Unit test

**1. Các Thư Viện Phổ Biến Để Viết Unit Test**

**a. JUnit**

* **JUnit 5** là một trong những thư viện viết unit test phổ biến nhất cho Java. Spring Boot mặc định hỗ trợ JUnit khi tạo dự án qua Spring Initializr.
* Cung cấp các chú thích như @Test, @BeforeEach, @AfterEach để quản lý vòng đời của test.

**b. Mockito**

* **Mockito** là thư viện giả lập (mocking) các thành phần phụ thuộc. Giúp bạn dễ dàng giả lập các đối tượng (objects) mà không cần cài đặt logic thực tế.
* Phù hợp cho các bài test không muốn phụ thuộc vào dữ liệu thực hay kết nối với cơ sở dữ liệu.
* Các chú thích thường dùng: @Mock, @InjectMocks, và Mockito.when().

**c. AssertJ**

* **AssertJ** giúp tạo ra các khẳng định (assertions) mạnh mẽ và dễ đọc hơn so với JUnit cơ bản.
* Các khẳng định kiểu fluent API (assertThat()) giúp mã test dễ hiểu và linh hoạt.

**d. Spring Test**

* **Spring Test** là một phần của Spring Framework, cung cấp các công cụ cần thiết để viết test cho các thành phần của Spring, như @SpringBootTest, @WebMvcTest, và @DataJpaTest.
* Giúp bạn dễ dàng kiểm tra các dịch vụ hoặc controller mà không cần khởi động toàn bộ ứng dụng.

**2. Viết Unit Test Cho Các Chức Năng Trong Dự Án**

Để bắt đầu viết unit test, dưới đây là một số bước cơ bản:

**a. Chuẩn Bị Môi Trường Test**

Trong pom.xml, đảm bảo các thư viện spring-boot-starter-test, junit-jupiter, và mockito đã được thêm. Thư viện spring-boot-starter-test sẽ tích hợp hầu hết các công cụ cần thiết.

**b. Viết Test Cho Service Layer**

Ví dụ, có một lớp dịch vụ UserService với phương thức getUserById.

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.Mockito;

import static org.assertj.core.api.Assertions.assertThat;

import static org.mockito.Mockito.when;

class UserServiceTest {

@Mock

private UserRepository userRepository; // Giả lập (mock) lớp UserRepository

@InjectMocks

private UserService userService; // Inject userRepository vào userService

@Test

void testGetUserById() {

User mockUser = new User(1, "John Doe");

when(userRepository.findById(1)).thenReturn(Optional.of(mockUser));

User user = userService.getUserById(1);

assertThat(user.getName()).isEqualTo("John Doe");

}

}

**c. Viết Test Cho Controller Layer**

Với Spring Boot, có thể sử dụng @WebMvcTest để test REST API trong các lớp Controller mà không cần khởi động toàn bộ ứng dụng.

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest(UserController.class)

class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@Test

void testGetUserById() throws Exception {

mockMvc.perform(get("/users/1"))

.andExpect(status().isOk());

}

}

**d. Kiểm Tra Các Kết Quả Mong Đợi**

Sử dụng các khẳng định (assertions) để xác minh kết quả như giá trị trả về, trạng thái HTTP, hoặc nội dung dữ liệu.

**3. Tips Viết Unit Test Hiệu Quả**

* **Nhắm vào từng phương thức riêng lẻ**: Unit test chỉ nên kiểm tra các phương thức hoặc lớp độc lập, không phải toàn bộ hệ thống.
* **Sử dụng các khẳng định (assertions)** mạnh mẽ và rõ ràng để kiểm tra kết quả.
* **Tránh kết nối với database** trong unit test: Sử dụng Mockito để giả lập dữ liệu thay vì thực hiện truy vấn SQL.