Lucas Antonio Ferreira Neto

Descrição do Sistema

Resumo do Projeto

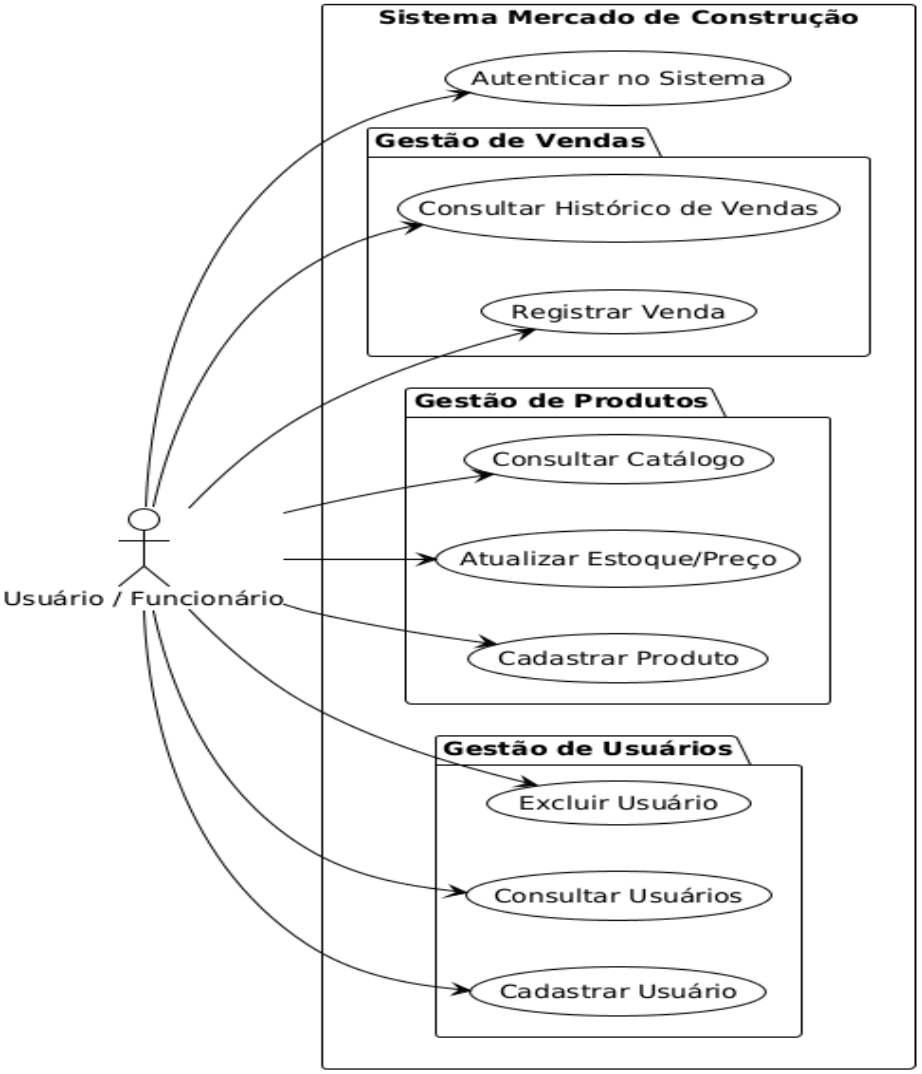
Este sistema é uma aplicação em Java desenvolvida para gerenciar o dia a dia de um mercado de construção civil. O objetivo principal é facilitar o controle da loja, permitindo cadastrar **produtos**, gerenciar **usuários** e registrar **vendas** através de uma interface gráfica amigável.

Por trás das telas, o projeto foi estruturado na arquitetura **MVC** (Model-View-Controller). Isso significa que a parte visual (telas), as regras de negócio (controladores) e os dados (modelo) estão bem separados, o que deixa o sistema mais organizado.

Para lidar com o banco de dados, utilizei o padrão **DAO**, garantindo que as informações sejam salvas e recuperadas de forma segura e eficiente. Além disso, apliquei boas práticas de programação, como os padrões **Singleton** e **Factory**, para otimizar as conexões e tornar o código mais fácil de manter ou evoluir no futuro.

