Cruzeiro do Sul Educacional

Introdução ao *React*

Conteudista
Prof. Me. Marco Antonio Sanches Anastacio

Revisão Textual
Esp. Pérola Damasceno

OBJETIVOS DA UNIDADE

- Entender o que é o *React* e como utilizá-lo na construção de páginas *web* interativas;
- Compreender os principais conceitos associados ao uso do *React* e como utilizar o JSX no desenvolvimento *web*.

Atenção, estudante! Aqui, reforçamos o acesso ao conteúdo *on-line* para que você assista à videoaula. Será muito importante para o entendimento do conteúdo.

Este arquivo PDF contém o mesmo conteúdo visto *on-line*. Sua disponibilização é para consulta *off-line* e possibilidade de impressão. No entanto, recomendamos que acesse o conteúdo *on-line* para melhor aproveitamento.

Afinal, o que é *React*?

É inegável o quanto o *JavaScript* se tornou uma linguagem fundamental para o desenvolvimento de páginas dinâmicas e interativas quando falamos sobre interface de usuário. Apesar de sua versatilidade, com o tempo foram surgindo diversas bibliotecas e *frameworks* que tornaram mais eficiente e rápida a construção de páginas *web*, com recursos que vão desde o gerenciamento de estado e interação com APIs externas até a criação e reutilização de componentes.

Leitura

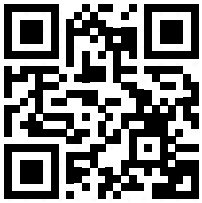
O que é uma API? Leia mais sobre no material a seguir.

Dentre as bibliotecas, o *React* destacou-se por sua simplicidade e eficiência no desenvolvimento de complexas interfaces de usuário, permitindo desde a criação de componentes reutilizáveis em *JavaScript* até o gerenciamento de estado, que permite a atualização da interface de forma eficiente e sem necessidade de atualizar a página inteira.

O *React* está entre as bibliotecas mais populares no ecossistema *JavaScript*, aparecendo como segunda na preferência dos mais de 70.000 desenvolvedores que responderam a uma pesquisa realizada pelo *StackOverFlow*, em maio de 2022.

Site

Conheça em detalhes a pesquisa do *StackOverFlow*.



Criada em 2011 por um engenheiro do *Facebook* para ser utilizado no *feed* de notícias, a biblioteca *React* foi apresentada ao público como código aberto dois anos depois, em 2013, na JSConf US. Mantido pelo *Facebook*, a grande comunidade de desenvolvedores trabalhando com o *React* não para de crescer e produzir recursos e ferramentas para facilitarem o desenvolvimento de aplicativos *web* de alta *performance* e escaláveis.

Em 2015, também na JSConf US, foi lançada a biblioteca *React Native*, destinada à criação de aplicações para dispositivos móveis que se utilizam de funções nativas das plataformas *Android* e iOS.

Site

O JSConf é uma organização de conferências *JavaScript* de todo o mundo.

Para a construção de interfaces de usuário, o *React* apropria-se de uma abordagem baseada em componentes e de uma representação virtual da árvore de elementos *DOM (Document Object Model)* da página, que chamaremos de *Virtual* DOM.

A principal vantagem dessa abordagem é que cada componente é uma unidade independente que pode ser reutilizada e combinada com outros componentes para criar uma interface completa. Já a utilização de uma *Virtual* DOM permite atualizar somente as partes da interface que foram modificadas, sem precisar recarregar a página inteira.

E, por falar nisso, a seguir, vamos sintetizar as principais vantagens da utilização do *React* em nossos projetos:

- **Reutilização de componentes:** torna o desenvolvimento mais eficiente;
- ***Performance:*** o *Virtual* DOM permite atualizações mais eficientes e rápidas da interface, sem recarregar a página inteira;
- **Facilidade de manutenção:** a estrutura baseada em componentes torna o código mais organizado e fácil de manter;
- **Comunidade ativa:** facilita ao desenvolvedor encontrar soluções para problemas e aprender novas técnicas.

Dentre as desvantagens, não podemos deixar de destacar a alta curva de aprendizagem para iniciantes, que podem sentir dificuldade na utilização dos componentes e do *Virtual* DOM.

