

INTEGRATION

# A ALTERNATIVA DO **FACEBOOK**

GUILHERME ALBERTO WACHS LOPES



**05**

**LISTA DE FIGURAS**

Figura 1 – Painel de criação de publicação do Facebook .....	6
Figura 2 – Diagrama de componentes para “CreatePublicationPanel” .....	6
Figura 3 – Comportamento de um componente TextField no Facebook.....	7
Figura 4 – Instalador do NodeJS.....	9
Figura 5 – Prompt de comando do Windows (terminal) .....	10
Figura 6 – Tela inicial do Visual Studio Code .....	11
Figura 7 – ES7 React plug-ins.....	13
Figura 8 – Primeira aplicação React .....	14
Figura 9 – Resultado da execução dos comandos para a criação de uma aplicação React.....	15
Figura 10 – Ciclo de vida de componentes .....	27
Figura 11 – Esboço de componentes para a aplicação proposta.....	36
Figura 12 – Visão principal da aplicação de lista de compras.....	37

**LISTA DE CÓDIGOS-FONTE**

Código-fonte 1 – Iniciando instalação de terminal aberto.....	14
Código-fonte 2 – index.js gerado por create-react-app .....	16
Código-fonte 3 – App.js gerado por create-react-app .....	19
Código-fonte 4 – Arquivo Teste.js .....	19
Código-fonte 5 – Arquivo index.js.....	20
Código-fonte 6 – Enviando uma propriedade a um componente .....	22
Código-fonte 7 – Recebendo propriedades em um componente .....	23
Código-fonte 8 – Enviando um objeto como uma propriedade a um componente ....	23
Código-fonte 9 – Recebendo um objeto em um componente .....	24
Código-fonte 10 – Alterando o componente App para mostrar a hora atual .....	25
Código-fonte 11 – Duas funções criadas de formas diferentes .....	26
Código-fonte 12 – Declaração do evento clique com JavaScript e HTML.....	28
Código-fonte 13 – Declaração do evento clique no React com JSX .....	28
Código-fonte 14 – Declaração do evento clique no React com JSX .....	29
Código-fonte 15 – Declaração do evento clique no React com JSX .....	30
Código-fonte 16 – Alterando uma propriedade de estilização de uma tag JSX por meio de uma ref .....	31
Código-fonte 17 – Uso de variáveis com conteúdo JSX .....	32
Código-fonte 18 – Uso de variáveis com conteúdo JSX .....	32
Código-fonte 19 – Uso de variáveis com conteúdo JSX .....	33
Código-fonte 20 – Arquivo MeuComponente.js.....	34
Código-fonte 21 – Importando seu componente no arquivo App.js.....	34
Código-fonte 22 – Arquivo MeuComponente.js.....	35
Código-fonte 23 – Importando seu componente no arquivo App.js.....	36
Código-fonte 24 – Arquivo App.js.....	39
Código-fonte 25 – Arquivo Formulario.js .....	39
Código-fonte 26 – Arquivo Produto.js.....	39

## SUMÁRIO

1 A ALTERNATIVA DO FACEBOOK .....	5
1.1 Introdução .....	5
1.2 Sobre o React .....	5
1.3 Pré-requisito deste capítulo .....	7
1.4 NodeJS: quando o JavaScript sai do browser .....	7
1.5 Visual Studio Code .....	10
1.5.1 Explorer .....	11
1.5.2 Terminal integrado .....	11
1.5.3 Controle de versionamento .....	12
1.5.4 Gerenciamento de extensões .....	12
1.6 Preparando o ambiente .....	13
1.7 Sua primeira aplicação React .....	13
1.8 Principais conceitos .....	16
1.9 JSX: a união entre HTML e JavaScript .....	18
1.10 Componentes .....	18
1.10.1 Importação e exportação de códigos .....	19
1.10.2 Codificação do componente .....	20
1.11 REACT: uma visão mais profunda .....	21
1.11.1 Props .....	21
1.11.2 Estados .....	24
1.11.3 Eventos .....	28
1.11.4 useRef .....	31
1.11.4 Técnicas comuns .....	31
1.11.4.1 Variáveis com tags HTML .....	31
1.11.4.2 Funções que retornam JSX .....	32
1.11.4.3 Uso de listas e a função map .....	32
1.11.4.4 Criando eventos em seus próprios componentes .....	33
1.11.4.5 A propriedade especial children .....	35
1.12 Aplicando o React na prática: Lista de Compras .....	36
1.13 Conclusão e próximos passos .....	39
REFERÊNCIAS .....	41

## 1 A ALTERNATIVA DO FACEBOOK

Você já aprendeu como é fácil manipular documentos HTML, eventos e até animações, vamos apresentar a você uma alternativa para a criação de componentes. O ReactJS é mantido pelo Facebook e é a sensação do mercado, sendo utilizado em aplicativos como Airbnb e Instagram. Você está pronto para se aventurar nessa tecnologia?

### 1.1 Introdução

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

### 1.2 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 e gerenciar estados e eventos no browser (Front-End). Para facilitar o entendimento desses frameworks, considere o painel de criação de publicação do Facebook, ilustrado a seguir.

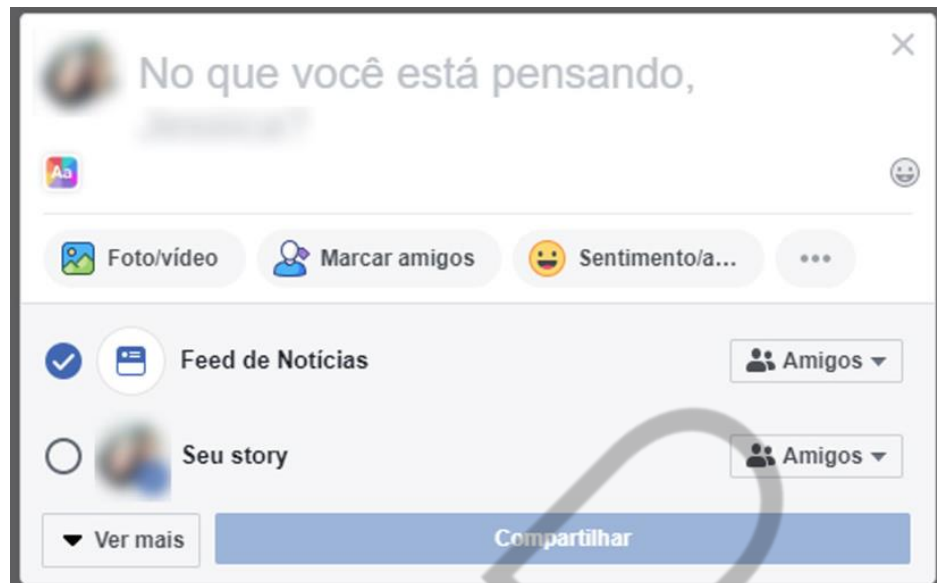


Figura 1 – Pannel de criação de publicação do Facebook  
Fonte: Facebook (2019)

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”.

