1. Componentes React: componentes de classe e funcional

Este capítulo apresenta mais alguns conceitos de sobre os componentes de classe e funcional do React.

1.1 Comunicação entre componentes

Como vimos anteriormente, React é uma biblioteca que controla os componentes da nossa aplicação web e estes componentes podem ser divididos da seguinte forma:

- Componentes de classe;
- Componentes funcionais.

Os componentes permitem que a interface com o usuário (UI - *user interface*) seja dividida em partes independentes e reutilizáveis, ou seja, trata cada parte da aplicação como um bloco isolado, livre de outras dependências externas.

1.2 Componentes de classe

Os componentes de classe, também chamados de statefull omponent, são componentes que possuem um alto nível de poder dentro da aplicação. Além gerenciar o próprio estado, herdam os chamados métodos de ciclo de vida do React, lidam com partes lógicas da aplicação e manipulam eventos através de métodos que podem ser invocados em qualquer lugar do componente ou em seus filhos.

Cabe ressaltar que os componentes de classe na verdade não são específicos do React, eles são uma implementação que surgiu no ES6. Segundo o ES6, uma classe nada mais é que uma **função especial** na qual você pode definir utilizando a palavra reservada `class`. Assim, um componente de classe no React estende o `Component`, da biblioteca.

```
import React, { Component } from 'react';

class Hello extends Component {
 render() {
   return <h3>Olá, {this.props.name}</h3>
 }
}
```

1.3 Componentes funcionais

Os componentes funcionais, também chamados de stateless component, são representados por funções JavaScript e, costumavam não se preocupar com o gerenciamento de estado do componente, mas apenas com a apresentação dos dados na aplicação. Com o **Hooks**, esta situação sobre o gerenciamento de estado e acesso aos métodos do ciclo de vida em um componente funcional mudou, mas veremos isso mais adiante no curso.

Veja um exemplo de componente funcional sem estado:

```
import React from 'react';

const Hello = (props) => <h3>Olá, {props.name}</h3>
```

1.4 Componentes de classe ou componentes funcionais?

Nos exemplos acima, uma vez atribuído seus valores, o resultado final de ambos os componentes será o mesmo. Porém, embora os componentes façam exatamente a mesma coisa, algo diferente ocorre na forma como acessamos os dados recebidos por `props`. No componente de classe precisamos estender a classe `Component`, padrão do React, e acessar seus valores através do objeto `this`.

Já no componente funcional, o `this` não existe, pois o mesmo é apenas uma função que pode acessar os valores do objeto diretamente, exatamente como se estivesse recebendo esses dados por parâmetros.

Outra diferença está na simplicidade e na quantidade de linhas de código que são usadas para criar cada tipo de componente. Isso reflete também na performance da aplicação, uma vez que funções são executadas um pouco mais rapidamente do que classes.

OBSERVAÇÃO: Anteriormente, era sugerido a escolha de componentes de classe sempre que existissem tarefas mais complexas a serem feitas, como lidar com alguma lógica, gerenciar o estado e manipular eventos. Também era sugerido a escolha de componentes funcionais quando você precisava apenas exibir alguma UI (por serem mais simples e mais rápidos). Ainda, não havia um meio de fazer o gerenciamento de estado de um componente funcional. Porém, com o advento do Hooks, isso mudou. Hooks são uma nova adição ao React 16.8 que permitem que você use o **state** e outros recursos do React sem escrever uma classe.Cabe ressaltar que a documentação oficial do React menciona uma “estratégia gradual de adoção ao Hooks”.

Vejamos alguns exemplos na prática a seguir.

