Introdução ao ReactJS

Aplicações para a Internet I

Steven Abrantes, Tiago Rebelo e Frederico Fonseca

Sumário

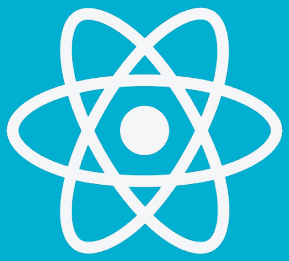
- Introdução ao ReactJS
- Componentes (*components*)
- Estado (*state*) e ciclo de vida (*lifecycle*)
- Manipulação de Eventos (*events*)
- Renderização condicional
- Listas e chaves
- Formulários
- Composição e herança

Introdução ao ReactJS

«É uma biblioteca JavaScript declarativa, eficiente e flexível para criar interfaces gráficas.»

(Facebook, 2011)

Introdução ao ReactJS



- Biblioteca **JavaScript** (JS) criada pelo **Facebook**;
  - Desenvolvido por **Jordan Walke** (2011);
  - *Open-Source* desde **2013**;
  - Utilizado pelo FB, Instagram, AirBNB, WhatsApp, Netflix, etc.;
- Tem como principal função a construção de interfaces gráficas;
  - Através de **componentes** UI reutilizáveis;
  - Baseado no conceito de *Web Components*;
- É parte integrante da arquitetura **MVC** - Model-View-Controller;
  - Neste caso da *View*;

Introdução ao ReactJS

- Faz a abstração do **DOM**, oferecendo um motor de manipulação mais simples - o chamado **Virtual DOM**;
- Suporte para ES6 ou superior (ES6+);
  - ES6 também pode ser chamado de **ECMAScript 6** ou **ES2015**;
  - De uma forma muito resumida, esta versão da linguagem resolve problemas antigos do JS, bem como permite a construção de aplicações mais complexas de uma forma mais simples, prática e escalável;

Introdução ao ReactJS

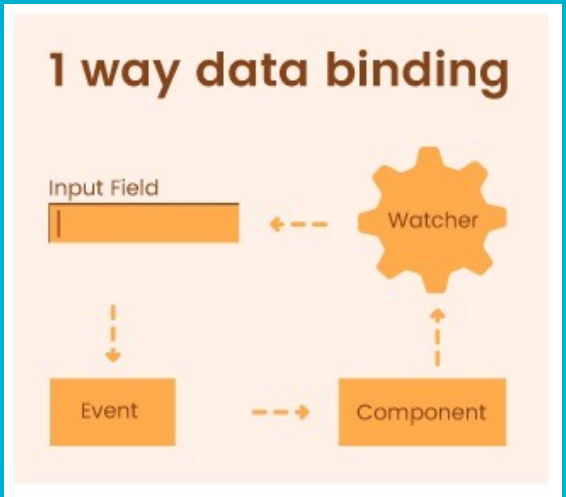
- Algumas das vantagens do *react* são:
  - Curva de aprendizagem rápida;
  - Sintaxe amigável (ES6+ ou JSX);
  - Possibilidade de reaproveitamento de componentes (logo, de código);
  - Otimização da manipulação do DOM *(Virtual DOM)*, alterando somente o necessário;
  - Comunidade muito ativa;
  - Entre outros...

Introdução ao ReactJS

- O **Virtual DOM** tem a capacidade de processar seletivamente subárvores de nós após a sua alteração de estado;
  - É esta "virtualização" do DOM que permite a sua renderização no servidor e a utilização de *views* no lado do servidor;
- Faz a menor quantidade possível de manipulações DOM para manter os componentes atualizados;
- Implementa o fluxo de dados reativo, de uma via (one-way), facilitando o *binding* dos dados;

Introdução ao ReactJS

- O chamado ***one-way data binding***:
  - É necessário invocar a função ***setState()*** para alterar o valor;
  - Desta forma temos muito mais previsibilidade e segurança na nossa aplicação;



* Retirado do artigo https://rubygarage.org/blog/the-angular-2-vs-react-contest-only-livens-up

