**КУРСОВА РОБОТА**

з дисципліни

**«Конструювання программного забезпечення Java»**

на тему

**«Веб додаток керування будівельними процесами»**

Опис:

**«платформа, що надає інструменти для планування, відстеження прогресу та управління ресурсами в будівельних проектах. »**

**Виконав**:

Студент групи ПД-34

Башич Владислав Андрійович

**Керівник курсової роботи:**

Викладач: Довженко Тимур Павлович

Київ – 2024

# **1 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ**

1.1 Опис User stories з діаграмами послідовностей

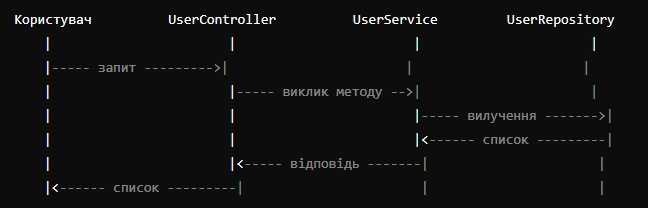
Проектування системи почнемо з опису User Stories. User story - це неформальне, загальне пояснення функції програмного забезпечення, написане з точки зору кінцевого користувача. Її мета - пояснити, як функція програмного забезпечення буде корисною для користувача. User Stories є одним з основних компонентів методу гнучкої розробки програмного забезпечення (Agile software development). Вони допомагають створити орієнтовану на користувача основу для щоденної роботи, що сприяє співпраці, творчості та покращенню якості продукту в цілому.

**Перегляд списку користувачів**

Опис:

Як користувач, я хочу переглядати список усіх користувачів у системі.

Діаграма послідовності:



Малюнок 1.1

Пояснення:

Користувач надсилає запит на отримання списку користувачів.

Контролер UserController отримує запит.

UserController викликає метод getAll() сервісу UserService.

Сервіс UserService повертає список користувачів із репозиторію.

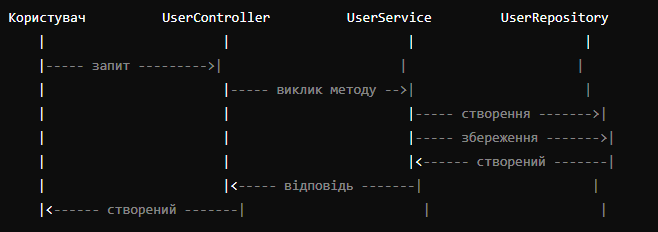
UserController відправляє список користувачів як відповідь користувачеві.

**Створення нового користувача**

Опис:

Як користувач, я хочу створювати нових користувачів у системі.

Діаграма послідовності:



Малюнок 1.2

Пояснення:

Користувач надсилає запит на створення нового користувача, надаючи дані про нового користувача.

Контролер UserController отримує запит із даними нового користувача.

UserController викликає метод createUser() сервісу UserService, передаючи дані нового користувача.

Сервіс UserService створює нового користувача та зберігає його в репозиторії.

Сервіс UserService повертає створеного користувача.

UserController відправляє створеного користувача як відповідь користувачеві.

1.2 Опис ERD бази даних

Далі розробимо ER-модель – це модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. Модель сутність-зв'язок є результатом систематичного процесу, який описує та визначає деяку предметну область. Вона не визначає сам процес, а лише візуалізує його. Дані представлені у вигляді компонентів (сутностей), які пов'язані між собою певними зв'язками, які виражають залежності і вимоги між ними, такі як: одна будівля може бути розділена на декілька квартир, але одна квартира може бути розташована лише в одній будівлі. Сутності можуть мати різні властивості (атрибути), які характеризують їх. Діаграми, створені для представлення цих сутностей, атрибутів і зв'язків графічно, називають сутність-зв'язок діаграмами.

Почнемо з опису сутностей системи.

**Сутність 1 "Проект"**:

* **id**: унікальний ідентифікатор проекту.
* **project\_name**: назва проекту.
* **description**: опис проекту.
* **start**: дата початку проекту.
* **end**: дата завершення проекту.
* **status**: статус проекту.

**Сутність 2 "Ресурси"**:

* **id**: унікальний ідентифікатор ресурсу.
* **resources\_name**: назва ресурсу.
* **quantity**: кількість ресурсу.
* **cost**: вартість ресурсу.
* **project\_id**: зовнішній ключ, що посилається на ідентифікатор проекту, до якого належить ресурс.

**Зв'язок між "Проект" та "Ресурси"**:

* Кожен проект може мати багато ресурсів, тому в таблиці "Ресурси" є зовнішній ключ project\_id, який посилається на id проекту в таблиці "Проект". Це означає, що кожні ресурси пов'язані з конкретним проектом. Коли проект видаляється, всі пов'язані з ним ресурси також автоматично видаляються (ON DELETE CASCADE), щоб уникнути появи посилань на неіснуючі дані.

Зобразимо всі вище перераховані сутності за допомогою діаграми:

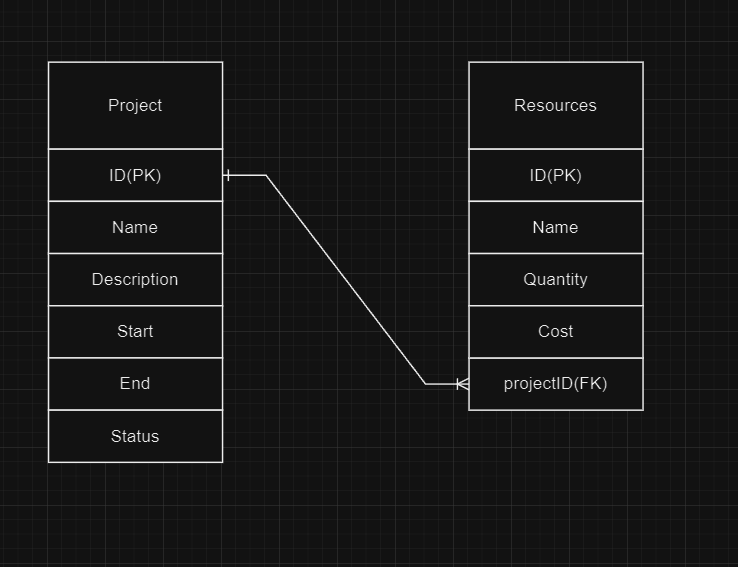


Рисунок 1.5 – ER-діаграма

1.3 Опис діаграми класів

Діаграма класів – це діаграма, на якій показані класи, інтерфейси та відносини між ними. Головний елемент діаграми класів – клас. При проектуванні об'єктно-орієнтованих систем діаграми класів обов'язкові. Класи використовуються в процесі аналізу предметної області для складання словника предметної області системи, що розробляється. Це можуть бути як абстрактні поняття предметної області, так і класи, на які спирається розробка та які описують програмні або апаратні сутності. Діаграма класів є набором статичних, декларативних елементів моделі. Класи зображуються у вигляді прямокутників, зазвичай розділених на дві або три частини. У верхній частині знаходиться ім'я класу. Середня частина містить список змінних класу, а нижня частина – методи класу. Символи, зазначені перед кожною змінною або методом, представляють собою індикатори видимості.

