece {

    public King(Position position, PlayerColor color) {
        super(position, color);
        this.setImgPath("images/" +
color.toString().toLowerCase().charAt(0) + "_king.png");
        this.setName("K");
    }

    @Override
    public List<List<Square>> getPossibleMoves(Board board) {
        List<List<Square>> possibleMoves = new ArrayList<>();
        int[][] moves = {{0, -1}, {1, -1}, {1, 0}, {1, 1}, {0,
1}, {-1, 1}, {-1, 0}, {-1, -1}};
```

```
        for (int[] move : moves) {
            int newX = this.getPosition().getX() + move[0];
            int newY = this.getPosition().getY() + move[1];
            if (newX >= 0 && newX < 8 && newY >= 0 && newY < 8)
{

possibleMoves.add(Collections.singletonList(board.getSquareFromP
os(new Position(newX, newY))));
            }
        }
        return possibleMoves;
    }

    public boolean canCastleLeft(Board board) {
        if (!this.isHasMoved()) {
            if (this.getColor() == PlayerColor.WHITE_PLAYER) {
                AbstractPiece leftRook =
board.getSquareFromPos(new Position(0, 7)).getOccupyingPiece();
                if (leftRook != null && !leftRook.isHasMoved())
{
                    boolean pathClear = true;
                    for (int i = 1; i <= 3; i++) {
                        if (board.getSquareFromPos(new
Position(i, 7)).getOccupyingPiece() != null) {
                            pathClear = false;
                            break;
                        }
                    }
                    if (pathClear) {
                        List<AbstractPiece> pieces = new
ArrayList<>();
                        for (Square square : board.getSquares())
{
                            if (square.getOccupyingPiece() !=
null && square.getOccupyingPiece().getColor() ==
PlayerColor.BLACK_PLAYER) {

pieces.add(square.getOccupyingPiece());
                            }
                        }
                        for (AbstractPiece piece : pieces) {
                            List<Square> squares =
piece.getMoves(board);
                            for (Square square : squares) {
                                for (int i = 1; i <= 3; i++) {
                                    if
(square.getPosition().equals(new Position(i, 7))) {
                                        return false;
                                    }
                                }
                            }
                        }
                        return true;
```

```
                    }
                }
            } else if (this.getColor() ==
PlayerColor.BLACK_PLAYER) {
                AbstractPiece leftRook =
board.getSquareFromPos(new Position(0, 0)).getOccupyingPiece();
                if (leftRook != null && !leftRook.isHasMoved())
{
                    boolean pathClear = true;
                    for (int i = 1; i <= 3; i++) {
                        if (board.getSquareFromPos(new
Position(i, 0)).getOccupyingPiece() != null) {
                            pathClear = false;
                            break;
                        }
                    }
                    if (pathClear) {
                        List<AbstractPiece> pieces = new
ArrayList<>();
                        for (Square square : board.getSquares())
{
                            if (square.getOccupyingPiece() !=
null && square.getOccupyingPiece().getColor() ==
PlayerColor.WHITE_PLAYER) {

pieces.add(square.getOccupyingPiece());
                            }
                        }
                        for (AbstractPiece piece : pieces) {
                            List<Square> squares =
piece.getMoves(board);
                            for (Square square : squares) {
                                for (int i = 1; i <= 3; i++) {
                                    if
(square.getPosition().equals(new Position(i, 0))) {
                                        return false;
                                    }
                                }
                            }
                        }
                        return true;
                    }
                }
            }
        }
        return false;
    }

    public boolean canCastleRight(Board board) {
        if (!this.isHasMoved()) {
            if (this.getColor() == PlayerColor.WHITE_PLAYER) {
                AbstractPiece rightRook =
board.getSquareFromPos(new Position(7, 7)).getOccupyingPiece();
```

```
                if (rightRook != null &&
!rightRook.isHasMoved()) {
                    boolean pathClear = true;
                    for (int i = 5; i <= 6; i++) {
                        if (board.getSquareFromPos(new
Position(i, 7)).getOccupyingPiece() != null) {
                            pathClear = false;
                            break;
                        }
                    }
                    if (pathClear) {
                        List<AbstractPiece> pieces = new
ArrayList<>();
                        for (Square square : board.getSquares())
{
                            if (square.getOccupyingPiece() !=
null && square.getOccupyingPiece().getColor() ==
PlayerColor.BLACK_PLAYER) {

pieces.add(square.getOccupyingPiece());
                            }
                        }
                        for (AbstractPiece piece : pieces) {
                            List<Square> squares =
piece.getMoves(board);
                            for (Square square : squares) {
                                for (int i = 5; i <= 6; i++) {
                                    if
(square.getPosition().equals(new Position(i, 7))) {
                                        return false;
                                    }
                                }
                            }
                        }
                        return true;
                    }
                }
            } else if (this.getColor() ==
PlayerColor.BLACK_PLAYER) {
                AbstractPiece rightRook =
board.getSquareFromPos(new Position(7, 0)).getOccupyingPiece();
                if (rightRook != null &&
!rightRook.isHasMoved()) {
                    boolean pathClear = true;
                    for (int i = 5; i <= 6; i++) {
                        if (board.getSquareFromPos(new
Position(i, 0)).getOccupyingPiece() != null) {
                            pathClear = false;
                            break;
                        }
                    }
                    if (pathClear) {
                        List<AbstractPiece> pieces = new
```

```
ArrayList<>();
                        for (Square square : board.getSquares())
{
                            if (square.getOccupyingPiece() !=
null && square.getOccupyingPiece().getColor() ==
PlayerColor.WHITE_PLAYER) {

pieces.add(square.getOccupyingPiece());
                            }
                        }
                        for (AbstractPiece piece : pieces) {
                            List<Square> squares =
piece.getMoves(board);
                            for (Square square : squares) {
                                for (int i = 5; i <= 6; i++) {
                                    if
(square.getPosition().equals(new Position(i, 0))) {
                                        return false;
                                    }
                                }
                            }
                        }
                        return true;
                    }
                }
            }
        }
        return false;
    }

    public List<Square> getValidMoves(Board board) {
        List<Square> output = new ArrayList<>();
        List<Square> moves = this.getMoves(board);
        for (Square square : moves) {
            if (!board.isInCheck(this.getColor(), new
Position[]{this.getPosition(), square.getPosition()})) {
                output.add(square);
            }
        }
        if (this.canCastleLeft(board)) {
            output.add(board.getSquareFromPos(new
Position(this.getPosition().getX() - 2,
this.getPosition().getY())));
        }
        if (this.canCastleRight(board)) {
            output.add(board.getSquareFromPos(new
Position(this.getPosition().getX() + 2,
this.getPosition().getY())));
        }
        return output;
    }

