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

**Курсова робота**

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

**Виконав:** Студент групи ПД-34

**Садлівський Владислав**

**Київ – 2024**

ЗМІСТ

1. **Проектування інформаційної системи**
   1. Опис User stories з діаграмами послідовностей
   2. Опис ERD бази даних
   3. Опис діаграми класів
   4. Написання тестових сценаріїв
2. **Реалізація API**
   1. Підготовка бази даних
   2. Створення Spring Boot Application
   3. Створення сутностей (Entities)
   4. Створення репозиторіїв (Repositories)
   5. Створення сервісів (Services)
   6. Створення контролерів (Controllers)
   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

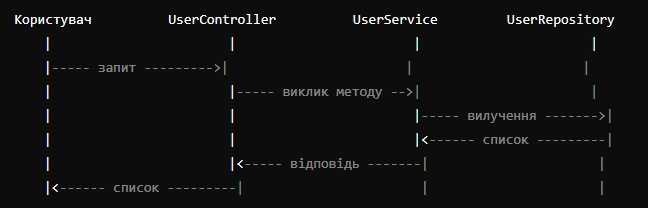
1. **Інтеграційне тестування системи**
   1. Написання інтеграційних автотестів для всіх User stories з позитивними і негативними сценаріями
2. **Використана література**
3. Проектування інформаційної системи
   1. Опис User stories з діаграмами послідовностей

**Перегляд списку користувачів**

Опис:

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

Діаграма послідовності:



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.

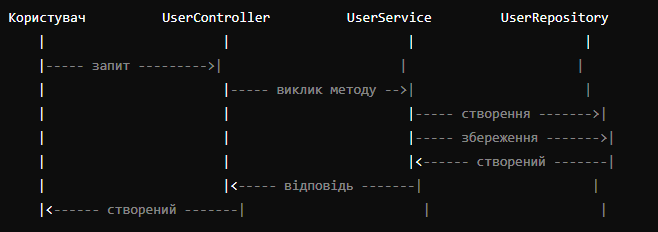
UserController відправляє список користувачів як відповідь користувачеві.

**Створення нового користувача**

Опис:

Як користувач, я хочу створювати нових користувачів у системі.

Діаграма послідовності:



Пояснення:

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

Контролер UserController отримує запит із даними нового користувача.

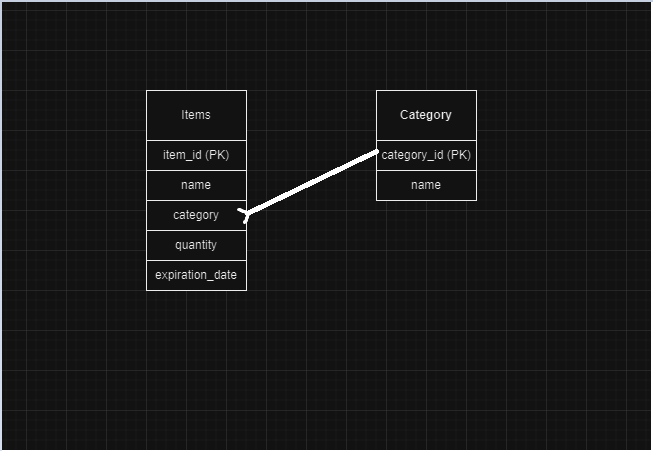
UserController викликає метод createUser() сервісу UserService, передаючи дані нового користувача.

Сервіс UserService створює нового користувача та зберігає його в репозиторії.

Сервіс UserService повертає створеного користувача.

UserController відправляє створеного користувача як відповідь користувачеві.

* 1. Опис ERD бази даних



**Сутність "Category":**

* **category\_id**: унікальний ідентифікатор категорії.
* **name**: назва категорії.

**Сутність "Item":**

* **item\_id**: унікальний ідентифікатор товару.
* **name**: назва товару.
* **category\_id**: зовнішній ключ, що посилається на category\_id у таблиці "Category".
* **quantity**: кількість товару.
* **expiration\_date**: дата завершення терміну придатності товару.

**Зв'язок між "Category" та "Item":**

* Кожна категорія може мати багато товарів, тому в таблиці "Item" є зовнішній ключ category\_id, який посилається на category\_id у таблиці "Category". Це означає, що кожен товар пов'язаний з конкретною категорією. Коли категорія видаляється, всі пов'язані з нею товари також автоматично видаляються (ON DELETE CASCADE), щоб уникнути появи посилань на неіснуючі дані.
  1. Опис діаграми класів

### Опис діаграми класів

**CategoryController:**

* getAllCategories(): List<Category>
* createCategory(CategoryDTO): Category
* updateCategory(Long, Category): ResponseEntity<Category>
* deleteCategory(Long): ResponseEntity<Void>