Primeiros passos com *React*

Para utilização do *React* em um projeto, precisamos inicialmente incluir a biblioteca em seu código. Isso pode ser feito de duas formas:

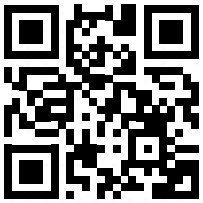
- **Gerenciador de pacotes:** podemos utilizar o npm ou yarn para instalar e gerenciar as dependências do projeto. Certifique-se que você possui o **npm** instalado e digite o seguinte comando no terminal:

```
npm install react react-dom
```

Esse comando fará a instalação das bibliotecas *React*, que é responsável pela criação e gerenciamento de componentes, e *ReactDOM*, que se encarrega da renderização dos componentes na página.

Leitura

O que é npm? Introdução básica para iniciantes.



- **Incluir as bibliotecas diretamente na página HTML:** ao iniciar um projeto *React*, importar a biblioteca a partir de uma rede de distribuição de conteúdo (CDN) pode ser uma ótima opção. Em apenas um minuto podemos configurar tudo, como mostraremos a seguir:

```
<script  src="https://unpkg.com/react/umd/react.development.js">
</script>
<script src="https://unpkg.com/react-dom/umd/react-dom.development.js"></script>
```

Leitura

Para fins didáticos, nessa Unidade utilizaremos a segunda opção, ou seja, iremos incluir as bibliotecas diretamente em uma página HTML. Entretanto, caso você deseje construir uma nova aplicação totalmente do zero em *React*, visite o tutorial disponível a seguir.

Nosso primeiro exemplo será uma página simples que utilizaremos para trabalhar alguns dos conceitos tratados:

```
1.<html>
2.<head>
3.  <meta charset="UTF-8">
4.  <title>Primeiros passos com React.js</title>
5.  <script src="https://unpkg.com/react/umd/react.development.js"></
    script>
6.  <script src="https://unpkg.com/react-dom/umd/react-dom.develo-
    pment.js"></script>
7.</head>
8.<body>
9.  <div id="root"></div>
10. <script>
11.   const root = ReactDOM.createRoot (document.querySelector('#root'));
12.   const element = React.createElement;
13.   root.render(element('h1', null, 'Olá, bem-vindo ao React!!!'));
14. </script>
15.</body>
16.</html>
```

O código pareceu um pouco confuso para você? Não se preocupe, pois, no Quadro 1, vamos comentar cada parte dele:

Quadro 1 – Exemplo comentado

Linha(s)	Descrição
linhas 5 e 6	Importamos as bibliotecas do .js e ReactDOM.js por meio de dois *scripts*.
linha 9	Definimos um elemento HTML com **id = "root"**, que será utilizado como ponto de entrada para a renderização dos elementos do *React*.
Linha 11	Utilizamos o método **createRoot()** do ReactDOM para criar um **"root"** *React* no elemento HTML com **id = "root"**.
Linha 12	Utilizamos o método **React.createElement()** para criar um elemento HTML.
Linha 13	Chamamos o método **render()** no **"root"** criado anteriormente, passando o elemento **<h1>** e o texto **"Olá, bem-vindo ao React!!!"** como parâmetro. Esse método será responsável por renderizar o elemento HTML na página, tornando-o visível para o usuário.

Componentes

Não seria exagero afirmar que os componentes são o coração e essência do *React*, pois permitem o desenvolvimento mais eficiente na medida em que a interface de usuário é dividida em partes que são independentes e reutilizáveis.

Na prática, os componentes fazem o encapsulamento da lógica e interface do usuário em elementos HTML, que trabalham isoladamente, provendo códigos independentes e reutilizáveis. Em outras palavras, um componente é uma unidade de códigos que pode ser combinada com outras, com a finalidade de criar interfaces mais complexas.

O modo mais simples para se definir um componente em *React* é escrevê-lo como uma função *JavaScript*:

```
<div id="root"></div>
<script>
  const { useState, createElement } = React;
  function OlaMundo(props) {
    const p = React.createElement("p", null,
                                  `Olá, ${props.name}!`);
    return p;
  }


  const root = ReactDOM.createRoot(
            document.querySelector("#root"));
  root.render(createElement(OlaMundo, {name: "Marco"}));
</script>
```

Nesse exemplo, o componente **OlaMundo** é definido por uma função que utiliza o método **createElement** para retornar um elemento **p**. A função recebe uma propriedade (**props**), chamada **name**, e o componente usa o método **useState** para definir um estado para **name**, que é atualizado quando passamos o nome da pessoa para o componente quando ele for renderizado.

