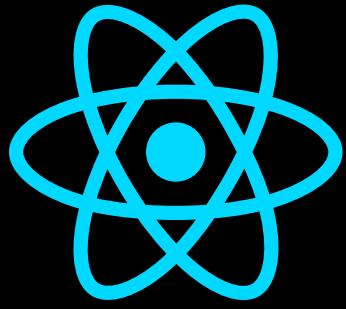


# Eventos e Estado no React



# O que iremos aprender?

- Introdução a Eventos no React
- Comunicação entre componentes
- Por que o nome handle?
- Introdução a Hooks no React
- Introdução a Estado no React
- Como usar o useState
- Manipulação de estado em React
- Funções de Atualização de Estado
- Revisando Conceitos
- Colocando em Prática - Projeto

# Introdução a Eventos no React

Em React, eventos são ações desencadeadas pelo usuário, como cliques de mouse, pressionamentos de teclas, submissões de formulários, entre outros. Assim como em JavaScript tradicional, você pode lidar com esses eventos em React para adicionar interatividade à sua aplicação.

## Ações do Usuário:

- Em uma aplicação React, os usuários interagem com a interface por meio de diferentes ações, como clicar em botões, digitar em campos de formulário, rolar a página, etc.
- Cada uma dessas ações é considerada um evento, e você pode escrever código para responder a esses eventos.

# Introdução a Eventos no React

Sintaxe de Eventos:

- Em React, os eventos são escritos em camelCase, como onClick, onChange, onSubmit, etc.
- A sintaxe de eventos em React é semelhante à usada em HTML, mas com diferenças importantes. Por exemplo, em HTML, você usa onclick, enquanto em React você usa onClick.

# Introdução a Eventos no React

```
//importando o react
import React from 'react';
//criando uma função de exemplo
function ExemploDeEvento() {
  //definindo o que será retornado:
  //um botão com o onClick em CamelCase, como
  // pede o React.
  //Dentro das chaves, estamos criando uma função
  //anônima através de uma arrow function, que
  //mostra uma mensagem de alerta na tela
  return (
    <>
    <button onClick={() => alert('Botão clicado!')}>Clique Aqui</button>
    </>
  );
}
//tornando a função exportável
export default ExemploDeEvento;
```

# Introdução a Eventos no React

No exemplo da página anterior, a função foi criada dentro do onclick, mas podemos criar a função da forma convencional também:

# Introdução a Eventos no React

```
import React from 'react';

// Função manipuladora de evento
function handleClick() {
  alert('Botão clicado!');
}

function ExemploAcoesUsuario() {
  return (
    <div>
      <h2>Clique no botão:</h2>
      {/* Passando a função handleClick como manipulador de evento */}
      <button onClick={handleClick}>Clique Aqui</button>
    </div>
  );
}

export default ExemploAcoesUsuario;
```

# Introdução a Eventos no React

Reparou que, na hora de chamar a função, não colocamos parênteses? Quando você passa uma função como propriedade em React, como em **onClick={handleClick}**, você está passando a própria função e não o resultado dela.

Em JavaScript, uma função é um tipo de dado especial que pode ser atribuído a uma variável, passada como argumento para outra função e até mesmo retornada por outra função. Quando você adiciona os parênteses, você está realmente chamando a função e passando o resultado dela.

# Introdução a Eventos no React

Se você quer apenas passar a função para ser chamada mais tarde, você não precisa adicionar os parênteses. O React vai chamar a função automaticamente quando o evento ocorrer.

É como se você estivesse dizendo: "Quando alguém clicar neste botão, aqui está a função que deve ser executada. Não a execute agora, espere até que o evento aconteça". E o React faz exatamente isso - ele executa a função quando o evento ocorre.

# Comunicação entre componentes

Quando você tem um componente pai e um componente filho em React, às vezes você quer que o componente filho possa comunicar alguma coisa de volta para o componente pai. Uma maneira de fazer isso é passando funções de evento do componente pai para o componente filho como propriedades.

Por exemplo, suponha que você tenha um botão "Play" em um componente filho e quando esse botão é clicado, você quer que uma ação seja executada no componente pai, como iniciar a reprodução de um vídeo.

# Comunicação entre componentes

Para fazer isso, você define uma função no componente pai que executa a ação desejada (como iniciar a reprodução do vídeo), e então passa essa função como uma propriedade para o componente filho. Quando o botão "Play" no componente filho é clicado, ele chama essa função que foi passada a ele como propriedade, permitindo que o componente pai saiba que a ação deve ser executada.

