# 📇 Sistema de Agenda de Contatos

## Documentação Técnica

---

## 1. Descrição do Problema e Objetivo da Solução

### 1.1 Problema

A necessidade de gerenciar contatos de forma eficiente e organizada é comum em diversos contextos. Um sistema de agenda de contatos deve permitir:

- **Armazenar informações** de contatos (nome, e-mail, telefone)
- **Gerenciar contatos** através de operações CRUD (Create, Read, Update, Delete)
- **Buscar contatos** de forma rápida e eficiente por diferentes critérios
- **Interface intuitiva** para facilitar o uso

### 1.2 Objetivo da Solução

Desenvolver um sistema completo de gerenciamento de contatos que atenda aos seguintes requisitos:

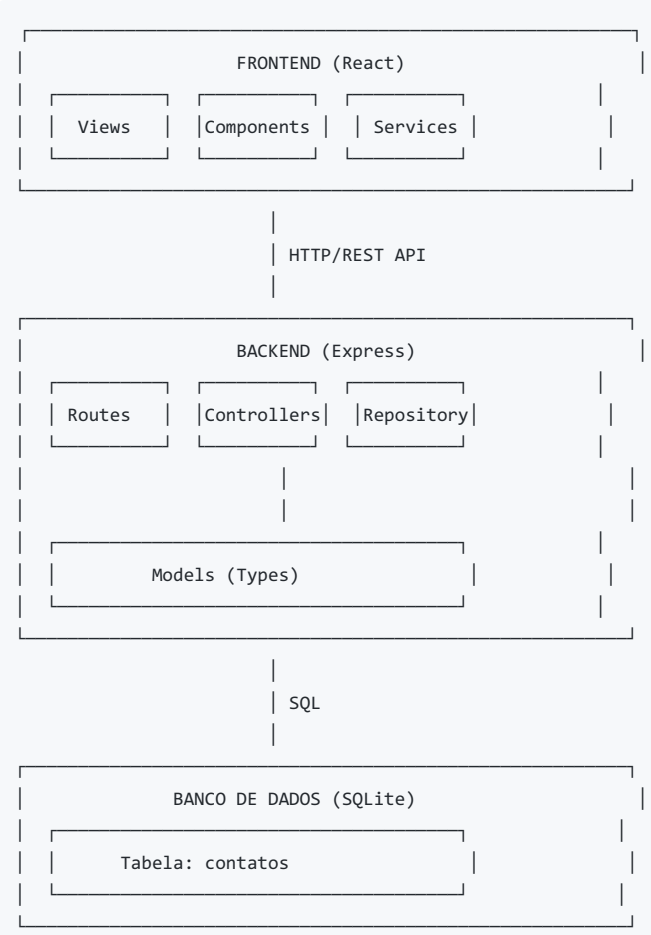
1. **Funcionalidades CRUD completas**:
    - Criar novos contatos
    - Listar todos os contatos
    - Editar contatos existentes
    - Excluir contatos
2. **Sistema de busca avançado**:
    - Busca por nome
    - Busca por e-mail
    - Busca por telefone
    - Busca em tempo real (conforme digitação)
3. **Persistência de dados**:
    - Armazenamento confiável dos dados
    - Banco de dados SQLite para simplicidade e portabilidade
4. **Interface moderna e responsiva**:
    - Design atrativo e intuitivo
    - Experiência de usuário fluida
    - Feedback visual para ações
5. **Arquitetura bem estruturada**:
    - Separação de responsabilidades
    - Código modular e reutilizável
    - Fácil manutenção e extensão

---

## 2. Arquitetura Adotada

### 2.1 Visão Geral

O sistema foi desenvolvido seguindo uma arquitetura em camadas (Layered Architecture) com separação clara entre frontend e backend, utilizando o padrão MVC (Model-View-Controller) no backend.




