

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

**Курсова робота
З дисципліни «Конструювання програмного
забезпечення (JAVA)»**

Виконав:

Студент групи ПД-34

Солов'ян Арсен

Київ – 2024

План:

1. Проектування інформаційної системи

- 1.1 Опис User stories з діаграмами послідовностей
- 1.2 Опис ERD бази даних
- 1.3 Опис діаграми класів
- 1.4 Написання тестових сценаріїв

2. Реалізація API

- 2.1 Підготовка бази даних
- 2.2 Створення Spring Boot Application
- 2.3 Створення сутностей (Entities)
- 2.4 Створення репозиторіїв (Repositories)
- 2.5 Створення сервісів (Services)
- 2.6 Створення контролерів (Controllers)
- 2.7 Написання Unit тестів

3. Підключення Swagger, системи авторизації з ролями, Docker Compose для розгортання проекту

- 3.1 Підключення системи авторизації OAuth2
- 3.2 Підключення Swagger UI з детальним описом REST API
- 3.3 Збірка проекту в контейнери Docker

4. Інтеграційне тестування системи

- 4.1 Написання інтеграційних автотестів для всіх User stories з позитивними і негативними сценаріями

5. Демонстрація

6. Висновок

GitHub: <https://github.com/KpoJleBapKa/CourseWorkJava>

1. Проектування інформаційної системи

1.1 Опис User stories з діаграмами послідовностей

User Story:

Як гравець у грі World of Tanks, я хочу мати можливість знайти статистику іншого гравця, щоб оцінити його навички.

Acceptance criteria

Якщо я на сторінці "Знайти статистику Гравця" введу нік-нейм гравця, та натисну кнопку "Розрахувати", то застосунок повинен відповісти з інформацією про статистику гравця з вказаним нік-неймом.

Інформація про гравця повинна містити його клан, загальну кількість боїв, відсоток перемог, відсоток влучень, середню шкоду за бій, середній досвід за бій, максимальну кількість знищених танків за бій та максимальний досвід за бій.

Якщо ім'я гравця не введено або не знайдено, застосунок повинен відповісти повідомленням "Ви не ввели нік-нейм гравця" або "Такого гравця не існує".

Інформація повинна бути представлена у зручному форматі для користувача, легко зрозумілому та зчитуваному.

User Story:

Як гравець у грі World of Tanks, я хочу мати можливість дізнатися кількість учасників клану, назву якого я вводжу.

Acceptance criteria

Якщо я на сторінці "Знайти статистику Клану" введу назву або тег клану, та натисну кнопку "Розрахувати", то застосунок повинен відповісти з інформацією про кількість гравців клану за вказаною назвою/тегом.

Інформація про клан повинна містити його назву, та кількість учасників клану.

Якщо назву/тег клану не введено або не знайдено, то застосунок повинен відповісти повідомленням "Ви не ввели назву клану" або "Такого клану не існує".

Інформація повинна бути представлена у зручному форматі для користувача, легко зрозумілому та зчитуваному.

User Story:

Як гравець у грі World of Tanks, я хочу мати можливість розрахувати кількість боїв на результат, щоб покращити свої ігровий досвід.

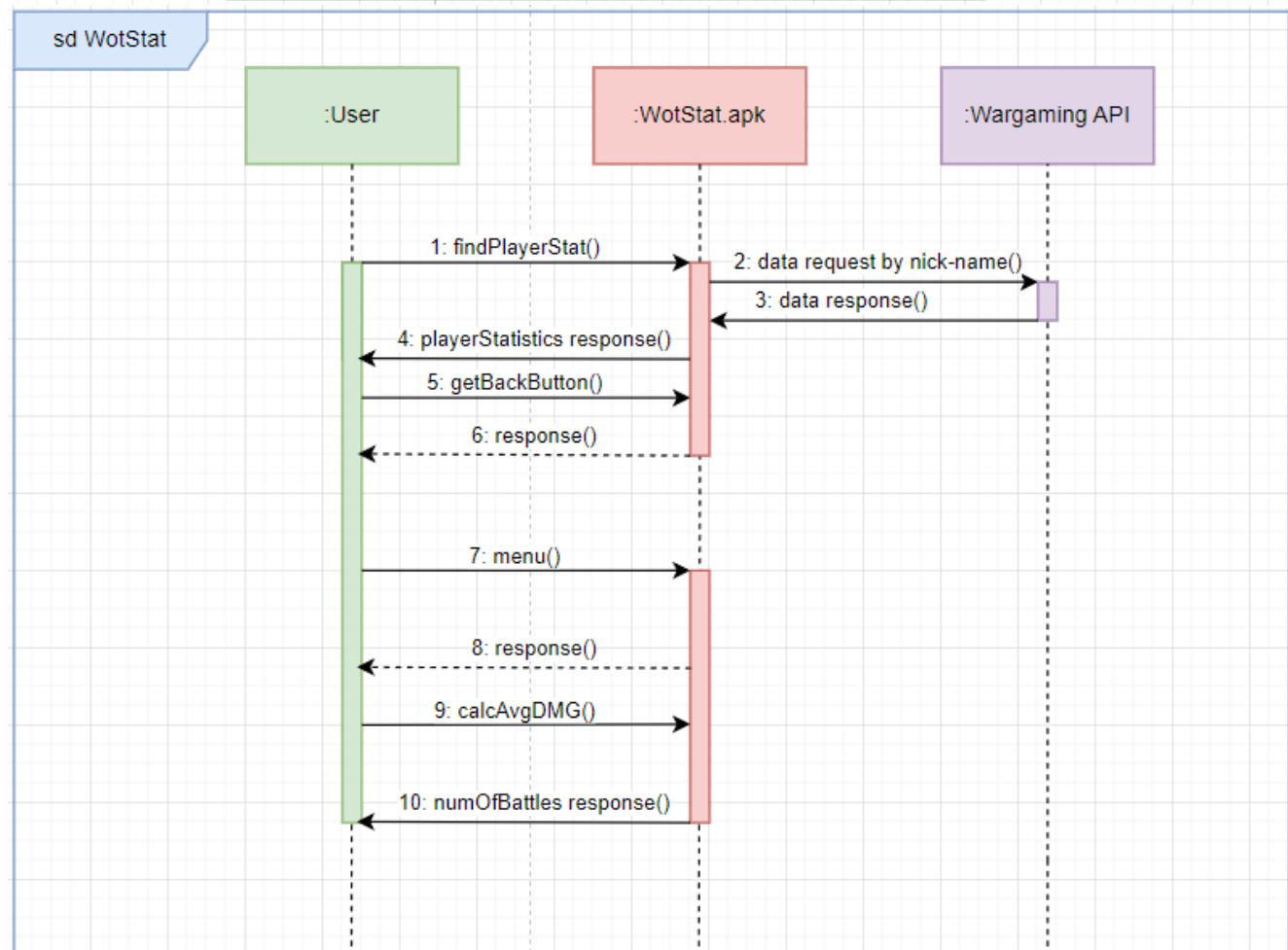
Acceptance criteria

Якщо я на сторінці "Меню" виберу 1 з 4 пунктів, введу дані, та натисну кнопку "Розрахувати", то застосунок повинен прокалькулювати ці дані, та відповісти з інформацією про кількість боїв з фіксованим результатом задля покращення статистики, яку я запросив.

Інформація повинна містити кількість боїв напроти кожного поля з фіксованим результатом.

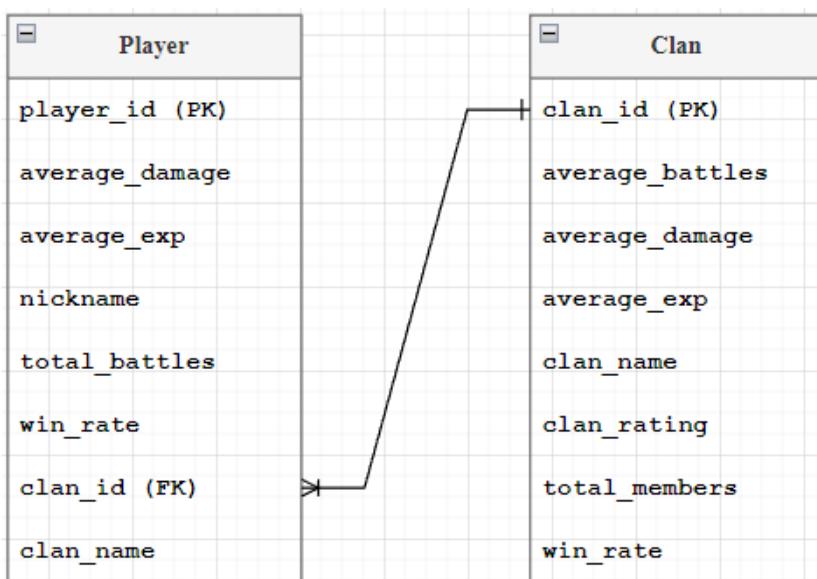
Якщо дані не введено, то застосунок повинен відповісти повідомленням "Заповніть всі поля".

Інформація повинна бути представлена у зручному форматі для користувача, легко зрозумілому та зчитуваному.



1. Проектування інформаційної системи

1.2 Опис ERD бази даних



Сутності та їх атрибути:

Player (Гравець)

player_id (PK) - Унікальний ідентифікатор гравця
average_damage - Середня шкода
average_exp - Середній досвід
nickname - Нікнейм гравця
total_battles - Загальна кількість боїв
win_rate - Відсоток перемог
clan_id (FK) - Ідентифікатор клану (якщо є)
clan_name - Назва клану, в якому перебуває гравець (якщо є)

Clan (Клан)

