# Resumo do Projeto

O **Sistema de Gerenciamento da Unidade Básica de Saúde (UBS)** possui três tipos de usuários: **Secretária**, **Médico** e **Administrador**, cada um com diferentes permissões de acesso e funcionalidades.

- **Secretária**: Responsável por monitorar e catalogar os resumos diários, como o número de pacientes agendados, atendidos e ausentes. O sistema oferece duas seções principais:
    - **Pacientes**: Permite o cadastro de novos pacientes, visualização e edição de informações de pacientes já registrados, além da emissão de prontuários.
    - **Consultas**: Inclui o agendamento de novas consultas, a visualização de uma lista de consultas já registradas, com opções para editar ou cancelar esses agendamentos.d
- **Médico**: O usuário médico tem acesso ao resumo diário, incluindo os pacientes e consultas designados a ele. Este perfil é dividido em duas seções:
    - **Pacientes**: Permite a visualização das informações dos pacientes sob seu cuidado.
    - **Relatórios**: Gera relatórios sobre os atendimentos realizados.
- **Administrador**: Com acesso total a todas as funcionalidades disponíveis no sistema, o administrador pode:
    - Gerenciar usuários, incluindo a criação de novos perfis e a designação de funções.
    - Aprovar solicitações de alteração de permissões.
    - Visualizar dashboards completos com diversas estatísticas sobre o funcionamento da UBS, além de exportar esses relatórios para análise externa.

# Documentação do Projeto

## Sumário

---

# Backend

## Visão Geral (Backend)

Este projeto fornece um serviço de backend com múltiplos controladores, serviços, repositórios e modelos de domínio. Ele inclui funcionalidades para autenticação de usuários, gerenciamento de clientes, cronogramas, e muito mais. O projeto é construído em **Java** com **Maven**, seguindo uma arquitetura típica do **Spring Boot**.

## Estrutura do Projeto (Backend)

Abaixo está uma visão geral dos principais componentes do projeto.

### Diretório Raiz (Backend)

- **pom.xml**: Gerencia as dependências do Maven e as configurações do projeto. É essencial para a construção do projeto.
- **Dockerfile**: Usado para containerizar a aplicação.
- **database/**: Contém arquivos necessários para configurar o banco de dados, incluindo:
    - **docker-compose.yml**: Define os serviços para rodar o banco de dados com Docker.
    - **Dockerfile**: Configurações Docker para o banco de dados.
    - **script-insert.sql**: Script SQL para a configuração inicial do banco de dados.
- **mvnw** e **mvnw.cmd**: Arquivos do Maven wrapper que permitem executar o Maven sem instalação local.

### Diretório de Código Fonte (Backend)

Este é o núcleo da aplicação. Ele contém todas as classes Java que implementam os serviços do backend, divididas em vários pacotes:

- **annotations/**:
  - **ExceptionHandlerMessage.java**: Lida com exceções com mensagens personalizadas.
  - **Permission.java**: Define anotações de permissão para controle de acesso baseado em funções.
- **components/**:
  - **JwtTokenUtil.java**: Lida com a criação, validação e expiração de tokens JWT, essencial para proteger os endpoints.
- **controller/**:
  - **AuthenticationController.java**: Gerencia o login e autenticação de usuários.
  - **ClienteController.java**, **ConsultaController.java**, etc.: Gerenciam operações relacionadas às diferentes entidades do sistema, como clientes, consultas, cronogramas, etc.
- **domain/**:
  - **DTOs**: Define objetos para transferência de dados (e.g., `BaseDTO.java`, `ClienteFiltroDTO.java`, `UsuarioResponseDTO.java`).
  - **Entidades**: Representam as tabelas do banco de dados (e.g., `Cliente.java`, `Consulta.java`).
  - **Enums**: Definem constantes, como raça e gênero (e.g., `CorRaca.java`, `Genero.java`).
- **repository/**:
  - Interage com o banco de dados via Spring Data JPA (e.g., `ClienteRepository.java`, `ConsultaRepository.java`).
- **service/**:
  - Define a lógica de negócio para operações específicas (e.g., `ClienteService.java`, `ConsultaService.java`).
- **utils/**:
  - Inclui utilitários para configuração e lógica personalizada (e.g., `CorsConfig.java`, `SecurityConfiguration.java`).

#### Diretório de Recursos (Backend)

Contém recursos não-Java, como scripts SQL e arquivos de configuração:

- **db/migration/**: Scripts para inicializar e atualizar o banco de dados.
  - **V1__create-tables.sql**: Cria as tabelas iniciais.
  - **V2__insert-default-values.sql**: Insere valores padrão no banco de dados.
- **error.properties**: Contém mensagens de erro usadas pela aplicação.

