1. Atividades Práticas - Integração com APIs

Nesta atividade prática, vamos criar um exemplo simples passo a passo para ilustrar como podemos fazer a integração com APIs com React.

1.1 Criando o projeto, instalando dependências e iniciando o servidor local

Crie um projeto chamado **compras**, digitando o seguinte comando no terminal:

```
npm create vite@latest my-api-react -- --template react
```

Este processo de configuração inicial do projeto leva alguns segundos. Ao terminar, o Vite repassa instruções para que você termine de instalar as dependências do seu projeto. Desta forma, digite o seguinte comando para **entrar no diretório** recém criado do nosso projeto:

```
cd my-api-react
```

Em seguida, **instale as dependências** necessárias do projeto executando no terminal o seguinte comando:

```
npm install
```

Antes de começar a usar o **Axios** em seus projetos, você precisa instalar essa biblioteca no seu projeto React:

```
npm install axios
```

Você pode, agora, **abrir a aplicação no editor Visual Studio Code**. Para isso, dentro do diretório do projeto digite o seguinte comando no terminal:

```
code .
```

Em seguida, você inicializará um servidor local e executará o projeto em seu navegador:

```
npm run dev
```

Ao executar esse script, você iniciará um servidor local de desenvolvimento, executará o código do projeto, iniciará um observador que detecta alterações no código e abrirá o projeto em um navegador web. Ele irá rodar a aplicação em modo desenvolvimento em http://localhost:5173/.

1.2 Passo 1: Conhecendo a API Reqres

A API Reqres é uma API gratuita e pública que fornece dados fictícios (mock) para simular requisições HTTP em aplicações front-end ou back-end durante o desenvolvimento e testes. Ela é muito utilizada por desenvolvedores para testar e demonstrar funcionalidades relacionadas a requisições de APIs, como operações de CRUD (Create, Read, Update, Delete).

Alguns dos recursos oferecidos pela API Reqres incluem:

- **Endpoints para usuários:** permite acessar, criar, atualizar e deletar dados de usuários fictícios.
- **Dados de exemplo:** disponibiliza informações como nomes, sobrenomes, endereços de e-mail e avatares de usuários.
- **Respostas em JSON:** todos os dados são retornados no formato JSON, que é amplamente utilizado em aplicações web.
- **Simulação de erros:** a API permite simular respostas de erro, como códigos de status 404 (não encontrado) ou 500 (erro interno do servidor), para testar o comportamento das aplicações diante de falhas.

A API Reqres é útil para desenvolvedores que estão criando ou testando a lógica de requisições HTTP em suas aplicações sem precisar se conectar a uma API real, proporcionando um ambiente seguro e controlado para esse propósito.

1.3 Passo 2: Obtendo recursos da API

Nesta etapa, vamos começar o desenvolvimento da nossa aplicação. Nosso objetivo é obter a listagem de usuários fornecida pela API Reqres. Para isso, altere crie o arquivo **MyGet.jsx** com o seguinte código:

```
import { useEffect } from "react";
import axios from "axios";

function MyGet() {
  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await axios.get("https://reqres.in/api/users");
        console.log(response);
```

```
      } catch (error) {
        console.error("Erro ao buscar os dados:", error);
      }
    };
    fetchData();
  }, []);

  return <p>Teste com API Axios</p>;
}
export default MyGet;
```

Uma requisição feita com **Axios** (ou qualquer outra biblioteca que use **Promises**) pode estar em um dos seguintes estados:

- **Pending (Pendente):** Esse é o estado inicial da requisição. Ele indica que a requisição foi feita, mas ainda não foi concluída. Enquanto a Promise estiver nesse estado, o Axios está aguardando a resposta do servidor.
- **Fulfilled (Concluída com sucesso):** Nesse estado, a requisição foi concluída com sucesso e o servidor retornou uma resposta válida. Quando isso acontece, a Promise é resolvida e você recebe os dados de resposta da API.
- **Rejected (Rejeitada):** Indica que a requisição falhou por algum motivo, como um erro de rede, um código de status HTTP que indica falha (como 404 ou 500), ou outro problema durante a comunicação com o servidor. Quando isso ocorre, a Promise é rejeitada e o código de tratamento de erros é executado.

