

1. Consumindo uma API RESTful com React Hooks

Nesta atividade prática, vamos trabalhar com os conceitos de sobre como consumir uma API RESTful com React Hooks.

1.1 Introdução

Com o advento do React Hooks, nós podemos ter uma lógica gerenciamento de estado em componentes funcionais - o que nos permite ter componentes funcionais como "contêineres". Desta forma, se quisermos que um componente carregue dados em sua montagem (*mounting*), não precisamos mais de um componente de classe. Pelo menos, não apenas porque precisamos de gerenciamento de estado ou de eventos de ciclo de vida (*lifecycle events*).

Existem hooks que nos auxiliam nestas tarefas:

- `useState` e `useReducer`: nos permite adicionar funcionalidade de gerenciamento de estado ao nosso componente.
- `useEffect`: nos dá a possibilidade de executar os chamados efeitos colaterais (*side effects*), tais como: buscar dados de uma API de forma assíncrona.

Em seguida vamos desenvolver um exemplo básico de um componente React que carrega e exibe uma lista de usuários obtidos de uma API RESTful. Assim, buscaremos usuários aleatórios do [JSONPlaceholder](#), uma API RESTful online que oferece dados fake para teste. Como resultado, temos uma resposta JSON similar a esta:

```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
```

```
    "name": "Romaguera-Crona",  
    "catchPhrase": "Multi-layered client-server neural-net",  
    "bs": "harness real-time e-markets"  
  },  
  ...  
]
```

1.2 Criando um Projeto React

Vamos utilizar um novo projeto nesta aula. Assim, vamos criar um projeto chamado my-api-hook, digitando o seguinte comando no terminal:

```
yarn create react-app my-api-hook
```

O processo leva alguns segundos e, logo após terminar, digite o seguinte comando para entrar no diretório recém criado do nosso projeto:

```
cd my-api-hook
```

Existem várias opções disponíveis para adicionar o Axios ao seu projeto, dependendo do seu gerenciador de pacotes. Estamos usando o Yarn, assim, digite a seguinte linha de comando:

```
yarn add axios
```

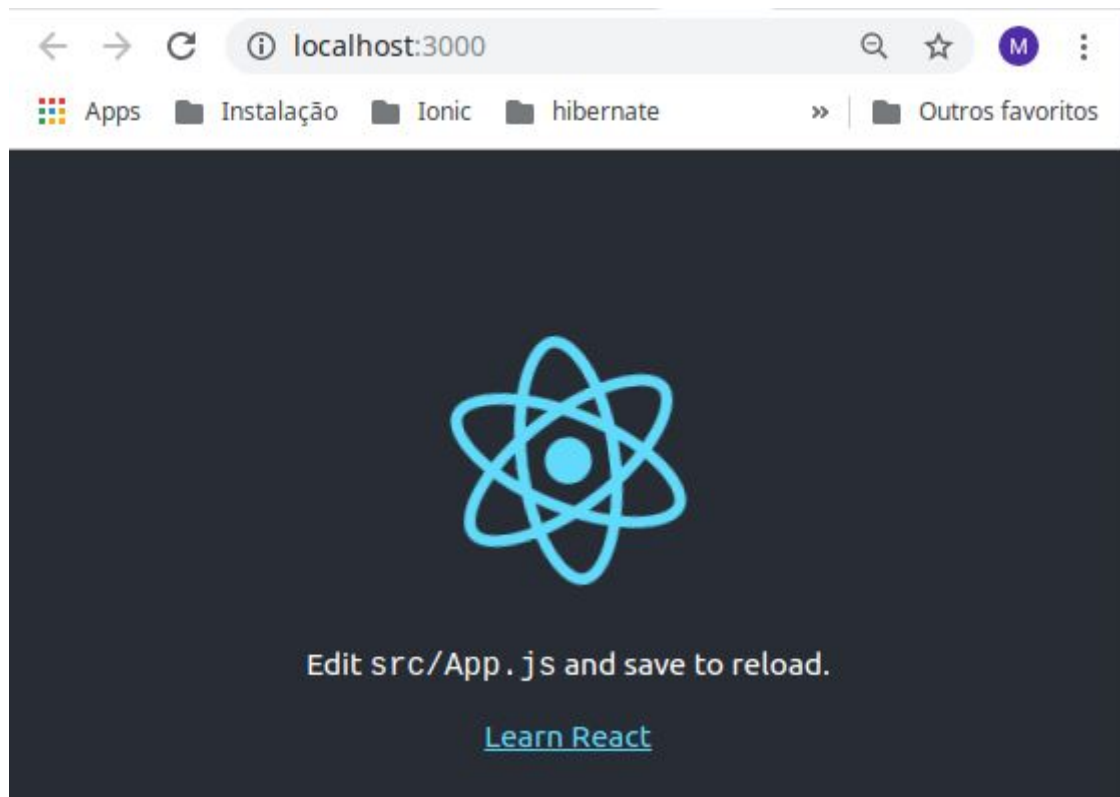
Você pode abrir a aplicação no editor Visual Studio Code. Para isso, dentro do diretório criado digite o seguinte comando:

```
code .
```

O Visual Studio Code deverá abrir com a nova pasta sendo acessada. Agora, vamos testar nossa aplicação. Digite a seguinte linha de comando:

```
yarn start
```

Ele irá rodar a aplicação em modo desenvolvimento em <http://localhost:3000>



A página será recarregada automaticamente se você fizer alterações no código.

