Sistema de E-commerce "Mugiwara Store"

Documento de Regras de Negócio: Sistema de E-commerce "Mugiwara Store"

1. Visão Geral do Sistema

Este documento estabelece as regras de negócio para o desenvolvimento do sistema "Mugiwara Store", uma plataforma de e-commerce no formato de website dedicada à comercialização de produtos temáticos da obra One Piece. O objetivo do sistema é fornecer um catálogo online de produtos, gerenciar o estoque, processar vendas, manter um registro de clientes e suas transações, e oferecer ferramentas de gerenciamento para os funcionários da loja. Este documento servirá como a principal referência para as fases de modelagem de dados e desenvolvimento da aplicação.

2. Regras de Negócio do Sistema Completo (Partes 1 e 2)

As regras a seguir descrevem a funcionalidade completa do sistema conforme idealizado para a entrega final do projeto.

2.1. Regras Gerais e de Acesso

- **Acesso Público:** O sistema deverá ser de acesso público, permitindo que qualquer visitante navegue livremente pelo catálogo de produtos para consultar itens, detalhes e preços, sem a necessidade de um cadastro prévio ou de efetuar login na plataforma.
- **Autenticação:** O sistema implementará um mecanismo de login e cadastro. Funcionalidades como realizar uma compra ou acessar o painel de gerenciamento serão restritas a usuários autenticados.
- **Perfis de Acesso:** O sistema contará com dois perfis principais de usuários:
    - **Cliente:** Após o cadastro e login, pode manter seus dados, realizar compras, verificar seu histórico de pedidos e consultar seus dados cadastrais.

    - **Funcionário (Vendedor):** Responsável por operar o sistema. Terá acesso a um painel de gerenciamento para executar todas as operações de CRUD de produtos, visualizar clientes e efetivar as vendas.

2.2. Regras de Produto e Gerenciamento de Estoque

- **Atributos Obrigatórios:** Cada produto deve possuir um identificador único, nome, descrição, preço, quantidade em estoque e uma categoria.
- **Validação de Preço:** O preço de qualquer produto deve ser, obrigatoriamente, um valor monetário superior a zero.
- **Validação de Estoque:** A quantidade em estoque de um produto nunca poderá ser um valor negativo. A lógica de negócio deve impedir que uma compra seja efetivada se a quantidade solicitada for superior à disponível.
- **Busca e Filtros (Público):** A plataforma deve oferecer aos usuários funcionalidades de busca para encontrar produtos por nome, faixa de preço e categoria.
- **Atributo Específico:** O sistema deve suportar um atributo que identifique se um produto foi fabricado na localidade de "Mari".
- **Filtro de Estoque (Funcionário):** Para usuários com perfil de funcionário, o sistema deve prover uma funcionalidade de filtro especial que liste todos os produtos cujo estoque seja inferior a 5 unidades.

2.3. Regras de Cliente

- **Cadastro Obrigatório para Compra:** A realização de uma compra está condicionada à identificação do cliente por meio de cadastro e login.
- **Atributos do Cliente:** O cadastro de um cliente deve conter um identificador único, nome completo, um endereço de e-mail válido, senha, endereço de entrega e um telefone de contato.
- **Política de Descontos:** Um cliente terá direito a um desconto se atender a pelo menos um dos seguintes critérios: ser torcedor do Flamengo, ser espectador da obra One Piece ou ser natural da cidade de Sousa.

2.4. Regras de Venda e Pedido

- **Histórico de Pedidos:** Um mesmo cliente pode realizar múltiplos pedidos, que ficarão registrados em seu histórico.
- **Associação com Vendedor:** Toda compra deve ser finalizada por um vendedor cadastrado no sistema, que será associado ao registro do pedido.
- **Composição do Pedido:** Um pedido é composto por, no mínimo, um item de produto, podendo conter vários itens distintos.
- **Forma de Pagamento:** Todo pedido deve ter uma forma de pagamento associada (ex: Cartão de Crédito, PIX, Boleto).
- **Status de Pagamento:** Para formas de pagamento que exigem confirmação (Cartão, Boleto), o pedido deve ter um status que indique o andamento (ex: "Aguardando Pagamento", "Pagamento Aprovado").

2.5. Regras de Relatórios

- **Relatório de Vendas por Vendedor:** O sistema deve ser capaz de emitir uma listagem mensal que detalhe o volume de vendas consolidado para cada vendedor.
- **Relatório de Estoque:** O sistema deve gerar um relatório de resumo que exiba a quantidade total de tipos de produto cadastrados e o valor total do inventário.