## 2.2 Camadas do Backend

### 2.2.1 Camada de Rotas (Routes)

**Localização**: `backend/src/routes/contatoRoutes.ts`

Responsável por:

- Definir os endpoints da API REST
- Mapear URLs para controladores
- Configurar métodos HTTP (GET, POST, PUT, DELETE)

**Endpoints**:

- `GET /api/contatos` - Lista todos os contatos
- `GET /api/contatos/search?term=termo` - Busca contatos
- `GET /api/contatos/:id` - Busca contato por ID
- `POST /api/contatos` - Cria novo contato
- `PUT /api/contatos/:id` - Atualiza contato
- `DELETE /api/contatos/:id` - Deleta contato

### 2.2.2 Camada de Controladores (Controllers)

**Localização**: `backend/src/controllers/ContatoController.ts`

Responsável por:

- Processar requisições HTTP
- Validar dados de entrada
- Chamar métodos do repositório
- Retornar respostas HTTP apropriadas
- Tratamento de erros

### 2.2.3 Camada de Repositório (Repository)

**Localização**: `backend/src/repositories/ContatoRepository.ts`

Responsável por:

- Abstrair acesso ao banco de dados
- Implementar operações CRUD
- Executar queries SQL
- Retornar dados no formato esperado

#### 2.2.4 Camada de Modelos (Models)

**Localização**: `backend/src/models/Contato.ts`

Responsável por:

- Definir tipos e interfaces TypeScript
- Estrutura de dados do domínio
- Contratos de dados entre camadas

#### 2.2.5 Camada de Configuração (Config)

**Localização**: `backend/src/config/database.ts`

Responsável por:

- Configuração do banco de dados
- Inicialização da conexão
- Criação de tabelas
- Funções auxiliares de acesso ao banco

### 2.3 Camadas do Frontend

#### 2.3.1 Camada de Apresentação (Views/Components)

**Localização**: `frontend/src/components/`

Componentes React:

- **ContatoForm**: Formulário para criar/editar contatos
- **ContatoList**: Lista de contatos com cards
- **SearchBar**: Barra de busca em tempo real

#### 2.3.2 Camada de Serviços (Services)

**Localização**: `frontend/src/services/api.ts`

Responsável por:

- Comunicação com a API REST
- Encapsular chamadas HTTP
- Gerenciar requisições Axios
- Tratamento de respostas

#### 2.3.3 Camada de Tipos (Types)

**Localização**: `frontend/src/types/Contato.ts`

Responsável por:

- Definir interfaces TypeScript
- Tipagem forte para dados
- Contratos de dados

#### 2.3.4 Camada de Aplicação (App)

**Localização**: `frontend/src/App.tsx`

Responsável por:

- Orquestração dos componentes
- Gerenciamento de estado
- Lógica de negócio da interface
- Coordenação entre componentes

### 2.4 Tecnologias Utilizadas

#### Backend

- **Node.js**: Runtime JavaScript
- **TypeScript**: Linguagem de programação tipada
- **Express**: Framework web para Node.js
- **SQLite**: Banco de dados relacional embutido
- **sqlite3**: Driver para acesso ao SQLite
- **CORS**: Middleware para Cross-Origin Resource Sharing
- **dotenv**: Gerenciamento de variáveis de ambiente
- **Jest**: Framework de testes

#### Frontend

- **React**: Biblioteca para construção de interfaces

- **TypeScript**: Linguagem de programação tipada
- **Vite**: Build tool e dev server
- **Axios**: Cliente HTTP para requisições
- **CSS**: Estilização dos componentes

