



DELIVEREI v1 - PLANEJAMENTO DA FASE 2

Data: 2025-10-08

Versão: 1.0

Status: Aguardando Aprovação



Resumo Executivo

Este documento detalha o planejamento completo da **Fase 2** do backend do projeto DELIVEREI, um sistema multi-tenant de delivery. A Fase 1 estabeleceu a fundação com autenticação, produtos e endpoints públicos. A Fase 2 focará em completar as funcionalidades core do sistema: pedidos, checkout, clientes, e funcionalidades administrativas.

Objetivos da Fase 2

- ✓ Implementar sistema completo de pedidos (CRUD + gestão de status)
- ✓ Criar fluxo de checkout com integração de pagamento
- ✓ Implementar gestão de clientes
- ✓ Desenvolver funcionalidades de SuperAdmin (empresas, assinaturas)
- ✓ Criar sistema de tickets de suporte
- ✓ Adicionar funcionalidades avançadas (cupons, notificações)



Status da Fase 1 (Concluída)

Implementado com Sucesso

1. Infraestrutura Base

- ✓ NestJS configurado com TypeScript
- ✓ Prisma ORM integrado com Supabase PostgreSQL
- ✓ Redis configurado (Docker Compose)
- ✓ Estrutura modular do projeto
- ✓ PrismaService e PrismaModule globais

2. Autenticação e Autorização

- ✓ JWT com Access Token (15min) e Refresh Token (7 dias)
- ✓ Endpoints: `/auth/login`, `/auth/signup`, `/auth/refresh`, `/auth/logout`
- ✓ Guards: `JwtAuthGuard`, `RolesGuard`
- ✓ Decorators: `@Public()`, `@Roles()`, `@CurrentUser()`, `@CurrentEmpresa()`
- ✓ Estratégia JWT com Passport
- ✓ Sistema de roles: `SUPER_ADMIN`, `ADMIN_EMPRESA`, `CLIENTE`

3. Multi-tenancy

- ✓ `TenantMiddleware` (extração de subdomínio)
- ✓ Isolamento de dados por empresa

- ☒ Suporte a slug público nas URLs

4. Modelos Prisma

- ☒ Empresa (id, nome, slug, subdominio, ativo)
- ☒ Usuario (id, email, senha, nome, role, empresaId)
- ☒ Produto (id, nome, descricao, preco, imagem, empresaId, estoque, categoria)
- ☒ Pedido (id, numero, status, total, clienteId, empresaId)
- ☒ RefreshToken (id, token, usuarioId, expiresAt)

5. Endpoints Implementados

Autenticação (Público)

- ☒ POST /auth/login - Login com email/senha
- ☒ POST /auth/signup - Criar conta
- ☒ POST /auth/refresh - Renovar access token
- ☒ POST /auth/logout - Logout

Produtos (Autenticado - Admin)

- ☒ GET /produtos - Listar produtos (paginado, filtros)
- ☒ POST /produtos - Criar produto
- ☒ GET /produtos/:id - Buscar produto
- ☒ PATCH /produtos/:id - Atualizar produto
- ☒ DELETE /produtos/:id - Soft delete
- ☒ DELETE /produtos/:id/hard - Hard delete (SUPER_ADMIN)

Público (Sem autenticação)

- ☒ GET /public/:slug/info - Info da loja
- ☒ GET /public/:slug/produtos - Produtos da loja (paginado)
- ☒ GET /public/:slug/produtos/:id - Detalhes do produto
- ☒ GET /public/:slug/categorias - Categorias disponíveis

6. Seed de Dados

- ☒ 2 empresas (Pizza Express, Burger King)
- ☒ 4 usuários (1 super admin, 2 admins empresa, 1 cliente)
- ☒ 8 produtos (5 Pizza Express, 3 Burger King)

7. Validações e Segurança

- ☒ DTOs com class-validator
- ☒ Bcrypt para hash de senhas
- ☒ CORS configurado
- ☒ Exception filters

Objetivos da Fase 2

Prioridade ALTA (Essencial para MVP)

1. **Sistema de Pedidos** - CRUD completo + gestão de status
2. **Checkout e Pagamentos** - Fluxo completo de compra
3. **Gestão de Clientes** - CRUD de clientes
4. **Integrações de Pagamento** - Asaas (PIX, Cartão, Boletão)

Prioridade MÉDIA (Importante para operação)

1. **Gestão de Empresas** - CRUD para SuperAdmin
2. **Sistema de Cupons** - Descontos e promoções
3. **Notificações** - Email e WhatsApp
4. **Dashboard Analytics** - Métricas e relatórios

Prioridade BAIXA (Nice to have)