3. Modelagem de Dados do Sistema

A modelagem de dados funciona como a planta do banco de dados, definindo a estrutura, as regras e os relacionamentos que os dados devem seguir para garantir a integridade, consistência e eficiência do sistema.

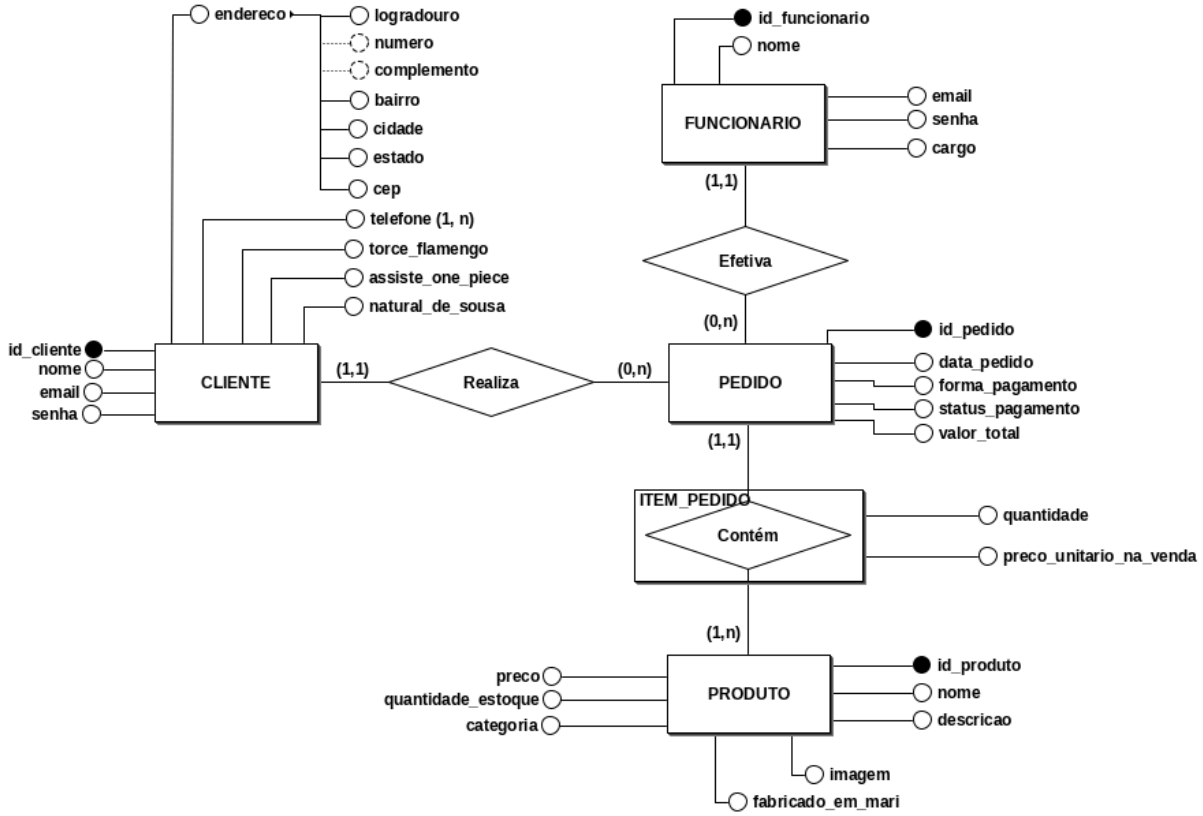
O desenvolvimento da modelagem de dados será realizado em três fases:

1. **Modelo Conceitual:** Representação de alto nível das principais entidades do sistema, focada nas regras de negócio.
2. **Modelo Lógico:** Tradução do modelo conceitual para uma estrutura mais próxima do SGBD (Sistema de Gerenciamento de Banco de Dados) a ser utilizado, como o modelo relacional.
3. **Modelo Físico:** A implementação final do modelo lógico no SGBD escolhido (PostgreSQL), definindo tipos de dados, índices e outras configurações de armazenamento.

