React: Hooks

Guia de Desenvolvimento Front-End, conhecendo os Hooks

O que são os Hooks?

Hooks nada mais são do que funções que permitem usar estados e ciclos de vida em componentes funcionais, permitindo que componentes tenham memória própria (estado) ou reajam a eventos de montagem e atualização

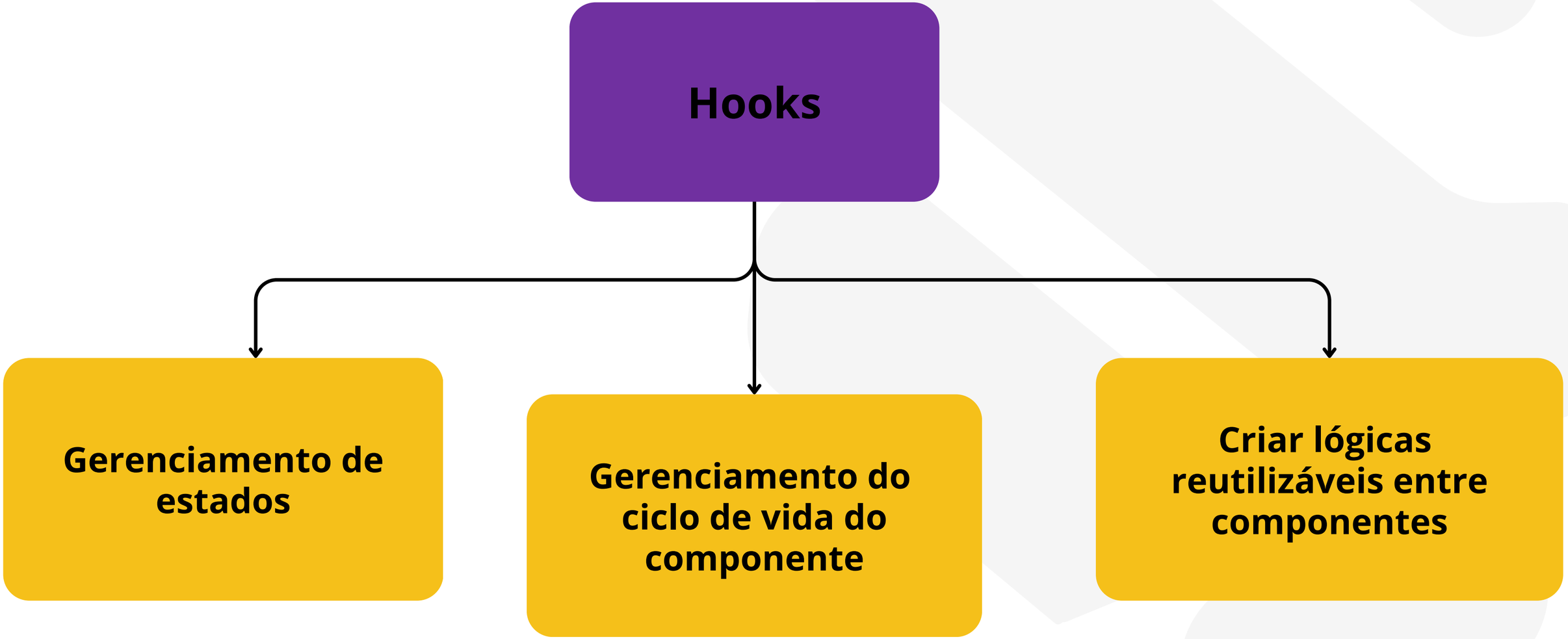
Mas porque usar os Hooks?

O uso de hooks evita o uso de classes, deixando o código mais simples e reutilizável.
Ou seja, escrevemos tudo em uma funções simples, feitas propriamente para uma utilidade específica, veja uma estrutura comum em funções hooks a baixo:

<declaração> [<variavel_de_estado>, <variavel_de_controle_de_estado>] = use<função_hook>

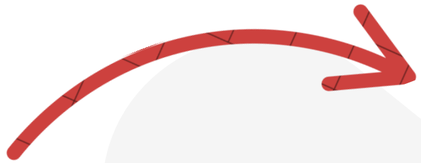
Davi Cândido de Almeida

Resumindo


Hooks
Gerenciamento de estados
Gerenciamento do ciclo de vida do componente
Criar lógicas reutilizáveis entre componentes

Aprofundando na estrutura de um hook

<declaração> [<variavel_de_estado>, <variavel_de_controle_de_estado>] = use<função_hook>