Кожен контролер відповідає за обробку HTTP-запитів і передачу даних до відповідного сервісу для обробки. Сервіси, у свою чергу, виконують основну бізнес-логіку і взаємодіють зі сховищем даних або іншими системами.

Діаграма відображає модульну структуру системи, де контролери відповідають за обробку запитів, а сервіси — за виконання бізнес-логіки. Така архітектура допомагає розподілити обов'язки і спрощує тестування та підтримку коду.

Діаграма класів зображена на рисунку 1. 6.

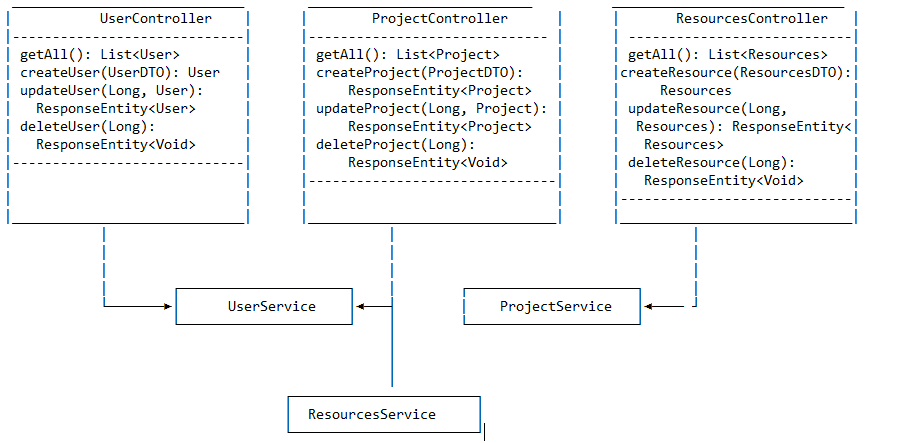


Рисунок 1.6 – Діаграма класів

1.4 Написання тестових сценаріїв

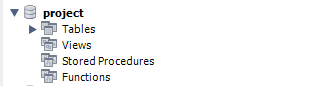
1. **UserController:**
   * **Тест на отримання всіх користувачів:**
     + Очікування: Отримати статус відповіді 200 OK та список користувачів у форматі JSON.
   * **Тест на створення нового користувача:**
     + Очікування: Отримати статус відповіді 200 OK та об'єкт нового користувача у форматі JSON.
   * **Тест на оновлення інформації про користувача:**
     + Очікування: Отримати статус відповіді 200 OK та оновлений об'єкт користувача у форматі JSON.
   * **Тест на видалення користувача:**
     + Очікування: Отримати статус відповіді 204 No Content.
2. **ProjectController:**
   * **Тест на отримання всіх проектів:**
     + Очікування: Отримати статус відповіді 200 OK та список проектів у форматі JSON.
   * **Тест на створення нового проекту:**
     + Очікування: Отримати статус відповіді 200 OK та об'єкт нового проекту у форматі JSON.
   * **Тест на оновлення інформації про проект:**
     + Очікування: Отримати статус відповіді 200 OK та оновлений об'єкт проекту у форматі JSON.
   * **Тест на видалення проекту:**
     + Очікування: Отримати статус відповіді 204 No Content.
3. **ResourcesController:**
   * **Тест на отримання всіх ресурсів:**
     + Очікування: Отримати статус відповіді 200 OK та список ресурсів у форматі JSON.
   * **Тест на створення нового ресурсу:**
     + Очікування: Отримати статус відповіді 200 OK та об'єкт нового ресурсу у форматі JSON.
   * **Тест на оновлення інформації про ресурс:**
     + Очікування: Отримати статус відповіді 200 OK та оновлений об'єкт ресурсу у форматі JSON.
   * **Тест на видалення ресурсу:**
     + Очікування: Отримати статус відповіді 204 No Content.

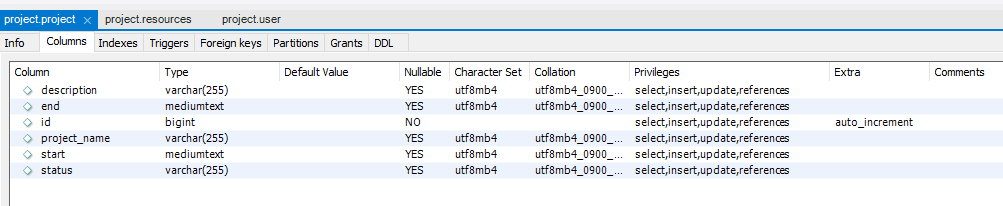
# **2 РЕАЛІЗАЦІЯ API**

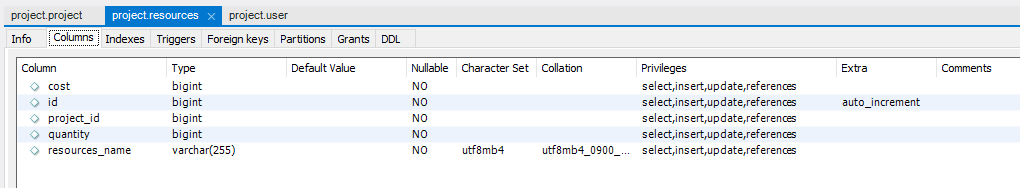
**2.1 Підготовка бази даних**

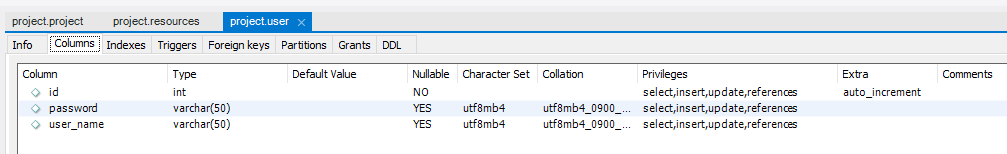
Оскільки схема бази даних спроектована у попередньому розділі, перейдемо до вибору СКБД. У проєкті будемо використовувати MySQL - це вільна та відкрита реляційна система керування базами даних, яка цінується за свою надійність, гнучкість та масштабованість. Вона використовується широким колом організацій, від стартапів до великих підприємств, для зберігання та аналізу даних.

