Bruno Pereira Gonçalves
Treinamento React

React Plano de Carreira

Carreira:

https://www.betrybe.com/guia-salarios-profissoes/desenvolvedor-react

Empresas Utilizam React:

https://github.com/react-brasil/empresas-que-usam-react-no-brasil

REACT

- React é uma biblioteca JavaScript para desenvolvimento de aplicações front-end;
- Estas aplicações são chamadas de SPA (Single Page Application);
- A arquitetura do React é baseada em componentes;
- Pode ser inserido em um aplicação ou podemos criar a aplicação apenas com React;
- É mantido pelo Facebook(Meta Open Source);

Site : https://pt-br.react.dev/

Instalação Ambiente

1-Instalar o Node.js:

- Node.js é um ambiente de execução JavaScript de código aberto e multiplataforma. É uma ferramenta popular para quase qualquer tipo de projeto!
- O Node.js executa o mecanismo V8 JavaScript, o núcleo do Google Chrome, fora do navegador. Isso permite que o Node.js tenha muito desempenho.
- Quando o Node.js executa uma operação de E/S, como ler da rede, acessar um banco de dados ou o sistema de arquivos, em vez de bloquear o thread e desperdiçar ciclos de CPU esperando, o Node.js retomará as operações quando a resposta retornar.
- O Node.js tem uma vantagem única porque milhões de desenvolvedores front-end que escrevem JavaScript para o navegador agora podem escrever o código do lado do servidor, além do código do lado do cliente, sem a necessidade de aprender uma linguagem completamente diferente.

Site : https://nodejs.org/en/download/package-manager

terminal: node -v

npm -v

Instalação Ambiente

2-Instalar Visual Studio Code(VsCode)

https://code.visualstudio.com/download

3-Instalar o Vite:

- No terminal do vscode ;
- npm install vite -g
- verificar versão: vite -v ;

Primeiro Projeto React

- Criar uma pasta no computador;
- Adicionar no Visual Studio;
- Acessar o terminal dentro da pasta do projeto desejado;
- Para criar as nossas aplicações em React vamos utilizar um executor de scripts do Node, que é o npx;
- Com o comando: npx create-react-app <nome> temos uma nova aplicação sendo gerada;
- Acessar a pasta do projeto: cd <nome da pasta>
- Podemos iniciar a aplicação com npm start;

Primeiro Projeto React+Vite

- Criar uma pasta no computador;
- Adicionar no Visual Studio;
- Acessar o terminal dentro da pasta do projeto desejado;
- Usar o comando no terminal: npm create vite@latest ;
- Defina o nome do projeto ;
- No menu escolha : React ;
- Escolha a linguagem : JavaScript :
- Execute as três ações: cd <nome_projeto> => npm install => npm run dev

Estrutura Base React

Há algumas pastas e arquivos chave para o desenvolvimento em React;
node_modules: Onde as dependências do projeto ficam;
public: Assets estáticos e HTML de inicialização;
src: Onde vamos programar as nossas apps;
src/index.js: Arquivo de inicialização do React;
src/App.js: Componente principal da aplicação;

Extensão Para React

1- Através de atalhos criar códigos React para facilitar a programação:

ES7+ React/Redux/React-Native snippets

2- Emmet é uma extensão nativa do VS Code que ajuda a escrever HTML mais rápido;

- Porém ela não vem configurada para o React!
- Temos que acessar File > Settings > Pesquise por: Emmet;
- Lá vamos incluir a linguagem: javascript - javascriptreact;

Componente React

Um componente React é uma parte reutilizável e independente de uma interface de usuário (UI) em aplicações desenvolvidas com a biblioteca React. Cada componente pode conter seu próprio estado e lógica, além de renderizar elementos HTML. Os componentes podem ser classificados em duas categorias principais:

1. **Componentes de Classe:** Usam a sintaxe de classes ES6 e têm acesso ao estado interno e a métodos de ciclo de vida.
2. **Componentes Funcionais:** Usam funções JavaScript e, com a introdução dos Hooks, também podem gerenciar estado e efeitos colaterais.

Os componentes podem ser combinados para formar interfaces complexas, e a modularidade dos componentes facilita a manutenção e o reuso do código. Além disso, eles permitem que os desenvolvedores construam interfaces dinâmicas que respondem a mudanças no estado da aplicação.

Componente React

O estado de um componente em React é um objeto que armazena dados que podem mudar ao longo do tempo e que afetam a renderização do componente. É uma maneira de o componente manter informações internas que podem ser usadas para controlar sua aparência e comportamento.

