



**SOFTEX**  
PERNAMBUCO

 **Softex**

MINISTÉRIO DA  
CIÊNCIA, TECNOLOGIA  
E INOVAÇÃO

GOVERNO FEDERAL  
**BRASIL**  
UNIÃO E RECONSTRUÇÃO



## Aula 22 | Módulo: React JS (Introdução)



- Conceito de componentes: reutilização e modularidade
- Configuração de ambiente com Vite ou Create React App (já foi ensinado na aula passada)



# React JS (Introdução)



## Configurando o ambiente para rodar o React

- Node.js (LTS recomendado  $\geq 20$ ):
  - **Windows/macOS:**
    - baixe o instalador LTS no site do Node.
  - **Linux (Deb/Ubuntu):**
    - `curl -fsSL https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash`
    - `nvm install --lts`
    - `nvm use --lts`
- Confirmar instalação rodando os comandos abaixo em algum terminal:
  - **node -v**
  - **npm -v**
- Se “npm/node não reconhecido”, feche o terminal e tente pelo Command Prompt.



## React JS (Introdução)



### Abrindo/Criando um projeto

- Para ABRIR um projeto existente:
  - Abra o VSCode, clique em File > Open Folder... (Arquivo > Abrir Pasta...) e procurar a pasta do projeto existente.
  - No terminal, rode o comando ao lado para iniciar: **npm run dev**
- Para CRIAR um projeto novo
  - Abra o VSCode, clique em File > Open Folder... (Arquivo > Abrir Pasta...).
  - Escolha um local para criar a sua pasta, crie uma nova pasta e dê o nome de **seunome\_aula\_react\_22**. Depois dê dois clique nessa pasta criada e clique em Selecionar pasta. O VSCode reabrirá dentro dessa pasta que foi criada.



## React JS (Introdução)



### Criar o projeto com Vite

- Criando um projeto com JavaScript puro:
  - **npm create vite@latest . -- --template react**
- Durante a instalação do React usando o Vite, algumas opções podem surgir:

```
PS E:\hygorrasec\curso_react> npm create vite@latest . -- --template react
Need to install the following packages:
create-vite@8.0.2
Ok to proceed? (y) y

> npx
> create-vite . --template react

◇ Use rolldown-vite (Experimental)?:
No

◇ Install with npm and start now?
Yes
```



## React JS (Introdução)



### Se algo travar: portas ou permissões

- Porta ocupada (erro ao iniciar ou tela vazia):
  - Iniciar em outra porta: **npm run dev -- --port 5174**
- Windows: “npm não reconhecido”
  - Feche e reabra o PowerShell/CMD.
  - Verifique se o Node foi instalado para “All Users” (às vezes a variável de ambiente não atualiza).
- Linux/macOS: erro de permissão (EACCES)
  - `mkdir -p ~/.npm-global`
  - `npm config set prefix ~/.npm-global`
  - `echo 'export PATH=$HOME/.npm-global/bin:$PATH' >> ~/.bashrc && source ~/.bashrc`



## React JS (Introdução)



### Para fixar: O que é um Componente?

- Um componente é uma parte independente da interface.
- Ele pode representar um botão, menu, card, formulário, etc.
- Vantagem: reutilização e organização do código.

### Exemplo visual:

App (geral)

- Header (*componente*)
- Menu (*componente*)
- Conteúdo (*componente*)
- Rodapé (*componente*)



## React JS (Introdução)



### O que é JSX e por que ele é tão importante no React?

- JSX significa JavaScript XML.
  - É uma forma especial de escrever JavaScript que se parece com HTML.
- Ele permite misturar código JavaScript com marcação visual (HTML) dentro de um mesmo arquivo.
- JSX não é HTML real, mas o React transforma JSX em JavaScript puro na hora de executar.
- Essa combinação facilita a leitura, a manutenção e a criação de interfaces interativas.






