Estudo de Caso: WIN Marketplace

Sumário Executivo

O WIN Marketplace é uma plataforma digital inovadora, criada para ser um ecossistema que conecta clientes, lojistas locais e motoristas de entrega em uma única interface. Com foco em fortalecer o comércio regional, o projeto visa digitalizar negócios de pequeno e médio porte, oferecer conveniência aos clientes e criar novas oportunidades de renda para entregadores autônomos. A solução se baseia em uma arquitetura robusta e escalável para garantir desempenho, segurança e disponibilidade.

O Desafio

O comércio local enfrenta uma concorrência crescente de grandes varejistas online, que oferecem conveniência e agilidade que muitas pequenas empresas não conseguem replicar. Ao mesmo tempo, os consumidores modernos demandam uma experiência de compra mais fluida e a entrega de produtos no mesmo dia. Existe uma lacuna no mercado para uma plataforma que não apenas digitalize essas lojas, mas também crie um sistema logístico eficiente e localizado. O desafio, portanto, é construir um marketplace que seja uma alternativa viável aos gigantes do e-commerce, mantendo a essência e o foco na economia regional.

A Solução Proposta

A resposta a esse desafio é o **WIN Marketplace**, uma plataforma multifuncional com três interfaces principais, projetadas para atender as necessidades de cada tipo de usuário:

- **Para Clientes:** Uma experiência de compra intuitiva, com funcionalidades de busca, carrinho de compras, múltiplos métodos de pagamento (PIX e cartão) e, crucialmente, o rastreamento do pedido em tempo real.
- **Para Lojistas:** Uma ferramenta completa para gerenciar o negócio online. O sistema permite o cadastro detalhado de produtos (incluindo variações e imagens), gestão de estoque, processamento de pedidos e acesso a relatórios de vendas e ganhos.
- **Para Motoristas:** Uma solução flexível para obter renda. Os motoristas podem definir sua disponibilidade, aceitar entregas, usar códigos de confirmação para a retirada e a entrega dos pedidos, e visualizar seus ganhos e histórico. A funcionalidade de saque permite a gestão financeira autônoma.

A plataforma é complementada por um painel de administração robusto, que permite o controle de usuários, a gestão de taxas e a monitoria completa de todas as operações.

Arquitetura e Implementação

A solução do WIN Marketplace é concebida com uma arquitetura moderna e escalável, para garantir um desempenho de alto nível e a capacidade de expansão futura.

- **Backend:** Desenvolvido em **Java** com **Spring Boot**, utilizando um banco de dados **PostgreSQL**. A escolha do PostgreSQL é estratégica, pois permite o uso de extensões poderosas como o **PostGIS** para o rastreamento em tempo real da geolocalização dos motoristas e o **pg_trgm** para buscas textuais eficientes nos produtos e lojas.
- **Frontend:** A interface web será construída com **React e Vite**, utilizando a flexibilidade do **Tailwind CSS** para um design responsivo e modular, garantindo uma ótima experiência tanto em computadores quanto em dispositivos móveis.
- **Integrações e Segurança:** O sistema utilizará **APIs RESTful** para a comunicação entre o frontend e o backend. A segurança é uma prioridade, com a implementação de **JWT para autenticação e autorização**, e **criptografia de hash para senhas**. O processamento de pagamentos será feito através da integração com o gateway **BR Fideliza**.

Resultados Esperados

A implementação do WIN Marketplace terá um impacto positivo e tangível em toda a comunidade:

- **Para os Lojistas:** Acesso a um novo canal de vendas, maior visibilidade e um crescimento no alcance de seus negócios, resultando em um aumento de receita e na sua competitividade.
- **Para os Clientes:** Uma experiência de compra mais conveniente, com a agilidade da entrega local e a possibilidade de apoiar negócios de sua própria região.
- **Para os Motoristas:** Uma nova fonte de renda flexível, com total autonomia sobre seus horários e a capacidade de gerenciar seus ganhos de forma simples.
- **Para a Comunidade:** O fortalecimento da economia regional, mantendo o capital circulando dentro da própria localidade e incentivando o desenvolvimento do comércio.

Requisitos do Projeto

Este documento detalha os requisitos funcionais e não funcionais para o desenvolvimento do WIN Marketplace, servindo como guia para as equipes de design, engenharia e testes.

1. Requisitos Funcionais (RF)

Os requisitos funcionais descrevem as funcionalidades específicas que o sistema deve executar para atender aos objetivos de negócio. Eles são organizados por ator principal.