Abra o Chrome DevTools para observar o resultado da execução da nossa aplicação:

Figura 1: Retorno da requisição GET para a API Reqres

OBS: Você pode estar vendo a mensagem duplicada no console devido ao uso do **<StrictMode>** no arquivo **main.jsx**. No React, quando você utiliza **<StrictMode>**, ele deliberadamente executa certos métodos e ciclos de vida duas vezes em modo de desenvolvimento. Isso é feito para detectar efeitos colaterais inesperados e garantir que seu código seja seguro e compatível com futuras versões do React.

Ao utilizar o **Axios** para realizar requisições HTTP, ele retorna uma **Promise**. Se a requisição for concluída com sucesso, essa **Promise** é resolvida e retorna um objeto de resposta que contém diversas informações úteis. O formato desse objeto é geralmente o seguinte:

- **data:** Contém os dados retornados pela API. Esse é o conteúdo principal da resposta e é onde você encontrará as informações solicitadas.
- **status:** O código de status HTTP da resposta (como 200 para sucesso, 404 para "não encontrado", etc.).
- **statusText:** Uma breve mensagem textual correspondente ao código de status (como "OK" para 200).
- **headers:** Um objeto contendo os cabeçalhos de resposta enviados pelo servidor.
- **config:** As configurações da requisição original, incluindo parâmetros como URL, método e cabeçalhos de solicitação.
- **request:** O objeto de requisição que foi usado para fazer a chamada à API.

Esse formato padrão do Axios facilita a manipulação e a análise das respostas da API, permitindo que você acesse rapidamente os dados que precisa e gerencie qualquer erro que possa ocorrer durante a requisição.

Neste exemplo, podemos ver na Figura 1 que o Axios retorna uma resposta contendo o status da requisição. Como o status é **200**, sabemos que a requisição foi bem-sucedida. Note que a lista de usuários fornecida pela API Reqres está localizada no campo **data**. Para apresentar esses dados de maneira mais clara e organizada, podemos utilizar o método **forEach()** do JavaScript para percorrer este array.

OBS: Como o objetivo é apenas imprimir os dados no console e não transformar ou gerar uma nova lista, o **forEach()** é mais apropriado do que o **map()**. O **map()** é geralmente usado quando precisamos retornar um novo array.

Atualize o arquivo **MyGet.jsx** conforme mostrado abaixo:

```
import { useEffect } from "react";
import axios from "axios";
function MyGet() {
  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await axios.get("https://reqres.in/api/users");
        // Itera sobre os dados da resposta e registra as informações de cada
usuário no console
        response.data.data.forEach((user) => {
          console.log(
            `ID: ${user.id} - Nome: ${user.first_name} ${user.last_name}`
          );
        });
      } catch (error) {
        console.error("Erro ao buscar os dados:", error);
      }
    };
    fetchData();
  }, []);
  return <p>Teste com API Axios</p>;
}
export default MyGet;
```

Abra o Chrome DevTools para observar o resultado da execução da nossa aplicação:

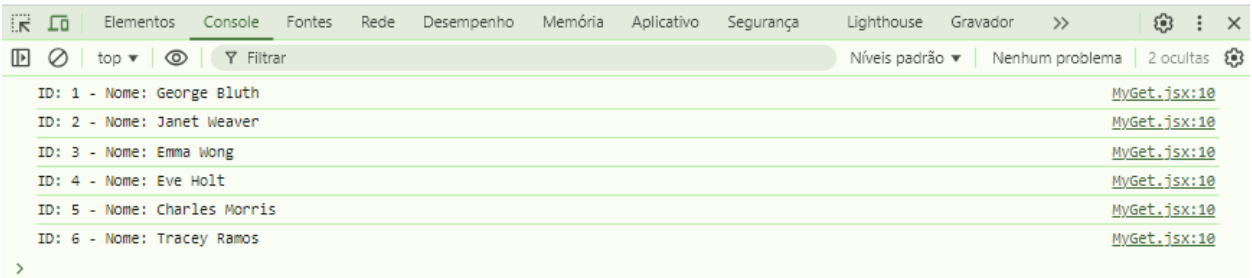

Figura 2: Customização dos dados vindos da API Reqres

Neste API, podemos realizar uma requisição HTTP GET passando parâmetros específicos. Por exemplo, se quisermos obter apenas as informações do usuário com ID=2, atualize o código em **MyGet.jsx** para o seguinte:

```
import { useEffect } from "react";
import axios from "axios";

function MyGet() {
  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await axios.get("https://reqres.in/api/users/2");

        // Registra as informações do usuário específico no console
        const user = response.data.data;
        console.log(`ID: ${user.id} - Nome: ${user.first_name}
${user.last_name}`);
      } catch (error) {
        console.error("Erro ao buscar os dados:", error);
      }
    };
    fetchData();
  }, []);
  return <p>Teste com API Axios</p>;
}
export default MyGet;
```

Como estamos buscando um único usuário com o ID 2, não há necessidade de usar **forEach**, pois a resposta não será um **array** de usuários, mas sim um **objeto** com as informações desse usuário específico.