1. **Sistema de Assinaturas** - Gestão de planos
2. **Sistema de Tickets** - Suporte ao cliente
3. **Logs e Auditoria** - Rastreamento de ações
4. **Upload de Imagens** - Cloudinary/S3



Funcionalidades Detalhadas da Fase 2



ETAPA 2.1: Sistema de Pedidos (ALTA PRIORIDADE)

Complexidade: ★★★ (Média-Alta)

Tempo Estimado: 2-3 dias

Dependências: Fase 1 completa

Objetivos

- Criar modelo completo de Pedido com itens
- Implementar CRUD de pedidos
- Gestão de status do pedido
- Histórico de mudanças de status
- Cálculos automáticos (subtotal, taxas, total)

Modelos Prisma a Criar/Atualizar

```

model Pedido {
    id String @id @default(uuid())
    numero String @unique
    status StatusPedido @default(PENDENTE)
    statusPagamento StatusPagamento @default(PENDENTE)
    subtotal Decimal @db.Decimal(10, 2)
    taxaEntrega Decimal @db.Decimal(10, 2) @default(0)
    desconto Decimal @db.Decimal(10, 2) @default(0)
    total Decimal @db.Decimal(10, 2)
    clienteId String
    empresaId String
    cupomId String?
    observacoes String?

    // Endereço de entrega
    enderecoRua String
    enderecoNumero String
    enderecoComplemento String?
    enderecoBairro String
    enderecoCidade String
    enderecoUF String
    enderecoCEP String

    // Dados do cliente (snapshot)
    clienteNome String
    clienteEmail String
    clienteTelefone String

    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt

    cliente Usuario @relation(fields: [clienteId], references: [id])
    empresa Empresa @relation(fields: [empresaId], references: [id])
    cupom Cupom? @relation(fields: [cupomId], references: [id])
    itens ItemPedido[]
    historico HistoricoPedido[]
    pagamento Pagamento?

    @@index([clienteId])
    @@index([empresaId])
    @@index([status])
    @@index([statusPagamento])
    @@index([createdAt])
}

model ItemPedido {
    id String @id @default(uuid())
    pedidoId String
    produtoId String
    quantidade Int
    precoUnitario Decimal @db.Decimal(10, 2)
    subtotal Decimal @db.Decimal(10, 2)

    // Snapshot do produto
    produtoNome String
    produtoImagem String?

    pedido Pedido @relation(fields: [pedidoId], references: [id], onDelete: Cascade)

    @@index([pedidoId])
    @@index([produtoId])
}

```

```

model HistoricoPedido {
  id          String   @id @default(uuid())
  pedidoId    String
  statusAnterior StatusPedido?
  statusNovo  StatusPedido
  usuarioId   String?
  observacao  String?
  createdAt   DateTime @default(now())

  pedido      Pedido   @relation(fields: [pedidoId], references: [id], onDelete: Cascade)

  @@index([pedidoId])
  @@index([createdAt])
}

enum StatusPedido {
  PENDENTE
  CONFIRMADO
  EM_PREPARO
  SAIU_ENTREGA
  ENTREGUE
  CANCELADO
}

enum StatusPagamento {
  PENDENTE
  APROVADO
  RECUSADO
  ESTORNADO
}

```

Endpoints a Implementar

```

// Módulo: pedidos
// Controller: pedidos.controller.ts

// Admin - Gestão de Pedidos
GET    /pedidos                // Listar pedidos (paginado, filtros)
GET    /pedidos/:id           // Detalhes do pedido
PATCH /pedidos/:id/status    // Atualizar status
GET    /pedidos/:id/historico // Histórico de status
DELETE /pedidos/:id           // Cancelar pedido

// Cliente - Meus Pedidos
GET    /pedidos/meus          // Pedidos do cliente logado
GET    /pedidos/meus/:id      // Detalhes do meu pedido

// Público - Rastreamento
GET    /public/pedidos/:numero/rastrear // Rastrear pedido por número

```

DTOs Necessários

```
// create-pedido.dto.ts
export class CreatePedidoDto {
  clienteNome: string;
  clienteEmail: string;
  clienteTelefone: string;
  enderecoRua: string;
  enderecoNumero: string;
  enderecoComplemento?: string;
  enderecoBairro: string;
  enderecoCidade: string;
  enderecoUF: string;
  enderecoCEP: string;
  itens: ItemPedidoDto[];
  cupomCodigo?: string;
  observacoes?: string;
}

export class ItemPedidoDto {
  produtoId: string;
  quantidade: number;
}

// update-status-pedido.dto.ts
export class UpdateStatusPedidoDto {
  status: StatusPedido;
  observacao?: string;
}

// query-pedidos.dto.ts
export class QueryPedidosDto {
  page?: number = 1;
  limit?: number = 20;
  status?: StatusPedido;
  statusPagamento?: StatusPagamento;
  dataInicio?: Date;
  dataFim?: Date;
  clienteId?: string;
  search?: string; // busca por número, cliente
}
```