RF.1 - Requisitos do Cliente (Usuário Final)

- **RF.1.1 Cadastro e Autenticação:** O sistema deve permitir que o cliente crie uma conta usando e-mail/senha ou redes sociais, e se autentique de forma segura.
- **RF.1.2 Gerenciamento de Perfil:** O cliente deve poder visualizar e editar seus dados pessoais (nome, telefone, etc.) e endereços de entrega.
- **RF.1.3 Navegação e Busca:** O sistema deve permitir a busca de produtos por nome, categoria ou loja, com filtros por preço, classificação, distância, etc.
- **RF.1.4 Carrinho de Compras:** O cliente deve poder adicionar, remover e ajustar a quantidade de produtos no carrinho.
- **RF.1.5 Finalização de Pedido:** O sistema deve permitir a finalização da compra com a seleção do endereço de entrega, opção de pagamento e visualização do total.
- **RF.1.6 Pagamento:** O cliente deve poder pagar por PIX ou cartão de crédito, e o sistema deve processar a transação com um gateway de pagamento (BR Fideliza).
- **RF.1.7 Acompanhamento de Pedido:** O cliente deve poder visualizar o status em tempo real do seu pedido (em preparo, em transporte, entregue) e a localização do motorista.
- **RF.1.8 Avaliações:** O cliente deve poder avaliar o produto e o motorista após a conclusão da entrega.
- **RF.1.9 Notificações:** O sistema deve enviar notificações (via e-mail, SMS ou push) sobre o status do pedido.

RF.2 - Requisitos do Vendedor (Lojista)

- **RF.2.1 Cadastro e Aprovação:** O lojista deve se cadastrar e fornecer os dados da sua loja para aprovação.
- **RF.2.2 Gerenciamento do Catálogo:** O lojista deve poder cadastrar, editar, ativar, desativar e remover produtos, incluindo informações como preço, estoque, descrição e imagens.
- **RF.2.3 Gestão de Pedidos:** O lojista deve receber notificações de novos pedidos e poder atualizar o status (ex: "em preparo", "pronto para entrega").
- **RF.2.4 Relatórios de Vendas:** O lojista deve ter acesso a relatórios detalhados sobre suas vendas, ganhos e históricos de transações.
- **RF.2.5 Gestão Financeira:** O lojista deve poder solicitar retiradas (saques) de seus ganhos acumulados.

RF.3 - Requisitos do Motorista (Entregador)

- **RF.3.1 Cadastro e Aprovação:** O motorista deve se cadastrar e fornecer os dados de seu veículo e CNH para aprovação.
- **RF.3.2 Status de Disponibilidade:** O motorista deve poder alternar seu status entre "disponível" e "indisponível" para receber chamadas de entrega.
- **RF.3.3 Recebimento de Entregas:** O motorista deve receber solicitações de entrega com informações do pedido, endereço e ganhos estimados.
- **RF.3.4 Gestão de Entrega:** O motorista deve poder atualizar o status da entrega (ex: "pedido retirado", "em rota", "entregue") e usar códigos de confirmação.
- **RF.3.5 Rastreamento de Localização:** O sistema deve coletar a localização do motorista em tempo real para o rastreamento do cliente.

- **RF.3.6 Ganhos e Histórico:** O motorista deve poder visualizar seu total de ganhos e o histórico de entregas concluídas.
- **RF.3.7 Gestão Financeira:** O motorista deve poder solicitar retiradas (saques) de seus ganhos acumulados.

RF.4 - Requisitos do Administrador

- **RF.4.1 Gerenciamento de Usuários:** O administrador deve poder visualizar, aprovar, suspender e remover contas de clientes, lojistas e motoristas.
- **RF.4.2 Gerenciamento de Taxas:** O administrador deve poder definir e ajustar as taxas de comissão cobradas dos lojistas e as taxas de entrega.
- **RF.4.3 Relatórios Gerenciais:** O administrador deve ter acesso a relatórios completos sobre desempenho do marketplace, finanças e métricas operacionais.
- **RF.4.4 Monitoramento:** O administrador deve poder monitorar o status de todos os pedidos e entregas em andamento.

2. Requisitos Não Funcionais (RNF)

Os requisitos não funcionais descrevem as qualidades do sistema, como ele deve funcionar, em vez de o que ele deve fazer.

RNF.1 - Desempenho

- **RNF.1.1 Tempo de Resposta:** A maioria das requisições (95%) deve ter um tempo de resposta inferior a 2 segundos.
- **RNF.1.2 Processamento de Pedidos:** O sistema deve ser capaz de processar 1.000 pedidos por minuto em horários de pico.
- **RNF.1.3 Latência de Rastreamento:** O rastreamento de motoristas deve ter uma latência máxima de 5 segundos.