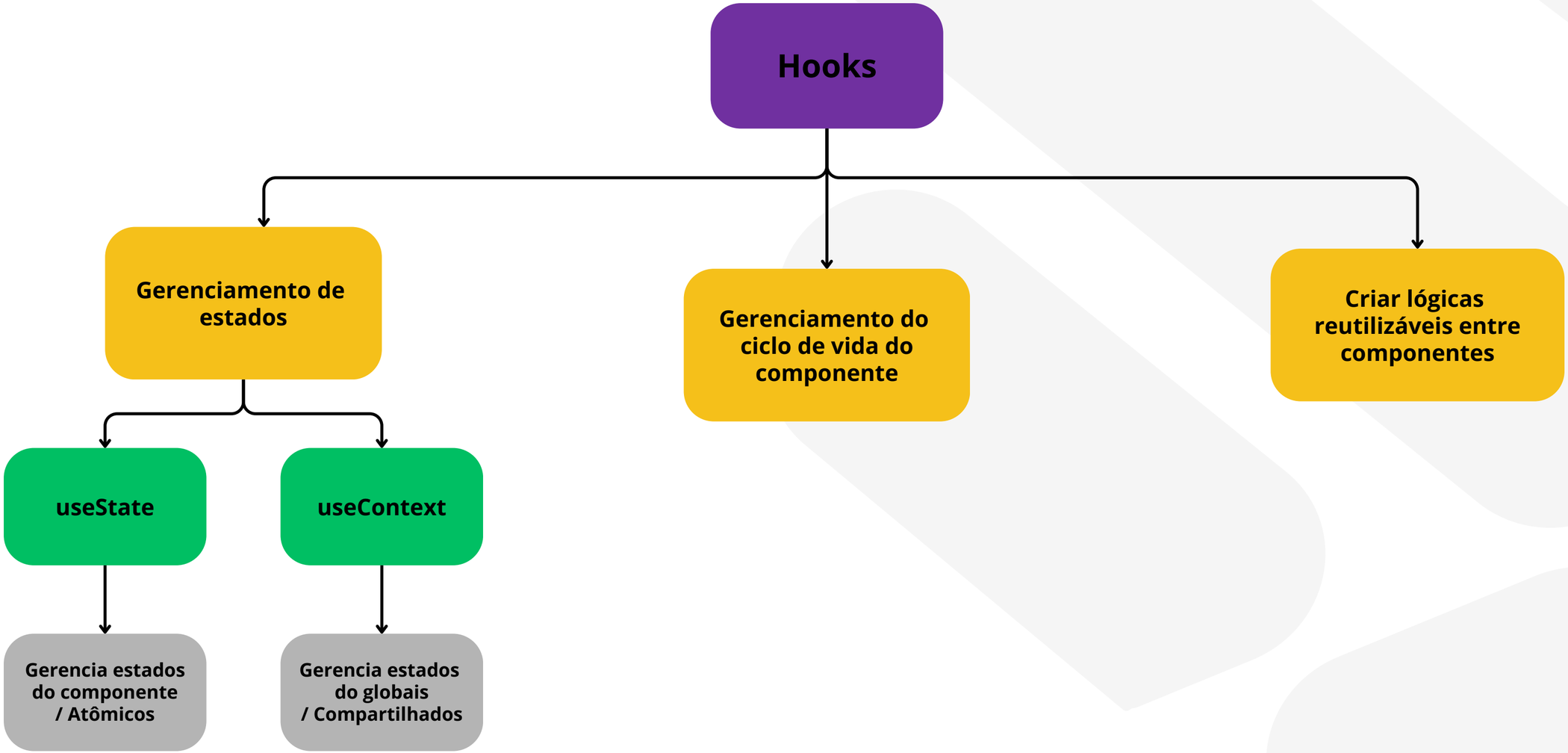
Veja mais:

