Documentação Técnica Completa - Pass Finance

Sistema moderno de gestão financeira desenvolvido com Next.js 16, React 19, TypeScript, Tailwind CSS, Prisma ORM e PostgreSQL.

Índice

Visão Geral da Arquitetura

O Pass Finance é um sistema **full-stack** que utiliza o framework **Next.js 16** com o paradigma **App Router**, combinando renderização server-side (SSR) e client-side (CSR). A arquitetura segue o padrão **MVC** adaptado para React, com separação clara entre camadas de apresentação, lógica de negócio e acesso a dados.

Princípios Arquiteturais

- **Componentização:** Interface dividida em componentes reutilizáveis e isolados
- **Single Source of Truth:** Estado gerenciado via React Context API
- **Type Safety:** TypeScript em todo o codebase para segurança de tipos
- **API RESTful:** Rotas de API para comunicação backend/frontend
- **Database-First:** Schema Prisma como fonte de verdade dos dados

---

Stack Tecnológica

Frontend

- **Next.js 16.0.1** - Framework React com SSR/SSG

- **React 19.2.0** - Biblioteca UI com hooks modernos
- **TypeScript 5** - Superset tipado do JavaScript
- **Tailwind CSS 4** - Framework CSS utility-first
- **Shadcn/UI** - Componentes UI acessíveis e customizáveis
- **Radix UI** - Primitives para componentes complexos
- **Lucide React** - Biblioteca de ícones
- **PapaParse** - Parser CSV para exportação de dados

Backend

- **Prisma ORM 6.1.0** - ORM TypeScript-first para PostgreSQL
- **PostgreSQL** - Banco de dados relacional
- **Zod** - Validação de schemas e dados
- **Next.js API Routes** - Endpoints serverless

DevTools

- **ESLint 9** - Linter para qualidade de código
- **PostCSS** - Processador CSS
- **TypeScript Compiler** - Compilação e type-checking

Estrutura de Diretórios

```
pass-finance/
  prisma/
     migrations/          # Histórico de migrações do banco
     schema.prisma        # Schema do banco de dados
  public/
     *.svg                # Assets públicos (ícones, imagens)
  src/
     app/
        favicon.ico       # Favicon do site
        globals.css       # Estilos globais
        layout.tsx        # Layout raiz da aplicação
        page.tsx          # Página principal (Home)
     components/
        ui/               # Componentes UI base (Shadcn)
        account-payable-modal.tsx
        add-payment-modal.tsx
        analytics-modal.tsx
        delete-confirmation-dialog.tsx
        export-button.tsx
        providers.tsx     # Providers de contexto
        sidebar.tsx
     contexts/
```

```
        language-context.tsx
        sidebar-context.tsx
        theme-context.tsx
    hooks/
        use-bills.ts     # Hook para buscar contas
    lib/
        prisma.ts        # Cliente Prisma singleton
        schemas/         # Schemas Zod de validação
            contaPagar.schema.ts
            pagamento.schema.ts
        utils.ts         # Utilitários gerais
    pages/
        api/             # API Routes
            analytics/
                index.ts
            contas/
                index.ts
                [id].ts
                [id]/
                    pagamentos.ts
            pagamentos/
                [pagamentoId].ts
            export-data.ts
    types/
        bill.ts          # Tipos TypeScript da aplicação
.env                     # Variáveis de ambiente
.gitignore
components.json          # Configuração Shadcn
eslint.config.mjs
next.config.ts
package.json
postcss.config.mjs
tsconfig.json
README.md
```

Backend & Banco de Dados

Schema Prisma

O banco de dados possui dois modelos principais: **ContaPagar** e **Pagamento**, com relacionamento 1:N.

Campos Principais: - **ContaPagar:** conta, credor, devedor, competência, vencimento, valor, status - **Pagamento:** total, caixa, tipo, dataPagamento

Recursos: - Enum `StatusConta` (PENDENTE, PAGO, ATRASADO, CAN-

CELADO) - Cálculo automático de `total`, `valorPago` e `saldo` - Cascata de exclusão (deletar conta remove pagamentos) - Timestamps automáticos (`criadoEm`, `atualizadoEm`)

API Endpoints

1. GET /api/contas?page=1&limit=10 Lista contas com paginação.

Resposta:

```
{
  "data": [...],
  "pagination": {
    "page": 1,
    "limit": 10,
    "total": 45,
    "totalPages": 5
  }
}
```