**ItemController:**

* getAllItems(): List<Item>
* createItem(ItemDTO): Item
* updateItem(Long, Item): ResponseEntity<Item>
* deleteItem(Long): ResponseEntity<Void>

**UserController:**

* getAllUsers(): List<User>
* createUser(UserDTO): User
* updateUser(Long, User): ResponseEntity<User>
* deleteUser(Long): ResponseEntity<Void>

**Services:**

* **CategoryService**
* **ItemService**
* **UserService**

Эта діаграма показує взаємозв'язок між контролерами та відповідними сервісами

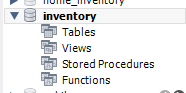
* 1. Написання тестових сценаріїв

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. **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.
3. Реалізація API
   1. Підготовка бази даних

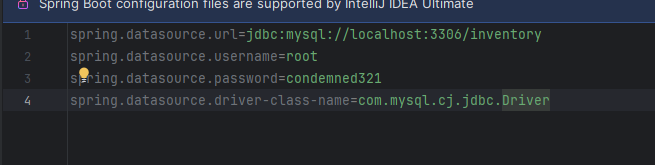
Для початку роботи з БД - треба її створити. Для цього проекту, я використовую СУБД MySQL Workbench.

Створюється БД

маємо створену БД:

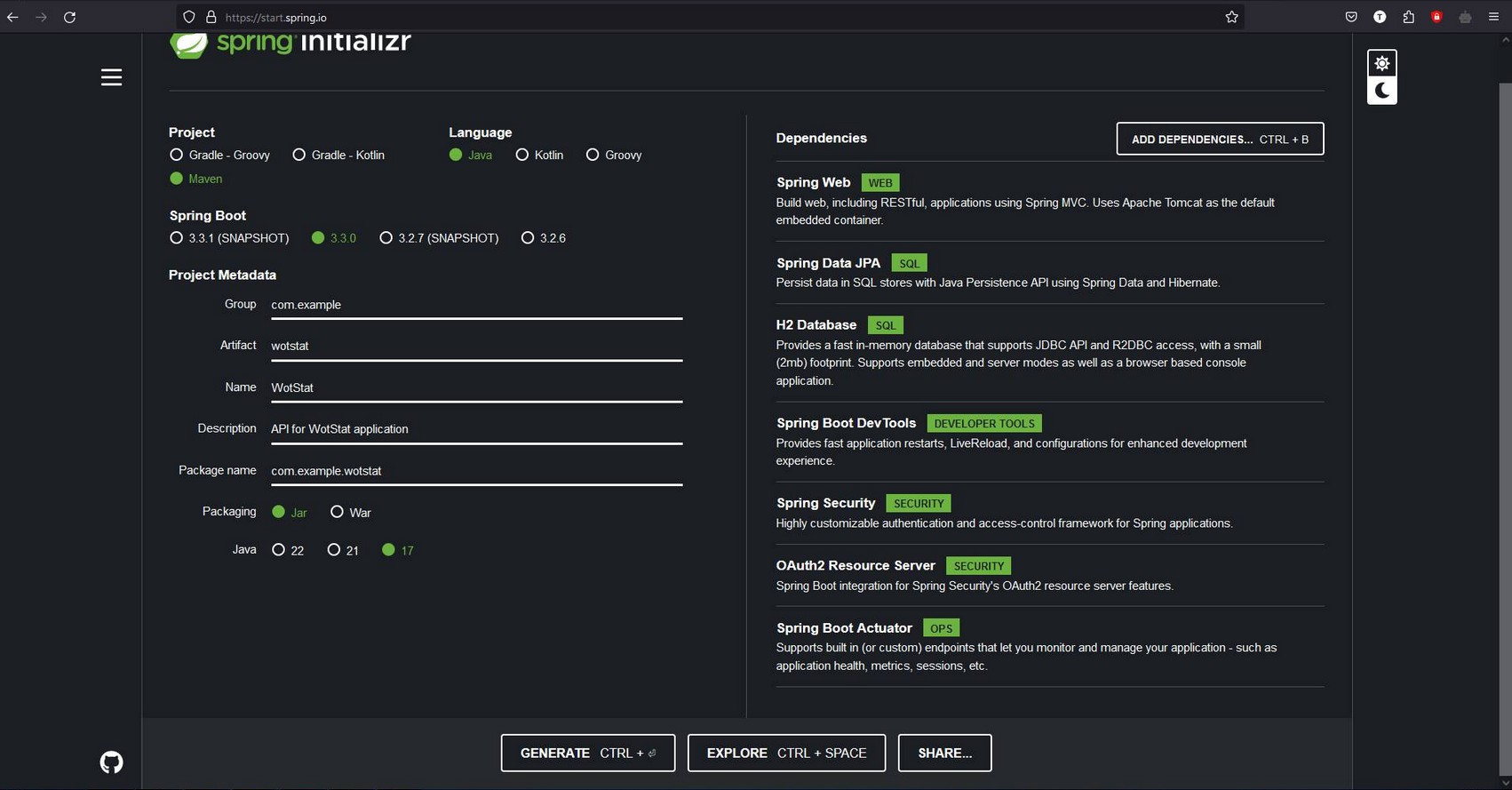


Далі, - локально підключаємось до цієї БД за допомогою команд у файлі application.properties:

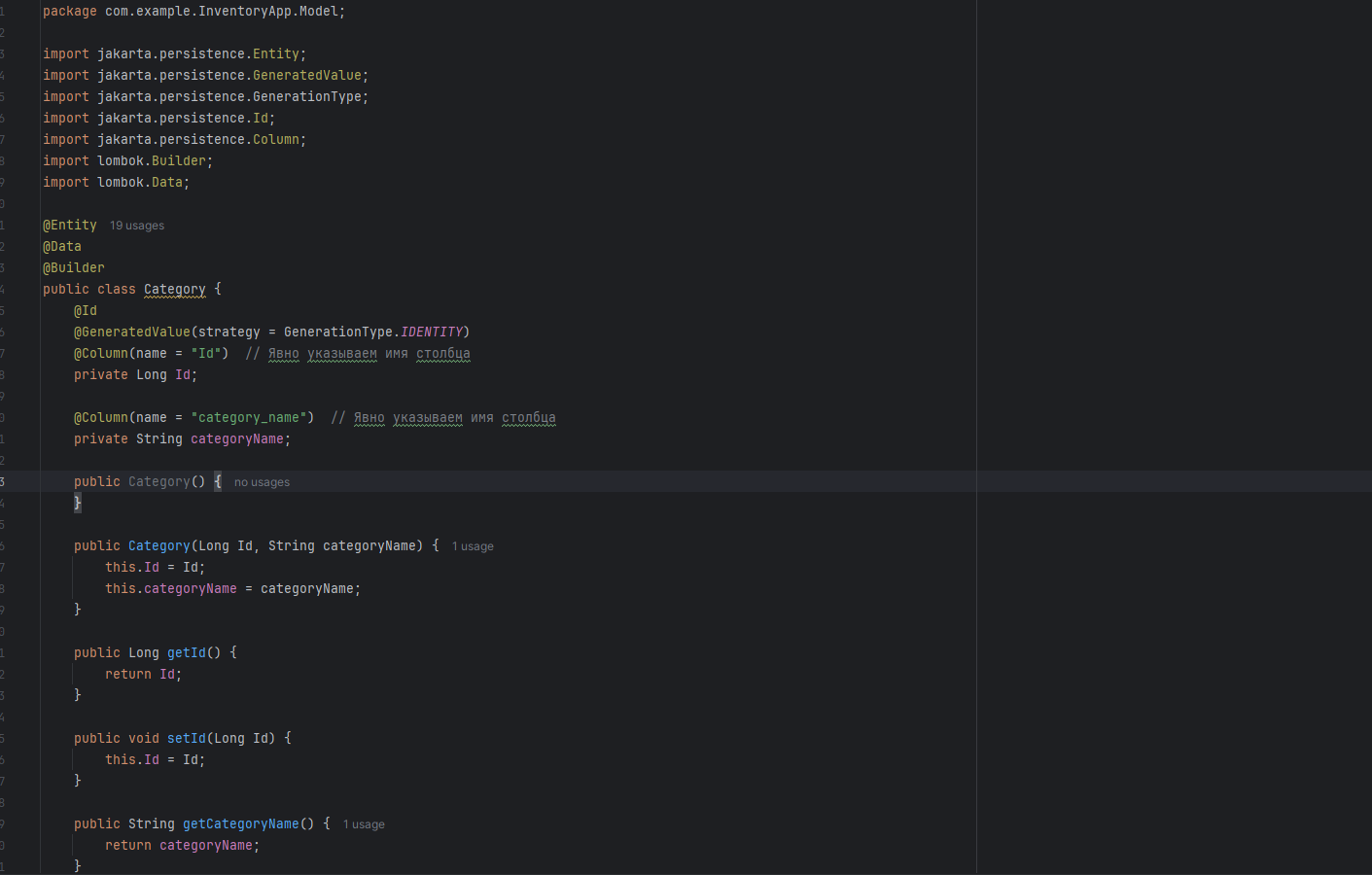


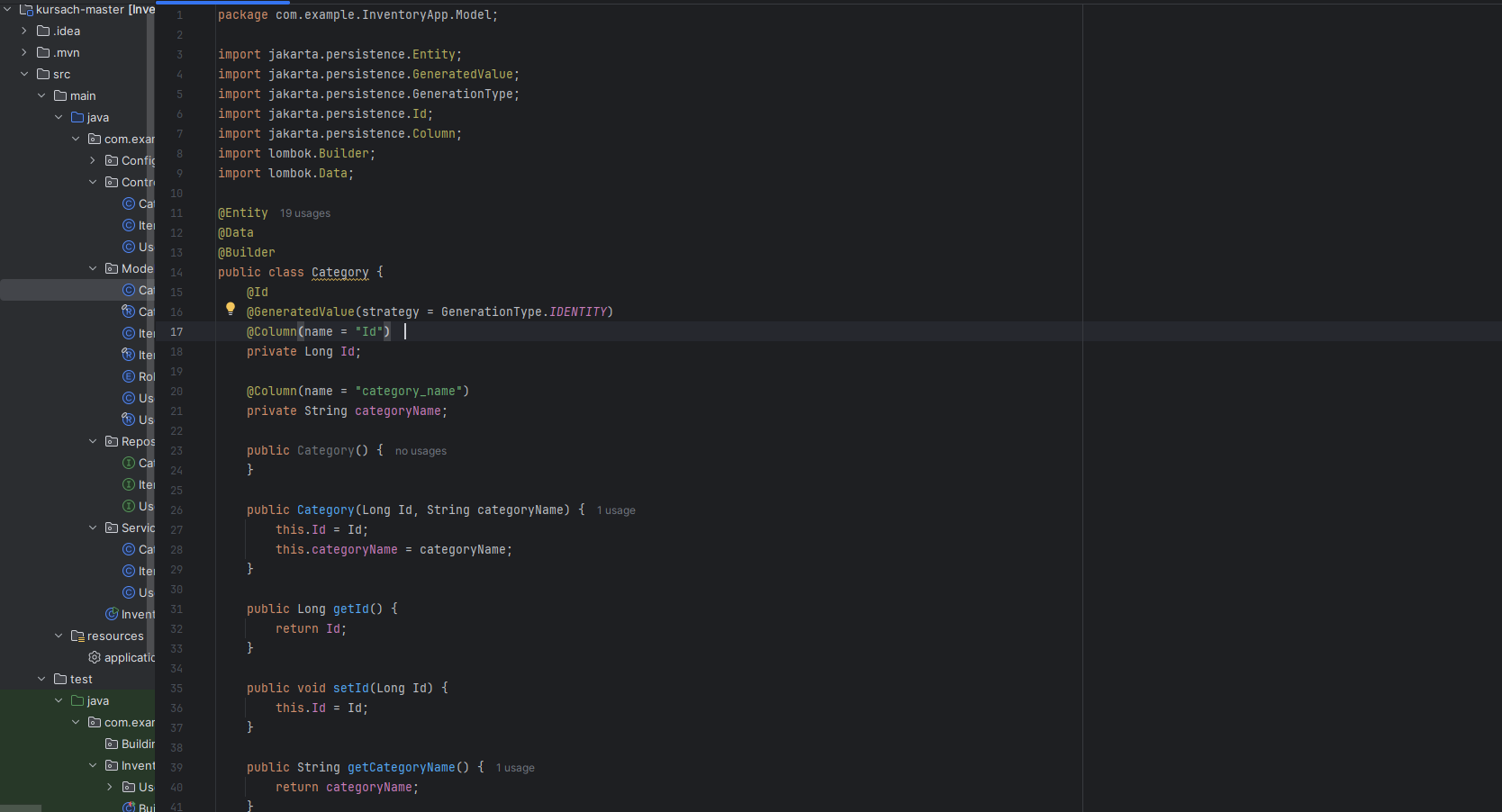
* 1. Створення Spring Boot Application

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



* 1. Створення сутностей (Entities)



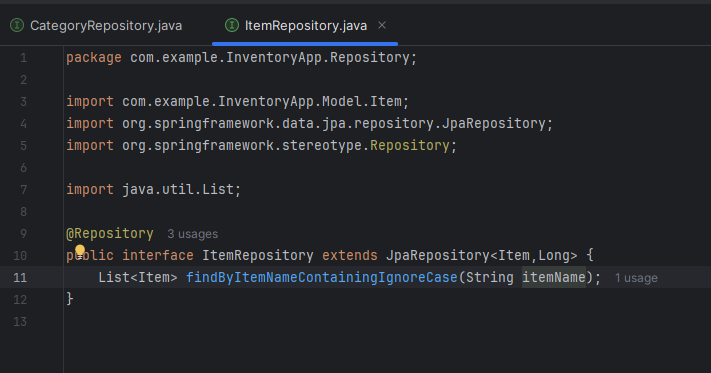


Для кожного поля в сутностях визначені геттери та сеттери, що дозволяє отримувати та встановлювати значення полів.