Aqui estão algumas características do estado:

1. **Mutable:** O estado pode ser atualizado, geralmente em resposta a ações do usuário (como cliques em botões) ou eventos do sistema.
2. **Local:** O estado é específico de um componente e não pode ser acessado diretamente por outros componentes, embora possa ser passado como props.
3. **Reatividade:** Quando o estado de um componente muda, o React automaticamente re-renderizar o componente para refletir essas alterações na UI.

Para gerenciar o estado em componentes funcionais, o React oferece o Hook useState. Em componentes de classe, o estado é definido no construtor e atualizado usando this.setState.

Exemplo de uso de estado em um componente funcional:

```
import React, { useState } from 'react';

function Contador() {
  const [contagem, setContagem] = useState(0);
  return (
    <div>
      <p>Contagem: {contagem}</p>
      <button onClick={() => setContagem(contagem + 1)}>Incrementar</button>
    </div>
  );
}
```

Criando Componentes React

- Na maioria dos projetos os componentes ficam em uma pasta chamada components, que devemos criar;
- Geralmente são nomeados com a camel case: firstComponent.js;
- No arquivo criamos uma função, que contém o código deste componente (a lógica e o template);
- E também precisamos exportar esta função, para reutilizá-lo;

Criando Componente Funcional

Os componentes funcionais são a forma mais comum e moderna de criar componentes em React, especialmente com o uso de Hooks.

```
import React from 'react';

function MeuComponente() {
  return (
    <div>
      <h1>Olá, Mundo!</h1>
    </div>
  );
}

export default MeuComponente;
```

Importando Componente

- A importação é a maneira que temos de reutilizar o componente;
- Utilizamos a sintaxe: import X from ‘./componentes/X’ onde X é o nome do componente;
- Para colocar o componente importado em outro componente, precisamos colocá-lo em forma de tag: <MeuComponente />
- E então finalizamos o ciclo de importação;

Importando e Usando o Componente

```
import React from 'react';
import MeuComponente from './components/MeuComponente';  {/* importando componente
*/}
function App() {
  return (
    <div>
      <MeuComponente />
    </div>
  );
}
export default App;
```

JSX do React

JSX (JavaScript XML) é uma extensão de sintaxe para JavaScript que permite escrever elementos de interface de usuário de forma semelhante ao HTML. JSX é usado no React para descrever como a UI deve se parecer.A sintaxe JSX em React combina JavaScript e HTML de maneira que permite criar interfaces de usuário de forma clara e concisa.

Característica do JSX

Sintaxe Semelhante ao HTML: JSX permite que você escreva marcação de maneira familiar, mas é Transpilado (conversão de um código escrito em uma linguagem para outra) ,para JavaScript. Exemplo:

```
const elemento = <h1>Olá, Mundo!</h1>;
```

Regras do JSX

1-Um Único Elemento Pai: Todos os elementos JSX devem ser encapsulados em um único elemento pai. Se você quiser retornar múltiplos elementos, **use um ou um fragmento**.

```
// Errado
return (
  <h1>Olá</h1>
  <h2>Mundo!</h2>
);

// Certo
return (
  <div>
    <h1>Olá</h1>
    <h2>Mundo!</h2>
  </div>
);
```

Regras do JSX

2-Atributos em CamelCase: Os atributos devem ser escritos em camelCase. Por exemplo, use className em vez de class, já que class é uma palavra reservada em JavaScript.

```
const elemento = <div className="minha-classe">Conteúdo</div>;
```

3-Expressões JavaScript: Pode incluir expressões JavaScript dentro de chaves {}. Isso permite que você insira variáveis e execute expressões diretamente no JSX.

```
const nome = "João";

const elemento = <h1>Olá, {nome}!</h1>;
```

Regras do JSX

4-Comentários: Os comentários em JSX devem ser escritos dentro de chaves, usando a sintaxe {/* comentário */}.

```
const elemento = (
  <div>
        {/* Este é um comentário */}
        <h1>Olá, Mundo!</h1>
  </div>
);
```

5-Nomes de Atributos: Nomes de atributos devem ser escritos em camelCase. Além de className, outros exemplos incluem htmlFor em vez de for.

```
const elemento = <label htmlFor="input">Nome:</label>;
```

Regras do JSX

6-Atributos Booleanos: Para atributos booleanos, você pode usar apenas o nome do atributo se ele estiver presente, ou undefined se não estiver.

```
const elemento = <input type="checkbox" checked={true} />; // checked
const elemento = <input type="checkbox" />; // não usa checked
```

7-Formatação de Texto: Texto em JSX pode ser tratado como uma string, mas se você quiser usar várias linhas, você deve usar um elemento pai ou fragmento.

```
const elemento = (
  <div>
      <p>Esta é uma linha.</p>
      <p>Esta é outra linha.</p>
  </div>
);
```

Regras do JSX

8-Keys em Listas: Quando renderizando listas de elementos, é importante fornecer uma key única para cada elemento, a fim de ajudar o React a identificar quais itens mudaram, foram adicionados ou removidos.