https://react.dev/reference/react/hooks

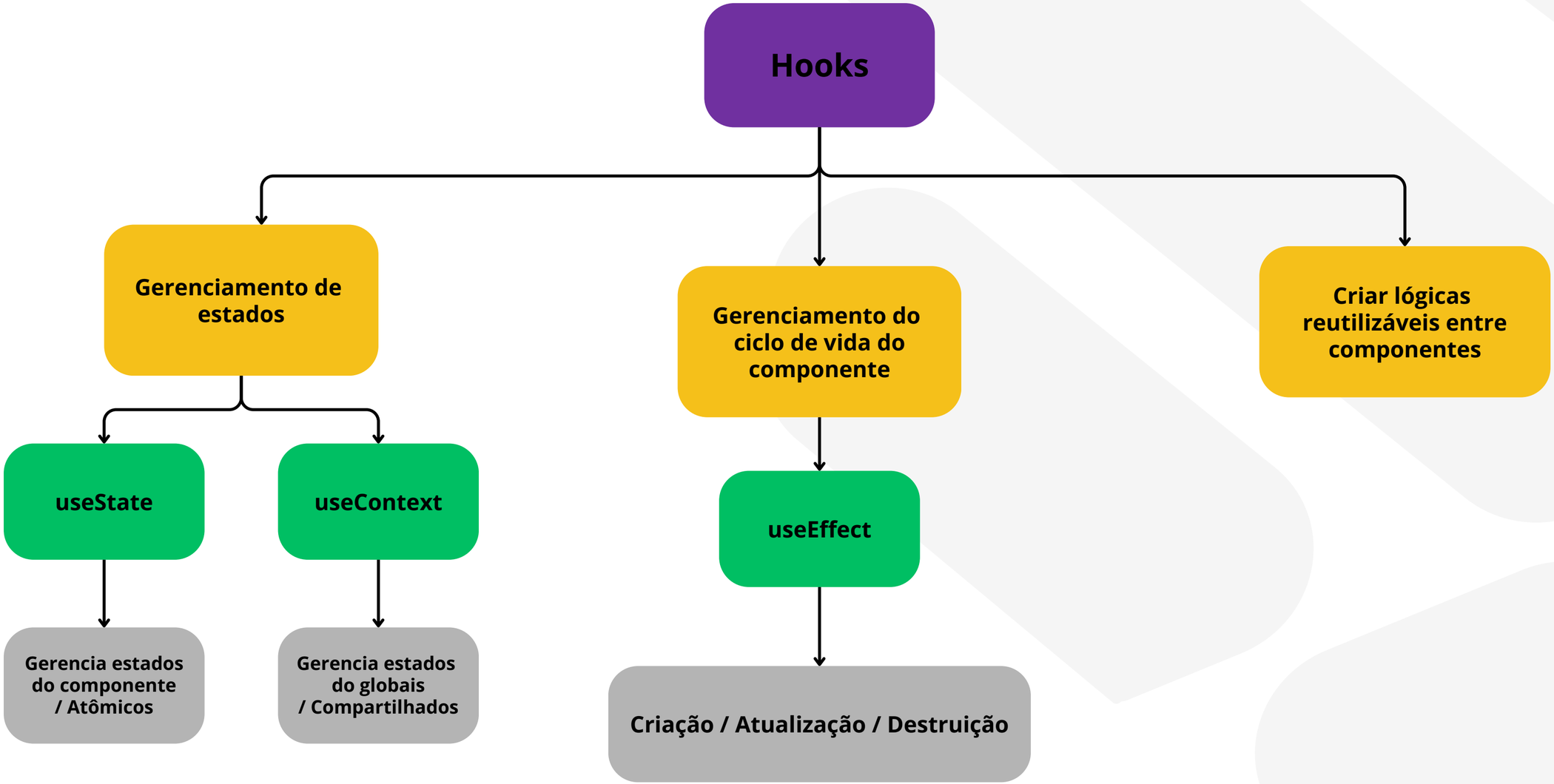
```
const [isPress, setIsPress] = useState( )

let [count, setCount] = useRef( )

var [colorTheme, setColorTheme] = useContext( )

useEffect( ( ) => {
  // Função
 }, [<variável>]) ;
```


