

RESPONSIVE WEB DEVELOPMENT

# REACTJS

## LISTA DE FIGURAS

Figura 7.1 – Painel de criação de publicação do Facebook .....	5
Figura 7.2 – Diagrama de componentes para "CreatePublicationPanel" .....	6
Figura 7.3 – Comportamento de um componente TextField no Facebook.....	6
Figura 7.4 – Instalador do NodeJS.....	8
Figura 7.5 – Prompt de comando do windows (terminal) .....	9
Figura 7.6 – Tela inicial do Visual Studio Code .....	9
Figura 7.7 – ES7 React plugins.....	11
Figura 7.8 – Primeira aplicação React .....	13
Figura 7.9 – Resultado da execução dos comandos para criação de uma aplicação React.....	13
Figura 7.10 – Ciclo de vida de um componente .....	25
Figura 7.11 – Esboço de componentes para a aplicação proposta.....	35
Figura 7.12 – Visão principal da aplicação de lista de compras.....	35

## LISTA DE CÓDIGO-FONTE

Código-fonte 7.1 – Exemplo de código-fonte HTML.....	12
Código-fonte 7.2 – index.js gerado por create-react-app. ....	14
Código-fonte 7.3 – App.js gerado por create-react-app. ....	16
Código-fonte 7.4 – Arquivo Teste.js. ....	17
Código-fonte 7.5 – Arquivo index.js.....	17
Código-fonte 7.6 – Enviando uma propriedade a um componente. ....	19
Código-fonte 7.7 – Recebendo propriedades em um componente. ....	20
Código-fonte 7.8 – Enviando um objeto como uma propriedade a um componente. ....	21
Código-fonte 7.9 – Recebendo um objeto em um componente. ....	22
Código-fonte 7.10 – Alterando o componente App para mostrar a hora atual. ....	23
Código-fonte 7.11 – Duas funções criadas de formas diferentes.....	24
Código-fonte 7.12 – Declaração do evento clique com JavaScript e HTML.....	26
Código-fonte 7.13 – Declaração do evento clique no React com JSX.....	26
Código-fonte 7.14 – Declaração do evento clique no React com JSX.....	27
Código-fonte 7.15 – Declaração do evento clique no React com JSX.....	28
Código-fonte 7.16 – Alterando o valor de uma tag JSX por meio de uma Ref. ....	29
Código-fonte 7.17 – Uso de variáveis com conteúdo JSX. ....	29
Código-fonte 7.18 – Uso de variáveis com conteúdo JSX. ....	30
Código-fonte 7.19 – Uso de variáveis com conteúdo JSX. ....	31
Código-fonte 7.20 – Arquivo MeuComponente.js.....	32
Código-fonte 7.21 – Importando seu componente no arquivo App.js.....	33
Código-fonte 7.22 – Arquivo MeuComponente.js.....	33
Código-fonte 7.23 – Importando seu componente no arquivo App.js.....	34
Código-fonte 7.24 – Arquivo App.js.....	37
Código-fonte 7.25 – Arquivo Formulario.js. ....	38
Código-fonte 7.26 – Arquivo Produto.js.....	39

## SUMÁRIO

7 REACTJS .....	5
7.1 Introdução .....	5
7.2 Sobre o React .....	5
7.3 Pré-requisito deste capítulo .....	7
7.4 NodeJS: quando o JavaScript sai do browser .....	7
7.5 Visual Studio Code .....	9
7.5.1 Explorer .....	10
7.5.2 Terminal integrado .....	10
7.5.3 Controle de versionamento .....	10
7.5.4 Gerenciamento de extensões .....	10
7.6 Preparando o ambiente .....	11
7.7 Sua primeira aplicação React .....	12
7.8 Principais conceitos .....	14
7.9 JSX: a união entre HTML e JAVASCRIPT .....	15
7.10 Componentes .....	16
7.10.1 Importação e exportação de códigos .....	17
7.10.2 Codificação do componente .....	18
7.11 REACT: uma visão mais profunda .....	19
7.11.1 Props .....	19
7.11.2 Estados .....	22
7.11.3 Eventos .....	25
7.11.4 Refs .....	28
7.11.4 Técnicas comuns .....	29
7.11.4.1 Variáveis com tags HTML .....	29
7.11.4.2 Funções que retornam JSX .....	30
7.11.4.3 Uso de Listas e a função map .....	30
7.11.4.4 Criando eventos em seus próprios componentes .....	31
7.11.4.5 A propriedade especial <i>children</i> .....	33
7.12 Aplicando o React na prática: Lista de Compras .....	34
Conclusão e Próximos Passos .....	39
REFERÊNCIAS .....	41

## 7 REACTJS

### 7.1 Introdução

Neste capítulo, você aprenderá o que é o Framework ReactJS e como utilizá-lo de maneira eficiente. Começaremos estudando algumas ferramentas necessárias para sua utilização e automatização do processo de codificação. Em seguida, estudaremos o React a fundo!

Prepare-se para conhecer um dos frameworks mais exigidos do mercado!

### 7.2 Sobre o React

O React foi originalmente desenvolvido no Facebook em 2012. Naquele mesmo ano, outro framework já era consolidado como “padrão” por muitas indústrias: AngularJS. Esses frameworks nada mais são do que bibliotecas JavaScript capazes de criar componentes, além de gerenciar estados e eventos no browser (Front-End).

Para facilitar o entendimento desses frameworks, considere o painel de criação de publicação do Facebook, ilustrado na Figura “Painel de criação de publicação do Facebook”.



Figura 7.1 – Painel de criação de publicação do Facebook  
Fonte: Facebook (2019)

Esse painel pode ser facilmente codificado como um componente React. Podemos dar o nome “CreatePublicationPanel”, para ele ser utilizado em qualquer outra página em que seja necessário. Mas algo interessante acontece aqui. Dentro desse componente, temos uma caixa de texto e quatro botões com ícones. E se esses elementos já fossem componentes? Ótimo! Podemos codificar, então, novos componentes, “TextField” e “IconButton”, e utilizá-los em “CreatePublicationPanel”.

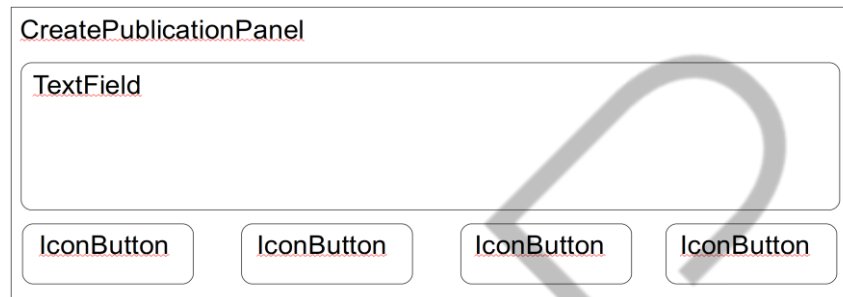


Figura 7.2 – Diagrama de componentes para "CreatePublicationPanel"  
Fonte: o autor (2019)

Você já reparou que os campos de texto do Facebook seguem o mesmo padrão? Certamente, trata-se de um mesmo componente “TextField”. Repare também que o **comportamento** desses campos de texto são os mesmos, ou seja, há uma explicação em cinza e, ao digitar algo, a explicação é escondida.



Figura 7.3 – Comportamento de um componente TextField no Facebook  
Fonte: Facebook

O React também é capaz de descrever esses comportamentos dos componentes. Assim, vimos aqui que tanto o padrão de exibição, descrito em HTML e CSS, como o comportamento, descrito em Javascript, podem ser unificados e relacionados a um componente.

Essas são só algumas vantagens em se utilizar o React. Outras vantagens estão relacionadas à interação desses componentes com API (Application Program Interface) de sistemas WEB e tratamento de estados desses componentes. Veremos isso mais tarde, neste capítulo.