## React JS (Introdução)



### Um componente simples

O que acontece aqui:

- Criamos uma função chamada **OlaMundo**.
- Ela retorna um código JSX (**<h1>Olá, mundo!</h1>**).
- O React entende esse retorno e exibe o texto na tela.

```
src > components >  OlaMundo.jsx > ...  
1   function OlaMundo() {  
2     |     return <h1>Olá, mundo!</h1>;  
3   }  
4  
5   export default OlaMundo;
```

O React transforma JSX em algo assim (em JavaScript puro):

➔ **React.createElement("h1", null, "Olá, mundo!");**

Ou seja, JSX é apenas uma forma mais amigável de escrever algo que o React entenderia de qualquer forma.




## React JS (Introdução)



### Regras importantes do JSX

- Todo JSX precisa ter apenas um elemento pai.

Errado:

```
src > components >  OlaMundo.jsx > ...  
1  function Ola() {  
2    return <h1>Olá</h1><p>Tudo bem?</p>;    JSX expressions must have one parent element.  
3  }  
4  
5  export default Ola;  
6
```

Certo:

```
src > components >  OlaMundo.jsx > ...  
1  function Ola() {  
2    // A <div> ... </div> está sendo um elemento pai  
3    return <div> <h1>Olá</h1><p>Tudo bem?</p> </div>;  
4  }  
5  
6  export default Ola;  
7
```



## React JS (Introdução)



### Regras importantes do JSX

- Os nomes dos componentes devem começar com letra maiúscula.
  - Isso diferencia componentes de tags HTML nativas.
    - Certo: `function Cabecalho()`
    - Errado: `function cabecalho()`



## React JS (Introdução)



### Regras importantes do JSX

- Usamos `{}` para inserir expressões JavaScript dentro do JSX.

```
src > components > BoasVindas.jsx > ...  
1  function BoasVindas() {  
2      const nome = "Hygor";  
3      return <h2>Bem-vindo, {nome}!</h2>;  
4  }  
5  
6  export default BoasVindas;  
7
```

- As propriedades (atributos) usam camelCase:
  - Exemplo: `className`, `onClick`, `backgroundColor`.
- O JSX não pode ter comentários comuns do HTML (`<!-- -->`). Use comentários do JS:
  - **`{/* Isso é um comentário dentro do JSX */}`**



## React JS (Introdução)



### Misturando HTML e JS

```
src > components >  BoasVindas.jsx > ...  
1  function BoasVindas() {  
2      const usuario = "Hygor";  
3      const hora = new Date().getHours();  
4      const saudacao = hora < 12 ? "Bom dia" : "Boa tarde";  
5  
6      return <h1>{saudacao}, {usuario}!</h1>  
7  }  
8  
9  export default BoasVindas;
```

- Dentro do JSX, usamos {} para colocar expressões JavaScript.
- O React avalia essas expressões e mostra o resultado na tela.
- Tudo o que estiver dentro de {} deve retornar um valor (não pode ser uma instrução como if ou for).



# React JS (Introdução)



## Exercício 1

- Crie o componente:
  - **OlaAluno.jsx**
- Esse componente deve mostrar o nome de um(a) aluno(a) e a data atual. Use a classe Date para trazer a data atual: **new Date().toLocaleDateString();**
- Use {} para exibir o nome em um H2 e a data dentro de um parágrafo.



## React JS (Introdução)



### Exercício 2

- Crie 2 componentes:
  - Cabecalho.jsx
  - Rodape.jsx
- No App.jsx, importe os dois e exiba na tela.
- Adicione uma <div> com o texto "Conteúdo principal" entre eles.



## React JS (Introdução)



### Por que separar o código em componentes?

- Imagine um site com dezenas de partes: cabeçalho, menu, produtos, rodapé...
- Se tudo estiver em um único arquivo, o código:
  - Fica difícil de entender...
  - É complicado de atualizar...
  - Repete muito código...
- Solução: dividir a interface em componentes independentes e organizados.
- Cada um tem sua função, e o conjunto forma o aplicativo inteiro.