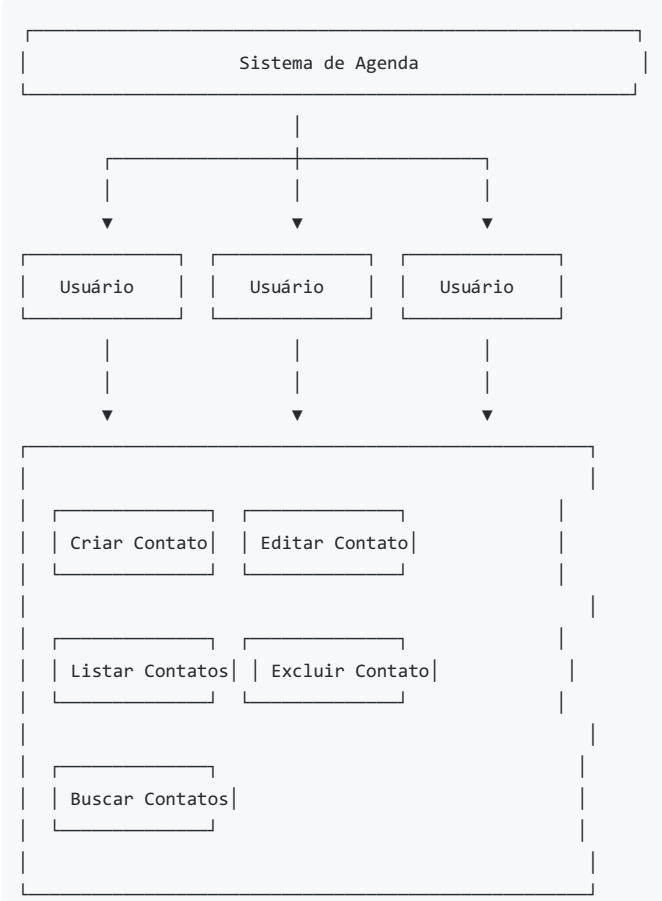
### 2.5 Estrutura de Pacotes

```
agenda/
├── backend/
│   ├── src/
│   │   ├── config/           # Configurações
│   │   │   └── database.ts
│   │   ├── controllers/      # Controladores
│   │   │   └── ContatoController.ts
│   │   ├── models/           # Modelos/Tipos
│   │   │   └── Contato.ts
│   │   ├── repositories/     # Repositórios
│   │   │   └── ContatoRepository.ts
│   │   ├── routes/           # Rotas
│   │   │   └── contatoRoutes.ts
│   │   ├── __tests__/        # Testes
│   │   │   └── ContatoRepository.test.ts
│   │   └── server.ts         # Servidor principal
│   ├── data/                 # Banco de dados SQLite
│   │   └── agenda.db
│   ├── package.json
│   └── tsconfig.json
│
└── frontend/
    ├── src/
    │   ├── components/       # Componentes React
    │   │   ├── ContatoForm.tsx
    │   │   ├── ContatoList.tsx
    │   │   └── SearchBar.tsx
    │   ├── services/         # Serviços de API
    │   │   └── api.ts
    │   ├── types/            # Tipos TypeScript
    │   │   └── Contato.ts
    │   ├── App.tsx           # Componente principal
    │   └── main.tsx          # Entry point
    ├── package.json
    └── vite.config.ts
```

