IOS – Instituto de Oportunidade Social

HTML, CSS e JS 31 - Introdução ao React

Temas abordados

- Introdução ao React;
- JS Syntax Extension (JSX);
- Renderização de elementos;
- Componentes;
- Propriedades (Props);
- Estado do componente.

IOS – Instituto de Oportunidade Social

Introdução ao React

Introdução ao React

O que é ?

React é uma **biblioteca** usada para a construção de **interfaces**, ou seja, ele é utilizado para o desenvolvimento de aplicações do lado do cliente. O React é frequentemente chamado de um **“framework”** para **Frontend**, pois ele possui capacidade semelhantes e é diretamente comparado com os frameworks **Angular** e **Vue**.

Single Page Application (SPA)

Uma aplicação construída com o React utiliza o conceito de **Single Page Application** (SPA, aplicação de página única). SPA é uma aplicação web ou um site de uma única página, isto é, ele interage com o usuário reescrevendo dinamicamente a página atual com o novo dado solicitado. Isso quer dizer que esse tipo de aplicação, **recarrega apenas parte da interface** do usuário (UI, User Interface), que necessita ser atualizada.

Vantagens do (SPA)

Utilizar SPA no desenvolvimento de aplicações web é vantajoso, pois o **gerenciamento** da interface é realizado do **lado do cliente** e as **consultas no servidor** só são necessárias para buscas **novos dados** e não páginas inteiras. Por isso, as aplicações desenvolvidas com SPA fazem as **transições entre páginas mais rápidas** e o usuário tem a sensação de estar navegando por um aplicativo nativo.

Manutenção e Uso

O React é uma biblioteca criado e mantida pelo **Facebook** (trocou de nome em 2021 para **Meta**) e atualmente é a ferramenta mais popular usada para o desenvolvimento de SPA de acordo com o **Google Trends**, disponível no link:

