



Aprenda com quem faz

# Vue

Prof. Guilherme Henrique de Assis

2023



## SUMÁRIO

Capítulo 1. Introdução	4
1.1. Introdução	4
Capítulo 2. Vue.js	10
2.1. Hello World	10
2.2. Sintaxe de Template	12
2.3. Métodos	13
2.4. Computed Properties	14
2.5. Watchers	16
2.6. Diretivas	17
2.7. Eventos	23
2.8. Estilização	27
2.9. Componentes	28
2.10. Vue Router	34
Referências	36



## Capítulo 1.Introdução

---

Neste capítulo, faremos uma breve introdução ao Vue.js, falando um pouco do que se trata, da sua história e de sua presença no mercado de trabalho.

### 1.1. Introdução

O Vue.js é um framework JavaScript open-source voltado para a criação de interfaces do tipo Single Page Applications (SPA's). Ele foi lançado por Evan You, em 2014, que trabalhou no Google com o AngularJS. Na época, observou a necessidade de um framework mais completo que lidasse melhor com grandes aplicações, resolvendo, então, iniciar a criação do Vue.js.

**Figura 1 – Vue.js**



**Fonte:** <https://vuejs.org/>.

Ele foi desenvolvido com o objetivo de que pudesse ser adotado de forma incremental, já que sua biblioteca principal diz respeito à camada visual, sendo possível até mesmo integrá-lo com outras bibliotecas em projetos já existentes.

A curva de aprendizado do framework não é alta, facilitando o aprendizado. Ele fornece um conjunto de ferramentas que podem ser utilizadas para facilitar o desenvolvimento, por exemplo, o Vue Cli, uma ferramenta de linha de comando que facilita a criação dos projetos. Uma outra vantagem do Vue.js é que ele é um framework bem leve, não pesando muito no carregamento das páginas. O Vue.js está em pleno desenvolvimento, estando atualmente na versão 3.0, que foi lançada em

setembro de 2020. A imagem abaixo retrata todas as versões do framework, com a sua respectiva data de lançamento e nome.

**Figura 2 – Versões do Vue.js.**

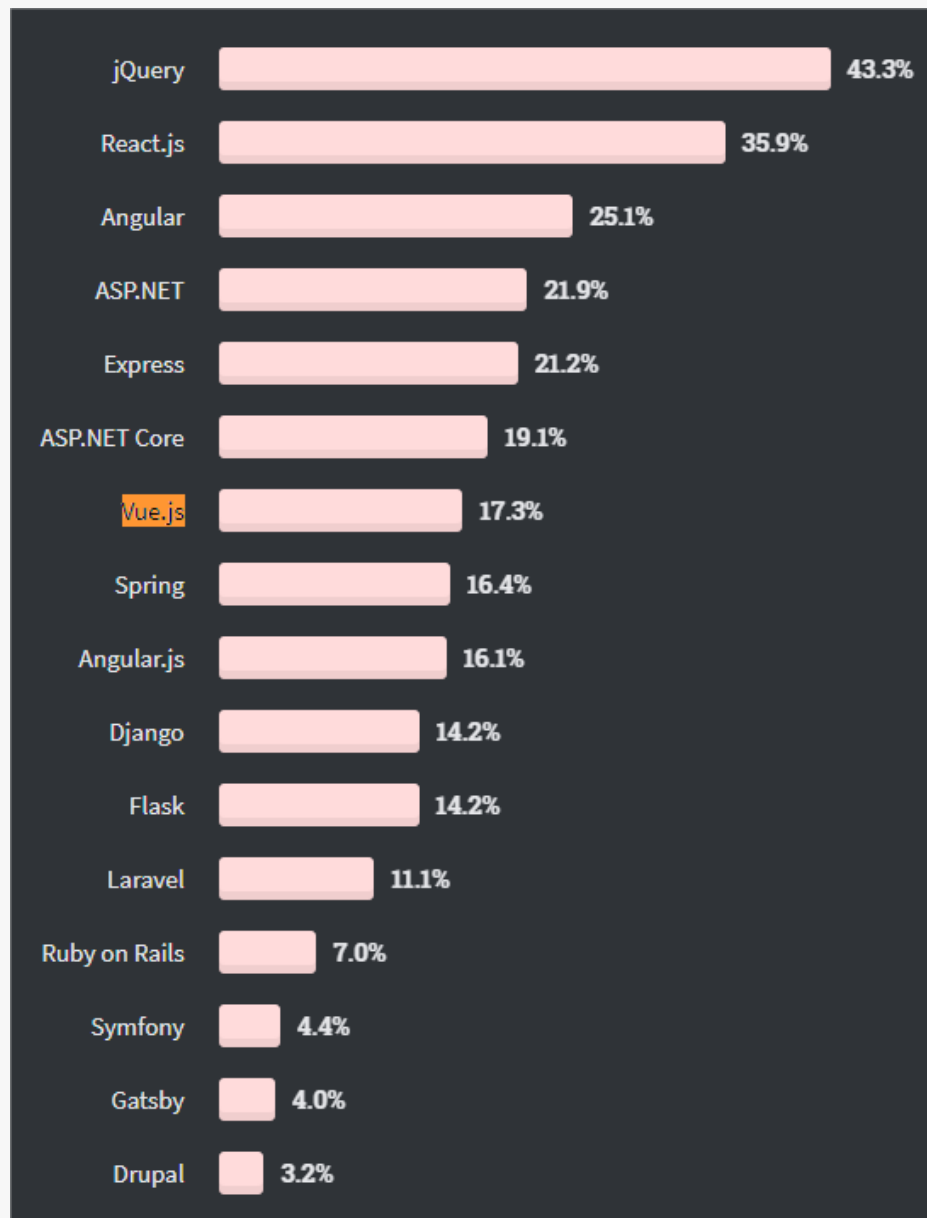
Version	Release date	Title
3.0	September 18, 2020	One Piece <sup>[17]</sup>
2.6	February 4, 2019	Macross <sup>[18]</sup>
2.5	October 13, 2017	Level E <sup>[19]</sup>
2.4	July 13, 2017	Kill la Kill <sup>[20]</sup>
2.3	April 27, 2017	JoJo's Bizarre Adventure <sup>[21]</sup>
2.2	February 26, 2017	Initial D <sup>[22]</sup>
2.1	November 22, 2016	Hunter X Hunter <sup>[23]</sup>
2.0	September 30, 2016	Ghost in the Shell <sup>[24]</sup>
1.0	October 27, 2015	Evangelion <sup>[25]</sup>
0.12	June 12, 2015	Dragon Ball <sup>[26]</sup>
0.11	November 7, 2014	Cowboy Bebop <sup>[27]</sup>
0.10	March 23, 2014	Blade Runner <sup>[28]</sup>
0.9	February 25, 2014	Animatrix <sup>[29]</sup>
0.8	January 27, 2014	N/A <sup>[30]</sup>
0.7	December 24, 2013	N/A <sup>[31]</sup>
0.6	December 8, 2013	VueJS <sup>[32]</sup>

