

Semana 8 - Aula REVM2

# Revisão do Módulo 2

Labenu\_



# O que vamos ver hoje?

- Projeto React
- Componentes
- Estados e Props
- Interações com Usuário
- Renderização de Arrays
- Renderização Condicional
- Ciclo de Vida
- Integração com APIs



# Projeto React

Labenu\_



# Todos os passos que seguimos 🦶



// Criar Projeto

```
npx create-react-app meu-app
```

```
npm install styled-components
```

```
npm install axios
```

// Rodar

```
npm run start
```

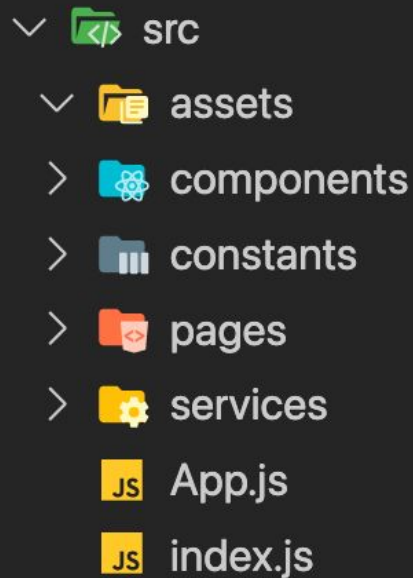
// Buildar

```
npm run build
```

```
surge ./build
```



# Estrutura do Projeto



```

└─ src
  └─ assets
  └─ components
  └─ constants
  └─ pages
  └─ services
  App.js
  index.js

```

- **assets:** imagens, ícones, fontes
- **components:** reutilizáveis
- **constants:** URLs, cores
- **pages:** telas do seu app
- **services:** integrações
- **App.js:** primeiro componente



# Componentes

Labenu\_



# Componentes

- Um componente é uma **função** ou **classe reutilizável** que retorna um **JSX** a ser exibido na tela
- **Quando criar um componente:**
  - Layout repetido
  - Código muito grande/complexo
  - Isolar funcionalidade
  - Nomear partes do projeto
  - ... Mas não tem um jeito "certo"!





# Exercício 1

- Ao lado temos uma tela de perfil de um site de publicação de artigos
- Qual seria a divisão de componentes que você faria?



**Juanita Phillips**

Desenvolvedora e Jornalista



## Como fazer animações

Neste artigo, aprenderemos a fazer animações

## Como fazer animações

Neste artigo, aprenderemos a fazer animações

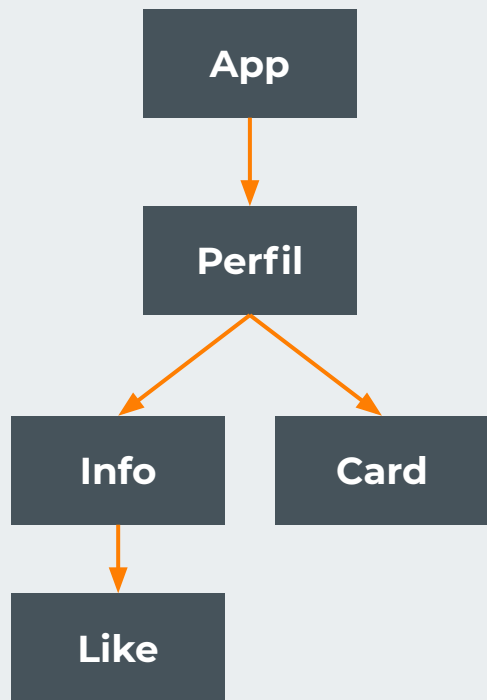
## Como fazer animações

Neste artigo, aprenderemos a fazer animações





# Exemplo Componentização



# Componentes Funcionais


```
1 import React from "react"
2
3 export const MeuComponente = (props) => {
4
5   const incrementar = (numero) => {
6     return numero + 1;
7   }
8
9   const textoTela = incrementar(props.valor)
10
11   return <p>{textoTela}</p>
12 }
```

- Recebe **props**
- Retorna **JSX**
- Pode ter funções
- Pode ter variáveis
- Não tem **state**
- Não tem **lifecycle**



# Componentes de Classe

```
1 import React from "react"
2
3 export class MeuComponente extends React.Component {
4
5   state = {
6     contador: 0
7   }
8
9   componentDidMount() {
10     // Requisições
11   }
12
13   incrementar = (numero) => {
14     return numero + 1;
15   }
16
17   textoTela = this.incrementar(this.props.valor)
18
19   render() {
20     return <p>{this.textoTela}</p>
21   }
22 }
23 }
```

