

**Planejamento de Teste**

**Nome dos Alunos:**

**Nome: Rafael Oliveira**

**RA: 823128825**

**Nome: Kauã Freitas Santana Silva**

**RA: 823116576**

**nome: Jonatan Mateus de Souza Alves**

**RA: 823121880**

**Nome: Letícia Girardi Marra**

**RA:823126898**

**Lorena Oliveira Solaci**

**RA: 823112471**

**2024**

**Plano de Teste com TDD**

**1. Introdução**

O objetivo deste plano de teste é garantir que todas as funcionalidades e requisitos descritos para o sistema de um mercado sejam devidamente testados utilizando a abordagem **Test-Driven Development (TDD)**.

Este plano de teste foca em validar tanto os requisitos funcionais quanto os não-funcionais, assegurando que o sistema atenda aos objetivos de automação, inovação e eficiência definidos no escopo do projeto. O ciclo de desenvolvimento do TDD é seguido em três etapas:

1. **Escrever o teste**: Antes de implementar a funcionalidade.
2. **Desenvolver o código**: O código é escrito apenas o suficiente para passar no teste.
3. **Refatorar**: O código é melhorado e otimizado após os testes passarem.

**2. Escopo**

O plano de teste abrangerá os seguintes módulos do sistema:

* **Gestão de Produtos**
* **Gestão de Fornecedores**
* **Gestão de Vendas**
* **Integração com Sistemas de Pagamento**
* **Gestão de Usuários**

Cada módulo será testado usando TDD, garantindo que o código seja desenvolvido em conformidade com os testes previamente definidos.

**3. Abordagem**

A abordagem será dividida em **testes funcionais** e **testes não-funcionais**:

* **Testes funcionais**: Validação das entradas e saídas das funcionalidades descritas.
* **Testes não-funcionais**: Foco em desempenho, segurança, usabilidade e compatibilidade.

Com o TDD, para cada funcionalidade, um teste será escrito antes de qualquer código. Quando o teste falhar (o que é esperado), o código será implementado para passar no teste. Uma vez que o teste for aprovado, o código será refatorado para garantir sua eficiência e clareza.

**4. Ambiente de Teste**

O ambiente de teste será configurado para suportar os testes automatizados e manuais:

* **Navegadores**: Chrome, Firefox, Edge.
* **Dispositivos**: Desktop, tablet, smartphone.
* **Integração**: Com sistemas de pagamento simulados e bancos de dados de teste.
* **Ferramentas de Teste**: Uso de frameworks de teste como **JUnit**, **pytest** ou **Selenium** para automatizar a execução de testes e permitir a integração contínua.

------------------------------------------------------------------------------------------------------------------------------

**Casos de Teste**

Caso de Teste 1: Cadastro de Produtos

**Objetivo:** Validar se o sistema permite o cadastro correto de novos produtos.  
**Pré-condição:** O sistema deve estar funcionando corretamente e o usuário deve ter permissão de administrador.  
**Entradas:** Nome do produto, código SKU, preço, quantidade em estoque.  
**Passos do Teste (em python):**

*def test\_cadastro\_produto():*

*produto = cadastrar\_produto(nome="Produto Teste", sku="12345", preco=10.99, estoque=100)*

*assert produto.nome == "Produto Teste"*

*assert produto.sku == "12345"*

*assert produto.preco == 10.99*

*assert produto.estoque == 100*

**Código a ser desenvolvido:** Após escrever o teste, implementar o código para passar o teste:

*def cadastrar\_produto(nome, sku, preco, estoque):*

*return Produto(nome=nome, sku=sku, preco=preco, estoque=estoque)*

**Refatoração:** Uma vez que o teste for aprovado, refatorar para incluir validações adicionais, como duplicidade de SKU e regras de preço.