Regras de Negócio

1. Criação de Pedido

- Validar estoque dos produtos
- Calcular subtotal (soma dos itens)
- Aplicar cupom de desconto (se válido)
- Calcular taxa de entrega (pode ser fixa ou por CEP)
- Calcular total final
- Gerar número único do pedido (ex: PED-20251008-0001)
- Criar snapshot dos dados do produto (nome, preço)

2. Mudança de Status

- Apenas admin pode mudar status
- Registrar no histórico quem mudou e quando
- Validar transições permitidas:
 - PENDENTE → CONFIRMADO, CANCELADO

- CONFIRMADO → EM_PREPARO, CANCELADO
- EM_PREPARO → SAIU_ENTREGA, CANCELADO
- SAIU_ENTREGA → ENTREGUE
- ENTREGUE → (final)
- CANCELADO → (final)

3. Cancelamento

- Apenas pedidos PENDENTE ou CONFIRMADO podem ser cancelados
- Devolver estoque dos produtos
- Se pagamento aprovado, iniciar estorno

4. Listagem

- Admin vê todos os pedidos da empresa
- Cliente vê apenas seus pedidos
- Filtros: status, data, cliente
- Ordenação padrão: mais recentes primeiro

ETAPA 2.2: Checkout e Pagamentos (ALTA PRIORIDADE)

Complexidade: ★★★★★ (Alta)

Tempo Estimado: 3-4 dias

Dependências: Etapa 2.1 (Pedidos)

Objetivos

- Criar fluxo completo de checkout
- Integrar com gateway de pagamento (Asaas)
- Suportar múltiplos métodos: PIX, Cartão, Boletto
- Webhooks para atualização de status
- Validação de cupons de desconto

Modelos Prisma a Criar

```

model Pagamento {
  id          String      @id @default(uuid())
  pedidoId    String      @unique
  metodo      MetodoPagamento
  status      StatusPagamento @default(PENDENTE)
  valor       Decimal      @db.Decimal(10, 2)

  // Dados do gateway (Asaas)
  gatewayId    String?     // ID do pagamento no Asaas
  gatewayStatus String?
  pixQrCode    String?     @db.Text
  pixCopiaECola String?    @db.Text
  boletoUrl    String?
  boletoLinhaDigitavel String?

  // Dados do cartão (tokenizado)
  cartaoToken   String?
  cartaoBandeira String?
  cartaoUltimosDigitos String?

  // Metadados
  tentativas    Int         @default(0)
  ultimaTentativa DateTime?
  erroMensagem  String?

  createdAt     DateTime    @default(now())
  updatedAt     DateTime    @updatedAt

  pedido        Pedido      @relation(fields: [pedidoId], references: [id], onDelete: Cascade)

  @@index([pedidoId])
  @@index([gatewayId])
  @@index([status])
}

enum MetodoPagamento {
  PIX
  CARTAO_CREDITO
  CARTAO_DEBITO
  BOLETO
}

```

Endpoints a Implementar

```

// Módulo: checkout
// Controller: checkout.controller.ts

POST /checkout/create-order // Criar pedido + iniciar pagamento
POST /checkout/process-payment // Processar pagamento
GET /checkout/payment-status/:pedidoId // Status do pagamento

// Webhooks (Público)
POST /webhooks/asaas // Webhook do Asaas
POST /webhooks/asaas/verify // Verificar assinatura

```

DTOs Necessários

```
// checkout.dto.ts
export class CheckoutDto {
  // Dados do cliente
  clienteNome: string;
  clienteEmail: string;
  clienteTelefone: string;
  clienteCPF?: string;

  // Endereço
  enderecoRua: string;
  enderecoNumero: string;
  enderecoComplemento?: string;
  enderecoBairro: string;
  enderecoCidade: string;
  enderecoUF: string;
  enderecoCEP: string;

  // Itens
  itens: ItemCheckoutDto[];

  // Pagamento
  metodoPagamento: MetodoPagamento;
  dadosCartao?: DadosCartaoDto;

  // Outros
  cupomCodigo?: string;
  observacoes?: string;
}

export class ItemCheckoutDto {
  produtoId: string;
  quantidade: number;
}

export class DadosCartaoDto {
  numero: string;
  nome: string;
  validade: string; // MM/YY
  cvv: string;
}

// asaas-webhook.dto.ts
export class AsaasWebhookDto {
  event: string;
  payment: {
    id: string;
    status: string;
    value: number;
    // ... outros campos do Asaas
  };
}
```