Перейдемо до створення бази даних.



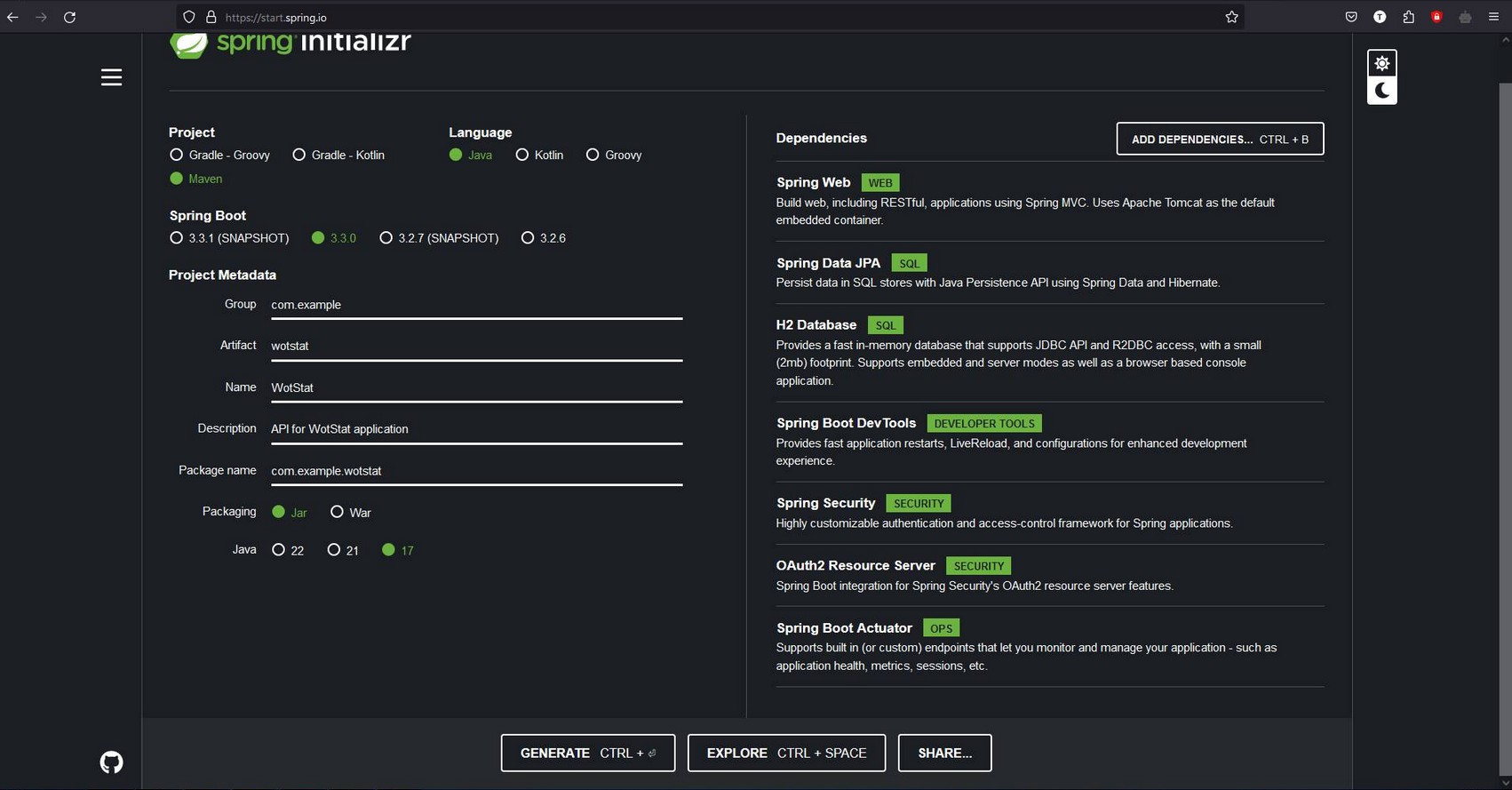






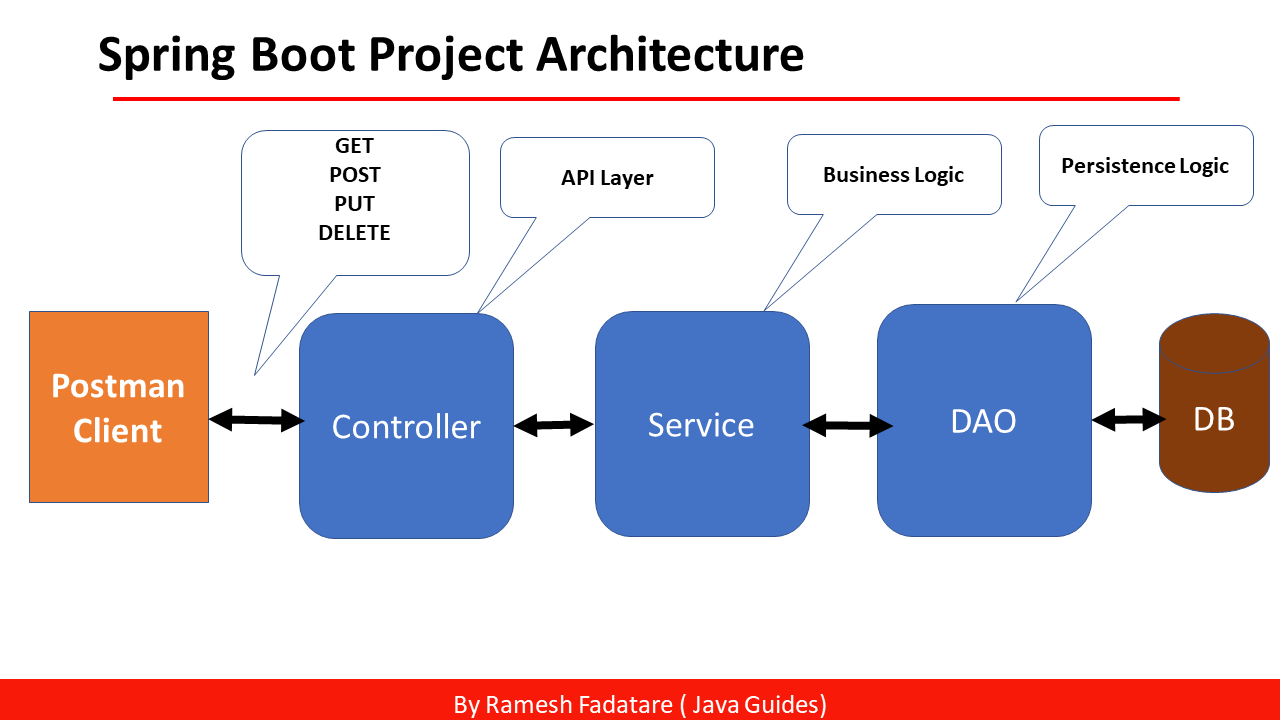
Малюнок 2.1 – Створення таблиць БД

**2.2 Створення Spring Boot Application**

Для створення Spring Boot Application використаємо інструмент spring initializr, який генерує стартовий код проекту з вибраними нами залежностями.

Малюнок 2.2 – Генерація стартового коду

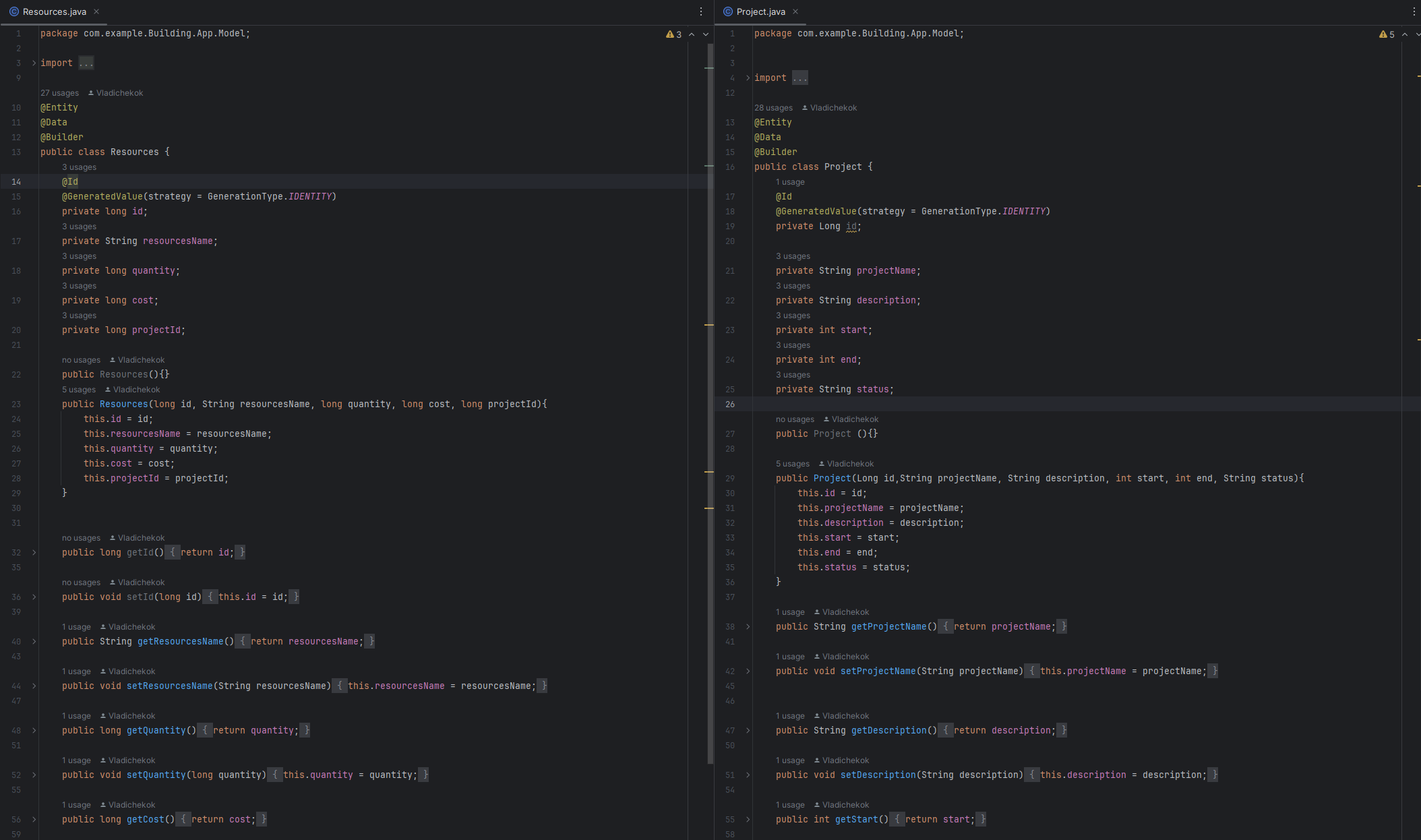
Для розробки будемо використовувати InteliJ Idea. Архітектуру проекту побудуємо за (Model-View-Controller), який часто використовується веб-додатках.



Малюнок 2.3 – Архітектура проекту

Кожен компонент взаємодіє з іншими компонентами за допомогою відповідних інтерфейсів або класів. Наприклад, контролер може викликати методи сервісів, а сервіси в свою чергу використовують репозиторії для доступу до даних. DTO використовуються для передачі даних між цими компонентами, щоб розділити модель даних (Entity) та внутрішню логіку додатку від зовнішнього інтерфейсу (Controller).

Створимо Entity для кожного об’єкту даних, що зберігаються в базі даних.



