🏗️ Arquitetura de Integração com API

Data: 18 de Novembro de 2025
Projeto: Portal Paranhos Frontend
Versão: 1.0.0

📋 Visão Geral

Este documento descreve a arquitetura de integração entre o frontend (Next.js) e o backend (API REST) do Portal Paranhos.

🎯 Objetivos

1. **Separação de Responsabilidades**: Configuração, lógica de negócio e apresentação separadas
2. **Type Safety**: Uso completo de TypeScript para garantir segurança de tipos
3. **Flexibilidade**: Fácil alternância entre dados mock e API real
4. **Manutenibilidade**: Código organizado e fácil de manter
5. **Performance**: Cache inteligente e otimizações do Next.js

---

📁 Estrutura de Arquivos

```
app/
├── config/
│   └── api.ts                  # Configurações da API
├── lib/
│   └── api-client.ts           # Cliente HTTP genérico
├── services/
│   └── municipios.ts           # Serviço de municípios
├── types/
│   └── municipio.ts            # Tipos TypeScript
└── municipios/
    └── page.tsx                # Página que consome o serviço
```

🔧 Camadas da Arquitetura

1. Camada de Configuração (`app/config/`)

Responsabilidade: Centralizar todas as configurações relacionadas à API.

Arquivo: `app/config/api.ts`

Conteúdo:
- URL base da API (via variável de ambiente)
- Endpoints disponíveis

- Configurações de timeout
- Configurações de cache/revalidação
- Headers padrão

Exemplo:

```
export const API_CONFIG = {
  BASE_URL: process.env.NEXT_PUBLIC_API_BASE_URL || 'https://api.paranhospr.com.br',
  TIMEOUT: 30000,
  REVALIDATE: 60,
};

export const API_ENDPOINTS = {
  MUNICIPIOS: '/municipios',
  MUNICIPIO_BY_ID: (id: number) => `/municipios/${id}`,
};
```

Vantagens:
- ✅ Único ponto de mudança para URLs
- ✅ Fácil manutenção
- ✅ Configuração via variáveis de ambiente

---

2. Camada de Tipos (`app/types/`)

Responsabilidade: Definir todos os tipos TypeScript usados na aplicação.

Arquivo: `app/types/municipio.ts`

Conteúdo:
- Interfaces de entidades (Municipio, Vereador, etc.)
- Tipos de classificação e enums
- Tipos de resposta da API
- Tipos de filtros e parâmetros

Exemplo:

```
export interface Municipio {
  id: number;
  nome: string;
  ibgeCode: string;
  territorioId: number;
  territorio?: TerritorioTuristico;
  detalhe?: MunicipioDetalhe;
  stats?: MunicipioStats;
}

export type ClassificacaoMunicipal = 'OURO' | 'PRATA' | 'BRONZE' |
'SEM_CLASSIFICACAO';
```

Vantagens:
- ✅ Type safety em toda a aplicação
- ✅ Autocomplete no editor
- ✅ Detecção de erros em tempo de desenvolvimento
- ✅ Documentação viva do código

3. Camada de Cliente HTTP (`app/lib/`)

Responsabilidade: Fornecer funções genéricas para fazer requisições HTTP.

Arquivo: `app/lib/api-client.ts`

Conteúdo:
- Função `apiGet<T>()` para requisições GET
- Função `apiPost<T>()` para requisições POST
- Tratamento de erros customizado (`ApiError`)
- Timeout automático
- Headers padrão

Exemplo:

```
export async function apiGet<T>(endpoint: string): Promise<T> {
  const url = `${API_CONFIG.BASE_URL}${endpoint}`;
  const response = await fetch(url, {
    headers: DEFAULT_HEADERS,
  });

  if (!response.ok) {
    throw new ApiError(/* ... */);
  }

  return await response.json();
}
```

Vantagens:
- ✅ Reutilização de código
- ✅ Tratamento de erros centralizado
- ✅ Fácil adicionar interceptors ou middlewares
- ✅ Tipagem genérica com TypeScript

---

4. Camada de Serviços (`app/services/`)

Responsabilidade: Fornecer funções de alto nível para consumir dados específicos.