Serviço de Integração Asaas

```
// asaas.service.ts
export class AsaasService {
  async createPayment(data: CreatePaymentDto): Promise<AsaasPaymentResponse>
  async getPayment(paymentId: string): Promise<AsaasPaymentResponse>
  async createPixPayment(data: CreatePixPaymentDto): Promise<AsaasPixResponse>
  async createBoletoPayment(data: CreateBoletoPaymentDto):
  Promise<AsaasBoletoResponse>
  async createCreditCardPayment(data: CreateCardPaymentDto):
  Promise<AsaasCardResponse>
  async cancelPayment(paymentId: string): Promise<void>
  async refundPayment(paymentId: string): Promise<void>
  async verifyWebhookSignature(signature: string, payload: string): boolean
}
```

Fluxo de Checkout

1. Cliente finaliza carrinho

- Frontend envia CheckoutDto para /checkout/create-order

2. Backend processa

- Valida dados do cliente e endereço
- Valida estoque dos produtos
- Aplica cupom (se válido)
- Calcula totais
- Cria pedido no banco (status: PENDENTE)
- Cria registro de pagamento

3. Inicia pagamento no Asaas

- **PIX**: Gera QR Code e Copia e Cola
- **Cartão**: Tokeniza e processa
- **Boleto**: Gera boleto e linha digitável

4. Retorna resposta

```
json
{
  "pedidoId": "uuid",
  "numero": "PED-20251008-0001",
  "status": "PENDENTE",
  "pagamento": {
    "metodo": "PIX",
    "status": "PENDENTE",
    "pixQrCode": "data:image/png;base64,...",
    "pixCopiaECola": "00020126..."
  }
}
```

5. Webhook do Asaas

- Asaas notifica mudança de status
- Backend valida assinatura
- Atualiza status do pagamento
- Atualiza status do pedido
- Envia notificação ao cliente

Regras de Negócio

1. Validação de Estoque

- Verificar disponibilidade antes de criar pedido
- Reservar estoque ao criar pedido
- Devolver estoque se pagamento falhar/expirar

2. Cupons de Desconto

- Validar código do cupom
- Verificar validade (data início/fim)
- Verificar limite de uso
- Verificar valor mínimo do pedido
- Aplicar desconto (percentual ou fixo)

3. Timeout de Pagamento

- PIX: 30 minutos
- Boletto: 3 dias
- Cartão: imediato
- Cancelar pedido se expirar

4. Segurança

- Nunca armazenar dados completos do cartão
- Usar tokenização do Asaas
- Validar assinatura dos webhooks
- Rate limiting nos endpoints

ETAPA 2.3: Gestão de Clientes (ALTA PRIORIDADE)

Complexidade: ★★ (Média)

Tempo Estimado: 1-2 dias

Dependências: Fase 1 completa

Objetivos

- CRUD completo de clientes
- Histórico de pedidos do cliente
- Endereços salvos
- Dados de contato

Modelos Prisma a Atualizar/Criar

```

model Usuario {
  // ... campos existentes ...
  cpf          String?
  telefone     String?
  dataNascimento DateTime?

  enderecos    Endereco[]
  // ... relações existentes ...
}

model Endereco {
  id          String @id @default(uuid())
  usuarioId   String
  nome        String // ex: "Casa", "Trabalho"
  rua         String
  numero      String
  complemento String?
  bairro      String
  cidade      String
  uf          String
  cep         String
  principal   Boolean @default(false)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  usuario     Usuario @relation(fields: [usuarioId], references: [id], onDelete: Cascade)

  @@index([usuarioId])
}

```

Endpoints a Implementar

```

// Módulo: clientes
// Controller: clientes.controller.ts

// Admin - Gestão de Clientes
GET    /clientes                // Listar clientes (paginado)
GET    /clientes/:id           // Detalhes do cliente
POST   /clientes               // Criar cliente
PATCH /clientes/:id           // Atualizar cliente
DELETE /clientes/:id           // Desativar cliente
GET    /clientes/:id/pedidos    // Pedidos do cliente
GET    /clientes/:id/enderecos  // Endereços do cliente

// Cliente - Meu Perfil
GET    /clientes/me            // Meus dados
PATCH /clientes/me            // Atualizar meus dados
GET    /clientes/me/enderecos  // Meus endereços
POST   /clientes/me/enderecos  // Adicionar endereço
PATCH /clientes/me/enderecos/:id // Atualizar endereço
DELETE /clientes/me/enderecos/:id // Remover endereço

