🏗️ ARQUITETURA DO SISTEMA DE PEDIDOS

📋 VISÃO GERAL

Este documento descreve a organização completa do projeto: estrutura de pastas, arquivos e a razão de cada um existir. É como um mapa detalhado que mostra onde cada peça fica e por que ela está lá.

🗂️ ESTRUTURA COMPLETA DO PROJETO

```
projeto-pedidos/
│
├── backend/
│   ├── src/
│   │   ├── config/
│   │   │   ├── database.ts
│   │   │   └── environment.ts
│   │   │
│   │   ├── controllers/
│   │   │   ├── authController.ts
│   │   │   ├── pedidosController.ts
│   │   │   ├── usuariosController.ts
│   │   │   ├── dashboardController.ts
│   │   │   └── relatoriosController.ts
│   │   │
│   │   ├── middlewares/
│   │   │   ├── auth.ts
│   │   │   ├── permissions.ts
│   │   │   ├── validation.ts
│   │   │   └── errorHandler.ts
│   │   │
│   │   ├── services/
│   │   │   ├── statusService.ts
│   │   │   ├── notificationService.ts
│   │   │   ├── authService.ts
│   │   │   └── relatoriosService.ts
│   │   │
│   │   ├── jobs/
│   │   │   └── atrasoJob.ts
│   │   │
│   │   ├── routes/
│   │   │   ├── auth.ts
│   │   │   ├── pedidos.ts
│   │   │   ├── usuarios.ts
│   │   │   ├── dashboard.ts
│   │   │   ├── relatorios.ts
│   │   │   └── index.ts
│   │   │
│   │   ├── types/
│   │   │   ├── index.ts
```

```
│   │   │   ├── api.ts
│   │   │   └── database.ts
│   │   │
│   │   ├── utils/
│   │   │   ├── formatters.ts
│   │   │   ├── validators.ts
│   │   │   └── errors.ts
│   │   │
│   │   └── server.ts
│   │
│   ├── migrations/
│   │   ├── 20260107000001_create_usuarios.ts
│   │   ├── 20260107000002_create_pedidos.ts
│   │   └── 20260107000003_create_pedidos_status_log.ts
│   │
│   ├── seeds/
│   │   └── 01_admin_inicial.ts
│   │
│   ├── .env
│   ├── .env.example
│   ├── .gitignore
│   ├── knexfile.ts
│   ├── package.json
│   ├── package-lock.json
│   ├── tsconfig.json
│   └── README.md
│
└── frontend/
    ├── assets/
    │   ├── css/
    │   │   ├── reset.css
    │   │   ├── variables.css
    │   │   ├── global.css
    │   │   ├── components.css
    │   │   ├── pages.css
    │   │   └── responsive.css
    │   │
    │   ├── js/
    │   │   ├── api.js
    │   │   ├── auth.js
    │   │   ├── components.js
    │   │   ├── utils.js
    │   │   └── validation.js
    │   │
    │   └── images/
    │       ├── logo.png
    │       └── icons/
    │
    ├── pages/
    │   ├── auth/
    │   │   ├── login.html
    │   │   └── register.html
    │   │
    │   ├── cliente/
```

```
│   │   ├── meus-pedidos.html
│   │   └── minhas-entregas.html
│   │
│   ├── colaborador/
│   │   ├── dashboard.html
│   │   ├── pedidos-pendentes.html
│   │   ├── meus-pedidos.html
│   │   └── finalizados.html
│   │
│   ├── admin/
│   │   ├── dashboard.html
│   │   ├── todos-pedidos.html
│   │   ├── gestao-clientes.html
│   │   ├── gestao-equipe.html
│   │   └── relatorios.html
│   │
│   └── shared/
│       └── perfil.html
│
├── index.html
└── README.md
```

BACKEND - DETALHAMENTO

src/config/

database.ts

- Configuração da conexão com MySQL via Knex
- Define pool de conexões (quantas conexões simultâneas)
- Exporta instância do Knex para usar em todo código

environment.ts

- Carrega variáveis do arquivo .env
- Valida se variáveis obrigatórias existem
- Exporta constantes (PORT, JWT_SECRET, DATABASE_URL)

Por que separar?