Arquivo: `app/services/municipios.ts`

Conteúdo:
- Funções públicas: `listarMunicipios()` , `buscarMunicipioPorId()` , etc.
- Funções mock para desenvolvimento
- Funções auxiliares: `getClassificacaoColor()` , `formatarWhatsApp()`
- Flag `USE_MOCK` para alternar entre mock e API

Exemplo:

```
export async function listarMunicipios(filtros?: MunicipiosFiltros): Promise<ListaMu-
nicipiosResponse> {
  if (USE_MOCK) {
    return listarMunicipiosMock(filtros);
  }
  return listarMunicipiosApi(filtros);
}
```

Vantagens:
- ✅ Abstração da fonte de dados (mock vs API)
- ✅ Lógica de negócio centralizada
- ✅ Fácil testar e desenvolver sem backend
- ✅ Funções auxiliares específicas do domínio

---

5. Camada de Apresentação (`app/municipios/`)

Responsabilidade: Renderizar a interface do usuário.

Arquivo: `app/municipios/page.tsx`

Conteúdo:
- Componentes React
- Consumo dos serviços
- Lógica de apresentação
- Estilos (TailwindCSS)

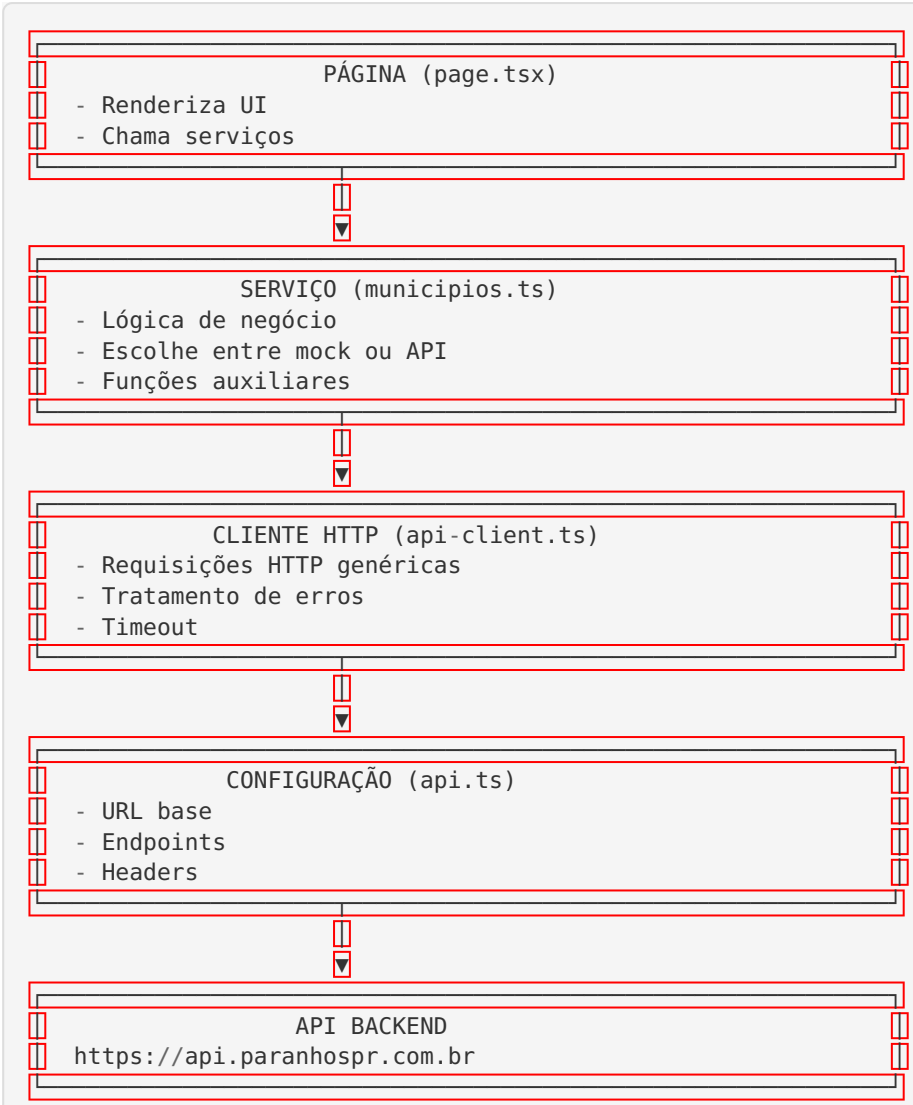
Exemplo:

```
export default async function MunicipiosPage() {
  const { municipios, total } = await listarMunicipios({ perPage: 50 });

  return (
    <main>
      {municipios.map(m => <MunicipioCard key={m.id} municipio={m} />)}
    </main>
  );
}
```

