DESENVOLVIMENTO FULL STACK

React - State

Caio Eduardo do Prado Ireno

caio.ireno@faculdadeimpacta.com.br

Estado

Os componentes frequentemente precisam alterar o que é exibido na tela em resposta a interações. Por exemplo, digitar em um formulário deve atualizar o campo de entrada, clicar em "próximo" em um carrossel de imagens deve mudar a imagem exibida, e clicar em "comprar" deve adicionar um produto ao carrinho.

Os componentes precisam "lembrar" de informações, como o valor atual do input, a imagem atual e o conteúdo do carrinho de compras. **No React, esse tipo de memória específica do componente é denominado estado**.

O estado de uma aplicação refere-se às informações que podem mudar ao longo do tempo e que influenciam a renderização da interface do usuário.

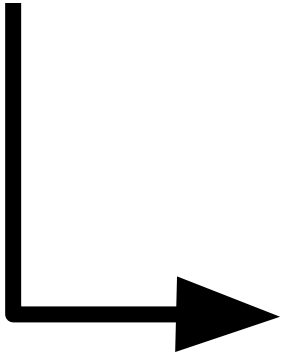
```
export const Teste = () => {
  const ativo = true;

  return (
    <button disabled={!ativo}>{ativo ? "Botão Ativo" : "Botão Inativo"}</button>
  );
};
```

https://stackblitz.com/edit/vitejs-vite-img5bqcw?file=src%2FApp.tsx

Botão Muda o Estado?

Constante Imutável

A variável ativo é declarada como uma constante (const). Isso significa que seu valor é **fixo** e não pode ser alterado após a sua definição.



Portanto, mesmo que você queira mudar o estado do botão, não há uma maneira de atualizar ativo dentro do componente.

Sem Gerenciamento de Estado

Como **ativo** não está sendo gerenciado por um sistema de estado, a interface não reage a mudanças.

O React não sabe que precisa **re-renderizar** o componente quando o estado muda porque não há um mecanismo para isso

POR QUE USAR HOOKS?

Hooks

Os hooks são funções que permitem que você **"conecte"** o estado e outras funcionalidades do React a componentes funcionais. O hook mais comum para gerenciamento de estado é o **useState**.

Versão do componente usando **useState**:

```
export const Teste = () => {
  const [ativo, setAtivo] = useState(true);

  const toggleAtivo = () => {
    setAtivo((prevAtivo) => !prevAtivo);
  };

  return (
    <div>
      <button disabled={!ativo} onClick={toggleAtivo}>
        {ativo ? "Botão Ativo" : "Botão Inativo"}
      </button>
    </div>
  );
};
```

https://stackblitz.com/edit/vitejs-vite-ny6bwwua?file=src%2FApp.tsx

Vantagens de Usar Hooks

Reatividade: O uso de hooks permite que o componente reaja a mudanças de estado, atualizando a interface automaticamente.

Simplicidade: Hooks simplificam a lógica do estado e permitem que você use estado e outras funcionalidades do React em componentes funcionais.

Composição: Você pode facilmente compartilhar lógica entre componentes usando hooks personalizados.

React.useState

O useState é um hook que retorna um array com dois elementos.:

1. O primeiro elemento representa o valor atual do estado, que pode ser de qualquer tipo, como strings, arrays, números, booleanos, null, undefined e objetos.
2. O segundo elemento é uma função que permite atualizar o estado do primeiro valor.

Quando essa função de atualização é chamada, todos os componentes que **dependem desse estado** são **re-renderizados**, assim como seus componentes filhos. Esse mecanismo assegura a reatividade dos componentes funcionais no React.

Múltiplos Estados

Não existem limites para o uso do useState, podemos definir diversos no mesmo componente.

```
export const Teste = () => {
  const [contar, setContar] = React.useState(0);
  const [ativo, setAtivo] = React.useState(false);
  const [dados, setDados] = React.useState({ nome: "", idade: "" });

  return <div></div>;
```

Reatividade

Não modifique o estado diretamente. Utilize sempre a função de atualização do estado, pois ela que garante a reatividade dos componentes.