Малюнок 2.4 - Моделі

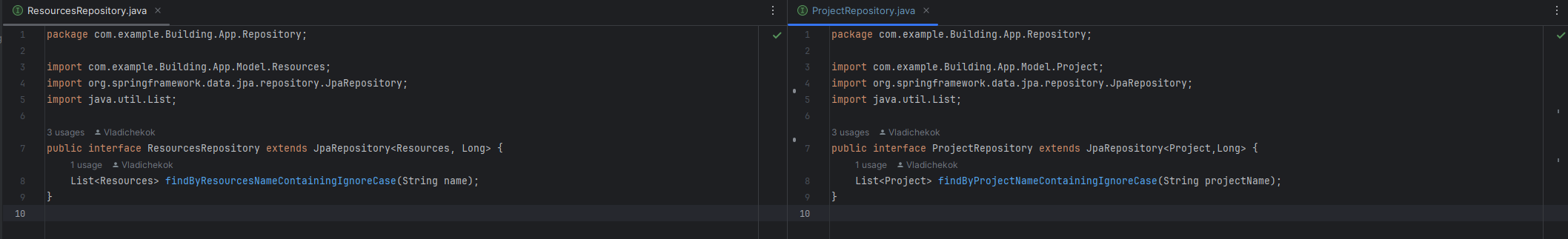
До кожної з моделі створимо DTO (Об'єкти переносу даних) - об'єкти, які використовуються для передачі даних між різними частинами додатка, зазвичай між контролером та сервісом.

Також створимо репозиторії - це компоненти, які взаємодіють з базою даних. Вони надають методи для отримання, збереження, оновлення та видалення об'єктів.

Створимо сервіси - компоненти, які виконують бізнес-логіку застосунка. Вони використовують репозиторії для доступу до даних та виконання операцій з ними.

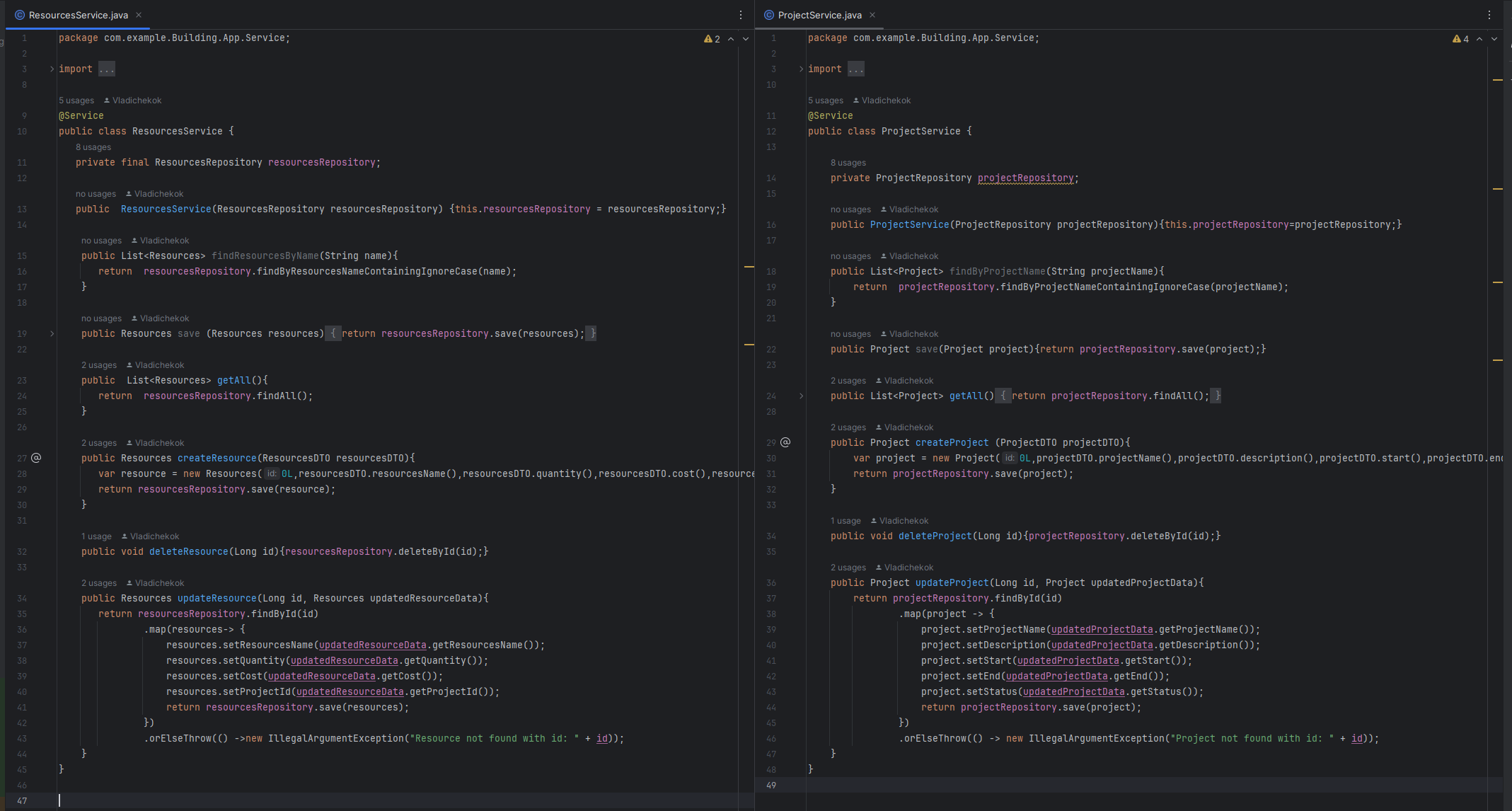
І в кінці створимо контролери - компоненти, які приймають HTTP-запити від клієнта, обробляють їх та повертають відповідь. Вони викликають методи сервісів для виконання потрібних операцій.

