



Componentes Funcionais

[Revisando o conceito de componente](#)

[Tipos de componentes](#)

[Estrutura de um componente](#)

[Componentes aninhados](#)

-
- Componentes são um dos principais conceitos do React;
 - Podemos dizer que um componente é a estrutura principal quando vamos desenvolver uma interface do usuário (UI) utilizando React.

Revisando o conceito de componente

- **Componente** é um bloco de conteúdo da interface do usuário (ou tela) que pode ser reutilizado e, preferencialmente, deve ter uma única responsabilidade dentro da nossa aplicação;
- Podemos pensar nos componentes como uma peça de Lego que podemos usar várias vezes, para construirmos partes específicas de uma página web;
- Ele é um bloco de código que contém:
 - HTML (através do JSX);
 - JavaScript;
 - CSS (opcionalmente).
- Ao dividirmos nossa página em pequenos componentes, nós temos um código mais organizado, mais fácil de entender e de manter;
- Sem contar que evitamos a repetição de código da nossa aplicação, já que podemos reutilizar esses componentes quantas vezes quisermos;
 - À medida que seu projeto cresce, você notará que muitos de seus designs podem ser compostos reutilizando componentes que você já escreveu, agilizando seu desenvolvimento.
- Além disso, hoje existem diversas bibliotecas disponíveis no mercado com componentes já prontos para que possamos instalar como dependência nos nossos projetos e utilizar em nossos projetos de forma livre e gratuita.

Tipos de componentes

- O React dá suporte a componentes feitos com a estrutura de classes e métodos (que chamamos de *Class Components*, ou Componentes de Classe;
 - Pode ver com mais detalhes no Curso de “[Diferentes Tipos de Componentes](#)”.
- Nas versões recentes, temos o que chamamos de componentes funcionais, que são componentes que são funções do JavaScript que podemos exportar e usar em outros pontos do nosso código;
- Iremos utilizar os componentes funcionais ao longo dos cursos, que são os mais recomendados atualmente e traz uma abordagem mais moderna do React.

Estrutura de um componente

Se pudermos dividir a base do componente em uma “receita de bolo”, ela seria mais ou menos assim:

Passo 1: Criando a função e o conteúdo do componente

- Criar uma função com o nome do componente;
 - O nome deve começar com **letra maiúscula**, para que o React saiba diferenciar o que é um elemento HTML de um componente React quando formos utilizar nosso componente;
 - Exemplo: `Card`, `Profile`, `ListItem`, `Gallery`, etc. Podemos utilizar nomes em português, mas, normalmente, nos projetos de mercado, é mais comum vermos os nomes dos componentes em inglês (por isso, nos exemplos, já iremos nos habituar com isso).
- Essa função do nosso componente deve ter um **retorno** com o conteúdo que deve ser **renderizado*** na nossa página.
 - Por exemplo, pode ser a combinação de uma imagem, títulos, parágrafos e outros;



* Um termo muito comum quando trabalhamos com front-end é o **renderizar**, que basicamente significa transformar o código em uma representação visual que os usuários podem ver. Dentro da programação, *renderizar* se refere ao processo de criar e exibir elementos na tela com base nas instruções do nosso código: textos, imagens ou qualquer outro elemento visual na UI.

```
function Profile() {  
  return (  
      
  )  
}
```

```
)  
}
```

- **Importante:** o conteúdo retornado deve ser um único elemento encapsulado:

- ❌ Errado

```
function Profile() {  
  return (  
    <h1>Katherine Johnson</h1>  
      
  )  
}
```

- ✅ Certo

```
function Profile() {  
  return (  
    <div>  
      <h1>Katherine Johnson</h1>  
        
    </div>  
  )  
}
```

Passo 2: Exportando e importando o componente

- Depois de criarmos o nosso componente funcional em um arquivo separado, precisamos **exportá-lo** para que ele fique disponível para ser **importado** em outros pontos do nosso código, caso contrário, não iremos conseguir o utilizar para que ele seja exibido na tela;
- Existem duas formas de exportar os componentes:



Essas duas formas de exportarmos não é uma particularidade do React, mas sim do JavaScript, na qual podemos exportar funções independente se estamos trabalhando com front-end, back-end, tampouco com aplicações feitas com ou sem React.

