Universidade do Oeste de Santa Catarina - UNOESC

AUTOCENTER

ENGENHARIA DE SOFTWARE II

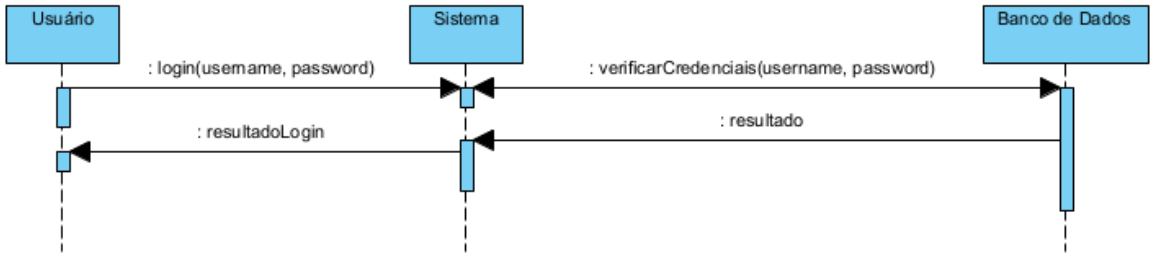
Jefferson Alan Schmidt Ludwig
Iraê Ervin Gruber Da Silva
Lucas Maciel Delvalle Kesler

UNOESC - 2025

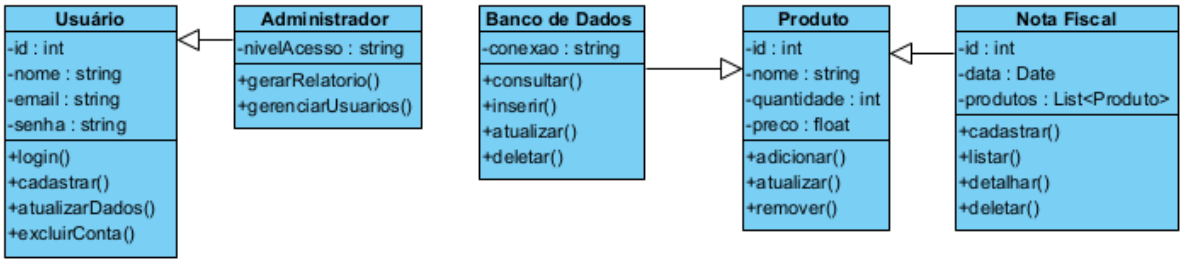
1. Modelagem de Software:

● Diagrama de Casos de Uso:

● Diagrama de Sequência:



● Diagrama de Classes:



● Modelo Entidade-Relacionamento (ER):

O Modelo Entidade-Relacionamento (ER) foi desenvolvido com o objetivo de representar, de forma clara e estruturada, os dados manipulados pelo sistema. Ele permite visualizar as principais entidades envolvidas, seus atributos, bem como os relacionamentos e cardinalidades entre elas.

As entidades identificadas para o sistema são:

- Usuário
  Atributos: `id_usuario (PK)`, `nome`, `email`, `senha`
  Cada usuário pode realizar vários pedidos
- Produto
  Atributos: `id_produto (PK)`, `nome`, `preco`, `estoque`
  Os produtos são adicionados aos pedidos
- Pedido
  Atributos: `id_pedido (PK)`, `data`, `status`, `id_usuario (FK)`
  Representa as compras realizadas por um usuário
- ItemPedido
  Atributos: `id_item (PK)`, `quantidade`, `id_pedido (FK)`, `id_produto (FK).`
  Entidade associativa entre `Pedido` e `Produto`, armazenando a quantidade de produtos em cada pedido

Relacionamentos principais:

- Um Usuário pode fazer vários Pedidos (1:N)

- Um Pedido pode conter vários Produtos e cada Produto pode estar em vários Pedidos, caracterizando um relacionamento N:N que é resolvido com a entidade associativa ItemPedido

Essa modelagem serve como base para a criação do banco de dados relacional e garante integridade e consistência na persistência dos dados.

