

Componentes Funcionais e props

Labenu_



Componentes (primeiro pilar do React)

Labenu_



Componentes

- Componentes são blocos de código reutilizáveis
- Agora esses blocos representam um pedaço de **interface**, “gerando” o HTML que aparece na página

Entrada
(**Props**)



Saída
(**JSX**)



O que deve ser um componente?

- **Não existe uma regra** de quando componentizar uma parte da tela
- Devemos considerar **criar** um componente quando:
 - **Layout repetido**
 - Código muito grande e/ou confuso
 - Queremos dar um nome significativo à uma parte da interface



Como assim repetido?

Gramma verde:
Página que é a
base para os
componentes.
Exemplo:
homepage

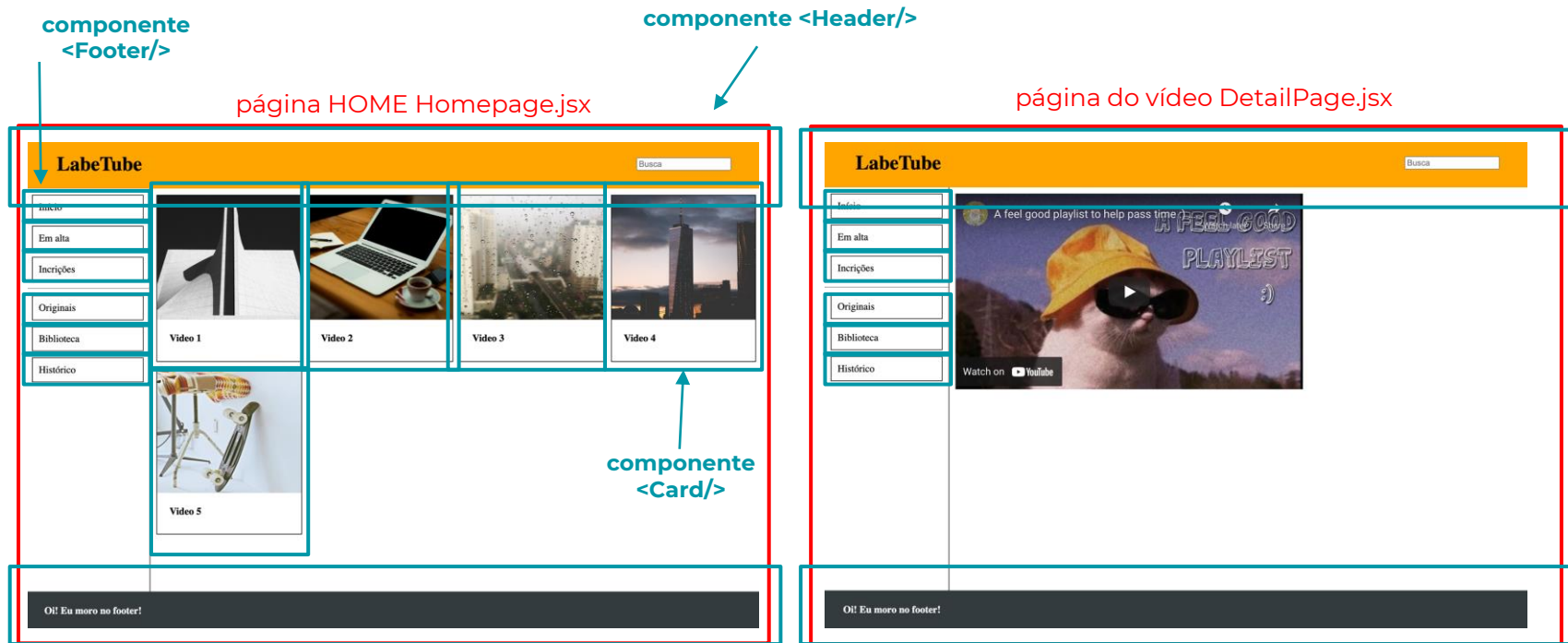


Componentes reutilizáveis:
botões, cards, headers,
footers...

Analogia com Lego

Os blocos são
componentes que se
repetem. São blocos
prontos de código que
podem repetir na
aplicação.
Ex: caixa de
comentário, botões...





