Curso Presencial Programação Fullstack

Aula 03

Prof. MSc. Kelson | Senior Software Engineer

PROFKELSON.DEV

BORA CODAR?

Cronograma

PROFKELSON.DEV
BORA CODAR?

01
React JS

02
Muita prática!

03
Mentoria

PROFKELSON.DEV
BORA CODAR?

ReactJS

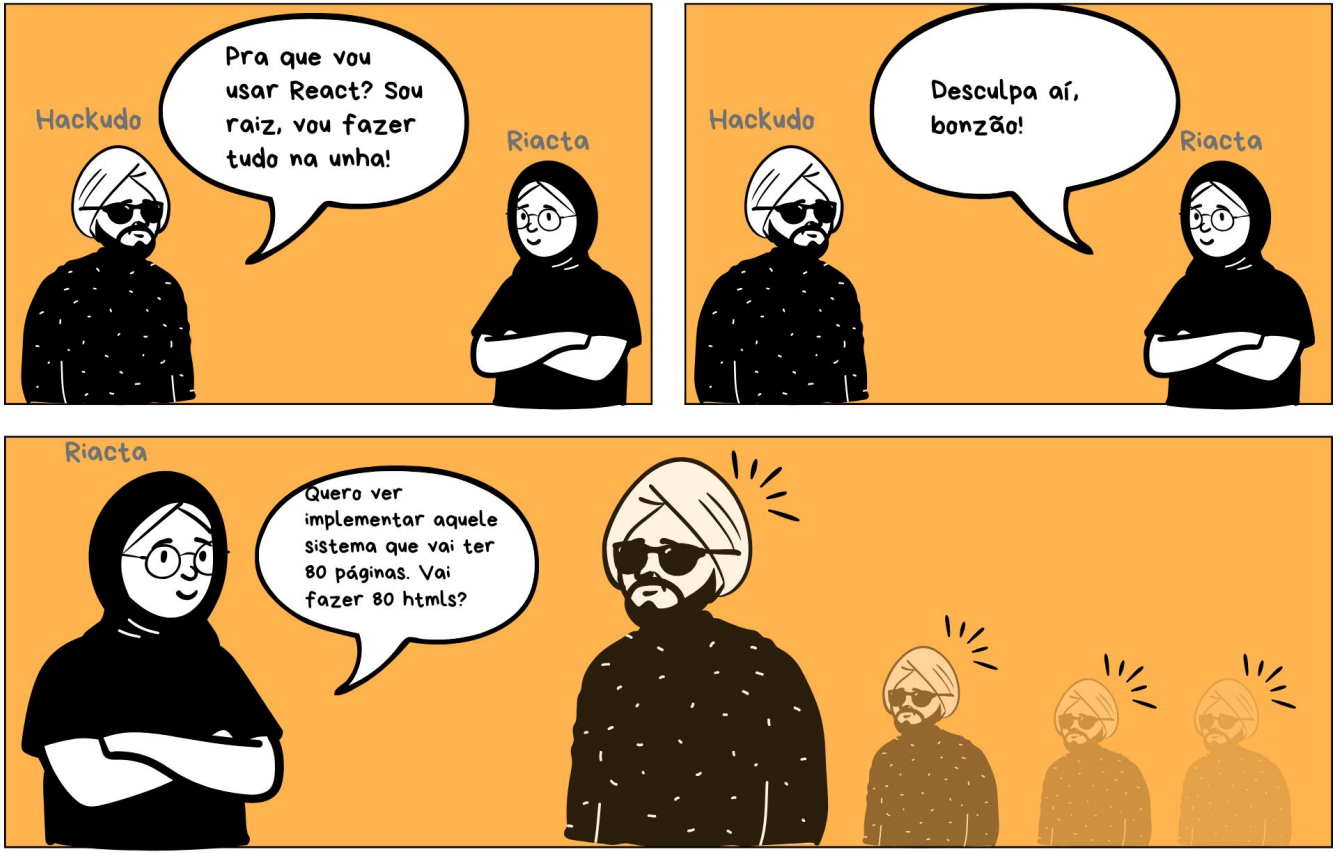
- React é uma biblioteca JS
  - Apesar de alguns autores afirmarem que o ReactJS é um framework, ele é oficialmente e mais amplamente difundido como uma biblioteca.
- Single Page Application (SPA)
  - SPA é uma abordagem de dev de aplicações em que todo o conteúdo é carregado em uma única página da web, em vez de carregar várias páginas diferentes.

PROFKELSON.DEV
BORA CODAR?

ReactJS

- No React, um SPA é criado utilizando componentes React que são utilizados para construir a interface do usuário.
- Mantida pelo **Facebook**
  - O que é ótimo, pois é mantido por uma grande empresa e há sempre possibilidade de novas atualizações para a biblioteca.

POR QUE USAR REACTJS???



PROFKELSON.DEV
BORA CODAR?

PROFKELSON.DEV
BORA CODAR?

NodeJS

- **Runtime** de JavaScript.
- Node.js é uma plataforma de desenvolvimento de aplicativos JavaScript que permite executar código JavaScript no **lado do servidor**.
- Embora o React possa ser executado em um navegador da web, ele também pode ser executado no lado servidor usando o Node.js.
- Isso permite que o React seja utilizado em aplicativos SPA ou em aplicações no lado do servidor.

PROFKELSON.DEV
BORA CODAR?

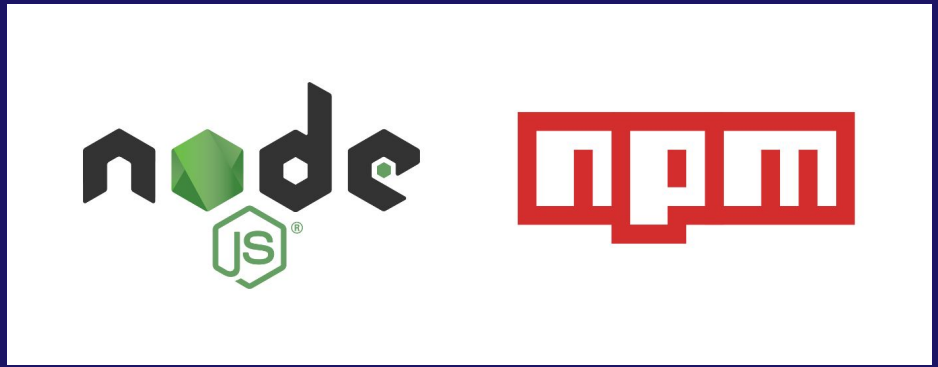
NodeJS

- Além disso, o node.js fornece várias ferramentas úteis para o desenvolvimento de **aplicações React**.
  - Executar o **servidor de desenvolvimento do React**
  - Permite que visualize as **alterações em tempo real** enquanto trabalha na aplicação
  - Executar **tarefas de compilação**
  - Gerenciamento de pacotes
  - Entre outros.

