

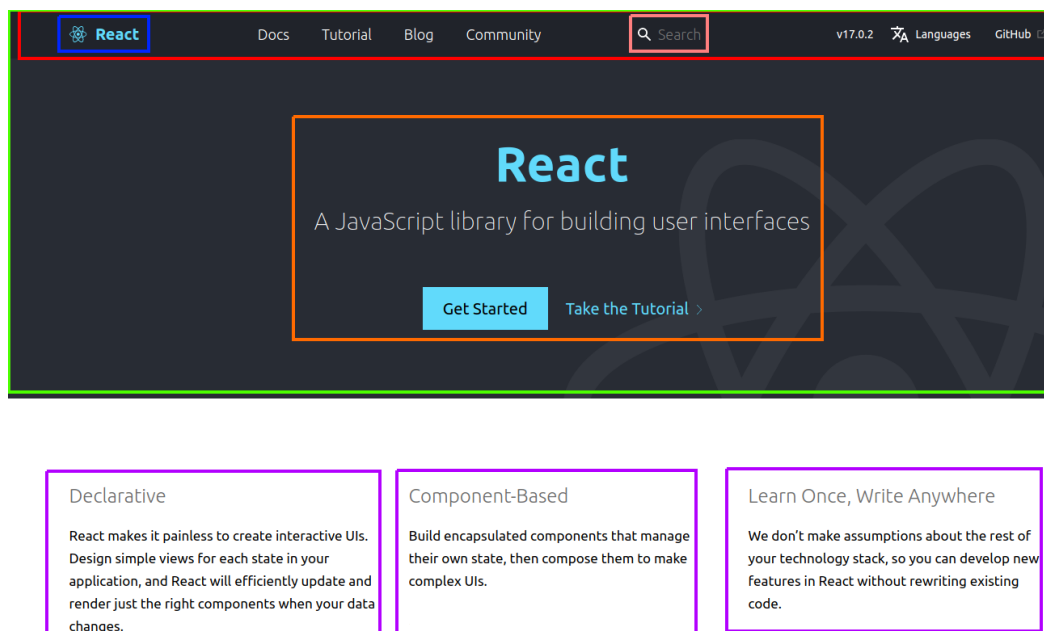
ReactJS

Componentes - JSX

1 Introdução

ReactJS é uma biblioteca Javascript de código aberto, própria para a construção de interfaces gráficas e componentes visuais. Criada em 2013 pelo Facebook, hoje é mantida também por um grupo de empresas e desenvolvedores independentes. O **cerne** da biblioteca ReactJS é caracterizado pela **criação de componentes**. Entre outras coisas, cada componente tem como responsabilidade **“explicar” como se dá a exibição visual de partes de uma interface gráfica mais complexa**. Cada componente gerencia seu próprio **estado** e eles podem se comunicar por meio de um mecanismo conhecido como **props**. Componentes lidam, também, com eventos gerados pelo usuário. Ao construir uma interface gráfica, cabe ao desenvolvedor detectar quais partes merecem ser definidas por componentes independentes. Uma vez definidos os componentes, eles são combinados a fim de obter-se a interface gráfica completa. Componentes ReactJS são usados por meio de um mecanismo semelhante a simples **tags HTML**. A Figura 1.1 mostra a página oficial do ReactJS e destaca **uma** forma como ela poderia ser definida usando componentes.

Figura 1.1



Na Figura 1.1, cada retângulo representa um possível componente ReactJS previamente definido. O código ReactJS que dá origem a essa página pode ser parecido com aquele exibido no Bloco de Código 1.1. Cada componente é utilizado por meio de algo semelhante a uma tag HTML. Seu uso implica a chamada de uma função que tem como finalidade produzir o código HTML que define o seu aspecto visual.

Bloco de Código 1.1

```
<PaginaPrincipal>
  <Toolbar>
    <Logo />
    <ul>
      <li>Docs</li>
      <li>Tutorial</li>
      <li>Blog</li>
      <li>Community</li>
    </ul>
    <Busca />
    <ul>
      <li>v17.0.2</li>
      <li>Languages</li>
      <li>GitHub</li>
    </ul>
  </Toolbar>
  <DefinicaoPrincipal>
    <p>React</p>
    <p>A Javascript library for building user interfaces</p>
    <button>Get Started</button>
    <button>Take the Tutorial</button>
  </DefinicaoPrincipal>
</PaginaPrincipal>
```

2 Desenvolvimento

Uma aplicação ReactJS pode ser criada de diferentes formas. Visite o Link 2.1 para conhecer as principais.