Vantagens:
- ✅ Componentes focados em apresentação
- ✅ Server Components do Next.js 14
- ✅ SEO otimizado
- ✅ Performance com cache automático

🔄 Fluxo de Dados



🎛️ Modo Mock vs API Real

Modo Mock (Desenvolvimento)

Quando usar:
- Backend ainda não está disponível
- Desenvolvimento de UI sem dependências
- Testes de interface
- Demonstrações

Como ativar:

```
// Em app/services/municipios.ts
const USE_MOCK = true;
```

Dados disponíveis:
- 8 municípios de exemplo

- Dados completos (prefeito, vice, stats, etc.)
- Estatísticas gerais

Vantagens:
- ✅ Desenvolvimento independente do backend
- ✅ Dados consistentes para testes
- ✅ Sem necessidade de conexão com API

Modo API (Produção)

Quando usar:
- Backend está disponível
- Testes de integração
- Produção

Como ativar:

```
// Em app/services/municipios.ts
const USE_MOCK = false;
```

Configuração:

```
# .env.local
NEXT_PUBLIC_API_BASE_URL="https://api.paranhospr.com.br"
```

Vantagens:
- ✅ Dados reais e atualizados
- ✅ Funcionalidades completas
- ✅ Integração real com backend

🔐 Variáveis de Ambiente

Desenvolvimento Local

Crie um arquivo `.env.local`:

```
# API Backend
NEXT_PUBLIC_API_BASE_URL="http://localhost:3001"

# Ou use a API de produção
# NEXT_PUBLIC_API_BASE_URL="https://api.paranhospr.com.br"
```

Produção (Vercel)

Configure no painel do Vercel:

1. Acesse: **Settings → Environment Variables**
2. Adicione:
   - **Name**: `NEXT_PUBLIC_API_BASE_URL`

- **Value**: `https://api.paranhospr.com.br`
- **Environment**: Production, Preview, Development

📡 Endpoints da API

Municípios

`GET /municipios`

Lista todos os municípios com filtros opcionais.

Query Parameters:
- `nome` (string, opcional): Filtrar por nome
- `territorioId` (number, opcional): Filtrar por território
- `classificacao` (string, opcional): Filtrar por classificação
- `page` (number, opcional): Página atual (padrão: 1)
- `perPage` (number, opcional): Itens por página (padrão: 10)

Response:

```
{
  municipios: Municipio[];
  total: number;
  page: number;
  perPage: number;
}
```

Exemplo:

```
GET /municipios?territorioId=1&classificacao=OURO&page=1&perPage=20
```

`GET /municipios/:id`

Busca município por ID com dados completos.

Response:

```
{
  id: number;
  nome: string;
  ibgeCode: string;
  territorioId: number;
  territorio: TerritorioTuristico;
  detalhe: MunicipioDetalhe;
  stats: MunicipioStats;
  vereadores: Vereador[];
}
```

Exemplo:

```
GET /municipios/1
```

Estatísticas

`GET /stats/gerais`

Retorna estatísticas gerais do portal.

Response:

```
{
  totalMunicipios: number;
  prefeitosComWhatsApp: number;
  vicesComWhatsApp: number;
  vereadoresComWhatsApp: number;
  totalVereadores: number;
}
```

Exemplo:

```
GET /stats/gerais
```

Territórios

`GET /territorios`

Lista todos os territórios turísticos.

Response:

```
{
  id: number;
  nome: string;
  descricao?: string;
}[]
```

Vereadores

`GET /vereadores/municipio/:municipioId`

Lista vereadores de um município específico.

Response:

```
{
  id: number;
  municipioId: number;
  nome: string;
  partido: string;
  votacao: number;
  whatsapp?: string;
  email?: string;
  instagram?: string;
  facebook?: string;
}[]
```

🚨 Tratamento de Erros

Tipos de Erro

`ApiError`

Erro customizado para erros da API.