- Configurações centralizadas
- Fácil trocar entre dev/produção
- Um lugar só para ajustar

---

src/controllers/

authController.ts

- Função: login()
- Função: register()

- Função: logout()
- Função: me() (retorna dados do usuário logado)

pedidosController.ts

- Função: listar()
- Função: buscarPorId()
- Função: criar()
- Função: assumir()
- Função: concluir()
- Função: cancelar()
- Função: buscarHistorico()

usuariosController.ts

- Função: listar() (apenas admin)
- Função: atualizar() (ativo, nivel_acesso)
- Função: desativar()

dashboardController.ts

- Função: estatisticasPessoais()
- Função: estatisticasGlobais() (apenas admin)
- Função: proximasEntregas()
- Função: pedidosAtrasados()

relatoriosController.ts

- Função: rankingProdutividade()
- Função: taxaCancelamento()
- Função: tempoMedioEntrega()
- Função: analiseAtrasos()
- Função: motivosCancelamento()

Por que um arquivo por domínio?

- Fica fácil achar onde está cada endpoint
- Cada arquivo cuida de uma área (pedidos, usuários, etc)
- Código organizado e escalável

🗁 src/middlewares/

auth.ts

- Verifica se token JWT existe no header
- Valida se token é válido
- Decodifica e anexa dados do usuário na requisição
- Bloqueia se token inválido/expirado

permissions.ts

- Verifica nível de acesso do usuário
- Exporta funções: apenasCliente(), apenasColaborador(), apenasAdmin()
- Bloqueia acesso se nível insuficiente

validation.ts

- Valida dados de entrada (body, query, params)
- Verifica campos obrigatórios, tipos, formatos
- Retorna erro descritivo se dados inválidos

errorHandler.ts

- Captura erros não tratados
- Formata resposta de erro padronizada
- Loga erros no console/arquivo
- Esconde detalhes internos do usuário

Por que middlewares?

- Código reutilizável (não repete validações)
- Proteção automática antes do controller
- Fácil adicionar novas validações

🗁 src/services/

statusService.ts

- Função central: mudarStatusPedido()
- Gerencia transação (tudo ou nada)
- Atualiza pedido + log em uma operação
- Único ponto que muda status (segurança)

notificationService.ts

- Função: criar()
- Função: marcarComoLida()
- Função: buscarNaoLidas()
- Função: buscarTodas()

authService.ts

- Função: verificarSenha()
- Função: gerarToken()
- Função: hashearSenha()

relatoriosService.ts

- Função: calcularEstatisticas()
- Função: buscarDadosRelatorio()
- Processa dados complexos para relatórios

Por que services?

- Lógica de negócio centralizada
- Reutilizável por vários controllers
- Facilita testes unitários
- Uma fonte de verdade

🗁 src/jobs/

atrasoJob.ts

- Agenda execução diária (00:00)
- Busca pedidos em andamento com prazo vencido
- Chama statusService para marcar como atrasado
- Registra logs de execução

Por que separar jobs?

- Tarefas automáticas isoladas
- Fácil desabilitar temporariamente
- Logs independentes
- Não depende de usuário

🗁 src/routes/

auth.ts

- POST /login
- POST /register
- POST /logout
- GET /me

pedidos.ts

- GET / (lista pedidos)
- GET /:id (detalhes)
- POST / (criar)
- POST /:id/assumir
- POST /:id/concluir
- POST /:id/cancelar
- GET /:id/historico

usuarios.ts

- GET / (listar usuários - admin)
- PATCH /:id (atualizar usuário - admin)

dashboard.ts

- GET /stats (estatísticas)

- GET /avisos (alertas)

relatorios.ts

- GET /produtividade
- GET /cancelamentos
- GET /tempo-entrega
- GET /atrasos
- GET /motivos-cancelamento

index.ts

- Importa todas as rotas
- Agrupa em /api
- Exporta router principal

Por que separar rotas?