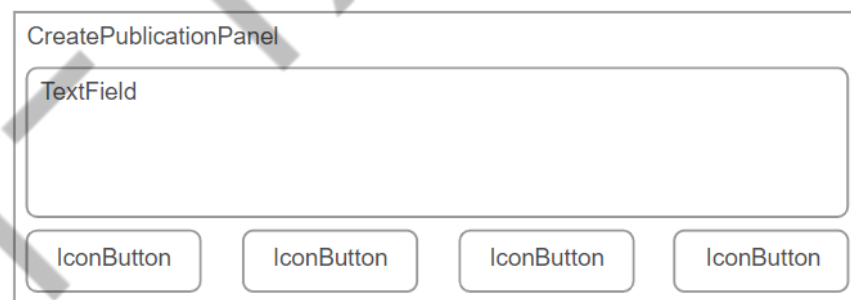


Figura 2 – Diagrama de componentes para “CreatePublicationPanel”  
Fonte: Elaborado pelo autor (2019)

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 é o mesmo, ou seja, há uma explicação em cinza e, ao digitar algo, a explicação é escondida.



Figura 3 – Comportamento de um componente TextField no Facebook  
Fonte: Elaborado pelo autor (2019)

O React também é capaz de descrever esses comportamentos dos componentes. Assim, vimos aqui que tanto o padrão de exibição descrito, em HTML e em 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 com a interação desses componentes com API (Application Program Interface) de sistemas WEB e tratamento de estados desses componentes. Veremos isso mais tarde neste capítulo.

### 1.3 Pré-requisito deste capítulo

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

### 1.4 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 a fim de 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 disso 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 16.17.1. Assim, 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.



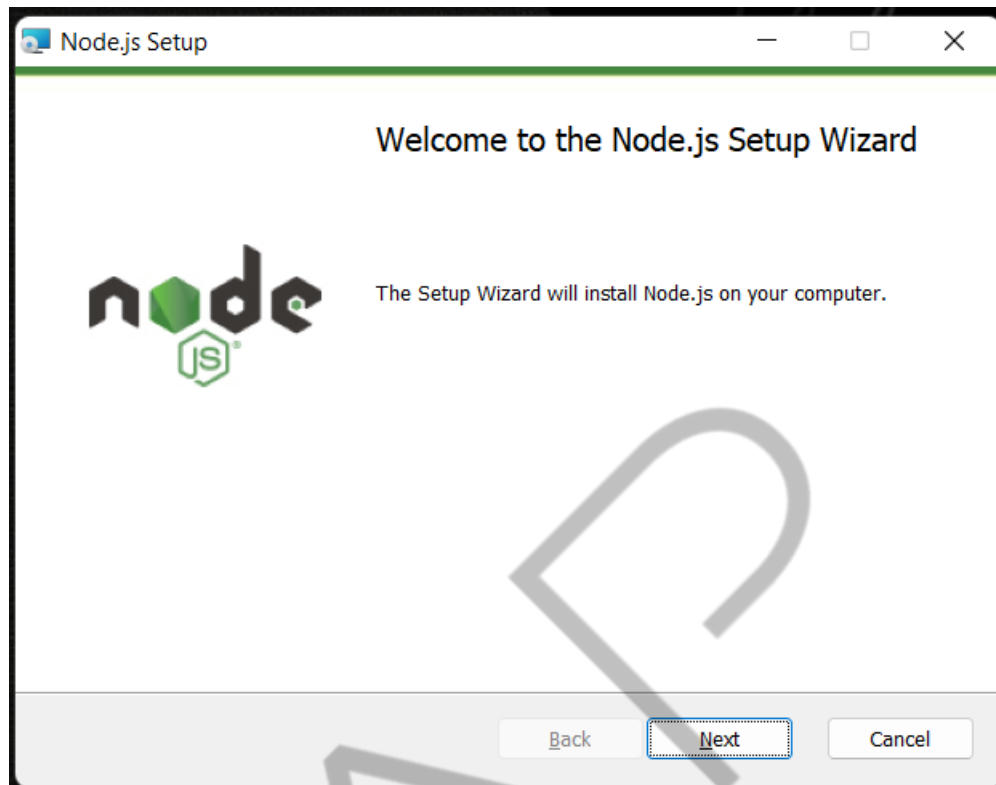



Figura 4 – Instalador do NodeJS  
Fonte: Elaborado pelo autor (2022)

Assim que instalado, vamos nos certificar de que tudo ocorreu corretamente. Para começar, 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.