Link 2.1

<https://reactjs.org/docs/create-a-new-react-app.html>

Neste material estudaremos sobre a criação de aplicações React, sobre a sua estrutura - conjunto de arquivos e diretórios que caracterizam uma aplicação - e sobre a sua execução.

2.1 (Seu workspace) É recomendável organizar-se separando seus projetos em diretórios apropriados para eles. Em geral, é comum a criação de um diretório com a finalidade de abrigá-los. É comum chamá-lo de algo como **workspace**, embora esse nome não seja obrigatório. Procure criá-lo em um diretório que

- não tenha espaços em branco e/ou caracteres especiais
- seja subdiretório do diretório "home" do usuário do sistema operacional, a fim de evitar problemas com permissões controladas pelo próprio sistema operacional. No Linux, um exemplo de workspace razoável é **/home/usuario/workspace**. No Windows, um bom exemplo é **C:\Users\usuario\Documents\workspace**.

Nota. No Windows, certifique-se de que você não está usando um diretório do **OneDrive**. Geralmente isso acontece quando se clica em Documents na barra lateral "Quick Launch" que fica à esquerda do Windows Explorer.

Vá em frente e crie o seu workspace. A seguir, abra um terminal e use

cd path/do/seu/workspace

para navegar até ele.

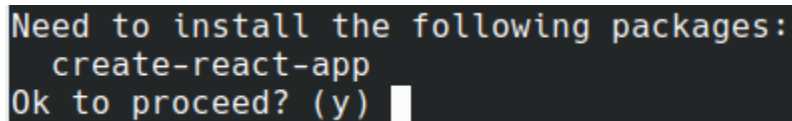
2.2 (Criando uma aplicação ReactJS) Use o comando

`npx create-react-app meu-primeiro-app-react`

para criar uma aplicação.

Caso seja a primeira execução do create-react-app usando o npx e a versão do npm seja 7+, a mensagem da Figura 2.2.1 deverá ser exibida. Basta confirmar para prosseguir.

Figura 2.2.1



```
Need to install the following packages:  
  create-react-app  
Ok to proceed? (y) █
```

Nota. A ferramenta npx faz o download do pacote executado - neste caso, o create-react-app - e o mantém instalado em sua memória "cache" - um simples diretório em seu sistema de arquivos. O diretório utilizado depende de onde o NodeJS está instalado. No Linux com NodeJS instalado usando o nvm, ele pode ser encontrado em um diretório cujo path é parecido com `/home/rodrigo/.npm/_npx/`. Os arquivos ali existentes não fazem parte de sua aplicação React. São arquivos referentes ao pacote instalado apenas.

Depois de criar o projeto, use

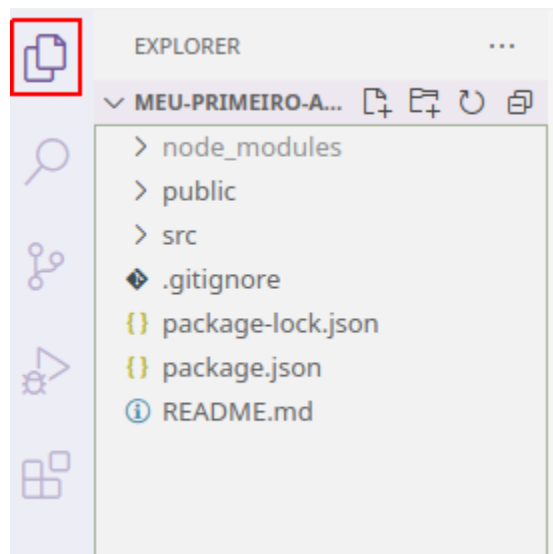
`cd meu-primeiro-app-react`

para navegar até o diretório recém-criado. Abra uma instância do VS Code vinculada ao diretório atual com

code .

2.3 (Diretórios e arquivos do projeto) Um projeto ReactJS é caracterizado por uma coleção de arquivos e diretórios. O Explorer do VS Code deve mostrar a estrutura exibida pela Figura 2.3.1.