---

## 3. Modelagem

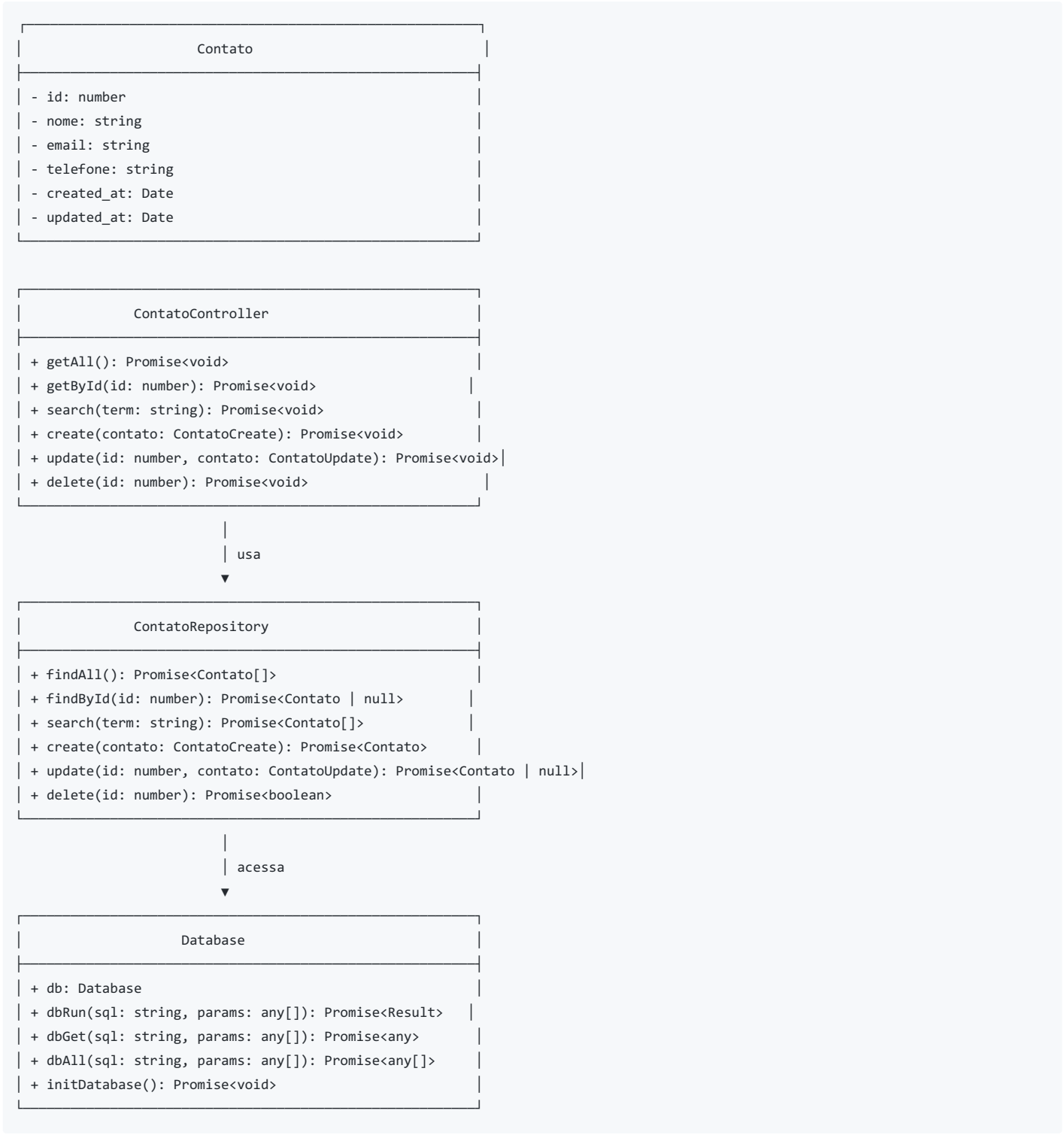
### 3.1 Diagrama de Casos de Uso



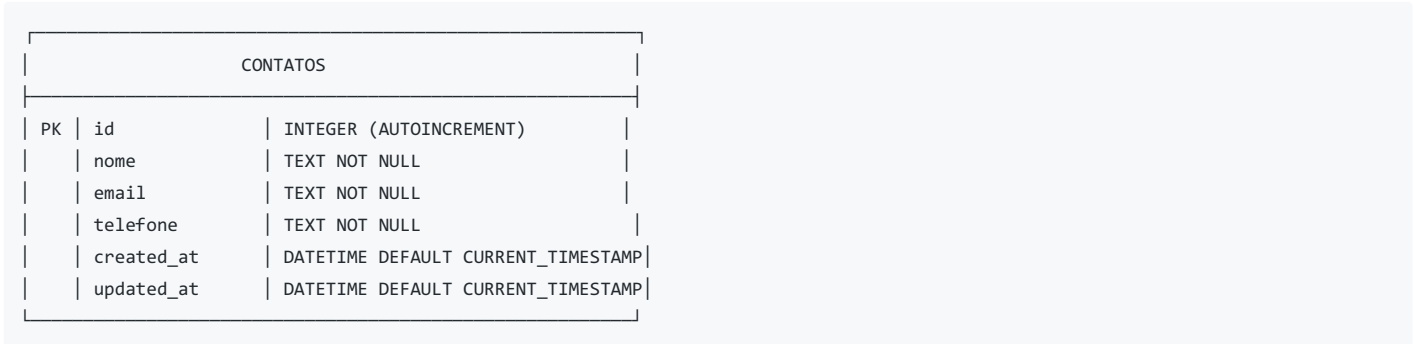

**Casos de Uso**:

1. **Criar Contato**: Usuário pode adicionar um novo contato com nome, e-mail e telefone
2. **Listar Contatos**: Usuário pode visualizar todos os contatos cadastrados
3. **Editar Contato**: Usuário pode modificar informações de um contato existente
4. **Excluir Contato**: Usuário pode remover um contato do sistema
5. **Buscar Contatos**: Usuário pode buscar contatos por nome, e-mail ou telefone

## 3.2 Diagrama de Classes




## 3.3 Modelo de Dados (Diagrama Entidade-Relacionamento)




**Descrição da Tabela**:

- **id**: Chave primária, auto-incremento
- **nome**: Nome completo do contato (obrigatório)
- **email**: Endereço de e-mail do contato (obrigatório)

- **telefone**: Número de telefone do contato (obrigatório)
- **created_at**: Data e hora de criação (automático)
- **updated_at**: Data e hora da última atualização (automático)