1.3 Consumindo uma API RESTful com React Hooks

Vamos ver um exemplo prático de como consumir dados vindos de uma API RESTful com React Hooks.

No arquivo App.js, insira o seguinte código:

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

function App() {
  // Definimos os estados necessários para o componente
  const [data, setData] = useState([]);

  useEffect(() => {
    // Busca os usuários da API JSONPlaceholder
    axios
      .get("https://jsonplaceholder.typicode.com/users")
      .then(result => setData(result.data))
  }, []);
```

```

return (
  <div>
    <h3>Consulta à API JSONPlaceholder</h3>
    <ul>
      {data.map(item => (
        <li key={item.id}>
          {item.username}: {item.name}
        </li>
      ))}
    </ul>
  </div>
);
}

export default App;

```

Vejam os a seguir como criamos este código.

1.3.1 Hook useState

O hook `useState` usa a atribuição via desestruturação (*destructuring assignment*) para fornecer o valor atual de um estado (que chamamos de `data`) e uma função para definir um novo valor de estado (que chamamos de `setData`). O argumento que passamos para `useState` é o valor inicial para `data` no estado do componente.

```
const [data, setData] = useState([]);
```

Logo, você pode imaginar que agora o estado do componente está assim:

```

{
  data: []
}

```

Então, agora temos um estado em nosso componente. Mas, precisamos definir como obter os dados da API RESTful nesse campo. Para esta tarefa, precisamos usar o hook `useEffect`.

1.3.2 Hook useEffect

A cada renderização do componente, o hook `useEffect` executa a função que passamos como o primeiro argumento. Por questão de performance, não queremos carregar os usuários vindos da API cada vez que o componente é renderizado novamente. Por isso, passamos um array vazio como segundo argumento.

O segundo argumento de `useEffect` nos permite passar dependências para o Hook. Por exemplo, um valor no estado ou uma string usada para filtrar os dados que serão retornados da API. Cada vez que uma dessas dependências mudar, o hook `useEffect` será executado novamente. Neste exemplo simples, como queremos buscar os dados apenas uma vez, passamos um array vazio.

```
useEffect(() => {  
  axios  
    .get("https://jsonplaceholder.typicode.com/users")  
    .then(result => setData(result.data));  
}, []);
```

Veja que dentro da *Promise* que recebemos de `axios`, nós atualizamos o estado de nosso componente (usando `setData`) com os dados recém buscados da API (`result.data`).

1.3.3 Carregando os dados na tela

Uma vez que buscamos os dados da API, vamos carregá-los na tela em formato de uma lista. Veja que criamos uma lista não ordenada (elemento ``) e usamos a função `map()` do Javascript para percorrer os valores vindos do array `data`. Retornamos um elemento `` para cada item de `data`. Ainda, cabe lembrar que `key` é um atributo especial que você precisa definir ao criar listas de elementos.

```
<ul>  
  {data.map(item => (  
    <li key={item.id}>  
      {item.username}: {item.name}  
    </li>  
  ))}  
</ul>
```

Veja o resultado no navegador:

Consulta à API JSONPlaceholder

- Bret: Leanne Graham
- Antonette: Ervin Howell
- Samantha: Clementine Bauch
- Karianne: Patricia Lebsack
- Kamren: Chelsey Dietrich
- Leopoldo_Corkery: Mrs. Dennis Schulist
- Elwyn.Skiles: Kurtis Weissnat
- Maxime_Nienow: Nicholas Runolfsdottir V
- Delphine: Glenna Reichert
- Moriah.Stanton: Clementina DuBuque

Funcionou! Porém, vamos fazer alguns ajustes neste código a seguir.

1.3.4 Usando funções async/await

Uma vez que o `Axios` é uma API baseada em Promises, podemos usar a sintaxe assíncrona do ES2017: a funcionalidade `async/await`. A proposta das funções `async/await` é de simplificar o uso de forma síncrona das Promises e executar alguns procedimentos em um grupo de Promises. Dessa forma, uma função assíncrona pode conter uma expressão `await`, que pausa a execução da função assíncrona e espera pela resolução da Promise passada, e depois retoma a execução da função assíncrona e retorna o valor resolvido.

Altere o arquivo `App.js` com o seguinte código:

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

function App() {
  // Definimos os estados necessários para o componente
  const [data, setData] = useState([]);
  // Busca os usuários da API JSONPlaceholder
  async function listUsers () {
    try {
      const result = await
        axios.get('https://jsonplaceholder.typicode.com/users');
      // Atualiza o estado de 'data' com o resultado da API
      setData(result.data);
    } catch (err) {
      console.error(err);
    }
  };
  // A cada renderização do componente, o hook useEffect
```

```

    //executa a função listUsers().
    useEffect(() => {listUsers()}, []);
    return (
      <div>
        <h3>Consulta à API JSONPlaceholder</h3>
        <ul>
          {data.map(item => (
            <li key={item.id}>
              {item.username}: {item.name}
            </li>
          ))}
        </ul>
      </div>
    );
  }
}
export default App;

```

Perceba que o `await` só poderá ser utilizado em funções marcadas com a palavra chave `async`. Ele funciona suspendendo a execução em seu contexto até que a Promise seja entregue.

Veja que o resultado no navegador não mudou.