Propriedades:
- `message`: Mensagem de erro
- `status`: Código HTTP (404, 500, etc.)
- `statusText`: Texto do status HTTP
- `data`: Dados adicionais do erro (opcional)

Exemplo:

```
try {
  const municipios = await listarMunicipios();
} catch (error) {
  if (error instanceof ApiError) {
    console.error(`Erro ${error.status}: ${error.message}`);
  }
}
```

Erros Comuns

408 - Request Timeout

Causa: Requisição demorou mais que o timeout configurado (30s)

Solução:
- Verificar conexão com a API
- Aumentar timeout em `app/config/api.ts`
- Ativar modo mock temporariamente

---

404 - Not Found

Causa: Endpoint ou recurso não encontrado

Solução:
- Verificar se a URL da API está correta
- Verificar se o endpoint existe no backend
- Verificar se o ID do recurso é válido

---

500 - Internal Server Error

Causa: Erro no servidor backend

Solução:
- Verificar logs do backend
- Verificar se os dados estão no formato esperado
- Reportar erro para equipe de backend

🎨 Funções Auxiliares

`getClassificacaoColor(classificacao)`

Retorna classes CSS do TailwindCSS para colorir badges de classificação.

Parâmetros:
- `classificacao`: 'OURO' | 'PRATA' | 'BRONZE' | 'SEM_CLASSIFICACAO'

Retorno:
- String com classes CSS

Exemplo:

```
const classes = getClassificacaoColor('OURO');
// 'bg-yellow-400/10 text-yellow-400 border-yellow-400/30'
```

`formatarWhatsApp(whatsapp)`

Formata número de telefone no padrão brasileiro.

Parâmetros:
- `whatsapp`: String com número de telefone

Retorno:
- String formatada: `+55 (XX) XXXXX-XXXX`

Exemplo:

```
const formatado = formatarWhatsApp('+5541999990001');
// '+55 (41) 99999-0001'
```

🔄 Cache e Revalidação

Next.js App Router

O Next.js 14 com App Router possui cache automático para requisições `fetch()`.

Configuração:

```
const response = await fetch(url, {
  next: { revalidate: 60 } // Revalida a cada 60 segundos
});
```

Estratégias:

1. Static Generation (padrão)

```
// Cache infinito, regenera no build
const response = await fetch(url);
```

2. Incremental Static Regeneration (ISR)

```
// Revalida a cada X segundos
const response = await fetch(url, {
  next: { revalidate: 60 }
});
```

3. Server-Side Rendering (SSR)

```
// Sem cache, sempre busca dados frescos
const response = await fetch(url, {
  cache: 'no-store'
});
```

📊 Performance

Otimizações Implementadas

1. **Server Components**: Renderização no servidor
2. **Cache Automático**: Next.js cache de requisições
3. **Timeout**: Evita requisições travadas
4. **Tipagem**: Detecção de erros em tempo de desenvolvimento
5. **Code Splitting**: Carregamento sob demanda

Métricas Esperadas

- **Time to First Byte (TTFB)**: < 200ms
- **First Contentful Paint (FCP)**: < 1.5s
- **Largest Contentful Paint (LCP)**: < 2.5s
- **Time to Interactive (TTI)**: < 3.5s

🧪 Testes

Testar com Mock

```
# 1. Ative o modo mock
# Em app/services/municipios.ts: USE_MOCK = true

# 2. Rode o projeto
npm run dev

# 3. Acesse http://localhost:3000/municipios
```

Testar com API Local

```
# 1. Configure a URL da API local
# .env.local: NEXT_PUBLIC_API_BASE_URL="http://localhost:3001"

# 2. Desative o modo mock
# Em app/services/municipios.ts: USE_MOCK = false

# 3. Rode o backend localmente (porta 3001)

# 4. Rode o frontend
npm run dev

# 5. Acesse http://localhost:3000/municipios
```

Testar com API de Produção

```
# 1. Configure a URL da API de produção
# .env.local: NEXT_PUBLIC_API_BASE_URL="https://api.paranhospr.com.br"

# 2. Desative o modo mock
# Em app/services/municipios.ts: USE_MOCK = false

# 3. Rode o frontend
npm run dev

# 4. Acesse http://localhost:3000/municipios
```

📚 Referências

- Next.js 14 Documentation (https://nextjs.org/docs)
- TypeScript Handbook (https://www.typescriptlang.org/docs/)
- Fetch API (https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)
- TailwindCSS (https://tailwindcss.com/docs)

🤝 Contribuindo

Para adicionar novos serviços ou endpoints:

1. **Adicione o endpoint** em `app/config/api.ts`
2. **Defina os tipos** em `app/types/`
3. **Crie o serviço** em `app/services/`
4. **Adicione dados mock** para desenvolvimento
5. **Consuma o serviço** nas páginas

Última atualização: 18 de Novembro de 2025
Autor: Equipe Portal Paranhos