1.5 Criando nosso primeiro Componente (se você ainda não fez…)

Vamos utilizar o projeto criado na primeira aula de componentes, o my-components. Se você já criou este projeto, **podes pular para o item 1.6**. Caso contrário, vamos criar um projeto chamado my-components, digitando o seguinte comando no terminal:

```
yarn create react-app my-components
```

O processo leva alguns segundos e, logo após terminar, digite o seguinte comando para entrar no diretório recém criado do nosso projeto:

```
cd my-components
```

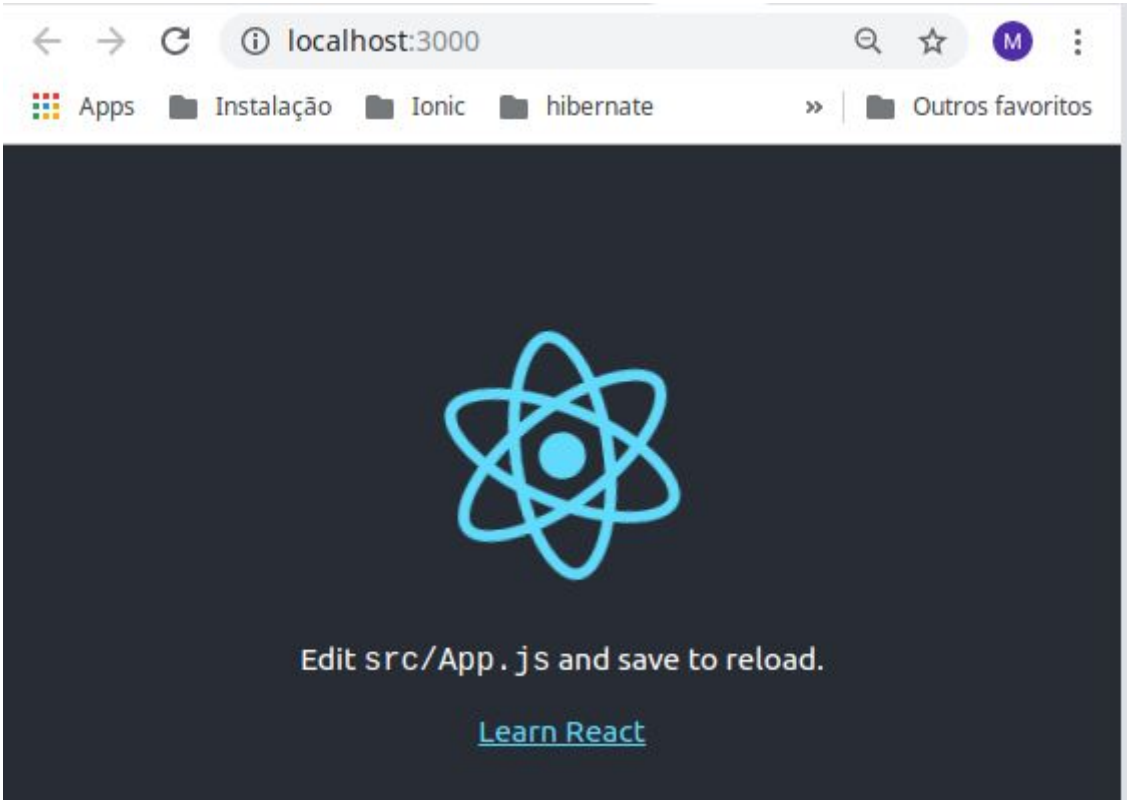
Você pode abrir a aplicação no editor Visual Studio Code. Para isso, dentro do diretório my-components digite o seguinte comando:

```
code .
```

O Visual Studio Code deverá abrir com a pasta my-components sendo acessada. Agora, vamos testar nossa aplicação. Digite a seguinte linha de comando:

```
yarn start
```

Ele irá rodar a aplicação em modo desenvolvimento em http://localhost:3000



A página será recarregada automaticamente se você fizer alterações no código. Agora, crie um diretório chamado componentes na pasta src.

1.6 Criando componentes funcionais e de classe

Dentro do diretório src/componentes, crie um arquivo chamado ComponenteClasse.js.

```
import React, { Component } from 'react';
export default class ComponenteClasse extends Component {
   render(){
       return (
           <h1>Olá Classe!</h1>
       )
   }
}
```

Veja que tem mais código um pouco mais verboso do que trabalhar com componente funcional.

Agora, vamos no index.js, para referenciar o componente ComponenteClasse:

```
import React from 'react';
import ReactDOM from 'react-dom';
```

```
import ComponenteClasse from './componentes/ComponenteClasse';

const elemento = document.getElementById('root');
ReactDOM.render(
   <div>
       <ComponenteClasse />
   </div>
   , elemento )
```

Veja como ficou o resultado na tela do navegador.