Consulta à API JSONPlaceholder

- Bret: Leanne Graham
- Antonette: Ervin Howell
- Samantha: Clementine Bauch
- Karianne: Patricia Lebsack
- Kamren: Chelsey Dietrich
- Leopoldo_Corkery: Mrs. Dennis Schulist
- Elwyn.Skiles: Kurtis Weissnat
- Maxime_Nienow: Nicholas Runolfsdottir V
- Delphine: Glenna Reichert
- Moriah.Stanton: Clementina DuBuque

Em seguida, vamos criar nosso próprio hook.

1.3.5 Escrevendo seu próprio hook

Começaremos escrevendo nosso hook para buscar os usuários da API JSONPlaceholder. Assim, crie o diretório `src/hooks`. Dentro dele crie o arquivo `useFindUsers.js` com o seguinte código:

```

import { useEffect, useState } from 'react';
import axios from "axios";

export function useFindUsers () {

  // Definimos os estados necessários para o nosso hook
  const [data, setData] = useState([]);
  // Busca os usuários da API JSONPlaceholder
  async function listUsers () {
    try {
      const result = await
        axios.get('https://jsonplaceholder.typicode.com/users');
      // Atualiza o estado de 'data' com o resultado da API
      setData(result.data);
    } catch (err) {
      console.error(err);
    }
  };
  // A cada renderização do componente, o hook useEffect
  //executa a função listUsers().
  useEffect(() => {listUsers()}, []);

  return data;
}

```

Observe que o hook `useFindUsers()` é apenas uma simples função que podemos exportar para outros componentes, tal como o `App.js`. Em seguida, nós definimos os estados necessários para o nosso hook com o código

```
const [data, setData] = useState([]);
```

Em seguida, com a função assíncrona `listUsers()`, nós buscamos os usuários vindos da API JSONPlaceholder usando a API Axios. E, finalmente, atualizamos o estado de 'data' com o resultado da API

```
setData(result.data);
```

Cabe lembrar que o hook `useEffect` pode ser comparado aos *lifecycles events* `componentDidMount`, `componentDidUpdate` e `componentDidUnmount` dos componentes de classe.

Agora que temos o hook `useFindUsers()`, podemos usá-lo em um componente da nossa aplicação, em nosso caso, vamos usar em `App.js`:


```

import React from 'react';
import { useFindUsers } from '../hooks/useFindUsers';

function App() {
  // use seu próprio hook para carregar os dados da API
  const data = useFindUsers()
  return (
    <div>
      <h3>Consulta à API JSONPlaceholder</h3>
      <ul>
        {data.map(item => (
          <li key={item.id}>
            {item.username}: {item.name}
          </li>
        ))}
      </ul>
    </div>
  );
}
export default App;

```

Veja que o resultado no navegador não mudou. Está tudo funcionando.

Consulta à API JSONPlaceholder

- Bret: Leanne Graham
- Antonette: Ervin Howell
- Samantha: Clementine Bauch
- Karianne: Patricia Lebsack
- Kamren: Chelsey Dietrich
- Leopoldo_Corkery: Mrs. Dennis Schulist
- Elwyn.Skiles: Kurtis Weissnat
- Maxime_Nienow: Nicholas Runolfsdottir V
- Delphine: Glenna Reichert
- Moriah.Stanton: Clementina DuBuque

1.3.6 Melhorando seu código

Uma vez que nosso hook `useFindUsers()` está funcionando, podemos implementar algumas melhorias. Por exemplo, podemos tratar o momento em que a API JSONPlaceholder foi chamada e ainda não retornou um dado. Também podemos melhorar o tratamento de erros vindos esta API.

```

const [loading, setLoading] = useState(false)
const [error, setError] = useState(null)

```

Assim, altere o arquivo useFindUsers.js com o seguinte código:

```
import { useEffect, useState } from 'react';
import axios from "axios";

// O hook é apenas uma simples função que podemos exportar
export function useFindUsers () {

  // Definimos os estados necessários para o nosso hook, isso inclui:
  // usuários (data), estado de carregamento (loading) e erros (error)
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  // Busca os usuários da API JSONPlaceholder
  async function listUsers () {
    try {
      const result = await
        axios.get('https://jsonplaceholder.typicode.com/users');

      // Atualiza o estado de 'loading'
      setLoading(false);
      // Atualiza o estado de 'data' com o resultado da API
      if (result.data) {
        setData(result.data);
      } else {
        setData([]);
      }
    }

    catch (err) {
      // Atualiza o estado de 'error', 'data' e de 'loading'
      setError("Lamento, ocorreu um erro!");
      setData([]);
      setLoading(false);
    }
  };

  // A cada renderização do componente, o hook useEffect
  // executa a função listUsers().
}
```

```

useEffect(() => {
  // Primeiro, definimos os estados de loading e error
  setLoading(true);
  setError(null);

  // executa a função listUsers().
  listUsers();
}, []);

return { data, loading, error };
}

```

Veja que nós definimos os estados necessários para o nosso hook, isso inclui: usuários (data), estado de carregamento (loading) e erros (error). Ainda, dentro do hook `useEffect()`, primeiro, nós definimos os estados de loading e error e, depois, executamos a função `listUsers()`.

