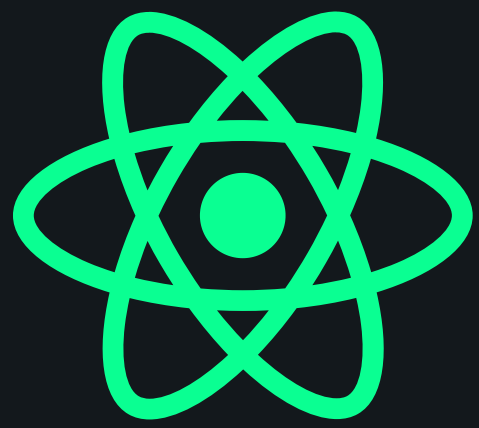
HOOKS

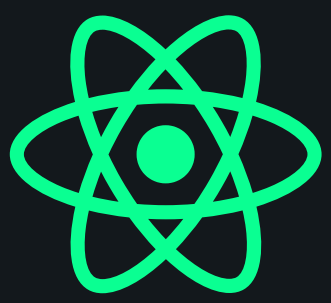


CONHEÇA OS PRINCIPAIS HOOKS DO REACT

useState

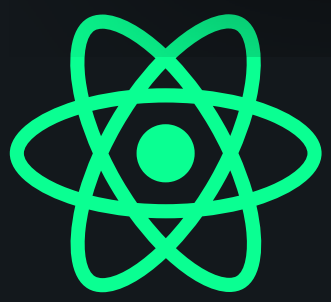
Este Hook permite que você adicione estado a componentes funcionais.

Ele retorna um array com duas posições: o valor atual do estado e uma função para atualizá-lo.



useState: na prática

```jsx
import { useState } from 'react';

function Example() {
  const [text, setText] = useState('Hello');

  const handleClick = () => {
    setText('World');
  };

  return (
    <>
      <div>{text}</div>
      <button onClick={handleClick}>Mudar texto</button>
    </>
  );
}
```

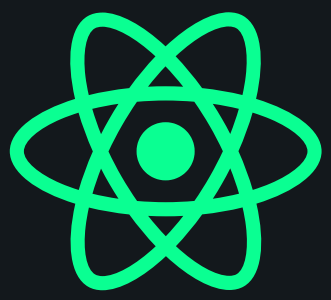


useState: explicação

Neste exemplo, estamos usando o useState para gerenciar o estado de uma string chamada "text".

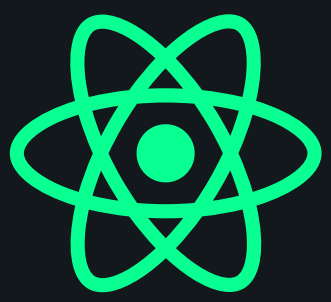
O useState retorna uma array com dois valores, o primeiro valor é o estado atual e o segundo valor é uma função para atualizar o estado.

Nós estamos desestruturando esses valores para as variáveis "text" e "setText". Inicialmente, o text é inicializado com o valor "Hello".



useEffect

Este Hook permite que você adicione efeitos colaterais a componentes funcionais, como carregar dados de uma API, ou atualizar o DOM.



useEffect: na prática

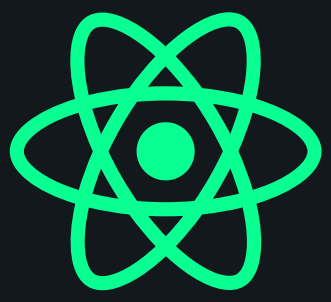
```jsx
import { useState, useEffect } from 'react';

function Example() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('https://example.com/data')
      .then(response => response.json())
      .then(data => setData(data))
      .catch(error => console.log(error));
  }, []);

  return (
    <>
      {data ? (
        <div>{JSON.stringify(data)}</div>
      ) : (
        <div>Carregando...</div>
      )}
    </>
  );
}
```

useEffect: explicação

Neste exemplo, estamos usando o useState para gerenciar o estado do componente, onde data é o estado e setData é a função para atualizar o estado.

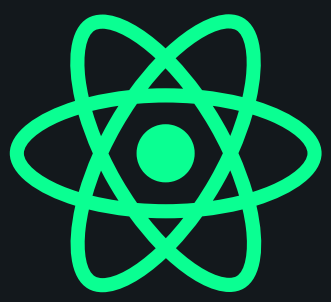
Dentro do useEffect, estamos fazendo uma requisição HTTP usando o fetch para recuperar os dados de uma URL específica.