Abra o Chrome DevTools para observar o resultado da execução da nossa aplicação:

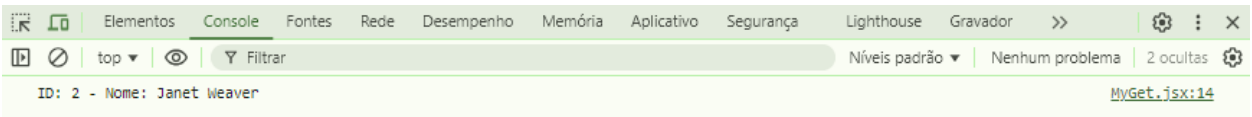

Figura 3: Retorno dos dados do usuário com ID 2

1.4 Passo 3: Enviando recursos para a API

Para criar um novo usuário na API Reqres, utilizaremos uma requisição HTTP POST. Para isso, crie o arquivo **MyPost.jsx** com o seguinte código:

```
import React, { useEffect } from 'react';
import axios from 'axios';

function MyPost() {
    useEffect(() => {
        const createUser = async () => {
            const user = {
                first_name: 'John',
                last_name: 'Lilly',
                job_title: 'Software Engineer',
            };
            try {
                const response = await
axios.post('https://reqres.in/api/users', user);
                console.log(response.data);
            } catch (error) {
                console.error('Erro ao criar usuário:', error);
            }
        };
        createUser();
    }, []);
    return <p>Teste de POST com API Axios</p>;
}
export default MyPost;
```

Observe que a função **createUser** é projetada para criar um novo usuário enviando uma requisição **POST** para a API. Ela utiliza a abordagem assíncrona para esperar pela resposta e tratar possíveis erros de forma eficaz. O objeto **user** com as informações do usuário é enviado como parte do **corpo da requisição**, e a resposta é manipulada para confirmar o sucesso ou registrar um erro, se houver.

Para ver os dados de uma requisição feita com Axios no Chrome DevTools, siga estas etapas abaixo:

1. Abra o Chrome DevTools:
    a. Clique com o botão direito do mouse na página e selecione "Inspecionar" ou pressione Ctrl + Shift + I (ou Cmd + Option + I no Mac) para abrir o DevTools.
2. Vá para a aba “Rede”:
    a. Esta aba mostra todas as requisições feitas pelo seu navegador, incluindo requisições HTTP feitas por Axios.
3. Recarregue a página:
    a. Se a requisição é feita automaticamente na montagem do componente (como no caso do uso do **useEffect**), você pode precisar recarregar a página para que a requisição apareça na lista.
4. Encontre a requisição na listagem:
    a. Procure a requisição correspondente à URL que você usou no Axios (https://reqres.in/api/users).
    b. Você pode usar o campo de filtro no canto superior esquerdo da aba "Rede" para facilitar a busca.
5. Visualize os detalhes da requisição:
    a. Clique na requisição desejada para ver mais detalhes. Isso abrirá uma nova seção com várias guias, como "**Cabeçalhos**", "**Payload**", "**Resposta**" e "**Visualização**":
        i. **Headers:** Mostra os cabeçalhos da requisição e da resposta, incluindo informações como URL, método (GET, POST), status da resposta e outros detalhes.
        ii. **Payload:** Exibe os dados enviados com a requisição (neste caso, os dados do usuário que você está criando).
        iii. **Resposta:** Mostra a resposta que o servidor retornou para a sua requisição, que pode incluir os dados do usuário criado.
        iv. **Visualização:** Apresenta uma visualização mais amigável e organizada da resposta do servidor.

