



# FRAMEWORKS FULL STACK

## Texto base

# 4

## Introdução ao ReactJS e listagem de produtos no front-end com dados de uma API back-end

Prof. André Nascimento Maia

Prof. Caio Nascimento Maia

### *Resumo*

*Nesta aula, aprofundaremos os conceitos que envolvem o principal framework utilizado para criação de web application, o React. Será criado, para demonstração, um componente para listar produtos de um e-commerce que, inicialmente, utiliza informações em memória e depois consome uma API REST através da API Fetch presente nos navegadores de versões mais recentes.*

### **4.1. React**

React é um *framework* Javascript para criar interfaces de usuários utilizando uma abordagem moderna e eficiente baseada em Single Page Application (React).

Utilizando React a criação de interfaces interativas de usuários acontece através de *views* declarativas para cada estado da aplicação, o que permite o *framework* atualizar de forma eficiente apenas os componentes que mudaram de estado e necessitam ser visualizados quando atualizados.

O *framework* é utilizado em diversos escopos diferentes. É possível utilizar React em aplicação de *front-end* que são executadas dentro de um navegador, em um *back-end* utilizando Node.js e em aplicações para serem executadas dentro de um dispositivo móvel através do *framework* React Native, que utiliza os mesmos conceitos do React, garantindo que não seja necessário aprender diversos *frameworks* diferentes para cada uma dessas tarefas.

React é baseado em componentes, o que permite a criação de interfaces de usuários complexas através da combinação de componentes e gerenciamento de estados.

### 4.1.1 Componente em React

As *tags* padrão do HTML têm diversos componentes autocontidos para resolver diversos problemas. São os casos das *tags*: *img*, *table*, *span*, *div*, e todas as outras disponíveis para uso. Normalmente, utilizamos essas *tags* de forma combinada para criar a visualização de aplicações *web* através de navegadores, sem nos darmos conta de como elas funcionam internamente.

```
<figure>
  
  <figcaption>O título para explicar a imagem</figcaption>
</figure>
```

#### **Código 4.1: Exemplo de HTML 5 para exibir uma imagem no navegador.**

O Código 4.1 é o exemplo de um código HTML composto pelas *tags* *figure*, *img* e *figcaption* (MDN, 2022), que são utilizadas para exibir uma imagem em um navegador informando uma URL para uma imagem específica da *web*, um título e uma descrição alternativa para garantir acessibilidade para quem não consegue visualizar uma imagem através de um navegador renderizando a informação em um monitor de computador.

Internamente, o navegador sabe exatamente como baixar a imagem, contornando qualquer problema de conexão remota; renderizar essa imagem na tela; e posicioná-la do modo adequado baseando-se em todo estilo já aplicado e calculado para a imagem.

O React permite a criação de novos componentes autocontidos e autossuficientes como as *tags* já existentes no HTML. Essa habilidade permite que o desenvolvedor crie suas próprias abstrações utilizando componentes padrão HTML, outros componentes React já existentes, ou até mesclar ambos.

Um componente React é nada mais, nada menos que uma função Javascript. O Código 4.2 cria um componente chamado Greeting que tem o objetivo de criar uma mensagem personalizada de boas vindas baseado no nome de uma pessoa.

```
1. <Greeting person="Alice" />
2.
3. function Greeting(props) {
4.   const person = props.person;
5.   const msg = `Hello ${person}!`;
6.   return <span>{msg}</span>
7. }
```

#### **Código 4.2: Trecho de código do componente React chamado Greeting.**

O componente Greeting, toda vez que utilizado, vai imprimir uma mensagem padrão de saudação.

Assim como as *tags* do HTML, você pode criar componentes dentro de componentes combinando-os e utilizando relação de parentesco para formar o melhor layout que preferir.

No Código 4.3, combinamos o componente Greeting do Código 4.2 para criar um novo componente chamado Letter, que representa uma estrutura padrão para uma carta de boas vindas.

```

1. <Letter>
2.   <Greeting person="Alice" />
3. </Letter>
4.
5. function Letter(props) {
6.   const greeting = props.children;
7.   return (
8.     <div>
9.       {greeting}<br/>
10.      <p>Seja bem-vindo.</p>
11.    </div>
12.  );
13. }
```

**Código 4.3: Trecho de código do componente React chamado Letter que tem relação de parentesco com o componente Greeting.**

Os componentes React podem ter internamente a parte visual, utilizando *tags* HTML, funcionalidades específicas através de sua função em Javascript e também gerenciar um estado.