* 1. 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.

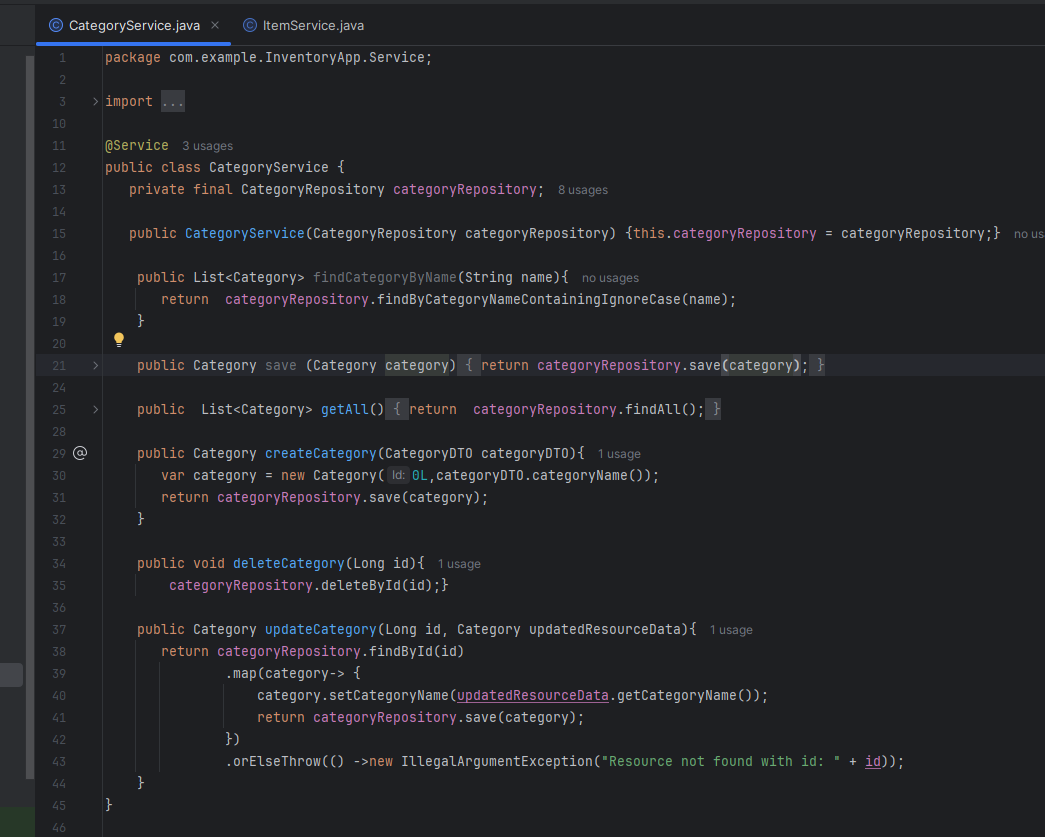


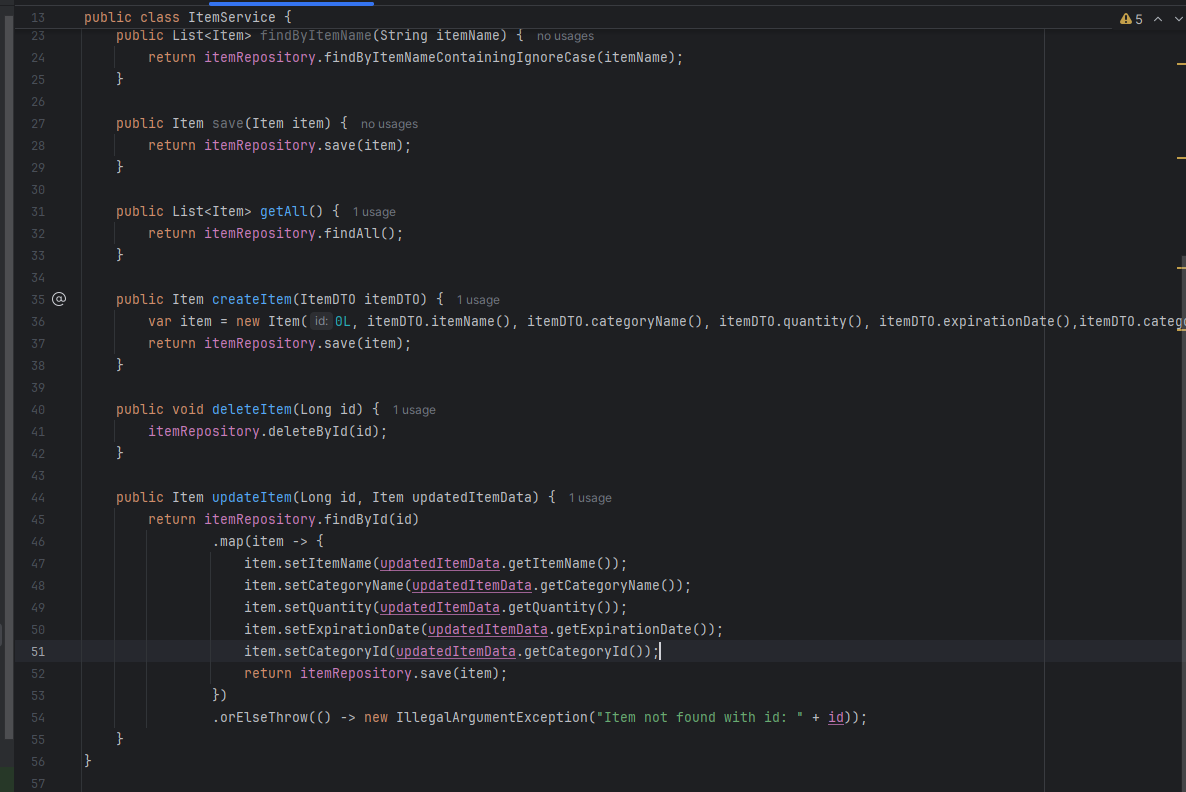


Репозиторії використовуються в сервісах для виконання операцій з базою даних.

* 1. Створення сервісів (Services)