Perceba que nosso exemplo requer a importação das funções **useState** e **createElement** da biblioteca *React*, o que é feito na linha: **const { useState, createElement } = React.**

Enquanto a função **useState** é responsável por criar um estado interno em um componente *React*, fornecendo uma forma de atualizar esse estado, a função **createElement** é usada para criar elementos *React* que compõem as interfaces em um aplicativo.

Saiba Mais

Uma boa prática para tornar nosso código mais legível é importar apenas as funcionalidades específicas que precisamos da biblioteca do *React* e usá-las diretamente em nosso código, sem precisar chamar a biblioteca toda vez.

Finalmente, o método **render** é chamado para renderização do componente, passando-se o resultado da chamada **createElement(OlaMundo, {name: "Marco"})** como argumento.

Saiba Mais

O parâmetro **props** da função é um objeto que armazena dados para uso na criação da marcação HTML. No exemplo citado, **props.name** é o nome de uma pessoa! Observe, também, que o nome de componentes segue a sintaxe **PascalCase** (exemplo: **OlaMundo**).

Nosso exemplo utiliza-se de um componente funcional que é mais simples e retorna um elemento por meio de funções *JavaScript*. No entanto, o *React* também permite a definição de componentes de classe, que são mais complexos que os funcionais e se utilizam de uma classe ES6. Os componentes de classe serão apresentados no final desta Unidade.

A implementação do *React* utilizada neste exemplo admite três parâmetros para criar a marcação HTML:

```
const elementoHTML = React.createElement(
                "el",
                {objeto javascript},
                "conteúdo"
              );
```

O primeiro atributo (**el**) destina-se à criação do elemento HTML; o segundo define um ou mais atributos para o elemento; e o terceiro é o conteúdo do elemento. O resultado é armazenado em uma constante ES6 que, neste exemplo, foi chamada de **elementoHTML**.

O segundo parâmetro pode ser declarado vazio **{}, null** ou conter regras CSS. Já o terceiro poderá ser vazio, conter um texto, ou outro elemento HTML, resultando em aninhamentos em tantos níveis quanto desejarmos. Veja o exemplo a seguir:

```
const elemento = React.createElement(
        'h1',
         {className : 'titulo'},
        'Olá mundo React!'
      );
```

Importante

Observe que o segundo parâmetro se utiliza da palavra reservada **className** e não **class** para definir o atributo HTML.

Ciclos de Vida em *React*

O ciclo de vida de um aplicativo é definido em programação como sendo o período entre a sua inicialização e finalização.

No contexto do *React*, os ciclos de vida permitem gerenciar o comportamento de um componente por meio de métodos que se destinam a executar tarefas específicas no instante mais apropriado da vida do componente.

Existem quatro principais fases do ciclo de vida de um componente:

- **Inicialização:** o componente está sendo preparado para a renderização;
- **Montagem:** o componente é inserido na árvore do DOM pela primeira vez;
- **Atualização:** o componente é atualizado com novas **props** ou estados;
- **Desmontagem:** o componente é removido da árvore do DOM.

Os principais métodos utilizados no ciclo de vida do *React* são:

- **constructor():** executado quando o componente é criado e inicializa o estado do componente e outras propriedades;

- **componentDidMount():** executado após o componente ser montado na página. Neste ponto, deve-se executar qualquer inicialização que o componente precise;

- **componentDidUpdate(prevProps, prevState):** executado após o componente ser atualizado. Neste ponto, deve-se executar qualquer ação que dependa do estado anterior do componente;

- **componentWillUnmount():** executado quando o componente está prestes a ser removido da página. Neste ponto, deve-se limpar qualquer recurso que o componente tenha utilizado.

