



DEV EM
DOBRRO.

Reutilizando lógica com hooks
customizados

Você vai aprender



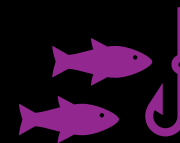
O que são Custom Hooks e
como escrever os seus
próprios



Como reutilizar lógica
entre componentes



Como nomear e estruturar
seus Custom Hooks



Quando e por que extrair
Custom Hooks

Compartilhando lógica entre componentes

React vem com vários Hooks embutidos

- `useState`
- `useContext`
- `useEffect`.

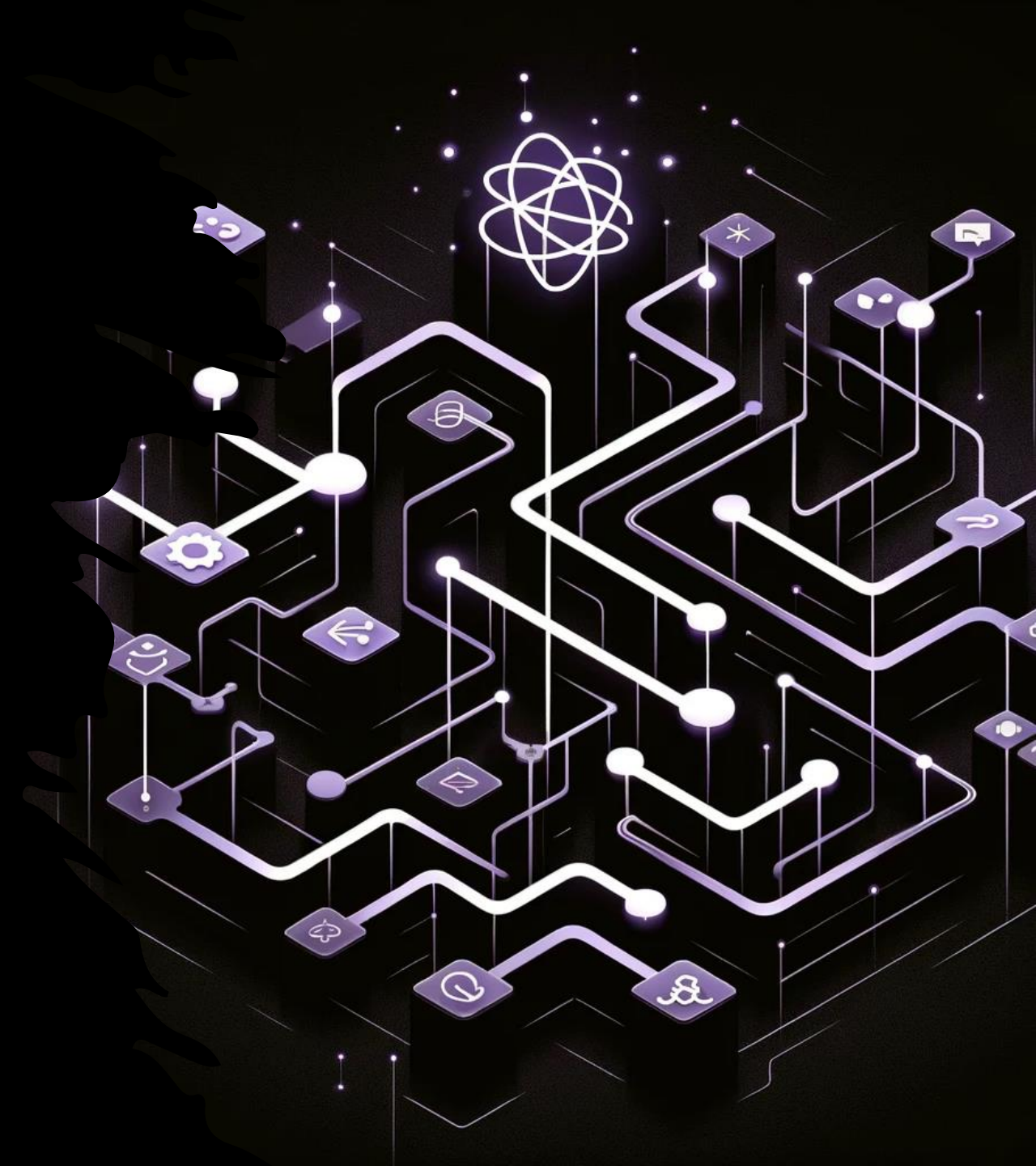
Às vezes, você vai querer que houvesse um Hook para algum propósito mais específico:

- Buscar dados: para acompanhar se o usuário está online
- Para se conectar a uma sala de bate-papo.

Extraindo seu próprio Hook personalizado de um componente

Imagine que você está desenvolvendo um aplicativo que depende fortemente da rede (como a maioria dos aplicativos faz). Você quer alertar o usuário se a conexão de rede deles foi desligada acidentalmente enquanto estavam usando seu aplicativo.