- Já possuí as **props**
- **state** e **lifecycle**
- Funções e variáveis sem **const** ou **let**
- Para acessar as coisas, usamos **this**
- Arrow functions 
- **render()** retorna JSX



```
1 export default class MeuComponente extends React.Component {
2   // Primeiro de tudo: iniciamos o state
3   state = { listaFrutas: ["Banana", "Maçã", "Uva"] }
4
5   // Depois, colocamos os métodos de lifecycle que precisarmos
6   componentDidMount(){}
7   componentDidUpdate(){}
8   componentWillUnmount(){}
9
10  // Caso precisemos de outras funções ou variáveis, podemos colocar elas aqui também
11  minhaVariavel = "Bananinha"
12  minhaFuncao = () => { // Uso de arrow functions fortemente recomendado
13    return "Olá mundo!"
14  }
15
16  // Por último temos o método render(), que retorna o JSX
17  // Tudo dentro da classe mas fora do render pode ser referenciado nele usando o this
18  render() {
19    // IMPORTANTE => Não fazer alterações de state aqui dentro!
20    // Podemos colocar aqui funções relacionadas com o layout, ex: função que renderiza uma lista
21
22    // Dentro do render, precisamos declarar const/let/function
23    const listaFrutas = this.state.listaFrutas.map((fruta) => return(<p key={fruta}>{fruta}</p>))
24
25    return (
26      <div>
27        <h1>Frutas que gosto:</h1>
28        <p>{this.minhaFuncao()}</p> // Declarada fora do render => precisa do this
29        {listaFrutas} // Declarada dentro do render => não precisa do this
30      </div>
31    )
32  }
33 }
```



# Estados e Props

Labenu\_



# State

- Um jeito de **guardar dados** no nosso **componente**
- São criados e consumidos **DENTRO** do componente
- Podemos **modificar** com a função `this.setState()`
- A mudança do estado causa **re-renderização**

