

Desafio 01

Partindo do *Olá Mundo* (<https://stackblitz.com/edit/react-7y9vcj>), deverá ser alterado da seguinte forma:

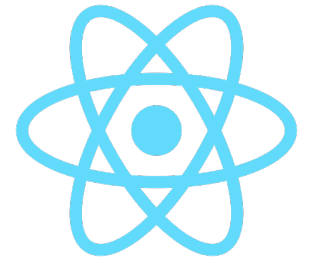
- na classe principal **App**, devem ser adicionados 2 novos métodos: **getTitulo** e **getParagrafo**.
- getTitulo deve receber um parâmetro de texto e retornar um JSX com o texto envolvido pela tag **h1**.
- getParagrafo deve receber dois parâmetros: nome e texto. O parâmetro nome deve ser envolvido pela tag de negrito (**b**), seguido do parâmetro texto, então a função deve retornar um JSX ambos envolvidos pela tag **p**.

Desenvolvedor Full-Stack

Objetivo: Aprender tecnologias incríveis para construir coisas magníficas

Tecnologias aprendidas: JavaScript, TypeScript, ReactJS, Angular, Python, NodeJS entre outras

← Resultado do desafio no navegador



Arquivo HTML

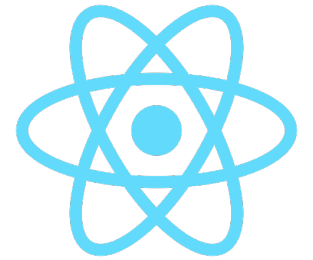
```
<div id="root"></div>
```

Desenvolvedor Full-Stack

Objetivo: Aprender tecnologias incríveis para construir coisas magníficas

Tecnologias aprendidas: JavaScript, TypeScript, ReactJS, Angular, Python, NodeJS entre outras

Resolução desafio anterior



Arquivo JS – parte 01

```
import React, { Component } from 'react';
import { render } from 'react-dom';
import './style.css';

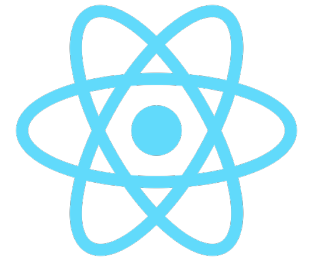
class App extends Component {
  constructor() {
    super();
    this.curso = {
      nome: 'Desenvolvedor Full-Stack',
      objetivo: 'Aprender tecnologias incríveis para construir coisas magníficas',
      tecnologias: 'JavaScript, TypeScript, ReactJS, Angular, Python, NodeJS entre outras'
    };
  }

  getTitulo(texto) {
    return <h1>{texto}</h1>
  }
}
```

Objeto representando o conteúdo do componente através de suas propriedades

Esse método pega um parâmetro e devolve o JSX como <h1>

Resolução desafio anterior



Arquivo JS – parte 02

```
getParagrafo(nome, texto) {  
  return <p>  
    <b>{nome}</b>  
    {texto}  
  </p>  
}
```

Esse método recebe dois parâmetros e devolve o JSX correspondente a cada um deles.

```
render() {  
  return (  
    <div>  
      { this.getTitulo( this.curso.nome ) }  
  
      { this.getParagrafo( 'Objetivo', this.curso.objetivo ) }  
  
      { this.getParagrafo( 'Tecnologias aprendidas', this.curso.tecnologias ) }  
    </div>  
  );  
}
```

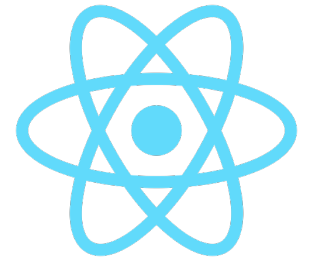
Esse é o método principal do componente que renderiza todo o seu conteúdo.

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



Esse é o método principal que renderiza a raiz do componente e coloca o seu conteúdo na página html.