2. POST /api/contas Cria nova conta a pagar.

Payload:

```
{
  "conta": "000123",
  "lancamento": "2025-01-15T00:00:00.000Z",
  "credor": "Fornecedor ABC",
  "devedor": "Empresa XYZ",
  "competencia": "2025-01-15",
  "vencimento": "2025-02-15",
  "valor": 1000,
  "desconto": 50,
  "juros": 10,
  "classificacaoGerencial": "Operacional"
}
```

3. GET /api/contas/[id] Busca conta específica com pagamentos relacionados.

4. PUT /api/contas/[id] Atualiza dados da conta.

5. DELETE /api/contas/[id] Deleta conta e pagamentos em cascata.

6. POST /api/contas/[id]/pagamentos Cria pagamento e atualiza saldo da conta automaticamente.

7. DELETE /api/pagamentos/[id] Deleta pagamento e reverte valores da conta.

8. GET /api/analytics Retorna dados agregados para análise: - Total pendente e pago - Gastos por classificação - Contagem por status

9. GET /api/export-data Exporta todos os dados formatados para CSV.

Validação com Zod

Todos os endpoints utilizam validação Zod server-side para garantir integridade dos dados:

```
// Exemplo: Criar Conta
export const criarContaPagarSchema = z.object({
  conta: z.string().min(1),
  lancamento: z.string().datetime(),
  credor: z.string().min(1),
  devedor: z.string().min(1),
  valor: z.number().positive(),
  // ...
});
```

Frontend & Componentes

Componentes Principais

1. Layout Principal Define estrutura da aplicação com Sidebar e área de conteúdo.

2. Página Principal (Home) Componente que orquestra toda a interface: - Header com busca, tema, idioma - Toolbar com filtros e ações - Tabela com dados paginados - Modais de cadastro e edição

3. AccountPayableModal Modal completo com 4 abas: - Dados Gerais - Dados Financeiros - Pagamentos - Adicionais (arquivos e notas)

Funcionalidades: - Edição inline - Validação client-side - Integração com API - Exclusão com confirmação

4. AddPaymentModal Modal para cadastro de novas contas: - Formulário completo - Sistema de parcelamento - Cálculo automático de totais - Validação de campos obrigatórios

5. AnalyticsModal Visualização de dados analíticos com gráficos.

6. ExportButton Exporta dados para CSV usando PapaParse: - Formatação em português - Encoding UTF-8 com BOM - Download automático

7. DeleteConfirmationDialog Dialog reutilizável para confirmação de exclusões.

8. Sidebar Navegação lateral com menu de opções.

Contextos & Estado Global

1. ThemeContext

Gerencia tema (light/dark) com persistência no localStorage.

Funcionalidades: - Toggle de tema - Persistência de preferência - Classe CSS no `<html>`

2. LanguageContext

Internacionalização com suporte a PT/EN/ES.

Funcionalidades: - Função `t()` para tradução - Mudança dinâmica de idioma - Traduções organizadas por chave

3. SidebarContext

Controla estado da sidebar (aberta/fechada).

Hooks Customizados

useBills

Hook principal para gerenciar dados de contas:

Funcionalidades: - Busca paginada de contas - Mapeamento ContaPagar → Bill - Estados de loading e erro - Função refetch para atualizar - Cálculo de dias para vencimento

Retorno:

```
{
  bills: Bill[],
  loading: boolean,
  error: string | null,
  pagination: {...},
  refetch: () => Promise<void>
}
```

Tipos TypeScript

Bill (Interface Principal)

```typescript
export interface Bill {
  id: string;
  code: string;
  competenceDate: string;
  dueDate: string;
  paymentInfo: string;
  status: 'Pendente' | 'Pago' | 'Vencido' | 'Cancelado';
  classification: {
    code?: string;
    description: string;
  };
  participants: {
    name: string;
    secondary?: string;
  };
  installment: string;
  amount: number;
  details?: {...};
}
```

Funcionalidades Implementadas

Gestão de Contas

- Listagem com paginação
- Cadastro completo
- Edição inline
- Exclusão com confirmação
- Filtros avançados
- Busca por ID ou participante

Pagamentos

- Registro de pagamentos
- Atualização automática de saldo
- Histórico de pagamentos
- Exclusão com reversão

Parcelamento

- Cadastro parcelado
- Cálculo automático por parcela
- Controle de parcelas

Análise

- Dashboard com gráficos
- Totais pendente e pago
- Gastos por classificação
- Contagem por status