```

DTOs Necessários

```
// create-cliente.dto.ts
export class CreateClienteDto {
  nome: string;
  email: string;
  telefone: string;
  cpf?: string;
  dataNascimento?: Date;
  senha?: string; // opcional, pode ser gerada
}

// update-cliente.dto.ts
export class UpdateClienteDto {
  nome?: string;
  telefone?: string;
  cpf?: string;
  dataNascimento?: Date;
}

// create-endereco.dto.ts
export class CreateEnderecoDto {
  nome: string;
  rua: string;
  numero: string;
  complemento?: string;
  bairro: string;
  cidade: string;
  uf: string;
  cep: string;
  principal?: boolean;
}
```

ETAPA 2.4: Sistema de Cupons (MÉDIA PRIORIDADE)

Complexidade: ★★ (Média)

Tempo Estimado: 1-2 dias

Dependências: Etapa 2.2 (Checkout)

Modelos Prisma a Criar

```

model Cupom {
  id          String    @id @default(uuid())
  codigo      String    @unique
  descricao   String
  tipo        TipoCupom
  valor       Decimal    @db.Decimal(10, 2)
  valorMinimo Decimal?   @db.Decimal(10, 2)
  limiteUso   Int?
  usosPorCliente Int?    @default(1)
  dataInicio  DateTime
  dataFim     DateTime
  ativo       Boolean    @default(true)
  empresaId   String
  createdAt   DateTime   @default(now())
  updatedAt   DateTime   @updatedAt

  empresa      Empresa    @relation(fields: [empresaId], references: [id], onDelete: Cascade)
  pedidos      Pedido[]
  usos          UsoCupom[]

  @@index([empresaId])
  @@index([codigo])
  @@index([ativo])
}

model UsoCupom {
  id          String    @id @default(uuid())
  cupomId     String
  usuarioId   String
  pedidoId    String?
  createdAt   DateTime   @default(now())

  cupom       Cupom      @relation(fields: [cupomId], references: [id], onDelete: Cascade)

  @@index([cupomId])
  @@index([usuarioId])
}

enum TipoCupom {
  PERCENTUAL
  VALOR_FIXO
  FRETE_GRATIS
}

```

Endpoints a Implementar

```

// Admin - Gestão de Cupons
GET    /cupons           // Listar cupons
POST   /cupons           // Criar cupom
GET    /cupons/:id       // Detalhes do cupom
PATCH /cupons/:id       // Atualizar cupom
DELETE /cupons/:id       // Desativar cupom
GET    /cupons/:id/usos  // Histórico de uso

// Público - Validação
POST   /cupons/validar   // Validar cupom

```

● ETAPA 2.5: Gestão de Empresas - SuperAdmin (MÉDIA PRIORIDADE)

Complexidade: ★★☆☆ (Média-Alta)

Tempo Estimado: 2-3 dias

Dependências: Fase 1 completa

Modelos Prisma a Atualizar

```
model Empresa {
  // ... campos existentes ...
  cnpj      String? @unique
  telefone  String?
  email     String?
  logo      String?
  status    StatusEmpresa @default(ATIVA)
  planoId   String?

  // Configurações
  taxaEntrega Decimal? @db.Decimal(10, 2)
  tempoPreparoMin Int? @default(30)
  horarioAbertura String?
  horarioFechamento String?
  diasFuncionamento String[] // ["seg", "ter", "qua", ...]

  plano      Plano? @relation(fields: [planoId], references: [id])
  // ... relações existentes ...
}

model Plano {
  id          String @id @default(uuid())
  nome        String
  descricao   String?
  valor       Decimal @db.Decimal(10, 2)
  limiteUsuarios Int?
  limiteProdutos Int?
  limitesPedidosMes Int?
  recursos    String[] // ["relatorios", "integracao_whatsapp", ...]
  ativo       Boolean @default(true)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  empresas    Empresa[]
}

enum StatusEmpresa {
  ATIVA
  INATIVA
  SUSPENSA
  TRIAL
}
```


Endpoints a Implementar

```
// SuperAdmin - Gestão de Empresas
GET    /admin/empresas           // Listar empresas
POST   /admin/empresas           // Criar empresa
GET     /admin/empresas/:id     // Detalhes da empresa
PATCH /admin/empresas/:id     // Atualizar empresa
DELETE /admin/empresas/:id     // Desativar empresa
PATCH /admin/empresas/:id/status // Mudar status
GET     /admin/empresas/:id/stats // Estatísticas da empresa

// SuperAdmin - Gestão de Planos
GET     /admin/planos           // Listar planos
POST    /admin/planos           // Criar plano
PATCH  /admin/planos/:id       // Atualizar plano
DELETE  /admin/planos/:id       // Desativar plano
```

● ETAPA 2.6: Notificações (MÉDIA PRIORIDADE)

Complexidade: ★★☆☆ (Média-Alta)

Tempo Estimado: 2-3 dias

Dependências: Etapa 2.1 (Pedidos)