https://trends.google.com/trends/explore?cat=31&q=Vue.js,React,Angular

Introdução ao React

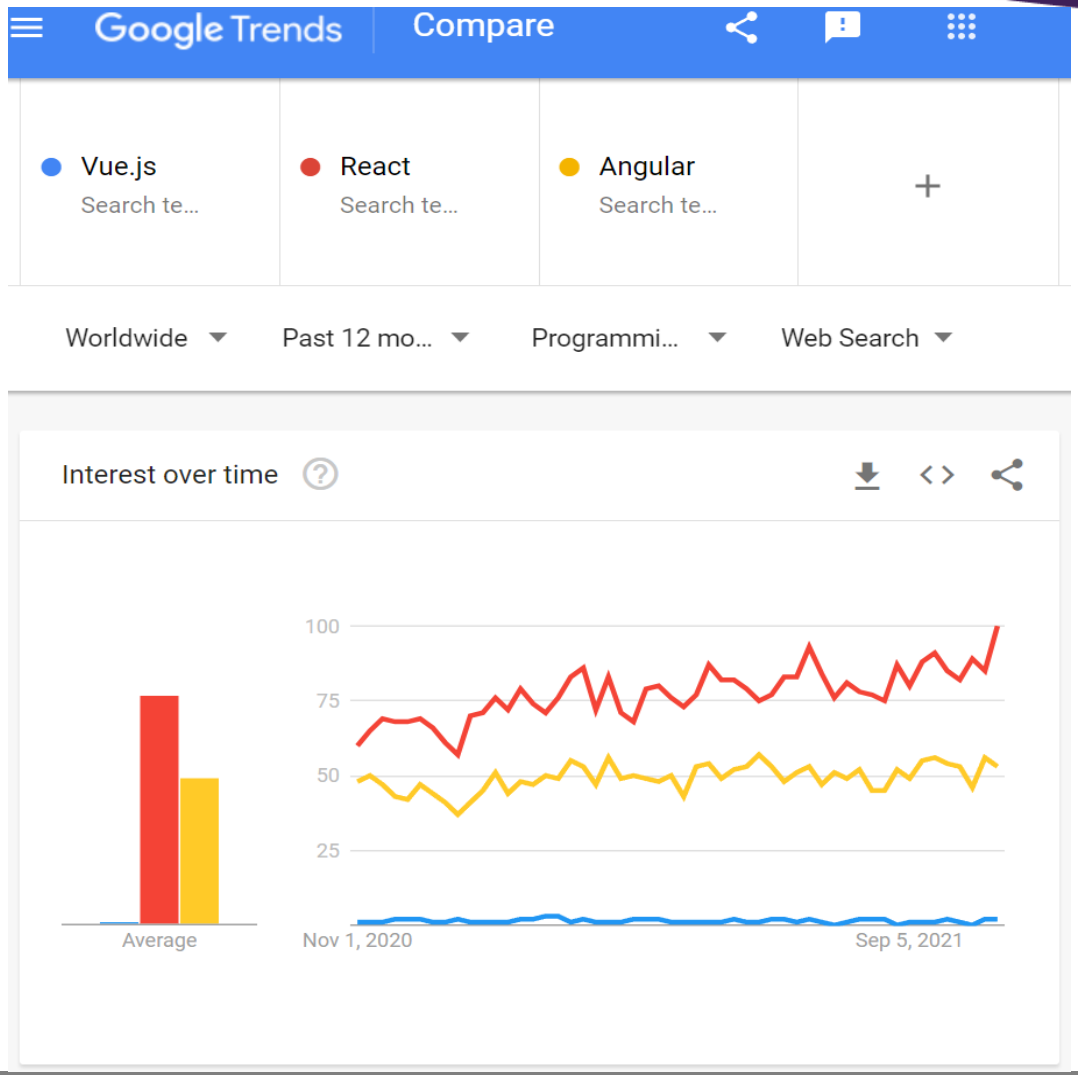


Gráfico de Utilização

IOS – Instituto de Oportunidade Social

JS Syntax Extension (JSX)

JS Syntax Extension (JSX)

O que é ?

O React utiliza uma sintaxe chamada **JSX** (JavaScript Syntax Extension), que é uma extensão da sintaxe do JavaScript e a documentação oficial (https://reactjs.org/docs) recomenda a utilização dessa linguagem para desenvolver a sua aplicação. A vantagem de desenvolver a sua aplicação React utilizando o JSX ao invés de JavaScript puro é que ela parece uma **linguagem de marcação**, que possui todo o poder de desenvolvimento provido pela JavaScript.

JS Syntax Extension (JSX)

Sintaxe JSX:

A sintaxe JSX é formada de algo que chamamos de elementos React:
const mensagem = <h1>Hello, world!</h1>;

A expressão acima não é uma string e nem uma expressão válida do JavaScript. Ela é simplesmente uma instrução em JSX que armazena na variável **mensagem** o elemento React **<h1>Hello, world!</h1>**. Esse elemento parece uma marcação de HTML.

JS Syntax Extension (JSX)

Mesclando JSX com JS:

Nesse exemplo abaixo, o conteúdo da variável **name** é incorporado na criação do elemento React <h1>, assim esse elemento pode mudar o conteúdo exibido dinamicamente. Porque quando o conteúdo da variável **nome** é alterado, o elemento React na variável **mensagem** também muda. Observe que, para incorporar a expressão JavaScript é necessário que você a insira entre chaves.

```
const nome = 'Irmão do Jorel';
const mensagem = <h1>Olá, {nome}</h1>;
```

JS Syntax Extension (JSX)

Invocando função JS mesclada com JSX:

Nesse outro exemplo, a função **formataNome** é chamada, construindo o nome completo (**firstName** + **lastName**)

```
function formataNome(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Irmão',
  lastName: 'do Jorel'
};

const mensagem = (
  <h1>
    Hello, {formataNome(user)}!
  </h1>
);
```

JS Syntax Extension (JSX)

Atributos nos elementos React:

Você pode usar atributos nos elementos React da mesma forma que no HTML, no exemplo abaixo note que para especificar o valor do atributo, você deve usar o literal entre aspas (Ex: **tabIndex="0"**) e quando você quer usar um valor de um objeto do JS, o mesmo deve ser colocado entre chaves e sem aspas (Ex: **src={user.avatarUrl}**).

```
const elemento01 = <div tabIndex="0"></div>;
const elemento02 = <img src={user.avatarUrl}></img>;
```

JS Syntax Extension (JSX)

Nomenclatura JSX:

Como o JSX está mais próximo do JavaScript do que do HTML, o **React DOM** usa a convenção **camelCase** de nomenclatura de propriedade em vez de nomes de atributos HTML. Sendo assim, o atributo **class** do HTML torna-se **className** em JSX e **tabindex** torna-se **tabIndex**.

JS Syntax Extension (JSX)

Hierarquia de elementos:

O React pode ter **apenas um elemento pai** e vários filhos, isto é, para ser possível a **renderização**. No exemplo a seguir teremos, uma sintaxe inválida para o React, pois as marcações <h1> e <h2> estão no primeiro nível da hierarquia, ou seja, temos dois elementos pais. Já a sintaxe válida com um elemento pai e dois elementos filhos <h1> e <h2>.

JS Syntax Extension (JSX)

Sintaxe inválida x válida:

Sintaxe inválida	Sintaxe válida

`const mensagem = (`
`  <h1>Olá!</h1>`
`  <h2>Como é bom ver vocês.</h2>`
`);`
	`const mensagem = (`
	`  <div>`
	`  <h1>Olá!</h1>`
	`  <h2>Como é bom ver vocês.</h2>`
	`  </div>`
	`);`

(a)	(b)

A sintaxe JSX também permite a marcação vazio como pai.

```
const mensagem = (
  <>
    <h1>Olá!</h1>
    <h2>Como é bom ver vocês.</h2>
  </>
);
```