#### Caso de Teste 2: Atualização de Preços (RF-02)

**Objetivo:** Verificar se o sistema permite a atualização de preços dos produtos existentes.  
**Pré-condição:** O produto deve já estar cadastrado no sistema.  
**Passos do Teste:**

*def test\_atualizacao\_preco\_produto():*

*produto = cadastrar\_produto(nome="Produto Teste", sku="12345", preco=10.99, estoque=100)*

*produto.atualizar\_preco(15.99)*

*assert produto.preco == 15.99*

**Código a ser desenvolvido:** Implementar o código para passar no teste:

*class Produto:*

*def \_\_init\_\_(self, nome, sku, preco, estoque):*

*self.nome = nome*

*self.sku = sku*

*self.preco = preco*

*self.estoque = estoque*

*def atualizar\_preco(self, novo\_preco):*

*self.preco = novo\_preco*

**Refatoração:** Incluir regras de negócio para validações de preço e logs de alteração.

#### Caso de Teste 3: Processamento de Vendas (RF-07)

**Objetivo:** Testar se o sistema processa uma venda corretamente, atualizando o estoque e gerando os relatórios necessários.  
**Pré-condição:** Produtos e clientes cadastrados, método de pagamento configurado.  
**Passos do Teste:**

*def test\_processamento\_venda():*

*produto = cadastrar\_produto(nome="Produto Teste", sku="12345", preco=10.99, estoque=100)*

*venda = processar\_venda(produto, quantidade=5, metodo\_pagamento="Cartão")*

*assert venda.quantidade == 5*

*assert produto.estoque == 95 # Estoque deve ser reduzido após a venda*

*assert venda.metodo\_pagamento == "Cartão"*

**Código a ser desenvolvido:** Após o teste, implementar o código para processar a venda:

*def processar\_venda(produto, quantidade, metodo\_pagamento):*

*if produto.estoque < quantidade:*

*raise Exception("Estoque insuficiente")*

*produto.estoque -= quantidade*

*return Venda(produto=produto, quantidade=quantidade, metodo\_pagamento=metodo\_pagamento)*

**Refatoração:** Incluir validações para estoque insuficiente e logs de transações.

----------------------------------------------------------------------------------------------------------

**Roteiro de Teste Usando TDD**

**Objetivo Geral:** Garantir a correta implementação dos requisitos do sistema, utilizando TDD para validar cada funcionalidade. Para cada caso de uso, serão seguidos os seguintes passos:

#### 1. Gestão de Produtos

1.1. Escrever testes para o cadastro de produtos, incluindo validação de campos obrigatórios e formatos.  
1.2. Implementar o código para passar os testes.  
1.3. Refatorar para garantir eficiência e atender a requisitos de negócio (e.g., duplicidade de SKU).

#### 2. Gestão de Fornecedores

2.1. Escrever testes para cadastrar e gerenciar fornecedores, simulando a criação de pedidos e relatórios.  
2.2. Implementar o código para processar as ações de gestão de fornecedores.  
2.3. Refatorar para incluir validações como a verificação de CNPJ e a notificação automática.

#### 3. Gestão de Vendas

3.1. Escrever testes para o processo de vendas, incluindo a atualização de estoque, método de pagamento e relatório de vendas.  
3.2. Implementar o código para processar a venda e atualizar o estoque.  
3.3. Refatorar o código para otimizar o desempenho e garantir segurança nas transações.

#### 4. Integração com Sistemas de Pagamento

4.1. Escrever testes para diferentes métodos de pagamento (cartão, boleto, etc.).  
4.2. Implementar a lógica de integração de pagamento.  
4.3. Refatorar o código para melhorar a segurança e a usabilidade.

#### 5. Gestão de Usuários

5.1. Escrever testes para garantir que permissões e acessos estão configurados corretamente.  
5.2. Implementar o código para a gestão de usuários e autenticação.  
5.3. Refatorar o código para incluir automação de processos e auditoria de acessos.

### Conclusão

O uso do **Test-Driven Development (TDD)** no plano de testes garante que cada parte do sistema seja devidamente validada e funcione conforme os requisitos funcionais e não-funcionais. O processo contínuo de escrever testes antes de codificar e refatorar o código melhora a qualidade, a confiabilidade e a eficiência do sistema.