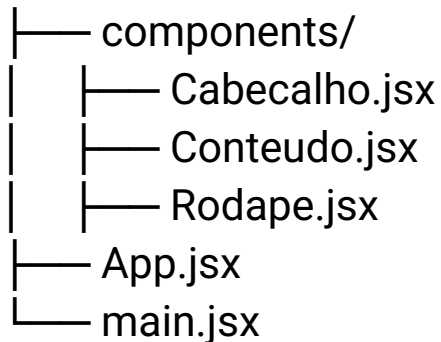
## React JS (Introdução)



### Estrutura de pastas organizada

- Um bom projeto React costuma ter uma estrutura parecida com esta:

src/



**components/** > guarda todos os componentes reutilizáveis.

**App.jsx** > componente principal que monta a página juntando os outros.

**main.jsx** > ponto de entrada que renderiza o App no navegador.



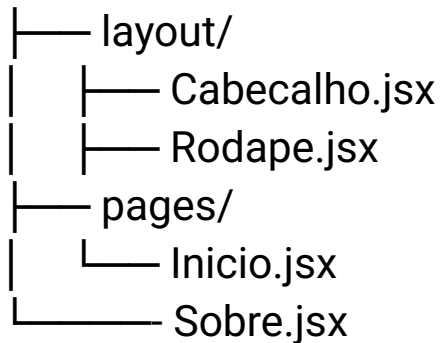
## React JS (Introdução)



### Boas práticas de modularização

- Nomeie os componentes com letra maiúscula (Header, Footer).
- Mantenha um componente por arquivo.
- Use nomes claros e descritivos:
  - Ruim: Comp1.jsx | Bom: MenuPrincipal.jsx
- Agrupe por função, se o projeto crescer:

components/





## React JS (Introdução)



### E se o projeto crescer?

- Nesse caso, podemos instalar o React Router se precisarmos ter páginas separadas.
- Ele servirá se:
  - O seu site tem múltiplas páginas (ex: /, /sobre, /contato, /produtos).
  - E você quiser mudar a URL sem recarregar a página.
  - Que o usuário possa usar o botão “Voltar” do navegador normalmente.
  - Construir um site tradicional (com navegação por links).



## React JS (Introdução)

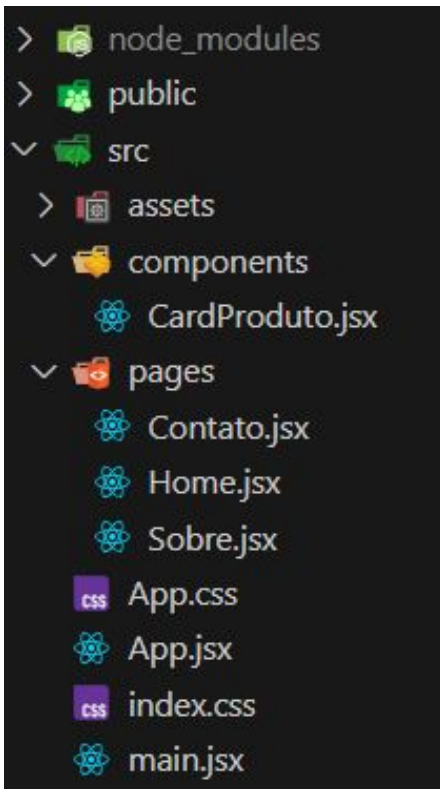


### Instalando o React Router (caso decida usar)

- Se quiser começar a estruturar um site com páginas, basta rodar no terminal:
  - **npm install react-router-dom**
- Para agilizar, vamos visualizar uma estrutura típica de projeto com páginas acessando o repositório abaixo:
  - **[https://github.com/hygorrasec/curso\\_react\\_bfd](https://github.com/hygorrasec/curso_react_bfd)**
- Faça o clone do repositório no computador de vocês usando o comando abaixo:
  - **git clone [https://github.com/hygorrasec/curso\\_react\\_bfd.git](https://github.com/hygorrasec/curso_react_bfd.git)**



## React JS (Introdução)



### Instalando o React Router (caso decida usar)

- Com o repositório clonado, abra o projeto no VSCode e instale as bibliotecas:
  - **npm install**
- Depois de instalar inicie o projeto com o comando abaixo:
  - **npm run dev**
- Observe a estrutura montada, veja que já estamos trabalhando com Router e a página muda o link da URL conforme clicamos nos links do menu.