IOS – Instituto de Oportunidade Social

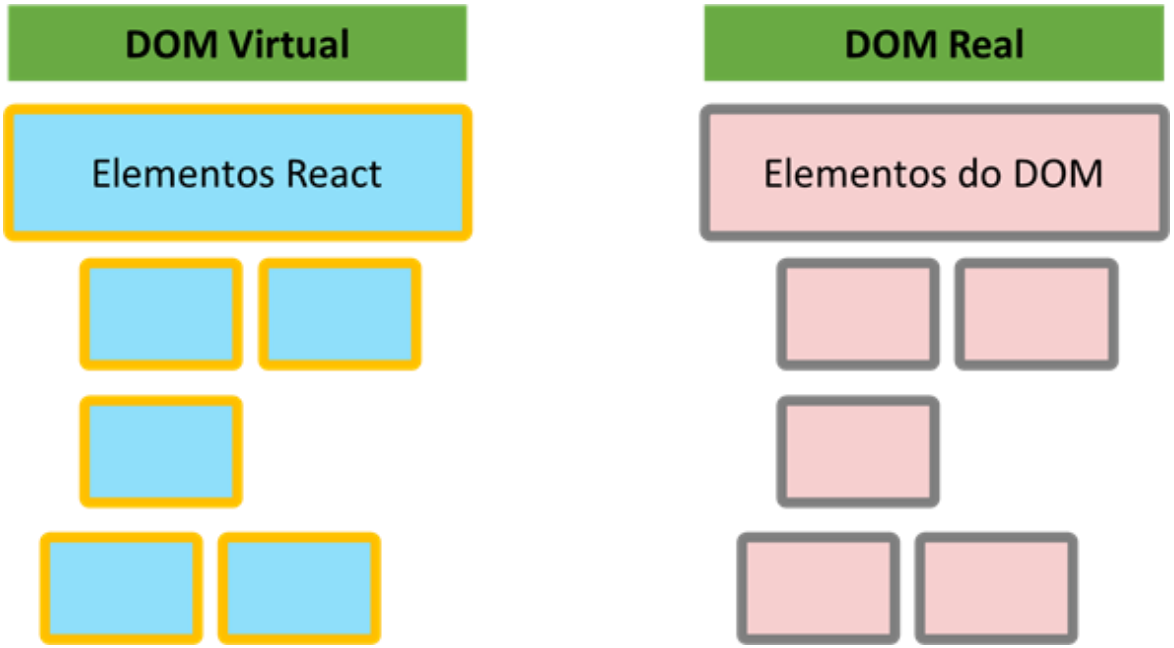
Renderização de elementos

Renderização de elementos

Um elemento React descreve o que você visualiza na interface que está sendo desenvolvida. Diferente de elementos DOM do navegador, elementos React são objetos simples e utilizam menos recursos para serem criados e gerenciados. O React possui um **DOM Virtual**, que é formando pelos elementos React e é responsável por atualizar o DOM do navegador (DOM Real) para exibir os **elementos React renderizados** como marcações HTML na página web. O DOM Virtual é simples e barato de criar (**requer menos recursos**), pois sempre que um elemento React é criado ou tem seu **estado alterado**, ele se encarrega de **atualizar o DOM Real**. Esse é o motivo da biblioteca se chamar **React (Reagir)**, pois ela "reage" a **mudanças de estado** e atualiza do DOM.

Renderização de elementos

Todo elemento React é **renderizado** na página web. Você deve ter em mente, que renderizar algo, nesse contexto, é transformar uma sintaxe do JSX em algo que vai ser **exibido** na página web.



Renderização de elementos

Método render

É comum que no arquivo **index.html** de uma aplicação React exista a marcação da seguinte forma:
** **

Essa marcação é chamada de **nó raiz do DOM**, pois toda a aplicação React será renderizada e gerenciada dentro dessa marcação. Para renderizar um elemento React em um nó raiz como o mostrado anteriormente, deve-se utilizar o método **render()**. Ex:

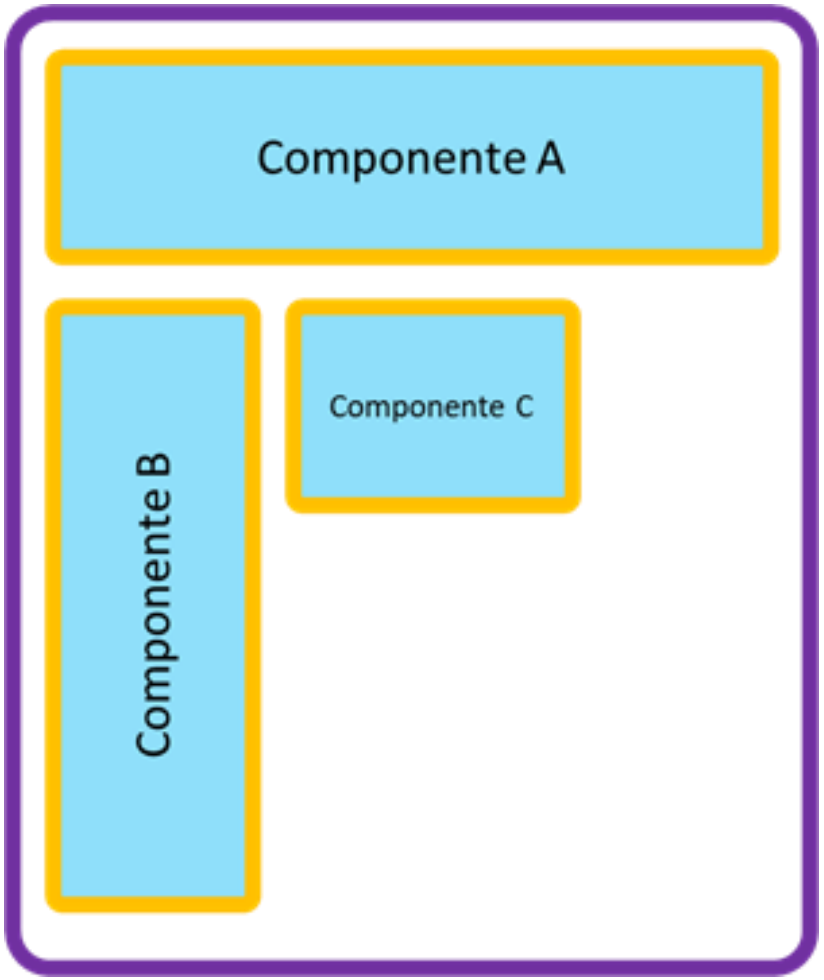
```
const mensagem = <h1>Hello, world</h1>;
ReactDOM.render(mensagem, document.getElementById('root'));
```