RNF.2 - Escalabilidade

- **RNF.2.1 Crescimento de Dados:** A arquitetura deve suportar o crescimento da base de dados sem degradação do desempenho, acomodando milhões de produtos e pedidos.
- **RNF.2.2 Picos de Demanda:** O sistema deve escalar horizontalmente para lidar com picos de tráfego durante promoções e feriados.

RNF.3 - Segurança

- **RNF.3.1 Autenticação e Autorização:** O sistema deve usar tokens JWT para garantir que apenas usuários autorizados acessem os recursos.
- **RNF.3.2 Criptografia:** As senhas dos usuários devem ser armazenadas de forma criptografada (hash). As transações de pagamento devem ser protegidas por HTTPS e criptografia de ponta a ponta.
- **RNF.3.3 Proteção contra Ameaças:** O sistema deve ser resistente a ataques de injeção de SQL, XSS, CSRF e outras ameaças web comuns.

- **RNF.3.4 Conformidade Legal:** O sistema deve estar em conformidade com as leis de proteção de dados locais (ex: LGPD no Brasil).

RNF.4 - Usabilidade

- **RNF.4.1 Interface Intuitiva:** As interfaces (web e mobile) devem ser claras, responsivas e fáceis de usar para todos os atores.
- **RNF.4.2 Feedback Visual:** O sistema deve fornecer feedback visual imediato para ações do usuário (botões de carregamento, mensagens de sucesso, etc.).

RNF.5 - Disponibilidade

- **RNF.5.1 Tempo de Atividade:** O sistema deve ter uma disponibilidade mínima de 99.5% por mês, excluindo janelas de manutenção planejadas.

RNF.6 - Manutenibilidade

- **RNF.6.1 Modularidade:** O código deve ser modular e seguir as melhores práticas para facilitar a manutenção e a adição de novas funcionalidades.
- **RNF.6.2 Documentação:** O código deve ser bem comentado e as APIs devem ter documentação clara.

Casos de Uso

Este documento detalha as interações principais de cada ator com o sistema WIN Marketplace, servindo como base para o desenvolvimento das funcionalidades.

Atores Principais

- **Cliente** → realiza compras, pagamentos, acompanha pedidos e avaliações.
- **Vendedor (Lojista)** → gerencia produtos e pedidos, acompanha vendas e ganhos.
- **Motorista (Entregador)** → gerencia sua disponibilidade, aceita entregas e acompanha seus ganhos.
- **Administrador** → gerencia todos os usuários, monitora o sistema e ajusta configurações.
- **Sistema de Pagamento (BR Fideliza)** → processa transações e valida pagamentos.

Casos de Uso por Ator

👤 Cliente

- **Cadastrar-se e Autenticar-se:** O cliente cria sua conta na plataforma e faz login.
- **Gerenciar Perfil e Endereços:** O cliente edita suas informações pessoais e gerencia seus endereços de entrega.
- **Navegar e Pesquisar Produtos:** O cliente navega no catálogo de produtos e usa a busca para encontrar itens específicos.
- **Gerenciar Carrinho de Compras:** O cliente adiciona, remove ou altera a quantidade de itens no carrinho.
- **Finalizar Pedido:** O cliente revisa seu pedido, seleciona um endereço e procede para o pagamento.
- **Realizar Pagamento:** O cliente paga o pedido usando PIX ou cartão de crédito através do gateway de pagamento.
- **Acompanhar Pedido:** O cliente visualiza o status do pedido em tempo real, desde o preparo até a entrega.
- **Avaliar Produto e Entrega:** Após a entrega, o cliente avalia o produto e o motorista.
- **Receber Notificações:** O cliente recebe alertas sobre o status do pedido.

🏬 Logista

- **Cadastrar-se e ser Aprovado:** O lojista se registra na plataforma e aguarda a aprovação do administrador.
- **Gerenciar Perfil da Loja:** O lojista edita informações da loja, como descrição e horários de funcionamento.
- **Gerenciar Catálogo de Produtos:** O lojista adiciona, edita ou remove produtos, incluindo estoque, preços e variações.
- **Gerenciar Pedidos:** O lojista recebe notificações de novos pedidos, atualiza seu status e os prepara para a entrega.
- **Gerenciar Ganhos e Saques:** O lojista visualiza seus ganhos e solicita retiradas para sua conta bancária.
- **Acessar Relatórios:** O lojista visualiza relatórios sobre suas vendas e desempenho.

