

# Custom Hooks

Labenu\_



# O que vamos ver hoje?

- Desestruturação
- O que são Custom Hooks
- Custom hook de requisição
- Tratamento de erros



# Desestruturação

Labenu\_



# Desestruturação

- Sintaxe mais simples para criar variáveis a partir de objetos e arrays
- Permite atribuir propriedades (em caso de objetos) ou posições (em caso de arrays) de forma mais **direta**
- Muito usado com funções que retornam mais de um valor, como o hook de useState!



# Desestruturação - Objetos

- Objeto pessoa com propriedades **nome** e **email**
- Antes: acessar propriedades e atribuir à variáveis
- Nova: se quisermos variáveis com os **mesmos nomes** das propriedades, é possível desestruturar

```
1 // Objeto
2 const pessoa = {
3   nome: "Alice",
4   email: "alice@lbn.com"
5 }
6
7 // Sintaxe tradicional
8 const nome = pessoa.nome
9 const email = pessoa.email
10
11 // Desestruturacao
12 const {nome, email} = pessoa
```



# Desestruturação - Array

- Array pessoas com string de **nome** da pessoa
- Antes: acessar as duas posições e atribuí-las a variáveis
- Nova: se soubermos o que **cada posição tem** e quisermos atribuir a uma variável, é possível desestruturar

```
1 // Array
2 const pessoas = ["Alice", "Bob"]
3
4 // Sintaxe tradicional
5 const alice = pessoas[0]
6 const bob = pessoas[1]
7
8 // Desestruturacao
9 const [alice, bob] = pessoas
```





# Exercício 1

- Crie uma função que retorna um objeto com os dados de uma pessoa: nome, idade e profissão
- Após executar a função, guarde seus dados em variáveis isoladas utilizando desestruturação





## Exercício 2

- Crie uma função que retorna um array com uma lista strings que representa os itens de uma lista de compras
- Após executar a função, guarde seus dados em variáveis isoladas utilizando desestruturação





# Pausa para relaxar 🤪

5 min

- Desestruturação é uma forma de guardar valores de arrays e objetos em variáveis



```
1 // Objeto
2 const pessoa = {
3   nome: "Alice",
4   email: "alice@lbn.com"
5 }
6
7 // Sintaxe tradicional
8 const nome = pessoa.nome
9 const email = pessoa.email
10
11 // Desestruturação
12 const {nome, email} = pessoa
```

```
1 // Array
2 const pessoas = ["Alice", "Bob"]
3
4 // Sintaxe tradicional
5 const alice = pessoas[0]
6 const bob = pessoas[1]
7
8 // Desestruturação
9 const [alice, bob] = pessoas
```



# Motivação dos Custom Hooks

Labenu\_



# Problema 🤯

## Reutilização de lógicas de estado e lifecycle

- Em componentes de classe, não é possível **reutilizar** lógica de estado e lifecycle naturalmente
- Ou seja: componentes que têm comportamento semelhantes, mas interfaces diferentes, possuem **código repetido**



```

1 import React from 'react'
2 import axios from 'axios'
3
4 export class TelaProdutos extends React.Component {
5   state = {
6     products: []
7   }
8
9   componentDidMount() {
10     axios.get('https://minha-api.com/products')
11       .then((response) => {
12         this.setState({products: response.data})
13       })
14   }
15
16   render() {
17     return (
18       <div>
19         {this.state.products.map((product) => {
20           return <p>{product.name}</p>
21         })}
22       </div>
23     )
24   }
25 }

```

```

1 import React from 'react'
2 import axios from 'axios'
3
4 export class TelaUsuario extends React.Component {
5   state = {
6     user: {}
7   }
8
9   componentDidMount() {
10     axios.get('https://minha-api.com/user')
11       .then((response) => {
12         this.setState({user: response.data})
13       })
14   }
15
16   render() {
17     return (
18       <div>
19         <p>{this.state.user.name}</p>
20       </div>
21     )
22   }
23 }

```

```

1 import React from 'react'
2 import axios from 'axios'
3
4 export class TelaProdutos extends React.Component {
5   state = {
6     products: []
7   }
8
9   componentDidMount() {
10     axios.get('https://minha-api.com/products')
11       .then((response) => {
12         this.setState({products: response.data})
13       })
14   }
15
16   render() {
17     return (
18       <div>
19         {this.state.products.map((product) => {
20           return <p>{product.name}</p>
21         })}
22       </div>
23     )
24   }
25 }

