Relatório de Testes

1. Introdução

Objetivo do Sistema

O sistema foi desenvolvido para gerenciar **usuários** e **produtos**, oferecendo operações de **cadastro**, **atualização**, **remoção** e **listagem**. Além disso, a aplicação implementa regras específicas para:

- **Usuários**: Validação de senha, unicidade de login e manipulação de uma lista de usuários.
- **Produtos**: Garantia de código único, validação de nome, descrição e preço.

O objetivo dos testes é validar a consistência, a integridade e a conformidade dessas operações e regras de negócio.

---

2. Requisitos Funcionais

2.1 Cadastro de Usuários

O sistema deve permitir o gerenciamento de usuários com as seguintes funcionalidades e restrições:

1. **Adicionar um novo usuário** com login e senha.
    - Login e senha devem ser strings.
    - A senha deve atender aos seguintes critérios:
        - Ter pelo menos 4 caracteres.
        - Não ser igual ao login.
        - Não conter espaços em branco no início ou no final.
    - Caso a senha não seja válida, o sistema deve lançar uma exceção do tipo IllegalArgumentException.
2. **Remover usuários existentes** com base no login.
3. **Atualizar a senha** de um usuário existente, garantindo que ela cumpra as regras de validação.
4. **Evitar duplicidade de usuários** com o mesmo login.
5. Retornar todos os usuários cadastrados no sistema através de um método que forneça uma lista ou coleção de usuários.

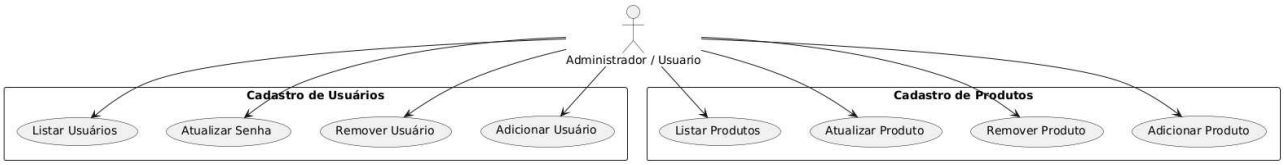
2.2 Cadastro de Produtos

O sistema deve permitir o gerenciamento de produtos com as seguintes funcionalidades e validações:

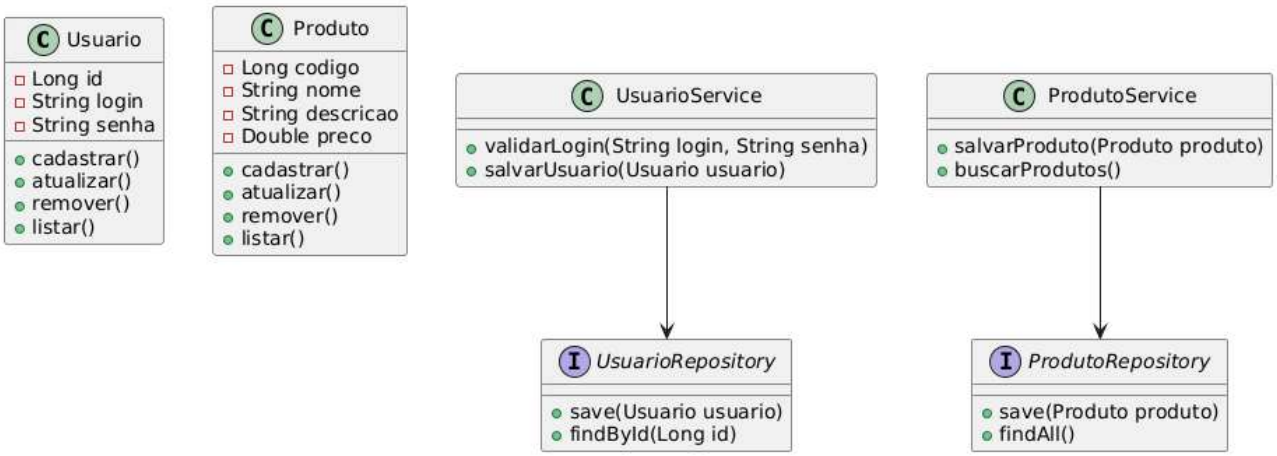
1. **Adicionar um novo produto** ao sistema.
    - Validar as seguintes regras antes de adicionar:

        - Cada produto deve ter um código único.
        - O nome do produto não pode ser nulo ou vazio.
        - A descrição deve ter no máximo 200 caracteres.
        - O preço do produto deve ser maior que zero.
    - Caso alguma validação falhe, o sistema deve lançar uma exceção do tipo IllegalArgumentException.
2. **Remover produtos existentes** do sistema com base no código.
3. **Atualizar dados do produto** (nome, descrição e preço) garantindo que as regras de validação sejam aplicadas.
4. Retornar todos os produtos cadastrados no sistema através de um método que forneça uma lista ou coleção de produtos.

3. Diagrama de Caso de Uso



4. Diagrama de Classes



5. Casos de Teste

5.1. UsuarioService

Caso de Teste 1: Adicionar Usuário Válido

- **Objetivo**: Verificar se o sistema adiciona um usuário com login e senha válidos.
- **Entradas**: Login: usuario1, Senha: senha123.
- **Procedimento**:
    1. Criar um objeto Usuario com os valores fornecidos.

2. Chamar o método adicionar(Usuario).
3. Verificar o retorno do método getAll().

- **Resultado Esperado**: O usuário é adicionado à lista, aumentando seu tamanho.
- **Resultado Obtido**: **Teste aprovado**.

Caso de Teste 2: Adicionar Usuário com Senha Inválida

- **Objetivo**: Impedir a adição de usuários com senhas inválidas.
- **Entradas**:
  - Login: usuario2, Senha: 123.
  - Login: usuario3, Senha: usuario3.
  - Login: usuario4, Senha: (em branco).
  - Login: usuario5, Senha: senha com espacos.
- **Procedimento**:
  - Criar objetos Usuario com cada combinação de login e senha.
  - Chamar adicionar(Usuario) para cada caso.
  - Verificar se o sistema lança IllegalArgumentException.
- **Resultado Esperado**: Exceção lançada em todos os casos.
- **Resultado Obtido**: **Teste aprovado**.

Caso de Teste 3: Adicionar Usuário com Login Duplicado

- **Objetivo**: Garantir a unicidade do login dos usuários.
- **Entradas**:
  - Login: usuario6, Senha: senha123.
  - Login: usuario6, Senha: senha456.
- **Procedimento**:
  - Adicionar o primeiro usuário.
  - Tentar adicionar o segundo usuário com o mesmo login.
- **Resultado Esperado**: Exceção lançada para o segundo usuário.
- **Resultado Obtido**: **Teste aprovado**.

Caso de Teste 4: Remover Usuário

- **Objetivo**: Verificar a remoção de usuários existentes.
- **Entradas**: Login: usuario7, Senha: senha123.
- **Procedimento**:
  1. Adicionar o usuário.
  2. Remover o usuário com o método remover(Usuario).

3. Verificar o tamanho da lista após a remoção.
- **Resultado Esperado**: O usuário é removido, e a lista está vazia.
- **Resultado Obtido**: **Teste aprovado**.

Caso de Teste 5: Atualizar Senha

- **Objetivo**: Atualizar a senha de um usuário existente.
- **Entradas**:
  - Login: usuario8, Senha Antiga: senha123.
  - Nova Senha: novaSenha123.
- **Procedimento**:
  - Adicionar o usuário.
  - Atualizar a senha com o método atualizar(login, novaSenha).
  - Verificar a nova senha.
- **Resultado Esperado**: A senha é atualizada.
- **Resultado Obtido**: **Teste aprovado**.

Caso de Teste 6: Retornar Todos os Usuários

- **Objetivo**: Garantir que o método getAll() retorna a lista de usuários cadastrados.
- **Entradas**: Dois usuários com login e senha válidos.
- **Procedimento**:
  1. Adicionar dois usuários.
  2. Verificar o conteúdo e o tamanho da lista retornada por getAll().
- **Resultado Esperado**: A lista contém os dois usuários adicionados.
- **Resultado Obtido**: **Teste aprovado**.

5.2. ProdutoService

Caso de Teste 1: Adicionar Produto com Sucesso

- **Objetivo**: Adicionar um produto válido ao catálogo.
- **Entradas**:
  - Código: 12345.
  - Nome: Produto Teste.
  - Descrição: Descrição válida.
  - Preço: 10.0.
- **Procedimento**:
  - Criar um objeto Produto.