### 7.3 Pré-requisito deste capítulo

O material a seguir assume a premissa de que o leitor tenha noções de lógica de programação ou desenvolvimento de algoritmos, além disso, que conheça tudo o que foi abordado sobre HTML, CSS e JavaScript em nossos materiais.

### 7.4 NodeJS: quando o JavaScript sai do browser

O JavaScript é muito conhecido por ser uma linguagem interpretada nos Browsers. Assim, um dos papéis dos browsers é entender esse código e reagir de forma adequada. O componente dos browsers que faz essa tarefa se chama “interpretador JavaScript”. Um dos interpretadores mais rápidos conhecidos atualmente é o V8, utilizado no Chrome.

O NodeJS é um programa que isola o V8 em um programa executável, tornando-o capaz de interpretar códigos JavaScript. Assim, o NodeJS é simplesmente um programa que entende JavaScript.

O NodeJS pode ser utilizado de diversas formas diferentes. A primeira, e mais frequente, é interpretar códigos no Back-End (servidor) para gerar páginas e mandá-las para o cliente. A segunda forma é utilizá-lo para executar códigos para aplicações Desktop. O VisualStudioCode é um bom exemplo disso, uma vez que ele é escrito em JavaScript e roda como um programa qualquer. A terceira forma é utilizá-lo como uma ferramenta para o desenvolvedor. Nesse caso, podemos utilizar códigos JavaScript para gerar outros códigos JavaScript. Um exemplo prático é o processo de “minificação”, que reescreve um arquivo js de forma a diminuir seu tamanho sem alterar a lógica de seu código. O React utiliza o NodeJS para gerar códigos que o Browser entende. Esse será o uso que faremos neste capítulo.

Junto à instalação do NodeJS, há outro programa, chamado npm (node package manager). O papel do npm é gerenciar bibliotecas de códigos JavaScript. Para quem conhece Python, o npm é similar ao pip. Com npm é possível fazer downloads automaticamente de bases de dados de código públicas para usar em qualquer projeto, o React está cadastrado no npm.

O NodeJS pode ser baixado em <https://nodejs.org/en/>. Na escrita deste capítulo, a versão atual do Node LTS (Long Term Support) é 10.15.3. É recomendável checar sua versão para que ela seja compatível com esse documento. Caso não tenha instalado, baixe o instalador e siga as instruções apresentadas na tela.

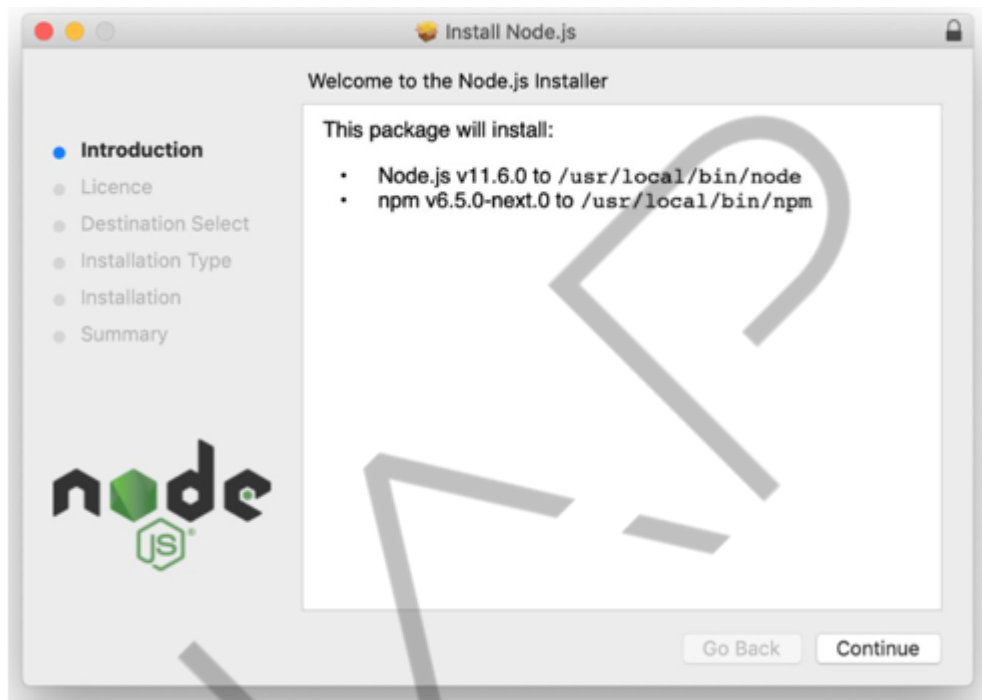

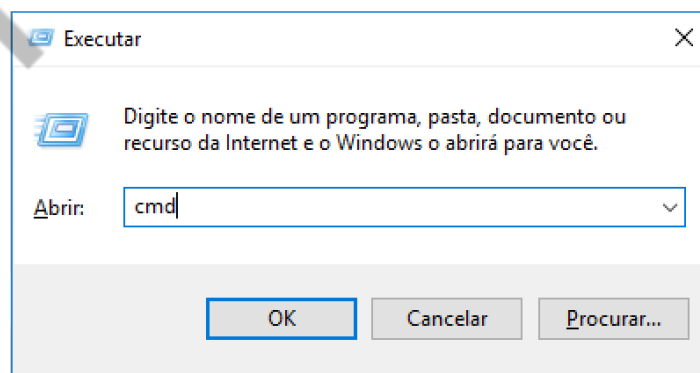


Figura 7.4 – Instalador do NodeJS  
Fonte: Elaborada pelo Autor (2019)

Assim que instalado, vamos nos certificar de que tudo ocorreu corretamente. Inicialmente, vamos abrir um “prompt de comando” por meio das teclas +r e digitar “cmd”. Clique em “OK” e um terminal será apresentado. Sempre que nos referirmos ao terminal do Windows, execute esses passos para abri-lo.





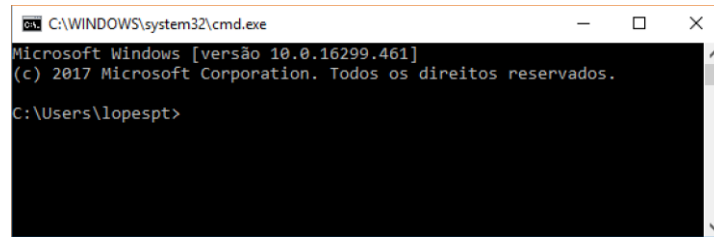


Figura 7.5 – Prompt de comando do windows (terminal)  
Fonte: Elaborada pelo Autor (2019)

Digite “node -v” e pressione a tecla enter no terminal. Você deverá observar a versão do NodeJS que instalou. Certifique-se de que essa versão seja, no mínimo, v10.15.3.

## 7.5 Visual Studio Code

O Visual Studio Code é uma IDE (*Integrated Development Environment*), ambiente de desenvolvimento integrado muito utilizado para programação WEB. Uma IDE integra diversas ferramentas que tornam o desenvolvimento muito mais ágil e prático, tais como: terminal integrado, substituição de palavras em múltiplos arquivos, formatação de códigos-fonte, entre muitas outras ferramentas. A tela inicial do VS Code é ilustrada na Figura “Tela inicial do Visual Studio Code”.

