Vue

Bootcamp: Desenvolvedor Front End

João Henrique Abreu

2020

Vue

Bootcamp: Desenvolvedor Front End

João Henrique Abreu

Sumário

Capítulo 1. Background Vue.js 5
Por que escolher o Vue? 5
Fatores para escolher qual framework utilizar em seu projeto 7
Estado atual do Vue 8
Quem usa? 9

Capítulo 2. Fundamentos de Vue 10
Pré-requisitos 10
Instalação 11
Estrutura do projeto 14
A instância Vue 16

Capítulo 3. Componentes 18
Estrutura de um componente 18
Ciclo de vida de um componente 22
Propriedades de um componente 24

Capítulo 4. Layout dinâmico – Diretivas 32
O que é reatividade? 32
Reatividade aprofundada 33
Diretivas 34

Capítulo 5. Roteamento 39
vue-router 39
Instalação 40
Configuração das rotas 41

Navegando 42

Rotas privadas 44

Capítulo 6. Estilo 46

Vuetify 47

Referências 52

Capítulo 1. Background Vue.js

Vue é um framework Javascript criado por Evan You Ex-google em 2015. Foi amplamente difundido em projetos de aplicações web interativas, se tornando já o queridinho do mercado e uma ferramenta poderosa para desenvolvedores que buscam agilidade e uma boa experiência de programação.

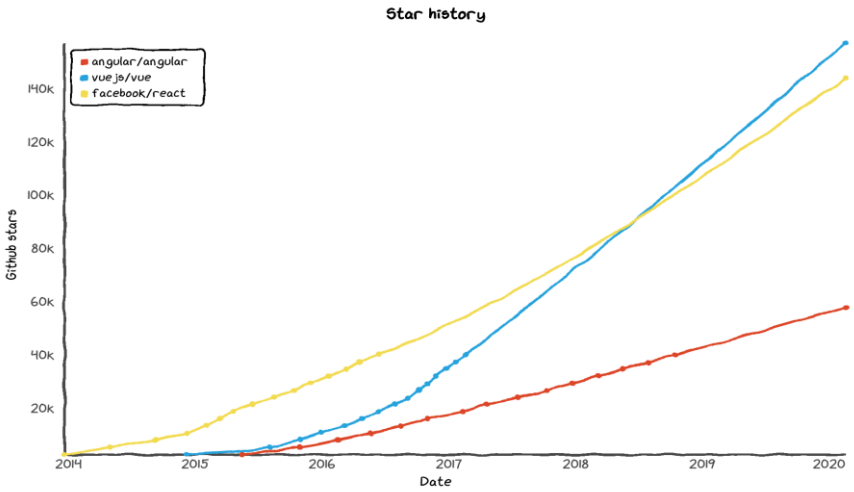
Por que escolher o Vue?

Sabemos que o Vue é o queridinho do mercado, framework mais amado etc., mas o que realmente o torna tão especial assim? Listamos alguns dos principais fatores que fazem os desenvolvedores realmente amarem o Vue:

- Resposta rápida (renderização seletiva), biblioteca otimizada (20kb), altamente escalável;
- Pode ser adotado em novos projetos ou incrementalmente em sistemas existentes;
- Rápida curva de aprendizado, alto desacoplamento de responsabilidades, comunidade ativa e diversas bibliotecas e extensões;
- Documentação localizada e detalhada, linguagem focada no desenvolvedor e criado a partir das experiências negativas dos demais.

Outro fator interessante, mas muito relevante na hora de escolher o seu framework, é o feedback da comunidade e sua avaliação no github. Vue conquistou o título de “framework mais amado” por ter o maior número de estrelas (Daityari, 2020). Mesmo que com uma quantidade menor de downloads, o Vue possui uma maior quantidade de avaliações positivas (estrelas) em seu repositório (Vue Official Documentation, 2020).