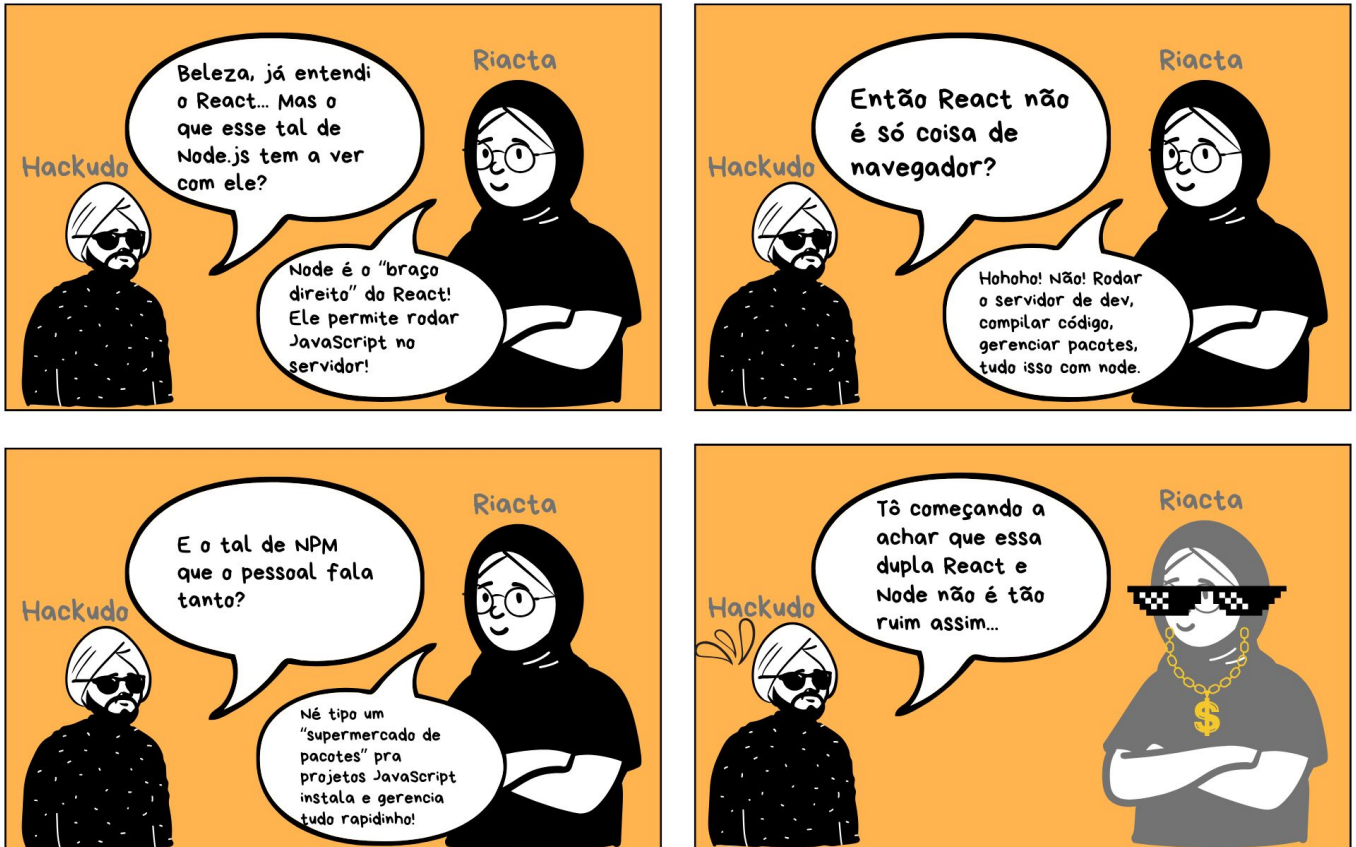
PROFKELSON.DEV
BORA CODAR?

NodeJS

- **npm:** (Node Package Manager) é uma ferramenta que permite aos desenvolvedores gerenciar e compartilhar pacotes de software reutilizáveis em projetos.
- Desenvolvedores costumam usar o **NPM para instalar e gerenciar pacotes** de dependências para seus projetos ReactJS.



DESCOBRINDO O NODEJS...



PROFKELSON.DEV
BORA CODAR?

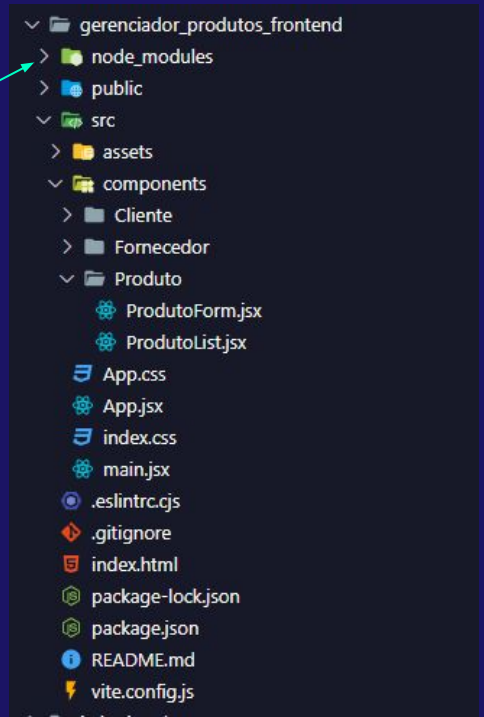
VAMOS CODAR? [COM O PROF/DEPOIS SOZINHO(A)]

- Entre na sua pasta do workspace do curso
- Digite o comando:
  - **npm create vite@latest**
- Esse comando irá criar um projeto ReactJS
  - Escolha um nome para o projeto:
    - aula03
    - JavaScript

PROFKELSON.DEV
BORA CODAR?

Estrutura de projeto ReactJS

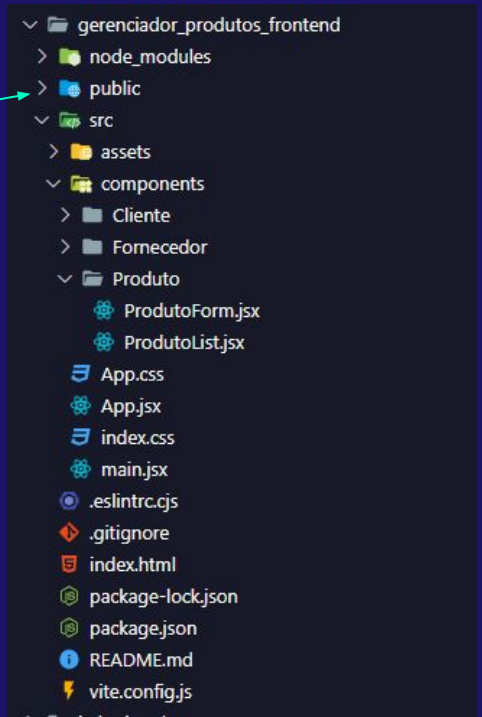
- **node_modules:** geralmente é criado em um projeto React quando se utiliza um gerenciador de pacotes npm (node package manager) para instalar as dependências do projeto. Quando se cria o projeto React e instala dependências usando o npm, essas dependências são baixadas e armazenadas nesse diretório.



PROFKELSON.DEV
BORA CODAR?

Estrutura de projeto ReactJS

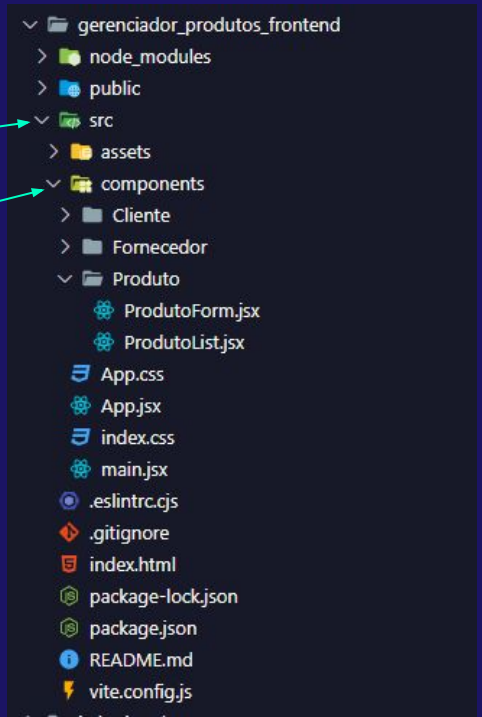
- **public:** Esse diretório é criado para armazenar todos os recursos estáticos da aplicação React que são acessados diretamente pelo navegador. HTML principal, imagens, arquivos de ícone, etc.