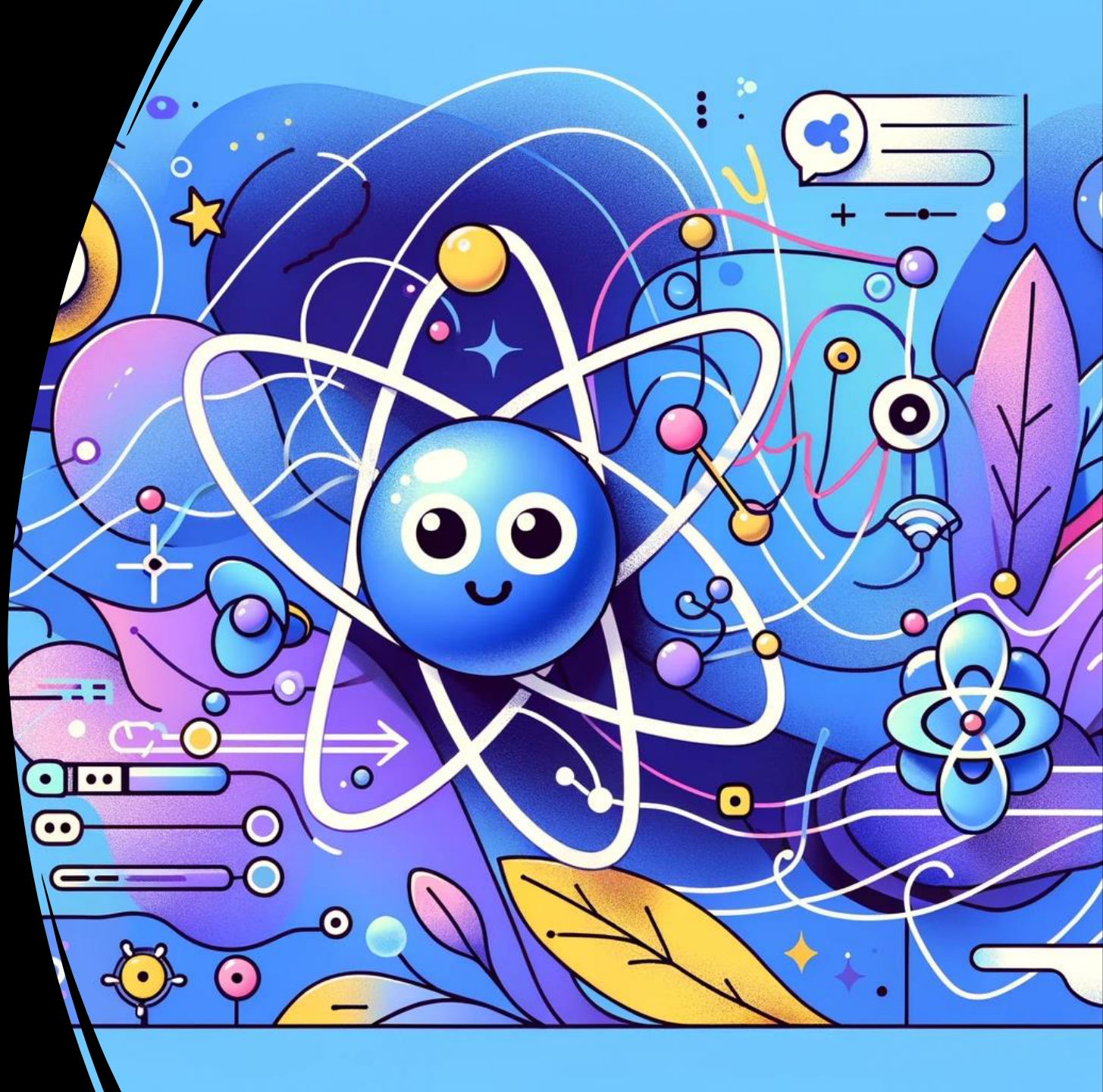
Como você faria isso?