Observe o exemplo a seguir, que, para fins didáticos, teve seu código dividido em dois arquivos:

ciclo_de_vida.html

```
<html>
<head>
   <title>Exemplo Ciclo de Vida</title>
  <script src="https://unpkg.com/react/umd/react.development.js"></
script>
    <script src="https://unpkg.com/react-dom/umd/react-dom.develo-
pment.js"></script>
</head>
<body>
   <div id="root"></div>
   <script src="ciclo_de_vida.js"></script>
</body>
</html>
```

ciclo_de_vida.js

```
class CicloDeVida extends React.Component {
  constructor(props) {
    super(props);
    this.state = { contador: 0 };
    console.log("Construtor");
  }
  componentDidMount() {
    console.log("Componente foi montado");
  }
  componentDidUpdate(prevProps, prevState) {
    console.log("Componente foi atualizado");
    console.log("Estado anterior:", prevState);
    console.log("Estado atual:", this.state);
  }
  componentWillUnmount() {
    console.log("Componente será desmontado");
  }
  incrementar = () => {
    this.setState({ contador: this.state.contador + 1 });
  };
  render() {
    console.log("Renderização");
    return React.createElement("div", null,
      React.createElement("h1", null,
            "Contador: ", this.state.contador),
      React.createElement("button",
             { onClick: this.incrementar }, "Atualizar")
    );
  }
}

const root = ReactDOM.createRoot(document.querySelector("#root"));
root.render(React.createElement(CicloDeVida));
```

O exemplo cria um componente chamado **CicloDeVida**, composto por um contador e um botão que incrementa o contador a cada clique. O método **render()** renderiza o contador e o botão, que possui um evento de clique, responsável por chamar a função **incrementar()**. A cada clique, o estado do componente é atualizado e o contador é renderizado novamente.

O ciclo de vida do componente é gerenciado pelos quatro métodos implementados:

- **constructor():** o componente foi criado tem seu estado inicializado. O console do navegador exibe a mensagem "Construtor";

- **componentDidMount():** o componente foi montado na página. O console do navegador exibe a mensagem "Componente foi montado";

- **componentDidUpdate():** controla ações com base no estado ou propriedades anteriores e atuais do componente toda vez que ele é atualizado. O console do navegador exibe a mensagem "Componente foi atualizado" e os estados anterior e atual;

- **componentWillUnmount():** executado antes do componente ser removido da página. O console do navegador exibe a mensagem "Componente será desmontado".

A Figura 1 mostra a renderização do exemplo no navegador *Chrome*, com a aba de ferramentas do desenvolvedor aberta à direita:

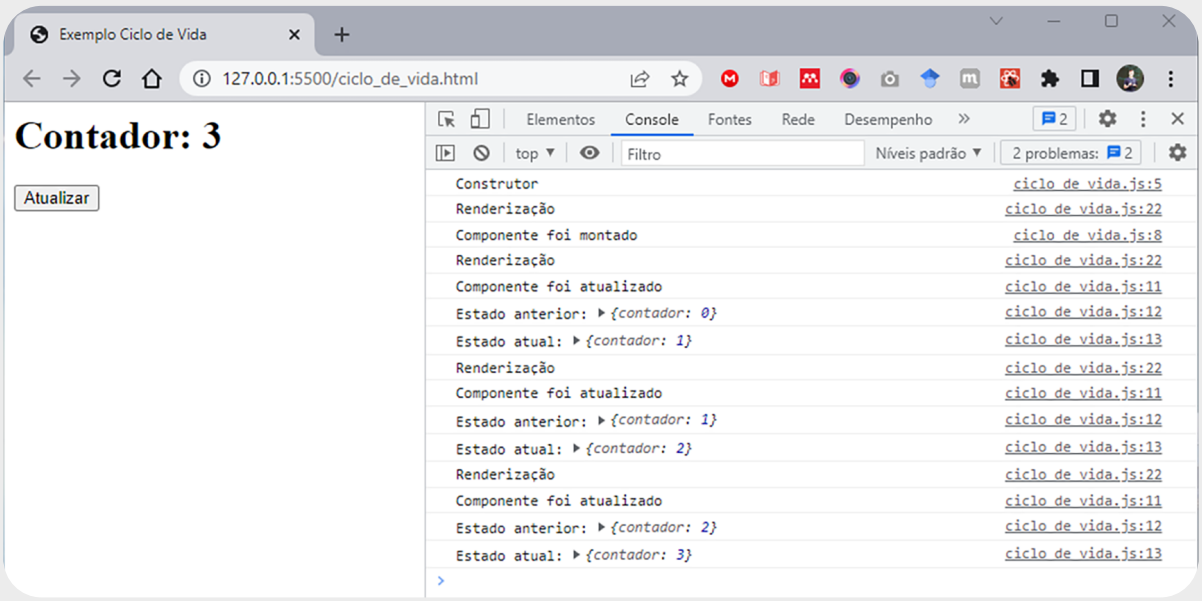


Figura 1 – Ciclo de vida React

#ParaTodosVerem: imagem do navegador *Chrome* da *Google* com fundo na cor branca. Ao lado esquerdo, há um título com o texto "Contador: 3" e, logo abaixo, um botão cinza que contém o texto "Atualizar". À direita, é exibida a aba de ferramentas do desenvolvedor, com a guia "Console" selecionada com um traço na cor azul na parte inferior. Logo abaixo, uma tabela exibe, do lado esquerdo, os estados do componente *React* e, à direita, o nome da aplicação (ciclo_de_vida.js) seguida pela linha do código que está sendo executada. Fim da descrição.

O que é o JSX?

JSX é o acrônimo de *JavaScript* XML, uma tecnologia criada com o objetivo de simplificar a criação de elementos por meio de códigos *JavaScript*, muito utilizada no desenvolvimento de interface do usuário como componentes do *React*.

Sua sintaxe declarativa, que permite criar um componente com uma estrutura HTML e CSS escrita em *JavaScript*, proporciona o desenvolvimento de interfaces mais intuitivas com códigos mais fáceis de ler e escrever, sem necessidade de recorrer à concatenação de ***Strings*** ou métodos para criação de elementos dinâmicos em HTML.

Considere, por exemplo, uma página HTML que exiba dinamicamente um elemento **h1** com o texto "Hello JSX com *React*". Na Figura 2, observe no código a tag <h1>Hello JSX com React</h1>.

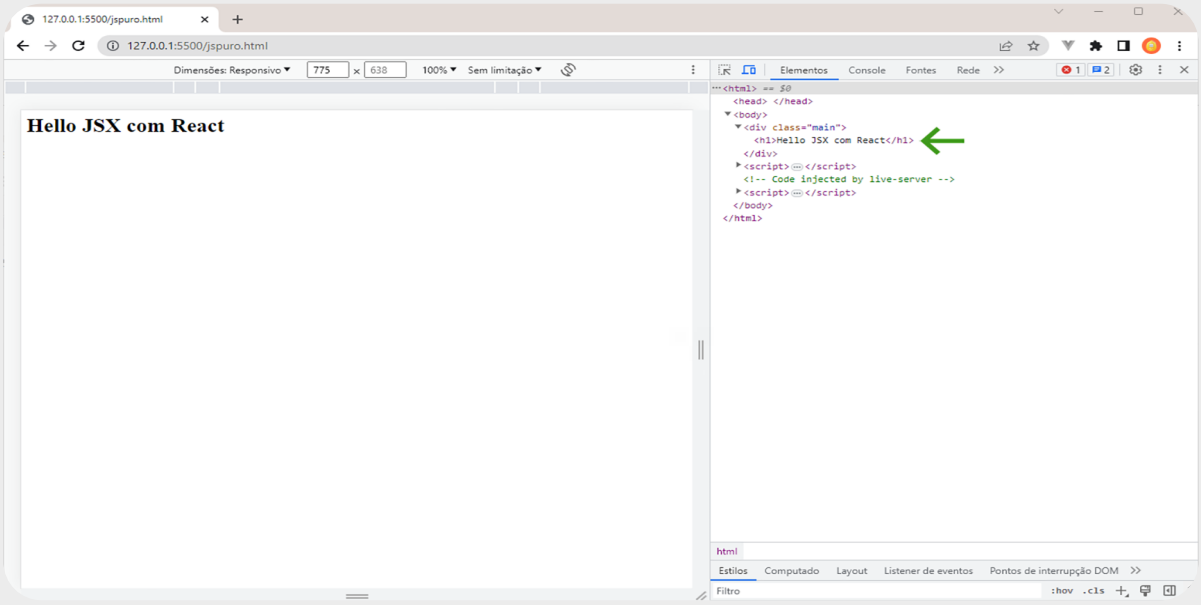


Figura 2 – *Hello* JSX com *React*

#ParaTodosVerem: imagem do navegador *Chrome*, da Google, com fundo na cor branca. Ao lado esquerdo, há um título com o texto "Hello JSX com *React*". À direita, é exibida a aba de ferramentas do desenvolvedor, com a guia "Elementos". Logo abaixo, uma seta na cor verde aponta para o elemento h1. Fim da descrição.