Analisando o código em ComponenteClasse.js, podemos observar que o método `render()` não recebe parâmetros. Então, como fazemos para acessar as propriedades? Para acessar os membros de uma classe nós usamos o `this`. Faça a seguinte alteração neste componente:

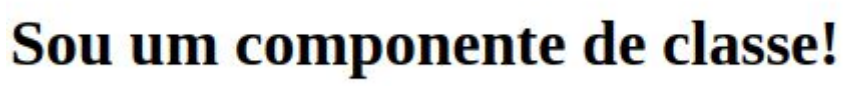
```
import React, { Component } from 'react';
export default class ComponenteClasse extends Component {
   render(){
       return (
           <h1>{this.props.valor}</h1>
       )
   }
}
```

Agora, vamos no index.js, para passar parâmetros ao componente ComponenteClasse:

```
import React from 'react';
import ReactDOM from 'react-dom';
import ComponenteClasse from './componentes/ComponenteClasse';

const elemento = document.getElementById('root');
ReactDOM.render(
   <div>
       <ComponenteClasse valor="Sou um componente de classe!"/>
   </div>
   , elemento );
```

Veja o resultado no navegador:



Também, podemos alterar a chamada ao componente ComponenteClasse para que ele pode mostre um `valor` que foi passado ou mostre um `valor` “PADRAO” quando o index não passar esta informação ao componente.

Altere o arquivo ComponenteClasse.js:

```
import React, { Component } from 'react';
export default class ComponenteClasse extends Component {
   render(){
       return (
           <h1>{this.props.valor || 'Padrão'}</h1>
       )
   }
}
```

E agora altere o index.js para não passar nenhuma propriedade para este componente.

```
import React from 'react';
import ReactDOM from 'react-dom';
import ComponenteClasse from './componentes/ComponenteClasse';

const elemento = document.getElementById('root');
ReactDOM.render(
   <div>
       <ComponenteClasse/>
   </div>
   , elemento );
```

Veja o resultado no navegador:

Agora, já trabalhamos anteriormente com a criação de diversos componente funcionais. Cabe ressaltar que em Javascript podemos ter uma função dentro de uma outra função. Assim, vamos criar um componente que contém uma outra função, dentro daquela função responsável por renderizar o componente.

Crie em src/componentes um arquivo chamado ComponenteComFuncao.js:

```
import React from 'react';
export default props => {
   const aprovados = ['Maria', 'Ana', 'João', 'Roberto'];
   const gerarItens = itens => {
       return itens.map(item => <li>{item}</li>);
   }
   return(
       <ul>
           {gerarItens(aprovados)}
       </ul>
   )
}
```

Veja que criamos um componente funcional que contém uma função arrow anônima (que recebe `props`). Dentro dela, temos uma constante de `aprovados` com uma lista com nome de pessoas. Vamos usar essa lista para renderizar os elementos de uma lista não ordenada (tag `<ul>` do HTML). Também temos uma arrow function anônima que atribui seu resultado para um constante `gerarItens`. Nesta função, esperamos receber um conjunto de `itens` como parâmetro. Veja que estamos transformando os `itens` em elementos JSX através do `map()`. O map transforma elementos de um array em outros tipos de elementos, neste caso, uma string de um array para um elemento JSX (serão itens de uma lista `<li>`).

No index.js, vamos chamar este componente funcional:

```
import React from 'react';
import ReactDOM from 'react-dom';
import ComponenteComFuncao from './componentes/ComponenteComFuncao';

const elemento = document.getElementById('root');
ReactDOM.render(
   <div>
       <ComponenteComFuncao/>
   </div>
   , elemento );
```

Veja o resultado na tela do navegador;

- Maria
- Ana
- João
- Roberto

Referências

- BANKS, Alex; PORCELLO, Eve. Learning React. O'Reilly Media, 2017.
- WIERUCH, Robin. The Road to learn React: Your journey to master plain yet pragmatic React. 2018
- Site Oficial do React. Disponível em: https://pt-br.reactjs.org/docs/glossary.html