

IOS – Instituto de
Oportunidade Social

React 03 - Componentes funcionais



- Compreender o uso de Componentes JSX;
- Criar o primeiro componente JSX;

IOS – Instituto de
Oportunidade Social

Criando componentes funcionais



Hooks

React Hooks são funções que nos permitem usar estado (**state**) e recursos do ciclo de vida de um componente quando utilizamos componentes funcionais. Os **hooks não funcionam dentro de classes**. Os **principais** hooks são:

- **useState**: usado para **criar o estado** de um componente, atribuir um **valor inicial** para o **estado** e, também, uma função para que possamos **atualizar** esse **estado**.
- **useEffect**: usado para produzir **efeitos colaterais** em componentes funcionais.

Demais Hooks:

Além desse dois existem o **useContext**, **useReducer**, **useRef**, **useLocation**, entre outros.

Para saber mais: Você pode acessar a página de Introdução aos Hooks da documentação oficial disponível em:

<https://pt-br.reactjs.org/docs/hooks-intro.html>

Criando componente JSX:

- Criar um diretório chamado **Components**
- O **nome do componente** deve obrigatoriamente **começar com letra maiúscula e não pode conter espaços em branco ou caracteres especiais**.
- A extensão do arquivo pode ser **.js** (de arquivo JavaScript) ou **.jsx (de arquivo JSX)**. A vantagem de usar a extensão **.jsx** é que o VS Code já interpreta como um arquivo com a **sintaxe JSX**. Ela é **usada no React** e faz as sugestões de correções para essa linguagem.

Criando componentes funcionais

Retornando o elemento pai React:

Isso **não** é permitido:

```
return (  <p>Você clicou {contador} vezes</p>
          <button onClick={() => setContador(contador + 1)}>
            Clique aqui
          </button>    );
```



Importante! Note, também, que você só pode ter um elemento React pai dentro do método **return**. Desse modo precisamos colocar o elemento **<div>** para envolver os outros dois elementos, **parágrafo** e **botão**. Dessa forma, a aplicação funcionará corretamente.

Isso **está** correto:

```
return (    <div>
              <p>Você clicou {contador} vezes</p>
              <button onClick={() => setContador(contador + 1)}>
                Clique aqui
              </button>
            </div>    );
```

Criando componentes funcionais



Extensão para React:



Importante! O VS Code possui uma extensão para agilizar na criação do código de um componente React: **ES7 React/Redux/GraphQL/React-Native snippets**

Comando **rafce** - Esse **snippet** gerará um código automático de um **componente funcional** (utiliza **Arrow function**) e com o mesmo nome dado ao arquivo do componente.



ES7 React/Redux/GraphQL/React-Native snippets v3.1.1

dsznajder | 3.829.441 | ★★★★★ (40)

Simple extensions for React, Redux and Graphql in JS/TS with ES7 syntax

Disable Uninstall

This extension is enabled globally.

IOS – Instituto de
Oportunidade Social

Inserindo componente na raiz



Inserindo componente na raiz

Até agora, nós apenas criamos o arquivo e inserimos o código do componente. Ainda precisamos **inserir esse componente** na **árvore de componentes** da aplicação. Como todo projeto React, temos um **componente raiz**, esse nosso componente criado será um componente filho do componente raiz App.

```
1. import './App.css';
2. import Exemplo from './Components/Exemplo';
3.
4. function App() {
5.   return (
6.     <div className="App">
7.       <Exemplo />
8.     </div>
9.   );
10. }
11.
12. export default App;
```

Inserindo componente na raiz

Notação para importar componente

Então, no arquivo App.js, devemos atualizar o código da seguinte forma:



IOS – Instituto de
Oportunidade Social

Informando props no Componente



Informando props no Componente

Exemplo de componente com props:

```
1. const Header = (props) => {  
2.   return (  
3.     <div>  
4.       <h1>Olá, {props.nome}, seja bem-vindo!</h1>  
5.     </div>  
6.   );  
7. };  
8.  
9. export default Header;
```

Observe que esse componente espera **pelo menos uma propriedade** (linha 1 do código), que será passada pelo componente pai. Um componente pode receber uma ou mais propriedades simplesmente indicando com a instrução **props** entre parênteses.

```
const Header = (props) => {
```

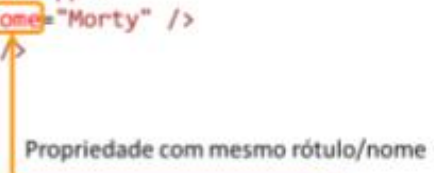
Indica que o componente pode receber dados através de propriedades

Informando props no Componente

Exemplo de passagem de props pelo componente pai:

```
1. import './App.css';
2. import Exemplo from './Components/Exemplo';
3. import Header from './Components/Header';
4.
5. function App() {
6.   return (
7.     <div className="App">
8.       <Header nome="Morty" />
9.       <Exemplo />
10.     </div>
11.   );
12. }
13.
14. export default App;
```

Note que o nome da propriedade precisa ser igual:



```
function App() {
  return (
    <div className="App">
      <Header nome="Morty" />
      <Exemplo />
    </div>
  );
}

const Header = (props) => {
  return (
    <div>
      <h1 style={estilo}>Olá, {props.nome}, seja bem-vindo!</h1>
    </div>
  );
};
```

Propriedade com mesmo rótulo/nome

Informando props no Componente



Valor padrão para uma propriedade (defaultProps)