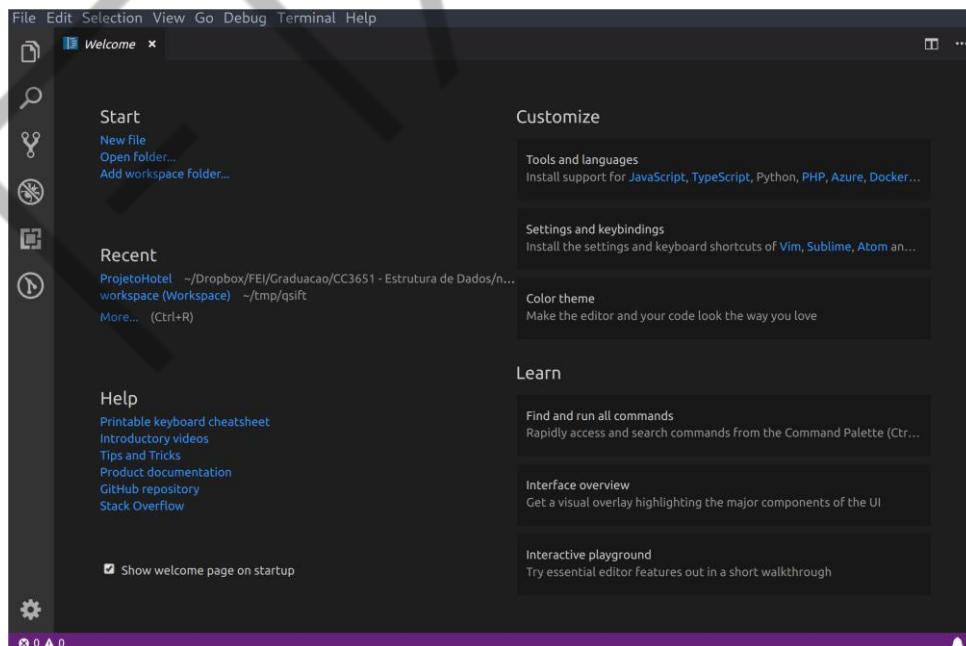



Figura 7.6 – Tela inicial do Visual Studio Code  
Fonte: Elaborada pelo Autor (2019)

Para instalação, basta acessar o site <https://code.visualstudio.com/> e fazer download da aplicação. A instalação poderá ser feita seguindo as instruções na tela.

Para criar um projeto no VS Code, precisamos apenas criar uma pasta nova e abri-la pelo menu **File**→**Open Folder...**

O VS Code conta com diversas ferramentas úteis. As próximas seções detalham essas ferramentas.


### 7.5.1 Explorer

O *Explorer* é um módulo do VS Code capaz de exibir os arquivos e pastas do seu projeto. Ele pode ser acessado por meio do ícone .

### 7.5.2 Terminal integrado


O terminal integrado pode ser aberto por meio do menu **Terminal**→**New Terminal**. Ele é muito útil para executar comandos de sistema, tais como *mkdir* (criação de pastas), *del* (deletar arquivos e/ou pastas), *node* (executar o node), *npm* (instalar pacotes no node), entre muitos outros. Esse terminal é o mesmo que utilizamos na **Erro! Fonte de referência não encontrada**. “Prompt de comando do Windows (terminal)”.

### 7.5.3 Controle de versionamento

Sempre que nos envolvemos em um grande projeto, precisamos gerenciar as diferentes versões de todos os arquivos envolvidos. Se alguma alteração em um arquivo “quebrar” a aplicação, podemos facilmente reverter a alteração que gerou o problema. A ferramenta integrada de versionamento no VS Code é acessada por meio do ícone . É possível utilizar diversos sistemas de versionamento, contudo, o GIT apresenta a melhor integração.

### 7.5.4 Gerenciamento de extensões

Extensões são módulos adicionais que trazem novas ferramentas ao VS Code. Um exemplo de extensão é “Brazilian Portuguese – Code Spell Checker”, que

habilita a verificação de ortografia em arquivos de código-fonte. O gerenciador de extensões pode ser encontrado por meio do ícone .

Neste capítulo, usaremos o plugin “ES7 React/Redux/GraphQL/React-Native snippets”, ilustrado na **Erro! Fonte de referência não encontrada.**. Para instalar o *plugin*, basta clicar em **Install**.

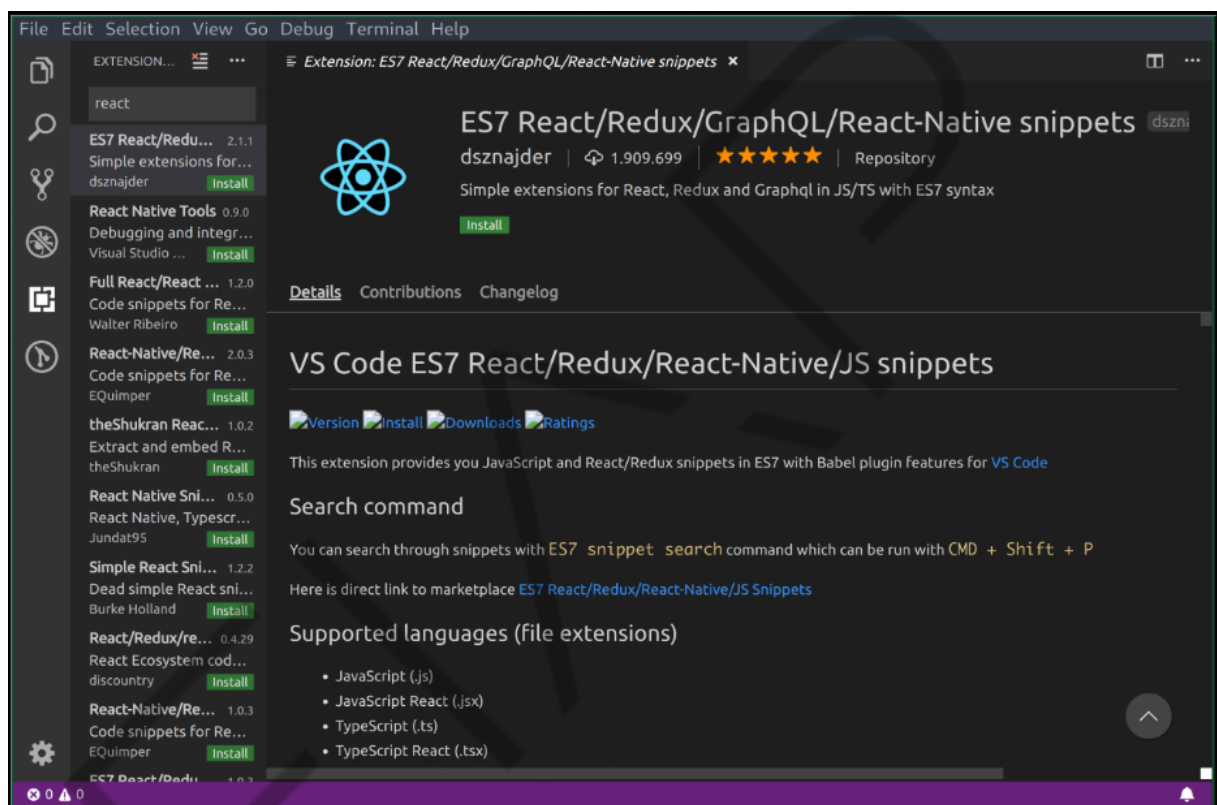


Figura 7.7 – ES7 React plugins  
Fonte: Elaborada pelo autor (2019)

## 7.6 Preparando o ambiente

Para manter todos os nossos arquivos organizados, este capítulo manterá todos os códigos dentro da pasta local `C:\CodigosReact`. Dessa forma, a primeira coisa a fazer é criar essa pasta em sua máquina.

## 7.7 Sua primeira aplicação React

Para a criação de uma aplicação React, utilizaremos uma ferramenta de auxílio chamada “Create React App”. Essa ferramenta pode ser instalada e executada através do *npx* (*executor de comandos npm*) em um terminal aberto por meio do método da Seção 10.5.2 – **Erro! Fonte de referência não encontrada.**, ou pelo método da Seção 10.4 – **Erro! Fonte de referência não encontrada.**). Aqui, será utilizado o terminal integrado ao VS Code.