Objetivos

- Email transacional (confirmação, status)
- WhatsApp (via API oficial ou Twilio)
- Notificações in-app
- Templates personalizáveis

Modelos Prisma a Criar

```

model Notificacao {
  id          String          @id @default(uuid())
  tipo        TipoNotificacao
  canal        CanalNotificacao
  destinatario String
  assunto      String?
  mensagem     String          @db.Text
  status       StatusNotificacao @default(PENDENTE)
  tentativas   Int             @default(0)
  erro         String?
  pedidoId     String?
  empresaId    String
  createdAt    DateTime        @default(now())
  enviadoEm    DateTime?

  @@index([empresaId])
  @@index([status])
  @@index([pedidoId])
}

enum TipoNotificacao {
  PEDIDO_CONFIRMADO
  PEDIDO_EM_PREPARO
  PEDIDO_SAIU_ENTREGA
  PEDIDO_ENTREGUE
  PEDIDO_CANCELADO
  PAGAMENTO_APROVADO
  PAGAMENTO_RECUSADO
}

enum CanalNotificacao {
  EMAIL
  WHATSAPP
  SMS
  IN_APP
}

enum StatusNotificacao {
  PENDENTE
  ENVIADO
  FALHOU
  CANCELADO
}

```

Serviços de Integração

```
// email.service.ts
export class EmailService {
  async sendTransactional(to: string, template: string, data: any)
  async sendPedidoConfirmado(pedido: Pedido)
  async sendPedidoStatusChanged(pedido: Pedido)
}

// whatsapp.service.ts
export class WhatsAppService {
  async sendMessage(to: string, message: string)
  async sendTemplate(to: string, template: string, params: any)
  async sendPedidoUpdate(pedido: Pedido)
}
```

● ETAPA 2.7: Dashboard e Analytics (MÉDIA PRIORIDADE)

Complexidade: ★★★★★ (Média-Alta)

Tempo Estimado: 2-3 dias

Dependências: Etapa 2.1 (Pedidos)

Endpoints a Implementar

```
// Admin - Dashboard
GET /dashboard/stats // Estatísticas gerais
GET /dashboard/vendas // Vendas por período
GET /dashboard/produtos-top // Produtos mais vendidos
GET /dashboard/clientes-top // Clientes que mais compram
GET /dashboard/pedidos-recentes // Últimos pedidos
GET /dashboard/receita // Receita por período

// SuperAdmin - Dashboard Global
GET /admin/dashboard/global // Estatísticas de todas empresas
GET /admin/dashboard/empresas-top // Empresas com mais vendas
```

DTOs de Resposta

```
export class DashboardStatsDto {
  pedidosHoje: number;
  pedidosMes: number;
  receitaHoje: number;
  receitaMes: number;
  ticketMedio: number;
  clientesNovos: number;
  produtosAtivos: number;
  taxaConversao: number;
}

export class VendasPeriodoDto {
  periodo: string; // "2025-10-08"
  pedidos: number;
  receita: number;
  ticketMedio: number;
}
```

● ETAPA 2.8: Sistema de Assinaturas (BAIXA PRIORIDADE)

Complexidade: ★★★★★ (Alta)

Tempo Estimado: 3-4 dias

Dependências: Etapa 2.5 (Empresas)

Modelos Prisma a Criar

```

model Assinatura {
  id          String      @id @default(uuid())
  empresaId   String      @unique
  planoId     String
  status      StatusAssinatura @default(ATIVA)
  dataInicio  DateTime
  dataFim     DateTime?
  proximaCobranca DateTime
  valor       Decimal      @db.Decimal(10, 2)

  // Dados de pagamento recorrente
  gatewayAssinaturaId String?
  metodoPagamento MetodoPagamento

  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt

  empresa      Empresa    @relation(fields: [empresaId], references: [id])
  plano        Plano      @relation(fields: [planoId], references: [id])
  faturas      Fatura[]

  @@index([empresaId])
  @@index([status])
}

model Fatura {
  id          String      @id @default(uuid())
  assinaturaId String
  numero      String      @unique
  valor       Decimal      @db.Decimal(10, 2)
  status      StatusFatura @default(PENDENTE)
  dataVencimento DateTime
  dataPagamento DateTime?
  gatewayId   String?
  createdAt   DateTime      @default(now())

  assinatura    Assinatura @relation(fields: [assinaturaId], references: [id])

  @@index([assinaturaId])
  @@index([status])
}

enum StatusAssinatura {
  ATIVA
  CANCELADA
  SUSPENSA
  TRIAL
  INADIMPLENTE
}

enum StatusFatura {
  PENDENTE
  PAGA
  VENCIDA
  CANCELADA
}

```

ETAPA 2.9: Sistema de Tickets (BAIXA PRIORIDADE)

Complexidade: ★★☆☆ (Média-Alta)

