



INSTITUTO DE GESTÃO E  
TECNOLOGIA DA INFORMAÇÃO

---

## Desenvolvimento Reativo

---

Raphael Ribeiro Gomide

2021

## **Desenvolvimento Reativo**

Raphael Ribeiro Gomide

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

## **Sumário**

---

Capítulo 1. Introdução ao desenvolvimento reativo.....	7
Sistemas Reativos.....	7
Desenvolvimento Reativo.....	8
Assíncrono .....	8
Orientado a eventos .....	9
Dados “imprevisíveis” .....	9
Reação às interações.....	10
Estado da aplicação .....	10
Escrita declarativa .....	10
Características funcionais .....	10
Um exemplo muito comum de reatividade – planilhas.....	12
O padrão de projeto Observer .....	12
RxJS – ReactiveX for JavaScript.....	13
Capítulo 2. Single Page Applications (SPA’s).....	18
Introdução.....	18
SPA’s e o frameworks JavaScript.....	20
Conceitos importantes relacionados a SPA’s.....	21
Requisições assíncronas: .....	21
Manipulação do DOM (Document Object Model).....	25
Roteamento .....	27
Ferramentas para o desenvolvimento .....	28
Node.js .....	28
Microsoft Visual Studio Code .....	29
Capítulo 3. Introdução ao Angular.....	31

Angular.....	31
Instalação e configuração .....	31
Características.....	33
TypeScript.....	37
Implementação simples com Angular – angular-intervalos .....	39
Implementando o template.....	40
Implementando o componente.....	42
Entendendo a implementação da aplicação.....	43
Capítulo 4. Introdução ao React.....	45
React.....	45
Instalação e configuração .....	45
Características.....	48
Arquitetura do React.....	49
O arquivo App.js .....	50
Implementação simples com React – react-intervalos.....	51
Entendendo a implementação da aplicação.....	54
Capítulo 5. Introdução ao Vue.....	56
Vue.....	56
Instalação e configuração .....	56
Características.....	59
Arquitetura do Vue.....	61
Implementação de Aplicações com Vue.....	62
Entendendo a implementação da aplicação.....	64

Capítulo 6. Comparativo: Angular, React e Vue .....	65
Angular.....	65
Vantagens.....	65
Desvantagens.....	66
React.....	66
Vantagens.....	66
Desvantagens.....	67
Vue.....	67
Vantagens.....	67
Desvantagens.....	68
Conclusões finais .....	69
Referências.....	70

## Capítulo 1. Introdução ao desenvolvimento reativo

---

### Sistemas Reativos

---

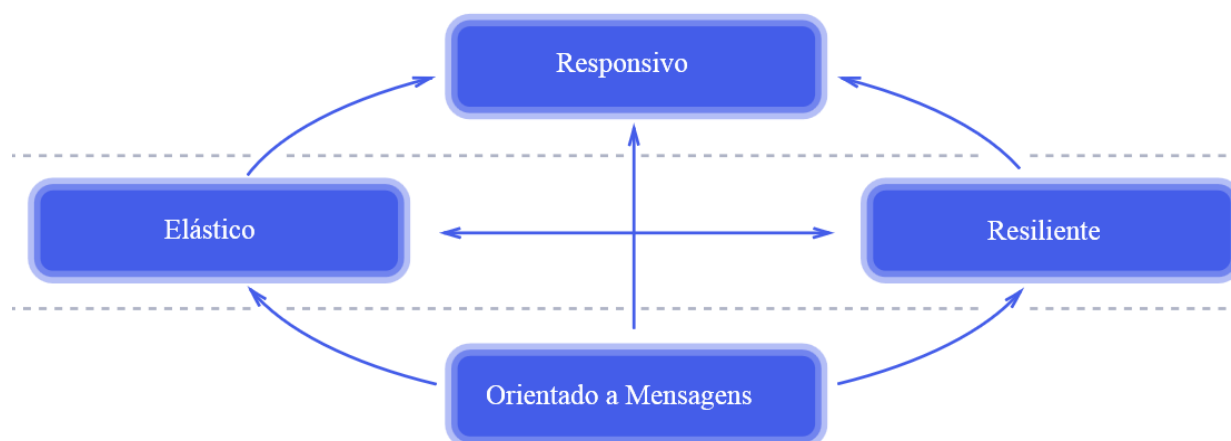
Antes de introduzir o assunto sobre o desenvolvimento reativo propriamente dito, é importante entender o conceito de **sistemas reativos**, que **não** faz parte do escopo desta disciplina.

Sistemas reativos estão relacionados a um conceito mais amplo e representam Sistemas de Informação que possuem basicamente quatro características importantes:

- **Responsivo:** o *feedback* ao usuário é constante e o usuário nunca se sente “abandonado” na utilização do sistema. Procedimentos mais lentos são “amenizados” graças a indicadores de processamento e barra de progresso, por exemplo.
- **Elástico:** o sistema, se escalado, deve manter o mesmo nível de desempenho, independente do seu tamanho.
- **Resiliente:** o sistema deve saber tratar os seus erros de forma amigável, sem prejudicar a experiência do usuário.
- **Orientado a mensagem:** este termo está relacionado ao modo de comunicação entre os componentes do sistema, que é baseado em conceitos de Orientação a Objetos. As mensagens devem ser trocadas de forma assíncrona.

Para mais detalhes sobre sistemas reativos, acesse o manifesto. A figura abaixo ilustra as principais características de um sistema reativo.

**Figura 1 – Características importantes de Sistemas reativos**



Fonte: <https://www.reactivemanifesto.org/pt-BR>

## Desenvolvimento Reativo

O desenvolvimento reativo pode ser definido como um paradigma de programação com as seguintes características:

### Assíncrono

As instruções nem sempre são executadas em sequência. No caso do JavaScript, requisições a API's, por exemplo, levam mais tempo para serem cumpridas e não são, portanto, instantâneas. O JavaScript, por ser não-blocante, não “espera” o término da requisição e o resultado acaba não sendo o esperado. Nesses casos, é preciso mudar a forma de programar. Isso será visto com mais detalhes posteriormente.



## Orientado a eventos

Eventos representam o “**quando**” e são bastante utilizados em *front end*. Um exemplo clássico é a programação de um clique em um botão qualquer. Para que a ação do clique execute alguma instrução é necessário, no mínimo, a seguinte implementação:

- Capturamos o botão através do DOM;
- Atribuímos uma função que será executada **quando** o usuário clicar no botão (evento), com *addEventListener*;
- Implementamos a função a ser executada;
- Exemplo:

**Figura 2 – Implementação de eventos**

```
const button = document.querySelector('#button1');
button.addEventListener('click', hello);

function hello() {
  alert('Hello, world!');
}
```

## Dados “imprevisíveis”

Nem sempre é possível determinar **quando** os dados estarão disponíveis. No exemplo acima, é impossível determinar **quando** o usuário irá clicar no botão. Entretanto, é interessante que o programador esteja “**preparado**” para isso. O comando *addEventListener* cumpre esse requisito.

## Reação às interações

---

O mapeamento de eventos permite que o *software* possa **reagir**. No exemplo acima, **quando** o usuário clica no botão, uma função é executada. Nesta função, é exibida uma mensagem em tela com o texto *“Hello, world!”*.

## Estado da aplicação

---

Em muitos casos, os elementos que são **observados** e/ou que **reagem** às interações são conhecidos como o **estado** da aplicação – são os dados que serão transformados com o tempo e que, com a **reatividade**, dão o *feedback* visual ao usuário.

## Escrita declarativa

---

A escrita de código se torna mais simples, pois o desenvolvedor foca mais em **o que deve ser feito** em vez de **como algo deve ser feito** (característica do paradigma **imperativo** de programação). Um bom exemplo de linguagem declarativa é o próprio HTML e consultas SQL (Structured Query Language) em Bancos de Dados que realizam cálculos e/ou agrupamentos (GROUP BY).

## Características funcionais

---

O desenvolvimento reativo pode englobar características de programação funcional, tais como:

- **Imutabilidade:** os dados não são alterados diretamente. Na verdade, são totalmente substituídos. Isso garante uma menor probabilidade de erros (*bugs*) em detrimento de mais consumo de memória (que atualmente não é mais um grande problema ou *gargalo*).
- **Funções puras (*pure functions*):** funções puras sempre retornam o mesmo valor para um conjunto de parâmetros. Isso garante mais estabilidade no código e portanto, menos *bugs*.

- **Funções que recebem outras funções como parâmetros:** são conhecidas como *call-backs* ou *higher order functions*. São essenciais à programação assíncrona.
- **Encadeamento de funções:** funções cujos resultados servem de entrada para uma outra função, de forma sequencial. Muito utilizado na biblioteca RxJS, que será vista posteriormente. Isso também pode ser feito como padrão de projeto Builder, implementado por algumas linguagens de programação.

A figura abaixo ilustra um exemplo de função pura com JavaScript:

**Figura 3 – Exemplo de função pura (*pure function*)**

```
function sum(op1, op2) {
  return op1 + op2;
}

sum(1, 3); //sempre será 4
sum(2, 3); //sempre será 5
sum(3, 3); //sempre será 6
```

## Um exemplo muito comum de reatividade – planilhas

Em planilhas, é muito comum a definição de fórmulas que irão **reagir** conforme alteração de valores nas células vinculadas a elas. Segue um exemplo:

**Figura 4 – Exemplo de reatividade com planilha**

	A	B
1	A	3
2	B	8
3	C	=B1+B2

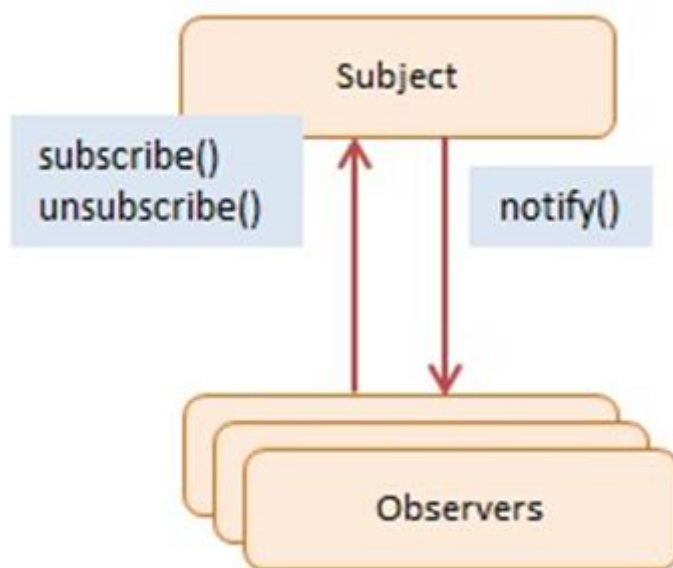
Na figura acima, o valor da célula B3 irá sempre **reagir** caso ocorram alterações em B1 e/ou B2.

Para mais detalhes sobre programação reativa, não deixe de conferir as videoaulas onde são demonstradas algumas aplicações com *Vanilla JS* (JavaScript puro).

## O padrão de projeto *Observer*

O padrão de projeto *Observer* é muito utilizado na manipulação de eventos e interfaces gráficas. A figura abaixo ilustra melhor esse padrão de projeto:

**Figura 5 – Padrão de projeto *Observer***



Fonte: <https://www.dofactory.com/javascript/observer-design-pattern>.

Uma boa analogia para explicar esse padrão é o modelo de assinatura de jornais e/ou revistas. Uma vez que você efetua a assinatura, passa a receber de tempos em tempos os exemplares. Quando você não se interessa mais pelo jornal/revista, pode cancelar a assinatura e deixar de recebê-los a partir de então.

Seguindo a analogia e considerando a figura acima, pode-se dizer que o jornal/revista é o *subject*, a assinatura é o processo de *subscribe*, o cancelamento é o *unsubscribe* e o recebimento de exemplares é o *notify*. Os assinantes são representados pelos *observers*.

Para mais detalhes de implementação desse padrão de projeto com JavaScript, verifique as videoaulas do capítulo 1.