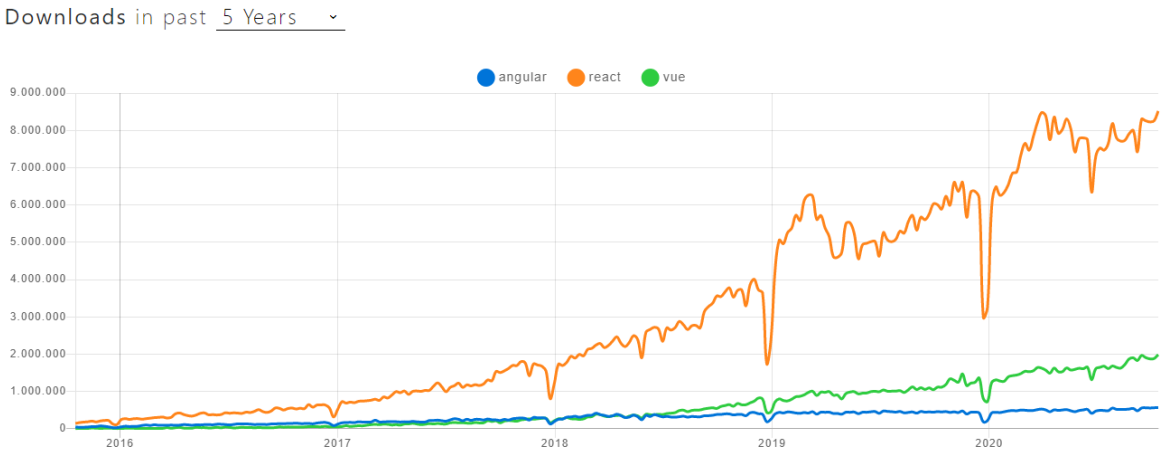
Figura 1 - Número de estrelas nos repositórios do Github.



Isto demonstra que o objetivo dos fundadores do Vue, que é dar a melhor experiência de desenvolvimento ao desenvolvedor, está sendo cumprido com méritos.

Em comparação com os demais frameworks, o Vue, apesar de mais recente, já se encontra em 2º lugar dos frameworks javascript mais baixados (NPM trends, 2020):

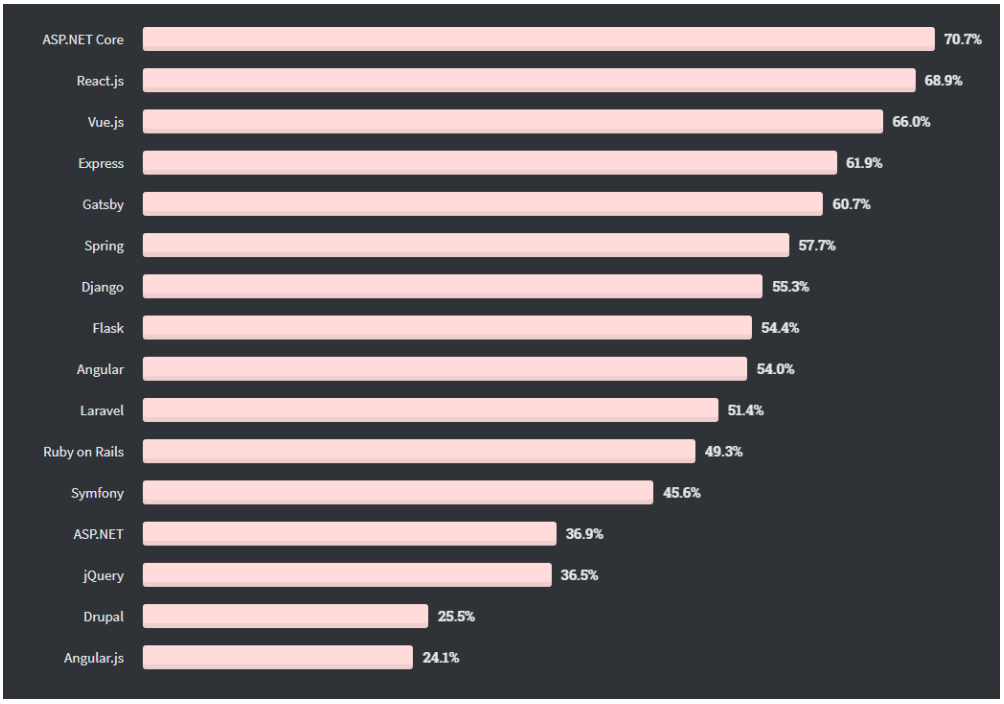
Figura 2 - Número de downloads dos principais frameworks javascript.