Em seguida, altere o arquivo `App.js` da seguinte forma:

```

import React from 'react';
import { useFindUsers } from './hooks/useFindUsers';

function App() {
  // use seu próprio hook para carregar os dados da API
  const { data, loading, error } = useFindUsers()

  if (loading) return <div>Carregando...</div>
  if (error) return <div>{error}</div>

  return (
    <div>
      <h3>Consulta à API JSONPlaceholder</h3>
      <ul>
        {
          data &&
          data.length > 0 &&
          data.map(item => (
            <li key={item.id}>
              {item.username}: {item.name}
            </li>
          ))
        }
      </ul>
    </div>
  )
}

```

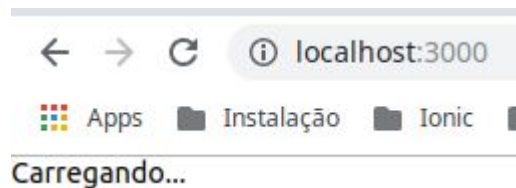
```

    </ul>
  </div>
);
}
export default App;

```

Observe que fazemos uso da renderização condicional para mostrar os dados vindos da API dentro do elemento ``.

Veja que o resultado no navegador mudou, um pouco. Ao carregar o componente pela primeira vez, recebemos a mensagem de “Carregando...”



Em seguida, ao receber os resultados da API JSONPlaceholder, a tela é atualizada.

Consulta à API JSONPlaceholder

- Bret: Leanne Graham
- Antonette: Ervin Howell
- Samantha: Clementine Bauch
- Karianne: Patricia Lebsack
- Kamren: Chelsey Dietrich
- Leopoldo_Corkery: Mrs. Dennis Schulist
- Elwyn.Skiles: Kurtis Weissnat
- Maxime_Nienow: Nicholas Runolfsdottir V
- Delphine: Glenna Reichert
- Moriah.Stanton: Clementina DuBuque

Agora, vamos testar chamar a API com o URL errado:

```

const result = await
  axios.get('https://jsonplaceholderERRADO.typicode.com/users');

```

Veja o resultado no navegador:

Lamento, ocorreu um erro!

Ou seja, ele entrou no tratamento de erros de nosso código, o método `catch()`.

Volte a URL para a correta, ou seja:

```

const result = await
  axios.get('https://jsonplaceholder.typicode.com/users');

```

1.3.7 Criando o componente Users

Vamos mover o código que trata sobre a listagem de usuários vindos da API para um novo componente. Assim, crie o diretório `src/components` e, em seguida, crie o arquivo `Users.js` neste novo diretório.

```
import React from 'react';
import { useFindUsers } from '../hooks/useFindUsers';

function Users() {

  // use seu próprio hook para carregar os dados da API
  const { data, loading, error } = useFindUsers();

  return (
    <div>
      {loading ? <p>Carregando...</p> :
        <ul>
          {
            data &&
            data.length > 0 &&
            data.map(item => (
              <li key={item.id}>
                {item.username}: {item.name}
              </li>
            ))
          }
        </ul>
        {error ? <p>{error}</p> : null}
      </div>
    );
  }

  export default Users;
```

Agora, vamos alterar o arquivo `App.js` para instanciar o novo componente `Users.js`.

```
import React from 'react';
import Users from '../components/Users';

function App() {
  return (
```

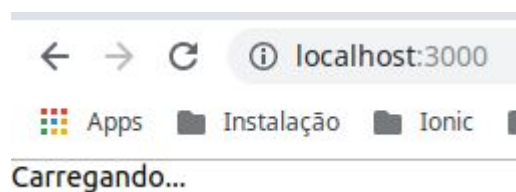
```

<div>
  <h3>Consulta à API JSONPlaceholder</h3>
  <Users />
</div>
);
}

export default App;

```

Veja que o resultado no navegador não mudou, ou seja, ao carregar o componente pela primeira vez, recebemos a mensagem de “Carregando...”



Em seguida, ao receber os resultados da API JSONPlaceholder, a tela é atualizada.

Consulta à API JSONPlaceholder

- Bret: Leanne Graham
- Antonette: Ervin Howell
- Samantha: Clementine Bauch
- Karianne: Patricia Lebsack
- Kamren: Chelsey Dietrich
- Leopoldo_Corkery: Mrs. Dennis Schulist
- Elwyn.Skiles: Kurtis Weissnat
- Maxime_Nienow: Nicholas Runolfsdottir V
- Delphine: Glenna Reichert
- Moriah.Stanton: Clementina DuBuque

1.3.8 Passando parâmetros para o hook useFindUsers

Uma vez que nosso hook `useFindUsers()` está funcionando, vamos considerar que você deseja procurar por um usuário específico da lista. Vamos alterar o arquivo `useFindUsers.js` para permitir isso.

```

import { useEffect, useState } from 'react';
import axios from "axios";

// O hook é apenas uma simples função que podemos exportar
export function useFindUsers(search) {

