

Executar os processos de codificação, manutenção e documentação de aplicativos computacionais para desktop

SENAC PE

Thiago Nogueira

Instrutor de Educação Profissional

React: JSX e elementos contêiner

Container Elements

Motivação

Por que precisamos estruturar o JSX?

- Pense no JSX como blocos de construção em um jogo ou empilhando tijolos para construir uma torre: Se você não organizar os blocos corretamente, a torre cairá.
- O JSX funciona de forma semelhante.

Ao estruturar o JSX corretamente, podemos:

- **Manter o código organizado:** Código estruturado é mais fácil de ler
- **Sites com boa aparência:** A maneira que o código é estruturado afeta diretamente a aparência do site
- **Evitar bugs:** React sem estrutura é mais propenso a erro e bugs

Container Elements

Componentes Pai

- O React tem uma regra muito importante: seu JSX deve sempre retornar apenas **um elemento pai**.
- Isso significa que dentro de cada componente, todos os seus elementos devem ser encapsulados dentro de um único contêiner.
- É como embalar itens em uma caixa — você não pode entregar vários itens separadamente sem colocá-los em uma caixa primeiro.

Container Elements

Componentes Pai

- Por exemplo, este código não funcionará:

```
function BadExample() {  
  return (  
    <h1>Hello!</h1>  
    <p>This won't work.</p>  
  );  
}
```

- o React não sabe como lidar com dois elementos irmãos (`<h1>` e `<p>`) sem um único pai envolvendo-os.
- É como tentar segurar dois copos de água em uma mão sem uma bandeja — vai ficar bagunçado!

Container Elements

Componentes Pai

- Para consertar isso, podemos envolver esses elementos em um contêiner.
- Normalmente, usamos um `<div>` (um contêiner de uso geral) para manter vários elementos juntos:

```
function GoodExample() {  
  return (  
    <div>  
      <h1>Hello!</h1>  
      <p>This works great!</p>  
    </div>  
  );  
}
```

- Agora o React entende que tudo dentro do está agrupado e pode renderizar o componente.

Container Elements

Usando elementos contêiner

- Um elemento container é como uma caixa que contém outros elementos dentro dela. Em HTML, diferentes elementos container são usados para organizar conteúdo.
- Em React, normalmente usamos `<div>`, mas há outras opções que tornam seu código mais significativo.
- Alguns dos elementos comuns de contêiner que você pode usar:
 - `<div>`: Este é o recipiente mais comumente usado. É como uma caixa simples que você pode usar em qualquer lugar.
 - `<section>`: Use isso para agrupar conteúdo relacionado. Por exemplo, você pode usá-lo para agrupar todo o conteúdo sobre um tópico específico.
 - `<article>`: Use isso quando o conteúdo fizer sentido por si só, como uma postagem de blog ou um artigo.
 - `<header>`: Isso é usado para a parte superior da sua página ou uma seção, normalmente onde fica um título ou navegação.
 - `<footer>`: Isto é para a parte inferior da sua página, onde geralmente ficam as informações de direitos autorais ou detalhes de contato.
 - `<nav>`: Use isso para links de navegação, como um menu ou índice.

Container Elements

Usando elementos contêiner

- Vamos ver um exemplo de como usar diferentes elementos de contêiner:

```
function SimpleWebPage() {  
  return (  
    <div>  
      <header>      { /* <header> contém o título principal */}  
        <h1>Welcome to My Website</h1>  
      </header>  
      <nav>          { /* <nav> contém os links de navegação */}  
        <a href="#home">Home</a>  
        <a href="#about">About</a>  
        <a href="#contact">Contact</a>  
      </nav>  
      <section>      { /* <section> contém o conteúdo "Sobre nós" */}  
        <h2>About Us</h2>  
        <p>We are a cool company that does amazing things!</p>  
      </section>  
      <footer>  
        <p>Copyright 2024</p>  
      </footer>      { /* <footer> contém o aviso de direitos autorais */}  
    </div>  
  )  
}
```


Container Elements

Elementos aninhados

- Aninhado é quando você coloca um elemento dentro de outro.
- É como colocar caixas menores dentro de maiores.
- Isso ajuda a agrupar conteúdo relacionado.

