



FIAP

Shift



Aula 07

React

Parte 2



O QUE O MATERIAL COBRE

1

REACT HOOKS: ESTADOS

2

REACT HOOKS: EFEITOS

3

ROTAS NO REACT

4

FORMULÁRIOS: REACT-HOOK-FORM

O QUE O MATERIAL COBRE

5

REQUISIÇÕES HTTP: AXIOS + SWR

6

SUGESTÕES DE EXERCÍCIOS

1

React Hooks

Estados

React Hooks: useState (Estados)

Os estados são como variáveis, eles armazenam dados específicos de um componente. Esse dado pode determinar como o componente deve se comportar ao ser renderizado, pode também ser utilizado para exibição de valores, seja com ou sem mutação.

Os estados estão diretamente conectados ao Virtual DOM e também ao Render Tree do React.

É possível utilizar os conceitos de const e let do Javascript, mas eles não podem estar diretamente ligados às mudanças na árvore de renderização.



React Hooks: useState (Estados)

```
import React from 'react';

export default function MyComponent() {
  const [count, setCount] = React.useState(0);

  return (
    <div className="flex gap-3 items-center">
      <div className="italic font-light text-lg text-gray-500">
        Contador: {count}
      </div>
      <button
        className="p-3 rounded bg-blue-500 text-gray-50"
        onClick={() => {
          setCount(10);
        }}
      >
        Atualizar contador para 10
      </button>
      <button
        className="p-3 rounded bg-blue-500 text-gray-50"
        onClick={() => {
          setCount(20);
        }}
      >
        Atualizar contador para 20
      </button>
    </div>
  );
}
```

Para utilizar um estado basta chamar o React Hook `useState()`, **sendo seu parâmetro o valor inicial.**

O retorno é um array com dois índices:

- **Primeiro** – o valor do estado;
- **Segundo** – o valor para atualizar o estado.



React Hooks: useState (Estados)

```
import React from 'react';

export default function MyComponent() {
  const [count, setCount] = React.useState(0);

  return (
    <div className="flex gap-3 items-center">
      <div className="italic font-light text-lg text-gray-500">
        Contador: {count}
      </div>
      <button
        className="p-3 rounded bg-blue-500 text-gray-50"
        onClick={() => {
          setCount((prevValue) => prevValue + 1);
        }}
      >
        Adicionar 1
      </button>
    </div>
  );
}
```

É possível utilizar uma função dentro do estado para coletar o valor anterior e assim ter mais flexibilidade em alterações caso precise, evitando utilizar o valor do estado em si para gerar um novo valor.

Normalmente o parâmetro pode ser chamado de **prevValue**.

2

React Hooks

Efeitos

React Hooks: useEffect (Efeitos)

```
import React from 'react';

export default function MyComponent() {
  const [count, setCount] = React.useState(0);

  React.useEffect(() => {
    setCount(10);
  }, []);

  return (
    <div className="italic font-light text-lg text-gray-500">
      Contador: {count}
    </div>
  );
}
```

Uma das bases principais do React são o **ciclos de vida (life cycles)** que também são conhecidos como **efeitos**. Esses efeitos acontecem com base em modificações geradas pelo componente.

Os efeitos ficam observando mudanças de estados e/ou props, e caso uma mudança aconteça a função do efeito será executada.

No React utilizamos o **useEffect()**:

- Primeiro parâmetro – a função que irá executar no efeito;
- Segundo parâmetro – um array de dependências que gatilham a chamada do efeito.