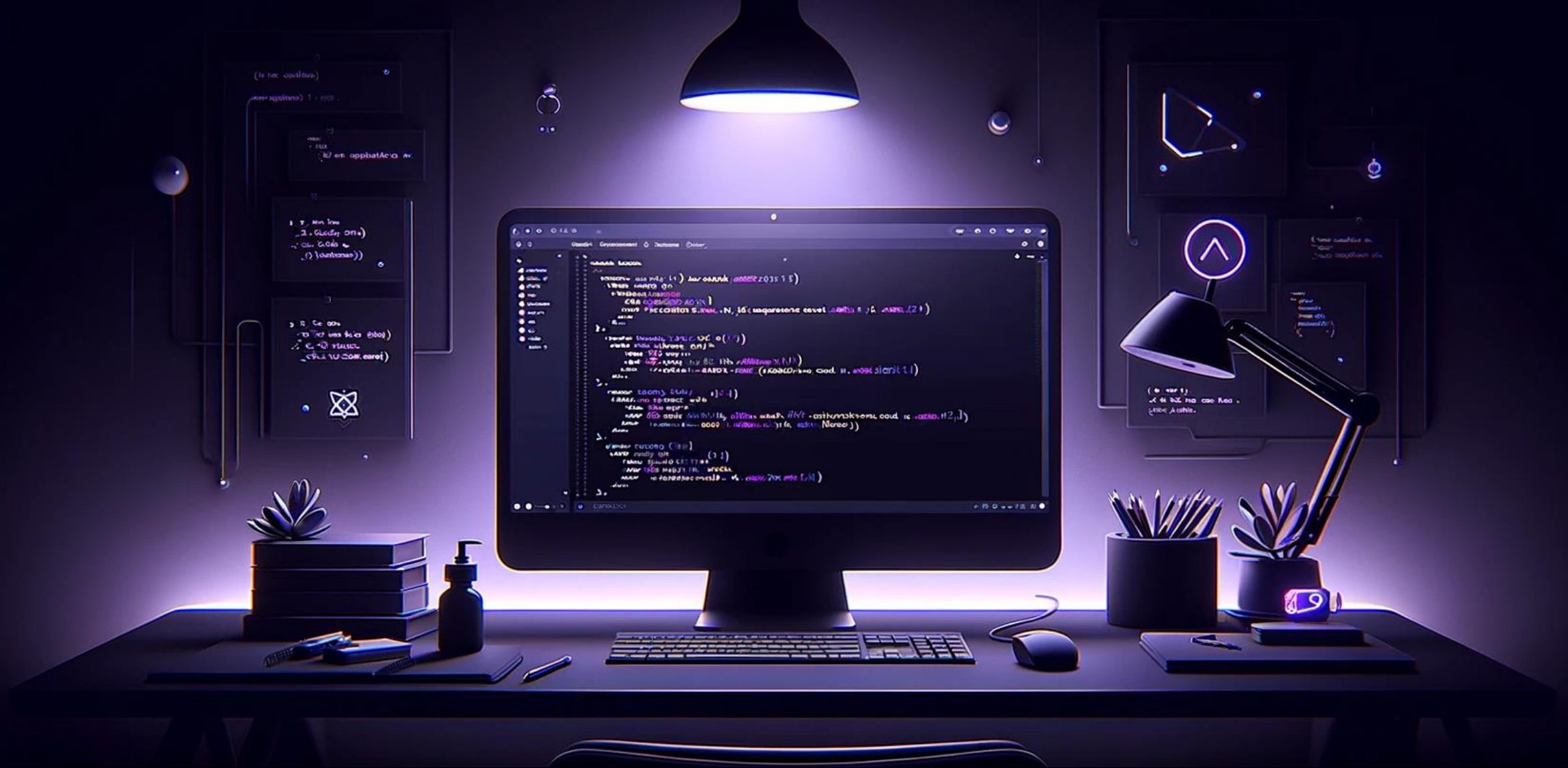


Extraindo seu próprio Hook personalizado de um componente

Você precisará de duas coisas no seu componente

1. Um pedaço de estado que rastreia se a rede está online.
2. Um Effect que se inscreve nos eventos globais online e offline e atualiza esse estado.



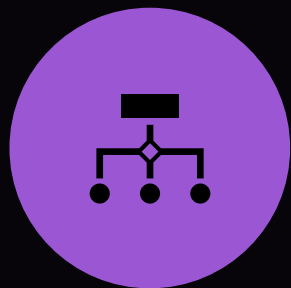


Agora vamos pro VSCODE!

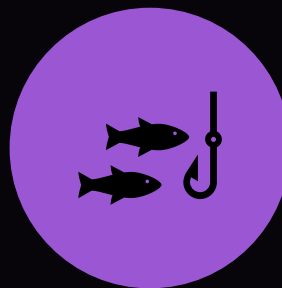
Reutilizando lógica com hooks customizados

Como nomear seus Custom Hooks

Nomes de Hooks sempre começam com "use"