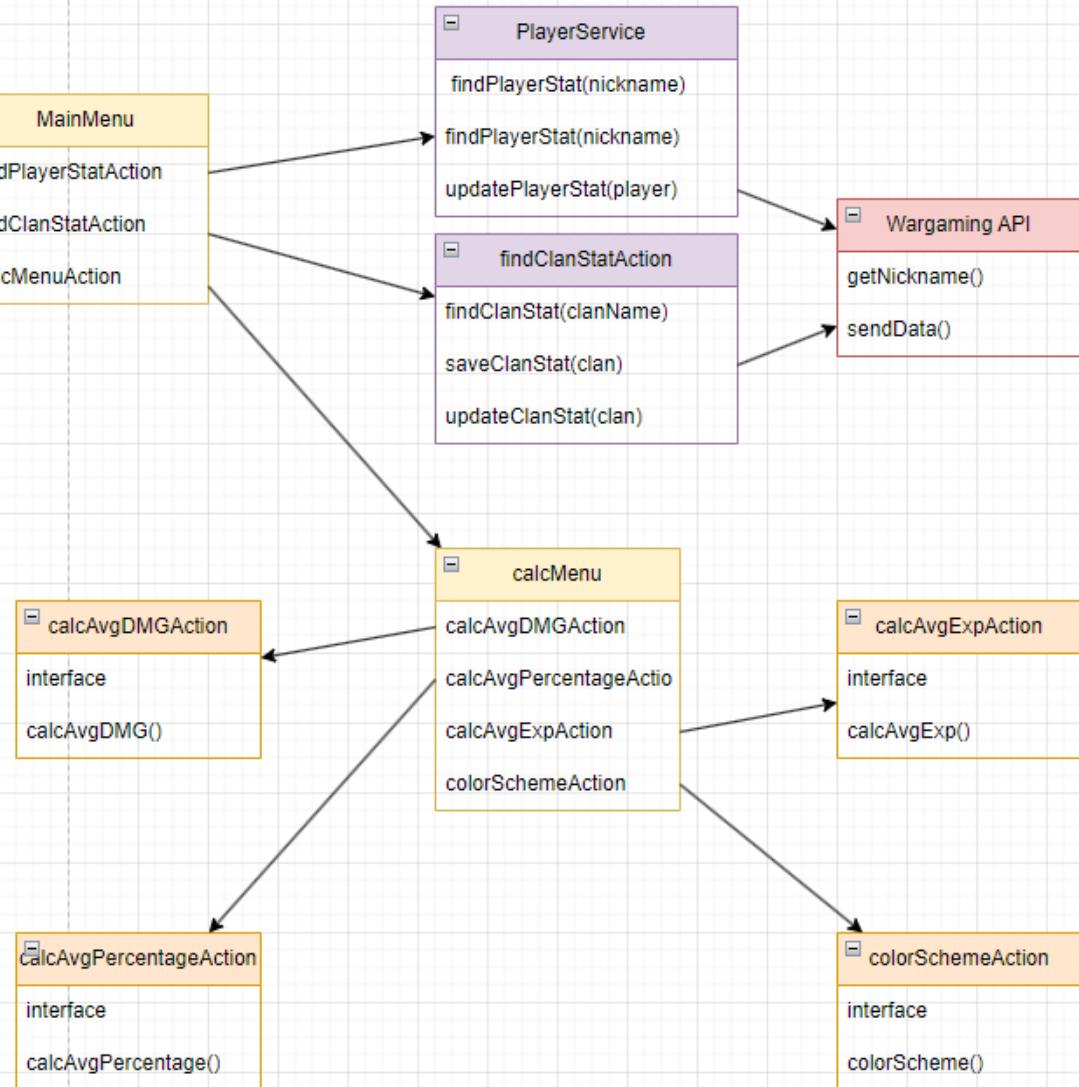
clan_id (PK) - Унікальний ідентифікатор клану
average_battles - Середня кількість боїв гравців клану
average_damage - Середня шкода гравців клану
average_exp - Середній досвід гравців клану
clan_name - Назва клану
clan_rating - Рейтинг клану
total_members - Загальна кількість учасників
win_rate - Відсоток перемог гравців клану

Зв'язки:

Один клан має багато гравців.

1. Проектування інформаційної системи

1.3 Опис діаграмами класів



1.4 Написання тестових сценаріїв

1. Тестовий сценарій: Введення неіснуючого нік-нейму для отримання статистики гравця

- **Кроки:**
 1. Користувач вводить неіснуючий нік-нейм в `findPlayerStatAction`.
- **Очікуваний результат:**
 - Застосунок відповідає повідомленням про те, що такого гравця не існує.

2. Тестовий сценарій: Введення існуючого нік-нейму для отримання статистики гравця

- **Кроки:**
 1. Користувач вводить нік-нейм в `findPlayerStatAction`
- **Очікуваний результат:**
 - Застосунок відповідає статистикою гравця за вказаним нік-неймом.

3. Тестовий сценарій: Перехід на меню калькуляцій

- **Кроки:**
 1. Користувач натискає на кнопку головного меню під назвою "Розрахунки"
- **Очікуваний результат:**
 - Застосунок перекидає користувача на сторінку меню розрахунків.

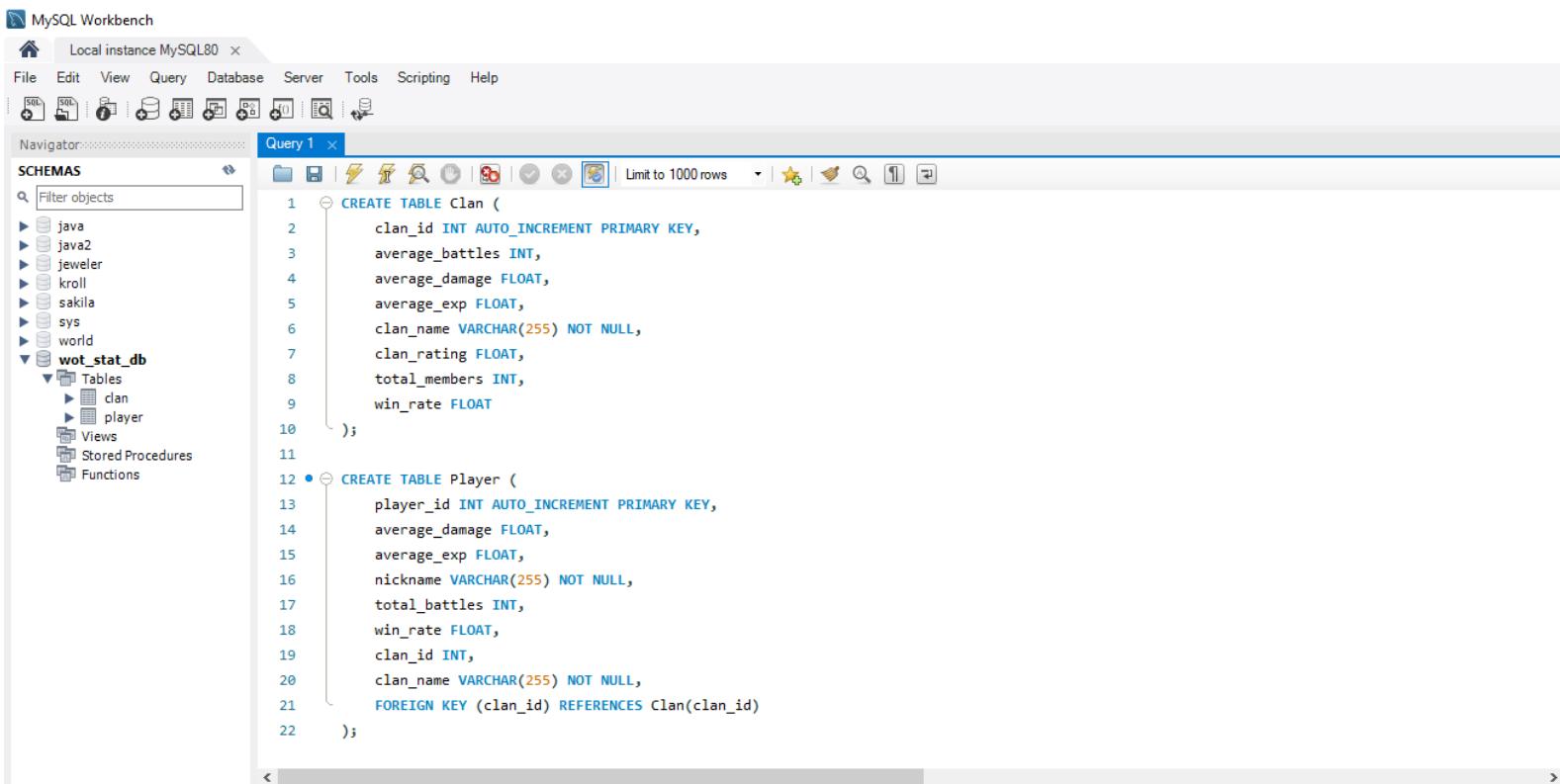
4. Тестовий сценарій: Введення даних на сторінці калькуляції середньої школи

- **Кроки:**
 1. Користувач вводить свою кількість боїв на танку
 2. Користувач вводить свою середню шкоду на танку
 3. Користувач вводить бажану кількість середньої школи на танку
- **Очікуваний результат:**
 - Застосунок калькулює дані, та відображає результат у вигляді кількості боїв на 5000, 4500, 4000, 3500 середньої школи задля покращення (бажаної) середньої шкоди на танку.

2. Реалізація API

2.1 Підготовка бази даних

Створення бази даних та таблиць:



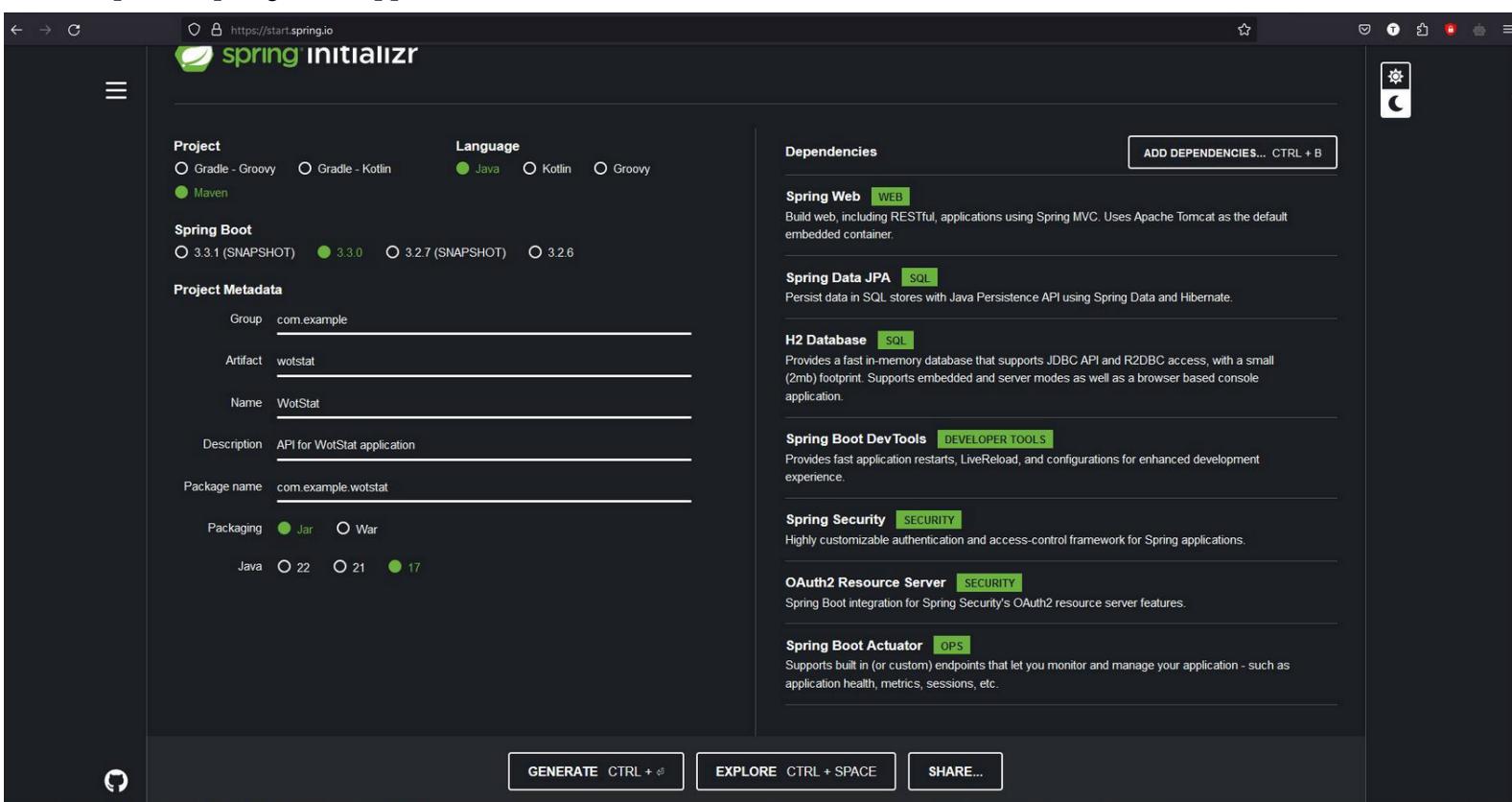
The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local instance MySQL80 contains the following databases: java, java2, jeweler, kroll, sakila, sys, world, and wot_stat_db.
- Tables:** Under wot_stat_db, there are two tables: clan and player.
- Query Editor:** The Query 1 window displays the SQL code for creating the 'Clan' and 'Player' tables.

```
CREATE TABLE Clan (
    clan_id INT AUTO_INCREMENT PRIMARY KEY,
    average_battles INT,
    average_damage FLOAT,
    average_exp FLOAT,
    clan_name VARCHAR(255) NOT NULL,
    clan_rating FLOAT,
    total_members INT,
    win_rate FLOAT
);

CREATE TABLE Player (
    player_id INT AUTO_INCREMENT PRIMARY KEY,
    average_damage FLOAT,
    average_exp FLOAT,
    nickname VARCHAR(255) NOT NULL,
    total_battles INT,
    win_rate FLOAT,
    clan_id INT,
    clan_name VARCHAR(255) NOT NULL,
    FOREIGN KEY (clan_id) REFERENCES Clan(clan_id)
);
```