# Comunicação entre componentes

```
// Componente filho
// o botão vai esperar receber uma função como
// propriedade. onPlayClick é só um parâmetro.
// um nome fictício, preciso de uma função mas ainda
// não sei o nome real dela
function Button(props) {
  return <button onClick={props.onPlayClick}>Play</button>;
}
```

# Comunicação entre componentes

```
// Componente pai
function PlayButton() {
  // Função que inicia a reprodução do vídeo
  function handlePlayClick() {
    console.log('O vídeo está sendo reproduzido...');

    // Lógica para iniciar a reprodução do vídeo
  }

  return (
    <div>
      <h2>Clique no botão para reproduzir o vídeo:</h2>
      {/* Passando a função de evento como uma prop para o componente filho */}
      <Button onClick={handlePlayClick} />
    </div>
  );
}

export default PlayButton;
```

# Comunicação entre componentes

- No componente pai PlayButton, definimos a função handlePlayClick que inicia a reprodução do vídeo.
- Passamos essa função como uma propriedade chamada onPlayClick para o componente filho Button.
- No componente filho Button, definimos um evento de clique no botão que chama a função onPlayClick quando o botão é clicado.
- Assim, quando o botão "Play" no componente filho é clicado, ele chama a função handlePlayClick no componente pai, permitindo que a reprodução do vídeo seja iniciada.

# Por que o nome handle?

A convenção de nomear funções de evento com prefixo "handle" é apenas uma convenção de nomenclatura comumente usada na comunidade de desenvolvimento React. Essa prática ajuda a tornar o código mais legível e compreensível para outros desenvolvedores, especialmente quando trabalham em equipe ou revisam o código de outras pessoas.

A palavra "handle" sugere que a função está lidando com algum tipo de evento, como um clique de mouse, uma mudança de valor em um campo de entrada, entre outros. Isso torna mais claro para outros desenvolvedores que a função está sendo usada como um manipulador de evento.

# Por que o nome handle?

Além disso, ao usar essa convenção, é mais fácil identificar rapidamente as funções de evento em um código React mais extenso, o que pode facilitar a manutenção e a compreensão do código.

No entanto, é importante observar que seguir essa convenção é uma prática recomendada, mas não é uma regra estrita. Você é livre para nomear suas funções de acordo com a convenção que você e sua equipe preferirem, desde que o código seja legível e comprehensível para todos os envolvidos no projeto.

# Introdução a Hooks no React

Um hook, em React, é uma função especial que permite adicionar funcionalidades específicas a componentes funcionais. Hooks são uma forma de reutilizar lógica de estado, efeitos colaterais e outras funcionalidades em componentes funcionais sem a necessidade de escrever uma classe.

Os hooks foram introduzidos no React a partir da versão 16.8 e são fundamentais para o desenvolvimento de componentes mais complexos e reutilizáveis. Eles permitem que você divida a lógica do seu componente em funções pequenas e isoladas, facilitando a leitura, teste e manutenção do código.

# Introdução a Hooks no React

Existem diversos hooks disponíveis no React, como **useState** para adicionar estado a componentes funcionais, **useEffect** para lidar com efeitos colaterais, **useContext** para acessar contextos, entre outros. Cada hook possui uma finalidade específica e pode ser usado para resolver diferentes problemas no desenvolvimento de aplicações React.

Os hooks seguem algumas regras de nomeação e de ordem de chamada para garantir que funcionem corretamente. Eles devem sempre começar com **use** no nome para que o React possa identificá-los como hooks. Além disso, os hooks devem ser chamados apenas em componentes funcionais ou em outros hooks personalizados.

# Introdução a Hooks no React

Em resumo, um hook em React é uma função que permite adicionar funcionalidades específicas a componentes funcionais, ajudando a tornar o código mais limpo, reutilizável e fácil de entender.

# Introdução a Estado no React

O estado no React é uma forma de armazenar dados dentro de um componente. Ele representa qualquer dado que pode mudar ao longo do tempo durante a vida útil do componente. O estado é fundamental para criar componentes dinâmicos e interativos em React.

# Introdução a Estado no React

O useState é um dos hooks fornecidos pelo React que permite adicionar estado a componentes funcionais. Ele é usado para declarar uma variável de estado dentro de um componente funcional.

## Como usar o useState?

Importação:

- Primeiro, importamos a função useState do pacote react.

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

# Como usar o useState?

Declaração de Estado:

- Dentro do componente funcional, chamamos a função useState e passamos o valor inicial do estado como argumento.
- A função useState retorna um array com duas posições: a primeira é o valor atual do estado, e a segunda é uma função que permite atualizar esse valor.

```
1 const [contador, setContador] = useState(0);
```

# Como usar o useState?

Uso do Estado:

- Podemos acessar o valor atual do estado diretamente usando a variável (contador, no exemplo acima).
- Para atualizar o estado, chamamos a função que recebemos como segundo elemento do array retornado pelo useState e passamos o novo valor desejado como argumento.

```
1 setContador(contador + 1);
```

# Como usar o useState?

Exemplo Completo:

Aqui está um exemplo completo de como usar o useState em um componente funcional para criar um contador simples:

```
//fazendo a importação do React
import React, { useState } from 'react';

function Contador() {
  // Declaração de estado com valor inicial de 0
  //e criando a função que muda o valor de contador
  const [contador, setContador] = useState(0);

  // Função para incrementar o contador
  function incrementar() {
    setContador(contador + 1);
  }

  return (
    <div>
      <h2>Contador: {contador}</h2>
      <button onClick={incrementar}>Incrementar</button>
    </div>
  );
}

export default Contador;
```

# Como usar o useState?

- Criamos um componente funcional chamado Contador.
- Usamos o useState para criar uma variável de estado chamada contador, com um valor inicial de 0.
- Renderizamos o valor atual do contador dentro de um <h2>.
- Criamos uma função chamada incrementar, que incrementa o contador quando o botão é clicado.
- Chamamos setContador dentro da função incrementar para atualizar o estado do contador.
- Finalmente, associamos a função incrementar ao evento onClick do botão.

# Manipulação de Estado em React

Quando você atualiza o estado em React, você está substituindo completamente o valor anterior pelo novo valor. Isso ocorre porque os elementos do estado são imutáveis, o que significa que não podem ser alterados diretamente.

Por exemplo, se você tiver um estado que é um objeto ou um array, e quiser fazer uma atualização que não substitua completamente o estado, você pode usar o operador de propagação do JavaScript, também conhecido como spread operator (...).

# Manipulação de Estado em React

O spread operator permite criar uma cópia do estado existente e fazer modificações nessa cópia sem afetar o estado original. Isso é útil quando você deseja atualizar apenas uma parte de um objeto ou adicionar um novo elemento a um array sem perder os dados existentes.

Por exemplo, suponha que você tenha um estado que é um objeto e deseja adicionar uma nova propriedade a ele. Você pode fazer isso usando o spread operator da seguinte forma:

# Manipulação de Estado em React

O spread operator permite criar uma cópia do estado existente e fazer modificações nessa cópia sem afetar o estado original. Isso é útil quando você deseja atualizar apenas uma parte de um objeto ou adicionar um novo elemento a um array sem perder os dados existentes.

Por exemplo, suponha que você tenha um estado que é um objeto e deseja adicionar uma nova propriedade a ele. Você pode fazer isso usando o spread operator da seguinte forma:

# Manipulação de Estado em React

```
const [pessoa, setPessoa] = useState({ nome: 'João', idade: 30 });

function atualizarNome() {
  // Atualiza apenas a propriedade 'nome', mantendo 'idade' inalterada
  setPessoa({ ...pessoa, nome: 'Maria' });
}
```

Neste exemplo, `{ ...pessoa, nome: 'Maria' }` cria uma cópia do objeto `pessoa` e substitui apenas a propriedade `nome` pelo novo valor '`Maria`', mantendo a propriedade `idade` inalterada.

# Manipulação de Estado em React

Da mesma forma, se você tiver um estado que é um array e quiser adicionar um novo elemento a ele, você pode usar o spread operator para criar uma cópia do array existente e adicionar o novo elemento:

```
const [itens, setItens] = useState(['maçã', 'banana', 'laranja']);

function adicionarItem(novoItem) {
  // Adiciona um novo item ao array
  setItens([...itens, novoItem]);
}
```

# Funções de Atualização de estado

Ao trabalhar com React, é fundamental entender como atualizar o estado de um componente de forma correta e segura. O hook useState nos permite gerenciar o estado em componentes funcionais, e a função setState é essencial para atualizar esse estado. Uma característica poderosa do setState é a capacidade de aceitar uma função de atualização como argumento, que nos permite acessar o estado anterior e atualizar o estado com base nele.

# Funções de Atualização de estado

As funções de atualização no `setState` em React são utilizadas para atualizar o estado de um componente com base em seu valor anterior. Ao invés de passar diretamente um novo valor de estado para o `setState`, passamos uma função que retorna esse novo valor de estado. Essa função recebe o estado anterior como parâmetro, permitindo-nos atualizar o estado de forma segura, especialmente em situações onde múltiplas atualizações podem ocorrer ou em operações assíncronas.