**Fonte:** <https://en.wikipedia.org/wiki/Vue.js>.

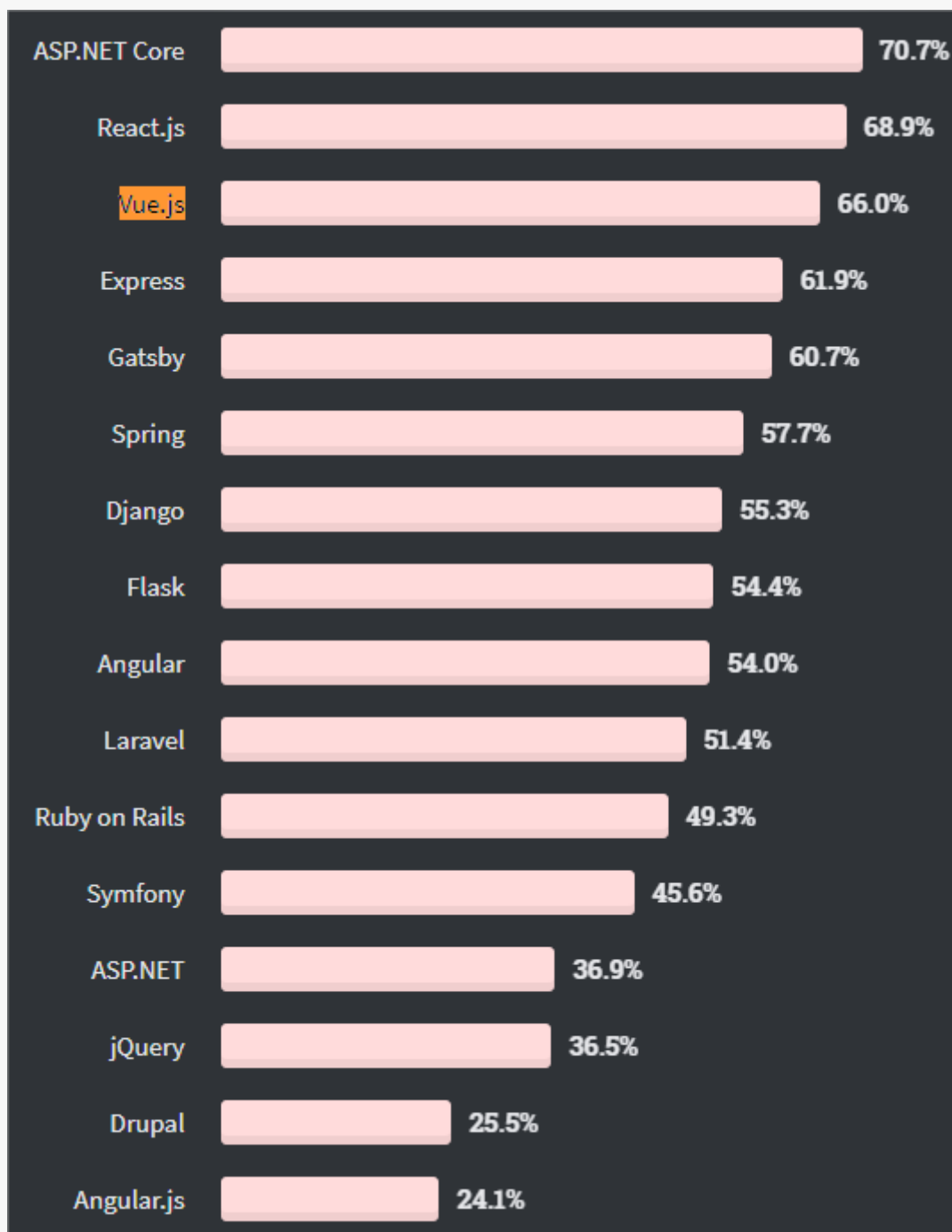
A respeito da sua utilização no mercado de trabalho, as duas imagens abaixo nos mostram algumas questões interessantes. Elas foram retiradas da pesquisa anual realizada pelo site Stack Overflow, no ano de 2020, com desenvolvedores. A primeira imagem diz respeito aos frameworks web mais utilizados pelos entrevistados. Neste quesito, podemos notar a forte presença do Vue.js, sendo utilizado por 17,8% dos respondentes, porém ficando atrás de seus principais concorrentes, o React e Angular. Já a segunda imagem diz respeito aos frameworks mais amados pelos desenvolvedores. Neste quesito, podemos notar que o Vue.js

se destaca bastante, superando o Angular e ficando praticamente empatado com o React.

**Figura 3 – Frameworks Web mais utilizados.**



**Fonte:** <https://insights.stackoverflow.com/survey/2020>.

**Figura 4 – Frameworks Web mais amados.**

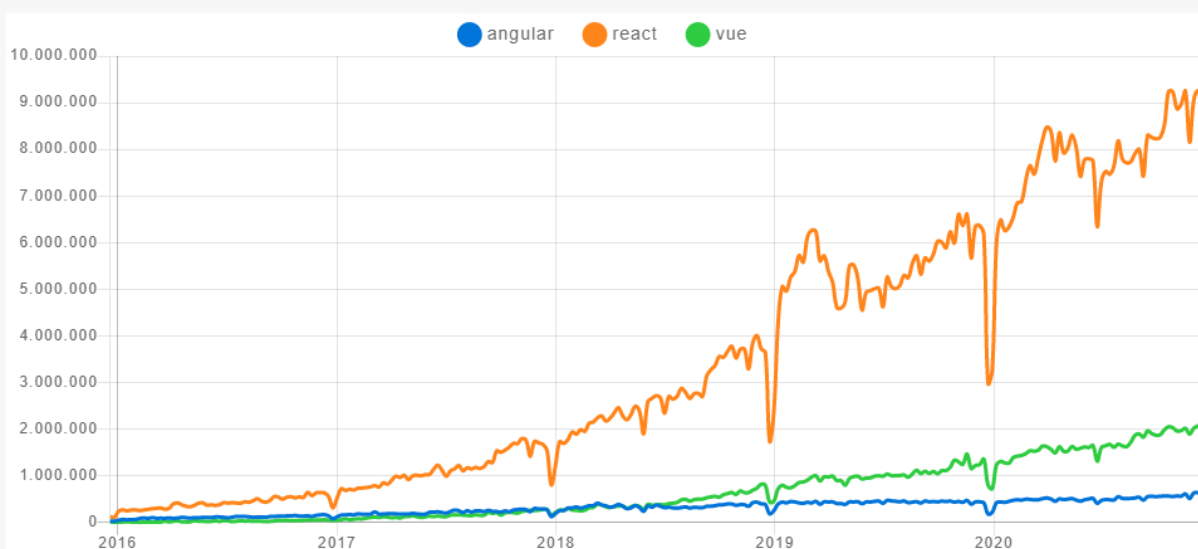
**Fonte:** <https://insights.stackoverflow.com/survey/2020>.