O código para criação de uma marcação HTML com uso de sintaxe de *JavaScript* puro ficaria assim:

```
<div id="root"></div>
<script>
  //Código JS para criar o elemento h1
  const elemento = document.createElement("h1");
  const titulo = document.createTextNode("Hello JSX com React");
  elemento.appendChild(titulo);
  //Código JS para anexar o elemento à página
  document.querySelector("#root").appendChild(elemento);
</script>
```

O mesmo resultado poderia ser obtido com a sintaxe JSX:

```
<div id="root"></div>
<script type="text/babel">
  //Código JSX para criar o elemento
  const elemento = <h1>Hello JSX com React</h1>;
  //Código React para anexar o elemento à página
   ReactDOM.createRoot(document.querySelector('#root')).render(elemento);
</script>
```

Site

Acesse o código completo dos exemplos citados no repositório disponível a seguir.

Perceba que estamos substituindo três linhas de código por apenas uma! Isso parece bastante promissor, você não acha?

Na prática, o *React* utiliza-se de uma programação declarativa para interagir com o DOM, o que significa que não é feita a manipulação direta do DOM. Somente precisamos indicar o que é preciso fazer que o *React* se encarrega de transformar a instrução *JavaScript* e executá-la corretamente.

No exemplo, a primeira implementação que fizemos utilizou-se de uma programação declarativa, ou seja, declaramos duas variáveis e associamos ao elemento HTML.

Os arquivos contendo códigos JSX podem ser salvos com as extensões .js ou .jsx, porém, para que os códigos sejam interpretados pelos navegadores, é necessário um tradutor, ou transpilador de código. O *React* usa o *Babel* para ler o conteúdo JSX e transformá-lo em *JavaScript* puro.

Leitura

Saiba mais sobre o que é babel lendo o material a seguir.

Observe que a sintaxe JSX permite a inserção de atributos, o que facilita o uso de CSS para formatação dos componentes. Vejamos um exemplo completo de tudo que vimos até agora sobre o JSX:

```
<html>
<head>
   <meta charset="UTF-8">
   <title>Primeiros passos com React.js</title>
   <script src="https://unpkg.com/@babel/standalone/babel.js"></script>
   <script src="https://unpkg.com/react/umd/react.development.js"></
script>
    <script src="https://unpkg.com/react-dom/umd/react-dom.develo-
pment.js"></script>
   <style>
      .hello {
         color: #c05959;
         font-size: 1.5em;
         text-transform: uppercase;
      }
   </style>
</head>
<body>
   <div id="root"></div>
   <script type="text/babel">
      function Card(props) {
         const element = (
            <div>
               <h1 className="hello"> JSX com React </h1>
               <p>Olá mundo, {props.name}</p>
            </div>
         )
         return element;
      }
      function Button(props) {
         return <button
```

```
                onClick={props.onClick}>{props.label}
            </button>;
        }
        const root = ReactDOM.createRoot(
                    document.querySelector('#root'));
        root.render(<div>
                <Card name="JSX" />
                <Button label="Clique aqui" onClick={()
                    => alert('Clicou no botão')} />
            </div>
        );
    </script>
</body>
</html>
```

No exemplo, é importante destacar a utilização do tradutor (transpilador) **babel.js:**

- Primeiro, importamos a biblioteca de um repositório: **<script src= "https://unpkg.com/@babel/standalone/babel.js"></script>**;
- A seguir, devemos especificar que nosso *script* se utiliza da biblioteca: **<script type="text/babel">.**

Agora, vamos ao código:

- Os componentes criados **Card** e **Button** são definidos como funções JavaScript que recebem propriedades (**props**);
- O componente **Button** recebe a propriedade que define sua **label** e o método **onClick**, responsável pelo evento clique;
- Na renderização, passamos os valores das **props**, o que permite uma personalização dos componentes exibidos na página;
- Observe que devemos utilizar **className** e não **class** para aplicar o CSS ao elemento **h1** do **HTML**.