Davi Cândido de Almeida

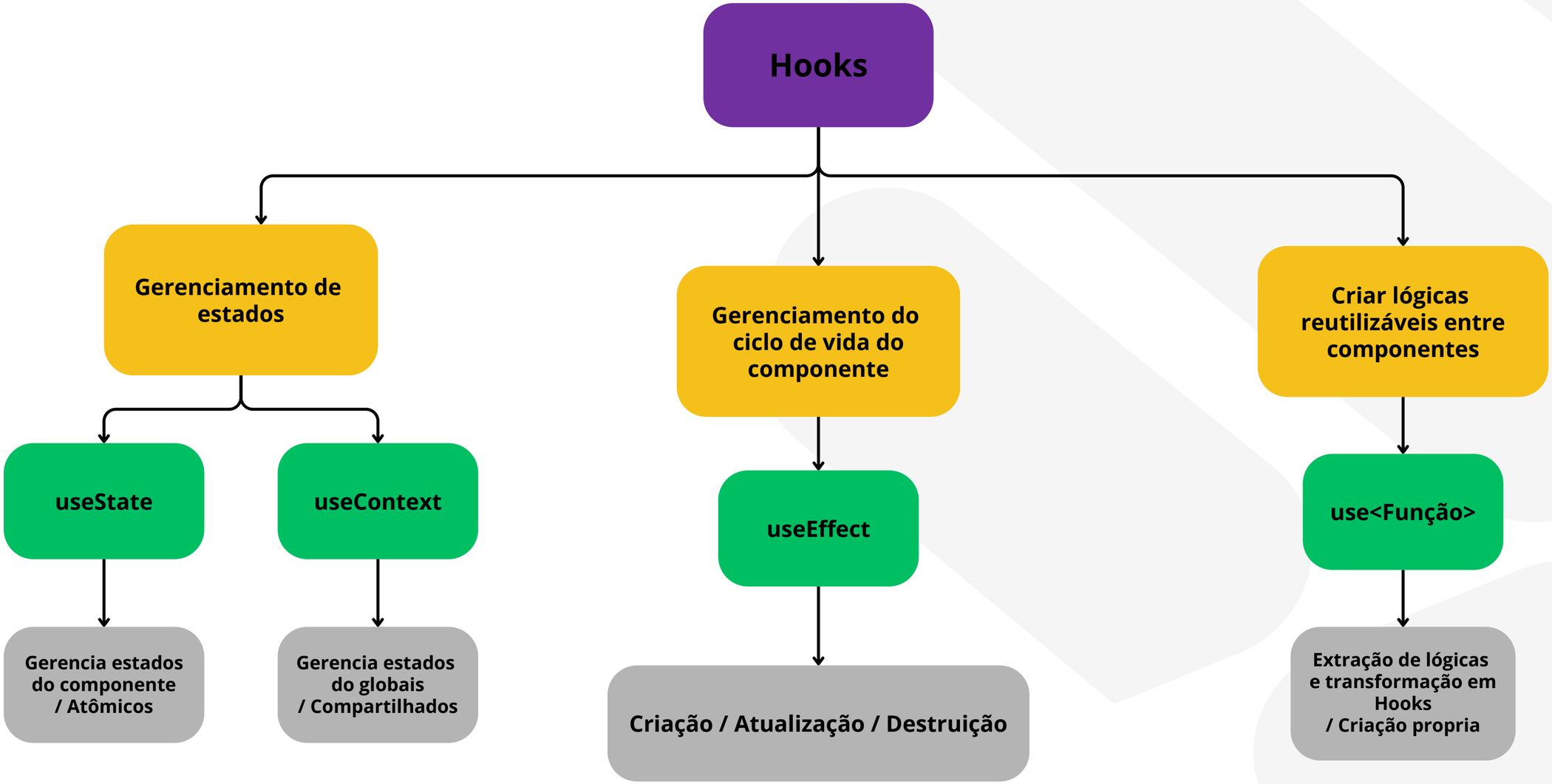
Conhecendo alguns Hooks



Conhecendo alguns Hooks



Conhecendo alguns Hooks



Conhecendo alguns Hooks

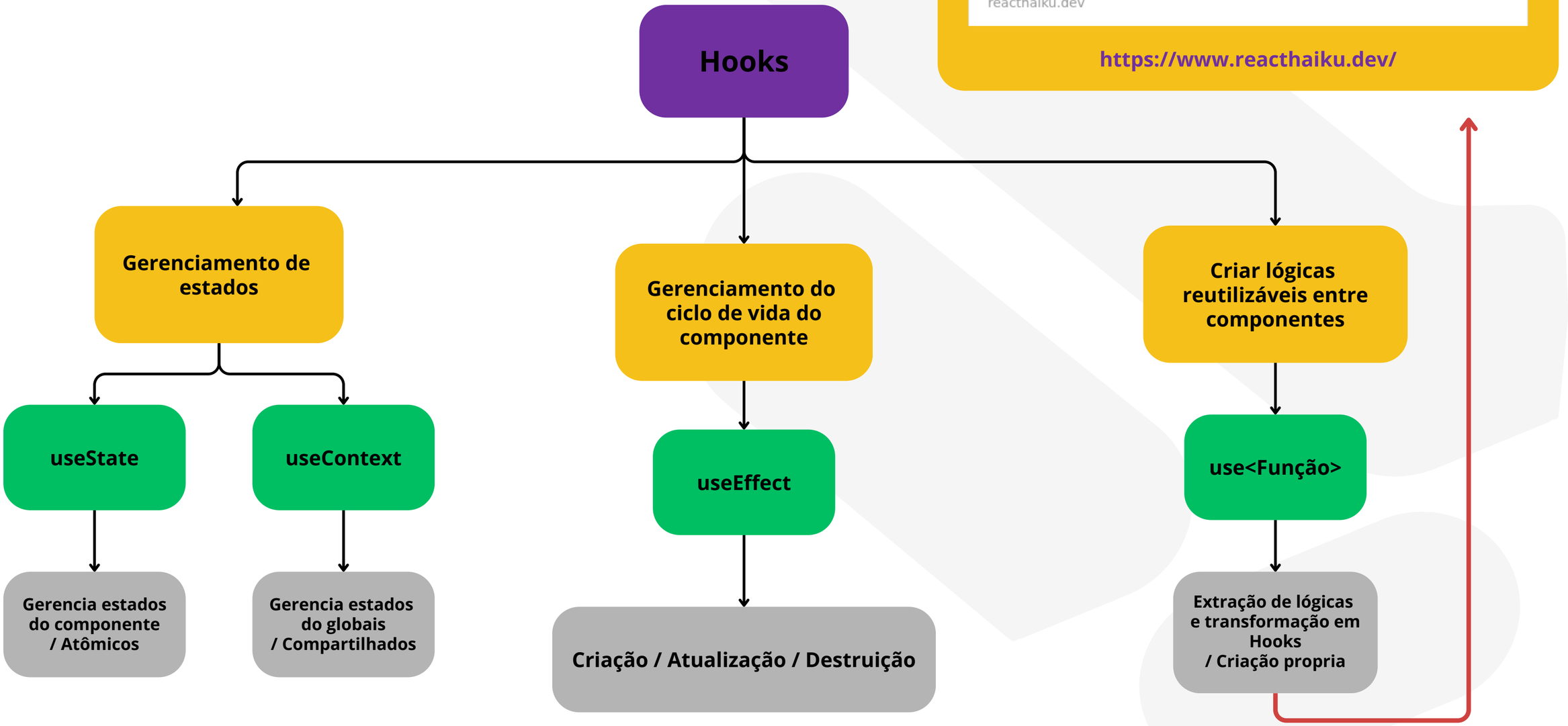
Explore Hooks Feitos pela comunidade:

Home | React Haiku

A minimal React Hooks library that saves you time and lines of code.

reacthaiku.dev

https://www.reacthaiku.dev/



Conhecendo um Hooks: UseState( )

- **useState Utilidade:** Armazena valores dinâmicos dentro do componente.
- **Funcionamento:** Retorna um estado e uma função para atualizá-lo.
- **Boas prática:** Inicialize o estado com um valor adequado ao tipo esperado

```jsx
export default function useStateComponent() {
  const [count, setCount] = useState(0);

  function incrementar() {
    setCount(count + 1);
  }

  function decrementar() {
    setCount(count - 1);
  }

  return (
    <ControlPageHooks>
      <h2>useContext</h2>
      <p>Contagem: {count}</p>
      <div className="flex gap-2">
        <button onClick={incrementar}>Incrementar</button>
        <button onClick={decrementar}>decrementar</button>
        <button onClick={() => setCount(0)}>Resetar</button>
      </div>
    </ControlPageHooks>
  );
}
```


Davi Cândido de Almeida

Conhecendo um Hooks: useEffect( )

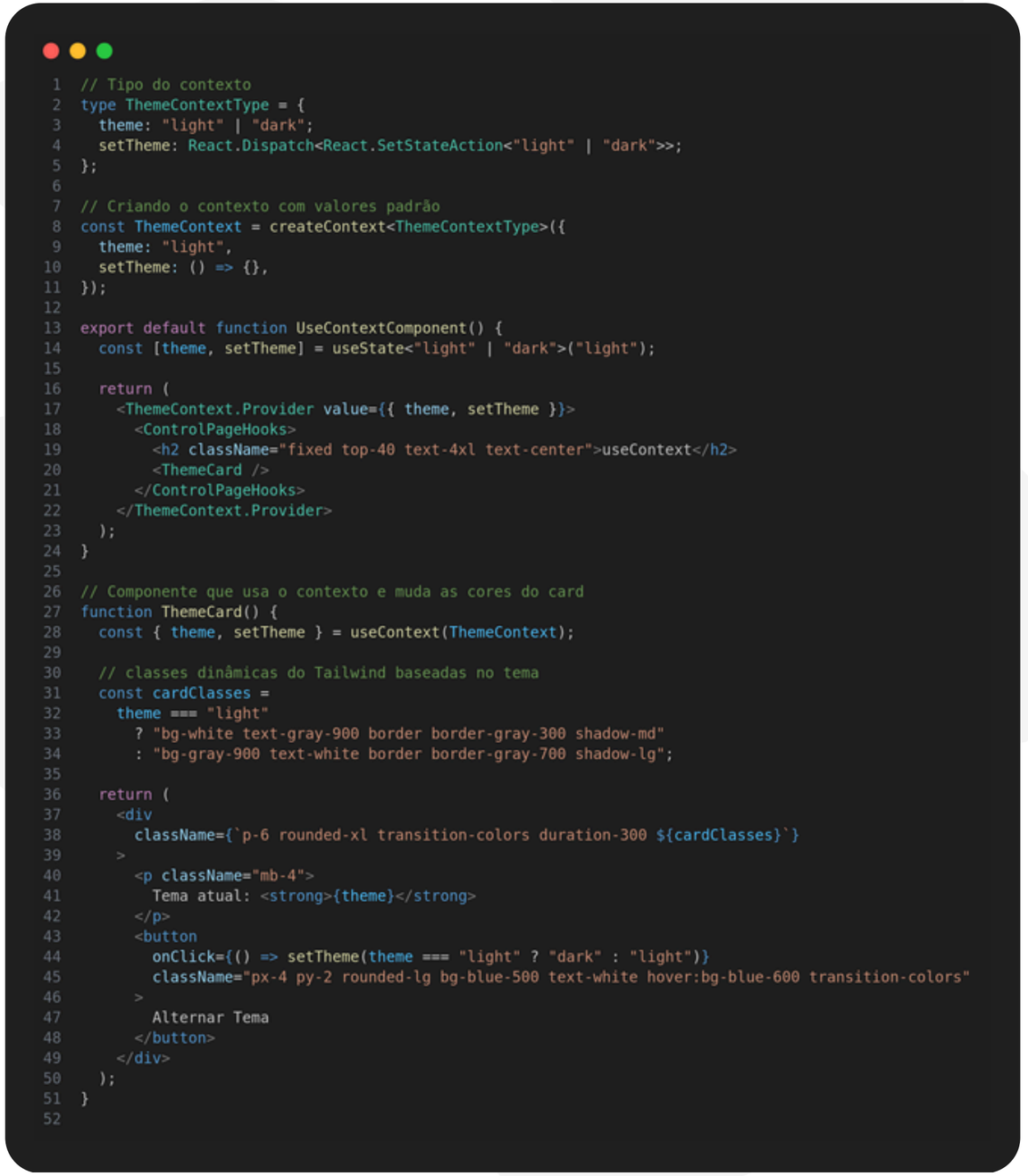
- **useEffect Utilidade:** Executa efeitos colaterais como timers, chamadas de API ou listeners.
- **Funcionamento:** Recebe uma função e um array de dependências para controlar quando rodar.
- **Boas prática:** Sempre declare as dependências necessárias para evitar reexecuções infinitas.

```jsx
export default function UseEffectComponent() {
  const [value, setValue] = useState(0);
  const [checked, setChecked] = useState(false);

  useEffect(() => {
    console.log("useEffect !!! value: " + value + " checked: " + checked);
  }, [value]);


  return (
    <ControlPageHooks>
      <h2 className="fixed top-40 text-4xl text-center ">
        useEffect
      </h2>
      <p className="fixed top-60 text-2xl text-center">
        Veja o console:
      </p>
      <div className="flex gap-2 justify-center items-center">
        <button onClick={() => setValue(value + 1)}>Incrementar</button>
        <input
          type="checkbox"
