

**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

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 сущностей (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

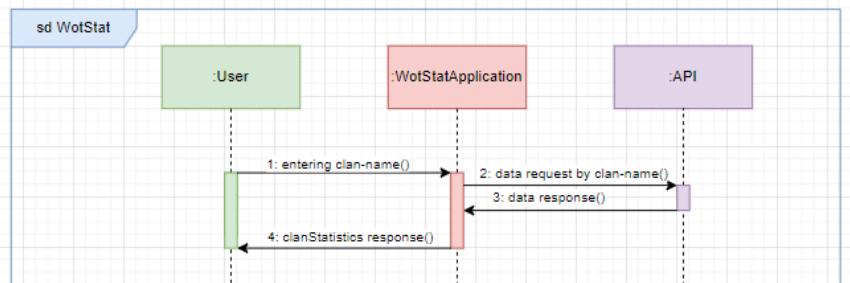
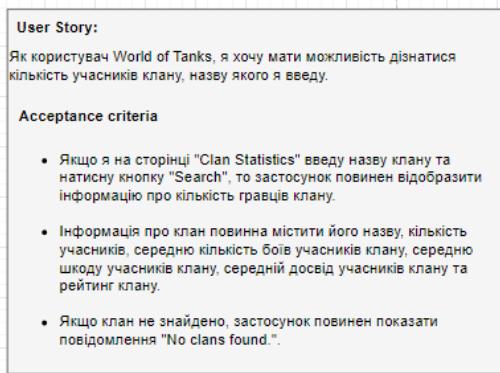
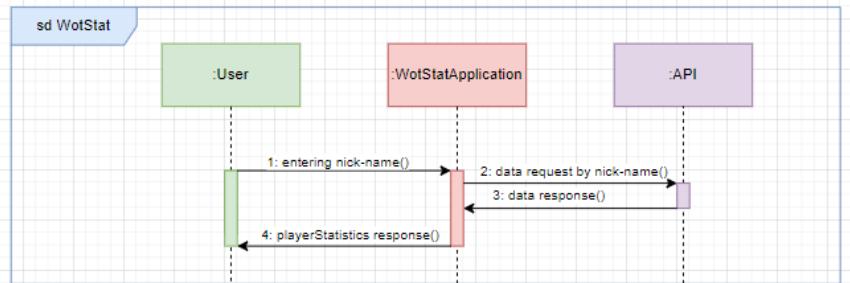
4. Інтеграційне тестування системи

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

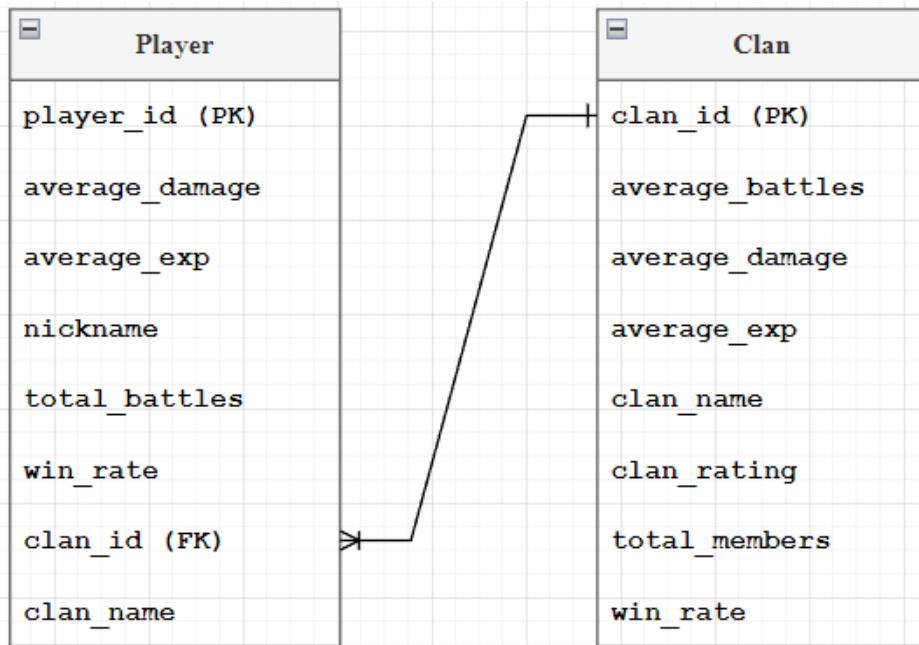
5. Використана література

1. Проектування інформаційної системи

1.1 Опис User stories з діаграмами послідовностей



1.2 Опис ERD бази даних



Дана ERD діаграма описує структуру бази даних для зберігання інформації про гравців та клані в грі World of Tanks. Діаграма складається з двох сутностей: Player (Гравець) та Clan (Клан).

Сутність Player (Гравець)

Сутність Player зберігає інформацію про окремих гравців. Вона містить такі атрибути:

- player_id (PK): Унікальний ідентифікатор гравця.
- average_damage: Середня шкода, нанесена гравцем за бій.
- average_exp: Середній досвід, отриманий гравцем за бій.
- nickname: Нікнейм гравця.
- total_battles: Загальна кількість боїв, зіграних гравцем.
- win_rate: Відсоток перемог гравця.
- clan_id (FK): Ідентифікатор клану, до якого належить гравець.
- clan_name: Назва клану, до якого належить гравець.

Сутність Clan (Клан)

Сутність Clan зберігає інформацію про клани. Вона містить такі атрибути:

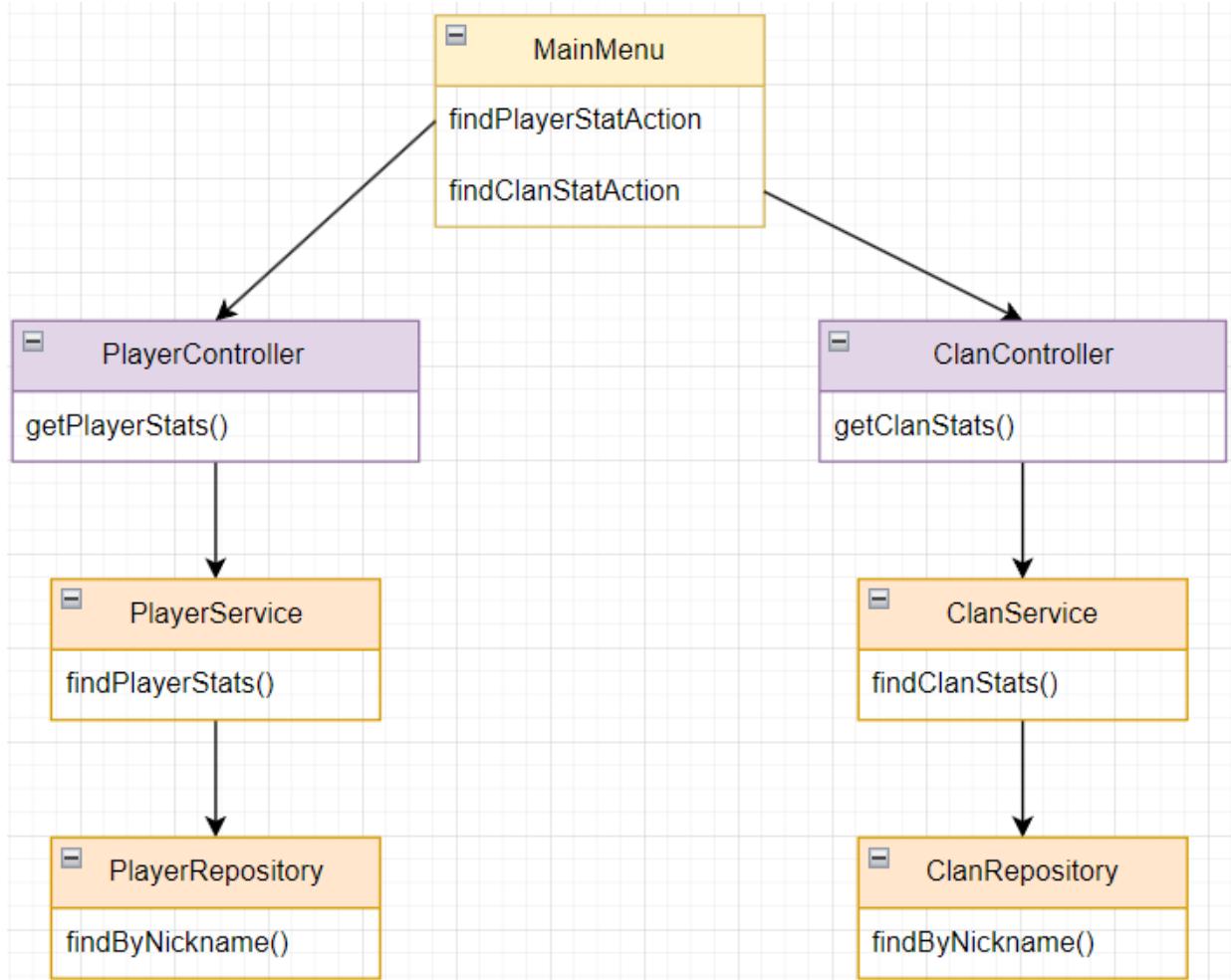
- clan_id (PK): Унікальний ідентифікатор клану.
- average_battles: Середня кількість боїв, зіграних учасниками клану.
- average_damage: Середня шкода, нанесена учасниками клану за бій.
- average_exp: Середній досвід, отриманий учасниками клану за бій.
- clan_name: Назва клану.
- clan_rating: Рейтинг клану.
- total_members: Загальна кількість учасників клану.
- win_rate: Відсоток перемог учасників клану.

Зв'язок між сутностями

Між сутностями Player та Clan існує зв'язок "багато до одного" (many-to-one):

- Один клан може мати багато гравців.
- Кожен гравець може належати тільки до одного клану.

1.3 Опис діаграми класів



Ця діаграма демонструє архітектуру взаємодії між різними компонентами системи для отримання статистики гравців та кланів у застосунку WotStat. Вона відображає, як запити від користувача обробляються через різні рівні контролерів, сервісів та репозиторій.

Компоненти діаграми:

MainMenu

`findPlayerStatAction`: Дія, яка ініціює запит на отримання статистики гравця.

`findClanStatAction`: Дія, яка ініціює запит на отримання статистики клану.

PlayerController

`getPlayerStats()`: Метод контролера, який обробляє запит на отримання статистики гравця.

PlayerService

`getPlayerByNickname()`: Метод сервісу, який виконує бізнес-логіку для отримання статистики гравця.

PlayerRepository

`findByNicknameContainingIgnoreCase()`: Метод репозиторію, який здійснює запит до бази даних для отримання інформації про гравця за його нікнеймом.

ClanController

`getClans()`: Метод контролера, який обробляє запит на отримання статистики клану.

ClanService

`getClanByName()`: Метод сервісу, який виконує бізнес-логіку для отримання статистики клану.

ClanRepository

`findByClanNameContainingIgnoreCase()`: Метод репозиторію, який здійснює запит до бази даних для отримання інформації про клан за його назвою.

1.4 Написання тестових сценаріїв

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

- **Кроки:**
 1. Користувач відкриває сторінку "Player Statistics".
 2. Користувач вводить неіснуючий нікнейм у текстове поле пошуку.
 3. Користувач натискає кнопку "Search".
- **Очікуваний результат:**
 - Застосунок відповідає повідомленням "No players found".

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

- **Кроки:**
 1. Користувач відкриває сторінку "Player Statistics".
 2. Користувач вводить існуючий нікнейм у текстове поле пошуку.
 3. Користувач натискає кнопку "Search".
- **Очікуваний результат:**
 - Застосунок відповідає статистикою гравця за вказаним нікнеймом, включаючи загальну кількість боїв, відсоток перемог, середню шкоду, середній досвід за бій та назву клану, в якому перебуває гравець.

3. Тестовий сценарій: Введення неіснуючої назви для отримання статистики клану

- **Кроки:**
 1. Користувач відкриває сторінку "Clan Statistics".
 2. Користувач вводить неіснуючий клан у текстове поле пошуку.
 3. Користувач натискає кнопку "Search".
- **Очікуваний результат:**
 - Застосунок відповідає повідомленням "No clans found".

4. Тестовий сценарій: Введення існуючої назви для отримання статистики клану

- **Кроки:**
 1. Користувач відкриває сторінку "Clan Statistics".
 2. Користувач вводить існуючий клан у текстове поле пошуку.
 3. Користувач натискає кнопку "Search".
- **Очікуваний результат:**
 - Застосунок відповідає статистикою клану за вказаною назвою, включаючи середню кількість боїв, відсоток перемог, шкоду, досвід гравців цього клану, рейтинг клану, та кількість учасників в цьому клані.

2. Реалізація API

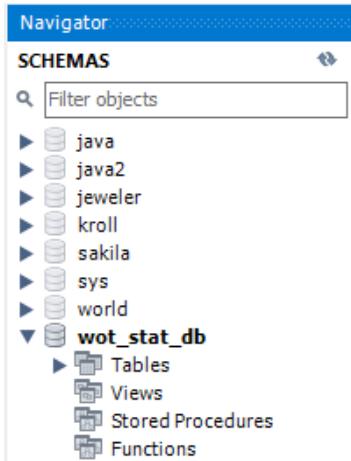
2.1 Підготовка бази даних

Для початку роботи з БД - треба її створити. Для цього проекту, я використовую СУБД mySQL Workbench.