Introdução ao ReactJS

- O *react* foi criado desde o início para ser utilizado de forma gradual;
  - É possível utilizar o React apenas para adicionar alguma interatividade a uma página HTML;
- Atualmente existe duas formas de utilizar o ReactJS:
  1. Via **NodeJS** *(a maneira mais correta e completa)*;
  2. Via **CDN** *(a maneira mais simples e que vamos utilizar em AI1)*;
     - A documentação do *react* refere esta abordagem como 'standalone';

Introdução ao ReactJS

- Um dos principais comandos em *react* é o ***createElement();***
  - A sua sintaxe é *React.createElement(**element**, **props**, **children**);*
    - ***element*** - div, p, h1, h2, span, header, main, etc.;
    - ***props*** - id, class, data, events, etc.;
    - ***children*** - outros elementos filho(s);
  - Analisemos um exemplo prático:

```
React.createElement('h1', {id: 'exemplo'}, 'Exemplo de elemento');
```

JS

O output seria:

<h1 id="exemplo">Exemplo de elemento</h1>

Introdução ao ReactJS

- Outro exemplo:

JSX

```
var menu = (
    <ul id="nav">
        <li><a href="#">Opção 1</a></li>
        <li><a href="#">Opção 2</a></li>
        <li><a href="#">Opção 3</a></li>
        <li><a href="#">Opção 4</a></li>
    </ul>
);
```

JS

```
var nav = React.createElement("ul", { id: "nav“ },
    React.createElement("li", null,
        React.createElement("a", { href: "#“ }, “Opção 1")),
    React.createElement("li", null,
        React.createElement("a", { href: "#“ }, “Opção 2")),
    React.createElement("li", null,
        React.createElement("a", { href: "#“ }, “Opção 3")),
    React.createElement("li", null,
        React.createElement("a", { href: "#“ }, “Opção 4"))
);
```

JSX

JavaScript Syntax eXtension

JavaScript Syntax eXtension (JSX)

- O **JSX** é uma extensão de sintaxe do JS;
  - É baseado em ECMAScript;
  - E permite-nos escrever XML/HTML dentro do JS;
    - Por exemplo, <tag />
- O código em JSX necessita de ser compilado (e.g, babel);
  - Este simplesmente converte o código do tipo XML/HTML em JS;

```
const element = <h1>Olá, Mundo!</h1>;
```

JSX

Seria convertido para...

```
const element = React.createElement("h1", null, "Olá, Mundo!");
```

JS

JavaScript Syntax eXtension (JSX)

- Algumas das vantagens do JSX são:
  - É mais fácil de entender e modificar por programadores menos experientes;
    - A maioria dos programadores *front-end* achará o JSX mais familiar do que o JS;
  - Permite tirar partido de todo o poder do JS utilizando HTML/XML;
    - O JSX é uma sintaxe declarativa utilizada para expressar uma estrutura em árvore de componentes da UI;
  - Entre outros;

Exemplo

Primeiro exemplo prático utilizando ReactJS...

Exemplo utilizando o ReactJS

- No ficheiro **HTML** escrevemos o código:

```
(...)
<head>
   (...)
   <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
   <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
   <script src="my_component.js"></script>
</head>

<body>

</body>
(...)
```

Exemplo utilizando o ReactJS

- No ficheiro **JS** (*my_component.js*) escrevemos o código:

JS

```
use 'strict';

class StateButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = { estado: false };
  }

  render() {
    let status;
    if (this.state.estado) { status = 'Estado atual: true'; } else { status = 'Estado atual: false'; }

    return React.createElement('div', null,
      React.createElement('h1', null, status),
      React.createElement('button', {
        onClick: () => this.setState({ estado: (this.state.estado==false) ? true : false; })
      }, 'Muda de Estado')
```

Component

do tipo *stateless* ou *stateful*

Componentes (*component*)

