**WebStore de Mercadinho — Planejamento (Etapa 1)**

Objetivo: documentar arquitetura, requisitos, modelagem, API e plano de execução para um e-commerce de mercadinho com **React + Vite + Sass + Axios** no frontend e **Django + Django REST Framework + SimpleJWT + MySQL** no backend. Deploy: **Vercel** (frontend) e **Render** (backend). Suporte a pagamento via link (ex.: Mercado Pago/Stripe, com PIX/Cartão via checkout hospedado).

**1) Visão e Escopo**

* **MVP**: catálogo, busca/filtragem, carrinho, cadastro/login, checkout com **link de pagamento**, gestão de estoque, área do cliente (pedidos), área admin (CRUD de produtos, estoque e pedidos).
* **Público**: clientes finais (compras rápidas), administradores/estoquistas.
* **Diferenciais**: interface acessível (alto contraste e rótulos claros), frete simples (ex.: taxa fixa por bairro, MVP), e checkout simplificado por link.

**2) Requisitos Funcionais (RF)**

1. **Autenticação**: cadastro, login JWT (access/refresh), recuperação de senha (MVP: reset por e-mail linkado ao backend; opcional pós-MVP).
2. **Catálogo**: categorias, lista de produtos, detalhe de produto, imagens múltiplas.
3. **Carrinho**: adicionar/editar/remover itens; carrinho persistente (localStorage + sincronização server-side após login).
4. **Estoque**: decrementar ao fechar pedido; alerta de baixo estoque; impedir compra se indisponível.
5. **Checkout**: endereço de entrega (MVP simples), resumo do pedido, criação de **link de pagamento** via gateway.
6. **Pedidos**: status (pending → paid → shipped/finished; canceled), histórico do cliente.
7. **Admin**: CRUD categorias/produtos, upload de imagem (Pillow), ajuste de estoque, acompanhar pedidos.
8. **Pagamentos**: geração de link e **webhook** para confirmação de pagamento.

**Requisitos Não Funcionais (RNF)**

* Performance: TTFB < 600ms backend, FCP < 2s em 4G.
* Segurança: HTTPS; JWT curto (15min) + refresh; CORS restrito; senhas com Argon2/BCrypt; validação sólida.
* Observabilidade: logs estruturados, request ID, métricas básicas.
* Escalabilidade: stateless no backend; cache de lista de produtos; paginação.
* Acessibilidade: contraste AA, foco visível, semântica ARIA.

**3) Stack Tecnológica**

**Frontend**

* React 18 + Vite, React Router, **Sass** (SCSS), **Axios**, React Hook Form + Zod (validação), TanStack Query (cache de requisições), Zustand (estado global leve) [opcional], Day.js, UUID.

**Backend**

* Django 5, Django REST Framework, **djangorestframework-simplejwt** (JWT), **mysqlclient** (MySQL), **Pillow** (imagens), **python-dotenv** (ou django-environ), **corsheaders**, drf-spectacular (OpenAPI/Swagger), whitenoise (assets estáticos admin), gunicorn (produção).

**Infra/DevOps**

* MySQL 8 (InnoDB, utf8mb4), Render (serviço web + job de migrações), Vercel (static hosting + proxy), Cloud storage para imagens (opcional: Cloudinary ou S3; MVP: FileSystem do Render/volume persistente).

**4) Arquitetura (Visão Geral)**

**Cliente (Web)** ⇄ **API REST (Django/DRF)** ⇄ **MySQL** ⇄ **Gateway de Pagamento**

* Frontend consome /api/v1/... via Axios.
* Autenticação: JWT Bearer no header.
* Webhook do gateway → endpoint /api/v1/payments/webhook atualiza pedido.
* Upload de imagens: multipart → armazenamento (FileSystem/Cloud).