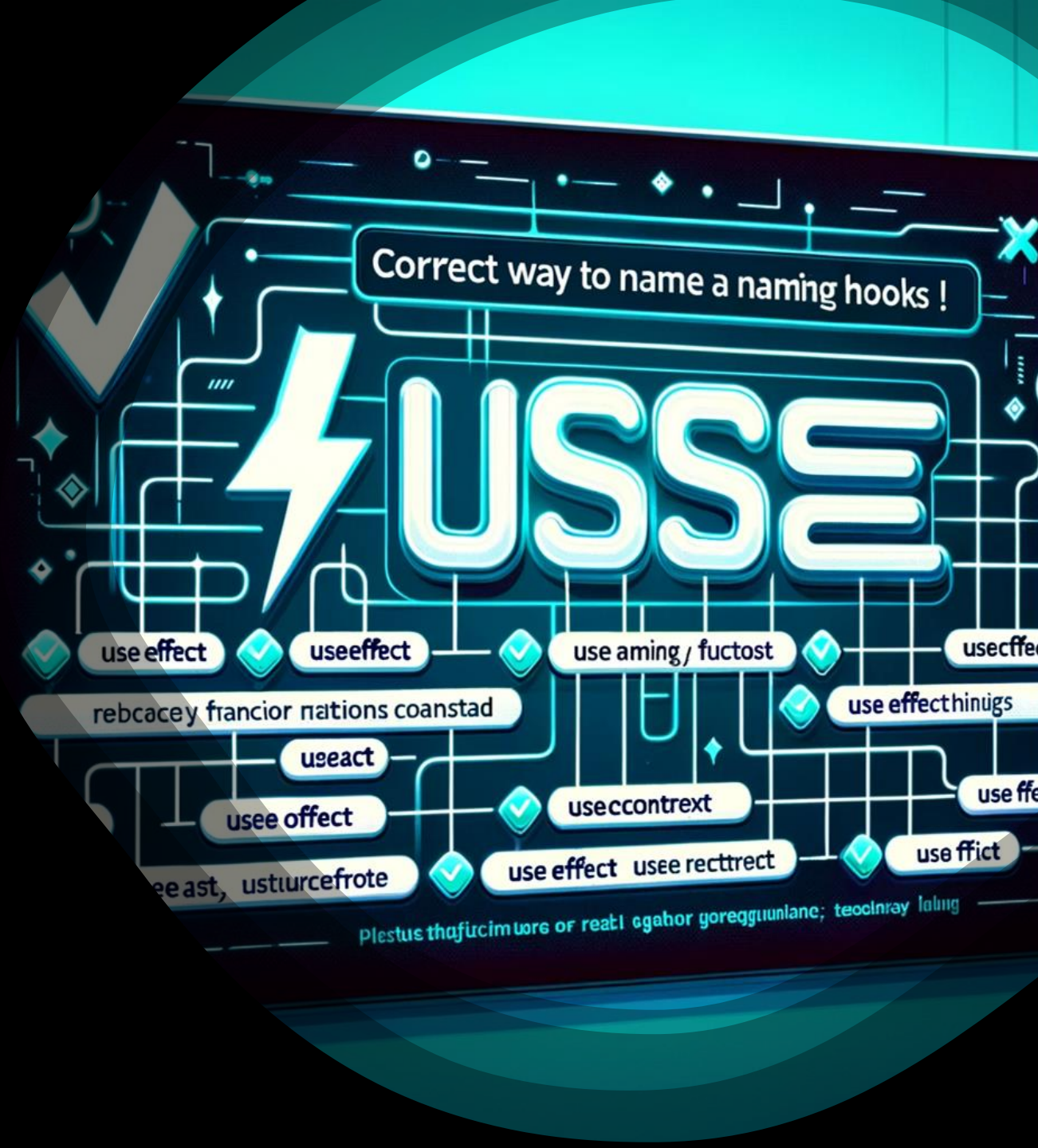
Aplicações React são construídas a partir de componentes. Componentes são construídos a partir de Hooks, sejam eles embutidos ou personalizados.



É provável que você frequentemente use Hooks personalizados criados por outros, mas ocasionalmente você pode escrever um você mesmo!

Nomes de Hooks sempre começam com "use"

- Você deve seguir estas convenções de nomenclatura:
 - Componentes React:
 - Iniciam com letra maiúscula (ex: StatusBar, SaveButton).
 - Retornam JSX ou outros tipos renderizáveis pelo React.
 - Hooks React:
 - Sempre começam com "use" seguido de letra maiúscula (ex: useState, useOnlineStatus).
 - Podem retornar valores arbitrários.



Nomes de Hooks sempre começam com "use"

- Por exemplo, se você ver uma chamada de função **getColor()** dentro do seu componente, você pode ter certeza de que ela não pode possivelmente conter estado do React dentro dela porque seu nome não começa com "use". No entanto, uma chamada de função como **useOnlineStatus()** provavelmente conterá chamadas para outros Hooks por dentro!



Hooks Personalizados permitem compartilhar lógica com estado, não o estado propriamente dito

- No exemplo anterior, quando você ligou e desligou a rede, ambos os componentes foram atualizados juntos. No entanto, é incorreto pensar que uma única variável de estado **isOnline** é compartilhada entre eles.

```
1 // StatusBar
2 const isOnline = useOnlineStatus();
3
4 // SaveButton
5 const isOnline = useOnlineStatus();
6
```

- Ambos StatusBar e SaveButton utilizam useOnlineStatus.
- Ainda são duas variáveis de estado independentes.
- Sincronizados pelo mesmo valor externo (status da rede).