        - Chamar o método adicionar(Produto).
        - Validar os dados retornados.
- **Resultado Esperado**: Produto é adicionado com dados consistentes.
- **Resultado Obtido**: **Teste aprovado**.

Caso de Teste 2: Adicionar Produto com Código Duplicado

- **Objetivo:** Garantir que o sistema impede a adição de produtos com código duplicado.
- **Entrada:**
    - Código: "12345"
    - Nome: "Produto Teste"
    - Descrição: "Descrição válida"
    - Preço: 10.0
- **Procedimento:**
    - Adicionar o primeiro produto.(Caso de teste 1)
    - Tentar adicionar um segundo produto com o mesmo código.
    - Verificar se o sistema lança uma exceção IllegalArgumentException.
- **Resultado Esperado:** O sistema deve lançar uma exceção com a mensagem "Produto com este código já existe".
- **Resultado Obtido:** Teste passou com sucesso.

Caso de Teste 3: Atualizar Produto com Sucesso

- **Objetivo:** Verificar se o sistema permite atualizar as informações de um produto existente.
- **Entrada:**
    - Código: "12345"
    - Nome: "Produto Atualizado"
    - Descrição: "Nova descrição"
    - Preço: 15.0
- **Procedimento:**
    - Adicionar um produto com os dados iniciais.
    - Atualizar o produto utilizando o código do produto existente.
    - Verificar se as informações foram atualizadas corretamente.
- **Resultado Esperado:** O produto deve ser atualizado com as novas informações.
- **Resultado Obtido:** Teste passou com sucesso.

Caso de Teste 4: Remover Produto com Sucesso

- **Objetivo:** Verificar se o sistema permite remover um produto existente.
- **Entrada:**
  - Código: "12345"
  - Nome: "Produto Teste"
  - Descrição: "Descrição válida"
  - Preço: 10.0
- **Procedimento:**
  - Adicionar um produto.
  - Remover o produto utilizando o código.
  - Verificar se o produto foi removido corretamente.
- **Resultado Esperado:** O produto deve ser removido e não deve mais estar presente no banco de dados.
- **Resultado Obtido:** Teste passou com sucesso.

Caso de Teste 5: Retornar Todos os Produtos

- **Objetivo:** Verificar se o método getAll() retorna todos os produtos armazenados no banco de dados.
- **Entrada:**
  - Produto 1: Código "12345", Nome "Produto Teste 1", Preço 10.0
  - Produto 2: Código "12346", Nome "Produto Teste 2", Preço 20.0
- **Procedimento:**
  - Adicionar dois produtos com códigos diferentes.
  - Chamar o método getAll() e verificar a quantidade de produtos retornados.
- **Resultado Esperado:** A lista retornada deve conter dois produtos.
- **Resultado Obtido:** Teste passou com sucesso.