Um outro dado interessante a ser analisado é a quantidade de estrelas que os projetos têm no GitHub. Apesar de não ser uma métrica oficial, ela pode ser utilizada para termos uma ideia da popularidade e aceitação de uma ferramenta. Neste quesito, o Vue tem, no momento da

escrita deste texto, 177.000 estrelas, enquanto o React possui 161.000 estrelas e o Angular 68.900 estrelas. Isso nos ajuda a ter uma ideia da popularidade e da aceitação do framework.

Um outro ponto interessante que podemos analisar é a quantidade de downloads das bibliotecas ao longo dos últimos cinco anos. Esta informação está representada na imagem abaixo e foi obtida no site NPM Trends. Pela imagem, podemos observar que o Vue.js vem com uma crescente no número dos downloads, ficando atrás apenas do React.

**Figura 5 – Quantidade de downloads no NPM.**



Fonte: <https://www.npmtrends.com/angular-vs-react-vs-vue>.





**XP**e

## > Capítulo 2



## Capítulo 2.Vue.js

---

Neste capítulo, falaremos sobre as funcionalidades do Vue.js, iniciando com um Hello World e indo até as funcionalidades mais avançadas, como Vue Router.

### 2.1. Hello World

A instalação do Vue.js pode ser feita a partir da inclusão direta do código do framework em uma tag script, apontando para uma CDN ou até mesmo para um arquivo local. Para a criação de aplicações mais robustas e complexas, é aconselhável que o projeto seja criado a partir da ferramenta de linha de comando oficial, chamada Vue CLI. Um projeto criado a partir dele já vem com as configurações prontas, como as configurações para a geração de build e hot-reload. A imagem abaixo ilustra a importação do Vue.js a partir de um script externo, sendo inserido na própria página.

**Figura 6 – Importação por script.**

```
<script src="https://unpkg.com/vue@3.0.4"></script>
```

**Fonte: Elaborado pelo próprio professor.**

Uma outra forma de criar o projeto Vue.js é a partir do Vue CLI. Para utilizá-lo, é preciso que o Node.js e o NPM estejam instalados na máquina. É possível realizar a sua instalação de forma global utilizando o NPM, para que assim ele possa ser acessado via linha de comando, independentemente de estar instalado diretamente no projeto. Ele pode ser instalado utilizando o comando “npm install –g @vue/cli”. Para criar um projeto, basta chamar o comando “vue create” com o nome do projeto, por exemplo: “vue create nome-do-projeto”. É recomendável fechar e abrir o prompt de comando após a instalação, para que ele possa reconhecer.

Com o Vue.js importado na página, uma instância dele pode ser criada a partir da chamada do método “createApp”. Esse método recebe

como parâmetro um objeto JSON, com todas as configurações da aplicação, como os dados que ela irá trabalhar e os métodos para realizações de ações. Uma das propriedades da instância mais utilizada é a chamada “data”. Ela é uma função que é chamada durante o processo da criação do componente e deve retornar um objeto. As informações inseridas aqui podem ser mostradas em tela utilizando a sintaxe de template, que será mostrada no próximo tópico. O próprio Vue.js se encarrega de atualizar as informações na tela assim que elas são alteradas.

O objeto que possui a instância do Vue.js tem um método chamado “mount”. Ele recebe, como parâmetro, o id de um elemento HTML, no qual será montada a aplicação. A imagem abaixo ilustra a criação de um objeto chamado app, com a opção de dados configurados no “data”. Ele possui somente uma propriedade chamada “name”, com o valor “John”. O método “createApp” retorna a instância do Vue.js, porém ele ainda não realizou a montagem dos elementos na tela. Para realizar a montagem, é preciso chamar o método “mount” passando o id elemento. Estes métodos são separados, pois há casos em que o desenvolvedor precisa manipular a instância do Vue.js retornada pelo “createApp” antes de realizar a montagem em tela.

**Figura 7 – Criação da instância do Vue.js.**

```
const app = {  
  data() {  
    return {  
      name: "John",  
    }  
  }  
}  
Vue.createApp(app).mount('#hello-world')
```

**Fonte: Elaborado pelo próprio professor.**

Uma ferramenta muito interessante para auxiliar o desenvolvimento com o Vue.js é o Vue Devtools. É uma extensão disponível

para o Chrome e para o Firefox, que permite inspecionar e debugar as aplicações, permitindo, por exemplo, que o desenvolvedor consulte a estrutura dos componentes da aplicação e qual o estado de cada um deles.

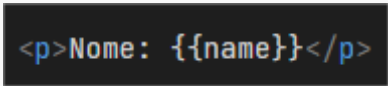
## 2.2. Sintaxe de Template

O Vue.js utiliza uma linguagem de template baseada no HTML que permite ao desenvolvedor vincular os elementos da interface aos dados. Este vínculo é o chamado data-binding.

O Vue.js trabalha com o conceito do DOM Virtual, mantendo uma cópia da estrutura do HTML nessa árvore de elementos virtual. Com isso em mãos, quando algum estado da aplicação é alterado e ele precisar atualizar a tela, o Vue.js consegue identificar somente as partes que precisam ser atualizadas, manipulando assim o DOM o mínimo necessário.

Para realizar a interpolação de dados ao longo do HTML, o Vue.js utiliza a sintaxe de chaves duplas. A imagem abaixo ilustra como a interpolação é realizada, fazendo com que após a execução, o valor da propriedade seja substituído pela informação em formato de texto.

**Figura 8 – Interpolação de texto.**



```
<p>Nome: {{name}}</p>
```

**Fonte: Elaborado pelo próprio professor.**

Conforme visto na imagem acima, o Vue.js interpreta os dados colocados entre as chaves duplas como texto simples. Caso seja necessário que essa informação seja interpretada como HTML, tendo inclusive os estilos interpretados, basta utilizar a diretiva “v-html” no elemento no qual o HTML será renderizado. Essa diretiva fará com que a informação contida na propriedade seja mostrada dentro do elemento no formato HTML. A imagem abaixo ilustra a utilização desta diretiva.

**Figura 9 – Interpolação de HTML.**

```
<div v-html="htmlContent"></div>
```

**Fonte: Elaborado pelo próprio professor.**

Dentro das chaves duplas, também é possível escrever expressões JavaScript, que serão resolvidas e terão o resultado apresentado na tela. O Vue.js suporta dentro das chaves duplas apenas uma expressão, não sendo possível colocar várias expressões juntas. A imagem abaixo ilustra esta utilização.

**Figura 10 – Interpolação com expressão.**

```
<p>{{age > 18 ? 'maior de idade' : 'menor de idade'}}</p>
```

**Fonte: Elaborado pelo próprio professor.**

### 2.3. Métodos

Vimos anteriormente que dentro da instância Vue.js podemos configurar, na propriedade “data”, os dados que poderão ser visualizados em tela. Para que possamos implementar as ações que manipulam esses dados, podemos implementar métodos que podem ser chamados a partir de alguma ação do usuário, como um click em um botão.

