Instrodução ao React.JS

Othon Oliveira

SENAC - PROA

Sumário

Criando componentes

Nossa pasta de projetos

- Vamos criar nossos Componentes React numa pasta chamada componentes, que fica no `src`
- Formato usual tipo "CamelCase" exemplo: **P**rimeiro**C**omponente.js
- Nesse arquivo criamos um função, esta vai conter o código desse componente (a lógica)
- Também precisamos "exportar" esta função, para reutilizá-la
- Bora praticar!!

Minha Imagem

Figura: Componentes React

Prmeiros passos

Primeiros passo!!

- Primeiro passo: criamos uma pasta para o projeto (no seu S.O.) ou no VSCode

Prmeiros passos

Primeiros passo!!

- Primeiro passo: criamos uma pasta para o projeto (no seu S.O.) ou no VSCode
- Segundo passo: $ npx create-react-app seu-app

Prmeiros passos

Primeiros passo!!

- Primeiro passo: criamos uma pasta para o projeto (no seu S.O.) ou no VSCode
- Segundo passo: $ npx create-react-app seu-app
- Terceiro passo: entrar na pasta “seu-app” (cd seu-app)

Prmeiros passos

Primeiros passo!!

- Primeiro passo: criamos uma pasta para o projeto (no seu S.O.) ou no VSCode
- Segundo passo: $ npx create-react-app seu-app
- Terceiro passo: entrar na pasta "seu-app" (cd seu-app)
- Quarto passo: inicia a aplicação: npm start

Prmeiros passos

Primeiros passo!!

- Primeiro passo: criamos uma pasta para o projeto (no seu S.O.) ou no VSCode
- Segundo passo: $ npx create-react-app seu-app
- Terceiro passo: entrar na pasta "seu-app" (cd seu-app)
- Quarto passo: inicia a aplicação: npm start
- Quinto passo: agora você cria a pasta componentes

Estrutura de Diretórios do Projeto - Exemplo

```
seu-app/
├── README.md
├── node_modules/
├── package.json
├── public/
│   ├── index.html
│   └── favicon.ico
├── src/
│   ├── App.js
│   ├── index.js
│   ├── logo.svg
│   └── componente/
│       └── MeuComponente.js
└── .gitignore
```

Partes de um componente

Terceira parte

Acompanhe também pelo aquivo que o prof vai disponibilizar

MeuComponente.js

```
1  import React from 'react';
2
3  function MeuComponente() {
4    return (
5      <div>
6        Meu primeiro componente
7      </div>
8    );
9  }
10
11 export default MeuComponente;
```

outra forma de: MeuComponente.js

Isso está na mesma estrutua de pastas

```
1 const MeuComponente = () => {
2     return(
3         <div>
4             <h1>Meu primeiro componente</h1>
5         </div>
6     );
7 };
8
9 export default MeuComponente;
```

Utilizando o componente criado

Importando é importante

- A importação é a maneira que temos de reutilizar um componente

Utilizando o componente criado

Importando é importante

- A importação é a maneira que temos de reutilizar um componente
- Utilizamos a sintaxe: import X from ’./componente/X, onde “X” é o nome do componente

Utilizando o componente criado

Importando é importante

- A importação é a maneira que temos de reutilizar um componente
- Utilizamos a sintaxe: import X from ’./componente/X, onde “X” é o nome do componente
- Para colocar o componente importado em outro componente precisamos colocar na forma de tag: $< MeuComponente/ >$

Utilizando o componente criado

Importando é importante

- A importação é a maneira que temos de reutilizar um componente
- Utilizamos a sintaxe: import X from ’./componente/X, onde “X” é o nome do componente
- Para colocar o componente importado em outro componente precisamos colocar na forma de tag: $< MeuComponente/ >$
- Após esse passo finalizamos a importação do componente.