2.2 Створення Spring Boot Application



The screenshot shows the Spring Initializr web application with the following configuration:

- Project:** Maven
- Language:** Java
- Spring Boot:** 3.3.0 (SNAPSHOT)
- Project Metadata:**
 - Group: com.example
 - Artifact: wotstat
 - Name: WotStat
 - Description: API for WotStat application
 - Package name: com.example.wotstat
 - Packaging: Jar
 - Java: 22
- Dependencies:**
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
 - H2 Database** (SQL): Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.
 - Spring Boot DevTools** (DEVELOPER TOOLS): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
 - Spring Security** (SECURITY): Highly customizable authentication and access-control framework for Spring applications.
 - OAuth2 Resource Server** (SECURITY): Spring Boot integration for Spring Security's OAuth2 resource server features.
 - Spring Boot Actuator** (OPS): Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.
- Buttons:** GENERATE (CTRL + F), EXPLORE (CTRL + SPACE), SHARE...

1. Spring Web - для створення веб-додатків.
2. Spring Data JPA - для роботи з базою даних за допомогою JPA.
3. H2 Database - для використання вбудованої бази даних H2.
4. Spring Boot DevTools - для полегшення розробки з автоматичним перезавантаженням додатка.
5. Spring Security - для реалізації безпеки.
6. OAuth2 Resource Server - для підтримки OAuth2 авторизації.
7. Spring Boot Actuator - для моніторингу та управління додатком.

2. Реалізація API

2.3 Створення сущностей (Entities)

У цьому проекті сущності представляють об'єкти, які будуть зберігатися в базі даних. Вони позначені як класи Java з анотаціями JPA (Java Persistence API), які визначають, як ці об'єкти мають бути відображені в реляційній базі даних.



```
Clan.java
package com.example.wotstat.model;

import jakarta.persistence.*;
import java.util.List;

@Builder
@Entity
public class Clan {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String clanName;
    private int totalMembers;
    private int averageBattles;
    private float winRate;
    private float clanRating;
    private float averageExp;
    private float averageDamage;

    @OneToMany(mappedBy = "clan")
    private List<Player> players;
}

Player.java
package com.example.wotstat.model;

import jakarta.persistence.*;
import java.util.List;

@Builder
@Entity
public class Player {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nickname;
    private int totalBattles;
    private float winRate;
    private float averageDamage;
    private float averageExp;

    @ManyToOne
    @JoinColumn(name = "clan_id")
    private Clan clan;

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getNickname() { return nickname; }

    public void setNickname(String nickname) { this.nickname = nickname; }
}
```

Для кожного поля в сущностях визначені геттери та сеттери, що дозволяє отримувати та встановлювати значення полів.

```
Clan.java
public Long getId() { return id; }

no usages
public void setId(Long id) { this.id = id; }

no usages
public String getClanName() { return clanName; }

3 usages
public void setClanName(String clanName) { this.clanName = clanName; }

no usages
public int getTotalMembers() { return totalMembers; }

4 usages
public void setTotalMembers(int totalMembers) { this.totalMembers = totalMembers; }

no usages
public int getAverageBattles() { return averageBattles; }

4 usages
public void setAverageBattles(int averageBattles) { this.averageBattles = averageBattles; }

no usages
public float getWinRate() { return winRate; }

4 usages
public void setWinRate(float winRate) { this.winRate = winRate; }

no usages
public float getClanRating() { return clanRating; }

4 usages
public void setClanRating(float clanRating) { this.clanRating = clanRating; }

no usages
public float getAverageExp() { return averageExp; }

4 usages
public void setAverageExp(float averageExp) { this.averageExp = averageExp; }

no usages
public String getClanName() { return clanName; }

2 usages
public void setClanName(String clanName) { this.clanName = clanName; }

Player.java
1 usage
public Long getId() { return id; }

no usages
public void setId(Long id) { this.id = id; }

no usages
public String getNickname() { return nickname; }

2 usages
public void setNickname(String nickname) { this.nickname = nickname; }

no usages
public int getTotalBattles() { return totalBattles; }

3 usages
public void setTotalBattles(int totalBattles) { this.totalBattles = totalBattles; }

no usages
public float getWinRate() { return winRate; }

3 usages
public void setWinRate(float winRate) { this.winRate = winRate; }

no usages
public float getAverageDamage() { return averageDamage; }

3 usages
public void setAverageDamage(float averageDamage) { this.averageDamage = averageDamage; }

no usages
public float getAverageExp() { return averageExp; }

3 usages
public void setAverageExp(float averageExp) { this.averageExp = averageExp; }

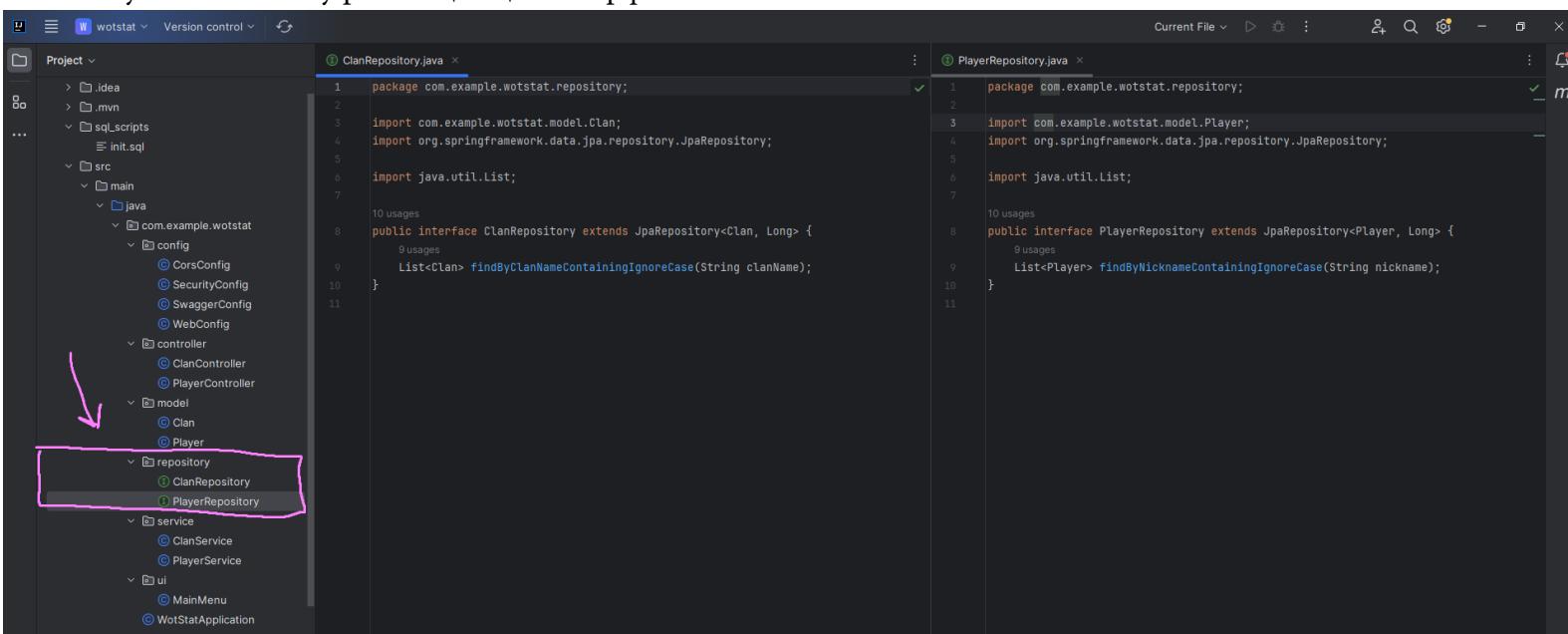
no usages
public String getClanName() { return clanName; }

2 usages
public void setClanName(String clanName) { this.clanName = clanName; }
```

2. Реалізація API

2.4 Створення репозиторіїв (Repositories)

Репозиторії в Spring Data JPA є інтерфейсами, які забезпечують доступ до бази даних та дозволяють виконувати CRUD (Create, Read, Update, Delete) операції без необхідності написання SQL запитів вручну. Spring Data JPA забезпечує автоматичну реалізацію цих інтерфейсів.