    @Override
```

```
    public void specificMove(Square square, Board board, Square
prevSquare) {
        if (this.getName().equals("K")) {
            if (prevSquare.getPosition().getX() -
this.getPosition().getX() == 2) {
                AbstractPiece rook = board.getPieceFromPos(new
Position(0, this.getPosition().getY()));
                rook.move(board, board.getSquareFromPos(new
Position(3, this.getPosition().getY())), true);
            } else if (prevSquare.getPosition().getX() -
this.getPosition().getX() == -2) {
                AbstractPiece rook = board.getPieceFromPos(new
Position(7, this.getPosition().getY()));
                rook.move(board, board.getSquareFromPos(new
Position(5, this.getPosition().getY())), true);
            }
        }
    }
}
```

Knight.java

```
package com.kubik.ChessWebapp.model.pieces;

import com.kubik.ChessWebapp.model.AbstractPiece;
import com.kubik.ChessWebapp.model.Board;
import com.kubik.ChessWebapp.model.Position;
import com.kubik.ChessWebapp.model.Square;
import com.kubik.ChessWebapp.statics.PlayerColor;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Knight extends AbstractPiece {
    public Knight(Position position, PlayerColor color) {
        super(position, color);
        this.setImgPath("images/" +
color.toString().toLowerCase().charAt(0) + "_knight.png");
        this.setName("N");
    }

    @Override
    public List<List<Square>> getPossibleMoves(Board board) {
        List<List<Square>> possibleMoves = new ArrayList<>();
        int[][] moves = {
                {1, -2}, {2, -1}, {2, 1}, {1, 2},
                {-1, 2}, {-2, 1}, {-2, -1}, {-1, -2}
        };
        for (int[] move : moves) {
            int newX = this.getPosition().getX() + move[0];
```

```
            int newY = this.getPosition().getY() + move[1];
            if (newX >= 0 && newX < 8 && newY >= 0 && newY < 8)
{
                Square square = board.getSquareFromPos(new
Position(newX, newY));

possibleMoves.add(Collections.singletonList(square));
            }
        }
        return possibleMoves;
    }

    @Override
    public void specificMove(Square square, Board board, Square
prevSquare) {
    }
}
```

Pawn.java

```
package com.kubik.ChessWebapp.model.pieces;

import com.kubik.ChessWebapp.model.AbstractPiece;
import com.kubik.ChessWebapp.model.Board;
import com.kubik.ChessWebapp.model.Position;
import com.kubik.ChessWebapp.model.Square;
import com.kubik.ChessWebapp.statics.PlayerColor;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Pawn extends AbstractPiece {
    boolean doubleMove;

    public Pawn(Position position, PlayerColor color) {
        super(position, color);
        this.setImgPath("images/" +
color.name().toLowerCase().charAt(0) + "_pawn.png");
        this.setName("P");
        this.doubleMove = false;
    }

    @Override
    public List<List<Square>> getPossibleMoves(Board board) {
        List<List<Square>> possibleMoves = new ArrayList<>();
        List<Position> direction = new ArrayList<>();
        if (this.getColor() == PlayerColor.WHITE_PLAYER) {
            direction.add(new Position(0, -1));
            doubleMove = false;
            if (!this.isHasMoved()) {
```

```
                direction.add(new Position(0, -2));
                doubleMove = true;
            }
        } else if (this.getColor() == PlayerColor.BLACK_PLAYER)
{
            direction.add(new Position(0, 1));
            doubleMove = false;
            if (!this.isHasMoved()) {
                direction.add(new Position(0, 2));
                doubleMove = true;
            }
        }
        for (Position move : direction) {
            int newY = this.getPosition().getY() + move.getY();
            if (newY >= 0 && newY < 8) {
                possibleMoves.add(Collections.singletonList(
                        board.getSquareFromPos(new
Position(this.getPosition().getX(), newY))));
            }
        }
        return possibleMoves;
    }

    @Override
    public List<Square> getMoves(Board board) {
        List<Square> result = new ArrayList<>();
        List<List<Square>> possibleMoves =
getPossibleMoves(board);
        for (List<Square> direction : possibleMoves) {
            for (Square square : direction) {
                if (square.getOccupyingPiece() != null) {
                    break;
                } else {
                    result.add(square);
                }
            }
        }
        if (this.getColor() == PlayerColor.WHITE_PLAYER) {
            if (this.getPosition().getX() + 1 < 8 &&
this.getPosition().getY() - 1 >= 0) {
                Square square = board.getSquareFromPos(
                        new Position(this.getPosition().getX() +
1, this.getPosition().getY() - 1)
                );
                if (square.getOccupyingPiece() != null &&
square.getOccupyingPiece().getColor() != this.getColor()) {
                    result.add(square);
                }
            }
            if (this.getPosition().getX() - 1 >= 0 &&
this.getPosition().getY() - 1 >= 0) {
                Square square = board.getSquareFromPos(new
Position(this.getPosition().getX() - 1,
```

```
this.getPosition().getY() - 1));
                if (square.getOccupyingPiece() != null &&
square.getOccupyingPiece().getColor() != this.getColor()) {
                    result.add(square);
                }
            }
        } else if (this.getColor() == PlayerColor.BLACK_PLAYER)
{
            if (this.getPosition().getX() + 1 < 8 &&
this.getPosition().getY() + 1 < 8) {
                Square square = board.getSquareFromPos(new
Position(this.getPosition().getX() + 1,
this.getPosition().getY() + 1));
                if (square.getOccupyingPiece() != null &&
square.getOccupyingPiece().getColor() != this.getColor()) {
                    result.add(square);
                }
            }
            if (this.getPosition().getX() - 1 >= 0 &&
this.getPosition().getY() + 1 < 8) {
                Square square = board.getSquareFromPos(new
Position(this.getPosition().getX() - 1,
this.getPosition().getY() + 1));
                if (square.getOccupyingPiece() != null &&
square.getOccupyingPiece().getColor() != this.getColor()) {
                    result.add(square);
                }
            }
        }
        return result;
    }

    @Override
    public void specificMove(Square square, Board board, Square
prevSquare) {
        if (this.getPosition().getY() == 0 ||
this.getPosition().getY() == 7) {
            square.setOccupyingPiece(new Queen(new
Position(this.getPosition().getX(), this.getPosition().getY()),
this.getColor()));
        }
    }
}
```

Queen.java

```
package com.kubik.ChessWebapp.model.pieces;

import com.kubik.ChessWebapp.model.AbstractPiece;
import com.kubik.ChessWebapp.model.Board;
import com.kubik.ChessWebapp.model.Position;
```

```
import com.kubik.ChessWebapp.model.Square;
import com.kubik.ChessWebapp.statics.PlayerColor;

import java.util.ArrayList;
import java.util.List;

public class Queen extends AbstractPiece {
    public Queen(Position position, PlayerColor color) {
        super(position, color);
        this.setImgPath("images/" +
color.name().toLowerCase().charAt(0) + "_queen.png");
        this.setName("Q");
    }

