FIAP

GRADUAÇÃO

45697056

TDS

Responsive Web Development

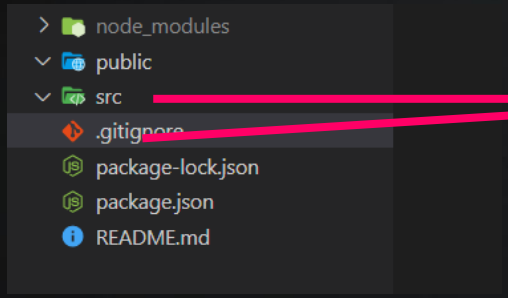
Prof. Alexandre Carlos profalexandre.jesus@fiap.com.br

Prof. Luís Carlos lsilva@fiap.com.br

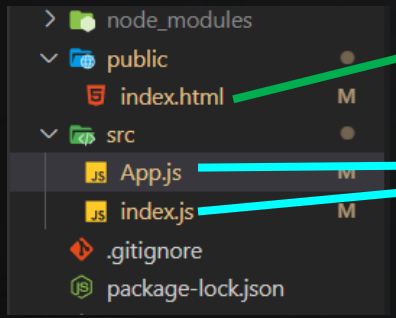
Novo projeto c/ pasta src e public do zero

45697056

Para iniciarmos nossos projeto da aula de hoje, vamos criar um novo projeto chamado react-aula5 e vamos apagar todos os arquivos das pastas src e public.



Apagamos todos os arquivos da pasta src e public



Logo após, na pasta public, recrie os arquivos index.html contendo a div c/ id='root'

E na pasta src os arquivos index.js com o ReactDOM.render() e o arquivo App.js como nosso componente principal.

Ciclo de Vida de Componentes de Classe

Para começarmos a entender o ciclo de vida de um componente vamos começar criando uma nova pasta chamada “componentes” e nela um arquivo chamado “CompClasse.js”, e insira o código ao lado.

45697056

src > componentes > CompClasse.js > CompClasse

```
1   import React, { Component } from 'react'
2
3
4   export default class CompClasse extends Component{
5
6       constructor(props){
7           super(props)
8           this.state={
9               valor: props.valor,
10          }
11      }
12
13      aumentarValor(){
14          this.setState({valor: this.state.valor + 1})
15      }
16
17      render(){
18          return(
19              <>
20                  <h2>Compomente Classe</h2>
21                  <p>Valor Classe: {this.state.valor}</p>
22                  <button onClick={()=>this.aumentarValor()}>Valor Classe + 1</button>
23              </>
24          )
25      }
26  }
```

Ciclo de Vida de Componentes de Classe

45697056

O componente de classe tem um método de ciclo de vida que, sempre que o componente ser criado (Montado), após o método render() ele é executado, o nome dele é: componenteDidMount()

```
13      aumentarValor(){
14          this.setState({valor: this.state.valor + 1})
15      }
16
17      componentDidMount(){
18          console.log("Opa, acabei de ser criado! :)");
19      }
20
21      render(){
```

Ciclo de Vida de Componentes de Classe

45697056

Assim que o elemento recebe uma atualização temos outro método que pode ser chamado após o render() ser executado, o nome dele é: componenteDidUpdate()

```
12  aumentarValor() {
13    this.setState({ valor: this.state.valor + 1 });
14  }
15
16  componentDidMount() {
17    console.log("Opa, acabei de ser criado! :)");
18  }
19
20  componentDidUpdate() {
21    console.log("alteraram algo em mim! :o");
22  }
23
24  render() {
```

Ciclo de Vida de Componentes de Classe

Para completar nosso ciclo de vida temos um método que é chamado o componente ser removido, o nome dele é: componenteWillUnmount()

```
16    componentDidMount() {
17      console.log("Opa, acabei de ser criado! :)");
18    }
19
20    componentDidUpdate() {
21      console.log("alteraram algo em mim! :o");
22    }
23
24    componentWillUnmount(){
25      console.log("Poxa, me removeram.... :(");
26    }
27
28    render() {
```

Mas calma, precisamos criar uma função de fora do componente para excluí-lo.

