Guia Prático de Testes em Java com JUnit, Mockito e MockMvc

# 1. Introdução

Este guia foi criado para ajudar o Grupo 10 a entender, implementar e expandir testes automatizados no projeto Techseller, usando boas práticas de Engenharia de Software com Java, JUnit, Mockito e Spring MockMvc.

# 2. Estrutura de Testes no Projeto

A estrutura de testes segue o padrão Maven:  
• Código fonte: src/main/java  
• Testes automatizados: src/test/java  
Os testes devem ser criados paralelamente às classes que testam (ex: ProdutoServiceTest testa ProdutoService).

# 3. Conceitos Importantes

## 3.1 JUnit

JUnit é o framework padrão de testes unitários em Java. Permite escrever métodos de teste para verificar se uma funcionalidade está funcionando como esperado. Exemplo básico:  
@Test  
void soma() {  
 assertEquals(5, 2 + 3);  
}

## 3.2 Mockito

Mockito é uma biblioteca de 'mocking' — ela permite simular comportamentos de objetos dependentes. Muito útil para testar serviços sem acessar o banco de dados real.  
Exemplo:  
when(userRepository.findByEmail("abc@xyz.com")).thenReturn(Optional.of(user));

## 3.3 MockMvc

MockMvc é uma ferramenta do Spring usada para testar controladores (Controllers) como se fosse uma requisição HTTP, sem precisar rodar o servidor. Muito útil para testar endpoints REST.  
Exemplo:  
mockMvc.perform(post("/produto")).andExpect(status().isOk());

# 4. Anotações (Annotations)

* @Test — Indica que o método é um teste unitário. Vem do JUnit.
* @Mock — Cria um mock (simulação) de uma dependência. Vem do Mockito.
* @InjectMocks — Injeta os mocks anotados em uma instância real da classe testada. Vem do Mockito.
* @BeforeEach — Executa o método antes de cada teste. Usado para configurar o ambiente.
* @WebMvcTest — Carrega apenas os componentes relacionados ao Controller, sem subir todo o contexto do Spring Boot.
* @Autowired — Injeta uma dependência gerenciada pelo Spring (como MockMvc, Service, etc).
* @MockBean — Usado em testes Spring para simular beans (serviços). Substitui um bean real por um mock.

# 5. Expansão futura dos testes

À medida que o projeto crescer, crie novos testes sempre que adicionar uma nova funcionalidade. Recomendações:  
• Para cada novo método em serviços (ex: CalcularDesconto), crie um método de teste correspondente.  
• Para novos endpoints, adicione testes em Controller usando MockMvc.  
• Mantenha cobertura de testes acima de 80% sempre que possível.  
• Use classes utilitárias para gerar dados de teste com facilidade.