



# FIAP

GRADUAÇÃO

45697056



TDS

# Responsive Web Development

Prof. Alexandre Carlos [profalexandre.jesus@fiap.com.br](mailto:profalexandre.jesus@fiap.com.br)

Prof. Luís Carlos [lsilva@fiap.com.br](mailto:lsilva@fiap.com.br)



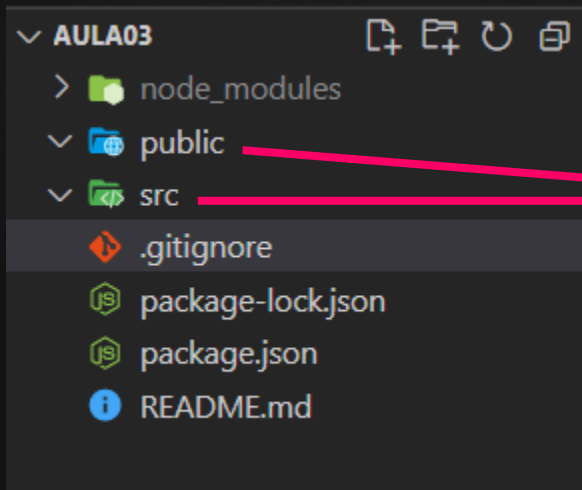


## Novo projeto c/ pasta src e public do zero

45697056



Para iniciarmos nossos projeto da aula de hoje, vamos criar um novo projeto chamado `react-aula3` e vamos **apagar** todos os arquivos das pastas `src` e `public`.



Apagamos todos  
os arquivos da  
pasta src e public



React

# Novo projeto c/ pasta src e public do zero

45697056



Na pasta `public` devemos criar novamente o arquivo `index.html`, não podemos esquecer de criar uma div com o `id= 'root'`.

The screenshot shows the VS Code interface. On the left, the 'EXPLORADOR' (Explorer) sidebar is open, showing the project structure. Under 'EDITORES ABERTOS' (Open Editors), 'inde.html public' is listed. In the 'AULA03' folder, the 'public' folder is expanded, showing 'inde.html' selected. Below it, the 'src' folder is also visible, containing files like '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The main editor area shows the content of 'inde.html' in the 'public' folder. The code is as follows:

```
public > inde.html > ...
1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Aula 03</title>
8  </head>
9  <body>
10   <div id='root'></div>
11 </body>
12 </html>
```



## Novo projeto c/ pasta src e public do zero

45697056  
■ ■ ■

O próximo passo será recriar o arquivo `index.js` na pasta `src`. Insira o código abaixo para que os componentes que vamos criar a seguir possam ser renderizados no arquivo `index.html`.

JS index.js M X

src > JS index.js

```
1 import React from 'react'
2 import ReactDOM from 'react-dom'
3
4 ReactDOM.render(
5   <h1>Conteúdo de Index.js</h1>,
6   document.getElementById('root')
7 )
8
```

Nos permite usar o JSX

Nos permite usarmos o VDOM

Método para renderizar os componentes na tela

Conteúdo que será renderizado

Local onde será renderizado

Repare em nossa página no navegador, agora só temos a tag `h1`.



React

## Novo projeto c/ pasta src e public do zero

45697056

■ ■ ■

Por boa prática, vamos criar um novo componente chamado `App.js`, ele será o nosso componente principal. A princípio vamos apenas colocar um `h1` dentro, repita o código abaixo:

```
JS index.js M JS App.js M X
src > JS App.js > ...
1 | import React from 'react'
2 |
3 | export default () =>{
4 |
5 |     return(
6 |         <h1>Conteúdo de App.js</h1>
7 |     )
8 | }
```

Podemos usar uma arrow function para deixar o código mais leve.



React

## Novo projeto c/ pasta src e public do zero

45697056



Para o nosso componente App ser reproduzido na tela agora temos que inserir ele no index.js.

JS index.js M X

JS App.js M

src > JS index.js

```
1 import React from 'react'
2 import ReactDOM from 'react-dom'
3 import App from './App'
4
5 ReactDOM.render(<App/>, document.getElementById('root'))
6
```

Repare que agora conseguimos até deixar em apenas uma linha.



# Usando arquivos JSX

45697056  
■ ■ ■

Quando criamos componentes inserimos no retorno das funções uma linguagem muito parecida com HTML, ela é chamada de JSX ou **Javascript XML**. As tags que utilizamos na verdade são de XML e não HTML, por isso percebemos algumas diferenças como fechar uma tag de quebra de linha com a barra no final ou usar o atributo `className`.