Figura 23.1



- A pasta **node_modules** abriga todas as dependências do projeto. Quando executamos o comando para criar o projeto, ele fez o download de muitos pacotes, os quais são armazenados nesta pasta. Os arquivos da biblioteca ReactJS, por exemplo, podem ser encontrado ali.
- A pasta **public** abriga arquivos estáticos, como: arquivos .html cujo conteúdo não se altera, figuras, arquivos de áudio etc.

- A pasta **src** abriga o código-fonte de sua aplicação. Observe que, no momento, ela possui alguns arquivos Javascript e CSS. Trabalharemos muito nesta pasta.
- O arquivo **.gitignore** é próprio do sistema de controle de versão Git. Ali especificamos os arquivos e diretórios que desejamos excluir do controle de versão.
- As dependências armazenadas na pasta **node_modules** são descritas textualmente no arquivo **package.json**. Assim, o controle de versão deste projeto inclui somente uma descrição de suas dependências. As dependências propriamente ditas nunca têm sua versão controlada. Um desenvolvedor que obtiver uma cópia deste projeto terá acesso somente a este arquivo e fará ele mesmo o download das dependências.
- O arquivo **package-lock.json** foi introduzido com a versão 5 do npm e tem como finalidade descrever a versão exata de cada pacote usado no projeto. Assim, mesmo que algum pacote seja atualizado no futuro pelos seus mantenedores, o projeto pode ainda ser reconstruído utilizando as versões originalmente utilizadas.
- O arquivo **README.md** (md vem de **markdown**) possui informações sobre o projeto que podem ser de interesse para outros desenvolvedores.

2.4 (Colocando o projeto em execução) Para colocar a aplicação em funcionamento, basta usar

npm start

Esse comando indica que desejamos executar aquilo que está associado ao nome "start", descrito no arquivo **package.json**. Abra o arquivo novamente e encontre a seção "scripts" para entender melhor. Veja, também, a Figura 2.4.1.

Figura 2.4.1



```
{ } package.json x

{ } package.json > { } scripts > start

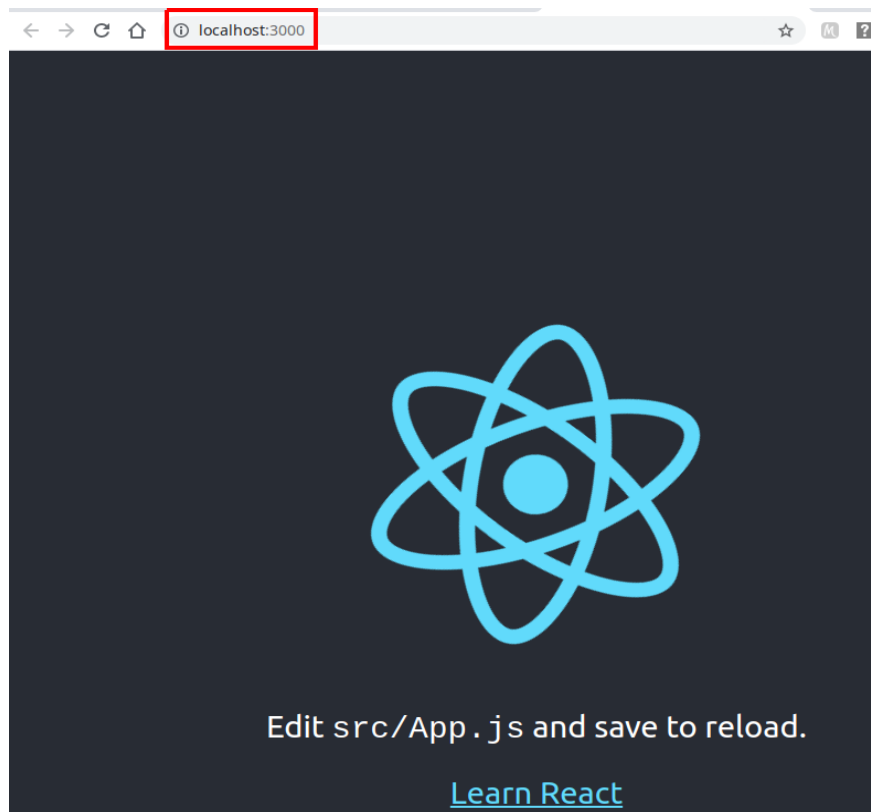
1  {
2    "name": "meu-primeiro-app-react",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^5.14.1",
7      "@testing-library/react": "^11.2.7",
8      "@testing-library/user-event": "^12.8.3",
9      "react": "^17.0.2",
10     "react-dom": "^17.0.2",
11     "react-scripts": "4.0.3",
12     "web-vitals": "^1.1.2"
13   },
14   "scripts": {
15     "start": "react-scripts start",
16     "build": "react-scripts build",
17     "test": "react-scripts test",
18     "eject": "react-scripts eject"
19   },
```