O Código 4.4, mostra como podemos criar um componente que gerencia um estado localmente. No código de exemplo, o componente gerencia a variável chamada *count*, declarada na linha 5, junto com a função *setCount* que é utilizada para o valor do estado gerenciado. A declaração da linha 5 utiliza atribuição via desconstrução (MDN, 2014) para extrair a referência para o estado na primeira posição do vetor retornado pela função *useState(...)*, e o mesmo conceito para atribuição da referência para a função de atualização do estado.

A função *useState* é chamada de *Hook*. *Hooks* são uma nova adição ao React 16.8. Eles permitem que você use o *state* e outros recursos do React sem escrever uma classe (React, 2022). O primeiro argumento da função *useState* é o valor inicial para o estado que será gerenciado.

Sempre que um *state* em React é atualizado, automaticamente, o *framework* inicia uma análise dos impactos da alteração *Document Object Model* (DOM) (MDN, 2022) e atualiza apenas o trecho necessário para renderizar a alteração do estado gerenciado. Esse mecanismo, é um dos grande diferenciais do React e o que faz ser eficiente em seu gerenciamento dos componentes da página.



```

1. import React, { useState } from 'react';
2.
3. function Counter() {
4.   // essa é uma variável de state usando o hook useState.
5.   const [count, setCount] = useState(0);
6.
7.   function increment() {
8.     setCount(count + 1);
9.   }
10.
11.  return (
12.    <div>
13.      <p>O valor do contador é: {count}</p>
14.      <button onClick={increment}>Incrementar</button>
15.    </div>
16.  );
17.}

```

**Código 4.4: Exemplo de gerenciamento de estados em um componente Counter em React**

## 4.2. Criando uma página de listagem de produtos

Utilizando os conceitos apresentados até aqui, vamos construir uma listagem de produtos baseada no protótipo da Figura 4.1. Imagine que tenhamos que construir um *e-commerce* qualquer e esse *e-commerce* necessita listar os seus produtos disponíveis para compra. Cada produto disponível deve conter uma foto, um título, um valor total, uma quantidade total para pagamento parcelado e o valor das parcelas, e também, se o parcelamento inclui juros.

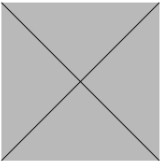
Geralmente, a lista de produtos em *e-commerces* é vasta e necessitam ser mantidas e sincronizadas com um sistema de estoque. Porém, para criarmos um componente de listagem de produtos, vamos supor uma lista de produtos fixas e em memória para que seja criado o código necessário para o componente em React.

Uma das formas de criar um componente React é criar todo o código HTML necessário e então, decompor esse código HTML em componentes React menores, até que a combinação desses vários componentes formem um componente maior que exibe as informações, garante funções básicas e gerencia estados da forma adequada.

O Código 4.5 apresenta trecho de código para a criação de um componente chamado *ProductsForSaleList*. O objetivo desse componente é apresentar uma lista de produtos disponíveis para venda de uma determinada loja on-line, um *e-commerce*, de acordo com os requisitos mínimos relacionados e baseando-se em uma experiência de visualização muito próxima do protótipo apresentado na Figura 4.1.

**Figura 4.1: Protótipo de página de listagem de produtos.**

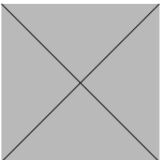
Buscar



Caneca Personalizada de Porcelana

**R\$ 123,45**

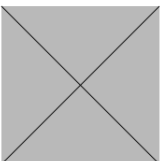
em 3x R\$ 41,15 sem juros



Caneca de Tulipa

**R\$ 123,45**

em 3x R\$ 41,15 sem juros



Caneca Importada Personalizada em Diversas Cores

**R\$ 123,45**

em 3x R\$ 41,15 sem juros



Caneca Personalizada de Porcelana

**R\$ 123,45**

em 3x R\$ 41,15 sem juros

**Fonte: dos Autores, 2022.**

```
1. function Installment(props) {
2.   const fees = props.installment.hasFee
3.     ? "com juros"
4.     : "sem juros";
5.   return (
6.     <p>
7.       em {props.installment.number}x
8.       de R$ {props.installment.total}
9.       {fees}
10.    </p>
11.  );
12. }
13.
14. function ProductListItem(props) {
15.   const defaultProductImage = "https://via.placeholder.com/150";
16.   return (
17.     <div className="d-flex position-relative border my-2">
18.       <img
19.         src={defaultProductImage}
20.         className="flex-shrink-0 me-3" />
21.       <div>
22.         <a href="#" className="stretched-link">
23.           <h3 className="mt-0">{props.product.title}</h3>
24.         </a>
25.         <h4>R$ {props.product.amount}</h4>
26.         <Installment installment={props.product.installments} />
27.       </div>
28.     </div>
29.   );
30. }
31.
32. function ProductsForSaleList(props) {
33.   const products = props.products.map(
34.     (x, index) => <ProductListItem product={x} key={index} />
35.   );
36.   return <div>{products}</div>
37. }
```

