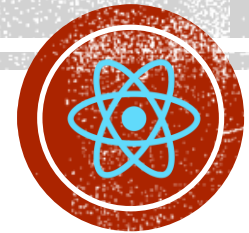


# SINGLE PAGE APPLICATIONS COM REACT-DOM



**Douglas Nassif Roma Junior**

 /douglasjunior

 /in/douglasjunior

 smarppy.com

 douglas@smarppy.com

# AGENDA

| Tópico          | Conteúdo  |
|-----------------|---|
| <b>webpack</b>  | <ul style="list-style-type: none"><li>- Introdução ao ambiente webpack</li><li>- Loaders</li><li>- Plugins</li><li>- Configurações</li></ul>  |
| <b>React JS</b> | <ul style="list-style-type: none"><li>- Introdução ao ReactJS</li><li>- Componentes<ul style="list-style-type: none"><li>-- Componente funcional</li><li>-- Componente de classe</li></ul></li><li>- ECMAScript 2015 e JSX</li><li>- Create-React-App</li><li>- Coleções de Componentes</li></ul> |

# WEBPACK

# PROJETO SEM WEBPACK

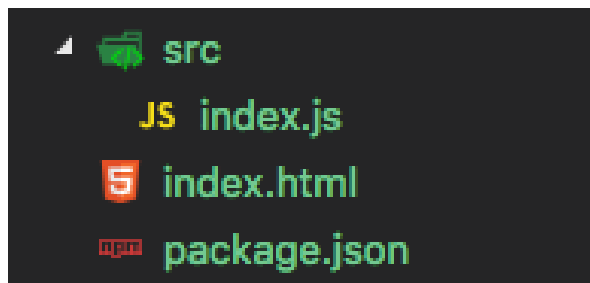
- Para entender o real papel do webpack, primeiro vamos criar uma estrutura tradicional de um projeto web.

```
$ mkdir meu-projeto-webpack
```

```
$ cd meu-projeto-webpack
```

```
$ npm init -y
```

- Em seguida crie a seguinte estrutura de arquivos:



# PROJETO SEM WEBPACK

- index.html

```
<html>

<head>
  <meta charset="utf-8" />
  <title>Introdução ao webpack</title>
  <script src="https://unpkg.com/lodash@4.16.6"></script>
</head>

<body>
  <script src="./src/index.js"></script>
</body>

</html>
```

# PROJETO SEM WEBPACK

- src/index.js

```
function component() {
  var element = document.createElement('div');
  // Lodash é utilizado como variável global por meio da
  // declaração do <script> no index.html
  element.innerHTML = _.join(['Olá', 'webpack', '!'], ' ');
  return element;
}

document.body.appendChild(component());
```

# PROJETO SEM WEBPACK

- Resultado:

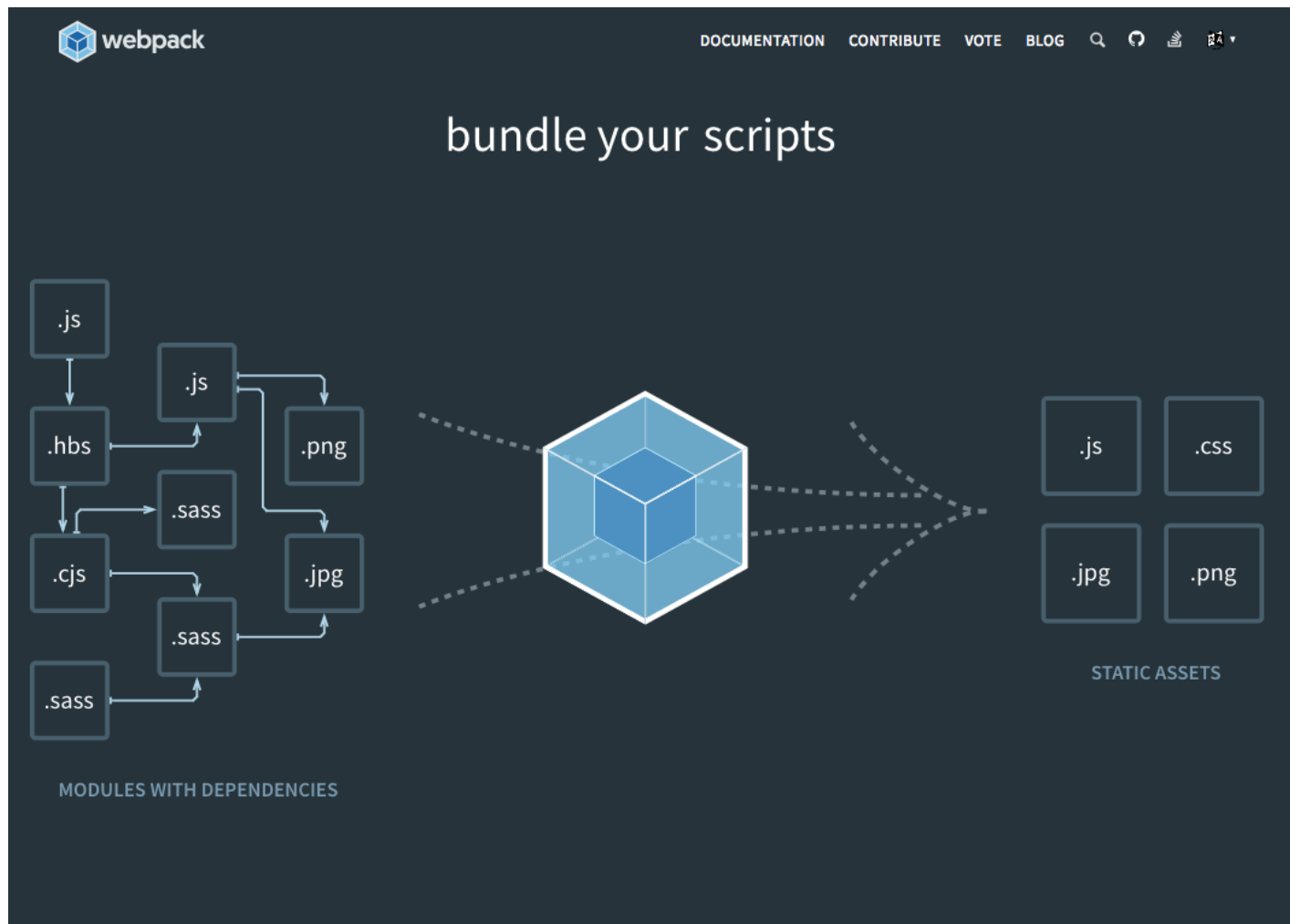


# INTRODUÇÃO AO WEBPACK

- webpack (com “w” minúsculo) é um empacotador de código para projetos web, assim como o browserify.
- O objetivo do webpack não é apenas unir todos os arquivos JS e CSS em um único pacote, mas também possibilitar a divisão do projeto em módulos reaproveitáveis e com isso auxiliando no trabalho em equipe.
- Entretanto, o webpack não é recomendado apenas para projetos grandes, tornando-se uma ferramenta muito útil também para projetos pequenos.