🚚 Motorista

- **Cadastrar-se e ser Aprovado:** O motorista se registra e fornece a documentação necessária para aprovação.
- **Gerenciar Status de Disponibilidade:** O motorista ativa e desativa sua disponibilidade para receber pedidos de entrega.
- **Receber e Aceitar Entregas:** O motorista recebe solicitações de entrega e pode aceitá-las para iniciar o trabalho.
- **Gerenciar Rota de Entrega:** O motorista segue a rota de coleta e entrega, atualizando o status do pedido.
- **Validar Coleta e Entrega:** O motorista utiliza códigos para confirmar a retirada do pedido na loja e a entrega ao cliente.

- **Gerenciar Ganhos e Saques:** O motorista visualiza seus ganhos e solicita saques para sua conta bancária.

⚙️ Administrador

- **Gerenciar Usuários:** O administrador visualiza, aprova, suspender ou remover contas de clientes, lojistas e motoristas.
- **Gerenciar Taxas e Comissão:** O administrador ajusta as taxas de serviço cobradas dos lojistas e as taxas de entrega.
- **Monitorar Sistema:** O administrador acompanha o status de todos os pedidos e entregas em tempo real.
- **Acessar Relatórios Gerenciais:** O administrador visualiza relatórios completos sobre o desempenho financeiro e operacional da plataforma.
- **Administrar Conteúdo:** O administrador pode gerenciar notificações globais, conteúdo da plataforma e aprovar lojas e produtos.

Exemplo de Especificação de Caso de Uso: Finalizar Pedido

Visão Geral

- **Nome do Caso de Uso:** Finalizar Pedido
- **Ator Principal:** Cliente
- **Atores Secundários:** Vendedor (Lojista), Sistema de Pagamento (BR Fideliza)
- **Pré-condição:** O cliente deve estar autenticado na plataforma e ter produtos no carrinho de compras.

Fluxo Principal (Caminho Feliz)

1. O Cliente navega até a tela do carrinho de compras e clica no botão "Finalizar Compra".
2. O sistema exibe a página de resumo do pedido, mostrando todos os produtos, subtotal, frete e total.
3. O Cliente seleciona um endereço de entrega salvo em seu perfil ou cadastra um novo endereço.
4. O sistema recalcula o frete e atualiza o total do pedido.
5. O Cliente escolhe o método de pagamento desejado (PIX ou Cartão de Crédito).
6. O Cliente insere as informações de pagamento e clica em "Pagar".

7. O sistema interage com o **Sistema de Pagamento (BR Fideliza)** para processar a transação.
8. O **Sistema de Pagamento (BR Fideliza)** retorna a confirmação de que o pagamento foi aprovado.
9. O sistema salva os detalhes do pedido no banco de dados, incluindo os itens e o status `PENDING` (ou `CONFIRMED`, dependendo da política de confirmação).
10. O sistema envia notificações para o **Cliente** (e-mail e push) e para o **Vendedor (Lojista)** informando sobre o novo pedido.
11. O sistema redireciona o Cliente para a página de confirmação do pedido, mostrando o número e os detalhes da compra.

Fluxos Alternativos

- **A1: Seleção de Endereço:** 1a. O Cliente não tem endereços salvos. 1b. O sistema solicita que o Cliente adicione um novo endereço antes de prosseguir.
- **A2: Aplicação de Cupom/Desconto:** 2a. Na página de resumo, o Cliente insere um código de cupom válido. 2b. O sistema valida o código do cupom e aplica o desconto, recalculando o valor total do pedido.
- **A3: Alteração de Itens no Carrinho:** 3a. Na página de resumo, o Cliente volta ao carrinho de compras. 3b. O Cliente altera a quantidade de um item ou remove-o. 3c. O sistema atualiza o carrinho e o Cliente pode voltar a finalizar o pedido.

Fluxos de Exceção

- **E1: Pagamento Recusado:** 1a. No passo 8 do Fluxo Principal, o **Sistema de Pagamento (BR Fideliza)** retorna uma falha na transação. 1b. O sistema exibe uma mensagem de erro para o Cliente, informando que o pagamento foi recusado. 1c. O Cliente é redirecionado para a tela de pagamento para tentar novamente ou escolher outro método.
- **E2: Item Fora de Estoque:** 2a. No passo 9 do Fluxo Principal, o sistema verifica o estoque e identifica que um ou mais itens não estão mais disponíveis. 2b. O sistema informa o Cliente quais itens estão indisponíveis e os remove do pedido. 2c. O Cliente pode finalizar o pedido com os itens restantes ou cancelá-lo.
- **E3: Endereço Fora da Área de Entrega:** 3a. No passo 4 do Fluxo Principal, o sistema detecta que o endereço de entrega está fora do raio de atuação do lojista. 3b. O sistema informa ao Cliente que não é possível realizar a entrega para o endereço selecionado e pede que ele escolha outro.

