Leandro de Pinho Monteiro apresenta:

Curso Gratuito de React

atualização: 02/2021

Como funciona este curso?

Este é um curso prático e rápido para interessados em aprender React baseado na documentação oficial. Muitas pessoas ficam confusas com a parte da documentação traduzida para o português ou com os saltos de conhecimento necessários para entender cada exemplo presente no site. Neste curso apresento os conhecimentos e exemplos de forma mais organizada e comentada para lhe ajudar na missão de aprender **React**.

Por que aprender React?

Aprender **React** permite ao profissional trabalhar em diferentes frentes. Um caminho é ser desenvolvedor **front-end**, programando sites e *widgets*, componentes interativos que podem ser facilmente adicionados a qualquer site, ou pode trabalhar no lado do servidor, como desenvolvedor **back-end** (com Node.js) ou, ainda, criando aplicativos com **React Native**.

Para quem é este curso?

Para quem já é desenvolvedor e tem boas noções da linguagem de programação JavaScript (JS). Se não é o seu caso, comece por aqui: https://universidadedatecnologia.com.br/a-linguagem-javascript/

1. O que é React?

React é uma biblioteca feita em JS para criar **interfaces de usuário** (UI) em sites e aplicativos. Seu objetivo é facilitar a criação de interfaces interativas, aquelas em que o usuário interage com uma ação e o sistema retorna uma reação. Trata-se de um recurso incrível para um desenvolvedor, o qual pode ser usado no front-end, back-end ou no desenvolvimento de apps.

2. Como começar?

Existem diferentes forma de começar a programar em React e a mais fácil de todas é usando um editor de código no *browser*, como o CodePen. Considere o tradicional *Hello World*, exemplo com o código necessário para imprimir o texto "Hello World" na tela. Neste caso, na tela significa no navegador. O exemplo pode ser rapidamente acessado pelo link abaixo:

https://pt-br.reactjs.org/redirect-to-codepen/hello-world

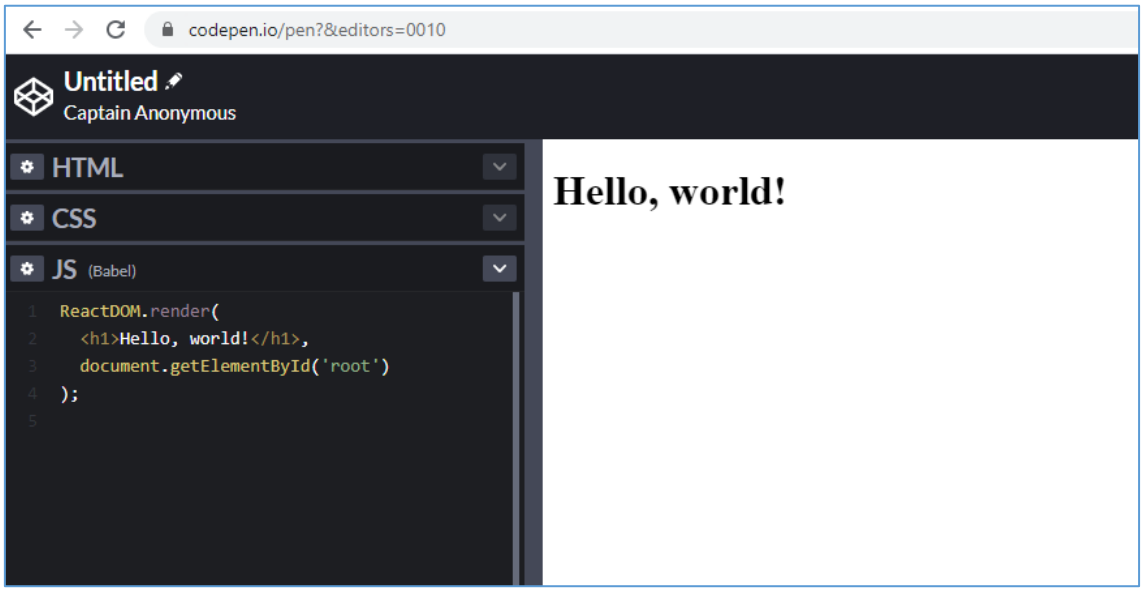

Figura 1: exemplo *Hello World* no Code Pen.

Daí o aluno vai lá, muda o texto de “Hello World” para “Olá Mundo”, ou qualquer texto que quiser, e nota a diferença no resultado. É efetivamente colocar a mão na massa nos primeiros minutos de aprendizado.