Tempo Estimado: 2-3 dias

Dependências: Fase 1 completa

Modelos Prisma a Criar

```

model Ticket {
  id          String      @id @default(uuid())
  numero      String      @unique
  titulo      String
  descricao   String      @db.Text
  status      StatusTicket @default(ABERTO)
  prioridade  PrioridadeTicket @default(MEDIA)
  empresaId   String
  autorId     String
  atribuidoId String?
  createdAt   DateTime    @default(now())
  updatedAt   DateTime    @updatedAt
  resolvidoEm DateTime?

  empresa      Empresa      @relation(fields: [empresaId], references: [id])
  autor        Usuario      @relation("TicketsAutor", fields: [autorId], references: [id])
  atribuido     Usuario?    @relation("TicketsAtribuido", fields: [atribuidoId], references: [id])
  mensagens    MensagemTicket[]

  @@index([empresaId])
  @@index([status])
  @@index([prioridade])
}

model MensagemTicket {
  id          String      @id @default(uuid())
  ticketId    String
  autorId     String
  mensagem    String      @db.Text
  interno     Boolean      @default(false)
  createdAt   DateTime    @default(now())

  ticket      Ticket      @relation(fields: [ticketId], references: [id], onDelete: Cascade)
  autor       Usuario     @relation(fields: [autorId], references: [id])

  @@index([ticketId])
}

enum StatusTicket {
  ABERTO
  EM_ANDAMENTO
  AGUARDANDO_CLIENTE
  RESOLVIDO
  FECHADO
}

enum PrioridadeTicket {
  BAIXA
  MEDIA
  ALTA
  URGENTE
}

```

● ETAPA 2.10: Upload de Imagens (BAIXA PRIORIDADE)

Complexidade: ★★ (Média)

Tempo Estimado: 1-2 dias

Dependências: Fase 1 completa

Objetivos

- Upload de imagens de produtos
- Upload de logo da empresa
- Integração com Cloudinary ou AWS S3
- Redimensionamento automático
- Validação de tipo e tamanho

Endpoints a Implementar

POST	/upload/produto	// Upload imagem de produto
POST	/upload/empresa/logo	// Upload logo da empresa
DELETE	/upload/:id	// Remover imagem



Resumo de Complexidade e Tempo

Etapa	Funcionalidade	Prioridade	Complexidade	Tempo Estimado
2.1	Sistema de Pedidos	ALTA	★★★	2-3 dias
2.2	Checkout e Pagamentos	ALTA	★★★★★	3-4 dias
2.3	Gestão de Clientes	ALTA	★★	1-2 dias
2.4	Sistema de Cupons	MÉDIA	★★	1-2 dias
2.5	Gestão de Empresas	MÉDIA	★★★★	2-3 dias
2.6	Notificações	MÉDIA	★★★★	2-3 dias
2.7	Dashboard Analytics	MÉDIA	★★★★	2-3 dias
2.8	Sistema de Assinaturas	BAIXA	★★★★★	3-4 dias
2.9	Sistema de Tickets	BAIXA	★★★★	2-3 dias
2.10	Upload de Imagens	BAIXA	★★	1-2 dias

Tempo Total Estimado: 20-30 dias de desenvolvimento



Ordem de Implementação Recomendada



Sprint 1 (MVP Core) - 6-9 dias

- 1. **Etapa 2.1** - Sistema de Pedidos (2-3 dias)
- 2. **Etapa 2.2** - Checkout e Pagamentos (3-4 dias)
- 3. **Etapa 2.3** - Gestão de Clientes (1-2 dias)

Resultado: Sistema funcional para vendas online




Sprint 2 (Funcionalidades Comerciais) - 5-8 dias

- 1. **Etapa 2.4** - Sistema de Cupons (1-2 dias)

2.  **Etapa 2.6** - Notificações (2-3 dias)
3.  **Etapa 2.7** - Dashboard Analytics (2-3 dias)

Resultado: Sistema completo para operação comercial

Sprint 3 (Gestão Avançada) - 5-7 dias

1.  **Etapa 2.5** - Gestão de Empresas (2-3 dias)
2.  **Etapa 2.10** - Upload de Imagens (1-2 dias)
3.  **Etapa 2.9** - Sistema de Tickets (2-3 dias)

Resultado: Sistema com gestão completa

Sprint 4 (Monetização) - 3-4 dias

1.  **Etapa 2.8** - Sistema de Assinaturas (3-4 dias)

Resultado: Sistema pronto para escala



Integrações Necessárias

1. Asaas (Pagamentos)

- **Documentação:** <https://docs.asaas.com>
- **Recursos necessários:**
 - API Key (Sandbox e Produção)
 - Webhook URL configurada
 - Certificado SSL válido
- **Métodos a implementar:**
 - PIX (QR Code dinâmico)
 - Cartão de Crédito (tokenização)
 - Boleto bancário
 - Webhooks de status