Перевіримо роботу:



Малюнок 2.5 – Репозиторії

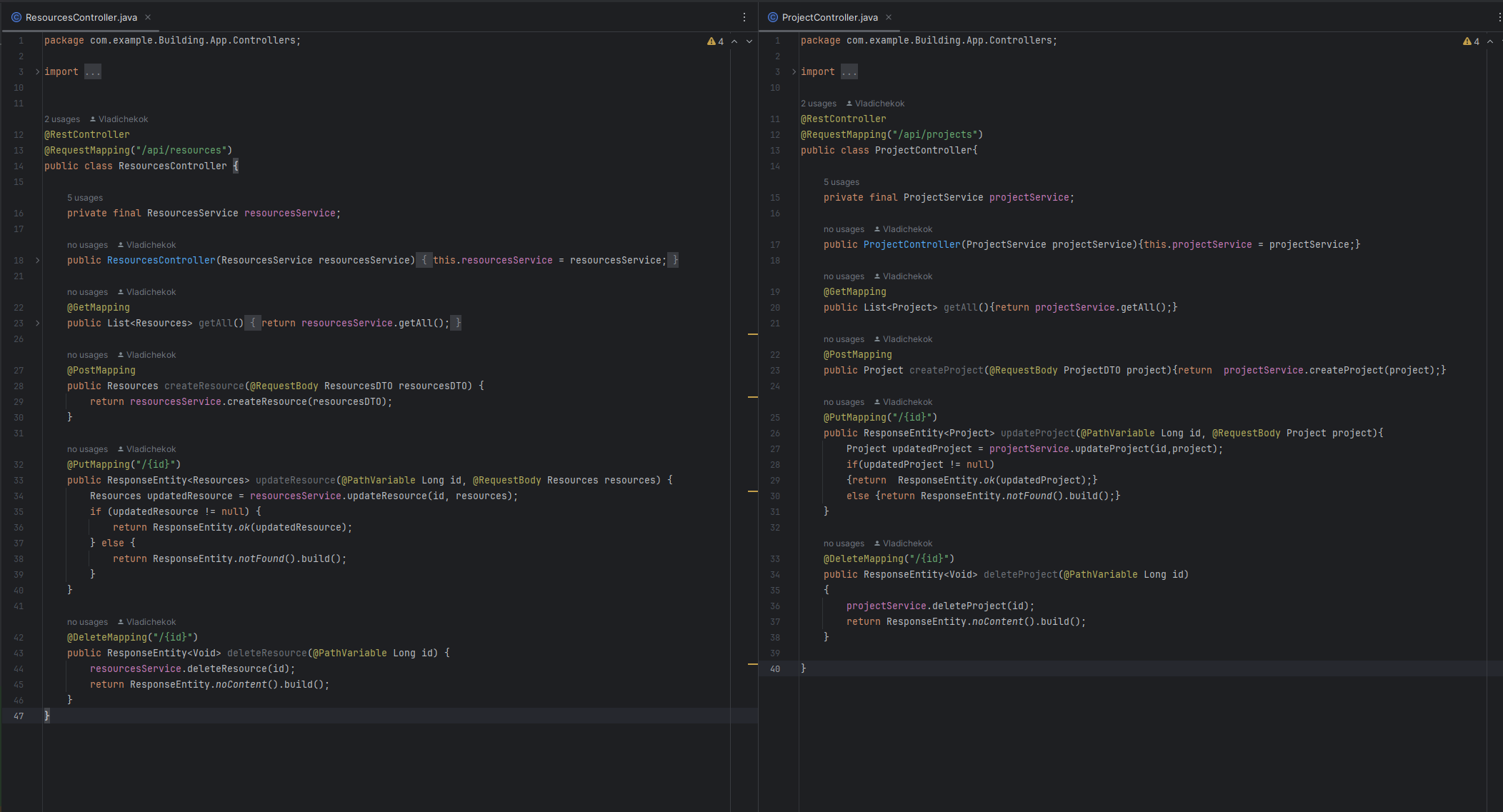
Репозиторії в Spring Data JPA є інтерфейсами, які забезпечують доступ до бази даних та дозволяють виконувати CRUD (Create, Read, Update, Delete) операції без необхідності написання SQL запитів вручну. Spring Data JPA забезпечує автоматичну реалізацію цих інтерфейсів.



Малюнок 2.6 – Сервіси

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.



Малюнок 2.7 – Контроллери

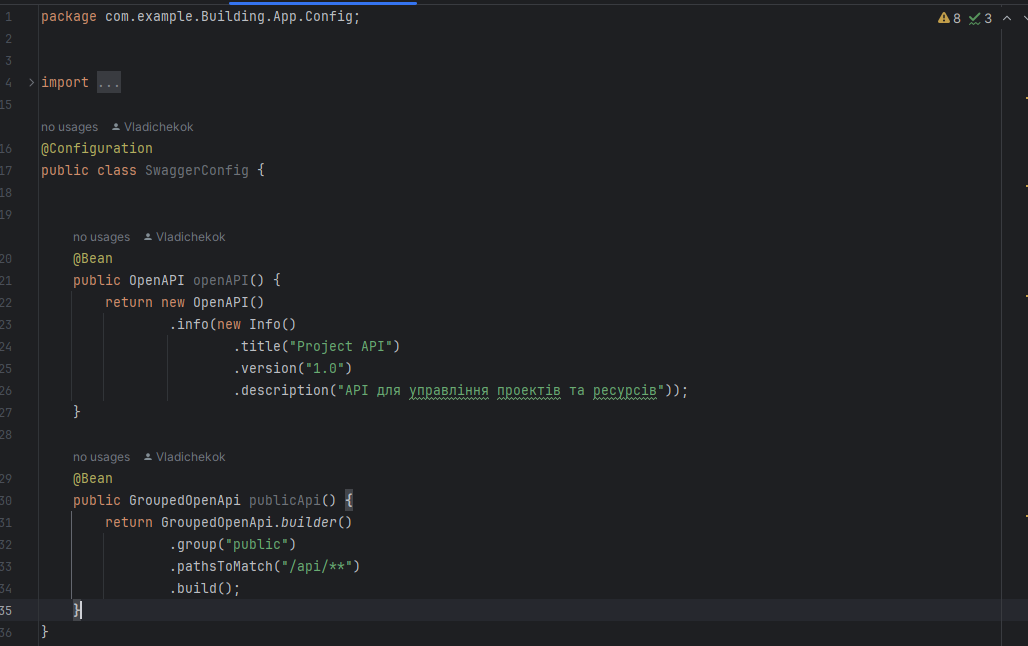
Контролери (Controllers) у Spring Framework відповідають за обробку HTTP-запитів і повернення відповідей клієнту. Вони є частиною архітектури MVC (Model-View-Controller) і взаємодіють із сервісами для виконання бізнес-логіки.

# **3 ПІДКЛЮЧЕННЯ SWAGGER**

**3.1 Підключення Swagger UI з детальним описом REST API**

Swagger - це набір інструментів для створення, документування та використання API. Основна мета Swagger - полегшити розробку та спілкування між розробниками та іншими учасниками проекту щодо використання API.