          checked={checked}
          onChange={(e) => setChecked(e.target.checked)}
        />
      </div>
    </ControlPageHooks>
  );
}
```


Davi Cândido de Almeida

Conhecendo um Hooks: useContext( )

- **useContext Utilidade:** Compartilha dados entre componentes sem precisar passar props manualmente.
- **Funcionamento:** Usa Context.Provider para fornecer valores e useContext para consumi-los.
- **Boa prática:** Use em dados realmente globais (tema, autenticação, idioma).



```tsx
// Tipo do contexto
type ThemeContextType = {
  theme: "light" | "dark";
  setTheme: React.Dispatch<React.SetStateAction<"light" | "dark">>;
};

// Criando o contexto com valores padrão
const ThemeContext = createContext<ThemeContextType>({
  theme: "light",
  setTheme: () => {},
});

export default function UseContextComponent() {
  const [theme, setTheme] = useState<"light" | "dark">("light");

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      <ControlPageHooks>
        <h2 className="fixed top-40 text-4xl text-center">useContext</h2>
        <ThemeCard />
      </ControlPageHooks>
    </ThemeContext.Provider>
  );
}

// Componente que usa o contexto e muda as cores do card
function ThemeCard() {
  const { theme, setTheme } = useContext(ThemeContext);

  // classes dinâmicas do Tailwind baseadas no tema
  const cardClasses =
    theme === "light"
      ? "bg-white text-gray-900 border border-gray-300 shadow-md"
      : "bg-gray-900 text-white border border-gray-700 shadow-lg";

  return (
    <div
      className={`p-6 rounded-xl transition-colors duration-300 ${cardClasses}`}
    >
      <p className="mb-4">
        Tema atual: <strong>{theme}</strong>
      </p>
      <button
        onClick={() => setTheme(theme === "light" ? "dark" : "light")}
        className="px-4 py-2 rounded-lg bg-blue-500 text-white hover:bg-blue-600 transition-colors"
      >
        Alternar Tema
      </button>
    </div>
  );
}
```


Davi Cândido de Almeida

Conhecendo um Hooks: useReducer( )

