

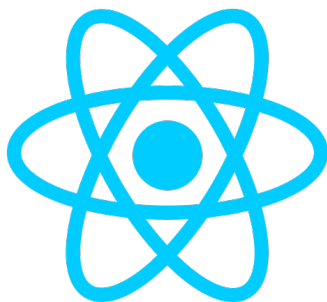
INSTITUTO FEDERAL
Sul-rio-grandense

Câmpus
Pelotas



DESENVOLVIMENTO FRONT-END II

Ciclo de Vida de Componentes e Principais Hooks



JavaScript

Prof. Gill Velleda Gonzales | gillgonzales@ifsul.edu.br | @g1ll
CSTSI – Março - 2025

Desenvolvimento Front-End II

Tópicos



- **Ciclo de Vida:**

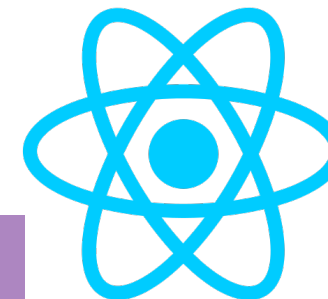
- Montagem (mount)
- Atualização (update)
- Desmontagem (unmount)
- Arquitetura de Componentes
- Ciclo de Vida de Componentes

- **Principais Hooks:**

- useState
- useEffect



JavaScript

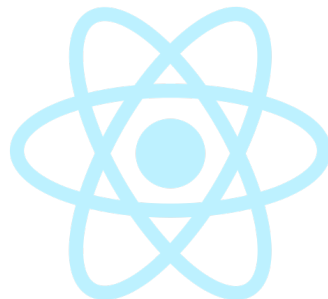


Ciclo de Vida

CONCEITOS PRINCIPAIS



- **Ciclo de Vida:**
 - O **ReactJS** se baseia nas três principais fases dos componentes:
 - **Montagem (mounted):**
 - Quando o componente é criado e adicionado ao DOM, visível na página.
 - **Atualização (updated):**
 - Quando as propriedades repassadas ou o próprio estado do componente é atualizado. Neste caso o componente é renderizado novamente.
 - **Desmontagem (unmounted):**
 - O componente é destruído, portanto removido do DOM.

