

# Renderização de Listas

Labenu\_



# O que vamos ver hoje?

- Renderizar Arrays
- Salvar arrays em estados



# Renderizando Listas

Labenu\_



# Renderizando Listas

- Frequentemente, os dados chegam para nós como **listas** (arrays) e precisamos renderizar os itens na tela
- Se apenas colocarmos um array no meio do código, o layout fica estranho ou simplesmente não funciona!
- Mas se colocarmos essa lista de dados em um **array de componentes**, o React entende que deve renderizar cada um deles!

Vamos ver na prática! 



# Renderizando Listas - Exemplo



```
const listaDeComponentes = [  
  <li>Item 1</li>,  
  <li>Item 2</li>,  
  <li>Item 3</li>  
]  
  
function App() {  
  return (  
    <div>  
      <ul>{listaDeComponentes}</ul>  
    </div>  
  );  
}
```

- Item 1
- Item 2
- Item 3



# Renderizando Listas

- Mas essa lista de dados, em geral, não vai chegar pra gente já dentro de um array de componentes
- Geralmente, a lista vem para a gente como uma lista de **strings** ou **objetos**

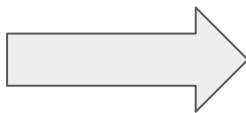
```
const listaDeStrings = [  
  "Item 1",  
  "Item 2",  
  "Item 3"  
]
```



# Renderizando Listas

- Nosso objetivo é **tratar essa lista** de strings/objetos para que se torne uma lista de componentes!

```
const listaDeStrings = [  
  "Item 1",  
  "Item 2",  
  "Item 3"  
]
```



```
const listaDeComponentes = [  
  <li>Item 1</li>,  
  <li>Item 2</li>,  
  <li>Item 3</li>  
]
```



# Renderizando Listas

- Para fazer isso, podemos usar a função de array **map** do JavaScript!
- Relembrando: essa função transforma um array em outro, mapeando cada posição
- No caso, com o uso da função, conseguiremos transformar um **array de strings** em um **array de componentes**





# Exemplo de map

```
const listaDeDados = [  
  'Item 1',  
  'Item 2',  
  'Item 3'  
]  
  
const listaDeComponentes = listaDeDados.map((dado) => {  
  return <li>{dado}</li>  
})  
  
function App() {  
  return (  
    <div>  
      <ul>{listaDeComponentes}</ul>  
    </div>  
  );  
}
```



# Exemplo de map

```
const listaDeDados = [  
  'Item 1',  
  'Item 2',  
  'Item 3'  
]  
  
const listaDeComponentes = listaDeDados.map((dado) => {  
  return <li>{dado}</li>  
})  
  
function App() {  
  return (  
    <div>  
      <ul>{listaDeComponentes}</ul>  
    </div>  
  );  
}
```

```
const listaDeStrings = [  
  "Item 1",  
  "Item 2",  
  "Item 3"  
]
```



```
const listaDeComponentes = [  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
]
```

Vamos ver na prática! 



# Renderizando Listas - Key

- **Keys dos componentes:**
  - Cada componente deve ter um **identificador único**
  - Serve para o React **otimizar** o processamento sabendo quais elementos mudaram
  - Deve ser passada como a prop **key** do componente da lista



# Renderizando Listas - Key

```
const listaDeDados = [  
  'Item 1',  
  'Item 2',  
  'Item 3'  
]  
  
const listaDeComponentes = listaDeDados.map((dado) => {  
  return <li key={dado}>{dado}</li>  
})  
  
function App() {  
  return (  
    <div>  
      <ul>{listaDeComponentes}</ul>  
    </div>  
  );  
}
```

Vamos ver na prática! 