Pós-condições

1. Um novo registro do pedido é criado no banco de dados na tabela `orders`.
2. Registros de cada item do pedido são criados na tabela `order_items`.
3. O status do pedido é alterado de acordo com o resultado do pagamento.
4. O estoque dos produtos comprados é decrementado na tabela `products` e/ou `product_variants`.

5. O Cliente recebe uma notificação de confirmação e o Vendedor recebe uma notificação de novo pedido.

Diagrama de Classes

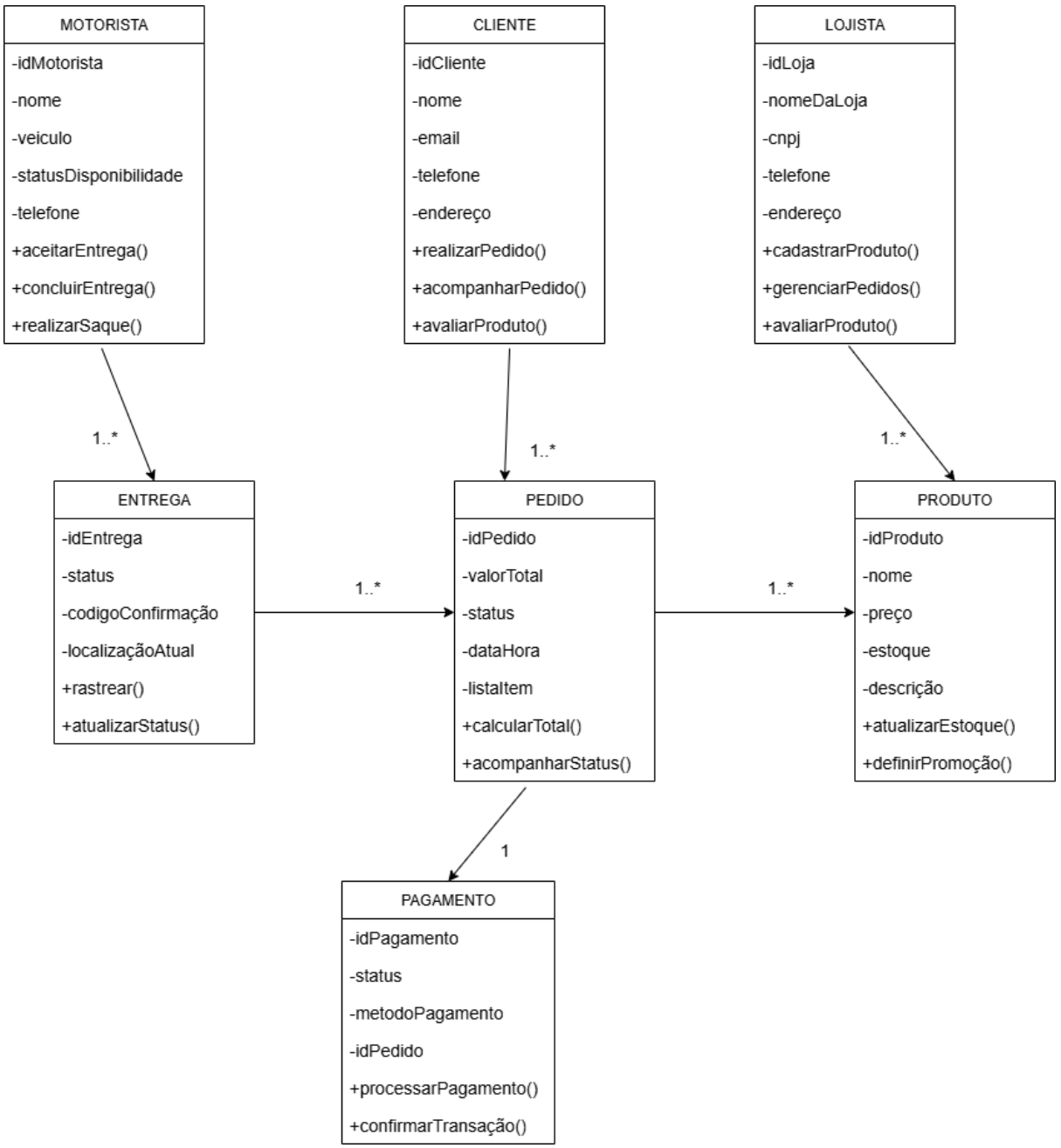

Diagrama de Classes

Visão Geral

Este documento descreve as classes principais do sistema WIN Marketplace, baseadas nas tabelas do banco de dados. Cada classe representa uma entidade, com seus atributos (propriedades) e seus métodos (comportamentos).