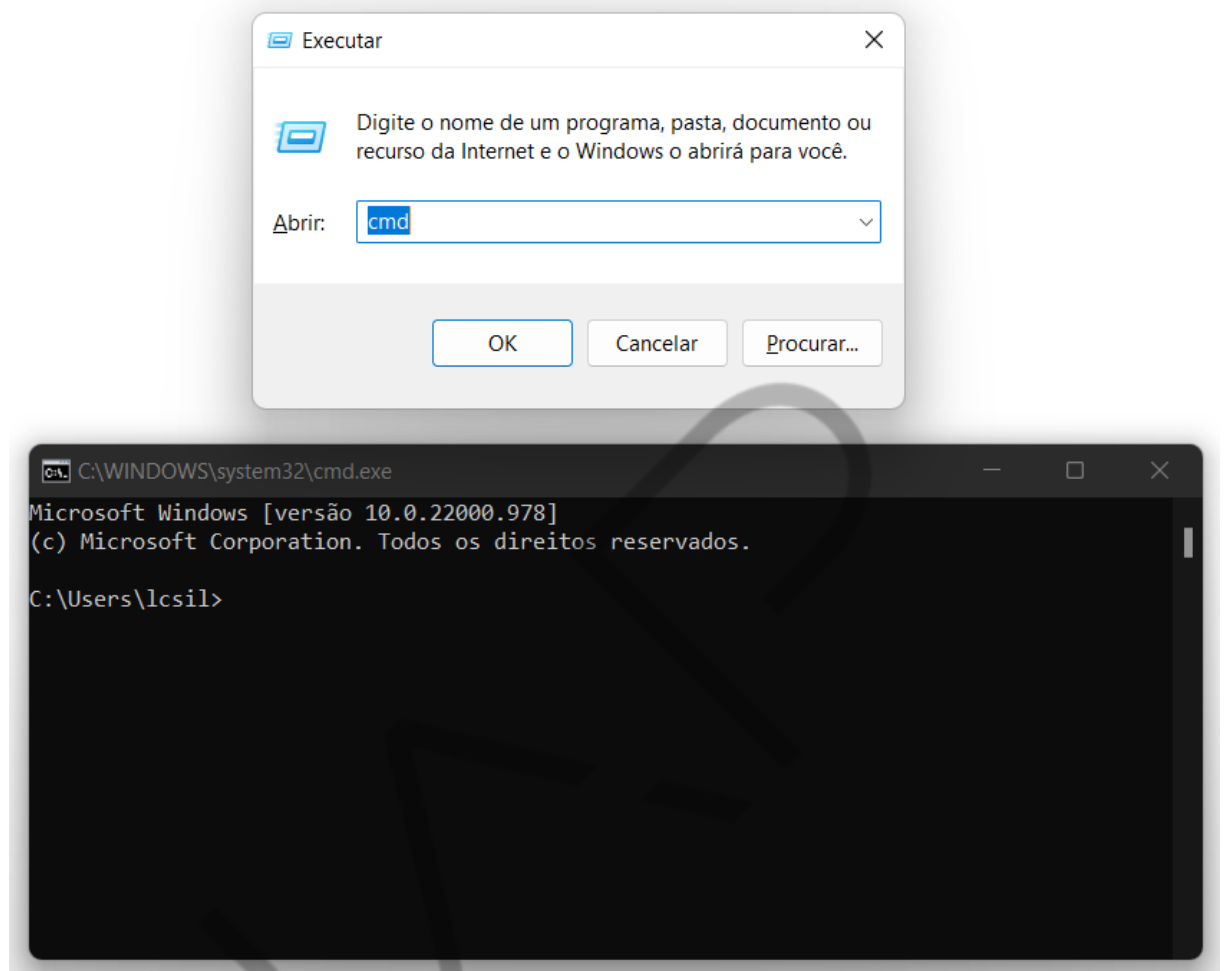


Figura 5 – Prompt de comando do Windows (terminal)  
Fonte: Elaborado pelo autor (2022)

Digite “node -v” e pressione a tecla enter no terminal. Você deve observar a versão do NodeJS que instalou. Certifique-se de que essa versão seja, no mínimo, v16.17.1.

## 1.5 Visual Studio Code

O Visual Studio Code é um IDE (*Integrated Development Environment*), ambiente de desenvolvimento integrado muito utilizado para programação WEB. Um 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 está ilustrada abaixo.

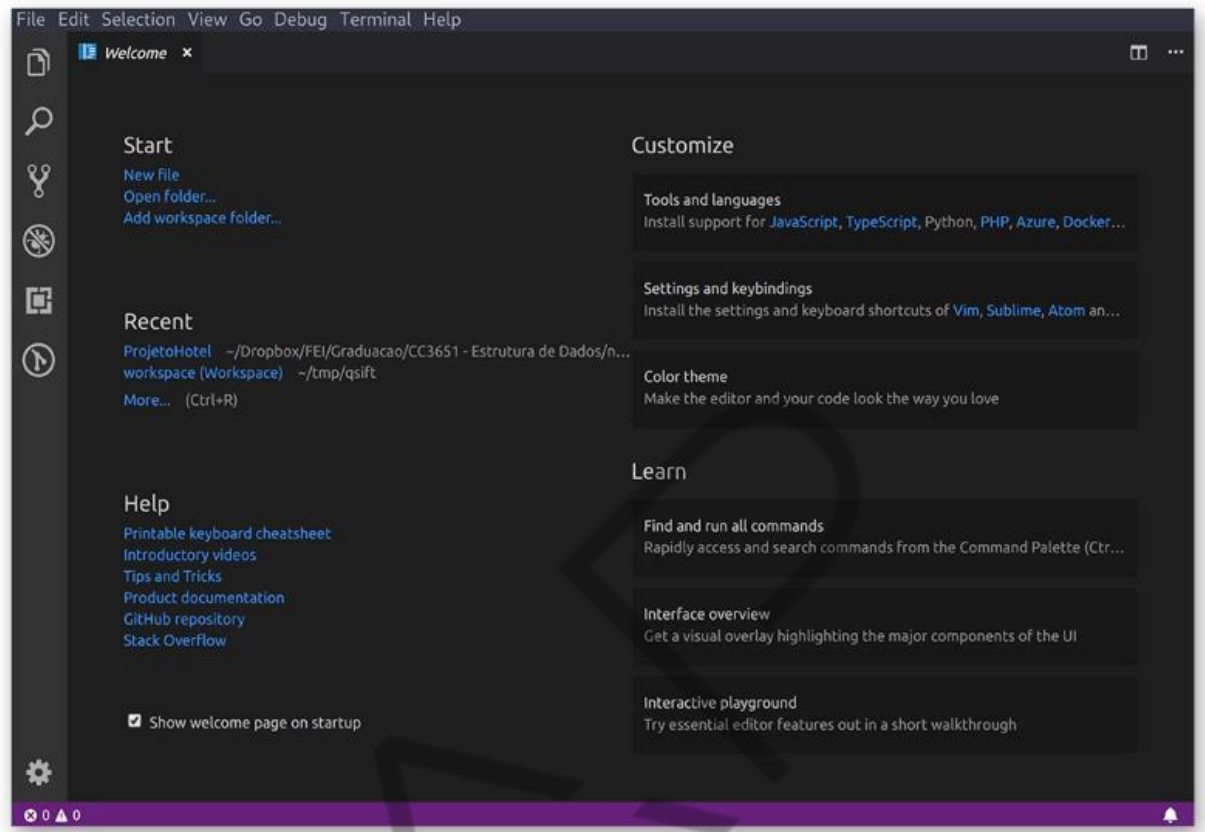



Figura 6 – Tela inicial do Visual Studio Code  
Fonte: Elaborado pelo autor (2019)

Para a instalação, basta acessar o site <<https://code.visualstudio.com/>> e fazer o 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.