Os métodos podem ser configurados na propriedade “methods” da instância do Vue.js. Esta propriedade recebe um objeto JSON com as funções dentro dele. O valor de “this” é automaticamente associado à instância do componente em questão, permitindo, assim, que dentro dos métodos sejam acessadas as propriedades definidas no “data” a partir do “this”. Não é recomendável definir os métodos como arrow functions, para evitar problemas de referência da instância. A imagem abaixo ilustra a configuração de dois métodos chamados “incrementAge” e “decrementAge” que acessam a propriedade “age” a partir do “this”.

**Figura 11 – Métodos.**

```
const app = {  
  data() {  
    return {  
      age: 30  
    }  
  },  
  methods: {  
    incrementAge() {  
      this.age++;  
    },  
    decrementAge() {  
      this.age--;  
    }  
  }  
}
```

**Fonte: Elaborado pelo próprio professor.**

## 2.4. Computed Properties

Uma outra funcionalidade do Vue.js são as *computed properties*. Assim como *data* e *methods*, elas também podem ser configuradas na instância do Vue.js, a partir da propriedade “computed”. As *computed properties* funcionam como os dados que são declarados no “data”, porém, com elas, o desenvolvedor pode montar uma expressão. A propriedade “computed” é um objeto JSON que recebe as propriedades como configurações desse objeto, sendo que estas devem ser uma função que retorna o valor calculado.

A imagem abaixo ilustra um exemplo de *computed property*, chamada “fullName”. Nela, estão sendo concatenadas as propriedades “firstName” e “lastName”, sendo assim retornado o nome completo.

**Figura 12 – Computed Property.**

```
data() {
  return {
    firstName: "John",
    lastName: "Snow",
  },
  computed: {
    fullName: function () {
      return `${this.firstName} ${this.lastName}`;
    }
  }
}
```

**Fonte: Elaborado pelo próprio professor.**

Essa mesma implementação da imagem anterior poderia ser realizada diretamente no template a partir da concatenação de “firstName” e “lastName”. O problema disso é que o código do template acaba ficando poluído, situação que piora mais ainda à medida que as expressões vão crescendo em complexidade. Outra vantagem da *computed property*, nesse caso, é o reaproveitamento de código, pois não há a necessidade de o desenvolvedor copiar e colar a expressão em vários locais, basta usar a *computed property*.

Essa mesma situação também poderia ser resolvida com a criação de um método, tal como “getFullName” para o exemplo da imagem anterior. Este método poderia realizar a concatenação internamente e retornar o resultado. O problema desta abordagem é que os métodos são executados todas as vezes em que é chamado, enquanto a *computed property* armazena seu valor em cache, sendo alterada somente caso algum de seus valores sofra alteração. Por isso, em cenários em que o valor da propriedade não muda com frequência, é mais interessante a *computed property* ao invés do método, já que ele irá manter o seu valor em cache, melhorando a performance da aplicação.

## 2.5. Watchers

Watcher é uma outra funcionalidade do Vue.js, que permite ao desenvolvedor escrever um código que poderá reagir a alteração de determinado valor que ele estiver observando. Um cenário de utilização seria quando é necessário fazer uma requisição assíncrona em resposta à mudança de alguns dados, por exemplo, consumir um web service quando o campo de CPF for alterado.

Para configurar watchers na instância do Vue.js, basta informá-los na propriedade “watch”. Essa propriedade deve receber um objeto JSON, sendo que o nome das suas propriedades será o nome do watcher, que deve ter o mesmo nome da propriedade que ele ficará observando as mudanças. O seu valor deve ser uma função que será executada quando a propriedade em questão for modificada. Essa função recebe dois parâmetros, o primeiro corresponde ao valor atual da propriedade, já com a modificação realizada, e o segundo com o valor antigo da propriedade, antes da modificação ter sido realizada.

A imagem abaixo ilustra a utilização de um watcher. Nela, foi criada um watcher para a propriedade “address”, e sempre que esta propriedade for alterada, o watcher será executado imprimindo em console o valor atual e anterior.

**Figura 13 – Watcher.**

```
data() {  
  return {  
    address: "",  
  },  
  watch: {  
    address: function (newValue, oldValue) {  
      console.log(newValue);  
      console.log(oldValue);  
    }  
  }  
}
```



**Fonte: Elaborado pelo próprio professor.**

## 2.6. Diretivas

No Vue.js, as diretivas são atributos especiais que podem ser adicionados nos elementos HTML para realizar ações específicas relacionadas a alterações no DOM em resposta a uma mudança do valor da expressão que lhe for passada. Elas são muito utilizadas no desenvolvimento das aplicações, pois possuem diversas funções, desde mostrar um valor em tela até a definir quando um elemento deve ou não ser mostrado. Existem várias diretivas. Abaixo são abordadas algumas das principais:

- v-on:

A diretiva v-on tem por objetivo vincular um método ou expressão à determinada ação. Após escrever o nome da diretiva v-on, basta colocar o sinal de dois pontos seguido pelo nome da ação que deseja vincular. Na imagem abaixo, temos um exemplo de sua utilização, no qual é vinculado os métodos “incrementAge” e “decrementAge” ao click de um botão. A diretiva v-on também pode ser escrita utilizando um “@”. No exemplo da imagem abaixo, podemos ver essa utilização, sendo que “v-on:click” e “@click” funcionam da mesma forma. Ao invés de informar um método, também é possível informar uma expressão JavaScript simples, como por exemplo “age++”.

**Figura 14 – Diretiva v-on.**

```
<button v-on:click="incrementAge">Incrementar idade</button>  
<button @click="decrementAge">Decrementar idade</button>
```

**Fonte: Elaborado pelo próprio professor.**

- v-if, v-else e v-elseif:

As diretivas v-if, v-else e v-elseif servem para definir condições para que um elemento seja removido ou permaneça na página. Para o v-if, é passada uma expressão, e se essa expressão for verdadeira, o elemento no qual ela está associada permanecerá na tela. Caso a expressão seja falsa, o elemento será removido da página.

Em conjunto com o v-if, é possível utilizar o v-else e o v-elseif. O funcionamento deles é igual ao das linguagens de programação. Se a diretiva v-else estiver posicionada abaixo da v-if, a expressão do v-else será avaliada caso a do v-if seja falsa. Se a expressão do v-else for verdadeira, o elemento no qual ele está localizado será incluído na página.

Entre o v-if e o v-else, é possível existir vários v-elseif. Sua expressão será avaliada caso o v-if ou v-elseif anterior seja falso. Caso a expressão seja verdadeira, o elemento será mostrado em tela e os demais v-elseif ou v-else não serão mais avaliados.