### RxJS – ReactiveX for JavaScript

RxJS é uma biblioteca JavaScript extremamente útil e poderosa. Ela, de certa forma, aumenta muito o poder da manipulação de eventos. Essa biblioteca está disponível atualmente nas seguintes tecnologias/linguagens de programação:

**Figura 6 – Versões da biblioteca ReactiveX**

## Languages

- Java: RxJava
- JavaScript: RxJS
- C#: Rx.NET
- C#(Unity): UniRx
- Scala: RxScala
- Clojure: RxClojure
- C++: RxCpp
- Lua: RxLua
- Ruby: Rx.rb
- Python: RxPY
- Go: RxGo
- Groovy: RxGroovy
- JRuby: RxJRuby
- Kotlin: RxKotlin
- Swift: RxSwift
- PHP: RxPHP
- Elixir: reaxive
- Dart: RxDart

## ReactiveX for platforms and frameworks

- RxNetty
- RxAndroid
- RxCocoa

Fonte: <http://reactivex.io/languages.html>.

Além de ser assíncrona e baseada em eventos, o RxJS é dependência obrigatória do Angular, como será visto posteriormente. Isso **não** significa que o RxJS é **exclusivo** do **Angular**, ou seja, ele pode muito bem ser utilizado por **qualquer framework** JavaScript, inclusive com JavaScript puro (Vanilla JS). O RxJS, em sua essência, contém:

1. O padrão *Observer*, conforme visto anteriormente.

2. O padrão *Iterator*, que é utilizado para iterações (repetições) de forma mais clara ao desenvolvedor e menos propensa a erros.
3. Algumas características de programação funcional como imutabilidade, funções puras, encadeamento de funções e escrita declarativa.

Segue alguns conceitos importantes utilizados pelo RxJS:

- **Observable**: coleção invocável de valores ou eventos futuros.
- **Observer**: coleção de *callbacks* que sabem **reagir** aos valores emitidos pelo *Observable*.
- **Subscription**: representa a execução de um *Observable*. Quando guardamos o *subscription* em uma variável, podemos efetuar o cancelamento da operação através do método *unsubscribe*.
- **Operators**: funções puras para manipular os dados de forma funcional.

A seguir, um exemplo prático – suponha que você precise, por algum motivo, exibir uma lista de todos os números divisíveis por *x*, sendo *x* um valor definido pelo usuário.

Com RxJS, isso é facilmente implementado com algumas linhas de código (considerando um ambiente já configurado).

Para mais detalhes sobre esse exemplo, consulte o seguinte repositório, que foi demonstrado nas videoaulas do capítulo 1 - <https://gitlab.com/rrgomide-rxjs/rxjs-examples.git>.

O exemplo a ser mostrado a seguir está na pasta “09-rxjs-apostila” e as explicações estão nos comentários de código:

**Figura 7 – Implementação simples com RxJS**

```

/**
 * Funções de criação de observables
 * e operadores
 */
const { interval } = rxjs;
const { filter } = rxjs.operators;

/**
 * Função auxiliar para incluir
 * elementos em tela
 */
function addValueToElement(element, value) {
  element.textContent += value + ' | ';
}

/**
 * Função que aciona o observable
 */
function filterFrom(x) {
  const element = document.querySelector('#observable');

  /**
   * A cada 100 milisegundos, o Observable
   * vai emitir um valor iniciando em 0
   */
  const observable$ = interval(100)
    /**
     * "pipe" é usado para encadear funções.
     * Neste caso, usamos somente uma "filter".
     * Estamos filtrando somente elementos divisíveis
     * por 'x'
     */
    .pipe(filter(value => value % x === 0));

  /**
   * Com o subscribe, o observable "passa a funcionar"
   */
  observable$.subscribe(filteredValue => {
    addValueToElement(element, filteredValue);
  });
}

/**
 * Executando a função considerando
 * o valor 7
 */
filterFrom(7);

```



**Figura 8 – Projeto com RxJS em execução**

```
0 | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 | 77 | 84 | 91 | 98 |
105 | 112 | 119 | 126 | 133 | 140 | 147 | 154 | 161 | 168 | 175 | 182
| 189 | 196 | 203 | 210 | 217 | 224 | 231 | 238 | 245 | 252 | 259 |
266 | 273 | 280 | 287 | 294 | 301 | 308 | 315 | 322 | 329 | 336 | 343
| 350 | 357 | 364 | 371 | 378 | 385 | 392 | 399 | 406 | 413 | 420 |
427 | 434 | 441 | 448 | 455 | 462 | 469 | 476 | 483 | 490 | 497 | 504
| 511 | 518 | 525 | 532 | 539 | 546 | 553 | 560 | 567 | 574 | 581 |
588 | 595 | 602 | 609 | 616 | 623 | 630 |
```

É fato que, a princípio, a implementação de RxJS pareça complexa. Ao longo da apostila e das videoaulas, serão vistos vários exemplos para fixar melhor o conteúdo. O mais importante é praticar para se acostumar. Alguns estudiosos dizem que o RxJS chega a ser uma própria linguagem de programação compatível com JavaScript devido à sua sintaxe diferenciada.

A documentação do RxJS em português ainda é, infelizmente, escassa. A documentação oficial em inglês tem evoluído bastante e está disponível em <https://rxjs-dev.firebaseapp.com/>.

Para mais detalhes de utilização e exemplos de implementações com RxJS, verifique as videoaulas de todos os capítulos.

## Capítulo 2. Single Page Applications (SPA's)

### Introdução

Os primeiros sites da Internet eram, em geral, compostos de múltiplas páginas, ou seja, de diversos arquivos .html. O acesso a cada página demandava uma nova requisição ao servidor e o usuário tinha aquele *feedback* visual do recarregamento do site (também conhecido como *refresh*).

Esse modelo é conhecido como MPA (*Multiple Page Applications*) e ainda é utilizado, pois ainda se aplicam bem a diversos cenários (blogs e sites de notícia, por exemplo).

MPA's podem possuir problemas de desempenho durante a navegação, o que pode prejudicar a experiência do usuário em alguns casos.

**Figura 9 – Exemplo de *site* com a tecnologia de *Multiple Page Applications***

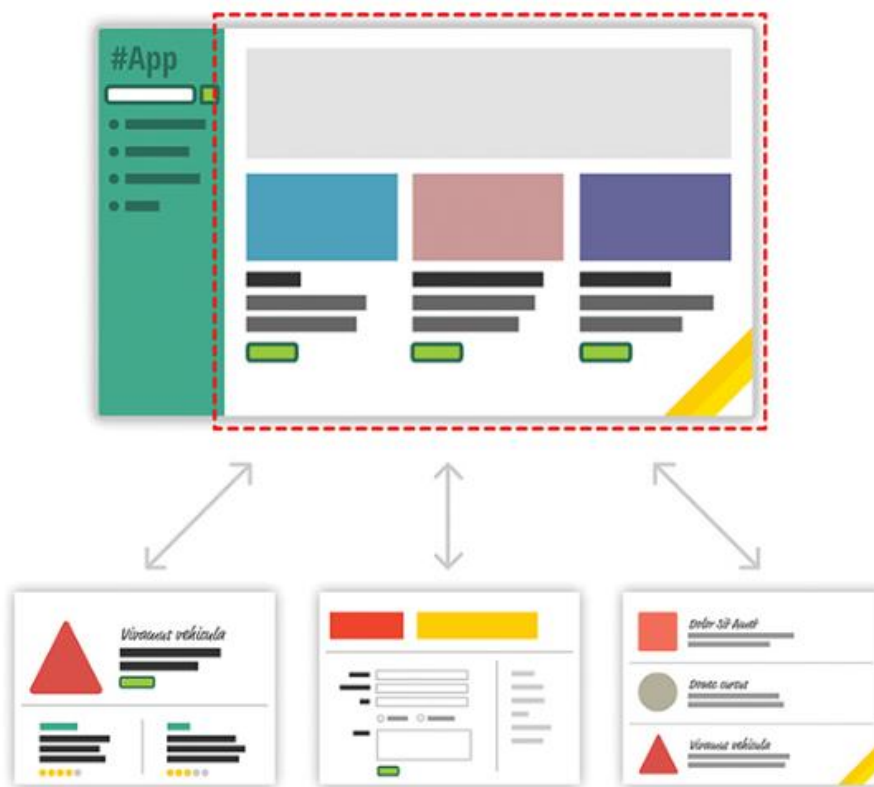


**Fonte: Chinnathambi (2018).**

Para solucionar os problemas das MPA's, surgiu o conceito de SPA's (*Single Page Applications*). Nesse modelo, o site utiliza uma arquitetura onde apenas uma página é carregada e o usuário tem a impressão de estar utilizando uma aplicação *desktop*.

Assim, as transições de telas e páginas são feitas sem a necessidade de *refresh* do navegador o que, em geral, melhora muito a experiência do usuário. Exemplos clássicos são os diversos sites do Google e do Facebook.

**Figura 10 – Exemplo de *site* com a tecnologia de *Single Page Applications***



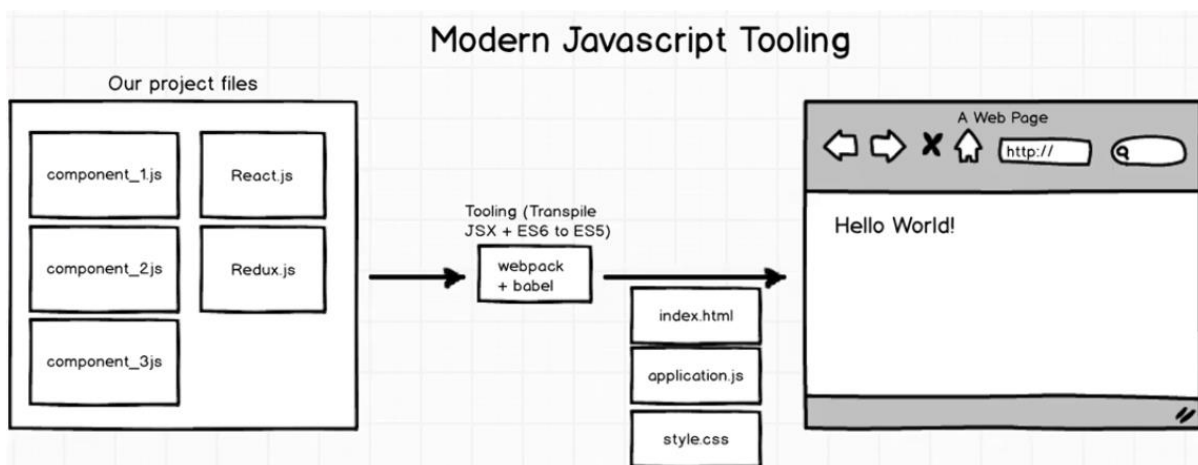
**Fonte: Chinnathambi (2018).**

Mais detalhes e alguns exemplos de SPA's podem ser vistos nas videoaulas do capítulo 2.

## SPA's e os frameworks JavaScript

Atualmente, é muito comum que os *frameworks* JavaScript utilizem o seguinte procedimento para a geração de SPA's:

**Figura 11 – Geração de SPA's em *frameworks* JavaScript**



**Fonte: Grider (2018).**

Resumidamente, o desenvolvedor tem acesso a uma arquitetura muito bem elaborada para o **desenvolvimento**, o que garante uma experiência de desenvolvimento eficiente e prazerosa na maioria dos casos. Entretanto, no momento em que a aplicação é gerada para entrar em **produção** são realizados, em geral, os seguintes procedimentos:

- **Transpilação do código moderno para código compatível:** o código moderno, escrito em ES6+ (ECMA Script 6+) é convertido para um código em ES5, que é mais compatível entre os navegadores atuais. O termo é conhecido como **transpilação** porque é feita uma transformação de código para uma mesma linguagem de programação. Atualmente, a principal ferramenta que faz esse tipo de processamento é o Babel (<https://babeljs.io/>), que está presente no ferramental da grande maioria dos *frameworks*.

- **Empacotamento do código (*packing*):** esse processo consiste em unir todo o código em um ou vários arquivos .js. O processo pode incluir também a minificação do código, em que variáveis são convertidas para identificadores menores, são excluídas quebras de linha, indentações etc. Isso garante uma execução mais eficiente do código. Atualmente, a principal ferramenta que faz esse tipo de trabalho é o Webpack (<https://webpack.js.org/>), que também está presente na grande maioria dos *frameworks*.

Ao final de todo o processo, o site geralmente dispõe de, basicamente, um arquivo .html, um arquivo .css e um arquivo .js, formando então a *Single Page Application*.

Assim, no fim das contas, os arquivos gerados são, em geral, semelhantes aos arquivos de um site feito sem qualquer tipo de *framework*. Com isso, pode-se concluir que *frameworks* dão uma melhor **experiência de desenvolvimento** e não necessariamente uma melhor **experiência de usuário**.

