Documentação Técnica - Sistema RM Papel

Arquitetura do Sistema

Visão Geral

O Sistema RM Papel é uma aplicação web full-stack desenvolvida com arquitetura moderna e escalável:

- **Frontend**: React.js com Vite
- **Backend**: Flask (Python) com APIs REST
- **Banco de Dados**: PostgreSQL (produção) / SQLite (desenvolvimento)
- **Autenticação**: JWT (JSON Web Tokens)
- **Deploy**: Railway (backend) + Vercel (frontend)

Estrutura de Diretórios

```
rm-papel-sistema/
├── rm-papel-backend/          # API Flask
│   ├── src/
│   │   ├── models/           # Modelos do banco de dados
│   │   │   ├── database.py   # Configuração SQLAlchemy
│   │   │   ├── user.py       # Modelo de usuário
│   │   │   ├── material.py   # Modelo de material
│   │   │   ├── product.py    # Modelo de produto
│   │   │   ├── sale.py       # Modelo de venda
│   │   │   └── ...
│   │   ├── routes/           # Rotas da API
│   │   │   ├── auth.py       # Autenticação
│   │   │   ├── materials.py  # CRUD materiais
│   │   │   ├── products.py   # CRUD produtos
│   │   │   ├── sales.py      # CRUD vendas
│   │   │   ├── dashboard.py  # Métricas
│   │   │   └── ...
│   │   └── main.py           # Aplicação principal
│   ├── requirements.txt      # Dependências Python
│   ├── Procfile             # Configuração Railway
│   └── railway.json         # Configuração Railway
├── rm-papel-frontend/        # Interface React
│   ├── src/
│   │   ├── components/       # Componentes React
```

```
│   │   │   ├── ui/          # Componentes shadcn/ui
│   │   │   ├── Layout.jsx   # Layout principal
│   │   │   └── ...
│   │   ├── pages/           # Páginas da aplicação
│   │   │   ├── Login.jsx
│   │   │   ├── Dashboard.jsx
│   │   │   └── ...
│   │   ├── contexts/        # Contextos React
│   │   │   └── AuthContext.jsx
│   │   ├── lib/             # Utilitários
│   │   │   └── api.js       # Configuração Axios
│   │   └── App.jsx          # Componente principal
│   ├── package.json         # Dependências Node.js
│   └── vite.config.js       # Configuração Vite
├── README.md                # Documentação principal
├── DEPLOY_GUIDE.md          # Guia de deploy
└── MANUAL_USUARIO.md        # Manual do usuário
```

Backend (Flask API)

Tecnologias Utilizadas

- **Flask**: Framework web Python
- **SQLAlchemy**: ORM para banco de dados
- **Flask-JWT-Extended**: Autenticação JWT
- **Flask-CORS**: Configuração CORS
- **Flask-Migrate**: Migrações de banco
- **ReportLab**: Geração de PDFs
- **Bcrypt**: Hash de senhas

Modelos de Dados

User (Usuário)

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(120), unique=True,
nullable=False)
    password_hash = db.Column(db.String(255), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
```

Material

```
class Material(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    category_id = db.Column(db.Integer,
db.ForeignKey('category.id'))
    supplier_id = db.Column(db.Integer,
db.ForeignKey('supplier.id'))
    purchase_price = db.Column(db.Numeric(10, 2))
    current_stock = db.Column(db.Integer, default=0)
    minimum_stock = db.Column(db.Integer, default=0)
    unit = db.Column(db.String(20))
```

Product (Produto)

```
class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    category_id = db.Column(db.Integer,
db.ForeignKey('category.id'))
    sale_price = db.Column(db.Numeric(10, 2))
    production_time = db.Column(db.Integer)  # em minutos
    description = db.Column(db.Text)
    image_url = db.Column(db.String(255))
    is_active = db.Column(db.Boolean, default=True)
```

Sale (Venda)