**5) Modelagem de Dados (ERD)**

Entidades principais:

* **User**(id, email\*, password\_hash, first\_name, last\_name, phone, role[customer|admin], is\_active, created\_at)
* **Address**(id, user\_id→User, label, street, number, complement, neighborhood, city, state, zip\_code, is\_default)
* **Category**(id, name, slug, parent\_id→Category null)
* **Product**(id, sku\*, name, slug, description, price DEC(10,2), cost DEC(10,2) opt, category\_id→Category, is\_active, created\_at, updated\_at)
* **ProductImage**(id, product\_id→Product, image, alt\_text, sort\_order)
* **Inventory**(id, product\_id→Product, quantity INT, min\_threshold INT)
* **Cart**(id, user\_id→User null, session\_id\*, status[open|converted|abandoned], total DEC(10,2), updated\_at)
* **CartItem**(id, cart\_id→Cart, product\_id→Product, quantity INT, unit\_price DEC(10,2), subtotal DEC(10,2))
* **Order**(id, code\*, user\_id→User, cart\_id→Cart, status[pending|paid|shipped|finished|canceled], total DEC(10,2), shipping\_address\_id→Address, created\_at)
* **OrderItem**(id, order\_id→Order, product\_id→Product, quantity INT, unit\_price DEC(10,2), subtotal DEC(10,2))
* **Payment**(id, order\_id→Order, gateway, status[pending|paid|failed|refunded], external\_reference, checkout\_link, amount DEC(10,2), currency 'BRL', paid\_at null)
* **AuditLog**(id, actor\_id→User, action, entity, entity\_id, payload JSON, created\_at) [opcional]

**Observações**

* Índices: User.email, Product.sku, Product.slug, Category.slug, Cart.session\_id.
* Colação DB: utf8mb4\_0900\_ai\_ci.
* Precisão de preço: Decimal(10,2).

**6) Design da API (v1)**

**Autenticação**

* POST /api/v1/auth/register → cria usuário
* POST /api/v1/auth/token → JWT (access, refresh)
* POST /api/v1/auth/token/refresh
* GET /api/v1/users/me (auth) → perfil
* PUT /api/v1/users/me (auth)

**Catálogo**

* GET /api/v1/catalog/categories
* GET /api/v1/catalog/products?search=&category=&page=&ordering=price|name|-price
* GET /api/v1/catalog/products/{id}

**Carrinho**

* GET /api/v1/cart (por session\_id ou auth)
* POST /api/v1/cart/items {product\_id, quantity}
* PATCH /api/v1/cart/items/{item\_id} {quantity}
* DELETE /api/v1/cart/items/{item\_id}

**Pedidos**

* POST /api/v1/orders (gera pedido a partir do carrinho)
* GET /api/v1/orders (cliente autenticado)
* GET /api/v1/orders/{id}

**Pagamentos**

* POST /api/v1/payments/{order\_id}/link → cria link no gateway
* POST /api/v1/payments/webhook → callback do gateway

**Admin (role=admin)**

* GET/POST /api/v1/admin/products
* GET/PUT/PATCH/DELETE /api/v1/admin/products/{id}
* PATCH /api/v1/admin/inventory/{product\_id} {quantity}
* GET/PATCH /api/v1/admin/orders/{id} (alterar status)

**Padrões**

* Resposta padrão: { success, data, error }
* Paginação DRF: { count, next, previous, results }
* Erros: códigos semânticos (ex.: stock\_unavailable).

**7) Fluxos Críticos**

**Cadastro/Login**

1. Usuário envia dados → validação → criação → retorno 201.
2. Login → token/refresh via SimpleJWT.

**Carrinho**

* Visitante: usa session\_id (cookie + localStorage); ao logar, mesclar carrinhos.

**Checkout**

1. Validar estoque + endereço → criar Order(pending) e itens.
2. Criar Payment(pending) e chamar gateway para **gerar link**.
3. Usuário paga no checkout hospedado → gateway envia **webhook**.
4. Webhook valida assinatura → Payment.status=paid, Order.status=paid e **decrementa estoque** (se ainda não decrementado).

**Estoque**

* Decrementar apenas quando Order vira paid (ou reserved por X minutos se necessário; MVP: sem reserva).