```

```
// Definimos os estados necessários para o nosso hook, isso inclui:
// usuários (data), estado de carregamento (loading) e erros (error)
const [data, setData] = useState([]);
const [loading, setLoading] = useState(false);
const [error, setError] = useState(null);

// Busca os usuários da API JSONPlaceholder
async function listUsers() {
  try {
    // Altera a apiUrl de acordo com a string 'search'
    const apiUrl = search ?
      `https://jsonplaceholder.typicode.com/users/${search}` :
      'https://jsonplaceholder.typicode.com/users';

    const result = await axios.get(apiUrl);
    // Atualiza o estado de 'loading'
    setLoading(false);
    // Atualiza o estado de 'data' com o resultado da API
    if (result.data) {
      //Caso esteja sendo feito uma pesquisa, retorna apenas 1 registro
      if(search!=='){
        //Adiciona a resposta em um array
        let res = [];
        res.push(result.data);
        //Atualiza o estado de 'data'
        setData(res);
      }else{
        //Senão, atualiza o estado de 'data' com o array vindo da API
        setData(result.data);
      }
    }

  } else {
    setData([]);
  }
}

catch (err) {
  // Atualiza o estado de 'error', 'data' e de 'loading'
  setError("Lamento, ocorreu um erro!");
}
```

```

        setData([]);
        setLoading(false);
    }
};

// A cada renderização do componente, o hook useEffect
// executa a função listUsers().
useEffect(() => {
    // Primeiro, definimos os estados de loading e error
    setLoading(true);
    setError(null);

    // executa a função listUsers().
    listUsers();

    //Passamos o novo parâmetro search atualizar o useEffect.
}, [search]);

return { data, loading, error };
}

```

Neste caso, adicionamos o parâmetro `search` para o hook `useFindUsers()`. De acordo com o valor passado em `search`, vamos chamar a devida URL para a API JSONPlaceholder. Ainda, de acordo com esta API, ao pesquisar um usuário específico (via ID), ele retorna um objeto contendo os dados deste usuário. Porém, quando não informamos nenhum ID, ela retorna um array com os dados de todos os usuários. Para mantermos um padrão de retorno no hook `useFindUsers()`, quando filtramos por um usuário específico, estamos adicionando este resultado a um array vazio, para melhor trabalharmos no componente `Users.js`

```

...
//Adiciona a resposta em um array
let res = [];
res.push(result.data);

```

Observe, e isso é muito importante, que em `useEffect()`, nós passamos o novo parâmetro `search` para o array que tínhamos antes (no segundo argumento desta função). Isso significa que o método `useEffect()` será executado novamente se este parâmetro for alterado.

```

...
listUsers();
//Passamos o novo parâmetro search para

```



```
// o array que tínhamos antes. Isso significa que o useEffect()  
// será executado novamente se este parâmetro for alterado  
}, [search]);  
...
```

Agora, vamos alterar o código de Users.js.

```
import React, {useState} from 'react';  
import { useFindUsers } from '../hooks/useFindUsers';  
  
function Users() {  
  
  //Cria um estado para o conteúdo da pesquisa  
  const [search, setSearch] = useState('');  
  
  // use seu próprio hook para carregar os dados da API  
  const { data, loading, error } = useFindUsers(search);  
  
  return (  
    <div>  
      {loading ? <p>Carregando...</p> :  
        <ul>  
          {  
            data &&  
            data.length > 0 &&  
            data.map(item => (  
              //Atualiza o estado de search para o id selecionado na lista  
              <li key={item.id} onClick={() => setSearch(item.id)}>  
                {item.username}: {item.name}  
              </li>  
            ))  
          }  
        </ul>  
        {error ? <p>{error}</p> : null}  
      </div>  
    );  
  }  
  
  export default Users;
```

Veja que ao clicar em um item da lista, atualizamos o estado de `search`, que, consequentemente, chama o `useEffect()` e, se este valor mudou, chama a API `JSONPlaceholder`.

```
<li key={item.id} onClick={() => setSearch(item.id)}>
```

Veja o resultado no navegador.

Consulta à API `JSONPlaceholder`

- Bret: Leanne Graham
- Antonette: Ervin Howell
- Samantha: Clementine Bauch
- Karianne: Patricia Lebsack
- Kamren: Chelsey Dietrich
- Leopoldo_Corkery: Mrs. Dennis Schulist
- Elwyn.Skiles: Kurtis Weissnat
- Maxime_Nienow: Nicholas Runolfsdottir V
- Delphine: Glenna Reichert
- Moriah.Stanton: Clementina DuBuque

Ao clicarmos em Karianne: Patricia Lebsack e, desta forma, temos o seguinte resultado na tela:

Consulta à API `JSONPlaceholder`

- Karianne: Patricia Lebsack

Funcionou!

1.3.9 Melhorando o State

Observe que em nosso hook `useFindUsers()` estamos fazendo o gerenciamento de estados para tratar das informações de usuários, estado de carregamento e erros da seguinte forma:

```
// Definimos os estados necessários para o nosso hook, isso inclui:  
// usuários (data), estado de carregamento (loading) e erros (error)  
const [data, setData] = useState([]);  
const [loading, setLoading] = useState(false);  
const [error, setError] = useState(null);
```

Podemos agrupar esta informação em um objeto único, uma vez que todos estão tratando sobre as chamadas a uma API externa.

```
// Definimos os estados necessários para o nosso hook, isso inclui:  
// usuários (data), estado de carregamento (loading) e erros (error)  
const [state, setDataState] =  
  useState({loading: true, data: [], error: null});
```

