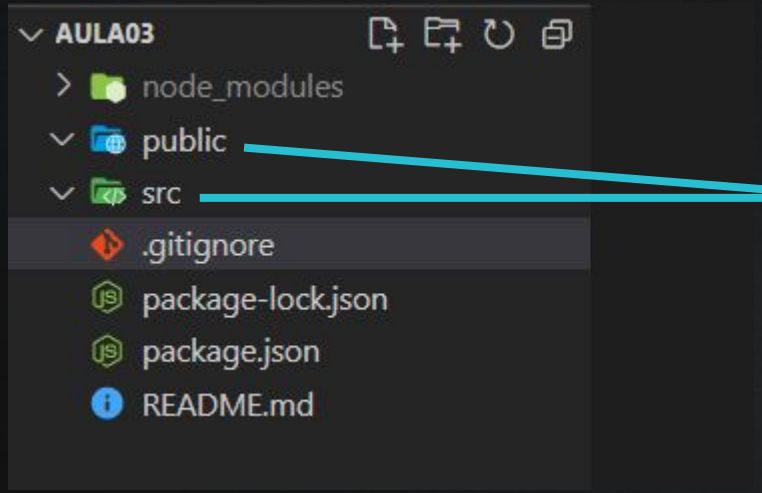
# Novo projeto c/ pasta src  e public do zero

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.

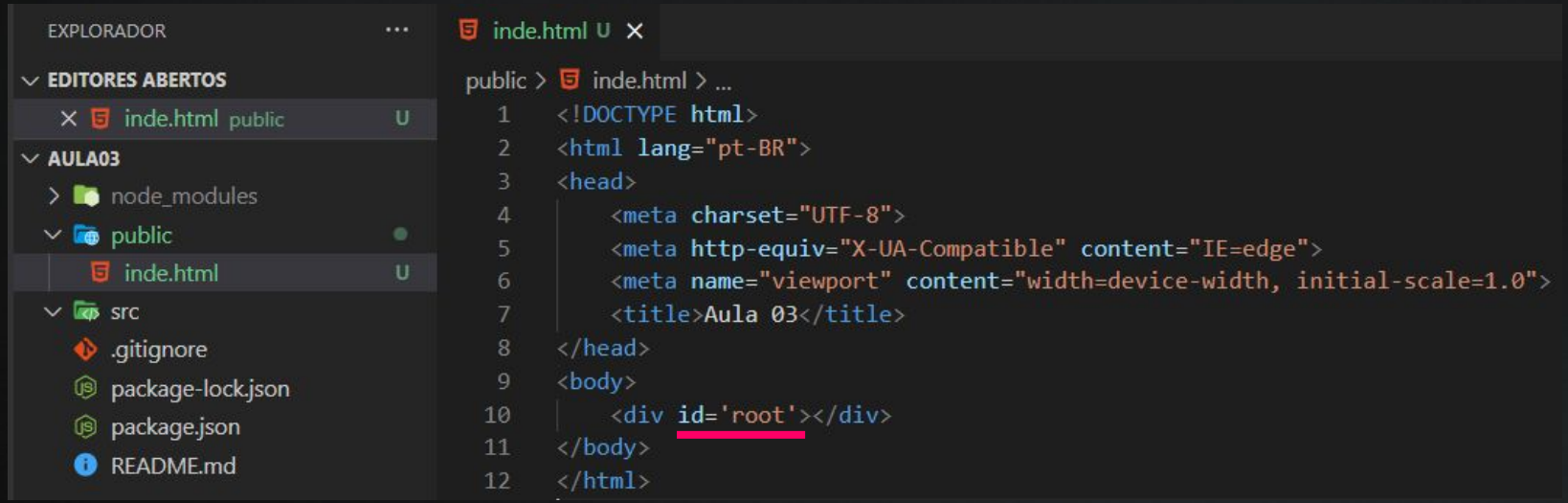



Apagamos todos os arquivos da pasta src e public

# Novo projeto c/ pasta src  e public do zero

React

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



# Novo projeto c/ pasta src e public do zero

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

index.js M

src > 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.

# Novo projeto c/ pasta src e public do zero

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:

index.js M | App.js M

src > 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.