    @Override
    public List<List<Square>> getPossibleMoves(Board board) {
        List<List<Square>> possibleMoves = new ArrayList<>();
        List<Square> direction = new ArrayList<>();
        for (int y = this.getPosition().getY() - 1; y >= 0; y--)
{
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX(), y)));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int i = 1; i < 8; i++) {
            if (this.getPosition().getX() + i > 7 ||
this.getPosition().getY() - i < 0) {
                break;
            }
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX() + i,
this.getPosition().getY() - i)));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int x = this.getPosition().getX() + 1; x < 8; x++)
{
            direction.add(board.getSquareFromPos(new Position(x,
this.getPosition().getY())));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int i = 1; i < 8; i++) {
            if (this.getPosition().getX() + i > 7 ||
this.getPosition().getY() + i > 7) {
                break;
            }
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX() + i,
this.getPosition().getY() + i)));
        }
        possibleMoves.add(new ArrayList<>(direction));
```

```
        direction.clear();
        for (int y = this.getPosition().getY() + 1; y < 8; y++)
{
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX(), y)));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int i = 1; i < 8; i++) {
            if (this.getPosition().getX() - i < 0 ||
this.getPosition().getY() + i > 7) {
                break;
            }
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX() - i,
this.getPosition().getY() + i)));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int x = this.getPosition().getX() - 1; x >= 0; x--)
{
            direction.add(board.getSquareFromPos(new Position(x,
this.getPosition().getY())));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int i = 1; i < 8; i++) {
            if (this.getPosition().getX() - i < 0 ||
this.getPosition().getY() - i < 0) {
                break;
            }
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX() - i,
this.getPosition().getY() - i)));
        }
        possibleMoves.add(new ArrayList<>(direction));
        return possibleMoves;
    }

    @Override
    public void specificMove(Square square, Board board, Square
prevSquare) {

    }
}
```

Rook.java

```
package com.kubik.ChessWebapp.model.pieces;

import com.kubik.ChessWebapp.model.AbstractPiece;
```

```
import com.kubik.ChessWebapp.model.Board;
import com.kubik.ChessWebapp.model.Position;
import com.kubik.ChessWebapp.model.Square;
import com.kubik.ChessWebapp.statics.PlayerColor;

import java.util.ArrayList;
import java.util.List;

public class Rook extends AbstractPiece {
    public Rook(Position position, PlayerColor color) {
        super(position, color);
        this.setImgPath("images/" +
color.name().toLowerCase().charAt(0) + "_rook.png");
        this.setName("R");
    }

    @Override
    public List<List<Square>> getPossibleMoves(Board board) {
        List<List<Square>> possibleMoves = new ArrayList<>();
        List<Square> direction = new ArrayList<>();
        for (int y = this.getPosition().getY() - 1; y >= 0; y--)
{
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX(), y)));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int x = this.getPosition().getX() + 1; x < 8; x++)
{
            direction.add(board.getSquareFromPos(new Position(x,
this.getPosition().getY())));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int y = this.getPosition().getY() + 1; y < 8; y++)
{
            direction.add(board.getSquareFromPos(new
Position(this.getPosition().getX(), y)));
        }
        possibleMoves.add(new ArrayList<>(direction));
        direction.clear();
        for (int x = this.getPosition().getX() - 1; x >= 0; x--)
{
            direction.add(board.getSquareFromPos(new Position(x,
this.getPosition().getY())));
        }
        possibleMoves.add(new ArrayList<>(direction));
        return possibleMoves;
    }

    @Override
    public void specificMove(Square square, Board board, Square
prevSquare) {
```

```
    }
}
```

AbstractPiece.java

```
package com.kubik.ChessWebapp.model;


import com.kubik.ChessWebapp.statics.PlayerColor;
import lombok.Getter;
import lombok.Setter;

import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
public abstract class AbstractPiece {
    private Position position;
    private PlayerColor color;
    private boolean hasMoved = false;
    private boolean attackingKing = false;
    private String name = null;
    private boolean enPassant = false;
    private String imgPath;

    public AbstractPiece(Position position, PlayerColor color) {
        this.position = position;
        this.color = color;
    }

    public boolean move(Board board, Square square, boolean
force) {
        for (Square s : board.getSquares()) {
            s.setHighlight(false);
        }
        if (getValidMoves(board).contains(square) || force) {
            Square prevSquare =
board.getSquareFromPos(position); //todo to conclude for what
this line
            this.setPosition(square.getPosition());
            square.setOccupyingPiece(this);
            prevSquare.setOccupyingPiece(null);
            this.hasMoved = true;
            board.setSelectedPiece(null);
            specificMove(square, board, prevSquare);
            return true;
        } else {
            board.setSelectedPiece(null);
            return false;
```

```
        }
    }

    public List<Square> getMoves(Board board) {
        List<Square> output = new ArrayList<>();
        List<List<Square>> possibleMoves =
getPossibleMoves(board);
        for (List<Square> direction : possibleMoves) {
            for (Square square : direction) {
                if (square.getOccupyingPiece() != null) {
                    if
(!square.getOccupyingPiece().getColor().equals(this.color)) {
                        output.add(square);
                        break;
                    } else {
                        break;
                    }
                } else {
                    output.add(square);
                }
            }
        }
        return output;
    }

    public List<Square> getValidMoves(Board board) {
        List<Square> result = new ArrayList<>();
        for (Square square : getMoves(board)) {
            if (!board.isInCheck(this.color, new
Position[]{this.getPosition(), square.getPosition()})) {
                result.add(square);
            }
        }
        return result;
    }

    public abstract List<List<Square>> getPossibleMoves(Board
board);

    public abstract void specificMove(Square square, Board
board, Square prevSquare);
}
```

Board.java

```
package com.kubik.ChessWebapp.model;

import com.kubik.ChessWebapp.entity.ChessUser;
import com.kubik.ChessWebapp.model.pieces.*;
import com.kubik.ChessWebapp.statics.BoardStatus;
import com.kubik.ChessWebapp.statics.GameResult;
```

```java
import com.kubik.ChessWebapp.statics.PlayerColor;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

@Getter
@Setter
@ToString
public class Board {
    private String id;
    private ChessUser firstChessPlayer;
    private ChessUser secondChessPlayer;
    private GameResult gameResult;
    private AbstractPiece selectedPiece;
    private PlayerColor turn;
    private List<Square> squares;
    private BoardStatus boardStatus;
    private final static String[][] config = {
            {"bR", "bN", "bB", "bQ", "bK", "bB", "bN", "bR"},
            {"bP", "bP", "bP", "bP", "bP", "bP", "bP", "bP"},
            {"  ", "  ", "  ", "  ", "  ", "  ", "  ", "  "},
            {"  ", "  ", "  ", "  ", "  ", "  ", "  ", "  "},
            {"  ", "  ", "  ", "  ", "  ", "  ", "  ", "  "},
            {"  ", "  ", "  ", "  ", "  ", "  ", "  ", "  "},
            {"wP", "wP", "wP", "wP", "wP", "wP", "wP", "wP"},
            {"wR", "wN", "wB", "wQ", "wK", "wB", "wN", "wR"}
    };