Usando um coponente criado

No arquivo App.js

```
1  //componentes
2  import MeuComponente from './componentes/MeuComponente';
3
4  // styles /css
5  import './App.css';
6
7  function App() {
8    return (
9      <div className="App"> // <-- className ??
10       <h1>Fundamentos React</h1>
11       <MeuComponente/> // <-- aqui vai nosso componente novo
12     </div>
13   );
14 }
15 export default App;
```

JSX

Algumas diferenças do HTML e Javascript original

- JSX é o HTML do React

JSX

Algumas diferenças do HTML e Javascript original

- JSX é o HTML do React
- É onde vamos declarar as **tags** que serão exibidas no navegador

JSX

Algumas diferenças do HTML e Javascript original

- JSX é o HTML do React
- É onde vamos declarar as **tags** que serão exibidas no navegador
- Ficam dentro do *return*... do componente

JSX

Algumas diferenças do HTML e Javascript original

- JSX é o HTML do React
- É onde vamos declarar as **tags** que serão exibidas no navegador
- Ficam dentro do *return*... do componente
- As instruções são semelhantes ao HTML e ao JS

JSX

Algumas diferenças do HTML e Javascript original

- JSX é o HTML do React
- É onde vamos declarar as **tags** que serão exibidas no navegador
- Ficam dentro do *return*... do componente
- As instruções são semelhantes ao HTML e ao JS
- Temos algums diferenças do HTML, por exemplo: class agora é **className**
- O JSX só pode ter **um elemento pai**

Exemplos de comentários

Comentários no JSX

```
1  const UmaFuncao = () => {
2   // Um comentario de uma linha
3
4   /*
5     Um comentario de varias linhas
6   */
7  return (
8         <div>
9          {/* Outro comentario */}
10          <h1>Meu React</h1>
11          <p className="Teste">Um texto</p>
12        </div>
13    );
14 };
```

JSX

Javascript "original" e JSX, trocando figurinhas

- Template Expression é o recurso que permite executar JS e JSX, interpolando variáveis

JSX

Javascript "original" e JSX, trocando figurinhas

- Template Expression é o recurso que permite executar JS e JSX, interpolando variáveis
- Isso é muito útil para seus projetos React

JSX

Javascript "original" e JSX, trocando figurinhas

- Template Expression é o recurso que permite executar JS e JSX, interpolando variáveis
- Isso é muito útil para seus projetos React
- A sintaxe e $\{AlgumcodigoemJS\}$ do componente

JSX

Javascript "original" e JSX, trocando figurinhas

- Template Expression é o recurso que permite executar JS e JSX, interpolando variáveis
- Isso é muito útil para seus projetos React
- A sintaxe e $\{AlgumcodigoemJS\}$ do componente
- Tudo que está entre as chaves .. é processado em JS e retorna um resultado

JSX

Javascript "original" e JSX, trocando figurinhas

- Template Expression é o recurso que permite executar JS e JSX, interpolando variáveis
- Isso é muito útil para seus projetos React
- A sintaxe e $\{AlgumcodigoemJS\}$ do componente
- Tudo que está entre as chaves .. é processado em JS e retorna um resultado
- Temos algums diferenças do HTML, por exemplo: class agora é **className**
- Vamos ver na prática

Estrutura de Diretórios do Projeto - Exemplo

```
seu-app/
├── README.md
├── node_modules/
├── package.json
├── public/
│   ├── index.html
│   └── favicon.ico
├── src/
│   ├── App.js
│   ├── index.js
│   ├── logo.svg
│   └── componente/
│       ├── MeuComponente.js
│       └── TemplateExpression.js
└── .gitignore
```

Arrow Function

Funções "seta"

Vamos criar esse componente??

```
const dobrar = (num) => num * 2;
console.log(dobrar(5)); // Isso imprimirá 10
```

Arrow Function

Funções "seta"

Vamos criar esse componente??

```
const dobrar = (num) => num * 2;
console.log(dobrar(5)); // Isso imprimirá 10
```

Neste caso

A função dobrar aceita um argumento "num" e retorna o dobro desse número de forma concisa.