## React JS (Introdução)



### Props (Propriedades): Passando Dados Entre Componentes

- O que são Props?
  - Elas servem para enviar informações de um componente pai para um componente filho.
  - São muito parecidas com parâmetros de uma função.



# React JS (Introdução)



## Exemplo simples usando props

- Temos um componente chamado Saudacao:

```
src > components > Saudacao.jsx > ...  
1  function Saudacao(props) {  
2    |    return <h2>Olá, {props.nome}</h2>;  
3  }  
4  
5  export default Saudacao;  
6
```

- E no App.jsx importamos esse componente:

```
src > App.jsx > ...  
You, 1 second ago | 1 author (You)  
1  import Saudacao from './components/Saudacao'  
2  
3  function App() {  
4    |    return <Saudacao nome="Hygor" />  
5  }  
6  
7  export default App;  
8
```

### Explicação

O componente Saudacao recebe um parâmetro chamado props.

Dentro dele, usamos {props.nome} para acessar o valor.

O componente App envia o dado nome="Hygor".

Resultado na tela: "Olá, Hygor!"



## React JS (Introdução)



### Props como parâmetros de função (forma mais moderna)

- Em vez de usar **props.nome**, podemos desestruturar as props:

```
src > components > Saudacao.jsx > ...  
1  function Saudacao({ nome }) {  
2    |    return <h2>Olá, {nome}</h2>;  
3  }  
4  
5  export default Saudacao;  
6
```

- É o mesmo resultado, mas o código fica mais limpo e legível.





## React JS (Introdução)



### Exemplo prático com múltiplas props

```
src > components > Produto.jsx > ...  
1  function Produto({ nome, preco }) {  
2      return (  
3          <div>  
4              <h3>{nome}</h3>  
5              <p>Preço: R$ {preco}</p>  
6          </div>  
7      );  
8  }  
9  
10 export default Produto;  
11
```

```
src > App.jsx > ...  
You, 27 seconds ago | 1 author (You)  
1  import Produto from './components/Produto'  
2  
3  function App() {  
4      return (  
5          <>  
6              <Produto nome="Teclado Gamer" preco={250} />  
7              <Produto nome="Mouse RGB" preco={150} />  
8          </>  
9      );  
10 }  
11  
12 export default App;  
13
```



## React JS (Introdução)



### Props e componentes com conteúdo interno

- Às vezes, queremos passar um bloco de conteúdo inteiro, e não apenas um valor.
- Isso é feito com a prop especial **children**.

```
src > components > Card.jsx > ...
1  function Card({ children }) {
2      return (
3          <div className="card">
4              {children}
5          </div>
6      );
7  }
8
9  export default Card;
10
```

```
src > App.jsx > ...
You, 1 second ago | 1 author (You)
1  import Card from './components/Card'
2
3  function App() {
4      return (
5          <Card>
6              <p>Este é um conteúdo dentro do Card.</p>
7          </Card>
8      );
9  }
10
11  export default App;
12
```

→ O children representa tudo o que está entre as tags **<Card>...</Card>**.



## React JS (Introdução)



### Exercício 3

- Crie um componente Aluno que receba três props:
  - nome
  - curso
  - nota
- Dentro do componente, exiba uma frase como:
  - O aluno {nome} do curso {curso} tirou nota {nota}.
- No App.jsx, exiba três alunos diferentes.
- Dica: use interpolação {} dentro do JSX para mostrar os valores.