#### Código 4.5: Relação de componentes para formar o ProductsForSaleList.

```

1. const products = [
2.   {
3.     title: "Caneca Personalizada de Porcelana",
4.     amount: 123.45,
5.     installments: { number: 3, total: 41.15, hasFee: true }
6.   },
7.   {
8.     title: "Caneca de Tulipa",
9.     amount: 123.45,
10.    installments: { number: 3, total: 41.15 },
11.  },
12. ];

```

#### Código 4.6: Lista fixa de informações de produtos.

O componente `ProductsForSaleList` é composto de três outros componentes chamados `ProductListItem` e `Installment`. Cada um dos componentes é responsável por um trecho da informação exibida na lista de produtos, o que facilita a leitura do código, manutenção e reutilização. O `ProductsForSaleList` utiliza uma lista de produtos fixa em memória (Código 4.6) para exibição, porém, é muito comum que essa lista de produtos seja obtida de uma API específica do *e-commerce*, que gerencia adequadamente o estoque desses produtos que serão colocados à venda.

### 4.3. Obtendo dados de uma API REST

Frequentemente *web applications* têm que utilizar chamadas remotas de APIs para executar uma operação de negócio, criar um pagamento, enviar um e-mail ou diversas outras milhares de possibilidades que atualmente estão disponíveis através de web APIs em um formato SaaS.

Uma das ideias por trás do conceito de componentes do React, é que os componentes sejam autossuficientes. Então, eles devem ser capazes de criar uma visão adequada de suas informações para o navegador, gerenciar o seu estado aplicando modificações na visão quando necessário e também consumir informações remotas quando pertinente. Assim como a *tag img* do HTML que recebe apenas o atributo *src* com uma URL e isso é o suficiente para que esse componente nativo saiba baixar a imagem remota e exibi-la na tela.

Existem diversas formas de fazer uma chamada remota a uma API REST. As formas mais comuns são através da biblioteca *Axios* (*Getting Started*, 2022) ou através da API dos navegadores chamada *Fetch* (*Fetch API - APIs Da Web* | MDN, 2022).

Utilizando o componente `ProductsForSaleList` criado no Código 4.5, vamos propor um novo código que faz a chamada a uma API REST fictícia que deveria



retornar o mesmo resultado do Código 4.6 com uma lista de produtos disponíveis para venda.

No Código 4.5 linha 33, é esperada uma lista de produtos informada através de propriedades (*props*). Isso significa que o modo de uso mais comum do componente para uma lista de produtos fixa é conforme a seguir:

```
<div>  
    <ProductsForSaleList products={products} />  
</div>
```

**Código 4.7: Exemplo de utilização do componente ProductsForSaleList.**

O Código 4.7 considera que a variável *products* já foi declarada no escopo como foi feito em Código 4.6.

Porém, se quisermos que o componente que foi renderizado faça uma requisição a uma API REST para obter a lista de produtos, devemos considerar o ciclo de vida do componente. O método específico para atualizar o estado de um componente quando os dados forem transferidos da API REST para o navegador é o *componentDidMount* (*Estado E Ciclo De Vida – React, 2022*). Esse método lhe permitirá utilizar o *setState* caso a sua implementação do componente for uma classe Javascript que estende um *React.Component*. Para o nosso exemplo, estamos utilizando a implementação de componentes como uma função, então, podemos utilizar outro *Hook* para nos auxiliar. O *Hook* que utilizaremos se chama *useEffect*. Este *Hook* tem esse nome porque permite que você cause efeitos colaterais na página que está sendo exibida, como quando atualizamos um *state* declarado em um componente (*Usando Effect Hook (Hook De Efeito) – React, 2022*).