A imagem abaixo ilustra um exemplo de utilização dessas diretivas, no qual é verificado qual o valor da variável “grade”, e de acordo com o seu valor, um texto diferente é mostrado em tela.

**Figura 15 – Diretivas v-if, v-else e v-elseif.**

```
<p v-if="grade < 60">Reprovado</p>
<p v-else-if="grade >= 60 && grade < 98">Aprovado</p>
<p v-else-if="grade >= 98">Aprovado com excelência</p>
<p v-else>Valor inválido</p>
```

**Fonte: Elaborado pelo próprio professor.**

- v-show:

A diretiva v-show, assim como a diretiva v-if, serve para verificar se um elemento será ou não mostrado em tela. Para a v-show, também deve ser informada uma expressão que, caso seja verdadeira, mostrará o

elemento na qual ela está localizada na tela e, caso seja falsa, não mostrará.

A diferença do v-if para o v-show é que, com relação ao v-if, caso a expressão seja falsa, o elemento é completamente removido do DOM. Já com o v-show, caso a expressão seja falsa, apesar de o elemento não aparecer na tela, ele não será removido completamente do DOM, e sim será oculto através de CSS. Esta, então, é a diferença entre os dois, o v-if remove o elemento completamente caso a expressão seja falsa, enquanto o v-show somente oculta o mesmo.

Usar o v-if ou o v-show vai depender da situação. Se a expressão que está sendo associada à diretiva sofrer alterações frequentes, talvez seja mais interessante utilizar o v-show, pois assim a aplicação terá uma melhor performance para tornar o elemento visível novamente, bastando alterar o CSS do elemento ao invés de ter que inserir ele por completo no DOM. Caso a expressão não sofra alterações frequentes, talvez seja mais interessante utilizar o v-if, pois assim o elemento não ficará no DOM de forma desnecessária, e quando ocorrer de a expressão se transformar em verdadeira, a aplicação então terá que inseri-lo no DOM.

A imagem abaixo ilustra a utilização do v-show. No exemplo, caso a variável “isUserLogged” seja verdadeira, é mostrado o botão de login e, caso seja falsa, é mostrado o botão de logout.

**Figura 16 – Diretiva v-show.**

```
<button v-show="!isUserLogged" @click="isUserLogged = true">Login</button>  
<button v-show="isUserLogged" @click="isUserLogged = false">Logout</button>
```

**Fonte: Elaborado pelo próprio professor.**

- v-text:

A diretiva v-text serve para mostrar o valor de uma propriedade como texto dentro do elemento. Ela pode ser substituída pelas chaves duplas diretamente dentro do elemento, sem a necessidade da diretiva. A imagem abaixo ilustra a utilização da diretiva e do atalho com as chaves duplas, sendo que as duas produzem o mesmo resultado.

**Figura 17 – Diretiva v-text.**

```
<p>{{firstName}}</p>  
<p v-text="firstName"></p>
```

**Fonte: Elaborado pelo próprio professor.**

- v-html:

A diretiva v-html tem objetivo parecido com a v-text, com a diferença que ela reproduz o conteúdo da propriedade no formato HTML, sendo mais indicada, então, em situações que o elemento deve replicar um código HTML com toda a formatação, e não somente um texto. A imagem abaixo ilustra a utilização dessa diretiva.

**Figura 18 – Diretiva v-html.**

```
<div v-html="htmlContent"></div>
```

**Fonte: Elaborado pelo próprio professor.**

- v-once:

A diretiva v-once faz com que o valor das propriedades inseridas dentro do elemento seja avaliado uma só vez, no momento da montagem da tela. Nas próximas vezes em que os dados forem atualizados, o conteúdo referente a eles não será atualizado dentro dos elementos marcados com v-once. A imagem abaixo ilustra essa utilização. O valor de “count” é carregado inicialmente no elemento com e no elemento sem

v-once. Ao clicar no botão que adiciona uma unidade ao “count”, o elemento com a diretiva v-once não será atualizado enquanto o outro será.

**Figura 19 – Diretiva v-once.**

```
<p v-once>{{ count }}</p>
<p>{{count}}</p>
<button @click="count++">Adicionar</button>
```

**Fonte: Elaborado pelo próprio professor.**

- v-for:

A diretiva v-for é muito utilizada no Vue.js, pois ela permite que sejam criadas iterações nos elementos. Ela pode receber como parâmetro, por exemplo, um array de dados, e o conteúdo que estiver dentro do elemento associado ao v-for será replicado à quantidade de vezes do tamanho do array, sendo que cada elemento replicado tem o valor do elemento em questão associado, sendo possível mostrá-lo em tela.

A imagem abaixo ilustra essa utilização, no qual “carBrands” é um array de strings com marcas de carro. Como a diretiva v-for está associada à tag “li”, essa tag será replicada uma vez para cada marca do array de marcas. Ao utilizar a expressão “brand in carBrands”, está sendo criada uma variável chamada “brand” para cada um dos elementos do array, e essa variável pode ser utilizada dentro do elemento para, por exemplo, exibi-lo na tela.

**Figura 20 – Diretiva v-for.**

```
<h4>Marcas</h4>
<ul>
  <li v-for="brand in carBrands">{{brand}}</li>
</ul>
```

**Fonte: Elaborado pelo próprio professor.**

- v-bind:

A diretiva v-bind é utilizada para vincular uma propriedade a um atributo. Ela pode ser utilizada com a abreviação de dois pontos, omitindo o v-bind. A imagem abaixo ilustra a utilização desta diretiva, no qual temos dois links que estão vinculando a propriedade “urlIgti” ao atributo href. Na primeira, é utilizada a versão com o v-bind escrito, e na segunda é utilizada a abreviação somente com os dois pontos. Ambos funcionam da mesma maneira.

**Figura 21 – Diretiva v-bind.**

```
<a v-bind:href="urlIgti">IGTI</a>  
<a :href="urlIgti">IGTI</a>
```

**Fonte: Elaborado pelo próprio professor.**

- v-model:

A diretiva v-model é utilizada para criar uma interligação do tipo two-way databinding entre os dados e os elementos de input, textarea e de select. Essa ligação do tipo two-way databinding permite que o dado seja atualizado independentemente do lado em que foi alterado. Se tiver sido alterado pelo usuário na tela, a alteração será refletida no modelo, e se tiver sido alterada no modelo, a alteração também é refletida na tela. Esta diretiva é muito utilizada em formulários para pegar os dados digitados pelo usuário. A imagem abaixo ilustra essa vinculação.

**Figura 22 – Diretiva v-model.**

```
<label for="input-address">Endereço:</label>  
<input id="input-address" type="text" v-model="address">
```

**Fonte: Elaborado pelo próprio professor.**

- v-pre:

A diretiva `v-pre` serve para fazer com que o elemento no qual ela está associada não seja compilado. Na imagem abaixo, por exemplo, mesmo colocando dentro do elemento uma propriedade dentro das chaves duplas, ela não será processada, sendo exibido na tela literalmente o “`{{firstName}}`”, e não o valor da propriedade, como ocorreria caso o Vue.js tivesse realizado o processamento do elemento.