# INTRODUÇÃO AO WEBPACK



# INSTALANDO O WEBPACK

- Essencialmente o `webpack` deve ser instalado como uma dependência de desenvolvimento do seu projeto.

```
$ npm install --save-dev webpack webpack-cli
```

- Em alguns casos também pode ser interessante instalar o `webpack` globalmente, para facilitar a execução de scripts e testes rápidos.

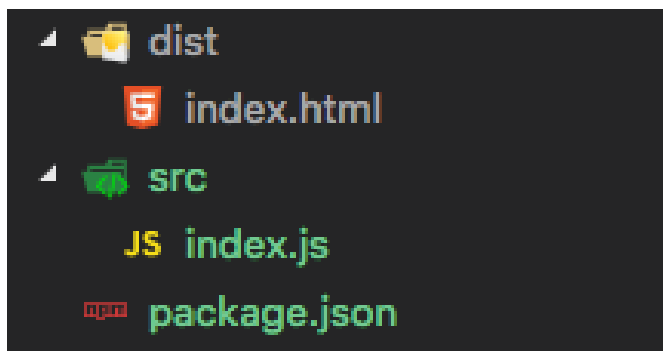
```
$ npm install --global webpack webpack-cli
```

# USO BÁSICO DO WEBPACK

- Para converter este projeto para utilizar o webpack, é preciso instalar as dependências necessárias:

```
$ npm install lodash
```

- Atualizar a estrutura do projeto para:



# USO BÁSICO DO WEBPACK

- Atualize o código do arquivo `dist/index.html`

```
<html>

<head>
  <meta charset="utf-8" />
  <title>Introdução ao webpack</title>
</head>

<body>
  <script src="main.js"></script>
</body>

</html>
```

# USO BÁSICO DO WEBPACK

- Atualize o código do "src/index.js"

```
import _ from 'lodash';

function component() {
  var element = document.createElement('div');

  // Lodash agora é importado do node_modules
  element.innerHTML = _.join(['Olá', 'webpack', '!'], ' ');

  return element;
}

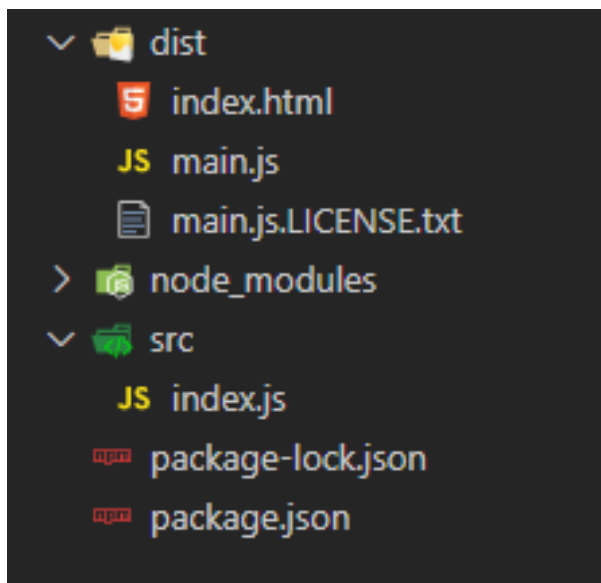
document.body.appendChild(component());
```

# USO BÁSICO DO WEBPACK

- Por fim, para empacotar o projeto execute o comando:

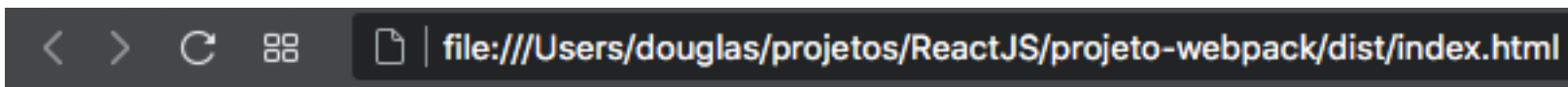
```
$ npx webpack --mode production
```

- Resultado:



# USO BÁSICO DO WEBPACK

- Resultado:



Olá webpack !

# CONFIGURAÇÃO DO WEBPACK

- A maioria dos projetos pode precisar de uma configuração mais complexa, para isso o `webpack` suporta a criação de arquivos de configuração.
- Para utilizar o arquivo de configuração, basta criar um arquivo chamado `webpack.config.js` na raiz do projeto e acrescentar o seguinte conteúdo.

```
const path = require('path');

module.exports = {
  entry: './src/index.js',
  mode: 'production',
  output: {
    filename: 'main.js',
    path: path.resolve(__dirname, 'dist')
  }
};
```

```
$ npx webpack --config webpack.config.js
```

<https://webpack.js.org/guides/getting-started/#using-a-configuration>



# WEBPACK LOADERS

# LOADERS

- Loaders são auxiliares que permitem que o webpack saiba trabalhar com outros tipos de arquivos além do JavaScript. Uma vez que o loader é configurado, você está dizendo ao webpack o que deve ser feito quando um arquivo daquele formato for encontrado.
- Por exemplo, para possibilitar que o webpack saiba carregar arquivos do tipo CSS, é preciso instalar e configurar os seguintes módulos:

```
$ npm install --save-dev style-loader css-loader
```

# LOADERS

## ■ webpack.config.js

```
const path = require('path');

module.exports = {
  // ...
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          'style-loader',
          'css-loader'
        ]
      }
    ]
  }
};
```



# LOADERS

- Uma vez que o loader de CSS foi configurado, é possível importar os arquivos de estilo utilizando o `import`.

src/index.js

```
import _ from 'lodash';
import './styles.css'; // adiciona importação do estilo

function component() {
  var element = document.createElement('div');
  element.innerHTML = _.join(['Olá', 'webpack', '!'], ' ');

  // adiciona a classe CSS para aplicar o estilo
  element.classList.add('hello');

  return element;
}

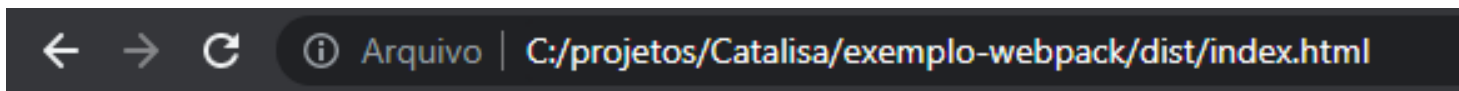
document.body.appendChild(component());
```

src/styles.css

```
.hello {
  color: red;
}
```

# LOADERS

- Resultado:



Olá webpack !