6. Código da Implementação dos testes.

6.1 ProdutoServiceTest:

```
package com.lojavirtual.gerenciamento_produtos;

import com.lojavirtual.gerenciamento_produtos.model.Produto;
import
com.lojavirtual.gerenciamento_produtos.repository.ProdutoRepository;
import com.lojavirtual.gerenciamento_produtos.service.ProdutoService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
```

```
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
public class ProdutoServiceTest {

    @Autowired
    private ProdutoService produtoService;

    @Autowired
    private ProdutoRepository produtoRepository;

    @BeforeEach
    public void setUp() {
        produtoRepository.deleteAll();
    }

    @Test
    public void testAdicionarProdutoComSucesso() {
        Produto produto = new Produto("12345", "Produto Teste", "Descrição
válida", 10.0);

        Produto produtoSalvo = produtoService.adicionar(produto);

        assertNotNull(produtoSalvo);
        assertEquals("Produto Teste", produtoSalvo.getNome());
        assertEquals("Descrição válida", produtoSalvo.getDescricao());
        assertEquals(10.0, produtoSalvo.getPreco());
    }

    @Test
    public void testAdicionarProdutoComCodigoDuplicado() {
        Produto produto = new Produto("12345", "Produto Teste", "Descrição
válida", 10.0);
        produtoService.adicionar(produto);

        IllegalArgumentException exception =
assertThrows(IllegalArgumentException.class,
                () -> produtoService.adicionar(produto));
        assertEquals("Produto com este código já existe",
exception.getMessage());
    }

    @Test
    public void testAtualizarProdutoComSucesso() {
        Produto produto = new Produto("12345", "Produto Teste", "Descrição
válida", 10.0);
        produtoService.adicionar(produto);
```

```
        produtoService.atualizar("12345", "Produto Atualizado", "Nova
descrição", 15.0);

        Produto produtoAtualizado =
produtoRepository.findByCodigo("12345").orElseThrow();
        assertEquals("Produto Atualizado", produtoAtualizado.getNome());
        assertEquals("Nova descrição", produtoAtualizado.getDescricao());
        assertEquals(15.0, produtoAtualizado.getPreco());
    }

    @Test
    public void testRemoverProduto() {
        Produto produto = new Produto("12345", "Produto Teste", "Descrição
válida", 10.0);
        produtoService.adicionar(produto);

        produtoService.remover("12345");

        assertFalse(produtoRepository.findByCodigo("12345").isPresent());
    }

    @Test
    public void testGetAllProdutos() {
        Produto produto1 = new Produto("12345", "Produto Teste 1", "Descrição
1", 10.0);
        Produto produto2 = new Produto("12346", "Produto Teste 2", "Descrição
2", 20.0);

        produtoService.adicionar(produto1);
        produtoService.adicionar(produto2);

        var produtos = produtoService.getAll();

        assertNotNull(produtos);
        assertEquals(2, produtos.size());
    }
}
```