CIM — Modelo Computacional Independente

Diagrama de Casos de Uso



Descrição dos Casos de Uso

O sistema Mercado de Construção permite que o **Usuário/Funcionário** realize atividades de autenticação, vendas, gestão de produtos e gestão de usuários.

- **Autenticar no Sistema:** O usuário informa login e senha para acessar o sistema.

- **Consultar Histórico de Vendas:** O usuário visualiza todas as vendas já realizadas.
- **Registrar Venda:** Permite inserir e confirmar os dados de uma nova venda.
- **Consultar Catálogo:** Exibe a lista de produtos disponíveis.
- **Atualizar Estoque/Preço:** O usuário altera quantidade ou preço de produtos cadastrados.
- **Cadastrar Produto:** Adiciona um novo produto ao sistema.
- **Excluir Usuário:** Remove um usuário do cadastro.
- **Consultar Usuários:** Lista todos os usuários registrados.
- **Cadastrar Usuário:** Insere um novo usuário/funcionário no sistema.

PIM — Modelo Independente de Plataforma

Diagrama de Classes UML

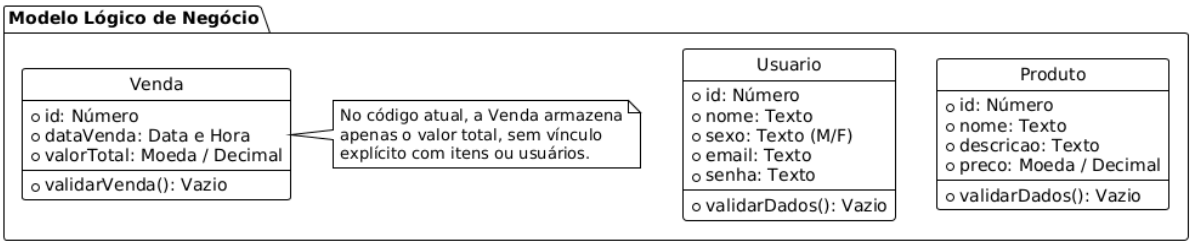
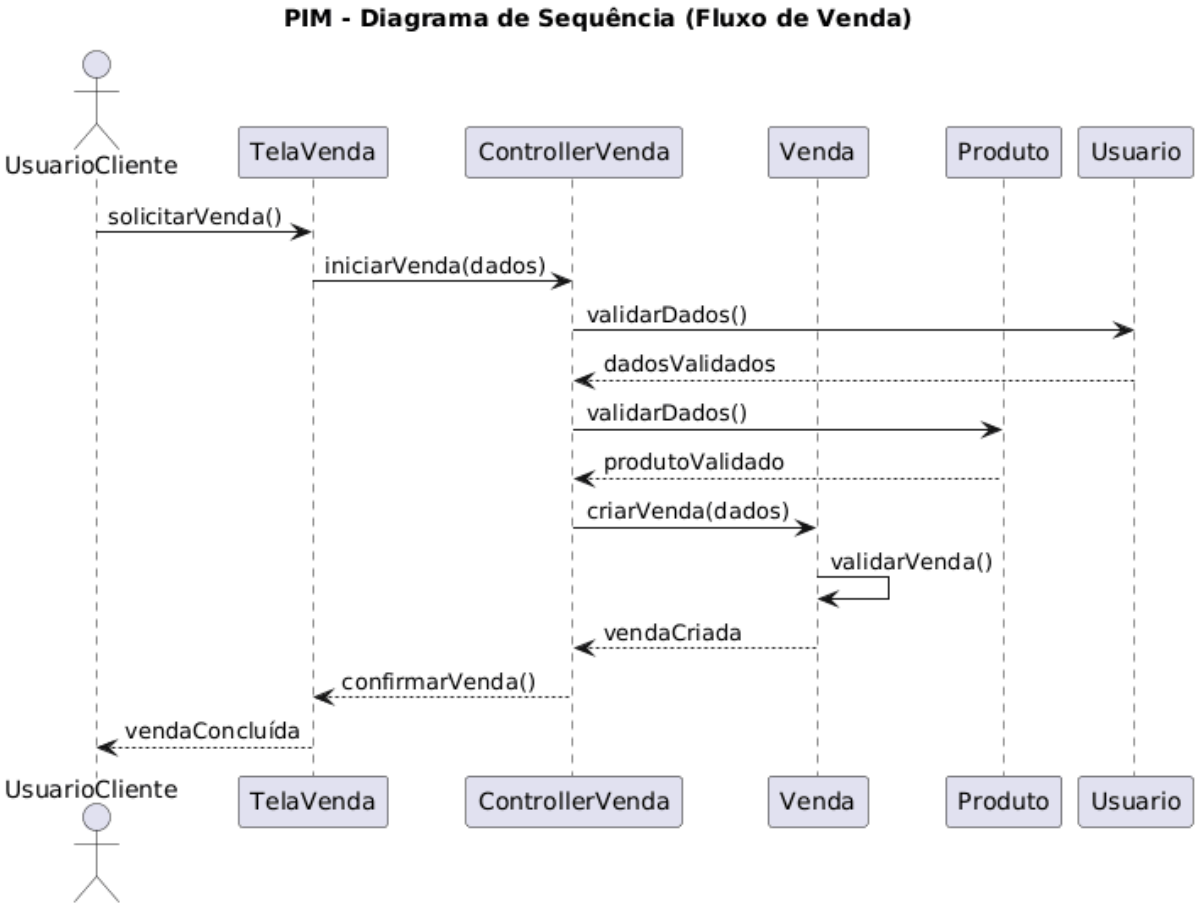
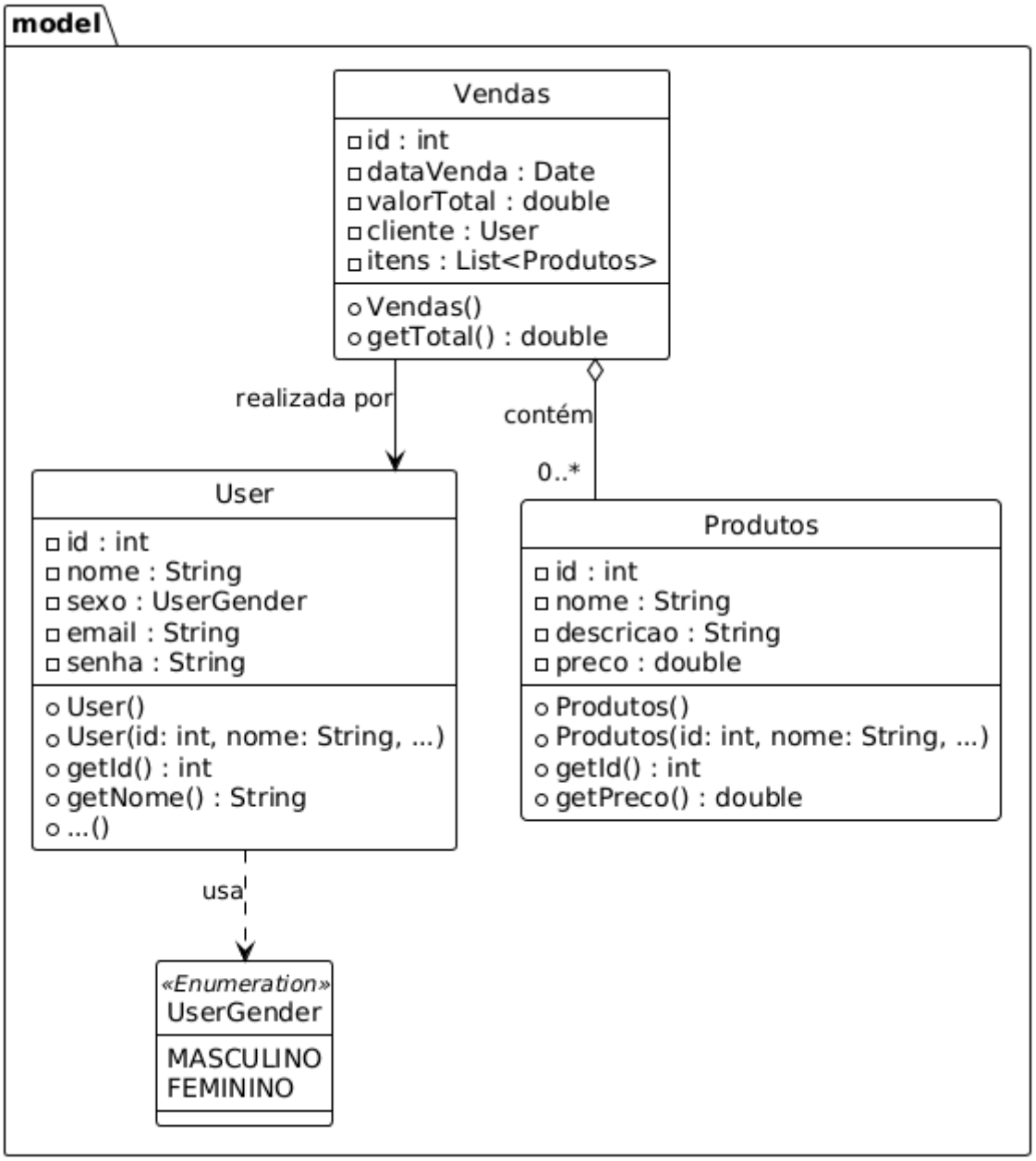