# LOADERS

- Loaders também podem ser utilizados para carregar imagens.

```
$ npm install --save-dev file-loader
```

webpack.config.js

```
const path = require('path');

module.exports = {
  // ...
  module: {
    rules: [
      // ... ,
      {
        test: /\. (png|svg|jpg|gif) $/,
        use: ['file-loader']
      }
    ]
  }
};
```

# LOADERS

- Uma vez que o arquivo de imagem for importado, será retornada uma `String` contendo o caminho para o arquivo.

src/index.js

```
// ...
import Logo from './logo.png';

function component() {
    // ...

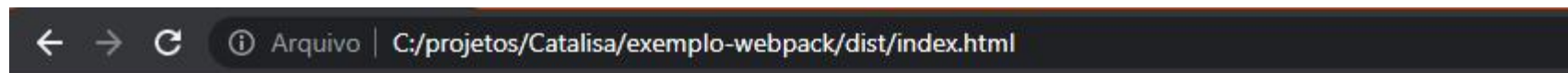
    var logoImg = new Image();
    logoImg.src = Logo;
    logoImg.width = 300;
    element.appendChild(logoImg);

    return element;
}

document.body.appendChild(component());
```

# LOADERS

- Resultado:



Olá webpack !





# WEBPACK PLUGINS

# PLUGINS

- Outro recurso interessante do `webpack` é a possibilidade de adição de `plugins`. Isso permite que você possa adicionar funcionalidades extras que o `webpack` não atende por si só.
- Por exemplo, podemos adicionar o `html-webpack-plugin` para gerenciar a criação do `index.html` à partir de um template.

```
$ npm install --save-dev html-webpack-plugin
```

# PLUGINS

- Configurando o `html-webpack-plugin`:

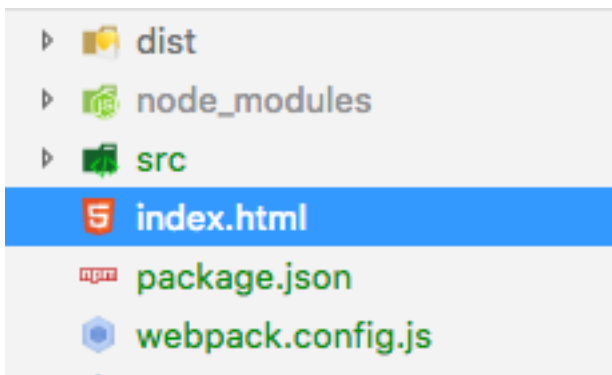
`webpack.config.js`

```
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  // ...,
  plugins: [
    new HtmlWebpackPlugin({
      template: './index.html'
    })
  ],
};
```

# PLUGINS

- Então, o arquivo de template `index.html` agora deve ficar na raiz do projeto, e conter o seguinte código.



```
<!DOCTYPE html>
<html>

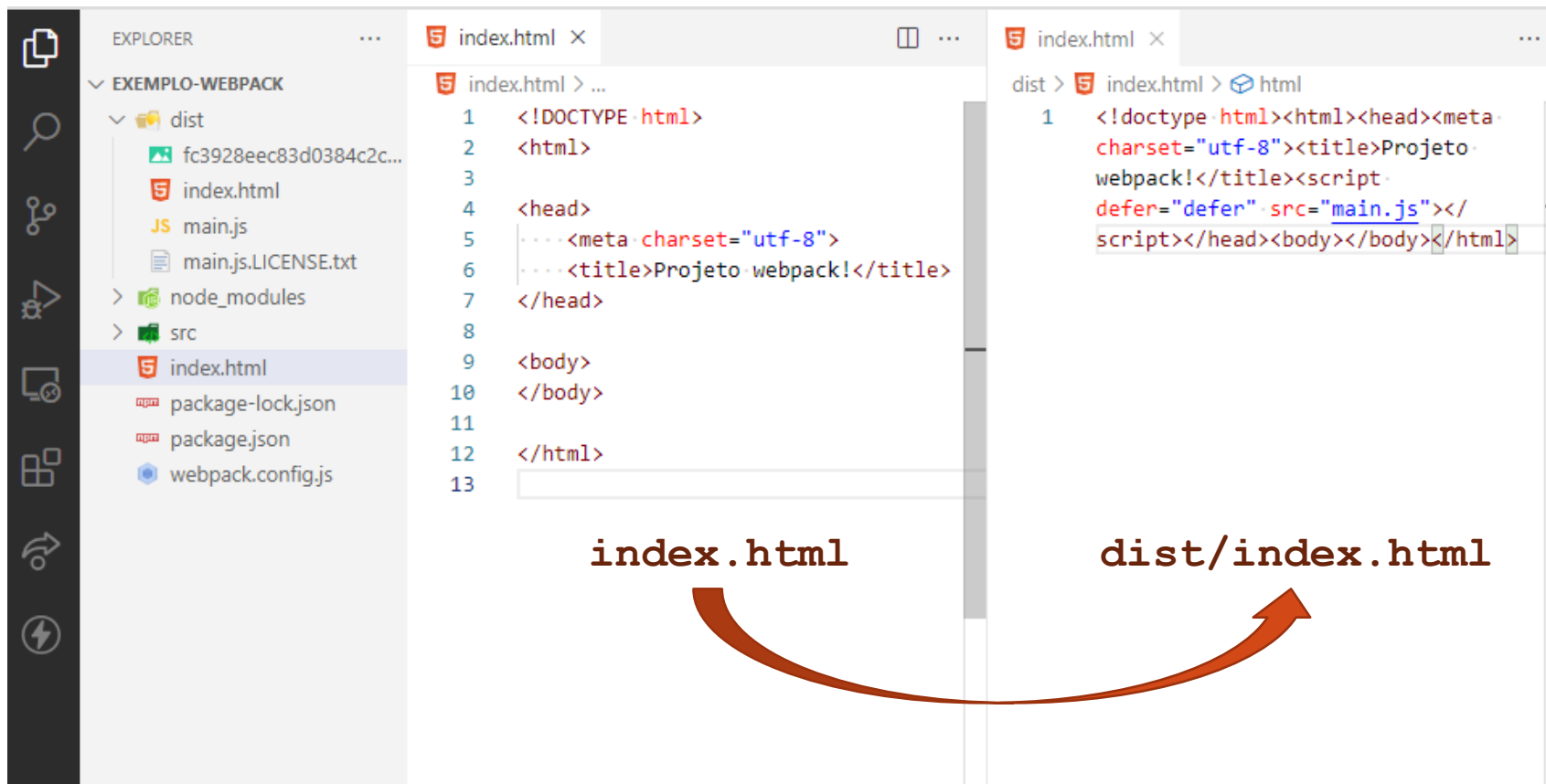
<head>
  <meta charset="utf-8">
  <title>Projeto webpack!</title>
</head>

<body>
</body>

</html>
```