### 1.5.1 Explorer


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

### 1.5.2 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)”.

### 1.5.3 Controle de versionamento

Sempre que participamos de 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.

### 1.5.4 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 plug-in “ES7 React/Redux/GraphQL/React-Native snippets”, ilustrado na Figura 7“ES7 React plug-ins”. Para instalar o plug-in, basta clicar em .

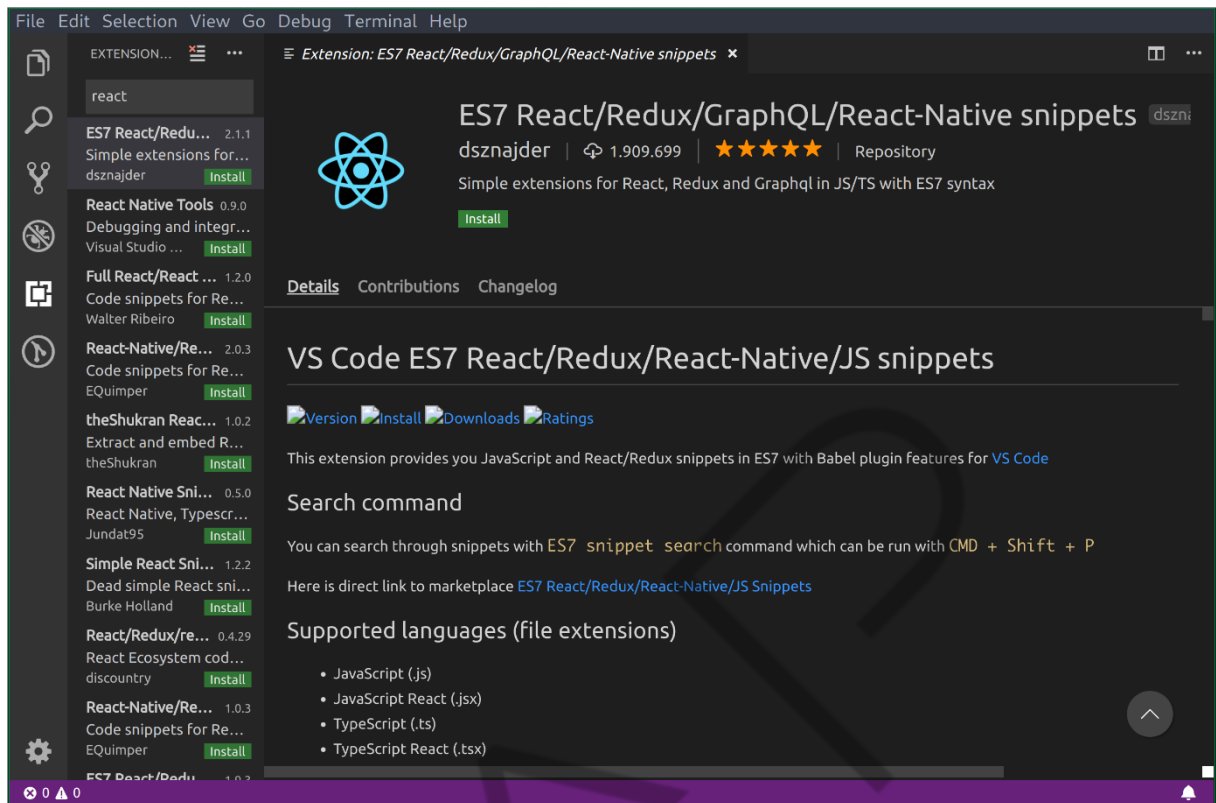


Figura 7 – ES7 React plug-ins  
Fonte: Elaborado pelo autor (2019)

## 1.6 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.

## 1.7 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 por meio do `npx` (executor de comandos npm) em um terminal aberto por meio do método da Seção “**Erro! Fonte de referência não encontrada.**”, ou pelo método da Seção “**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\  
npm create-react-app primeiro_app
```

Código-fonte 1 – Iniciando instalação de terminal aberto  
Fonte: Elaborado pelo autor (2019)

O primeiro comando altera a pasta de trabalho do terminal para a nossa pasta de códigos. O segundo comando executa um módulo do node para a criação de uma aplicação padrão React. A

Figura 9 “Resultado da execução dos comandos para a criação de uma aplicação React” 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 start`. Espere até que um browser seja aberto e apareça uma tela semelhante à ilustrada abaixo.

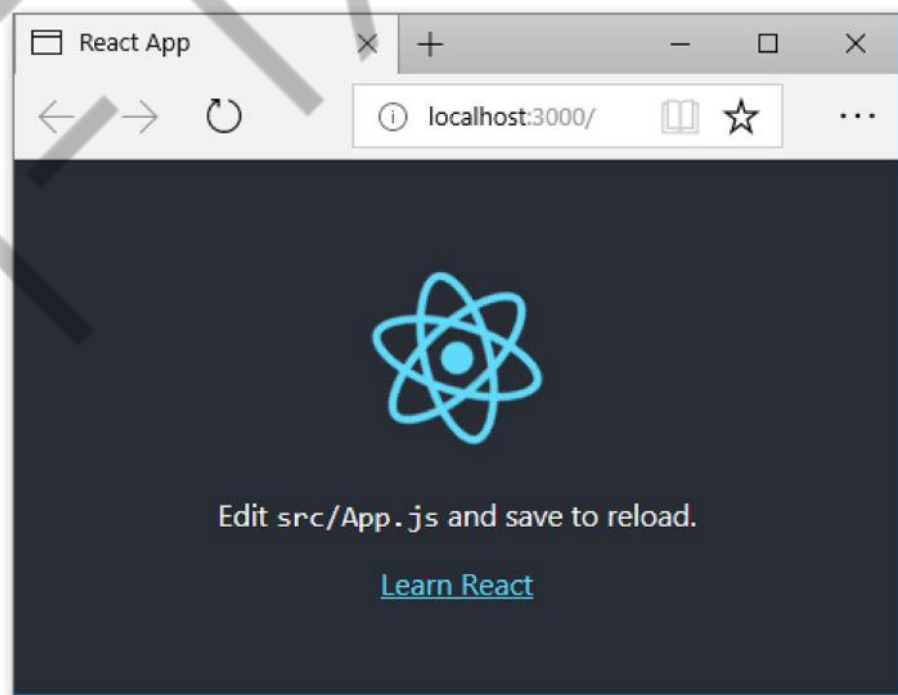


Figura 8 – Primeira aplicação React  
Fonte: Elaborado pelo autor (2019)

```
run `npm fund` for details
Removing template package using npm...

removed 1 package, and audited 1450 packages in 4s

210 packages are looking for funding
  run `npm fund` for details

6 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

Created git commit.