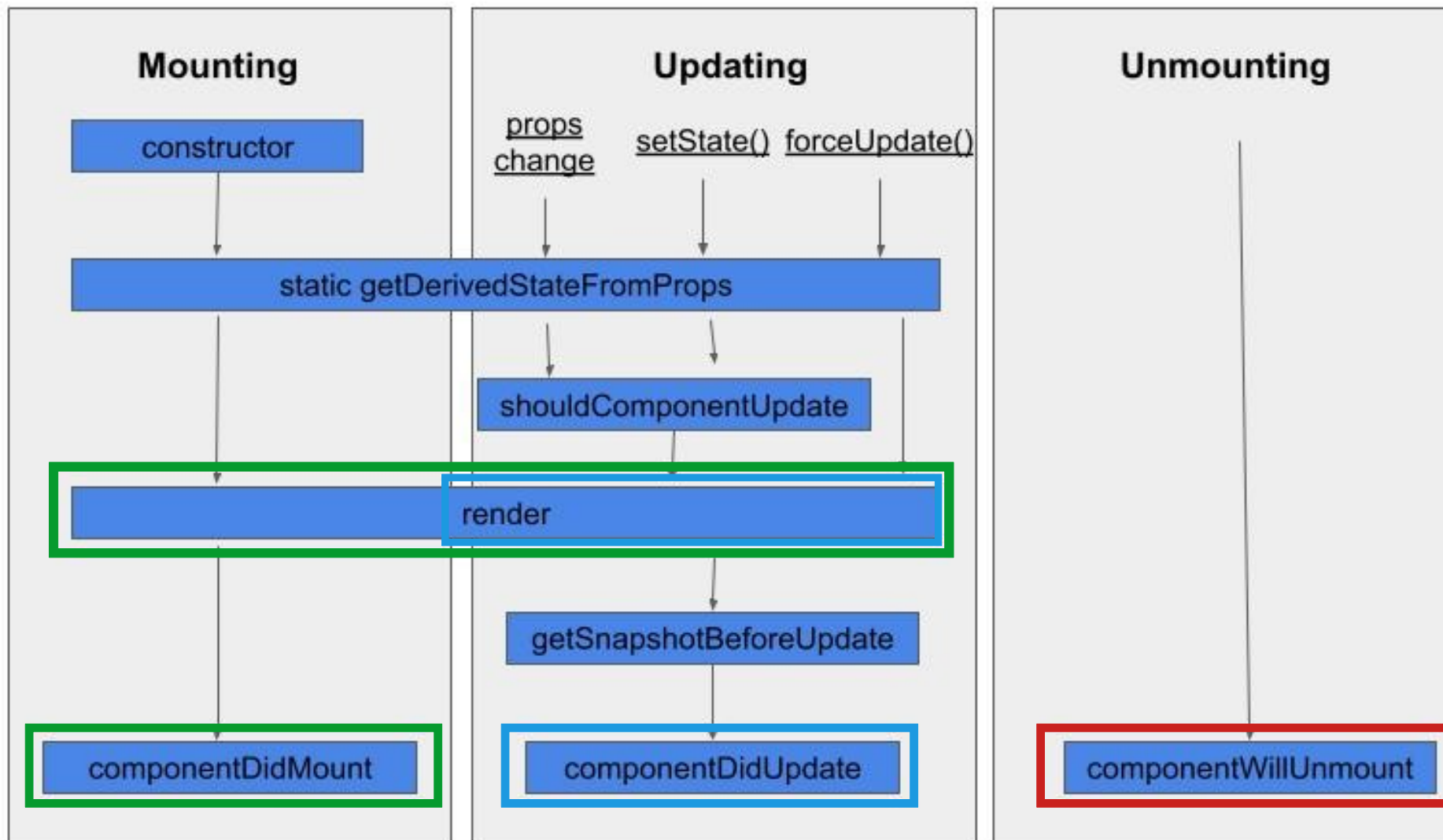


Ciclo de Vida

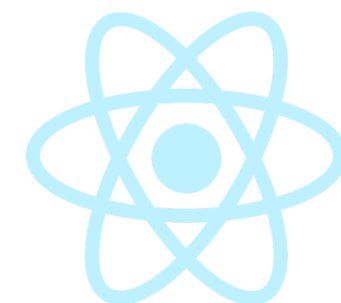
CONCEITOS PRINCIPAIS



React Component Lifecycle

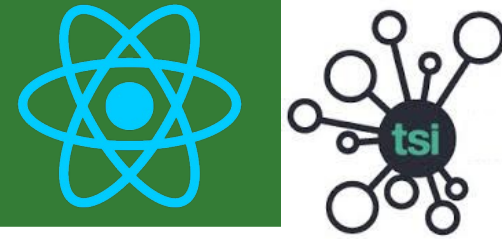


Fonte: <https://zerotomastery.io/cheatsheets/react-cheat-sheet/>



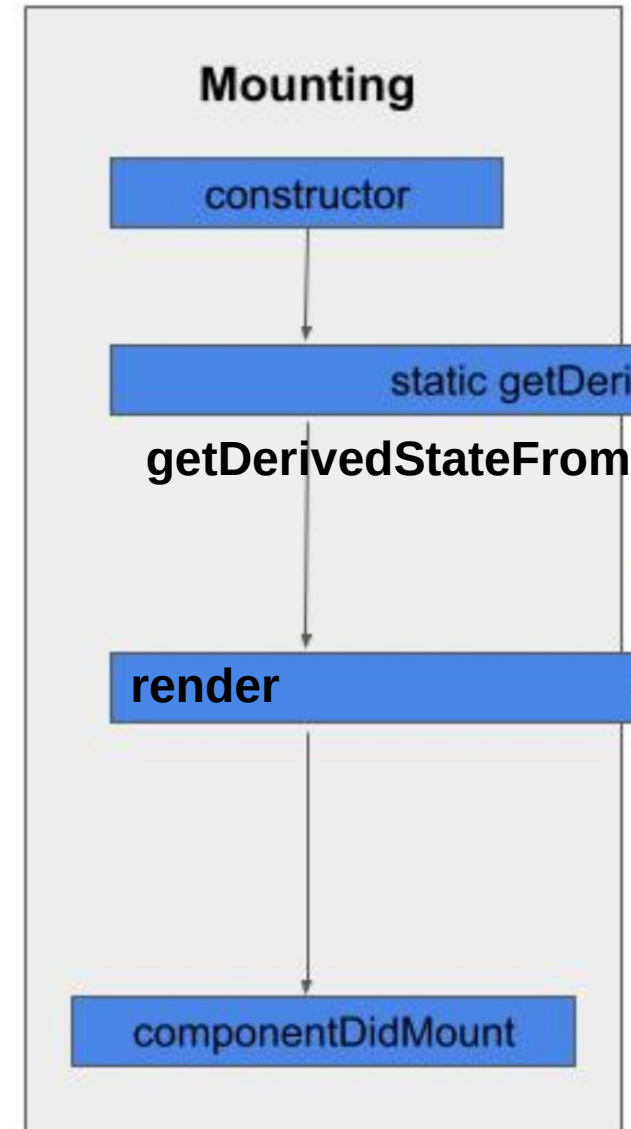
Ciclo de Vida do Componente

CONCEITOS PRINCIPAIS



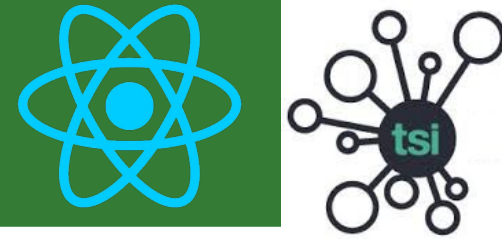
Montagem (mounted):