## React JS (Introdução)



### Desestruturação e Boas Práticas com Props

- O que é desestruturação?
  - Desestruturação é uma forma mais limpa e direta de acessar valores de objetos.
  - Como props é um objeto que contém todos os dados enviados ao componente, podemos “desmontar” esse objeto em variáveis individuais.
- Analogia:
  - Imagine que props é uma caixa com vários itens.
  - Em vez de abrir a caixa toda hora para pegar algo (props.nome, props.idade), você tira tudo de dentro de uma vez:
  - **const { nome, idade } = props;**



# React JS (Introdução)



## Exemplos para comparação

- Exemplo sem desestruturação (forma longa)

```
src > components > Saudacao.jsx > ...  
1  function Saudacao(props) {  
2    |    return <h1>Olá, {props.nome}! Você tem {props.idade} anos.</h1>;  
3  }  
4  
5  export default Saudacao;
```

→ Aqui, sempre que queremos usar algo, precisamos digitar props.algumaCoisa. Em componentes grandes, isso fica cansativo e confuso.

- Exemplo com desestruturação (forma limpa)

```
src > components > Saudacao.jsx > ...  
1  function Saudacao({ nome, idade }) {  
2    |    return <h1>Olá, {nome}! Você tem {idade} anos.</h1>;  
3  }  
4  
5  export default Saudacao;
```

### Vantagens de usar desestruturação:

- Código mais curto e legível.
- Evita repetição de props.
- Facilita entender o que o componente precisa receber.



## React JS (Introdução)



### Valores padrão (default props)

- Às vezes, o componente pode ser usado sem que todas as props sejam informadas.
- Para evitar **undefined**, podemos definir valores padrão.

```
src > components > Aluno.jsx > ...
1  function Aluno({ nome = "Aluno Desconhecido", nota = 0 }) {
2    |    return <p>{nome} tirou nota {nota}</p>;
3  }
4
5  export default Aluno;
```

Isso é ótimo para deixar o componente mais seguro e previsível.

```
src > App.jsx > ...
You, 23 seconds ago | 1 author (You)
1  import Aluno from './components/Aluno'
2
3  function App() {
4    |    return (
5      |      <>
6        |      <Aluno nome="Lucymar" nota={10} />
7        |      <Aluno />
8      |      </>
9    |    );
10   }
11
12   export default App;
```



## React JS (Introdução)



### Validando props (para projetos maiores - conceito opcional)

- O React permite validar o tipo das props com uma biblioteca chamada **prop-types**.
  - **npm install prop-types**

```
src > components >  Produto.jsx > ...  
1  import PropTypes from 'prop-types';  
2  
3  function Produto({ nome, preco }) {  
4    |    return <p>{nome}: R$ {preco}</p>;  
5  }  
6  
7  Produto.propTypes = {  
8    |    nome: PropTypes.string.isRequired,  
9    |    preco: PropTypes.number  
10 };  
11  
12 export default Produto;
```

Assim, se alguém tentar passar **preco="barato"**, o React mostrará um aviso no console.



## React JS (Introdução)



### Exercício 4

- Crie um componente Usuario que receba:
  - nome
  - idade
  - cidade (opcional, com valor padrão “Maricá”)
- Exiba a mensagem:
  - {nome}, {idade} anos, mora em {cidade}.
- No App.jsx, chame o componente com três usuários diferentes.





## React JS (Introdução)



### Props Dinâmicas e Expressões JavaScript

- Por que props dinâmicas são importantes?
  - Até agora, usamos props com valores fixos, como nome="Renan".
  - Mas o React permite usar variáveis e expressões JavaScript dentro das props. Isso torna o componente dinâmico e reutilizável.
  - Ou seja: o conteúdo exibido muda conforme os dados mudam.



## React JS (Introdução)



### Exemplo simples: props com variáveis

```
src > components > Saudacao.jsx > ...
1  function Saudacao({ nome }) {
2    |    return <h2>Olá, {nome}</h2>;
3    |  }
4
5  export default Saudacao;
```

```
src > App.jsx > ...
You, 51 seconds ago | 1 author (You)
1  import Saudacao from './components/Saudacao'
2
3  function App() {
4    |    const usuario = "Maria";
5    |    return <Saudacao nome={usuario} />;
6    |  }
7
8  export default App;
```

- Criamos uma variável usuario.
- Passamos ela como prop (nome={usuario}).
- O React substitui automaticamente {usuario} por "Maria".