Arrow Function

Funções "seta"

Vamos criar esse componente??

```
const dobrar = (num) => num * 2;
console.log(dobrar(5)); // Isso imprimirá 10
```

Neste caso

A função dobrar aceita um argumento "num" e retorna o dobro desse número de forma concisa.

Em resumo

As funções de seta são uma adição útil ao JavaScript, especialmente para funções curtas e simples. Elas são particularmente úteis quando você deseja preservar o contexto do this do escopo circundante e tornar seu código mais legível e conciso em muitos casos.

Para praticar

Create React App

Certifique-se de ter um projeto React configurado

```
// slide 1/3
import React, { useState } from 'react';
function Calculator() {
  const [inputValue, setInputValue] = useState(''); // Estado
  const [result, setResult] = useState(''); // Estado para
  armazenar o resultado
  // Função para lidar com a alteração do valor de entrada
  const handleInputChange = (event) => {
    setInputValue(event.target.value);
  };
// continua no proximo slide
  // Função para dobrar o valor inserido
  const doubleValue = () => {
```

Para praticar

Create React App

Certifique-se de ter um projeto React configurado

```
// slide 2/3

  // Função para dobrar o valor inserido
  const doubleValue = () => {
    const value = parseFloat(inputValue); // Converte a entrad
    if (!isNaN(value)) {
      setResult(`O dobro de ${value} é ${value * 2}`);
    } else {
      setResult('Por favor, insira um número válido.');
    }
  };
// continua no proximo slide

//continua no próximo slide
```

Para praticar

```
// slide 3/3
  return (
    <div>
      <h1>Calculadora Simples</h1>
      <input
        type="text"
        placeholder="Insira um valor"
        value={inputValue}
        onChange={handleInputChange}
      />
      <button onClick={doubleValue}>Dobrar</button>
      <p>{result}</p>
    </div>
  ); }
export default Calculator;
```

Importar o componente

Agora é só importar o componente ‘Calculator” para App.js

```
import React from 'react';
import Calculator from './Calculator';

function App() {
  return (
    <div className="App">
      <Calculator />
    </div>
  );
}
export default App;
```

E para rodar ??

Passo 1

Certifique-se de que seu servidor de desenvolvimento React esteja em execução (usando npm start ou algo semelhante).

E para rodar ??

Passo 1

Certifique-se de que seu servidor de desenvolvimento React esteja em execução (usando npm start ou algo semelhante).

Passo 2

Abra o navegador e acesse a página da aplicação. Você verá a interface da calculadora com um campo de entrada, um botão "Dobrar"e um espaço para exibir o resultado.

E para rodar ??

Passo 1

Certifique-se de que seu servidor de desenvolvimento React esteja em execução (usando npm start ou algo semelhante).

Passo 2

Abra o navegador e acesse a página da aplicação. Você verá a interface da calculadora com um campo de entrada, um botão "Dobrar"e um espaço para exibir o resultado.

Passo 3

Insira um número no campo de entrada e clique no botão "Dobrar". O aplicativo calculará o dobro do valor inserido e o exibirá.

Introdução ao JSX

O que é JSX?

JSX (JavaScript XML) é uma extensão da sintaxe do JavaScript que permite você escrever código HTML/XML dentro do código JavaScript.

Por que usar JSX?

- Facilita a criação de interfaces de usuário. - Permite o uso de componentes React. - Pode ser transformado em código JavaScript puro pelo Babel. - Melhora a legibilidade do código.

Sintaxe Básica

- Use chaves ‘‘ para incorporar expressões JavaScript. - Use ‘¡elemento¿‘ para criar elementos JSX. - Os elementos JSX podem ter atributos e valores entre aspas.