Sem dúvida é uma forma interessante de aprender, mas é preciso também saber rodar os exemplos no seu computador. Testar a sintaxe da linguagem em um editor é muito útil e interessante, porém, no começo, para aprender como funciona o fluxo de trabalho com **React**, é importante criar seus arquivos do zero pelo menos algumas vezes. E é isso que nós faremos aqui.

3. Escolhendo um editor ou IDE para trabalhar com React

Se você procura um IDE, isto é, um ambiente de desenvolvimento integrado, use o Reactide. Acesse o site, baixe e instale. Você conseguirá trabalhar com o código e com o *preview* do resultado na tela lado a lado, similar à interface do CodePen.

Agora, se você quer só um editor de código para trabalhar com o React, rodando diretamente os arquivos no navegador, use o Visual Studio Code.

Eu gosto mais da segunda opção, mas ambas são produtivas e, caso não goste de nenhuma, existem outras.

4. Como codificar (na teoria)?

Existem duas formas de programar em **React**.

A primeira é usando **React puro**, ou seja, usando JavaScript.

A segunda forma é usando **JSX**, que significa, Java Script Extended.

5. Como codificar (na prática)?

Primeiro, crie um arquivo HTML com as *tags* convencionais (*html*, *head* e *body*) e salve-o. Em seguida, importe as bibliotecas usando a *tag* <script>:

```
<script src="https://unpkg.com/react@17/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
```

Ao final desta etapa você estará com o arquivo HTML pronto para receber código em **React** dentro da *tag* <script>.

> Alternativamente, para separar o código em **React**, você pode criar um novo arquivo JS (como seuarquivocomreact.js), importando-o no arquivo HTML com a *tag* <script> assim como fez no primeiro passo. Nesse caso, você precisará colocar seus arquivos em um servidor HTTP para evitar o bloqueio *cross-origin*. Se não está acostumado com este problema, siga adicionando código React no mesmo arquivo HTML.

Não precisa sair fazendo, logo apresentarei o primeiro código de exemplo e comentarei cada linha dele.

6. O que é JSX?

JSX é uma extensão de **sintaxe** para **JavaScript** que produz elementos do **React**. Quando você terminar as instruções da seção 5, o arquivo aceitará a entrada de código em **React puro** (JavaScript), mas não em **JSX**. Para usar esta extensão é preciso adicionar no arquivo HTML, dentro da tag <head>, a seguinte linha:

```
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

7. Devo aprender React Puro ou JSX?

Você encontrará em vários trechos da documentação oficial que o React **não requer** o uso do JSX. Ou seja, é um recurso opcional do React. Você não precisa obrigatoriamente usá-lo. Esta afirmação parece tentadora ao ver as primeiras linhas de código em JSX pelo fato delas parecerem um pouco diferentes das usadas em outras linguagens de programação, mas não se engane: **o uso do JSX facilita as coisas e você deve aprendê-lo**. Deixei neste curso alguns exemplos equivalentes escritos em React puro e em JSX. A diferença de produtividade é notável.

Outro argumento é que, segundo a documentação, a maioria das pessoas acha prático usar JSX como uma **ajuda visual** quando se está trabalhando com uma interface dentro do código em JavaScript. Um último motivo é que JSX permite ao React mostrar mensagens mais úteis de erro e de aviso.

8. Exemplo 1: Hello World em React com JSX

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8" />
5.  <title>Hello World</title>

6.  <!-- Adiciona o React -->
7.  <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
8.  <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>

9.  <!-- Adiciona o JSX -->
10. <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
11. </head>
12. <body>
13. <div id="root"></div>

14. <script type="text/babel">

15. ReactDOM.render(
16.      <h1>Hello, world!</h1>,
17.      document.getElementById('root')
18. );
19. </script>

20. </body>
21. </html>
```

Exemplo 1: código completo do *Hello World* em React, usando JSX.

A primeira linha é o padrão/versão do arquivo HTML. Quando ela começa assim, quer dizer ao *browser* que o documento HTML é da versão 5.

As linhas 2, 3, 11, 12, 20 e 21 correspondem a abertura e fechamento das principais *tags* HTML: <html>, <head>, <body>. As linhas 6 e 9 são comentários. As linhas 4 e 5 trazem *tags* bem conhecidas em HTML.

Tirando todas essas linhas, as de relevância para este estudo são as bibliotecas React, importadas nas linhas 7 e 8, bem como a linha 10 que adiciona a extensão JSX.

