TRYBE
Modulo II – Front-end

Bloco 11 – React

“JavaScript library for building user interfaces.”

Principal **biblioteca de construção de UI**'s, criada por facebook. Para construir telas.
Referências oficiais: https://pt-br.reactjs.org/ e https://github.com/facebook/create-react-app .

1) Primeiros conceitos do React

Conceito

Caracteristicas importantes (2):
- Permite construir telas de forma **declarativa**: declara como a tela vai **reagir** a um estado, a dados.
- É composta por **componentes**, ou seja permite criar uma tela como uma junção de diferentes pequenas peças reutilizáveis e com lógica isolada.

Vantagens:
- declarar e testar apenas a reação ajuda no processo de epuração do código
- todo em JS
- várias variações de React (web, mobile…).

JSX

JSX e é uma **extensão de sintaxe para JavaScript** parecida com html para descrever como o UI devia parecer, montando o DOM. React não requer obrigatoriamente o uso do JSX mas é uma ajuda visual.

```
const element = <h1>Hello, world!</h1>;
```

JSX é uma **expressão** que retorna objeto javascript, portanto pode usar em function, variável, argumento, etc.

Estes dois exemplos são idênticos:

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

```
const JSX = <div>
  <h1>Heading.</h1>
  <p>Paragraph</p>
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
</div>;
```

```
const element = React.createElement();
```

Especificidades da sintax JSX:
- Para adicionar atributos
“” para integrar string
{} para integrar expressões Js (equivalente de tag <script> basicamente)
- Algumas palavras diferentes como className, pois class já é palavra-chave do JS (click – onClick, onchange – onChange…).
- Convenção de sempre iniciar nomes dos componentes com **maiúsculas**, porque poraue React trata componentes começando com letras minúsculas como tags do DOM.
- **Elementos filhos**: podem ter, porem apenas um pai.
- {/*comment*/}
- fechar tags:
 <hr />

ReactDOM.render

Responsável por **renderizar o código dentro do HTML**. Ou seja “tornar visual”.
Diferente de elementos DOM do navegador, elementos React são objetos simples e utilizam menos recursos. O React DOM é o responsável por **atualizar o DOM para exibir os elementos React**.

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

Sintax
ReactDOM.render(componentToRender, DOMtargetNodeYouWantToRenderTo)
A ser chamado depois dos elementos JSX declarados.

Atualizar elementos React:
São imutáveis, porém ReactDOM é o único jeito de alterar. E React somente atualiza o necessário.

Criação de componentes funcionais e de classe

Componentes podem ser criados de duas formas: mediante funções ou com a criação de classes.

Classe
Definição: espécie de molde para criação de novos objetos.
Sintax:

```
class Pen {
  constructor(point, color, ink = 100) {
    this.point = point;
    this.color = color;
    this.ink = ink;
  }
}
```

constructor auxilia a criar, em forma de parâmetros de função, os atributos e métodos do objeto criado pela class.
Apenas pode existir um dentro do class.

Pode ter *default value* (*via =)* no construtor

*Conceito de herença: precisará inserir **super(props)** na linha imediatamente abaixo do constructor.*

Instância

Instanciar = Ato de criar um novo objeto a partir da class definida.

```
const myPen = new Pen (0.7, 'black');
console.log(myPen) // Pen { point: 0.7, color: 'black', ink: 100 }

const redPen = new Pen (1, 'red', 80);
console.log(redPen); // Pen { point: 1, color: 'red', ink: 80 }
```

//MyPen e redPen são dois novos objetos com a estrutura do primeiro.

Métodos

Uma classe pode possuir métodos, que nada mais são do que **ações** que todo objeto derivado de uma classe pode realizar, baseado em seus atributos.

```
sign() {
  this.loseInk(10);
  return `'I've signed a text with a ${this.color} pen. charge: ${this.ink}%'`;
}
```

//Esse code está localizado dentro do {} da classe Pen. Assim, todo objeto derivado desta classe possuirá, dentro de si, a capacidade de utilizar o método sign.