Todo ano, o site StackOverflow, referência em fórum de discussão sobre programação, divulga sua pesquisa anual sobre o mercado de desenvolvimento de

software, e confirmou a estatística de apreciação (most loved framework) dos desenvolvedores sobre o Vue (StackOverflow, 2020):

Figura 3 - Pesquisa StackOverflow - Most Loved Frameworks.



Fatores para escolher qual framework utilizar em seu projeto

A escolha do framework é fundamental para sucesso de um projeto. Fatores de negócios, como familiaridade da equipe e até mesmo custo e oferta de contratação de desenvolvedores, devem ser levados em consideração. No Bootcamp Frontend, tratamos dos 3 principais frameworks Javascript disponíveis no mercado. Vamos apresentar aqui alguns itens que devem ser levados em consideração ao escolher qual frontend escolher para seu projeto ou para investir tempo e dedicação para incrementar sua carreira:

- Familiaridade com o framework;
- Alinhamento das forças do framework e adaptabilidade com as principais funcionalidades do seu projeto;

- Capacidade de escalabilidade do framework x necessidade de escalabilidade do projeto;
- Extensões, tamanho e atividade da comunidade, duração do suporte, apoiadores, portabilidade etc.;
- Custos de contratação de programadores (quanto mais específicos, mais caros);
- Oferta de desenvolvedores no mercado.

Estado atual do Vue

O Vue se encontra em pleno desenvolvimento e suporte, tendo lançado recentemente uma nova versão (major) a 3.0 com uma série de melhorias de desempenho e novas funcionalidades mais poderosas para incrementar a experiência do usuário (Vue Official Documentation, 2020).

Figura 4 - Versões do Vue.

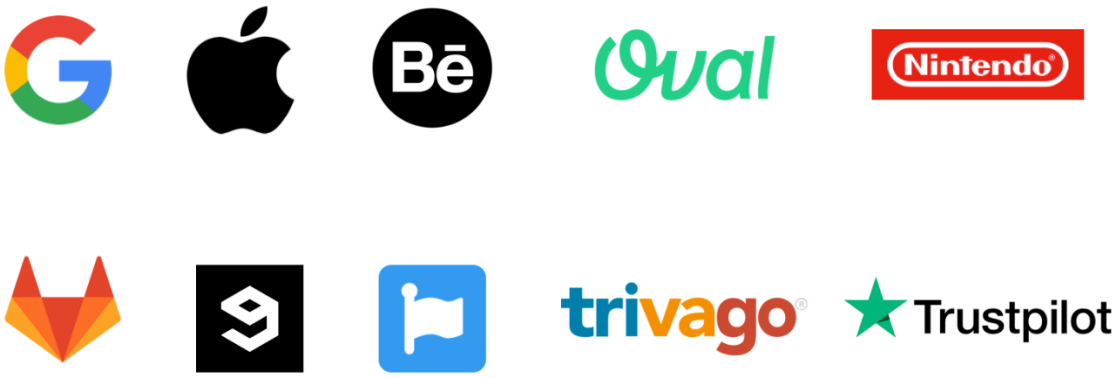
Versions [edit]

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

Quem usa?

Uma série de gigantes do mercado já adotaram o Vue totalmente ou parcialmente em seus projetos, demonstrando, assim, a maturidade do framework (Spezzano, 2019) e (Made with Vue, 2019).

Figura 5 - Grandes empresas que usam Vue.



Vue já é um framework maduro, utilizado em diversos projetos já em produção. Conta com uma comunidade apaixonada e engajada – o que é extremamente importante.

Ele apresenta boas justificativas para ser adotado em novos projetos ou migrar gradativamente para obter maior interatividade e melhorar a experiência do usuário.

Com sua sintaxe simples e intuitiva, apresenta um crescimento muito interessante e deve continuar na prateleira de cima dos grandes frameworks da web.

Capítulo 2. Fundamentos de Vue

Neste capítulo, iremos abordar um pouco os conceitos básicos do Vue, os primeiros passos para se inicializar uma aplicação com o framework, assim como um pouco de suas engrenagens que nos permite utilizar uma sintaxe simples, porém poderosa, para criar layouts extremamente interativos e compatível com diversos browsers.