```

```

1 import React from 'react'
2 import axios from 'axios'
3
4 export class TelaUsuario extends React.Component {
5   state = {
6     user: {}
7   }
8
9   componentDidMount() {
10     axios.get('https://minha-api.com/user')
11       .then((response) => {
12         this.setState({user: response.data})
13       })
14   }
15
16   render() {
17     return (
18       <div>
19         <p>{this.state.user.name}</p>
20       </div>
21     )
22   }
23 }

```

# Reutilização de Lógica 💡

- Os dois componentes têm coisas em comum:
  - Um **estado** com uma propriedade
  - Um `componentDidMount()` com uma **requisição**
  - **Estado atualizado** com resposta da requisição
- Hooks resolvem esse problema ❤️



# Exemplo: requisição

Partindo de um exemplo  
análogo usando Hooks

```
1 import React, {useState} from 'react'
2 import axios from 'axios'
3
4 export function TelaProdutos() {
5   const [products, setProducts] = useState([])
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/products')
9       .then((response) => {
10         setProducts(response.data)
11       })
12   }, [])
13
14   return (
15     <div>
16       {products.map((product) => {
17         return <p>{product.name}</p>
18       })}
19     </div>
20   )
21 }
```



# Exemplo: requisição

Partindo de um exemplo  
análogo usando Hooks

**A lógica de estado e lifecycle  
é a mesma dos componentes  
de classe, escrita de outro  
jeito**

```
1 import React, {useState} from 'react'
2 import axios from 'axios'
3
4 export function TelaProdutos() {
5   const [products, setProducts] = useState([])
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/products')
9       .then((response) => {
10         setProducts(response.data)
11       })
12   }, [])
13
14   return (
15     <div>
16       {products.map((product) => {
17         return <p>{product.name}</p>
18       })}
19     </div>
20   )
21 }
```





# Exemplo: requisição

Partindo de um exemplo análogo usando Hooks

A lógica de estado e lifecycle é a mesma dos componentes de classe, escrita de outro jeito

**Mas, como estamos em uma função, é possível extrair essa lógica para outra função**

```
1 import React, {useState} from 'react'
2 import axios from 'axios'
3
4 export function TelaProdutos() {
5   const [products, setProducts] = useState([])
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/products')
9       .then((response) => {
10         setProducts(response.data)
11       })
12   }, [])
13
14   return (
15     <div>
16       {products.map((product) => {
17         return <p>{product.name}</p>
18       })}
19     </div>
20   )
21 }
```



# Extraindo lógica

Uma função separada é criada,  
chamada useProducts

```
1 import {useState, useEffect} from 'react'
2 import axios from 'axios'
3
4 export function useProducts() {
5   const [products, setProducts] = useState([])
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/products')
9       .then((response) => {
10         setProducts(response.data)
11       })
12   }, [])
13
14   return products
15 }
```



# Extraindo lógica

Uma função separada é criada, chamada useProducts

**Ela contém a mesma lógica que o componente**

```
1 import {useState, useEffect} from 'react'
2 import axios from 'axios'
3
4 export function useProducts() {
5   const [products, setProducts] = useState([])
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/products')
9       .then((response) => {
10         setProducts(response.data)
11       })
12   }, [])
13
14   return products
15 }
```



# Extraindo lógica

Uma função separada é criada, chamada useProducts

Ela contém a mesma lógica que o componente

**E retorna o array de products, pois de todo esse código, é o que é realmente usado para a renderização**

```
1 import {useState, useEffect} from 'react'
2 import axios from 'axios'
3
4 export function useProducts() {
5   const [products, setProducts] = useState([])
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/products')
9       .then((response) => {
10         setProducts(response.data)
11       })
12   }, [])
13
14   return products
15 }
```



# Usando a lógica extraída

## Voltando para o componente

```
1 import React, {useState} from 'react'
2 import axios from 'axios'
3 import {useProducts} from './useProducts'
4
5 export function TelaProdutos() {
6   const products = useProducts()
7
8   return (
9     <div>
10       {products.map((product) => {
11         return <p>{product.name}</p>
12       })}
13     </div>
14   )
15 }
```