    public Board() {
        this.id = UUID.randomUUID().toString();
        this.selectedPiece = null;
        this.turn = PlayerColor.WHITE_PLAYER;
        this.boardStatus = BoardStatus.NEW;
        this.gameResult = null;
        this.squares = squaresInit();
        setField();
    }

    private List<Square> squaresInit() {
        List<Square> result = new ArrayList<>();
        for (int y = 0; y < 8; y++) {
            for (int x = 0; x < 8; x++) {
                result.add(new Square(new Position(x, y)));
            }
        }
        return result;
    }

    public Square getSquareFromPos(Position position) {
```

```
        for (Square square : squares) {
            Position squarePosition = square.getPosition();
            if (squarePosition.getX() == position.getX() &&
squarePosition.getY() == position.getY()) {
                return square;
            }
        }
        return null;
    }

    public AbstractPiece getPieceFromPos(Position position) {
        Square square = getSquareFromPos(position);
        return square != null ? square.getOccupyingPiece() :
null;
    }

    private void setField() {
        for (int y = 0; y < config.length; y++) {
            for (int x = 0; x < config[y].length; x++) {
                String piece = config[y][x];
                if (!"  ".equals(piece)) {
                    Square square = getSquareFromPos(new
Position(x, y));
                    PlayerColor color = piece.charAt(0) == 'w' ?
PlayerColor.WHITE_PLAYER : PlayerColor.BLACK_PLAYER;
                    switch (piece.charAt(1)) {
                        case 'R':
                            square.setOccupyingPiece(new
Rook(new Position(x, y), color));
                            break;
                        case 'N':
                            square.setOccupyingPiece(new
Knight(new Position(x, y), color));
                            break;
                        case 'B':
                            square.setOccupyingPiece(new
Bishop(new Position(x, y), color));
                            break;
                        case 'Q':
                            square.setOccupyingPiece(new
Queen(new Position(x, y), color));
                            break;
                        case 'K':
                            square.setOccupyingPiece(new
King(new Position(x, y), color));
                            break;
                        case 'P':
                            square.setOccupyingPiece(new
Pawn(new Position(x, y), color));
                            break;
                    }
                }
            }
```

```
        }
    }

    public void mouseClick(int x, int y) {
        Square clickedSquare = getSquareFromPos(new Position(x,
y));
        if (selectedPiece == null) {
            if (clickedSquare.getOccupyingPiece() != null &&
clickedSquare.getOccupyingPiece().getColor().equals(turn)) {
                selectedPiece =
clickedSquare.getOccupyingPiece();
                showMoves();
            }
        } else if (selectedPiece.move(this, clickedSquare,
false)) {
            turn = PlayerColor.togglePlayerTurn(turn);
        }
        checkGameOver();
    }

    public boolean isInCheck(PlayerColor color, Position[]
AttackingKingPositions) {
        boolean result = false;
        Position kingPos = null;
        AbstractPiece changingPiece = null;
        Square oldSquare = null;
        Square newSquare = null;
        AbstractPiece newSquareOldPiece = null;
        if (AttackingKingPositions != null) {
            for (Square square : squares) {
                if
(square.getPosition().equals(AttackingKingPositions[0])) {
                    changingPiece = square.getOccupyingPiece();
                    oldSquare = square;
                    oldSquare.setOccupyingPiece(null);
                    break;
                }
            }
            for (Square square : squares) {
                if
(square.getPosition().equals(AttackingKingPositions[1])) {
                    newSquare = square;
                    newSquareOldPiece =
newSquare.getOccupyingPiece();
                    newSquare.setOccupyingPiece(changingPiece);
                    break;
                }
            }
        }
        List<AbstractPiece> pieces = new ArrayList<>();
        for (Square square : squares) {
            if (square.getOccupyingPiece() != null) {
                pieces.add(square.getOccupyingPiece());
```

```
            }
        }
        if (changingPiece != null &&
"K".equals(changingPiece.getName())) {
            kingPos = newSquare.getPosition();
        }
        if (kingPos == null) {
            for (AbstractPiece piece : pieces) {
                if ("K".equals(piece.getName()) &&
color.equals(piece.getColor())) {
                    kingPos = piece.getPosition();
                    break;
                }
            }
        }
        for (AbstractPiece piece : pieces) {
            if (!color.equals(piece.getColor())) {
                for (Square square : piece.getMoves(this)) {
                    if (square.getPosition().equals(kingPos)) {
                        result = true;
                        break;
                    }
                }
            }
        }
        if (AttackingKingPositions != null) {
            oldSquare.setOccupyingPiece(changingPiece);
            newSquare.setOccupyingPiece(newSquareOldPiece);
        }
        return result;
    }

    public boolean isInCheckmate(PlayerColor color) {
        if (isInCheck(color, null)) {
            List<AbstractPiece> kingPieces = new ArrayList<>();
            for (Square square : squares) {
                if (square.getOccupyingPiece() != null &&
color.equals(square.getOccupyingPiece().getColor())) {
                    kingPieces.add(square.getOccupyingPiece());
                }
            }
            for (AbstractPiece piece : kingPieces) {
                if (!piece.getValidMoves(this).isEmpty()) {
                    return false;
                }
            }
            return true;
        }
        return false;
    }

    public boolean isStalemate(PlayerColor color) {
        for (Square square : squares) {
```

```
            if (square.getOccupyingPiece() != null &&
color.equals(square.getOccupyingPiece().getColor())) {
                if
(!square.getOccupyingPiece().getValidMoves(this).isEmpty()) {
                    return false;
                }
            }
        }
        return true;
    }

    public void showMoves() {
        if (selectedPiece != null) {

getSquareFromPos(selectedPiece.getPosition()).setHighlight(true)
;
            for (Square square :
selectedPiece.getValidMoves(this)) {
                square.setHighlight(true);
            }
        }
    }

    public GameResult checkGameOver() {
        if (isInCheckmate(PlayerColor.WHITE_PLAYER)) {
            return GameResult.BLACK_PLAYER_WIN;
        } else if (isInCheckmate(PlayerColor.BLACK_PLAYER)) {
            return GameResult.WHITE_PLAYER_WIN;
        } else if (isStalemate(turn)) {
            return GameResult.STALEMATE;
        }
        return null;
    }
}
```

Move.java

```
package com.kubik.ChessWebapp.model;

import lombok.Data;

@Data
public class Move {
    private String gameId;
    private String squareIndex;
}
```

Position.java

```java
package com.kubik.ChessWebapp.model;


import lombok.Data;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Data
public class Position {
    private int x;
    private int y;

    public Position(int x, int y){
        this.x = x;
        this.y = y;
    }
}
```

Square.java

```java
package com.kubik.ChessWebapp.model;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@ToString
public class Square {
    private Position position;
    private String color;
    private AbstractPiece occupyingPiece;
    private boolean isHighlight;

    public Square(Position position) {
        this.position = position;
        this.color = (position.getX() + position.getY()) % 2 ==
0 ? "light" : "dark";
        this.occupyingPiece = null;
        this.isHighlight = false;
    }
}
```

BoardRepository.java