Pré-requisitos

Para confeccionar este material, levamos em consideração que o leitor já possua alguns conhecimentos prévios, que então não iremos estender ou aprofundar, assumindo que o mesmo já possui familiaridade com os seguintes temas:

- Javascript;
- HTML;
- CSS;
- npm;
- Servidores web;
- Estrutura básica para publicar um projeto web;
- Orientação a objetos;
- Git e controle de versão.

É recomendado também algum conhecimento em inglês, pois mesmo a documentação localizada para português, sua sintaxe, métodos, conceitos e propriedades estão todas em inglês e utilizaremos bastante as palavras-chave do Vue em sua forma nativa (e não traduzida), para expressar melhor seu conteúdo e contexto.

Instalação

Vue, assim como diversas bibliotecas javascript, pode ser instalada de diversas formas:

- Download do código e inclusão no corpo do HTML;
- CDN do Vue;
- npm;
- CLI.

A versão de produção pode ser baixada, incluída via CDN ou usando NPM (requer configuração base) segundo as instruções em (Vue Official Documentation, 2020). Aqui iremos abordar a linha de comando do Vue, que já nos entrega uma estrutura pronta para iniciar um projeto com Vue.

Primeiramente, devemos instalar a cli globalmente no PC para podermos utilizar a linha de comando.

Figura 6 - Instalando a CLI.

```
> npm install -g @vue/cli
```

Teste se a CLI está instalada corretamente rodando o comando vue –version, deve-se exibir a versão atual da CLI no console. Finalmente, utilize o comando para criar um projeto novo projeto do Vue (o “.” serve para criar o projeto no diretório atual).

```
> vue create .
```

A interface fará uma série de perguntas (se deseja utilizar uma configuração pré-definida ou escolher manualmente), para configuração básica do projeto e instalação de bibliotecas adicionais. As opções são:

- Seleção pré-definida (versão 2.x ou 3.0);
- Manual:
  - Instalar o Babel;
  - Instalar suporte a Typescript;
  - Instalar suporte a PWAs (progressive web apps – não será coberto no curso);
  - Instalar o vue-router;
  - Instalar o vuex;
  - Instalar pré-processadores CSS;
  - Instalar o linter e formatador de código;
  - Instalar suporte para testes unitários;
  - Instalar suporte para testes integrados (E2E).

Além disso, já por padrão, é inicializado um repositório git no diretório selecionado para a instalação do projeto. Todas as opções acimas já realizam a pré-configuração dos pacotes selecionados também. Em nosso curso, iremos utilizar as seguintes bibliotecas (e iremos explorar de forma mais aprofundada):

- **Babel, vue-router e pré-processadores CSS.**

Em um projeto mais complexo utilizando Vue, é bastante interessante utilizar as demais bibliotecas.

- **Babel**

Junto com o Webpack, é responsável pela compilação do código na sintaxe própria do Vue, com seus markups para um código javascript compatível com browsers que suportam a partir de ES5 (provavelmente 90%+ dos browsers e suas versões atuais).

- **Typescript**

Permite escrever código Vue e sua lógica utilizando Typescript. Typescript é uma alternativa ao javascript, porém com *tipagem forte*.

- **PWA**

Progressive web apps são aplicações híbridas cujo objetivo é funcionar tanto como uma aplicação web, quanto um aplicativo mobile (enviando notificações para o celular etc.).

- **vue-router**

Biblioteca de roteamento do Vue. Responsável pelo funcionamento SPA (S*ingle Page Application*) e navegação entre páginas em um projeto do Vue. Iremos explorar bastante esta biblioteca.

- **vuex**

Gerenciamento do estado de uma aplicação. Funciona como um repositório de modelos global, que pode ser compartilhado entre os diversos módulos e componentes do Vue.

- **linter**

Biblioteca utilizada para formatação de código, conforme as melhores práticas e recomendações do mercado. Com o suporte do Visual Studio, pode não somente logar erros de formatação como também corrigir os menos severos automaticamente.