O primeiro passo é abrir o Visual Studio Code. Certifique-se de que nenhum projeto esteja aberto. Para isso, clique no menu **File**→**Close Folder**. Após a abertura, acesse um novo terminal e digite:

```
cd C:\CodigosReact\  
npx create-react-app primeiro_app
```

Código-fonte 7.1 – Exemplo de código-fonte HTML.  
Fonte: Elaborado pelo autor (2019)

O primeiro comando altera a pasta de trabalho do terminal para nossa pasta de códigos. O segundo comando executa um módulo do node para criação de uma aplicação padrão React. A **Erro! Fonte de referência não encontrada.** ilustra a execução desses comandos. Note que, após a execução, uma árvore de pastas e arquivos é criada em uma pasta chamada `primeiro_app`.

O próximo passo é abrir o projeto `primeiro_app` no VS Code. Clique no menu **File**→**Open Folder...** e selecione a pasta `C:\primeiro_app`. Agora, abra um novo terminal e digite `npm run start`. Espere até que um browser seja aberto e apareça uma tela semelhante à ilustrada na Figura “Primeira aplicação React”.

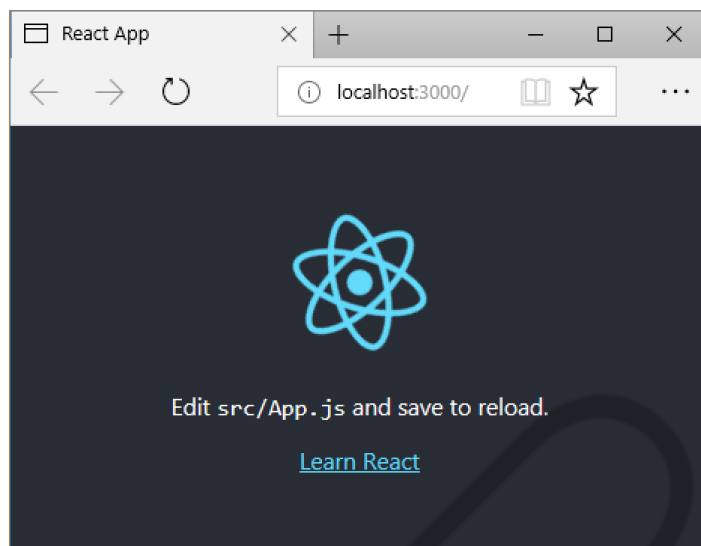


Figura 7.8 – Primeira aplicação React  
Fonte: Elaborada pelo autor (2019)

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

PS C:\Users\lopespt> cd C:\CodigosReact\
PS C:\CodigosReact> npx create-react-app primeiroapp
npx: instalou 63 em 7.469s

Creating a new React app in C:\CodigosReact\primeiroapp.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...

+ react-dom@16.8.5
+ react@16.8.5
+ react-scripts@2.1.8
added 1843 packages from 718 contributors and audited 36244 packages in 152.133s
found 63 low severity vulnerabilities
  run `npm audit fix` to fix them, or `npm audit` for details

Success! Created primeiroapp at C:\CodigosReact\primeiroapp
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 primeiroapp
  npm start

Happy hacking!
PS C:\CodigosReact> █
```

Figura 7.9 – Resultado da execução dos comandos para criação de uma aplicação React  
Fonte: Elaborado pelo autor (2019).

Sua Primeira aplicação React está criada! Nas próximas seções, vamos entender como alterar essa aplicação para deixar do jeito que você quiser.

## 7.8 Principais conceitos

O primeiro conceito importante a saber sobre o React é que ele é baseado em componentes. Tais componentes são os mesmos que abordamos na Seção **Erro! Fonte de referência não encontrada.** Cada componente apresenta uma representação em HTML (para ser renderizado) e comportamentos programados em JavaScript (para interações com o usuário ou o próprio sistema).

Na aplicação criada na Seção **Erro! Fonte de referência não encontrada.**, pode-se perceber que existem três pastas: `node_modules`, `public` e `src`. A pasta `node_modules` contém os pacotes (instalados com o `npm`) que são necessários para a utilização do React. A pasta `public` contém o arquivo `index.html`, a página principal do site. Note que esse arquivo é muito simples e, se você abri-lo diretamente em seu navegador, verá apenas uma página em branco – em breve entenderemos o porquê disso.

Finalmente, a pasta `src` contém diversos arquivos JavaScript, CSS, entre outros. Aqui está a parte mais importante da aplicação.

Observe o arquivo `index.js`, que está na pasta `src`:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

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

// If you want your app to work offline and load faster, you
// can change
// unregister() to register() below. Note this comes with
// some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

Código-fonte 7.2 – `index.js` gerado por create-react-app.  
Fonte: Create-react-app (2019)

A primeira observação sobre esse código é a instrução `import`. Essa instrução é interpretada pelo NodeJS para inserir códigos-fonte que estão em outros arquivos, permitindo a modularização (separação) de códigos-fonte.

Outra parte interessante do código é a tag `<App />`. `App` é um componente que foi importado anteriormente (na linha 4 do código). Usamos um componente da mesma forma como fossemos escrever um código HTML – por meio de tags.

Experimente trocar a linha:

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

por:

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

Note que `div` deve estar em minúsculo. Recarregue a página e verifique que sua aplicação agora renderiza “Teste” na tela. Caso não apareça nada, você pode ter terminado o comando `npm run start`. Se esse for o caso, execute-o novamente.

## 7.9 JSX: a união entre HTML e JAVASCRIPT

Você deve ter notado que o arquivo `index.js` parece conter “tags” html dentro de parâmetros de funções, até escrevemos uma `div` dentro da chamada `render`. Essa sintaxe se chama JSX e faz referência à “JavaScript Extension”. Contudo, o JavaScript citado aqui é o ES6 que, na escrita deste capítulo, é a última versão do JavaScript.

Essa nova versão adiciona funcionalidades à linguagem, tais como: classes, herança, *arrow functions*, *destructuring*, entre muitas outras.

O JSX é fundamental para a escrita e o uso de componentes React, uma vez que é por meio dele que escrevemos o código HTML que será inserido na página quando o componente for utilizado.

A versão original do arquivo `index.js` utilizou o componente `App` por meio da tag `<App/>`. No React, nossos componentes sempre devem ter a primeira letra

maiúscula e é assim que o React sabe quando deve utilizar as “tags” originais do HTML ou buscar em outros arquivos.

No decorrer deste capítulo, você terá mais contato com o JSX. Por enquanto, saiba que esse é o meio pelo qual podemos escrever códigos HTML em um componente React.

## 7.10 Componentes

Um dos principais destaques do React está justamente na capacidade de se criar Componentes (códigos independentes). O arquivo `App.js` nos dá uma dica de como um componente é criado.

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save to reload.
          </p>
          <a
            className="App-link"
            href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer"
          >
            Learn React
          </a>
        </header>
      </div>
    );
  }
}

export default App;
```

Código-fonte 7.3 – App.js gerado por create-react-app.  
Fonte: create-react-app (2019)

Vamos separar esse código em duas partes: importação e exportação de códigos e codificação do componente.



As próximas seções descrevem cada parte do código “App.js gerado por create-react-app”.

### 7.10.1 Importação e exportação de códigos

No Código-fonte App.js gerado por create-react-app, a importação e exportação de códigos fonte é feita por meio das palavras-chave `import` e `export`. O comportamento dessas duas palavras-chave será explicado por este exemplo:

```
let x = 10;
let y = "um texto";

export {x as default, y};
```

Código-fonte 7.4 – Arquivo Teste.js.  
Fonte: O Autor (2019)

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
import g,{y} from './Teste';

ReactDOM.render(<div>{g},{y}</div>,
document.getElementById('root'));

// If you want your app to work offline and load faster, you
// can change
// unregister() to register() below. Note this comes with
// some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

Código-fonte 7.5 – Arquivo index.js.  
Fonte: Adaptado de create-react-app (2019)

A instrução `export` tem a seguinte sintaxe:

```
export {var1, var2, var3... [as default]};
```

em que: `var1, var2...` são variáveis ou objetos que se deseja exportar e `as default` é um parâmetro opcional que exporta a variável como elemento principal do arquivo.