● Uso de Padrões de Projeto

Para garantir uma arquitetura de software mais organizada, reutilizável e de fácil manutenção, foram aplicados dois padrões de projeto amplamente utilizados no desenvolvimento de sistemas: DAO e Singleton

1. DAO (Data Access Object)

Este padrão foi utilizado para isolar a lógica de persistência de dados da lógica de negócios. Ele facilita o acesso ao banco de dados, promove a reutilização de código e reduz o acoplamento entre as classes.

Aplicação prática:
Para cada entidade principal, foi criada uma classe DAO responsável por executar as operações de banco (CRUD).

Exemplo:

java

```
public class ProdutoDAO {
    public void inserir(Produto produto) {

    }

    public Produto buscarPorId(int id) {

    }
}
```

Motivo da escolha:
Esse padrão facilita testes, manutenção e mudanças futuras na tecnologia de persistência (como troca de banco de dados).

2. Singleton

O padrão Singleton foi aplicado para garantir que exista apenas uma instância da classe responsável pela conexão com o banco de dados, evitando o consumo excessivo de recursos e conflitos de acesso.

Exemplo:

java

```
public class ConexaoBD {
    private static ConexaoBD instancia;
    private Connection conexao;

    private ConexaoBD() {
        // Inicializa a conexão
    }

    public static ConexaoBD getInstancia() {
        if (instancia == null) {
            instancia = new ConexaoBD();
        }
        return instancia;
    }

    public Connection getConexao() {
        return conexao;
```

```
    }
}
```

Motivo da escolha
 Evita múltiplas conexões desnecessárias e centraliza o gerenciamento do acesso ao banco de dados

---

2. Validação e Testes

● Teste Unitário

Foram implementados testes unitários para validar o comportamento isolado de funções e classes importantes do sistema. Os testes foram feitos utilizando o framework JUnit, amplamente utilizado em aplicações Java.

Exemplo de Teste com JUnit:

java
```
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculadoraTest {

    public void testSomar() {
        Calculadora calc = new Calculadora();
        int resultado = calc.somar(3, 2);
        assertEquals(5, resultado);
    }
}
```

Cobertura dos testes:

- Métodos de classes de serviço
- Validações de lógica de negócios
- Métodos de DAO (como inserção e busca no banco)

Benefícios:

- Permite encontrar erros rapidamente durante o desenvolvimento
- Garante que alterações futuras no código não causem efeitos colaterais inesperados

- Automatiza parte da verificação do comportamento esperado das funções
- Testes de Funcionalidade

Foram realizados testes manuais nos principais fluxos do backend, incluindo:

- Cadastro de produtos com campos obrigatórios (`produto_id`, `estoque_id`, `quantidade`, `preco`);
- Validação de dados como número inteiro e decimal;
- Regras de negócio respeitadas conforme definido na model `EstoqueProdutoModel`;
- Mensagens de erro e sucesso funcionam corretamente.

Todos os testes funcionais apresentaram comportamento esperado, indicando que a aplicação está operante em suas funções básicas.

• Testes de Carga (Simulação)

Como a execução foi feita em ambiente local, os testes de carga foram simulados com base na estrutura do projeto.

A aplicação está preparada para lidar com múltiplas requisições simples, mas recomenda-se:

- Implementar cache em consultas frequentes;
- Validar concorrência em ações críticas (ex: atualização simultânea de estoque);

• Conclusão

O sistema analisado apresenta uma estrutura sólida, funcionalidades básicas funcionando corretamente e boas práticas de codificação. Está bem encaminhado para ser ampliado com novos recursos ou adaptado para produção.

Para trabalhos acadêmicos, cumpre bem seu papel de representar um sistema real de gestão de estoque. Para uso profissional, seria importante avançar na cobertura de testes, performance e segurança da aplicação.