Success! Created primeiro_app at C:\CodigosReact\primeiro_app
Inside that directory, you can run several commands:

  cd primeiro_app
  npm start

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

Figura 9 – Resultado da execução dos comandos para a criação de uma aplicação React  
Fonte: Elaborado pelo autor (2022)

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

## 1.8 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 entre seus arquivos o `index.html`, a página principal do site. Note que esse arquivo é muito simples e se você o abrir 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/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass
// a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more:
// https://bit.ly/CRA-vitals
reportWebVitals();
```

Código-fonte 2 – `index.js` gerado por create-react-app  
Fonte: create-react-app (2022)



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 se fôssemos escrever um código HTML – por meio de tags. Experimente trocar o trecho:

```
const root =  
  ReactDOM.createRoot(document.getElementById('root'));  
  root.render(  
    <React.StrictMode>  
      <App />  
    </React.StrictMode>  
  );
```

por

```
const root =  
  ReactDOM.createRoot(document.getElementById('root'));  
  root.render(  
    <React.StrictMode>  
      <div>Teste</div>  
    </React.StrictMode>  
  );
```

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 start`. Se for esse o caso, execute-o novamente.

## 1.9 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.

## 1.10 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 logo from './logo.svg';
import './App.css';

function App() {
  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>
      </header>
    </div>
  );
}
```

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

export default App;
```

Código-fonte 3 – App.js gerado por create-react-app  
Fonte: create-react-app (2022)

Vamos separar esse código em duas partes: importação e exportação de códigos e codificação do componente. As próximas seções descrevem cada parte do código acima.

### 1.10.1 Importação e exportação de códigos

No Código-fonte “App.js gerado por create-react-app”, a importação e a exportação de códigos-fonte são feitas por meio das palavras-chave `import` e `export`. O comportamento dessas duas palavras-chave será explicado no exemplo abaixo.

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

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

Código-fonte 4 – Arquivo Teste.js  
Fonte: Elaborado pelo autor (2019)

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
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  
reportWebVitals();
```

Código-fonte 5 – Arquivo index.js  
Fonte: Adaptado de create-react-app (2019)

A instrução `export` tem a seguinte sintaxe:

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

Onde `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};
```

Onde `[var]` é o nome com o qual 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 com que foram exportadas. Contudo, a variável exportada como principal pode ter sua importação realizada com qualquer nome.

### 1.10.2 Codificação do componente

No Código-fonte “App.js gerado por create-react-app”, o componente exportado é construído por meio da definição de uma função, chamada `App`, que herda os padrões do React. Note que o nome da função que representa o componente deve iniciar com letra maiúscula, assim como o nome do arquivo criado.

Todo componente React espera como retorno da função que o representa um código composto das 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 função `App`, que é um componente React.
4. A função do componente, em seu `return( )` permite ao componente ser renderizado na tela com as tags `JSX`.
5. Finalmente, o comando `npm start` aciona o NodeJS para efetuar uma série de processamentos a fim de 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.

## 1.11 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.

### 1.11.1 Props

Uma *prop* é a maneira como o React recebe dados de entrada para um componente. As *props* são recebidas por meio 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 modo a passar uma propriedade para o componente *App*.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

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>  
reportWebVitals();

Código-fonte 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 logo from './logo.svg';
import './App.css';

function App(props) {
  return (
    <div className="App">
      <header className="App-header">
        <p>{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 – Recebendo propriedades em um componente  
Fonte: Adaptado de create-react-app (2022)

O código destacado ilustra a parte que foi alterada. Note que **props** foi passado como parâmetro no construtor na função do componente.

Não se preocupe caso você não entenda esses termos. O importante aqui é que saiba que toda vez que desejar receber propriedades em um componente, a palavra **props** deve estar indicando isso no construtor da função.

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

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

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
reportWebVitals();
```

Código-fonte 8 – Enviando um objeto como uma propriedade a um componente  
Fonte: Adaptado de create-react-app (2022)

```
import logo from './logo.svg';
import './App.css';

function App(props) {
  return (
    <div className="App">
      <header className="App-header">
```

```
<p>{props.minhaProp.a}</p>
<p>{props.minhaProp.b}</p>
<img src={props.minhaProp.c} />

    <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 9 – Recebendo um objeto em um componente  
Fonte: Adaptado de create-react-app (2022)

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

### 1.11.2 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 passarmos a hora atual via *props* para o componente, uma vez que o próprio componente deve ter controle sobre sua exibição, ou estado.

É muito fácil definir um estado em um componente. Para isso, devemos utilizar um Hook do React chamado `useState()` com ele podemos controlar o estado de uma variável, de um array, de um objeto e até de um array de objetos. Os hooks são



ferramentas do React para controlarmos os componentes, com eles podemos ter propriedades como ciclo de vida, estado e muitas outras características. O próximo exemplo temos o Código-fonte “Alterando o componente App para mostrar a hora atual”.