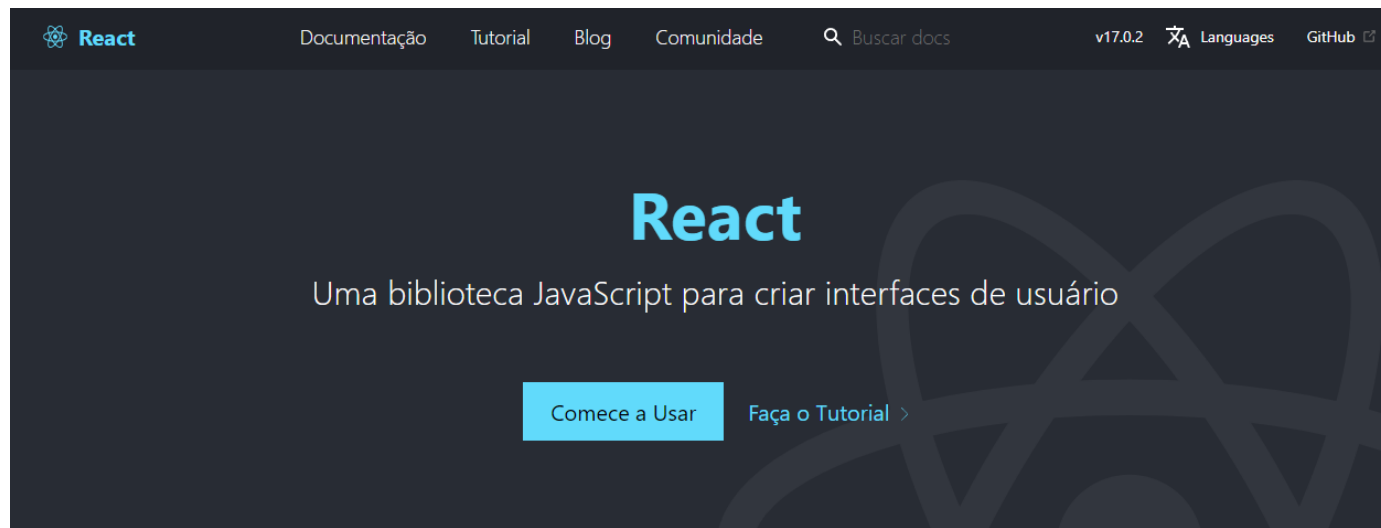
# PLUGINS

- Após compilar o projeto novamente, um novo `index.html` será gerado automaticamente dentro da pasta `dist/`:



# REACT-DOM

# INTRODUÇÃO AO REACT-DOM



## Declarativo

React faz com que a criação de UIs interativas seja uma tarefa fácil. Crie views simples para cada estado na sua aplicação, e o React irá atualizar e renderizar de forma eficiente apenas os componentes necessários na medida em que os dados mudam.

Views declarativas fazem com que seu código seja mais previsível e simples de depurar.

## Baseado em componentes

Crie componentes encapsulados que gerenciam seu próprio estado e então, combine-os para criar UIs complexas.

Como a lógica do componente é escrita em JavaScript e não em templates, você pode facilmente passar diversos tipos de dados ao longo da sua aplicação e ainda manter o estado fora do DOM.

## Aprenda uma vez, use em qualquer lugar

Não fazemos suposições sobre as outras tecnologias da sua stack, assim você pode desenvolver novos recursos com React sem reescrever o código existente.

O React também pode ser renderizado no servidor, usando Node, e ser usado para criar aplicações mobile, através do [React Native](#).

<http://reactjs.org>

# INTRODUÇÃO AO REACT-DOM

## ■ Declarativo

- React facilita a criação de UIs interativas. Crie *views* simples para cada estado em seu aplicativo e o React irá atualizar e renderizar eficientemente apenas os componentes certos quando seus dados forem alterados.
- Views declarativas tornam seu código mais previsível e mais fácil de depurar.

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



# INTRODUÇÃO AO REACT-DOM

- **Baseado em Componentes**
- Crie componentes encapsulados que gerenciem seu próprio estado, e então, use-os para compor UIs complexas.
- Uma vez que a lógica dos componentes está escrita em JavaScript, você pode facilmente passar dados através do seu aplicativo e manter o estado fora do DOM.

```
class Timer extends React.Component {  
  tick() {  
    this.setState((prevState) => ({  
      seconds: prevState.seconds + 1  
    }));  
  }  
  componentDidMount() {  
    setInterval(() => this.tick(), 1000);  
  }  
  render() {  
    return (  
      <div>  
        Seconds: {this.state.seconds}  
      </div>  
    );  
  }  
}
```

# INTRODUÇÃO AO REACT-DOM

- **Aprenda uma vez, use em qualquer lugar**
- Não exige que você altere o conjunto de tecnologias utilizados em sua stack, evitando reescrever o código de sua aplicação.
- React também pode renderizar no servidor usando **Node** ou alimentar aplicativos móveis usando o **React Native**.
- React-DOM (Web), React-Native (Android, iOS, Windows), React XP (Mobile, Desktop, Web), React-Ink (Terminal)

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

class WhyReactNativeIsSoGreat extends Component {
  render() {
    return (
      <View>
        <Text>
          Se você gosta do React na web, você
          vai gostar do React Native.
        </Text>
        <Text>
          Você apenas usa componentes nativos
          como 'View' e 'Text', em vez de um
          componentes web como 'div' e 'span'.
        </Text>
      </View>
    );
  }
}
```

# INTRODUÇÃO AO REACT-DOM

- Para utilizar o ReactJS na web, você precisa apenas importar as bibliotecas React e React-DOM no `<head>` de sua página.