- Métodos chamados na versão com **Classes** são:
 - ▶ **Construtor**: inicializa o componente, ações necessárias na inicialização são processadas aqui.
 - ▶ **GetDerivedStateFromProps**: método estático chamado para compor o estado do componente a partir de propriedades passadas pelo componente pai.
 - ▶ **Render**: Acontece a primeira renderização, ou seja, o componente é mostrado na tela, adicionado ao DOM.
 - ▶ **ComponenteDidMount**: Ações necessárias **após** a montagem do componente são realizadas aqui.
 - ▶ **Exemplo com estilo Classes e Functions**



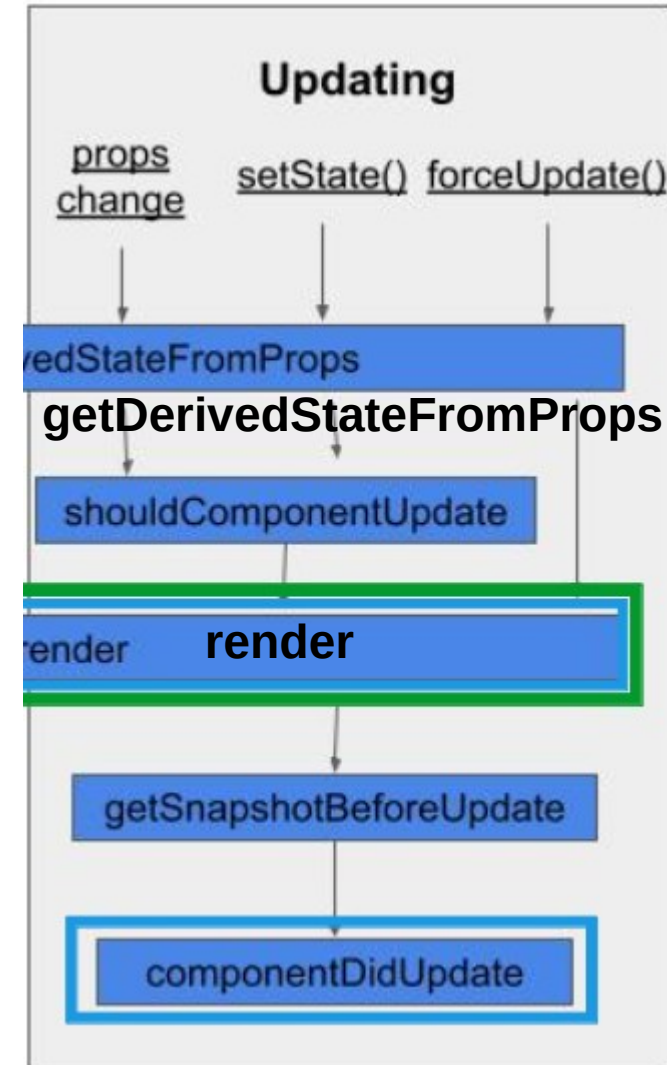
Ciclo de Vida do Componente

CONCEITOS PRINCIPAIS



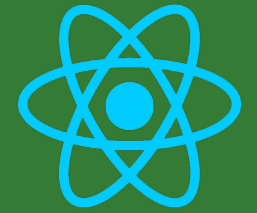
Atualização (updated):

- Métodos chamados na versão com **Classes** são:
 - ▶ ***GetDerivedStateFromProps***: quando propriedades e estados são alterados ou o método ***forceUpdate*** é chamado, executado logo antes do render.
 - ▶ ***shouldComponentUpdate***: avisa o react se o componente deverá ser re-renderizado.
 - ▶ **Render**: realiza a re-renderização, ou seja, o componente é redesenhado na tela, atualizando o seu estado no DOM.
 - ▶ ***getSnapshotBeforeUpdate***: retorna estado após render
 - ▶ **ComponentDidUpdate**: executa após a atualização.
 - ▶ **Exemplo com estilo Classes e Functions**



Ciclo de Vida do Componente

CONCEITOS PRINCIPAIS



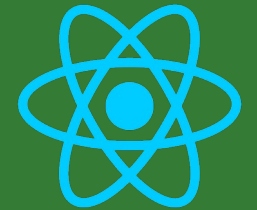
Desmontagem (unmounted):

- Métodos chamados na versão com **Classes** são:
 - ▶ ***componentWillUnmount***: executada antes do componente ser **removido** do DOM.
- *Não se programa mais com Classes no React.*
- Ao em vez de **Classes** utilizaremos **componentes Funcionais**, criados com funções.
- No entanto, é importante compreender os **fundamentos das classes**, pois utilizaremos os **Hooks** que cumprem funções semelhantes aos antigos **métodos de classe**.



Ciclo de Vida do Componente

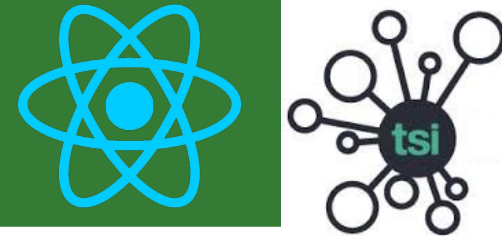
CONCEITOS PRINCIPAIS



- **Classes vs Functions :**
- **Equivalência dos métodos com componente Functions:**
- **Hooks:** são funções que podem ser chamadas dentro de componentes declarados como **Functions**.
 - ▶ Os principais **Hooks** são:
 - **useState() :**
 - Substituirá a propriedade **state** e o método **setState**.
 - **useEffect() :**
 - Substituirá os métodos **componentDidMount**, **componentDidUpdate** e **componentWillUnmount**.

Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useState()

- Fará o papel do atributo **state** e do método **setState** dos componentes do tipo **Classe**, e será responsável por **acessar** e **atualizar** os estados do componente.

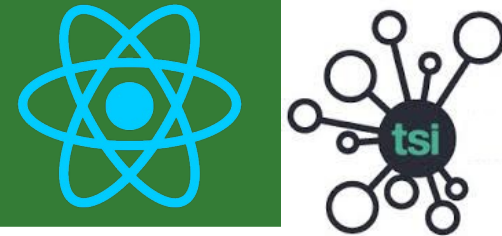
```
AdaClass.jsx x  Ada.jsx x
10 export default class AdaClass extends Component {
11   //useState
12   state = {
13     data: _data,
14     countRender: 1,
15   };
16
10 export default function Ada() {
11   //states
12   const [data, setData] = useState(_data); //setState
13   const [countRender, setCountRender] = useState(1); //setState
14
15
16
```

- O **useState** retorna um **array** com o estado e a função para atualizar este estado, a qual cumpre o papel do **this.setState** para cada estado.

 ***Veja este exemplo***

Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useState()

- Fará o papel do atributo **state** e do método **setState** dos componentes do tipo **Classe**, e será responsável por **acessar** e **atualizar** os estados do componente.

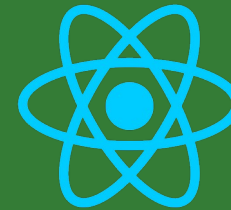
Exemplo

```
AdaClass.jsx x
17 //useEffect
18 componentDidMount() {
19   console.log(this.state.countRender, 'render')
20   ;
21 }
22 //useEffect
23 componentDidUpdate(prevProps, prevState) {
24   if (this.state.data !== prevState.data) {
25     console.log(this.state.countRender,
26       'render');
27   }
28 }
29 //função retornada no useEffect
30 componentWillUnmount() {
31   console.log('Ada será removida!!');
32 }

Ada.jsx x
18 useEffect(() => {
19   //componentDidMount && componentDidUpdate
20   console.log(countRender, 'render');
21   return () => {
22     //componentWillUnmount
23     console.log('Ada será removida!!');
24   };
25 }, [data]);
26
27
28
29
30
31
32
33
34
35
```

Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useState()

- Nova versão do **CardImc** guardando o **imc** como estado.

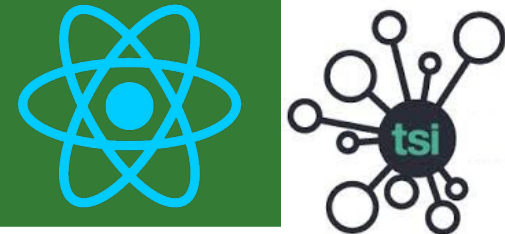
```
CardImc.jsx ×  
1  import { useState } from 'react';  
2  import './style.css';  
3  
4  export default function CardImc({ pessoa }) {  
5    const peso = pessoa.peso;  
6    const alt = pessoa.altura;  
7    const calcImc = () => peso / alt ** 2;  
8    const [imc, setImc] = useState(calcImc);  
9  }
```



[Veja este exemplo](#)