### Conceitos importantes relacionados a SPA's

---

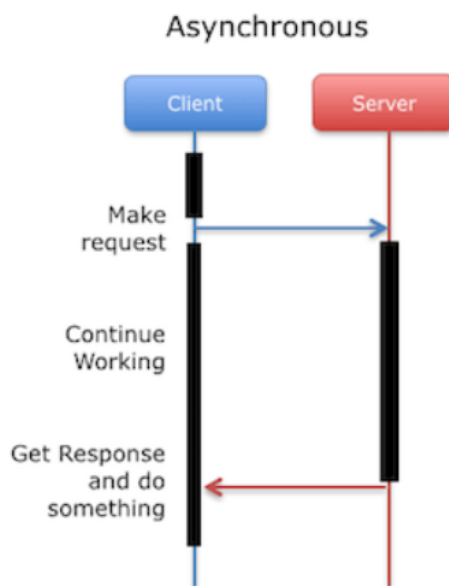
A construção de SPA's gira em torno dos seguintes conceitos, que são de extrema importância:

#### *Requisições assíncronas:*

---

Requisições assíncronas são essenciais às SPA's, pois não bloqueiam o funcionamento do site, o que possibilita uma melhoria na experiência do usuário, que pode ter a impressão de uma utilização praticamente instantânea do site, ou seja, sem muitos atrasos e com constante *feedback*.

**Figura 12 – Exemplo de requisição assíncrona**



Fonte: [Smashing Magazine](#).

Considerando o JavaScript, requisições assíncronas podem ser abordadas com as seguintes técnicas:

- **XMLHttpRequest** – precursor do padrão AJAX. Raramente são utilizadas atualmente, pois existem API's mais robustas e com código mais legível.
- **Callbacks** – funções que são passadas através de parâmetros de outras funções e executadas ao final de determinado processamento. O excesso de *callbacks* pode levar a um grave problema de manutenção de código conhecido como "*callback hell*".

Figura 13 – *Callback hell*



```

1  function hell(win) {
2    // for listener purpose
3    return function() {
4      loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5        loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6          loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7            loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8              loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9                loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                     async.eachSeries(SERIALS, function(src, callback) {
14                       loadScript(win, BASE_URL+src, callback);
15                     });
16                   });
17                 });
18               });
19             });
20           });
21         });
22       });
23     });
24   });
25 });
26 }

```

Fonte: [medium.com](https://medium.com).

- **Promise:** estrutura mais moderna que busca resolver os problemas acarretados pelo *callback hell*. Assim, ao invés de *callbacks*, programamos a execução de métodos assíncronos com a palavra-chave *then* (“então”, em inglês), que trata a execução feita com sucesso. Para erros, podemos utilizar a palavra-chave *catch*. Ainda assim, muitos consideram o código com *promises* pouco legível.

**Figura 14 – Exemplo de código JavaScript com utilização de *promise***

```
function searchWithPromise() {
    const currentUsername = inputUserGithub.value;

    fetch(`${urlGithub}${currentUsername}`)
        .then(res => {
            return res.json();
        })
        .then(userData => {
            console.log(userData);
        });
}
```

- **Async/await:** tem funcionalidade equivalente às *promises*, mas possuem uma melhor legibilidade de código. Estão presentes em versões mais modernas de JavaScript como o ES7+. De qualquer forma, o Babel já dá suporte à transpilação para código compatível. Perceba que o código abaixo, que implementa a mesma funcionalidade da figura 14 (*promise*), tem melhor legibilidade e dá a impressão/sensação de que a execução é síncrona.

**Figura 15 – Exemplo de código JavaScript com *async/await***

```
async function searchWithAsyncAwait() {
    const currentUsername = inputUserGithub.value;
    const res = await fetch(`${urlGithub}${currentUsername}`);
    const userData = await res.json();
    console.log(userData);
}
```



- **Observables:** conforme visto anteriormente nas videoaulas do capítulo 1, a utilização de implementações baseadas no padrão de projeto *Observer* permitem grande flexibilidade no tratamento de requisições assíncronas, sendo o RxJS a principal biblioteca utilizada para este fim atualmente.

### Manipulação do DOM (Document Object Model)

---

SPA's manipulam o DOM constantemente com o objetivo de entregar ao usuário uma melhor experiência através de *feedback* visual com atualizações instantâneas do estado da aplicação, por exemplo.

Sabe-se também que a manipulação direta no DOM é um processo caro (lento). Portanto, devem ser adotadas estratégias para manipulá-lo da maneira mais eficiente possível.

Considerando Vanilla JS (JavaScript puro), a manipulação do DOM pode ser feita conforme trecho de código do exemplo abaixo:

**Figura 16 – Manipulação direta do DOM com Vanilla JS**

```
var n1 = document.querySelector('#n1');
var n2 = document.querySelector('#n2');
var sum = document.querySelector('#sum');

function init() {
  n1.addEventListener('input', react);
  n2.addEventListener('input', react);

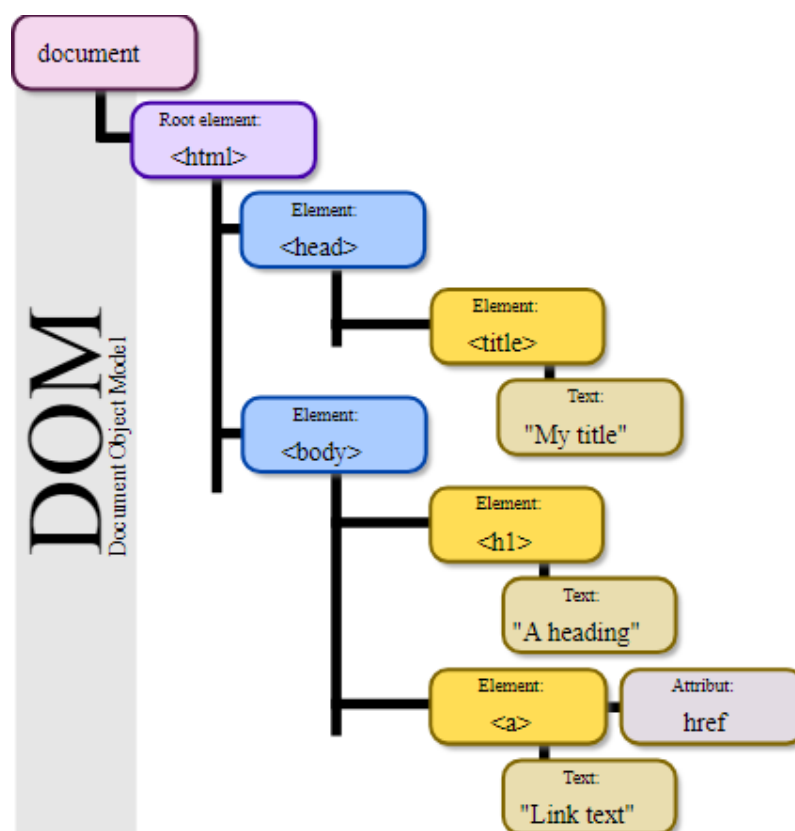
  /**
   * Já invocando react() pela
   * primeira vez.
   */
  react();
}

/**
 * Reagindo às mudanças nos inputs
 */
function react() {
  sum.value = +n1.value + +n2.value;
}
```

Os *frameworks* atuais modernos utilizam algumas estratégias eficientes para manipular o DOM, como por exemplo a utilização de um Virtual DOM, que será visto com mais detalhes posteriormente.

A figura abaixo demonstra uma abstração do DOM em formato de árvore.

Figura 17 – Representação gráfica do DOM (*Document Object Model*)



Fonte: [Wikipedia](#).

### Roteamento

SPA's de grande porte geralmente possuem mais de um módulo (separação lógica), como por exemplo "Administração", "Autenticação", "Compras", "Estoque", "Vendas", etc. Além disso, esses módulos podem ser acessados diretamente através de URL's específicas.

Considerando que as SPA's possuem em geral apenas uma página, é necessário que elas adotem alguma estratégia para "entender" essa utilização de links que, se digitados na barra de endereços, vai obrigatoriamente causar um *refresh* na página, o que, de certa forma, foge aos princípios de uma SPA.

Para esses casos, são adotadas técnicas de **roteamento** que farão internamente a troca de páginas que, no fim das contas, são trocas de componentes.

O principal detalhe que o usuário comum está acostumado é o botão “Voltar” do navegador, que deve ser também tratado pelo processo de roteamento.

Considerando os três principais *frameworks* do mercado atualmente (Angular, React e Vue), pode-se dizer que os três possuem, mesmo que não oficialmente, estratégias para o roteamento, que são:

- **Angular:** módulo próprio para roteamento, já incluso na plataforma.
- **React:** pacote [react-router](#), mantido por terceiros.
- **Vue:** pacote [vue-router](#), mantido pela própria equipe do Vue.

O tema de **roteamento** vai além do escopo da disciplina e portanto, não será abordado. Vale ressaltar que é perfeitamente possível construir SPA's mais simples sem a necessidade de roteamento.

### Ferramentas para o desenvolvimento

---

A seguir serão mostradas algumas ferramentas que serão utilizadas durante o desenvolvimento nos capítulos seguintes:

#### Node.js

---

O Node.js é um ambiente de *runtime* JavaScript compatível com os principais sistemas operacionais (Windows / Linux / MacOS), que permite a criação de servidores *web* com JavaScript, por exemplo.

O Node.js utiliza internamente a *engine* JavaScript V8, criada pelo Google em linguagem C++ multiplataforma. Esta *engine* é utilizada no Google Chrome.

Além disso, seu ecossistema de pacotes, o *npm*, é conhecido por ser o maior ecossistema de bibliotecas *open source* mundial.

Os três *frameworks* que serão estudados nesta disciplina estão hospedados no ambiente do Node.js. A instalação de cada *framework* pode ser feita através de pacotes do npm (Node Package Manager), semelhante ao *apt* do Linux.

A figura abaixo ilustra alguns comandos importantes do npm:

**Figura 18 – Comandos importantes do npm**

Comando	Descrição
<code>node -v</code>	Verifica a versão instalada do Node.js
<code>npm i nome_do_pacote -g</code>	Instalação <b>global</b> do pacote
<code>npm i nome_do_pacote</code>	Instalação <b>local</b> do pacote
<code>npm i nome_do_pacote --save</code>	Instalação <b>local</b> do pacote e registro como dependência do projeto
<code>npm i nome_do_pacote --save-dev</code>	Instalação <b>local</b> do pacote e registro como dependência do desenvolvedor
<code>npm r nome_do_pacote</code> <code>(-g --save --save-dev)</code>	Exclusão do pacote
<code>npm view nome_do_pacote</code>	Visualiza informações importantes sobre o pacote, incluindo as versões
<code>npm i -g npm</code>	Atualiza o próprio npm (que também é um pacote do Node.js)

O Node.js será utilizado nesta disciplina basicamente como a base para o desenvolvimento de aplicações com os *frameworks*. Os *frameworks* possuem diversos *scripts* criados com o Node.js, com destaque para ferramentas de *live reloading*, que consiste na atualização instantânea do site em modo de desenvolvimento para cada alteração feita no código pelo desenvolvedor.

Sendo assim, não há necessidade de nenhum detalhamento da tecnologia que será utilizada normalmente durante o desenvolvimento, sem necessidade de nenhuma configuração em específico, por exemplo.

Mais detalhes sobre o Node.js podem ser encontrados em <https://nodejs.org>.

### Microsoft Visual Studio Code<sup>1</sup>

<sup>1</sup> Para instalar o Visual Studio Code, acesse <https://code.visualstudio.com/>

Esta ferramenta *open source*, concebida pela Microsoft e também conhecida como VSCode, tem sido bastante utilizada para o desenvolvimento JavaScript recentemente, superando editores concorrentes como o SublimeText e o Atom. É um editor leve que possui recursos interessantes através de funcionalidades nativas e de *plugins*. O VSCode é compatível com Windows, MacOS e Linux. Além de JavaScript, suporta diversas outras linguagens de programação e tem integração nativa com o *git*<sup>2</sup>.

Todos os exemplos de código-fonte e aulas gravadas foram feitos no VSCode, que é a ferramenta recomendada pelo professor. Entretanto, nada impede que seja utilizado um outro editor de sua preferência.