Створюється БД за допомогою наступного запиту:

```
CREATE DATABASE wot_stat_db
```

В результаті - маємо створену БД:



Далі, - локально підключаємось до цієї БД за допомогою команд у файлі application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/wot_stat_db
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

Для контейнера використовуємо інший порт:

```
spring.datasource.url=jdbc:mysql://localhost:3307/wot_stat_db
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

2.2 Створення Spring Boot Application

Заходимо на сайт <https://start.spring.io/>, вибираємо потрібні залежності, генеруємо проект, завантажуємо, та відкриваємо в IDE.

The screenshot shows the Spring Initializr web interface. In the 'Dependencies' section, the following options are selected:

- Spring Web** (WEB)
- Spring Data JPA** (SQL)
- H2 Database** (SQL)
- Spring Boot DevTools** (DEVELOPER TOOLS)
- Spring Security** (SECURITY)
- OAuth2 Resource Server** (SECURITY)
- Spring Boot Actuator** (OPS)

At the bottom, there are buttons for **GENERATE** (CTRL + ⌘), **EXPLORE** (CTRL + SPACE), and **SHARE...**.

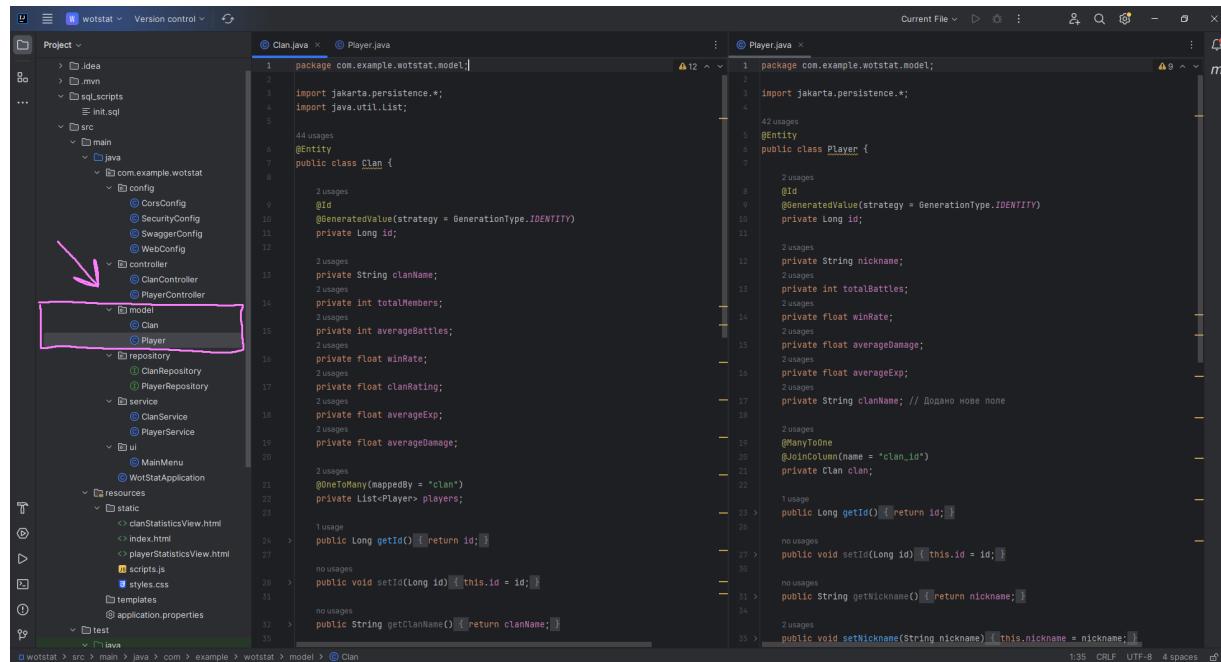
1. Spring Web - для створення веб-додатків.
2. Spring Data JPA - для роботи з базою даних за допомогою JPA.
3. H2 Database - для використання вбудованої бази даних H2.
4. Spring Boot DevTools - для полегшення розробки з автоматичним перезавантаженням додатка.
5. Spring Security - для реалізації безпеки.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

Spring Boot Actuator - для моніторингу та управління додатком.

This screenshot is identical to the one above, showing the Spring Initializr interface with the same set of dependencies selected: Spring Web, Spring Data JPA, H2 Database, Spring Boot DevTools, Spring Security, OAuth2 Resource Server, and Spring Boot Actuator.

2.3 Створення сутностей (Entities)

У цьому проекті сутності 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;

@javax.persistence.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 averageExp;
    private float averageDamage;

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

    public Long getId() { return id; }

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

    public String getClanName() { return clanName; }

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

    public int getTotalMembers() { return totalMembers; }

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

    public int getAverageBattles() { return averageBattles; }

