React – Hooks - useState

O que é

Os Hooks no React representam uma revolução na forma como os desenvolvedores gerenciam o estado e o ciclo de vida em componentes funcionais. Com a adição dos Hooks, é possível extrair lógica de componentes, reutilizar código de forma mais eficiente e simplificar o gerenciamento de estados.

Hooks são funções especiais fornecidas pelo React que permitem adicionar funcionalidades de componente em componentes funcionais. Eles permitem que você use estado e outras características do React em componentes que são funcionalmente equivalentes aos componentes de classe.

Antes dos Hooks, os componentes funcionais não podiam conter estado local, o que os limitava em termos de funcionalidades. Agora, com Hooks, os componentes funcionais podem ter estados locais, efeitos colaterais e lógica de ciclo de vida, tornando-os tão poderosos quanto os componentes de classe.

Principais Hooks

- useState
- useEffect
- useContext

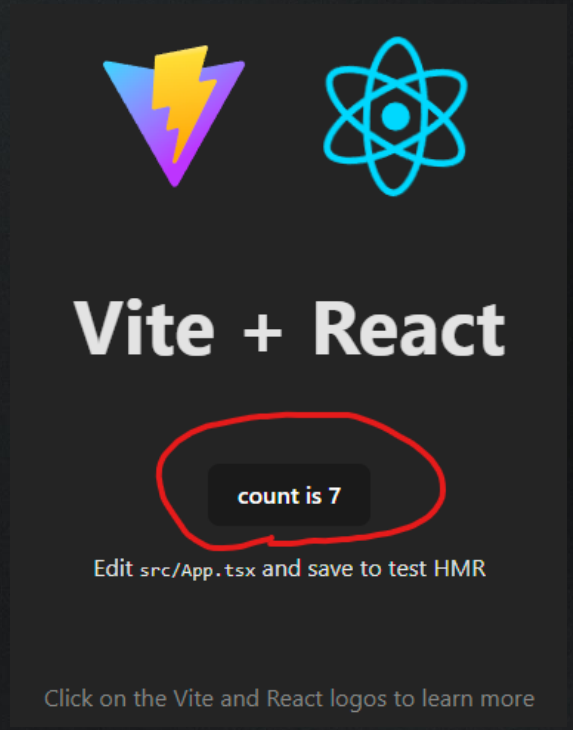
useState

O useState é utilizado para adicionar estado a componentes funcionais. Ele retorna um par de valores: o estado atual e uma função para atualizá-lo. Em TypeScript, é possível também fornecer o tipo esperado para o estado.

O useState é uma função que retorna dois valores: o estado atual e uma função para atualizar esse estado. A função useState aceita um argumento opcional, que é o valor inicial do estado.

Vamos criar uma aplicação com o comando do vite, instalar a dependências e abrir o App.tsx que vem por padrão do vite.

A tela padrão do vite vem um botão, onde ao clicarmos ele adicionar +1 ao contador. Esse contado é controlado pelo useState



useState

Vamos criar um componente novo para entendermos o funcionado do useState

1. Criar uma pasta components, e vamos criar um arquivo Exemple.tsx
2. Vamos criar o componente que vai retornar um HTML básico com um texto e um botão de interação
3. Vamos deletar novamente as coisas dentro do App.tsx e colocar nosso componente lá

default > src > components > Exemple.tsx > ...

```tsx
const ExempleComponent = () => {
  return (
    <>
      <h6>
        Exemplo de estado
      </h6>
      <p>
        Nome:
        <span>
          Nome aqui
        </span>
      </p>
      <button>
        Clique Aqui
      </button>
    </>
  )
}

export default ExempleComponent
```

default > src > App.tsx > ...

```tsx
import ExempleComponent from "./components/Exemple"

function App() {
  return (
    <>
      <ExempleComponent />
    </>
  )
}

export default App
```

useState

4. Vamos importar o Hook

```
1  import { useState } from "react"
2
3  const ExempleComponent = () => {
4    return (
```

5. Vamos declarar o uso dele, criando em uma variavel de nome

```
1  import { useState } from "react"
2
3  const ExempleComponent = () => {
4    const [name, setName] = useState<string>('')
5
```

