**Java testing**

1. Concepts
2. Common types

A diagram of a software testing process

AI-generated content may be incorrect.

Figure 1: common types of testing

* Unit test:

Dùng để test 1 đơn vị đơn lẻ (1 đơn vị thì sẽ có phần phụ thuộc và cũng có phần độc lập, có thể là method, class,…).

Chỉ test code không test các phụ thuộc dependencies khác như database, service,…

Chạy nhanh và dễ thực hiện test nhiều lần.

Unit test không cải thiện hiệu xuất hay hiệu quả của code nhưng nó khiến ta có thể cô lập được lỗi, phục vụ cho quá trình phát triển và bảo trì trong tương lai.

Thường dùng cho tầng service vì ctrl chỉ là phần trung gian để đưa yều cầu về cho service, còn repo thì chỉ lấy dữ liệu từ trong db nếu fake dữ liệu thì không còn ý nghĩa nữa.

* Integration test

Dùng để test sự tương tác giữa các unit như order với payment, product vớ category, service với repository, controller với service, repo với database, full luồng ctr-ser-repo-database…

Có thể liên quan đến databse thực hoặc in-memory, messaging system,…

Chạy chậm hơn do phức tạp hơn và phải tích hợp nhiều unit nhưng có tính thực tế hơn.

* E-E test:

Mô phỏng như người dùng thật để kiểm tra luồng hoạt động của ứng dụng.

1. Test doubles

Dùng để cô lập và mô phỏng hành vi của dependencies.

Gồm nhiều loại test: Dummy, Stub, Mock, Fake, Spy. Nhưng thường gọi chung là Mock vì rất khó phân biệt các loại và trong các thư viện lớn để test double thì phương thức để tạo đều được gọi chung là mock.

1. Best practices

Mỗi assert tập trung vào 1 unit, hành vi, vấn đề cụ thể.

Tuân thủ naming convention, patterns, priciples (AAA, TDD).

Thực hiện data test có ý nghĩa.

Mỗi lần test nên tập trung vào duy nhất một hành vi.

Chỉ nên test các public interface.

1. JUnit5

JUnit là 1 java testing framework được sử dụng phổ biến trong Java. Junit5 là phiên bản mới nhất của JUnit.

Khi viết test nên tuân theo AAA pattern.

1. Architecture

JUnit Platform: ra mắt framework cho việc test

JUnit Jupiter: cung cấp các API cho việc viết tests và extensions

JUnit Vintage: cho phép chạy JUnit3,4 chạy trên JUnit5

1. Annotation

Annotation cung cấp các thông tin bổ sung về mã nguồn, giúp làm rõ cấu trúc và mục đích của mã.

Các annotatin thường gặp:

@Test: biểu thị rằng đây là 1 method cho việc testing

@BeforeEach: biểu thị rằng method gắn anotation này sẽ được chạy trước mỗi test method \*(trước @Before)

@AfterEach: sẽ được thực hiện sau tất cả các method (trước @After)

@BeforeAll: sẽ được thực hiện trước toàn bộ test methods trong class đó (trước @BeforClass)

@AfterAll: sẽ được thực hiện sau toàn bộ test methods trong class đó (trước @AfterClass)

1. Assertion

Assertion hỗ trợ kiểm tra tính đúng đắn của kết quả trả về trong quá trình testing.

Các hàm thông dụng cho việc kiểm tra: *Assertions.assertNotNull, Assertions.assertEquals, Assertions.assertTrue, Assertions.assertSame…*

1. Parameterized tests (Nâng cao)

Parameterized tests hỗ trợ param hóa các giá trị đầu vào nhằm mục đích thực hiện cùng 1 hàm test nhưng test được nhiều trường hợp với các input dầu vào khác nhau thay vì fix cứng và viết nhiều test case cho 1 hàm.

Các annotations thông dụng: @ParameterizedTest, @ValueSource, @NullSource, @EmptySource, @EnumSource,…

1. Mockito

Mockito là 1 thư viện phổ biến của java nhằm hỗ trợ mock các đối tượng và dependencíe trong việc testing giúp test unit 1 cách độc lập.

1. Annotation

@ExtendWith(MockitoExtension.class): khai báo rằng class này là 1 test class.

@Mock: khai báo mock object cho các dependencies. Tương đương với Mockito.mock(MyDependency.class).