```
<script crossorigin src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
<script crossorigin src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js"></script>
```

- E então você já pode renderizar seu primeiro componente.

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

  <script>
    const myDiv = React.createElement('div', null, 'Olá React!');
    ReactDOM.render(myDiv, document.getElementById('root'));
  </script>
</body>
```

# INTRODUÇÃO AO REACT-DOM

- Código completo:

```
<!DOCTYPE html>

<head>
  <meta charset="UTF-8">
  <title>My React Website</title>

  <script crossorigin src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js"></script>
</head>

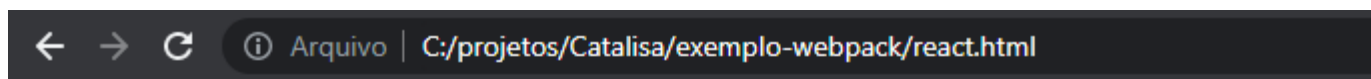
<body>
  <div id="root"></div>

  <script>
    const myDiv = React.createElement('div', null, 'Olá React!');
    ReactDOM.render(myDiv, document.getElementById('root'));
  </script>
</body>

</html>
```

# INTRODUÇÃO AO REACT-DOM

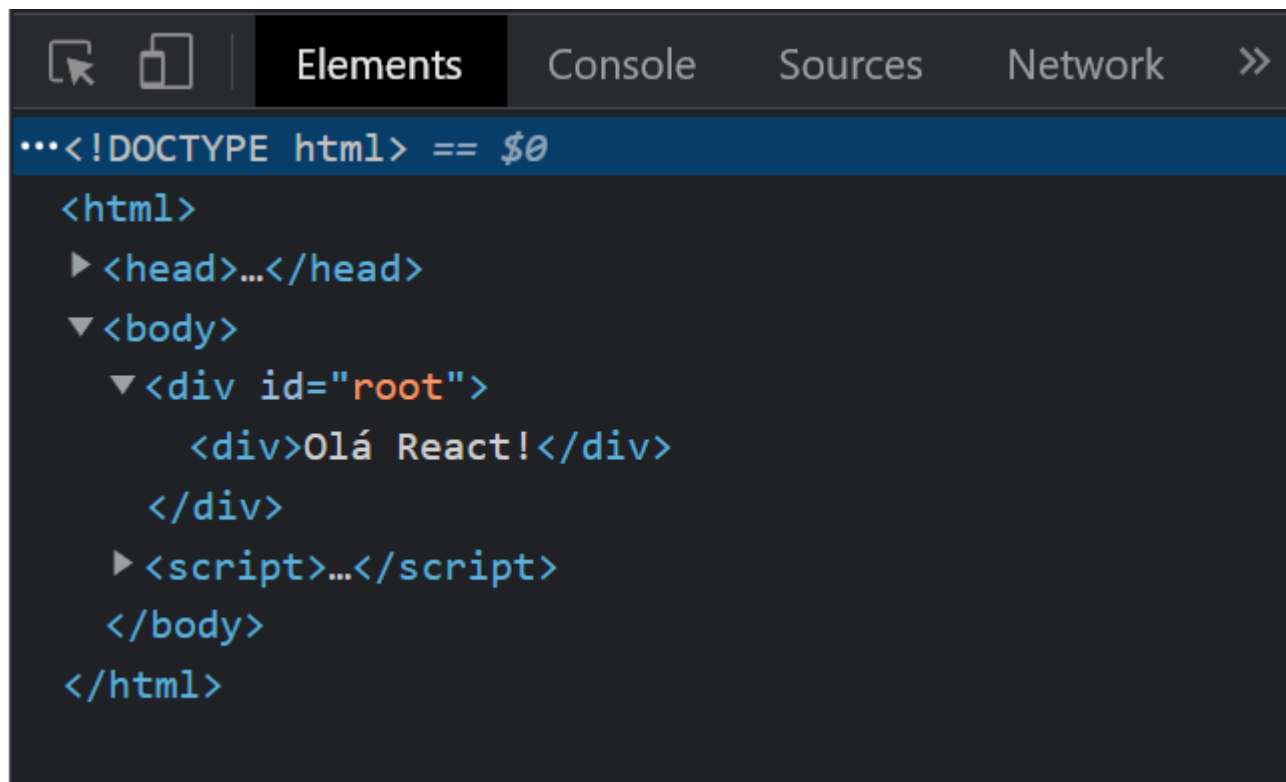
- Resultado:



Olá React!

# INTRODUÇÃO AO REACT-DOM

- Código renderizado:



```

...<!DOCTYPE html> == $0
<html>
  ><head>...</head>
  ▼<body>
    ▼<div id="root">
      <div>Olá React!</div>
    </div>
    ><script>...</script>
  </body>
</html>
  
```

# COMPONENTES

# TUDO SÃO COMPONENTES

- Os componentes permitem que você divida a interface (UI) em partes independentes, reutilizáveis e pensar em cada uma isoladamente.
- Conceitualmente, os componentes são como funções JavaScript. Eles aceitam entradas arbitrárias (chamados de “props ou propriedades”) e retornam elementos descrevendo o que deve aparecer na tela.



LabelComponent  
ItemComponent

NETFLIX

Navegar ▾

Kids

Buscar

9+

Douglas ▾

Populares na Netflix

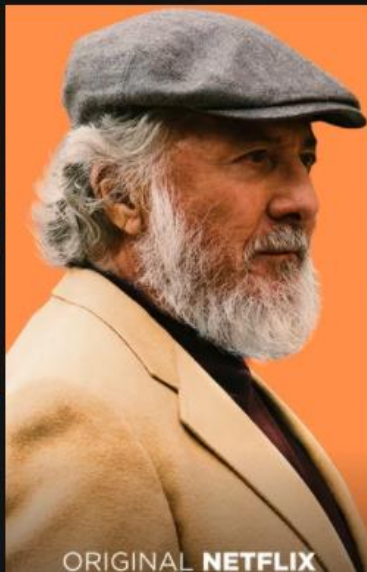
SectionComponent



Em alta



ORIGINAIS NETFLIX



# COMPONENTES FUNCIONAIS

# COMPONENTES FUNCIONAIS

- A maneira mais simples de se criar um Componente em ReactJS, é declarando uma função JavaScript.

```
function WelcomeComponent(props) {
  return React.createElement('h1', null, 'Hello, ' + props.name);
}
```

- Esta função é um componente válido pois ela recebe um parâmetro único chamado “props” e retorna um elemento React.
- É chamado de componente “funcional” pois, literalmente, é uma função JavaScript.

