Como implementar Testes de Integração no Projeto Biblioteca Digital

Este tutorial demonstra como criar testes de integração para a classe BibliotecaService, focando em sua interação com AutorRepository em um projeto Quarkus.

Aprenderemos a usar o Mockito para "simular" o comportamento do repositório. Isso é crucial porque nos permite testar a lógica do BibliotecaService de forma isolada, garantindo que ele funcione como esperado, sem a necessidade de uma conexão real com o banco de dados. Essa abordagem torna os testes mais rápidos, confiáveis e focados na lógica de negócio da camada de serviço.

O que são Testes de Integração?
São testes para verificar como os módulos ou componentes de um aplicativo funcionam juntos. O objetivo principal é garantir que a comunicação e a interação entre essas partes sejam bem-sucedidas, identificando interoperabilidade e troca de informações que não seriam detectadas se cada módulo fosse testado isoladamente.

Pré-requisitos:

- O projeto "Sistema de Biblioteca Digital" funcionando.
- JUnit5 e Mockito.

Passo a Passo:

Seção 1: No arquivo POM.xml adicione as seguintes dependências.

```
<dependency>
   <groupId>org.mockito</groupId>
   <artifactId>mockito-core</artifactId>
   <version>5.11.0</version>
   <scope>test</scope>
</dependency>
<dependency>
   <groupId>io.quarkus</groupId>
   <artifactId>quarkus-junit5-mockito</artifactId>
   <scope>test</scope>
</dependency>
```

Seção 2: Entendendo a Interação a ser testada primeiro, vamos observar que o BibliotecaService injeta e utiliza o AutorRepository para acessar os dados dos autores. Os métodos que vamos testar são:

•listarTodosAutores(): Chama autorRepository.findAll().
•listarTodosAutoresComSeusLivros(): Chama autorRepository.findAllComLivros().

•contarTotalAutores(): Chama autorRepository.count().Nosso objetivo é garantir que o BibliotecaService chame esses métodos e manipule os resultados corretamente.

Seção 3: Criando a Classe de Teste e Simulando o Repositório (Mocking).
Vamos criar o arquivo BibliotecaServiceTest.java no diretório src/test/java/br/upf/ads175/service/.

Dentro desta classe, em vez de injetar o AutorRepository real com @Inject, usaremos @InjectMock. Esta anotação especial do Quarkus substitui a implementação real do AutorRepository por um "mock" (um objeto simulado).

```java
package br.upf.ads175.service;

import br.upf.ads175.entity.Autor;
import br.upf.ads175.repository.AutorRepository;
import io.quarkus.test.InjectMock;
import io.quarkus.test.junit.QuarkusTest;
import jakarta.inject.Inject;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

import java.util.ArrayList;
import java.util.List;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;

@QuarkusTest  new *
public class BibliotecaServiceTest {

    @Inject  3 usages
    private BibliotecaService bibliotecaService;

    @InjectMock  3 usages
    private AutorRepository autorRepository;
```

Seção 4: Vamos escrever os testes para cada método, usando o padrão "Arrange, Act, Assert" (Organizar, Agir, Verificar).

1- Organizar: Preparamos o ambiente, principalmente dizendo ao nosso mock como se comportar usando Mockito.when().

2- Agir: Executamos o método do bibliotecaService que estamos testando.

3- Verificar: Usamos asserções (assertEquals, assertNotNull, etc.) para confirmar se o resultado foi o esperado.

Dentro da classe BibliotecaServiceTest:

```java
@Test  new *
public void testListarTodosAutores() {
    List<Autor> autoresMock = new ArrayList<>();
    autoresMock.add(new Autor());
    autoresMock.add(new Autor());

    Mockito.when(autorRepository.findAll()).thenReturn(autoresMock);

    List<Autor> autores = bibliotecaService.listarTodosAutores();

    assertNotNull(autores);
    assertEquals( expected: 2, autores.size());
}

@Test  new *
public void testListarTodosAutoresComSeusLivros() {
    List<Autor> autoresMock = new ArrayList<>();
    autoresMock.add(new Autor());
    autoresMock.add(new Autor());

    Mockito.when(autorRepository.findAllComLivros()).thenReturn(autoresMock);

    List<Autor> autores = bibliotecaService.listarTodosAutoresComSeusLivros();

    assertNotNull(autores);
    assertEquals( expected: 2, autores.size());
}
```

```java
    @Test  new *
    public void testContarTotalAutores() {
        Mockito.when(autorRepository.count()).thenReturn( t: 5L);

        Long count = bibliotecaService.contarTotalAutores();

        assertNotNull(count);
        assertEquals( expected: 5L, count);
    }
}
```

Verificando o resultado:

Para executar os testes, execute o comando “quarkus test” no terminal, na raiz do projeto. Após a execução, você verá um relatório indicando que todos os testes passaram com sucesso.

```
--
2025-10-29 20:47:32,030 INFO  [io.qua.test] (Test runner thread) Running 1/3. Running: br.upf.ads175.service.BibliotecaServiceTest#testContarTotal
tores()
2025-10-29 20:47:32,276 INFO  [io.qua.test] (Test runner thread) Running 2/3. Running: br.upf.ads175.service.BibliotecaServiceTest#testListarTodos
toresComSeusLivros()
2025-10-29 20:47:32,284 INFO  [io.qua.test] (Test runner thread) Running 3/3. Running: br.upf.ads175.service.BibliotecaServiceTest#testListarTodos
tores()
2025-10-29 20:47:32,331 INFO  [io.quarkus] (Test runner thread) TDE-biblioteca(test application) stopped in 0.039s


--
All 3 tests are passing (0 skipped), 3 tests were run in 5657ms. Tests completed at 20:47:32.
Press [r] to re-run, [:] for the terminal, [h] for more options>
```