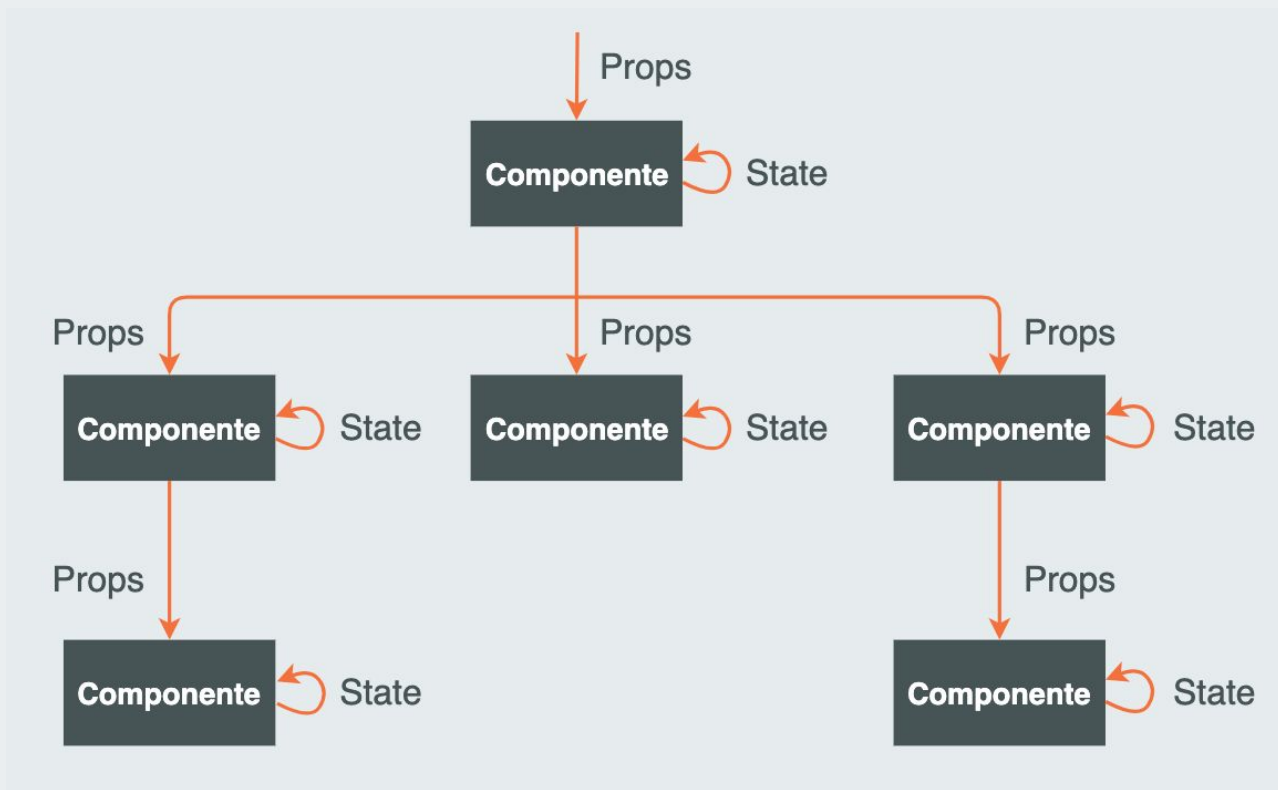


# Props

- Dados que vêm **DE FORA** do componente
- Usadas para torná-lo mais reutilizável
- Podem ter **qualquer formato** (string, array, função...)
- Props **não mudam** dentro do componente que a recebe



# Árvore de Componentes





# Funcional x Classe

```
1 import React from "react"
2
3 export const MeuComponente = (props) => {
4
5   const incrementar = (numero) => {
6     return numero + 1;
7   }
8
9   const textoTela = incrementar(props.valor)
10
11   return <p>{textoTela}</p>
12 }
```

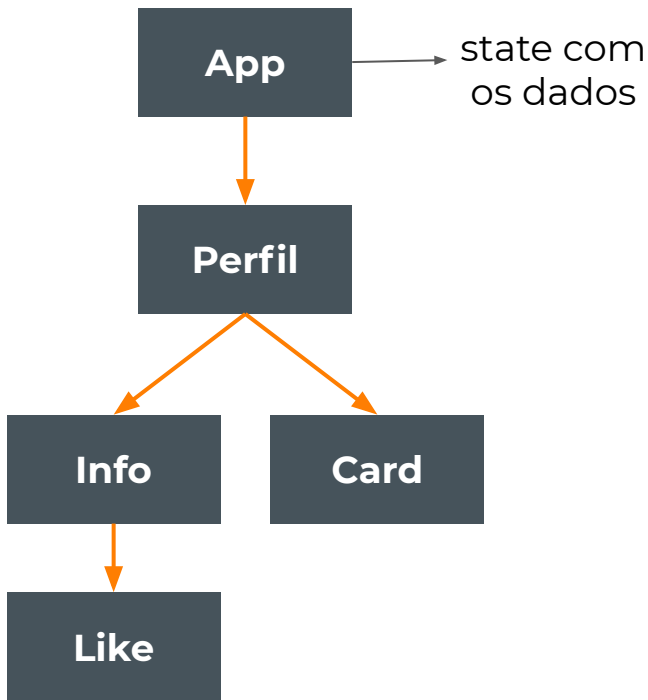
```
1 import React from "react"
2
3 export class MeuComponente extends React.Component {
4
5   state = {
6     contador: 0
7   }
8
9   componentDidMount() {
10     // Requisições
11   }
12
13   incrementar = (numero) => {
14     return numero + 1;
15   }
16
17   textoTela = this.incrementar(this.props.valor)
18
19
20   render() {
21     return <p>{this.textoTela}</p>
22   }
23 }
```





## Exercício 2

- Temos o layout do perfil mostrado no exercício 1 e as informações estão todas no estado do componente App
- Usar props para passar todas as informações necessárias para todos os filhos
- [Link do Exercício](#)



# Interações com Usuário

Labenu\_



# Interações com Usuário

- O usuário pode interagir com nosso site através do mouse e do teclado
- Quando isso acontece, normalmente desejamos:
  - realizar alguma **ação imediata**
  - **guardar** alguma informação
  - ou ambas as opções anteriores



# Eventos dos Elementos

- Passamos **funções** para os **eventos** de cada componente com o qual o usuário pode interagir
  - `onClick()`
  - `onChange()`
  - `onKeyDown()`
- Ações imediatas acontecem **no momento do evento**
- Para guardar dados usamos **inputs controlados**



# Eventos Com Parâmetro

- Às vezes, quando passamos uma função para um evento, precisamos passar também um **parâmetro**
- Nesses casos usamos a sintaxe da **arrow function**

```
1 <button onClick={funcaoSemParametro}>Botão 1</button>  
2  
3 <button onClick={() => funcaoComParametro(param)}>Botão 2</button>
```