Percebemos que os arquivos JS aceitam sem problema nenhum a linguagem JSX, mas para organizar nosso código é bom criarmos nossos componentes com a extensão JSX. O único arquivo que devemos tomar uma atenção especial é o App, pois se importamos um arquivo `App.jsx` no arquivo `index.js` apenas colocando o caminho  `'./App'`, ele vai procurar o `App.js` e não o `App.jsx`.

Aconselho a deixar o App com a extensão `js` e os demais componentes com a extensão `jsx`.







React

# Usando arquivos JSX

45697056

■ ■ ■

Crie uma pasta chamada componentes e dentro dela um arquivo chamado Componente1.jsx.

```
Componente1.jsx U X JS App.js M
src > componentes > Componente1.jsx > default
1  import React from 'react'
2
3  export default ()=>{
4
5
6      return(
7          <>
8              <h2>Componente 1</h2>
9          </>
10     )
11 }
```

Você vai perceber que em alguns editores de código, como o VSCode, com a extensão jsx ele reconhece os elementos de forma mais fácil.



React

## Trabalhando com props.children

45697056

■ ■ ■

Podemos também passar elementos de pai para filho, para isso utilizamos o `props.children`, desta forma podemos carregar elementos de dentro do pai para o componente filho. Crie na pasta componentes, um arquivo jsx chamado `Componente2`.

Componente1.jsx U

Componente2.jsx U X

App.js M

src > componentes > Componente2.jsx > ...

```
1  import React from 'react'
2
3  export default props=>{
4
5      return(
6          <>
7          <h2>Componente 2</h2>
8          {props.children}
9          </>
10     )
11 }
```

Repare que estamos preparando uma  
receber um `props.children` logo abaixo  
do `h2`.



React

## Trabalhando com props.children

45697056

■ ■ ■

Para em Componente 1 para definirmos quais elementos serão passados para o Componente2 devemos usá-lo com tag de abertura e fechamento, assim definimos o conteúdo de `props.children`.

```
Componente1.jsx U x Componente2.jsx U js App.js M
src > componentes > Componente1.jsx > ...
1  import React from 'react'
2  import Componente2 from './Componente2'
3
4
5  export default ()=>{
6
7      return(
8          <>
9              <h1>Componente 1</h1>
10             <Componente2>
11                 <p>Esse texto vem do Componente 1</p>
12             </Componente2>
13         </>
14     )
15 }
```

Chame o Componente2 utilizando tag de abertura e fechamento. Assim você poderá colocar o conteúdo dentro.



React

# Renderizar componente de forma Condicional

Assim como usamos a lógica no javascript para definir valores, também podemos usar para decidir sobre a utilização de componentes na montagem de nossas páginas. Vamos utilizar a expressão ternária como exemplo, mas poderia ser qualquer outra. Aplique o código ao lado no Componente1:

```
45697056
■ ■ ■

Componente1.jsx U x Componente2.jsx U JS App.js M

src > componentes > Componente1.jsx > ...
1  import React, { useState } from 'react'
2  import Componente2 from './Componente2'
3
4
5  export default ()=>{
6
7      const [num,setNum]=useState(0)
8
9      return(
10         <>
11             <h1>Componente 1</h1>
12             <Componente2>
13                 <p>0 número {num} é:
14                 { //No caso de valores nos elementos:
15                 num % 2 == 0 ? 'par' : 'impar' }</p>
16                 { //No caso de um elemento:
17                 num % 2 == 0 ?
18                 <p>Este número é par!!!</p> :
19                 <p>Este número é impar!!!</p>
20                 }
21             </Componente2>
22             <button onClick={()=>setNum(num + 1)}>Num + 1</button>
23         </>
24     )
25 }
```



React

# Utilizando o método map() para criar elementos

Uma forma bem prática de criar elementos é utilizando o método map. Podemos ter um array com nome ou valores e manipular adicionando eles a tags HTML ou manipulando seus valores.

```
Componente1.jsx U  Componente2.jsx U x  JS App.js M

src > componentes > Componente2.jsx > ...
1  import React from 'react'
2
3  export default props=>{
4
5      const frutas=['Maça','Banana','Pera','Uva','Goiaba']
6
7
8      return(
9          <>
10             <h2>Componente 2</h2>
11             <ul>
12                 {frutas.map((f)=><li>{f+ ' fresquinha'}</li>)}
13             </ul>
14             {props.children}
15
16             </>
17         )
18     }
```