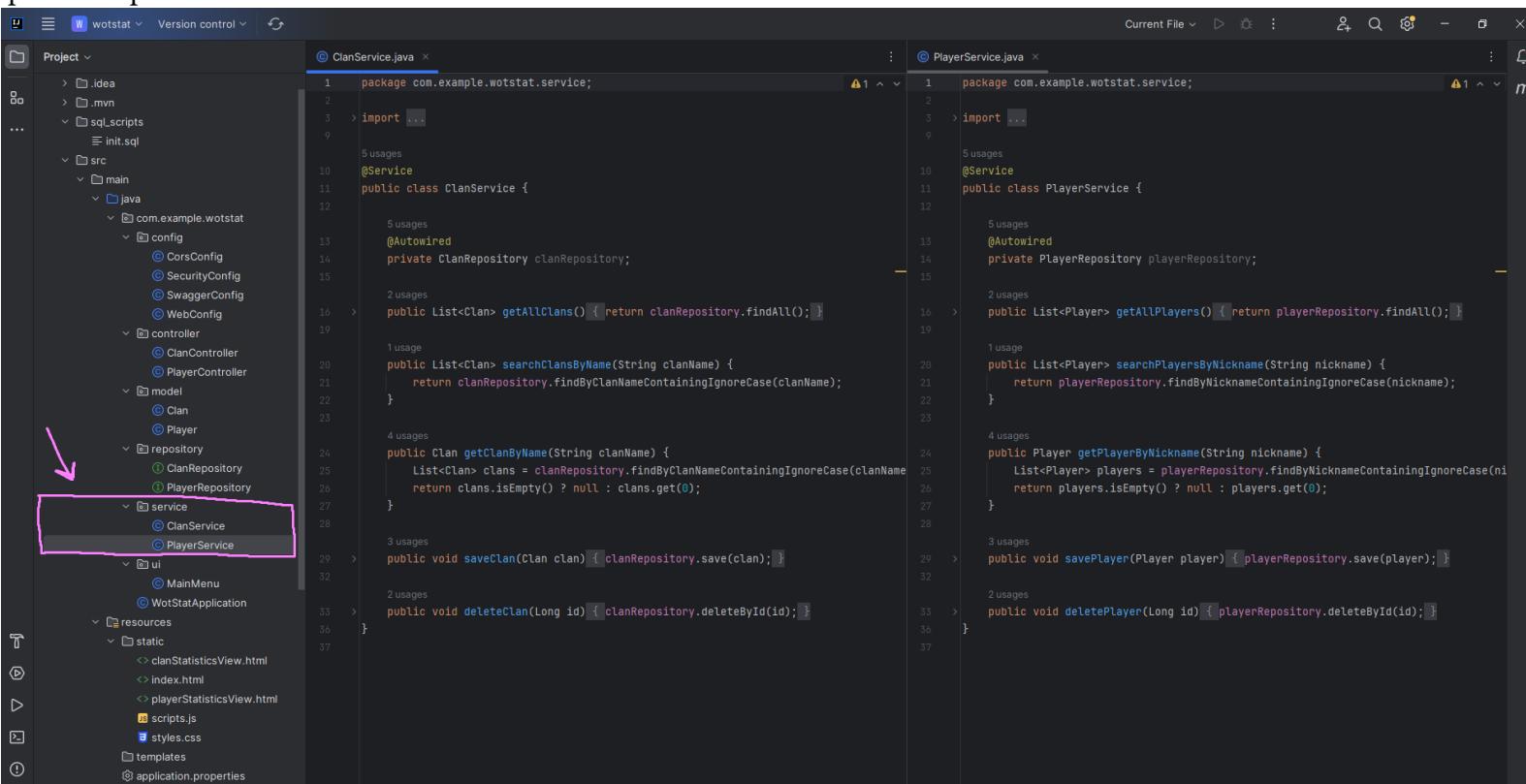
```
ClanRepository.java
1 package com.example.wotstat.repository;
2
3 import com.example.wotstat.model.Clan;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 import java.util.List;
7
8 public interface ClanRepository extends JpaRepository<Clan, Long> {
9     List<Clan> findByClanNameContainingIgnoreCase(String clanName);
10 }
11

PlayerRepository.java
1 package com.example.wotstat.repository;
2
3 import com.example.wotstat.model.Player;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 import java.util.List;
7
8 public interface PlayerRepository extends JpaRepository<Player, Long> {
9     List<Player> findByNicknameContainingIgnoreCase(String nickname);
10 }
11
```

Репозиторії використовуються в сервісах для виконання операцій з базою даних.

2.5 Створення сервісів (Services)

Сервіси (Services) у Spring Framework є важливим компонентом, який відповідає за бізнес-логіку додатку. Вони використовують репозиторії для взаємодії з базою даних і забезпечують абстракцію між контролерами та репозиторіями.



```
ClanService.java
1 package com.example.wotstat.service;
2
3 import ...
4
5 @Service
6 public class ClanService {
7
8     @Autowired
9     private ClanRepository clanRepository;
10
11     public List<Clan> getAllClans() { return clanRepository.findAll(); }
12
13     public List<Clan> searchClansByName(String clanName) {
14         return clanRepository.findByClanNameContainingIgnoreCase(clanName);
15     }
16
17     public Clan getClanByName(String clanName) {
18         List<Clan> clans = clanRepository.findByClanNameContainingIgnoreCase(clanName);
19         return clans.isEmpty() ? null : clans.get(0);
20     }
21
22     public void saveClan(Clan clan) { clanRepository.save(clan); }
23
24     public void deleteClan(Long id) { clanRepository.deleteById(id); }
25
26 }
27
28

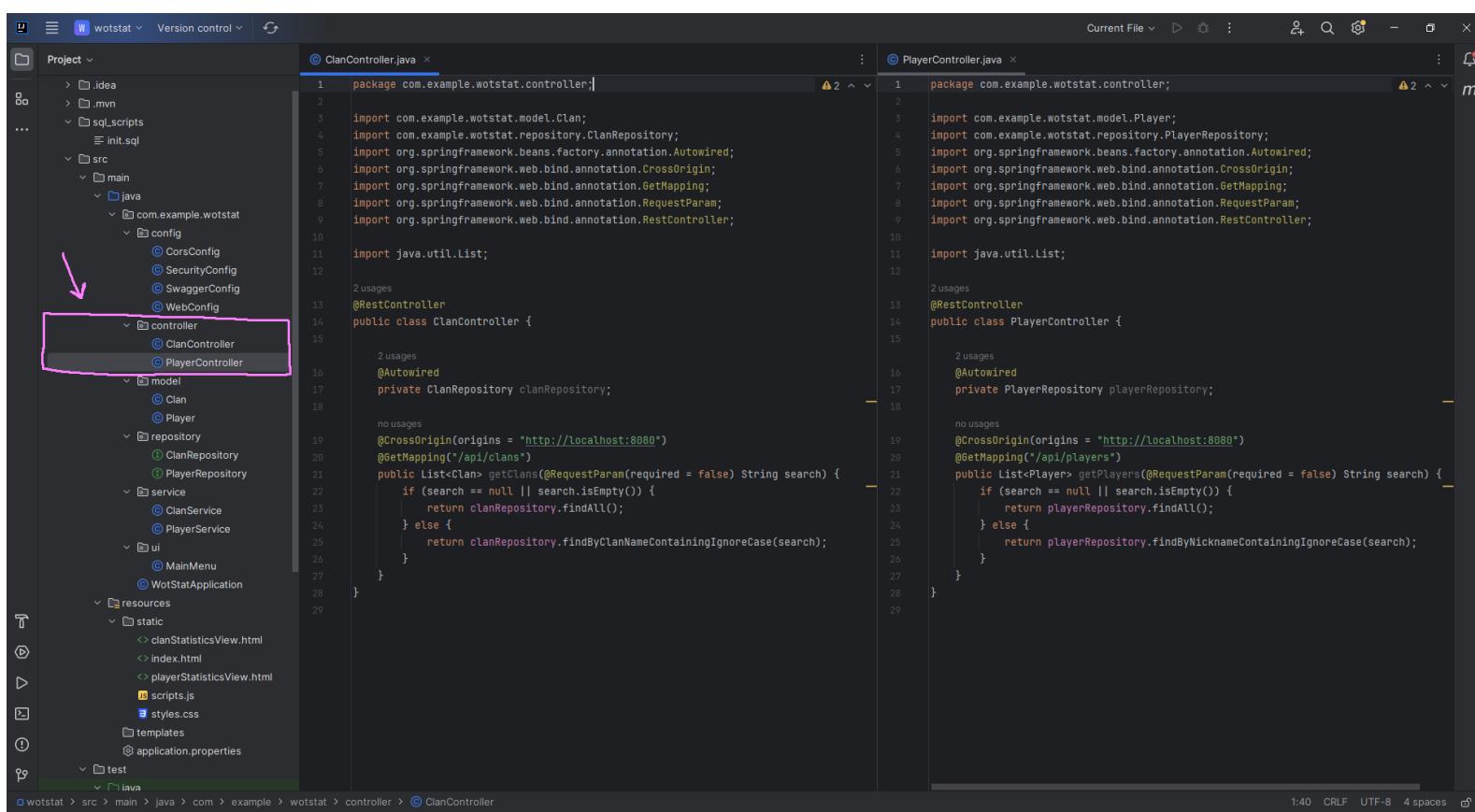
PlayerService.java
1 package com.example.wotstat.service;
2
3 import ...
4
5 @Service
6 public class PlayerService {
7
8     @Autowired
9     private PlayerRepository playerRepository;
10
11     public List<Player> getAllPlayers() { return playerRepository.findAll(); }
12
13     public List<Player> searchPlayersByNickname(String nickname) {
14         return playerRepository.findByNicknameContainingIgnoreCase(nickname);
15     }
16
17     public Player getPlayerByNickname(String nickname) {
18         List<Player> players = playerRepository.findByNicknameContainingIgnoreCase(nickname);
19         return players.isEmpty() ? null : players.get(0);
20     }
21
22     public void savePlayer(Player player) { playerRepository.save(player); }
23
24     public void deletePlayer(Long id) { playerRepository.deleteById(id); }
25
26 }
27
28
```

Також, сервіси використовуються в контролерах для обробки HTTP-запитів і виконання відповідних операцій з базою даних через сервіси.

2. Реалізація API

2.6 Створення контролерів (Controllers)