# Usando a lógica extraída

Voltando para o componente

**Substituímos toda aquela  
lógica pela chamada da  
função useProducts**

```
1 import React, {useState} from 'react'
2 import axios from 'axios'
3 import {useProducts} from './useProducts'
4
5 export function TelaProdutos() {
6   const products = useProducts()
7
8   return (
9     <div>
10       {products.map((product) => {
11         return <p>{product.name}</p>
12       })}
13     </div>
14   )
15 }
```



# Usando a lógica extraída

Voltando para o componente

Substituímos toda aquela lógica pela chamada da função `useProducts`

**E, se ela estiver em outro arquivo, devemos importá-la da mesma forma que fazemos com componentes**

```
1 import React, {useState} from 'react'
2 import axios from 'axios'
3 import {useProducts} from './useProducts'
4
5 export function TelaProdutos() {
6   const products = useProducts()
7
8   return (
9     <div>
10       {products.map((product) => {
11         return <p>{product.name}</p>
12       })}
13     </div>
14   )
15 }
```



# Repetindo...

É possível fazer a mesma com o componente de TelaUsuario

```
1 import React, {useState, useEffect} from 'react'
2 import axios from 'axios'
3
4 export function TelaUsuario() {
5   const [user, setUser] = useState([])
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/user')
9       .then((response) => {
10         setUser(response.data)
11       })
12   }, [])
13
14   return (
15     <div>
16       <p>{user.name}</p>
17     </div>
18   )
19 }
```





# Repetindo...

É possível fazer a mesma com o componente de TelaUsuario

**Extraímos a lógica para uma função useUser**

```
1 import {useState, useEffect} from 'react'
2 import axios from 'axios'
3
4 export function useUser() {
5   const [user, setUser] = useState({})
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/user')
9       .then((response) => {
10         setUser(response.data)
11       })
12   }, [])
13
14   return user
15 }
```



# Repetindo...

É possível fazer a mesma com o componente de TelaUsuario

Extraímos a lógica para uma função useUser

**E substituímos essa lógica no componente pela no função**

```
1 import React from 'react'
2 import {useUser} from './useUser'
3
4 export function TelaUsuario() {
5   const user = useUser()
6
7   return (
8     <div>
9       <p>{user.name}</p>
10    </div>
11  )
12 }
```



# Criamos duas funções...

```
1 import {useState, useEffect} from 'react'
2 import axios from 'axios'
3
4 export function useProducts() {
5   const [products, setProducts] = useState([])
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/products')
9       .then((response) => {
10         setProducts(response.data)
11       })
12   }, [])
13
14   return products
15 }
```

```
1 import {useState, useEffect} from 'react'
2 import axios from 'axios'
3
4 export function useUser() {
5   const [user, setUser] = useState({})
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/user')
9       .then((response) => {
10         setUser(response.data)
11       })
12   }, [])
13
14   return user
15 }
```



# ...muito parecidas

```
1 import {useState, useEffect} from 'react'
2 import axios from 'axios'
3
4 export function useProducts() {
5   const [products, setProducts] = useState([])
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/products')
9       .then((response) => {
10         setProducts(response.data)
11       })
12   }, [])
13
14   return products
15 }
```

```
1 import {useState, useEffect} from 'react'
2 import axios from 'axios'
3
4 export function useUser() {
5   const [user, setUser] = useState({})
6
7   useEffect(() => {
8     axios.get('https://minha-api.com/user')
9       .then((response) => {
10         setUser(response.data)
11       })
12   }, [])
13
14   return user
15 }
```



Código comum



Código variável



Nomes arbitrários



# Função reutilizável

Podemos unir as duas funções em uma só, e chamá-las dos dois componentes, apenas variando os parâmetros

```
1 import { useEffect, useState } from "react"
2 import axios from "axios"
3
4 export function useRequestData(url) {
5   const [data, setData] = useState(undefined)
6
7   useEffect(() => {
8     axios.get(url)
9       .then((res) => setData(res.data))
10      .catch((err) => console.log(err.response.data))
11   }, [url])
12
13   return data
14 }
```



# Função reutilizável

Podemos unir as duas funções em uma só, e chamá-las dos dois componentes, apenas variando os parâmetros

● **Código comum é mantido**

```
1 import { useEffect, useState } from "react"
2 import axios from "axios"
3
4 export function useRequestData(url) {
5   const [data, setData] = useState(undefined)
6
7   useEffect(() => {
8     axios.get(url)
9       .then((res) => setData(res.data))
10      .catch((err) => console.log(err.response.data))
11   }, [url])
12
13   return data
14 }
```