Para melhorar a produtividade durante o desenvolvimento, é sugerida a prática e utilização das seguintes teclas de atalho:

- Alt + Shift + ↓      → duplica linhas
- Ctrl + Shift + D      → exclui linhas
- Ctrl + ← ou →      → navega mais rapidamente entre os elementos
- Alt + ↓ ou ↑      → move linhas

Para mais dicas e teclas de atalho que ajudam a melhorar a produtividade no Visual Studio Code, acesse: <https://code.visualstudio.com/docs/getstarted/tips-and-tricks>.

---

<sup>2</sup> O *git* é um *software* livre para controle de versões de código. Os *apps* disponibilizados pelo professor são controlados pelo *git* e hospedados no *Gitlab*. É extremamente recomendável que os desenvolvedores estudem e dominem esta tecnologia. Para mais detalhes, acesse <https://git-scm.com/>.

## Capítulo 3. Introdução ao Angular

### Angular

O Angular é um *framework* desenvolvido por colaboradores do Google para a criação de SPA's (*Single Page Applications*).

Atualmente, o Angular é considerado não somente um *framework* e sim uma **plataforma de desenvolvimento**, pois fornece muito mais funcionalidades do que um *framework*, em geral.

- Site oficial: <https://angular.io>.
- Repositório no Github: <https://github.com/angular/angular>.

### Instalação e configuração

A maneira mais simples de se instalar o Angular, considerando o Node.js já instalado, um terminal de comandos aberto e acesso em qualquer pasta do seu computador, é a seguinte:

**Figura 19 – Comando de instalação global do Angular**

```
d:\igti
λ npm install -g @angular/cli
```

O comando acima instala o Angular globalmente, ou seja, permite que sejam criados projetos a partir de qualquer pasta de seu computador.

Para criar um projeto, defina uma pasta base e execute o seguinte comando:

**Figura 20 – Criação de projetos com Angular**

```
d:\igti\projetos-angular
λ ng new MeuPrimeiroProjetoAngular
```

Com esse comando, o Angular vai montar um *scaffolding* de um projeto base e efetuar o download de todas as dependências necessárias do projeto.

Para executar o projeto, acesse a pasta do mesmo e utilize o seguinte comando:

**Figura 21 – Executando um projeto Angular**

```
d:\igti\projetos-angular\MeuPrimeiroProjetoAngular (master)
λ ng serve -o
```

O parâmetro ‘-o’ é opcional e abre o navegador padrão com uma aba em <http://localhost:4200/>, após o carregamento do projeto em memória. Sem a utilização do parâmetro, o desenvolvedor deve abrir a url do servidor de desenvolvimento manualmente.

É importante salientar que todos os *frameworks* que serão vistos nesta disciplina possuem suporte a um servidor de desenvolvimento com funcionalidade de *live-reloading*, que é geralmente provido pelo Webpack. Esta disciplina não entrará em detalhes sobre o Babel (transpilação de JavaScript) e Webpack, pois fogem do escopo. De maneira geral, eles já estão muito bem configurados em cada *framework* e raramente são necessárias alterações nessas configurações.

A estrutura de um projeto Angular é, na minha opinião, a mais complexa dentre os três *frameworks*.

Para mais detalhes sobre como criar e codificar projetos com Angular, verifique as videoaulas dos capítulos 3 e 6.



## Características

---

Atualmente, o Angular se encontra na versão 10.x e foi totalmente reescrito a partir da versão 2. Assim, ficou convencionado que a versão 1.x seria denominada AngularJS ou Angular.js para não haver confusão na busca de documentação, por exemplo.

É esperado que o AngularJS, que está atualmente na versão 1.8.x, seja descontinuado em breve. Não é recomendado estudá-lo, a não ser que seja necessário dar manutenção em algum projeto legado feito com essa tecnologia.

O Angular geralmente lança novas versões *major* a cada seis meses. A seguir serão listadas algumas características importantes sobre o Angular a partir da versão 2:

- Adesão ao *semver* (*Semantic Versioning*), que controla melhor as versões, evitando *breaking changes*.
- Utilização de TypeScript como linguagem de programação padrão. Mais detalhes sobre o TypeScript serão vistos a seguir.
- Para auxiliar na organização e reutilização do código, além da separação de responsabilidades, o Angular possui as seguintes entidades:
  - Módulos
  - Componentes
  - Metadados
  - Diretivas
  - *Pipes*
  - *Templates*
  - *Services*

As figuras a seguir ilustram melhor esses conceitos. Mais detalhes podem ser vistos nas videoaulas do capítulo 3.

**Figura 24 – Módulos e componentes do Angular**

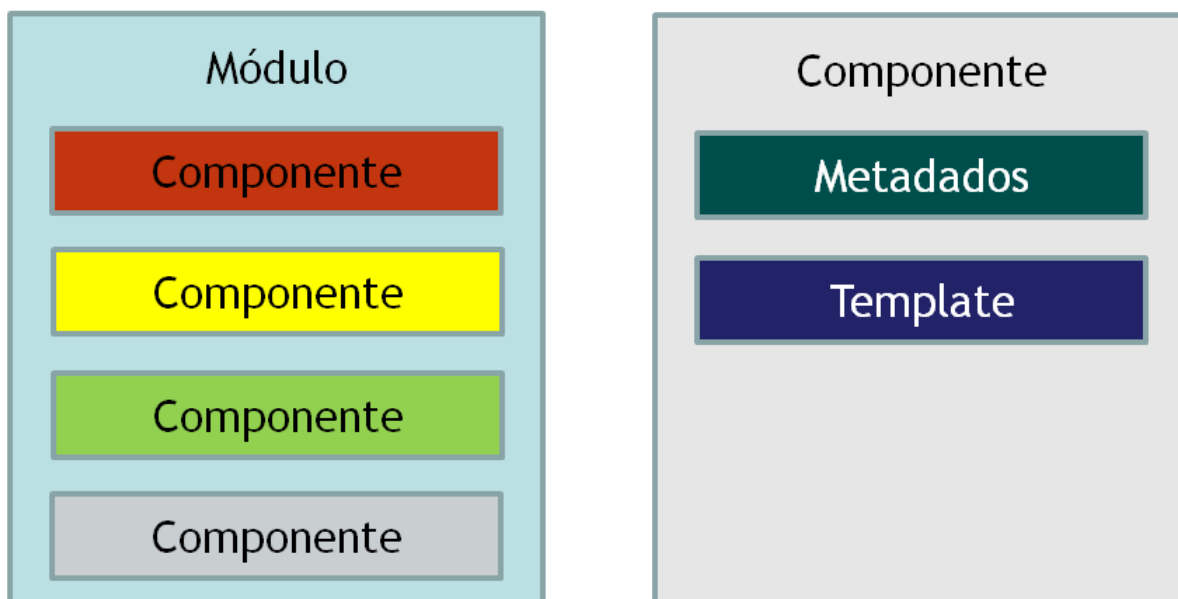
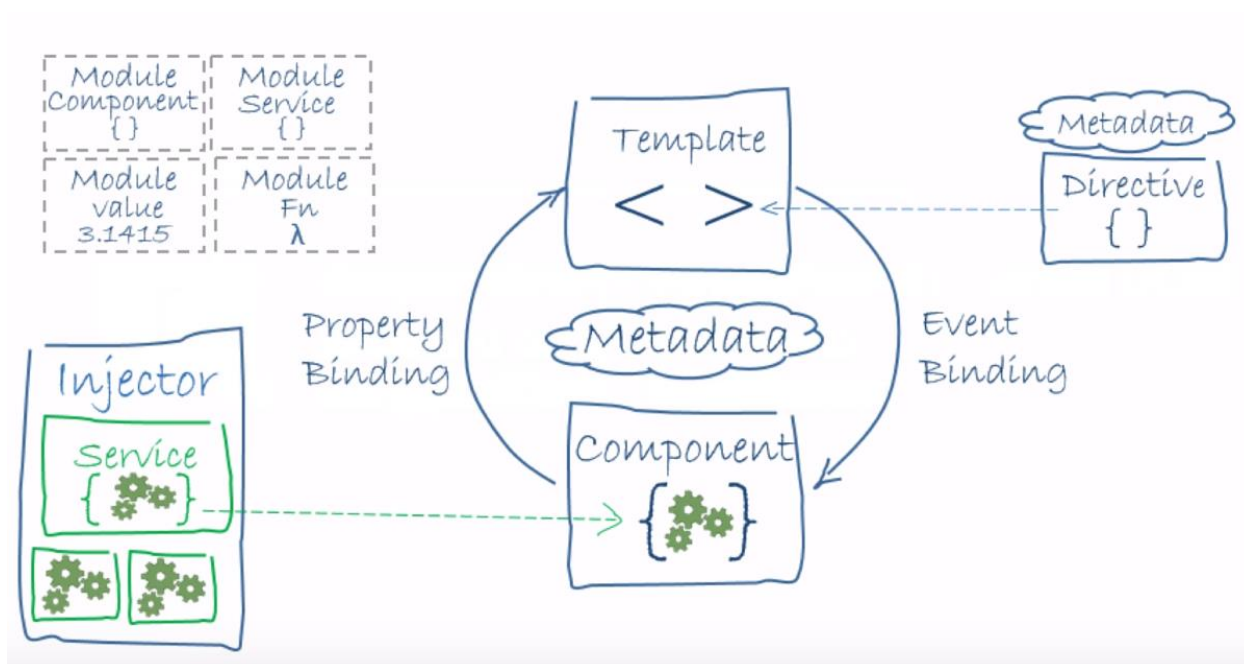


Figura 25 – Arquitetura do Angular



Fonte: <https://angular.io/guide/architecture>.

Sobre as figuras acima:

- **Component:** classe que agrupa uma lógica bem definida. Possui **metadados** que identificam a classe como um componente para o Angular. Esses metadados conectam a classe a um *template*.
- **Template:** contém o código HTML vinculado ao componente. A comunicação entre componente e *template* se dá nos dois sentidos.
- Quando o sentido da comunicação é **do componente ao template**, há o **Property Binding** (vínculo de propriedades), onde um atributo do componente pode ser exibido no *template* com a notação de duas chaves à esquerda e duas chaves à direita, também conhecido como *double curly braces* – {{ atributo\_do\_componente }}. Em alguns casos o *property binding* pode vir acompanhado de um ou mais **pipes**, que são estruturas que encapsulam funções transformadoras de dados. Os *pipes* sempre são precedidos pelo

caractere '|', que é conhecido como "*pipe*". O Angular já possui diversos *pipes* como o *date* e *upperCase*.

- Uma outra notação disponível de **property binding** é a de colchetes – [ atributo\_HTML ], que normalmente é utilizada em atributos de elementos HTML. Quando um atributo é decorado com colchetes, o valor passa a ser código JavaScript e não somente uma *string*. Assim, é possível implementar lógicas simples possibilitando aplicar **reatividade**.
- Quando o sentido da comunicação é do template ao componente, geralmente ocorre o **Event Binding**, que se refere aos eventos (cliques do mouse, edição de campos via teclado etc.) e possui a notação de parênteses – (nome\_do\_evento). Com essa notação é possível criar novos eventos e emití-los no sentido inverso, ou seja, de componentes filhos para o componente pai.
- Os templates podem conter **diretivas** que auxiliam o desenvolvedor na criação do *layout* e disposição dos atributos do componente. As principais diretivas são o *\*ngIf* (estrutura de decisão) e *\*ngFor* (estrutura de repetição). O desenvolvedor também pode criar suas próprias diretivas.
- **Serviços** são classes especiais que modelam regras de negócio e/ou comunicação remota, auxiliando na organização do código e separação de responsabilidades. Em geral, os serviços são instanciados nos componentes através de **injeção de dependência**, onde o próprio Angular *sabe* como instanciá-los, retirando uma responsabilidade a mais do desenvolvedor.
- Todos os elementos acima podem ser agrupados em **módulos**, que podem conter um ou mais **componentes**. Esse conjunto de módulos interligados de forma organizada pode constituir um sistema Angular complexo e ao mesmo tempo mais fácil de ser mantido.
- O Angular também possui o **two-way data binding**, que geralmente é aplicado em *tags* do tipo `<input />`. Nesses casos, a escrita de dados por parte do usuário é instantaneamente refletida no atributo correspondente do componente. A notação é `[(ngModel)]`, que também é conhecida como "*banana in a box*". Para que o *two-way data binding* funcione, é necessário importar

*FormsModule* na aplicação, que não é inserida por padrão durante a criação de um app Angular.