▼ Exportação padrão / Default export:

- Quando utilizamos a exportação padrão `export default`, nós indicamos que, ao importar o arquivo `profile.js`, estaremos utilizando *por padrão* a função `Profile`, que é o componente que renderiza uma imagem no exemplo acima:

```
export default function Profile() {  
  return (  
      
  )  
}
```

- Agora, ao **importarmos** esse componente em outra página na qual queremos utilizar, podemos usar esse componente da seguinte maneira:

```
import React from 'react';  
import Profile from './Profile';  
  
function App() {  
  return (  
    <div>  
      <h1>Minha Aplicação React</h1>  
      <Profile />  
    </div>  
  );  
}  
  
export default App;
```

- Nesse caso, como não passamos nenhum conteúdo dentro desse componente, nós abrimos a tag e a fechamos logo em seguida (padrão React): `<Profile />`.



Quando importamos uma função que utiliza `export default` (ou exportação padrão), nós podemos dar nomes diferentes aos componentes em diferentes partes do nosso código quando o importamos. Mas, tome cuidado, pois isso pode comprometer a organização e manutenibilidade do seu código.

▼ Exportação nomeada / Named export:

- A exportação nomeada permite que você exporte várias coisas diferentes de um mesmo arquivo, como, por exemplo, dois componentes diferentes;
- Ao invés de escolhermos uma função para ser a principal, como foi no caso da exportação padrão, nós conseguimos exportar vários componentes/funções com um nome específico;

```
export function Profile() {
  return (
    
  )
}
```

- A vantagem é que, momento que vamos importar esse(s) componente(s) em outros arquivos, nós não podemos alterar seu nome como podemos fazer na exportação padrão.

```
import React from 'react';
import { Profile } from './Botao';

function App() {
  return (
    <div>
      <h1>Minha Aplicação React</h1>
      <Profile />
    </div>
  );
}

export default App;
```

- Quando trabalhamos com uma exportação nomeada, quando vamos importar esse componente, temos que utilizar chaves em volta do nome:

```
import { FunctionOne, FunctionTwo, FunctionThree } from 'path/file';
```

▼ Resumo

Sintaxe	Exportação	Importação
Padrão / Default	<code>export default function Button() {}</code>	<code>import Button from './Button.js';</code>
Nomeada / Named	<code>export function Button() {}</code>	<code>import { Button } from './Button.js';</code>

Passo 3: Finalize a estrutura e estilização do componente

- Agora que já temos a função do componente criada, renderizando um conteúdo inicial e já podemos vê-lo em tela, podemos finalizar o conteúdo dele, assim, fica mais fácil de termos um feedback visual do que está acontecendo conforme as nossas alterações.

Passo 4: Adicione interatividade ao componente

- O quarto passo é adicionarmos interatividade ao nosso componente, mas, veremos como podemos fazer isso de diferentes maneiras em um curso específico.

Componentes aninhados

- Assim como acontece com as tags HTML, nós podemos ter um componente dentro de outro, o que chamamos de componentes aninhados:


```
function Profile() {  
  return (  
      
  );  
}  
  
export default function Gallery() {  
  return (  
    <section>  
      <h1>Amazing scientists</h1>  
      <Profile />  
      <Profile />  
      <Profile />  
    </section>  
  );  
}
```

Referências

Your First Component – React

The library for web and native user interfaces

 <https://react.dev/learn/your-first-component>

 React

Learn React

REACT.DEV/LEARN

Importing and Exporting Components – React

The library for web and native user interfaces

 <https://react.dev/learn/importing-and-exporting-components>



Learn React

REACT.DEV/LEARN