Classes, Atributos e Métodos

Class: User

- **Atributos:**
  - `id`: UUID
  - `email`: String
  - `password_hash`: String
  - `user_type`: String (enum: 'CUSTOMER', 'MERCHANT', 'DRIVER', 'ADMIN')
  - `status`: String (enum: 'ACTIVE', 'INACTIVE', 'SUSPENDED', 'PENDING')
  - `email_verified`: Boolean
  - `phone`: String
  - `created_at`: Date
  - `updated_at`: Date
  - `last_login`: Date
  - `profile_image_url`: String
- **Métodos:**
  - `authenticate(email, password)`: Boolean
  - `updateProfile(profileData)`: void
  - `changePassword(newPassword)`: Boolean
  - `deactivateAccount()`: void

Class: Customer

- **Atributos:**
  - `id`: UUID
  - `first_name`: String
  - `last_name`: String
  - `cpf`: String
  - `birth_date`: Date
  - `gender`: String
  - `preferences`: JSONB (objeto de preferências)
  - `loyalty_points`: Integer
  - `total_orders`: Integer
  - `total_spent`: Decimal

- **Métodos:**
  - `createOrder(cart, deliveryAddress)`: Order
  - `trackOrder()`: OrderStatus
  - `addAddress(addressData)`: Address
  - `reviewProduct(product, rating, comment)`: ProductReview
  - `reviewDriver(driver, rating, comment)`: DriverReview
- **Relacionamentos:**
  - `1:1` com `User` (herda ou está diretamente ligado a um `User`).
  - `1:N` com `Order` (um cliente pode ter muitos pedidos).

Class: Merchant

- **Atributos:**
  - `id`: UUID
  - `store_name`: String
  - `owner_name`: String
  - `cnpj`: String
  - `cpf_owner`: String
  - `description`: String
  - `category`: String
  - `is_approved`: Boolean
  - `commission_rate`: Decimal
  - `total_sales`: Decimal
  - `rating`: Decimal
  - `rating_count`: Integer
  - `operating_hours`: JSONB
  - `delivery_radius`: Integer
  - `minimum_order`: Decimal
- **Métodos:**
  - `addProduct(productData)`: Product
  - `updateProduct(productId, productData)`: void
  - `updateOrderStatus(orderId, newStatus)`: void
  - `viewSalesReports(period)`: ReportData
  - `processWithdrawal(amount)`: void
- **Relacionamentos:**
  - `1:1` com `User`.
  - `1:N` com `Product` (uma loja pode ter muitos produtos).
  - `1:N` com `OrderItem` (uma loja recebe muitos itens de pedidos).

Class: Driver

- **Atributos:**
  - `id`: UUID
  - `first_name`: String

    - `last_name`: String
    - `cpf`: String
    - `cnh`: String
    - `cnh_category`: String
    - `vehicle_type`: String
    - `vehicle_plate`: String
    - `is_approved`: Boolean
    - `is_available`: Boolean
    - `current_location`: Point (geometria)
    - `rating`: Decimal
    - `rating_count`: Integer
    - `total_deliveries`: Integer
    - `total_earnings`: Decimal
    - `bank_account`: JSONB
- **Métodos:**
    - `setAvailability(isAvailable)`: void
    - `acceptDelivery(deliveryId)`: void
    - `updateDeliveryStatus(deliveryId, newStatus)`: void
    - `viewEarnings(period)`: ReportData
- **Relacionamentos:**
    - `1:1` com `User`.
    - `1:N` com `Delivery` (um motorista realiza muitas entregas).

Class: Admin

- **Atributos:**
    - `id`: UUID
    - `name`: String
    - `role`: String
    - `permissions`: JSONB
    - `last_activity`: Date
- **Métodos:**
    - `manageUsers(userId, action)`: void
    - `approveMerchant(merchantId)`: void
    - `generateReports(reportType)`: ReportData
    - `setCommissionRate(newRate)`: void
- **Relacionamentos:**
    - `1:1` com `User`.