Isso quer dizer que estamos tentando executar o pacote **react-scripts**. Ele se encontra na pasta **node_modules/.bin**. Perceba o ponto que precede a palavra bin. Trata-se de um diretório oculto.

Nota. Pode ser conveniente utilizar o terminal “embutido” no VS Code. Para isso, no VS Code, clique Terminal >> New Terminal.

Um servidor web simples, utilizado somente para testes e desenvolvimento, é colocado em funcionamento. A partir daí, a aplicação pode ser acessada utilizando um navegador comum no endereço <http://localhost:3000>. Uma instância do navegador padrão da máquina deve ter sido colocada em funcionamento automaticamente. O navegador deve exibir algo parecido com aquilo que a Figura 2.4.2 ilustra.

Figura 2.4.2

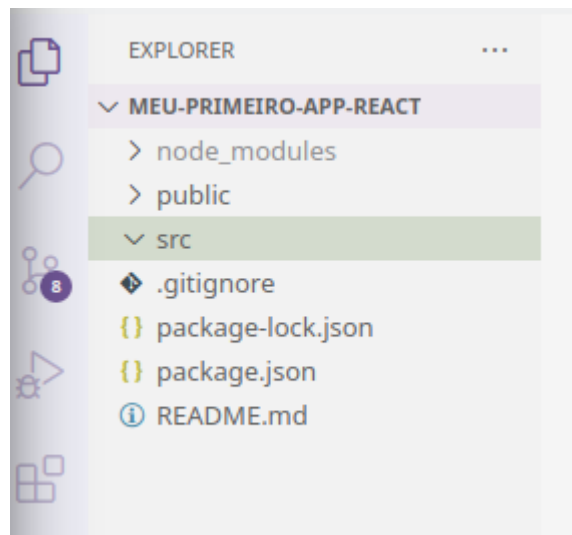


Caso deseje interromper a execução do servidor, basta apertar **CTRL+C** no terminal. Para executar novamente, use

npm start

2.5 (Experimentos iniciais - Definindo um componente ReactJS) O projeto criado possui bastante código gerado automaticamente que não nos é de interesse no momento. Por isso, vamos **apagar todos os arquivos existentes na pasta src**. Assim teremos a oportunidade de começar o projeto do zero. A pasta src deve ficar totalmente vazia, como na Figura 2.5.1.

Figura 2.5.1



- Na pasta **src**, crie um arquivo chamado **index.js**. Para isso, basta clicar sobre ela e escolher **New File**.
- No navegador, visite novamente o endereço **localhost:3000**. Perceba que a página está agora vazia.

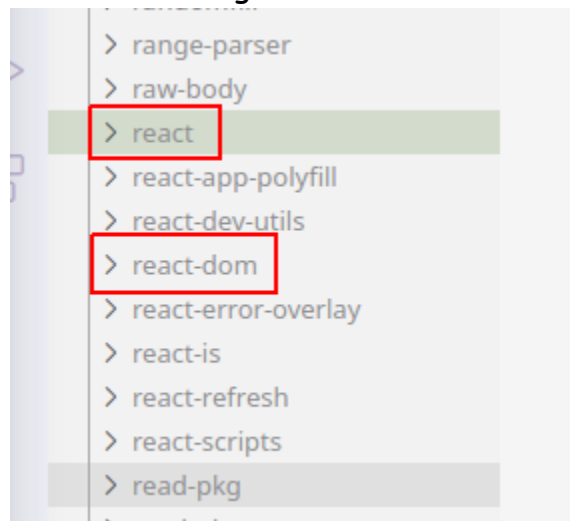
- O uso do React requer a importação das bibliotecas **React** e **ReactDOM**. Isso pode ser feito como ilustra o Bloco de Código 2.5.1.