- Em termos gerais, podemos dizer que os **componentes** são pedaços (ou partes) que compõem um todo;
- No *react* os componentes permitem dividir a UI em partes independentes e reutilizáveis, permitindo pensar isoladamente cada um deles;
  - Os componentes são como funções em JS, aceitam parâmetros de entrada (chamados **props**) e retornam elementos **react**;
  - Estes dividem-se em 2 categorias: **presentational (ou stateless - sem estado)** e **container (ou stateful – com estado)**;

Componentes (*component*)

- Os componentes em *react* iniciam-se sempre com letra maiúscula;
  - Caso contrário será tratado como uma tag do DOM (body, p, div, etc.);

Componentes (*component*)

- **Componente Stateless**
  - Os componentes deste tipo somente se preocupam com a apresentação dos dados, portanto não tem estado;
  - Não devem alterar o valor dos *props* (propriedades de entrada);
    - É uma regra muito importante em *react*;
  - A maneira mais fácil de definir um componente deste tipo é escrevendo uma função JS;

```
function ExemploComponente() {
    return React.createElement("h1", null, "Olá, mundo!");
}
```

JS

O output seria:
<h1>Olá Mundo!</h1>

Componentes (*component*)

- Também é possível utilizar *props* neste tipo de componente;

```
function ExemploComponente(props) {
    return React.createElement("h1", null, "Olá, "+props.name+"!");
}
ReactDOM.render(React.createElement(ExemploComponente, {name:"AI1"}),
document.body);
```

JS

O output seria:
<h1>Olá , AI1!</h1>

- Também é possível definir esta função em **ES6+**:

```
const elemento = () => { return React.createElement("h1", null, "Olá Mundo!")};
```

O output seria:
<h1>Olá Mundo!</h1>

Componentes (*component*)

- **Componente Stateful**
  - É a forma mais recomendada de definir um componente;
  - Este tipo de componentes lida com a lógica de transformação dos dados, incluindo a apresentação dos mesmos;

JS

```
class ExemploComponente extends React.Component {
    constructor(props) {
        super(props);
        this.state = { nome:'AI1' };
    }

    render() {
        return React.createElement("h1", null, "Olá "+this.state.nome+"!");
    }
}
```

O output seria:
<h1>Olá AI1!</h1>

Componentes (*component*)

- **Componente Proprietário**
  - São iguais aos restantes componentes;
  - Mas utilizam uma ***tag*** criada por nós;
  - Analisemos um exemplo da sintaxe:

```
const elemento = React.createElement(ExemploComponente, { name: "valor_a_enviar“ });
```

JS

O output seria:
<ExemploComponente name=“valor_a_enviar” />

Componentes (*component*)

- Em qualquer um dos tipos de componentes é possível passar **parâmetros de entrada** (*props*) para o componente;
- Vejamos um exemplo:

JS

```
function ExemploComponente(props) {
    return React.createElement("exemplocomponente", null,
                React.createElement("h1", null, "Olá, "+props.name));
}
ReactDOM.render(React.createElement(ExemploComponente, { name: "Pedro" }), document.body);
```

O output seria:

```
<ExemploComponente>
   <h1>Olá, Pedro</h1>
</ExemploComponente>
```

Componentes (*component*)

- Por último, o *react* também permite criar composição de componentes, isto é, componentes que referenciam outros componentes, permitindo a mesma abstração para qualquer nível de detalhe;
  - Um botão, um formulário, uma caixa de diálogo, etc., são expressos como componentes;
- Vejamos um exemplo de um componente “Welcome” que é invocado (renderizado) várias vezes:

Componentes (*component*)

- Vejamos um exemplo de um componente “Welcome” que é invocado (renderizado) várias vezes:

JS

```
'use strict';

function Welcome(props) {
    return React.createElement("h1", null, “Olá, “+ props.name);
}

function App() {
    return React.createElement("div", null,
        React.createElement(Welcome, {name:"Pedro"}),
        React.createElement(Welcome, {name:"Tiago"})
    );
}
ReactDOM.render(App(), document.body);
```

O output seria:

```
<div>
  <h1>Olá, Pedro</h1>
  <h1>Olá, Tiago</h1>
</div>
```