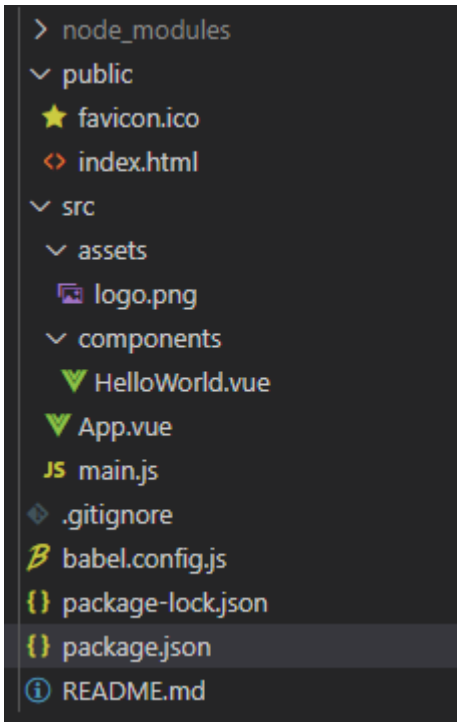
- **Testes unitários e Testes Integrados**

Instalação e configuração base para bibliotecas para automação de testes unitários (mocha, jest, etc.) e testes integrados (cypress, selenium etc.).

Estrutura do projeto

Uma vez instalado o projeto Vue, via CLI, recebemos a seguinte estrutura pré-definida:

Pastas (expandido):



Vamos analisar os principais arquivos gerados pelo Vue:

- **package.json e package-lock**

Para aqueles que estão familiarizados com os projetos javascript e utilizando npm, o arquivo package é o principal arquivo, contendo a configuração das bibliotecas utilizadas, informações sobre o projeto e comandos de entrada (rodar servidor, criar distribuição para produção etc.).

- **/src/main.js**

Ponto de entrada do Vue e principal arquivo de configuração da instância. Responsável pela configuração global, gerenciamento das bibliotecas e plugins, inicialização da instância do Vue. Caso haja plugins, filtros, componentes e outras

bibliotecas que devem ser instaladas no escopo global do Vue, deve ser feito neste arquivo (iremos abordar todos esses conceitos nos próximos capítulos).

- **/src/App.vue**

Arquivos com extensão .vue são tratados como componentes no Vue, e este aqui é o componente de entrada que será a base para a toda a aplicação Vue (o componente pai de todos os componentes derivados do projeto). Note que ele é referenciado no arquivo *main.js* e utilizado como alvo da renderização.

- **/src/components**

Pasta sugerida (pode ser alterada livremente) para armazenar os componentes (pense nos componentes como classes) que carregam todo o código e funcionalidades de fato de nosso projeto Vue.

- **/src/public**

A maioria dos servidores web (apache, nginx etc.) buscam um arquivo de entrada html para inicializar a renderização do conteúdo. Para uma configuração básica do Vue, utiliza-se (já previamente configurado pela CLI) o index.html pelo servidor do npm. Este arquivo também é importante, pois contém o elemento base para renderização (pelo Vue) de todo o conteúdo dinâmico que será exibido pela página.

- **/src/assets**

Pasta recomendada para inclusão dos assets como estilos globais, bibliotecas javascript específicas (que não serão gerenciadas pelo npm).

- **/node_modules**

Pasta de instalação das bibliotecas e suas dependências utilizadas pelo npm (já vem ignorada pelo git por padrão – recomenda-se manter assim).

Tudo se inicia com a criação de uma nova instância Vue, dentro de um determinado elemento HTML. Ao inicializar sua instância Vue, você deve indicar um elemento de sua página que irá renderizar todo o conteúdo daquela instância do Vue. Todo conteúdo é renderizado dinamicamente (após a página ser carregada). Utilizando a CLI, o Vue já nos entrega essa configuração pronta:

Arquivo **main.js:**

Figura 7 - Inicialização da instancia do Vue.

```
const vm = new Vue({
  render: h => h(App)
}).$mount('#app');
```

Figura 8 - Sintaxe alternativa para inicialização da instância Vue.

```
const vm = new Vue({
  router,
  el: '#app',
  render: h => h(App)
});
```

Podemos notar aqui alguns pontos importantes:

- Atribui-se a um elemento (el) com id “#app” a instância Vue. Este é o processo de renderização. Todo o conteúdo proveniente dos componentes que criarmos em nosso projeto será injetado pelo Vue dentro deste elemento *app* (pode-se alterar esse id a gosto). Este elemento pode ser localizado no arquivo index.html;
- Atribuímos a uma função *render* um objeto chamado App. Este objeto é o nosso componente base (veja a sintaxe de importação na parte inicial do arquivo), que será o ponto de entrada para renderização de todo o conteúdo do nosso projeto. A partir dele, podemos importar novos componentes, renderizar páginas sob

demanda, configurar um layout base para a aplicação e inúmeros outros cenários (ou simplesmente escrever todo nosso projeto nele – o que não recomendamos);

- Uma vez inicializada a instância do Vue, ela pode ser acessada nos componentes via *this* ou pelo objeto vm criado.

Para rodarmos nossa aplicação e vermos o resultado, vamos utilizar o comando “serve” configurado no arquivo package.json:

```
> npm run serve
```

Isto irá inicializar o servidor web (de desenvolvimento) do npm. Recomendamos rodar a aplicação (por padrão, ela deve estar disponível no endereço http://localhost:8080 – observe no log do npm o resultado da inicialização do seu projeto) e explorar o código fonte da página gerada. Recomendamos também que faça alterações no conteúdo do arquivo App.vue para visualizar melhor como o código de um componente é **injetado** em uma página web.

Capítulo 3. Componentes

Mencionamos os termos componentes diversas vezes nesta apostila e em nossas aulas. Os componentes são a base fundamental do Vue, o coração de toda aplicação escrita em Vue js. Vamos entender profundamente a natureza dos componentes, sua sintaxe, estrutura e como organizar nosso código da melhor maneira, respeitando os conceitos SOLID de programação e como os componentes contribuem fortemente para termos um código otimizado, de fácil manutenção, poderoso e bastante elegante.

Estrutura de um componente

Todo componente do Vue é composto por 3 partes fundamentais:

Figura 9 - Estrutura base de um componente.

- **<template>**

Responsável pela definição do layout do componente. Suporta utilização de código HTML puro (inclusive HTML5) além das marcações específicas da linguagem

Vue (como utilização de componentes customizados) que se unificam ao código HTML incorporado no template, e é injetado dentro dos elementos alvo onde serão renderizados.

Uma das principais características dos frameworks javascript da atualidade é a sua capacidade de renderizar conteúdos dinamicamente. Aqui no template, podemos fazer isso a partir do vínculo de dados com sintaxe Mustache {{ }} (Mustache, 2020).

Veremos à frente que todo código escrito dentro de um componente recebe observadores do Vue que reagem à medida que estes elementos têm seu estado alterado, seja por alguma ação do usuário, ou evento disparado propositalmente pelo seu código. A sintaxe aqui nos permite também manipular da forma como quisermos a renderização do layout do componente. Pode-se (a partir de diretivas) modificar o comportamento de elementos, alterar classes dinamicamente, habilitar ou desabilitar elementos, condicionar a renderização de um elemento a uma condição, mudar posicionamento, encaixe no grid, enfim, uma infinidade de possibilidades de manipulação do DOM do componente.

Iremos utilizar diversas propriedades do nosso componente enquanto definimos o nosso template. Para isso, acessamos a instância do componente, geralmente via *this*, porém uma facilidade que o Vue nos oferece é que não é necessário utilizar o atributo *this* dentro de um template, pois todo o template está vinculado a instância do componente. O *this* aqui está implícito nas chamadas a propriedades ou métodos existentes da instância (ao declarar uma variável aqui, ela deve estar previamente registrada no componente).