3.1. Modelo Conceitual (Diagrama Entidade-Relacionamento - DER)

O primeiro passo na modelagem de dados é a criação do Modelo Conceitual. Esta fase foca em traduzir as regras de negócio que definimos em uma representação gráfica e de alto nível, utilizando o Diagrama Entidade-Relacionamento (DER). O objetivo é criar um mapa claro das principais "coisas" (entidades) sobre as quais o sistema precisa armazenar informações e como elas se relacionam, sem se preocupar com detalhes técnicos de implementação do banco de dados. Para a elaboração do diagrama, foi utilizada a ferramenta **brModelo**, que permite a criação de

modelos conceituais seguindo as notações acadêmicas. O diagrama a seguir representa o sistema de e-commerce "Mugiwara Store" em sua totalidade.



Descrição das Entidades

PRODUTO: Entidade forte que representa os itens comercializados na loja. É a base do catálogo de vendas e possui atributos como id_produto (identificador), nome, descricao, preco, quantidade_estoque e categoria.

CLIENTE: Entidade forte que representa os usuários compradores do sistema. Além de informações de identificação e contato como id_cliente (identificador), nome e email, esta entidade possui:

- Um atributo composto endereco, que detalha o local de entrega em cep, logradouro, numero, bairro, cidade, estado e um complemento opcional.

- Um atributo multivalorado telefone, indicando que um cliente pode cadastrar um ou mais números de contato.

- Atributos booleanos (torce_flamengo, assiste_one_piece, natural_de_sousa) para suportar a regra de negócio de descontos promocionais.

FUNCIONARIO: Entidade forte que representa um membro da equipe da loja, responsável por gerenciar o catálogo e efetivar os pedidos. Possui atributos de identificação e acesso, como id_funcionario (identificador), nome, email e cargo.

PEDIDO: Entidade forte que representa a transação de compra. É a entidade central do processo de venda e possui atributos como id_pedido (identificador), data_pedido, forma_pagamento, status_pagamento e valor_total.

ITEM_PEDIDO: É uma Entidade Associativa (ou Entidade Fraca) que surge do relacionamento N:M (muitos-para-muitos) entre PEDIDO e PRODUTO. Sua existência e identificação dependem das duas entidades que ela conecta. Ela é fundamental para detalhar o conteúdo de cada compra, armazenando a quantidade de um produto específico e o preco_unitario_na_venda no momento da transação.

Descrição dos Relacionamentos e Cardinalidades

As cardinalidades, no formato (mínima, máxima), definem as regras de negócio que governam as interações entre as entidades.

Realiza (entre CLIENTE e PEDIDO):

- Um CLIENTE pode realizar (0,n) (nenhum ou vários) PEDIDOS.
- Um PEDIDO deve ser realizado por (1,1) (um e somente um) CLIENTE.

Efetiva (entre FUNCIONARIO e PEDIDO):

- Um FUNCIONARIO pode efetivar (0,n) (nenhum ou vários) PEDIDOS.
- Um PEDIDO deve ser efetivado por (1,1) (um e somente um) FUNCIONARIO.

Contém (entre PEDIDO, PRODUTO e ITEM_PEDIDO):

- Um PEDIDO deve conter (1,n) (um ou vários) ITENS_PEDIDO. Isso garante que não existam pedidos vazios.
- Um PRODUTO pode estar contido em (0,n) (nenhum ou vários) ITENS_PEDIDO, permitindo que um produto nunca tenha sido vendido ou seja vendido várias vezes.
- Um ITEM_PEDIDO está ligado a (1,1) PEDIDO e a (1,1) PRODUTO.

Este modelo conceitual servirá como base para a criação do modelo lógico relacional, onde as regras de normalização serão aplicadas para garantir a integridade e eficiência do banco de dados.