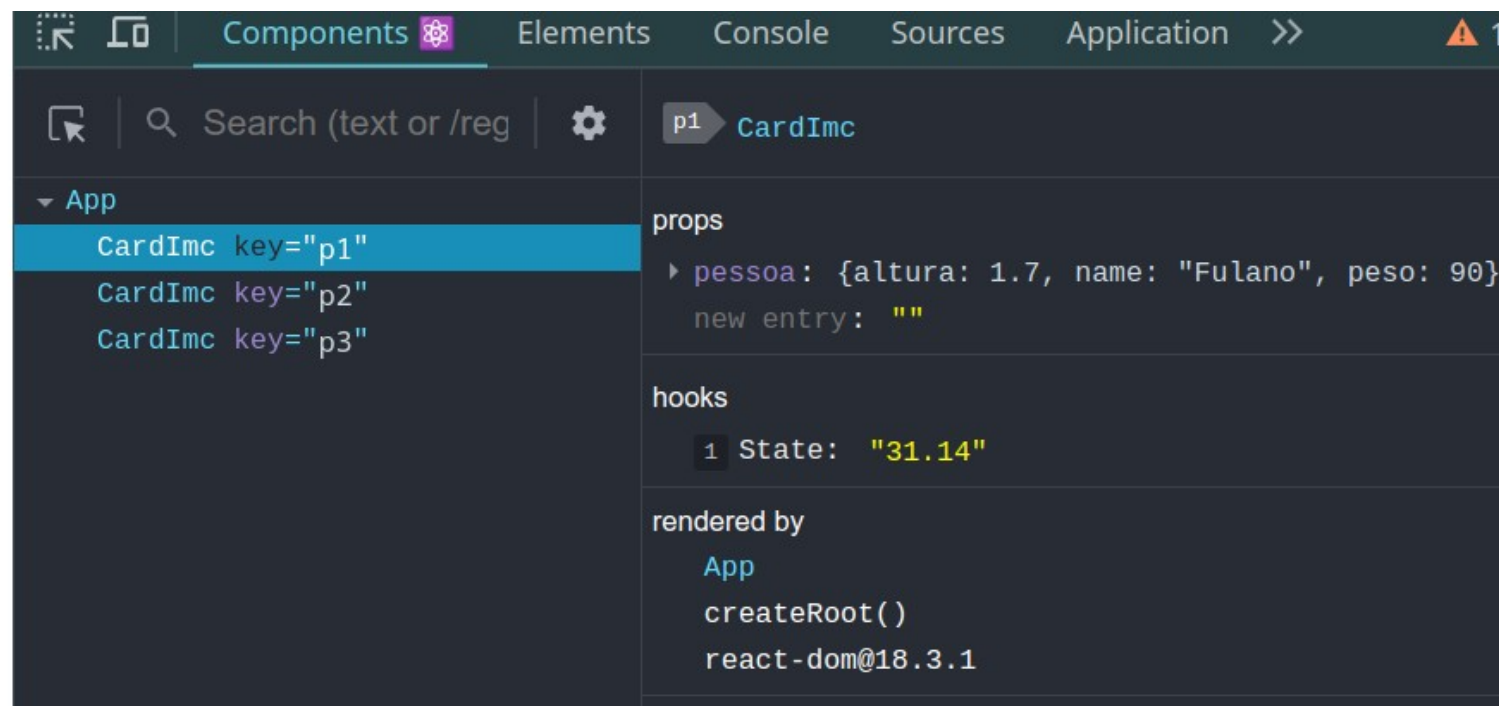
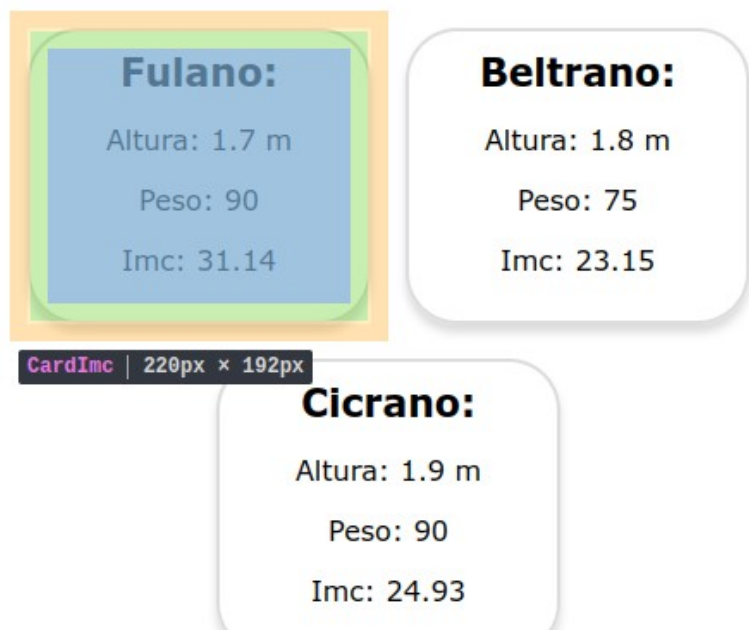
Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useState()

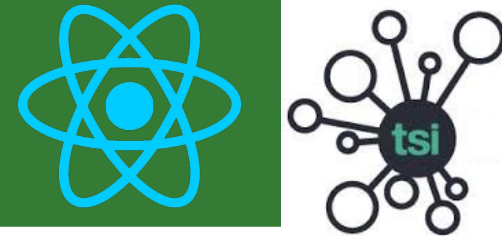
- Nova versão do **CardImc** guardando o **imc** como um estado.



[Veja este exemplo](#)

Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useState()

- Um erro comum quando aprendemos este hook é tentar usar o novo estado ***imediatamente após*** a chamada ao ***setState***.

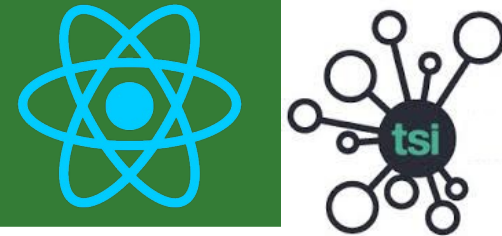
```
CardImc.jsx X ... CardImc.jsx X
9      const incrementaPeso = () => {
10        setPeso(peso + 1);
11        setImc(peso / alt ** 2);
12      };
13
14      const decrementaPeso = () => {
15        setPeso(peso - 1);
16        setImc(peso / alt ** 2);
17      };
18
19      return (
20        <div className="imcCard">
21          <h1>{pessoa.name}</h1>
22          <p>Altura: {alt} m</p>
23          <p>
24            Peso: {peso}
25            <span onClick={incrementaPeso}>&nbsp;&nbsp;&nbsp;+&nbsp;&nbsp;&nbsp;</span>
26            <span onClick={decrementaPeso}>&nbsp;&nbsp;&nbsp;-&nbsp;&nbsp;&nbsp;</span>
27          </p>
28          <p>Imc: {imc.toFixed(2)}</p>
29        </div>
30      );
31    }
```



Veja este exemplo

Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useState()

- O **valor** de **peso** após o **setPeso** (**linhas 11 e 16**), ainda é o **valor atual**, e **NÃO** **peso+1** ou **peso-1**.
- Portanto, o **setImc** estará **calculando** o valor do **imc** de forma **errada**.

