■ Documentação do Frontend – Gerenciador de Eventos (React)

1. Visão Geral

Aplicação web desenvolvida em **React + TypeScript** com **Vite**, estilização com **TailwindCSS**, consumo de API via **Axios** e roteamento com **React Router**. O frontend autentica-se no backend usando **JWT** e permite que administradores cadastrem-se, façam login e gerenciem seus eventos (listar, criar por URL ou upload, editar e excluir), com paginação no grid e carregamento seguro de imagens.

2. Stack e Dependências

Categoria	Tecnologias
Linguagem/Build	TypeScript, Vite
UI/Estilo	TailwindCSS
Roteamento	react-router-dom
HTTP Client	axios (com interceptor para JWT)

3. Estrutura do Projeto

A estrutura abaixo foca nos arquivos relevantes mostrados e suas responsabilidades.

Caminho	Descrição
src/services/api.ts	Instância do Axios + interceptor que injeta o token JWT
src/routes/AppRoutes.tsx	Definição das rotas públicas e privadas
src/pages/Login.tsx	Tela de login do administrador
src/pages/Register.tsx	Tela de cadastro do administrador
src/pages/HomeEventos.tsx	Listagem, paginação e ações (criar/editar/excluir)
src/components/Header.tsx	Cabeçalho com logomarca
src/components/ImagemEvento.tsx	Carrega imagem via URL externa ou endpoint protegido (blob)
src/components/EventoCard.tsx	Card visual de evento com ações
src/components/EventoModal.tsx	Modal para criar evento (URL ou upload) com validações
src/components/EventoEditModal.tsx	Modal para editar data/localização com validações
tailwind.config.js / postcss.config.js	Setup do TailwindCSS
vite-env.d.ts / tsconfig.json	Configurações do TypeScript

4. Configuração e Execução

1) **Pré-requisitos**: Node.js 18+. 2) **Instalação**: `npm install` 3) **Variáveis de ambiente** (opcional): defina `VITE_API_BASE_URL` para apontar para o backend. 4) **Execução em desenvolvimento**: `npm run dev` 5) **Build**: `npm run build` e `npm run preview` para testar o build.

5. Serviço HTTP – api.ts (Axios)

- `baseURL`: por padrão `http://localhost:8080`. - **Interceptor de requisição**: antes de cada chamada, tenta coletar o token de `localStorage` ou `sessionStorage` e injeta `Authorization: Bearer ` no header. - Vantagem: centraliza autenticação e simplifica chamadas em todo o app.

6. Roteamento – AppRoutes.tsx

Rotas definidas com `react-router-dom`: - `/login` (Login) - `/register` (Cadastro de administrador) - `/home` (Home de eventos – requer estar autenticado para funcionar corretamente) O componente `Header` é renderizado globalmente.

7. Telas e Componentes

7.1 Login.tsx

- Campos: e-mail, senha + opção **Gravar Senha** para escolher `localStorage` (persistente) ou `sessionStorage` (sessão). - Ação: `POST /auth/login` → salva `token` e `administrador` e redireciona para `/home`. - UI: Tailwind com feedback de erro e botão para alternar visibilidade da senha.

7.2 Register.tsx

- Campos: nome, e-mail, senha, confirmar senha (validação local de coincidência). - Ação: `POST /auth/register` → exibe mensagem de sucesso e redireciona para login. - Tratamento de erros ajustado para status comuns (400/409) em caso de e-mail duplicado.

7.3 HomeEventos.tsx

- Busca eventos do admin autenticado: `GET /eventos/admin/{admin.id}`. - **Paginação client-side**: `itemsPerPage = 6` com navegação Anterior/Próxima. - **Criação de eventos**: • Via URL: `POST /eventos/url` (JSON puro). • Via upload: `POST /eventos/upload` (FormData com parte `dados` em JSON + parte `imagem`). - **Edição**: `PUT /eventos/{id}` (altera data/localização) via `EventoEditModal`. - **Exclusão**: `DELETE /eventos/{id}`. - **Estado**: local por componente (useState/useEffect); chamadas centralizadas em `api`.

7.4 ImagemEvento.tsx – carregamento seguro de imagens

- Se `imagemUrl` inicia com `http`, usa a fonte externa diretamente. - Caso contrário (ex.: `/eventos/{id}/imagem` protegido por JWT), realiza `GET` com `responseType: 'blob'` usando a instância do Axios (que injeta o token automaticamente) e converte para `ObjectURL` para exibir. - Fallback visual é exibido quando a imagem não é reconhecida.

7.5 EventoModal.tsx

- Controle de modo de imagem: **URL** ou **Upload** (radio buttons). - Validações obrigatórias: nome, data, localização e fonte da imagem conforme o modo escolhido. - Ao salvar, chama `onSave` com os dados para que a página execute a requisição correta.

7.6 EventoEditModal.tsx

- Edição de **data** e **localização** com validações. - Nome do evento é exibido como somente leitura. - Ao confirmar, invoca `onSave(id, novaData, novaLocal)`.

7.7 EventoCard.tsx

- Exibe imagem (via `ImagemEvento`), nome, data (formatada com `toLocaleDateString`) e localização. - Botões **Editar** (abre modal) e **Excluir** (chama API e recarrega lista).

7.8 Header.tsx

- Cabeçalho com logotipo e título do sistema. Presente em todas as páginas.

8. Fluxo de Autenticação no Frontend

1) Usuário se cadastra (opcional) e realiza login. 2) Backend retorna `token` e dados do administrador. 3) O frontend armazena o token conforme escolha (**Gravar Senha**): • `localStorage` para persistir entre sessões. • `sessionStorage` apenas para a sessão atual. 4) O interceptor do Axios injeta o token em todas as chamadas subsequentes. 5) Componentes que exigem dados protegidos (ex.: imagens) usam a instância `api`.

9. Estilo e Design System

- Paleta com fundo em tons de azul escuro e acento ciano (`#00ADB5`). - TailwindCSS aplicado com gradientes, bordas arredondadas e sombras sutis. - Componentes seguem consistência visual e acessibilidade básica (foco/hover/disabled).

10. Boas Práticas Implementadas

- Separação por camadas (services, components, pages, routes). - Interceptor central de autenticação. - Validações de formulário antes de submeter. - Paginação client-side para listas grandes. - Carregamento seguro de imagens protegidas via **blob**.

11. Considerações de Segurança

- Nunca expor o token JWT em querystring; uso exclusivo do header `Authorization`. - Evitar salvar o token em `localStorage` em ambientes altamente sensíveis; considerar cookies HttpOnly se houver risco elevado (para este projeto educacional, a abordagem atual é suficiente). - Sanitização básica de entradas e mensagens de erro genéricas para evitar vazamento de detalhes.

12. Limitações & Próximos Passos

- A paginação atualmente é client-side; para muitas páginas, migrar para paginação server-side. - Falta controle de rota privada (guard) explícito; pode-se criar um wrapper `PrivateRoute`. - Melhorar acessibilidade (ARIA), testes (unitários/E2E) e feedbacks de carregamento/erro. - Acrescentar tratamento de logout e expiração de token (refresh/redirect automático).