

IOS – Instituto de
Oportunidade Social

HTML CSS JS 33 - Exemplos de Componentes Funcionais



- > Hooks
 - > useState
 - > useEffect
- > Criando componentes funcionais
- > Inserindo um novo componente na hierarquia
 - > props
 - > Header.defaultProps
 - > PropTypes
- > Adicionando um estilo a um componente

IOS – Instituto de
Oportunidade Social

Hooks



> Hooks

- `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.

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

IOS – Instituto de
Oportunidade Social

Criando componentes funcionais



Na aula de hoje vamos criar um componente funcional que possui um parágrafo e um botão, onde o parágrafo mostra quantas vezes você clicou no botão da interface.

1. Criar um diretório chamado Components

> No diretório crie um componente chamado Exemplo.jsx

- 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, que é a usada no React, e faz as sugestões de correções para essa linguagem. Se você criar o arquivo com a extensão .js, você terá que mudar manualmente a linguagem selecionada.

Criando componentes funcionais

3) Agora insira o código abaixo no arquivo **Exemplo.jsx**:

```
1. import { useState } from 'react';
2.
3. const Exemplo = () => {
4.   const [contador, setContador] = useState(0);
5.
6.   return (
7.     <div>
8.       <p>Você clicou {contador} vezes</p>
9.       <button onClick={() => setContador(contador + 1)}>
10.        Clique aqui
11.      </button>
12.    </div>
13.  );
14. };
15.
16. export default Exemplo;
```

Observe que esse componente possui um estado chamado contador, que é iniciado com o valor zero, **useState(0)**, e uma função **setContador**, que é usada para atualizar o valor do estado, **setContador(contador + 1)**. Você só pode atualizar o estado de um componente com a função gerada pelo hook **useState**, que nesse exemplo é a **setContador**. Para utilizar o hook **useState** é necessário importá-lo no componente através da instrução na linha 1.

Criando componentes funcionais

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 não é permitido 

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

Isso está correto 

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

Vamos ver o resultado a nossa aplicação React.

1. No terminal entre no arquivo da sua aplicação React e execute a aplicação com o comando `npm start`.

Criando componentes funcionais

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



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.

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

```
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;
```

Criando uma aplicação em React



A linha 7 contém o nosso componente criado, indicando o local (ordem) que ele deverá ser renderizado. Note que devemos sempre usar a notação:

Abre o componente com o símbolo de menor que	Nome do componente	Espaço	Fechar o componente com barra e o sinal de maior que
<Exemplo />			

Vão ter casos que iremos colocar propriedade dentro da instrução do componente, mas por enquanto vamos entender de forma simples a instrução para renderização do componente.

IOS – Instituto de
Oportunidade Social

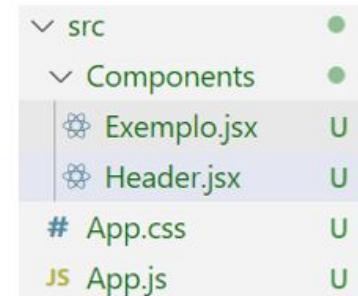
Inserindo um novo componente na
hierarquia



Um novo componente na hierarquia



1) Crie o arquivo Header.jsx dentro do diretório **Components**.



Um novo componente na hierarquia


2) Insira o seguinte código no arquivo Header.jsx.

```
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;
```