Flávio Mota

Resolução desafio anterior

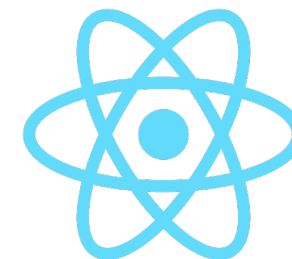


Arquivo CSS

```
h1, p {  
  font-family: Verdana, Geneva, Tahoma, sans-serif;  
}  
  
h1 {  
  color:  darkblue  
}  
  
p {  
  color:  cadetblue  
}
```

Link resolução

<https://stackblitz.com/edit/react-fuh1tg>



Renderizando um Elemento no DOM

Suponhamos que exista um `<div>` em algum lugar do seu código HTML:

```
<div id="root"></div>
```

Nós o chamamos de nó raiz do DOM porque tudo dentro dele será gerenciado pelo React DOM.

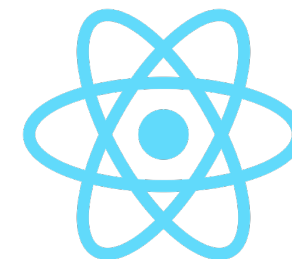
Aplicações construídas apenas com React geralmente tem apenas um único nó raiz no DOM. Se deseja integrar o React a uma aplicação existente, você pode ter quantos nós raiz precisar.

Para renderizar um elemento React em um nó raiz, passe ambos para `ReactDOM.render()`:

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

Elementos são os menores blocos de construção de aplicativos React.

Um elemento descreve o que você quer ver na tela:



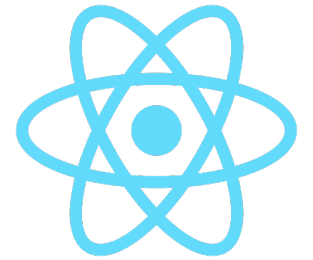
Atualizando o Elemento Renderizado

Elementos React são imutáveis. Uma vez criados, você não pode alterar seus elementos filhos ou atributos.

Com o que aprendemos até agora, a única forma de atualizar a interface é criar um novo elemento e passá-lo para `ReactDOM.render()`.

Veja o seguinte exemplo de um relógio:

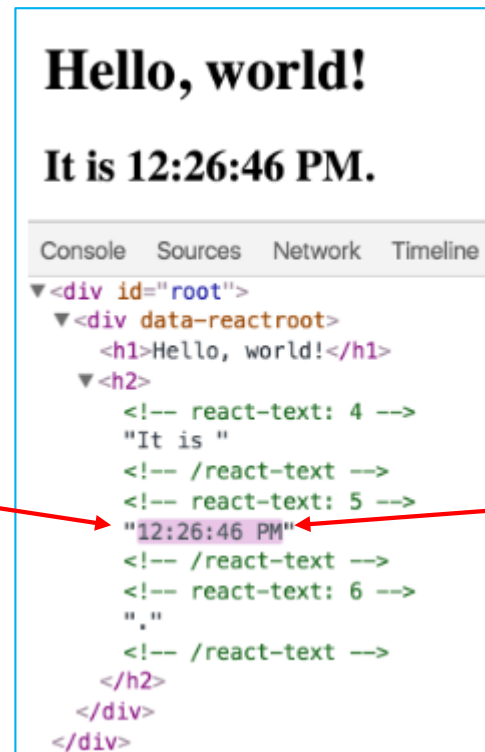
```
function tick() {  
  const element = (  
    <div>  
      <h1>Hello, world!</h1>  
      <h2>It is {new Date().toLocaleTimeString()}</h2>  
    </div>  
  );  
  ReactDOM.render(element, document.getElementById('root'));  
}  
  
setInterval(tick, 1000);
```



O React atualiza somente o Necessário

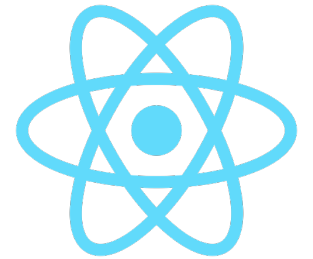
O React DOM compara o elemento novo e seus filhos com os anteriores e somente aplica as modificações necessárias no DOM para levá-lo ao estado desejado.