IOS – Instituto de Oportunidade Social

Componentes

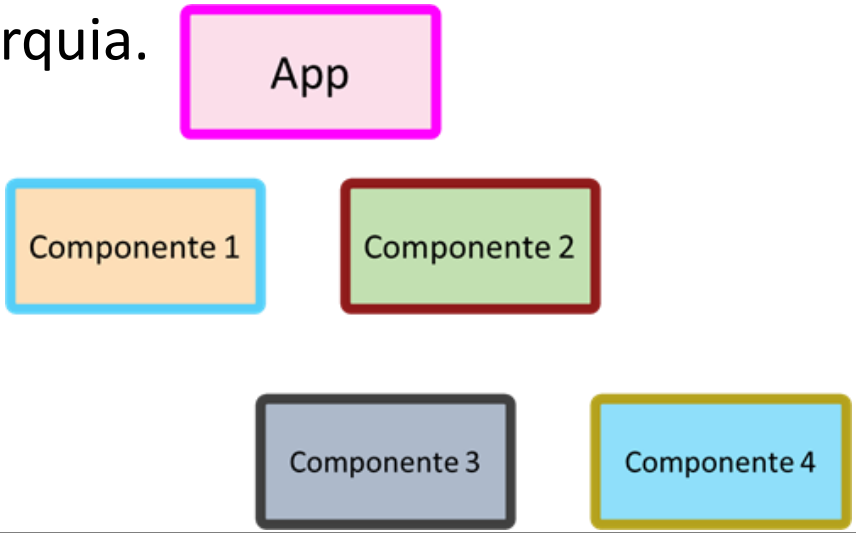
Componentes

O coração de toda aplicação desenvolvida em React são componentes. Os componentes são essencialmente uma parte da interface do usuário e podem ser formados por um ou mais elementos React. Então quando estamos construindo aplicação com React, nós estamos construindo vários **componentes independentes**, isolados e reutilizáveis. E esses componentes juntos são usados para compor uma **interface complexa**.



Componentes

Todas as aplicações React têm pelo menos um componente, que é referenciado como **componente raiz** e na maioria das aplicações que você encontrar o nome desse componente é **App**. Esse componente representa a aplicação interna e contém outros componentes filhos. Então, toda aplicação React é essencialmente uma árvore de componentes, onde existe um único componente raiz (App), que está no topo da hierarquia.



Componentes

Divisão de componentes:

A seguir vamos ver um exemplo, de divisão de componentes da página do Facebook do IOS de forma detalhada. E esses componentes em conjunto constroem a interface da página do Facebook.

Componentes

Componente NavBar1

Componente ImgSup

Componente Cabeçalho

Componente NavBar2

Componente CreatePost

Componente Posts

Componente About

Componentes

Implementação por Classe ou Função:

Em termos de implementação, o componente pode ser implementado como uma **classe** ou como uma **função** do JavaScript. **Componentes funcionais** foram introdução no React a partir da **versão 16.8** no final de 2019 com a inserção do conceito de **hooks**, até então os componentes eram criados apenas utilizando classes. Atualmente, é bem provável que você irá encontrar em uma empresa uma **implementação híbrida**, que utiliza a abordagem de componentes com classe e componentes funcionais.

