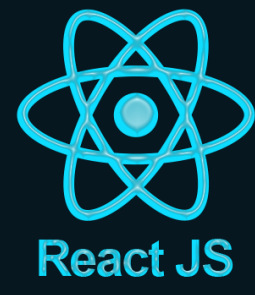
USE EFFECT

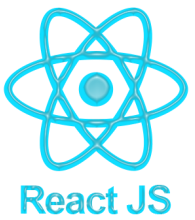
Boas práticas do Hook useEffect

Bruno Rodrigues

01

DEPENDÊNCIAS CLARAS





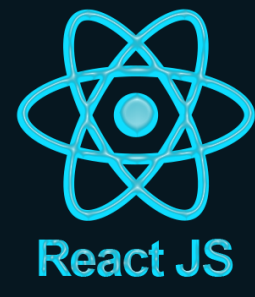
Melhor Prática 1: Dependências Claras

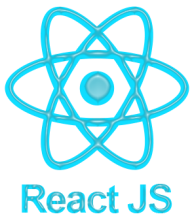
O segundo argumento do useEffect é uma array de dependências que indica quando o efeito deve ser reexecutado. Sempre declare todas as dependências usadas dentro do efeito para evitar bugs difíceis de depurar.

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

function UserProfile({ userId }) {
  const [user, setUser] = useState(null);

  useEffect(() => {
    fetch(`https://api.example.com/users/${userId}`)
      .then(response => response.json())
      .then(data => setUser(data));
  }, [userId]); // Reexecuta quando userId mudar

  if (!user) return <div>Loading...</div>;

  return <div>{user.name}</div>;
}
```

02

LIMPEZA DE EFEITOS





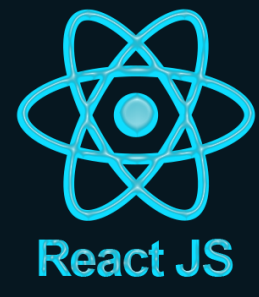
Melhor Prática 2: Limpeza de Efeitos

Quando seu efeito pode causar vazamentos de memória, como no caso de timers ou assinaturas de eventos, use uma função de limpeza para limpar o efeito.

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

function Timer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const timer = setInterval(() => {
      setCount(prevCount => prevCount + 1);
    }, 1000);

    return () => clearInterval(timer);
// Limpa o timer quando o componente desmontar
  }, []);

  return <div>Count: {count}</div>;
}
```

03

EFEITOS CONDICIONAIS



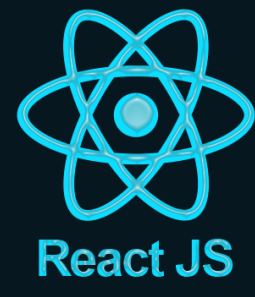
Melhor Prática 3: Efeitos Condicionais

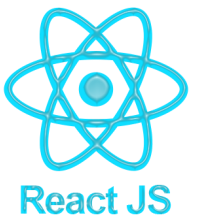
Evite execuções desnecessárias de efeitos condicionando-os às mudanças específicas das dependências.

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

function Search({ query }) {
  const [results, setResults] = useState([]);

  useEffect(() => {
    if (!query) return;

    const fetchData = async () => {
      const response = await fetch(
`https://api.example.com/search?q=${query}`);
      const data = await response.json();
      setResults(data);
    };

    fetchData();
  }, [query]); // Executa apenas quando query mudar

  return (
    <div>
      <ul>
        {results.map(result => (
          <li key={result.id}>{result.name}</li>
        ))}
      </ul>
    </div>
  );
}
```

04

EFEITOS CONDICIONAIS

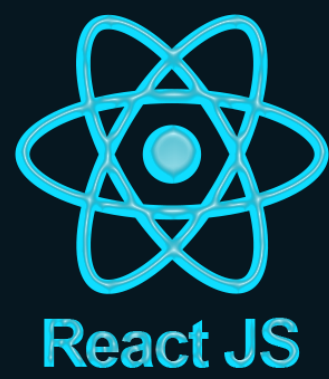




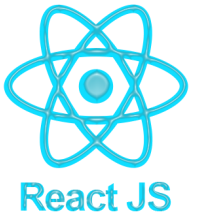
Melhor Prática 4: Separar Lógica de Negócio

Para manter seu código limpo e fácil de entender, separe a lógica de negócios em funções externas ao useEffect.

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

function fetchUserData(userId) {
  return fetch(`https://api.example.com/users/${
userId}`)
    .then(response => response.json());
}

function UserProfile({ userId }) {
  const [user, setUser] = useState(null);

  useEffect(() => {
    fetchUserData(userId).then(data => setUser(data
));
  }, [userId]);

  if (!user) return <div>Loading...</div>;

  return <div>{user.name}</div>;
}
```



CONCLUSAO



Conclusão

O useEffect é uma ferramenta essencial no arsenal de um desenvolvedor React. Ao seguir as melhores práticas apresentadas neste ebook, você garantirá que seus componentes funcionais sejam eficientes, legíveis e livres de bugs. Use dependências claras, limpe efeitos quando necessário, condicione seus efeitos, separe a lógica de negócios e evite manipular estado diretamente dentro do useEffect.

Espero que este guia tenha sido útil e que você se sinta mais confiante ao usar o useEffect em seus projetos React!