As linhas 14 a 19, mais especificamente, o conteúdo entre elas, contêm de fato o código em **React**. Assim, podemos dizer que o menor exemplo de React é o seguinte código:

```
ReactDOM.render(
       <h1>Hello, world!</h1>,
       document.getElementById('root')

     );
```

Código 1: trecho em React chama o método render() com dois argumentos.

Para fechar a análise das linhas do código, observe pontos importantes:

- A linha 14 adiciona o atributo *type="text/babel"*, que diz ao navegador que nesta *tag* teremos **JSX**.
- A linha 13 contém o único elemento do body, um container vazio, porém com *id* **root**.
- Este mesmo elemento, **root**, é usado na linha 17 pelo código React, como segundo argumento do método render().

9. Criando variáveis em React

O React cria variáveis usando as palavras reservadas **let** e **const**. Elas são equivalentes ao uso da palavra **var**. Assim, o código **React** do Exemplo 1 pode ser escrito de forma mais separada, como:

```
const element = <h1>Hello, world2!</h1>;

      ReactDOM.render(
        element,
        document.getElementById('root')
      );
```

Código 2: uso de variável em React.

A primeira linha cria uma variável chamada **element** e atribuí a ela uma *tag* <h1> com o texto “Hello world2!”.

Então, na chamada do método **render()**, a variável **element** é passada como seu primeiro argumento. No Exemplo 1, a prória *tag* <h1> era passada como argumento. O resultado é o mesmo, mas essa segunda forma permite estruturar o código de maneira mais organizada usando variáveis.

10. Referenciando variáveis

Um recurso poderoso ao usar JSX é inserir expressões válidas em JS entre chaves, conforme o trecho abaixo:

```
      const nome = "Variável em React";
      const elemento = <h1>Olá, {nome}!</h1>;
      ReactDOM.render(
        elemento,
        document.getElementById('root')
      );
```

Código 3: uso de referência à variável dentro de chaves.

No trecho, a variável **elemento** possui um valor depende da variável **nome**, colocada entre os símbolos de abre e fecha chaves.

Qualquer expressão válida em JS pode ser colocada entre chaves, por isso, você pode criar textos interativos à vontade. Por exemplo, modifique o Código 3 para o trecho abaixo:

```
const nome = "Variável em React";
const elemento = <h1>Olá, {nome}! Seu número é {25+4}.</h1>;
ReactDOM.render(
  elemento,
  document.getElementById('root')
);
```

Código 4: uso de expressão JS dentro de chaves.

11. Exemplo 2: Olá Variável em React com JSX

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8" />
5.  <title>Exemplo 1 - Hello World em React com JSX</title>
6.  <!-- Adiciona React -->
7.  <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
8.  <script src="https://unpkg.com/react-dom@17/umd/react-
    dom.development.js"></script>
9.  <!-- Adiciona JSX -->
10. <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
11. </head>
12. <body>
13. <div id="root"></div>
14. <script type="text/babel">
15.      const nome = "Variável em React";
16.      const elemento = <h1>Olá, {nome}! Seu número é {25+4}.</h1>;
17.      ReactDOM.render(elemento, document.getElementById('root'));
18. </script>
19. </body>
20. </html>
```

Exemplo 2: código completo do exemplo “Olá Variável” em React, usando JSX.

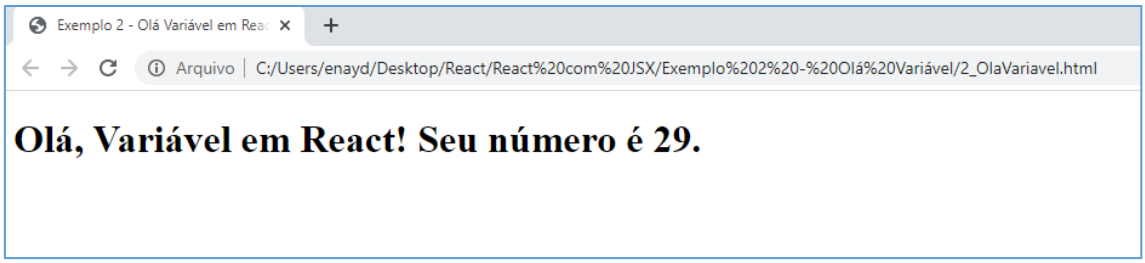

Figura 2: resultado do Exemplo 2 no navegador.

12. Exemplo 3: Hello World em React sem JSX

Antes de avançar, vamos entender a diferença de algo simples, como os exemplos até aqui, escrito em React puro. O Exemplo 1 pode ser feito com o seguinte código:

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset="UTF-8" />
5. <title>Exemplo 1 - Hello World em React com JSX</title>
6. <!-- Adiciona React -->
7. <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
8. <script src="https://unpkg.com/react-dom@17/umd/react-
   dom.development.js"></script>
9. </head>
10. <body>
11. <div id="root"></div>
12. <script>
13.     const element = React.createElement(
14.           'h1',
15.           {className: 'suaclasse'},
16.           'Hello, world sem JSX!'
17.     );
18.     ReactDOM.render(
19.          element,
20.          document.getElementById('root')
21.     );
22. </script>
23. </body>
24. </html>
```