```
import logo from './logo.svg';
import './App.css';
import { useState } from 'react';

function App() {

  const [hora, setHora] = useState(new Date())

  setInterval(() => setHora(new Date()), 1000)

  return (
    <div className="App">
      <header className="App-header">

        <p>Hora: {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 10 – Alterando o componente App para mostrar a hora atual

Fonte: Adaptado de create-react-app (2022)

A linha mais importante do código acima é:

```
const [hora, setHora] = useState(new Date());
```

Aqui, definimos o estado da constante hora com o `useState()`, assim criamos um estado para ele e todas as vezes que o seu valor é alterado, ele renderiza o componente na view do navegador. Um state deve ser importado do pacote do React e criado utilizando colchetes para definir seu nome e o método que nos permite alterar seu valor. No construtor do `useState` devemos passar seu valor inicial. No return podemos chamar como uma constante normal, nos referimos a esta linha:

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

Assim, toda vez que quiser utilizar um campo de estado do componente, basta acessá-lo por meio de seu nome e para alterá-lo devemos sempre usar o seu método, que em nosso caso foi o `setHora`, é uma ótima prática sempre criar este método usando a palavra `set` seguida do nome do state. Repare que usamos uma função que 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 esta 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 11 – Duas funções criadas de formas diferentes  
Fonte: Elaborado pelo autor (2019)

Voltando ao exemplo do Código-fonte “Recebendo uma constante em um componente”, o bloco de código da *arrow function* utiliza o método `setHora` para alterar a constante `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

```
hora = new Date();
```

Não é a forma correta de alterar o campo. Em vez disso, utilize:

```
setHora(new Date());
```

Podemos aproveitar, melhor dizendo, monitorar as mudanças no ciclo de vida de um componente para criarmos ações a partir delas, tanto na tela como internamente em nossas informações, para isso, podemos utilizar outro Hook do React, o **useEffect**. Com ele conseguimos criar ações sempre que um componente é alterado, quando é criado, quando um state específico for alterado e até quando ele for desmontado, ou seja, tirado da página.

Para todos estes controles temos mudanças sutis na criação do `useEffect`. Veja na Figura 10 “Ciclo de vida de componentes”, que está abaixo, como criar cada um deles.

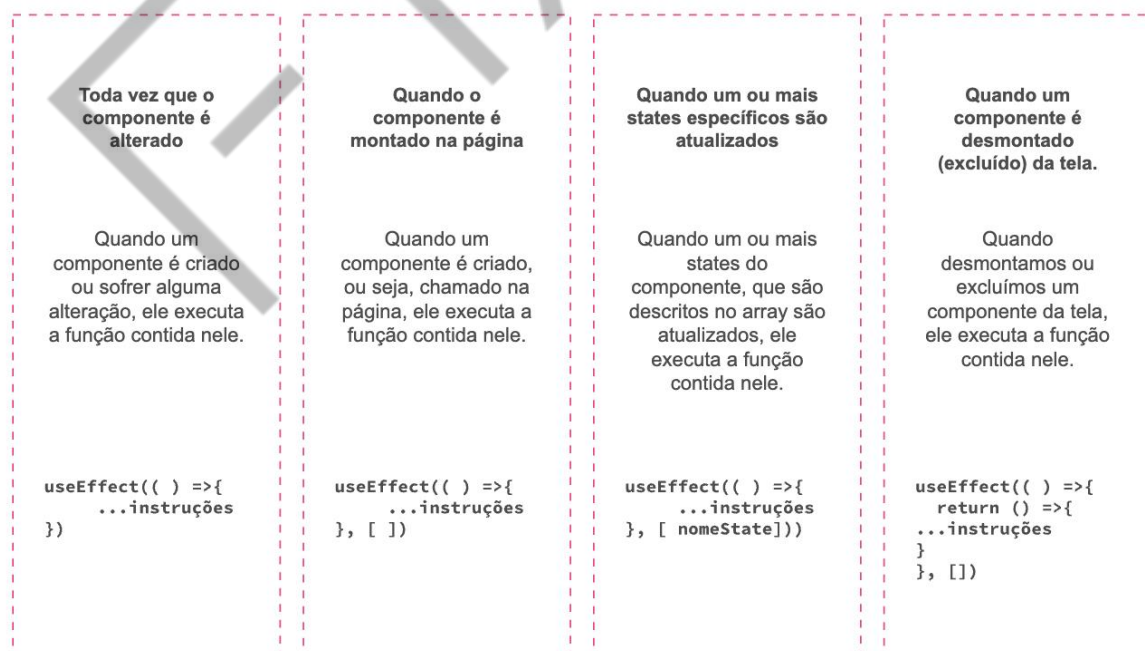


Figura 10 – Ciclo de vida de componentes  
Fonte: Elaborado pelo autor (2022)

Vamos fazer alguns testes com eles, mas primeiro aprenderemos a trabalhar com os eventos.

### 1.11.3 Eventos

Os componentes React reagem à forma como o usuário interage com uma aplicação. Programar o que um componente deve fazer após um clique de mouse, ou 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-fontes “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 12 – Declaração do evento clique com JavaScript e HTML  
Fonte: Elaborado pelo autor (2019)

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

Código-fonte 13 – Declaração do evento clique no React com JSX  
Fonte: Elaborado pelo autor (2019)

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

```
import { useEffect, useState } from 'react'
import logo from './logo.svg';
import './App.css';

function App() {

  const [numero, setNumero] = useState(0)

  function clicou(){
    setNumero(numero + 1)
  }

  useEffect(()=>{ console.log(`aumentou para: ${numero}`)
}, [numero])
```

```
return (
  <div className="App">
    <header className="App-header">
      <h2>{numero}</h2>
      <button onClick={clique}>Clique Aqui</button>
      <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 14 – Declaração do evento clique no React com JSX  
Fonte: Elaborado pelo autor (2022)

Teste sua aplicação e clique no botão. Você deverá visualizar o valor do state número aumentando na Tag H2, onde temos a chamada dela dentro da expression. Também aproveitamos para testar o `useEffect`, fazendo com que ele captasse todas as interações com o state número e se houvesse, ele imprimiria no console uma frase mostrando o aumento do número. Vamos a um segundo exemplo. A ideia é criar um novo state e exibir um segundo número que sempre será o dobro do primeiro, mas ele só vai atualizar quando mandarmos, clicando no segundo botão.