Bloco de Código 2.5.1

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

- Os nomes '**react**' e '**react-dom**' são os nomes dos diretórios em que se encontram as respectivas bibliotecas. Eles podem ser encontrados na pasta **node_modules**, como a Figura 2.5.2 ilustra.

Figura 2.5.2



- Por outro lado, **React** e **ReactDOM** são **nomes de variáveis** que darão acesso aos conteúdos das respectivas bibliotecas. Embora por razões óbvias seja uma convenção utilizar esses nomes, nada impede que sejam utilizados nomes quaisquer, como ilustra o Bloco de Código 2.5.2.

Bloco de Código 2.5.2

```
import x from 'react'  
import y from 'react-dom'
```

Esse é apenas um detalhe técnico que vale destacar. Iremos, entretanto, **manter as convenções**, prosseguindo com o uso dos nomes **React** e **ReactDOM**.

- Um componente ReactJS pode ser definido de duas formas diferentes: utilizando uma **função** ou uma **classe**. Em ambos os casos, ele deve **produzir código HTML que explica como se dá a sua exibição**. Além disso, ele pode definir **funções responsáveis pela manipulação de eventos gerados pelo usuário**.

- Embora seja perfeitamente viável utilizar código HTML "puro" para fazer a descrição visual dos componentes, é muito comum o uso de **JSX**, que vem de "Javascript XML". Trata-se de uma extensão sintática à linguagem Javascript que, em geral, simplifica essa tarefa. A documentação oficial do ReactJS sugere o seu uso. Visite o Link 2.5.1 para saber mais.

Link 2.5.1

<https://reactjs.org/docs/introducing-jsx.html>

- O Bloco de Código 2.5.3 mostra a definição de um componente ReactJS feita por meio de uma função. A expressão JSX que ela devolve está destacada.

Bloco de Código 2.5.3

```
const App = () => {  
  return <div>Meu primeiro componente ReactJS</div>  
}
```

- Para utilizar o componente, devemos especificar um contêiner HTML (um **div**, por exemplo) cujo conteúdo será substituído pelo HTML produzido pelo componente. A aplicação que criamos já possui um desses no arquivo **public/index.html**. Ele é o primeiro filho do elemento **body**. Abra o arquivo e verifique a sua existência.
- No arquivo **src/index.js**, utilizamos o componente como uma simples tag. Veja o Bloco de Código 2.5.4.

Nota. O método **querySelector** é capaz de buscar referências na árvore DOM utilizando qualquer seletor CSS. O método **getElementById**, por sua vez, faz buscas utilizando somente o id de um elemento. Ambos resolvem o problema, neste caso.

Bloco de Código 2.5.4

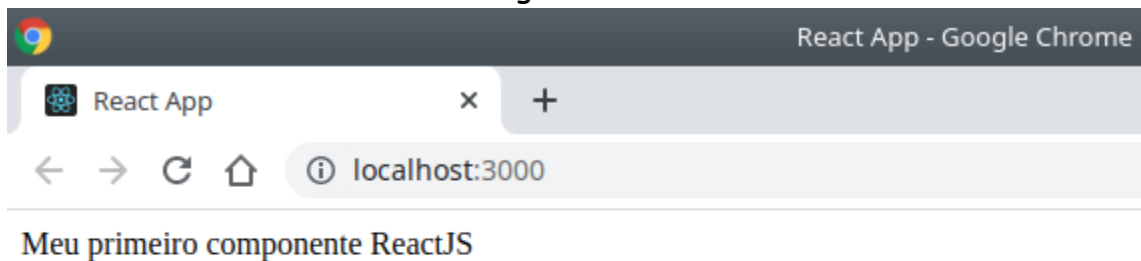
```
import React from 'react'
import ReactDOM from 'react-dom'

const App = () => {
  return <div>Meu primeiro componente ReactJS</div>
}

ReactDOM.render(
  <App />,
  document.querySelector("#root")
)
```

