**Documentação Técnica do Projeto: Previsão Climática Full-Stack**

**1. Visão Geral do Projeto**

O sistema "Previsão Climática" é uma aplicação **Full-Stack** desacoplada, projetada para ser escalável e de fácil manutenção.

* **Front-end:** Single Page Application (SPA) construída com Angular, responsável pela interface e consumo de dados.
* **Back-end:** API RESTful desenvolvida em ASP.NET Core, responsável pela lógica de negócios, segurança (JWT) e persistência de dados (SQL Server).
* **Integração Externa:** Comunicação com a API **OpenWeatherMap** para dados de previsão do tempo em tempo real.

|  |  |  |  |
| --- | --- | --- | --- |
| **Camada** | **Tecnologia** | **Servidor Padrão** | **Principal Responsabilidade** |
| Front-end | Angular (Standalone) | <http://localhost:4200> | Interface, Rotas, Fluxo Reativo (RxJS). |
| Back-end | ASP.NET Core (API) | <http://localhost:5168> | Lógica de Negócios, Segurança (JWT), EF Core. |
| Banco de Dados | SQL Server | (Config. via appsettings.json) | Persistência de Usuários e Cidades Favoritas. |

## 2. Documentação Técnica do Front-end (Angular)

### 2.1. Arquitetura e Estrutura

O Front-end utiliza a arquitetura moderna de **Standalone Components**, eliminando a necessidade de *NgModules*. A aplicação é estruturada em:

* **src/app/app.routes.ts:** Define todas as rotas e aplica as Guardas de Rota.
* **src/app/components/:** Contém os componentes de página (ex: home, login, favoritos).
* **src/app/services/:** Contém a lógica de negócio e comunicação com a API (ex: auth.service.ts, favoritos.service.ts).

### 2.2. Gerenciamento de Estado e Reatividade (RxJS)

O projeto depende fortemente da biblioteca RxJS para lidar com assincronia e estado:

|  |  |  |
| --- | --- | --- |
| **Conceito RxJS** | **Arquivos de Uso** | **Descrição** |
| BehaviorSubject | auth.service.ts | Usado para armazenar o estado global de login (isLoggedIn$). Qualquer componente subscrito recebe atualizações imediatas. |
| Pipe async | app.html, favoritos.html | Usado no template (`\*ngIf="isLoggedIn$ |
| switchMap & forkJoin | favoritos.service.ts | Orquestração de Chamadas: Utilizado para buscar a lista de cidades favoritas do usuário e, em seguida, fazer N chamadas em paralelo (uma para a API de clima de cada cidade). |

### 2.3. Segurança e Comunicação

O fluxo de segurança e comunicação com a API é centralizado:

|  |  |  |
| --- | --- | --- |
| **Elemento** | **Caminho** | **Função na Segurança** |
| auth.service.ts | src/app/services/ | Gerencia o localStorage para salvar/recuperar o token JWT (authToken). |
| auth.interceptor.ts | src/app/core/interceptors/ | Middleware Global: Intercepta todas as requisições HTTP de saída e anexa o Authorization: Bearer <token> no header para todas as rotas protegidas do Back-end. |
| authGuard | src/app/guards/ | Guarda de Rota: Impede o acesso às rotas (/home, /favoritos) se o AuthService retornar false para o estado de login, redirecionando o usuário para /login. |

### 2.4. Observações de Manutenção do Front-end

1. **Responsividade:** O menu *hamburger* só funciona porque o JavaScript do Bootstrap (bootstrap.bundle.min.js) foi explicitamente incluído no array scripts do **angular.json**.
2. **Mapeamento de Ícones:** A lógica de tradução dos códigos de clima da OpenWeatherMap para as classes CSS do Bootstrap Icons está centralizada no **weather.service.ts** (variável ICON\_MAP).

## 3. Documentação Técnica do Back-end (ASP.NET Core API)

### 3.1. Arquitetura e Padrões

O Back-end segue o padrão de **Injeção de Dependência (DI)** e utiliza o modelo **Repository** para abstrair o acesso a dados.

* **Program.cs:** Configura todos os serviços (DI), CORS, Autenticação JWT e EF Core.
* **Controllers:** Recebem as requisições HTTP e orquestram a lógica (ex: AuthController, FavoritosController).
* **Repositories (I...Repository):** Classes responsáveis por toda a interação direta com o AppDbContext (SQL Server).
* **Services:** Classes responsáveis por lógica externa, como o OpenWeatherMapService.

### 3.2. Persistência de Dados (Entity Framework Core)

O acesso ao SQL Server é feito através do EF Core.

|  |  |  |  |
| --- | --- | --- | --- |
| **Entidade** | **Tabela SQL** | **Propriedades Críticas (Mapeamento)** | **Padrão** |
| Usuario.cs | Usuarios | [Column("Nome")] no Username, [Column("Senha")] no Password. | Acesso por \_context.Usuarios. |
| CidadeFavorita.cs | CidadesFavoritas | Id (INT IDENTITY(1,1) no SQL), UsuarioId (INT). | Gerenciado por IFavoritosRepository. |
| AppDbContext.cs | Contexto | Define os DbSets e o relacionamento de chave estrangeira. | Deve ser o único ponto de acesso ao banco. |

3.3. Configuração de Segurança (JWT)

|  |  |  |
| --- | --- | --- |
| **Item de Configuração** | **Local** | **Propósito** |
| Chave Secreta | appsettings.json (AppSettings:Token) | Usada para assinar o token JWT. Deve ser longa e complexa. |
| [Authorize] | FavoritosController | Garante que o endpoint só possa ser acessado se um JWT válido for fornecido no cabeçalho Authorization. |
| Extração de UserId | FavoritosController | O UserId do usuário logado é extraído diretamente dos Claims do token JWT para filtrar os favoritos. |

3.4. Lógica de Integração Externa (OpenWeatherMap)

|  |  |  |
| --- | --- | --- |
| **Arquivo** | **Função Específica** | **Ponto de Atenção** |
| OpenWeatherMapService.cs | Comunica-se com a API. Usa o IHttpClientFactory para garantir resiliência e a chave API configurada no appsettings.json. | Configuração da URL e API Key centralizadas. |
| PrevisaoController.cs | GET /5dias | Tratamento de Data: É o único local que usa DateTime.ParseExact() para converter o formato de string de data específico do JSON da OpenWeatherMap para um objeto DateTime antes de agrupar os dados por dia. |