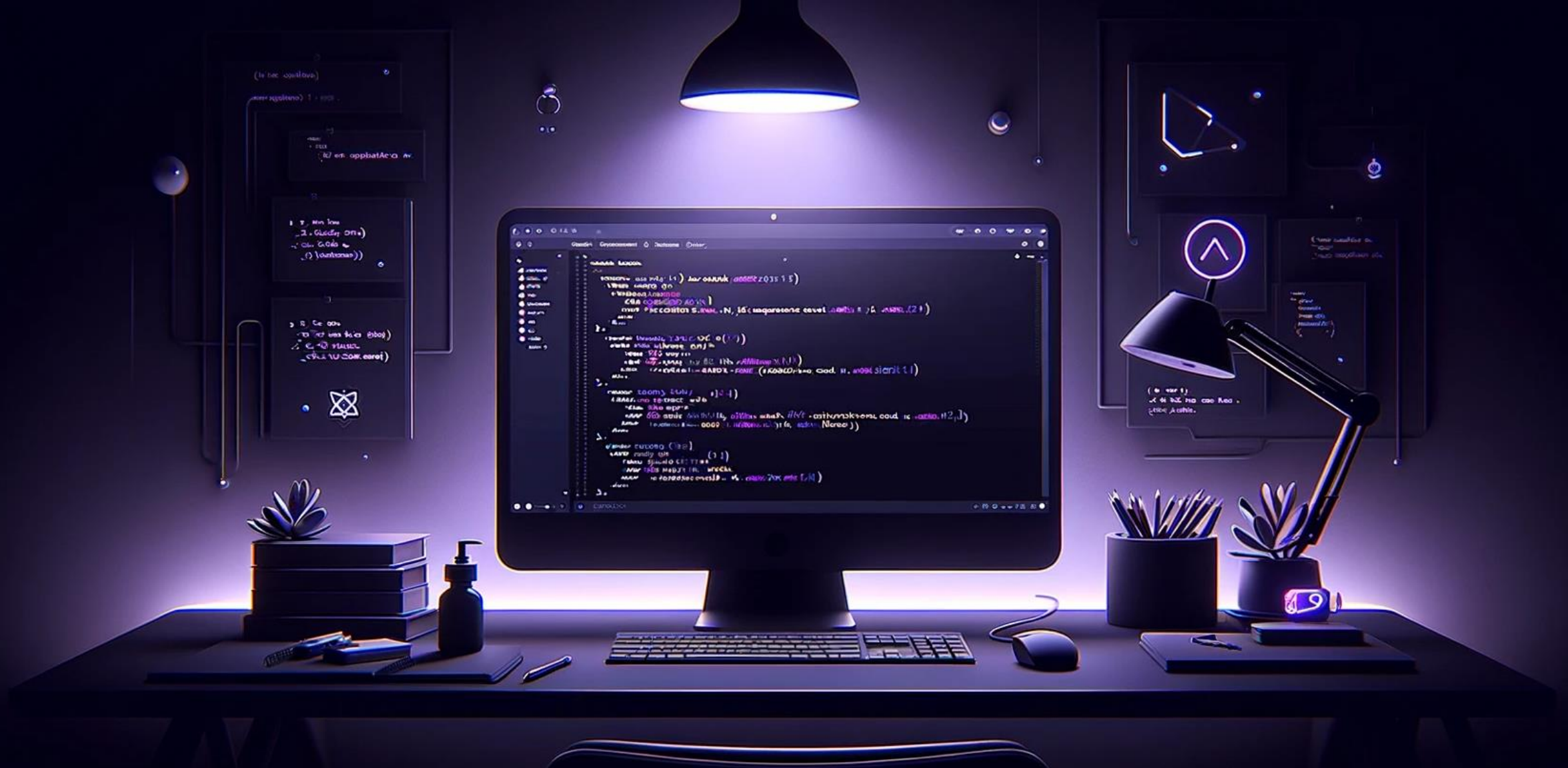
```
1 // StatusBar
2 const [isOnline, setIsOnline] = useState(true);
3 useEffect(() => { /* ... */ }, []);
4
5 // SaveButton
6 const [isOnline, setIsOnline] = useState(true);
7 useEffect(() => { /* ... */ }, []);
```

- Cada componente tem sua própria variável de estado e Effect independentes.

Hooks Personalizados permitem compartilhar lógica com estado, não o estado propriamente dito

- Para ilustrar melhor isso, precisaremos de um exemplo diferente.
- Vamos criar agora um componente Form para realizar os testes!





Agora vamos pro VSCODE!

Reutilizando lógica com hooks customizados

Quando usar Hooks personalizados

Reexecução com Re-renderização do Componente

- Quando você usa um Hook personalizado em um componente do React, o código dentro desse Hook é executado novamente sempre que o componente é re-renderizado. A re-renderização pode acontecer por vários motivos, como mudanças no estado ou nas props do componente.



Hooks Personalizados Devem Ser Puros

- Diz-se que um Hook (ou função) é "puro" quando ele não causa efeitos colaterais externos e seu retorno depende apenas de seus argumentos.
- Isso significa que um Hook personalizado não deve modificar diretamente estados ou variáveis fora de seu escopo.
- Ele deve ser uma função 'pura' no sentido de que, dados os mesmos inputs (props, estado), ele sempre retornará o mesmo output (resultado).



