Objetivo:

I. Requisição HTTP.

I. Requisição HTTP

Em aplicações web, requisições HTTP são o mecanismo padrão para que o navegador se comunique com um servidor, permitindo buscar ou enviar dados.

Existem diversas bibliotecas que facilitam a realização de requisições HTTP no React. As mais populares são:

- fetch API: nativa do JS, oferece uma forma simples de fazer requisições;
- Axios: uma biblioteca popular que disponibiliza uma API mais amigável, além de recursos adicionais, como interceptores e cancelamento de requisições.

Neste contexto, utilizaremos Axios para realizar nossas requisições HTTP.

Instalação do Axios:

```
npm i axios
```

Organização do código:

Uma boa prática é organizar o código em pastas e arquivos para facilitar a manutenção e escalabilidade. Ao lado, apresentamos uma estrutura recomendada:

- Pasta services: contém os arquivos responsáveis pelas requisições HTTP:
  - api.ts (Figura 1): responsável por configurar a instância do Axios;
  - cep.ts (Figura 2): contém a função responsável por fazer uma requisição HTTP do tipo GET;
- Pasta types (Figura 3): armazena as definições de tipos usando interfaces para garantir segurança e clareza no código.

Como exemplo o código a seguir faz requisições HTTP GET no serviço da ViaCEP (https://viacep.com.br).

Explicação do código api.ts (Figura 1):

- Criamos uma instância do Axios com uma URL base para o serviço ViaCEP;
- A instância configurada facilita a reutilização dessa configuração ao longo da aplicação.

```
import axios from "axios";
```

```
const api = axios.create({
  baseURL: "https://viacep.com.br/ws/",
});

export default api;
```

Figura 1 – Código do arquivo services/api.ts.

Explicação do código cep.ts (Figura 2):

- A função getCep faz uma requisição HTTP do tipo GET para buscar informações de um CEP;
- Utilizamos unknown no catch para garantir que a variável de erro seja verificada antes de acessar suas propriedades;
- Caso o erro seja uma instância de Error, retornamos a mensagem; caso contrário, retornamos uma mensagem genérica de erro.

```
import { CepProps, ErrorProps } from "../types";
import api from "./api";

export async function getCep(numero: string): Promise<CepProps | ErrorProps> {
  try {
    const { data } = await api.get<CepProps | ErrorProps>(`${numero}/json`);
    return data;
  } catch (e: unknown) {
    /* O tipo unknown é mais seguro que any, pois exige que façamos
    uma verificação de tipo antes de acessar suas propriedades */
    if (e instanceof Error) {
      return { erro: e.message };
    }
    return { erro: "Erro desconhecido" };
  }
}
```

Figura 2 – Código do arquivo services/cep.ts.

Explicação do código types/index.ts (Figura 3):

- CepProps: define o tipo dos dados retornados pela API do ViaCEP;
- ErrorProps: define o formato do objeto de erro retornado em caso de o CEP não existir.

```
export interface CepProps {
  cep: string;
  logradouro: string;
  complemento: string;
  unidade: string;
```

```
  bairro: string;
  localidade: string;
  uf: string;
  estado: string;
  regiao: string;
  ibge: string;
  gia: string;
  ddd: string;
  siafi: string;
}

export interface ErrorProps {
  erro: string;
}
```

Figura 3 – Código do arquivo types/index.ts.

Na função getCep a instrução

```
await api.get<CepProps | ErrorProps>(`${numero}/json`);
```

faz uma requisição HTTP assíncrona. O `await` pausa a execução da função até que a Promise retornada pelo Axios seja resolvida ou rejeitada. Desta forma, a função getCep precisa ser `async` e o seu retorno é uma Promise, o que significa que o resultado dessa função não estará disponível imediatamente.

Por este motivo, no componente App (Figura 4), o uso da função getCep precisa ser precedido por `await`, o que por sua vez, precisa estar em uma função que seja `async`. A seguir está o resultado em caso de sucesso e falha:

CEP 12328070
Buscar
{"cep":"12328-070","logradouro":"Avenida Faria Lima","complemento":"","unidade":"","bairro":"Jardim Santa Maria","localidade":"Jacareí","uf":"SP","estado":"São Paulo","regiao":"Sudeste","ibge":"3524402","gia":"3920","ddd":"12","siafi":"6589"}

CEP 12328071
Buscar
{"erro":"true"}

```
import { useState } from "react";
import { getCep } from "./services/cep";
import { CepProps, ErrorProps } from "./types";

export default function App() {
  const [entrada,setEntrada] = useState("");
  const [resposta, setResposta] = useState<CepProps | ErrorProps | null>(null);

  async function obter(){
```

```
    const res = await getCep(entrada);
    setResposta(res);
  }

  return (
    <>
      <div>
        <label htmlFor="cep">CEP</label>
        <input id="cep" value={entrada} onChange={(e) => setEntrada(e.target.value)} />
      </div>
      <div>
        <button onClick={obter}>Buscar</button>
      </div>
      {resposta && <div>{JSON.stringify(resposta)}</div>}
    </>
  );
}
```

Figura 4 – Código do arquivo App.tsx.

II. Exercícios

Exercício 1 – O IBGE possui uma API para disponibilização de dados de localidades do país. Nela é possível listar as regiões, estados e mesorregiões do país (https://servicodados.ibge.gov.br/api/docs/localidades).

Fazer um aplicativo para listar as regiões do país ao carregar a aplicação.

Use o código a seguir no arquivo services/api.ts.

Dica:

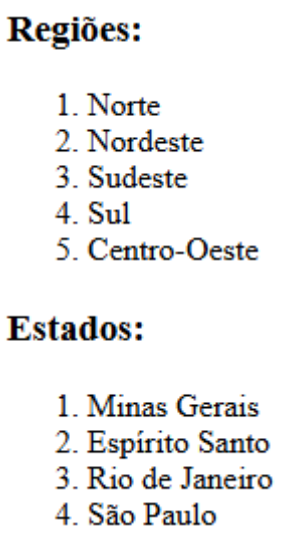
- Faça a requisição ao montar o componente App. Use o Hook useEffect.

```
import axios from "axios";

const api = axios.create({
  baseURL: "https://servicodados.ibge.gov.br/api/v1/localidades/regioes",
});

export default api;
```

Exercício 2 – Alterar o aplicativo do Exercício 1 para o usuário clicar sobre o nome de uma região e serem listados os estados dessa região.

Dica:

- Adicione `${id}/estados` na URL base para obter os estados de uma determinada região.



Exercício 3 – Alterar o aplicativo do Exercício 2 para o usuário clicar sobre o nome de um estado e serem listadas as mesorregiões desse estado.

Dica:

- A URL a seguir lista as mesorregiões do estado do Mato Grosso do Sul:

https://servicodados.ibge.gov.br/api/v1/localidades/estados/50/mesorregioes