**Figura 26 – Exemplo de *template* com Angular utilizando *property binding* e *event binding***

```
<li
  (click)="changeTab('repos')"
  [class]="currentTab === 'repos' && 'active'">
  Repositórios {{ getReposCount() }}
</li>
```

**Figura 27 – Exemplo de *template* com Angular utilizando *two-way data binding***

```
<div>
  <!-- Exemplo de two-way data-binding -->
  <label>
    Nome:
    <input
      [(ngModel)]="hero.name"
      placeholder="name"
    >
  </label>
</div>
```

## TypeScript

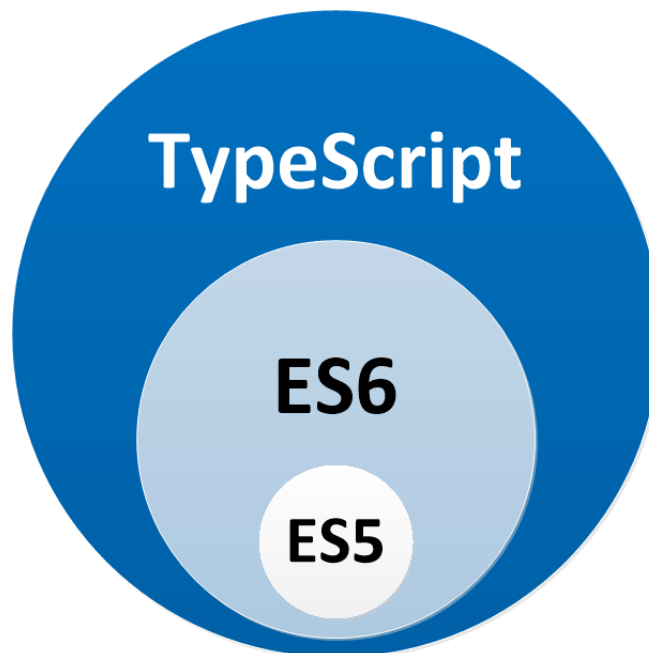
O *TypeScript* é uma linguagem de programação criada pela Microsoft. É *open source* e pode ser encontrada em <https://github.com/Microsoft/TypeScript>.

O *TypeScript* é um *superset* do *JavaScript* e seu principal objetivo é servir de *açúcar sintático* (*syntactic sugar*) ao *JavaScript* através do provimento de novas

funcionalidades, em especial a tipagem de dados (característica encontrada em linguagens de *backend* como o Java e o C#).

Além disso, o TypeScript provê todas as funcionalidades já existentes em novas versões do JavaScript ainda não homologadas oficialmente (EcmaScript 2015 (ES2015/ES6)). O site oficial do TypeScript é o <https://www.typescriptlang.org/>. O Angular passou a adotar o TypeScript por padrão a partir da versão 2. A imagem abaixo ilustra melhor o TypeScript em relação ao JavaScript (ES5 e ES6):

**Figura 28 – TypeScript**



**Fonte:** [medium.com](https://medium.com).

A seguir, são elencadas algumas características importantes do TypeScript:

- Possui verificação de tipos em tempo de compilação, o que pode acarretar em redução de erros em produção.
- O custo de manutenção de um projeto pode diminuir.
- Possui suporte a *arrays* (vetores) tipados e *decorators* (muito utilizados pelo Angular).

- Possui um tipo especial para se referir a qualquer valor: *any*.
- O *açúcar sintático* provê um melhor suporte à OO (Orientação a Objetos) com atributos e métodos públicos e privados, por exemplo.
- O funcionamento do TypeScript se dá através da transpilação do seu código para JavaScript, de forma que o resultado fique compatível com a maioria dos navegadores atuais.

Para mais detalhes sobre o TypeScript, acompanhe as videoaulas do capítulo 3.

---

### Implementação simples com Angular – angular-intervalos

---

Para todos os *frameworks* abordados por esta disciplina, será criada, da maneira mais simples possível<sup>3</sup>, uma aplicação **reativa** que, a partir de um número escolhido pelo usuário, como por exemplo o 34, exibe os números divisíveis por 2 a 9 considerando o intervalo de 2 a 34, conforme imagem abaixo:

---

<sup>3</sup> É fato que sempre existirão melhores formas de implementação, refatoramento do código etc. Mas o foco neste caso é a implementação mais simples considerando um desenvolvedor novato aprendendo a tecnologia.

**Figura 29 – Aplicação a ser implementada**

## Reatividade com intervalos de números

Contador

34

Números divisíveis por 2: 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34

Números divisíveis por 3: 3 6 9 12 15 18 21 24 27 30 33

Números divisíveis por 4: 4 8 12 16 20 24 28 32

Números divisíveis por 5: 5 10 15 20 25 30

Números divisíveis por 6: 6 12 18 24 30

Números divisíveis por 7: 7 14 21 28

Números divisíveis por 8: 8 16 24 32

Números divisíveis por 9: 9 18 27

### Implementando o template

Para não nos preocuparmos muito com CSS e estilização, inclua seguinte o *link* do CSS do Materialize<sup>4</sup> no arquivo index.html, conforme imagem abaixo:

- <https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css>

**Figura 30 – Inclusão do Materialize CSS no projeto Angular**

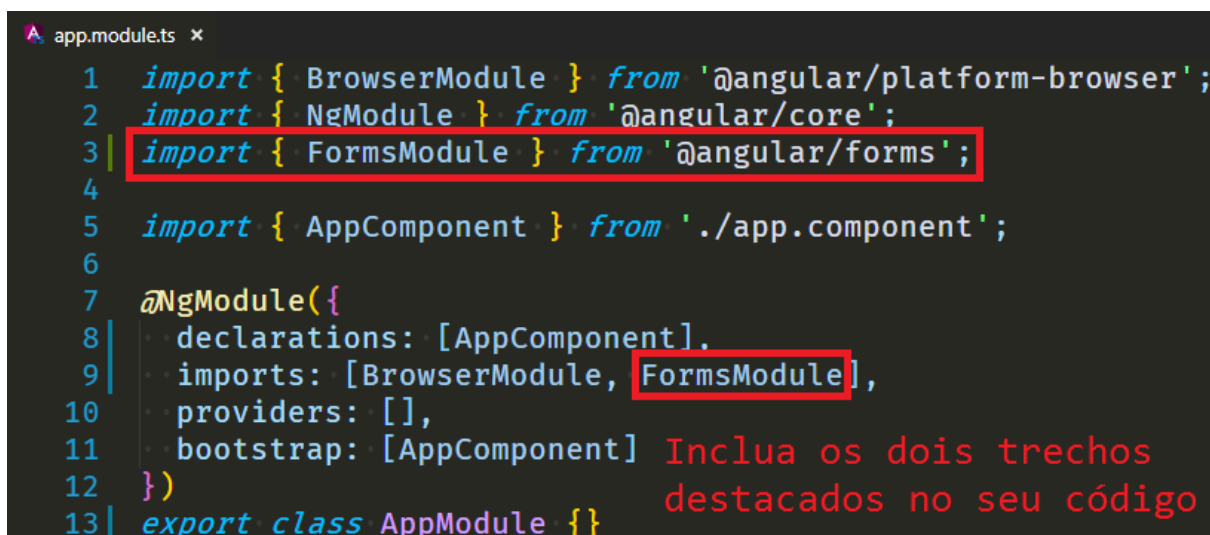
```
index.html x
1 <html lang="en">
2   <head>
3     <meta charset="utf-8">
4     <title>AngularIntervalos</title>
5     <base href="/">
6
7     <meta
8       name="viewport"
9       content="width=device-width, initial-scale=1"
10    >
11    <link rel="icon" type="image/x-icon" href="favicon.ico">
12    <!-- Materialize -->
13    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css">
14  </head>
15  <body>
16    <app-root></app-root>
17  </body>
18 </html>
```

<sup>4</sup> Para mais detalhes sobre o Materialize, acesse <https://materializecss.com/getting-started.html>.



No caso do Angular, precisaremos trabalhar com *two-way data binding*, que não vem ativado por padrão. Por isso, devemos alterar o arquivo *app.module.ts* com o seguinte código, que irá incluir *FormsModule* no projeto:

**Figura 31 – Incluindo *FormsModule* no projeto**



```

1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3  import { FormsModule } from '@angular/forms';
4
5  import { AppComponent } from './app.component';
6
7  @NgModule({
8    declarations: [AppComponent],
9    imports: [BrowserModule, FormsModule],
10   providers: [],
11   bootstrap: [AppComponent]
12 })
13 export class AppModule {}
  
```

Inclua os dois trechos destacados no seu código

Em seguida, acesse o arquivo *app.component.html* e inclua o seguinte código:

**Figura 32 – Código do *template***

```

1  <h3>
2  Reatividade com intervalos de números - Angular
3  </h3>
4  <div>
5    <div>
6      <label>
7        Contador
8        <input
9          type="number"
10         min="1"
11         max="200"
12         [(ngModel)]="currentValue" Two-way data binding
13       >
14     </label>
15   </div>
16   <ul> Iterando sobre os números 2 a 9
17     <li *ngFor="let divisor of divisors">
18       Números divisíveis por {{ divisor }}:
19       <span *ngFor="let number of getDivisiveisPor(divisor)">{{ number + ' ' }}</span>
20     </li> Iterando sobre os divisores do número definido pelo usuário
21   </ul>
22 </div>
23

```

### Implementando o componente

Analizando o código do template, é possível perceber que vários trechos ainda não existem na aplicação. Esses trechos devem ser implementados em *app.component.ts*. Portanto, deixe esse arquivo da seguinte forma:

Figura 33 – Código do componente

```

1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    divisors = [2, 3, 4, 5, 6, 7, 8, 9];
10   currentValue = 1;  Atributos monitorados
11
12   getDivisiveisPor(num) {
13     const numbers = [];
14
15     for (let i = 1; i <= this.currentValue; i++) {
16       if (i % num === 0) {
17         numbers.push(i);
18       }
19     }  Método que itera e obtém os
20       números divisíveis por x
21     return numbers;
22   }
23 }

```

### Entendendo a implementação da aplicação

Analisando *app.component.html*, que representa o template, pode-se perceber a utilização de *two-way data-binding* na linha 12, que irá monitorar o valor de *currentValue*. O usuário, ao alterar o valor do *input* irá alterar automaticamente o valor de *currentValue* que fará com que os componentes que o observam **reajam**.

A **reatividade** pode ser percebida nas linhas 17 a 22, que renderizam uma lista não-ordenada com os divisores. Para isso, foram utilizadas duas diretivas **\*ngFor**. A primeira diretiva (linha 18) itera sobre os números 2 a 9, que estão fixos no código do componente. A segunda (linha 20) itera sobre os valores de 1 a

*currentValue*, exibindo somente os que possuem divisão inteira. As implementações estão no componente.

Analisando *app.component.ts*, é possível visualizar os atributos monitorados nas linhas 9 e 10. O método *getDivisiveisPor(num)* está implementado entre as linhas 12 e 22 e contém a lógica principal da aplicação.

Perceba que em nenhum momento nos preocupamos em manipular o DOM para exibir e/ou excluir valores. Tudo isso foi feito pelo Angular de forma automática, reativa e eficiente!

O professor irá postar o código-fonte desta aplicação no fórum da disciplina.

Para mais detalhes de implementação de aplicações com Angular, consulte as videoaulas dos capítulos 3 e 6.

## Capítulo 4. Introdução ao React

---

### React

---

O React foi criado por colaboradores do Facebook e se denomina uma biblioteca JavaScript para construção de interfaces com o usuário. Foi inicialmente concebido para resolver um problema do Facebook de manter o estado das notificações que os usuários recebiam.

- Site oficial: <https://reactjs.org/>.
- Repositório no Github: <https://github.com/facebook/react>.

### Instalação e configuração

---

Assim como o Angular, o React utiliza diversos pacotes do Node.js e necessita de diversas configurações de transpilação e empacotamento, principalmente se for utilizado o JSX (será visto posteriormente).

Para resolver esse problema que poderia afastar entusiastas e principalmente novos desenvolvedores, foi criada uma ferramenta para simplificar o *scaffolding* de um novo projeto, denominada *create-react-app*. Todos os projetos de React desta disciplina a utilizam. Para instalá-la, faça o seguinte:

**Figura 34 – Instalação do *create-react-app***

```
C:\igti  
λ npm i -g create-react-app
```

O comando acima instala a ferramenta globalmente, ou seja, permite que sejam criados projetos a partir de qualquer pasta de seu computador.