Ciclo de Vida de Componentes de Classe

Para fazermos o teste de remoção do elemento, vamos fazer os seguintes ajustes na nosso componente App

src > App.js > ...

```
1  import React, {useState}from 'react'
2  import CompClasse from './componentes/CompClasse'
3
4  export default () =>{
5
6      const [classe, setClasse] = useState(false)
7      const criarClasse = ()=>{
8          setClasse(!classe)
9      }
10
11     return(
12         <>
13             <h1>Ciclos de Vida</h1>
14             {classe === true ? <CompClasse valor={0}/> : ""}
15             <button onClick={()=>criarClasse()}>
16                 {classe === true ? "Remover Classe" : "Criar Classe"}
17             </button>
18         </>
19     )
20 }
```

Ciclo de Vida de Componentes de Função

45697056

Agora que já conhecemos um pouco sobre as classes e seu ciclo de vida, vamos ver como podemos trabalhar com o ciclo de vida de nossas funções. Dentro da pasta componentes, crie um componente chamado CompFuncao.js.

src > componentes > CompFuncao.js > ...

```
1   import React, { useState } from 'react'
2
3   export default props => {
4
5       const [valor, setValor] = useState(props.valor)
6
7       return(
8           <>
9               <h2>Componente de Função</h2>
10              <p>Valor Função: {valor}</p>
11              <button onClick={()=>setValor(valor + 1)}>Aumentar Valor</button>
12          </>
13      )
14  }
```

Não se esqueça em chamar no App passando um atributo: valor={0}

Ciclo de Vida de Componentes de Função (useEffect)

Os métodos de ciclo de vida que vimos na classe não servem para um componente de função, mas para isso temos o Hook chamado useEffect( ). Vamos inserir ele em nosso código.

```
4
5       const [valor, setValor] = useState(props.valor)
6
7       useEffect(()=>{
8           console.log("É só chamar que eu vou! :P");
9       })
10
11      return(
12          <>
13              <h2>Componente de Função</h2>
14              <p>Valor Função: {valor}</p>
15              <button onClick={()=>setValor(valor + 1)}>Aumentar Valor</button>
16          </>
17      )
18  }
```

Repare que ele é chamado sempre que temos uma ação no componente.

Ciclo de Vida de Componentes de Função (useEffect)

O método useEffect( ) na verdade espera dois parâmetros, o primeiro é uma função e o segundo é um array que vai dizer quando ele deve ser chamado. Para que ele execute quando o componente é criado usamos um array vazio.

```
5     const [valor, setValor] = useState(props.valor)
6
7     useEffect(()=>{
8         console.log("É só chamar que eu vou! :P");
9     })
10
11    useEffect(()=>{
12        console.log("Acabei de ser criado! :)");
13    },[])
14
15    return(
```

Ciclo de Vida de Componentes de Função (useEffect)

Podemos usar o useEffect( ) para executar tarefas quando um componente é atualizado, para isso precisamos apresentar no array que ele recebe o valor que ele deve ficar “escutando”, aguardando a mudança.

```
7     useEffect(()=>{
8         console.log("É só chamar que eu vou! :P");
9     })
10
11    useEffect(()=>{
12        console.log("Acabei de ser criado! :)");
13    },[])
14
15    useEffect(() => {
16      console.log("Alteraram algo em mim! :o");
17    }, [valor]);
18
19    return(
```

Ciclo de Vida de Componentes de Função (useEffect)

Para conseguirmos trabalhar com a remoção de nosso componente, devemos passar uma nova função no retorno da função de callback, assim ela executará antes do elemento ser removido.

```
14
15    useEffect(() => {
16      console.log("Alteraram algo em mim! :o");
17    }, [valor]);
18
19    useEffect(() => {
20        return () => console.log("Poxa, me removeram! :(");
21    }, []);
22
23
24    return(
```

Para removermos o componente devemos fazer isso de dentro do componente pai.

Ciclo de Vida de Componentes de Função (useEffect)

Aqui no componente App vamos fazer um state e um botão para criarmos uma criação condicional do CompFuncao, como fizemos com o da classe.

```
7     const [classe, setClasse] = useState(false)
8     const criarClasse = ()=>{
9         setClasse(!classe)
10    }
11    const [funcao, setFuncao] = useState(false)
12
13
14    return (
15      <>
16        <h1>Ciclos de Vida</h1>
17        {classe === true ? <CompClasse valor={0} /> : ""}
18        <button onClick={() => criarClasse()}>
19          {classe === true ? "Remover Classe" : "Criar Classe"}
20        </button>
21        <br/>
22        {funcao === true ? <CompFuncao valor={0} /> : ""}
23        <button onClick={() => setFuncao(!funcao)}>
24          {funcao === true ? "Remover Função" : "Criar Função"}
25        </button>
26      </>
27    );
```

Exercício

45697056

Crie um novo projeto chamado exercicio5-1, limpe todos os arquivos das pastas public e src, após isso recrie o arquivo index.html na pasta public e os arquivos index.js e App.js na pasta src.