Container Elements

Elementos aninhados

Exemplo:


- Neste exemplo, temos dois elementos `<div>` aninhados dentro de uma `<section>`
- Isso mantém o conteúdo sobre produtos agrupado em uma seção.

```
function NestedExample() {  
  return (  
    <div>  
      <section>  
        <h2>Our Products</h2>  
        <div>  
          <h3>Product 1</h3>  
          <p>This is a great product!</p>  
        </div>  
        <div>  
          <h3>Product 2</h3>  
          <p>This product is even better!</p>  
        </div>  
      </section>  
    </div>  
  );  
}
```

Container Elements

Fragmentos

- Às vezes, você não quer adicionar um elemento HTML extra como um `<div>`.
- Talvez você só precise de algo para agrupar elementos sem afetar seu layout. É aí que os fragmentos entram!
- Um fragmento é um contêiner invisível que contém elementos sem adicionar nada extra ao HTML.
- Isso é útil quando você precisa agrupar elementos, mas não quer bagunçar o design com tags HTML extras.



```
import React from 'react';

function FragmentExample() {
  return (
    <React.Fragment>
      <h1>Hello!</h1>
      <p>This uses a Fragment.</p>
    </React.Fragment>
  );
}
```

Container Elements

JavaScript no JSX

- Uma das coisas mais legais sobre o JSX é que ele permite que você combine sintaxe semelhante a HTML com JavaScript.
- Para adicionar JavaScript dentro do seu JSX, você usa chaves `{}`.
- Dentro dessas chaves, você pode incluir variáveis, chamadas de função ou até mesmo cálculos simples.

`{name}` insere o valor da variável `name` dentro do JSX e `{2+2}` calcula o resultado E mostra na tela.

```
function Greeting() {  
  const name = "Alice";  
  return (  
    <div>  
      <h1>Hello, {name}!</h1>  
      <p>The result of 2 + 2 is {2 + 2}.</p>  
    </div>  
  );  
}
```

React: Criando e renderizando arquivos JSX

Criando e renderizando JSX

Overview

- Para começar a usar JSX, você precisa criar novos arquivos no seu projeto React.
- Esses arquivos conterão seu código JSX e podem ser usados para construir diferentes partes da sua interface de usuário.
- Vamos percorrer as etapas para criar um novo arquivo JSX.

Criando e renderizando JSX

Criando um novo arquivo JSX

1. Crie um novo arquivo

- Na pasta **src** do seu projeto , crie um novo arquivo.
- Você pode nomeá-lo como **MyComponent.js** ou **Header.js** , dependendo do que você está construindo.

2. Escreva o código JSX

- Abra seu novo arquivo e comece a escrever seu código JSX.
- Por exemplo, vamos criar um componente simples que exibe uma mensagem de boas-vindas.
 - Neste exemplo, **MyComponent** é um componente React que renderiza um **div** com um cabeçalho **h1** .
 - **Este arquivo exporta o componente para que ele possa ser usado em outras partes do seu aplicativo.**

```
import React from 'react';

const MyComponent = () => {
  return (
    <div>
      <h1>Welcome to My React App!</h1>
    </div>
  );
};

export default MyComponent;
```

Criando e renderizando JSX

Renderizando um novo arquivo JSX

- Após criar seu arquivo JSX, você precisa renderizá-lo em seu aplicativo para vê-lo em ação.
- Isso envolve importar o componente para outro arquivo e incluí-lo na árvore de componentes do seu aplicativo.

Criando e renderizando JSX

Renderizando um novo arquivo JSX

Importar o componente

- No arquivo principal do seu aplicativo (geralmente **App.js**), importe o componente que você acabou de criar:
- Aqui, importamos **MyComponent** e o incluimos dentro do componente **App**.
- Dessa forma, **MyComponent** será exibido onde quer que **App** seja usado.

```
import React from 'react';
import MyComponent from './MyComponent'; // Caminho do arquivo

const App = () => {
  return (
    <div>
      <MyComponent />
    </div>
  );
};

export default App;
```

Criando e renderizando JSX

Renderizando um novo arquivo JSX

Renderize a aplicação

- Por fim, certifique-se de que seu aplicativo React esteja configurado para renderizar o componente **App**.
- Isso geralmente é feito no arquivo **index.js**

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App'; // Importa o component App

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root') // Aqui é onde é renderizado em HTML
);
```

Exercícios

Exercícios

Instruções


- Seguir o exercício roteiro da aula 16
- Documentar o que foi feito no README.md
- A documentação deve ter a explicação de cada bloco de código do novo componente criado bem como a explicação do que é o novo componente
- Enviar o exercício roteiro da aula 16 para um repositório no GitHub
- Anexar link do repo na atividade do Teams


*“Ensinar é impregnar
de sentido o que
fazemos a cada
instante”*


Paulo Freire

Obrigado!

Thiago Nogueira

 [linkedin.com/tdn](https://www.linkedin.com/tdn)

 thiago.nogueira@pe.senac.br

 (81) 9 9627-0419