Nesse caso será atualizado somente esse bloco da página



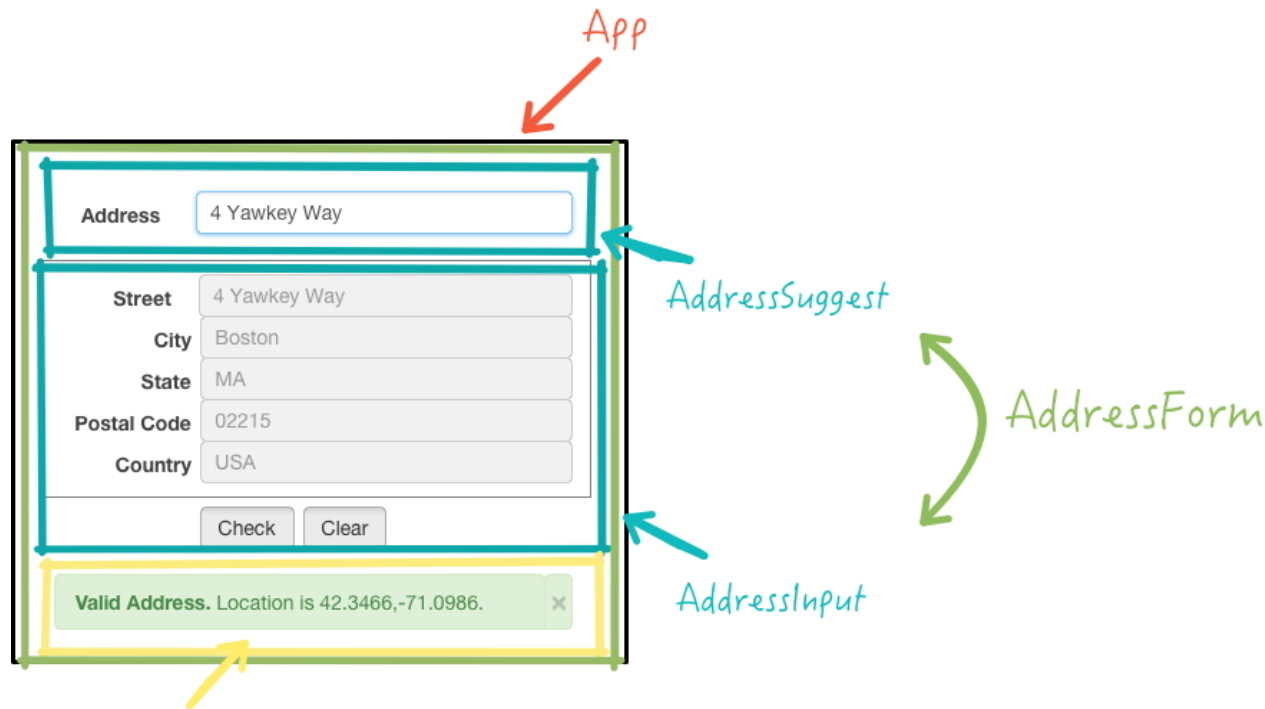
```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById('root'));
}

setInterval(tick, 1000);
```

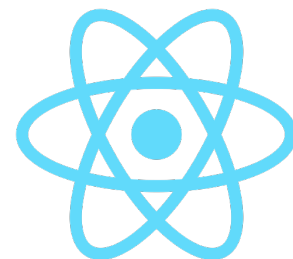
Componentes permitem você dividir a UI em partes independentes, reutilizáveis e pensar em cada parte isoladamente.

Conceitualmente, componentes são como funções JavaScript. Eles aceitam entradas arbitrárias (chamadas “props”) e retornam elementos React que descrevem o que deve aparecer na tela.