Crie uma pasta chamada componentes e dentro um arquivo chamado Aviao.js, ele deve ter um h2 o identificando e um state chamado altura, que deve começar em “0” e apresentar a altura em um parágrafo.

Você deve criar um botão que quando clicado aumente a altura em 100.

Após a primeira parte apresente no console, usando o useEffect, mensagens avisando que “o avião decolou” (quando o componente for criado), “O avião está em XXX pés”(quando ele subir através do botão) e “O avião foi derrubado”(quando o componente for removido).

Utilizando Styled Components

45697056

O Styled Component é uma biblioteca de estilização para agilizar a resolver alguns probleminhas de estilização nos componentes do React. Para instalar, vamos abrir um novo terminal e digitar “npm install --save styled-componentes”

```
Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS D:\_React\aula5-react> npm install --save styled-components
npm WARN @babel/plugin-bugfix-v8-spread-parameters-in-optional-chaining@7.14.5 requires a peer
of @babel/core@^7.13.0 but none is installed. You must install peer dependencies yourself.
npm WARN tsutils@3.21.0 requires a peer of typescript@>=2.8.0 || >= 3.2.0-dev || >= 3.3.0-dev |
| >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev || >= 3.7.0-bet
a but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {
```

Utilizando Styled Components

45697056

Mas antes de começarmos a usar, vamos entender um possível problema. Crie um componente chamado Componente1.js na pasta componentes:

src > componentes > JS Componente1.js > ...

```
1   import React from 'react'
2   import './Componente1.css'
3
4   export default ()=>{
5
6       return(
7           <div>
8               <p>Teste</p>
9           </div>
10      )
11  }
```

Utilizando Styled Components

45697056

Como já deve ter reparado no arquivo que acabamos de criar, estamos importando um arquivo chamado Componente1.css, certo? Então vamos criá-lo:

src > componentes > Componente1.css > ...

```css
div{
    border: 3px solid blue;
    padding: 20px;
}

p{
    color: blue;
}
```

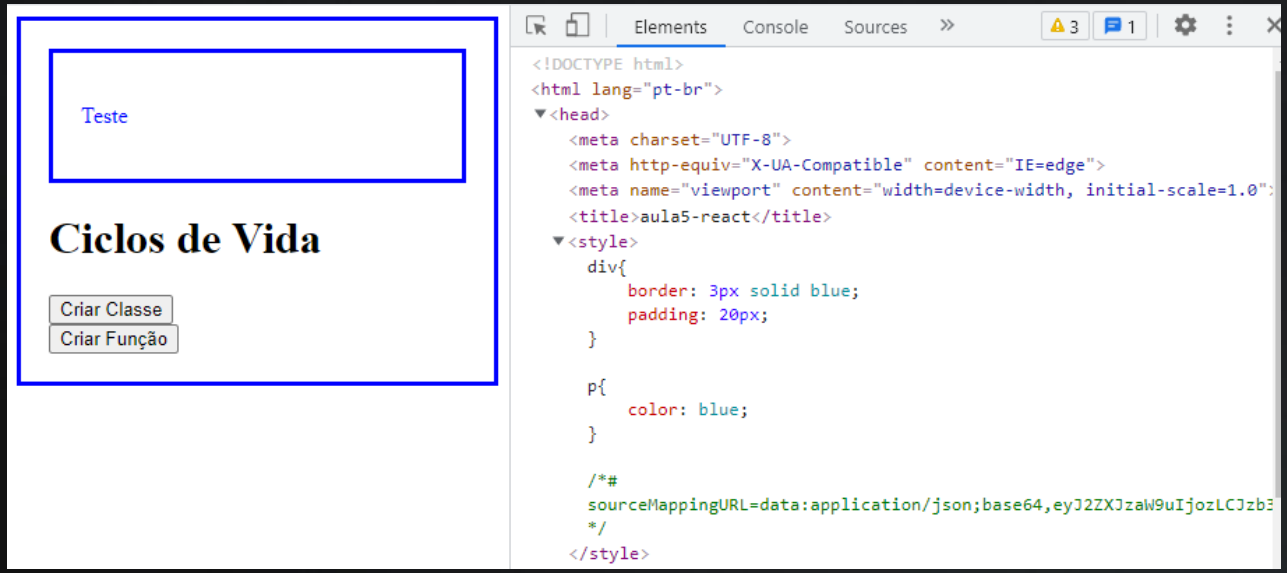
Ah! Não se esqueça em chamar o Componente1.js no App.js,

```
14
15      return (
16        <>
17          <Componente1/>
18          <h1>Ciclos de Vida</h1>
19          {classe === true ? <CompClasse
20          <button onClick={() => criarCla
```

Utilizando Styled Components

45697056

O resultado não será como esperamos, perceba que a nossa div ‘root’ também recebeu a estilização, isso porque o react renderiza o css em uma tag style.