Para evitar esse problema de não passar o valor desejado e não aparecer nenhuma informação, você pode configurar um **valor padrão** para ser exibido.

```
1. const Header = (props) => {  
2.   return (  
3.     <div>  
4.       <h1>Olá, {props.nome}, seja bem-vindo!</h1>  
5.     </div>  
6.   );  
7. };  
8.  
9. Header.defaultProps = {  
10.   nome: 'Nome padrão',  
11. };  
12.  
13. export default Header;
```

Checagem de tipos de uma propriedade

Outra informação importante ao trabalhar com propriedades é que você pode definir o tipo de dado que deverá ser passado na propriedade. Isso é feito pelo recurso **PropTypes** do React. Você pode utilizar os seguintes validadores de tipo de dados:

| Validador | Descrição |
|-------------------|---------------------------|
| PropTypes.array, | Verifica se é um array |
| PropTypes.bool, | Verifica se é um booleano |
| PropTypes.func, | Verifica se é uma função |
| PropTypes.number, | Verifica se é um número |
| PropTypes.object, | Verifica se é um objeto |
| PropTypes.string, | Verifica se é uma string |
| PropTypes.symbol, | Verifica se é um símbolo |

Informando props no Componente



Utilizando PropTypes:

```
1. import PropTypes from 'prop-types';
2.
3. const Header = (props) => {
4.   return (
5.     <div>
6.       <h1>Olá, {props.nome}, seja bem-vindo!</h1>
7.     </div>
8.   );
9. };
10.
11. Header.defaultProps = {
12.   nome: 'Nome padrão',
13. };
14.
15. Header.propTypes = {
16.   nome: PropTypes.string,
17. }
18.
19. export default Header;
```

Observe na linha 1 a necessidade de **importar o recurso** no componente e nas linhas 15 a 17 é onde definimos que o valor passado pela propriedade nome **deve ser uma string**.

Informando props no Componente



Erro na renderização

Se na renderização do componente Header dentro do componente App passarmos um valor que não seja string, por exemplo:

<Header nome=1 />

O terminal mostrará erro de compilação:

```
./src/App.js
SyntaxError: C:\react\tema_03_conta_cliques\src\App.js: JSX value should be either an expression or a quoted
JSX text. (8:25)

   6 |     return (
   7 |       <div className="App">
>  8 |         <Header nome=1 />
     |                        ^
   9 |         <Exemplo />
  10 |       </div>
  11 |     );
```

This error occurred during the build time and cannot be dismissed.

IOS – Instituto de
Oportunidade Social

Adicionando estilo ao componente



Adicionando estilo ao componente



Existem várias formas de configurar estilos em um componente **React**: **inline**, pelo arquivo **App.css** ou **index.css**, através de **Bootstrap**, etc. Vamos ver um primeiro exemplo de aplicar estilos CSS inline no componente.

No componente Header podemos utilizar o atributo **style** e definir a cor da fonte do texto que aparecer no heading `<h1>`.

```
1. import PropTypes from 'prop-types';
2.
3. const Header = (props) => {
4.   return (
5.     <div>
6.       <h1 style={{ color: 'red' }}>Olá, {props.nome}, seja bem-vindo!</h1>
7.     </div>
8.   );
9. };
10.
11. Header.defaultProps = {
12.   nome: 'Nome padrão',
13. };
14.
15. Header.propTypes = {
16.   nome: PropTypes.string,
17. };
18.
19. export default Header;
```

Adicionando estilo ao componente



Observe que, o atributo `style={{ color: 'red' }}` na linha 6 configura a **cor da fonte** como **vermelha**. Note que, quando utilizamos a propriedade CSS devemos colocá-la entre duas aberturas e fechamento de chaves.



Importante! Note que o **valor da propriedade** deve ser colocado **entre aspas**, diferentemente de como fazemos no arquivo CSS.

Adicionando estilo ao componente



Para ficar mais organizado, podemos **criar uma variável** com os **estilos** que queremos aplicar e colocar somente a variável dentro de uma abertura e fechamento de chaves do atributo **style**.

```
1. import PropTypes from 'prop-types';
2.
3. const estilo = { color: 'red' };
4.
5. const Header = (props) => {
6.   return (
7.     <div>
8.       <h1 style={estilo}>Olá, {props.nome}, seja bem-vindo!</h1>
9.     </div>
10.   );
11. };
12.
13. Header.defaultProps = {
14.   nome: 'Nome padrão',
15. };
16.
17. Header.propTypes = {
18.   nome: PropTypes.string,
19. };
20.
21. export default Header;
```

IOS – Instituto de
Oportunidade Social

Vamos Praticar



Apostila de React

05.Apostila-React

Páginas 30 a 42

OBS: Acompanhar o passo a passo com o instrutor

IOS – Instituto de
Oportunidade Social

Exercícios



Criar 2 Componentes (criar a estrutura **Components**), o primeiro **BoasVindas.jsx** contendo uma `<section>` e dentro dela um `<h1>` e um `<p>` e o componente **Sobre.jsx** contendo um `<article>` e dentro dele um `<h2>` e um `<p>` falando sobre React. No componente BoasVindas receber parâmetro **props** com **nome**, utilizando **defaultProps** e **PropTypes**, aplicar estilo com a tag **style** `{{ }}`. No componente Sobre utilizar estilo criando **variável estilo** como objeto e passar na **tag style**. Realizar o processo de Build (**npm build**) do projeto praticado anteriormente e subir no GitHub.