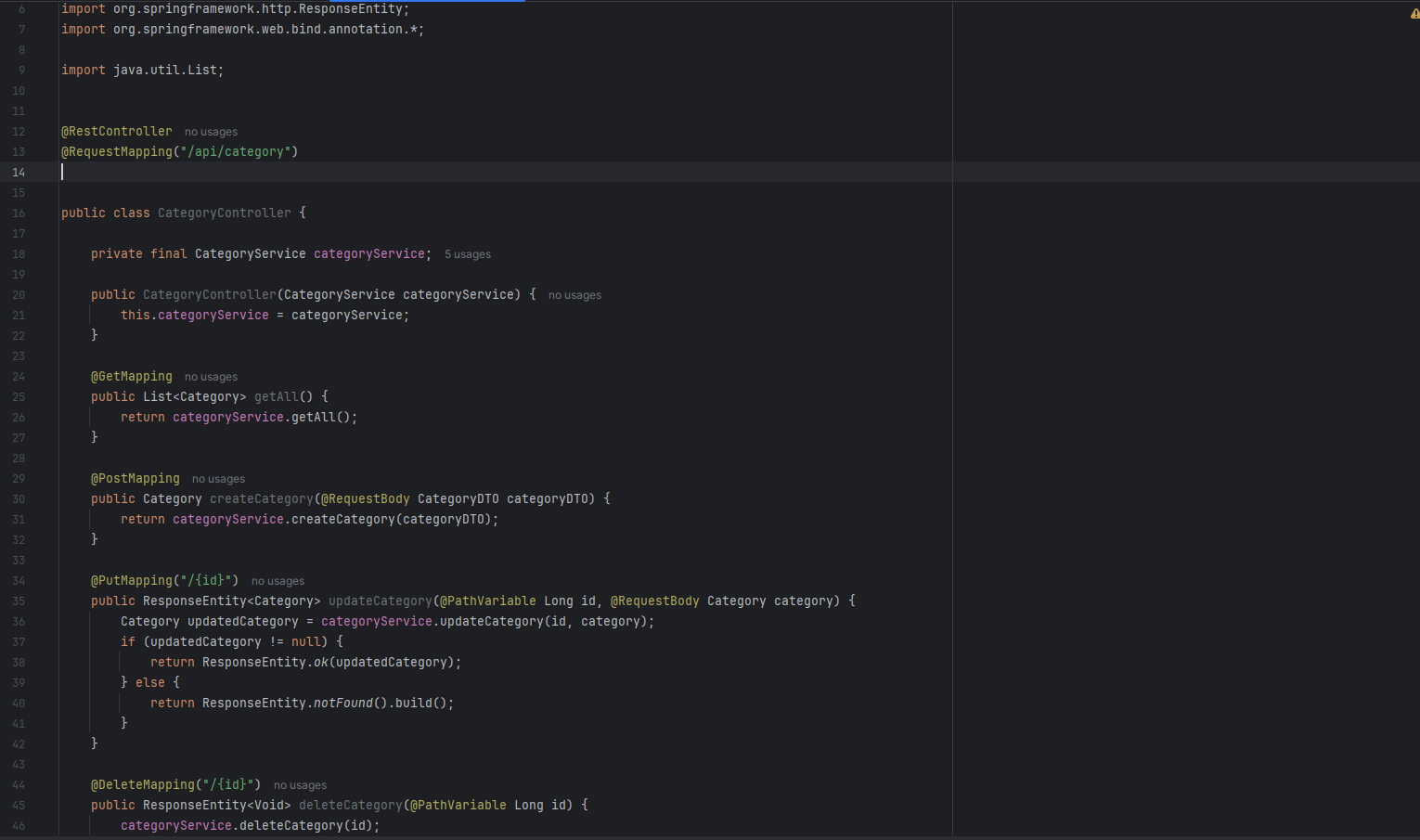
Сервіси (Services) у Spring Framework є важливим компонентом, який відповідає за бізнес-логіку додатку. Вони використовують репозиторії для взаємодії з базою даних і забезпечують абстракцію між контролерами та репозиторіями.