Parte do Corpo do Componente

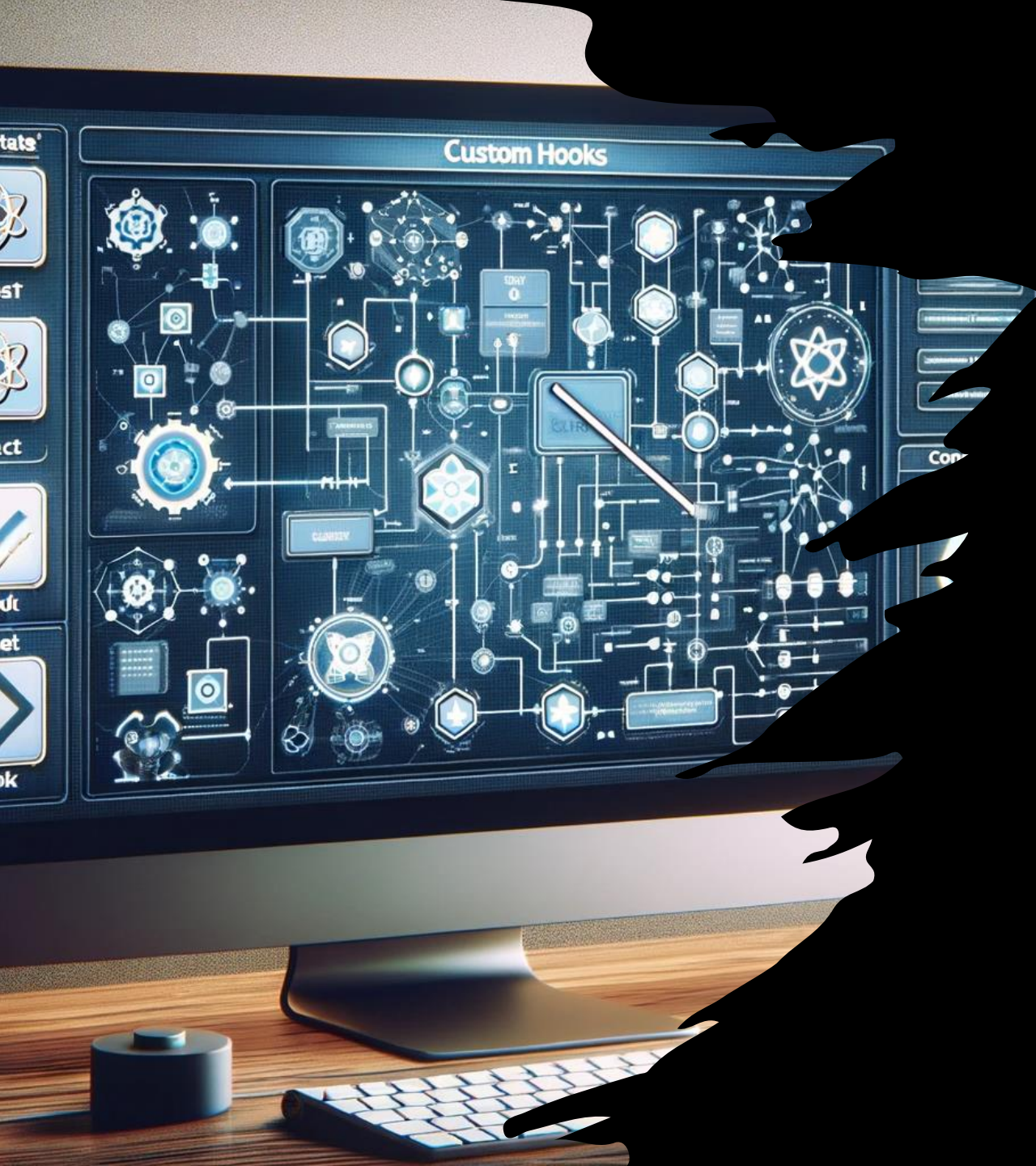
- O código dentro do seu Hook personalizado deve ser considerado como parte integrante do componente em que é usado.
- Assim como você pensa cuidadosamente sobre o código que coloca diretamente no corpo de um componente, você deve tratar o código dentro dos Hooks personalizados com o mesmo nível de consideração.
- Isso é importante para manter a previsibilidade e a manutenibilidade do componente.



Acesso a Props e Estado Atualizados

- Como os Hooks personalizados são reexecutados com cada re-renderização do componente, eles têm acesso ao estado e às props mais recentes.
- Isso significa que, se o estado ou as props do componente mudarem, o Hook personalizado receberá essas mudanças na próxima execução.
- Isso é útil para garantir que o Hook esteja sempre trabalhando com as informações mais atuais do componente.





Quando usar Hooks personalizados

- Evitar para duplicações simples, como um `useState`
- Ideal para Effects que interagem com sistemas externos

Quando usar Hooks personalizados

Antes do Hook Personalizado:
ShippingForm

```
1 function ShippingForm({ country }) {  
2   const [cities, setCities] = useState(null);  
3   useEffect(() => {  
4     let ignore = false;  
5     fetch(`/api/cities?country=${country}`)  
6       .then(response => response.json())  
7       .then(json => {  
8         if (!ignore) setCities(json);  
9       });  
10    return () => { ignore = true; };  
11  }, [country]);  
12  
13  const [city, setCity] = useState(null);  
14  const [areas, setAreas] = useState(null);  
15  useEffect(() => {  
16    if (city) {  
17      let ignore = false;  
18      fetch(`/api/areas?city=${city}`)  
19        .then(response => response.json())  
20        .then(json => {  
21          if (!ignore) setAreas(json);  
22        });  
23      return () => { ignore = true; };  
24    }  
25  })
```