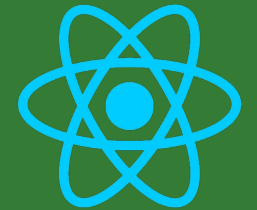
```
CardImc.jsx X ... CardImc.jsx X
9      const incrementaPeso = () => {
10         setPeso(peso + 1);
11         setImc(peso / alt ** 2);
12     },
13
14     const decrementaPeso = () => {
15         setPeso(peso - 1);
16         setImc(peso / alt ** 2);
17     };
18
19     return (
20         <div className="imcCard">
21             <h1>{pessoa.name}</h1>
22             <p>Altura: {alt} m</p>
23             <p>
24                 Peso: {peso}
25                 <span onClick={incrementaPeso}>&nbsp;&nbsp;&nbsp;+&nbsp;&nbsp;&nbsp;</span>
26                 <span onClick={decrementaPeso}>&nbsp;&nbsp;&nbsp;-&nbsp;&nbsp;&nbsp;</span>
27             </p>
28             <p>Imc: {imc.toFixed(2)}</p>
29         </div>
30     );
31 }
```



[Veja este exemplo](#)

Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useState()

- Uma forma de resolver essa situação é criar uma **variável local** da função para calcular o **novo peso**, e usá-la para **atualizar o imc**.
- Neste caso estaremos utilizando o **valor atualizado do novo peso** para também atualizar o **novo valor de imc**.
- O valor do **state peso** só será igual ao **_peso** na próxima **renderização**.



Veja esta correção



Veja documentação

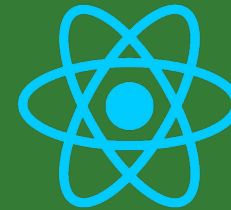


CardImcFix1.jsx X

```
9      const incrementaPeso = () => {
10        let _peso = peso + 1
11        setPeso(_peso);
12        setImc(_peso / alt ** 2);
13      };
14
15      const decrementaPeso = () => {
16        let _peso = peso - 1
17        setPeso(_peso);
18        setImc(_peso / alt ** 2);
19      };
20
```

Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useState()

- O hook useState também aceita uma **função como argumento**, onde podemos receber como parâmetro o valor **antigo** do **state**.
- Então atualizar o seu valor internamente e **retornar o valor atualizado**. Veja ao lado como ficaria a correção passando uma função como argumento do **setPeso**.

🌟 [Veja esta correção](#)

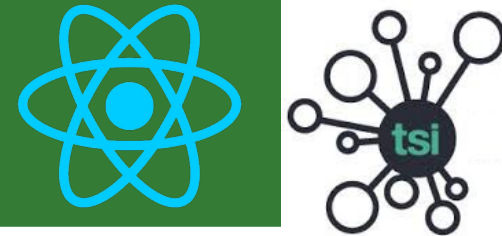
🌟 [Veja documentação](#)

🌟 CardImcFix2.jsx ×

```
9      const incrementaPeso = () => {
10        setPeso((peso) => {
11          let novo_peso = peso + 1;
12          setImc(novo_peso / alt ** 2);
13          return novo_peso;
14        });
15      };
16
17      const decrementaPeso = () => {
18        setPeso((peso) => {
19          let novo_peso = peso - 1;
20          setImc(novo_peso / alt ** 2);
21          return novo_peso;
22        });
23      };
24
```

Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useEffect() → Side Effects ou Efeitos Colaterais

- Executa no primeiro render(), **componentDidMount**, e depois cada vez que um estado **observado** é atualizado, **componentDidUpdate**.

useEffect(**callback**, **dependências**)

 **Veja o exemplo da Ada**

AdaClass.jsx X

...

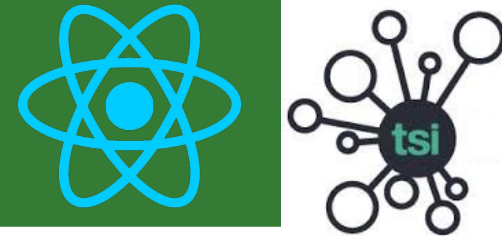
Ada.jsx X

```
17 //useEffect
18 componentDidMount() {
19   | console.log(this.state.countRender, 'render');
20   | }
21
22 //useEffect
23 componentDidUpdate(prevProps, prevState) {
24   | if (this.state.data !== prevState.data) {
25   |   | console.log(this.state.countRender, 'render');
26   |   | }
27   | }
28
```

```
18 useEffect(() => {
19   | //componentDidMount && componentDidUpdate
20   | console.log(countRender, 'render');
21   | return () => {
22   |   | //componentWillUnmount
23   |   | console.log('Ada será removida!!');
24   |   | };
25   | }, [data]); //observa mudanças em data
26
27
28
29
```


Hooks useState e useEffect


CONCEITOS PRINCIPAIS



Hook useEffect() → Side Effects ou Efeitos Colaterais

- **Callback ()=>{}:** Função a ser executada a cada modificação do estado observado.
- **Dependências []:** Lista de **estados** a serem **observados**, passados em um **array []**.

