
otterwise

HOOKS

otterwise

Hooks

- Funções que permitem a conexão de uma funcionalidade ao componente.
- Torna possível o acesso ao estado e manipulação do ciclo de vida do componente, além de outras coisas como memorização de funções e valores.
- Só podem ser utilizados em componentes funcionais ou Hook customizado.
- Facilitam a reutilização de código e melhoram a coesão dos componentes.

Referência: Hooks de forma resumida – React

ESTADO DE UM COMPONENTE

otterwise

Estado de um Componente

- Funcionalidade que permite a manipulação de informações internamente no componente
- Cada componente guarda seu próprio estado
- O componente é renderizado novamente quando o estado muda, de forma inteligente.
- Hook: *useState*.

Referência: API de Referência dos Hooks – React

Estado de um Componente

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

function Example() {
  const [count, setCount] = useState(0);
  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Referência: [Hooks FAQ – React]

otterwise

EXERCÍCIO

Exercício

Crie um contador onde o usuário pode incrementar ou decrementar o valor na tela.

Mostre o valor atual do contador na tela e crie dois botões, um para incremento e outro para decremento.

Utilize o hook *useState* para criar a lógica.

CICLO DE VIDA

otterwise

Ciclo de Vida de um Componente

- **Montagem do componente:** quando o componente é carregado na tela pelo React.
- **Atualização das propriedades:** toda vez que alguma propriedade que é passada para o componente atualiza, o componente renderiza novamente aplicando as mudanças da propriedade.
- **Modificação do estado:** cada vez que modificamos o estado de um componente, ele renderiza novamente.
- **Desmontagem do componente:** quando o componente não precisa ser mais mostrado na tela, ele é desmontado pelo React.

useEffect

- Com o hook ***useEffect*** podemos atrelar execuções de funções específicas aos momentos do ciclo de vida do componente.
- **Atenção**: A função presente no ***useEffect*** também será executada na montagem do componente.

```
import { useEffect } from "react";

function User(props) {
  const { userId } = props

  useEffect(() => {
    const request = async () => {
      const response = await (
        await
fetch(`https://jsonplaceholder.typicode.com/users/${userId}`)
      ).json();
      console.log(response);
    };

    request();
  }, [userId]);
  return <h2>Fetch user with id: {userId}</h2>;
}
export default User;
```

Referências: [API de Referência dos Hooks – React]

otterwise

EXERCÍCIO

Exercício

Crie um componente que carrega os dados da fake api (https://jsonplaceholder.typicode.com/), mas o recurso a ser carregado pode ser de “users”, “todos” ou “posts”. Através de um botão na tela, deve ser possível gerar aleatoriamente um dos 3 recursos que será carregado.

O resultado do carregamento desses recursos deve ser mostrado na tela em formato de string.

Dica: utilize JSON.stringify para mostrar os dados na tela

HOOKS ADICIONAIS

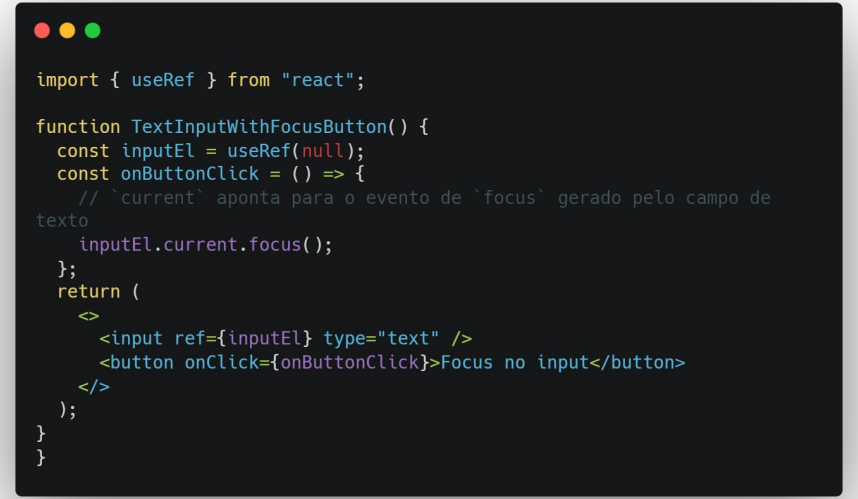
otterwise

useRef

otterwise

useRef

Guarda uma referência ao elemento, onde o valor dele pode ser acessado e modificado sem disparar uma nova renderização.



```
import { useRef } from "react";

function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` aponta para o evento de `focus` gerado pelo campo de
texto
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus no input</button>
    </>
  );
}
}
```

Referência: [API de Referência dos Hooks – React]

otterwise

useCallback

otterwise

useCallback

Memoriza uma função para que não seja redefinido a cada nova renderização.

```
import { useCallback } from "react";

const memoizedCallback = useCallback(
  () => {
    doSomethingExpensive(a, b);
  },
  [a, b],
);
```

Referências: [API de Referência dos Hooks – React]

otterwise

useMemo

useMemo

Memoriza um valor para que não seja redefinido a cada nova renderização.

```
import { useMemo } from "react";

const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

Referências: [API de Referência dos Hooks – React]

otterwise

useReducer

useReducer

Uma outra maneira de lidar com estado de um componente é utilizando um reducer para manter o estado sempre atualizado. Esse hook é utilizado quando o estado que deve ser mantido é complexo.

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

Referência: [API de Referência dos Hooks – React]

otterwise

EXERCÍCIO

Exercício

Faça um componente que contém um botão, e ao clicar neste botão, carrega uma lista de usuários através do endpoint (https://jsonplaceholder.typicode.com/users) da fake api. Mostre para o usuário na tela cada status da requisição.

Os status possíveis são:

- **idle**: a requisição ainda não foi feita (vulgo estado inicial);
- **pending**: a requisição já foi feita para o servidor mas ainda não retornou;
- **error**: a requisição deu errado;
- **success**: a requisição deu certo.

Utilize o hook useReducer para controlar o status da requisição.