## React JS (Introdução)



### Props com expressões JavaScript

- Podemos colocar qualquer expressão JavaScript dentro de {}:

```
src > components > Mensagem.jsx > ...  
1  function Mensagem({ nome }) {  
2    |    return <p>Bem-vindo, {nome.toUpperCase()}!</p>;  
3  }  
4  
5  export default Mensagem;  
6
```

```
src > App.jsx > ...  
You, 15 seconds ago | 1 author (You)  
1  import Mensagem from './components/Mensagem'  
2  
3  function App() {  
4    |    return <Mensagem nome="hygor" />;  
5  }  
6  
7  export default App;  
8
```

Tudo dentro de {} é avaliado como JavaScript e o resultado é mostrado.



# React JS (Introdução)



## Props com cálculos e condições

- Podemos até passar **expressões matemáticas** ou **condições**:

```
src > components > Nota.jsx > ...
1  function Nota({ aluno, valor }) {
2      return (
3          <p>
4              {/* Usando operador ternário */}
5              {aluno} tirou nota {valor} - {valor >= 7 ? "Aprovado" : "Reprovado"}
6          </p>
7      );
8  }
9
10 export default Nota;
```

```
src > App.jsx > ...
You, 16 seconds ago | 1 author (You)
1  import Nota from './components/Nota'
2
3  function App() {
4      return (
5          <>
6              <Nota aluno="Marcio" valor={10} />
7              <Nota aluno="Juliana" valor={10} />
8          </>
9      );
10 }
11
12 export default App;
```

→ O operador **? (ternário)** é muito usado no React para condições simples.



## React JS (Introdução)



### Componentes Reutilizáveis e Personalizáveis com Props

- Em um site, muitas partes se repetem: botões, cards, alertas, inputs...
- Em vez de copiar e colar o mesmo código, criamos um único componente genérico, e mudamos apenas o que for diferente usando props.



## React JS (Introdução)



### Componentes Reutilizáveis e Personalizáveis com Props

- Exemplo: criando um botão genérico!
- Componente **Botao.jsx**:

```
src > components > Botao.jsx > ...  
1  function Botao({ texto, cor_fundo, cor_texto }) {  
2      return (  
3          <button style={{ backgroundColor: cor_fundo, color: cor_texto }}>  
4              {texto}  
5          </button>  
6      );  
7  }  
8  
9  export default Botao;
```



# React JS (Introdução)

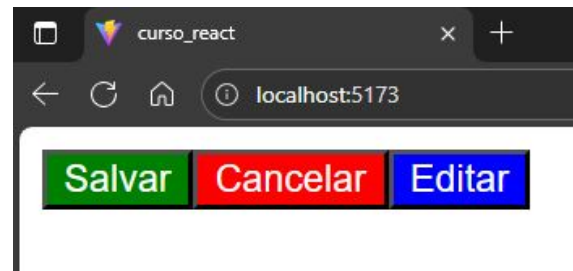


## Componentes Reutilizáveis e Personalizáveis com Props

- Exemplo: criando um botão genérico!
- Importamos no **App.jsx**:

```
src > App.jsx > ...  
You, 1 minute ago | 1 author (You)  
1 import Botao from './components/Botao'  
2  
3 function App() {  
4   return (  
5     <>  
6       <Botao texto="Salvar" cor_fundo="green" cor_texto="white" />  
7       <Botao texto="Cancelar" cor_fundo="red" cor_texto="white" />  
8       <Botao texto="Editar" cor_fundo="blue" cor_texto="white" />  
9     </>  
10  );  
11 }  
12  
13 export default App;
```

Resultado:





## React JS (Introdução)



### Adicionando comportamento (funções como props)

- Props também podem receber funções, por exemplo, ações ao clicar em um botão:

```
src > components > Botao.jsx > ...
1  function Botao({ texto, aoClicar }) {
2      return (
3          <button onClick={aoClicar}>
4              {texto}
5          </button>
6      );
7  }
8
9  export default Botao;
```

Agora as props controlam tanto o visual quanto o comportamento.

```
src > App.jsx > ...
You, 15 seconds ago | 1 author (You)
1  import Botao from './components/Botao'
2
3  function App() {
4      function salvar() {
5          alert("Dados salvos com sucesso!");
6      }
7
8      return (
9          <>
10             <Botao texto="Salvar" cor="green" aoClicar={salvar} />
11          </>
12      );
13  }
14
15  export default App;
```





# React JS (Introdução)



## Exercício 1 - Gabarito

```
src > components > OlaAluno.jsx > ...
1  function OlaAluno() {
2      const nome = "Maria";
3      const data = new Date().toLocaleDateString();
4
5      return (
6          <div>
7              <h2>Olá, {nome}!</h2>
8              <p>Hoje é {data}.</p>
9          </div>
10     );
11 }
12
13 export default OlaAluno;
14
```

```
src > App.jsx > ...
1  import OlaAluno from './components/OlaAluno';
2
3  export default function App() {
4      return (
5          <>
6              <OlaAluno />
7          </>
8      );
9  }
10
```



# React JS (Introdução)



## Exercício 2 - Gabarito

```
src > components > Cabecalho.jsx > ...
1  function Cabecalho() {
2    |    return <p>Cabeçalho</p>
3  }
4
5  export default Cabecalho;
```

```
src > components > Rodape.jsx > ...
1  function Rodape() {
2    |    return <p>Rodapé</p>
3  }
4
5  export default Rodape;
```

```
src > App.jsx > ...
1  import Cabecalho from './components/Cabecalho';
2  import Rodape from './components/Rodape';
3
4  export default function App() {
5    |    return (
6    |        <>
7    |            <Cabecalho />
8    |            <div>Conteúdo principal</div>
9    |            <Rodape />
10   |        </>
11   |    );
12 }
```



# React JS (Introdução)



## Exercício 3 - Gabarito

```
src > components > Aluno.jsx > ...
1  function Aluno({ nome, curso, nota }) {
2      return (
3          <p>
4              0 aluno {nome} do curso {curso} tirou nota {nota}.
5          </p>
6      );
7  }
9  export default Aluno;
```

```
src > App.jsx > ...
You, 43 seconds ago | 1 author (You)
1  import Aluno from './components/Aluno'
2
3  function App() {
4      return (
5          <>
6              <Aluno nome="Webert" curso="React" nota={10} />
7              <Aluno nome="Ana" curso="JavaScript" nota={10} />
8              <Aluno nome="João" curso="CSS" nota={10} />
9          </>
10     );
11 }
13 export default App;
14
```



# React JS (Introdução)



## Exercício 4 - Gabarito

```
src > components > Usuario.jsx > ...  
1  function Usuario({ nome, idade, cidade = "Maricá" }) {  
2    |    return <p>{nome}, {idade} anos, mora em {cidade}</p>;  
3  }  
4  
5  export default Usuario;
```

```
src > App.jsx > ...  
You, 8 seconds ago | 1 author (You)  
1  import Usuario from './components/Usuario'  
2  
3  function App() {  
4    return (  
5      <>  
6        <Usuario nome="Hygor" idade={29} cidade="Niterói" />  
7        <Usuario nome="Maria" idade={25} />  
8        <Usuario nome="João" idade={32} cidade="Rio de Janeiro" />  
9      </>  
10   );  
11 }  
12  
13 export default App;
```



# ATÉ A PRÓXIMA AULA!

*Front-end - Design. Integração. Experiência.*

**Professor:** Hygor Rasec

<https://www.linkedin.com/in/hygorrasec>

<https://github.com/hygorrasec>