Programação Mobile
React Native - Aula 06

Professor: João Felipe Bragança

Já aprendemos até aqui:

- As diferenças entre Aplicativos Nativos e Híbridos.
- O que é React Native e quando podemos utilizá-lo.
- Como configurar o ambiente de desenvolvimento (VSCode, Node, Expo, Expo Go, emulador do Android Studio),
- Como criar um projeto novo com Expo.
- Identificar os arquivos que vem no projeto (package.json, node_modules, etc).
- O que são Componentes.
- Os principais Componentes do React Native (View, Text, StyleSheet, etc).
- Como criar nossos próprios Componentes.
- Como passar propriedades do Componente Pai para o Componente Filho.
- Como trabalhar com FlexBox para criar layouts.
- Revisão básica de JavaScript (variáveis (var, let, const), funções (arrow functions), objetos, arrays, for, foreach e map).
- e muito mais…

Objetivos da Aula

- Entender o que são Hooks.
- Entender quando uma variável não é suficiente.
- State - A memória de um Componente
- State vive dentro de um Componente
- Como compartilhar States Entre Componentes

O que são Hooks?

- São funções especiais que nos permitem acessar funcionalidades do React, como Estados e Efeitos Colaterais, em componentes funcionais. Tais métodos só estavam disponíveis em componentes de Classe.
- Foram introduzidos no React como uma forma de simplificar o uso de estados e métodos do ciclo de vida.
- Com Hooks, é possível utilizar apenas componentes funcionais, que são funções JavaScript simples e eficientes.
- Tornou possível a escrita de componentes complexos de forma muito mais “limpa” do que utilizando componentes de Classe.
- O uso de Hooks nos componentes funcionais tornou o uso de classes obsoleto.
- Não existe previsão para descontinuar componentes de Classe, para manter a compatibilidade com projetos legados.

Componente de Classe

```jsx
// demonstrating a Class component
class Counter extends React.Component {
     constructor(props) {
       super(props);
       this.state = { count: 0 };
     }

     componentDidMount() {
       this.setState({ count: this.state.count + 1 });
     }

    handleIncrement = () => {
       this.setState({ count: this.state.count + 1 });
     };

    handleDecrement = () => {
       this.setState({ count: this.state.count - 1 });
    };

     render() {
       return (
         <div className="counter">
           <h1 className="count">{this.state.count}</h1>

           <button type="button" onClick={this.handleIncrement}>
             Increment
           </button>
           <button type="button" onClick={this.handleDecrement}>
             Decrement
           </button>
         </div>
       );
     }
}

export default Counter;
```

Hooks mais Utilizados

- useState: Permite adicionar estado a um componente funcional. Ele retorna um par: o valor atual do estado e uma função para atualizá-lo.

- useEffect: Permite executar efeitos colaterais em um componente funcional. É uma alternativa aos métodos de ciclo de vida de componentes de classe, como componentDidMount ou componentDidUpdate.

- useContext: Permite acessar o valor de um contexto no React em um componente funcional.

Alguns Hooks mais avançados

- useReducer: Uma alternativa ao useState para gerenciar estados mais complexos com ações e uma função redutora.

- useMemo: Permite memorizar valores calculados para evitar recalcular uma função de alta carga sempre que o componente renderizar.

- useCallback: Permite memorizar uma função para evitar recriá-la em cada renderização.

- useRef: Permite criar uma referência mutável que persiste durante todo o ciclo de vida do componente.

Quando uma variável não é suficiente

- Variáveis locais não mantém o valor entre renderizações.
- Atualizar uma variável local não executa uma re-renderização.
- O exemplo do contador ilustra bem, apesar do valor da variável mudar ao pressionar o botão, a tela não muda, nenhuma re-renderização é acionada.
- Caso alguma renderização fosse executada, o valor da variável seria perdido.
- Para atualizar o componente com os dados novos, são preciso duas coisas:
    - Manter os dados entre as renderizações
    - Acionar a re-renderização.

        - *Para isso, temos o Hook **useState***

State - A memória de um componente

- Frequentemente, os componentes precisam alterar o que está na tela como resultado de uma interação. Digitar no formulário deverá atualizar o campo de entrada, clicar em “próximo” em um carrossel de imagens deverá alterar a imagem exibida, clicar em “comprar” deverá colocar um produto no carrinho de compras.

- Os componentes precisam “lembrar” coisas: o valor de entrada atual, a imagem atual, o carrinho de compras. No React, esse tipo de memória específica do componente é chamada state.

O Hook useState

- useState oferece duas coisas importantes:
  - Uma variável de estado (getter), que retém o dado entre renderizações.
  - Uma função de estado (setter), que atualiza o valor do state e aciona uma nova renderização.