React Hooks: useEffect (Efeitos) com deps

```
import React from 'react';

export default function MyComponent() {
  const [count, setCount] = React.useState(0);
  const [updateTimes, setUpdateTimes] = React.useState(0);

  React.useEffect(() => {
    setUpdateTimes((prevValue) => prevValue + 1);
  }, [count]);

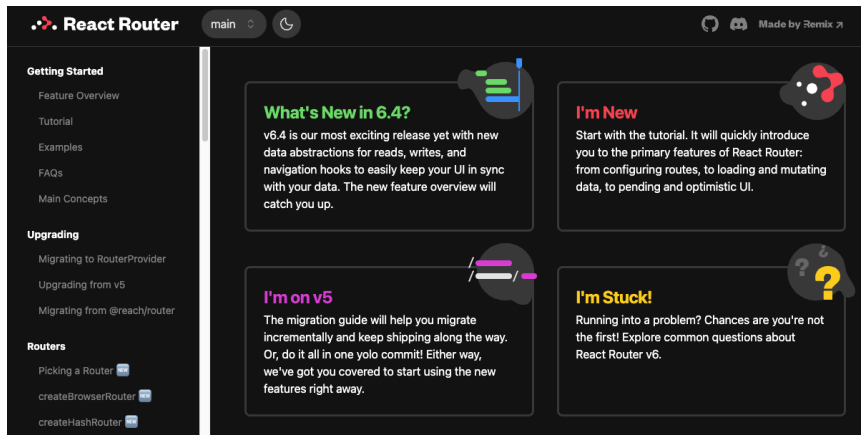
  return (
    <div className="flex gap-3 items-center">
      <div className="italic font-light text-lg text-gray-500">
        Contador: {count} / Atualizações: {updateTimes}
      </div>
      <button
        className="p-3 rounded bg-blue-500 text-gray-50"
        onClick={() => {
          setCount((prevValue) => prevValue + 10);
        }}
      >
        Adicionar 10
      </button>
    </div>
  );
}
```

Ao utilizar uma dependência, sempre que o valor for alterado o efeito será chamado.

3

Rotas no **React**

React Router: biblioteca de rotas



Por se tratar de uma biblioteca, o React não possui um sistema de roteamento acoplado diretamente nele. Felizmente existe uma outra biblioteca incrível que nos permite juntar a parte de renderização com rotas através da barra de navegação do browser.

Com funções e componentes o **React Router (v6)** nos permite criar rotas fixas e dinâmicas, permitindo o envio de parâmetros.

React Router : criando rotas básicas

```
npm i react-router-dom
```

```
import { createBrowserRouter, RouterProvider } from
'react-router-dom';

const router = createBrowserRouter([
  {
    path: '/',
    element: <Root />,
    children: [
      {
        path: 'sobre',
        element: <About />,
      },
      {
        path: '',
        element: <Home />,
      },
    ],
  },
],
);

function App() {
  return <RouterProvider router={router} />;
}
```

O primeiro passo é instalar o pacote via NPM.

Logo depois criar a sequencia de rotas através de uma lista na função **createBrowserRouter()**.

Por último chamar o componente **<RouterProvider />** passando a lista de rotas via props.

Importante entender que o sistema de rotas no React Router funciona como se fosse um **switch/case**, o primeiro match da URL na lista é o que ele vai renderizar.

React Router: criando rotas dinâmicas

```
const router = createBrowserRouter([
  {
    path: '/',
    element: <Root />,
    children: [
      {
        path: 'blog/:slug',
        element: <BlogPost />,
      },
    ],
  },
]);
```

É possível também criar rotas dinâmicas passando parâmetros pela URL, muito útil para coletar dados específicos, por exemplo os detalhes de um post no blog. Basta apenas mudar um pouco o path para receber esse parâmetro.

React Router: useParams()

```
import { useParams } from 'react-router-dom';

export default function BlogPost() {
  const { slug } = useParams();

  return (
    <div className="grid gap-5">
      
      <div className="font-light italic text-lg text-gray-500">
        Parâmetro slug: <span className="font-bold">{slug}</span>
      </div>
    </div>
  );
}
```

Para coletar o parâmetro enviado na URL utilize a função **useParams()** do react router.

+

.

•

•

. . .