Class: Address

- **Atributos:**
    - `id`: UUID
    - `user_id`: UUID

    - `type`: String
    - `label`: String
    - `cep`: String
    - `street`: String
    - `number`: String
    - `complement`: String
    - `neighborhood`: String
    - `city`: String
    - `state`: String
    - `coordinates`: Point
    - `is_default`: Boolean
    - `created_at`: Date
- **Métodos:**
    - `validateCEP(cep)`: Boolean
- **Relacionamentos:**
    - `N:1` com `User` (muitos endereços pertencem a um único usuário).

Class: Category

- **Atributos:**
    - `id`: UUID
    - `name`: String
    - `slug`: String
    - `parent_id`: UUID
    - `icon`: String
    - `description`: String
    - `is_active`: Boolean
    - `sort_order`: Integer
    - `created_at`: Date
- **Relacionamentos:**
    - `1:N` com `Product` (uma categoria pode ter muitos produtos).
    - `1:N` consigo mesma (categorias podem ter subcategorias).

Class: Product

- **Atributos:**
    - `id`: UUID
    - `merchant_id`: UUID
    - `category_id`: UUID
    - `name`: String
    - `slug`: String
    - `description`: String
    - `short_description`: String

    - `sku`: String
    - `barcode`: String
    - `price`: Decimal
    - `compare_price`: Decimal
    - `cost_price`: Decimal
    - `stock_quantity`: Integer
    - `low_stock_alert`: Integer
    - `track_stock`: Boolean
    - `weight`: Decimal
    - `length`: Decimal
    - `width`: Decimal
    - `height`: Decimal
    - `status`: String
    - `is_featured`: Boolean
    - `meta_title`: String
    - `meta_description`: String
    - `keywords`: String
    - `views_count`: Integer
    - `sales_count`: Integer
    - `rating`: Decimal
    - `rating_count`: Integer
    - `created_at`: Date
    - `updated_at`: Date
- **Métodos:**
    - `calculatePriceWithDiscount()`: Decimal
    - `updateStock(quantityChange)`: void
    - `addRating(rating)`: void
- **Relacionamentos:**
    - `N:1` com `Merchant` (muitos produtos pertencem a uma loja).
    - `N:1` com `Category` (muitos produtos pertencem a uma categoria).
    - `1:N` com `ProductImage` (um produto tem muitas imagens).
    - `1:N` com `ProductVariant` (um produto pode ter muitas variações).
    - `1:N` com `ProductReview` (um produto pode ter muitas avaliações).
    - `1:N` com `OrderItem` (um produto pode estar em muitos itens de pedido).

Class: ProductImage

- **Atributos:**
    - `id`: UUID
    - `product_id`: UUID
    - `image_url`: String
    - `alt_text`: String
    - `sort_order`: Integer

    - `is_primary`: Boolean
- **Relacionamentos:**
    - `N:1` com `Product`.

Class: ProductVariant

- **Atributos:**
    - `id`: UUID
    - `product_id`: UUID
    - `name`: String
    - `sku`: String
    - `price`: Decimal
    - `stock_quantity`: Integer
    - `attributes`: JSONB
    - `is_active`: Boolean
    - `created_at`: Date
- **Relacionamentos:**
    - `N:1` com `Product`.

Class: Order

- **Atributos:**
    - `id`: UUID
    - `order_number`: String
    - `customer_id`: UUID
    - `status`: String
    - `subtotal`: Decimal
    - `discount_amount`: Decimal
    - `shipping_amount`: Decimal
    - `tax_amount`: Decimal
    - `total_amount`: Decimal
    - `delivery_address`: JSONB
    - `created_at`: Date
    - `confirmed_at`: Date
    - `delivered_at`: Date
    - `delivery_code`: String
    - `tracking_number`: String
    - `customer_notes`: String
    - `internal_notes`: String
- **Métodos:**
    - `calculateTotal()`: Decimal
    - `cancelOrder()`: void
- **Relacionamentos:**
    - `N:1` com `Customer`.

    - `1:N` com `OrderItem` (um pedido contém muitos itens).
    - `1:1` com `Payment` (um pedido tem um pagamento).
    - `1:1` com `Delivery` (um pedido tem uma entrega).

Class: OrderItem

- **Atributos:**
    - `id`: UUID
    - `order_id`: UUID
    - `product_id`: UUID
    - `variant_id`: UUID
    - `merchant_id`: UUID
    - `product_name`: String
    - `product_sku`: String
    - `quantity`: Integer
    - `unit_price`: Decimal
    - `total_price`: Decimal
    - `status`: String
    - `created_at`: Date
- **Relacionamentos:**
    - `N:1` com `Order`.
    - `N:1` com `Product`.
    - `N:1` com `Merchant`.