- Organização por feature
- Middlewares específicos por grupo
- Documentação implícita (arquivo = endpoints)

🗁 src/types/

index.ts

- Type: Usuario
- Type: Pedido
- Type: PedidoStatusLog
- Type: Notificacao
- Enum: NivelAcesso
- Enum: StatusPedido
- Enum: Prioridade

api.ts

- Type: LoginRequest
- Type: LoginResponse
- Type: CreatePedidoRequest
- Type: PedidoResponse
- Tipos de requisições e respostas da API

database.ts

- Type: UsuariosTable
- Type: PedidosTable
- Type: PedidosStatusLogTable
- Representa estrutura exata do banco

Por que types separados?

- TypeScript valida tipos em tempo de desenvolvimento
- Autocomplete no editor
- Previne erros de tipo
- Documentação viva

🗁 src/utils/

formatters.ts

- Função: formatarData()
- Função: formatarMoeda()
- Função: formatarTelefone()
- Funções de formatação de dados

validators.ts

- Função: validarEmail()
- Função: validarSenha()
- Função: validarCPF()
- Funções de validação

errors.ts

- Class: AppError (erros customizados)
- Class: UnauthorizedError
- Class: ForbiddenError
- Class: NotFoundError

Por que utils?

- Funções pequenas e reutilizáveis
- Não repete código
- Fácil testar isoladamente

🗎 src/server.ts

Responsabilidades:

- Importa Express
- Configura middlewares globais (cors, json)
- Registra todas as rotas
- Inicia jobs agendados
- Conecta no banco
- Sobe servidor na porta configurada
- Trata erros globais

Por que é o arquivo principal?

- Ponto de entrada único

- Orquestra todo o backend
- Centraliza inicialização

🗁 migrations/

20260107000001_create_usuarios.ts

- Cria tabela usuarios
- Define colunas, tipos, constraints
- Função up() cria, down() remove

20260107000002_create_pedidos.ts

- Cria tabela pedidos
- Define foreign keys
- Índices para performance

20260107000003_create_pedidos_status_log.ts

- Cria tabela de histórico
- Foreign keys para pedidos e usuários

Por que migrations?

- Versionamento do banco de dados
- Histórico de mudanças
- Aplicação automática (CI/CD)
- Rollback se necessário

🗁 seeds/

01_admin_inicial.ts

- Cria usuário admin padrão
- Email: admin@example.com
- Senha: admin123 (criptografada)
- Para facilitar primeiro acesso

Por que seeds?

- Dados iniciais para desenvolvimento
- Não precisa cadastrar admin manualmente
- Facilita testes

🗎 Arquivos na raiz do backend/

.env

- Variáveis de ambiente secretas

- DATABASE_URL, JWT_SECRET, PORT
- NÃO VAI PRO GIT (gitignore)
- Cada desenvolvedor tem o seu

.env.example

- Modelo do arquivo .env
- Mostra quais variáveis são necessárias
- VAI PRO GIT (sem valores reais)
- Outros devs copiam e preenchem

.gitignore

- Lista arquivos/pastas que não vão pro Git
- node_modules/, .env, dist/, logs/
- Evita enviar código compilado e senhas

knexfile.ts

- Configuração do Knex
- Define ambientes (development, production)
- Caminho das migrations e seeds
- Conexão com banco por ambiente

package.json

- Lista dependências (express, knex, bcrypt, etc)
- Scripts de desenvolvimento:
  - `npm run dev` → Roda servidor em desenvolvimento
  - `npm run build` → Compila TypeScript
  - `npm run migrate` → Roda migrations
  - `npm run seed` → Roda seeds
- Metadados do projeto (nome, versão, autor)

package-lock.json

- Versões exatas das dependências instaladas
- Gerado automaticamente pelo npm
- Garante mesmas versões em todos ambientes
- VAI PRO GIT

tsconfig.json

- Configuração do TypeScript
- Define como compilar (ES6, CommonJS)
- Onde procurar arquivos (.ts)
- Onde salvar compilados (dist/)
- Opções de checagem de tipos (strict)

README.md

- Documentação do backend
- Como instalar dependências
- Como rodar migrations
- Como iniciar servidor
- Estrutura de pastas resumida