**8) Estrutura de Repositórios e Pastas**

**Opção A — Monorepo**

mercadinho/

frontend/

src/

api/ # axios, hooks TanStack Query

pages/ # Home, Produtos, Produto, Carrinho, Login, Cadastro, Conta, Admin

components/ # UI reutilizável

styles/ # Sass (variables, mixins, base, components)

store/ # Zustand (opcional)

utils/

public/

.env

backend/

mercadinho/ # settings, urls

apps/

accounts/

catalog/

cart/

orders/

payments/

adminpanel/

media/ # imagens (se FileSystem)

requirements.txt

.env

**Opção B — Repositórios separados**

* mercadinho-frontend (Vercel)
* mercadinho-backend (Render)

**9) Convenções e Boas Práticas**

* **API Prefix**: /api/v1/ (versionamento desde o início).
* **JWT**: expiração access=15min, refresh=7d; rota de blacklist opcional.
* **CORS**: permitir domínios do Vercel + localhost.
* **Upload**: limite 2MB por imagem; formatos: JPG/PNG/WebP; Pillow para validação/redimensionamento.
* **Commit**: Conventional Commits (feat:, fix:, chore:...).
* **Código**: isort + black (backend), eslint + prettier (frontend).

**10) Pagamentos (MVP por Link)**

* Gateways sugeridos: **Mercado Pago** (PIX + cartão) ou **Stripe Checkout** (cartão; PIX via parceiros).
* **Estratégia**: o backend cria a preferência/checkout e retorna checkout\_link para o frontend redirecionar.
* **Webhook**: rota dedicada; validar assinatura secreta; reconciliar idempotente (não marcar pago duas vezes).

**11) Observabilidade & Segurança**

* **Logs**: request ID por requisição; logar eventos de pedido/pagamento.
* **Proteção**: rate limit para /auth/\* e /payments/webhook.
* **Headers**: Secure, HttpOnly, SameSite=Lax para cookies (session\_id).

**12) Variáveis de Ambiente (.env)**

**Backend**

DJANGO\_SECRET\_KEY=

DJANGO\_DEBUG=false

ALLOWED\_HOSTS=api.mercadinho.com,localhost

DB\_HOST=

DB\_PORT=3306

DB\_NAME=

DB\_USER=

DB\_PASSWORD=

CORS\_ALLOWED\_ORIGINS=https://mercadinho.vercel.app,http://localhost:5173

JWT\_ACCESS\_LIFETIME\_MIN=15

JWT\_REFRESH\_LIFETIME\_DAYS=7

MEDIA\_ROOT=/opt/render/project/media

MEDIA\_URL=/media/

PAYMENT\_PROVIDER=mercadopago

MP\_ACCESS\_TOKEN=

MP\_WEBHOOK\_SECRET=

**Frontend** (.env)

VITE\_API\_BASE\_URL=https://api.mercadinho.render.com/api/v1

VITE\_PAYMENT\_SUCCESS\_URL=https://mercadinho.vercel.app/sucesso

VITE\_PAYMENT\_CANCEL\_URL=https://mercadinho.vercel.app/cancelado

**13) Plano de Execução (Sprints)**

**Sprint 0 – Setup (0.5–1 dia)**

* Repos, lint/format, dependências, CI básico, templates PR.

**Sprint 1 – Backend Fundacional (2–3 dias)**

* Django+DRF, MySQL, SimpleJWT, apps accounts, catalog (Category/Product/Image/Inventory), cart.
* Endpoints de catálogo + carrinho.

**Sprint 2 – Pedidos & Pagamentos (2–3 dias)**

* orders + payments, criação de pedido, integração a gateway (somente geração de link), webhook e atualização de status/estoque.

**Sprint 3 – Frontend Base (2–3 dias)**

* Layout, rotas, home, catálogo, detalhe, carrinho, login/cadastro, integração com API.

**Sprint 4 – Admin (2 dias)**

* Painel simples para CRUD de produtos/estoque e acompanhamento de pedidos.

**Sprint 5 – Hardening & Deploy (1–2 dias)**

* A11y, loading states, testes essenciais, build, deploy (Vercel/Render), variáveis de ambiente e domínios.

**14) Definition of Done (DoD)**

* API v1 documentada (Swagger), testes de fumaça, dados seed.
* Fluxo completo: navegar → adicionar ao carrinho → checkout → pagar (simulado ou sandbox) → status pago.
* Admin consegue criar/editar produto com imagem e gerenciar estoque.

**15) Próxima Etapa (Etapa 2)**

**Criar o esqueleto do backend** e explicar arquivo por arquivo:

1. Criar projeto Dj