- **Utilidade:** Gerencia estados mais complexos com múltiplas transições.
- **Funcionamento:** Usa uma função reducer que recebe estado + ação e retorna um novo estado.
- **Boa prática:** Organize as ações em constantes ou enums para evitar erros de digitação.

```tsx
const reducer = (state: any, action: any) => {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    case "RESET":
      return { count: 0 };
    default:
      return state;
  }
};

export default function useReducerComponent() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <ControlPageHooks>
      <h2 className="fixed top-40 text-4xl text-center">useReducer</h2>
      <div className="flex flex-col gap-4 justify-center mt-24 mb-2">
        <p>Contagem: {state.count}</p>
      </div>
      <div className="flex gap-2">
        <button onClick={() => dispatch({ type: "INCREMENT" })}>
          Incrementar
        </button>
        <button onClick={() => dispatch({ type: "DECREMENT" })}>
          decrementar
        </button>
        <button onClick={() => dispatch({ type: "RESET" })}>Resetar</button>
      </div>
    </ControlPageHooks>
  );
}

```


Davi Cândido de Almeida

Conhecendo um Hooks: useRef( )

- **Utilidade:** Armazena valores mutáveis que não causam re-render.
- **Funcionamento:** Retorna um objeto .current que mantém o valor entre renderizações.
- **Boa prática:** Use para manipular DOM ou guardar valores que não precisam atualizar a UI.

```jsx
export default function UseRefComponent() {
  const [name, setName] = useState("");
  const renders = useRef(0);

  useEffect(() => {
    renders.current = renders.current + 1;
  });

  return (
    <ControlPageHooks>
      <h2 className="fixed top-40 text-4xl text-center">useRef</h2>
      <div className="flex gap-4 justify-center mt-24">
        <input
          value={name}
          type="text"
          placeholder="Digite algo"
          onChange={(e) => setName(e.target.value)}
        />
      </div>
      <div className="flex flex-col gap-4 justify-center mt-24">
        <p>Hello! My name is {name}</p>
        <p>Render count: {renders.current}</p>
      </div>
    </ControlPageHooks>
  );
}

```


Davi Cândido de Almeida

Conhecendo um Hooks: useMemo( )

- **Utilidade:** Memoriza valores derivados de cálculos pesados para evitar recomputações.
- **Funcionamento:** Recebe uma função de cálculo e dependências.
- **Boa prática:** Só use quando notar gargalos de performance — não abuse.

```tsx
"use client";
import { useState, useMemo } from "react";

export default function useMemoComponent() {
  const [number, setNumber] = useState(1);
  const [text, setText] = useState("");

  const doubleNumber = useMemo(() => {
    return slowFunction(number);
  }, [number]);

  return (
    <>
      <p>{number}</p>
      <input value={text} onChange={(e) => setText(e.target.value)} />
      <button onClick={() => setNumber(2)}>Increment</button>
      <p>text: {text}</p>
    </>
  );
}

const slowFunction = (num: number) => {
  console.log("Slow function is being called!");
  for (let i = 0; i <= 10000; i++) {}
  return num * 2;
};
```


Davi Cândido de Almeida

Conhecendo um Hooks: use( ) (novo)

- **use Utilidade:** Permite usar Promises diretamente dentro do componente (principalmente em React Server Components no Next.js 13+).
- **Funcionamento:** Você passa uma Promise para o use e ele “pausa” a renderização até que o valor seja resolvido.
- **Boa prática:** Use apenas em Server Components (não no cliente)

```tsx
// Função que retorna uma promise
async function fetchUser() {
  const res = await fetch("https://jsonplaceholder.typicode.com/users/1");
  if (!res.ok) {
    throw new Error("Error fetching user");
  }
  return res.json();
}

// Componente do servidor que já retorna os dados
function UserDetails() {
  const user = use(fetchUser()); // ✅ só Server Component pode usar use()
  return (
    <div className="p-4 bg-white rounded shadow">
      <h2 className="text-xl font-bold">Usuário</h2>
      <p>
        <strong>Nome:</strong> {user.name}
      </p>
      <p>
        <strong>Email:</strong> {user.email}
      </p>
    </div>
  );
}

// Componente principal do servidor
export default function UseComponent() {
  return (
    <ControlPageHooks noBackButton={true}>
      <h1 className="text-2xl mb-4">Demo do novo `use()` Hook</h1>
      <UserDetails /> {/* Não precisa de Suspense no servidor */}
    </ControlPageHooks>
  );
}

```


Davi Cândido de Almeida

Criação de Hooks personalizados: use<logica>( )