Diagrama de Sequência



PSM — Modelo Específico de Plataforma (Java)



O PSM apresenta a implementação das classes Vendas, User, Produtos e o enum UserGender, definidas com tipos e estruturas específicas da linguagem Java.

A classe Vendas possui atributos como *id*, *dataVenda*, *valorTotal*, referência para o *User* que realizou a venda e uma lista de *Produtos*. Inclui também o método getTotal() responsável por calcular o valor total com base nos itens.

A classe User contém informações do usuário (id, nome, sexo, email, senha) e utiliza o enum UserGender para representar o gênero de forma tipada. Inclui construtores e métodos de acesso.

A classe Produtos armazena dados de cada produto (id, nome, descrição e preço), com construtores e métodos getters.

O modelo especifica ainda os relacionamentos:

- Vendas → contém vários Produtos (0..*).
- Vendas → é realizada por um User.
- User → utiliza o enum UserGender.

Esse PSM já define a estrutura pronta para implementação em Java, incluindo tipos concretos, listas, métodos e relacionamentos.

Código Gerados:

```java
public class ProdutoModel {

    private int id;

    private String nome;

    private double preco;



    public int getId() {
```

```
        return this.id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return this.nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public double getPreco() {
        return this.preco;
    }

    public void setPreco(double preco) {
        this.preco = preco;
    }
}

import java.util.List;
```

```java
public class ProdutoDAO {

    public void salvar(ProdutoModel produto) {

    }


    public List listar() {


        return null;

    }


    public void atualizar(ProdutoModel produto) {

    }


    public void excluir(int id) {

    }

}
```

Vantagens da abordagem orientada por modelos

Percebi que ela ajuda muito a visualizar e entender o sistema antes de codar. Tudo fica mais organizado e claro, e até agiliza porque parte do código já nasce pronta.

Limitações ao gerar código automaticamente

O código gerado nem sempre fica do jeito ideal. Muitas vezes eu precisei ajustar manualmente, porque a ferramenta não consegue representar todo o comportamento que o sistema realmente precisa.

Projetos onde a MDA é mais vantajosa

Acho que faz mais sentido em sistemas maiores, onde existe muita estrutura e a padronização é importante. Nesses casos, o modelo realmente facilita a manutenção e o desenvolvimento.

Análise Crítica

O projeto apresenta uma base sólida graças à arquitetura **MVC** e ao uso de **DAOs**, o que mantém o código organizado e facilita futuras manutenções. A interface em **Java Swing** atende bem aos requisitos funcionais, embora já exista espaço para evoluir visualmente ou migrar para uma solução web mais moderna.

Como pontos de melhoria, eu identifiquei que as regras de negócio poderiam ser centralizadas em uma camada de Serviço (para evitar que fiquem divididas entre Controllers e DAOs) e que o tratamento de erros pode ser aprofundado. No geral, o sistema é consistente, funcional e aplica corretamente as boas práticas de desenvolvimento.