● páginas não se repetem, são a base dos componentes

● componentes que se repetem em várias páginas

componente <Footer/>



Componentes

- Na prática, componentes em React são **funções** (por enquanto) com algumas **regras** específicas:
 - Primeira letra do nome maiúscula
 - Deve retornar um JSX (com um único pai)

```
1 function MeuComponente() {  
2   return <div>  
3     <h1>Meu primeiro componente!</h1>  
4     <p>Este é o meu primeiro componente React</p>  
5   </div>  
6 }
```



Componentes

- Na prática, componentes em React são **funções** (por enquanto) com algumas **regras** específicas:
 - **Primeira letra do nome maiúscula**
 - Deve retornar um JSX (com um único pai)

```
1 function MeuComponente() {  
2   return <div>  
3     <h1>Meu primeiro componente!</h1>  
4     <p>Este é o meu primeiro componente React</p>  
5   </div>  
6 }
```



Componentes

- Na prática, componentes em React são **funções** (por enquanto) com algumas **regras** específicas:
 - Primeira letra do nome maiúscula
 - **Deve retornar um JSX (com um único pai)**

```
1 function MeuComponente() {  
2   return <div>  
3     <h1>Meu primeiro componente!</h1>  
4     <p>Este é o meu primeiro componente React</p>  
5   </div>  
6 }
```

Vamos ver na prática! 



Usando um Componente

- Para colocar o componente na tela, chamamos ele em um **componente pai**, dentro do nosso JSX
- Quando colocamos um componente A dentro de um componente B, falamos que o componente A é **filho** do componente B



Usando um Componente

- Lembrando que o componente **App.js**, criado por padrão quando criamos um app React, **é o pai de todos os outros componentes**

`<MeuComponente/>`
é filho do `<App/>`



Usando um Componente

- Para **chamar** o componente, usamos uma sintaxe semelhante à do **HTML**
- Nome deve ser mantido, **com a letra maiúscula**

App.js

```
1 function App() {  
2   return <div>  
3     <MeuComponente></MeuComponente>  
4   </div>  
5 }
```

MeuComponente.js

```
1 function MeuComponente() {  
2   return <div>  
3     <h1>Meu Primeiro Componente</h1>  
4     <p>Este é o meu primeiro componente React</p>  
5   </div>  
6 }
```



Self-Closing Tag

- Quando um componente não possui nada entre a abertura e o fechamento de sua tag, é preferível que se use a sintaxe **self-closing**

App.js

```
1 function App() {  
2   return <div>  
3     <MeuComponente />  
4   </div>  
5 }
```

Vamos ver na prática! 



Exercício 1

- Além do componente App, crie um novo componente chamado `NovoComponente()`, que contenha informações de **nome**, **idade** e **email**
- Chame o `NovoComponente` dentro do componente `App()`

Perfil

Nome: Lua

idade: 27

email: lua@labenu.com



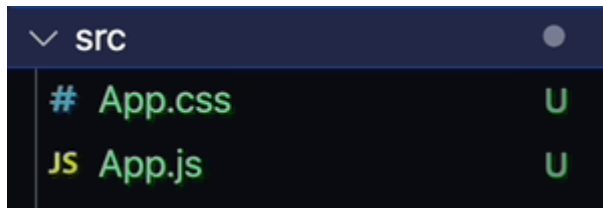
Separando em Arquivos

- É permitido ter mais de um componente em um mesmo arquivo (afinal, eles são só funções)
- No entanto, é uma boa prática criar **um arquivo por componente**
- Podemos comunicar os arquivos por meio de **imports** e **exports**



Criando um Componente Separado

- Recomenda-se a criação de uma pasta chamada **components**, que guarde todos os componentes criados
- O arquivo deve ter o **mesmo nome** do componente que ele guarda



Exportando um Componente Separado

- Todo arquivo que possui um componente deve **importar o React no topo**
- Devemos **exportar** o componente (antes do nome)

MeuComponente.js

```
1 import React from 'react'
2
3 export function MeuComponente() {
4   return <div>
5     <h1>Meu primeiro componente!</h1>
6     <p>Este é o meu primeiro componente React</p>
7   </div>
8 }
```



Importando um Componente Separado ☆

- Precisamos **importá-lo** no arquivo onde o usamos
- Fazemos isso por meio da palavra **import**
 - Atenção para as **{chaves}** em volta do nome do componente

App.js

```
1 import React from 'react'
2 import { MeuComponente } from './components/MeuComponente'
3
4 function App() {
5   return <div>
6     <MeuComponente />
7   </div>
8 }
```

Vamos ver na prática! 