Fonte: <https://pt-br.reactjs.org/docs/components-and-props.html>

Fonte: <https://developer.here.com/blog/street-address-validation-with-reactjs-and-here-geocoder-autocomplete>



Componentes de Função e Classe

A maneira mais simples de definir um componente é escrever uma função JavaScript:

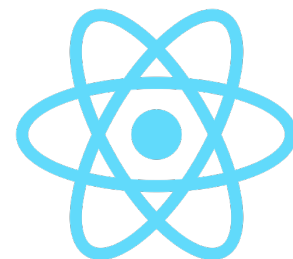
```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

Essa função é um componente React válido porque aceita um único argumento de objeto "props" (que significa propriedades) com dados e retorna um elemento React. Nós chamamos esses componentes de "componentes de função" porque são literalmente funções JavaScript.

Você também pode usar uma classe ES6 para definir um componente:

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

Os dois componentes acima são equivalentes do ponto de vista do React.

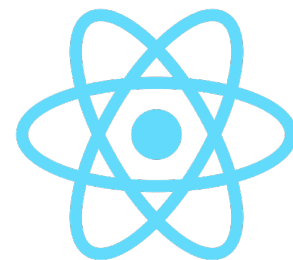


```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
const element = <Welcome name="Sara" />;  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
);
```

Try it on CodePen

Vamos recapitular o que acontece nesse exemplo:

1. Nós chamamos `ReactDOM.render()` com o elemento `<Welcome name="Sara" />`.
2. React chama o componente `Welcome` com `{name: 'Sara'}` como props.
3. Nosso componente `Welcome` retorna um elemento `<h1>Hello, Sara</h1>` como resultado.
4. React DOM atualiza eficientemente o DOM para corresponder `<h1>Hello, Sara</h1>`.



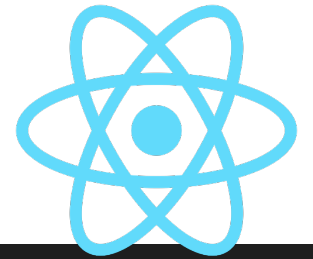
Compondo Componentes

Componentes podem se referir a outros componentes em sua saída. Isso nos permite usar a mesma abstração de componente para qualquer nível de detalhe. Um botão, um formulário, uma caixa de diálogo, uma tela: em aplicativos React, todos esses são normalmente expressos como componentes.

Por exemplo, nós podemos criar um componente `App` que renderiza `Welcome` muitas vezes:

```
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')  
);
```

Exemplo - Props



Index.js parte 01

```
class MyButton extends React.Component {
  render() {
    return (
      <button>{this.props.nome}</button>
    )
  }
}

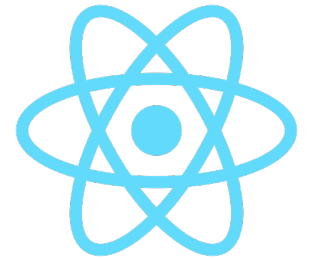
class MyLabel extends React.Component {
  render() {
    return (
      <p>{this.props.texto}</p>
    )
  }
}
```

Index.js parte 02

```
class App extends React.Component {
  render() {
    return (
      <div>
        <MyLabel texto="Recode Pro 2019"/>
        <MyButton nome="Botão 01"/>
        <MyButton nome="Botão 02"/>
        <MyButton nome="Botão 03"/>
      </div>
    )
  }
}

ReactDOM.render(<App />, document.getElementById("conteudo"))
```

Exemplo - Props



Index.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset='utf-8'>
  <meta http-equiv='X-UA-Compatible' content='IE=edge'>
  <title>Exemplo Props</title>
  <meta name='viewport' content='width=device-width, initial-scale=1'>
</head>

<body>

  <div id="conteudo"></div>

  <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="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
  <script type="text/babel" src="index.js"> </script>

</body>

</html>
```

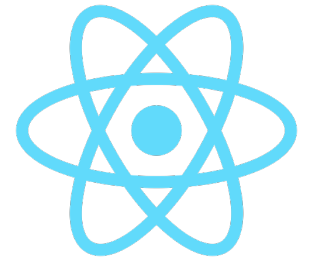
Saída no navegador

Recode Pro 2019

Botão 01

Botão 02

Botão 03



Desafio

Neste desafio, vamos alterar o projeto anterior e construir 2 novos componentes: *CursoHeader* e *CursoContent*. O primeiro cria o título e o segundo criando as linhas seguintes, ambos recebendo os dados por **props**.

Desenvolvedor Full-Stack

Objetivo: Aprender tecnologias incríveis para construir coisas magníficas

Tecnologias aprendidas: JavaScript, TypeScript, ReactJS, Angular, Python, NodeJS entre outras

Resultado no navegador