# Novo projeto c/ pasta src e public do zero

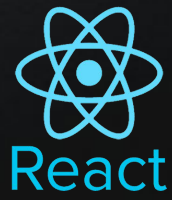
Para o nosso componente App ser reproduzido na tela agora temos que inserir ele no index.js.

index.js M × App.js M

src > 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

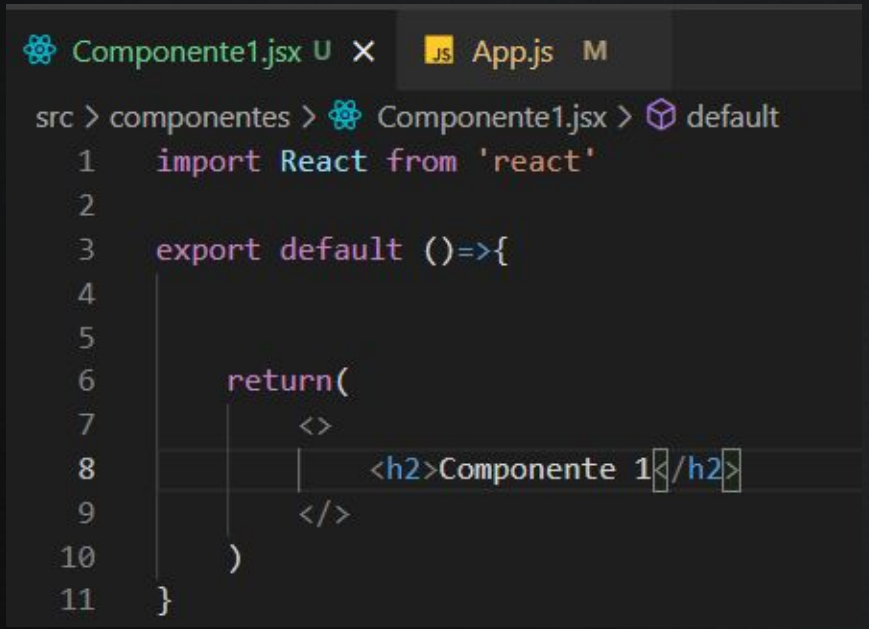
Quando criamos componentes inserimos no retorno das funções uma linguagem muito parecida com HTML, ela é chamada de JSX ou Javascript XML. As tags que utilizamos na verdade são de XML e não HTML, por isso percebemos algumas diferenças como fechar uma tag de quebra de linha com a barra no final ou usar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

# Usando arquivos JSX

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




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

# Trabalhando com props.children

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 | App.js M

src > componentes > Componente2.jsx > ...

```jsx
import React from 'react'

export default props=>{

    return(
        <>
        <h2>Componente 2</h2>
        {props.children}
        </>
    )
}
```

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

# Trabalhando com props.children

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 | Componente2.jsx U | App.js M

src > componentes > Componente1.jsx > ...

```jsx
import React from 'react'
import Componente2 from './Componente2'


export default ()=>{

    return(
        <>
            <h1>Componente 1</h1>
            <Componente2>
                <p>Esse texto vem do Componente 1</p>
            </Componente2>
        </>
    )
}
```

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

# Renderizar componente de forma Condicional

React

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:

Componente1.jsx U | Componente2.jsx U | App.js M

src > componentes > Componente1.jsx > ...

```jsx
import React, { useState } from 'react'
import Componente2 from './Componente2'


export default ()=>{

    const [num,setNum]=useState(0)

    return(
        <>
            <h1>Componente 1</h1>
            <Componente2>
                <p>O número {num} é:
                {//No caso de valores nos elementos:
                 num % 2 == 0 ? 'par' : 'impar' }</p>
                {//No caso de um elemento:
                    num % 2 == 0 ?
                    <p>Este número é par!!!</p> :
                    <p>Este número é impar!!!</p>
                }
            </Componente2>
            <button onClick={()=>setNum(num + 1)}>Num + 1</button>
        </>
    )
}
```

# Utilizando o método map() para criar elementos

React

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 | App.js M

src > componentes > Componente2.jsx > ...

```jsx
import React from 'react'

export default props=>{

    const frutas=['Maça','Banana','Pera','Uva','Goiaba']


    return(
        <>
        <h2>Componente 2</h2>
        <ul>
            {frutas.map((f)=><li>{f+' fresquinha'}</li>)}
        </ul>
        {props.children}

        </>
    )
}
```

# Utilizando o método map() para criar elementos

React

Se precisarmos de uma referencia 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 | App.js M

src > componentes > Componente2.jsx > ...

```jsx
import React from 'react'

export default props=>{

    const frutas=['Maça','Banana','Pera','Uva','Goiaba']


    return(
        <>
        <h2>Componente 2</h2>
        <ul>
            {frutas.map((f,ind)=><li>{f+` é a fruta nº ${ind +1}`}</li>)}
        </ul>
        {props.children}

        </>
    )
}
```

# Usando eventos para capturar valores de campos

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 | Componente2.jsx U | App.js M

src > componentes > NovoValor.jsx > ...

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

export default ()=>{

    const [frase,setFrase]=useState('')

    return(
        <>
            <h2>Vamos escrever uma frase:</h2>
            <label>Motivação:</label>
            <input type="text"  onChange={(e)=>setFrase(e.target.value)}/>
            <p>Motivação do dia: {frase}</p>

        </>
    )
}
```

Cuidado ao pegar o valor do evento

OBS. Não esqueça de depois chamar ele em App.

# Usando eventos para capturar valores de campos

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:

```jsx
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.