Logo, altere o arquivo useFindUsers.js para o seguinte código:

```
import { useEffect, useState } from 'react';
import axios from "axios";

// O hook é apenas uma simples função que podemos exportar
export function useFindUsers(search) {

  // Definimos os estados necessários para o nosso hook, isso inclui:
  // usuários (data), estado de carregamento (loading) e erros (error)
  const [state, setDataState] = useState({loading: true,data: [],error:
null});

  // Busca os usuários da API JSONPlaceholder
  async function listUsers() {
    try {
      // Altera a apiUrl de acordo com a string 'search'
      const apiUrl = search ?
        `https://jsonplaceholder.typicode.com/users/${search}` :
        'https://jsonplaceholder.typicode.com/users';

      const result = await axios.get(apiUrl);
      // Atualiza o estado de 'loading'
      setDataState({loading:false});
      // Atualiza o estado de 'data' com o resultado da API
      if (result.data) {
        //Caso esteja sendo feito uma pesquisa, retorna apenas 1 registro
        if(search!==''){
          //Adiciona a resposta em um array
          let res = [];
          res.push(result.data);
          //Atualiza o estado de 'data' e 'loading'
          setDataState({data: res,loading:false});
        }else{
          //Senão, atualiza o estado de 'loading' e 'data' com o array
vindo da API
          setDataState({data:result.data,loading:false});
        }
      }

    } else {
      //Atualiza o estado de 'data' e 'loading'
```

```

        setDataState({data: [],loading:false});
    }

}

catch (err) {
    // Atualiza o estado de 'error', 'data' e de 'loading'
    setDataState({data: [], loading:false, error: "Lamento, ocorreu um
erro!"});
}
};

// A cada renderização do componente, o hook useEffect
// executa a função listUsers().
useEffect(() => {
    // Primeiro, definimos os estados de loading e error
    setDataState({loading:true, error: null});

    // executa a função listUsers().
    listUsers();

    //Passamos o novo parâmetro search atualizar o useEffect.
}, [search]);

return { state };
}

```

Ainda, veja que estamos retornando o este objeto mais complexo, o state.

```
return { state };
```

Em seguida, altere o arquivo User.js:

```

import React, {useState} from 'react';
import { useFindUsers } from '../hooks/useFindUsers';

function Users() {

    //Cria um estado para o conteúdo da pesquisa
    const [search, setSearch] = useState('');

    // use seu próprio hook para carregar os dados da API
    const { state } = useFindUsers(search);

```

```
// Desestructure isLoading, data e error de state
const { data, loading, error } = state;

return (
  <div>
    {loading ? <p>Carregando...</p> :
      <ul>
        {
          data &&
          data.length > 0 &&
          data.map(item => (
            //Atualiza o estado de search para o id selecionado na lista
            <li key={item.id} onClick={() => setSearch(item.id)}>
              {item.username}: {item.name}
            </li>
          ))
        }
      </ul>
    }
    {error ? <p>{error}</p> : null}
  </div>
);
}

export default Users;
```

Veja que estamos desestruturando isLoading, data e error de state.

```
const { data, loading, error } = state;
```

Assim, não precisamos usar state.data ou state.loading ou state.error no código. Veja que não houve alteração do resultado no navegador.

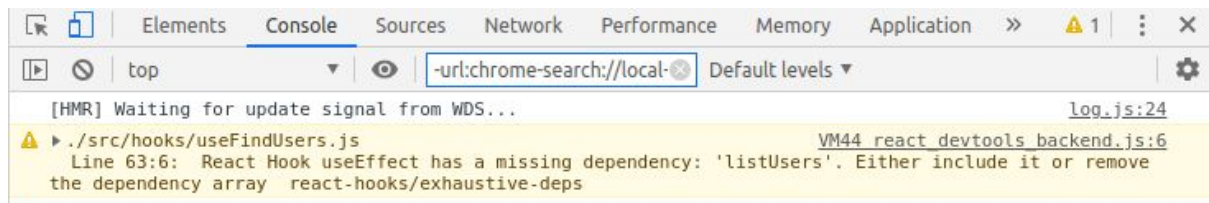
Consulta à API JSONPlaceholder

- Bret: Leanne Graham
- Antonette: Ervin Howell
- Samantha: Clementine Bauch
- Karianne: Patricia Lebsack
- Kamren: Chelsey Dietrich
- Leopoldo_Corkery: Mrs. Dennis Schulist
- Elwyn.Skiles: Kurtis Weissnat
- Maxime_Nienow: Nicholas Runolfsdottir V
- Delphine: Glenna Reichert
- Moriah.Stanton: Clementina DuBuque

Funcionou!!!

1.3.10 Removendo erros de dependência do useEffect

Observe na aba “console” do Google DevTools (clique em F12) que o React está disparando um warning relacionado ao nosso código.



A mensagem “React Hook useEffect has a missing dependency: 'listUsers'. Either include it or remove the dependency array” . Isso ocorre porque o hook `useFindUsers()` depende da função `listUsers()` . Segundo a [documentação oficial](#) do React, é recomendado que você declare as funções necessárias para um efeito (tal como atualizar os dados da tela de acordo com um determinado valor) dentro de `useEffect()` . Assim, fica mais fácil de se ver quais são os valores do escopo do componente que dependem desse efeito.