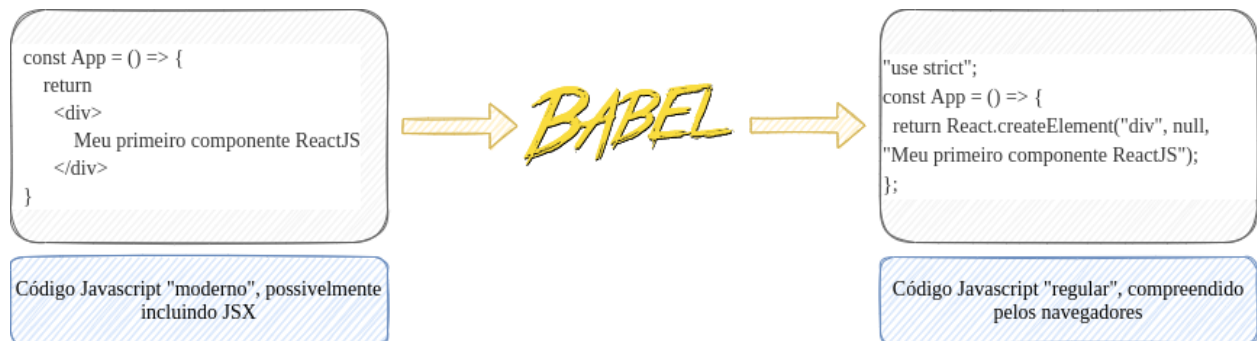
- No seu navegador, visite **localhost:3000** e verifique o resultado. Ele deve ser parecido com aquele exibido pela Figura 2.5.3.

Figura 2.5.3



2.6 (JSX? Por quê?) Quando utilizamos uma expressão JSX, essencialmente estamos criando um ou mais elementos HTML e adicionando-os em algum momento na árvore DOM. Embora isso possa ser feito por meio de chamadas de função comuns, seu uso tende a ser bastante trabalhoso. JSX é uma espécie de **syntax sugar** que visa simplificar essa tarefa. Para entender melhor, é importante saber que estamos utilizando um compilador Javascript chamado **Babel**. Ele recebe código Javascript moderno, potencialmente incluindo expressões JSX, e traduz para código Javascript comum, que os navegadores têm condições de entender. Veja a Figura 2.6.1.

Figura 2.6.1

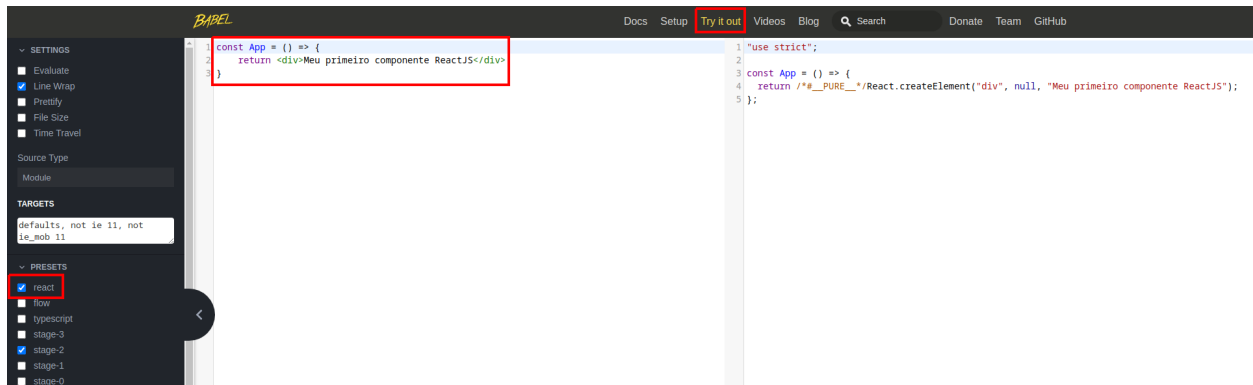


- Caso deseje, você pode realizar testes visitando o Link 2.6.1.

Link 2.6.1
<https://babeljs.io/>

Clique **Try it out** e cole o seu código Javascript que contém JSX, como mostra a Figura 2.6.2.