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

    public float getWinRate() { return winRate; }

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

    public float getAverageExp() { return averageExp; }

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

    public float getAverageDamage() { return averageDamage; }

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

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

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

@Entity
public class Player {

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

    private String nickname;
    private int totalBattles;
    private float winRate;
    private float averageExp;
    private String clanName; // Додано нове поле

    @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; }

    public int getTotalBattles() { return totalBattles; }

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

    public float getWinRate() { return winRate; }

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

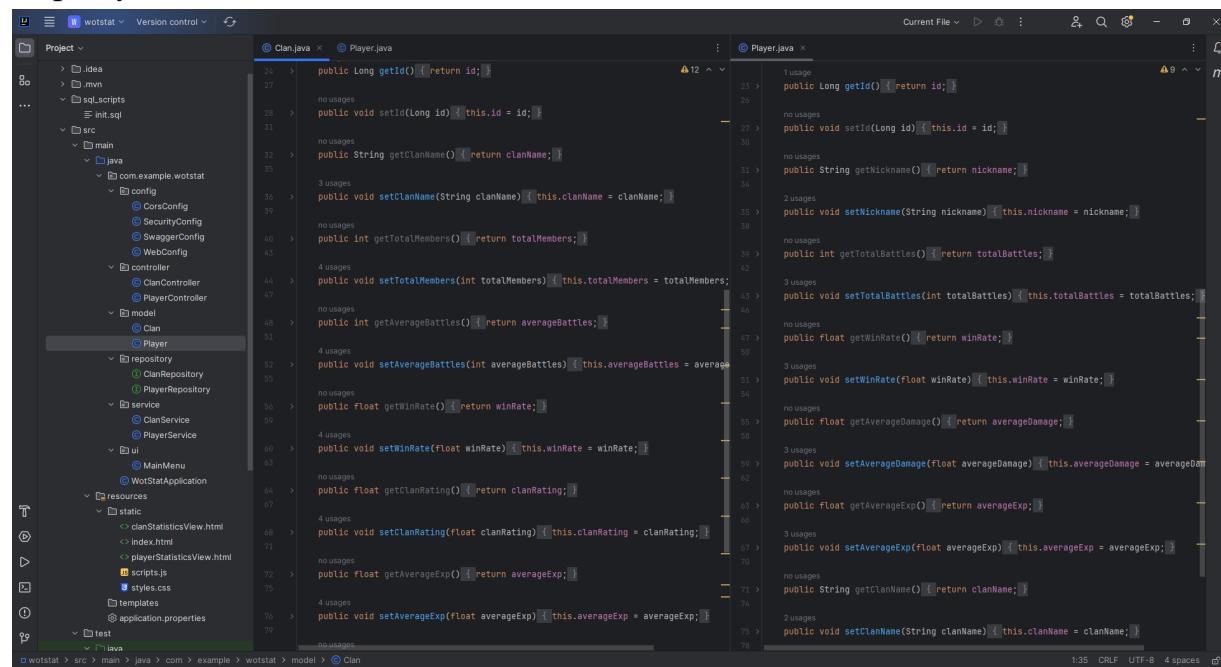
    public float getAverageExp() { return averageExp; }

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

    public String getClanName() { return clanName; }

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

Для кожного поля в сутностях 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 getAverageExp() { return averageExp; }

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

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

no usages
public void setAverageDamage(float averageDamage) { this.averageDamage = averageDamage; }

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; }

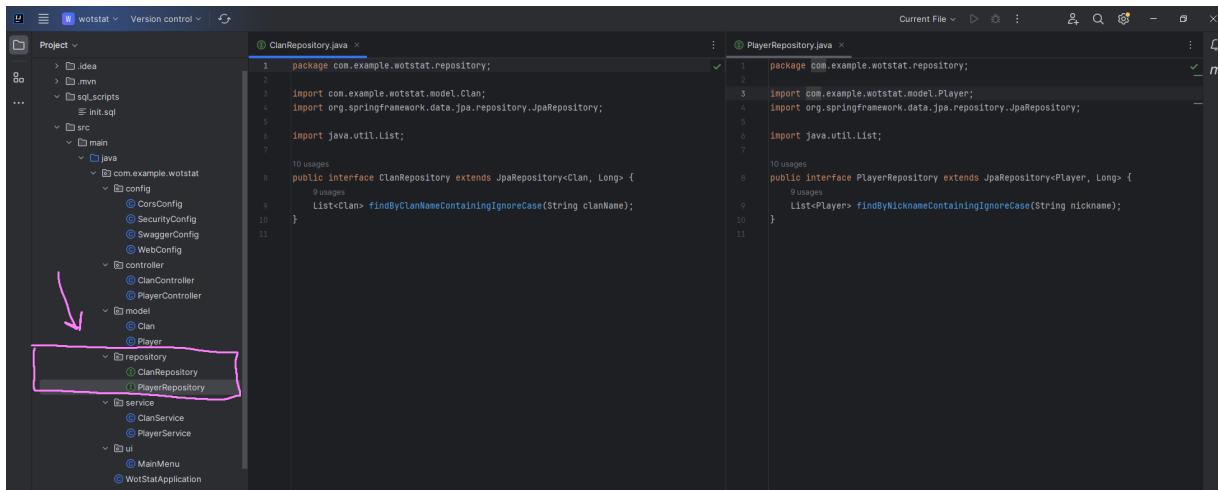
3 usages
public void setAverageExp(float averageExp) { this.averageExp = averageExp; }

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

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

2.4 Створення репозиторіїв (Repositories)

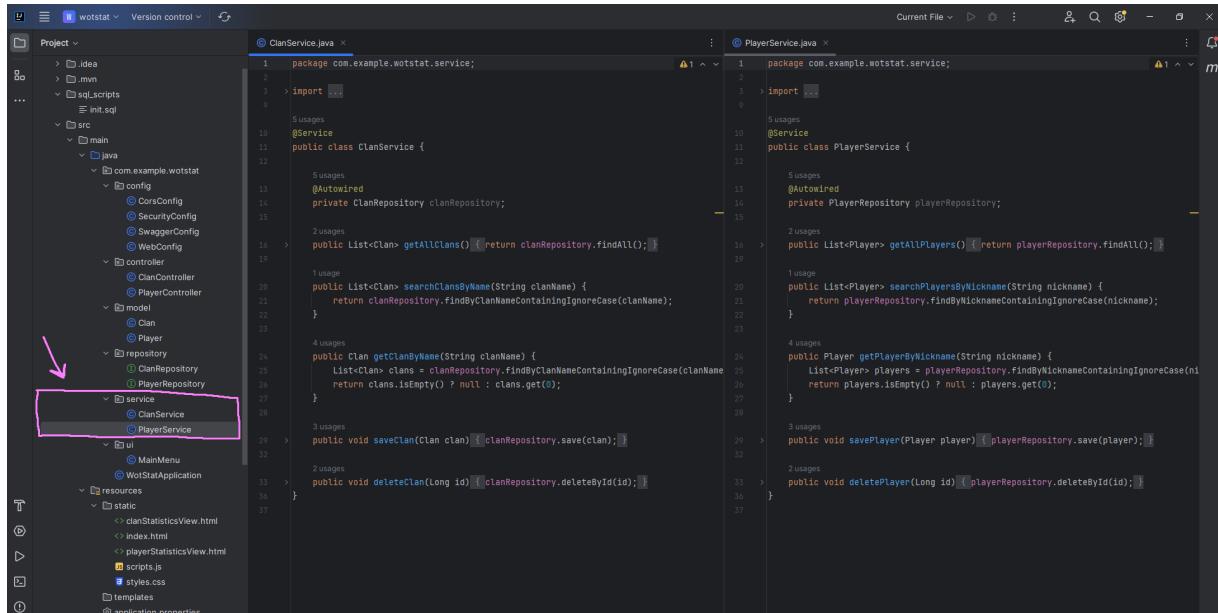
Репозиторії в Spring Data JPA є інтерфейсами, які забезпечують доступ до бази даних та дозволяють виконувати CRUD (Create, Read, Update, Delete) операції без необхідності написання SQL запитів вручну. Spring Data JPA забезпечує автоматичну реалізацію цих інтерфейсів.



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.



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

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.