Já no Código-fonte Arquivo index.js., a palavra-chave `import` é utilizada para “capturar” as variáveis do Código Fonte Arquivo Teste.js.

A instrução `import` tem a seguinte sintaxe:

```
import [var],{var1, var2, var3};
```

Em que `[var]` é o nome de como a variável exportada como elemento principal será importada e `var1, var2, ...` são os nomes das outras variáveis exportadas do código externo que devem ser importadas no código local. Dessa forma, teremos `g = 10` e `y = "um texto"`. Note que as variáveis dentro de `{ }` devem ser importadas com o mesmo nome em que foram exportadas. Contudo, a variável exportada como principal pode ter sua importação com qualquer nome.

### 7.10.2 Codificação do componente

No Código-fonte `App.js`, gerado por `create-react-app`, o componente exportado é construído por meio da definição de uma classe, chamada `App`, que herda as propriedades e os métodos de um componente padrão do React, chamado `Component`. Essa é a maneira mais tradicional de se criar componentes. Note que o nome da classe que representa o componente deve iniciar com letra maiúscula.

Todo componente React que precisa ser renderizado na página deve implementar a função `render()`. Essa função deve retornar as tags `JSX` que serão enviadas ao HTML da página principal.

Agora, conseguimos entender todo o código gerado pelo `create-react-app`:

1. Uma página `index.html` é criada com uma `div` e um `id` conhecido;
2. Na pasta `src`, o arquivo `index.js` renderiza o componente `App` na `div` criada no passo 1;
3. O componente `App` foi importado do arquivo `App.js`. Esse arquivo exporta a classe `App`, que é um componente React (uma vez que herda a classe do React);
4. A função `render()` é implementada nessa classe, o que permite ao componente ser renderizado na tela com as tags `JSX`.

5. Finalmente, o comando `npm run start` aciona o NodeJS para efetuar uma série de processamentos para tornar o código React um arquivo JavaScript legível para a maioria dos browsers. Além disso, um servidor é criado e disponibilizado para o programador testar a aplicação.

## 7.11 REACT: uma visão mais profunda

Agora que você já sabe tudo o que está acontecendo na aplicação padrão do React, é hora de conhecer um pouco mais a fundo as funcionalidades principais desse incrível framework.

### 7.11.1 Props

Uma *prop* é a maneira com que o React recebe dados de entrada para um componente. As *props* são recebidas através de atributos, assim como o HTML faz quando alguma propriedade de tag é configurada:

```
<a href="www.fiap.com.br">Fiap</a>
```

Note que *href* é um atributo que altera o comportamento de um link. A mesma coisa acontece com o React. Podemos, então, alterar o código de exemplo do React de forma a passar uma propriedade para o componente *App*.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App minhaProp="testando" />,
  document.getElementById('root'));

// If you want your app to work offline and load faster, you
// can change
// unregister() to register() below. Note this comes with
// some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

Código-fonte 7.6 – Enviando uma propriedade a um componente.  
Fonte: Adaptado de create-react-app (2019)

Nesse código, enviamos a string “testando” para o componente App. Agora, a próxima etapa é “capturar” essa string do lado do componente:

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <p>{this.props.minhaProp}</p>
          <img src={logo} className="App-logo" alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save
            to reload.
          </p>
          <a
            className="App-link"
            href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer"
            >
              Learn React
            </a>
        </header>
      </div>
    );
  }
}

export default App;
```

Código-fonte 7.7 – Recebendo propriedades em um componente.

Fonte: Adaptado de create-react-app (2019)

O código destacado ilustra a parte que foi alterada. Note que um construtor foi criado. Um construtor é uma função que sempre será executada quando um componente for criado. A palavra-chave `super` invoca o construtor da classe pai, que é `Component`. Não se preocupe, caso você não entenda esses termos. O importante, aqui, é saber que toda vez que desejar receber propriedades em um componente, esse bloco de código deve estar presente.

A outra parte do código que foi alterada imprime a propriedade `minhaProp` dentro de uma tag `<p>`. Nesse trecho `this.props` é um objeto JavaScript que contém todas as propriedades do componente e o bloco `{this.props.minhaProp}` renderiza o conteúdo de `minhaProp`.

A passagem de *props* para um componente é uma parte fundamental do React. Ela permite até mesmo que você passe objetos JavaScript para outros componentes. Tome como exemplo os seguintes códigos:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App minhaProp={{a: 1, b: "abc", c:
"https://picsum.photos/200"}} />,
document.getElementById('root'));

// If you want your app to work offline and load faster, you
// can change
// unregister() to register() below. Note this comes with
// some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

Código-fonte 7.8 – Enviando um objeto como uma propriedade a um componente.

Fonte: Adaptado de create-react-app (2019)

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <p>{this.props.minhaProp.a}</p>
          <p>{this.props.minhaProp.b}</p>
          <img src={this.props.minhaProp.c} />
          <img src={logo} className="App-logo"
alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save
to reload.
        </p>
      </div>
    );
  }
}
```

```
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

Código-fonte 7.9 – Recebendo um objeto em um componente.  
Fonte: Adaptado de create-react-app (2019)

No Código-fonte “Enviando um objeto como uma propriedade a um componente”, note que utilizamos o bloco `{ }` para enviar um objeto ao componente. Sempre que quisermos interpretar algum código como JavaScript, utilize esse bloco.

Agora, tente adicionar uma nova propriedade e renderizá-la no componente.

### 7.11.2 Estados

A seção anterior abordou o conceito de propriedades como uma forma de entrada de dados para um componente. Contudo, um componente deve ter controle sobre seu comportamento. Para ilustrar essa seção, vamos considerar que nosso componente App deva mostrar a hora atual e atualizar-se automaticamente.

Nesse contexto, não faz sentido algum passarmos a hora atual via *props* para o componente, uma vez que o próprio componente deve ter controle sobre sua própria exibição, ou estado.

É muito fácil definir um estado em um objeto. Para isso, devemos especificar um objeto `state` no construtor do componente, como mostrado no Código-fonte “Alterando o componente App para mostrar a hora atual”.

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {hora: new Date()};
  }

  componentDidMount() {
    setInterval( () => {
      this.setState({hora: new Date()});
    }, 1000);
  }

  render() {
    return (
      <div className="App">
        <header className="App-header">
          <p>Hora:
            {this.state.hora.toLocaleTimeString()}
          </p>
          <img src={logo} className="App-logo"
alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save
to reload.
          </p>
          <a
            className="App-link"
            href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer"
          >
            Learn React
          </a>
        </header>
      </div>
    );
  }
}

export default App;
```

Código-fonte 7.10 – Alterando o componente App para mostrar a hora atual.

Fonte: Adaptado de create-react-app (2019)

A linha mais importante desse código é

```
this.state = {hora: new Date()};
```

Aqui, definimos que o estado do componente é composto por um campo chamado “hora” e inicializado como um novo objeto do tipo Date. Na função render, nos referimos a esse objeto através da linha:

```
<p>Hora:{this.state.hora.toLocaleTimeString()}</p>
```

Assim, toda vez que quiser utilizar um campo de estado do componente, basta acessá-lo por meio de `this.state`.

Apesar de essas linhas serem suficientes para mostrar a hora na tela do usuário, ela não atualiza de segundo a segundo. Assim, precisamos definir um temporizador para que seja possível alterar o estado do componente.

Foi isso o que fizemos quando utilizamos a função `componentDidMount`. Essa função é chamada toda vez que um componente é renderizado na tela. A função `setInterval` recebe dois parâmetros. O primeiro deve ser uma função que é executada de tempos em tempos e o período (em milissegundos) no qual essa função será chamada.