Exemplo 3: código completo do exemplo *Hello World* em React, sem usar JSX.

Reparem que não existe mais a linha que antes importava o JSX. No entanto, foi preciso usar o método **React.createElement()** que possui uma sintaxe maior quando comparado com **JSX** para obter o mesmo resultado.

13. Elementos

Até aqui todos os códigos de React utilizam a declaração de um elemento, representado pela variável **element** ou **elemento** nos exemplos, que é chamado como primeiro argumento no método **ReactDOM.render()**.

```
ReactDOM.render(<primeiro argumento>, <segundo argumento>);
```

O segundo argumento é o nó raiz do React, normalmente um elemento com um atributo *id*. No Exemplo 3, este elemento está na linha 11.

Logo, a chamada do método **render()** aciona uma atualização do elemento (primeiro argumento) dentro do nó raiz (segundo argumento).

Elementos são os menores blocos de construção possíveis em **React**.

Neste ponto é preciso entender o conceito de DOM, que significa, *Document Object Model* ou, em português, modelo de objeto de documento. Um arquivo HTML é um DOM que possui tags bem definidas. Para o arquivo ser HTML ele precisa seguir uma estrutura específica, ou, em outras palavras, preicsa seguir o modelo de documentos HTML.

O navegador lê cada um dos elementos (criado com as *tags*) HTML e as renderiza (exibe, mostra na tela). Quando o React está em ação, ele é o responsável por atualizar somente os elementos React dentro do DOM. Repare que nesta última afirmação, posso trocar a palavra DOM por

documento HTML. Esta é uma importante parte para entender, pois contém um dos propósitos do React. O método **render()** será chamado apenas para atualizar um elemento dentro do nó raiz. O restante da página HTML continuará funcionando sem intereferência do React. Ele só atualiza o necessário, as partes do documento HTML que realmente mudam.

14. Componentes (via funções JavaScript)

Em um documento HTML temos suas *tags* bem definidas, as quais podemos chamar de elementos HTML (head, body, div, tablet etc). Adicionalmente, podemos ter elementos React, como os que temos visto. **Componentes** são compostos de **elementos**, representando um conceito mais amplo.

Vamos entender esse conceito com um exemplo lúdico. Imagine um game online em modo texto, em que todas as interações ocorrem no navegador.

Ao logar no game, o jogador recebe uma mensagem de bem-vindo. A mensagem vem estruturada em duas partes: um título, com o nome e level do jogador, e, abaixo, como um sub-título, a classe de guerreiro que ele representa e o número com o total de experiência (XP) que ele possui.

> Seja bem-vindo Leapmon - level 14
>
> Dark Elf, 847 de XP

```
Seja bem-vindo <nome do personagem> - level <XX>
<classe>, <XP> de XP
```

Para criar esse trecho em HTML usaríamos, por exemplo, as tags <h1> e <h2>. Porém, vamos usar um componente chamado **Status** para exibir as principais informações de um personagem do game.

Pense que o componente **Status** resultará em uma *tag* <Status>, a qual deverá ser definida, via componente, para então ser usada no código React.

Criar um componente React equivale a a criar uma função em JavaScript. Eles aceitam entradas arbitrárias (chamadas *props*, de propriedades) e retornam **elementos React** que descrevem o que deve aparecer na tela.

Assim, os componentes dividem a interface em partes independentes e reutilizáveis, de modo a pensar e programar cada uma isoladamente.