PROFKELSON.DEV
BORA CODAR?

Estrutura de projeto ReactJS

- **src:** (abreviação de "**source**"), é geralmente onde **armazenamos o código-fonte da aplicação** (JS, CSS e outros arquivos necessários para a codificação).
  Pode haver a pasta uma pasta **"components"** que contém os componentes React da aplicação, uma pasta **"pages"** que contém os componentes que representam as páginas da aplicação e uma pasta **"utils"** que contém funções utilitárias.



PROFKELSON.DEV
BORA CODAR?

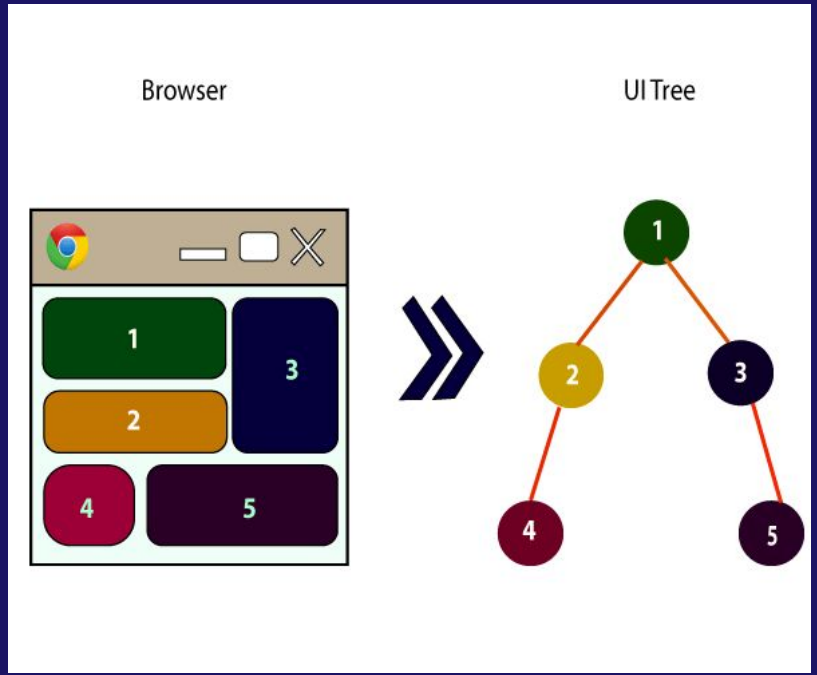
Componentes

- Um **componente é uma unidade** fundamental de uma aplicação que **pode ser reutilizada em diferentes partes da interface do usuário.**
  Um componente **pode ser definido como uma função** ou uma classe que encapsula uma parte da interface do usuário e possui suas **próprias propriedades, estado e comportamentos.**

PROFKELSON.DEV
BORA CODAR?

Componentes

- Os **componentes** podem ser compostos uns dentro dos outros, p**ermitindo a construção de interfaces de usuário complexas a partir de componentes mais simples e reutilizáveis.**
- Os componentes podem ser divididos em dois tipos principais:
    - **Componentes Funcionais**
    - Componentes de Classe



PROFKELSON.DEV
BORA CODAR?

JSX

- JSX **é uma extensão de sintaxe para o JavaScript** que é comumente usada no desenvolvimento de aplicações React.
- O **JSX permite que você escreva tags HTML e outros elementos da interface do usuário dentro do seu código JavaScript**, tornando mais fácil criar componentes React.
- Em vez de criar esses elementos em uma API separada, como o DOM, o **JSX permite criar esses elementos diretamente dentro do seu código JS.**

jsx

```jsx
const element = <h1>Hello, world!</h1>;
```

PROFKELSON.DEV
BORA CODAR?

JSX

- No exemplo ao lado, a sintaxe JSX é usada para criar um elemento <h1> que contém o texto **“Hello, world!”**, o elemento é **armazenado em uma variável chamada “element”**.
- Atenção: **JSX não é JavaScript válido**. Então é **necessário um compilador para converter o código JSX em JavaScript**. O conversor mais comum usado com o React é o **Babel**.

HTML

```jsx
const element = <h1>Hello, world!</h1>;
```

JavaScript

JSX

- Além disso, o JSX pode incluir expressões JavaScript dentro das tags usando chaves {}.
- Chamamos de Templates Expressions.
- Isso permite que você crie elementos de interface do usuário dinamicamente com base em dados ou variáveis.
- No exemplo ao lado, a expressão “{name}” é avaliada como o valor da variável “name”, que é “John Doe”, o resultado é um elemento <h1> que contém o texto “Hello, John Doe!”.

jsx

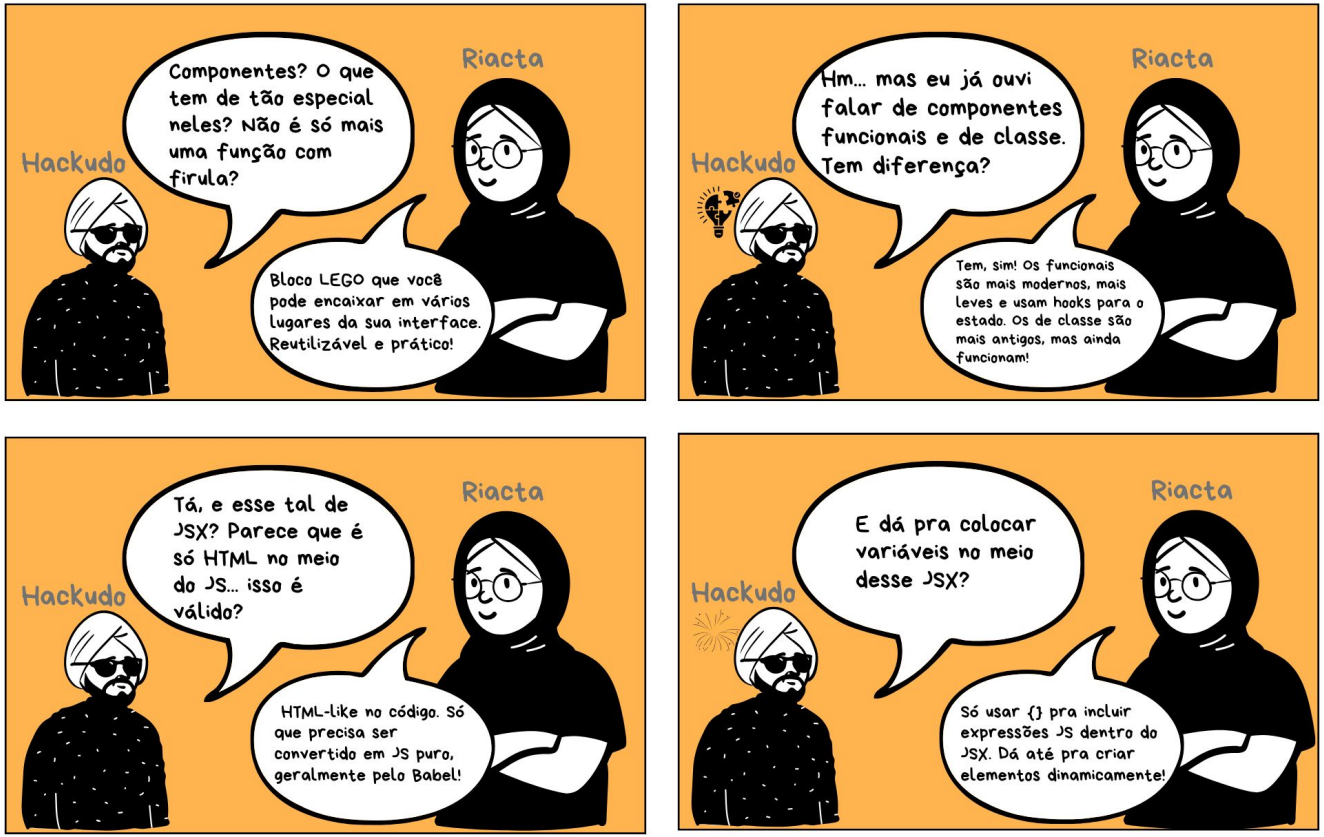
```jsx
const name = 'John Doe';
const element = <h1>Hello, {name}!</h1>;
```