React

# Utilizando o método map() para criar elementos

Se precisarmos de uma referência dos valores criados com map podemos utilizar o valor do índice ou posição dos valores. Vamos ver como ela funciona:

```
Componente1.jsx U  Componente2.jsx U x  App.js M

src > componentes > Componente2.jsx > ...
1  import React from 'react'
2
3  export default props=>{
4
5      const frutas=['Maça','Banana','Pera','Uva','Goiaba']
6
7
8      return(
9          <>
10             <h2>Componente 2</h2>
11             <ul>
12                 {frutas.map((f,ind)=><li>{f+` é a fruta nº ${ind +1}`}</li>)}
13             </ul>
14             {props.children}
15
16             </>
17         )
18     }
```



React

# Usando eventos para capturar valores de campos

45697056  
■ ■ ■

Vamos ver como é possível alterar o valor de um state através de campos. Crie um componente chamado NovoValor.jsx e insira o código abaixo:

```
Componente1.jsx U  NovoValor.jsx U X  Componente2.jsx U  App.js M
src > componentes > NovoValor.jsx > ...
1  import React, { useState } from 'react'
2
3  export default ()=>{
4
5      const [frase,setFrase]=useState('')
6
7      return(
8          <>
9              <h2>Vamos escrever uma frase:</h2>
10             <label>Motivação:</label>
11             <input type="text" onChange={(e)=>setFrase(e.target.value)}/>
12             <p>Motivação do dia: {frase}</p>
13         </>
14     )
15 }
16
```

Cuidado ao pegar o valor do evento

**OBS.** Não esqueça de depois chamar ele em App.



React

# Usando eventos para capturar valores de campos

45697056

■ ■ ■

Podemos até guardar vários valores no mesmo state, para isso podemos ter um objeto. Crie um componente chamado NovosValores.jsx e digite o código abaixo:

```
import React, { useState } from 'react'

export default ()=>{
  const [carro,setCarro]=useState({'marca':'','modelo':''})
  const mostrarCarro = (e)=>{
    const {name, value} = e.target;
    if(name == 'marca'){
      setCarro({'marca': value, 'modelo': carro.modelo})
    }else if(name == 'modelo'){
      setCarro({'marca': carro.marca, 'modelo': value})
    }
  }

  return(
    <>
      <h2>Dados do Carro</h2>
      Marca: <input name="marca" onChange={mostrarCarro}/> <br />
      Modelo: <input name="modelo" onChange={mostrarCarro}/> <br />
      <p>Meu carro é um {carro.marca} {carro.modelo}</p>
    </>
  )
}
```

Como o elemento é um objeto é possível fazer uma desestruturação

Analisa qual evento está sendo chamado para pegar o valor certo

**OBS.** Não esqueça de depois chamar ele em App.





## Exercício (5 pontos no 1ºCheckpoint)

45697056



- 1 – Crie uma pasta chamada exercicio3 e nela crie um projeto React.
- 2 – Apague todo o conteúdo das pastas public e src e crie os arquivos index.html, index.js e App.js, deixando um H1 escrito “Seu nome e rm e turma”.
- 3 – Crie um componente chamado Semaforo.jsx contendo uma div e dentro 3 elementos h2 com as palavras pare, atenção e siga e um botão que cada vez que for clicado troque a exibição deles. Deve ser mostrado 1 de cada vez.
- 4 - Aplique as cores do semáforo aos respectivos elementos.
- 5 – Crie um elemento chamado Carros que deverá receber uma lista com 10 carros do elemento App via props.children.
- 6 – Crie um componente chamado carteirinha, ele deverá receber uma foto, nome , rm, curso e turma de App. Apresente 3 carteirinhas devidamente formatada na tela e usando o componente.
- 7 – Crie a 4ª carteirinha, mas desta vez ela só deve vir com as foto, os demais dados devem vir de campos para preencher como no último exemplo.

**OBS.** Deve ser enviado um arquivo compactado contendo apenas a pasta public e src do projeto.

# DUVIDAS





Copyright © 2015 - 2021 Prof. Luís Carlos S. Silva  
Prof. Alexandre Carlos de Jesus

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).