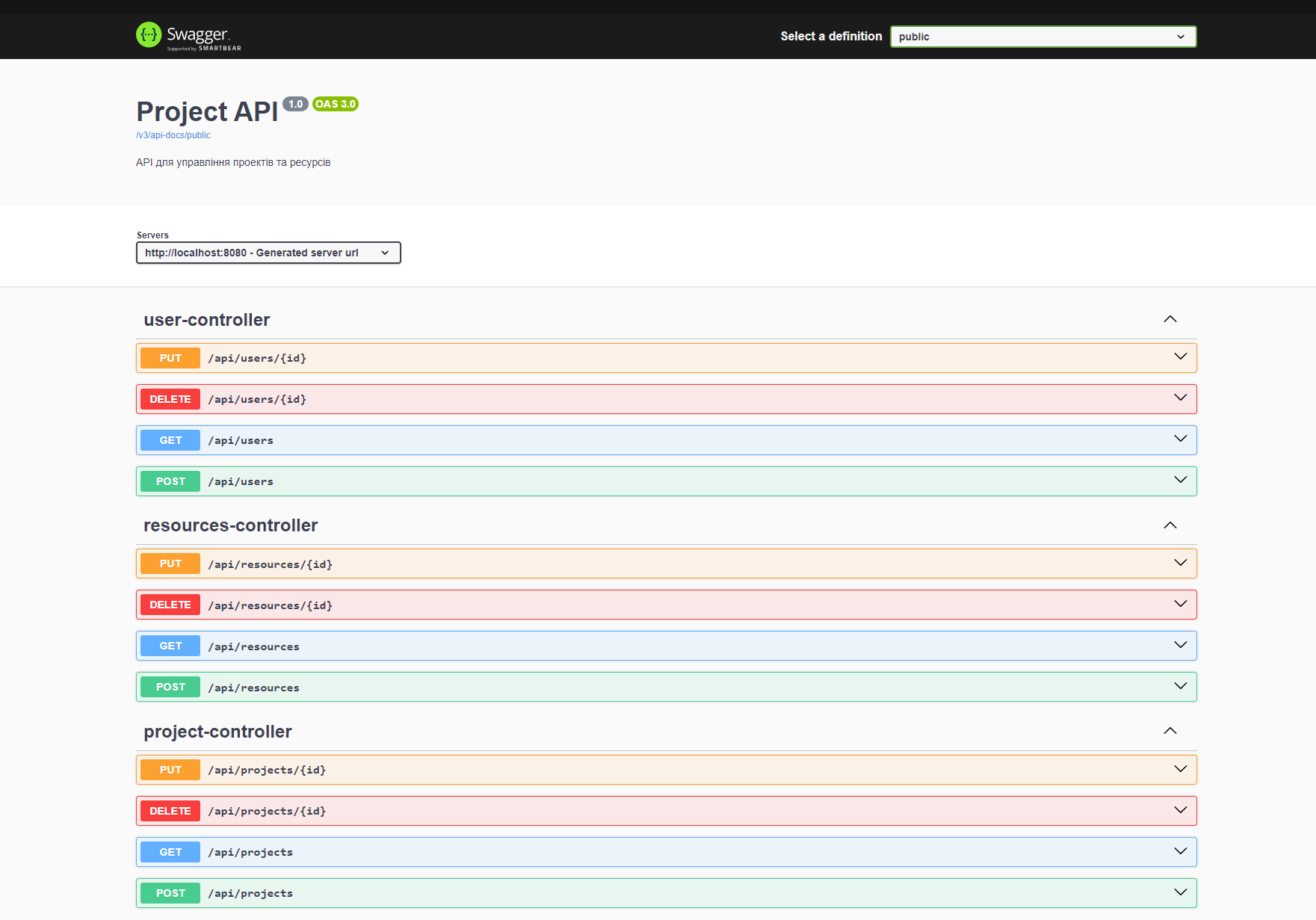
Swagger UI – це інтерактивний веб-інтерфейс, що автоматично генерує документацію API на основі специфікації OpenAPI. З Swagger UI можна переглядати, тестувати та спілкуватися з вашим API безпосередньо в браузері.