Контролери (Controllers) у Spring Framework відповідають за обробку HTTP-запитів і повернення відповідей клієнту. Вони є частиною архітектури MVC (Model-View-Controller) і взаємодіють із сервісами для виконання бізнес-логіки.



```
ClanController.java
package com.example.wotstat.controller;

import com.example.wotstat.model.Clan;
import com.example.wotstat.repository.ClanRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;

@.RestController
public class ClanController {

    @Autowired
    private ClanRepository clanRepository;

    no usages
    @CrossOrigin(origins = "http://localhost:8080")
    @GetMapping("/api/clans")
    public List<Clan> getClans(@RequestParam(required = false) String search) {
        if (search == null || search.isEmpty()) {
            return clanRepository.findAll();
        } else {
            return clanRepository.findByClanNameContainingIgnoreCase(search);
        }
    }
}

PlayerController.java
package com.example.wotstat.controller;

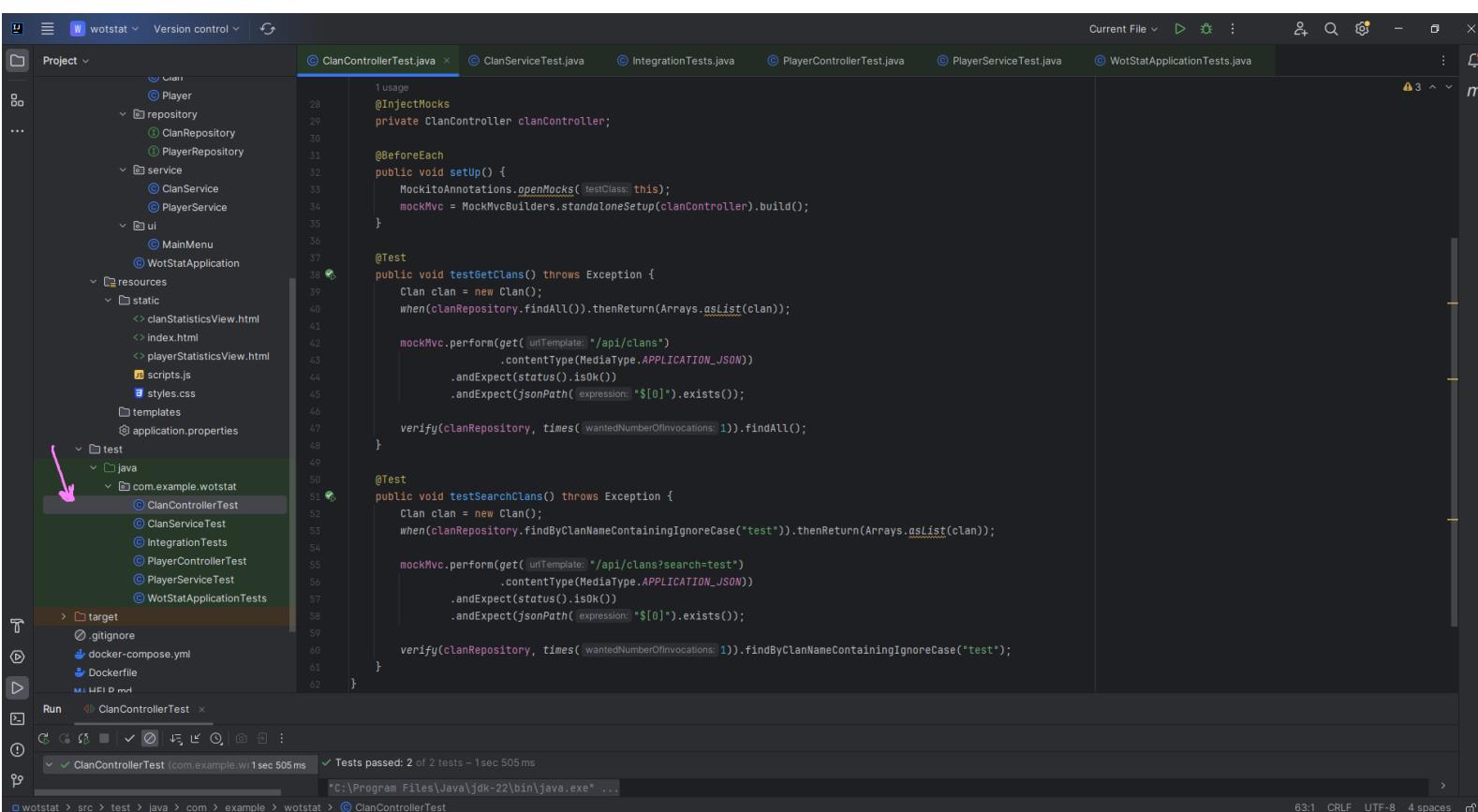
import com.example.wotstat.model.Player;
import com.example.wotstat.repository.PlayerRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;

@.RestController
public class PlayerController {

    @Autowired
    private PlayerRepository playerRepository;

    no usages
    @CrossOrigin(origins = "http://localhost:8080")
    @GetMapping("/api/players")
    public List<Player> getPlayers(@RequestParam(required = false) String search) {
        if (search == null || search.isEmpty()) {
            return playerRepository.findAll();
        } else {
            return playerRepository.findByNicknameContainingIgnoreCase(search);
        }
    }
}
```

2.7 Написання Unit тестів



```
ClanControllerTest.java
package com.example.wotstat;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;
import com.example.wotstat.model.Clan;
import com.example.wotstat.repository.ClanRepository;
import com.example.wotstat.service.ClanService;
import com.example.wotstat.ui.MainMenu;
import com.example.wotstat.WotStatApplication;
import static org.mockito.Mockito.when;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest
@MockitoAnnotations.openMocks(this)
@MockMvc = MockMvcBuilders.standaloneSetup(clanController).build();

@Test
public void testGetClans() throws Exception {
    Clan clan = new Clan();
    when(clanRepository.findAll()).thenReturn(Arrays.asList(clan));

    mockMvc.perform(get("/api/clans")
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.exists()"));

    verify(clanRepository, times(wantedNumberOfInvocations: 1)).findAll();
}

@Test
public void testSearchClans() throws Exception {
    Clan clan = new Clan();
    when(clanRepository.findByClanNameContainingIgnoreCase("test")).thenReturn(Arrays.asList(clan));

    mockMvc.perform(get("/api/clans?search=test")
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.exists()"));

    verify(clanRepository, times(wantedNumberOfInvocations: 1)).findByClanNameContainingIgnoreCase("test");
}
```

Тест testGetClans:

Перевіряє, що статус відповіді HTTP є 200 OK.

Перевіряє, що JSON відповідь містить хоча б один елемент.

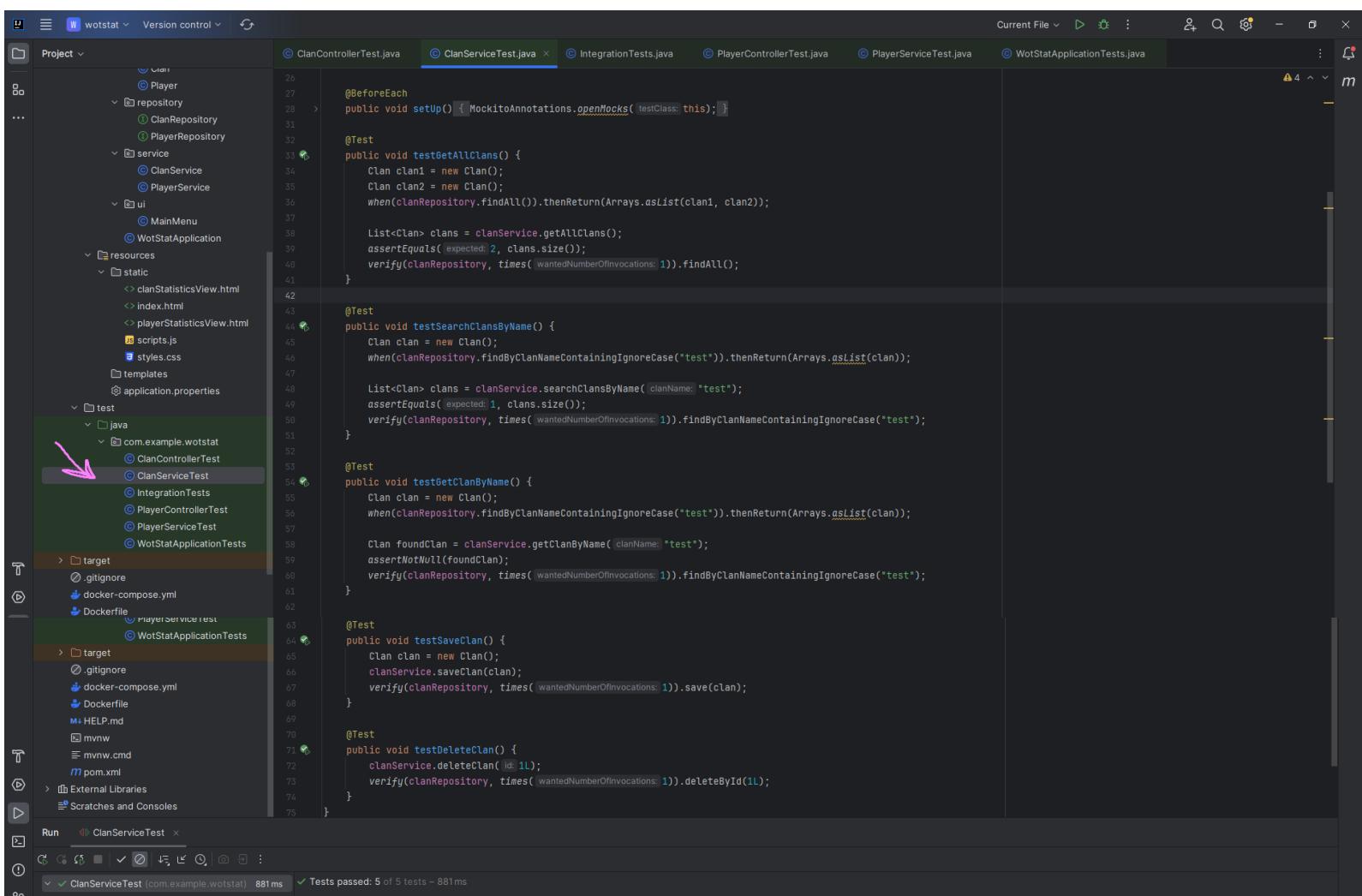
Викликає verify(clanRepository, times(1)).findAll(), щоб переконатися, що метод findAll() був викликаний один раз.

Тест testSearchClans:

Перевіряє, що статус відповіді HTTP є 200 OK.

Перевіряє, що JSON відповідь містить хоча б один елемент.

Викликає verify(clanRepository, times(1)).findByClanNameContainingIgnoreCase("test"), щоб переконатися, що метод findByClanNameContainingIgnoreCase() був викликаний один раз.



```
Project: wotstat
  - ClanControllerTest.java
  - ClanServiceTest.java (selected)
  - IntegrationTests.java
  - PlayerControllerTest.java
  - PlayerServiceTest.java
  - WotStatApplicationTests.java

  - Clan
    - Player
      - repository
        - ClanRepository
        - PlayerRepository
      - service
        - ClanService
        - PlayerService
      - ui
        - MainMenu
        - WotStatApplication
      - resources
        - static
          - clanStatisticsView.html
          - index.html
          - playerStatisticsView.html
          - scripts.js
          - styles.css
        - templates
          - application.properties
    - test
      - java
        - com.example.wotstat
          - ClanControllerTest
          - ClanServiceTest (highlighted with a pink arrow)
          - IntegrationTests
          - PlayerControllerTest
          - PlayerServiceTest
          - WotStatApplicationTests
```

```
ClanServiceTest.java
  - @BeforeEach
    public void setUp() { MockitoAnnotations.openMocks(this); }

  - @Test
    public void testGetAllClans() {
      Clan clan1 = new Clan();
      Clan clan2 = new Clan();
      when(clanRepository.findAll()).thenReturn(Arrays.asList(clan1, clan2));

      List<Clan> clans = clanService.getAllClans();
      assertEquals(expected: 2, clans.size());
      verify(clanRepository, times(wantedNumberOfInvocations: 1)).findAll();
    }

  - @Test
    public void testSearchClansByName() {
      Clan clan = new Clan();
      when(clanRepository.findByClanNameContainingIgnoreCase("test")).thenReturn(Arrays.asList(clan));

      List<Clan> clans = clanService.searchClansByName(clanName: "test");
      assertEquals(expected: 1, clans.size());
      verify(clanRepository, times(wantedNumberOfInvocations: 1)).findByClanNameContainingIgnoreCase("test");
    }

  - @Test
    public void testGetClanByName() {
      Clan clan = new Clan();
      when(clanRepository.findByClanNameContainingIgnoreCase("test")).thenReturn(Arrays.asList(clan));

      Clan foundClan = clanService.getClanByName(clanName: "test");
      assertNotNull(foundClan);
      verify(clanRepository, times(wantedNumberOfInvocations: 1)).findByClanNameContainingIgnoreCase("test");
    }

  - @Test
    public void testSaveClan() {
      Clan clan = new Clan();
      clanService.saveClan(clan);
      verify(clanRepository, times(wantedNumberOfInvocations: 1)).save(clan);
    }

  - @Test
    public void testDeleteClan() {
      clanService.deleteClan(id: 1L);
      verify(clanRepository, times(wantedNumberOfInvocations: 1)).deleteById(1L);
    }
}
```

Run: ClanServiceTest

Tests passed: 5 of 5 tests - 881ms

Тест testGetAllClans:

Перевіряє, що список кланів містить два елементи.