```java
package com.kubik.ChessWebapp.repository;
```

```
import com.kubik.ChessWebapp.model.Board;
import com.kubik.ChessWebapp.statics.BoardStatus;
import org.springframework.data.mongodb.core.mapping.Document;
import
org.springframework.data.mongodb.repository.MongoRepository;

import java.util.Optional;

@Document(collection = "boards")
public interface BoardRepository extends MongoRepository<Board,
Long> {
    Optional<Board>
findFirstBySecondChessPlayerIsNullAndBoardStatus(BoardStatus
boardStatus);
    Optional<Board> findById(String gameId);
}
```

ChessUserRepository.java

```
package com.kubik.ChessWebapp.repository;

import com.kubik.ChessWebapp.entity.ChessUser;
import
org.springframework.data.mongodb.repository.MongoRepository;

public interface ChessUserRepository extends
MongoRepository<ChessUser, Long> {
    ChessUser findByNickname(String nickname);
}
```

ChessGameService.java

```
package com.kubik.ChessWebapp.service;

import com.kubik.ChessWebapp.entity.ChessUser;
import com.kubik.ChessWebapp.model.Board;
import com.kubik.ChessWebapp.model.Move;
import com.kubik.ChessWebapp.repository.BoardRepository;
import com.kubik.ChessWebapp.statics.BoardStatus;
import com.kubik.ChessWebapp.statics.GameResult;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
@RequiredArgsConstructor
public class ChessGameService {
    private final BoardRepository boardRepository;
```

```
    public Board createGame(ChessUser user) {
        Board board = new Board();
        board.setFirstChessPlayer(user);
        boardRepository.save(board);
        return board;
    }

    public Board connectToRandom(ChessUser user) {
        Optional<Board> optionalGame =
boardRepository.findFirstBySecondChessPlayerIsNullAndBoardStatus
(BoardStatus.NEW);
        optionalGame.orElseThrow(() -> new RuntimeException("All
games are occupied"));
        Board board = optionalGame.get();
        board.setSecondChessPlayer(user);
        board.setBoardStatus(BoardStatus.IN_PROGRESS);
        boardRepository.save(board);
        return board;
    }

    public Board connectToSpecific(ChessUser user, String
gameId) {
        Optional<Board> optionalGame =
boardRepository.findById(gameId);
        optionalGame.orElseThrow(() -> new
RuntimeException("There is no game with provided id"));
        Board board = optionalGame.get();
        board.setSecondChessPlayer(user);
        board.setBoardStatus(BoardStatus.IN_PROGRESS);
        boardRepository.save(board);
        return board;
    }

    public Board getGame(String gameId) {
        Optional<Board> optionalGame =
boardRepository.findById(gameId);
        optionalGame.orElseThrow(() -> new RuntimeException("All
games are occupied"));
        return optionalGame.get();
    }

    public Board move(Move move) {
        Optional<Board> optionalGame =
boardRepository.findById(move.getGameId());
        optionalGame.orElseThrow(() -> new
RuntimeException("There is no game with provided id"));
        Board board = optionalGame.get();
        String squareIndex = move.getSquareIndex();

board.mouseClick(Character.getNumericValue(squareIndex.charAt(0)
), Character.getNumericValue(squareIndex.charAt(1)));
        GameResult gameResult = board.checkGameOver();
```

```
        if (gameResult != null) {
            board.setGameResult(gameResult);
            board.setBoardStatus(BoardStatus.FINISHED);
        }
        boardRepository.save(board);
        return board;
    }
}
```

ChessUserService.java

```
package com.kubik.ChessWebapp.service;

import com.kubik.ChessWebapp.dto.ChessUserDto;
import com.kubik.ChessWebapp.entity.ChessUser;
import com.kubik.ChessWebapp.repository.ChessUserRepository;
import lombok.RequiredArgsConstructor;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class ChessUserService {
    private final ChessUserRepository chessUserRepository;
    private final ModelMapper modelMapper;

    public String createUser(ChessUserDto chessUserDto) {
        return
chessUserRepository.save(modelMapper.map(chessUserDto,
ChessUser.class)).getId();
    }

    public ChessUser getUserByNickname(String nickname) {
        return chessUserRepository.findByNickname(nickname);
    }

    public List<ChessUser> getAllUsers() {
        return (List<ChessUser>) chessUserRepository.findAll();
    }
}
```

BoardStatus.java

```
package com.kubik.ChessWebapp.statics;

public enum BoardStatus {
```

```
    NEW, IN_PROGRESS, FINISHED
}
```

GameResult.java

```
package com.kubik.ChessWebapp.statics;

public enum GameResult {
    WHITE_PLAYER_WIN, BLACK_PLAYER_WIN, STALEMATE;
}
```

PlayerColor.java

```
package com.kubik.ChessWebapp.statics;

public enum PlayerColor {
    WHITE_PLAYER, BLACK_PLAYER;

    public static PlayerColor togglePlayerTurn(PlayerColor
currentPlayerColor) {
        switch (currentPlayerColor) {
            case WHITE_PLAYER: return BLACK_PLAYER;
            case BLACK_PLAYER: return WHITE_PLAYER;
            default: throw new RuntimeException("Invalid player
turn");
        }
    }
}
```

GameController.java

```
package com.kubik.ChessWebapp.web;

import com.kubik.ChessWebapp.dto.ChessPlayerDto;
import com.kubik.ChessWebapp.dto.GameConnect;
import com.kubik.ChessWebapp.model.Board;
import com.kubik.ChessWebapp.entity.ChessUser;
import com.kubik.ChessWebapp.model.Move;
import com.kubik.ChessWebapp.service.ChessGameService;
import com.kubik.ChessWebapp.service.ChessUserService;
import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
```

```
@Controller
@AllArgsConstructor
public class GameController {
    private final ChessGameService chessGameService;
    private final ChessUserService chessUserService;
    private final SimpMessagingTemplate simpMessagingTemplate;

    @PostMapping("/board-create")
    public String createBoard(@RequestBody String gameId, Model
model) {
        model.addAttribute("board",
chessGameService.getGame(gameId));
        return "fragments/game-page :: game-board";
    }

    @PostMapping("/game-create-json")
    public ResponseEntity<Board> createGame(@RequestBody
ChessPlayerDto chessPlayer) {
        ChessUser userByNickname =
chessUserService.getUserByNickname(chessPlayer.getNickname());
        return
ResponseEntity.ok(chessGameService.createGame(userByNickname));
    }

    @PostMapping("/connect-random-json")
    public ResponseEntity<Board> connectRandom(@RequestBody
ChessPlayerDto chessPlayer) throws RuntimeException {
        ChessUser userByNickname =
chessUserService.getUserByNickname(chessPlayer.getNickname());
        return
ResponseEntity.ok(chessGameService.connectToRandom(userByNicknam
e));
    }