6. Vamos colocar a variavel de name no nosso HTML e no click do botão chamar o setName para alterar o nome

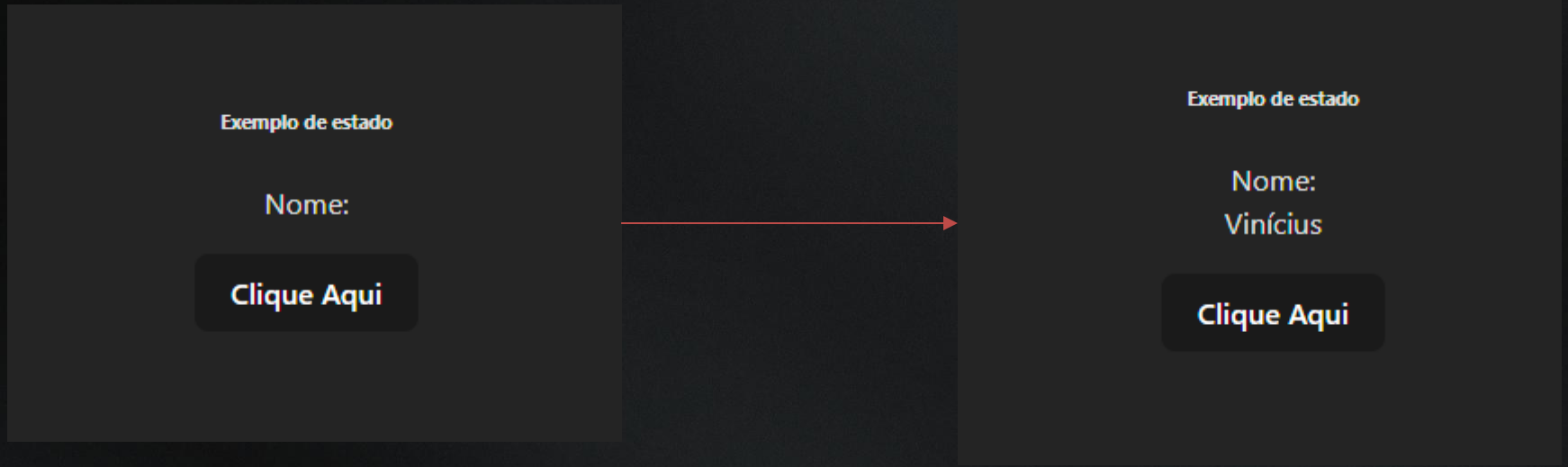
useState

6. Vamos colocar a variavel de name no nosso HTML e no click do botão chamar o setName para alterar o nome. Nosso componente deve ficar mais ou menos assim:

```tsx
import { useState } from "react"

const ExempleComponent = () => {
  const [name, setName] = useState<string>('')

  return (
    <>
      <h6>
        Exemplo de estado
      </h6>
      <p>
        Nome:
        <span>
          {name}
        </span>
      </p>
      <button onClick={() => setName('Vinícius')}>
        Clique Aqui
      </button>
    </>
  )
}

export default ExempleComponent
```

useState

Resultado



useState

Agora vamos criar uma manipulação de estado mais interessante, vamos manipular um formulário.
1. Vamos criar um novo arquivo Exemple2.tsx dentro da pasta de components, com a seguinte estrutura inicial. Também já podemos usar ele no nosso App.tsx

default > src > components > Exemple2.tsx > ...

```tsx
import { useState } from "react"

const Exemple2Component = () => {


  return (
    <>

    </>
  )
}

export default Exemple2Component

```

default > src > App.tsx > ...

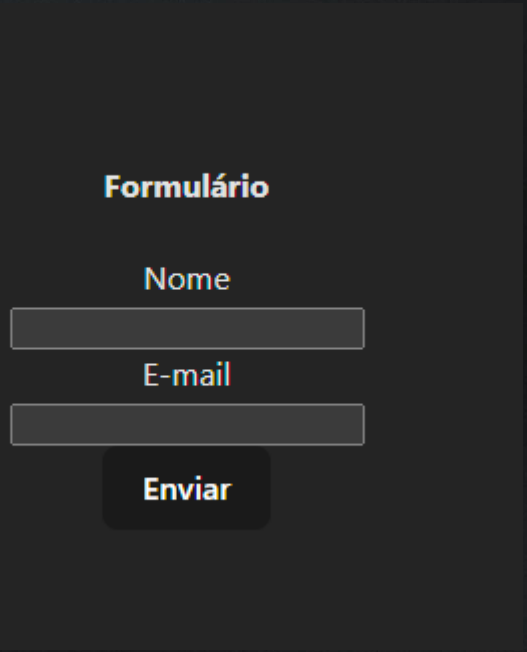
```tsx
import './App.css'
import Exemple2Component from "./components/Exemple2"

function App() {
  return (
    <>
      <Exemple2Component />
    </>
  )
}

export default App