```
securityConfig.java  ClanController.java  PlayerController.java  PlayerRepository.java  PlayerController.java
```

```
1 > /...
29
30 package com.example.wotstat.controller;
31
32 > import ...
40
41 @RestController
42 @RequestMapping("/api/clans")
43 public class ClanController {
44
45     8 usages
46     @Autowired
47     private ClanRepository clanRepository;
48
49     no usages
50     @CrossOrigin(origins = "http://localhost:8080")
51     @GetMapping
52     public List<Clan> getClans(@RequestParam(required = false) String search) {
53         if (search == null || search.isEmpty()) {
54             return clanRepository.findAll();
55         } else {
56             return clanRepository.findByNameContainingIgnoreCase(search);
57         }
58
59     no usages
60     @CrossOrigin(origins = "http://localhost:8080")
61     @PostMapping
62     public Clan createClan(@RequestBody Clan clan) { return clanRepository.save(clan);
63
64     no usages
65     @CrossOrigin(origins = "http://localhost:8080")
66     @PutMapping("/{id}")
67     public Clan updateClan(@PathVariable Long id, @RequestBody Clan clanDetails) {
68         Optional<Clan> optionalClan = clanRepository.findById(id);
69         if (optionalClan.isPresent()) {
70             Clan clan = optionalClan.get();
71             clan.setClanName(clanDetails.getClanName());
72             clan.setTotalMembers(clanDetails.getTotalMembers());
73             clan.setAverageBattles(clanDetails.getAverageBattles());
74             clan.setWinRate(clanDetails.getWinRate());
75             clan.setClanRating(clanDetails.getClanRating());
76             clan.setAverageExp(clanDetails.getAverageExp());
77             clan.setAverageDamage(clanDetails.getAverageDamage());
78             return clanRepository.save(clan);
79         } else {
80             throw new RuntimeException("Clan not found with id " + id);
81         }
82
83     no usages
84     @CrossOrigin(origins = "http://localhost:8080")
85     @PatchMapping("/{id}")
86     public Clan partiallyUpdateClan(@PathVariable Long id, @RequestBody Map<String,
87         Optional<Clan> optionalClan = clanRepository.findById(id);
88         if (optionalClan.isPresent()) {
89             Clan clan = optionalClan.get();
90             updates.forEach((key, value) -> {
91                 switch (key) {
92                     case "clanName":
93                         clan.setClanName((String) value);
94                         break;
95                     case "totalMembers":
96                         clan.setTotalMembers((Integer) value);
97                         break;
98                     case "averageBattles":
99                         clan.setAverageBattles((Integer) value);
100                    break;
101                   case "winRate":
102                     clan.setWinRate((Float) value);
103                     break;
104                     case "clanRating":
105                     clan.setClanRating((Float) value);
106                     break;
107                     case "averageExp":
108                     clan.setAverageExp((Float) value);
109                     break;
110                     case "averageDamage":
111                     clan.setAverageDamage((Float) value);
112                     break;
113                 }
114             });
115             return clanRepository.save(clan);
116         } else {
117             throw new RuntimeException("Clan not found with id " + id);
118         }
119
120     no usages
121     @CrossOrigin(origins = "http://localhost:8080")
122     @DeleteMapping("/{id}")
123     public void deleteClan(@PathVariable Long id) { clanRepository.deleteById(id);
124 }
```

```
1 > /...
29
30 package com.example.wotstat.controller;
31
32 > import ...
40
41 @RestController
42 @RequestMapping("/api/players")
43 public class PlayerController {
44
45     8 usages
46     @Autowired
47     private PlayerRepository playerRepository;
48
49     no usages
50     @CrossOrigin(origins = "http://localhost:8080")
51     @GetMapping
52     public List<Player> getPlayers(@RequestParam(required = false) String search) {
53         if (search == null || search.isEmpty()) {
54             return playerRepository.findAll();
55         } else {
56             return playerRepository.findByNameContainingIgnoreCase(search);
57         }
58
59     no usages
60     @CrossOrigin(origins = "http://localhost:8080")
61     @PostMapping
62     public Player createPlayer(@RequestBody Player player) { return playerRepository.s
63
64     no usages
65     @CrossOrigin(origins = "http://localhost:8080")
66     @PutMapping("/{id}")
67     public Player updatePlayer(@PathVariable Long id, @RequestBody Player playerDetail
68         Optional<Player> optionalPlayer = playerRepository.findById(id);
69         if (optionalPlayer.isPresent()) {
70             Player player = optionalPlayer.get();
71             player.setNickname(playerDetails.getNickname());
72             player.setTotalBattles(playerDetails.getTotalBattles());
73             player.setWinRate(playerDetails.getWinRate());
74             player.setAverageDamage(playerDetails.getAverageDamage());
75             player.setAverageExp(playerDetails.getAverageExp());
76             player.setClanName(playerDetails.getClanName());
77             return playerRepository.save(player);
78         } else {
79             throw new RuntimeException("Player not found with id " + id);
80         }
81
82     no usages
83     @CrossOrigin(origins = "http://localhost:8080")
84     @PatchMapping("/{id}")
85     public Player partiallyUpdatePlayer(@PathVariable Long id, @RequestBody Map<String,
86         Optional<Player> optionalPlayer = playerRepository.findById(id);
87         if (optionalPlayer.isPresent()) {
88             Player player = optionalPlayer.get();
89             updates.forEach((key, value) -> {
90                 switch (key) {
91                     case "nickname":
92                         player.setNickname((String) value);
93                         break;
94                     case "totalBattles":
95                         player.setTotalBattles((Integer) value);
96                         break;
97                     case "winRate":
98                         player.setWinRate((Float) value);
99                         break;
100                    case "averageDamage":
101                     player.setAverageDamage((Float) value);
102                     break;
103                     case "averageExp":
104                     player.setAverageExp((Float) value);
105                     break;
106                     case "clanName":
107                     player.setClanName((String) value);
108                     break;
109                 }
110             });
111             return playerRepository.save(player);
112         } else {
113             throw new RuntimeException("Player not found with id " + id);
114         }
115
116     no usages
117     @CrossOrigin(origins = "http://localhost:8080")
118     @DeleteMapping("/{id}")
119     public void deletePlayer(@PathVariable Long id) { playerRepository.deleteById(id)
120 }
```

2.7 Написання Unit тестів

UNIT тести використовуються для перевірки окремих частин коду на правильність їхньої роботи. В контексті Spring-додатків, одиницею може бути окремий метод у контролері, сервісі або репозиторії. Мета UNIT тестів — забезпечити, що кожен компонент працює ізольовано від інших і відповідає очікуванням.

ClanControllerTest:

Тест testGetClans:

Перевіряє, що статус відповіді HTTP є 200 OK.

Перевіряє, що JSON відповідь містить хоча б один елемент.

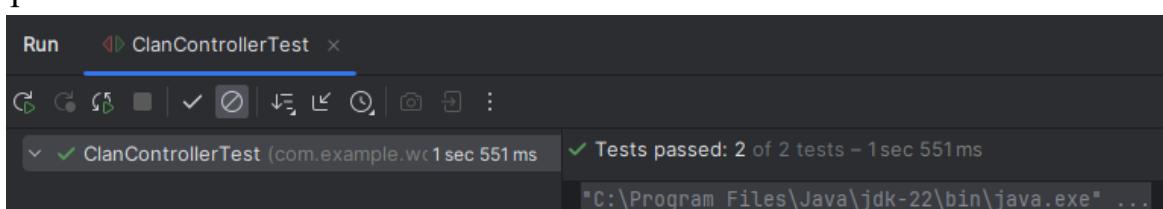
Викликає verify(clanRepository, times(1)).findAll(), щоб переконатися, що метод findAll() був викликаний один раз.

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.



ClanServiceTest:

Тест test GetAllClans:

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.

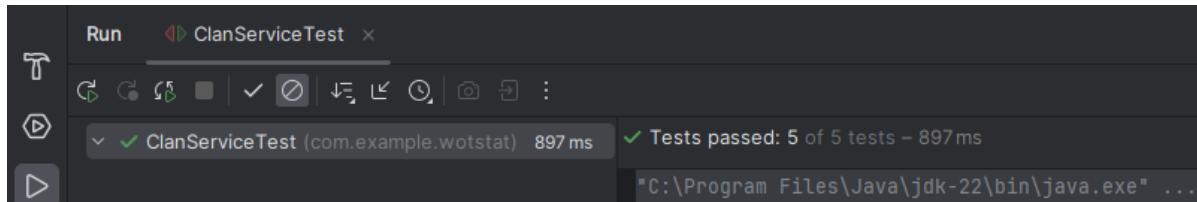
Викликає verify(clanRepository, times(1)).save(clan), щоб переконатися, що метод save() був викликаний один

раз.

Тест testDeleteClan:

Перевіряє, що клан видалився.

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



PlayerControllerTest:

Тест testGetPlayers:

Перевіряє, що статус відповіді HTTP є 200 OK.

Перевіряє, що JSON відповідь містить хоча б один елемент.

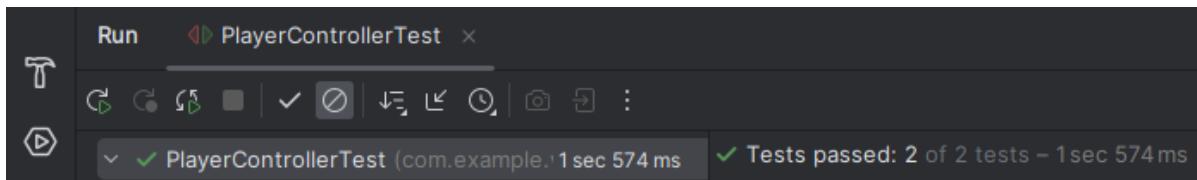
Викликає verify(playerRepository, times(1)).findAll(), щоб переконатися, що метод findAll() був викликаний один раз.

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.



PlayerServiceTest:

Тест test GetAllPlayers:

Перевіряє, що список гравців містить два елементи.

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

Тест testSearchPlayersByNickname:

Перевіряє, що список гравців містить один елемент.

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

Тест testGetPlayerByNickname:

Перевіряє, що знайдений гравець не є null.

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

Тест testSavePlayer:

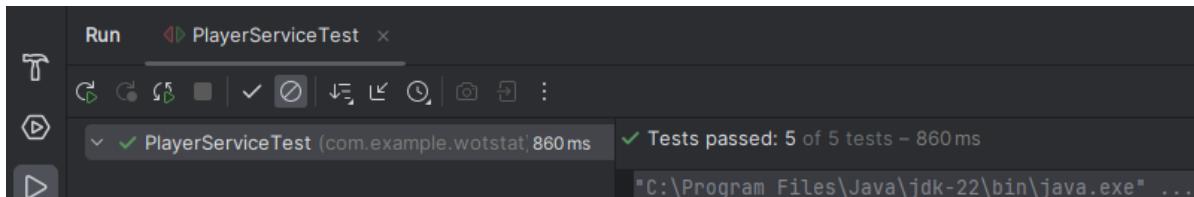
Перевіряє, чи зберігся гравець.

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

Тест testDeletePlayer:

Перевіряє, чи видалився гравець.

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



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

3.1 Підключення системи авторизації OAuth2

Система авторизації OAuth2 забезпечує безпечний доступ до додатку. Це дозволяє користувачам аутентифікуватися за допомогою сторонніх провайдерів (в даному випадку Google), що значно підвищує безпеку і зручність використання.

В цьому проекті, спочатку, був створений застосунок на <https://console.cloud.google.com> під назвою "wotstat"

The screenshot shows the 'Client ID for Web application' configuration page. The 'Name' field is set to 'WotStat Web App'. The 'Authorized JavaScript origins' section contains the URL 'http://localhost:8080/login/oauth2/code/google'. The 'Authorized redirect URIs' section also contains the same URL. The 'Client secrets' section displays the Client ID (1076523933284-cag1iqi5ltj7tqq8k2b864lo4l82p6f.apps.googleusercontent.com) and Client secret (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). The status is marked as 'Enabled'. At the bottom, there are 'SAVE' and 'CANCEL' buttons.

Потім йде налаштування, та отримання Client ID з Client secret для підключення до нашого застосунку в application.properties

OAuth client created

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

The screenshot shows the detailed view of the created OAuth client. It includes the 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). There is a 'DOWNLOAD JSON' button at the bottom left and an 'OK' button at the bottom right.

Налаштовуємо application.properties:

```
spring.security.oauth2.client.registration.google.client-
id=1076523933284-
cagliqi5ltj7tqq8k2b864loia182p6f.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-
secret=GOCSPEX-2pEm0jQEsdhZK2EmYgppzWCTC5fQ
spring.security.oauth2.client.registration.google.scope=openid, profile,
email
spring.security.oauth2.client.provider.google.issuer-uri=https://
accounts.google.com
```

Далі вже йде налаштування конфігурації.

CorsConfig - потрібен для дозволу крос-доменних запитів.

Цей клас дозволяє запити з `http://localhost:8080`, дозволяє всі заголовки та методи HTTP. Це необхідно для забезпечення роботи фронтенду з бекендом.

```
@Configuration
public class CorsConfig {

    no usages
    @Bean
    public CorsFilter corsFilter() {
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowCredentials(true);
        config.addAllowedOrigin("http://localhost:8080"); // надає доступ цьому фронтенду до ресурсів сервера
        config.addAllowedHeader("*"); // дозвіл на всі хедери
        config.addAllowedMethod("*"); // дозвіл на всі методи GET PUT POST OPTIONS DELETE
        source.registerCorsConfiguration(pattern: "/**", config);
        return new CorsFilter(source);
    }
}
```

SecurityConfig - потрібен для налаштування конфігурації безпеки.

Конфігурація безпеки:

Авторизація запитів:

Запити до `/api/**` потребують аутентифікації.

Всі інші запити дозволені без аутентифікації.

OAuth2 Login:

Налаштування аутентифікації через OAuth2 з використанням Google.

Використання OidcUserService для отримання інформації про користувача.

Додавання кастомного фільтра:

Додавання кастомного фільтра для перевірки email користувача при доступі до API.

Якщо email не відповідає заданому, доступ дозволяється лише для GET запитів.

В результаті маємо, що запити на отримання статистики (GET) можуть виконувати всі авторизовані користувачі, а всі запити може виконувати тільки авторизований користувач `toadkillergamer@gmail.com`.

3.2 Підключення Swagger UI з детальним описом REST API

Swagger — це інструмент для документування REST API, який дозволяє легко переглядати та тестувати API. Для інтеграції Swagger у Spring-додаток використана бібліотека Springdoc OpenAPI.