O fetch retorna uma promise, então usamos o método then para obter o objeto json e usando o setData para atualizar o estado com os dados retornados



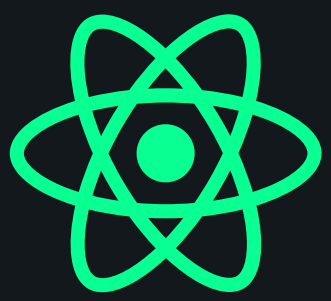
useReducer

Este Hook é semelhante ao useState, mas é mais adequado para gerenciar estados complexos ou estados compartilhados entre múltiplos componentes.



useReducer: na prática

```jsx
import React, { useReducer } from 'react';

const initialState = { count: 0 };

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>+</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
    </>
  );
}
```

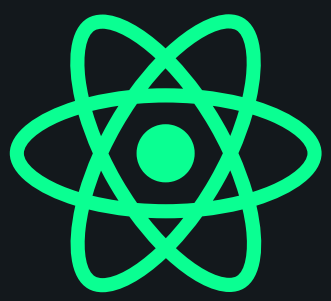


useReducer: explicação

Neste exemplo, usamos o useReducer para gerenciar o estado do contador.

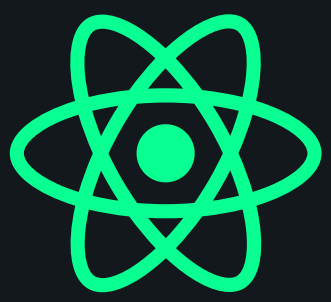
Ele tem um estado inicial e um reducer que é responsável por lidar com as ações de incremento e decremento.

A função dispatch é usada para enviar ações para o reducer, e o estado atualizado é retornado e armazenado na variável state.



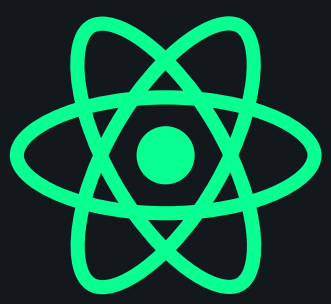
useCallback

Este Hook permite que você retorne uma função com uma referência constante, evitando a recriação desnecessária de funções dentro de componentes.



useCallback: na prática

```jsx
import React, { useCallback } from 'react';

function Parent({ value }) {
  const handleClick = useCallback(() => {
    console.log(value);
  }, [value]); // useCallback passa um array de dependencia,
  //a função só será recriada quando o valor for alterado

  return (
    <Child onClick={handleClick} />
  );
}

function Child({ onClick }) {
  return <button onClick={onClick}>Click me!</button>;
}
```

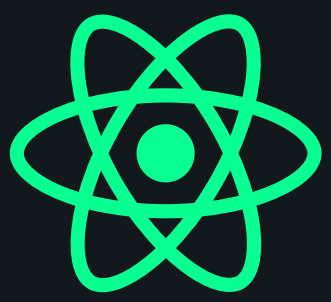


useCallback: explicação

Neste exemplo, o componente Parent passa a função handleClick como prop para o componente Child.

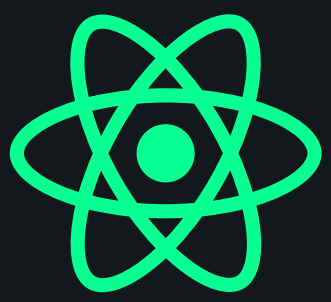
Usando useCallback, garantimos que a função handleClick só será recriada se o valor de value mudar.

Isso evita que o componente Child seja redesenhado desnecessariamente quando o componente Parent é renderizado, o que pode melhorar o desempenho da aplicação.



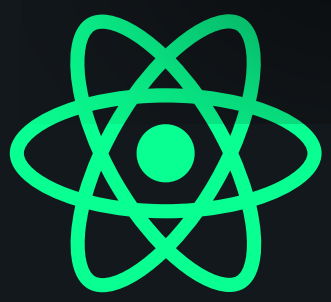
useRef

Este Hook permite que você crie referências para elementos do DOM e outros valores, permitindo acessá-los fora da árvore de componentes.



useRef: na prática

```jsx
import { useRef } from 'react';

function Example() {
  const inputRef = useRef(null);

  const handleClear = () => {
    inputRef.current.value = '';
  };

  return (
    <>
      <input ref={inputRef} type="text" />
      <button onClick={handleClear}>Limpar</button>
    </>
  );
}
```



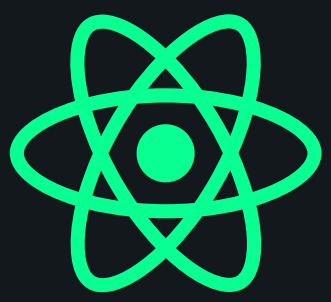
useRef: explicação

Neste exemplo, estamos usando o useRef para criar uma referência ao elemento de input.

A referência é armazenada na variável inputRef. Nós então passamos essa referência como um atributo ref para o elemento de input.

Isso permite que o React atribua o elemento de input para inputRef.current.

Dentro do método handleClear, podemos acessar o valor do elemento de input através do objeto inputRef.current e limpá-lo.



COMUNIDADE
HORA DE CODAR

Buscando a sua primeira vaga como dev?

Confira o link no post para mais informações

@horadecodar

Matheus Battisti - Hora de Codar