React Router: useNavigate()

```
import { useNavigate } from 'react-router-dom';

export default function About() {
  const navigate = useNavigate();

  return (
    <div className="grid gap-5">
      <button
        className="w-28 p-2 rounded bg-blue-500 text-white"
        onClick={() => {
          // Ir para a home
          navigate('/');
        }}
      >
        Voltar
      </button>
    </div>
  );
}
```

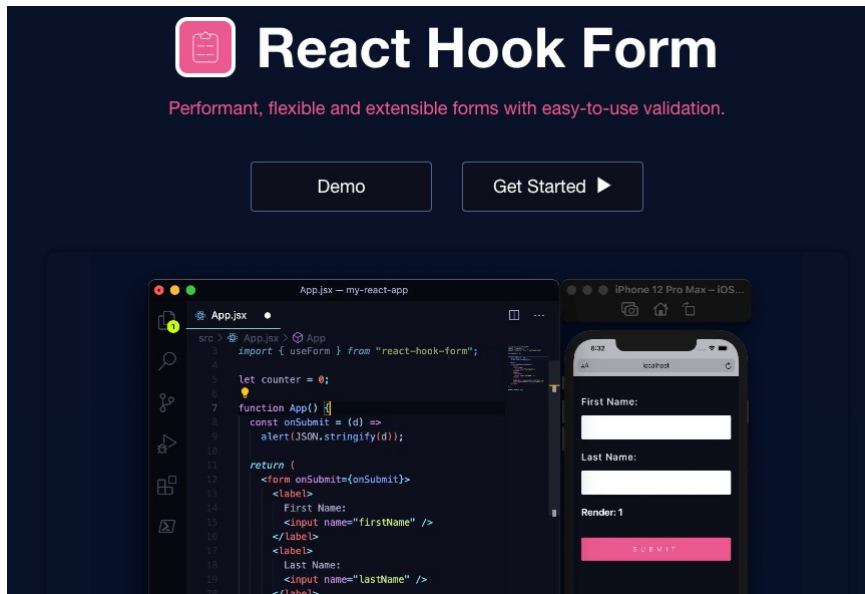
Para trocar de rota sem utilizar o componente **<Link>** utilize a função **useNavigate()** que retorna outra função na qual deve ser passada o parâmetro com a rota desejada.

4

Formulários

React Hook Form

React Hook Form: biblioteca de formulários



Muitos sistemas e aplicações tem formulários e lidar com eles na mão pode ser bem complexo, pois exige muitos estados.

A comunidade criou uma biblioteca para usar com formulários que facilita muito o trabalho chamada **react-hook-form**.

Com essa biblioteca é possível juntar os elementos HTML como form, input e outros e coletar os valores ao submeter o formulário.

Site: <https://react-hook-form.com/>

React Hook Form: exemplo

```
import { useForm } from 'react-hook-form';

export default function Login() {
  const form = useForm();
  const email = form.watch('email');

  function onSubmit(data) {
    console.log('Dados do formulário', data);
  }

  return (
    <form className="grid gap-2 p-4" onSubmit={form.handleSubmit(onSubmit)}>
      <h1 className="text-2xl font-bold">bem vindo: {email}</h1>

      <div className="grid gap-1">
        <input
          className="border"
          type="email"
          {...form.register('email', { required: true })}>
        </input>
        {form.formState.errors.email && <label>Campo obrigatório</label>}
      </div>

      <div className="grid gap-1">
        <input
          className="border"
          type="password"
          {...form.register('password', { required: true })}>
        </input>
        {form.formState.errors.password && <label>Campo obrigatório</label>}
      </div>

      <button type="submit" className="mt-2">
        Entrar
      </button>
    </form>
  );
}
```

Utilizando o useForm da biblioteca é possível inicializar um formulário, existem algumas propriedades principais:

- **register** – registra um campo do formulário e cria uma chave ao fazer a submissão de dados – é possível criar validações nele;
- **handleSubmit** – executa a função de submissão do formulário;
- **formState.errors** – o objeto que contém os erros de validação de cada campo;
- **formState.isValid** –
- **watch** –