```

useState

2. Vamos criar um formulário com nome e email

default > src > components > Exemple2.tsx > ...

```tsx
import { useState } from "react"

const Exemple2Component = () => {


  return (
    <form>
      <h4>
        Formulário
      </h4>
      <label htmlFor="name">
        Nome<br />
        <input id="name" name="name" type="text" />
      </label>
      <br />
      <label htmlFor="email">
        E-mail<br />
        <input id="email" name="email" type="email" />
      </label>
      <br />
      <button type="submit">
        Enviar
      </button>
    </form>
  )
}

export default Exemple2Component

```



useState

3. Agora vamos criar o nosso estado e usar uma interface TypeScript para tipar o nosso estado

```
interface MyData {
  name: string
  email: string
}

const Exemple2Component = () => {
  const [data, setData] = useState<MyData>({
    name: '',
    email: ''
  })
```

4. Em seguida vamos setar o atributo value dos nossos inputs

```
<label htmlFor="name">
  Nome<br />
  <input id="name" name="name" type="text" value={data.name} />
</label>
<br />
<label htmlFor="email">
  E-mail<br />
  <input id="email" name="email" type="email" value={data.email} />
</label>
```

useState

FIAP

5. Vamos criar uma função chamada da de inputChange que fara alterações no nosso estado

```
const inputChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const { name, value } = e.target
  setData({
    ...data,
    [name]: value
  })
}
```

6. Agora vamos aplicar a função nos inputs

```
<label htmlFor="name">
  Nome<br />
  <input id="name" name="name" type="text" value={data.name} onChange={inputChange} />
</label>
<br />
<label htmlFor="email">
  E-mail<br />
  <input id="email" name="email" type="email" value={data.email} onChange={inputChange} />
</label>
```

useState

7. Só isso já é suficiente para alterarmos os valores do estado, mas vamos um pouco além. Vamos criar um evento de envio do formulário e exibir em tela, como já fizemos nas aulas de HTML. Vamos criar um outro estado para controlarmos se o submit já foi realizado e uma função para alterar esse estado

```
3   const [isSubmit, setIsSubmit] = useState<boolean>(false)
4
5   const handleSubmit = (e: React.FormEvent<HTMLFormElement>) => {
6     e.preventDefault()
7     if (data.name && data.email) {
8       setIsSubmit(true)
9     }
0   }
```

8. Agora vamos colocar no onsubmit do nosso formulário

```
30    return (
31  >   <form onSubmit={handleSubmit}>···
48        </form>
49    )
```