**Figura 23 – Diretiva `v-pre`.**

```
<p v-pre>{{firstName}}</p>
```

**Fonte: Elaborado pelo próprio professor.**

- `v-cloak`:

A diretiva `v-cloak` tem por característica permanecer dentro do elemento somente até o Vue.js terminar a compilação inicial. Após a finalização, o próprio Vue.js se encarrega de remover a tag do elemento. Isso pode ser utilizado em conjunto com regras de CSS para fazer com que elementos que dependam de uma propriedade do Vue.js não sejam exibidos antes que o Vue.js tenha terminado a inicialização.

No exemplo da imagem abaixo, temos dois parágrafos com a propriedade “`firstName`” dentro deles. O segundo elemento possui a diretiva `v-cloak`, e no CSS existe uma regra fazendo com que elementos que possuam essa diretiva não sejam visíveis. Dessa forma, ao abrir a página no primeiro carregamento por alguns instantes, é possível ver o primeiro elemento com “`{{firstName}}`” como texto puro na tela, e instantes depois, assim que o Vue.js termina o processamento, ele é substituído para o valor da propriedade. O segundo elemento, que está com o `v-cloak`, não é mostrado enquanto a compilação não termina, evitando assim essa situação.

**Figura 24 – Diretiva `v-cloak`.**

```
<style>
  [v-cloak] {
    display: none;
  }
</style>

<p>{{firstName}}</p>
<p v-cloak>{{firstName}}</p>
```

**Fonte: Elaborado pelo próprio professor.**

## 2.7. Eventos

Para escutar e tratar eventos no Vue.js, utilizamos a diretiva `v-on`, como visto anteriormente. Dentro da diretiva, é possível passar um método que será executado quando o evento que estiver configurado na diretiva ocorrer. Também é possível passar expressões simples do JavaScript, como um `“count++”`.

É possível passar parâmetros para os métodos, bastando passá-los entre parênteses. Além disso, caso seja necessário acessar o objeto do evento do DOM, basta passá-lo com parâmetro utilizando `“$event”`. Caso o método tenha sido associado diretamente à diretiva, sem a passagem de parâmetros, o objeto do evento é enviado automaticamente, bastando ao método recebê-lo como único parâmetro.

A imagem abaixo ilustra a utilização do `v-on` associado ao evento de click do mouse. No primeiro exemplo, é passado um parâmetro para o método, enquanto no segundo ele é somente associado, sem passagem de parâmetros. Neste segundo exemplo, caso o método queira acessar o evento do DOM, basta ele receber um parâmetro com qualquer nome, pois o Vue.js automaticamente envia o evento como parâmetro neste cenário. O terceiro exemplo consiste na utilização do `@` no lugar do `v-on`, com passagem de parâmetros. Ao passar um parâmetro, o Vue.js não envia mais o evento do DOM automaticamente, dessa forma, caso o desenvolvedor queira acessar o evento do DOM, ele deve enviá-lo como parâmetro, passando o `“$event”`.



**Figura 25 – Eventos.**

```
<button v-on:click="eventTest('click')">On Click</button>  
<button v-on:click="eventTest">On Click Event Param Implicit</button>  
<button @click="eventTestParam('@click', $event)">On Click Event Param Explicit</button>
```

**Fonte: Elaborado pelo próprio professor.**

Além do evento de click com botão esquerdo do mouse, também existem outros eventos de mouse. Abaixo listamos alguns deles:

- **dblclick:** utilizado para capturar o duplo clique do mouse com o botão esquerdo.
- **click.left:** utilizado para capturar o click simples do mouse com o botão esquerdo, tem o mesmo efeito de utilizar somente o click.
- **click.right:** utilizado para capturar o click do mouse com o botão direito.
- **click.middle:** utilizado para capturar o click do mouse com o botão do meio.
- **mouseenter:** utilizado para realizar alguma ação quando o mouse passar sobre o elemento.
- **mouseleave:** utilizado para realizar alguma ação quando o mouse entrar em cima do elemento.

A imagem abaixo ilustra a utilização dos eventos listados acima. Uma outra funcionalidade interessante que o Vue.js possui é o modificador “prevent”. Uma ação muito comum em sistemas web é receber o evento do DOM como parâmetro para poder bloquear o funcionamento padrão dele. Por exemplo, no caso do click com o botão direito do mouse, é possível prevenir que o menu de opções padrão do navegador apareça. O modificador “prevent” do Vue.js tem essa função, de prevenir a execução

padrão do navegador nesses casos, evitando que o desenvolvedor tenha que pegar o evento manualmente e realizar esse tratamento. O terceiro exemplo da imagem abaixo ilustra esta utilização.

**Figura 26 – Eventos de Mouse.**

```
<button @dblclick="eventTest('dbl click')">On Double Click</button>
<button @click.left="eventMouseTest">Mouse Left</button>
<button @click.right.prevent="eventMouseTest">Mouse Right</button>
<button @click.middle="eventMouseTest">Mouse Middle</button>
<p @mouseenter="eventMouseTest" @mouseleave="eventMouseTest">mouse hover test</p>
```

**Fonte: Elaborado pelo próprio professor.**

Além dos eventos de mouse, também temos os eventos de teclado, representados por `keyup`, que é executado quando a tecla é solta, e `keydown` quando a tecla é apertada. Na frente do `keyup` ou `keydown`, deve ser informado o nome da tecla, o Vue.js já nos fornece algumas opções. Na imagem abaixo está ilustrada algumas opções, a maioria delas o próprio nome indica qual tela é. A tecla “meta” depende do sistema operacional, no Windows ela se refere à tecla do Windows, que geralmente fica entre o `Ctrl` e o `Alt`.

**Figura 27 – Eventos de Teclado.**

```
<input v-on:keyup.enter="eventKeyTest" placeholder="Press Enter">
<input v-on:keydown.tab="eventKeyTest" placeholder="Press Tab">
<input v-on:keyup.delete="eventKeyTest" placeholder="Press Delete">
<input v-on:keyup.esc="eventKeyTest" placeholder="Press Esc">
<input v-on:keyup.space="eventKeyTest" placeholder="Press Space">
<input v-on:keyup.up="eventKeyTest" placeholder="Press Up">
<input v-on:keyup.down="eventKeyTest" placeholder="Press Down">
<input v-on:keyup.left="eventKeyTest" placeholder="Press Left">
<input v-on:keyup.right="eventKeyTest" placeholder="Press Right">
<input v-on:keydown.ctrl="eventKeyTest" placeholder="Press Ctrl">
<input v-on:keydown.alt="eventKeyTest" placeholder="Press Alt">
<input v-on:keydown.shift="eventKeyTest" placeholder="Press Shift">
<input v-on:keydown.meta="eventKeyTest" placeholder="Press Meta">
```

**Fonte: Elaborado pelo próprio professor.**

Uma outra possibilidade interessante é associar o click do mouse com a tecla Ctrl. Para isso, basta utilizar o evento click com o Ctrl na descrição da diretiva. Nestes casos, um modificador útil é o “exact”. Ele faz com que o evento escute caso as teclas sejam exatamente as listadas, sem teclas adicionais pressionadas simultaneamente. A imagem abaixo ilustra essa utilização. No segundo exemplo, a ação será executada somente caso o click seja feito com a tecla Ctrl pressionada. Caso a tecla Alt também esteja pressionada no segundo exemplo, a ação não ocorrerá, por causa do modificador exact.