# COMPONENTES FUNCIONAIS

## ■ Código completo:

```
<script>
  function WelcomeComponent(props) {
    return React.createElement('h1', null, 'Hello, ' + props.name);
  }

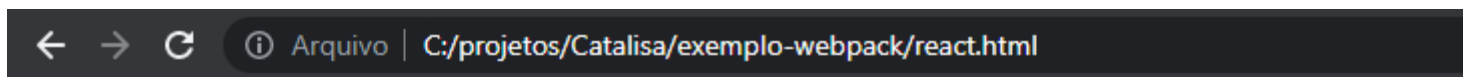
  const myComponent = React.createElement(
    WelcomeComponent,
    {
      name: 'Douglas'
    },
    null
  );

  const myDiv = React.createElement('div', null, 'Olá React!', myComponent);

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

# COMPONENTES FUNCIONAIS

- Resultado:



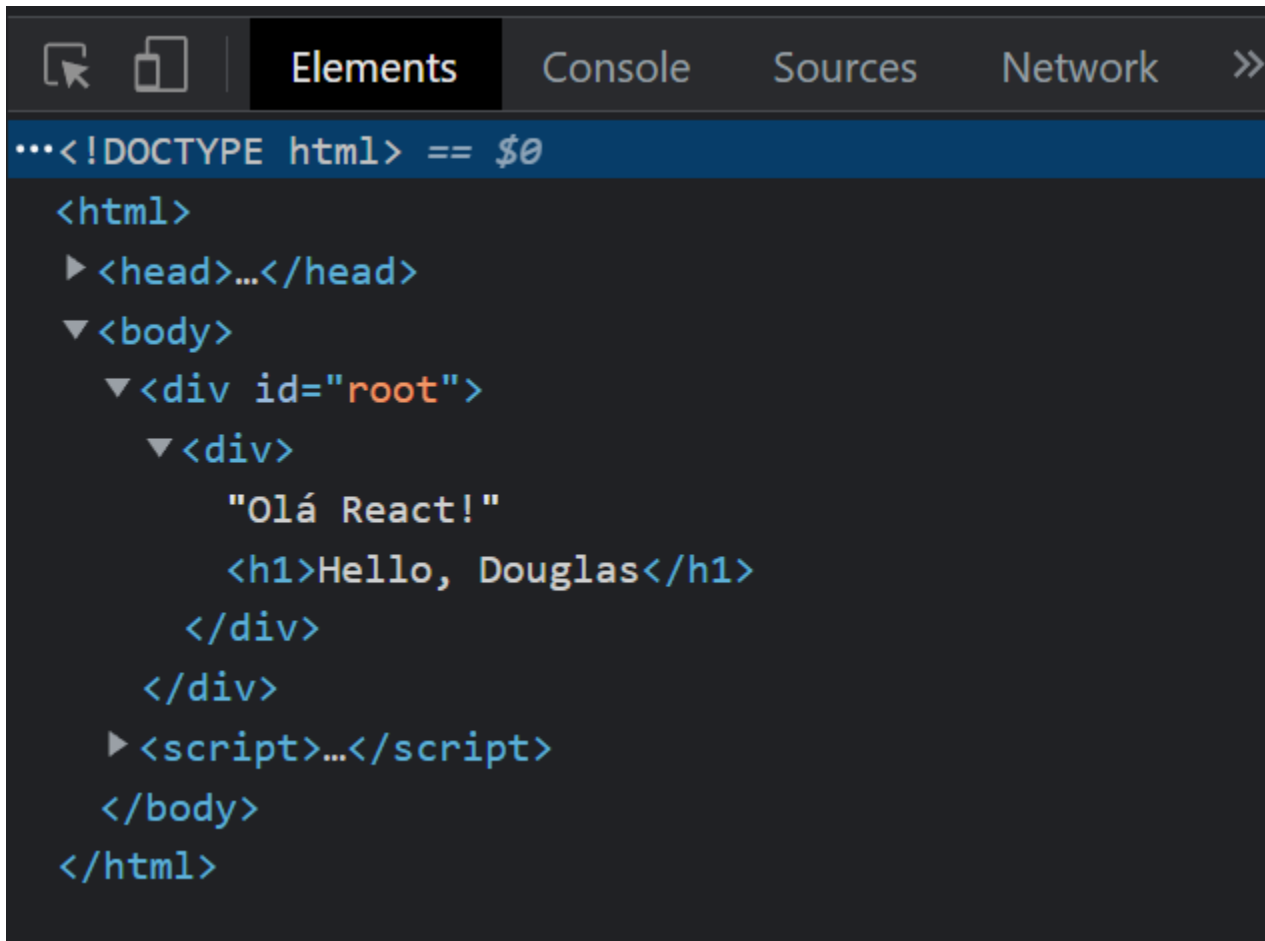
Olá React!

Hello, Douglas



# COMPONENTES FUNCIONAIS

- Código renderizado:



The screenshot shows the 'Elements' tab of a browser's developer tools. The DOM tree is expanded, showing the following structure:

```

<!DOCTYPE html> == $0
<html>
  <head>...</head>
  <body>
    <div id="root">
      <div>
        "Olá React!"
        <h1>Hello, Douglas</h1>
      </div>
    </div>
    <script>...</script>
  </body>
</html>
  
```

# COMPONENTES DE CLASSES

# COMPONENTES DE CLASSES

- Os componentes de classe também recebem valores através de “`props`” e podem renderizar um ou mais elementos React.
- Adicionalmente, **os componentes de classe são capazes de gerenciar seu próprio estado.**
- Se usado **sem webpack**:
  - À partir da versão 16, função de criação de classes foi movida para um pacote separado, então precisamos importar:

```
<script crossorigin src="https://unpkg.com/create-react-class@15.7.0/create-react-class.js"></script>
```



# COMPONENTES DE CLASSES

- Os componentes de classe podem ser declarados assim:

```
const CounterComponent = createReactClass({
  getInitialState: function () {
    return {
      count: 0,
    };
  },
  componentDidMount: function () {
    const self = this;
    setInterval(function () {
      self.setState({ count: self.state.count + 1 })
    }, 1000);
  },
  render: function () {
    return React.createElement('p', null, this.state.count);
  }
});
```

# COMPONENTES DE CLASSES

## ■ Código completo:

```
<script>
  function WelcomeComponent(props) {
    return React.createElement('h1', null, 'Hello, ' + props.name);
  }

  const myComponent = React.createElement(
    WelcomeComponent,
    {
      name: 'Douglas'
    },
    null
  );

  const CounterComponent = createReactClass({
    getInitialState: function () {
      return {
        count: 0,
      };
    },
    componentDidMount: function () {
      const self = this;
      setInterval(function () {
        self.setState({ count: self.state.count + 1 })
      }, 1000);
    },
    render: function () {
      return React.createElement('p', null, this.state.count);
    }
  });

  const myCounter = React.createElement(
    CounterComponent,
    null,
    null
  );

  const myDiv = React.createElement('div', null, 'Olá React!', myComponent, myCounter);