# Inputs Controlados

- Um input controlado é aquele no qual o **valor guardado** está sob **nossa responsabilidade**
- Guardar os dados é interessante para que possamos **manipulá-los** na nossa aplicação
- Precisamos passar para o input os atributos **value** e **onChange**, relacionando-os ao nosso **state**



# Inputs Controlados

```
1 export default class App extends React.Component {
2   state = {
3     nome: ""
4   }
5
6   mudouNome = (event) => {
7     this.setState({ nome: event.target.value });
8   }
9
10  render() {
11    return(
12      <div>
13        <input
14          value={this.state.nome}
15          onChange={this.mudouNome}
16        />
17        <h1>{this.state.nome}</h1>
18      </div>
19    )
20  }
21 }
```

Letícia Chijo

**Letícia Chijo**





# Renderização de Arrays

Labenu\_



# Funções de Array

Função	Utilização	Retorna <i>array</i> ?	Tamanho do array
<b><i>forEach</i></b>	Ler ou utilizar os itens do <i>array</i>	Não	-
<b><i>map</i></b>	Criar um novo <i>array</i> com elementos modificados em relação ao original	Sim	Igual ao original
<b><i>filter</i></b>	Criar um novo <i>array</i> com alguns elementos do original	Sim	Igual ou menor que o original



# Renderização de listas



```
1 import React from "react"
2
3 const App = () => {
4   const array = ["banana", "morango", "maçã"]
5
6   const listaFrutas = array.map((fruta) => {
7     return <li key={fruta}>{fruta}</li>
8   })
9
10  return <ul>{listaFrutas}</ul>
11 }
```

- Para mostrar a lista na tela, precisamos retornar o layout **de cada item**
- Para isso, usamos a função **map**



# Arrays no state



```
1 state = {  
2   listaFrutas: ["banana", "morango", "maçã"]  
3 }  
4  
5 adicionarAbacate = () => {  
6   const novaLista = [...this.state.listaFrutas, "abacate"];  
7   this.setState({ listaFrutas: novaLista });  
8 }
```

- Não devemos manipular arrays **diretamente** no state
- Fazemos uma **cópia**, manipulamos e guardamos



# Renderização Condicional

Labenu\_



# Renderização Condicional

- Às vezes, queremos mostrar coisas na tela que dependem de **condições**
- Para isso, podemos usar 4 estruturas diferentes:
  - if/else
  - switch case
  - ternário
  - curto circuito



# Renderização Condicional

- **if/else** e **switch case** são mais usados para escolher uma tela ou componente inteiro
- **Ternários** e **curto-circuitos** são colocados no meio do JSX para detalhes da tela



# Troca de Telas com switch case

```
1 export default class App extends React.Component {
2   state = {
3     telaAtual: "home"
4   }
5
6   mudarTela = (nomeTela) => {
7     this.setState({telaAtual: nomeTela})
8   }
9
10  escolherTelaAtual = () => {
11    switch (this.state.telaAtual){
12      case "home":
13        return <Home mudarTela={mudarTela}/>
14      case "perfil":
15        return <Perfil mudarTela={mudarTela}/>
16      default:
17        return <Home mudarTela={mudarTela}/>
18    }
19  }
20
21  render() {
22    return this.escolherTelaAtual()
23  }
24 }
```





# Renderização Condicional



- **Ternários:** duas opções diferentes

```
1 export default class App extends React.Component {  
2   state = {  
3     darkMode: false  
4   }  
5  
6   render() {  
7     return {  
8       <div>  
9         {this.state.darkMode ? <ComponenteDark/> : <ComponenteLight/>}  
10      </div>  
11    }  
12  }  
13 }
```



# Renderização Condicional

- **Curto-circuito:** uma opção que depende de condições

```
1 export default class App extends React.Component {  
2   state = {  
3     gostaBanana: false  
4   }  
5  
6   render() {  
7     return {  
8       <div>  
9         //this.state.gostaBanana ? <Banana /> : null  
10        {this.state.gostaBanana && <Banana />}  
11      </div>  
12    }  
13  }  
14 }
```