Abra o Chrome DevTools para observar a aba **Rede**, guia **Visualização**:

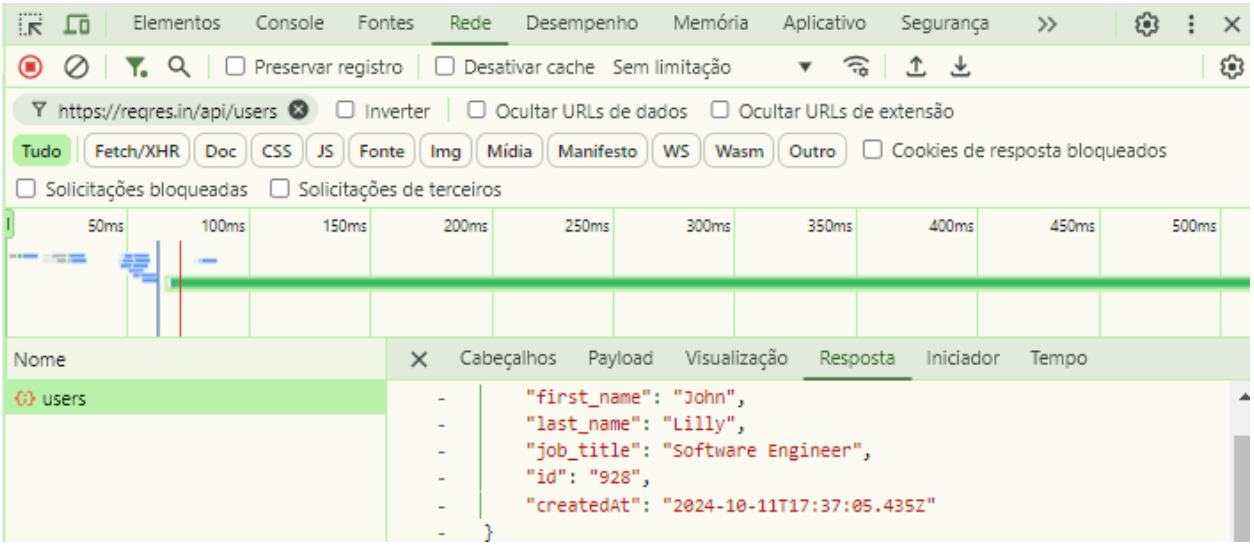

Figura 4: Guia Visualização da aba Rede

Observe que a API Reqres nos retorna os dados incluindo um **ID** gerado e a **data/hora** em que o registro foi criado.

1.5 Passo 4: Excluindo recursos da API

Para excluir um usuário, utilizamos uma requisição HTTP DELETE com o método **axios.delete()**. Nesse tipo de requisição, não é necessário enviar dados no corpo (**data**), pois ela remove o recurso especificado diretamente pela URL.

Vamos simular a exclusão do usuário com **ID=2** na API Reqres. De acordo com a documentação, precisamos utilizar a seguinte URL: **https://reqres.in/api/users/2**. Para isso, crie o arquivo **MyDelete.jsx** com o seguinte código:

```
import { useEffect } from 'react';
import axios from 'axios';
function MyDelete() {
    useEffect(() => {
        const deleteUser = async () => {
            try {
                const response = await
axios.delete('https://reqres.in/api/users/2');
                console.log('Usuário deletado com sucesso:', response.status);
            } catch (error) {
```

```
                console.error('Erro ao deletar usuário:', error);
            }
        };
        deleteUser();
    }, []);

    return <p>Teste de DELETE com API Axios</p>;
}
export default MyDelete;
```

Abra o Chrome DevTools para observar a aba **Rede**, guia **Cabeçalho**:

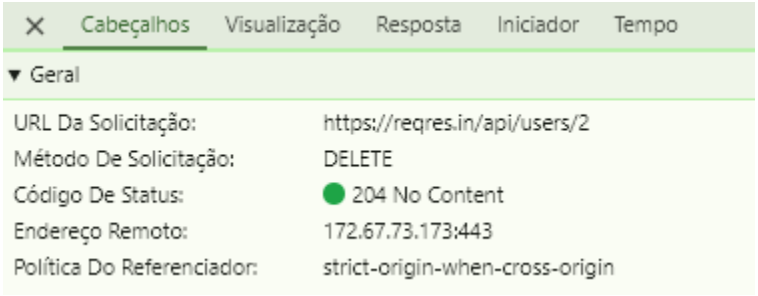

Figura 5: Guia Cabeçalho da aba Rede

O código de status **HTTP 204 No Content** indica que a requisição foi bem-sucedida e que o servidor processou a requisição, mas não está retornando nenhum conteúdo no corpo da resposta. Esse código é frequentemente utilizado em respostas de requisições DELETE, para confirmar que o recurso foi deletado com sucesso.