@InjectMocks: khai báo class được dùng để test.

1. Stubbing

Sử dụng Mock.when() để định nghĩa giá trị trả về của 1 mock method khi được call đến với các giá trị biết đầu vào cụ thể.

Mockito.when(myDependency.someMethod()).thenReturn("mocked result");

1. Verification

Kiểm tra nếu 1 method được gọi và ngoài ra có thể kiểm tra số lần gọi và giá trị đầu vào.

Các hàm thông dụng:

Mockito.verify(mock, times(2)).methodCall(args);

Mockito.verifyNoMoreInteractions(mock);

1. Agrument matchers

Là tính năng đặc biệt trong việc stubbing và verification, cung cấp các tùy biến linh hoạt cho biến để mock objects thay vì nhập chính xác giá trị của biến đầu vào của method.

Khi dùng agrument matchers, bắt buộc các args đều phải dùng agrument matchers, không được mix giữa biến đầu vào cố định và agrument matchers.

Được dùng trong method được gọi bởi stubbing hoặc verification, không được dùng để trả về trong thenReturn().

Common agrument matchers:

Any(), anyIn(), anyList(), any(Class<T>), eq(value), argThat()

1. AgrumentCaptor

Được sử dụng trong quá trình verification, dùng để lấy giá trị thực khi được gọi bởi mock method.

Thường được dùng trong các trường hợp như:

* Giá trị của biến đầu vào không được lấy chính xác bởi đầu vào mà được xử lý bên trong hàm hoặc dùng hàm any.
* Cần kiểm tra giá trị của trường sau khi test.
* Test callback.
* Kiểm tra giá trị trong hàm void, vì hàm void không return giá trị.

Cách sử dụng:

* @Captor + AgrumentCaptor<T>: dùng để khai báo instance của AgrumentCaptor.
* verify(myMockedDependency).methodCall(myObjectCaptor.capture()): dùng để bắt giá trị các biến đầu vào sau khi thực hiện hàm.
* captor.getValue(): lấy giá trị của biến đầu vào.
* captor.getAllValue(): lấy list của giá trị đầu vào.

1. Spring
2. Phân loại

Unit Test: test class đơn lẻ một cách riêng lẻ (không cần Spring context)

Slice Test: test 1 tầng layer đặc biệt hoặc 1 phần của project và load 1 phần Spring context.

Integration Test: Xử dụng Spring Context để test beans.

E2E: test toàn bộ ứng dụng với tất cả components thường là với HTTP.

1. Load Spring Context

* Load toàn bộ context:

Quan trọng trong integration test.

Using @SpringBootTest annotation để load toàn bộ application context, cùng với tất cả beans và config sẽ được load khi test.

* Load 1 phần context:

Giới hạn components và config được load để thực hiện test, phù hợp slice test.

Annotations:

* @WebMvcTest: Chỉ load các bean liên quan đến và controller được thực hiện test. Và có cung cấp MockMvc bean.
* @DataJpaTest: Chỉ load các JPA components và repository được thực hiện test, và tự động cấu hình 1 in-memory database.
* @RestClientTest: Cấu hình các components cần thiết cho việc test REST client (client layer bean). Có cung cấp WebTestClient bean.

1. Mock bean

Spring boot cung cấp annotation @MockBean để mock bean thay thế cho real bean trong Spring Context trong quá trình test.

Được sử dụng trong slice test và integration test trong mocking dependencies.

1. MockMvc

MockMvc cho phép test HTTP endpoints mà không cần khởi chạy toàn bộ HTTP server. Cung cấp 1 cách để thực hiện testing web requests và xác minh kết quả (status code, headers, response body,..)

Spring Boot tự động config MockMvc khi sử dụng @WebMvcTest, hoạt động khi gọi ngăn xếp Sping MVC trong bộ nhớ không phải qua mạng.

1. Use cases

Test controller layer trong unit test hoặc sử dụng trong slice test, với annotation @WebMvcTest.

Trong integration test sử dụng annotation @AutoConfigureMockMvc và E2E test sử dụng annotaion @SpringBootTest.

1. Test Spring Security

Thực hiện test hoạt động của security configuration, author và authen rules có được hực thi đúng như mong đợi hay không.

Các annotations thông dụng: @WithMockUser, @WithAnonymousUser, @WithUserDetails, @WithSecurityContext.