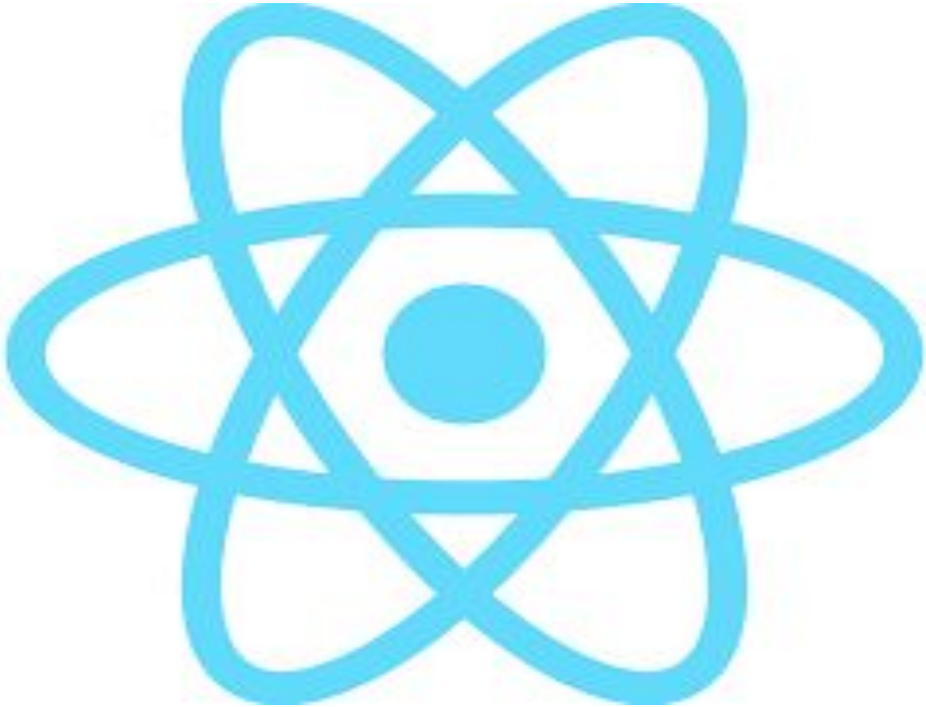
```
const itens = ['Maçã', 'Banana', 'Laranja'];
const lista = (
  <ul>
    {itens.map((item, index) => (
      <li key={index}>{item}</li>
    ))}
  </ul>
);
```

Regras do JSX

9-Fragmentos: Use fragmentos (<></> ou <React.Fragment>) para agrupar elementos sem adicionar nós extras ao DOM.

```
const elemento = (
  <>
      <h1>Olá</h1>
      <h2>Mundo!</h2>
  </>
);
```

IMAGENS REACT



IMAGENS REACT

Trabalhar com imagens em React é bastante flexível, permitindo que você utilize imagens locais, externas ou em diretórios públicos. A escolha do método depende da estrutura do seu projeto e das suas necessidades específicas.

Pasta Public: As imagens públicas ou as que ficaram permanente no nosso projeto podem ficar na pasta public.Podem ser chamadas pelas tags img diretamente pelo /nome.jpg;
Obs: Pois a pasta public fica linkada com o src das imagens;

```
function MeuComponente() {

  return <img src="/imagem.jpg" alt="Descrição da imagem" />;

}
```

IMAGENS REACT

Pasta assets: Padrão bem utilizada para as imagens dos projetos é colocar em uma pasta chamada assets, em src,ou seja,encontrara projetos com as duas abordagens. Em assets precisamos importar as imagens, e o src é dinâmico com o nome de importação;
import React from 'react';

```
import minhaImagem from './assets/nomeimagem.jpg';

function MeuComponente() {

  return <img src={minhaImagem} alt="Descrição da imagem" />;

}

export default MeuComponente;
```

IMAGENS REACT

Imagens de URLs Externas: Se você estiver usando imagens hospedadas em outros sites, pode referenciá-las diretamente pela URL.

```
function MeuComponente() {

  return <img src="https://exemplo.com/imagem.jpg" alt="Descrição da imagem"
/>;

}
```

Exercício

1- Faça um projeto React com três componentes: um componente acessando imagem da pasta public, um componente acessando imagem da pasta assets e um componente acessando imagem externa. Cada componente com o título respectivo ao acesso da imagem;

2- Projeto Final do Curso:

React Router

O que são Rotas em React?