https://stackblitz.com/edit/vitejs-vite-ngafcuyb?file=src%2FApp.tsx

```jsx
export const Teste = () => {
  const [carro, setCarro] = React.useState(["Carro 1", "Carro 2"]);

  function handleClick() {
    // Errado. Modifique o estado apenas com a função de atualização (setCarro)
    carro.push("Novo Carro");
  }

  function handleClickReativo() {
    // Correto. Eu desestruturo a array atual, criando uma nova e adiciono um novo elemento
    setCarro([...carro, "Novo Carro"]);
  }

  return (
    <>
      {carro.map((item, i) => (
        <li key={i}>{item + " " + i}</li>
      ))}
      <button onClick={handleClick}>Adicionar Carro</button>
      <button onClick={handleClickReativo}>Adicionar Reativo</button>
    </>
  );
```

Callback

É possível passar uma função de callback para atualizar o estado. Essa função recebe um parâmetro que representa o valor anterior do estado e deve retornar o novo valor que será atribuído ao estado.

https://stackblitz.com/edit/vitejs-vite-twrjamna?file=src%2FApp.tsx

```jsx
import React from "react";

const Callback = () => {
  const [ativo, setAtivo] = React.useState(true);

  function handleClick() {
    // usando um callback
    setAtivo((anterior) => !anterior);
  }

  return (
    <button onClick={handleClick} style={{ height: "50px",
width: "100px" }}>
      {ativo ? "Está Ativo" : "Está Inativo"}
    </button>
  );
};

export default Callback;
```

Exercício: Consumindo API com ViaCEP

Objetivo: Criar uma interface que consome a API do ViaCEP e exibe dados de endereços.

Instruções: API do ViaCEP: Utilize o seguinte endpoint para obter os dados de um endereço em formato JSON: **https://viacep.com.br/ws/{cep}/json/**Interface:

Crie uma interface com três botões, cada um associado a um CEP diferente.

Fetch de Dados: Ao clicar em um dos botões, faça uma requisição fetch para a API do ViaCEP correspondente ao CEP do botão e obtenha os dados do endereço.
Exiba os dados do endereço na tela.

Exibição: Mostre apenas um endereço por vez.
Enquanto a requisição está sendo realizada, exiba a mensagem "Carregando...".

Dicas:
Utilize o hook **useState** para gerenciar o estado dos dados do endereço e o estado de carregamento.
Utilize **interface** para tipar a resposta de requisição.

Parte 2: Ao invez de botão, faça um formulário para inserir o CEP

USEEFFECT

useEffect

Todo componente possui um ciclo de vida, que abrange os principais eventos de renderização, atualização e destruição. Com o **React.useEffect()**, é possível definir um callback que será executado em determinados momentos desse ciclo de vida do componente.

```
import React from "react";

export default function UseEffectHooks() {
  const [contar, setContar] = React.useState(0);

  React.useEffect(() => {
    console.log("Ocorre ao renderizar e ao atualizar");
  });

  return <button onClick={() => setContar(contar +
1)}>{contar}</button>;
}
```

useEffect

Clique no Botão: Quando o botão é clicado, setContar é chamado, o que atualiza o estado contar.

Essa atualização provoca uma nova renderização do componente.

Renderização Após Atualização: Após a atualização do estado, o useEffect é novamente executado, e a mesma mensagem é impressa no console.

Array de Dependências

No **useEffect** podemos definir dois argumentos:

- Função de callback que será executada,
- Array com uma lista de dependências.

A lista de dependências serve para informarmos quando o efeito deve ocorrer.

https://stackblitz.com/edit/vitejs-vite-gf1g065p?file=src%2FApp.tsx

Componente Montou

O **useEffect** é particularmente útil quando precisamos estabelecer um efeito que deve ser executado apenas uma vez, como ao **buscar dados de um servidor**, por exemplo.

https://stackblitz.com/edit/vitejs-vite-99ekcqxf?file=src%2FApp.tsx

Exercício: Consumindo API com ViaCEP

Refaça o exercício anterior, porem utilizando o Hooks UseEffect.

DICAS: Neste caso, o fluxo funciona da seguinte forma:

1. O formulário captura o valor do CEP e chama o manipulador de envio (HandleSubmit).
2. O estado do CEP (cep) é atualizado pelo formulário.
3. O useEffect é disparado sempre que o estado cep for alterado, iniciando a busca dos dados.

Obrigado!

EXERCICIO 1:
https://stackblitz.com/edit/vitejs-vite-fk7q2scf?file=src%2FApp.tsx

EXERCICIO 2:
https://stackblitz.com/edit/vitejs-vite-e4tvhx1m?file=src%2FApp.tsx