## 3.4 Fluxo de Dados (Sequência)

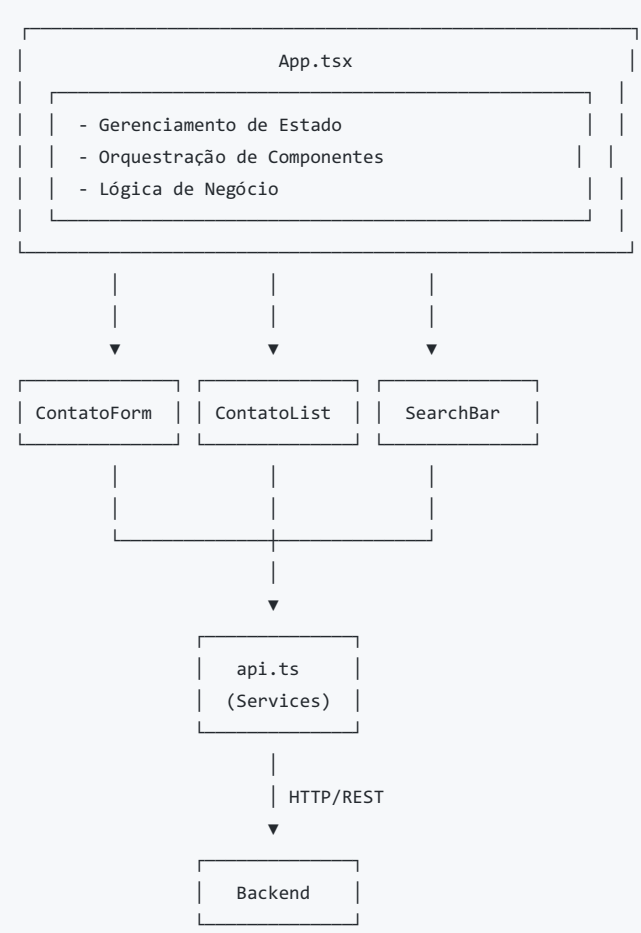
**Criar Contato**:

```
Usuário → ContatoForm → App → api.ts → Backend API
                                            |
                                            ▼
                                    ContatoController
                                            |
                                            ▼
                                    ContatoRepository
                                            |
                                            ▼
                                        SQLite DB
                                            |
                                            ▼
                                    Retorna Contato criado
                                            |
                                            ▼
Usuário ← ContatoList ← App ← api.ts ← Backend API
```

**Buscar Contatos**:

```
Usuário → SearchBar → App → api.ts → Backend API
                                        |
                                        ▼
                                ContatoController
                                        |
                                        ▼
                                ContatoRepository
                                        |
                                        ▼
                                    SQLite DB
                                        |
                                        ▼
                                Retorna Array de Contatos
                                        |
                                        ▼
Usuário ← ContatoList ← App ← api.ts ← Backend API
```

## 3.5 Diagrama de Componentes (Frontend)




---

## 4. Padrões de Projeto Utilizados

### 4.1 Repository Pattern

O padrão Repository foi utilizado para abstrair o acesso ao banco de dados, permitindo:

- Isolamento da lógica de acesso a dados
- Facilidade para trocar o banco de dados
- Testabilidade (mock do repositório)

### 4.2 MVC (Model-View-Controller)

- **Model**: Representa os dados (Contato)
- **View**: Interface do usuário (Componentes React)
- **Controller**: Processa requisições e coordena (ContatoController)

### 4.3 Service Layer

Camada de serviços no frontend para encapsular comunicação com API.

---

## 5. Considerações de Implementação

### 5.1 Segurança

- Validação de dados no backend
- Sanitização de inputs
- CORS configurado adequadamente

### 5.2 Performance

- Índices no banco de dados (implícitos no SQLite)
- Busca otimizada com LIKE
- Componentes React otimizados

### 5.3 Manutenibilidade

- Código modular e organizado
- TypeScript para type safety
- Separação de responsabilidades
- Testes unitários

### 5.4 Escalabilidade

- Arquitetura permite adicionar novos recursos
- Estrutura preparada para crescimento
- Fácil migração para banco de dados mais robusto se necessário

---

## 6. Conclusão

O sistema de Agenda de Contatos foi desenvolvido seguindo boas práticas de engenharia de software, com arquitetura bem definida, código limpo e organizado, e interface moderna. A solução atende completamente aos requisitos propostos, oferecendo funcionalidades CRUD completas, sistema de busca eficiente e experiência de usuário agradável.

A escolha do SQLite permite simplicidade na instalação e uso, enquanto a arquitetura em camadas garante manutenibilidade e extensibilidade do sistema.

---

**Versão**: 1.0
**Data**: 2025
**Tecnologias**: React, TypeScript, Node.js, Express, SQLite