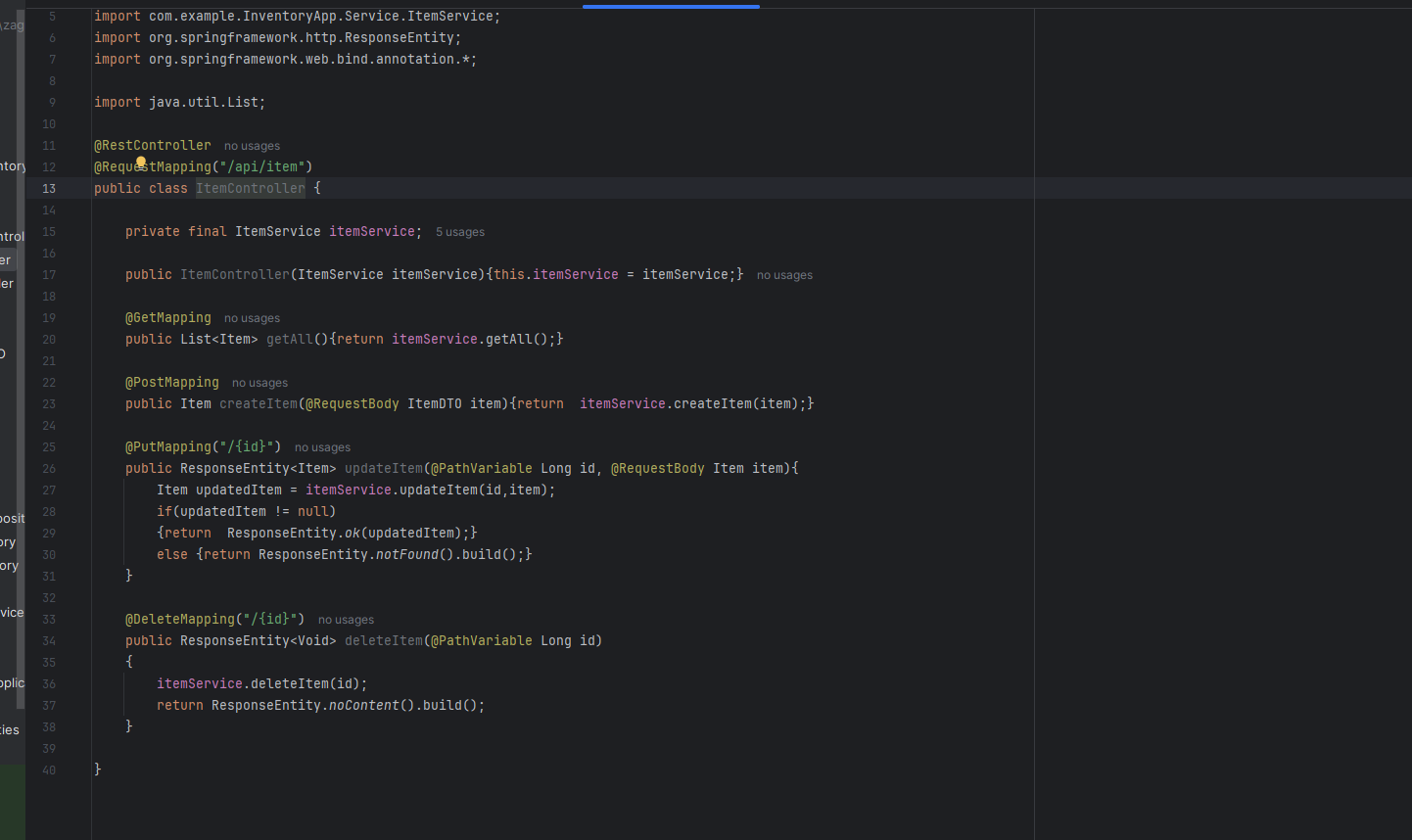




Також, сервіси використовуються в контролерах для обробки HTTP- запитів і виконання відповідних операцій з базою даних через сервіси.