Викликає verify(clanRepository, times(1)).findAll(), щоб переконатися, що метод findAll() був викликаний один раз.

Тест testSearchClansByName:

Перевіряє, що список кланів містить один елемент.

Викликає verify(clanRepository, times(1)).findByClanNameContainingIgnoreCase("test"), щоб переконатися, що метод findByClanNameContainingIgnoreCase() був викликаний один раз.

Тест testGetClanByName:

Перевіряє, що знайдений клан не є null.

Викликає verify(clanRepository, times(1)).findByClanNameContainingIgnoreCase("test"), щоб переконатися, що метод findByClanNameContainingIgnoreCase() був викликаний один раз.

Тест testSaveClan:

Перевіряє, що клан зберігся.

Викликає verify(clanRepository, times(1)).save(clan), щоб переконатися, що метод save() був викликаний один раз.

Тест testDeleteClan:

Перевіряє, що клан видалився.

Викликає verify(clanRepository, times(1)).deleteById(1L), щоб переконатися, що метод deleteById() був викликаний один раз.

```
Project: WotStatApplication
File: PlayerControllerTest.java

public class PlayerControllerTest {
    @Test
    public void testGetPlayers() throws Exception {
        Player player = new Player();
        when(playerRepository.findAll()).thenReturn(Arrays.asList(player));

        mockMvc.perform(get("/api/players")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.exists()"));

        verify(playerRepository, times(1)).findAll();
    }

    @Test
    public void testSearchPlayers() throws Exception {
        Player player = new Player();
        when(playerRepository.findByNicknameContainingIgnoreCase("test")).thenReturn(Arrays.asList(player));

        mockMvc.perform(get("/api/players?search=test")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.exists()"));

        verify(playerRepository, times(1)).findByNicknameContainingIgnoreCase("test");
    }
}
```

Run: PlayerControllerTest

Tests passed: 2 of 2 tests – 1sec 503ms

Тест testGetPlayers:

Перевіряє, що статус відповіді HTTP є 200 OK.

Перевіряє, що JSON відповідь містить хоча б один елемент.

Викликає verify(playerRepository, times(1)).findAll(), щоб переконатися, що метод findAll() був викликаний один раз.

Тест testSearchPlayers:

Перевіряє, що статус відповіді HTTP є 200 OK.

Перевіряє, що JSON відповідь містить хоча б один елемент.

Викликає verify(playerRepository, times(1)).findByNicknameContainingIgnoreCase("test"), щоб переконатися, що метод findByNicknameContainingIgnoreCase() був викликаний один раз.

```
Project: WotStatApplication
File: PlayerServiceTest.java

public class PlayerServiceTest {
    @Test
    public void testGetAllPlayers() {
        Player player1 = new Player();
        Player player2 = new Player();
        when(playerRepository.findAll()).thenReturn(Arrays.asList(player1, player2));

        List<Player> players = playerService.getAllPlayers();
        assertEquals(2, players.size());
        verify(playerRepository, times(1)).findAll();
    }

    @Test
    public void testSearchPlayersByNickname() {
        Player player = new Player();
        when(playerRepository.findByNicknameContainingIgnoreCase("test")).thenReturn(Arrays.asList(player));

        List<Player> players = playerService.searchPlayersByNickname("test");
        assertEquals(1, players.size());
        verify(playerRepository, times(1)).findByNicknameContainingIgnoreCase("test");
    }

    @Test
    public void testGetPlayerByNickname() {
        Player player = new Player();
        when(playerRepository.findByNicknameContainingIgnoreCase("test")).thenReturn(Arrays.asList(player));

        Player foundPlayer = playerService.getPlayerByNickname("test");
        assertNotNull(foundPlayer);
        verify(playerRepository, times(1)).findByNicknameContainingIgnoreCase("test");
    }

    @Test
    public void testSavePlayer() {
        Player player = new Player();
        playerService.savePlayer(player);
        verify(playerRepository, times(1)).save(player);
    }
}
```

```
com.example.wotstat
  ClanControllerTest
  ClanServiceTest
  IntegrationTests
  PlayerControllerTest
  PlayerServiceTest
  WotStatApplicationTests

target
  .gitignore
  docker-compose.yml
  Dockerfile
  LIFI.D.mvnt

Run PlayerServiceTest ×
  Tests passed: 5 of 5 tests - 1 sec 57 ms
  *C:\Program Files\Java\jdk-22\bin\java.exe* ...
```

Test testGetAllPlayers:

Перевіряє, що список гравців містить два елементи.

Викликає verify(playerRepository, times(1)).findAll(), щоб переконатися, що метод findAll() був викликаний один раз.

Test testSearchPlayersByNickname:

Перевіряє, що список гравців містить один елемент.

Викликає verify(playerRepository, times(1)).findByNicknameContainingIgnoreCase("test"), щоб переконатися, що метод findByNicknameContainingIgnoreCase() був викликаний один раз.

Test testGetPlayerByNickname:

Перевіряє, що знайдений гравець не є null.

Викликає verify(playerRepository, times(1)).findByNicknameContainingIgnoreCase("test"), щоб переконатися, що метод findByNicknameContainingIgnoreCase() був викликаний один раз.

Test testSavePlayer:

Перевіряє, чи зберігся гравець.

Викликає verify(playerRepository, times(1)).save(player), щоб переконатися, що метод save() був викликаний один раз.

Test testDeletePlayer:

Перевіряє, чи видалився гравець.

Викликається verify(playerRepository, times(1)).deleteById(1L), щоб переконатися, що метод deleteById() був викликаний один раз.

3.1 Підключення системи авторизації OAuth2

OAuth client created

The client ID and secret can always be accessed from Credentials in APIs & Services

Info OAuth access is restricted to the [test users](#) listed on your [OAuth consent screen](#)

Client ID	1076523933284-cag1iqi5ltj7tqq8k2b864lo4l82p6f.apps.googleusercontent.com	
Client secret	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	
Creation date	May 26, 2024 at 3:43:55 PM GMT+3	
Status	Enabled	

[DOWNLOAD JSON](#)

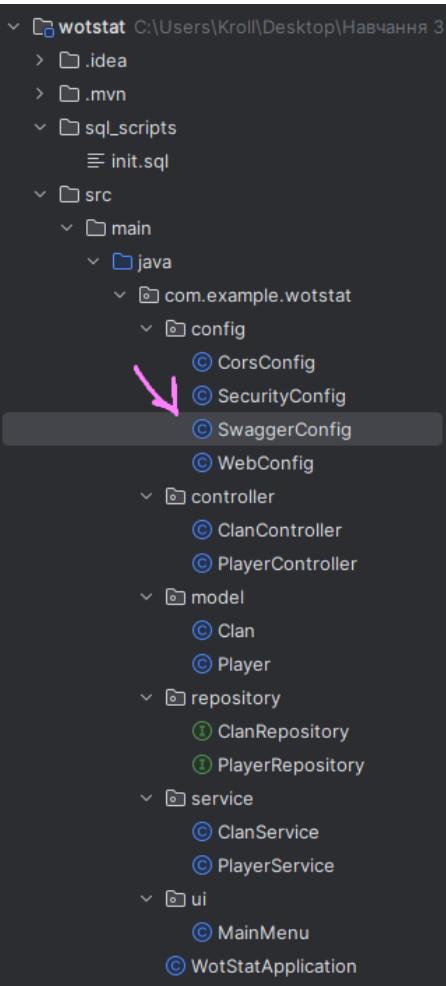
Система авторизації OAuth забезпечує безпечний доступ до додатку. Це дозволяє користувачам аутентифікуватися за допомогою сторонніх провайдерів (в даному випадку Google), що значно підвищує безпеку і зручність використання.

OK

The screenshot shows the Google Cloud Platform API & Services interface for managing OAuth clients. The left sidebar has a 'Credentials' section selected. The main area displays the configuration for a client named 'WotStat Web App'. It includes fields for 'Client ID' (1076523933284-cag1iqi5ltj7tqq8k2b864lo4l82p6f.apps.googleusercontent.com), 'Client secret' (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), 'Creation date' (May 26, 2024 at 3:43:55 PM GMT+3), and 'Status' (Enabled). Below these, there are sections for 'Authorized JavaScript origins' (with a field for 'http://localhost:8080/login/oauth2/code/google') and 'Authorized redirect URIs' (with a field for 'URIs 1' containing 'http://localhost:8080/login/oauth2/code/google'). At the bottom, there are 'SAVE' and 'CANCEL' buttons.

3.2 Підключення Swagger UI з детальним описом REST API

Swagger дозволяє автоматично генерувати документацію для API, що спрощує процес розробки і тестування.



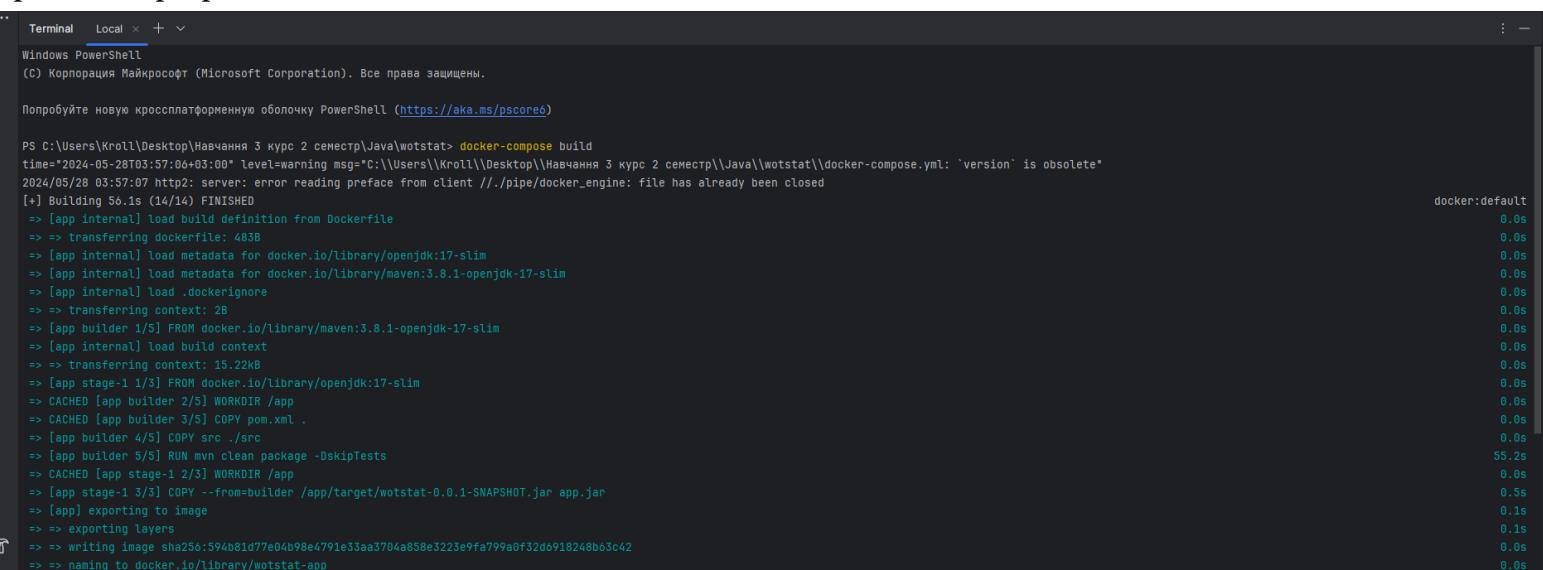
```
1 package com.example.wotstat.config;
2
3 import io.swagger.v3.oas.models.OpenAPI;
4 import io.swagger.v3.oas.models.info.Info;
5 import org.springdoc.core.GroupedOpenApi;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8
9 no usages
10 @Configuration
11 public class SwaggerConfig {
12
13     no usages
14     @Bean
15     public OpenAPI customOpenAPI() {
16         return new OpenAPI()
17             .info(new Info()
18                 .title("WotStat API")
19                 .version("1.0")
20                 .description("API для управління статистикою кланів та гравців"));
21
22     no usages
23     @Bean
24     public GroupedOpenApi publicApi() {
25         return GroupedOpenApi.builder()
26             .group("public")
27             .pathsToMatch("/api/**")
28             .build();
29 }
```

OpenAPI - це центральний об'єкт, який містить всю інформацію про API.