#### Diretório de Testes (Backend)

- **TesteApplicationTests.java**: Testes unitários para verificar a lógica da aplicação.

## Fluxo de Execução (Backend)

#### Fluxo de Autenticação (Backend)

1. O usuário envia as credenciais para o endpoint `/auth/login`.
2. O **AuthenticationController** encaminha a solicitação ao **AuthService**, que valida o usuário e gera um token JWT.
3. O **JwtTokenUtil** cria o token JWT com uma data de expiração e informações do usuário.
4. O token JWT é retornado ao cliente, que o utiliza nas requisições futuras como **Bearer Token**.

## Como Construir e Executar (Backend)

### Requisitos

- Java 11 ou superior
- Maven
- Docker (se for usar a configuração com Docker)

### Construção

Para construir o projeto, execute o comando:

Copiar código

- `mvn clean install`

### Execução

Você pode executar o projeto usando Docker:

Copiar código

- `docker-compose up`

Ou diretamente com o Spring Boot:

Copiar código

- `mvn spring-boot:run`

## Endpoints da API (Backend)

- **Autenticação**: `/auth/login`
- **Gerenciamento de Clientes**: `/client/`
- **Consultas**: `/consult/`
- **Status da Conta**: `/account-status/`
- **Cronogramas**: `/schedule/`
- **Usuários**: `/users/`

---

# Frontend

## Visão Geral (Frontend)

Este projeto fornece uma interface frontend desenvolvida com **Next.js** e **TypeScript**, que consome os serviços de backend e oferece funcionalidades relacionadas ao agendamento de consultas, gerenciamento de pacientes, entre outros. Ele inclui componentes reutilizáveis, serviços de API, contexto de autenticação e estilização CSS.

## Estrutura do Projeto (Frontend)

Abaixo está uma visão geral dos principais componentes do projeto.

### Diretório Raiz (Frontend)

- **package.json**: Gerencia as dependências e scripts de execução.
- **Dockerfile**: Usado para containerizar a aplicação frontend.
- **.env.local.example**: Exemplo de variáveis de ambiente necessárias.
- **next.config.js**: Arquivo de configuração do Next.js.
- **.eslintrc.json**: Configurações do ESLint para manter o padrão de código.
- **.prettierrc.json**: Configurações do Prettier para formatação de código.
- **tsconfig.json**: Arquivo de configuração para o TypeScript.
- **README.md**: Instruções básicas de uso e instalação.

### Diretório de Configuração (Frontend)

- **config/pt.json**: Arquivo de configuração para strings de tradução ou textos, usado para internacionalização ou configurações regionais.

### Diretório Público (Frontend)

- **public/images/**: Contém imagens estáticas (logos, ícones).

### Diretório de Código Fonte (Frontend)

- **Componentes (`src/components/`)**: Componentes reutilizáveis como `AgendamentosBox/`, `CardPaciente/`, `Header/`, e `Sidebar/`.
- **Contexto (`src/contexts/`)**: Gerencia o contexto de autenticação através de `AuthContext.tsx`.
- **Páginas (`src/pages/`)**: Contém as principais páginas da aplicação (e.g., `index.tsx`, `login.tsx`, `pacientes.tsx`).
- **Serviços (`src/services/`)**: Conecta o frontend ao backend via chamadas HTTP (e.g., `nextApi.ts`, `serverApi.ts`).

- **Estilos (`src/styles/`)**: Estilos globais e específicos de componentes.
- **Tipos (`src/types/`)**: Define interfaces e tipos TypeScript usados no projeto.

## Fluxo de Execução (Frontend)

### Integração com Backend (Frontend)

1. O usuário realiza login na página `/login`.
2. O frontend envia as credenciais para o backend via `/api/login`.
3. Após autenticação, o token JWT retornado é armazenado e usado para autorizar futuras requisições do frontend.
4. A partir daí, o frontend pode acessar endpoints para listar clientes, agendamentos, etc.

## Como Construir e Executar (Frontend)

### Requisitos

- Node.js 14 ou superior
- Docker (se for usar a configuração com Docker)

### Instalação

Para instalar as dependências, execute:

Copiar código

- `npm install`

### Execução

Você pode executar o frontend com Docker:

Copiar código

- `docker-compose up`

Ou diretamente com Next.js:

Copiar código

- `npm run dev`

### Construção para Produção

Para gerar uma build de produção:

Copiar código

- `npm run build`

## Estrutura de APIs (Frontend)

O frontend se comunica com o backend via uma série de chamadas API. Exemplos incluem:

- **GET /api/cliente/listar**: Lista clientes.
- **GET /api/profissional/listar**: Lista profissionais.
- **POST /api/login**: Faz a autenticação do usuário.