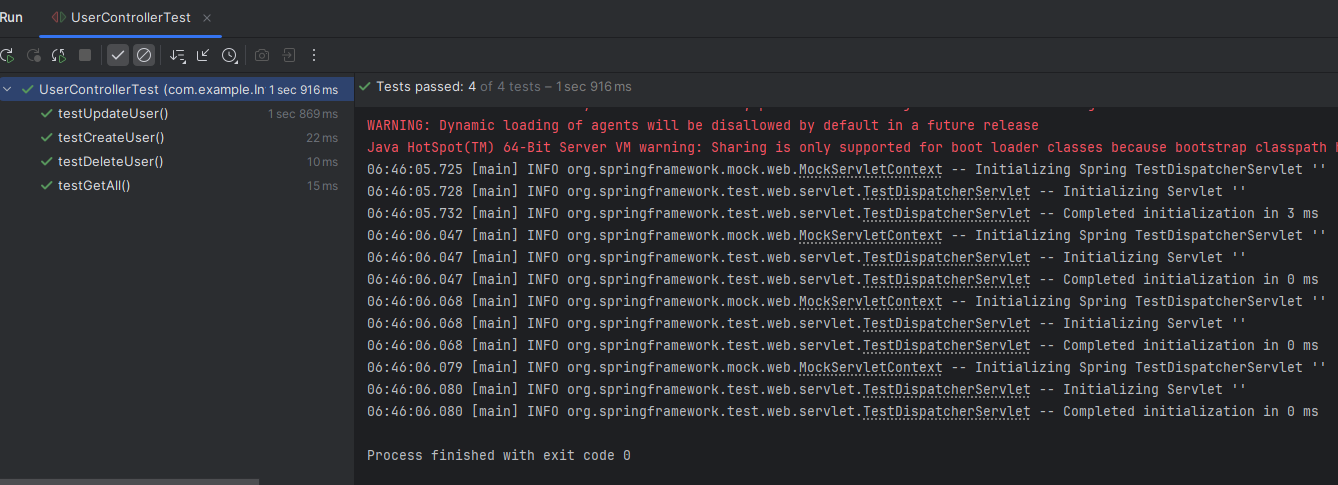
* 1. Створення контролерів (Controllers)





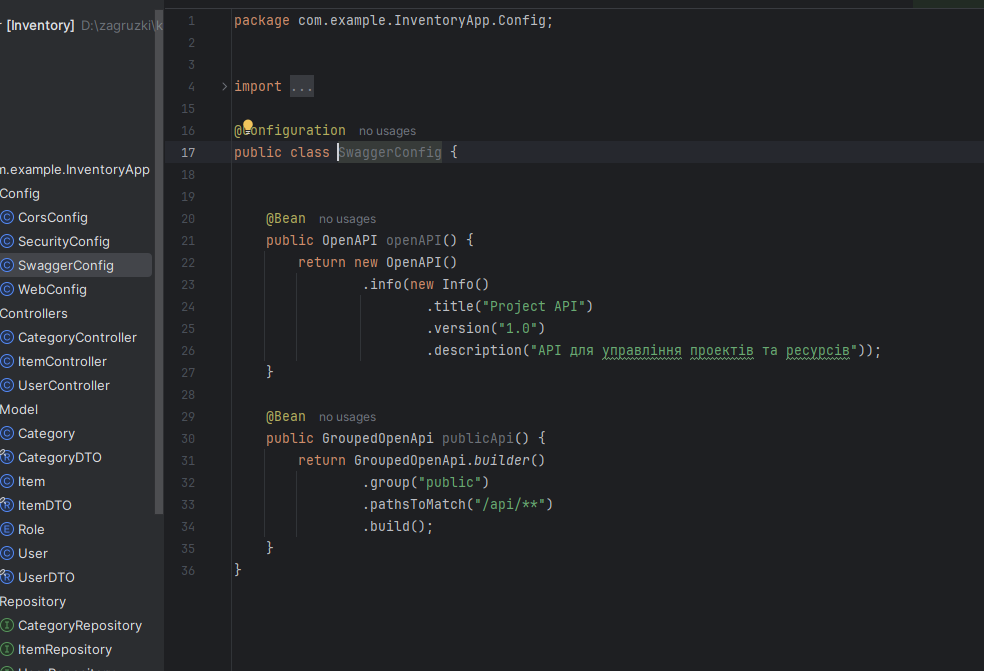
Контролери (Controllers) у Spring Framework відповідають за обробку HTTP-запитів і повернення відповідей клієнту. Вони є частиною архітектури MVC (Model-View-Controller) і взаємодіють із сервісами для виконання бізнес-логіки.

* 1. Написання Unit тестів



* 1. 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.



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:



1. 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