Componentes

Exemplos Function x Class:

```
function Mensagem() {
  return <h1>Olá, pessoal!</h1>;
}
```

```
class Mensagem extends React.Component {
  render() {
    return <h1>Olá, pessoal!</h1>;
  }
}
```

IOS – Instituto de Oportunidade Social

Propriedades (Props)

Propriedades (Props)

Componentes podem ter **props** (Abreviação de propriedades), que é a maneira de **passar uma informação** para um **componente**. Props também são usadas para passar dados de um componente pai para um componente filho na hierarquia de componentes. Por exemplo, o **componente Mensagem** mostrado no exemplo a seguir possui uma props chamada **name**, que é passada na criação do elemento React (**const elemento = <Mensagem name="Irmão do Jorel" />**). Na instrução <Mensagem name="Irmão do Jorel" />, a propriedade é passada como se fosse um **atributo** de uma marcação em **HTML**.

Propriedades (Props)

Exemplo prop name:

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

const elemento = <Mensagem name="Irmão do Jorel" />;

ReactDOM.render(
  elemento,
  document.getElementById('root')
);
```

Propriedades (Props)

Nomes das propriedades:

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

const elemento = <Mensagem name="Irmão do Jorel" />;
```

Nomes iguais

```
function Mensagem(props) {
  return <h1>Olá, {props.lala}</h1>;
}

const elemento = <Mensagem lala="Irmão do Jorel" />;
```

Nomes iguais

IOS – Instituto de Oportunidade Social

Estado do componente

Estado do componente

Os componentes no React não são apenas marcações estáticas, eles têm um conteúdo dinâmico chamado **estado**. O estado é basicamente **um objeto**, que determina como um componente será **renderizado**, ou seja, o seu comportamento dentro da UI. Um estado criado no componente **raiz (App)** é considerado **global**, ou seja, esse estado pode ser visto por todos os componentes da aplicação e não somente pelo componente raiz.

Estado do componente

Se o **estado** de um componente for **modificado**, isso irá provocar a **renderização** do componente e **atualização automática** da parte da **UI** que esse componente representa. Essa é a **"magia"** por trás do React, reagir atualizando automaticamente as partes que compõe da interface do usuário.

Os exemplos de código a seguir são de um **componente funcional** chamado **Exemplo** com o estado contador e a utilização do **hook useState**, em componentes implementados com função, e o código utilizando classe com o **estado (state)** contendo o campo **contador**.

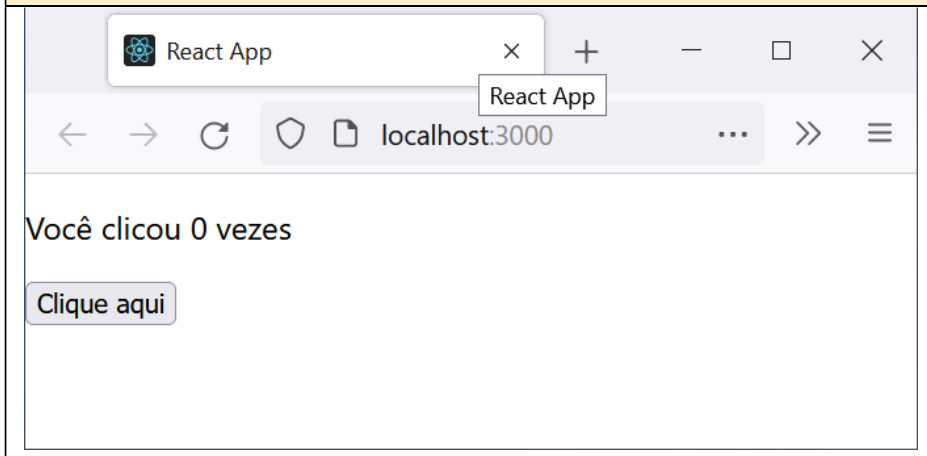
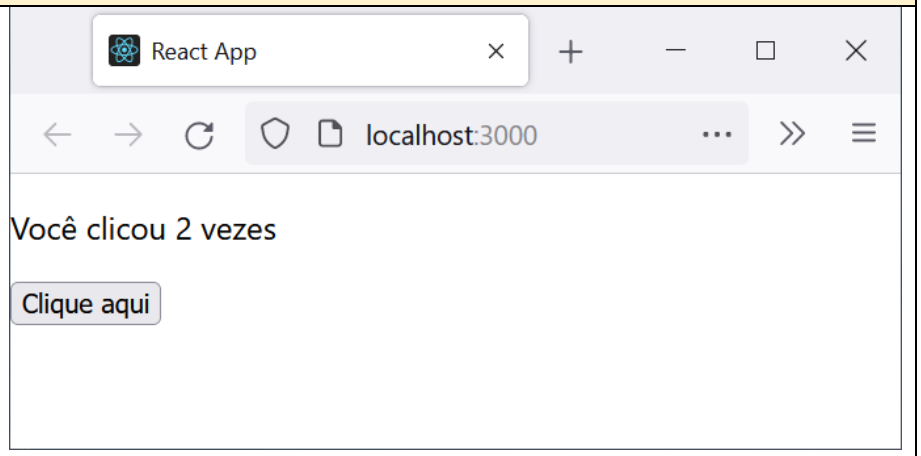
Estado do componente

```
import { useState } from 'react';

function Exemplo() {
  // Declara uma nova variável de estado, a qual chamaremos de "contador"
  const [contador, setContador] = useState(0);

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

Estado do componente