Class: Delivery

- **Atributos:**
    - `id`: UUID
    - `order_id`: UUID
    - `driver_id`: UUID
    - `status`: String
    - `pickup_address`: JSONB
    - `delivery_address`: JSONB
    - `current_location`: Point
    - `assigned_at`: Date
    - `picked_up_at`: Date
    - `delivered_at`: Date
    - `delivery_fee`: Decimal
    - `driver_earnings`: Decimal
    - `confirmation_code`: String
    - `proof_of_delivery`: JSONB
    - `delivery_notes`: String
    - `created_at`: Date

- **Métodos:**
  - `updateLocation(newLocation)`: void
- **Relacionamentos:**
  - `N:1` com `Order`.
  - `N:1` com `Driver`.
  - `1:N` com `DeliveryTracking` (uma entrega pode ter muitos pontos de rastreamento).

Class: DeliveryTracking

- **Atributos:**
  - `id`: UUID
  - `delivery_id`: UUID
  - `location`: Point
  - `timestamp`: Date
  - `status`: String
  - `notes`: String
- **Relacionamentos:**
  - `N:1` com `Delivery`.

Class: Payment

- **Atributos:**
  - `id`: UUID
  - `order_id`: UUID
  - `payment_method`: String
  - `status`: String
  - `amount`: Decimal
  - `fee_amount`: Decimal
  - `net_amount`: Decimal
  - `gateway_provider`: String
  - `transaction_id`: String
  - `gateway_response`: JSONB
  - `splits`: JSONB
  - `processed_at`: Date
  - `refunded_at`: Date
  - `created_at`: Date
- **Métodos:**
  - `processPayment()`: Boolean
  - `refund()`: Boolean
- **Relacionamentos:**
  - `N:1` com `Order`.

Class: ProductReview

- **Atributos:**
  - `id`: UUID
  - `product_id`: UUID
  - `customer_id`: UUID
  - `order_id`: UUID
  - `rating`: Integer
  - `comment`: String
- **Relacionamentos:**
  - `N:1` com `Product`.
  - `N:1` com `Customer`.
  - `N:1` com `Order`.

Class: DriverReview

- **Atributos:**
  - `id`: UUID
  - `driver_id`: UUID
  - `customer_id`: UUID
  - `order_id`: UUID
  - `rating`: Integer
  - `comment`: String
- **Relacionamentos:**
  - `N:1` com `Driver`.
  - `N:1` com `Customer`.
  - `N:1` com `Order`.

Explicação sobre a Class: `ProductVariant`

A classe **`ProductVariant`** serve para gerenciar as diferentes versões de um mesmo produto. Em vez de criar um novo produto para cada variação (por exemplo, uma camiseta azul e uma camiseta vermelha), você cria um único produto principal (a camiseta) e utiliza a classe `ProductVariant` para registrar suas opções.

Para entender melhor, imagine que um lojista de roupas vende a "Camiseta Básica WIN". Essa camiseta existe em duas cores e três tamanhos. Sem a classe `ProductVariant`, o lojista teria que cadastrar seis produtos diferentes:

- Camiseta Básica WIN - Azul - P
- Camiseta Básica WIN - Azul - M
- Camiseta Básica WIN - Azul - G

- Camiseta Básica WIN - Vermelha - P
- Camiseta Básica WIN - Vermelha - M
- Camiseta Básica WIN - Vermelha - G

Isso tornaria o gerenciamento do catálogo e do estoque muito difícil.

Com a classe `ProductVariant`, o processo fica mais organizado:

1. Você tem a classe principal **`Product`** com os atributos genéricos do produto (nome: "Camiseta Básica WIN", descrição, etc.).
2. Cada opção específica (Azul, Vermelha, P, M, G) é uma instância da classe **`ProductVariant`**.

Essa classe é essencial para:

- **Organização do Catálogo:** Agrupar produtos semelhantes.
- **Gestão de Estoque:** Controlar o estoque de cada variação de forma independente.
- **Flexibilidade:** Permitir que o cliente escolha as opções que deseja (tamanho, cor, etc.) em uma única página de produto.

Os atributos que você incluiu no diagrama, como **`name`**, **`sku`**, **`price`** e **`stock_quantity`**, são cruciais porque permitem que cada variação tenha seu próprio preço e seu próprio controle de estoque. O atributo **`attributes`** (`{"color": "red", "size": "M"}`) é fundamental para descrever as características de cada variação de forma dinâmica.