Bastante usado  
para verificar se  
dado existe antes  
de renderizar



# Ciclo de Vida

Labenu\_



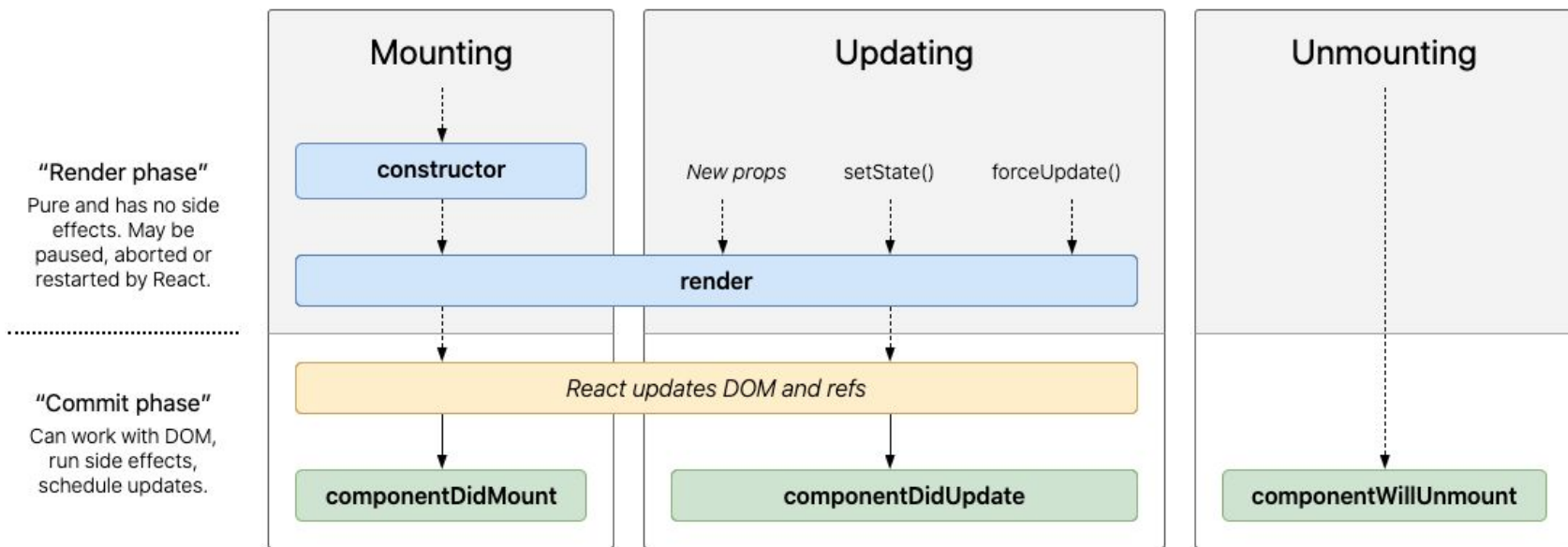
# Ciclo de vida de um componente 🧠

- **Ciclos de vida** são métodos executados em momentos diferentes da vida de um componente

<b>Montagem</b>	<code>componentDidMount()</code>
<b>Atualização</b>	<code>componentDidUpdate()</code>
<b>Desmontagem</b>	<code>componentWillUnmount()</code>



# Ciclo de vida de um componente 🧠



# Integração com APIs

Labenu\_



# Integração com APIs

- Usamos APIs principalmente para interagir com um **banco de dados**
- No código React, para utilizar essas APIs, usamos a lib **axios**
- Precisamos nos preocupar com **assincronicidade** e **tratamento de erros**



# .then() e .catch()

```
1 export default class App extends React.Component {
2   state = {
3     nomeAtividade: ""
4   }
5
6   componentDidMount() {
7     this.pegarAtividade()
8   }
9
10  pegarAtividade = () => {
11    axios
12      .get("https://www.boredapi.com/api/activity/")
13      .then((resposta) => {
14        this.setState({
15          nomeAtividade: resposta.data.activity,
16        })
17      })
18      .catch((erro) => {
19        console.log(erro);
20      })
21  }
22
23  render() {
24    return <p>{this.state.nomeAtividade}</p>
25  }
26 }
27
```





# async / await

```
1 export default class App extends React.Component {
2   state = {
3     nomeAtividade: ""
4   }
5
6   componentDidMount() {
7     this.pegarAtividade()
8   }
9
10  pegarAtividade = async() => {
11    try {
12      const dados = await axios.get("https://boredapi.com/api/activity/")
13      this.setState({nomeAtividade: dados.data.activity})
14    } catch (erro) {
15      console.log(erro)
16    }
17  }
18
19  render() {
20    return <p>{this.state.nomeAtividade}</p>
21  }
22 }
```





Obrigado(a)!