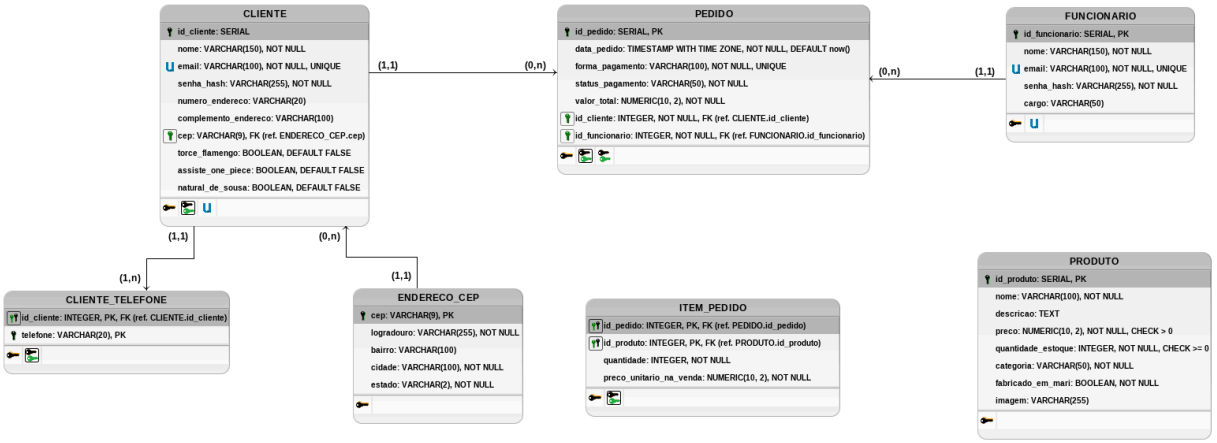
3.2. Ferramenta de Modelagem: brModelo

Para a elaboração dos diagramas, foi utilizada a ferramenta brModelo. Trata-se de um software educacional desenvolvido no Brasil, amplamente adotado no ensino de banco de dados por ser didático e focado nos principais conceitos da modelagem. Ele permite a criação de modelos conceituais (DER), lógicos e a geração de scripts SQL a partir dos diagramas, facilitando a transição entre as fases do projeto de banco de dados. Sua interface simples e a aderência à notação mais comum no meio acadêmico a tornam uma escolha ideal para este projeto.

3.3. Modelo Lógico (Relacional)

O Modelo Lógico traduz o Modelo Conceitual (DER) para o formalismo do Modelo Relacional, que é a base para a implementação em SGBDs como o PostgreSQL. Nesta fase, as entidades são mapeadas para relações (tabelas) e seus atributos para colunas, com a definição explícita de chaves primárias (PK), chaves estrangeiras (FK) e restrições de integridade.

O processo de mapeamento seguiu as regras apresentadas na disciplina e aplicou os conceitos de normalização para garantir que o esquema final estivesse na Terceira Forma Normal (3FN), eliminando redundâncias e anomalias de dados.



Justificativa do Mapeamento e Normalização

A transição do modelo conceitual para o lógico envolveu as seguintes decisões de design:

Mapeamento de Entidades Fortes: As entidades PRODUTO, FUNCIONARIO e PEDIDO foram mapeadas diretamente para suas respectivas tabelas, com seus atributos simples se tornando as colunas.

Mapeamento de Relacionamentos 1:N: Os relacionamentos "Realiza" (Cliente-Pedido) e "Efetiva" (Funcionario-Pedido) foram implementados adicionando a chave primária do lado "1" da relação como uma chave

estrangeira (FK) na tabela do lado "N". Por isso, a tabela PEDIDO contém id_cliente (FK) e id_funcionario (FK).

Normalização da Entidade CLIENTE: A entidade CLIENTE do modelo conceitual continha atributos que violavam as formas normais, exigindo decomposição:

- **Atributo Multivalorado telefone:** Para satisfazer a Primeira Forma Normal (1FN), que não permite atributos com múltiplos valores, o atributo telefone foi extraído para uma nova tabela, CLIENTE_TELEFONE.
- **Atributo Composto endereco:** Para satisfazer a Terceira Forma Normal (3FN), o atributo endereco foi decomposto. Foi identificada uma dependência transitiva, onde atributos como logradouro e cidade dependiam do cep, e não diretamente da chave id_cliente. Para resolver isso, foi criada a tabela ENDERECO_CEP, que armazena as informações dependentes do CEP, eliminando a redundância.
- **Mapeamento de Entidade Associativa (N:M):** O relacionamento "Contém" (N:M) entre PEDIDO e PRODUTO foi mapeado para a tabela ITEM_PEDIDO. A chave primária desta tabela é uma chave composta pelas chaves estrangeiras id_pedido e id_produto, garantindo a unicidade de um produto dentro de um pedido.