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

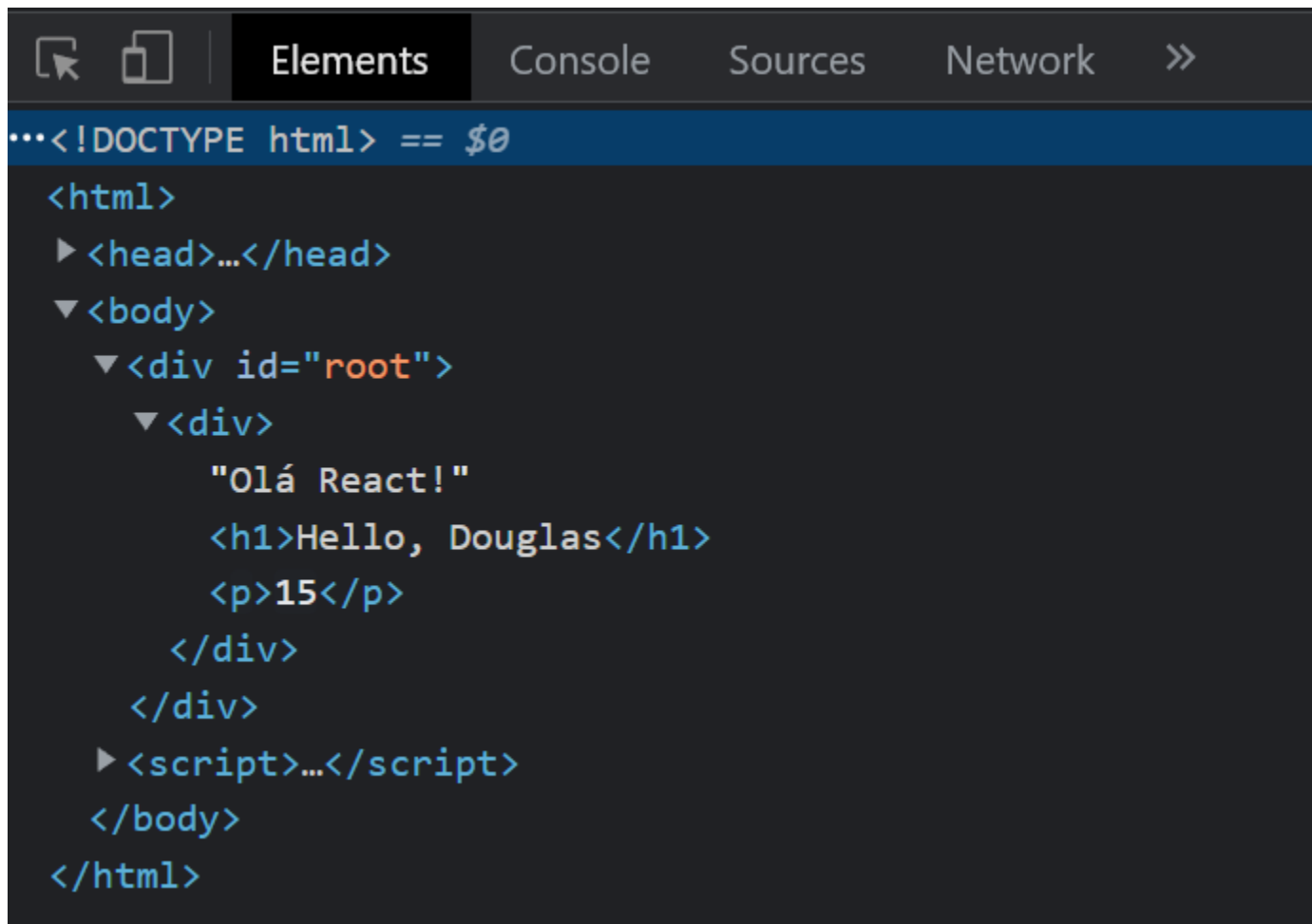
# COMPONENTES DE CLASSES

- Resultado:



# COMPONENTES DE CLASSES

- Código renderizado:



```

    ...<!DOCTYPE html> == $0
    <html>
      <head>...</head>
      <body>
        <div id="root">
          <div>
            "Olá React!"
            <h1>Hello, Douglas</h1>
            <p>15</p>
          </div>
        </div>
        <script>...</script>
      </body>
    </html>
  
```

# REACT HOOKS

# HOOKS

- A partir do **React 16.8**, os **componentes funcionais** podem trabalhar com `hooks` para gerenciar estado, criar referências, receber eventos, dentre outras coisas.
- Até o **React 16.8**, esses recursos existiam apenas nos componentes de classe.
- **OPINIÃO:** *Ainda não existe nada oficial, mas tudo indica que os componentes de classe serão depreciados e deixarão de existir no futuro.*

# HOOKS

- Exemplo de uso do useState.

```
// ...

const StepComponent = () => {
  const [step, setStep] = React.useState(0);
  const incrementStep = () => {
    setStep(s => s += 1);
  };
  const decrementStep = () => {
    setStep(s => s -= 1);
  };
  return React.createElement('div', null,
    React.createElement('button', { onClick: incrementStep }, '+'),
    React.createElement('span', null, step),
    React.createElement('button', { onClick: decrementStep }, '-'),
  );
};

const myStep = React.createElement(
  StepComponent,
  null,
  null
);

// ...
```

# HOOKS

## ▪ Resultado





# HOOKS

- Código renderizado:

```
...<html> == $0
  ▶ <head>...</head>
  ▼ <body>
    ▼ <div id="root">
      ▼ <div>
        "Olá React!"
        <h1>Hello, Douglas</h1>
        <p>83</p>
      ▼ <div>
        <button>+</button>
        <span>5</span>
        <button>-</button>
      </div>
    </div>
  </div>
  ▶ <script>...</script>
</body>
</html>
```

# ECMAScript 2015 e JSX

# ECMAScript 2015 E JSX

- Visando aproveitar todo o poder das versões mais recentes do JavaScript, o React dispõe de um *plugin* para o **Babel** que permite o uso do ECMAScript 2015 (ES6) e JSX.

```
// ECMAScript 2009 (ES 5) sem JSX
function WelcomeComponent(props) {
  return React.createElement('h1', null, 'Hello, ' + props.name);
}

// ECMAScript 2015 (ES 6) com JSX
const WelcomeComponent = (props) => {
  return <h1>Hello, {props.name}</h1>
}
```

- Para isso, é recomendado o uso de ferramentas como “webpack” ou “Browserify” para auxiliar na “transpilação” e empacotamento do código.
- Felizmente, existe uma ferramenta chamada “create-react-app” que faz todo o trabalho necessário com um único comando.