Para criar um projeto, defina uma pasta base e execute o seguinte comando:

**Figura 35 – Criação de projetos com o *create-react-app***

```
C:\igti\projetos-react  
λ create-react-app meu-primeiro-projeto-react
```

Com esse comando, o *create-react-app* vai montar um *scaffolding* de um projeto base e efetuar o download de todas as dependências necessárias do projeto.

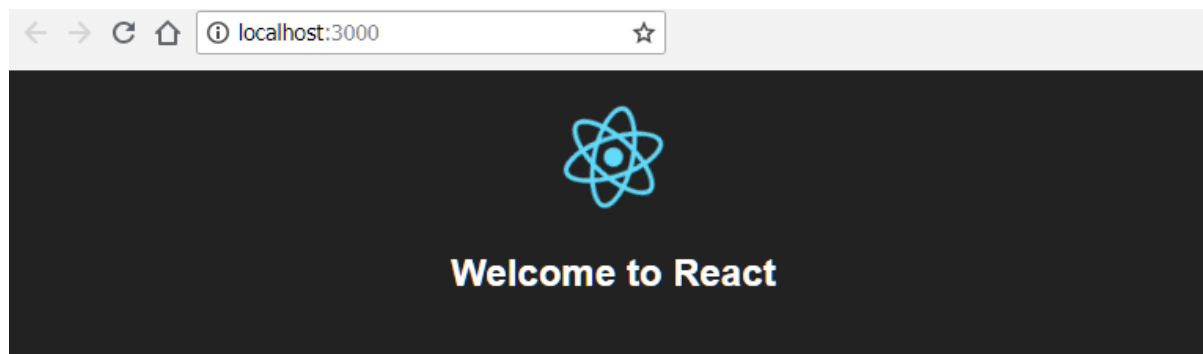
Por padrão o React utiliza o *yarn*, que é uma ferramenta também criada pela equipe do Facebook que funciona como uma alternativa ao *npm*.

O servidor de desenvolvimento do React é executado, por padrão, na porta 3000. Para executar o projeto, acesse a pasta do mesmo e utilize o seguinte comando:

**Figura 36 – Executando um projeto com o *create-react-app***

```
C:\igti\projetos-react\meu-primeiro-projeto-react
λ yarn start
```

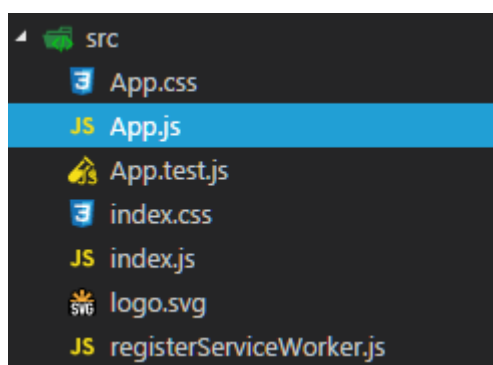
**Figura 37 – Projeto inicial gerado pelo *create-react-app***



To get started, edit `src/App.js` and save to reload.

A estrutura inicial de um projeto React é considerada bem simples. Segue abaixo os principais arquivos onde o desenvolvedor vai trabalhar:

**Figura 38 – Principais arquivos iniciais de um projeto React**



Para mais detalhes sobre como criar e codificar projetos com React, verifique as videoaulas dos capítulos 4 e 6.

## Características

---

Atualmente, o React se encontra na versão 16.x e foi totalmente reescrito internamente após a versão 15, sem afetar os projetos. A seguir são listadas algumas características importantes sobre o React:

- **Componentizável:** o React também preza pela criação de componentes, assim como grande parte dos *frameworks* de JavaScript modernos.
- **Declarativa:** seguindo os princípios do desenvolvimento reativo, a criação de componentes é bastante declarativa (ao invés de imperativa), o que faz com que o React **reaja** a mudanças no estado da aplicação de forma **eficiente**.
- **Virtual DOM:** a manipulação do DOM é de responsabilidade do Virtual DOM, que cria uma estrutura em memória do DOM e só efetua as atualizações realmente necessárias, o que garante melhor desempenho e, conseqüentemente, melhor experiência do usuário.
- **One-way data flow:** o React recomenda que concentremos o estado da aplicação no componente pai. Os filhos recebem dados do estado através de propriedades (*props*), que são somente leitura. Esse processo faz com que a lógica de alteração do estado se concentre em somente um componente, o que acarreta em menos *bugs* na aplicação.
- **Learn Once, Write Anywhere:** o React dá suporte ao desenvolvimento em diversas outras plataformas, com destaque para o React Native, que suporta por padrão a criação de *apps* nativos nas plataformas Android e iOS. Entendendo bem o funcionamento do React para *web* faz com que se aprenda a desenvolver para outras plataformas mais facilmente.
- **JavaScript moderno (ES6+):** O React, ao contrário do Angular e do Vue, não possui uma DSL (*Domain Specific Language*), que é utilizada



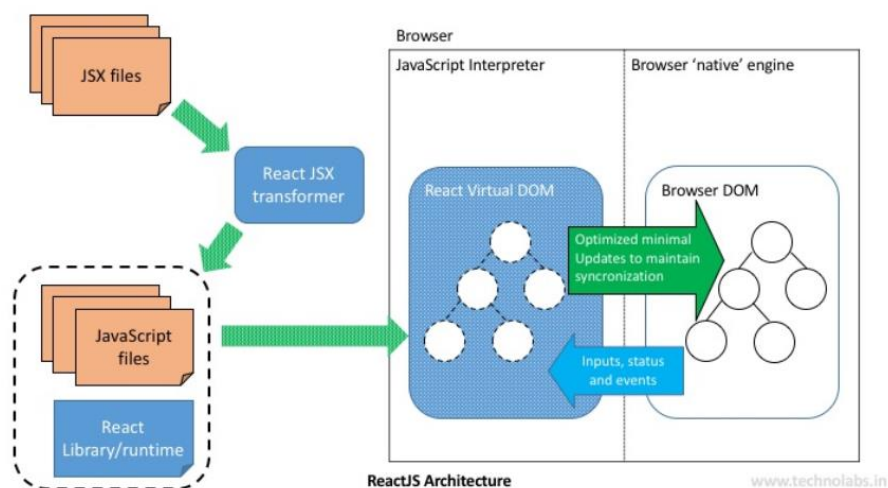
em diretivas (\*ngIf, \*ngFor), por exemplo. Com o React, basta que o desenvolvedor aprenda JavaScript e conheça a arquitetura do React. Isso permite mais flexibilidade.

- **JSX (JavaScript XML):** O React suporta o JSX para facilitar a criação de componentes. A escrita com JSX torna o código muito semelhante ao HTML. Entretanto, é necessária a utilização obrigatória do Babel para transpilar o código JSX tornando a aplicação compatível aos navegadores. Isso é configurado automaticamente com o *create-react-app*.

## Arquitetura do React

A imagem abaixo ilustra a arquitetura de aplicações com React:

**Figura 39 – Arquitetura de um projeto React**



Fonte: [biznomy.com](http://biznomy.com).

Analisando a imagem da esquerda para a direita e de cima para baixo, é possível perceber o seguinte:

1. É feita uma transformação (transpilação) de componentes feitos em JSX para código JavaScript, que junto ao código do React propriamente dito são hospedados com a aplicação (em produção).
2. Durante a execução da aplicação, o React cria o VirtualDOM que monitora o DOM e só efetua a manipulação quando necessário e de forma eficiente.

### O arquivo App.js

A imagem a seguir mostra os principais componentes de App.js, que é criado automaticamente com o *create-react-app*:

Figura 40 – Arquivo App.js

```

1  import React, { Component } from 'react';  Importação de módulos
2  import logo from './logo.svg';              e assets
3  import './App.css';
4
5  class App extends Component {  Classes React devem herdar de Component
6    render() {                    Método de renderização de um componente
7      return (
8        <div className="App">
9          <header className="App-header">
10             <img src={logo} className="App-logo" alt="logo" />  Código JSX
11             <h1 className="App-title">Welcome to React</h1>
12           </header>
13           <p className="App-intro">
14             To get started, edit <code>src/App.js</code> and save to reload.
15           </p>
16         </div>
17       );
18     }
19   }
20
21   export default App;  Exportação do componente para ser utilizado
22                       externamente
  
```

Algumas observações sobre a imagem acima:

- Perceba que o método *render* deve obrigatoriamente retornar algo. Esse retorno deve ser de **apenas um elemento**. Assim, para agrupar diversos elementos, pode ser utilizado um elemento *container*, como a

<div>. O React possui o componente React.Fragment, que também faz esse papel.

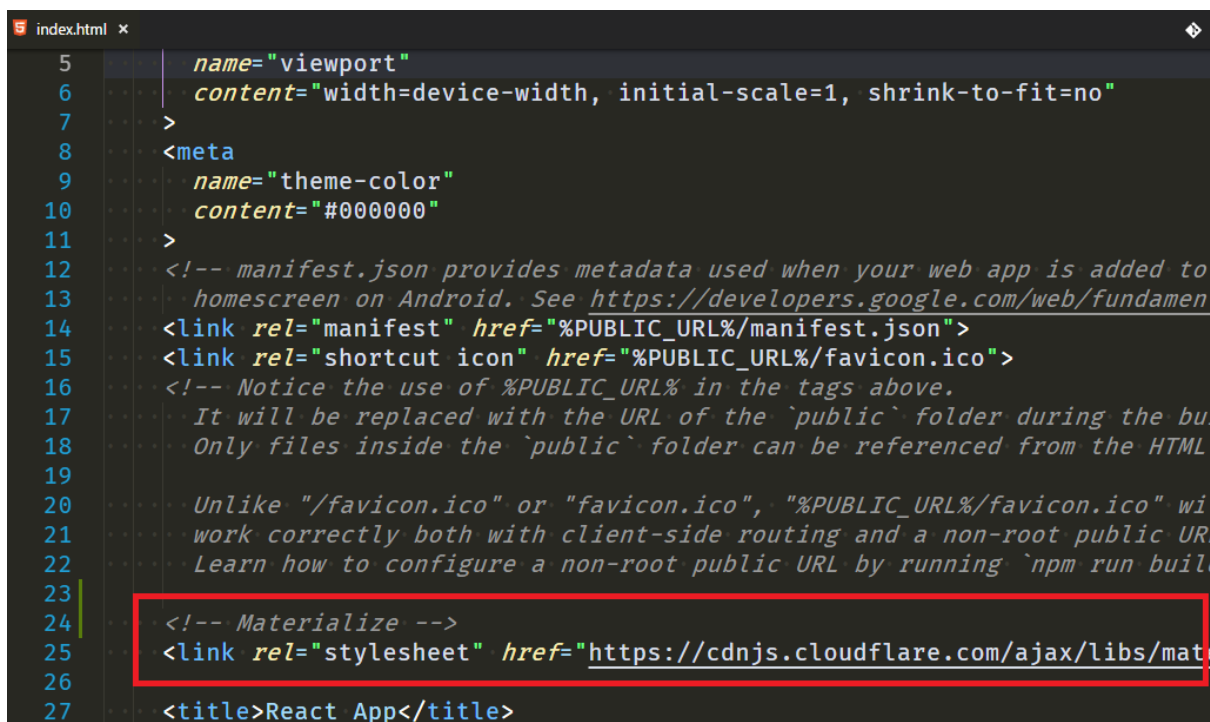
- O trecho de código `src={logo}` indica que o caminho da imagem aponta para `./logo.svg`. Assim, utilizamos `{ }` para representar instruções JavaScript dentro de JSX.
- Perceba que o código JSX é bastante semelhante ao HTML. Uma das diferenças é a utilização de `className` ao invés de `class`, já que esta última é uma palavra reservada do JavaScript. É importante lembrar que JSX é JavaScript e não HTML. Outra diferença é a inclusão de `</>` em `tags` que não possuem pares de fechamento, como `<input />`. Atualmente, esse tipo de fechamento é desnecessário no HTML, mas obrigatório em JSX.
- Para economizar texto e linhas, a linha 21 pode ser excluída e a linha 5 pode ser escrita como: **`export default class App...`**

### Implementação simples com React – react-intervalos

---

Primeiramente, incluímos o `link` do Materialize no arquivo `public/index.html` de forma semelhante ao que foi feito no projeto com Angular, conforme imagem abaixo:

**Figura 41 – Inclusão do Materialize na aplicação React**



```

5      name="viewport"
6      content="width=device-width, initial-scale=1, shrink-to-fit=no"
7    >
8    <meta
9      name="theme-color"
10     content="#000000"
11   >
12   <!-- manifest.json provides metadata used when your web app is added to
13   homescreen on Android. See https://developers.google.com/web/fundamen
14   <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
15   <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
16   <!-- Notice the use of %PUBLIC_URL% in the tags above.
17   It will be replaced with the URL of the `public` folder during the bu
18   Only files inside the `public` folder can be referenced from the HTML
19
20   Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" wi
21   work correctly both with client-side routing and a non-root public UR
22   Learn how to configure a non-root public URL by running `npm run buil
23
24   <!-- Materialize -->
25   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/mat
26
27   <title>React App</title>
  
```

No caso do React, para implementar o exemplo de divisores, como foi feito com Angular, basta apenas um arquivo, que será o App.js. É claro que é possível quebrar o código em diversos componentes, mas esse não é o foco nesse momento.

