# Entendendo Hooks no React: Ciclo de Vida e Gerenciamento de Estado

Por Escola Dnc

# Introdução

O React é uma biblioteca poderosa para desenvolvimento web que se destaca pela sua capacidade de criar interfaces de usuário reativas e dinâmicas. Um dos conceitos fundamentais para entender e aproveitar ao máximo o React são os Hooks. Este ebook explorará em detalhes o que são Hooks, como eles se relacionam com o ciclo de vida dos componentes React e como eles revolucionaram o desenvolvimento de aplicações React.

### O Ciclo de Vida em React

### Entendendo os Eventos do Ciclo de Vida

O ciclo de vida de um componente React consiste em três eventos principais:

- 1. **Montagem (Mounting)**: Quando um componente é criado e inserido no DOM.
- 2. **Atualização (Updating)**: Quando um componente é re-renderizado devido a mudanças em props ou estado.
- 3. **Desmontagem (Unmounting)**: Quando um componente é removido do DOM.

# Exemplo Prático do Ciclo de Vida

Imagine uma aplicação React com várias páginas:

- Ao navegar da página de contato para a homepage:
  - Os componentes da página de contato são desmontados
  - Os componentes da homepage são montados
- Entre a montagem e desmontagem, ocorrem atualizações quando o usuário interage com a página

Ciclo de Vida React:1. Montagem (Mounting)2. Atualização (Updating)3. Desmontagem (Unmounting)

### O Conceito de Estado em React

# O que é Estado?

O estado em React é uma forma de armazenar e gerenciar dados que podem mudar ao longo do tempo dentro de um componente. É uma parte crucial para tornar os componentes reativos.

### **Exemplo de Estado**

Considere um componente de boas-vindas:

- 1. Estado Inicial: Exibe "Hello" e pede o nome do usuário
- 2. Atualização de Estado: Usuário insere o nome "João"
- 3. Novo Estado: Exibe "Hello João"

Este exemplo ilustra como o estado muda e afeta a renderização do componente.

# Introdução aos Hooks

### O que são Hooks?

Hooks são funções especiais que permitem que você "se conecte" aos recursos de estado e ciclo de vida do React a partir de componentes funcionais.

### Principais Benefícios dos Hooks

- Simplificam o código
- Facilitam o reuso de lógica entre componentes
- Permitem usar estado e outros recursos do React sem escrever uma classe

### Hooks e Ciclo de Vida

Os Hooks se relacionam diretamente com o ciclo de vida dos componentes:

- useState: Gerencia o estado do componente
- useEffect : Lida com efeitos colaterais, substituindo métodos de ciclo de vida como componentDidMount e componentDidUpdate

# useState: O Hook de Estado

### Como Funciona o useState

useState é um Hook que permite adicionar estado a componentes funcionais:

```
const [count, setCount] = useState(0);
```

- count : variável que armazena o estado atual
- setCount : função para atualizar o estado
- 0 : valor inicial do estado

# **Exemplo Prático de useState**

Vamos criar um contador simples:

Este exemplo mostra como useState simplifica o gerenciamento de estado em componentes funcionais.

# Comparação: Antes e Depois dos Hooks

### Código Antes dos Hooks

Antes, era necessário usar classes e métodos de ciclo de vida explícitos:

```
class ExemploContador extends React.Component {  constructor(props)
    super(props); this.state = { count: 0 }; }
componentDidMount() {
                       document.title = `Você clicou
${this.state.count} vezes`; } componentDidUpdate() {
document.title = `Você clicou ${this.state.count} vezes`; }
                         <div>
                                      Você clicou
render() {
             return (
{this.state.count} vezes <button onClick={() =>
this.setState({ count: this.state.count + 1 })}>
                                                      Clique
aqui
           </button>
                         </div>
                                  );
```

# Código Com Hooks

Com Hooks, o mesmo componente fica mais simples e conciso:

```
import React, { useState, useEffect } from 'react';function
ExemploContador() { const [count, setCount] = useState(0);
useEffect(() => { document.title = `Você clicou ${count} vezes`;
}); return ( <div> Você clicou {count} vezes
<button onClick={() => setCount(count + 1)}> Clique aqui
</button> </div> );}
```

### **Principais Diferenças**

- Sintaxe mais limpa: Menos código para a mesma funcionalidade
- Lógica mais clara: Estado e efeitos são agrupados por funcionalidade, não por método de ciclo de vida
- Reutilização mais fácil: Lógica de estado pode ser extraída e reutilizada entre componentes

### Conclusão

Os Hooks representam uma evolução significativa no desenvolvimento React, simplificando o gerenciamento de estado e ciclo de vida dos componentes. Eles permitem escrever código mais limpo, mais fácil de entender e manter, além de promover a reutilização de lógica entre componentes.

Ao dominar o uso de Hooks como useState e useEffect, os desenvolvedores podem criar aplicações React mais eficientes e com menos código. À medida que você se aprofunda no uso de Hooks, descobrirá como eles podem tornar seu código mais expressivo e sua lógica de componente mais organizada.

Lembre-se, a prática é essencial para dominar os Hooks. Experimente implementá-los em seus projetos e observe como eles podem melhorar sua experiência de desenvolvimento com React.