Rotas permitem que você defina a navegação dentro de uma aplicação React, controlando qual componente ou página deve ser renderizado com base no URL atual. Para gerenciar rotas em uma aplicação React, geralmente usamos a biblioteca react-router-dom.

Como Funcionam as Rotas?

Router: (<Router>) é o componente que envolve todo o sistema de rotas. Ele fornece o contexto necessário para que as rotas funcionem.

Route: (<Routes>) é o componente que contém as definições das rotas específicas (<Route path="/" element={<Home />} />). Ele verifica a URL atual e renderiza o componente correspondente.

Link: O componente Link é usado para criar links de navegação dentro da aplicação.

Configurar Rotas

1-Instalação Primeiro, instale a biblioteca react-router-dom:

npm install react-router-dom

Certifique-se de que todas as versões das dependências essenciais estão atualizadas e corretas:

- react: 18.3.1
- react-dom: 18.3.1
- react-router-dom: 6.26.1

npm ls

Caso precise configurar:

1-Remova:

npm uninstall react-router-dom

2-Limpar Cache do NPM

npm cache clean --force

3-Instalar a Versão Correta de react-router-dom

npm install react-router-dom@6.26.1

4-Verificar a versão Instalada

npm ls react-router-dom

```
import React from 'react'
import {Routes,Route} from 'react-router-dom'
import Home from './paginas/home'
import Sobrenos from './paginas/sobrenos'
function Rotas(){
   return(
      <Routes>
            <Route path='/' element={<Home/>}></Route>
            <Route path='/sobrenos' element={<Sobrenos/>}></Route>
      </Routes>
   );
}export default Rotas
```

Usando as Rotas

O que é Link: O componente Link do react-router-dom é usado para criar links de navegação **interna em uma aplicação React**. Ele **permite a navegação entre diferentes rotas sem recarregar a página inteira,** o que proporciona uma experiência de usuário mais suave e rápida.

Importação

Para usar Link, primeiro importe-o da biblioteca react-router-dom:

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

Como Usar o Link

O Link é usado de maneira semelhante à tag HTML <a>, mas é específico para navegação interna em React. Exemplo básico:

Benefícios do Uso de Link

1. **Navegação sem Recarregar:** Proporciona uma navegação suave sem recarregar a página inteira.
2. **Manutenção do Estado:** Permite manter o estado e o contexto da aplicação durante a navegação.
3. **Sintaxe Familiar:** Semelhante à tag <a>, mas otimizado para SPA (Single Page Applications).

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch, Link } from 'react-router-dom';
function App() {
    return (
            <nav>
                <ul>
                    <li>
                        <Link to="/">Home</Link>
                    </li>
            </nav>
    );
}
```

Comparação entre <a> e Link

Característica	**Tag <a>**	**Componente Link**
Navegação Completa	Sim	Não (Navegação interna)
Recarregamento de Página	Sim	Não
SEO	Reconhecida por motores de busca	Navegação interna, precisa de configuração adicional para SEO
Usabilidade	Links externos	Navegação interna em aplicações React

Melhores Práticas para Criar Rotas em React

Modularidade: Modularizar as rotas em um arquivo dedicado, como **Routes.js**, é uma prática altamente recomendada. Isso mantém o código do App.js limpo e focado em sua responsabilidade principal, enquanto as rotas são gerenciadas separadamente.

Desempenho e Otimização: Utilize o carregamento sob demanda (lazy loading) para carregar componentes de forma assíncrona, melhorando o desempenho da aplicação.

Organização de Pastas: Manter uma estrutura de pastas clara e intuitiva é essencial para a manutenção e escalabilidade do projeto. Seguindo a estrutura modular, pode-se organizar componentes e páginas de forma lógica.

ROTAS: Carregamento Sem Sob Demanda (Eager Loading)

Carregar componentes sem lazy loading, também conhecido como eager loading, significa carregar todos os componentes de uma vez, independentemente de serem necessários imediatamente ou não.

Vantagens:

1. **Implementação Simples:** Menos código e complexidade, mais fácil de implementar.
2. **Navegação Imediata:** Componentes já estão carregados, então a navegação pode parecer instantânea após o carregamento inicial.

Desvantagens:

1. **Tempo de Carregamento Inicial:** Pode ser significativamente maior, pois todo o código é carregado na inicialização da aplicação.
2. **Maior Uso de Recursos:** Usa mais memória e poder de processamento, já que todos os componentes são carregados de uma vez.