Note que aqui utilizamos uma funcionalidade do ES6: *arrow function*. Uma *arrow function* é uma função definida como uma expressão JavaScript. A sintaxe de uma *arrow function* segue a seguinte forma:

```
(par1, par2, par3...) => { bloco de código }
```

No trecho, `par1...` é uma lista de parâmetros da função e `bloco de código` é o corpo da função. Para ilustrar como uma *arrow function* funciona, o Código-fonte “Duas funções criadas de formas diferentes” define a mesma função de duas formas diferentes.

```
function digaOi(texto){  
  alert(texto);  
}  
  
const digaOi = (texto) => {  
  alert(texto);  
}
```

Código-fonte 7.11 – Duas funções criadas de formas diferentes  
Fonte: Elaborado pelo Autor (2019)

Voltando ao exemplo do Código Fonte “Alterando o componente App para mostrar a hora atual”, o bloco de código da *arrow function* utiliza o método



`this.setState` para alterar o campo *hora*. Isso é tudo para que o React atualize a hora.

Uma observação importante a fazer é que a alteração de estados não deve ser feita de forma direta; isto é, a linha: `this.state.hora = new Date();` não é a forma correta de alterar o campo. Em vez disso, utilize `this.setState({hora: new Date()});`.

Com relação ao ciclo de vida de um componente, a **Erro! Fonte de referência não encontrada** ilustra quando cada função de um componente é chamada.

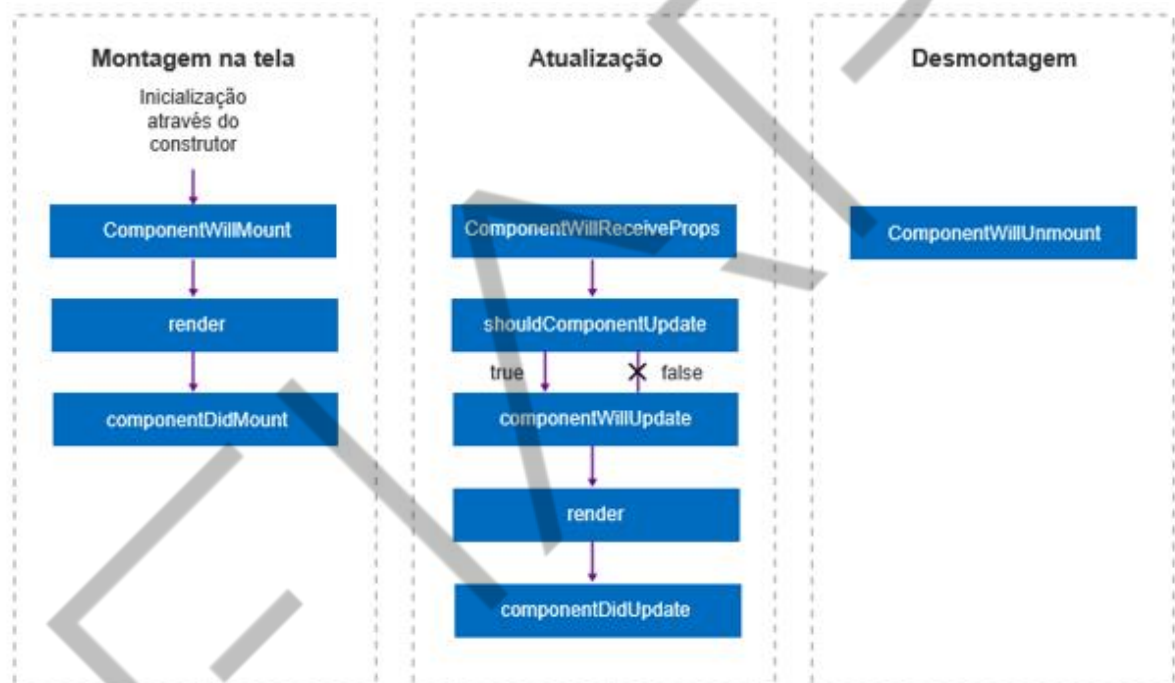


Figura 7.10 – Ciclo de vida de um componente

Fonte: adaptado de [https://rangle.github.io/react-training/react-lifecycles/\(2019\)](https://rangle.github.io/react-training/react-lifecycles/(2019))

### 7.11.3 Eventos

O comportamento dos componentes React reagem à forma com que o usuário interage com uma aplicação. Programar o que um componente deve fazer em um clique de mouse, ao teclar “enter”, ou ao rolar o mouse sobre uma imagem são tarefas fundamentais para trazer ao usuário essa interatividade.

O conceito por trás dessas ações é o recurso de Eventos. Um evento em React nada mais é que uma **declaração** do que deve ser feito **quando** algo acontece.

A maneira mais fácil de declarar um evento é bem semelhante ao HTML. Veja os Códigos Fonte “Declaração do evento clique com JavaScript e HTML” e “Declaração do evento clique no React com JSX”.

```
<input type="button" value="clique aqui" onclick="funcao()" />
```

Código-fonte 7.12 – Declaração do evento clique com JavaScript e HTML

Fonte: Elaborado pelo Autor (2019)

```
<input type="button" value="clique aqui" onclick={funcao} />
```

Código-fonte 7.13 – Declaração do evento clique no React com JSX

Fonte: Elaborado pelo o Autor (2019)

Note que, no caso do React, a função não é chamada, e, sim, somente enviada ao evento click. Para ilustrar um exemplo completo, considere o mesmo código do create-react-app. Vamos ao componente App e alterá-lo conforme o Código-fonte “Declaração do evento clique no React com JSX”.

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  constructor(props) {
    super(props);
  }

  clicou() {
    alert("Usuário clicou no botão");
  }

  render() {
    return (
      <div className="App">
        <header className="App-header">
          <input type="button" value="clique aqui"
onOnClick={this.clicou} />
          <img src={logo} className="App-logo"
alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save
to reload.
          </p>
          <a
            className="App-link"
            href="https://reactjs.org"
            target="_blank">
            Learn React
          </a>
        </header>
      </div>
    );
  }
}
```

```
        href="https://reactjs.org"
        target="_blank"
        rel="noopener noreferrer"
      >
        Learn React
      </a>
    </header>
  </div>
);
}
}

export default App;
```

Código-fonte 7.14 – Declaração do evento clique no React com JSX  
Fonte: Elaborado pelo Autor (2019)

Teste sua aplicação e clique no botão. Você deverá visualizar um alerta assim que o fizer.

Vamos a um segundo exemplo. A ideia é exibir o texto “clique no botão” em uma div assim que o botão for pressionado. O Código-fonte “Declaração do evento clique no React com JSX” faz exatamente isso.

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  constructor(props) {
    super(props);
    this.state={texto: ""};
    this.clicou = this.clicou.bind(this);
  }

  clicou() {
    this.setState({texto: "clique no botão"});
  }

  render() {
    return (
      <div className="App">
        <header className="App-header">
          <input type="button" value="clique aqui"
onClick={this.clicou}/>
          <div>{this.state.texto}</div>
          <img src={logo} className="App-logo" alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save to reload.
          </p>
          <a className="App-link" href="https://reactjs.org"

```

```
        target="_blank"
        rel="noopener noreferrer" >
        Learn React
      </a>
    </header>
  </div>
);
}
}

export default App;
```

Código-fonte 7.15 – Declaração do evento clique no React com JSX  
Fonte: Elaborado pelo Autor (2019)

Perceba, no código, que utilizamos a estratégia de *States* para definir qual será o texto a ser exibido pela *div*. A função *clicou* é chamada quando o botão é clicado.

A novidade aqui é a linha: `this.clicou = this.clicou.bind(this);`

Quando apontamos ao clique do botão essa função, o objeto `this` perde a referência do componente. Se essa linha não estiver presente, o “`this`” dentro da função `clicou` estará com o conteúdo `undefined`. Como uma forma de reapontar o `this` da função para o componente, essa linha redefine a função colocando o componente como o objeto `this` por meio do método `bind()`.