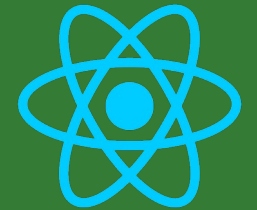
 [Veja o exemplo da Ada](#)

 Ada.jsx ×

```
18  useEffect(() => { //callback -> componentDidMount && componentDidUpdate
19    |   console.log(countRender, 'render');
20    |   return () => { //componentWillUnmount
21    |     |   console.log('Ada será removida!!');
22    |   };
23  }, [data] ); // dependência -> observa mudanças em data
```

Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useEffect()

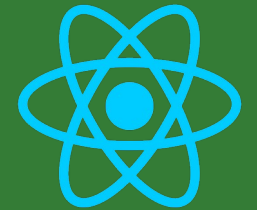
- Se a função passada como primeiro parâmetro **retornar outra função**, esta será executada a cada vez que o componente é **desmontado** (**componentWillUnmount**).
- Ela é executada em atualizações, pois o componente é removido e inserido novamente ao DOM.

🔗 Reveja o exemplo da Ada

```
AdaClass.jsx x  Ada.jsx x
29 //função retornada no useEffect
30 componentWillUnmount() {
31   console.log('Ada será removida!!');
32 }
33
34
35
36
18 useEffect(() => { //callback -> componentDidMount &&
   componentDidUpdate
19   console.log(countRender, 'render');
20   return () => { //componentWillUnmount
21     console.log('Ada será removida!!');
22   };
23 }, [data] ); // dependência -> observa mudanças em data
```

Hooks useState e useEffect

CONCEITOS PRINCIPAIS



Hook useEffect()

- Diferentes casos na passagem do segundo parâmetro **array**:
 - ▶ Se passar um **array vazio []**, **não será executado na atualização** dos estados, **componentDidUpdate**, portanto, executará **uma vez apenas (componentDidMount)**.
 - ▶ Se não passar nenhum parâmetro (**null**), fará com que ele execute a cada renderização.
 - ▶ A **recomendação** é sempre procurar passar as **dependências**, ou seja, os **estados a serem monitorados** no parâmetro **array**.

🔗 ***Veja o exemplo 1***

🔗 ***Veja o exemplo 2***

Hooks Flow

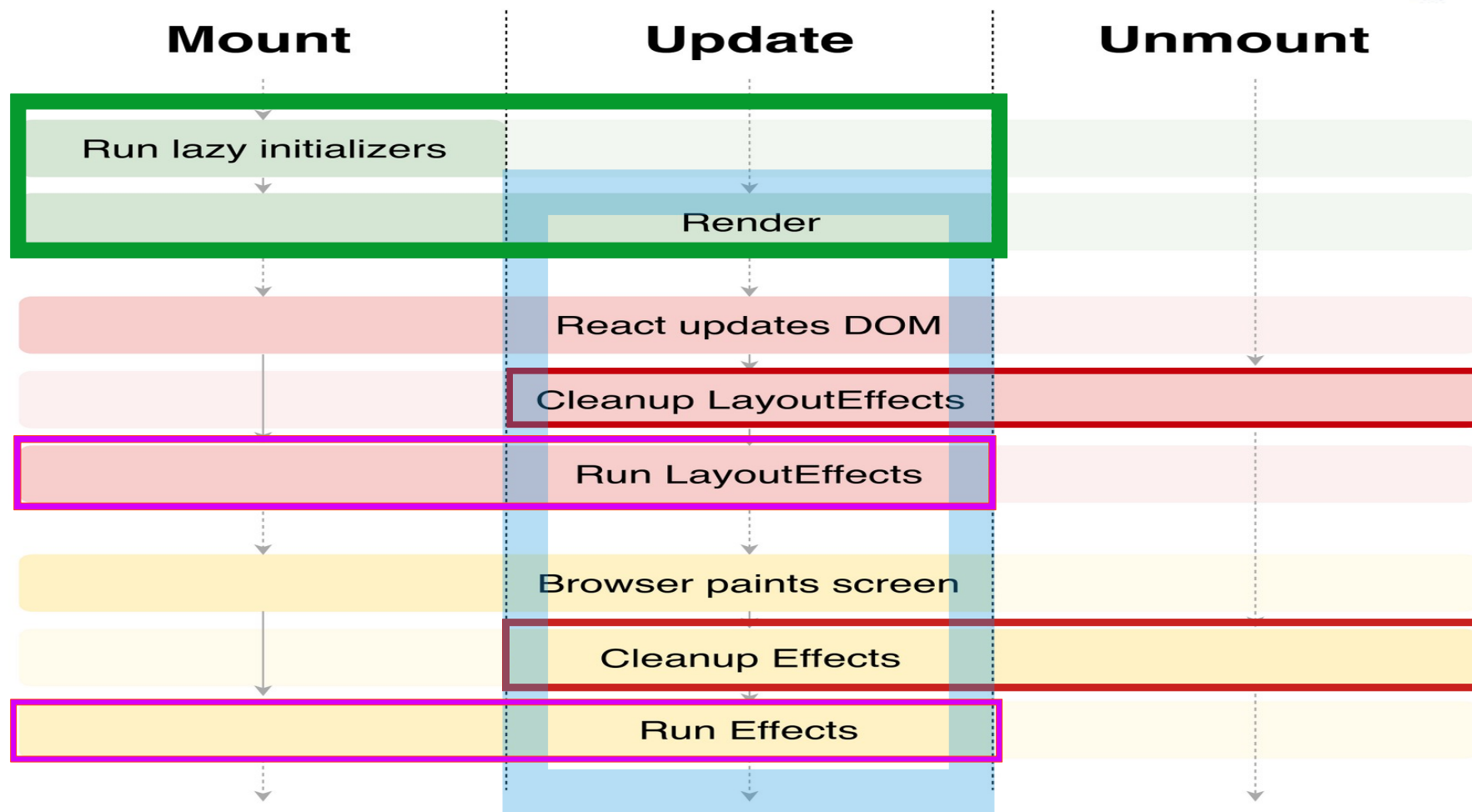
ORDEM DOS HOOKS:



React Hook Flow Diagram

v1.3.1 github.com/donavon/hook-flow

- **Mount: useState**
 - Inicialização
 - Callback
- **useEffects**
 - Callback
 - Side Effects
- **Update: useEffects**
 - Callback
 - State change
- **Unmount**
 - Returned Callback



Fonte: <https://github.com/donavon/hook-flow/blob/master/hook-flow.pdf>

Notes:

1. Updates are caused by a parent re-render, state change, or context change.
2. Lazy initializers are functions parent passed to useState and useReducer.

Veja o exemplo 3

Desenvolvimento Front-End II

Atividades



• Atividades

- **1)** Atualizar o exemplo do **CardImc** para alterar o estilo do **card** de acordo com o valor do IMC, como no exemplo do componente **ImcPessoa** abaixo:

- **Verde:** ≤ 24.5
- **Amarelo:** $> 24.5 < 30$
- **Vermelho:** ≥ 30
- Adicione botões de + e – ao lado do peso, cada vez que **atualizar o peso**, a cor do componente deve **mudar** de acordo com o **imc calculado**.
 - Crie um estado para a cor.
 - Utilize **useEffect** para atualizar a cor toda vez que o **IMC** for alterado.

```
JS (Babel)
19  <>
20  <ImcPessoa pessoa={{name: 'Fulano', alt: 1.7, peso: 80}} />
21  <ImcPessoa pessoa={{name: 'Mengano', alt: 1.9, peso: 109}} />
22  <ImcPessoa pessoa={{name: 'Esbelto', alt: 1.8, peso: 70}} />
23  </>
```

Fulano:	Mengano:	Esbelto:
Altura: 1.7 m	Altura: 1.9 m	Altura: 1.8 m
Peso: 80	Peso: 109	Peso: 70
IMC: 27.68	IMC: 30.19	IMC: 21.60



- **Atividades**

- **2)** Desenvolver um app ***ToDoList*** (Lista de Tarefas) utilizando os Hooks **useState** e **useEffect**:

- **Requisitos:**

- As tarefas deverão ser guardadas no ***localStorage***
- Cada tarefa deverá ter um lista de passos, etapas a serem cumpridas.
- Deverá ser possível criar e remover tarefas, assim como passos de cada tarefa.
- Deverá ser possível marcar os passos de cada tarefa como realizado, neste caso, mudar a aparência de passos realizados.
- Tarefas que tiverem todos os passos realizados deverão também mudar de aparência, sendo marcadas como cumpridas.



- **Atividades**

- **3)** Pesquise o Hook ***useRef*** e refatore o seu app ***TodoList*** utilizando este hook.
 - Links úteis:
 - <https://react.dev/reference/react/useRef>
 - https://www.w3schools.com/react/react_useref.asp
 - https://www.youtube.com/watch?v=BwRxBGsT_f0
- Entregue as atividades em três diferentes projetos criados com o vite com o template react.
- Para enviar ao moodle, compacte os três projetos em uma única pasta, mas remova a ***node_modules*** de cada projeto.



- **REFERÊNCIAS**

SILVA, Maurício Samy. **React Aprenda Praticando**. São Paulo: Novatec, 2021.

Quick Start – React: 2022 Meta Platforms, Inc. Disponível em: <https://react.dev/learn>. Acesso em: 17/10/2024.

SILVA, Maurício Samy. **CSS3 – Desenvolva aplicações web profissionais com o uso dos poderosos recursos de estilização das CSS3**. São Paulo: Novatec, 2011.

SILVA, Maurício Samy. **HTML5 – A linguagem de marcação que revolucionou a web**. São Paulo: Novatec, 2011.

SILVA, Maurício Samy. **JavaScript: guia do programador**. São Paulo: Novatec, 2010.