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

function Exemplo() {
  const [contador, setContador] = useState(0);

  function incrementar() {
    // Usando uma função de atualização no setState
    setContador(prevContador => prevContador + 1);
  }

  return (
    <div>
      <h2>Contador: {contador}</h2>
      <button onClick={incrementar}>Incrementar</button>
    </div>
  );
}

export default Exemplo;
```

# Funções de Atualização de estado

Por que Utilizar Funções de Atualização: A utilização de funções de atualização no setState é recomendada para evitar erros durante a renderização, especialmente em cenários onde o setState é chamado em mais de um lugar ou em operações assíncronas. Ao utilizar uma função de atualização, garantimos que estamos acessando o estado mais recente e evitamos possíveis conflitos de estado que poderiam ocorrer se tentássemos atualizar o estado diretamente.

# Funções de Atualização de estado

```
// Função de atualização sem utilizar uma função de atualização
function incrementarSemFuncao() {
  // Possível erro durante a renderização se chamado em operações assíncronas
  setContador(contador + 1);
}

// Função de atualização utilizando uma função de atualização
function incrementarComFuncao() {
  // Evita erro durante a renderização, acessa o estado anterior de forma segura
  setContador(prevContador => prevContador + 1);
}
```

# Funções de Atualização de estado

`prevContador` é o valor do estado anterior, ou seja, é o valor do estado antes da atualização ocorrer. Ele é o parâmetro da função de atualização que é passada para `setContador` quando você usa uma função de atualização.

# Revisando conceitos

O que são eventos no React?

Eventos em React referem-se a ações acionadas pelo usuário, como cliques de mouse, pressionamentos de teclas, movimentos do mouse, entre outros, que são capturados e manipulados pelos componentes React. Esses eventos são tratados através de funções de evento que são associadas aos elementos da interface do usuário, permitindo que os componentes reajam dinamicamente às interações do usuário.

# Revisando conceitos

Qual a sintaxe de eventos no React?

Em React, os eventos são escritos em camelCase, como onClick, onChange, onSubmit, etc.

Precisamos colocar parênteses ao lado da função dentro do onClick?

Não. Se você quer apenas passar a função para ser chamada mais tarde, você não precisa adicionar os parênteses. O React vai chamar a função automaticamente quando o evento ocorrer.

É como se você estivesse dizendo: "Quando alguém clicar neste botão, aqui está a função que deve ser executada. Não a execute agora, espere até que o evento aconteça". E o React faz exatamente isso - ele executa a função quando o evento ocorre.

# Revisando conceitos

Como é feita a comunicação entre componentes?

1. **Props (Propriedades)**: Os componentes pais podem passar dados para os componentes filhos através de props. Esses dados são passados como atributos HTML e podem ser acessados dentro do componente filho.
2. **Estado Compartilhado**: Quando dois ou mais componentes precisam compartilhar o mesmo estado, esse estado pode ser levantado para um componente pai e passado para os componentes filhos como props.

# Revisando conceitos

Por que o nome das funções começam com handle?

A palavra "handle" sugere que a função está lidando com algum tipo de evento, como um clique de mouse, uma mudança de valor em um campo de entrada, entre outros. Isso torna mais claro para outros desenvolvedores que a função está sendo usada como um manipulador de evento.

# Revisando conceitos

O que são Hooks no React?

Em resumo, um hook em React é uma função que permite adicionar funcionalidades específicas a componentes funcionais, ajudando a tornar o código mais limpo, reutilizável e fácil de entender.

O que é o estado no React?

O estado no React é uma forma de armazenar dados dentro de um componente. Ele representa qualquer dado que pode mudar ao longo do tempo durante a vida útil do componente. O estado é fundamental para criar componentes dinâmicos e interativos em React.

# Revisando conceitos

O que é o useState?

O useState é um dos hooks fornecidos pelo React que permite adicionar estado a componentes funcionais. Ele é usado para declarar uma variável de estado dentro de um componente funcional.

Quais os 3 passos para uso do useState?

Importação, que é quando estamos chamando o hook, declaração de estado, que é criar a variável e a função que muda seu estado, e uso do estado, que é usar a função para alterar o valor da variável.

# Colocando em prática - Projeto catálogo de filmes

Projeto: Catálogo de Filmes

Neste projeto, os alunos irão criar um catálogo de filmes onde poderão adicionar, visualizar e remover filmes da lista.

Funcionalidades:

1. Componentes:

- App: Componente principal que contém toda a aplicação.
- ListaDeFilmes: Componente para exibir a lista de filmes.
- Filme: Componente para representar cada filme na lista.
- FormularioFilme: Componente para adicionar novos filmes à lista.

# Colocando em Prática

## Estado:

- Use o estado para armazenar a lista de filmes.
- Cada filme pode ser representado como um objeto com propriedades como **id**, **titulo**, **ano**, **genero**, etc.

## Renderização de Listas:

- Use o método **map()** para iterar sobre a lista de filmes e renderizar cada filme na lista.

## Eventos:

- Adicione eventos para adicionar um novo filme à lista, remover um filme da lista, etc.

# Colocando em Prática

## Spread Operator:

- Use o spread operator para adicionar novos filmes à lista sem modificar o estado existente.
- Também pode ser usado para remover um filme da lista sem modificar os outros filmes.

## Props:

- Passe as propriedades necessárias para cada componente, como o título do filme, o ano de lançamento, o gênero, etc.