```jsx
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Home from './pages/Home';
import About from './pages/About';
import NotFound from './pages/NotFound';

function App() {
    return (
        <Router>
            <Switch>
                <Route exact path="/" component={Home} />
                <Route path="/about" component={About} />
                <Route component={NotFound} />
            </Switch>
        </Router>
    );
}

export default App;
```

<Router>

- Significado: <Router> é o componente que fornece a funcionalidade de roteamento para a aplicação. Em uma aplicação típica,usará BrowserRouter (apelidado de Router).
- Função: Envolve a aplicação inteira para permitir o uso do histórico de navegação do navegador e sincronizar a interface do usuário com o URL atual.

```
import { BrowserRouter as Router } from 'react-router-dom';
```

<Route path="/" element={<Home/>} />

- Significado: <Route> é o componente que define uma rota específica, especificando o caminho (path) e o componente a ser renderizado (component).
- Função:

  path="/": Define o caminho para o qual a rota corresponde. Neste caso, é a raiz (/).O Sistema inicia pela raiz .

  element={<Home/>}: Especifica o componente Home a ser renderizado quando o caminho corresponde.

  import { Route } from 'react-router-dom';

  import Home from './pages/Home';

Arquivos que devem ser verificados

- **App.jsx :** Verificar se a arquivo rota foi inicializado de forma direto ou dentro componente;
- **main.jsx:** Verificar se o <App/> estar envolvido pelo Router;
- **vite.config:** adicionar: **server: { historyApiFallback: true, }, ;**

ROTAS:Carregamento Sob Demanda (Lazy Loading)

Lazy loading é uma técnica que permite carregar componentes somente quando eles são necessários. Isso é especialmente útil em aplicações grandes ou com muitas rotas, pois evita carregar todo o código de uma vez, melhorando o tempo de carregamento inicial.

Vantagens:

1. **Desempenho Melhorado:** Carrega apenas o que é necessário inicialmente, reduzindo o tempo de carregamento da página.
2. **Menos Uso de Recursos:** Diminui o uso de memória e processamento, pois os componentes são carregados sob demanda.
3. **Experiência de Usuário:** Proporciona uma navegação mais rápida e suave, especialmente em dispositivos com recursos limitados.

Desvantagens:

1. **Complexidade Adicional:** Adicionar lazy loading requer mais planejamento e pode complicar um pouco o gerenciamento de dependências e estados.

```jsx
// routes.jsx
import React, { Suspense, lazy } from 'react';
import { Routes, Route } from 'react-router-dom';

// Use React.lazy para carregar os componentes
const Home = lazy(() => import('./components/Home'));
const About = lazy(() => import('./components/About'));
const User = lazy(() => import('./components/User'));

const AppRoutes = () => {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/user" element={<User />} />
      </Routes>
    </Suspense>
  );
};

export default AppRoutes;
```

```
import React, { Suspense, lazy } from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
```

import React: Importa a biblioteca React, que é essencial para criar componentes React.

{ Suspense, lazy }: Importa os hooks Suspense e lazy do React, usados para carregamento sob demanda (lazy loading) de componentes.

{ BrowserRouter as Router, Route, Switch }: Importa os componentes de roteamento do react-router-dom.

Carregamento Sob Demanda (Lazy Loading) dos Componentes