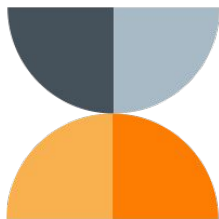


# Exercício 1

- Suponha que temos um array de objetos onde cada objeto possui 2 propriedades: **nome** e **idade**
- Mostre esse array na tela como mostrado ao lado:

Caio	26
Chijo	27
Laís	28
Mandi	29





# Pausa para relaxar 🧘

10 min

- Para renderizar uma **lista** (de números, strings, objetos... ) na tela, precisamos transformá-la em uma **lista de componentes**
- Para fazer isso, usamos a função **map**
- Cada item da lista deve ter uma prop chamada **key**, que é um valor **único**



# Listas no estado

Labenu\_



# Lista no State

- Geralmente nossas listas serão **dinâmicas**: ou seja, poderão ter itens **adicionados, alterados** ou **apagados**
- Nesse caso, é interessante **armazenar a lista de dados no estado**, pois queremos que nossa interface reflita o conteúdo da lista

Vamos ver na prática! 





# Temos que falar sobre o JavaScript

- No Javascript, existem dois tipos de dados:
  - Os que são passados por **valores** (primitivos)
    - boolean, string, number, null, undefined
  - Os que são passados por **referência**
    - funções, arrays e objetos



# O que significa isso?

- Para criar uma cópia de uma variável que contenha um tipo **primitivo**, podemos fazer:

```
1 let a = 20
2 let b = a
3 console.log("a", a) // a é 20
4 console.log("b", b) // b é 20
5
6 a = 50
7 console.log("a", a) // a é 50
8 console.log("b", b) // b é 20
```

Vamos ver na prática! 



# O que significa isso?


- Vamos tentar criar uma cópia de um **array** da mesma forma para ver o que acontece!

```
1 let c = [1, 2, 3]
2 let d = c
3 console.log("c", c) // c é [1, 2, 3]
4 console.log("d", d) // d é [1, 2, 3]
5
6 c.push(4)
7 console.log("c", c) // c é [1, 2, 3, 4]
8 console.log("d", d) // d ?
```

Vamos ver na prática! 



# O que significa isso?

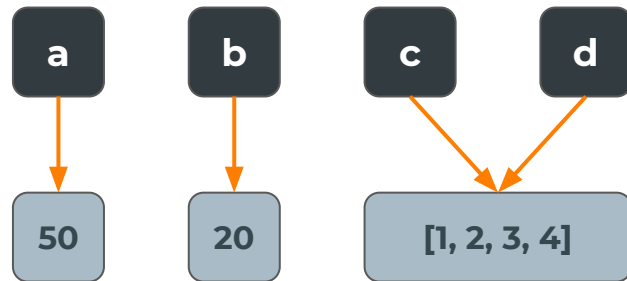
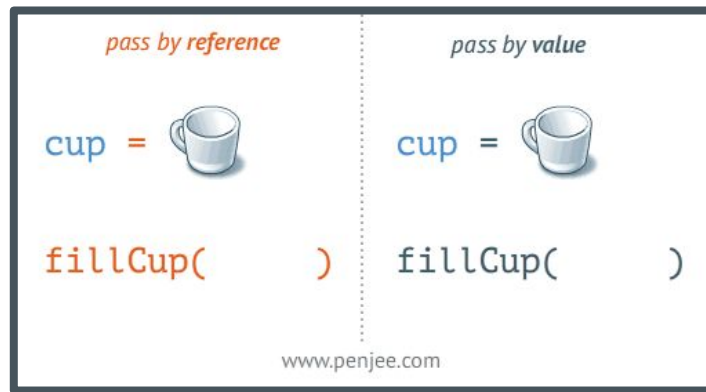
- Ao alterar o valor do array original, alteramos também o valor da cópia! 

```
1 let c = [1, 2, 3]
2 let d = c
3 console.log("c", c) // c é [1, 2, 3]
4 console.log("d", d) // d é [1, 2, 3]
5
6 c.push(4)
7 console.log("c", c) // c é [1, 2, 3, 4]
8 console.log("d", d) // d é [1, 2, 3, 4]
```



# O que está acontecendo? 🤯

- Dados passados por **valor** ocupam um **novo espaço** na memória e são algo totalmente diferente
- Dados passados por **referência** basicamente são o **mesmo dado** com nomes diferentes



# Então como copiar um array? 🤔

- Precisamos criar **um novo array** que ocupe **outro** lugar na memória
- Existem diversas maneiras de fazer isso, mas na versão ES6 do javascript foi inserido o **spread operator**, que é atualmente a forma mais indicada