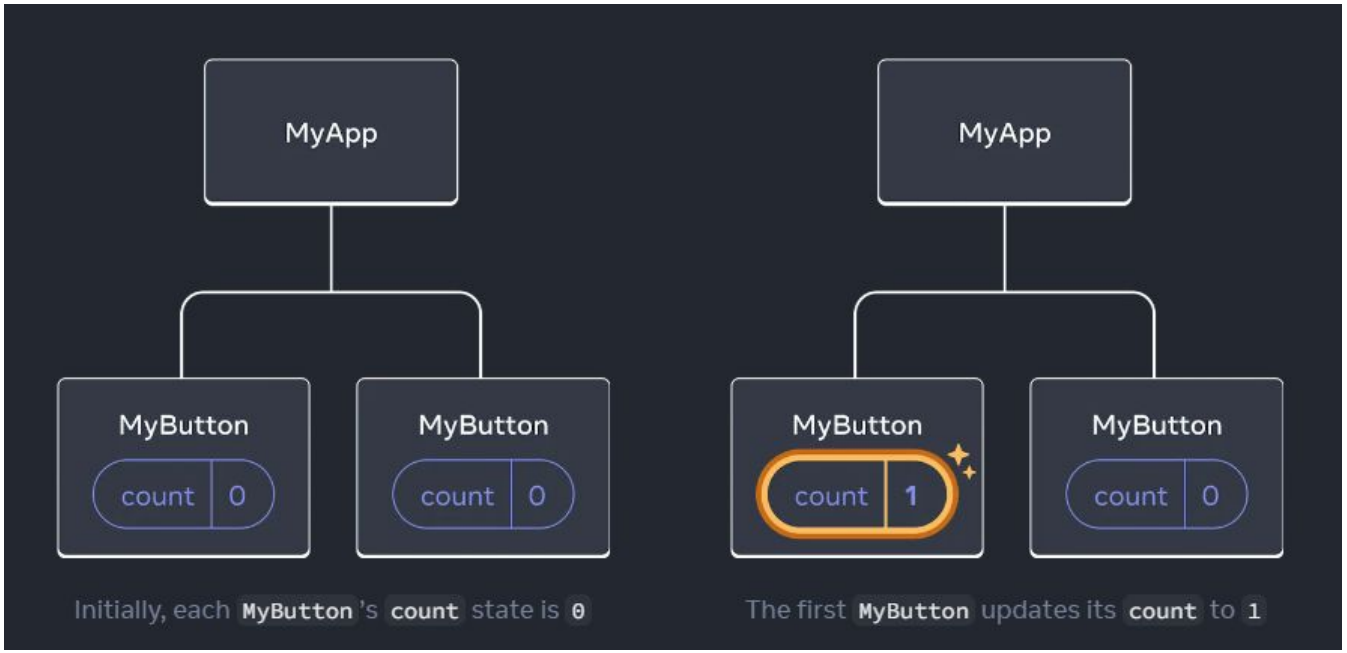
- Para utilizar precisamos importas e declarar da seguinte forma:

```
import { useState } from ‘react;
const [state, setState] = useState(initialValue);
```

- Sempre obtemos o valor através do getter e atualizamos através do setter. É importante pois o setter aciona uma re-renderização.

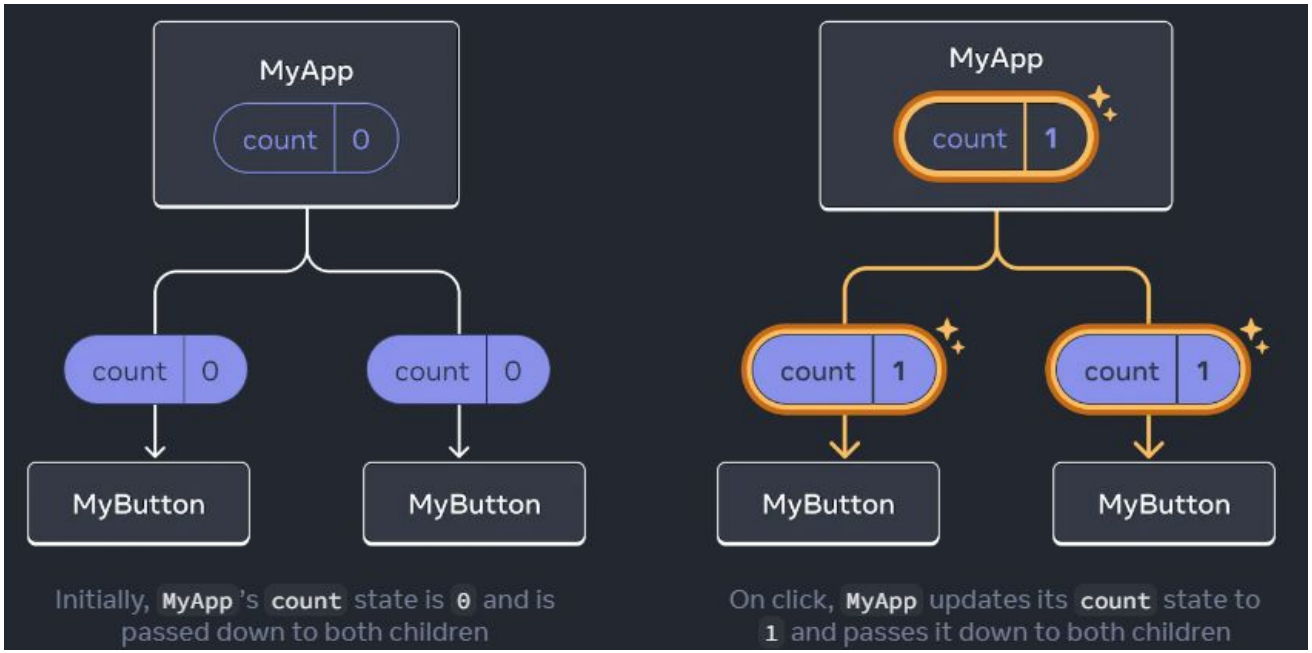
- *Lembrando que o `[]` após o const é um array destructuring*

State vive dentro de um componente



- Um state existe apenas dentro de seu componente, ele é isolado e privado.
- Mesmo que um componente seja chamado várias vezes, cada “instância” desse componente vai ter seu próprio state e um não afetará o outro.

Compartilhando States



- Frequentemente precisamos compartilhar dados entre componentes e fazer com que todos atualizem juntos.
- Para isso, basta levar o state para o componente superior (para cima), isso é chamado de *“lifting state up”.*
- Após, basta passar o state e a função de manipulação para o componente filho, “para baixo”, isso é chamado de *“props”*.

Obrigado(a)!