FRONTEND - DETALHAMENTO

assets/css/

reset.css

- Remove estilos padrão do navegador
- Margin, padding zerados
- Box-sizing consistente
- Todos os navegadores começam igual

variables.css

- Define variáveis CSS (--primary-color, --font-size, etc)
- Cores do sistema
- Tamanhos de fonte
- Espaçamentos padrão
- Muda uma variável, afeta tudo

global.css

- Estilos globais (body, html)
- Fontes padrão
- Links, títulos, parágrafos
- Classes utilitárias (.container, .btn)

components.css

- Estilos de componentes reutilizáveis
- .card, .modal, .badge, .notification
- Usado em várias páginas
- Consistência visual

pages.css

- Estilos específicos de páginas
- .dashboard-grid, .pedidos-list
- Layouts únicos de cada tela

responsive.css

- Media queries para mobile/tablet
- @media (max-width: 768px)
- Adaptações de layout

- Menu hamburger, cards empilhados

Por que separar CSS assim?

- Organização modular
- Fácil manutenção
- Reutilização de estilos
- Performance (carrega só o necessário)

🗁 assets/js/

api.js

- Funções que fazem fetch para backend
- buscarPedidos(), criarPedido(), assumirPedido()
- Adiciona token no header automaticamente
- Trata erros de rede
- Retorna dados JSON

auth.js

- Gerencia autenticação
- salvarToken(), getToken(), removerToken()
- estaLogado(), getUsuarioAtual()
- redirecionar se não logado
- Armazena token no localStorage

components.js

- Funções que criam elementos HTML dinamicamente
- criarCardPedido(dados)
- criarModal(titulo, conteudo)
- mostrarNotificacao(mensagem, tipo)
- Componentes reutilizáveis em JS

utils.js

- Funções auxiliares
- formatarData(data)
- formatarMoeda(valor)
- calcularDiasRestantes(prazo)
- debounce(funcao, tempo)

validation.js

- Valida formulários no cliente
- validarEmail(input)
- validarCamposObrigatorios(form)
- mostrarErro(campo, mensagem)
- Feedback visual de erros

Por que separar JS assim?

- Organização por responsabilidade
- Reutilização de código
- Fácil importar só o necessário

🗁 assets/images/

logo.png

- Logo da empresa
- Aparece no header, login

icons/

- Ícones SVG ou PNG
- icon-pedido.svg
- icon-usuario.svg
- icon-notificacao.svg

Por que pasta de imagens?

- Centraliza assets visuais
- Fácil encontrar e atualizar
- Cache do navegador

🗁 pages/

Organização por nível de acesso:

auth/login.html

- Formulário de login
- Email, senha, botão entrar
- Link para registro

auth/register.html

- Formulário de cadastro
- Nome, email, senha, confirmar senha
- Cria conta como cliente

cliente/meus-pedidos.html

- Lista pedidos do cliente (pendente, andamento, atrasado)
- Botão criar novo pedido
- Botão cancelar pedido

cliente/minhas-entregas.html

- Lista pedidos entregues e cancelados

- Visualização apenas

colaborador/dashboard.html

- Estatísticas pessoais
- Gráficos de pedidos
- Próximas entregas
- Pedidos atrasados

colaborador/pedidos-pendentes.html

- Lista todos pendentes
- Botão assumir pedido
- Filtros por tipo/prioridade

colaborador/meus-pedidos.html

- Pedidos que assumiu (andamento, atrasado)
- Botão concluir
- Botão cancelar

colaborador/finalizados.html

- Pedidos entregues/cancelados por ele
- Histórico visual

admin/dashboard.html

- Estatísticas pessoais + globais
- Produtividade da equipe
- Alertas do sistema

admin/todos-pedidos.html

- Lista TODOS os pedidos
- Filtros avançados
- Editar qualquer pedido
- Ver histórico de qualquer pedido

admin/gestao-clientes.html

- Lista clientes
- Editar ativo, nivel_acesso
- Pesquisa por nome/email

admin/gestao-equipe.html

- Lista colaboradores e admins
- Editar ativo, nivel_acesso
- Avisos de inatividade

admin/relatorios.html

- Ranking produtividade
- Taxa cancelamento
- Tempo médio entrega
- Análise atrasos
- Motivos cancelamento

shared/perfil.html

- Editar nome, email, senha
- Ver nível de acesso
- Botão logout

Por que organizar por pastas de acesso?