```
class Sale(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    customer_id = db.Column(db.Integer,
db.ForeignKey('customer.id'))
    total_amount = db.Column(db.Numeric(10, 2), nullable=False)
    payment_method = db.Column(db.String(50))
    sale_date = db.Column(db.DateTime, default=datetime.utcnow)
    status = db.Column(db.String(20), default='completed')
```

APIs Principais

Autenticação

- `POST /api/auth/register` - Registrar usuário
- `POST /api/auth/login` - Fazer login
- `GET /api/user/profile` - Obter perfil do usuário

Materiais

- `GET /api/materials` - Listar materiais
- `POST /api/materials` - Criar material
- `PUT /api/materials/<id>` - Atualizar material
- `DELETE /api/materials/<id>` - Excluir material

Produtos

- `GET /api/products` - Listar produtos
- `POST /api/products` - Criar produto
- `PUT /api/products/<id>` - Atualizar produto
- `DELETE /api/products/<id>` - Excluir produto

Vendas

- `GET /api/sales` - Listar vendas
- `POST /api/sales` - Registrar venda
- `GET /api/sales/<id>` - Obter venda específica

Dashboard

- `GET /api/dashboard/summary` - Resumo de métricas
- `GET /api/dashboard/sales-chart` - Dados para gráfico de vendas
- `GET /api/dashboard/top-products` - Produtos mais vendidos

Configuração de Ambiente

Variáveis de Ambiente

```
SECRET_KEY=sua-chave-secreta-aqui
JWT_SECRET_KEY=sua-chave-jwt-aqui
DATABASE_URL=postgresql://user:pass@host:port/dbname
FLASK_ENV=production
```

Configuração CORS

```
CORS(app, origins=["*"], supports_credentials=True)
```

Frontend (React)

Tecnologias Utilizadas

- **React 18**: Biblioteca JavaScript
- **Vite**: Build tool e dev server
- **React Router**: Roteamento
- **Tailwind CSS**: Framework CSS
- **shadcn/ui**: Componentes UI
- **Lucide React**: Ícones
- **Recharts**: Gráficos
- **Axios**: Cliente HTTP
- **React Hook Form**: Formulários

Estrutura de Componentes

Layout Principal

```
const Layout = () => {
  const [sidebarOpen, setSidebarOpen] = useState(false);
  const { user, logout } = useAuth();

  return (
    <div className="flex h-screen bg-gray-100">
      <Sidebar />
      <MainContent />
    </div>
  );
};
```

Contexto de Autenticação

```
const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  // Lógica de autenticação

  return (
    <AuthContext.Provider value={{ user, login, logout,
loading }}>
      {children}
    </AuthContext.Provider>
```

```
  );
};
```

Configuração da API

```
const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL || 'http://localhost:
5000/api',
  headers: {
    'Content-Type': 'application/json',
  },
});

// Interceptor para adicionar token
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
```

Roteamento

```
<Routes>
  <Route path="/login" element={<Login />} />
  <Route path="/register" element={<Register />} />
  <Route path="/*" element={
    <ProtectedRoute>
      <Layout />
    </ProtectedRoute>
  }>
    <Route index element={<Dashboard />} />
    <Route path="materials" element={<Materials />} />
    <Route path="products" element={<Products />} />
    <Route path="sales" element={<Sales />} />
    {/* ... outras rotas */}
  </Route>
</Routes>
```

Banco de Dados

Esquema Principal

```sql
 -- Usuários
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(120) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Categorias
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    type VARCHAR(20) NOT NULL -- 'material' ou 'product'
);

-- Fornecedores
CREATE TABLE suppliers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    contact_name VARCHAR(100),
    phone VARCHAR(20),
    email VARCHAR(120),
    address TEXT
);

-- Materiais
CREATE TABLE materials (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    category_id INTEGER REFERENCES categories(id),
    supplier_id INTEGER REFERENCES suppliers(id),
    purchase_price DECIMAL(10,2),
    current_stock INTEGER DEFAULT 0,
    minimum_stock INTEGER DEFAULT 0,
    unit VARCHAR(20),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Produtos
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    category_id INTEGER REFERENCES categories(id),
    sale_price DECIMAL(10,2),
    production_time INTEGER, -- em minutos
```

```
    description TEXT,
    image_url VARCHAR(255),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Clientes
CREATE TABLE customers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    document VARCHAR(20), -- CPF/CNPJ
    phone VARCHAR(20),
    email VARCHAR(120),
    address TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Vendas
CREATE TABLE sales (
    id SERIAL PRIMARY KEY,
    customer_id INTEGER REFERENCES customers(id),
    total_amount DECIMAL(10,2) NOT NULL,
    payment_method VARCHAR(50),
    sale_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) DEFAULT 'completed'
);