    @PostMapping("/connect-json")
    public ResponseEntity<Board> connectSpecific(@RequestBody
GameConnect connect) throws RuntimeException {
        ChessUser userByNickname =
chessUserService.getUserByNickname(connect.getPlayer().getNickna
me());
        return
ResponseEntity.ok(chessGameService.connectToSpecific(userByNickn
ame, connect.getGameId()));
    }


    @PostMapping("/game-move")
    public ResponseEntity<Board> move(@RequestBody Move move)
throws RuntimeException {
        Board board = chessGameService.move(move);
        simpMessagingTemplate.convertAndSend("/topic/game-
progress/" + board.getId(), board);
```

```
        return ResponseEntity.ok(board);
    }
}
```

IndexController.java

```
package com.kubik.ChessWebapp.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class IndexController {
    @GetMapping("/")
    public String index() {
        return "redirect:/login";
    }
}
```

LoginController.java

```
package com.kubik.ChessWebapp.web;

import com.kubik.ChessWebapp.dto.ChessUserDto;
import com.kubik.ChessWebapp.entity.ChessUser;
import com.kubik.ChessWebapp.service.ChessUserService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;


@Controller
@RequiredArgsConstructor
public class LoginController {
    private final ChessUserService chessUserService;

    @GetMapping("/login")
    public String showWelcomePage(Model model) {
        return "login";
    }

    @GetMapping("/registration")
    public String showRegistration(ChessUserDto chessUser, Model
model) {
        return "registration";
    }
```

```
    @PostMapping("/main-page")
    public String logIn(ChessUserDto chessUser, BindingResult
bindingResult, Model model) {
        ChessUser userFromDB =
chessUserService.getUserByNickname(chessUser.getNickname());
        if (userFromDB == null ||
!userFromDB.getPassword().equals(chessUser.getPassword())) {
            bindingResult.rejectValue("password",
"error.password", "Invalid password");
            return "login-failed";
        }
        model.addAttribute("user", userFromDB);
        return "main-page";
    }
}
```

RegistrationController.java

```
package com.kubik.ChessWebapp.web;

import com.kubik.ChessWebapp.dto.ChessUserDto;
import com.kubik.ChessWebapp.service.ChessUserService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
@RequiredArgsConstructor
public class RegistrationController {
    private final ChessUserService chessUserService;

    @PostMapping("/registration-result")
    public String registerUser(@Valid ChessUserDto chessUser,
BindingResult bindingResult, Model model) {
        if (bindingResult.hasErrors()) {
            return "registration";
        }
        if
(!chessUser.getPassword().equals(chessUser.getConfirmPassword())
) {
            bindingResult.rejectValue("confirmPassword",
"error.user", "Passwords do not match");
            return "registration";
        }
        chessUserService.createUser(chessUser);
        model.addAttribute("user", chessUser);
        return "registration-successful";
```

```
    }
}
```

ChessWebappApplication.java

```
package com.kubik.ChessWebapp;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ChessWebappApplication {
    public static void main(String[] args) {
        SpringApplication.run(ChessWebappApplication.class,
args);
    }
}
```

game-style.css

```
body{
    background-color: #9fe093;
}
table{
    border-spacing: 0;
}
th, td{
    height: 60px;
    width: 60px;
}
img {
    display: block;
    margin: auto;
}
#board{
    position: absolute;
    top: 10%;
    left: 20%;
}

.playerNickname{
    text-indent: 60px;
    font-size: 2rem;
    color: #888888;
    text-align: center;
    margin-bottom: 30px;
}
.playerTurn {
```

```
    color: #000000;
}

.light{
    background-color: antiquewhite;
}
.dark{
    background-color:saddlebrown;
}
.light.highlighted{
    background-color: chartreuse !important;
}
.dark.highlighted{
    background-color: darkgreen !important;
}
```

login-style.css

```
body, html {
    margin: 0;
    padding: 0;
    height: 100%;
}

#background-login{
    width: 100vw;
    height: 100vh;
    display: flex;
    align-items: center;
    justify-content: center;
    flex-direction: column;

    background-color: rgb(165, 165, 165);
}
#window-login {
    padding-top: 4vw;
    padding-bottom: 4vw;
    width:50%;
    background-color: rgb(112, 112, 112);
    border-radius: 50px;
    text-indent: 8vw;
    border: solid rgb(239, 239, 239);
}
.form-div{
    margin: 2vw;
}
.error-span{
    color: rgb(255, 0, 0);
}
```

main-style.css

```
body{
    background-color: #9fe093;
}

#buttonsFrame{
    position: absolute;
    top: 10vh;
    left: 35vw;
    width: fit-content;
    text-align: center;
    background: #35e314;
    border-radius: 10px;
}
.box{
    background-color: #9fe093;
    border-radius: 7px;
    margin: 5vh;
    padding: 1vh;
}
button{
    padding: 3vh;
    border-radius: 7px;
}

button:hover {
    cursor: pointer;
    background: #e1ffe1;
}
```

script.js

```
var playerTurnNow = "";
var imgTransparent =
"
AICTAEAOw=="
let chessSquaresAndIndexSquaresMap = {
    "00": "18", "10": "28", "20": "38", "30": "48", "40": "58",
"50": "68", "60": "78", "70": "88",
    "01": "17", "11": "27", "21": "37", "31": "47", "41": "57",
"51": "67", "61": "77", "71": "87",
    "02": "16", "12": "26", "22": "36", "32": "46", "42": "56",
"52": "66", "62": "76", "72": "86",
    "03": "15", "13": "25", "23": "35", "33": "45", "43": "55",
"53": "65", "63": "75", "73": "85",
    "04": "14", "14": "24", "24": "34", "34": "44", "44": "54",
"54": "64", "64": "74", "74": "84",
    "05": "13", "15": "23", "25": "33", "35": "43", "45": "53",
"55": "63", "65": "73", "75": "83",
    "06": "12", "16": "22", "26": "32", "36": "42", "46": "52",
```

```
"56": "62", "66": "72", "76": "82",
    "07": "11", "17": "21", "27": "31", "37": "41", "47": "51",
"57": "61", "67": "71", "77": "81"
};

const getKeyByValue = (object, value) => {
    return Object.keys(object).find(key => object[key] ===
value);
};

function playerTurn(id) {
    if (playerTurnNow !== playerType) {
        alert("It's not your turn!")
    } else {
        makeAMove(id);
    }
}

function makeAMove(id) {
    $.ajax({
        url: url + "/game-move",
        type: 'POST',
        dataType: "json",
        contentType: "application/json",
        data: JSON.stringify({
            "gameId": gameId,
            "squareIndex": id
        }),
        success: function (board) {
            refreshChessBoard(board);
        },
        error: function (error) {
            console.log(error);
        }
    })
}

function refreshChessBoard(board) {
    console.log(board);
    let squares = board.squares;
    squares.forEach((square) => {
        var piece = square.occupyingPiece;
        var position = square.position;
        var htmlPosition = chessSquaresAndIndexSquaresMap['' +
position.x + position.y];
        if (piece != null) {
            var imgPath = "/" + piece.imgPath;
            $("#square_" + htmlPosition + " img").attr("src",
imgPath);
        } else {
            $("#square_" + htmlPosition + " img").attr("src",
imgTransparent);
        }
```