jsx

```jsx
const element = <h1>Hello, world!</h1>;
```

```
</div>
<h1>Kelson Almeida {2+2}</h1>
<div className="card">
```

DESCOBRINDO COMPONENTES E JSX...



PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? (1) [COM O PROF]

- No React crie 4 componentes no seu projeto: Adicao, Subtracao, Multiplicacao e Divisao
- Esses componentes devem renderizar a seguinte frase dentro de uma tag <h1>, “O resultado de num1 + num2 é igual a resultado”
  - (-) para subtração
  - (*) para multiplicação
  - (/) para divisão
  - num1 , num2 são atributos do seu componente.
- Importe esses componentes criados para o App.jsx e passe os valores de num1 e num2 como propriedade com o objetivo de exibir a frase inteira na tela.
-

PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? (2) [COM O PROF]

- No React crie um componente chamado “PrecisoEstudar.jsx”
- Esse componente deve renderizar a seguinte frase dentro de uma tag <h1>, “Preciso estudar NOME-DE-ALGUMA-TECNOLOGIA”
- O componente deve ter uma propriedade chamada “nomeDaTecnologia” que irá exibir o nome da tecnologia na frase.
- Importe esse componente criado para o App.jsx e passe o nome da tecnologia como propriedade com o objetivo de exibir a frase inteira na tela.

PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? (2_1) [SOZINHO(A)]

- No React crie 4 componentes no seu projeto: CelsiusParaFahrenheit, FahrenheitParaCelsius, QuilometroParaMilhas, MilhasParaQuilometro
  - Faça com que esses componentes retornem no componente principal (App.jsx) e exiba em tela:
    - X° C é o mesmo que X °F
    - Fazer o mesmo para as outras 3 conversões
  - Pesquisar na internet qual é a forma de conversão de todas elas e aplicar no seu JSX.

PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? (3) [COM O PROF]

- Crie um componente Livro que recebe titulo, autor e ano como propriedades e renderiza essas informações dentro de uma tag .
- Crie um componente ListaDeLivros que recebe uma lista de livros como propriedade e renderiza um componente Livro para cada item na lista.
- A lista de livros deve ser um array de objetos com titulo, autor e ano.
- Importe e utilize os componentes Livro e ListaDeLivros no componente principal App.
- Passe uma lista de livros como propriedade para o componente ListaDeLivros.

PROFKELSON.DEV
BORA CODAR?

Renderizando com Funções

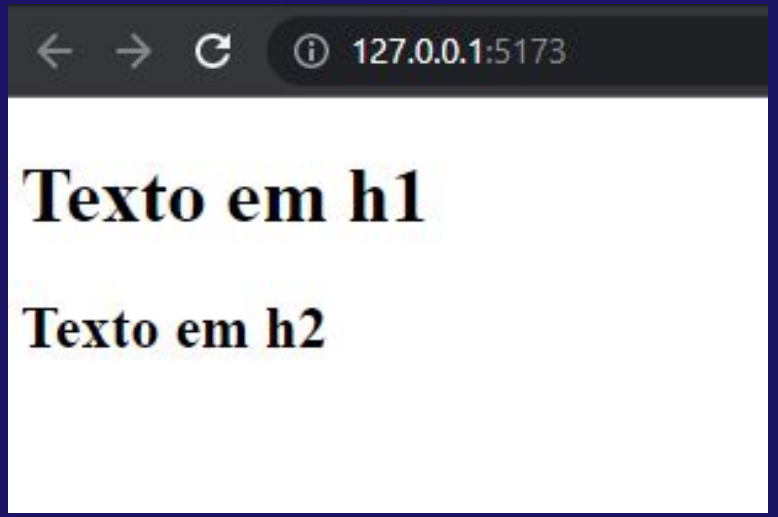
- É possível criar funções que retornem JSX.
- Situações que dependam de outras condições
- Exemplo: O JSX a ser renderizado pode mudar de acordo com alguma variável.

```
10
11      const renderSomething = (x) => {
12        if (x) {
13          return <h1>Renderizando isso!</h1>
14        } else {
15          return <h1>Renderizando aquilo!</h1>
16        }
17      }
18
19    return (
20      <div>
21          <div>
22              <button onClick={handleMyEvent}>Clique aqui</button>
23          </div>
24          <div>
25            {renderSomething(true)}
26            {renderSomething(false)}
27          </div>
28      </div>
29    )
30  }
31
32  export default Events
```

PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? (4) [COM O PROF]

- Crie um componente chamado: **RenderizandoComFuncoes**
- Neste componente crie uma função chamada: escolhaDeRenderizacao que terá um parâmetro chamado: **oQueRenderizar**
- Nesta função, se o valor de "**oQueRenderizar**" for a string "PB", terá como retorno um JSX com o texto "Paraíba" envolto pela tag h1.
- Se a string for diferente de "PB", o retorno será: "João Pessoa" envolto pela tag h2.
- Na tela chame essa função através de duas template expressions, uma passando como parâmetro "PB" e outra "JP".
- Segue o exemplo do resultado na tela ao lado:



PROFKELSON.DEV
BORA CODAR?

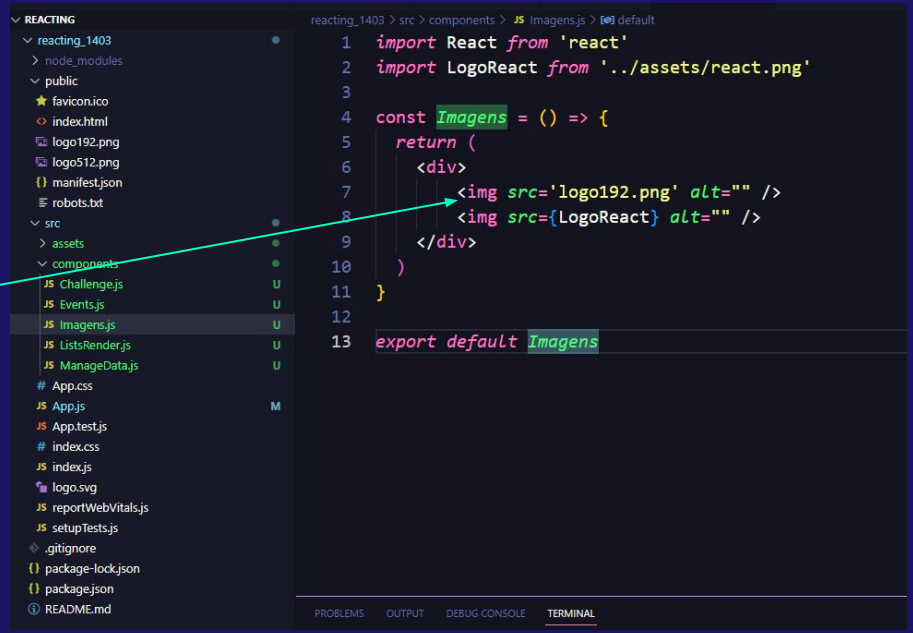
VAMOS CODAR? (4_1) [SOZINHO(A)]