Exercício 2

- Crie uma pasta chamada **components**
- Dentro dessa pasta, crie um **arquivo** chamado `NovoComponente.js`
- Transfira o componente antes criado no app para esse arquivo separado, lembrando de **exportar**
- **Importe** o componente dentro do arquivo `App.js`

Perfil

Nome: Lua

idade: 27

email: lua@labenu.com



Exercício 3

- Agora, repita o componente dentro do App() 3 vezes, mantendo as informações repetidas

Perfil

Nome: Lua

idade: 27

email: lua@labenu.com

Perfil

Nome: Lua

idade: 27

email: lua@labenu.com

Perfil

Nome: Lua

idade: 27

email: lua@labenu.com



10 min



Pausa para relaxar 🧘



Props (segundo pilar do React)

Labenu_



Propriedade dos Componentes

- Já vimos que **componentes são funções** que renderizam HTML na tela
- Os componentes podem funcionar de formas diferentes. Para isso, usamos as propriedades (props) no React.
Propriedades podem fornecer algumas funcionalidades como : **Passar dados personalizados para os componentes e disparar mudanças no estado.**



Props

- Componentes pais podem passar **propriedades** para os componentes filhos
- Chamamos essas propriedades de **props**
- Elas podem assumir **qualquer tipo** de valor que vimos em Javascript
 - Strings, números, arrays, objetos, funções, componentes, etc...



Props

- Imagine que temos uma caixa de cereal, ela não tem marca, nome e nem sabor. É uma caixa genérica.
- Temos 1000 caixas e queremos que:
 - 250 caixas sejam de Sucrilhos Original;
 - 250 caixas sejam de Sucrilhos Chocolate;
 - 500 caixas sejam de Snow Flakes de morango.
- Para isso, vamos passar para cada caixa (componente), as informações (props) necessárias.



Props

Entrada (**props**)

```
ingredientes=[["Flocos de milho", "Açúcar", "Chocolate"]]  
nome={"Sucrilhos"}  
sabor={"Chocolate"}
```



Caixa genérica (**componente**)



Saída: Sucrilhos Chocolate! (**JSX**)

(produto pronto, embalagem bonita para o usuário final)



Enviando Props

- **A sintaxe é a mesma dos atributos HTML**
 - Damos um nome arbitrário
 - Passamos o valor entre {chaves}

```
1 import React from 'react'
2 import { MeuComponente } from '../components/MeuComponente'
3
4 function App() {
5   return <div>
6     <MeuComponente texto={'Testando props'} />
7   </div>
8 }
```



Enviando Props

- A sintaxe é a mesma dos atributos HTML
 - **Damos um nome arbitrário**
 - Passamos o valor entre {chaves}

```
1 import React from 'react'
2 import { MeuComponente } from '../components/MeuComponente'
3
4 function App() {
5   return <div>
6     <MeuComponente texto={'Testando props'} />
7   </div>
8 }
```



Enviando Props

- A sintaxe é a mesma dos atributos HTML
 - Damos um nome arbitrário
 - **Passamos o valor entre {chaves}**

```
1 import React from 'react'
2 import { MeuComponente } from '../components/MeuComponente'
3
4 function App() {
5   return <div>
6     <MeuComponente texto={'Testando props'} />
7   </div>
8 }
```



Recebendo Props

- Quando um componente filho recebe props, temos um **argumento** na função que o define

```
1 import React from 'react'
2
3 export function MeuComponente(props) {
4   return <div>
5     <h1>Meu primeiro componente!</h1>
6     <p>Este é o meu primeiro componente React</p>
7     <p>{props.texto}</p>
8   </div>
9 }
```