A figura a seguir mostra a implementação em React.

Figura 42 – Implementação com React

```

1  import React, { Component, Fragment } from 'react';
2
3  export default class App extends Component {
4    constructor() {
5      super();
6
7      this.state = {
8        currentValue: 1
9      };
10
11     this.divisors = [2, 3, 4, 5, 6, 7, 8, 9];
12   }
13
14   getDivisiveisPor(number) {
15     const numbers = [];
16     for (let i = 1; i <= this.state.currentValue; i++) {
17       if (i % number === 0) {
18         numbers.push(i);
19       }
20     }
21     return numbers;
22   }
23
24   render() {
25     return (
26       <Fragment>
27         <h3>Reatividade com intervalos de números - React</h3>
28         <div>
29           <div>
30             <label>
31               Contador
32               <input
33                 type="number"
34                 min="1"
35                 max="200"
36                 value={this.state.currentValue}
37                 onChange={event =>
38                   this.setState({ currentValue: event.target.value })
39                 }
40             />
41             </label>
42           </div>
43           <ul>
44             {this.divisors.map(divisor => {
45               return (
46                 <li key={divisor}>
47                   Números divisíveis por {divisor}:{' '}
48                   {this.getDivisiveisPor(divisor).map(number => {
49                     return <span key={number}>{number + ' '}</span>;
50                   })}
51                 </li>
52               );
53             })}
54           </ul>
55         </div>
56       </Fragment>
57     );
58   }
59 }
60

```

## Entendendo a implementação da aplicação

---

Na linha 1, é feita a importação da biblioteca do React, com destaque para `Fragment`, que será utilizada para retornar apenas um elemento no método `render()`.

Na linha 3 declaramos a classe `App` herdando de `Component`, que é uma classe do React.

Nas linhas 4 a 12 criamos o construtor da classe que, como herda de `Component`, exige a invocação de `super()`. Além disso, criamos o estado da aplicação com `this.state` e instanciamos o atributo `this.divisors` que, sendo “fixo”, não necessita estar vinculado ao estado. Em geral, `this.state` recebe um objeto com um ou vários valores. O React trata `this.state` de forma especial, tornando-o reativo. No caso dessa aplicação, o único dado que necessita ser realmente reativo é, de fato, `currentValue`.

Nas linhas 14 a 22, temos o método `getDivisiveisPor(number)`, cuja implementação é praticamente idêntica à que foi feita com Angular, com a única exceção de utilizarmos `this.state.currentValue` em vez de `this.currentValue` (que não existe na aplicação React).

Nas linhas 24 a 59, temos o principal método da aplicação, o `render()`, que vai fazer a renderização do conteúdo. Este também é um método com tratamento especial do React. Perceba que `render()` retorna apenas um elemento (`<Fragment>`) que é um elemento *container* que agrupa todo o conteúdo com JSX.

Nas linhas 36 a 39, fazemos uma implementação semelhante ao *two-way data-binding* do Angular, ou seja, vinculamos o estado ao `value` do `<input>` e também definimos um novo valor do estado no evento `onChange` do `<input>`. Como o React não possui DSL (*Domain Specific Language*) devemos realizar esse tipo de implementação manualmente. Perceba na implementação de `onChange` que alteramos o estado de forma imutável através de `setState`, enviando um novo objeto como parâmetro. Isso acarreta em uma nova execução de `render()`, que então vai alterar os elementos que **observam** *state*, efetuando então a **reatividade**. Com o apoio do VirtualDOM, a execução de `render` é a mais eficiente possível e é feita de forma transparente ao desenvolvedor.

Nas linhas 43 a 54, é feita a renderização dos elementos em tela de forma reativa. Essa renderização é semelhante à do Angular com a exceção de não possuímos DSL no React (no caso, `*ngFor`). Por isso realizamos a implementação com ES6+ e a função `map`, que em essência realiza alguma transformação de dados e **retorna** um novo elemento (imutabilidade). No JSX, para invocarmos expressões JavaScript, basta utilizar a notação de *chaves (single braces)* - `{ }`. As cores das chaves facilitam a identificação do escopo de cada uma. Outro detalhe importante é a utilização de `key`, que é um atributo recomendado pelo React que auxilia a reconciliação dos elementos no VirtualDOM. Se não utilizarmos `key` o React emite um alerta no console do navegador.

O código-fonte deste projeto será disponibilizado pelo professor no fórum da disciplina.

Para mais detalhes de implementação de aplicações com React, consulte as videoaulas dos capítulos 4 e 6.

## Capítulo 5. Introdução ao Vue

### Vue

O Vue é um *framework* criado pelo chinês Evan You para a criação de SPA's (*Single Page Applications*). O Vue é atualmente mantido por uma comunidade de desenvolvedores.

- Site oficial: <https://vuejs.org/>.
- Repositório no Github: <https://github.com/vuejs/vue>.

### Instalação e configuração

Assim como o Angular, o Vue possui um CLI (*Command Line Interface*), com diversas opções de criação de projetos. Será mostrada a forma mais útil e simples possível.

Para instalar o Vue CLI<sup>5</sup> globalmente (forma recomendada), execute o comando abaixo a partir de qualquer pasta de um terminal de comandos qualquer:

**Figura 43 – Instalação do Vue CLI**

```
d:\projetos
λ npm install -g vue-cli
```

Para criar um projeto, defina uma pasta base e execute o seguinte comando, que irá acionar um assistente e fazer algumas perguntas sobre o projeto, conforme imagem abaixo:

---

<sup>5</sup> Por questões de compatibilidade com o material das vídeo aulas, será utilizada a versão 2.x do Vue CLI. A versão 3 pode ser instalada através do pacote @vue/cli.



**Figura 44 – Criação de um projeto com o Vue CLI**

```
d:\projetos
λ vue init webpack-simple meu-primeiro-projeto-vue

? Project name meu-primeiro-projeto-vue
? Project description A Vue.js project
? Author Raphael Gomide
? License MIT
? Use sass? No

vue-cli · Generated "meu-primeiro-projeto-vue".

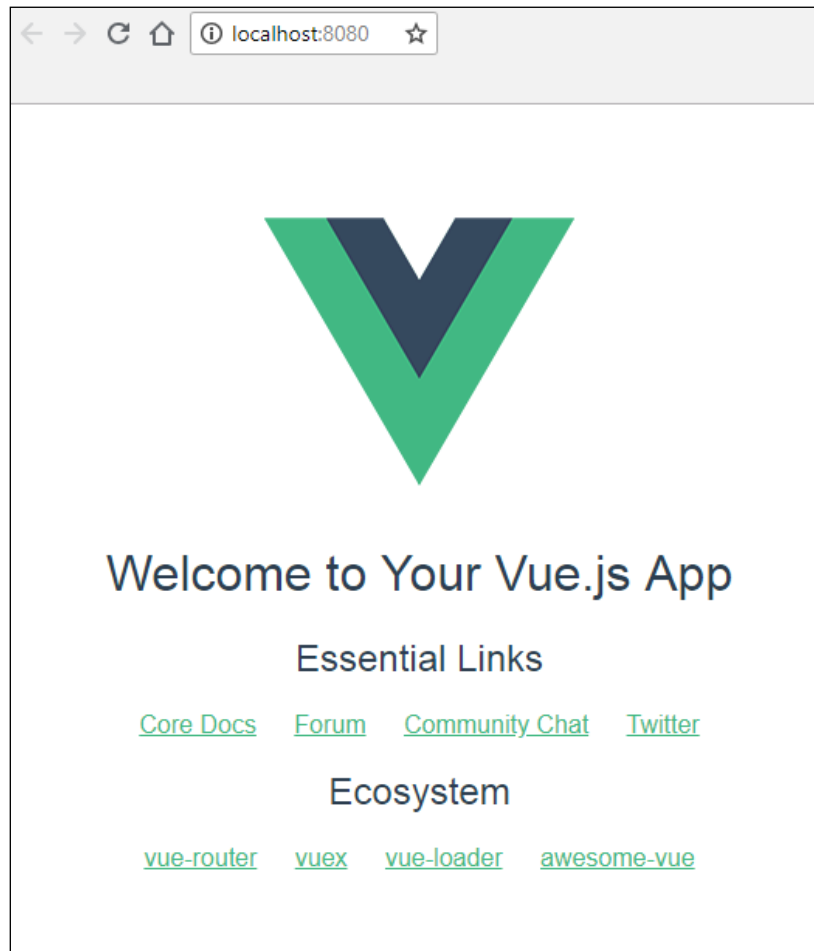
To get started:

  cd meu-primeiro-projeto-vue
  npm install
  npm run dev
```

**Observação:** o parâmetro *webpack-simple* já cria o projeto com capacidade de execução através de um servidor de desenvolvimento, assim como o Angular e o React já fazem por padrão. Esta é a forma mais simples e recomendada para quem está começando a utilizar o Vue com apoio do Node.js. Outro detalhe é que, ao contrário do Angular e do React, o Vue CLI não efetua o download das dependências automaticamente. O próprio comando instrui o desenvolvedor a fazer isso com *npm install*.

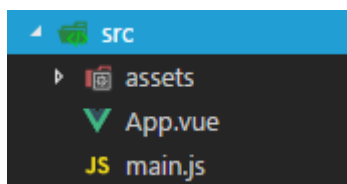
Para executar o projeto Vue, acesse a pasta do projeto e, após realizar o download das dependências com “npm install”, execute o comando “npm run dev”, conforme instrução na imagem acima. Por padrão, o servidor de desenvolvimento do Vue é executado na porta 8080:

**Figura 45 – Tela inicial de um projeto Vue**



A estrutura de um projeto Vue é a mais simples dentre os três *frameworks*. Segue abaixo os principais arquivos onde o desenvolvedor vai trabalhar.

**Figura 46 – Arquivos iniciais de um projeto Vue**



Para mais detalhes sobre como criar e codificar projetos com Vue, verifique as videoaulas dos capítulos 5 e 6.

### Características

A seguir são listadas algumas características importantes sobre o Vue:

- **Componentizável:** o Vue também preza pela criação de componentes, assim como grande parte dos *frameworks* de JavaScript modernos.
- **Declarativo:** seguindo os princípios do desenvolvimento reativo, a criação de componentes é bastante declarativa (ao invés de imperativa), o que faz com que o Vue **reaja** a mudanças no estado da aplicação de forma **eficiente**, assim como é feito no React.
- **Virtual DOM:** a manipulação do DOM é de responsabilidade do Virtual DOM, que cria uma estrutura em memória do DOM e só efetua as atualizações realmente necessárias, o que garante melhor desempenho e, conseqüentemente, melhor experiência do usuário, assim como é feito no React.
- **Excelente documentação:** o Vue é conhecido por possuir uma excelente documentação (<https://vuejs.org/v2/guide/index.html>) que

inclusive possui suporte colaborativo para o Português do Brasil (<https://br.vuejs.org/v2/guide/index.html>).