**Figura 28 – Ctrl com Click.**

```
<p @click.ctrl="eventTest('ctrl + click')">Ctrl + Click</p>
<p @click.ctrl.exact="eventTest('ctrl + click')">Ctrl + Click Exact</p>
<p @click.ctrl="eventTest('ctrl + click')" @click.alt="eventTest('alt + click')">Ctrl + Click / Alt + Click</p>
```

**Fonte: Elaborado pelo próprio professor.**

## 2.8. Estilização

No Vue.js, é possível vincular, a partir do data binding, propriedades a estilos dos componentes. Para isso, podemos utilizar a diretiva v-bind, realizando, então, o vínculo de uma propriedade ao atributo style ou class de um elemento.

A imagem abaixo ilustra alguns exemplos desta utilização. No primeiro exemplo, “textColor” é uma variável, e, ao utilizar o v-bind no atributo style, podemos passar o “textColor” diretamente para a propriedade “color”. Dessa forma, caso a propriedade “textColor” seja alterada, automaticamente o elemento também será.

No segundo exemplo da imagem, podemos ver uma outra opção de passar estilos para um elemento, que é enviando um objeto diretamente para o style através do bind. Quando estamos passando o estilo dessa forma, temos que utilizar as propriedades no formato camel case, pois se

trata de um objeto JSON, e quem fará a conversão para o formato do CSS será o próprio Vue.js.

No terceiro exemplo da imagem, uma classe é enviada utilizando o `v-bind` no `class`. No quarto exemplo, é passado pelo `bind` uma lista de classes, diretamente no elemento. No quinto exemplo, essa lista vem da propriedade. No último exemplo, é mostrado que também é possível trabalhar com expressões dentro do estilo, fazendo até condicionais para mostrar ou não a classe.

**Figura 29 – Estilos.**

```
<p v-bind:style="{ color: textColor }">inline style object</p>
<p v-bind:style="textColorObject">inline style object bind</p>
<p class="test-class-1" :class="classTest2">class bind</p>
<p :class="[classTest1, classTest2]">class list inline</p>
<p :class="classList">class bind list</p>
<p :class="[1 > 0 ? classList : classList]">class expression</p>

data() {
  return {
    textColorObject: {
      color: "green",
      fontSize: "25px"
    },
    classTest1: "test-class-1",
    classTest2: "test-class-2",
    classList: ["test-class-1", "test-class-2"]
  }
}
```

**Fonte: Elaborado pelo próprio professor.**

## 2.9. Componentes

A maioria dos frameworks JavaScript da atualidade trabalham muito com o conceito de componentes, e o Vue.js não é diferente. Desenvolver aplicações visando a componentização favorece a organização e a reutilização de código.

Os componentes no Vue.js são criados utilizando o conceito de “single file components”, no qual os códigos responsáveis pela estrutura do

componente, estilização e comportamento ficam todos no mesmo arquivo. Este tipo de organização é o padrão que obtemos ao criar uma aplicação utilizando a ferramenta de linha de comando do Vue.js, a Vue CLI.

O arquivo neste formato possui a extensão “.vue”, e é dividido em três partes. A primeira parte é o template, no qual é definida a estrutura da página. É neste local que o código HTML é escrito, juntamente com as diretivas do Vue.js que farão a ligação com o JavaScript. A segunda parte é o script, no qual é definido todo o código JavaScript do componente. Neste local, é codificado grande parte das funcionalidades que vimos anteriormente, como methods, computed properties e watchers, por exemplo. A terceira parte é o style, responsável pela estilização do componente, sendo o CSS em si. Se dentro da tag do style tiver a palavra “scoped”, indicará que aquele CSS só tem valor dentro do componente, caso contrário, ele valerá para toda a aplicação. É possível colocar no mesmo componente uma tag style com scoped e outra sem. A imagem abaixo ilustra a organização descrita acima.

**Figura 30 – Single File Component.**

```
<template>
| <h1>Component</h1>
</template>

<script>
export default {
}
</script>

<style scoped>
</style>
```

**Fonte: Elaborado pelo próprio professor.**

Geralmente, o arquivo de um componente é criado utilizando o PascalCase, ou senão o kebab-case. PascalCase é um padrão no qual se escreve o nome utilizando a primeira letra de cada palavra em caixa alta, enquanto o kebab-case se refere a escrever tudo em minúsculo e

separando as palavras com um traço. É recomendado que utilize o mesmo estilo em todo o projeto. Para utilizar um componente no template, ele é utilizado como se fosse uma tag e sempre em kebab-case, mesmo que ele tenha sido declarado utilizando o PascalCase. A imagem abaixo ilustra a utilização de um componente dentro do template.

**Figura 31 – Utilização de um componente.**

```
<template>  
  <my-component></my-component>  
</template>
```

**Fonte: Elaborado pelo próprio professor.**

No Vue.js, podemos registrar componentes de forma global ou de forma local. Ao registrar um componente de forma global, ele poderá ser acessado em todos os demais componentes da aplicação, sem a necessidade de realizar sua importação localmente em cada um deles. Ao importar e registrar um componente localmente, ele estará disponível somente naquele local. O recomendável é que os componentes sejam registrados globalmente somente quando realmente fizer sentido eles serem utilizados em todos os demais componentes, caso contrário, é preferível utilizar localmente. Para registrar um componente globalmente, basta utilizar o método componente diretamente na instância do Vue.js antes de realizar a sua montagem. A imagem abaixo ilustra este processo.

**Figura 32 – Componente global.**

```
const app = createApp(App);  
app.component('GlobalComponent', GlobalComponent);  
app.mount('#app');
```

**Fonte: Elaborado pelo próprio professor.**

Para registrar um componente localmente, basta passá-lo diretamente na propriedade “components”, dentro do objeto de configuração do componente, da mesma forma como fazemos com as

propriedades “data” e “computed”, por exemplo. A imagem abaixo ilustra este processo.

**Figura 33 – Componente local**

```
export default {  
  name: "App",  
  data: function () {  
    return {  
      title: "My Component",  
    };  
  },  
  components: {  
    LocalComponent,  
  }  
}
```

**Fonte: Elaborado pelo próprio professor.**

As duas imagens abaixo ilustram comportamentos importantes dos componentes no Vue.js. A primeira imagem se refere ao código de um componente, enquanto a segunda imagem se refere a sua utilização em outro componente.