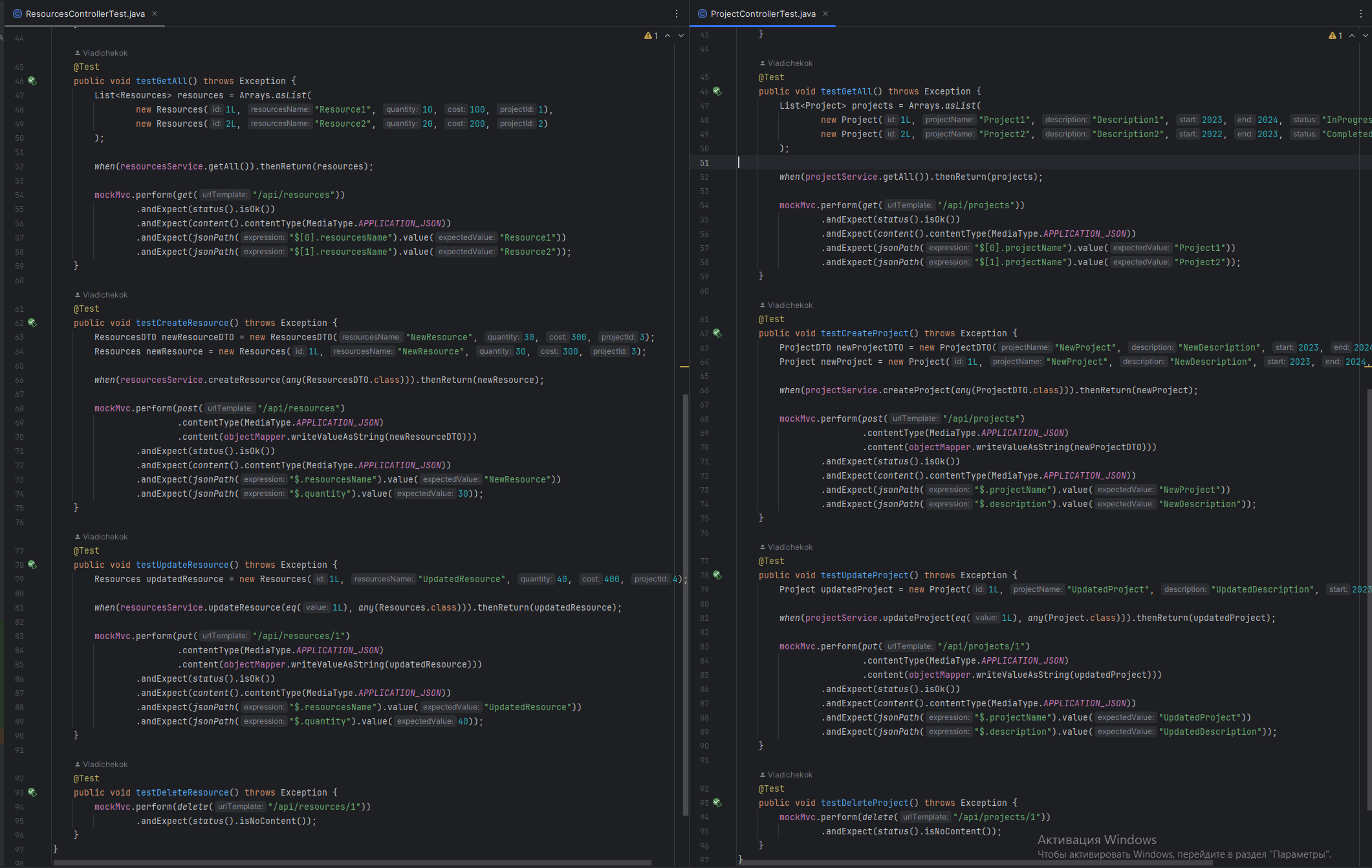
Малюнок 3.1 – Файл свагер конфіг

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

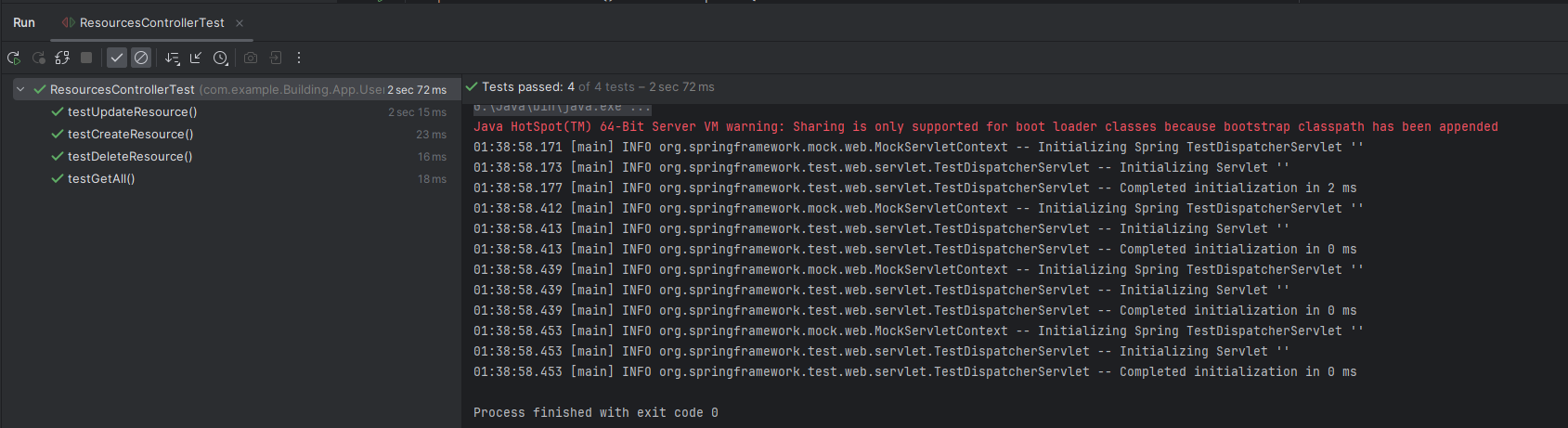
В результаті, маємо наступне:

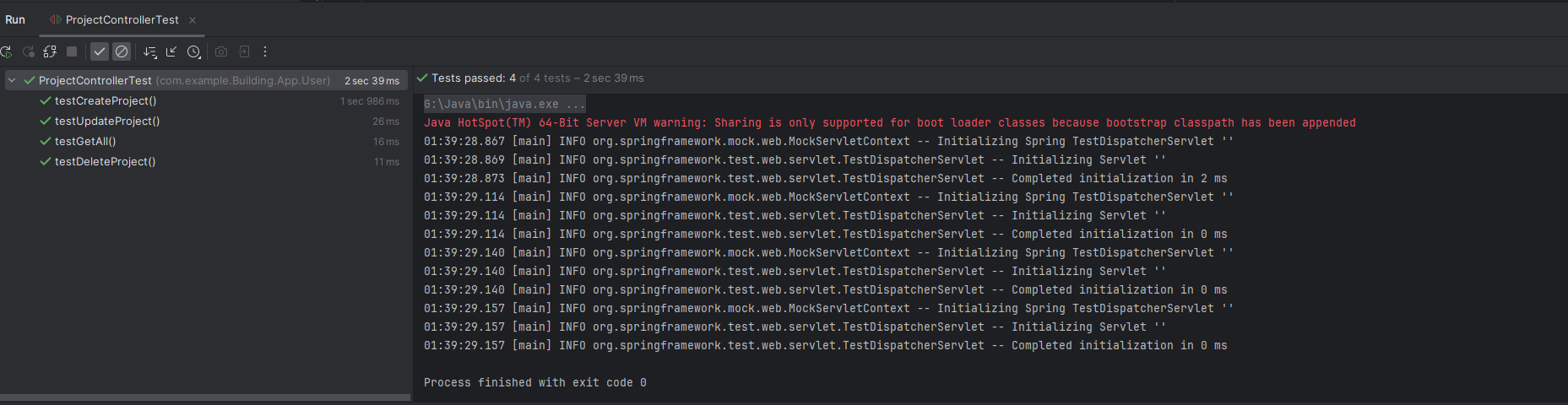
  
Малюнок 3.2 – Відтворений свагер

# **4 НАПИСАННЯ UNIT ТЕСТІВ**

Згадаємо описані в першій частині User Stories, та зробимо тести для ношої програми.  


Малюнок 4.1 – Тести

  
Малюнок 4.2 – Перевірка першого тесту

  
Малюнок 4.3 – Перевірка другого тесту