Figura 2.6.2



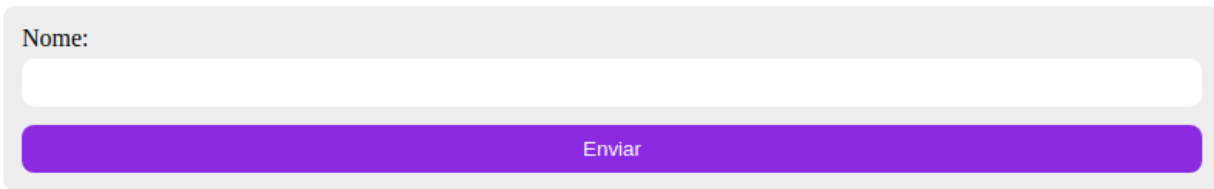
2.7 (HTML versus JSX) Nesta seção vamos desenvolver um pequeno exemplo com HTML "puro" para então verificar o seu equivalente em JSX. O primeiro passo é **criar um arquivo de extensão HTML fora do projeto ReactJS** em que estamos trabalhando no momento. Basta criar um arquivo textual em qualquer pasta do seu sistema de arquivos e alterar a sua extensão para **html**. Abra-o utilizando um editor de texto qualquer - possivelmente o próprio VS Code - e utilize o conteúdo do Bloco de Código 2.7.1.

Bloco de Código 2.7.1

```
<div style="margin: auto; width: 768px; background-color: #EEE; padding: 12px;  
border-radius: 8px">  
  <label for="nome" style="display: block; margin-bottom: 4px">Nome:</label>  
  <input type="text" id="nome" style="padding-top: 8px; padding-bottom: 8px;  
border-style: hidden; width: 100%; border-radius: 8px; outline: none; box-sizing:  
border-box"/>  
  <button style="margin-top: 12px; padding-top: 8px; padding-bottom: 8px;  
background-color: blueviolet; color: white; border: none; width: 100%; border-  
radius: 8px">Enviar</button>  
</div>
```

O resultado esperado é exibido pela Figura 2.7.1.

Figura 2.7.1



A screenshot of a web form. It features a light gray background. At the top, the text 'Nome:' is displayed in a small, dark font. Below it is a white rectangular text input field with rounded corners. Underneath the input field is a solid purple button with the word 'Enviar' written in white text, centered on the button.

O Bloco de Código 2.7.2 mostra o equivalente em JSX.

Bloco de Código 2.7.2

```
const App = () => {  
  return(  
    <div style={{margin: 'auto', width: 768, backgroundColor: '#EEE', padding:  
12, borderRadius: 8}}>  
      <label htmlFor="nome" style={{display: 'block', marginBottom:  
4}}>Nome:</label>  
      <input type="text" id="nome" style={{paddingTop: 8, paddingBottom: 8,  
borderStyle: 'hidden',width: '100%', borderRadius: 8, outline: 'none', boxSizing:  
'border-box'}}/>  
      <button style={{marginTop: 12, paddingTop: 8, paddingBottom: 8,  
backgroundColor: 'blueviolet', color: 'white', border: 'none', width: '100%',  
borderRadius: 8}}>Enviar</button>  
    </div>  
  )  
}
```

Cabe destacar os seguintes pontos.

- Em JSX, **style** é um **prop** (mais sobre isso adiante) ao qual atribuímos uma expressão Javascript. Usamos { } para indicar que o que vem a seguir é uma expressão Javascript. Quando usamos { } pela segunda vez, estamos especificando um objeto JSON.

- As propriedades CSS que são escritas em **kebab case** passam a ser escritas em **camel case**. Exemplo: **background-color** em CSS se escreve **backgroundColor** em JSX.
- **for** é uma palavra reservada da linguagem Javascript. Por isso, usamos **htmlFor** como atributo do elemento label.
- A documentação oficial explica o seguinte sobre unidades de medida:

"React will automatically append a "px" suffix to certain numeric inline style properties. If you want to use units other than "px", specify the value as a string with the desired unit."

Veja mais sobre isso no Link 2.7.1.

Link 2.7.1

<https://reactjs.org/docs/dom-elements.html>

- As constantes como **white**, **none** e **auto** devem ser expressas entre aspas simples ou duplas.
- Até então, fizemos uso somente de CSS **inline**. Obviamente, também é possível utilizar CSS definido em arquivos à parte. Para fazer esse teste, crie um arquivo chamado **styles.css** - o nome é você quem escolhe - no diretório **src**. Seu conteúdo é dado no Bloco de Código 2.7.3.

Bloco de Código 2.7.3

```
.rotulo{
  font-size: 18px;
  color: blueviolet;
}
```

No arquivo **index.js**, faça o ajuste destacado no Bloco de Código 2.7.4.