Quando usar Hooks personalizados

- Refatorando com Hook Personalizado: `useData`

```
1 function useData(url) {  
2   const [data, setData] = useState(null);  
3   useEffect(() => {  
4     let ignore = false;  
5     if (url) {  
6       fetch(url)  
7         .then(response => response.json())  
8         .then(json => {  
9           if (!ignore) setData(json);  
10        });  
11     return () => { ignore = true; };  
12   }  
13 }, [url]);  
14 return data;  
15 }  
16
```


Quando usar Hooks personalizados

- ShippingForm Simplificado com useData

```
1 function ShippingForm({ country }) {  
2   const cities = useData(`/api/cities?country=${country}`);  
3   const [city, setCity] = useState(null);  
4   const areas = useData(city ? `/api/areas?city=${city}` : null);  
5   // ...  
6 }  
7
```


Quando usar Hooks personalizados

- Benefícios dos Hooks Personalizados
 - Extrair Hooks personalizados torna o fluxo de dados explícito e previne a adição de dependências desnecessárias ao Effect, que promove melhores práticas e manutenção mais fácil do código.



Hooks Personalizados ajudam a migrar para padrões melhores



Função dos Efeitos:

Atuam onde o React não oferece soluções nativas.

Permitem interações com sistemas externos ou casos de uso exclusivos.



Minimizar a necessidade de Efeitos através de soluções direcionadas.

Aprimorar a API do React para cobrir mais cenários.



Simplificam a gestão de Efeitos e sincronização de dados.

Isolam complexidades, deixando o código principal mais limpo.



Preparação para o Futuro:

A estruturação em Hooks personalizados posiciona o código para aproveitar futuras atualizações do React sem grandes refatorações.

Hooks
Personalizados
ajudam a
migrar para
padrões
melhores

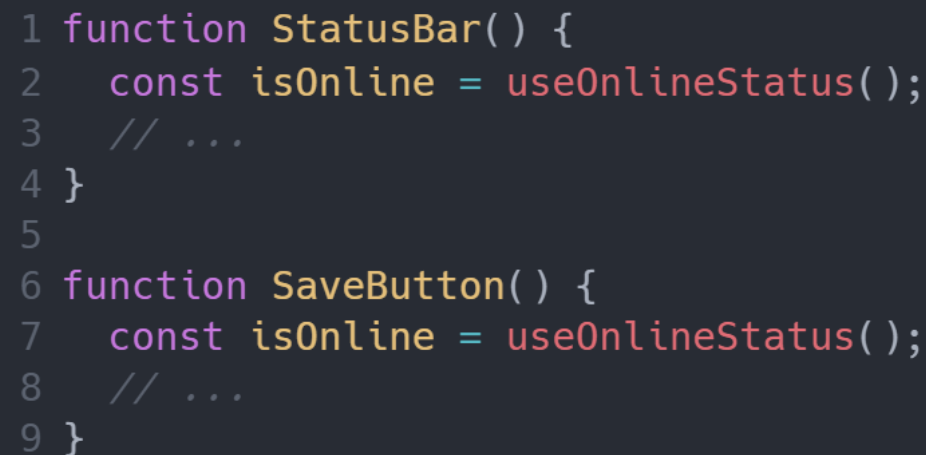
```
1 import { useState, useEffect } from 'react';
2
3 export function useOnlineStatus() {
4   const [isOnline, setIsOnline] = useState(true);
5   useEffect(() => {
6     function handleOnline() {
7       setIsOnline(true);
8     }
9     function handleOffline() {
10      setIsOnline(false);
11    }
12    window.addEventListener('online', handleOnline);
13    window.addEventListener('offline', handleOffline);
14    return () => {
15      window.removeEventListener('online', handleOnline);
16      window.removeEventListener('offline', handleOffline);
17    };
18  }, []);
19  return isOnline;
20 }
21
```


Hooks
Personalizados
ajudam a
migrar para
padrões
melhores

```
1 import { useSyncExternalStore } from 'react';
2
3 function subscribe(callback) {
4   window.addEventListener('online', callback);
5   window.addEventListener('offline', callback);
6   return () => {
7     window.removeEventListener('online', callback);
8     window.removeEventListener('offline', callback);
9   };
10 }
11
12 export function useOnlineStatus() {
13   return useSyncExternalStore(
14     subscribe,
15     () => navigator.onLine, // How to get the value on the client
16     () => true // How to get the value on the server
17   );
18 }
```

Hooks Personalizados ajudam a migrar para padrões melhores

- Perceba como você não precisou mudar nenhum dos componentes para fazer essa migração



```
1 function StatusBar() {  
2   const isOnline = useOnlineStatus();  
3   // ...  
4 }  
5  
6 function SaveButton() {  
7   const isOnline = useOnlineStatus();  
8   // ...  
9 }
```


Hooks Personalizados ajudam a migrar para padrões melhores

- Isso é mais uma razão pela qual encapsular Effects em Hooks personalizados é frequentemente benéfico:
 - Você torna o fluxo de dados de e para seus Effects muito explícito.
 - Você permite que seus componentes se concentrem na intenção em vez de na exata implementação dos seus Effects.
 - Quando o React adiciona novos recursos, você pode remover aqueles Effects sem alterar nenhum dos seus componentes.