#### 7.11.4 Refs

Refs são referências a tags JSX. Elas são utilizadas para manipular tais tags a partir de métodos ou funções. Uma ref é criada através da função `React.createRef()`. Depois de criadas, é possível atribuir uma tag JSX através do atributo `ref`.

```
import React, { Component } from 'react';

class Produto extends Component {
  constructor(props) {
    super(props);
    this.referencia = React.createRef();
  }

  componentDidMount() {
    this.referencia.current.value = "Valor configurado via ref";
  }
}
```

```
}

render() {
  return (
    <input type="text" ref={this.referencia} value="" />
  );
}
}

export default Produto;
```

Código-fonte 7.16 – Alterando o valor de uma tag JSX por meio de uma Ref.  
Fonte: Elaborado pelo Autor (2019)

No Código-fonte Alterando o valor de uma tag JSX por meio de uma Ref, utilizamos a `ref this.referencia.current` para alterar o conteúdo de uma caixa de texto.

#### 7.11.4 Técnicas comuns

Essa seção aborda algumas técnicas de programação baseadas em React e JSX. O uso de tais técnicas é importante uma vez que pode reduzir consideravelmente o tamanho e complexidade do código fonte. Além disso, você aprenderá sobre novas funcionalidades do ES6 em conjunto com o JSX.

##### 7.11.4.1 Variáveis com tags HTML

É possível criar variáveis e utilizá-las dentro da função render de um componente.

```
import React, { Component } from 'react';

class MeuComponente extends Component {
  render() {
    let f = <a href="http://www.fiap.com.br">fiap</a>;

    return <div>
      <div>O link da {f}</div>
      <div>Outro link da {f}</div>
    </div>;
  }
}
```

```
export default MeuComponente;
```

Código-fonte 7.17 – Uso de variáveis com conteúdo JSX.

Fonte: Elaborado pelo Autor (2019)

#### 7.11.4.2 Funções que retornam JSX

É possível criar funções e utilizá-las dentro da função render de um componente.

```
import React, { Component } from 'react';

class MeuComponente extends Component {
  constructor(props) {
    super(props);
  }
  criaLink(nome, url) {
    return <a href={url}>{nome}</a>;
  }
  render() {
    return <div>
      <div>{this.criaLink("Fiap",
"http://fiap.com.br")}</div>
      <div>{this.criaLink("Google",
"http://google.com.br")}</div>
    </div>;
  }
}

export default MeuComponente;
```

Código-fonte 7.18 – Uso de variáveis com conteúdo JSX.

Fonte: Elaborado pelo Autor (2019)

#### 7.11.4.3 Uso de Listas e a função map

É possível criar funções e utilizá-las dentro da função render de um componente.

```
import React, { Component } from 'react';

class MeuComponente extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    let v = [1,2,3,4,5];
```

```
    let h = v.map( (valor, indice) => {  
      return <li key={indice} >{valor}</li>  
    });  
  
    return <div>  
      <div><ul>{h}</ul></div>  
    </div>;  
  }  
}  
  
export default MeuComponente;
```

Código-fonte 7.19 – Uso de variáveis com conteúdo JSX.

Fonte: Elaborado pelo Autor (2019)

Primeiramente, criamos um vetor identificado por `v`. Após isso, definimos a variável `h` como resultado da função `map` aplicada em `v`.

A função `map` tem o seguinte protótipo: `v.map( f );`

A função `map` itera por elemento do vetor `v` e aplica a função `f`, que, nesse caso, é uma *arrow function*. O resultado da função `map` é um vetor com a mesma quantidade de elementos, contudo, transformados pela função `f`.

É importante notar que a função `f` deve receber dois parâmetros: o valor de cada elemento e o índice desse elemento. Além disso, sempre que utilizarmos o `map`, devemos aplicar o índice de cada elemento no atributo `key` da tag retornada.

Finalmente, o Código-fonte “Uso de variáveis com conteúdo JSX” aplica uma *arrow function* que transforma cada elemento do vetor `v` em uma tag `li`, resultando em um vetor de tags `li`.

#### 7.11.4.4 Criando eventos em seus próprios componentes

É possível criar eventos utilizando o recurso de *props* já visto anteriormente. A estratégia para isso é assumir que uma *prop* é uma função recebida e, a partir daí, chamar a função através do acesso `this.props`. O Código-fonte “Arquivo MeuComponente.js.” ilustra como tudo deve ser feito. É possível criar funções e utilizá-las dentro da função `render` de um componente.

```
import React, { Component } from 'react';
```

```
class MeuComponente extends Component {
  constructor(props) {
    super(props);
    this.clicou = this.clicou.bind(this);
  }

  clicou() {
    if(this.props.funcao !== undefined)
      this.props.funcao();
  }

  render() {
    return <input type="button" value={this.props.texto}
onClick={this.clicou} />
  }
}

export default MeuComponente;
```

Código-fonte 7.20 – Arquivo MeuComponente.js.  
Fonte: Elaborado pelo Autor (2019)

```
import React, { Component } from 'react';
import logo from './logo.svg';
import MeuComponente from './MeuComponente';
import './App.css';

class App extends Component {
  constructor(props) {
    super(props);
    this.clicou = this.clicou.bind(this);
  }

  acaoDoBotao() {
    alert("Evento disparado");
  }

  render() {
    return (
      <div className="App">
        <header className="App-header">
          <MeuComponente texto="Clique Aqui!"
funcao={this.acaoDoBotao} />
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}
```



```
        <a className="App-link" href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer" >
            Learn React
        </a>
      </header>
    </div>
  );
}
}

export default App;
```

Código-fonte 7.21 – Importando seu componente no arquivo App.js.  
Fonte: Elaborado pelo Autor (2019)

#### 7.11.4.5 A propriedade especial *children*

Todo componente React tem uma propriedade especial chamada *children*, que é utilizada para “injetar” o corpo da tag no componente representado por ela. Um exemplo é dado nos Códigos Fonte “Arquivo MeuComponente.js” e “Importando seu componente no arquivo App.js”.

```
import React, { Component } from 'react';

class MeuComponente extends Component {
  constructor(props) {
    super(props);
  }

  render() {
    return <div>
      <h2>{this.props.titulo}</h2>
      {this.props.children}
    </div>
  }
}

export default MeuComponente;
```

Código-fonte 7.22 – Arquivo MeuComponente.js.  
Fonte: Elaborado pelo Autor (2019)

```
import React, { Component } from 'react';
import logo from './logo.svg';
import MeuComponente from './MeuComponente';
import './App.css';

class App extends Component {
  constructor(props) {
```

```
    super(props);
  }

  render() {
    return (
      <div className="App">
        <header className="App-header">

          <MeuComponente titulo="Esse é o título">
            <ul>
              <li>Item 1</li>
              <li>Item 2</li>
              <li>Item 3</li>
            </ul>
          </MeuComponente>
          <img src={logo} className="App-logo" alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save to reload.
          </p>
          <a className="App-link" href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer" >
            Learn React
          </a>
        </header>
      </div>
    );
  }
}

export default App;
```

Código-fonte 7.23 – Importando seu componente no arquivo App.js.

Fonte: Elaborado pelo Autor (2019)

Note que o conteúdo declarado dentro da tag `<MeuComponente>` pôde ser acessado através da propriedade `this.props.children`.

## 7.12 Aplicando o React na prática: Lista de Compras

Agora que já entendemos como tudo funciona, vamos fazer nossa primeira aplicação React. A ideia, aqui, é construir uma lista de compras, contendo itens e seus valores. A aplicação permitirá a adição e remoção de itens e exibirá o total no final da lista.

Para realizar essa aplicação, vamos esboçar como os componentes se comunicarão. O diagrama da **Erro! Fonte de referência não encontrada.** ilustra esse esboço.