```
        if (square.highlight) {
            $("#square_" +
htmlPosition).addClass("highlighted");
        } else {
            $("#square_" +
htmlPosition).removeClass("highlighted");
        }
    });

    if (board.gameResult != null) {
        alert(board.gameResult);
    }
    playerTurnNow = board.turn;

    if (board.firstChessPlayer != null &&
board.secondChessPlayer != null) {
        $("#first_player").text(board.firstChessPlayer.nickname
== null ? "" : board.firstChessPlayer.nickname);

$("#second_player").text(board.secondChessPlayer.nickname ==
null ? "" : board.secondChessPlayer.nickname);
    }

    if (playerTurnNow == "WHITE_PLAYER") {
        $("#first_player").addClass("playerTurn");
        $("#second_player").removeClass("playerTurn");
    } else {
        $("#second_player").addClass("playerTurn")
        $("#first_player").removeClass("playerTurn");
    }
}


$(document).on("click", ".light, .dark", function () {
    var squareId = $(this).attr('id');
    var s = squareId.substring(7);
    var keyByValue =
getKeyByValue(chessSquaresAndIndexSquaresMap, s);
    playerTurn(keyByValue);
});
```

socket_js.js

```
const url = 'http://192.168.0.103:8080'; //change on local
server ip for local game
let stompClient;
let gameId;
let playerType;

function connectToSocket(gameId) {
    let socket = new SockJS(url + "/move");
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function (frame) {
```

```
        stompClient.subscribe("/topic/game-progress/" + gameId,
function (response) {
            let board = JSON.parse(response.body);
            console.log(board);
            refreshChessBoard(board);
        })
    })
}

 function create_game(name) {
    $.ajax({
        url: url + "/game-create-json",
        type: 'POST',
        dataType: "json",
        contentType: "application/json",
        data: JSON.stringify({
            "nickname": name
        }),
        success: function (board) {
            gameId = board.id;
            playerType = "WHITE_PLAYER";
            createBoardHtml(gameId, board);

            connectToSocket(gameId);
            alert("You created a game. Game id: " + board.id);
        },
        error: function (error) {
            console.log(error);
        }
    });

}


function connectToRandom(name) {
    $.ajax({
        url: url + "/connect-random-json",
        type: 'POST',
        dataType: "json",
        contentType: "application/json",
        data: JSON.stringify({
            "nickname": name
        }),
        success: function (board) {
            gameId = board.id;
            playerType = "BLACK_PLAYER";
            createBoardHtml(gameId, board);
            connectToSocket(gameId);

            alert("You are playing with: " +
board.firstChessPlayer.nickname);
        },
        error: function (error) {
```

```
            console.log(error);
        }
    })
}

function connectToSpecific(name) {
    gameId = document.getElementById("game_id").value;
    if (gameId === '' || gameId == null) {
        alert("Incorrect game id");
    }
    $.ajax({
        url: url + "/connect-json",
        type: 'POST',
        dataType: "json",
        contentType: "application/json",
        data: JSON.stringify({
            "player":{
                "nickname":name
            },
            "gameId": gameId
        }),
        success: function (board) {
            playerType = "BLACK_PLAYER";
            createBoardHtml(gameId, board);
            connectToSocket(gameId);

            alert("You are playing with: " +
board.firstChessPlayer.nickname);
        },
        error: function (error) {
            console.log(error);
        }
    })
}

function createBoardHtml(gameId, board){
    $.ajax({
        url: url + "/board-create",
        type: 'POST',
        dataType: "html",
        contentType: "application/json",
        data: gameId,
        success: function (fragment) {
            $("#buttonsFrame").remove();
            $("#boardFrame").html(fragment);
            refreshChessBoard(board);
        },
        error: function (error) {
            console.log(error);
        }
    });
}
```

game-page.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
<div th:fragment="game-board" id="board">
    <div class="playerNickname" id="second_player"></div>

    <table>
        <tr>
            <th>8</th>
            <td class="light" id="square_18"><img></td>
            <td class="dark" id="square_28"><img></td>
            <td class="light" id="square_38"><img></td>
            <td class="dark" id="square_48"><img></td>
            <td class="light" id="square_58"><img></td>
            <td class="dark" id="square_68"><img></td>
            <td class="light" id="square_78"><img></td>
            <td class="dark" id="square_88"><img></td>
        </tr>
        <tr>
            <th>7</th>
            <td class="dark" id="square_17"><img></td>
            <td class="light" id="square_27"><img></td>
            <td class="dark" id="square_37"><img></td>
            <td class="light" id="square_47"><img></td>
            <td class="dark" id="square_57"><img></td>
            <td class="light" id="square_67"><img></td>
            <td class="dark" id="square_77"><img></td>
            <td class="light" id="square_87"><img></td>
        </tr>
        <tr>
            <th>6</th>
            <td class="light" id="square_16"><img></td>
            <td class="dark" id="square_26"><img></td>
            <td class="light" id="square_36"><img></td>
            <td class="dark" id="square_46"><img></td>
            <td class="light" id="square_56"><img></td>
            <td class="dark" id="square_66"><img></td>
            <td class="light" id="square_76"><img></td>
            <td class="dark" id="square_86"><img></td>
        </tr>
        <tr>
            <th>5</th>
            <td class="dark" id="square_15"><img></td>
            <td class="light" id="square_25"><img></td>
            <td class="dark" id="square_35"><img></td>
            <td class="light" id="square_45"><img></td>
            <td class="dark" id="square_55"><img></td>
            <td class="light" id="square_65"><img></td>
            <td class="dark" id="square_75"><img></td>
            <td class="light" id="square_85"><img></td>
```

```
        </tr>
        <tr>
            <th>4</th>
            <td class="light" id="square_14"><img></td>
            <td class="dark" id="square_24"><img></td>
            <td class="light" id="square_34"><img></td>
            <td class="dark" id="square_44"><img></td>
            <td class="light" id="square_54"><img></td>
            <td class="dark" id="square_64"><img></td>
            <td class="light" id="square_74"><img></td>
            <td class="dark" id="square_84"><img></td>
        </tr>
        <tr>
            <th>3</th>
            <td class="dark" id="square_13"><img></td>
            <td class="light" id="square_23"><img></td>
            <td class="dark" id="square_33"><img></td>
            <td class="light" id="square_43"><img></td>
            <td class="dark" id="square_53"><img></td>
            <td class="light" id="square_63"><img></td>
            <td class="dark" id="square_73"><img></td>
            <td class="light" id="square_83"><img></td>
        </tr>
        <tr>
            <th>2</th>
            <td class="light" id="square_12"><img></td>
            <td class="dark" id="square_22"><img></td>
            <td class="light" id="square_32"><img></td>
            <td class="dark" id="square_42"><img></td>
            <td class="light" id="square_52"><img></td>
            <td class="dark" id="square_62"><img></td>
            <td class="light" id="square_72"><img></td>
            <td class="dark" id="square_82"><img></td>
        </tr>
        <tr>
            <th>1</th>
            <td class="dark" id="square_11"><img></td>
            <td class="light" id="square_21"><img></td>
            <td class="dark" id="square_31"><img></td>
            <td class="light" id="square_41"><img></td>
            <td class="dark" id="square_51"><img></td>
            <td class="light" id="square_61"><img></td>
            <td class="dark" id="square_71"><img></td>
            <td class="light" id="square_81"><img></td>
        </tr>
        <tr>
            <th></th>
            <th>a</th>
            <th>b</th>
            <th>c</th>
            <th>d</th>
            <th>e</th>
            <th>f</th>
```

```
            <th>g</th>
            <th>h</th>
        </tr>
    </table>
    <div class="playerNickname" id="first_player"></div>
</div>
</body>
</html>
```

login.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Chess</title>
    <link type="text/css" href="/css/login-style.css"
rel="stylesheet">
</head>