Classes e React

Uso de componentes por classe é diferente (comparado com por function) por ter acesso a métodos e ao estado próprios de qualquer componente React gerado via classe.

```
import React from 'react';

class Pen extends React.Component {
  render() {
    return (
      <h1>My first React Class Component!</h1>
    )
  }
}
```

método *render()* permite renderizar todo o conteúdo criado na tela, sintax:

//extends é uma herança: Pen herde propriedades do React.Component

Sintax

class Nameclass extends React.Component {
***render()** {*
return (
<insert JSX html stuff/>
)}}

Dicas diversas

Instalar

- Para criar uma aplicação em REACT (Instalar SEMPRE antes de fazer cada aplicação)
npx create-react-app (NOME DO PROJETO)
(npx é do npm e permite pegar apenas o que for de interesse)
- *CD* na ir pra pasta do projeto
- *npm start* (para ver o react no browser – é como um live server no react)
- *npm run build* (faz uma pasta com todos os arquivos necessários para a implementação)

```
File sizes after gzip:

  39.39 KB  build/static/js/2.9f22b3a7.chunk.js
  774 B     build/static/js/runtime-main.e2e49a21.js
  646 B     build/static/js/main.0ccbe467.chunk.js
  547 B     build/static/css/main.5f361e03.chunk.css

The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:

  npm install -g serve
  serve -s build

Find out more about deployment here:

  bit.ly/CRA-deploy
```

- *npm test* (comando de test, que já vem com um test default - similar ao Jest)

```
Success! Created react11.1 at /home/juliette/Documents/Curso Trybe/Trainings/exercises/11_1/react11.1
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd react11.1
  npm start
```

Organização de arquivos

- *HTML* - não tem html, ele depois irá na pasta build.
- *JS* - podem ter vários arquivos .js desde que apenas um app, ligado com todos. O importante é renderizar dentro do aplicativo mesmo. (*"it helps to separate the code responsible for the UI from the code responsible for handling your application logic")*.
- *CSS* - criar index.css (que seria como um css um nivel acima do style.css) e criar import.

index.js:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './App.css';

class MyHeader extends React.Component {
  render() {
    return (
      <div>
      <h1>Hello Style!</h1>
      <p>Add a little style!.</p>
      </div>
    );
  }
}

ReactDOM.render(<MyHeader />, document.getElementById('root'));
```

Importar com: *import nomeescolhido from ''* (exemplo: *import React from 'react';*) no começo do arquivo do App;
Exportar com: *export default nomecomponent* no final do arquivo .js do componente.

---

2) Componentes React

Uma **aplicação desenvolvida em React é composta de componentes.** Componentes permitem dividir a UI em partes independentes, reutilizáveis e pensar em cada parte isoladamente.

(O equivalente em Js de componente é a function, cada uma com responsabilidade independente facilitando trabalho de manutenção e identificação de possíveis erros).

```
var getProfilePic = function (username) {
  return 'https://photo.fb.com/' + username
}

var getProfileLink = function (username) {
  return 'https://www.fb.com/' + username
}

var getAvatarInfo = function (username) {
  return {
    pic: getProfilePic(username),
    link: getProfileLink(username)
  }
}

getAvatarInfo('tylermcginnis')
```

```
var ProfilePic = function() {
  return (
    <img src={'https://photo.fb.com/' + this.props.username} />
  )
}

var ProfileLink = function() {
  return (
    <a href={'https://www.fb.com/' + this.props.username}>
      {this.props.username}
    </a>
  )
}

var Avatar = function() {
  return (
    <div>
      <ProfilePic username={this.props.username} />
      <ProfileLink username={this.props.username} />
    </div>
  )
}