Recebendo Props

- Esse argumento é um objeto chamado **props** e contém informações **passadas pelo componente pai**

```
1 import React from 'react'
2
3 export function MeuComponente(props) {
4   return <div>
5     <h1>Meu primeiro componente!</h1>
6     <p>Este é o meu primeiro componente React</p>
7     <p>{props.texto}</p>
8   </div>
9 }
```

`{ texto: 'Testando props' }`

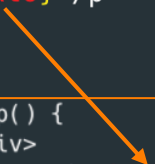


Recebendo Props

- O valor passado pelo pai está **disponível dentro do objeto** e pode ser acessado pelo **nome da propriedade(props)** passada

```
1 import React from 'react'
2
3 export function MeuComponente(props) {
4   return <div>
5     <h1>Meu primeiro componente!</h1>
6     <p>Este é o meu primeiro componente React</p>
7     <p>{props.texto}</p>
8   </div>
9 }
```

```
function App() {
  return <div>
    <MeuComponente texto={'Testando props'} />
  </div>
}
```



Agora, um momento muito esperado...



Enviando e recebendo Props

Enviando Props para filhos

1.

```
1 import React from 'react'
2 import { MeuComponente } from './components/MeuComponente'
3
4 function App() {
5   return <div>
6     <MeuComponente texto={'Testando props'} />
7   </div>
8 }
```

App.js

Recebendo Props do pai

2.

```
1 import React from 'react'
2
3 export function MeuComponente(props) {
4   return <div>
5     <h1>Meu primeiro componente!</h1>
6     <p>Este é o meu primeiro componente React</p>
7     <p>{props.texto}</p>
8   </div>
9 }
```

3.

```
1 import React from 'react'
2
3 export function MeuComponente(props) {
4   return <div>
5     <h1>Meu primeiro componente!</h1>
6     <p>Este é o meu primeiro componente React</p>
7     <p>{props.texto}</p>
8   </div>
9 }
```

MeuComponente.js



Regra Importante



- Props são *read-only* ⇒ **Somente Leitura**

- Componentes filhos não podem **nunca** alterar o objeto de props



Exercício 4

- No App.js, **envie** as props de cada componente
- No componente, **receba** as props e **utilize** elas nos campos correspondentes
- O resultado deve se parecer com o exemplo ao lado, **os dados devem ser dinâmicos**

Perfil

Nome: Lua

idade: 27

email: lua@labenu.com

Perfil

Nome: Yuzo

idade: 27

email: yuzo@labenu.com

Perfil

Nome: Lau

idade: 24

email: lau@labenu.com



Resumo

Labenu_



Resumo

- Para reutilizar um componente, usamos as **props**
- Props são dados passados de componente pai para componente filho e são **read-only**
- **Passando props:** nome arbitrário e valor entre chaves
- **Recebendo props:** objeto props, propriedade com o nome arbitrário escolhido



Resumo

Enviando Props para filhos

1.

```
1 import React from 'react'
2 import { MeuComponente } from '../components/MeuComponente'
3
4 function App() {
5   return <div>
6     <MeuComponente texto={'Testando props'} />
7   </div>
8 }
```

App.js

Recebendo Props do pai

2.

```
1 import React from 'react'
2
3 export function MeuComponente(props) {
4   return <div>
5     <h1>Meu primeiro componente!</h1>
6     <p>Este é o meu primeiro componente React</p>
7     <p>{props.texto}</p>
8   </div>
9 }
```

3.

```
1 import React from 'react'
2
3 export function MeuComponente(props) {
4   return <div>
5     <h1>Meu primeiro componente!</h1>
6     <p>Este é o meu primeiro componente React</p>
7     <p>{props.texto}</p>
8   </div>
9 }
```

MeuComponente.js



Resumo

- Componentes são **funções** que retornam um **JSX**
- Usar componente:
 - `<MeuComponente></MeuComponente>`
 - `<MeuComponente/>`
- Criamos um arquivo com o componente
 - **Exportar:** `export` ou `export default`
 - **Importar:**
 - `import {MeuComponente} from './path'`
 - `import MeuComponente from './path'` (no caso do `export default`)



Dúvidas? 🙋

Labenu_