15. Exemplo 4 – Componente em React com JSX

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <meta charset="UTF-8" />
5.  <title>Exemplo 4 - Componente em React com JSX</title>
6.  <!-- Adiciona React -->
7.  <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
8.  <script src="https://unpkg.com/react-dom@17/umd/react-
    dom.development.js"></script>
9.  <!-- Adiciona JSX -->
10. <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
11. </head>
12. <body>
13. <div id="root"></div>
14. <script type="text/babel">
15.      function Status(props) {
16.      return <div>
17.              <h1>Seja bem-vindo, {props.nome} - level {props.level}</h1>
18.              <h2>{props.classe}, {props.xp} de XP</h2>
19.             </div>;
20. }
21. const elemento = <Status nome="Phaet" level="14" classe="Paladino" xp="847" />;
22.      ReactDOM.render(
23.          elemento,
24.          document.getElementById('root')
25.      );
26. </script>
27. </body>
28. </html>
```

Exemplo 4: código completo com a criação de um componente em React.

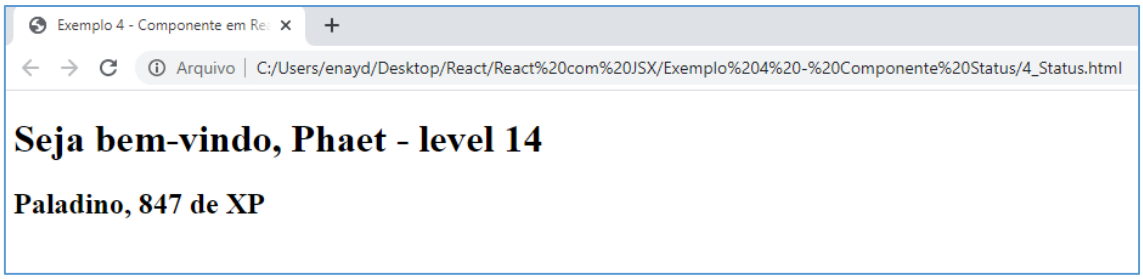

Figura 3: resultado do Exemplo 4 no navegador.

Vamos analisar o código e o que acontece durante e após sua execução:

1. A linha 22 chama a função **render()** com o elemento <Status /> como primeiro argumento, conforme a linha 23.
2. Em seguida, **React** chama o componente (a função) **Status** com as propriedades que escrevemos na linha 21 (*nome*, *level*, *classe* e *xp*).
3. O componente (função) retorna um elemento React, já formatado da forma idealizada.
4. React DOM atualiza o DOM somente com a parte dinâmica.

Dessa forma, elementos React podem representar componentes definidos pelo desenvolvedor, como foi o caso do componente **Status** neste exemplo.

***Dica**: Sempre inicie os nomes dos componentes com uma letra maiúscula. O React trata componentes começando com letras minúsculas como tags do DOM. Por exemplo, representa uma tag div do HTML, mas <Status/> representa um componente e requer que ele esteja previamente definido no escopo.*

16. Componentes (via funções classes)

Para criar esse trecho em HTML usaríamos, por exemplo, as tags <h1> e <h2>. Porém, vamos usar um componente chamado **Status** para exibir as principais informações de um personagem do game.

Por que JSX?

O React adota o fato de que a lógica de renderização é inerentemente acoplada com outras lógicas de UI: como eventos são manipulados, como o state muda com o tempo e como os dados são preparados para exibição.

Ao invés de separar tecnologias artificialmente colocando markup e lógica em arquivos separados, o React separa conceitos com unidades pouco acopladas chamadas "componentes" que contém ambos.

Características do React

React permite a criação de *views* simples para cada estado na sua aplicação, atualizando e renderizando de forma eficiente apenas os componentes necessários na medida em que os dados mudam. *Views* declarativas fazem com que seu código seja mais previsível e simples de depurar.

React é baseado em componentes, que encapsulam e gerenciam seu próprio estado. Ao combiná-los é possível criar UIs complexas.

Como a lógica do componente é escrita em JavaScript e não em templates, você pode facilmente passar diversos tipos de dados ao longo da sua aplicação e ainda manter o estado fora do DOM.

Lista de Exemplos

Todos os exemplos podem ser encontrados na pasta CursoReact do GitHub :

Exemplo 1: Hello World com JSX

Exemplo 2: Olá Variável em React

Exemplo 3: Hello World sem JSX

Exemplo 4: Componente Status

Referências

[1] Documentação Oficial da biblioteca React. Disponível em: < https://pt-br.reactjs.org/docs/getting-started.html>. Acesso em: 21 fev. 2021.

[2] 8 Best React IDE & React Editors. Disponível em: <https://www.dunebook.com/best-react-ide-react-editors/>. Acesso em: 21 fev. 2021.