- **Funcionalidades para SPA's e de outros *frameworks*:** o Vue dá suporte a funcionalidades importantes para SPA's como por exemplo o roteamento. Diferentemente do Angular, essas funcionalidades não são incluídas por padrão. Diferentemente do React, essas funcionalidades são mantidas pela própria equipe do Vue. Além disso, o Vue também suporta opcionalmente a utilização de JSX e TypeScript. Assim, pode-se concluir que o Vue possui uma arquitetura gradativamente adotável, ou seja, o desenvolvedor pode incluir e aprender novos conceitos aos poucos.
- **Simplicidade:** o Vue preza muito pela simplicidade. Inclusive é possível trabalhar com Vue apenas incluindo sua biblioteca principal na *tag* `<script>` das páginas HTML, assim como é feito com o JQuery, por exemplo. Isso é interessante para quem está começando a estudar o Vue e também para sites de pequeno porte.
- **DSL (*Domain Specific Language*):** assim como o Angular, o Vue possui algumas diretivas para os *templates*, como por exemplo o *v-if* para condicionais, o *v-for* para iterações e o *v-model* para *two-way data-binding*. Além disso, possui o *v-bind* para o *binding* de propriedades e o *v-on* para tratamento de eventos.
- **Single File Components:** o Vue possui esse conceito que une código HTML (*template*), código JavaScript (*script*) e código CSS (*style*) em um mesmo arquivo (com extensão. *vue*), conforme imagem abaixo:

**Figura 47 – Exemplo de um *Single File Component***

```
<template>
  <h1>{{ msg }}</h1>
</template>

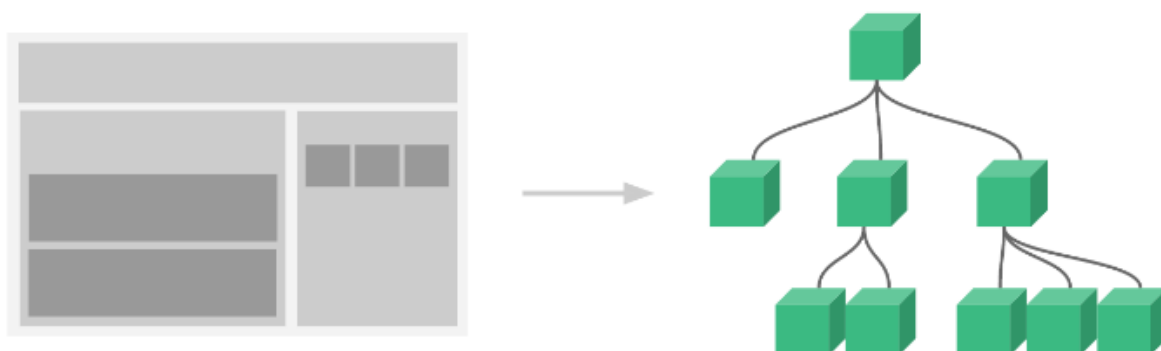
<script>
export default {
  data() {
    return {
      msg: 'Olá, Vue!',
    };
  },
};
</script>

<style scoped>
p {
  color: #333;
}
</style>
```

### Arquitetura do Vue

A imagem abaixo ilustra a arquitetura componentizável de aplicações com Vue. Com ela, é possível perceber que o Vue organiza os seus componentes em uma estrutura semelhante a uma árvore genealógica (com componentes pais e filhos), bem semelhante aos demais *frameworks*.

**Figura 46 – Arquitetura de componentes do Vue**



Fonte: [vuejs.org](https://vuejs.org).

## Implementação de Aplicações com Vue

Será vista, a seguir, a implementação do projeto “vue-intervalos”.

Primeiramente, incluímos o *link* do Materialize no arquivo index.html do projeto, conforme imagem abaixo:

**Figura 47 – Inclusão do Materialize CSS em um projeto Vue**



```

1 <html lang="en">
2 <head>
3 <meta charset="utf-8">
4 <title>vue-intervalos</title>
5 </head>
6 <!-- Materialize -->
7 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/css/materialize.min.css">
8 <body>
9 <div id="app"></div>
10 <script src="/dist/build.js"></script>
11 </body>
12 </html>

```

No caso do Vue, para implementar o exemplo de divisores basta apenas um arquivo, que será o App.vue. É claro que é possível quebrar o código em diversos componentes, mas esse não é o foco nesse momento. A figura a seguir mostra a implementação com Vue:

Figura 48 – Implementação com Vue

```

1 <template>
2   <div>
3     <h3>
4       Reatividade com intervalos de números - Vue
5     </h3>
6     <div>
7       <div>
8         <label>
9           Contador
10          <input
11            type="number"
12            min="1"
13            max="200"
14            v-model="currentValue"
15          >
16        </label>
17      </div>
18
19      <ul>
20        <li
21          v-for="divisor in divisors"
22          :key="divisor"
23        >
24          Números divisíveis por {{ divisor }}:
25          <span
26            v-for="number in getDivisiveisPor(divisor)"
27            :key="number"
28          >{{ number + ' ' }}</span>
29        </li>
30      </ul>
31    </div>
32  </div>
33 </template>
34
35 <script>
36 export default {
37   name: 'app',
38
39   created() {
40     this.divisors = [2, 3, 4, 5, 6, 7, 8, 9];
41   },
42
43   data() {
44     return {
45       currentValue: 1
46     }
47   },
48
49   methods: {
50     getDivisiveisPor(num) {
51       const numbers = [];
52       for (let i = 1; i <= this.currentValue; i++) {
53         if (i % num === 0) {
54           numbers.push(i);
55         }
56       }
57       return numbers;
58     },
59   },
60 }
61 </script>
62
63 <style>
64 body {
65   padding: 20px;
66 }
67 </style>

```

## Entendendo a implementação da aplicação

---

Nas linhas 1 a 33, está definido o template, que é o código HTML que pode possuir comandos específicos do Vue, conhecidos como **diretivas** (assim como no Angular).

Na linha 14, é feito o *two-way data-binding* através da diretiva v-model. Ao contrário do Angular, não é necessária nenhuma configuração adicional para que o v-model funcione.

Nas linhas 19 a 30, é feita a renderização dos valores utilizando a diretiva v-for (semelhante ao \*ngFor do Angular). O Vue também recomenda a utilização de *keys* assim como é feito no React. Quando o conteúdo de um atributo deve ser código ou expressão JavaScript, devemos decorá-lo com v-bind-atributo ou :atributo (como pode ser visto nas linhas 22 e 27). Isso é semelhante à notação de colchetes [ atributo ] do Angular.

Nas linhas 35 a 61, é implementado o código JavaScript da aplicação. Em geral, o Vue exporta um objeto com diversos agrupamentos de valores, sendo:

- *name*: nome do componente.
- *created*: método de ciclo de vida que indica que o componente terminou de ser criado. Nesse local criamos o atributo de divisores, cujos valores são fixos e, portanto, não necessitam pertencer ao estado do componente.
- *data*: **função** que pode retornar diversos valores e indica o **estado** do componente.
- *methods*: local para implementação de funções para manipular o estado do componente.

Nas linhas 63 a 67, está definido o estilo do componente.

O professor irá postar o código-fonte desta aplicação no fórum da disciplina.

Para mais detalhes de implementação de aplicações com Vue, consulte as videoaulas dos capítulos 5 e 6.



## Capítulo 6. Comparativo: Angular, React e Vue

---

Antes de ler o restante do capítulo, é interessante que você estude a aplicação *angular/react/vue-github* que foi demonstrada nas videoaulas do capítulo 6. Esses apps serão postados pelo professor no fórum da disciplina.

### Angular

---

A seguir, são listadas algumas vantagens e desvantagens em relação ao Angular.

#### Vantagens do Angular

---

- O Angular se tornou uma plataforma de desenvolvimento *front end* bastante consolidada a partir da versão 2, após ser totalmente reescrito.
- O Angular é suportado por uma grande empresa (Google).
- O Angular adotou o TypeScript por padrão, o que pode garantir menos *bugs* em desenvolvimento e, conseqüentemente, um sistema mais estável em produção.
- O Angular possui um CLI bastante poderoso, que permite a criação automática de diversas estruturas como componentes, serviços, *pipes*, etc.
- O Angular possui um conjunto completo de ferramentas para a construção do site, como módulos de tratamento de formulários e roteamento.
- A comunidade do Angular é bastante ativa na Internet.
- O Angular possui o RxJS como dependência e o utiliza bastante internamente. Já se sabe da importância e capacidade desta excelente biblioteca baseada em *Observables*.

- Por todo esse ferramental e estrutura, o Angular é fortemente recomendado para a construção de sistemas de grande porte.

### Desvantagens do Angular

---

- A curva de aprendizagem do Angular é, em geral, alta.
- Por possuir um ferramental completo, o Angular é, de certa forma, um *framework* “fechado”, o que impede algumas customizações.
- O Angular depende do Node.js para funcionar, principalmente para a transpilação e empacotamento do código (Babel/Webpack).
- É necessário um estudo adicional da DSL do Angular.

### React

---

A seguir, são listadas algumas vantagens e desvantagens em relação ao React.

### Vantagens do React

---

- O React é suportado por uma grande empresa (Facebook).
- O React possui suporte opcional ao TypeScript.
- O React possui suporte ao JSX, que permite uma escrita mais declarativa e menos imperativa.
- A comunidade do React é bastante ativa na Internet.
- O React tem sido bastante reconhecido no mercado nos últimos anos.
- O React é conhecido por manipular muito bem o DOM com o Virtual DOM.

- O React possui um excelente suporte ao ES6+, ou seja, você utiliza o React com as tecnologias mais modernas em se tratando de JavaScript.
- Por ser uma biblioteca simples, o React dá mais liberdade ao desenvolvedor para escolher o ferramental.
- No React não há uma DSL a ser estudada. Basta aprender JavaScript ES6+.

### Desvantagens do React

---

- A curva de aprendizagem do React é, em geral, média.
- Algumas dependências necessárias à construção de SPA's são mantidas por terceiros, como por exemplo o *react-router*.
- O React é bastante verboso caso não seja adotado o JSX.
- Apesar de não possuir DSL é necessário aprender JSX.
- Para utilizar o JSX, é necessário que o React seja executado em ambiente Node.js com transpilação e empacotamento (Babel/Webpack).

### Vue

---

A seguir, são listadas algumas vantagens e desvantagens em relação ao Vue.

### Vantagens do Vue

---

- O Vue é bastante customizável.
- O Vue possui suporte opcional ao TypeScript e ao JSX.
- A comunidade do Vue é bastante ativa na Internet.
- O Vue também possui um Virtual DOM, semelhante ao do React.

- O Vue é conhecido por possuir uma excelente documentação.
- Por ser uma biblioteca simples, o Vue dá mais liberdade ao desenvolvedor para escolher o ferramental.
- As dependências para a construção de SPA's são mantidas pela própria equipe do Vue, como por exemplo o *vue-router*.
- É perfeitamente possível utilizar o Vue sem a necessidade do Node.js, transpiladores e empacotadores.

### Desvantagens do Vue

---

- Não há uma grande empresa suportando o Vue, que é mantido por uma comunidade de cerca de 40 desenvolvedores.
- O Vue infelizmente é ainda menos reconhecido no mercado nacional em comparação ao Angular e React.
- Para utilizar o Vue, é necessário um estudo de sua DSL.

## Conclusões finais

---

Tanto o Angular quanto o React e o Vue são excelentes *frameworks* JavaScript de *front end* e tendem a dominar o mercado por muito tempo.

Algumas vantagens e desvantagens citadas no tópico anterior podem ter o efeito contrário para alguns desenvolvedores, pois muitos argumentos são pessoais.

Um grande fator a ser considerado na escolha de qual *framework* utilizar é também o gosto pessoal do desenvolvedor, ou seja, com qual *framework* ele se sente mais confortável para trabalhar. Isso é conhecido como DX (Developer eXperience).

## Referências

---

ANGULAR DOCUMENTATION. Disponível em: <<https://angular.io/>>. Acesso em: 23 abr. 2021

BERNHARDT, Manuel. Reactive Web Applications: Covers Play, Akka, and Reactive Streams. New York: Manning, 2016.

FILIPOVA, Olga. Learning Vue.js 2. USA: Packt Publishing, 2016.

PORCELLO, Eve; BANKS, Alex. Learning React: Functional Web Development with React and Redux. USA: O'Reilly, 2017.

POWELL, Josh C.; MIKOWSKI, Michael S. Single Page Web Applications: JavaScript end-to-end. New York: Manning, 2014.

REACT DOCUMENTATION. Disponível em: <<https://reactjs.org/>>. Acesso em: 23 abr. 2021.

VUE.JS. Disponível em: <<https://vuejs.org/>>. Acesso em: 23 abr. 2021.