6.2 UsuarioServiceTest:

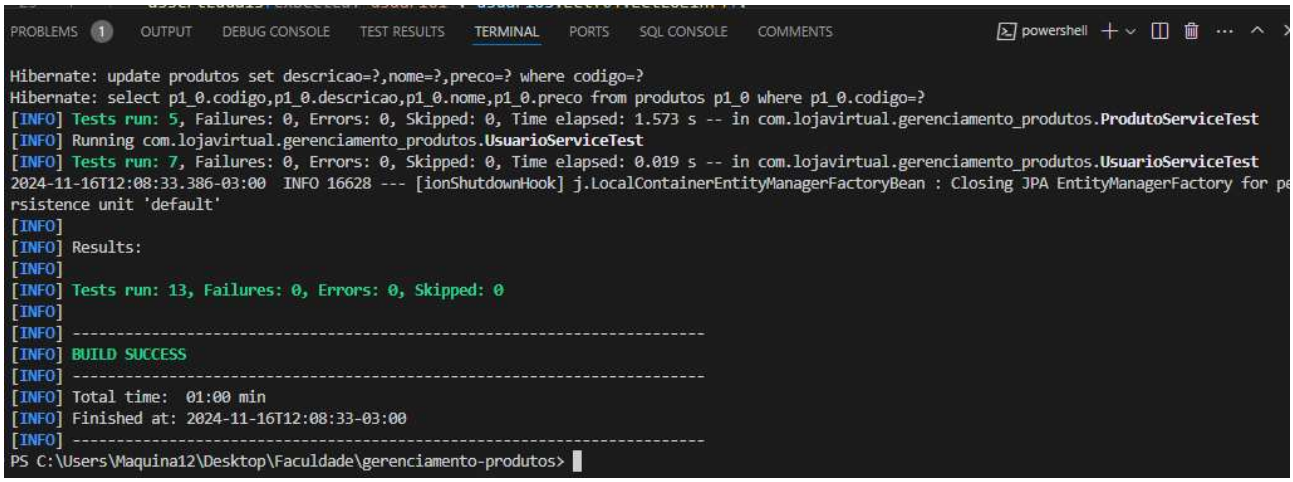
```
package com.lojavirtual.gerenciamento_produtos;

import com.lojavirtual.gerenciamento_produtos.model.Usuario;
import com.lojavirtual.gerenciamento_produtos.service.UsuarioService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.List;
```

```
import static org.junit.jupiter.api.Assertions.*;

class UsuarioServiceTest {

    private UsuarioService usuarioService;

    @BeforeEach
    void setUp() {
        usuarioService = new UsuarioService();
    }

    @Test
    void deveAdicionarUsuarioValido() {
        Usuario usuario = new Usuario("usuario1", "senha123");

        usuarioService.adicionar(usuario);

        List<Usuario> usuarios = usuarioService.getAll();
        assertEquals(1, usuarios.size());
        assertEquals("usuario1", usuarios.get(0).getLogin());
    }

    @Test
    void naoDeveAdicionarUsuarioComSenhaInvalida() {
        Usuario usuarioComSenhaCurta = new Usuario("usuario2", "123");
        Usuario usuarioComSenhaIgualAoLogin = new Usuario("usuario3",
"usuario3");
        Usuario usuarioComSenhaVazia = new Usuario("usuario4", " ");
        Usuario usuarioComEspacos = new Usuario("usuario5", "senha com
espacos");

        assertThrows(IllegalArgumentException.class, () ->
usuarioService.adicionar(usuarioComSenhaCurta));
        assertThrows(IllegalArgumentException.class, () ->
usuarioService.adicionar(usuarioComSenhaIgualAoLogin));
        assertThrows(IllegalArgumentException.class, () ->
usuarioService.adicionar(usuarioComSenhaVazia));
        assertThrows(IllegalArgumentException.class, () ->
usuarioService.adicionar(usuarioComEspacos));
    }

    @Test
    void naoDeveAdicionarUsuarioComLoginDuplicado() {
        Usuario usuario1 = new Usuario("usuario6", "senha123");
        Usuario usuario2 = new Usuario("usuario6", "senha456");

        usuarioService.adicionar(usuario1);
```

```
        assertThrows(IllegalArgumentException.class, () ->
usuarioService.adicionar(usuario2));
    }

    @Test
    void deveRemoverUsuario() {
        Usuario usuario = new Usuario("usuario7", "senha123");

        usuarioService.adicionar(usuario);
        usuarioService.remover(usuario);

        List<Usuario> usuarios = usuarioService.getAll();
        assertTrue(usuarios.isEmpty());
    }

    @Test
    void deveAtualizarSenhaDoUsuario() {
        Usuario usuario = new Usuario("usuario8", "senha123");

        usuarioService.adicionar(usuario);
        usuarioService.atualizar("usuario8", "novaSenha123");

        List<Usuario> usuarios = usuarioService.getAll();
        assertEquals("novaSenha123", usuarios.get(0).getSenha());
    }

    @Test
    void deveLancarExcecaoSeUsuarioNaoExistirParaAtualizar() {
        assertThrows(IllegalArgumentException.class, () ->
usuarioService.atualizar("usuario9", "novaSenha"));
    }

    @Test
    void deveRetornarListaDeUsuarios() {
        Usuario usuario1 = new Usuario("usuario10", "senha123");
        Usuario usuario2 = new Usuario("usuario11", "senha456");

        usuarioService.adicionar(usuario1);
        usuarioService.adicionar(usuario2);

        List<Usuario> usuarios = usuarioService.getAll();
        assertEquals(2, usuarios.size());
    }
}
```

7. Execução dos Testes e Ferramentas Utilizadas

- **Editor:** Visual Studio Code (VS Code).
- **Framework de Teste:** JUnit 5.
- **Ambiente: Projeto Spring Boot configurado para execução local.**



8. Conclusão

Os testes realizados no sistema de gerenciamento de usuários e produtos foram fundamentais para validar a implementação das funcionalidades e garantir que o sistema atenda aos requisitos estabelecidos. Através dos casos de teste, foi possível assegurar que as operações de adição, remoção, atualização e listagem tanto de usuários quanto de produtos estão funcionando corretamente, respeitando as regras de validação e as exceções adequadas quando necessário.

Em relação aos usuários, os testes garantiram que:

- Usuários com senhas válidas são adicionados corretamente ao sistema.
- O sistema impede a adição de usuários com senhas inválidas ou duplicidade de login.
- A remoção e a atualização de senhas funcionam conforme esperado.
- A integridade da lista de usuários é mantida com a correta adição e remoção de registros.

No que diz respeito aos produtos, os testes confirmaram que:

- Produtos com dados válidos são corretamente adicionados.
- O sistema impede a adição de produtos com códigos duplicados, garantindo a unicidade.
- A atualização de informações de produtos é realizada com sucesso, mantendo as validações em vigor.
- Produtos podem ser removidos corretamente do sistema.
- A listagem de todos os produtos cadastrados é precisa e consistente.