```
@Configuration
public class SwaggerConfig {

    no usages

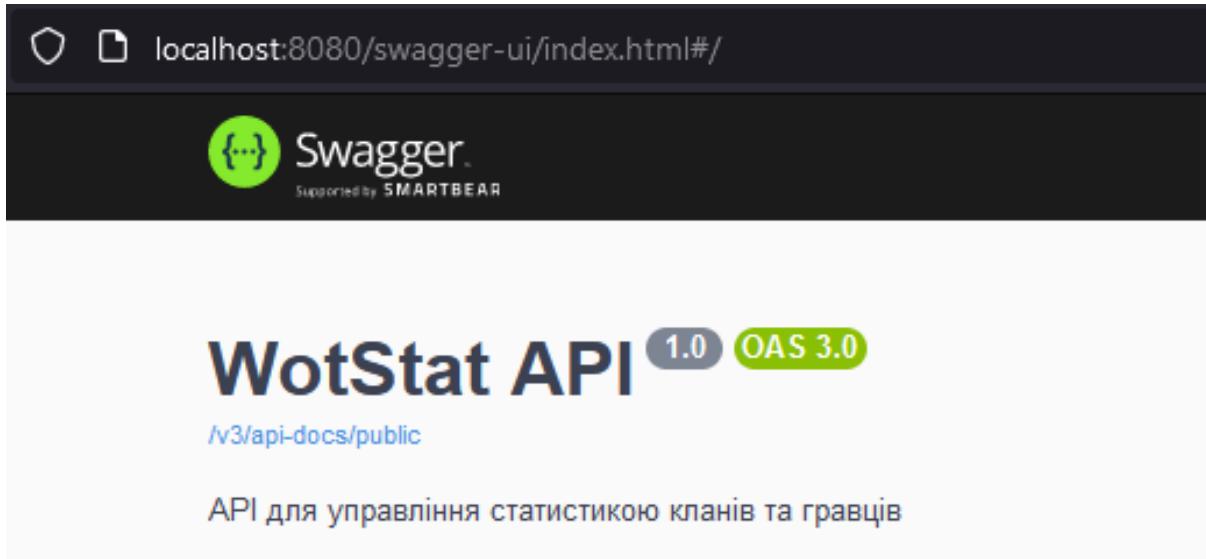
    @Bean
    public OpenAPI openAPI() {
        return new OpenAPI()
            .info(new Info()
                .title("WotStat API")
                .version("1.0")
                .description("API для управління статистикою кланів та гравців"));
    }

    no usages

    @Bean
    public GroupedOpenApi publicApi() {
        return GroupedOpenApi.builder()
            .group("public")
            .pathsToMatch("/api/**")
            .build();
    }
}
```

OpenAPI - це центральний об'єкт, який містить всю інформацію про API.
GroupedOpenApi - це об'єкт, який дозволяє групувати і фільтрувати ендпоїнти для документування.

В результаті, маємо наступне:



The screenshot shows the Swagger UI homepage for the WotStat API. At the top, there's a navigation bar with icons for shield, file, and URL (localhost:8080/swagger-ui/index.html#/). Below the bar, the Swagger logo (a green circle with three white curly braces) and the text "Supported by SMARTBEAR" are visible. The main title is "WotStat API" in large blue text, with "1.0" and "OAS 3.0" in smaller circles next to it. Below the title, there's a blue link to "/v3/api-docs/public". At the bottom, a description reads "API для управління статистикою кланів та гравців".

3.3 Збірка проекту в контейнери Docker

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

Перш за все - налаштовуємо application.properties, для докер-контейнера він окремий.

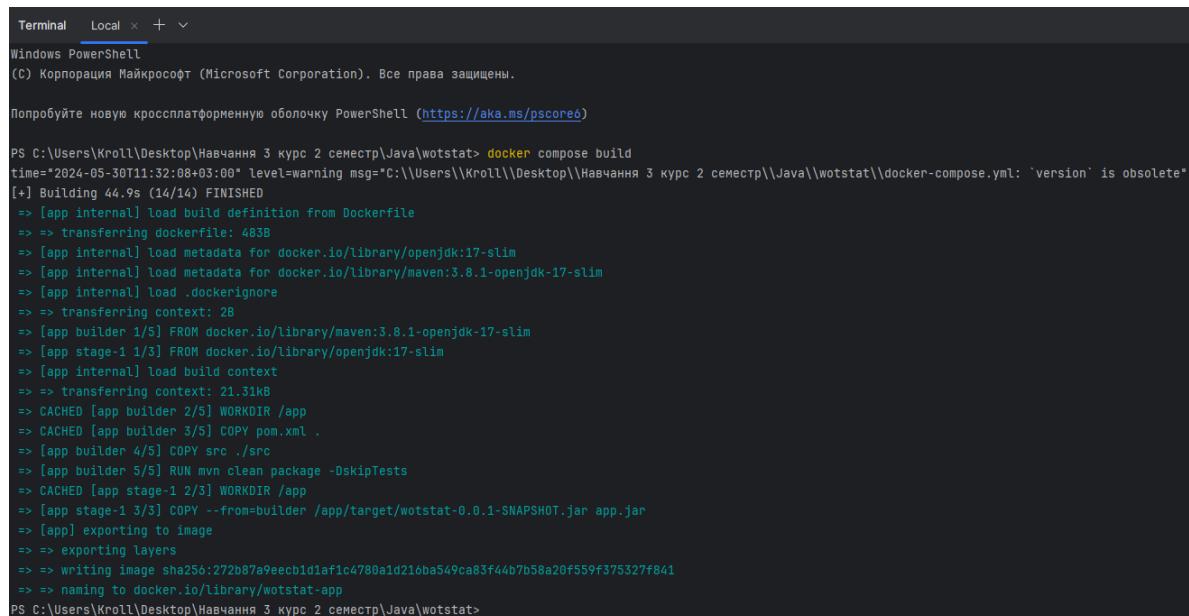
```
spring.application.name=WotStat
spring.datasource.url=jdbc:mysql://localhost:3307/wot_stat_db
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.security.oauth2.client.registration.google.client-id=1076523933284-cag1iqi5ltj7tqq8k2b864loia182pof.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-secret=GOCSPX-2pEm0jQEbhZK2EmYgppzWCTC5fQ
spring.security.oauth2.client.registration.google.scope=openid, profile, email
spring.security.oauth2.client.provider.google.issuer-uri=https://accounts.google.com
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.html
server.port=8080
springdoc.swagger-ui.path=/custom-path
```

Тут ми використовуємо інший порт для БД.

Налаштування docker-compose.yml та DockerFile.

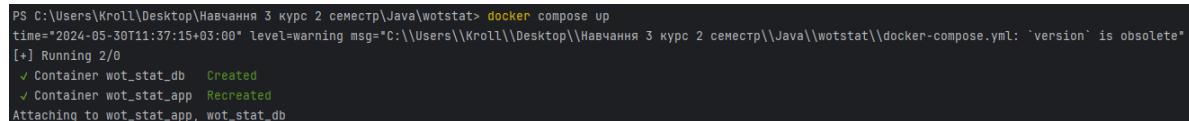
Їх налаштування дозволяє розгорнати та запускати додаток в ізольованих контейнерах.

Білдимо контейнер:



```
PS C:\Users\Kroll\Desktop\Навчання 3 курс 2 семестр\Java\wotstat> docker compose build
time="2024-05-30T11:32:08+03:00" level=warning msg="C:\\Users\\Kroll\\Desktop\\Навчання 3 курс 2 семестр\\Java\\wotstat\\docker-compose.yml: 'version' is obsolete"
[+] Building 44.9s (14/14) FINISHED
=> [app internal] load build definition from Dockerfile
=> => transferring dockerfile: 483B
=> [app internal] load metadata for docker.io/library/openjdk:17-slim
=> [app internal] load metadata for docker.io/library/maven:3.8.1-openjdk-17-slim
=> [app internal] load .dockerrcignore
=> => transferring context: 2B
=> [app builder 1/5] FROM docker.io/library/maven:3.8.1-openjdk-17-slim
=> [app stage-1 1/3] FROM docker.io/library/openjdk:17-slim
=> [app internal] load build context
=> => transferring context: 21.31KB
=> CACHED [app builder 2/5] WORKDIR /app
=> CACHED [app builder 3/5] COPY pom.xml .
=> [app builder 4/5] COPY src ./src
=> [app builder 5/5] RUN mvn clean package -DskipTests
=> CACHED [app stage-1 2/3] WORKDIR /app
=> [app stage-1 3/3] COPY --from=builder /app/target/wotstat-0.0.1-SNAPSHOT.jar app.jar
=> [app] exporting to image
=> => exporting layers
=> => writing image sha256:272b87a9eecd1af1c4780a1d216ba549ca83f44b7b58a20f559f375327f841
=> => naming to docker.io/library/wotstat-app
PS C:\Users\Kroll\Desktop\Навчання 3 курс 2 семестр\Java\wotstat>
```

Запускаємо контейнер:



```
PS C:\Users\Kroll\Desktop\Навчання 3 курс 2 семестр\Java\wotstat> docker compose up
time="2024-05-30T11:37:15+03:00" level=warning msg="C:\\Users\\Kroll\\Desktop\\Навчання 3 курс 2 семестр\\Java\\wotstat\\docker-compose.yml: 'version' is obsolete"
[+] Running 2/0
  ✓ Container wot_stat_db   Created
  ✓ Container wot_stat_app  Recreated
Attaching to wot_stat_app, wot_stat_db
```

В результаті маємо запущений контейнер, який виконує функції застосунку:

Container CPU usage: 0.00% / 1200% (12 CPUs available)

Container memory usage: 0B / 15.2GB

Show charts

Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
wotstat	mysql:8.0	Running (2/2)	3307:3306	0%	1 minute ago	[Stop] [Logs] [Details]
wot_stat_db	99110c26d1ef	Running	8080:8080	0%	1 minute ago	[Stop] [Logs] [Details]
wotstat-app	99110c26d1ef	Running	8080:8080	0%	1 minute ago	[Stop] [Logs] [Details]

4. Інтеграційне тестування системи

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

Інтеграційне тестування використовується для тестування всієї системи разом.

Project: wotstat

IntegrationTests.java

```
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@SpringBootTest
@AutoConfigureMockMvc
public class IntegrationTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testIndex() throws Exception {
        mockMvc.perform(get("/"))
                .andExpect(status().isOk())
                .andExpect(content().string("Welcome to WotStat!"));
    }
}
```

Run: IntegrationTests

Output:

```
2024-05-30T12:00:00+03:00 INFO 18100 --- [wotstat] [main] o.h.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-05-30T12:00:03:748+03:00 INFO 18100 --- [wotstat] [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 6.2.Final
2024-05-30T12:00:03:781+03:00 INFO 18100 --- [wotstat] [main] o.h.intern.RegionFactoryInitiator : HHH000026: Second-level cache disabled
2024-05-30T12:00:04:106+03:00 INFO 18100 --- [wotstat] [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup; ignoring JPA class transformer
2024-05-30T12:00:04:132+03:00 INFO 18100 --- [wotstat] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2024-05-30T12:00:04:455+03:00 INFO 18100 --- [wotstat] [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 Added connection com.mysql.cj.jdbc.Connection
2024-05-30T12:00:04:456+03:00 INFO 18100 --- [wotstat] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2024-05-30T12:00:04:499+03:00 WARN 18100 --- [wotstat] [main] org.hibernate.orm.deprecation : HHHW000025: MySQLDialect does not need to be specified explicitly. MySQLDialect has been deprecated; use org.hibernate.dialect.MySQLDialect instead.
2024-05-30T12:00:04:497+03:00 WARN 18100 --- [wotstat] [main] org.hibernate.orm.deprecation : HHHW000026: MySQLDialect has been deprecated; use org.hibernate.dialect.MySQLDialect instead.
2024-05-30T12:00:05:129+03:00 INFO 18100 --- [wotstat] [main] o.h.e.t.t.j.p.AopPlatformInitiator : HHHW000049: No JTA platform available (set 'hibernate.transaction.platform_name' to 'true' to enable)
2024-05-30T12:00:05:485+03:00 WARN 18100 --- [wotstat] [main] j.LocalContainerEntityManagerFactoryBean : Initializing JPA EntityManagerFactory for persistence unit 'default'. PersistenceUnitProperties: properties enabled by default. Therefore, don't expose 'hibernate.jpa.platform'.
2024-05-30T12:00:05:485+03:00 INFO 18100 --- [wotstat] [main] JpaBaseConfiguration$JpaWebConfiguration : Spring Data JPA configuration is disabled by default. Therefore, do not expose 'spring.data.jpa.repositories.enabled' property.
2024-05-30T12:00:06:169+03:00 INFO 18100 --- [wotstat] [main] o.s.w.a.WebEndpointLinksResolver : Exposing 1 endpoint beneath base path '/actuator'
2024-05-30T12:00:06:274+03:00 INFO 18100 --- [wotstat] [main] o.s.w.DefaultSecurityFilterChain : Will secure any request with [org.springframework.security.web.DefaultSecurityFilterChain]
2024-05-30T12:00:06:853+03:00 INFO 18100 --- [wotstat] [main] o.s.w.DispatcherServlet : Initializing Servlet ''
2024-05-30T12:00:06:853+03:00 INFO 18100 --- [wotstat] [main] o.s.w.DispatcherServlet : Completed initialization in 1 ms
2024-05-30T12:00:06:916+03:00 INFO 18100 --- [wotstat] [main] com.example.wotstat.IntegrationTests : Started IntegrationTests in 5.679 seconds (process running for 1. Manage Clans
2. Manage Players
3. Exit
```

Тест був написаний за User Story, які були описані в першому пункті.

5. Використана література

<https://uk.wikipedia.org/wiki/Java>

<https://qagroup.com.ua/publications/what-is-swagger/>

<https://foxminded.ua/shcho-take-rest-api/>

<https://console.cloud.google.com>

<https://stackoverflow.com/questions/74055657/bean-of-type-org-springframework-security-oauth2-client-userinfo-oauth2userservi>

<https://freehost.com.ua/ukr/faq/articles/swagger-scho-tse-take-ta-jak-z-nim-pratsjuvati/>

<https://stackoverflow.com/questions/4119448/the-import-javax-servlet-cant-be-resolved>

<https://stackoverflow.com/questions/24319662/from-inside-of-a-docker-container-how-do-i-connect-to-the-localhost-of-the-mach>