Veja como ficaria o *ProductsForSaleList* adaptado para, assim que for construído, fazer uma busca em uma API REST fictícia que retorna uma lista de produtos exatamente no mesmo formato que definimos em Código 4.6.

```
01. function ProductsForSaleList() {
02.   const [error, setError] = useState(null);
03.   const [isLoading, setIsLoaded] = useState(false);
04.   const [products, setProducts] = useState([]);
05.
06.   useEffect(() => {
07.     fetch("https://api-ficticia.com/products")
08.       .then((response) => response.json())
09.       .then(
10.         (json) => {
11.           setIsLoaded(true);
12.           setProducts(json);
13.         },
14.         (error) => {
15.           setIsLoaded(true);
16.           setError(error);
17.         }
18.       );
19.   }, []);
20.
21.   if (error) {
22.     return <div>Error: {error.message}</div>;
23.   } else if (!isLoading) {
24.     return <div>Carregando...</div>;
25.   } else {
26.     const p = products.map(
27.       (x, index) => <ProductListItem
28.         product={x} key={index} />;
29.     return <div>{p}</div>;
30.   }
31. }
```

**Código 4.8: ProductsForSaleList utilizando a API Fetch para obter dados de uma API REST.**

No Código 4.8 fizemos uma mudança significativa em ProductsForSaleList. Primeiro, incluímos 3 diferentes estados chamados *error*, *isLoading* e *products*. O primeiro é utilizado para controlar o estado das chamadas remotas. Se qualquer erro acontecer quando fizermos uma requisição para uma API REST, uma mensagem amigável será exibida para o usuário. Veja as linhas 14 e 21.

O estado *isLoading* é utilizado para sabermos quando devemos exibir uma mensagem de "carregando" para o usuário. Essa mensagem sempre ficará visível quando o estado de *isLoading* for *false*.

Por fim, o *state products* controla a lista de produtos que deve ser exibida. Inicialmente esta lista está vazia (veja linha 04), mas após a chamada com sucesso da API REST em *api-ficticia.com/products*, os produtos retornados na resposta da API são convertidos em JSON e atualizamos o *state* de *products*. Isso faz com que automaticamente, a árvore de componente do React identifique que um *state* foi atualizado e é necessário causar um efeito colateral no DOM da página que está sendo exibida no navegador.

Após as alterações de código propostas no Código 4.8, não precisamos mais informar uma propriedade chamada *products* para a utilização do componente *ProductsForSaleList*. Agora, a sua utilização deve ser conforme o Código 4.9.

```
<div>  
  <ProductsForSaleList />  
</div>
```

**Código 4.9: Utilização do *ProductsForSaleList* com uma lista de produtos disponível através de uma API REST.**

#### 4.4. Considerações Finais

Nesta aula, vimos o que são componentes React e os seus conceitos básicos. Também criamos um componente complexo chamado *ProductsForSaleList*, responsável por exibir uma lista de produtos à venda para um *e-commerce* baseado em uma lista de produtos disponíveis em estoque obtida através de uma API REST.

Os conceitos utilizados, permitem a junção entre o código produzido para *front-end* e o código de *back-end* que é requisito mínimo para qualquer solução *full-stack*, de ponta a ponta, para a construção de produtos que se baseiam em uma solução *web*.

## Referências

**React:** Uma biblioteca JavaScript para criar interfaces de usuário. Disponível em: <https://pt-br.reactjs.org/>. Acesso em: 10 dez. 2021.

**Atribuição via desestruturação (destructuring assignment) - JavaScript | MDN.** (2014, April 14). MDN Web Docs. Acesso em: 16 jan. 2022. Disponível em: [https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Destructuring\\_assignment](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)

**Estado e Ciclo de Vida – React.** (2022). React. Disponível em: <https://pt-br.reactjs.org/docs/state-and-lifecycle.html>. Acesso em: 17 jan. 2022.

**Fetch API - APIs da Web | MDN.** (2022, January 15). MDN Web Docs. Disponível em: [https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch_API). Acesso em: 17 jan. 2022.

**Getting Started.** (2022). Axios. Disponível em: <https://axios-http.com/docs/intro>. Acesso em: 17 jan. 2022.

**<img> - HTML: Linguagem de Marcação de Hipertexto | MDN.** (2022, January 13). MDN Web Docs. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/img>. Acesso em: 16 jan. 2022.

**Introdução ao DOM - APIs da Web | MDN.** (2022). MDN Web Docs. Disponível em: [https://developer.mozilla.org/pt-BR/docs/Web/API/Document\\_Object\\_Model/Introduction](https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model/Introduction). Acesso em: 16 jan. 2022.

**Usando Effect Hook (Hook de Efeito) – React.** (2022). React. Disponível em: <https://pt-br.reactjs.org/docs/hooks-effect.html>. Acesso em: 17 jan. 2022.

**Usando o State do Hook – React.** (2022). React. Disponível em: <https://pt-br.reactjs.org/docs/hooks-state.html>. Acesso em: 16 jan. 2022.