Um novo componente na hierarquia

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

Um novo componente na hierarquia

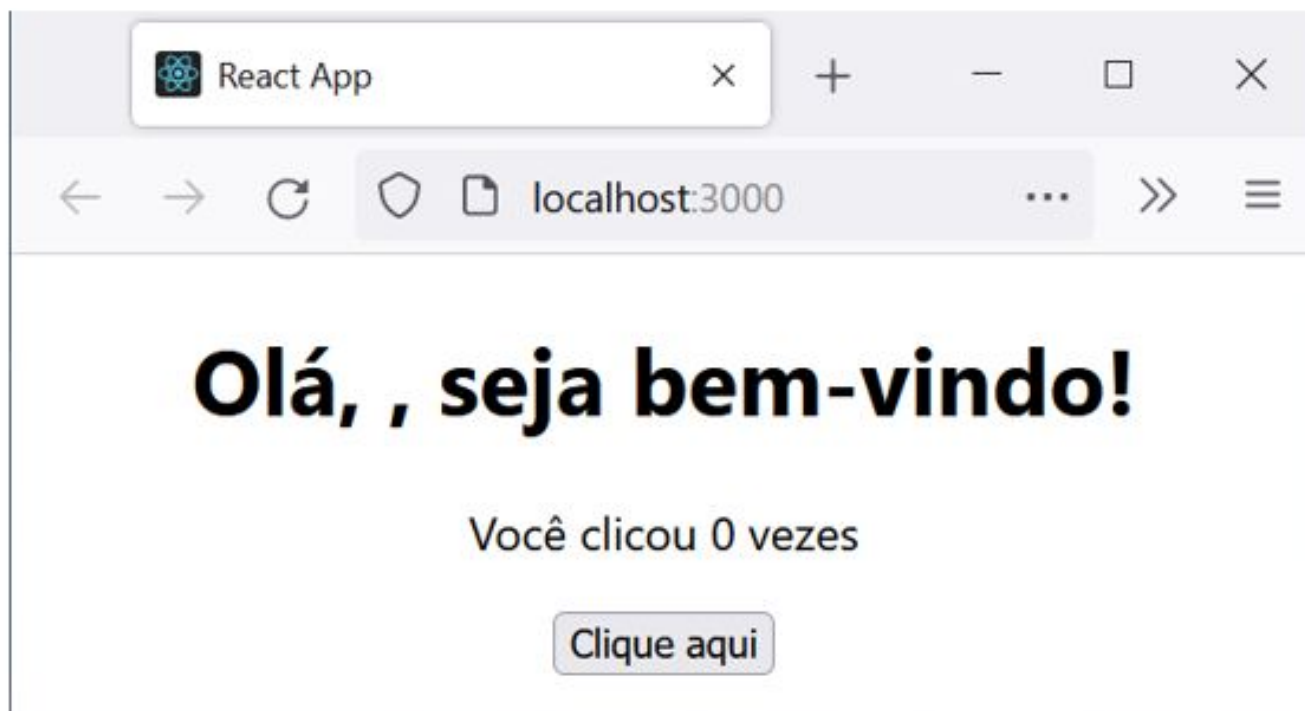


3) Com o componente criado, devemos atualizar o componente raiz App com o seguinte código:

```
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 />
9.       <Exemplo />
10.    </div>
11.  );
12. }
13.
14. export default App;
```

Um novo componente na hierarquia

Veja o resultado no navegador Web



Um novo componente na hierarquia

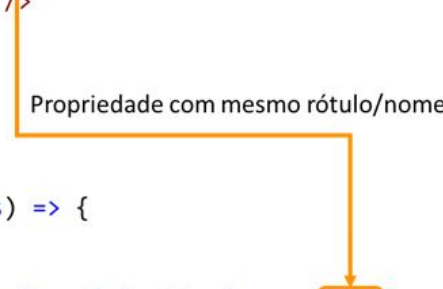
5) Agora, se atualizarmos o código do componente App na linha 8 passando o valor da propriedade nome:

```
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;
```

Um novo componente na hierarquia

Note que o nome da propriedade deve ser o mesmo que o esperado/acessado pelo componente:

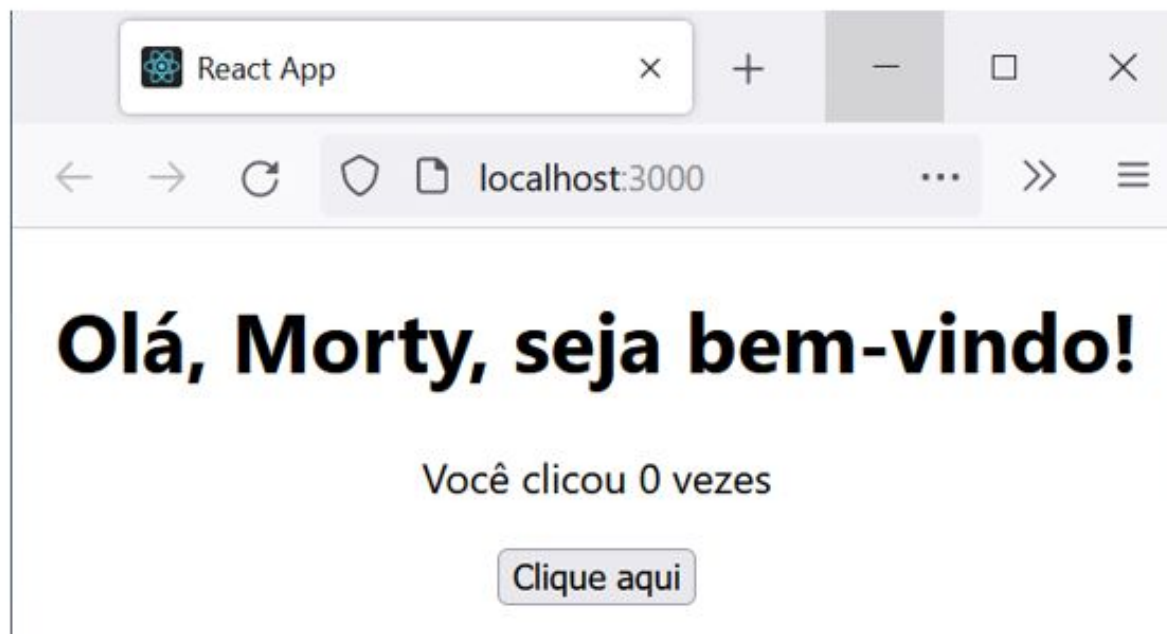
```
function App() {  
  return (  
    <div className="App">  
      <Header nome="Morty" />  
      <Exemplo />  
    </div>  
  );  
}  
  
const Header = (props) => {  
  return (  
    <div>  
      <h1 style={estilo}>Olá, {props.nome}, seja bem-vindo!</h1>  
    </div>  
  );  
};
```



Propriedade com mesmo rótulo/nome

Um novo componente na hierarquia

Veja o resultado no navegador Web



Um novo componente na hierarquia



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;
```

A instrução `Header.defaultProps` (linhas 9 a 11) configura valores padrão para as propriedades esperadas pelo componente. Desse modo, se o valor da propriedade não for passado pelo componente pai, teremos a exibição de valor padrão.

Um novo componente na hierarquia



Outra informação importante ao trabalhar com propriedades é que você pode definir o tipo de dados 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
<code>PropTypes.array</code> ,	Verifica se é um array
<code>PropTypes.bool</code> ,	Verifica se é um booleano
<code>PropTypes.func</code> ,	Verifica se é uma função
<code>PropTypes.number</code> ,	Verifica se é um número
<code>PropTypes.object</code> ,	Verifica se é um objeto
<code>PropTypes.string</code> ,	Verifica se é uma string
<code>PropTypes.symbol</code> ,	Verifica se é um símbolo

Criando uma aplicação em React



Por exemplo, na propriedade do componente Header estamos esperando uma string. Então podemos atualizar o código do componente com a seguinte instrução:

```
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;
```

IOS – Instituto de
Oportunidade Social

Adicionando um estilo a um
componente



Adicionando estilo a um componente



A partir desse ponto você irá seguir com o passo a passo na apostila esse será seu exercício de casa.

Suba o código no GitHub e disponibilize o link da atividade no Moodle.