Antes de encerrarmos esta Unidade, no próximo exemplo vamos retomar o componente de classe do *React* e explorar alguns recursos que essa sintaxe oferece:

```
<html>
<head>
   <meta charset="UTF-8">
   <title>Primeiros passos com React.js</title>
   <script src="https://unpkg.com/@babel/standalone/babel.js"></script>
   <script src="https://unpkg.com/react/umd/react.development.js"></
script>
    <script src="https://unpkg.com/react-dom/umd/react-dom.develo-
pment.js"></script>
</head>



<body>
   <div id="root"></div>



   <script type="text/babel">
      class Carro extends React.Component {
         constructor(props) {
            super(props);
            this.state = {
               marca: "Ford",
               modelo: "Camaro",
               cor: "amarela",
               ano: 1975
            };
         }
         render() {
            return (
                  <div>
                  <h1>Meu carro JSX</h1>
                  <ul>
```

```
                <li>Marca: {this.state.marca}</li>
                <li>Modelo: {this.state.modelo}</li>
                <li>Cor: {this.state.cor}</li>
                <li>Ano: {this.state.ano}</li>
              </ul>
            </div>
          );
        }
      }


      const root = ReactDOM.createRoot(
                    document.getElementById('root'));
      root.render(<Carro />);
    </script>
  </body>
```

Neste exemplo, criamos uma classe "Carro", que estende a classe **React. Component**. O método **constructor(props)** é utilizado para inicializar o estado interno do componente (**state**) com as propriedades (**props**).

O método **render()** é o responsável pela estrutura HTML do componente, que se utiliza das propriedades armazenadas em **this.state** para exibir as características do carro. Observe que o uso do **state** pode ser bastante útil quando precisamos armazenar e manipular dados recebidos de **props** ao longo da vida do componente.

Apesar de ser opcional, o uso do JSX, em *React*, é altamente recomendado, pois torna a escrita e manutenção de código muito mais fácil e eficiente, já que utiliza marcação HTML, CSS e JavaScript em um único arquivo.

Além da simplicidade da sintaxe, outras vantagens da utilização do JSX são:

- **Renderização eficiente:** permite que o *React* faça atualizações de forma mais rápida e eficiente do que outras abordagens de renderização;

- **Validação de sintaxe:** como é compilado para *JavaScript*, o JSX é validado pela sintaxe do *JavaScript*, evitando erros comuns de digitação e garantindo que o código esteja correto antes de ser executado;

- **Facilidade de integração com outras bibliotecas:** é compatível com outras bibliotecas *JavaScript*, como *jQuery* e *Bootstrap*, facilitando a integração do *React* com outras ferramentas e tecnologias;

- **Prevenção de ataques de injeção de código:** possui recursos integrados de segurança que garantem que os dados sejam exibidos como texto, e não como código executável.

Em Síntese

O JSX é uma extensão da sintaxe do *JavaScript* que permite escrever componentes *React* de forma mais expressiva, uma vez que permite a mistura de *JavaScript* e HTML em um único arquivo. Porém, lembre-se que o JSX precisa ser convertido em *JavaScript* puro (transpilado) antes de ser executado, já que não é uma sintaxe reconhecida pelo navegador ou pelo interpretador *JavaScript* padrão.

MATERIAL COMPLEMENTAR

Sites

***React* – Uma Biblioteca *JavaScript* para Criar Interfaces de Usuário**
Visite e explore a página oficial do *React* e aprofunde seus estudos.
https://bit.ly/4872GTH

W3Schools – React
https://bit.ly/3Z9MJIx

React Native
Crie aplicativos nativos para *Android*, iOS e muito mais usando o *React*.
https://bit.ly/3PgUNmb

Leituras

O que é *Babeljs*
Saiba mais sobre a importância do *Babeljs* para o *JavaScript* moderno.
https://bit.ly/461fzxr

REFERÊNCIAS BIBLIOGRÁFICAS

CLARK, R. *et al*. **Introdução ao HTML5 e CSS3:** a evolução da web. Rio de Janeiro: Alta *Books*, 2014.

MDN WEB DOCS. **Começando com *React*.** [s.d.]. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started>. Acesso em: 01/07/2023.

SILVA, M. S. ***React:*** Aprenda praticando. São Paulo: Novatec, 2022.

STEFANOV, S. **Primeiros passos com *React:*** construindo aplicações web. São Paulo: Novatec, 2016.

TERUEL, E. C. **HTML 5:** guia prático. 2. ed. São Paulo: Erica, 2013.