2. Email (Transacional)

- **Opções:**
 - SendGrid (recomendado)
 - AWS SES
 - Mailgun
- **Templates necessários:**
 - Confirmação de pedido
 - Mudança de status
 - Pagamento aprovado/recusado
 - Boas-vindas

3. WhatsApp

- **Opções:**
 - WhatsApp Business API (oficial)
 - Twilio WhatsApp
 - Evolution API (self-hosted)

- **Mensagens:**

- Confirmação de pedido
- Status de entrega
- Promoções (opt-in)

4. Upload de Imagens

- **Opções:**

- Cloudinary (recomendado)
- AWS S3 + CloudFront
- Supabase Storage

- **Configurações:**

- Redimensionamento automático
- Compressão
- CDN



Checklist de Implementação

Antes de Começar

- ☐ Revisar e aprovar este planejamento
- ☐ Configurar conta Asaas (Sandbox)
- ☐ Configurar serviço de email
- ☐ Configurar serviço de upload
- ☐ Definir prioridades finais

Durante o Desenvolvimento

- ☐ Criar migrations do Prisma
- ☐ Implementar DTOs com validações
- ☐ Criar services com lógica de negócio
- ☐ Implementar controllers
- ☐ Adicionar testes unitários
- ☐ Adicionar testes de integração
- ☐ Documentar endpoints (Swagger)
- ☐ Testar fluxos completos

Após Cada Etapa

- ☐ Code review
 - ☐ Testes manuais
 - ☐ Atualizar documentação
 - ☐ Deploy em staging
 - ☐ Validação com stakeholders
-

Riscos e Mitigações

Risco 1: Integração com Asaas

Impacto: Alto

Probabilidade: Média

Mitigação:

- Usar ambiente sandbox primeiro
- Implementar retry logic
- Logs detalhados de erros
- Fallback para pagamento manual

Risco 2: Performance com Muitos Pedidos

Impacto: Alto

Probabilidade: Média

Mitigação:

- Índices otimizados no banco
- Cache com Redis
- Paginação em todas as listagens
- Background jobs para processamento pesado

Risco 3: Webhooks Não Recebidos

Impacto: Alto

Probabilidade: Baixa

Mitigação:

- Polling de status como backup
- Retry automático de webhooks
- Logs de todos os webhooks recebidos
- Alertas de webhooks falhados

Risco 4: Estoque Negativo

Impacto: Médio

Probabilidade: Média

Mitigação:

- Validação de estoque antes de criar pedido
- Transações atômicas no banco
- Reserva de estoque ao criar pedido
- Devolução de estoque ao cancelar

Recursos e Referências

Documentação Técnica

- [NestJS Docs](https://docs.nestjs.com) (<https://docs.nestjs.com>)
- [Prisma Docs](https://www.prisma.io/docs) (<https://www.prisma.io/docs>)
- [Asaas API](https://docs.asaas.com) (<https://docs.asaas.com>)
- [Supabase Docs](https://supabase.com/docs) (<https://supabase.com/docs>)

Padrões e Boas Práticas

- [REST API Best Practices](https://restfulapi.net) (<https://restfulapi.net>)
- [NestJS Best Practices](https://github.com/nestjs/nest/tree/master/sample) (<https://github.com/nestjs/nest/tree/master/sample>)
- [Prisma Best Practices](https://www.prisma.io/docs/guides/performance-and-optimization) (<https://www.prisma.io/docs/guides/performance-and-optimization>)

Segurança

- [OWASP Top 10](https://owasp.org/www-project-top-ten/) (<https://owasp.org/www-project-top-ten/>)
- [JWT Best Practices](https://tools.ietf.org/html/rfc8725) (<https://tools.ietf.org/html/rfc8725>)
- [PCI DSS Compliance](https://www.pcisecuritystandards.org) (<https://www.pcisecuritystandards.org>)



Próximos Passos

1. **Revisar este planejamento** com a equipe
2. **Aprovar prioridades** e ordem de implementação
3. **Configurar integrações** (Asaas, Email, etc)
4. **Iniciar Sprint 1** com Etapa 2.1 (Sistema de Pedidos)
5. **Manter comunicação** constante sobre progresso



Contato e Suporte

Para dúvidas ou sugestões sobre este planejamento:

- Criar issue no repositório
- Discussão no canal do projeto
- Reunião de alinhamento

Documento criado em: 2025-10-08

Versão: 1.0

Status: Aguardando Aprovação

Próxima Revisão: Após aprovação e início da implementação



Vamos construir um sistema incrível! 🚀