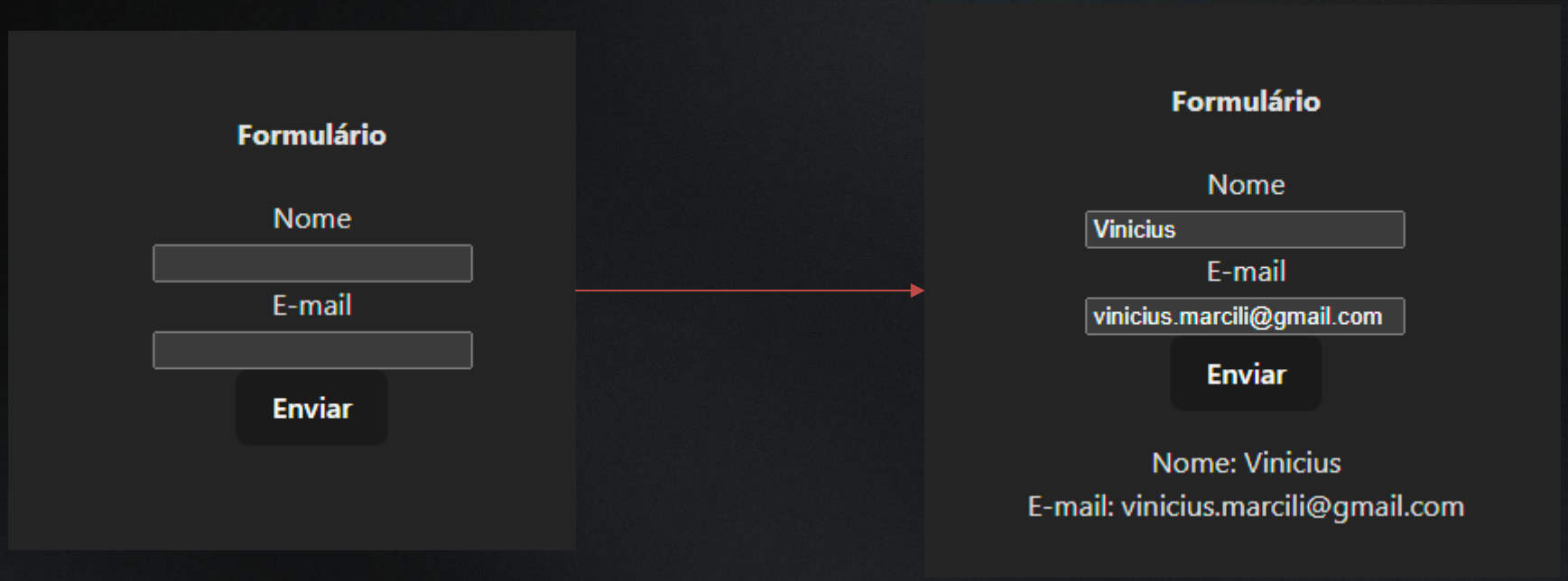
useState

9. Feito isso, vamos criar no nosso HTML um local para mostrarmos o resultado validando a nossa variavel isSubmit. Vamos fazer como se fosse um IF, caso a primeira condição seja verdadeira o JSX vai executar a segunda condição que é o nosso HTML, como na imagem abaixo:

```
30  return (
31    <form onSubmit={handleSubmit}>
32      <h4>
33        Formulário
34      </h4>
35      <label htmlFor="name">
36        Nome<br />
37        <input id="name" name="name" type="text" value={data.name} onChange={inputChange} />
38      </label>
39      <br />
40      <label htmlFor="email">
41        E-mail<br />
42        <input id="email" name="email" type="email" value={data.email} onChange={inputChange} />
43      </label>
44      <br />
45      <button type="submit">
46        Enviar
47      </button>
48      {isSubmit && (
49        <p>
50          Nome: {data.name}<br />
51          E-mail: {data.email}
52        </p>
53      )}
54    </form>
```

useState

Resultado:



useState – Combinando Props com useState

Como vimos anteriormente, podemos passar uma função como uma Prop de um componente no React. Também podemos combinar o uso dos dois para alterar o valor do useState do componente pai com uma informação que está no componente filho.
Vamos pegar de exemplo a aplicação inicial do Vite com React, que tem um botão de contador para exemplificar

useState – Combinando Props com useState

1. Vamos criar um componente chamado Counter.tsx e vamos copiar o HTML da div ‘card’ do nosso App.tsx
2. Vamos passar via props o valor de count e a função setCount

default > src > components > Counter.tsx > ...

```tsx
interface Props {
  count: number
  setCount: (count: number) => void
}

const Counter = ({ count, setCount }: Props) => {
  return (
    <div className="card">
      <button onClick={() => setCount(count + 1)}>
        count is {count}
      </button>
      <p>
        Edit <code>src/App.tsx</code> and save to test HMR
      </p>
    </div>
  )
}

export default Counter
```

useState – Combinando Props com useState

FIAP

3. Agora vamos ir até o App.tsx e colocar o nosso componente. Vamos ver que tudo está funcionando como antes, porém agora o filho está atualizando o valor do count que está no pai

```
function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <Counter count={count} setCount={setCount} />
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}
```

useState

Exercícios

1. Desenvolva um componente de contador simples em React utilizando o Hook useState. O componente deve exibir um número inicializado em 0 e dois botões, um para incrementar o número e outro para decrementá-lo. Mostre o valor atual do contador na interface.

2. Crie um componente de lista de tarefas (to-do list) em React que permite adicionar e remover tarefas. Utilize o Hook useState para gerenciar o estado das tarefas. O componente deve incluir um formulário para adicionar novas tarefas, uma lista de tarefas exibindo seus nomes e botões para remover cada tarefa individualmente.

Dúvidas, críticas ou sugestões?


FIAP