```
class Exemplo extends Component {
    // Declare uma nova variável de estado, a qual chamaremos de "contador"
    // e é um campo do objeto state
    constructor(props) {
        super(props);
        this.state = {
            contador: 0,
        };
    }

    render() {
        return (
            <div>
                <p>Você clicou {this.state.contador} vezes</p>
                <button
                    onClick={() =>
                        this.setState({ contador: this.state.contador + 1 })
                    }
                >
                    Clique aqui
                </button>
            </div>
        );
    }
}
```

Estado do componente

Aplicação iniciado no navegador.	Aplicação React depois de ter clicado por duas vezes no botão.
	

IOS – Instituto de Oportunidade Social

Porque usar React ?

Porque usar React ?

Existem diversos motivos para você utilizar o React no desenvolvimento de suas aplicações, podemos citar alguns:

O React estrutura a parte visual da sua aplicação. Se você tentar construir uma **aplicação grande** e **complexa** utilizando apenas **Vanilla JavaScript**, a sua implementação tem grandes chances de se tornar uma bagunça com os códigos HTML, estilos CSS e JavaScript por todo lugar que for possível.

O React permite você construir uma interface do usuário utilizando **componentes reutilizáveis**. Desse modo, cada parte da interface é representada por um componente com estado, dados e métodos.

Porque usar React ?

O React utiliza a **sintaxe JSX**, que é uma **linguagem de marcação** que não necessita separar a parte de marcação e a parte lógica como acontece com o HTML e JavaScript. A linguagem é basicamente o JavaScript, porém é formatada como se fosse o HTML.

As aplicações construídas com o React são bem interativas, pois elas usam o **DOM Virtual**, que permite **atualizar parte da página** sem a necessidade de recarregá-la completamente. O gerenciamento da página sendo feito pelo DOM Virtual, torna a aplicação mais rápida, dinâmica e interativa.

O React traz benefícios de **performance** e para a **realização de testes** da sua aplicação.

IOS – Instituto de Oportunidade Social

Exercícios

Exercícios

- Avaliar a estrutura da página Web que está sendo criada através do projeto e realizar um desmembramento visual de componentes, como no exemplo utilizado do **Facebook do IOS**.

OBS: Não é necessário gerar código fonte nesse momento e nem saber como será feito a quebra por componentes, apenas uma avaliação empírica da página em forma de planejamento (nesse momento tentar pensar como um Gestor de Projetos e não como Desenvolvedor). **Integrantes do grupo podem ter visões diferentes**.