CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE FRANCA

“Dr. THOMAZ NOVELINO”

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

BRUNA ERCOLINO DE SOUZA

PESQUISA: DIFERENÇAS ENTRE COMPONENTES DE CLASSE E COMPONENTES DE FUNÇÃO

FRANCA/SP

2025

O *React* é uma biblioteca de código aberto amplamente utilizado para construir interface de usuário (UI) para aplicativos *web*, permitindo que os desenvolvedores dividam a interface em componentes reutilizáveis e independentes, tornando o desenvolvimento mais eficiente e fácil de manter. Existem duas maneiras principais de criar componentes: **componentes de função** e **componentes de classe**, cada um tem sua própria sintaxe e casos de uso. Uma de suas principais características é sua abordagem baseada em componentes.

O que são componentes de *React*?

- Os componentes são os blocos de construção que encapsulam a lógica e a aparência da IU;
- São reutilizáveis, independentes e representam uma parte da IU. O *React* permite que você divida sua interface de usuário em componentes menores, o que facilita o gerenciamento e a manutenção da sua base de código.

Como dito acima, existem dois tipos principais de componentes no *React*:

1. Componente de Classe

São classes ES6 (sintaxe para criar objetos no *JavaScript*) que estendem de *React.Component* ou *React.PureComponent*. Elas possuem um *render()*, método onde você define a estrutura da interface do usuário do seu componente usando JSX. Esses componentes são utilizados quando há necessidade de gerenciar estado interno ou controlar o ciclo de vida do componente.

Sua sintaxe é mais extensa e requer mais código em relação aos componentes de função, dificultando sua leitura e manutenção, além de que sua reutilização lógica pode se tornar complicada, pois muitas vezes ela fica acoplada ao ciclo de vida do componente, tornando mais difícil sua separação e reutilização para outros contextos.

1.1 Ciclo de Vida: Os componentes de classe têm acesso a vários métodos de ciclo de vida, como *componentDidMount*, *componentDidUpdate* e *componentWillUnmount*, que permitem que você se conecte a diferentes estágios do ciclo de vida de um componente.

1.2 Gerenciamento de Estados: Os componentes de classe podem armazenar e gerenciar o estado local usando a *this.statepropriedade*. Eles também podem atualizar o estado usando *this.setState()*.

- *componentDidMount*: usado para buscar dados iniciais quando o componente é montado.
- *componentDidUpdate*: usado para registrar uma mensagem sempre que o estado dos dados muda.
- *componentWillUnmount*: usado para registrar uma mensagem antes que o componente seja desmontado.

Esses métodos de ciclo de vida fornecem ganchos para diferentes estágios do ciclo de vida de um componente, permitindo que você execute tarefas de configuração, atualização e limpeza conforme necessário.

2. Componente de Função

São funções *JavaScript* simples que recebem *props* (abreviação de *Properties*) como entrada e retornam elementos JSX. São frequentemente usadas para componentes de apresentação ou sem estado.

Sintaxe: Os componentes de função são definidos utilizando a palavra-chave *function* ou funções de seta (=>). Ambas as formas são válidas e produzem o mesmo resultado. Esse tipo de componente possui uma sintaxe mais concisa e direta, o que torna o código mais legível e fácil de compreender. No entanto, não gerenciam seu próprio estado (*state*) nem possuem acesso direto aos métodos de ciclo de vida fornecidos pelo *React* (ou *React Native*), diferente dos componentes de classe, que permitem o controle e gerenciamento de cada etapa do ciclo de vida do componente.

2.1 Ciclo de Vida: Componentes de função não possuem métodos de ciclo de vida. No entanto, com *React Hooks*, você pode usar o *useEffectHook* para replicar o comportamento do ciclo de vida.

Ao aproveitar o *useEffectHook*, os componentes de função agora podem atingir o mesmo comportamento de ciclo de vida que os componentes de classe, confundindo ainda mais a distinção entre os dois tipos de componentes.

2.2 Gerenciamento de Estados: Tradicionalmente, os componentes de função não tinham estado e não conseguiam manter seu próprio estado. No entanto, com a introdução dos *React Hooks* (como *useState*), os componentes de função agora podem gerenciar o estado usando *Hooks*.

O *useStateHook* retorna um *array* com dois elementos: o valor do estado atual (*count*) e uma função (*setCount*) para atualizar o estado.

Quando o botão é clicado, *setCount* é chamado com o novo valor de *count*, acionando uma nova renderização com o valor de estado atualizado exibido. Isso demonstra como os componentes de função agora podem manter e gerenciar seu próprio estado usando *React Hooks*, tornando-os mais poderosos e versáteis.

Os componentes de função são, geralmente, mais concisos e fáceis de ler, especialmente para componentes mais simples.

Com a introdução dos *React Hooks*, os componentes funcionais passaram a ter a capacidade de gerenciar estados e lidar com o ciclo de vida, tornando menos evidente a distinção entre **componentes de função** e **componentes de classe**. Atualmente, muitas das tarefas que antes exigiam componentes de classe podem ser implementadas de forma mais simples, legível e elegante por meio de componentes funcionais.

Os *Hooks* são funções especiais que podem utilizar outros *hooks* dentro delas, permitindo isolar e reutilizar lógicas de forma eficiente. Por convenção, o nome de um *hook* deve iniciar com o prefixo “*use*”, para que o *React* o reconheça e aplique o comportamento adequado.

Hooks como *useState*, *useEffect*, *useContext* e outros oferecem uma forma mais direta e precisa de gerenciar estados, tratar efeitos colaterais e compartilhar lógica entre componentes. Essa evolução possibilitou que os desenvolvedores

escrevessem códigos mais modulares, reutilizáveis e de fácil compreensão, reduzindo a complexidade e diminuindo a necessidade do uso de classes.

Embora os **componentes de classe** ainda tenham relevância, especialmente em projetos legados, os Hooks consolidaram os **componentes funcionais** como a principal escolha no desenvolvimento de aplicações *React* modernas, graças à sua versatilidade, clareza e facilidade de manutenção.

Além disso, devido à sua estrutura mais simples, os **componentes funcionais** geralmente apresentam melhor desempenho em comparação aos de classe, pois não possuem o custo adicional associado ao gerenciamento interno de estado e aos métodos de ciclo de vida.

De forma geral, embora os **componentes de classe** ainda tenham seu valor e continuem úteis em códigos antigos ou complexos, a tendência atual do *React* é o uso de **componentes funcionais** com *Hooks*, que oferecem uma abordagem mais moderna, eficiente e agradável para a construção de interfaces de usuário.