# CREATE REACT APP

# CREATE-REACT-APP

- Ferramenta de linha de comando para auxiliar na criação e configuração de um projeto “webpack” com ReactJS.
- Adicionalmente, `create-react-app` traz um conjunto de configurações e módulos auxiliares que facilitam o processo de desenvolvimento e implantação.
- Para instalar o `create-react-app` utilize:  

```
$ npm install --global create-react-app
```

# CREATE-REACT-APP

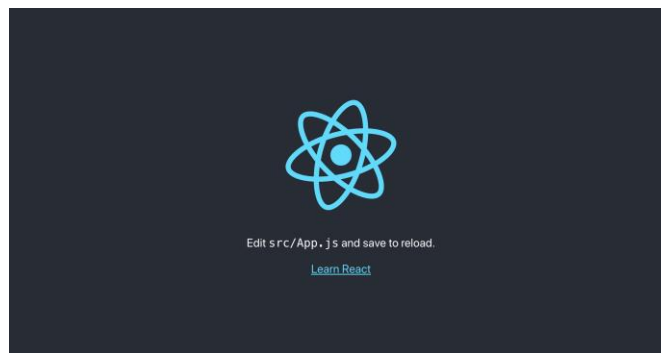
- Para criar um novo projeto, execute o comando:

```
$ create-react-app meu-projeto-reactjs
```

- Entre na pasta do projeto e execute e inicie o `webpack-dev-server` em modo de desenvolvimento:

```
$ cd meu-projeto-reactjs
```

```
$ npm start
```



# ESTADO E CICLO DE VIDA

# ESTADO E CICLO DE VIDA

- Componentes funcionais vs Componentes de classe

```
function Titulo(props) {
  return (
    <h1>
      {props.children}
    </h1>
  );
}
```

```
<Titulo>
  Olá mundo!
</Titulo>
```

```
class Contador extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      valor: props.valorInicial
    };
  }
  render() {
    return (
      <div>
        Contando: {this.state.valor}
      </div>
    );
  }
}
```

```
<Contador valorInicial={100} />
```



# ESTADO E CICLO DE VIDA

- Estado é semelhante às propriedades, porém ele é privado (visível apenas dentro do componente) e totalmente controlado pelo componente de classe ou *hook*.
- Para entender a diferença, vamos criar um componente Relógio que recebe o tempo via “`props`” e em seguida alterá-lo para controlar seu próprio estado.

# ESTADO E CICLO DE VIDA

- Recebendo data e hora via “props”.

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

function Relogio(props) {
  return (
    <div>
      <h1>Olá React!</h1>
      <h2>Hora certa: {props.date.toLocaleTimeString()}</h2>
    </div>
  );
}
```

```
<Relogio date={new Date()} />
```

# ESTADO E CICLO DE VIDA

- Controlando seu próprio estado.

```
import React, {
  useEffect, useState
} from 'react';

const Relogio = () => {
  const [date, setDate] = useState(new Date());

  useEffect(() => {
    const contar = () => {
      setDate(new Date());
    }
    setInterval(contar, 1000);
  }, []);

  // continua
```

```
    return (
      <div>
        <h1>Olá React!</h1>
        <h2>
          Hora certa: {date.toLocaleTimeString()}.
        </h2>
      </div>
    );
  }

  export default Relogio;
```

# ESTADO E CICLO DE VIDA

- Componentes de classe também possuem eventos que podem ser utilizados para detectar momentos do ciclo de vida dos componentes.
- **Montagem:** Eventos disparados quando o componente é criado e anexado ao DOM.

| Componente de classe  | Componente funcionar com Hooks  |
|---|---|
| <ul style="list-style-type: none"> <li>• <code>constructor()</code></li> <li>• <del><code>componentWillMount()</code></del></li> <li>• <code>render()</code></li> <li>• <code>componentDidMount()</code></li> </ul> | <ul style="list-style-type: none"> <li>• <code>useState()</code></li> <li>• <code>useEffect()</code></li> <li>• <code>useLayoutEffect()</code></li> </ul> |

# ESTADO E CICLO DE VIDA

- **Atualização:** Uma atualização pode ocorrer por uma mudança nas `props` ou no estado. Estes eventos são disparados quando o componente é re-renderizado.

| Componente de classe   | Componente funcionar com Hooks   |
|--|--|
| <ul style="list-style-type: none"> <li>• <del><code>componentWillReceiveProps()</code></del></li> <li>• <code>shouldComponentUpdate()</code></li> <li>• <del><code>componentWillUpdate()</code></del></li> <li>• <code>render()</code></li> <li>• <code>componentDidUpdate()</code></li> </ul> | <ul style="list-style-type: none"> <li>• <code>useEffect()</code></li> <li>• <code>useLayoutEffect()</code></li> </ul> |

- **Desmontagem:** Evento chamado quando o componente será destruído.

| Componente de classe  | Componente funcionar com Hooks   |
|---|--|
| <ul style="list-style-type: none"> <li>• <code>componentWillUnmount()</code></li> </ul> | <ul style="list-style-type: none"> <li>• <code>useEffect()</code></li> <li>• <code>useLayoutEffect()</code></li> </ul> |

# COLEÇÕES DE COMPONENTES

# COLEÇÕES DE COMPONENTES

- Com a popularização do React JS, é comum encontrar bibliotecas, frameworks e conjuntos de componentes já prontos para o uso com React.
- Quatro exemplos que devemos citar são:
  - ReactStrap (Bootstrap 4)
  - Ant Design (Alibaba/Aliexpress)
  - Office UI Fabric (Microsoft)
  - Material UI (implementa o Material Design)
  - Blueprint
  - Atlas Kit (Atlasian)

# COLEÇÕES DE COMPONENTES

- Para instalar o **AntDesign**, basta executar:

```
$ npm install antd
```

- E então, importar o CSS globalmente no `index.js`:

```
import 'antd/dist/antd.css';
```



# COLEÇÕES DE COMPONENTES

Código:

```
import { Alert } from 'antd';

// ...

<div className="App">
  <Alert message="Success Text" type="success" />
  <Alert message="Info Text" type="info" />
  <Alert message="Warning Text" type="warning" />
  <Alert message="Error Text" type="error" />
</div>
```

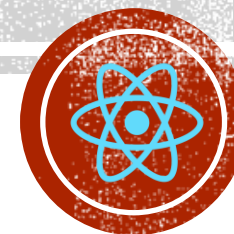
Resultado:

|              |
|--------------|
| Success Text |
| Info Text    |
| Warning Text |
| Error Text   |

# REFERÊNCIAS

- webpack - <https://webpack.js.org/>
- webpack guides - <https://webpack.js.org/guides/>
- Lodash - <https://lodash.com>
- React JS - <http://reactjs.org>
- JSX - <https://reactjs.org/docs/introducing-jsx.html>
- Documentação React - <https://reactjs.org/docs/getting-started.html>
- Create-react-app - <https://create-react-app.dev/>
- Podcast - <https://hipsters.tech/react-o-framework-onipresente-hipsters-66/>

# OBRIGADO!



**Douglas Nassif Roma Junior**

 /douglasjunior

 /in/douglasjunior

 smarppy.com

 douglas@smarppy.com