Um exemplo prático de JSX

O JSX permite incorporar valores de variáveis, criar elementos HTML como $< h1 >$ e $< p >$, e também criar listas como $< ul >$ e $< li >$. Ele facilita a construção de interfaces de usuário em React de forma legível e expressiva.

```
import React from 'react';
function App() {
  const name = 'React.js';
  return (
    <div>
      <h1>Exemplo de JSX</h1>
      <p>Bem-vindo ao {name}</p>
      <ul>
        <li>Componente 1</li>
        <li>Componente 2</li>
      </ul>
    </div>
```

Passo a passo para criar um componente

Passo 1 Desenvolvimento dos componentes

1. **Desenvolvimento local** Desenvolva os dois componentes em um ambiente local. Você pode criar os componentes em diretórios separados e desenvolvê-los usando um servidor de desenvolvimento local.
2. **Teste e comportamento** Certifique-se de que os componentes funcionem conforme o esperado e tenham o comportamento desejado em um ambiente de desenvolvimento local.

Passo 2: Configuração do Ambiente de Produção

3. Certifique-se de que os componentes funcionem conforme o esperado e tenham o comportamento desejado em um ambiente de desenvolvimento local.

Passo a passo para criar um componente

Passo 3: Configuração do Ambiente de Produção

1. Certifique-se de que os componentes funcionem conforme o esperado e tenham o comportamento desejado em um ambiente de desenvolvimento local.

Passo 4: Criação de um Aplicativo de Exemplo

1. Crie um aplicativo de exemplo que importa e usa os dois componentes. Isso permitirá que você teste os componentes em um contexto de aplicação..

Passo 5: Implantação dos Componentes

1. Implante os componentes em um servidor ou serviço de hospedagem. Você pode usar uma variedade de opções, como Netlify, Vercel, GitHub Pages, ou implantar em seu próprio servidor.

Passo a passo para criar um componente

Passo 6: Publicação dos Componentes

1. Disponibilize os componentes para uso público. Isso pode envolver a criação de um pacote npm privado, publicando-os em um repositório de pacotes privado ou até mesmo como arquivos estáticos no servidor.

Passo 7: Integração com Outros Projetos

1. Em outros projetos onde você deseja usar esses componentes, você pode instalá-los como pacotes npm ou incluir os arquivos diretamente em seu código, dependendo de como você os implantou.

Passo 8: Monitoramento e Manutenção

1. Monitore o desempenho e a funcionalidade dos componentes em produção. Esteja pronto para realizar correções e atualizações conforme necessário.

Um exemplo prático de JSX

Dentro do diretório components, crie um arquivo chamado Componente1.js:

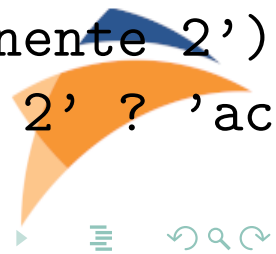
```
// touch components/Componente1.js (um editor de texto qq)
import React from 'react';

function Componente1() {
  return (
    <div>
      <h2>Componente 1</h2>
      <p>Este é o Componente 1.
      Clique em "Componente 1" para vê-lo em ação.</p>
    </div>
  );
}
export default Componente1;
```

Outro exemplo prático de JSX

Ajuste na 1ª função (que tem componente1..)

```
  return (
    <div>
      <h1>Exemplo de Clique em Componente</h1>
      <ul>
        <li
          onClick={() => handleComponentClick('Componente 1')}
          className={activeComponent === 'Componente 1' ? 'act
        >
          Componente 1
        </li>
        <li
          onClick={() => handleComponentClick('Componente 2')}
          className={activeComponent === 'Componente 2' ? 'act
        > continua abaixo ....
```



Um exemplo prático de JSX

Ajuste na 1ª função (que tem componente2..)

```
.. contunuação ...
    Componente 2
  </li>
</ul>
{activeComponent === 'Componente 1' && (
  <div>
    <h2>Componente 1</h2>
    <p>Este é o Componente 1. Clique em "Componente 1" p
  </div>
)}
{activeComponent === 'Componente 2' && (
  <div>
    <h2>Componente 2</h2>
    <p>Este é o Componente 2. Clique em "Componente 2" p
  </div>
```