<body>
    <div id="background-login">
        <div>
            <h1>Welcome to chess website</h1>
        </div>
        <div id="window-login">
            <form th:action="@{/main-page}" method="POST">
                <div class="form-div">
                    <label for="nickname">Username:</label>
                    <input type="text" id="nickname"
name="nickname" required>
                </div>

                <div class="form-div">
                    <label for="password">Password:</label>
                    <input type="password" id="password"
name="password" required>
                </div>

                <div class="form-div">
                    <button type="submit">Log in</button>
                    <a th:href="@{/registration}"
class="register-button">Register</a>
                </div>
            </form>
        </div>
    </div>
</body>

</html>
```

login-failed.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Chess</title>
    <link type="text/css" href="/css/login-style.css"
rel="stylesheet">
</head>

<body>
<div id="background-login">
    <div>
        <h1>Welcome to chess website</h1>
    </div>
    <div id="window-login">
        <form th:action="@{/main-page}"
th:object="${chessUserDto}" method="POST">
            <div class="form-div">
                <label for="nickname">Username:</label>
                <input type="text" id="nickname" name="nickname"
required>
            </div>

            <div class="form-div">
                <label for="password">Password:</label>
                <input type="password" id="password"
name="password" required>
                <span class="error-span"
th:if="${#fields.hasErrors('password')}"
th:errors="*{password}"></span>
            </div>

            <div class="form-div">
                <button type="submit">Log in</button>
                <a th:href="@{/registration}" class="register-
button">Register</a>
            </div>
        </form>
    </div>
</div>

</body>

</html>
```

main-page.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Chess</title>
    <link type="text/css" href="/css/main-style.css"
rel="stylesheet">
    <link type="text/css" href="/css/game-style.css"
rel="stylesheet">
    <!--  sockjs, stomp and jquery-->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/sockjs-
client/1.4.0/sockjs.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp
.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.1/jquery.
min.js"></script>
</head>

<body>
<div id="buttonsFrame">
    <p th:text="'Welcome, ' + ${user.nickname}"></p>
    <div class="box">
        <button name="createNewGameBtn"
th:attr="onclick=|create_game('${user.nickname}')|">Create a new
game</button>
    </div>
    <div class="box">
        <button
th:attr="onclick=|connectToRandom('${user.nickname}')|">Connect
to random game</button>
    </div>
    <div class="box">
        <input id="game_id">
        <button
th:attr="onclick=|connectToSpecific('${user.nickname}')|">Connec
t to specific game</button>
    </div>
</div>
<div id="boardFrame">

</div>
<script src="js/script.js"></script>
<script src="js/socket_js.js"></script>
</body>

</html>
```

registration.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Registration Page</title>
    <link type="text/css" href="/css/login-style.css"
rel="stylesheet">
</head>

<body>
    <div id="background-login">
        <div>
            <h1>Registration Form</h1>
        </div>
        <div id="window-login">
            <form th:action="@{/registration-result}"
th:object="${chessUserDto}" method="post">
                    <div class="form-div">
                    <label for="nickname">Username:</label>
                    <input type="text" th:board="*{nickname}"
id="nickname" name="nickname" required>
                    <span class="error-span"
th:if="${#fields.hasErrors('nickname')}"
th:errors="*{nickname}"></span>
                    </div>

                    <div class="form-div">
                    <label for="password">Password:</label>
                    <input type="password"
th:board="*{password}" id="password" name="password" required>
                    <span class="error-span"
th:if="${#fields.hasErrors('password')}"
th:errors="*{password}"></span>
                    </div>

                    <div class="form-div">
                    <label for="confirmPassword">Confirm
Password:</label>
                    <input type="password"
th:board="*{confirmPassword}" id="confirmPassword"
name="confirmPassword" required>
                    <span class="error-span"
th:if="${#fields.hasErrors('confirmPassword')}"
th:errors="*{confirmPassword}"></span>
                    </div>

                    <div class="form-div">
                    <label for="email">Email:</label>
                    <input type="email" th:board="*{email}"
```

```
id="email" name="email" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-
z]{2,}$" required>
                    <span class="error-span"
th:if="${#fields.hasErrors('email')}"
th:errors="*{email}"></span>
                    </div>
                    <div class="form-div">
                    <button type="submit" class="register-
button">Register</button>
                    <a th:href="@{/}" class="register-button">Go
Back to Welcome Page</a>
                    </div>
            </form>
        </div>
    </div>
</div>
</body>
</html>
```

registration-successful.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Registration Result</title>
</head>
<body>
        <h1 class="neon-title">Registration Successful</h1>
        <p th:text="'Thank you for joining us, ' +
${user.nickname}"></p>
        <a th:href="@{/}">Go Back</a>
</body>
</html>
```

application.properties

```
spring.application.name=ChessWebapp
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=chess
spring.mongodb.embedded.storage.oplogSize=1024
spring.main.allow-bean-definition-overriding=true
spring.autoconfigure.exclude=org.springframework.boot.autoconfig
ure.mongo.embedded.EmbeddedMongoAutoConfiguration

server.port=8080
server.address=0.0.0.0
```

messages.properties

```
error.user=Passwords do not match
error.password=Invalid password
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>3.3.0</version>
       <relativePath/>
    </parent>
    <groupId>com.kubik</groupId>
    <artifactId>ChessWebapp</artifactId>
    <version>1.0.0</version>
    <name>ChessWebapp</name>
    <description>chess game</description>
    <properties>
       <java.version>17</java.version>
    </properties>
    <dependencies>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-thymeleaf</artifactId>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-
validation</artifactId>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-web</artifactId>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-websocket</artifactId>
       </dependency>
       <dependency>
          <groupId>org.projectlombok</groupId>
          <artifactId>lombok</artifactId>
          <optional>true</optional>
       </dependency>
       <dependency>
```

```
            <groupId>org.modelmapper</groupId>
            <artifactId>modelmapper</artifactId>
            <version>3.2.0</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-
mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```