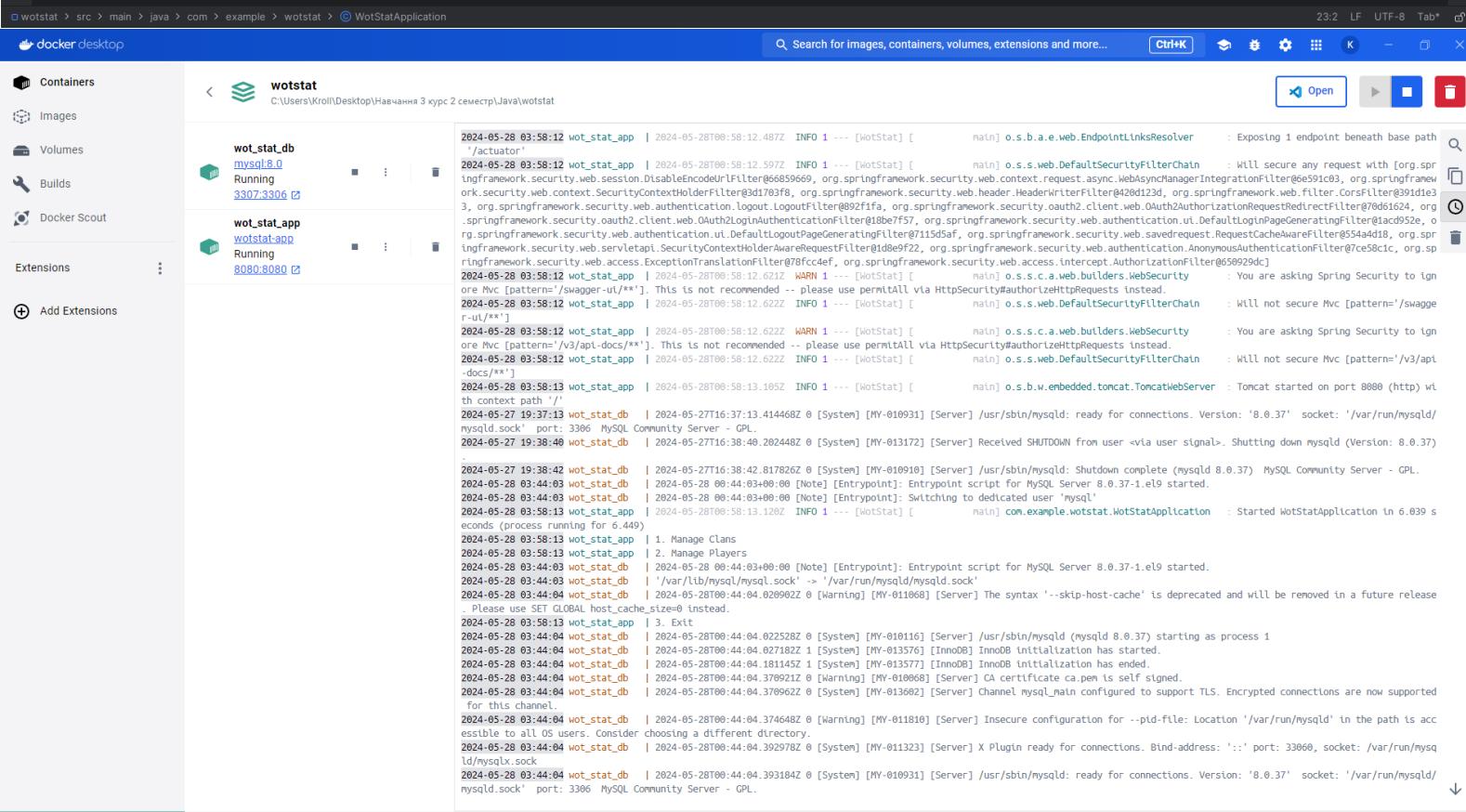
GroupedOpenApi - це об'єкт, який дозволяє групувати і фільтрувати ендпоїнти для документування.

3.3 Збірка проекту в контейнери Docker

Збірка проекту в контейнери Docker дозволяє ізолювати додаток і його залежності, забезпечуючи консистентність середовища розробки.



```
PS C:\Users\Kroll\Desktop\Навчання 3 курс 2 семестр\Java\wotstat> docker-compose build
time=2024-05-28T03:57:04+03:00 level=warning msg="C:\\Users\\Kroll\\Desktop\\\\Навчання 3 курс 2 семестр\\Java\\wotstat\\docker-compose.yml: 'version' is obsolete"
2024/05/28 03:57:07 http2: server: error reading preface from client //./pipe/docker_engine: file has already been closed
[+] Building 56.1s (14/14) FINISHED                                            docker:default
 => [app internal] load build definition from Dockerfile                  0.0s
=> => transferring dockerfile: 483B                                         0.0s
=> [app internal] load metadata for docker.io/library/openjdk:17-slim    0.0s
=> [app internal] load metadata for docker.io/library/maven:3.8.1-openjdk-17-slim 0.0s
=> [app internal] load .dockerignore                                       0.0s
=> => transferring context: 28                                         0.0s
=> [app builder 1/5] FROM docker.io/library/maven:3.8.1-openjdk-17-slim   0.0s
=> [app internal] load build context                                     0.0s
=> => transferring context: 15.22kB                                      0.0s
=> [app stage-1 1/3] FROM docker.io/library/openjdk:17-slim              0.0s
=> CACHED [app builder 2/5] WORKDIR /app                                0.0s
=> CACHED [app builder 3/5] COPY pom.xml .                               0.0s
=> [app builder 4/5] COPY src ./src                                     0.0s
=> [app builder 5/5] RUN mvn clean package -DskipTests                55.2s
=> CACHED [app stage-1 2/3] WORKDIR /app                                0.0s
=> [app stage-1 3/3] COPY --from=builder /app/target/wotstat-0.0.1-SNAPSHOT.jar app.jar 0.5s
=> [app] exporting to image                                              0.1s
=> => exporting layers                                                 0.1s
=> => writing image sha256:594b81d77e04b98e4791e33aa3704a858e3223e9fa799a0f32d6918248b63c42 0.0s
=> => naming to docker.io/library/wotstat-app                           0.0s
```



4. Інтеграційне тестування системи

4.1 Написання інтеграційних автотестів для всіх User stories з позитивними і негативними сценаріями

User Story: Отримати статистику гравця

Позитивний сценарій:

Тест testGetPlayerStatistics_Success перевіряє успішне отримання статистики для існуючого гравця.

Негативний сценарій:

Тест testGetPlayerStatistics_PlayerNotFound перевіряє випадок, коли гравець не знайдений (отримання статистики для неіснуючого гравця).

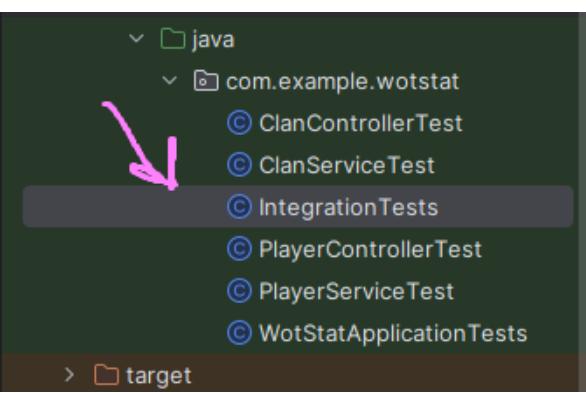
User Story: Отримати статистику клану

Позитивний сценарій:

Тест testGetClanStatistics_Success перевіряє успішне отримання статистики для існуючого клану.

Негативний сценарій:

Тест testGetClanStatistics_ClanNotFound перевіряє випадок, коли клан не знайдений (отримання статистики для неіснуючого клану).



5. Демонстрація

Запускаємо застосунок, та переходимо на <http://localhost:8080/>

Вибираємо статистику клану

Після натискання кнопки пошуку - нічого не стається. В консолі бачимо, що для взаємодії з застосунком - потрібна аутентифікація.

```
⚠️ Шаблон було примусово застосовано перед появним завантаженням сторінки. Якщо таблиця стилі ще не завантажені, це може спричинити flash неініціалізованого виступу.
⚠️ файл cookie "JSESSIONID" не має належного значення атрибути "SameSite". Недорозі файлів cookie без атрибути "SameSite" або з недійсним значенням вважатимуться як "Lax", що означає, що файлів cookie більше не надсилються в сторонніх контекстах. Якщо ваша програма залежить від доступності цих файлів cookie clans в таких контекстах, додайте до них атрибут "SameSite=None", щоб дізнатися більше про атрибут "SameSite", прошуєте https://developer.mozilla.org/docs/Web/HTTP/Headers/Set-Cookie#SameSite
⚠️ Запит зі стороннього джерела заблоковано: Попітка одного джерела не дозволяє читання віддаленого ресурсу на https://accounts.google.com/o/oauth2/v2/auth?response\_type=code&client\_id=1076523933284-cogilqisltjzgpa82b8e4l0id18zpsf.apps.googleusercontent.com&scope=openid%20profile%20email&state=plgrwafiu0mo-rglfuyx7n7m0uymx7x2dr&redirect\_uri=http://localhost:8080/login/oauth2/code/google&nonce=ss8952074016npo0x1t8mso1nmcsegt9tlo. ( причиня: відсутній заголовок CORS 'Access-Control-Allow-Origin', код стану: 302). [\[Зокнайдіть\]]
⚠️ Запит зі стороннього джерела заблоковано: Попітка одного джерела не дозволяє читання віддаленого ресурсу на https://accounts.google.com/o/oauth2/v2/auth?response\_type=code&client\_id=1076523933284-cogilqisltjzgpa82b8e4l0id18zpsf.apps.googleusercontent.com&scope=openid%20profile%20email&state=plgrwafiu0mo-rglfuyx7n7m0uymx7x2dr&redirect\_uri=http://localhost:8080/login/oauth2/code/google&nonce=ss8952074016npo0x1t8mso1nmcsegt9tlo. ( причиня: запин CORS не вавася). код стану: (null). [\[Зокнайдіть\]]
⚠️ > Error loading clan statistics: TypeError: NetworkError when attempting to fetch resource.
URI джерела: moz-extension://e1e1dbafc-aedc-44b1-be88-d47528b621d4/ad-blocker/content.js
URI карты джерела: panels:content.js.map [\[Зокнайдіть\]]
⚠️ Запит зі стороннього джерела заблоковано: Попітка одного джерела не дозволяє читання віддаленого ресурсу на https://accounts.google.com/o/oauth2/v2/auth?response\_type=code&client\_id=1076523933284-cogilqisltjzgpa82b8e4l0id18zpsf.apps.googleusercontent.com&scope=openid%20profile%20email&state=veo-h0z0znjswf7bu9y3620m1f8m0ccv792zysv7t32dr&redirect\_uri=http://localhost:8080/login/oauth2/code/google&nonce=vnxXlb-2b3o1st44-4d1nf7x3scvhpauif9gsFICl. ( причиня: відсутній заголовок CORS 'Access-Control-Allow-Origin', код стану: 302). [\[Зокнайдіть\]]
⚠️ Запит зі стороннього джерела заблоковано: Попітка одного джерела не дозволяє читання віддаленого ресурсу на https://accounts.google.com/o/oauth2/v2/auth?response\_type=code&client\_id=1076523933284-cogilqisltjzgpa82b8e4l0id18zpsf.apps.googleusercontent.com&scope=openid%20profile%20email&state=veo-h0z0znjswf7bu9y3620m1f8m0ccv792zysv7t32dr&redirect\_uri=http://localhost:8080/login/oauth2/code/google&nonce=vnxXlb-2b3o1st44-4d1nf7x3scvhpauif9gsFICl. ( причиня: запин CORS не вавася). код стану: (null). [\[Зокнайдіть\]]
⚠️ > Error loading clan statistics: TypeError: NetworkError when attempting to fetch resource.
```