- Estrutura clara e documentada
- Fácil aplicar proteções (middleware)
- Navegação intuitiva

📄 index.html

Responsabilidades:

- Página inicial (geralmente redireciona)
- Se logado → dashboard
- Se não logado → login
- Carrega CSS e JS globais
- Define estrutura HTML base

Por que index.html?

- Ponto de entrada padrão do navegador
- Servidor web busca index por padrão

📄 README.md (frontend)

Conteúdo:

- Como abrir o projeto
- Estrutura de pastas
- Como conectar com backend
- Páginas disponíveis
- Variáveis de configuração (API_URL)

🔄 FLUXO DE COMUNICAÇÃO

```
NAVEGADOR
    ↓
```

```
index.html (carrega CSS/JS)
   ↓
auth.js verifica se está logado
   ↓
   SIM → dashboard.html
   NÃO → login.html
   ↓
Usuário faz ação (assumir pedido)
   ↓
components.js captura clique
   ↓
api.js faz fetch para backend
   ↓
auth.js adiciona token
   ↓
BACKEND recebe requisição
   ↓
routes → middlewares → controller → service → database
   ↓
Resposta JSON
   ↓
api.js recebe
   ↓
components.js atualiza DOM
   ↓
utils.js formata dados
   ↓
Usuário vê resultado na tela
```

🎯 PRINCÍPIOS ARQUITETURAIS

Modularidade

- Cada arquivo tem uma responsabilidade única
- Fácil encontrar onde mexer
- Código escalável

Separação Frontend/Backend

- Desenvolvimento independente
- Backend serve API (JSON)
- Frontend consome API
- Pode trocar frontend sem tocar backend

Convenção de Nomenclatura

- Arquivos em camelCase: authController.ts
- Pastas em lowercase: controllers/, middlewares/
- Componentes descritivos: criarCardPedido()
- Rotas RESTful: GET /pedidos, POST /pedidos/:id/assumir

Single Source of Truth

- Configurações: config/
- Tipos: types/
- Estilos: assets/css/
- Lógica de negócio: services/

Don't Repeat Yourself (DRY)

- Código repetido vira função (utils/)
- Estilos repetidos viram classes (components.css)
- Componentes HTML viram funções (components.js)

📊 ESCALABILIDADE

Adicionar nova funcionalidade:

```
1. Backend:
   - Criar migration (se precisar nova tabela/coluna)
   - Adicionar types em types/
   - Criar service (lógica de negócio)
   - Criar controller (endpoints)
   - Adicionar rotas em routes/

2. Frontend:
   - Criar página HTML em pages/
   - Adicionar estilos em pages.css
   - Criar funções em api.js
   - Conectar com backend
```

Adicionar novo desenvolvedor:

```
1. Clonar repositório
2. Copiar .env.example para .env
3. Instalar dependências: npm install
4. Rodar migrations: npm run migrate
5. Rodar seeds: npm run seed
6. Iniciar servidor: npm run dev
7. Abrir frontend: index.html
```

🎓 RESUMO

Por que essa arquitetura?

✅ **Organização clara:** Sabe onde cada coisa está
✅ **Escalável:** Fácil adicionar funcionalidades
✅ **Manutenível:** Mudanças localizadas
✅ **Testável:** Cada parte independente
✅ **Documentada:** Estrutura autoexplicativa
✅ **Profissional:** Padrões de mercado

Arquivos críticos:

Backend:

- `server.ts` → Inicializa tudo
- `database.ts` → Conexão com banco
- `statusService.ts` → Lógica central de status
- `knexfile.ts` → Configuração do Knex
- `.env` → Segredos e configurações

Frontend:

- `index.html` → Ponto de entrada
- `auth.js` → Gerencia sessão
- `api.js` → Comunicação com backend
- `variables.css` → Design system
- `components.js` → Componentes reutilizáveis