React Hook Form: exemplo

```
import { useForm } from 'react-hook-form';

export default function Login() {
  const form = useForm();
  const email = form.watch('email');

  function onSubmit(data) {
    console.log('Dados do formulário', data);
  }

  return (
    <form className="grid gap-2 p-4" onSubmit={form.handleSubmit(onSubmit)}>
      <h1 className="text-2xl font-bold">bem vindo: {email}</h1>

      <div className="grid gap-1">
        <input
          className="border"
          type="email"
          {...form.register('email', { required: true })}>
        </div>
        {form.formState.errors.email && <label>Campo obrigatório</label>}

        <div className="grid gap-1">
          <input
            className="border"
            type="password"
            {...form.register('password', { required: true })}>
          </div>
          {form.formState.errors.password && <label>Campo obrigatório</label>}

        <button type="submit" className="mt-2">
          Entrar
        </button>
      </form>
    );
  }
}
```

E outras:

- **formState.isValid** – é a propriedade que indica se o formulário está válido ou se contém qualquer erro;
- **watch** – a função que permite observar algum valor do formulário.

Utilize essas funções juntamente com um formulário HTML.

5

Requisições HTTP

Axios + SWR

Axios: biblioteca de requisições HTTP

A X I O S

Promise based HTTP client for the browser and node.js

Axios is a simple promise based HTTP client for the browser and node.js. Axios provides a simple to use library in a small package with a very extensible interface.

Get Started

```
import axios from "axios";
axios.get('/users')
  .then(res => {
    console.log(res.data);
  });
```

Get Started

View on GitHub

Com a função fetch é possível fazer requisições HTTP, essa função é mais recente nativa do navegador.

Existe também uma biblioteca muito famosa e utilizada por diversos projetos no mundo toda chamada **axios** e ela também é focada em requisições HTTP facilitando muito o processo.

Axios: instalação e utilização

```
npm i axios
```

```
import React from 'react';
import axios from 'axios';

export default function Request() {
  const [user, setUser] = React.useState(null);

  React.useEffect(() => {
    (async () => {
      const { data } = await axios.get(
        'https://api.github.com/users/guscsales'
      );

      setUser(data);
    })();
  }, []);

  return (
    <pre className="p-4">
      {user ? JSON.stringify(user, null, 2) : 'no data yet'}
    </pre>
  );
}
```

O primeiro passo é instalar a biblioteca via NPM.

Existem algumas maneiras de chamar o axios, a mais simples é **axios.get()** (ou o **nome de outro protocolo**) passando a URL por parâmetro.

Para utilizar o valor da resposta da API é possível combinar com um **useState**.

Axios: utilização

```
import axios from 'axios';
import { useForm } from 'react-hook-form';

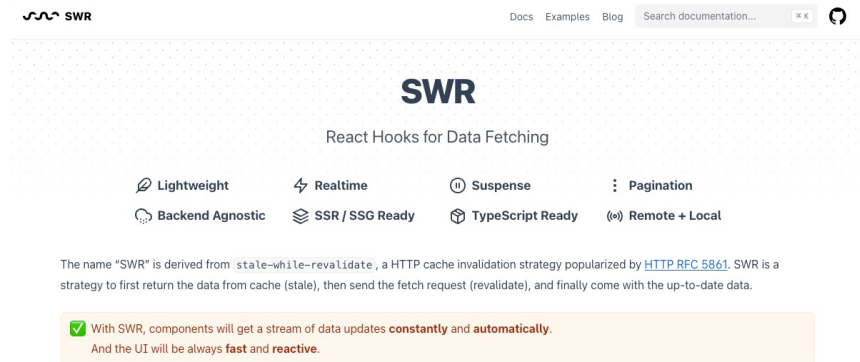
export default function Login() {
  const form = useForm();

  async function onSubmit(data) {
    try {
      await axios.post('/login', {
        email: data.email,
        password: data.password,
      });
    } catch (e) {
      console.error('Erro ao fazer login', e);
    }
  }

  return (
    <form className="grid gap-2 p-4"
      onSubmit={form.handleSubmit(onSubmit)}>
      ...
    </form>
  );
}
```

Assim como é possível fazer requisições com o GET, é possível também fazer com outros protocolos (POST, PUT, PATCH ou DELETE) passando os parâmetros como corpo da requisição.