- **Utilidade:** Extrai e reutiliza lógica entre componentes.
- **Funcionamento:** É uma função que pode chamar outros Hooks e retorna valores ou funções.
- **Boa prática:** Sempre nomeie com use no início (useAuth, useForm) e mantenha foco em uma responsabilidade.
  - Cria uma pasta **hooks/** para armazernar todos os seus hooks

```jsx
export default function useCriadoComponent() {
  const { count, increment, decrement, reset } = useCounter();

  return (
    <ControlPageHooks>
      <h2 className="fixed top-40 text-4xl text-center ">
        Hook criado (useCounter)
      </h2>
      <div className="flex flex-col gap-4 justify-center mt-24 items-center-safe">
        <p>Contagem: {count}</p>
        <div className="flex gap-2">
          <button onClick={increment}>Incrementar</button>
          <button onClick={decrement}>decrementar</button>
          <button onClick={reset}>Resetar</button>
        </div>
      </div>
    </ControlPageHooks>
  );
}
```

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

export default function useCounter() {
  const [count, setCount] = useState(0);

  function increment() {
    setCount(count + 1);
  }

  function decrement() {
    setCount(count - 1);
  }

  function reset() {
    setCount(0);
  }

  return {
    count,
    increment,
    decrement,
    reset,
  }
}
```


Davi Cândido de Almeida

Um universo rico de Hooks...

// Hooks Básicos

- **useState( )** → Armazena e atualiza estado local dentro do componente.
- **useEffect( )** → Executa efeitos colaterais (ex.: chamadas de API, timers, eventos).
- **useContext( )** → Acessa valores do Context API sem precisar de props.
- **useRef( )** → Guarda valores mutáveis ou acessa elementos do DOM sem causar re-render.

// Hooks de Performance

- **useMemo( )** → Memoriza o resultado de cálculos caros para evitar recomputar sempre.
- **useCallback( )** → Memoriza funções para não recriar em cada renderização.

// Hooks de Estrutura/Estado

- **useReducer( )** → Alternativa ao useState, útil para estados mais complexos com várias transições.
- **useTransition( )** → Marca atualizações como não urgentes, deixando a UI mais fluida.
- **useDeferredValue( )** → Retarda um valor e mantém a UI responsiva durante cálculos pesados.

Davi Cândido de Almeida

Um universo rico de Hooks...

// Hooks de Refs e Efeitos Avançados

- **useLayoutEffect( )** → Similar ao useEffect, mas roda antes da tela ser atualizada (sincronização com o layout/DOM).
- **useImperativeHandle( )** → Permite expor uma API customizada quando se usa forwardRef.
- **useInsertionEffect( )** → Executa antes do browser aplicar estilos CSS (usado por libs de estilização).

// Hooks de Novas Features (React 18+)

- **useId( )** → Gera um ID único estável, útil para inputs, acessibilidade, labels.
- **useSyncExternalStore( )** → Lê o estado de stores externos (ex.: Redux, Zustand) de forma consistente.
- **useOptimistic( )** → (React 18.2+) Permite lidar com atualizações otimistas (ex.: mostrar valor antes da resposta do servidor).
- **useFormStatus( )** → (para React Server Components + Forms) Lida com o status de formulários assíncronos.
- **useFormState( )** → Garante estado controlado em formulários no lado do servidor.

Davi Cândido de Almeida

Um universo rico de Hooks...

// Hooks para Server Components (Next.js 13+)

- use( ) → Hook experimental para await em Promises diretamente no componente.

Obs: (nem todos são usados no cliente, mas fazem parte do ecossistema atual do React)

Resumo de boas praticas

- Use Hooks básicos **(useState, useEffect, useContext)** no dia a dia.
- Crie **Custom Hooks** quando notar repetição de lógica.
- Evite efeitos desnecessários em **useEffect (cada dependência deve ser realmente necessária).**
- Use **useMemo e useCallback** apenas quando houver problema real de performance.

Conclusão

- Os Hooks trouxeram **simplicidade** e **poder** aos componentes funcionais.
- Permitem **gerenciar estado**, **controlar efeitos colaterais** e compartilhar dados sem a complexidade das classes.
- Com eles, a lógica pode ser extraída e reutilizada em **Custom Hooks**, deixando o código mais limpo e organizado.

Links Uteis:

- https://react.dev/
- https://www.reacthaiku.dev/

Davi Cândido de Almeida

Considerações Finais

Lembre-se que isso não é tudo. Muito mais pode ser explorado.
Ainda existe um universo de Hooks avançados para explorar (Refs, Performance, Server Components). No entanto o mais importante é começar pelo básico e ir evoluindo conforme a necessidade.

Acesse os código no GitHub: https://github.com/webtech-network/lab-introducao-react.git

Esse tutorial foi escrito por Davi Cândido – PUC Minas. Compartilhe com colegas desenvolvedores!