```
import { useEffect, useState } from 'react'
import logo from './logo.svg';
import './App.css';

function App() {

  const [numero, setNumero] = useState(0)
  const [dobro, setDobro] = useState(0)

  function clique(){
    setNumero(numero + 1)
  }
}
```

```
useEffect(()=>{ console.log(`aumentou para: ${numero}`)
},[numero])

function dobrou(){
  setDobro(numero * 2)
}
useEffect(()=>{ console.log(`o dobro de número é:
${dobro}`) })

return (
  <div className="App">
    <header className="App-header">
      <h2>{numero} e o dobro é {dobro}</h2>
      <button onClick={clicou}>Clique Aqui</button> <br/>
      <button onClick={dobrou}>Clique Dobro</button>
      <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 15 – Declaração do evento clique no React com JSX

Fonte: Elaborado pelo autor (2019)

Perceba no código que utilizamos o `useEffect` sem o segundo argumento, o array, assim ele executa sempre que há uma atualização no componente. Quando clicamos no primeiro botão as duas mensagens aparecem no console, mas quando clicamos no segundo botão, somente a mensagem mostrando o dobro aparece no console.

#### 1.11.4 useRef

Refs são referências a tags JSX. Elas são utilizadas para manipular tais tags a partir de métodos ou funções. Uma ref é criada por meio de um Hook do react. Depois de criadas, pode-se atribuir uma tag JSX por meio do atributo `ref`.

```
import { useRef } from 'react'

function Produto() {

  const inputElement = useRef();

  const focusInput = () => {
    inputElement.current.style.backgroundColor = 'yellow';
  };

  return (
    <>
      <input type="text" ref={inputElement} />
      <button onClick={focusInput}>Focus Input</button>
    </>
  );
}

export default Produto
```

Código-fonte 16 – Alterando uma propriedade de estilização de uma tag JSX por meio de uma ref  
Fonte: Elaborado pelo autor (2022)

No Código-fonte “Alterando o valor de uma tag JSX por meio de uma ref”, utilizamos a `ref .current` para alterar a propriedade de estilização do input.

#### 1.11.4 Técnicas comuns

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

##### 1.11.4.1 Variáveis com tags HTML

É possível criar variáveis e utilizá-las no return da função de um componente.

```
function MeuComponente() {
```

```
let f = <a href="https://www.fiap.com.br">FIAP</a>

return(
  <div>
    <p>Link da FIAP: {f}</p>
    <p>Outro Link da FIAP: {f}</p>
  </div>
)
}
export default MeuComponente
```

Código-fonte 17 – Uso de variáveis com conteúdo JSX  
Fonte: Elaborado pelo autor (2022)

#### 1.11.4.2 Funções que retornam JSX

É possível criar funções e utilizá-las dentro do return de um componente.

```
function MeuComponente() {

  function criarLink(nome, url) {
    return <a href={url}>{nome}</a>
  }

  return(
    <div>
      <p>{criarLink('FIAP',
'https://www.fiap.com.br')}</p>
      <p>{criarLink('GOOGLE',
'https://www.google.com')}</p>
    </div>
  )
}
export default MeuComponente
```

Código-fonte 18 – Uso de variáveis com conteúdo JSX  
Fonte: Elaborado pelo autor (2022)

#### 1.11.4.3 Uso de listas e a função map

É possível criar funções e utilizá-las dentro do return de um componente.

```
function MeuComponente() {

  const numeros = [1,2,3,4,5]

  let loop = numeros.map((valor, indice)=>{
```



```
        <li key={indice}>{'O valor é ${valor}`}</li>
      ))

      return(
        <div>
          <ul>{loop}</ul>
        </div>
      )
    }
  }
  export default MeuComponente
```

Código-fonte 19 – Uso de variáveis com conteúdo JSX

Fonte: Elaborado pelo autor (2022)

Primeiramente, criamos um vetor identificado por `numeros`. Depois disso, definimos a variável `loop` como resultado da função `map` aplicada em `numeros`. A função `map` tem o seguinte protótipo:

```
numeros.map( loop );
```

A função `map` itera por cada elemento do vetor `numeros` e aplica a função `loop`, 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 `loop`.

É importante notar que a função `loop` deve receber dois parâmetros: o valor e o índice de cada elemento. Além disso, sempre que utilizarmos o `map`, devemos aplicar o índice de cada elemento no atributo `key` da tag retornada. Finalmente, o código acima aplica uma *arrow function* que transforma cada elemento do vetor `numeros` em uma tag `li`, resultando em um vetor de tags `li`.

#### 1.11.4.4 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 por meio do acesso `props`. O código-fonte a seguir ilustra como tudo deve ser feito. É possível criar funções e utilizá-las dentro do `return` de um componente.

```
function MeuComponente(props) {

  return(
```

```
        <div>
            <button
onClick={props.funcao}>{props.texto}</button>
            </div>
        )
    }
    export default MeuComponente
```

Código-fonte 20 – Arquivo MeuComponente.js  
Fonte: Elaborado pelo autor (2022)

```
import logo from './logo.svg';
import './App.css';
import MeuComponente from './components/MeuComponente';

function App() {

    function acaoDoBotao() {
        alert('Evento disparado')
    }

    return (
        <div className="App">
            <header className="App-header">
                <MeuComponente texto='Clique Aqui'
funcao={acaoDoBotao}/>
                <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 21 – Importando seu componente no arquivo App.js  
Fonte: Elaborado pelo autor (2022)

#### 1.11.4.5 A propriedade especial children

Todo componente React tem uma propriedade especial chamada children. Ela 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”.

```
function MeuComponente(props) {  
  
  return(  
    <div>  
      <h2>{props.titulo}</h2>  
      {props.children}  
    </div>  
  )  
}  
export default MeuComponente
```

Código-fonte 22 – Arquivo MeuComponente.js  
Fonte: Elaborado pelo autor (2022)

```
import logo from './logo.svg';  
import './App.css';  
import MeuComponente from './components/MeuComponente';  
  
function App() {  
  
  return (  
    <div className="App">  
      <header className="App-header">  
        <MeuComponente titulo='Este 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 23 – Importando seu componente no arquivo App.js  
Fonte: Elaborado pelo autor (2022)

Note aqui que o conteúdo declarado na tag `<MeuComponente>` pôde ser acessado por meio da propriedade `props.children`.

## 1.12 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 a 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 Figura 11 “Esboço de componentes para a aplicação proposta” ilustra esse esboço.