Exportação

- Export CSV
- Formatação em português
- UTF-8 com BOM (Excel)

Interface

- Dark Mode
- Internacionalização
- Design responsivo
- Loading states
- Validação de formulários
- Feedback visual

Fluxo de Dados

Fluxo de Listagem

```
Usuário → page.tsx → useBills → GET /api/contas
→ Prisma → PostgreSQL → Mapeia dados → Renderiza
```

Fluxo de Criação

```
Usuário → AddPaymentModal → Validação Zod
→ POST /api/contas → Prisma INSERT → refetch()
```

Fluxo de Edição

```
Usuário → Clica linha → AccountPayableModal
→ GET /api/contas/[id] → Edita → PUT /api/contas/[id]
```

Fluxo de Pagamento

```
Usuário → Adicionar Pagamento → POST /api/contas/[id]/pagamentos
→ Transação Prisma (INSERT + UPDATE) → Commit
```

Boas Práticas

TypeScript

- Tipagem estrita
- Interfaces bem definidas
- Enums para valores fixos
- Type guards

React

- Componentes funcionais com hooks
- Custom hooks para lógica reutilizável
- Context API para estado global
- Memoização para otimização

Backend

- API RESTful consistente
- Validação com Zod
- Tratamento de erros
- Transações para operações críticas
- Prisma Client singleton

Segurança

- Validação server-side obrigatória
- Sanitização de inputs
- Variáveis de ambiente para secrets
- Queries parametrizadas (Prisma)

Performance

- Paginação no backend
- Lazy loading de modais
- Otimização de re-renders
- Code splitting automático

Código Limpo

- Nomes descritivos
- Funções pequenas e focadas

- Separação de responsabilidades
- Formatação consistente

Configuração e Deploy

Variáveis de Ambiente

```
DATABASE_URL="postgresql://user:password@localhost:5432/passfinance"
NEXT_PUBLIC_API_URL="http://localhost:3000"
```

Variável para viabilidade de Migrate

```
Postgres_URL_non_pooling = "postgresql://postgres:NovaSenhaDB@db.plereohpxdjyneuxgcuz.supaba
```

Variável para viabilidade de deploy via Vercel

```
DATABASE_URL = "postgresql://postgres.plereohpxdjyneuxgcuz:NovaSenhaDB@aws-1-us-east-2.poole
```

Scripts

```
# Desenvolvimento
npm run dev

# Build
npm run build
npm run start

# Banco de Dados
npx prisma generate
npx prisma migrate dev
npx prisma migrate deploy
npx prisma studio

# Linting
npm run lint
```

Deploy (Vercel)

1. Configurar variáveis de ambiente
2. Build Command: `npm run build`
3. Install Command: `npm install && npx prisma generate`
4. Aplicar migrations: `npx prisma migrate deploy`

Bancos recomendados: Neon, Supabase, Railway

Métricas do Projeto

- **Linhas de Código:** ~3.500
- **Componentes React:** 15+
- **API Endpoints:** 9
- **Contextos:** 3
- **Hooks Customizados:** 1
- **Schemas Zod:** 2
- **Modelos Prisma:** 2

Roadmap Futuro

Funcionalidades Planejadas

- ☐ Autenticação de usuários
- ☐ Permissões e roles
- ☐ Contas a Receber
- ☐ Relatórios em PDF
- ☐ Anexos de arquivos
- ☐ Notificações de vencimento
- ☐ Dashboard interativo
- ☐ Histórico de alterações
- ☐ Importação de CSV/Excel
- ☐ Integração bancária

Melhorias Técnicas

- ☐ Testes unitários
- ☐ Testes E2E
- ☐ Storybook
- ☐ CI/CD
- ☐ Monitoring
- ☐ Analytics
- ☐ Rate limiting
- ☐ Cache

Conclusão

O **Pass Finance** é um sistema completo, moderno e escalável para gestão financeira. A arquitetura bem definida, uso de tecnologias de ponta e boas práticas garantem um código de qualidade, fácil manutenção e pronto para evolução.

A separação clara entre frontend e backend, o uso de TypeScript para segurança de tipos, a validação robusta com Zod e o ORM Prisma para acesso ao banco

de dados criam uma base sólida para um sistema empresarial confiável.

Desenvolvido com por Manuel Sereno | © 2025 PASS