# Função reutilizável

Podemos unir as duas funções em uma só, e chamá-las dos dois componentes, apenas variando os parâmetros

- Código comum é mantido
- **Código variável é passado por parâmetro da função**

```
1 import { useEffect, useState } from "react"
2 import axios from "axios"
3
4 export function useRequestData(url) {
5   const [data, setData] = useState(undefined)
6
7   useEffect(() => {
8     axios.get(url)
9       .then((res) => setData(res.data))
10      .catch((err) => console.log(err.response.data))
11   }, [url])
12
13   return data
14 }
```



# Função reutilizável

Podemos unir as duas funções em uma só, e chamá-las dos dois componentes, apenas variando os parâmetros

- Código comum é mantido
- Código variável é passado por parâmetro da função
- Código arbitrário é deixado mais genérico

```
1 import { useEffect, useState } from "react"
2 import axios from "axios"
3
4 export function useRequestData(url) {
5   const [data, setData] = useState(undefined)
6
7   useEffect(() => {
8     axios.get(url)
9       .then((res) => setData(res.data))
10      .catch((err) => console.log(err.response.data))
11   }, [url])
12
13   return data
14 }
```





# E agora, usamos nos componentes

```
1 import React from "react"
2 import { BASE_URL } from "../constants/urls"
3 import { useRequestData } from "../hooks/useRequestData"
4
5 export function TelaProdutos() {
6   const products = useRequestData(`${BASE_URL}/products`)
7
8   return(
9     <div>
10       {products.map((prod) => {
11         return <p>{product.name}</p>
12       })}
13     </div>
14   )
15 }
```

```
1 import React from "react"
2 import { BASE_URL } from "../constants/urls"
3 import { useRequestData } from "../hooks/useRequestData"
4
5 export function TelaUsuario() {
6   const user = useRequestData(`${BASE_URL}/user`)
7
8   return(
9     <div>
10       <p>{user.name}</p>
11     </div>
12   )
13 }
```



# Custom Hooks

Labenu\_



# Custom Hooks

- Criamos nosso 1º Custom Hook, **useRequestData!**
- Um Custom Hook é uma função que chama os hooks básicos (useState, useEffect...)
- O nome deve começar com **use...** para que fique claro que é um hook e que as regras dos hooks devem ser seguidas

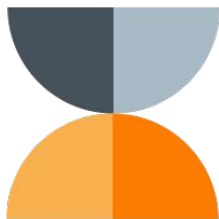


# Custom Hooks

- Deve ser usado da mesma forma que os hooks tradicionais, seguindo as mesmas regras
- Quando criar um custom hook:
  - Código/lógica repetida
  - Componente muito complexo ou longo
  - Mesmos critérios usados para criar componentes ou funções

**Vamos ver na prática!** 





# Pausa para relaxar 🧘

10 min

- Custom Hooks são funções que usam hooks básicos
- Devem ser criados quando lógicas de estado e side-effects se repetem
- Nome deve começar com **use...** para que seja facilmente identificado como um hook e para que as regras possam ser seguidas



# Loading e Tratamento de Erros

Labenu\_



# Loadings e Erros

- Até agora, nós estávamos fazendo nossos loadings com renderização condicional
- Porém, quando acontece um **erro** ou quando, por exemplo, nos é retornada **uma lista vazia**, nosso programa não sabe dizer a diferença e continua mostrando um **loading** na tela para o usuário!

Vamos ver na prática! 



# Loadings e Erros

- O loading deve começar quando a **requisição** começa e terminar quando ela termina
- Podemos fazer esse controle criando um estado **isLoading**

```
1 export const useRequestData = (url) => {  
2   const [data, setData] = useState(undefined)  
3   const [isLoading, setIsLoading] = useState(false)  
4  
5   useEffect(() => {  
6     setIsLoading(true)  
7     axios  
8       .get(url)  
9       .then((res) => {  
10        setIsLoading(false)  
11        setData(res.data)  
12      })  
13      .catch((err) => {  
14        setIsLoading(false)  
15        console.log(err.response)  
16      })  
17    }, [url])  
18  
19    return [data, isLoading]  
20  }
```