-- Itens da venda
CREATE TABLE sale_items (
    id SERIAL PRIMARY KEY,
    sale_id INTEGER REFERENCES sales(id),
    product_id INTEGER REFERENCES products(id),
    quantity INTEGER NOT NULL,
    unit_price DECIMAL(10,2) NOT NULL,
    total_price DECIMAL(10,2) NOT NULL
);
```

Relacionamentos

- **User**: Não possui relacionamentos diretos (sistema single-tenant)
- **Material**: Pertence a uma Category e um Supplier
- **Product**: Pertence a uma Category
- **Sale**: Pertence a um Customer e possui vários SaleItems
- **SaleItem**: Pertence a uma Sale e um Product

Segurança

Autenticação JWT

```
# Geração do token
access_token = create_access_token(
    identity=user.id,
    expires_delta=timedelta(hours=24)
)

# Verificação do token
@jwt_required()
def protected_route():
    current_user_id = get_jwt_identity()
    # Lógica da rota protegida
```

Hash de Senhas

```
from werkzeug.security import generate_password_hash,
check_password_hash

# Ao criar usuário
password_hash = generate_password_hash(password)

# Ao verificar login
if check_password_hash(user.password_hash, password):
    # Login válido
```

CORS

```
CORS(app,
     origins=["https://seu-frontend.vercel.app"],
     supports_credentials=True,
     allow_headers=["Content-Type", "Authorization"])
```

Deploy

Backend (Railway)

1. **Configuração automática**: Railway detecta Python automaticamente
2. **Variáveis de ambiente**: Configuradas no painel do Railway
3. **Banco de dados**: PostgreSQL provisionado automaticamente

4. **Build**: `pip install -r requirements.txt`
5. **Start**: `python src/main.py`

Frontend (Vercel)

1. **Configuração automática**: Vercel detecta Vite automaticamente
2. **Build**: `npm run build`
3. **Output**: `dist/`
4. **Variáveis de ambiente**: `VITE_API_URL`

Configuração de Produção

Backend

```
# Configuração para produção
if os.environ.get('FLASK_ENV') == 'production':
    app.config['DEBUG'] = False
    app.config['SQLALCHEMY_DATABASE_URI'] =
os.environ.get('DATABASE_URL')
```

Frontend

```
// Configuração da API para produção
const API_URL = import.meta.env.VITE_API_URL || 'http://
localhost:5000/api';
```

Monitoramento e Logs

Backend Logs

```
import logging

# Configuração de logs
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Uso nos endpoints
@app.route('/api/sales', methods=['POST'])
def create_sale():
    logger.info(f"Nova venda criada: {sale.id}")
```

Frontend Error Boundary

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    console.error('Error caught by boundary:', error,
errorInfo);
  }

  render() {
    if (this.state.hasError) {
      return <h1>Algo deu errado.</h1>;
    }
    return this.props.children;
  }
}
```

Performance

Backend

- **Paginação**: Implementada em todas as listagens
- **Índices**: Criados em campos de busca frequente
- **Cache**: Headers de cache para recursos estáticos
- **Compressão**: Gzip habilitado

Frontend

- **Code Splitting**: Componentes carregados sob demanda
- **Lazy Loading**: Imagens carregadas conforme necessário
- **Memoização**: React.memo em componentes pesados
- **Bundle Optimization**: Vite otimiza automaticamente

Testes

Backend Tests

```python
import unittest
from main import app

class TestAPI(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()

    def test_login(self):
        response = self.app.post('/api/auth/login',
                                 json={'email': 'test@test.com',
'password': '123456'})
        self.assertEqual(response.status_code, 200)
```

Frontend Tests

```javascript
import { render, screen } from '@testing-library/react';
import Dashboard from './Dashboard';

test('renders dashboard', () => {
  render(<Dashboard />);
  const element = screen.getByText(/Dashboard/i);
  expect(element).toBeInTheDocument();
});
```

Manutenção

Backup do Banco

```bash
# PostgreSQL backup
pg_dump $DATABASE_URL > backup.sql

# Restore
psql $DATABASE_URL < backup.sql
```

Atualizações

1. **Backend**: Push para GitHub → Deploy automático no Railway
2. **Frontend**: Push para GitHub → Deploy automático no Vercel

3. **Banco**: Migrações automáticas via Flask-Migrate

Monitoramento

- **Railway**: Logs e métricas automáticas
- **Vercel**: Analytics e performance
- **Uptime**: Monitoramento de disponibilidade

Esta documentação técnica serve como referência para desenvolvedores que trabalham com o Sistema RM Papel. Para dúvidas específicas sobre implementação, consulte o código-fonte ou entre em contato com a equipe de desenvolvimento.