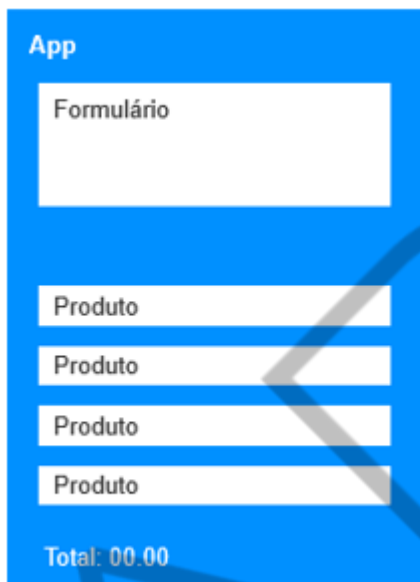


Figura 7.11 – Esboço de componentes para a aplicação proposta  
Fonte: Elaborada pelo Autor (2019)

Na figura, App é o componente principal e contém um componente Formulário para o cadastro de itens na lista. O componente App também tem uma lista de componentes do tipo Produto, que renderizará as informações do produto. Finalmente, o componente App mostra o valor total da lista.

Uma visão da tela é mostrada na Figura “Visão principal da aplicação de lista de compras”.

### Lista de Compras

Adicionar Produto

Nome:

Preço:

Quantidade:

Nome	Preço	Quantidade	SubTotal	
Suco de Laranja	9.50	3.00	28.50	<input type="button" value="deletar"/>
Salgadinho	8.29	2.00	16.58	<input type="button" value="deletar"/>
Manteiga	19.00	3.00	57.00	<input type="button" value="deletar"/>

Total: 102.08

Figura 7.12 – Visão principal da aplicação de lista de compras

Fonte: Elaborada pelo Autor (2019)

O código-fonte é disponibilizado a seguir.

```
import React, { Component } from 'react';
import Formulario from './Formulario';
import Produto from './Produto';

class App extends Component {
  constructor(props) {
    super(props);
    this.adicionar = this.adicionar.bind(this);
    this.remover = this.remover.bind(this);
    this.state = {lista: [], total: "0.00"};
  }

  adicionar(produto) {
    this.setState({lista: this.state.lista.concat(produto)},
      () => {
        let total = 0;
        for(let p in this.state.lista) {
          console.log(p);
          total += this.state.lista[p].preco *
this.state.lista[p].quantidade;
        }
        this.setState({total: total.toFixed(2)});
      });
  }

  remover(indice) {
    this.state.lista.splice(indice,1);
    this.setState({lista: this.state.lista});
    console.log("teste");
  }

  render() {
    return (
      <div>
        <h2>Lista de Compras</h2>
        <fieldset>
          <legend>Adicionar Produto</legend>
          <Formulario evtAdicionar={this.adicionar}/>
        </fieldset>
        <table border="1" cellSpacing="0">
          <thead>
            <tr>
              <th>Nome</th>
```

```

        <th>Preço</th>
        <th>Quantidade</th>
        <th>SubTotal</th>
        <th></th>
      </tr>
    </thead>
    <tbody>
      {
        this.state.lista.map( (prod,idx) => {
          return <Produto evtDeletar={this.remover}
key={idx} indice={idx} produto={prod} />
        })
      }
    </tbody>
  </table>
  <div>Total: {this.state.total}</div>
</div>
);
}
}

export default App;

```

Código-fonte 7.24 – Arquivo App.js.  
Fonte: Elaborado pelo Autor (2019)

```

import React, { Component } from 'react';

class Produto extends Component {
  constructor(props) {
    super(props);
    this.state = {
      nome: "",
      preco: "",
      quantidade: ""
    }
    this.adicionar = this.adicionar.bind(this);
    this.atualiza = this.atualiza.bind(this);
    this.nome = React.createRef();
    this.preco = React.createRef();
    this.quantidade = React.createRef();
  }

  atualiza() {
    if(isNaN(this.preco.current.value)){
      alert("Valor inválido para o campo Preço");
      this.preco.current.value="";
    }
    if(isNaN(this.quantidade.current.value) ||
    this.quantidade.current.value.indexOf(".") >= 0){
      alert("Valor inválido para o campo Quantidade");
      this.quantidade.current.value="";
    }
  }
}

```

```
    }

    this.setState(
      {
        nome: this.nome.current.value,
        preco: parseFloat(this.preco.current.value),
        quantidade: parseFloat(this.quantidade.current.value)
      }
    );
  }

  adicionar() {
    if (this.props.evtAdicionar) {
      this.props.evtAdicionar(this.state);
    }
    this.nome.current.value = "";
    this.preco.current.value = "";
    this.quantidade.current.value = "";
    this.setState({
      nome: "",
      preco: "",
      quantidade: "",
    });
  }

  render() {
    return (
      <div>
        Nome: <input ref={this.nome} type="text"
value={this.props.nome} onChange={this.atualiza} /><br/>
        Preço: <input ref={this.preco} type="text"
value={this.props.preco} onChange={this.atualiza}/><br/>
        Quantidade: <input ref={this.quantidade} type="text"
value={this.props.quantidade} onChange={this.atualiza}/>
<br/>
        <input type="button" value="Adicionar"
onClick={this.adicionar}/>
      </div>
    );
  }
}

export default Produto;
```

Código-fonte 7.25 – Arquivo Formulario.js.  
Fonte: Elaborado pelo Autor (2019)

```
import React, { Component } from 'react';

class Produto extends Component {
  constructor(props) {
    super(props);
  }
```

```
    this.deleta = this.deleta.bind(this);
  }

  deleta() {
    if (this.props.evtDeletar) {
      this.props.evtDeletar(this.props.indice);
    }
  }

  render() {
    return (
      <tr>
        <td>{this.props.produto.nome}</td>
        <td>{this.props.produto.preco.toFixed(2)}</td>
        <td>{this.props.produto.quantidade.toFixed(2)}</td>
        <td>{(this.props.produto.preco *
this.props.produto.quantidade).toFixed(2)}</td>
        <td><input type="button" value="deletar"
onClick={this.deleta} /></td>
      </tr>
    );
  }
}

export default Produto;
```

Código-fonte 7.26 – Arquivo Produto.js.

Fonte: Elaborado pelo Autor (2019)

## Conclusão e Próximos Passos

Este capítulo apresentou o framework React, uma biblioteca escrita em JavaScript para geração de páginas baseadas em componentes. Foram apresentadas algumas funcionalidades e técnicas que reduzem consideravelmente a quantidade e a complexidade de códigos-fonte.

O conteúdo deste capítulo é uma pequena parte do que o React pode fazer. Assim, é importante que o leitor tenha algumas sugestões de conteúdos adicionais e próximos passos para se aprofundar na ferramenta.

O leitor deve ter percebido que a comunicação entre os componentes é dada pela passagem de objetos através das *props*. Em aplicações maiores, isso se torna um problema, uma vez que essas ligações de dados podem se tornar confusas. Essa é a principal motivação para a biblioteca *Redux*. No *Redux*, todos os estados

são mantidos em um único local e os componentes acessam esse local para ler ou gravar dados. Dessa forma, a comunicação entre os componentes fica muito mais fácil.

Uma extensão muito interessante do React é o React Native, que é capaz de criar aplicações nativas para plataformas móveis, como iOS e Android. A vantagem de sua utilização está no fator de um único código poder ser executado tanto para Web como para celulares.

Finalmente, outra questão importante em aplicações React é a navegação entre diferentes páginas. De forma geral, deve-se criar uma aplicação React por página, porém, isso gera lentidão no sistema e muito *overhead* do processador. O React-Router resolve esse problema por meio de uma biblioteca de componentes que definem diferentes “telas” em uma única página. O melhor de tudo isso é que a URL também é interpretada e, para o usuário final, parece que a navegação entre os componentes é uma navegação entre páginas diferentes.



## REFERÊNCIAS

FACEBOOK INC. **React**. 2019. Disponível em: <<http://www.reactjs.org>>. Acesso em: 12 mar. 2019.

FLANAGAN, David. **JavaScript: O Guia Definitivo**. São Paulo: Bookman, 2012.

W3Schools. **EcmaScript 6**. 2015. Disponível em: <[https://www.w3schools.com/js/js\\_es6.asp](https://www.w3schools.com/js/js_es6.asp)>. Acesso em: 12 mar. 2019.

EMANIP