Проводимо авторизацію

Увійдіть в обліковий запис Google

Виберіть обліковий запис

щоб перейти в додаток WotStat

- Arsen Solovian
toadkillergamer@gmail.com
- Ceh9 Slav
krolvalakasa@gmail.com
- Kroll PWNZ
krollpwnz@gmail.com
- KpoJleBapKa Shopper
kpojlebapo4ka@gmail.com
- Arsen Solovian
kroll.valakasa@gmail.com
- Krol Valakasa
kpojlebapka@gmail.com
- Velaskes Gladiko
velaskesgladiko@gmail.com
- Вибрати інший обліковий запис

Щоб продовжити, ми надамо додатку WotStat ваші ім'я, електронну адресу, налаштування мови й зображення профілю.

Українська Довідка Конфіденційність Умови

Войдите в аккаунт Google

Приложению "WotStat" нужны дополнительные права доступа к вашему аккаунту Google

toadkillergamer@gmail.com

У приложения "WotStat" уже есть некоторые права доступа

Узнайте, к каким сервисам ([Сервисов: 3](#)) у приложения "WotStat" уже есть некоторые права доступа.

Убедитесь в надежности сервиса "WotStat"

Этот сайт или приложение сможет получить доступ к конфиденциальной информации. Ознакомьтесь с условиями использования и политикой конфиденциальности сервиса "WotStat", чтобы узнать, как будут обрабатываться ваши данные. Посмотреть или удалить приложения и сайты с доступом к вашему аккаунту можно на странице [Аккаунт Google](#).

[Подробнее об угрозах безопасности...](#)

Отмена

Продолжить

Русский

Справка Конфиденциальность Условия

Після цих кроків вже можемо взаємодіяти з застосунком

localhost:8080/clanStatisticsView.html

Clan Statistics

Login with Google

Search

Clan Name	Average Battles	Average Damage	Average Experience	Win Rate	Clan Rating	Total Members
MANKI	5629	1077	531	50.26	2654	28

Аналогічно з гравцями

localhost:8080/playerStatisticsView.html

Player Statistics

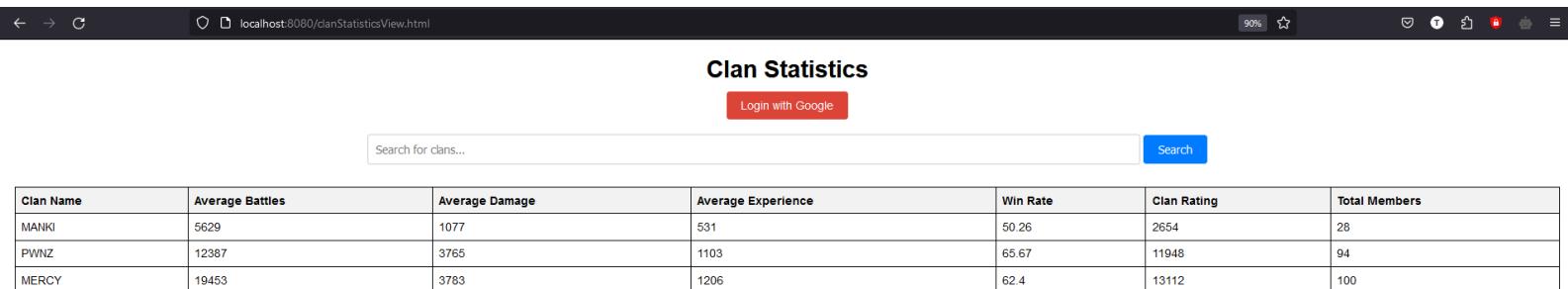
Login with Google

Search

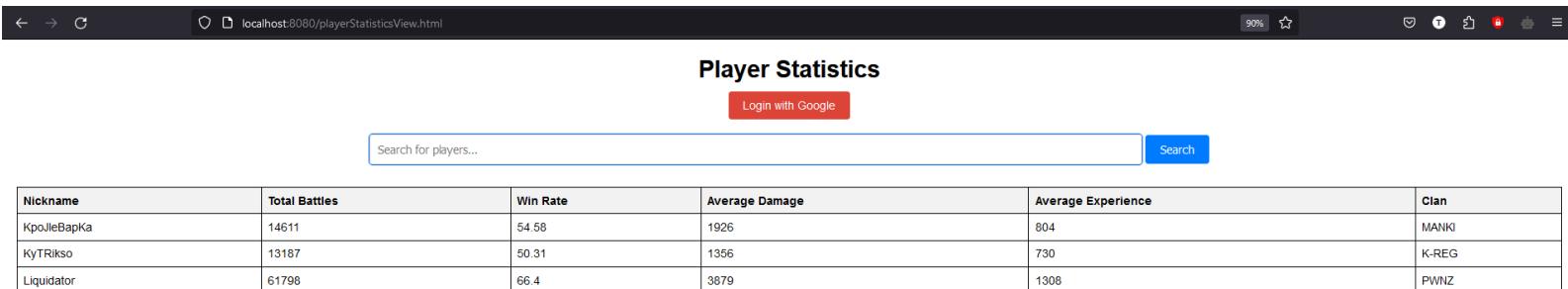
Nickname	Total Battles	Win Rate	Average Damage	Average Experience	Clan
KroJleBapKa	14611	54.58	1926	804	MANKI

Вивід даних:

Якщо нічого не вписати:



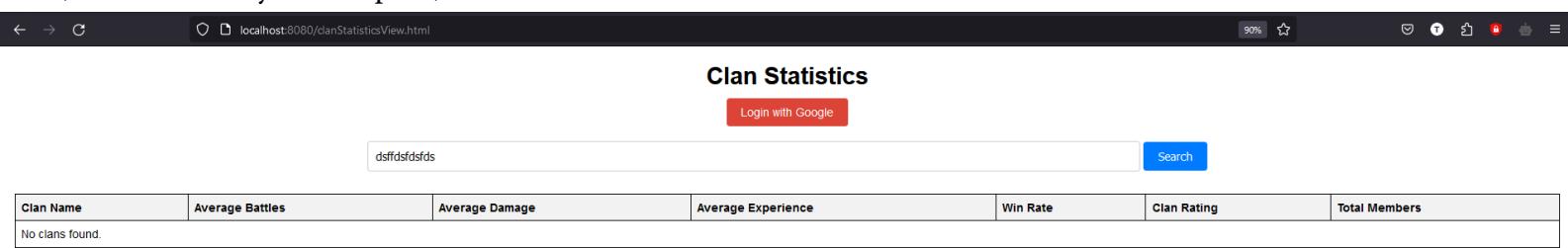
Clan Name	Average Battles	Average Damage	Average Experience	Win Rate	Clan Rating	Total Members
MANKI	5629	1077	531	50.26	2654	28
PWNZ	12387	3765	1103	65.67	11948	94
MERCY	19453	3783	1206	62.4	13112	100



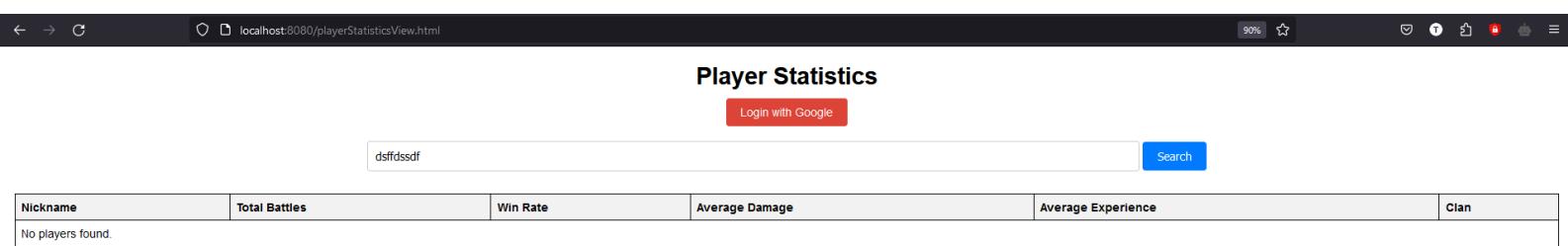
Nickname	Total Battles	Win Rate	Average Damage	Average Experience	Clan
KpolleBapKa	14611	54.58	1926	804	MANKI
KyTRikso	13187	50.31	1356	730	K-REG
Liquidator	61798	66.4	3879	1308	PWNZ

Маємо вивід всіх даних

Якщо ввести неіснуючого гравця/клан:



Clan Name	Average Battles	Average Damage	Average Experience	Win Rate	Clan Rating	Total Members
No clans found.						



Nickname	Total Battles	Win Rate	Average Damage	Average Experience	Clan
No players found.					

7. Висновок

В процесі створення цього проекту було вивчено:

Розробка RESTful API: Створення та впровадження RESTful сервісів для взаємодії з користувачами.

Spring Boot: Використання Spring Boot для швидкої розробки та конфігурації веб-додатків.

Spring Data JPA: Інтеграція з базою даних для збереження та отримання даних через репозиторії.

Інтеграційне тестування: Використання MockMvc для тестування контролерів та перевірки різних сценаріїв роботи API.

Безпека OAuth2: Захист API за допомогою OAuth2 для контролю доступу.

Swagger: Документування API для полегшення розробки та тестування.

CORS: Налаштування CORS для підтримки взаємодії з фронтенд-застосунками.

Docker: Використання Docker та Docker Compose для контейнеризації додатку та спрощення розгортання.

Ці знання допомогли створити надійний, масштабований та безпечний веб-додаток, який відповідає сучасним стандартам розробки.