- Crie um componente Mensagem que recebe uma propriedade tipo.
- O componente deve renderizar diferentes mensagens baseadas no valor da propriedade tipo:
- Se tipo for "sucesso", renderizar "Operação realizada com sucesso!" dentro de uma tag <h1>.
- Se tipo for "erro", renderizar "Ocorreu um erro na operação." dentro de uma tag <h2>.
- Se tipo for "aviso", renderizar "Atenção! Verifique os dados." dentro de uma tag <h3>.

PROFKELSON.DEV
BORA CODAR?

Trabalhando com Imagens

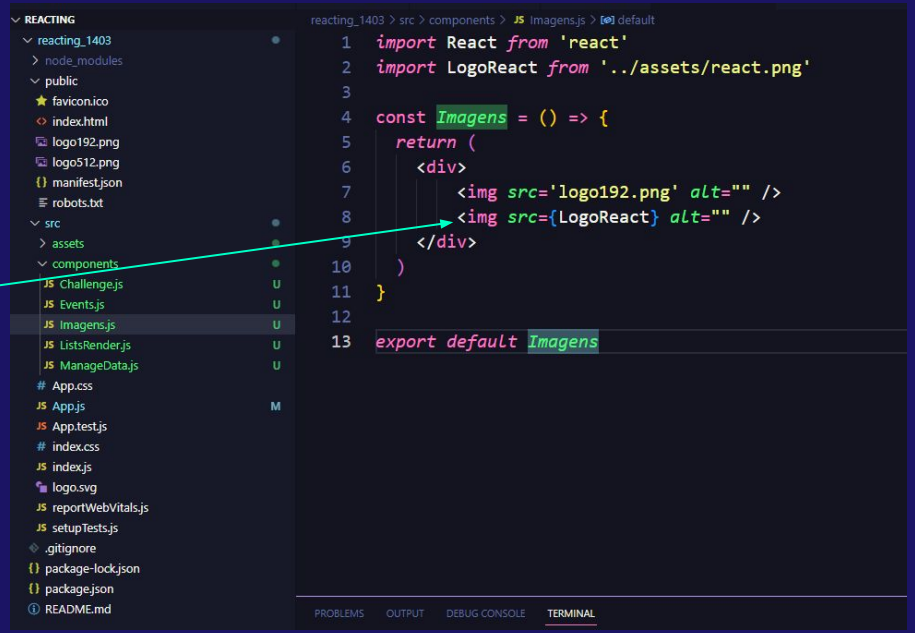
- A **pasta public** do nosso projeto pode abrigar as imagens públicas.
- Quando as imagens estão nessa pasta, podemos **chamá-las nos nossos componentes através da tag img e /nome-da-imagem.jpg**
- Isso se dá, pois a pasta public fica conectada ao src
- Por boas práticas, sempre devemos ao menos inicializar a propriedade “alt” do img.
-



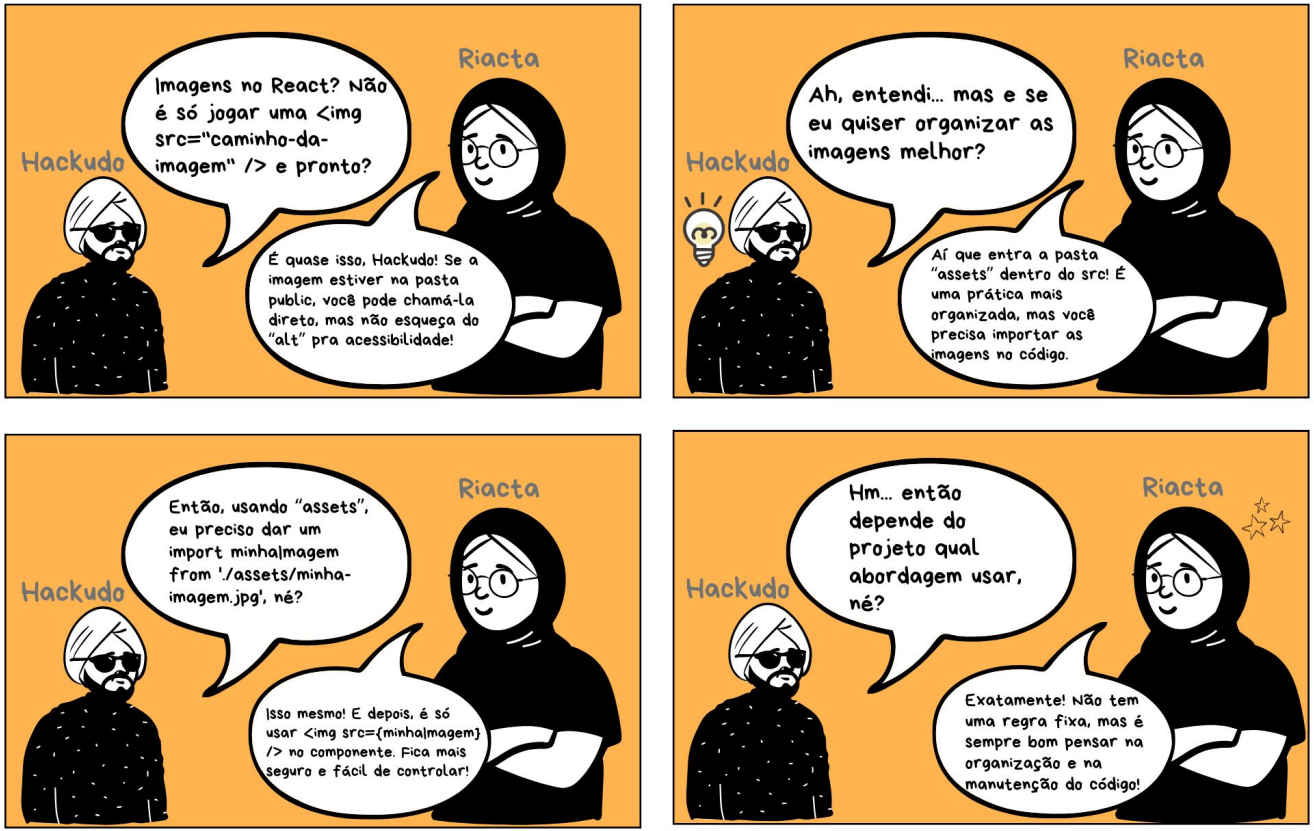
PROFKELSON.DEV
BORA CODAR?

Trabalhando com Imagens

- **Outro padrão** bem utilizado para inserir imagens em projetos, é **utilizar uma pasta chamada "assets" dentro de src.**
- Podemos encontrar projetos utilizando as duas abordagens.
- Caso optemos por assets, precisamos importar as imagens.



TRABALHANDO COM IMAGENS NO REACT...



PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? (5) [COM O PROF]

- Crie um componente chamado TrabalhandoComImagens
- Baixe na internet duas imagens quaisquer.
- Exiba essas duas imagens na tela, uma através da pasta public e a outra através de assets importado.

PROFKELSON.DEV
BORA CODAR?

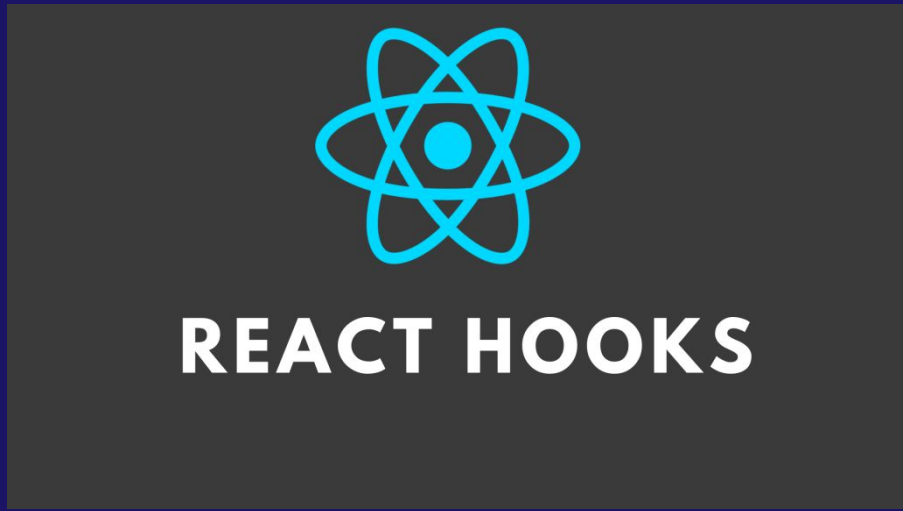
VAMOS CODAR? (5_1) [SOZINHO(A)]

- Crie um componente chamado TrabalhandoComImagensCondicionais
- Baixe na internet duas imagens quaisquer.
- Faça uma renderização condicional:
  - Se a função receber a string “PUBLIC”: mostrar a imagem pública
  - Se receber a string “ASSET”: Exiba a imagem de asset importado.

PROFKELSON.DEV
BORA CODAR?

Hooks

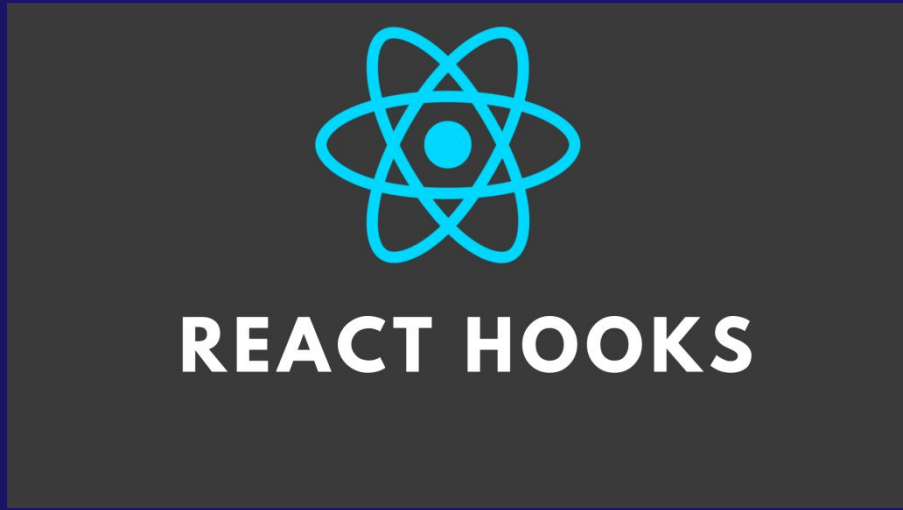
- Recurso do React que tem muitas funções!
- Guardar/alterar o estado de algum dado.
- Em nomenclatura, os hooks iniciam com a palavra “use”.
    - Exemplo: useState.
- Precisam ser importados!
- São bastante utilizados nas aplicações.
- Utilizaremos bastante durante as nossas práticas.



PROFKELSON.DEV
BORA CODAR?

Hooks

- Até a versão 16.7, algumas funcionalidades só eram possíveis serem utilizadas através de classes.
- Na versão 16.8 do React, foram lançados os hooks, que permitem uso de vários recursos através de funções. Ajudando também a organizar a lógica dentro dos componentes.



PROFKELSON.DEV
BORA CODAR?

Hooks

- Uma forma de adicionar funcionalidades de estado e ciclo de vida a componentes funcionais do React.
- Existem vários tipos de Hooks disponíveis, os mais comuns são o "**useState**" e o "**useEffect**".
- No "useState" é **permitido adicionar estado a um componente funcional**