<Avatar username="tylermcginnis" />
```

// function na esquerda, componentes na direita.

Definição de componentes

Existem **dois jeitos de criar componentes**:

1. Componente de função: literalmente function Js. || 2. Componente criado com classe

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

// todo componente com class tem que usar render dentro

Renderizando um componente

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Props

"props" significa **propriedades**.
Props são **objetos** em que a gente **passa as informações para componentes** a partir de **chaves e valores** (igual qualquer outro objeto), é como o torna **reutilizável** os componentes.

Pensando em equivalente no Js, props são como parâmetros de uma function.
Inclusive, a chamada do componente lembra a chamada de uma função com passagem de parâmetros.

//componente Greeting

```
import React from 'react';

class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name} {this.props.lastName}</h1>;
  }
}

export default Greeting;
```

//chamado dentro do componente App

```
import Greeting from './Greeting';
function App() {
  return (
    <div>
      <Greeting name="Samuel" lastName="Silva" />
    </div>
  );
}
```

"In React, you can pass props, or properties, to child components."

Escrever **valor padrão** de props:
MyComponentName.defaultProps = { key: value };

To override a default props value, the syntax to be followed is

```
<Component propsName={Value}/>
```

Composição de componentes

Compondo componentes
"How do you build a big app? By building various small apps".
Componentes podem se referir a outros componentes em sua saída.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Sintax da chamada de componente dentro de componente:
<Namecomponente (action)/>.

*Importante também reparar que é **dentro da App** que atribuimos valores para **props**.*

Pais e filhos
Treinar de sempre detectar esse relacionamento entre componentes. Os dados são repassados de componente pai para componente(s) filho(s).

```
class ParentComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>I am the parent</h1>
< ChildComponent />
      </div>
    );
  }
};
```

Extraindo componentes
Boa prática de dividir componentes em componentes menores para ter uma paleta de componentes reutilizáveis. Critérios para separar: se usado mais de uma vez ou se já complexo em si.

```
const TypesOfFruit = () => {
  return (
    <div>
      <h2>Fruits:</h2>
      <ul>
        <li>Apples</li>
        <li>Blueberries</li>
        <li>Strawberries</li>
        <li>Bananas</li>
      </ul>
    </div>
  );
};

const Fruits = () => {
  return (
    <div>
< TypesOfFruit />
    </div>
  );
};

class TypesOfFood extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div>
        <h1>Types of Food:</h1>
< Fruits />
      </div>
    );
  }
};
```

Props: somente leitura
Todos os componentes React têm que agir como **funções puras em relação ao seus props**, ou seja nunca modificar eles (somente leitura).

Lista de componentes e chaves

Pode precisar **atribuir uma chave, um identificador,** para cada um dos elementos que renderiza.

→ Renderizando Múltiplos Componentes
Ex.1

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```

→

```
ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```

Ex. 2

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

//foi preciso adicionar key para evitar erro.

*// **chaves devem ser atribuídas aos elementos dentro do array** para dar uma identidade **estável** aos elementos.*

→ Chaves

Chaves ajudam o React a identificar quais itens sofreram alterações, foram adicionados ou removidos. Apenas fazem sentido no contexto do **array** que está encapsulando os itens.

Inside ChildElement: props.key
Inside ParentElement : <ChildElement key={[‘item array 1’, ‘item array 2’]} />

Regras de uso:

```
const todoItems = todos.map((todo) =>
  <li key={todo.id}>
    {todo.text}
  </li>
);
```

- Para definir a chave, pegar preferencialmente o **id** dos seus dados e na ausência deles, o **index** (evitar se ordem puder ser alterada).

- Chaves devem ser ***Únicas*** apenas entre Elementos *Irmãos.*

- Chaves não são passadas/acessíveis para os componentes filhos (precisando do mesmo valor, tem que criar prop).

Dicas diversas

.toLocateString(‘pt-BR’)
https://www.w3schools.com/jsref/jsref_tolocalestring.asp “The toLocaleString() method converts a Date object to a string, using locale settings.”
–
this: nele, componente jà pronto
–
Parsing para transformar em JS para leitura

–
Dentro de component ou function, return (code) funciona tanto quanto { js diretamente e return <> } :

→