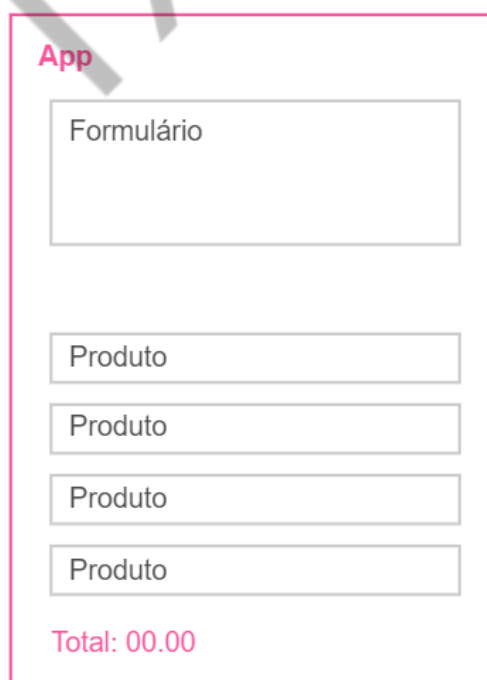
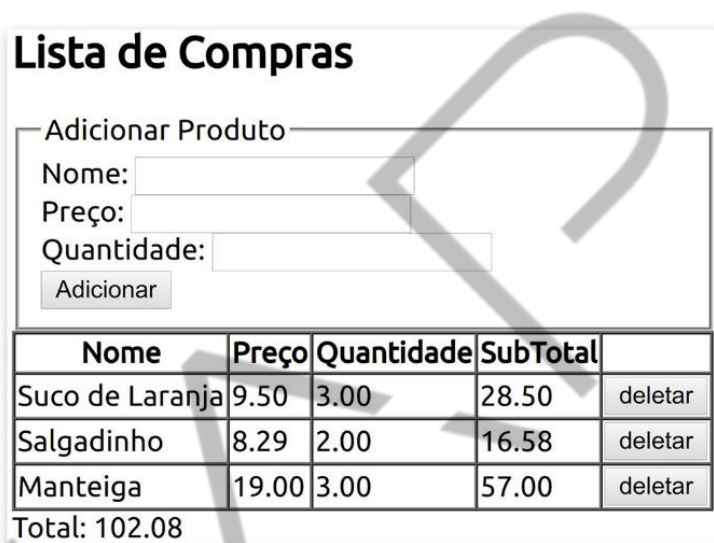


Figura 11 – Esboço de componentes para a aplicação proposta

Fonte: Elaborado 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 possui 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 **Erro! Fonte de referência não encontrada.** “Visão principal da aplicação de lista de compras”.



Nome	Preço	Quantidade	SubTotal	
Suco de Laranja	9.50	3.00	28.50	deletar
Salgadinho	8.29	2.00	16.58	deletar
Manteiga	19.00	3.00	57.00	deletar
Total: 102.08				

Figura 12 – Visão principal da aplicação de lista de compras  
Fonte: Elaborado pelo autor (2019)

O código-fonte é disponibilizado a seguir:

```
import logo from './logo.svg';
import './App.css';
import { useState } from 'react';
import Formulario from './components/Formulario';
import Produto from './components/Produto';

function App() {

  const [produto, setProduto] = useState({'nome': '',
'preco': '', 'quantidade': ''})
  const [lista, setLista] = useState([])
  const [total, setTotal] = useState(0)

  function totalCompra(){
  }

  function adicionar(e){
    e.preventDefault()
```

```

    setLista([...lista, produto])
    setTotal(total + (produto.preco * produto.quantidade))
    setProduto({'nome': '', 'preco': '', 'quantidade': ''})
  }

  function captar(e) {
    setProduto({...produto, [e.target.name]: e.target.value})
  }

  function deletar(p) {
    setTotal(total - (p.preco * p.quantidade))
    let listaNova = lista.filter((pro) => pro !== p)
    setLista(listaNova)
  }

  return (
    <div className="App">
      <h2>Lista de Compras</h2>
      <fieldset>
        <legend>Adicionar Produto</legend>
        <Formulario produto={produto} adicionar={adicionar}
captar={captar}/>
      </fieldset>
      <br/>
      <table border="1" cellPadding={0}>
        <thead>
          <tr>
            <th>Nome</th>
            <th>Preço</th>
            <th>Quantidade</th>
            <th>SubTotal</th>
            <th></th>
          </tr>
        </thead>
        <tbody>
          {
            lista.map((p, i) => (
              <Produto key={i}
                nome={p.nome}
                preco={p.preco}
                quantidade={p.quantidade}
                deletar={deletar.bind(this, p)}
              />
            ))
          }
        </tbody>
      </table>
      <p>Total: {total.toFixed(2)}</p>
    </div>
  );
}

```

```
export default App;
```

Código-fonte 24 – Arquivo App.js  
Fonte: Elaborado pelo autor (2022)

```
function Formulario(props) {  
  
  return(  
    <form onSubmit={props.adicionar}>  
      <input name='nome' value={props.produto.nome}  
placeholder="Nome"  
      onChange={props.captar}/> <br/>  
      <input name='preco' value={props.produto.preco}  
placeholder="Preço"  
      onChange={props.captar}/> <br/>  
      <input name='quantidade'  
value={props.produto.quantidade} placeholder="Quantidade"  
      onChange={props.captar}/> <br/>  
      <button type='submit'>Adicionar</button>  
    </form>  
  )  
}  
export default Formulario
```

Código-fonte 25 – Arquivo Formulario.js  
Fonte: Elaborado pelo autor (2022)

```
function Produto(props) {  
  
  let subtotal = props.preco * props.quantidade  
  return (  
    <tr>  
      <td>{props.nome}</td>  
      <td>{Number(props.preco).toFixed(2)}</td>  
      <td>{Number(props.quantidade).toFixed(2)}</td>  
      <td>{Number(subtotal).toFixed(2)}</td>  
      <td><button  
onClick={props.deletar}>Deletar</button></td>  
    </tr>  
  );  
}  
export default Produto
```

Código-fonte 26 – Arquivo Produto.js  
Fonte: Elaborado pelo autor (2022)

## 1.13 Conclusão e próximos passos

Este capítulo apresentou o framework React, uma biblioteca escrita em JavaScript para a 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 por meio 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 fato de um único código poder ser executado tanto para Web como para celulares.

Por fim, 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. **React**. 2019. Disponível em: <<http://www.reactjs.org>>. Acesso em: 13 set. 2021.

FLANAGAN, D. **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: 13 set. 2021.

EMANIP