# Então como copiar um array? 🤔

```
1 let c = [1, 2, 3]
2 let d = [...c]
3 console.log("c", c) // c é [1, 2, 3]
4 console.log("d", d) // d é [1, 2, 3]
5
6 c.push(4)
7 console.log("c", c) // c é [1, 2, 3, 4]
8 console.log("d", d) // d é [1, 2, 3]
```

O spread operator (...)  **copia**  tudo que estava no array **c** e **cria um novo** array **d**



# Voltando para o React...

- **Reatividade**: quando algo muda no **estado**, essa mudança é refletida automaticamente na **tela**
- Para que o React entenda que ocorreu uma mudança no array, precisamos **trocar a referência!**
- Ou seja, precisamos **criar uma cópia do array com as mudanças desejadas** e **colocar esse novo array no estado**





# Na Prática: adicionando itens



```
1 export default class App extends React.Component {
2   state = {
3     listaDeFrutas: ["Batata", "Maçã", "Laranja"]
4   };
5
6   adicionaFruta1 = () => {
7     const novasFrutas = [...this.state.listaDeFrutas];
8     novasFrutas.push("Abacate");
9     this.setState({ listaDeFrutas: novasFrutas });
10  };
11
12  adicionaFruta2 = () => {
13    const novasFrutas = [...this.state.listaDeFrutas, "Abacate"];
14    this.setState({ listaDeFrutas: novasFrutas });
15  };
16
17  adicionaFruta3 = () => {
18    this.setState({ listaDeFrutas: [...this.state.listaDeFrutas, "Abacate"] });
19  };
20
21  render() {
22    ...
23  }
24 }
25
```



# Na Prática: alterando itens



```
1 export default class App extends React.Component {
2   state = {
3     listaDeFrutas: ["Batata", "Maçã", "Laranja"]
4   };
5
6   alteraFruta = () => {
7     const novasFrutas = [...this.state.listaDeFrutas];
8     novasFrutas[0] = "Banana"
9     this.setState({ listaDeFrutas: novasFrutas });
10  };
11
12  render() {
13    ...
14  }
15 }
16
```



# Na Prática: removendo itens



```
1 export default class App extends React.Component {
2   state = {
3     listaDeFrutas: ["Batata", "Maçã", "Laranja"]
4   };
5
6   // Filter
7   removeFruta1 = () => {
8     const frutasFiltradas = this.state.listaDeFrutas.filter((fruta) => {
9       return fruta !== "Batata";
10     })
11
12     this.setState({ listaDeFrutas: frutasFiltradas });
13   };
14
15   // Splice
16   removeFruta2 = () => {
17     const indiceBatata = this.state.listaDeFrutas.findIndex((fruta) => {
18       return fruta === "Batata";
19     })
20
21     const novasFrutas = [...this.state.listaDeFrutas];
22     novasFrutas.splice(indiceBatata, 1);
23     this.setState({ listaDeFrutas: novasFrutas });
24   };
25
26   render() {
27     ...
28   }
29 }
```





## Exercício 2

- Na página do exercício anterior, crie um formulário que permita adicionar novos professores

Nome	Idade	Adicionar
Caio	26	
Chijo	27	
Laís	28	
Mandi	29	
Bruno	23	



# Pausa para relaxar 🐾💤

5 min





## Exercício 3

- Na página do exercício anterior, crie um botões em cada usuário que permitam deletá-los da lista

Nome	Idade	Adicionar
Caio	26	X
Chijo	27	X
Laís	28	X
Mandi	29	X



# Resumo

Labenu\_



# Resumo

- Para renderizar uma **lista** na tela (strings, números, objetos...), precisamos transformá-la em uma **lista de componentes**
- Para fazer isso, usamos a função **map**
- Cada item da lista deve ter uma prop chamada **key**, que é um valor **único**





# Resumo

- Quando colocamos listas em estados, temos que tomar **cuidado** na hora de **alterar seus elementos**
- Precisamos criar uma cópia do array e fazer as alterações
  - **Adicionar itens:** push ou spread operator
  - **Alteração:** alterar o índice específico
  - **Remover itens:** splice ou filter



# Dúvidas? 🤔

Labenu\_





Obrigado(a)!