Logo, vamos alterar o arquivo `useFindUsers.js` da seguinte forma:

```
import { useEffect, useState } from 'react';
import axios from "axios";

// O hook é apenas uma simples função que podemos exportar
export function useFindUsers(search) {

  // Definimos os estados necessários para o nosso hook, isso inclui:
  // usuários (data), estado de carregamento (loading) e erros (error)
  const [state, setDataState] = useState({ loading: true, data: [],
error: null });

  // A cada renderização do componente, o hook useEffect
  // executa a função listUsers().
  useEffect(() => {

    // Ao mover a função listUsers para dentro do useEffect,
    // podemos ver claramente os valores que ela usa.
    async function listUsers() {
      try {
        // Altera a apiUrl de acordo com a string 'search'
        const apiUrl = search ?
`https://jsonplaceholder.typicode.com/users/${search}` :
```

```

    'https://jsonplaceholder.typicode.com/users';

    const result = await axios.get(apiUrl);
    // Atualiza o estado de 'loading'
    setDataState({ loading: false });
    // Atualiza o estado de 'data' com o resultado da API
    if (result.data) {
      //Caso esteja sendo feito uma pesquisa, retorna apenas 1 registro
      if (search !== '') {
        //Adiciona a resposta em um array
        let res = [];
        res.push(result.data);
        //Atualiza o estado de 'data' e 'loading'
        setDataState({ data: res, loading: false });
      } else {
        //Senão, atualiza o estado de 'loading' e 'data' com o array
vindo da API
        setDataState({ data: result.data, loading: false });
      }

    } else {
      //Atualiza o estado de 'data' e 'loading'
      setDataState({ data: [], loading: false });
    }

  }

  catch (err) {
    // Atualiza o estado de 'error', 'data' e de 'loading'
    setDataState({ data: [], loading: false, error: "Lamento,
ocorreu um erro!" });
  }
};

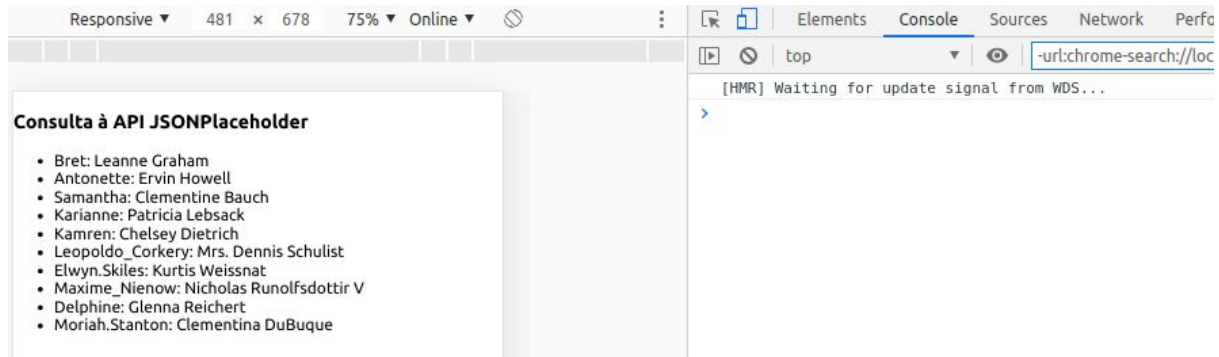
// Primeiro, definimos os estados de loading e error
setDataState({ loading: true, error: null });
// executa a função listUsers().
listUsers();

//Passamos o novo parâmetro search atualizar o useEffect.
}, [search]);

```

```
return { state };
}
```

Assim, ao mover a função `listUsers()` para dentro do `useEffect`, podemos ver claramente os valores que ela usa. Assim, eliminamos o warning disparado pelo React.



1.3.11 Utilizando o hook `useReducer`

Após aprendermos e experimentarmos os conceitos básicos do hook `useState`, vamos analisar como o hook `useReducer` poder ser usado na execução de ações assíncronas, tais como fazer solicitações para uma API externa.

O hook `useReducer` também pode ser usado para atualizar o estado, tal como o `useState`, mas ele o faz de uma maneira mais sofisticada: ele aceita uma função redutora (*reducer function*) e um estado inicial e, depois, retorna o estado real e uma função de despacho (dispatch function). A função `dispatch` altera o estado de maneira implícita, mapeando ações para suas respectivas transições de estado.

```
const [state, dispatch] = useReducer(reducer, initialArg, init);
```

Assim, crie em `src/hooks` o arquivo `useFindUsersReducer.js` com o seguinte código:

```
import { useEffect, useReducer } from 'react';
import axios from "axios";

const initialState = {
  loading: true,
  data: [],
  error: null
}

const reduce = (state, action) => {
```



```

// Atualiza o estado de 'error', 'data' e de 'loading'
// De acordo com o tipo de ação
switch (action.type) {
  case 'OnFetching':
    return {
      loading: true,
      data: [],
      error: null
    }
  case 'OnSuccess':
    return {
      loading: false,
      data: action.payload,
      error: null
    }
  case 'OnFailure':
    return {
      loading: false,
      data: [],
      error: 'Lamento, ocorreu um erro!'
    }
  default:
    return state;
}
}

// O hook é apenas uma simples função que podemos exportar
export function useFindUsersReducer(search) {

  // Definimos os estados necessários para o nosso hook, isso inclui:
  // usuários (data), estado de carregamento (loading) e erros (error)
  const [state, dispatch] = useReducer(reduce, initialState)
  // A cada renderização do componente, o hook useEffect
  // executa a função listUsers().
  useEffect(() => {

    // Ao mover a função listUsers para dentro do useEffect,
    // podemos ver claramente os valores que ela usa.
    async function listUsers() {
      try {