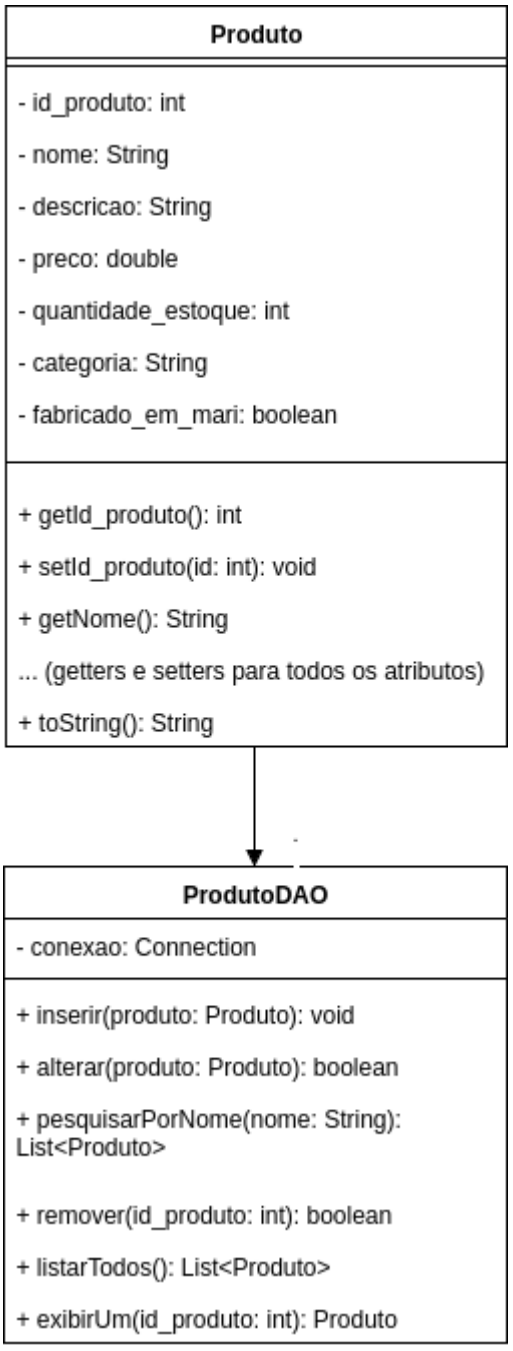
4.4. Modelo Físico (Script SQL)

```sql
-- Tabela PRODUTO (Entidade principal da loja)
CREATE TABLE PRODUTO (
    id_produto SERIAL NOT NULL,
    nome VARCHAR(100) NOT NULL,
    descricao TEXT,
    preco NUMERIC(10,2) NOT NULL,
    quantidade_estoque INTEGER NOT NULL,
    categoria VARCHAR(50) NOT NULL,
    fabricado_em_mari BOOLEAN NOT NULL,
    imagem VARCHAR(255),
    CONSTRAINT pk_produto PRIMARY KEY (id_produto),
    CONSTRAINT ck_produto_preco CHECK (preco > 0),
    CONSTRAINT ck_produto_estoque CHECK (quantidade_estoque >= 0)
);

-- Tabela ENDERECO_CEP (Criada para atender a 3ª Forma Normal)
CREATE TABLE ENDERECO_CEP (
    cep VARCHAR(9) NOT NULL,
    logradouro VARCHAR(255) NOT NULL,
    bairro VARCHAR(100),
    cidade VARCHAR(100) NOT NULL,
    estado VARCHAR(2) NOT NULL,
    CONSTRAINT pk_endereco_cep PRIMARY KEY (cep)
);

-- Tabela CLIENTE
CREATE TABLE CLIENTE (
    id_cliente SERIAL NOT NULL,
    nome VARCHAR(150) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    senha_hash VARCHAR(255) NOT NULL,
    numero_endereco VARCHAR(20),
    complemento_endereco VARCHAR(100),
    cep VARCHAR(9),
    torce_flamengo BOOLEAN DEFAULT FALSE,
    assiste_one_piece BOOLEAN DEFAULT FALSE,
    natural_de_sousa BOOLEAN DEFAULT FALSE,
    CONSTRAINT pk_cliente PRIMARY KEY (id_cliente),
    CONSTRAINT fk_cliente_endereco_cep FOREIGN KEY (cep) REFERENCES
ENDERECO_CEP(cep)
);
```

```sql
-- Tabela CLIENTE_TELEFONE (Criada para atender a 1ª Forma Normal)
CREATE TABLE CLIENTE_TELEFONE (
    id_cliente INTEGER NOT NULL,
    telefone VARCHAR(20) NOT NULL,
    CONSTRAINT pk_cliente_telefone PRIMARY KEY (id_cliente,
telefone),
    CONSTRAINT fk_telefone_cliente FOREIGN KEY (id_cliente)
REFERENCES CLIENTE(id_cliente) ON DELETE CASCADE
);

-- Tabela FUNCIONARIO
CREATE TABLE FUNCIONARIO (
    id_funcionario SERIAL NOT NULL,
    nome VARCHAR(150) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    senha_hash VARCHAR(255) NOT NULL,
    cargo VARCHAR(50),
    CONSTRAINT pk_funcionario PRIMARY KEY (id_funcionario)
);

-- Tabela PEDIDO
CREATE TABLE PEDIDO (
    id_pedido SERIAL NOT NULL,
    data_pedido TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    forma_pagamento VARCHAR(50) NOT NULL,
    status_pagamento VARCHAR(50) NOT NULL,
    valor_total NUMERIC(10, 2) NOT NULL,
    id_cliente INTEGER NOT NULL,
    id_funcionario INTEGER NOT NULL,
    CONSTRAINT pk_pedido PRIMARY KEY (id_pedido),
    CONSTRAINT fk_pedido_cliente FOREIGN KEY (id_cliente)
REFERENCES CLIENTE(id_cliente),
    CONSTRAINT fk_pedido_funcionario FOREIGN KEY (id_funcionario)
REFERENCES FUNCIONARIO(id_funcionario)
);

-- Tabela ITEM_PEDIDO (Entidade Associativa)
CREATE TABLE ITEM_PEDIDO (
    id_pedido INTEGER NOT NULL,
    id_produto INTEGER NOT NULL,
    quantidade INTEGER NOT NULL,
    preco_unitario_na_venda NUMERIC(10, 2) NOT NULL,
```

```
    CONSTRAINT pk_item_pedido PRIMARY KEY (id_pedido, id_produto),
    CONSTRAINT fk_item_pedido_pedido FOREIGN KEY (id_pedido)
REFERENCES PEDIDO(id_pedido) ON DELETE CASCADE,
    CONSTRAINT fk_item_pedido_produto FOREIGN KEY (id_produto)
REFERENCES PRODUTO(id_produto)
);
```