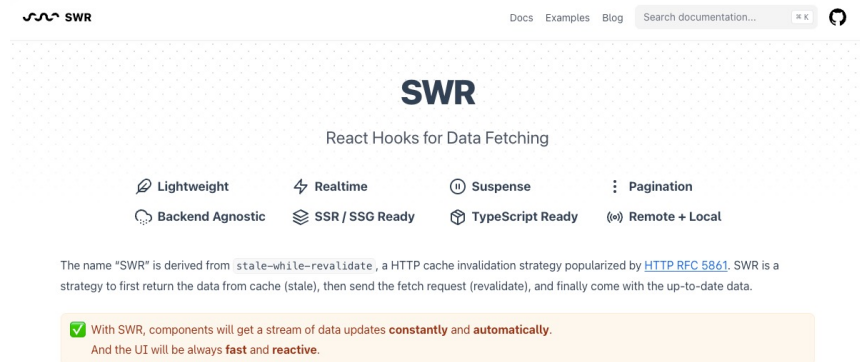
Axios + SWR: não crie estados, utilize-os



Utilizar o axios diretamente com o GET é muito custoso e dificilmente escalável, porque exige a criação de muitos estados intermediários como carregamento, erro e o proprio dado. Normalmente o protocolo que mais é utilizado nas aplicações é o GET.

Pensando nisso podemos utilizar um third party hook chamado **useSWR()** criado pela Vercel.

Axios + SWR: não crie estados, utilize-os



O useSWR utiliza o conceito de cache **SWR (Stale-While-Revalidate)**, que basicamente guarda os dados em cache e recarrega os mesmos em background atualizando assim que tiver a atualização já pronta.

É possível utilizar ele com várias bibliotecas de requisição e o axios é uma delas.

Axios + SWR: instalação e utilização

```
npm i swr
```

```
import useSWR from 'swr';

export default function Request() {
  const {
    data: { data },
    isLoading,
  } =
  useSWR('https://api.github.com/users/guscsales');


  console.log('Dados retornados da API', data);


  return (
    <pre className="p-4">
      {!isLoading ? JSON.stringify(data, null, 2) :
      'no data yet'}
    </pre>
  );
}
```

O useSWR utiliza o conceito de cache **SWR (Stale-While-Revalidate)**, que basicamente guarda os dados em cache e recarrega os mesmos em background atualizando assim que tiver a atualização já pronta. E todos os estados durante a requisição são retornados em um objeto para ser utilizado.

É possível utilizar ele com várias bibliotecas de requisição e o axios é uma delas.

Projeto CEP

 Busca CEP

 CEP 90040001 via Correios

Endereço: Avenida João Pessoa	Bairro: Farroupilha	Cidade: Porto Alegre
Estado: RS	Longitude: -51.2182889	Latitude: -30.0388643

Fazer nova pesquisa

Vamos construir um projeto utilizando os conceitos acima. O objetivo será consumir a API pública para consulta de CEP e de acordo com o número retornar os resultados do endereço em uma segunda página.

+

•

•

•

• • •

Projeto CEP – Tela Inicial

 Busca CEP

Busque qualquer endereço...

Digite o valor do CEP do endereço para coletar mais informações sobre o mesmo.

Exemplo: 01001000

Buscar

+

.

.

.

.

.

.

Projeto CEP – Tela Inicial com Erro

 Busca CEP

Busque qualquer endereço...