CUSTOM HOOKS

Hooks Personalizados ajudam a migrar para padrões melhores



Consolidação de Padrões: Assim como um sistema de design padroniza elementos de UI, Hooks personalizados ajudam a unificar e reutilizar lógicas comuns entre componentes.



Foco na Intenção: Centraliza a complexidade dos Effects, permitindo que componentes se concentrem na funcionalidade, não na implementação.



Redução de Complexidade: Diminui a necessidade de implementar diretamente Effects, simplificando a manutenção.

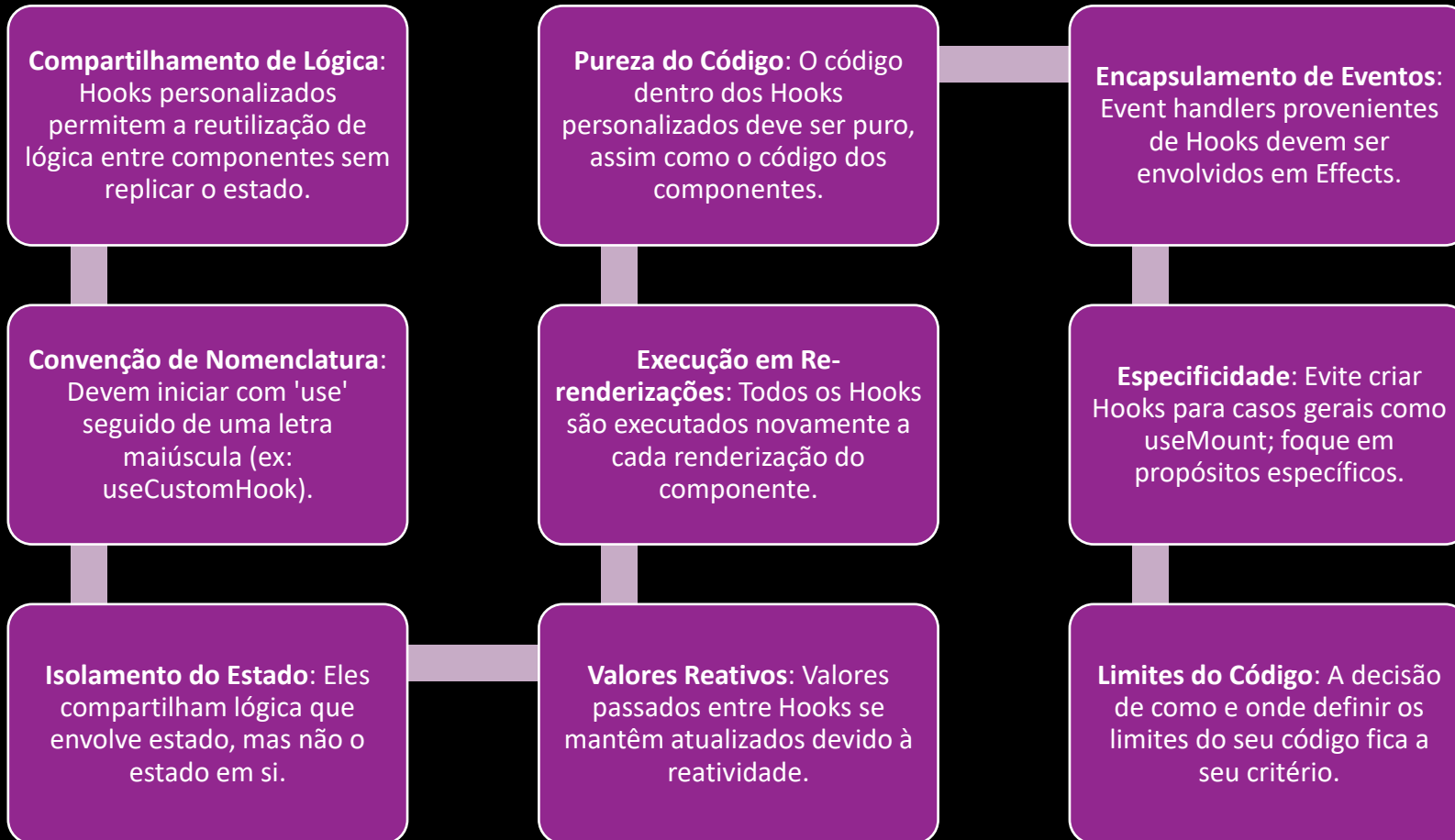


Comunidade Ativa: Aproveite Hooks bem elaborados e mantidos pela comunidade React para enriquecer seus projetos.

Existem várias maneiras de fazer isso

- Animação Fade-in Personalizada: Utilizar o API `requestAnimationFrame` do navegador para criar um efeito de desvanecimento.
- Configuração do Loop de Animação: Iniciar com um Effect que estabelece um ciclo de animação, alterando a opacidade do nó DOM referenciado até que alcance 1.
- Código Inicial Exemplificado: A implementação começa definindo o processo de animação, ajustando-se gradualmente até atingir a visibilidade total.

Conclusão



Referências

- Documentação Oficial React: <https://react.dev/learn/reusing-logic-with-custom-hooks#when-to-use-custom-hooks>