```

```

    // Altera a apiUrl de acordo com a string 'search'
    const apiUrl = search ?
      `https://jsonplaceholder.typicode.com/users/${search}` :
      'https://jsonplaceholder.typicode.com/users';

    const result = await axios.get(apiUrl);

    // Atualiza o estado de 'data' com o resultado da API
    if (result.data) {
      //Caso esteja sendo feito uma pesquisa, retorna apenas 1 registro
      if (search !== '') {
        //Adiciona a resposta em um array
        let res = [];
        res.push(result.data);
        //Atualiza o estado em caso de sucesso
        dispatch({ type: 'OnSuccess', payload: res });
      } else {

        //Senão, atualiza o estado em caso de sucesso com o array
vindo da API
        dispatch({ type: 'OnSuccess', payload: result.data });
      }

    } else {
      //Atualiza o estado em caso de erro
      dispatch({ type: 'OnFailure' });
    }

  }
  catch (err) {
    //Atualiza o estado em caso de erro
    dispatch({ type: 'OnFailure' });
  }
};

// Primeiro, definimos os estados em caso de pesquisa
dispatch({ type: 'OnFetching' });

// executa a função listUsers().
listUsers();

```

```

    //Passamos o novo parâmetro search atualizar o useEffect.
  }, [search]);

  return { state };
}

```

Observe que, inicialmente, nós estruturamos nosso estado.

```

const initialState = {
  loading: true,
  data: [],
  error: null
}

```

Essa estrutura nos permite obter facilmente os dados de status e resposta de uma requisição a uma API externa. Agora que temos nossa estrutura de dados do estado, nós precisávamos implementar três ações.

1. OnFetching - a ação a ser despachada quando uma solicitação de API está em andamento
2. OnSuccess - a ação a ser despachada quando uma solicitação da API é concluída com êxito
3. OnFailure - a ação a ser despachada e solicitações de uma API resulta em erro

Assim, a função `dispatch` altera o estado de maneira implícita, mapeando ações para suas respectivas transições de estado.

```

const reduce = (state, action) => {
  // Atualiza o estado de 'error', 'data' e de 'loading'
  // De acordo com o tipo de ação
  switch (action.type) {
    case 'OnFetching':
      return {
        loading: true,
        data: [],
        error: null
      }
    case 'OnSuccess':
      return {
        loading: false,
        data: action.payload,
        error: null
      }
    case 'OnFailure':

```

```

        return {
            loading: false,
            data: [],
            error: 'Lamento, ocorreu um erro!'
        }
    default:
        return state;
    }
}

```

O objeto que está sendo enviado com a função `dispatch` possui uma propriedade obrigatória `type` e uma propriedade opcional chamada `payload`. O `type` informa à função `reduce` qual transição de estado precisa ser aplicada e o `payload` pode ser usado adicionalmente pela função redutora para definir o novo estado.

```
dispatch({ type: 'OnSuccess', payload: result.data });
```

Após analisarmos o código do hook `useFindUsersReducer`, vamos instanciar este novo hook no componente `Users.js`:

```

import React, {useState} from 'react';
import { useFindUsersReducer } from '../hooks/useFindUsersReducer';

function Users() {

    //Cria um estado para o conteúdo da pesquisa
    const [search, setSearch] = useState('');

    // use seu próprio hook para carregar os dados da API
    const { state } = useFindUsersReducer(search);

    // Desestructure isLoading, data e error de state
    const { data, loading, error } = state;

    return (
        <div>
            {loading ? <p>Carregando...</p> :
                <ul>
                    {
                        data &&
                        data.length > 0 &&
                        data.map(item => (

```

```

        //Atualiza o estado de search para o id selecionado na
lista
        <li key={item.id} onClick={() => setSearch(item.id)}>
            {item.username}: {item.name}
        </li>
    ))
}
</ul>
{error ? <p>{error}</p> : null}
</div>
);
}

export default Users;

```

Finalmente, veja que o comportamento no navegador não alterou.

Consulta à API JSONPlaceholder

- Bret: Leanne Graham
- Antonette: Ervin Howell
- Samantha: Clementine Bauch
- Karianne: Patricia Lebsack
- Kamren: Chelsey Dietrich
- Leopoldo_Corkery: Mrs. Dennis Schulist
- Elwyn.Skiles: Kurtis Weissnat
- Maxime_Nienow: Nicholas Runolfsdottir V
- Delphine: Glenna Reichert
- Moriah.Stanton: Clementina DuBuque

Ao clicarmos em Karianne: Patricia Lebsack, desta forma, temos o seguinte resultado na tela:

Consulta à API JSONPlaceholder

- Karianne: Patricia Lebsack

Funcionou!

1.3.12 Quando usar o hook useState e o useReducer?

Abaixo, veremos alguns cenários onde podemos usar estes hooks:

Cenário	useState()	useReducer()
Tipo de estado	Use ao trabalhar com Number, Boolean e String	Use ao trabalhar com objeto ou array

Número de transições de estado	Deve ser usado quando você tiver apenas 1 ou 2 chamadas setState	Deve ser usado quando tivermos muitas transições de estado
Lógica de negócios	O hook useState não tem lógica de negócios	Quando sua aplicação envolve transformação ou manipulação de dados complexos
Local vs Global	Deve ser usado ao lidar com um único componente e precisar executar operações localmente	Quando quiser lidar com vários componentes, ou seja, para passar dados de um componente para outro.

Referências

- Site Oficial do React. Using the Effect Hook. Disponível em: <https://reactjs.org/docs/hooks-effect.html>
- Site Oficial do React. Hooks FAQ. Disponível em: <https://reactjs.org/docs/hooks-faq.html#is-it-safe-to-omit-functions-from-the-list-of-dependencies>
- Site MDN. Funções assíncronas. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/funcoes_assincronas