Bloco de Código 2.7.4

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

const App = () => {
  return(
    <div style={{margin: 'auto', width: 768, backgroundColor: '#EEE', padding:
12, borderRadius: 8}}>
      <label className="rotulo" htmlFor="nome" style={{display: 'block',
marginBottom: 4}}>Nome:</label>
      <input type="text" id="nome" style={{paddingTop: 8, paddingBottom: 8,
borderStyle: 'hidden', width: '100%', borderRadius: 8, outline: 'none', boxSizing:
'border-box'}}/>
      <button style={{marginTop: 12, padding: 8, backgroundColor: 'blueviolet', color: 'white', border: 'none', width: "100%",
borderRadius: 8}}>Enviar</button>
    </div>
  )
}

ReactDOM.render(
  <App />,
  document.querySelector("#root")
)
```

Nota. Utilizamos **className** pelo fato de **class** ser uma palavra reservada da linguagem Javascript.

- Os objetos JSON que agrupam configurações CSS podem, também, ser definidos pela função que define o componente ReactJS. Isso ocorre pois podemos

referenciá-los a partir de contextos JSX. Veja o Bloco de Código 2.7.5. Apenas removemos o objeto JSON associado ao botão e o associamos a uma constante. A seguir, ele é referenciado por meio do nome da constante.

Bloco de Código 2.7.5

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

const App = () => {
  const estilosBotao = {marginTop: 12, paddingTop: 8, paddingBottom: 8,
    backgroundColor: 'blueviolet', color: 'white', border: 'none', width: "100%",
    borderRadius: 8};
  return(
    <div style={{margin: 'auto', width: 768, backgroundColor: '#EEE', padding:
12, borderRadius: 8}}>
      <label className="rotulo" htmlFor="nome" style={{display: 'block',
marginBottom: 4}}>Nome:</label>
      <input type="text" id="nome" style={{paddingTop: 8, paddingBottom: 8,
borderStyle: 'hidden',width: '100%', borderRadius: 8, outline: 'none', boxSizing:
'border-box'}}/>
      <button style={estilosBotao}>Enviar</button>
    </div>
  )
}

ReactDOM.render(
  <App />,
  document.querySelector("#root")
)
```

- Isso também é verdade para texto simples. Veja o Bloco de Código 2.7.6.

Bloco de Código 2.7.6

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

const App = () => {
  const estilosBotao = {marginTop: 12, paddingTop: 8, paddingBottom: 8,
    backgroundColor: 'blueviolet', color: 'white', border: 'none', width: "100%",
    borderRadius: 8};
  const textoDoRotulo = "Nome:";
  return(
    <div style={{margin: 'auto', width: 768, backgroundColor: '#EEE', padding:
    12, borderRadius: 8}}>
      <label className="rotulo" htmlFor="nome" style={{display: 'block',
    marginBottom: 4}}>{textoDoRotulo}</label>
      <input type="text" id="nome" style={{paddingTop: 8, paddingBottom: 8,
    borderStyle: 'hidden', width: '100%', borderRadius: 8, outline: 'none', boxSizing:
    'border-box'}}/>
      <button style={estilosBotao}>Enviar</button>
    </div>
  )
}

ReactDOM.render(
  <App />,
  document.querySelector("#root")
)
```

- Também é possível chamar funções neste contexto, como ilustra o Bloco de Código 2.7.7.

Bloco de Código 2.7.7

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

const App = () => {
  const estilosBotao = {marginTop: 12, paddingTop: 8, paddingBottom: 8,
    backgroundColor: 'blueviolet', color: 'white', border: 'none', width: "100%",
    borderRadius: 8};
  const textoDoRotulo = "Nome:";
  const obterTextoDoBotao = () => {
    return "Enviar";
  }
  return(
    <div style={{margin: 'auto', width: 768, backgroundColor: '#EEE', padding:
12, borderRadius: 8}}>
      <label className="rotulo" htmlFor="nome" style={{display: 'block',
marginBottom: 4}}>{textoDoRotulo}</label>
      <input type="text" id="nome" style={{paddingTop: 8, paddingBottom: 8,
borderStyle: 'hidden', width: '100%', borderRadius: 8, outline: 'none', boxSizing:
'border-box'}}/>
      <button style={estilosBotao}>{obterTextoDoBotao()}</button>
    </div>
  )
}

ReactDOM.render(
  <App />,
  document.querySelector("#root")
)
```

Referências

React - A JavaScript library for building user interfaces. 2021. Disponível em <<https://reactjs.org/>>. Acesso em agosto de 2021.