1.6 Passo 5: Alternando recursos da API

Para alterar recursos utilizando Axios, podemos fazer uso dos métodos HTTP **PUT** e **PATCH**. O método **axios.put() substitui todos os dados** atuais do recurso de destino pelos dados fornecidos na requisição, ou seja, realiza uma atualização completa do recurso. Em outras palavras, ele sobrescreve completamente as informações existentes.

Por outro lado, o método **axios.patch()** é utilizado para aplicar **atualizações parciais** a um recurso, modificando apenas as partes específicas que foram fornecidas na requisição, sem alterar o restante dos dados. Portanto, recomenda-se usar o método PUT quando você precisar atualizar todo o recurso e PATCH quando desejar alterar apenas um subconjunto dos dados.

Vamos realizar uma atualização parcial do usuário com ID = 2 na API Reqres, modificando apenas o primeiro nome e o cargo do usuário. Para isso, crie o arquivo chamado **MyUpdate.jsx** com o seguinte código:

```
import { useEffect } from 'react';
import axios from 'axios';

function MyUpdate() {
    useEffect(() => {
        const updateUser = async () => {
            const user = {
                first_name: 'morpheus',
                job_title: 'zion resident',
            };
            try {
                const response = await
axios.patch('https://reqres.in/api/users/2', user);
                console.log('Usuário atualizado:', response.data);
            } catch (error) {
                console.error('Erro ao atualizar usuário:', error);
            }
        };
        updateUser();
    }, []);

    return <p>Teste de PATCH com API Axios</p>;
}
export default MyUpdate;
```

Abra o Chrome DevTools para observar a aba **Rede**, guia **Visualização**:

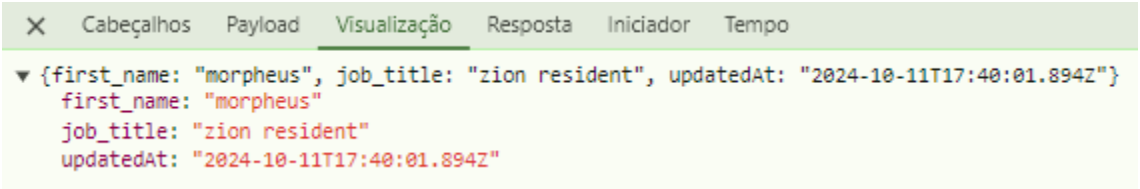

Figura 6: Guia Visualização da aba Rede

Observe que a API Reqres nos retorna os dados enviados, incluindo a informação sobre a data e hora da última modificação.

1.7 Passo 6: Realizando requisições simultâneas

Um dos principais benefícios do uso da API Axios é a capacidade de executar requisições HTTP em paralelo. Para isso, basta passar um array contendo várias URLs para o método **axios.all()**. Esse método retorna uma única Promise, que é resolvida apenas quando todas as requisições incluídas foram concluídas. Ao final, você receberá um array com os objetos de resposta, na mesma ordem em que as requisições foram feitas.

Por exemplo, se quisermos obter dados dos usuários com **ID=1** e **ID=2**, podemos crie o arquivo **MyAll.jsx** com o seguinte código:

```
import { useEffect } from 'react';
import axios from 'axios';
function MyAll() {
    useEffect(() => {
        const fetchUsers = async () => {
            try {
                const responses = await axios.all([
                    axios.get('https://reqres.in/api/users/1'),
                    axios.get('https://reqres.in/api/users/2')
                ]);
                // Extraindo os dados das respostas
                const [user1, user2] = responses.map(response => response.data);

                console.log('Usuário 1:', user1);
                console.log('Usuário 2:', user2);
            } catch (error) {
                console.error('Erro ao buscar usuários:', error);
            }
        };
        fetchUsers();
    }, []);
    return <p>Teste de ALL com API Axios</p>;
}
export default MyAll;
```

Observe que extraímos os dados das respostas usando **map**, para facilitar a leitura.

Abra o Chrome DevTools para observar o resultado da execução da nossa aplicação:

Figura 7: Visualização das duas respostas obtidas no console