javascript

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

function Example() {
  // Declara uma variável de estado chamada "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Clique aqui
      </button>
    </div>
  );
}
```

PROFKELSON.DEV
BORA CODAR?

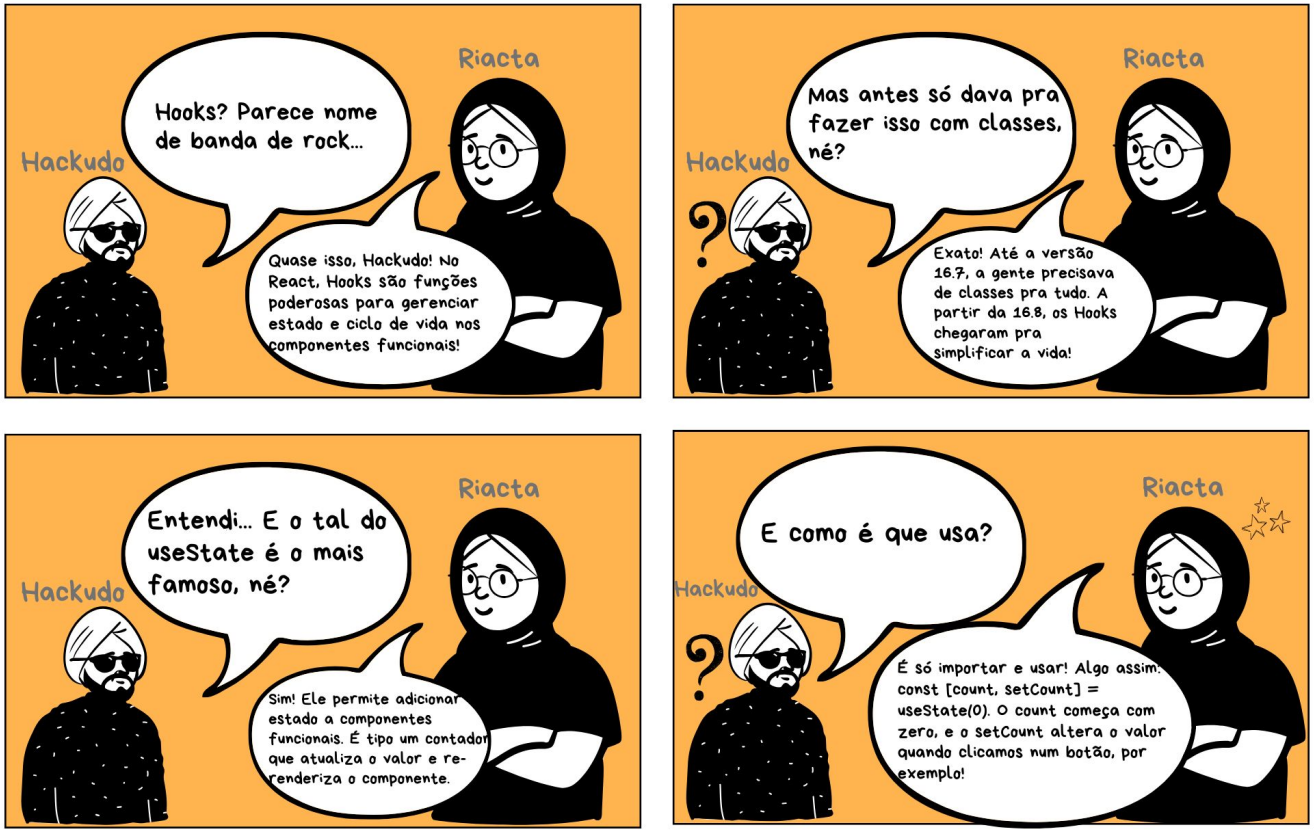
Hooks

- No exemplo ao lado, o “useState” é utilizado para adicionar um estado chamado “count” ao componente “Example”. O “count” começa com o valor de 0 e é atualizado sempre que o botão é clicado.
- Quando o estado é atualizado, o componente é renderizado novamente com o valor do estado.

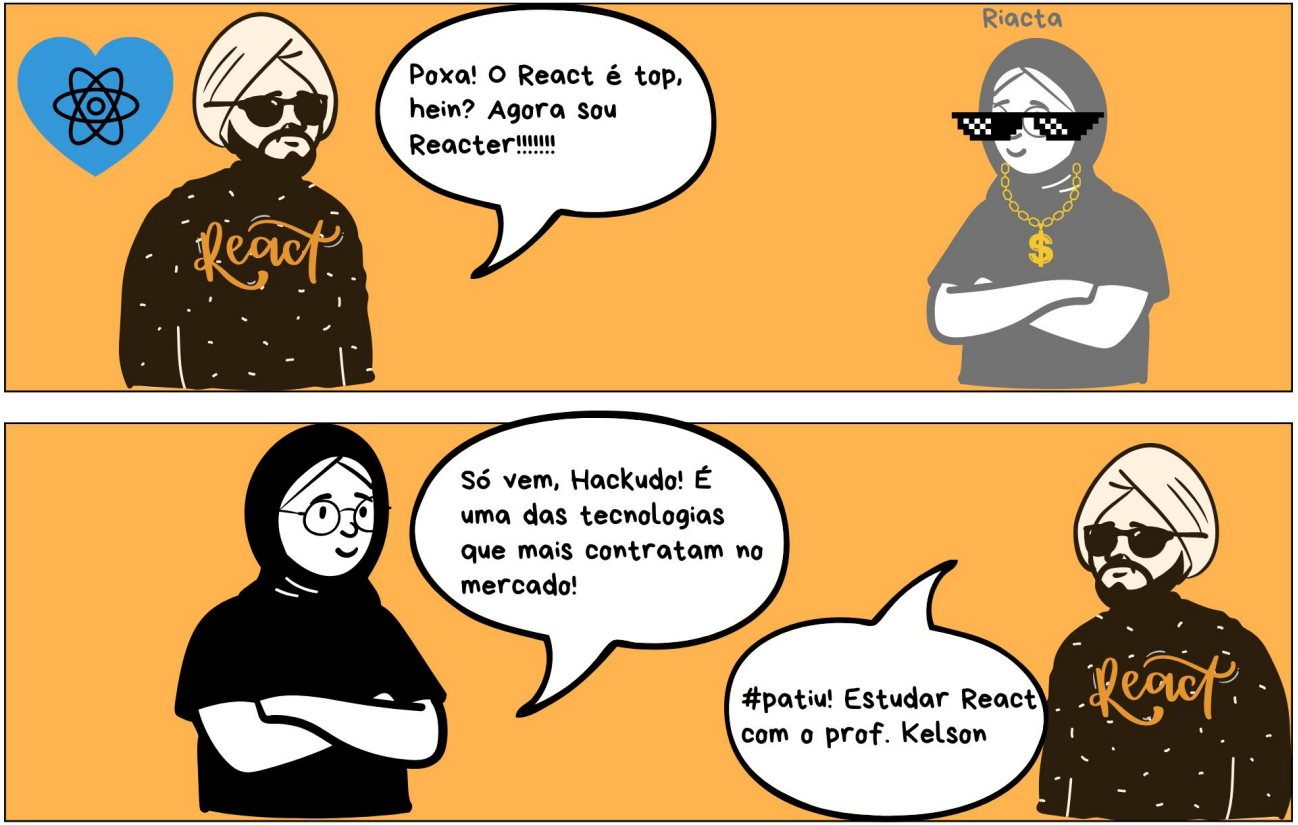
javascript

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

function Example() {
  // Declara uma variável de estado chamada "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Clique aqui
      </button>
    </div>
  );
}
```

DESCOBRINDO O USESTATE...



E NASCE MAIS UM FANBOY REACT!



PROFKELSON.DEV
BORA CODAR?

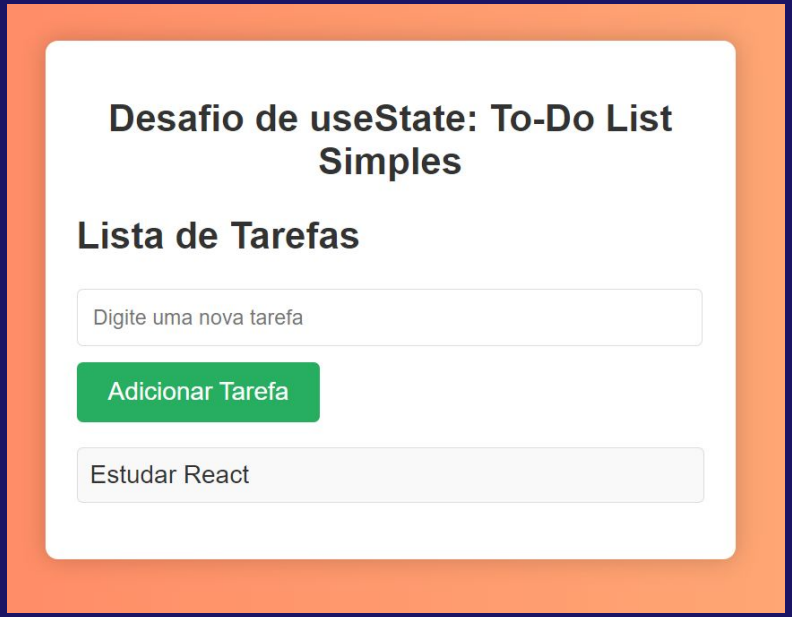
VAMOS CODAR? (6) [COM O PROF]

- Crie um componente chamado: HookContador
- Seu componente terá um useState “contador” com estado inicial de valor 1.
- Crie uma função “incrementarContador” que altera o valor do contador para valor anterior+1
- A página de exibição deve seguir o exemplo ao lado:
- Crie o button “Incrementar contador” para exibir o valor de cada incremento.

PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? (6_1) [SOZINHO(A)]

- Crie um componente TodoList que utiliza o hook useState para gerenciar o estado de uma lista de tarefas.
- Inicialize a lista de tarefas com um array vazio.
- Crie uma função adicionarTarefa que adiciona uma nova tarefa à lista.
- Renderize um campo de texto para inserir novas tarefas e um botão para adicionar a tarefa à lista.
- Renderize a lista de tarefas abaixo do campo de texto e do botão.



PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? (6_1) [SOZINHO(A)]

```
4   const [tarefas, setTarefas] = useState([]);
5   const [novaTarefa, setNovaTarefa] = useState('');
6
7   const adicionarTarefa = () => {
8     if (novaTarefa.trim() !== '') {
9       setTarefas([...tarefas, novaTarefa]);
10      setNovaTarefa('');
11    }
12  };
```

PROFKELSON.DEV
BORA CODAR?

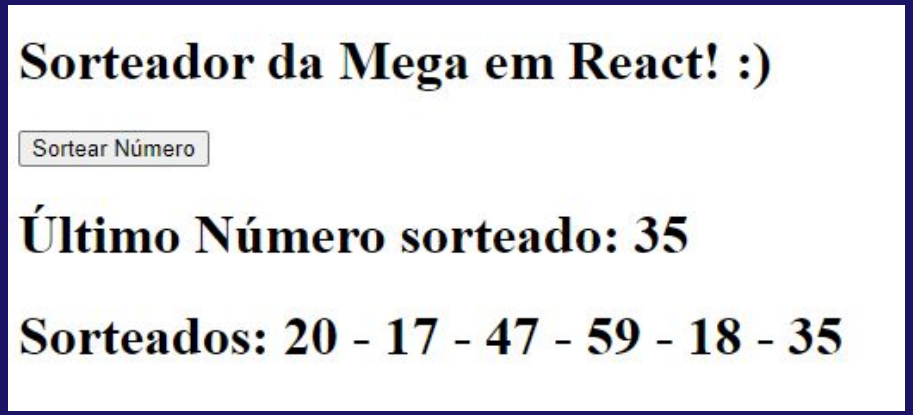
VAMOS CODAR? (6_1) [SOZINHO(A)]

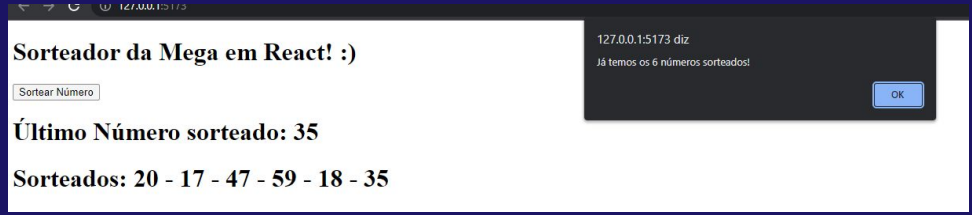
```
14    return (
15      <div className="todo-list">
16        <h2>Lista de Tarefas</h2>
17        <input
18          type="text"
19          value={novaTarefa}
20          onChange={(e) => setNovaTarefa(e.target.value)}
21          placeholder="Digite uma nova tarefa"
22        />
23        <button onClick={adicionarTarefa}>Adicionar Tarefa</button>
24        <ul>
25          {tarefas.map((tarefa, index) => (
26            <li key={index}>{tarefa}</li>
27          ))}
28        </ul>
29      </div>
30    );
31  };
```

PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? (7) [COM O PROF]

- Crie um componente chamado “HookMegaSena”
- Nele, inicialize um useState que armazena um número sorteado, que tem o estado inicial vazio.
- Crie outro useState com o estado inicial de um array vazio para armazenar os números sorteados
- Seu componente terá uma função chamada: sortearNumero
- Essa função deve abastecer o “useState” de número sorteado com um número aleatório entre 1 e 60.
    - Sintaxe: Math.floor(Math.random() * 60) + 1
- Também deve ir armazenando os valores já sorteados em um array.
    - Sintaxe: [...arrayNumerosSorteados, sorteado]
- Não permitir que um novo número seja sorteado se o array já tiver o tamanho de 6 elementos.
- Se isso acontecer, exibir um alert: “Já temos 6 números sortedos!”
- Exibir na tela as informações como demonstra o print ao lado.
- Criar o button Sortear Número para chamar a sua função.





PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? (7_1) [SOZINHO(A)]

- Agora refaça o sorteador em outro componente, agora para quina.
- Devem ser sorteados 5 números que vão de 1 até 80

PROFKELSON.DEV
BORA CODAR?

Props

- Abreviação de propriedades.
- Podemos passar valores de componente pai para um componente filho.
- No exemplo ao lado, se quisermos passar um dado para o componente “Filho”, podemos fazer isso através do props.
- Neste caso, foi passada a string “Olá, mundo!” do Pai para o Filho através de “props.mensagem”.
- Note a sintaxe do envio do dado.
- Note a sintaxe do recebimento.

jsx

```jsx
function Pai() {
  return <Filho />;
}
```

jsx

```jsx
function Pai() {
  return <Filho mensagem="Olá, mundo!" />;
}

function Filho(props) {
  return <p>{props.mensagem}</p>;
}
```

Destructuring em Props

- E se quisermos passar várias props através de um único componente?
- Existe uma solução chamada “Destructuring em Props”, ou Desestruturando Props, que vai nos permitir separar, de forma mais concisa, os dados que vamos passar de um componente pai para um componente filho.
- Não vamos mais precisar usar: props.algumDado

jsx

```jsx
function Filho(props) {
  return <p>{props.nome} tem {props.idade} anos.</p>;
}
```

Desestruturando as props nome e idade

jsx

```jsx
function Filho({ nome, idade }) {
  return <p>{nome} tem {idade} anos.</p>;
}
```

jsx

```jsx
function Pai() {
  return <Filho nome="Ana" idade={30} />;
}
```

PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? [COM O PROF/DEPOIS SOZINHO(A)]

- Crie um componente chamado “Aluno” que vai renderizar informações sobre um aluno, como: nome, email e curso.
- Em vez de acessar cada propriedade individualmente dentro do componente, utilize a técnica de destructuring para extrair as propriedades “nome”, “email” e “curso” de uma vez só
- Renderize o componente “Aluno” em um elemento da página.
- Para isso crie um array com 3 alunos em objetos literais.
- Percorra esse array através de .map para renderizar esses 3 alunos na tela.

PROFKELSON.DEV
BORA CODAR?

Fragments

- Quando precisar ter mais de um elemento pai em um componente.
- Outros elementos como uma div pode exigir que criemos mais regras de CSS desnecessárias...
- E se criarmos um elemento "vazio" que evite elementos pais que não irão ser necessariamente utilizados?

jsx

```jsx
function App() {
  return (
    <>
      <h1>Título</h1>
      <p>Parágrafo 1</p>
      <p>Parágrafo 2</p>
    </>
  );
}
```

PROFKELSON.DEV
BORA CODAR?

VAMOS CODAR? [COM O PROF/DEPOIS SOZINHO(A)]

- Refaça o exercício anterior utilizando Fragments no componente “Aluno” para evitar o elemento pai “ ”:
- Crie um componente chamado “Aluno” que vai renderizar informações sobre um aluno, como: nome, email, curso, media e status.
- Em vez de acessar cada propriedade individualmente dentro do componente, utilize a técnica de destructuring para extrair as propriedades “nome”, “email”, “curso”, “media” de uma vez só
- Renderize o componente “Aluno” em um elemento da página.
- Para isso crie um array com 3 alunos em objetos literais.
- Percorra esse array através de .map para renderizar esses 3 alunos na tela.
- Para o item “status” deve ser exibida a mensagem “APROVADO(A)” ou “REPROVADO(A)”
    - Se a média for maior ou igual a 7.00 -> Aprovado
    - Se não -> Reprovado.