O primeiro ponto a se observar nas imagens abaixo é a passagem de parâmetros utilizando props. Props é a forma de se enviar parâmetros de um componente para outro no Vue.js. Na segunda imagem, podemos ver que, ao utilizar o componente, é realizado o bind de uma propriedade chamada title. Essa propriedade, por sua vez, é utilizada na primeira imagem dentro do componente, como se fosse uma propriedade declarada no “data”. Para isso, é preciso realizar sua declaração na propriedade chamada “props”. É possível informar o tipo dessa propriedade que, no caso do exemplo, é string.

Outro comportamento a observar nas imagens abaixo é como ocorre a comunicação entre os componentes a partir de eventos. Na Figura 34, quando o usuário clica no botão de adicionar, é chamado um método “add”, que dispara um evento utilizando o “this.\$emit”. Essa função pode

ser utilizada para emitir eventos entre componentes, bastando passar o nome do evento e, caso seja necessário, os parâmetros como segundo argumento da função. Na Figura 35, podemos observar como é realizada a escuta do evento. Diretamente na utilização do componente é utilizada a diretiva `v-on` juntamente com o nome do evento, sendo assim possível associar um código para ser executado quando o evento ocorrer. Os parâmetros enviados podem ser capturados através do `$event`.

Outra funcionalidade importante que podemos observar nas imagens abaixo é a utilização dos slots. Com os slots, um componente consegue receber como parâmetro porções de template, que serão exibidas dentro do componente no local desejado. Na Figura 35, podemos observar o envio de dois slots com nomes específicos, `header` e `content`, e na Figura 34 podemos ver como utilizá-los, sendo possível inserir o código recebido no local do componente desejado. Informar o nome do slot é opcional, caso seja utilizado somente um slot, não há essa necessidade.

**Figura 34 – Declaração do Componente.**



```
<template>
  <h1>Local Component</h1>
  <h2>{{title}}</h2>
  <button @click="add">Add</button>
  <br/>
  <slot name="header"/>
  <slot name="content"/>
</template>

<script>
export default {
  props: {
    title: String,
  },
  methods: {
    add: function () {
      this.$emit('addclick', 2);
    }
  }
}
</script>
```

Fonte: Elaborado pelo próprio professor.

Figura 35 – Utilização do Componente.

```
<template>
  <local-component :title="title" @addclick="count += $event">
    <template v-slot:header>
      Slot Header
    </template>
    <template v-slot:content>
      Slot Content
    </template>
  </local-component>
</template>
```

Fonte: Elaborado pelo próprio professor.

Todo componente no Vue.js tem um ciclo de vida, e o framework fornece um método para cada estágio desse ciclo, sendo possível executar códigos em momentos específicos, que vão desde antes da criação do componente até a sua destruição. Abaixo estão listados os métodos do ciclo de vida de um componente no Vue.js. Para inserir códigos nestes

momentos, basta declarar o método dentro do objeto de configuração do componente, no mesmo nível das demais configurações.

- **beforeCreate:** executado no momento da inicialização do componente. Neste momento, o “data” ainda não está disponível.
- **created:** é executado depois que o componente é criado. Neste momento, o “data” já está acessível. Pode ser utilizado para realizar chamadas assíncronas, porém o Vue.js não aguarda a resolução delas antes de seguir a construção. Neste momento, o DOM ainda não está acessível, por isso não é possível criar um loading na tela para aguardar as requisições assíncronas.
- **beforeMount:** executado antes da primeira renderização do componente.
- **mounted:** executado depois que o componente foi montado, sendo possível acessar os elementos do DOM. Muito utilizado para realizar chamadas assíncronas, pois como neste momento o DOM está acessível, pode ser utilizado para mostrar um loading na tela enquanto requisições assíncronas realizadas não finalizam.
- **beforeUpdate:** executado antes que uma atualização de dados seja realizada.
- **updated:** executado após as propriedades do componente e o DOM serem atualizados.
- **beforeDestroy:** executado antes da destruição do componente. Muito utilizado para limpar listeners ou eventos que estavam sendo escutados.

- destroyed: executado quando o componente foi destruído.

## 2.10. Vue Router

O roteamento em aplicações do tipo single page application é muito importante, pois permite que a aplicação faça o efeito de trocar de página, até mesmo trocando a URL no navegador, porém sem efetuar um refresh na tela. Isso ocorre pois o roteador é capaz de sincronizar a URL com o componente em exibição no momento. Isso é muito bom para a usabilidade do usuário, porque permite que ele acesse uma parte da aplicação diretamente a partir da URL.

O Vue.js possui uma biblioteca oficial para roteamento, chamada Vue Router. Ela não vem por padrão acoplada ao framework. Para utilizá-la, é preciso instalá-la a partir do NPM ou via inserção de script na página. O nome do pacote no NPM é “vue-router”. A imagem abaixo ilustra como pode ser realizada a importação e a configuração do Vue Router no projeto, utilizando 3 rotas como exemplo.

**Figura 36 – Vue Router.**

```
import { createApp } from 'vue'
import App from './App.vue'
import PageA from './components/PageA';
import PageB from './components/PageB';
import PageC from './components/PageC';
import { createRouter, createWebHashHistory } from "vue-router";

const router = createRouter({
  history: createWebHashHistory(),
  routes: [
    { path: "/pageA", component: PageA },
    { path: "/pageB", component: PageB },
    { path: "/pageC", component: PageC },
  ]
});

const app = createApp(App);
app.use(router);
app.mount('#app');
```

**Fonte: Elaborado pelo próprio professor.**

Após realizar a importação e a configuração, o router pode ser acessado utilizando “this.\$router”. A imagem abaixo ilustra formas de realizar a navegação a partir do objeto do roteador. O método “push” empilha a página no histórico, ou seja, se o usuário clicar em voltar, ele voltará para a página que estava anteriormente. O método “replace” substitui a página atual pela da navegação e, caso o usuário clique em voltar, ele não voltará para a página que estava anteriormente, pois ela terá sido substituída no histórico.

**Figura 37 – Navegação Vue Router.**

```
goToPageA() {  
  this.$router.push("/pageA");  
},  
replacePageA() {  
  this.$router.replace("/pageA");  
}
```

**Fonte:** Elaborado pelo próprio professor.

## Referências

---

2020 Developer Survey. **Stack Over Flow**. Disponível em: <https://insights.stackoverflow.com/survey/2020>. Acesso em: 18 mar. 2022.

ANGULAR vs. react vs vue. **NPM Trends**. Disponível em: <https://www.npmtrends.com/angular-vs-react-vs-vue>. Acesso em: 18 mar. 2022.

VUE.JS. Disponível em: <https://v3.vuejs.org/>. Acesso em: 18 mar. 2022.