Digite o valor do CEP do endereço para coletar mais informações sobre o mesmo.

Exemplo: 01001000

Buscar

Valor inválido, digite um CEP que contenha 8 dígitos

Projeto CEP – Tela de Resultados

 Busca CEP



CEP 90040001 via Correios

Endereço: Avenida João Pessoa

Bairro: Farroupilha

Cidade: Porto Alegre

Estado: RS

Longitude: -51.2182889

Latitude: -30.0388643

Fazer nova pesquisa

Projeto CEP – Tela de Carregamento

 Busca CEP

Endereço:

Bairro:

Cidade:


Estado:

Longitude:

Latitude:

Fazer nova pesquisa

Projeto CEP - URL da API

 **CEP 90040001 via Correios**

Endereço: Avenida João Pessoa

Bairro: Farroupilha

Cidade: Porto Alegre

Estado: RS

Longitude: -51.2182889

Latitude: -30.0388643

Fazer nova pesquisa

ElementsConsoleSourcesNetworkPerformanceMemory>>1⚙️⋮✕

1000 ms2000 ms3000 ms4000 ms5000 ms6000 ms

Name✕HeadersPreviewResponseInitiatorTiming

90040001

General

Request URL:https://brasilapi.com.br/api/cep/v2/90040001

Request Method:GET

Status Code:200

Remote Address:104.21.89.107:443

Referrer Policy:strict-origin-when-cross-origin

Response Headers

Access-Control-Allow-Credentials:true

Access-Control-Allow-Origin:*

Age:0

Alt-Svc:h3=":443"; ma=86400

Cache-Control:public, max-age=0, must-revalidate

Cf-Cache-Status:DYNAMIC

Cf-Ray:7de788c2e883011f-GRU

Content-Encoding:br

Content-Type:application/json; charset=utf-8


Date:Wed, 28 Jun 2023 17:08:52 GMT

Etag:W/"e6-MBDKm2U0V/7nmQfV7ryCG/UmENo"

Net:{"success_fraction":0,"report_to":"cf-

1 / 5 requests460 B / 46

Projeto CEP – Retorno da API

 **CEP 90040001 via Correios**

Endereço: Avenida João Pessoa

Estado: RS

Bairro: Farroupilha

Longitude: -51.2182889

Cidade: Porto Alegre

Latitude: -30.0388643

Fazer nova pesquisa

ElementsConsoleSourcesNetworkPerformanceMemory>>1⚙⋮✕

1000 ms2000 ms3000 ms4000 ms5000 ms6000 ms

Name✕HeadersPreviewResponseInitiatorTiming

90040001

```
{cep: "90040001", state: "RS", city: "Porto Alegre", neighborhood: "Farroupilha", cep: "90040001", city: "Porto Alegre", location: {type: "Point", coordinates: {longitude: "-51.2182889", latitude: "-30.0388643"}}, coordinates: {longitude: "-51.2182889", latitude: "-30.0388643"}, latitude: "-30.0388643", longitude: "-51.2182889", type: "Point", neighborhood: "Farroupilha", service: "correios", state: "RS", street: "Avenida João Pessoa"}
```

1 / 5 requests460 B / 4E

6

Sugestões de **Exercícios**

Exercícios para treinar

Estados, Efeitos, Rotas e mais

Com o projeto SWAPI

- Monte novamente o projeto utilizando os componentes já feitos, mas agora em uma versão React, utilizando o axios, SWR, formulários e outras ferramentas do React;
- Crie a página de busca e também uma página de detalhes para mostrar mais informações sobre a busca feita;
- Crie uma página 404 para mostrar um erro de página não encontrada em caso de acessar uma URL que não existe no React Router.

Bons estudos! 😊



OBRIGADO

@guscsales - gsales.io

FIAP

Copyright © 2022 | Professor Gustavo Campos Sales

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.





FIAP