Vamos ver como podemos resolver isso usando o styled-component

Utilizando Styled Components

45697056

A primeira coisa que devemos fazer é criar nosso arquivo para criarmos as estilizações, vamos cria-lo na pasta src mesmo e vamos chamar de styled.js.

src > styled.js

```
1  import styled from 'styled-components'
2
3
4
5
```

Devemos fazer o import do styled-components

Utilizando Styled Components

45697056

No arquivo Componente1.js vamos transformar nossa div em um novo componente chamado DivComp1 e importa-lo de nosso arquivo styled.js. Calma ainda vamos criar ele.

Importamos ele e usamos no lugar da div

src > componentes > Componente1.js > ...

```
1   import React from 'react'
2   import './Componente1.css'
3
4   import { DivComp1 } from '../styled'
5
6   export default ()=>{
7
8       return(
9           <DivComp1>
10              <p>Teste</p>
11          </DivComp1>
12      )
13  }
```

Utilizando Styled Components

45697056

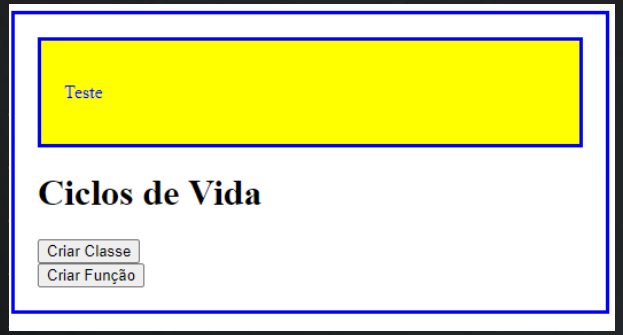
Vamos fazer um teste para deixar o fundo da div na cor amarela, volte para o arquivo styled.js e crie o código abaixo:

src > styled.js > ...

```
1  import styled from 'styled-components'
2
3
4  export const DivComp1 = styled.div`
5      background-color : yellow;
6  `
```

Repare que criamos e exportamos nosso novo elemento.

Cuidado, a estilização deve estar entre crases.



Utilizando Styled Components

Agora vamos arrumar nosso problema atual passando toda a nossa estilização para o nosso novo elemento:

src > styled.js > ...

```
1   import styled from 'styled-components'
2
3
4   export const DivComp1 = styled.div`
5     background-color: yellow;
6     border: 3px solid blue;
7     padding: 20px;
8
9     p{
10      color: blue;
11    }
12  `;
```

Não esqueça de comentar a importação do css no Componente1.js

src > componentes > Componente1.js > ...

```
1   import React from 'react'
2   //import './Componente1.css'
3
```

Utilizando Styled Components

45697056

src > componentes > Componente2.js > ...

```
1  import React from 'react'
2  import styled from 'styled-components'
3
4  const DivComp2 = styled.div`
5      border: 3px solid green;
6      padding: 20px;
7      background-color: #99ff99;
8
9      h2{ color: purple; }
10
11     p{  color: orange;  }
12 `
13
14 export default ()=>{
15
16     return(
17         <DivComp2>
18             <h2>Estilização Componente 2</h2>
19             <p>Desta vez dentro do componente.</p>
20         </DivComp2>
21     )
22 }
```

Nós podemos também criar nossos componentes de estilização na própria página. Vamos criar o Componente2.js e criar nosso componente dentro dele:

Não esqueça de chamar o componente2.js no App.js

Utilizando Styled Components

Observe o resultado final, ele cria um controle interno em classes para garantir que não teremos problemas como aqueles do início.

Exercício

45697056

Crie um novo projeto chamado exercicio5-2, limpe todos os arquivos das pastas public e src, após isso recrie o arquivo index.html na pasta public e os arquivos index.js e App.js na pasta src.

Pensando em um tema de sorveteria, crie um componente Cabecalho.js, dentro dele um h1 e um parágrafo.

Logo após crie um componente chamado Corpo.js, dentro dele um h2 e uma lista de sabores, outro h2 e uma lista de acompanhamentos.

Por fim crie um componente chamado Rodape.js, dentro um parágrafo com o endereço da sorveteria.

Após todos os componentes criados, use um styled-componentes externo para o cabeçalho e rodapé e um interno para o corpo.

DUVIDAS