```
const Home = lazy(() => import('./pages/Home'));
const About = lazy(() => import('./pages/About'));
const NotFound = lazy(() => import('./pages/NotFound'));
```

lazy(() => import('./pages/Home')): Define que o componente Home deve ser carregado sob demanda.

import('./pages/Home'): Importa dinamicamente o módulo Home quando ele é necessário.

const Home: Declara uma constante que armazena o componente carregado sob demanda.

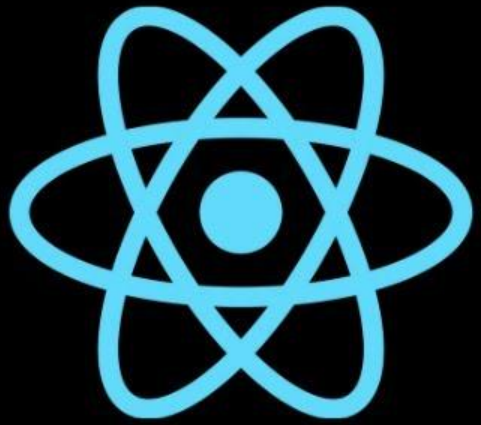
lazy: é uma função fornecida pelo React que permite carregar componentes sob demanda.Ele aceita uma função que retorna uma promessa que resolve o módulo do componente.

Componente Principal

<Suspense fallback={<div>Loading...</div>}>: Envolve os componentes carregados sob demanda e exibe um fallback (neste caso, <div>Loading...</div>) enquanto os componentes estão sendo carregados.

- **fallback={ Loading... }:** Define o conteúdo a ser exibido enquanto os componentes estão sendo carregados.

All React Hooks Explained



1. useState
2. useEffect
3. useContext
4. useReducer
5. useCallback
6. useMemo
7. useRef
8. useImperativeHandle
9. useLayoutEffect

React Hook

Hooks são funções especiais em React que permitem que você utilize o **estado e outras funcionalidades** do React em **componentes funcionais.**

Introduzidos na versão 16.8,os Hooks simplificam o gerenciamento de estado e a execução de efeitos colaterais em componentes funcionais, promovendo uma abordagem mais limpa e reutilizável no desenvolvimento com React. Eles também facilitam a divisão de lógica de estado em funções menores, melhorando a legibilidade e a manutenção do código.

- Todos os hooks começam com **use**, por exemplo: **useState**;
- Podemos criar os nossos hooks, isso é chamado de **custom hook**;
- Os hooks precisam ser importados;

Principais Hook- useState

1. O hook de useState é um dos mais utilizados;
2. Utilizamos para **gerenciar o estado de algum dado**, variáveis não funcionam corretamente, o componente não re-renderiza;
3. Permite adicionar estado local a **um componente funcional;**
4. Retorna **um array** com o valor atual do estado e uma função para atualizá-lo;
5. Para guardar o dado definimos o **nome da variável** e para alterar vamos utilizar setNome, onde nome é o nome do nosso dado;

```
import React, { useState } from 'react';

function Contador() {
  const [contagem, setContagem] = useState(0);
  return (
    <div>
      <p>Contagem: {contagem}</p>
      <button onClick={() => setContagem(contagem + 1)}>Incrementar</button>
    </div>
  );
}
```

Principais Hook- useEffect

1. Permite realizar efeitos colaterais em componentes funcionais, como chamadas de API, manipulação de DOM ou subscrições.
2. Pode ser configurado para rodar em diferentes momentos do ciclo de vida do componente.

```
import React, { useState, useEffect } from 'react';
function Exemplo() {
  const [dados, setDados] = useState([]);
  useEffect(() => {
    fetch('https://api.exemplo.com/dados')
      .then(response => response.json())
      .then(data => setDados(data));
  }, []); // O array vazio significa que o efeito roda apenas uma vez
  return (
        <ul>
              {dados.map(item => (
              <li key={item.id}>{item.nome}</li>
              ))}
        </ul>
  );
}
```

Principais Hook- useContext

Permite acessar o contexto de uma maneira mais simples, sem a necessidade de consumir o contexto diretamente em cada nível da árvore de componentes.

```
import React, { useContext } from 'react';

const MeuContexto = React.createContext();
function Componente() {
  const valor = useContext(MeuContexto);
  return <p>O valor é: {valor}</p>;
}
```

Principais Hook- useReducer

Uma alternativa ao useState para gerenciar estados mais complexos. Baseia-se no padrão Redux, permitindo que você escreva lógica de atualização de estado em um único lugar.

```
import React, { useReducer } from 'react';
const reducer = (state, action) => {
  switch (action.type) {
    case 'incrementar':
      return { count: state.count + 1 };
    case 'decrementar':
      return { count: state.count - 1 };
    default:
      return state;
  }
};
```

```
function Contador() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <div>
        <p>Contagem: {state.count}</p>

        <button onClick={() => dispatch({ type: 'incrementar' })}>+</button>

        <button onClick={() => dispatch({ type: 'decrementar' })}>-</button>

    </div>
  );
}
```