# Loadings e Erros

- Para o caso do erro, basta colocá-lo também em um **estado** que dirá se aconteceu um erro na requisição!
- Ao fim, retornamos ele também

```
1 export const useRequestData = (url) => {
2   const [data, setData] = useState(undefined)
3   const [isLoading, setIsLoading] = useState(false)
4   const [error, setError] = useState("")
5
6   useEffect(() => {
7     setIsLoading(true);
8     axios
9       .get(url)
10      .then((res) => {
11        setIsLoading(false)
12        setData(res.data)
13      })
14      .catch((err) => {
15        setIsLoading(false)
16        setError(err)
17      })
18    }, [url])
19
20   return [data, isLoading, error]
21 }
```



# Loadings e Erros

- Para acessar todos esses dados, utilizamos a sintaxe da desestruturação
- Teremos acesso ao **loading**, aos **dados** (se a requisição der certo) e ao **erro** (se a requisição der errado)

```
1 import React from "react"
2 import { useRequestData } from "../hooks/useRequestData"
3
4 export default function App() {
5
6   const [products, isLoading, error] = useRequestData(
7     "https://minha-api.com/products"
8   )
9
10 }
```



# Loadings e Erros

- Na tela, há **4 situações possíveis**:
  - A requisição ainda não terminou ⇒ **Carregando**
  - A requisição terminou e deu um **erro**
  - A requisição terminou e deu **sucesso**:
    - Os dados retornados estão completos
    - Os dados retornados estão vazios (Ex: carrinho vazio, nenhum match correspondido)



# useRequestData

```
1 export const useRequestData = (url) => {
2   const [data, setData] = useState(undefined)
3   const [isLoading, setIsLoading] = useState(false)
4   const [error, setError] = useState("")
5
6   useEffect(() => {
7     setIsLoading(true);
8     axios
9       .get(url)
10      .then((res) => {
11        setIsLoading(false)
12        setData(res.data)
13      })
14      .catch((err) => {
15        setIsLoading(false)
16        setError(err)
17      })
18   }, [url])
19
20   return [data, isLoading, error]
21 }
```

```
1 import React from "react"
2 import { useRequestData } from "../hooks/useRequestData"
3
4 export default function App() {
5   const [products, isLoading, error] = useRequestData(
6     "https://minha-api.com/products"
7   )
8
9   const productsList =
10     products && products.map((prod) => {
11       return <li key={prod.id}>{prod.name}</li>
12     })
13
14   return (
15     <div>
16       {isLoading && <p>Carregando...</p>}
17       {!isLoading && error && <p>0correu um erro</p>}
18       {!isLoading && products && products.length > 0 && productsList}
19       {!isLoading && products && products.length === 0 && (
20         <p>Não há nenhum produto</p>
21       )}
22     </div>
23   )
24 }
```





## Exercício 3

- Utilizando o hook **useRequestData**, faça uma integração com a API de Harry Potter para mostrar a lista de personagens na tela
- Link da API: <https://hp-api.herokuapp.com/>



# Resumo

Labenu\_



# Resumo

- Custom Hooks são funções compostas por outros hooks básicos (useState, useEffect)
- Servem para isolar lógicas de estado e ciclo de vida
- Seguem as regras dos hooks que vimos e seus nomes devem obrigatoriamente começar com **use**



# Resumo



- Desestruturação é uma maneira de acessar e guardar valores que estão dentro de arrays e objetos

```
1 // Objeto
2 const pessoa = {
3   nome: "Alice",
4   email: "alice@lbn.com"
5 }
6
7 // Sintaxe tradicional
8 const nome = pessoa.nome
9 const email = pessoa.email
10
11 // Desestruturação
12 const {nome, email} = pessoa
```

```
1 // Array
2 const pessoas = ["Alice", "Bob"]
3
4 // Sintaxe tradicional
5 const alice = pessoas[0]
6 const bob = pessoas[1]
7
8 // Desestruturação
9 const [alice, bob] = pessoas
```





# Resumo



```
1 export const useRequestData = (url) => {
2   const [data, setData] = useState(undefined)
3   const [isLoading, setIsLoading] = useState(false)
4   const [error, setError] = useState("")
5
6   useEffect(() => {
7     setIsLoading(true);
8     axios
9       .get(url)
10      .then((res) => {
11        setIsLoading(false)
12        setData(res.data)
13      })
14      .catch((err) => {
15        setIsLoading(false)
16        setError(err)
17      })
18    }, [url])
19
20   return [data, isLoading, error]
21 }
```



# Dúvidas? 🧐

Labenu\_





Obrigado(a)!