5. Modelagem da Aplicação (Diagrama de Classes UML)



O diagrama UML apresentado ilustra uma parte essencial da arquitetura da aplicação, focando no padrão de projeto DAO (Data Access Object). Este padrão é fundamental para a organização do código, pois separa a lógica de negócio das regras de acesso e persistência de dados. O diagrama detalha a estrutura da entidade Produto e de seu respectivo objeto de acesso a dados, ProdutoDAO.

Classe Produto

A classe Produto é uma classe de modelo (ou Model) que representa a entidade "Produto" no sistema. Ela atua como uma estrutura de dados que encapsula os atributos de um produto comercializado na loja. Seus principais componentes são:

- **Atributos (privados):** Contém os dados que definem um produto, como id_produto, nome, preco, e quantidade_estoque. O encapsulamento (uso de atributos privados) garante que o acesso aos dados seja controlado.
- **Métodos (públicos):** Expõe os métodos de acesso (getters) e modificação (setters) para cada atributo, permitindo que outras partes do sistema interajam com os dados do objeto Produto de forma segura e controlada. Inclui também o método toString() para uma representação textual do objeto, útil para fins de depuração e logs.

Classe ProdutoDAO

A classe ProdutoDAO é responsável por centralizar toda a comunicação com o banco de dados referente à entidade Produto. Ela abstrai a complexidade das operações SQL do resto da aplicação. Suas responsabilidades incluem:

- **Atributo (privado):** Mantém uma referência à conexão com o banco de dados (conexao).
- **Métodos (públicos):** Cada método corresponde a uma operação CRUD (Create, Read, Update, Delete) ou a outras consultas necessárias:
    - inserir(produto: Produto): Recebe um objeto Produto e o persiste no banco de dados.
    - alterar(produto: Produto): Atualiza um registro existente no banco com base nos dados de um objeto Produto.
    - remover(id_produto: int): Remove um produto do banco de dados a partir de seu ID.

- listarTodos(): Retorna uma lista com todos os objetos Produto cadastrados.

- exibirUm(id_produto: int): Retorna um único objeto Produto correspondente ao ID fornecido.

- pesquisarPorNome(nome: String): Busca produtos cujo nome corresponda ao termo de pesquisa.

A seta de dependência partindo de ProdutoDAO para Produto indica que a classe de acesso a dados depende da classe de modelo para executar suas operações, recebendo ou retornando objetos do tipo Produto.

Para manter a clareza e focar no padrão de arquitetura principal, as demais classes do sistema, como Cliente, Pedido, Funcionario e seus respectivos DAOs, não foram representadas. A estrutura dessas classes segue o mesmo princípio de design da classe Produto e seu DAO, tornando sua representação gráfica trivial e repetitiva neste contexto.