Figura 10 - Exemplo de código de template.

```
<template>
  <div>
    <h3>Fila de Banho</h3>
    <table>
      <tr>
        <th>Nome</th>
        <th>Idade</th>
        <th>Raça</th>
        <th>Tosa?</th>
        <th></th>
      </tr>
      <tr v-for="(cliente, index) in clientes" :key="index">
        <td>{{ cliente.nome }}</td>
        <td>{{ cliente.idade }}</td>
        <td>{{ cliente.raca }}</td>
        <td>{{ cliente.servico.extra | pergunta }}</td>
        <td><button @click="realizarServico(cliente)">Dar banho</button></td>
      </tr>
    </table>
  </div>
</template>
```

- **<script>**

Aqui vem a lógica do componente. Tudo que virá a ser utilizado para alterar o estado do componente deve ser declarado aqui, sejam parâmetros de entrada, propriedades internas, funções, cálculos, condições, formatações, observadores de estado etc. Se desejamos adicionar algum comportamento no layout, que será utilizado no <template>, o mesmo deve ser definido aqui no script (é possível adicionar lógicas rápidas no template, mas recomendamos que caso a lógica ultrapasse uma linha de código, ela seja adicionada aqui dentro). Podemos também importar e registrar novos componentes filhos que serão utilizados pelo respectivo componente.

O script também nos permite um vínculo com a criação da instância do Vue. Chamamos isso de *lifecycle hooks* (gatilhos de ciclo de vida). Esses *hooks* são extremamente poderosos, pois nos permitem alterar o estado e a forma como o componente é renderizado em diversos instantes de sua inicialização, permitindo alterar o DOM de forma mais otimizada, chamar APIs externas, condicionar a renderização a determinados cenários etc.

Figura 11 - Exemplo de código em um <script>.

```
<script>
import axios from 'axios';
import Cachorro from '@/models/cachorro'

export default {
  data() {
    return {
      cliente: new Cachorro(),
      racas: [],
      servicos: []
    }
  },
  async mounted() {
    try {
      this.racas = await this.buscarRacas();
      this.servicos = await this.buscarServicos();
    } catch (error) {
      console.log(error);
    }
  },
  methods: {
    async buscarRacas() {
      const { data } = await axios.get('http://localhost:3000/racas');
      return data;
    },
    async buscarServicos() {
      const { data } = await axios.get('http://localhost:3000/servicos');
      return data;
    },
    cadastrarCliente() {
      if (this.cliente.nome.length === 0 || this.cliente.idade === 0 || this.cliente.raca.length === 0) {
        alert('Favor preencher todas as informações do cadastro do cliente');
      }

      this.$emit('novo-cadastro', this.cliente);
      this.cliente = new Cachorro();
    }
  }
}
</script>
```

- **<style>**

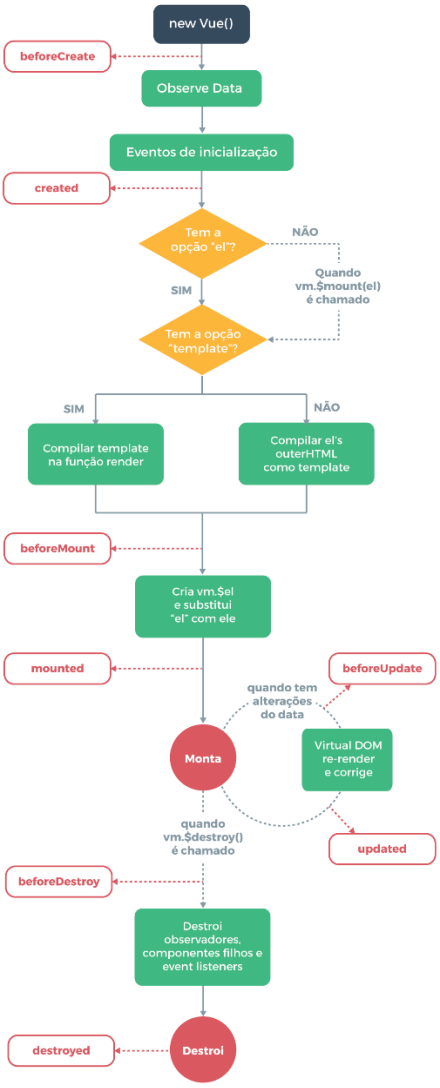
Finalmente e não menos importante, a tag de style permite declararmos estilos (com CSS ou utilizando pré-processadores) exclusivos para o componente. Chega de arquivos CSS quilométricos ou dezenas de CSSs para modificar o layout de sua aplicação. Com a tag style – utilizando a propriedade *scoped* – desacoplamos os estilos de uma camada externa, trazendo-o para o contexto do componente, facilitamos a gestão de regras específicas para determinados elementos, deixando o código mais enxuto e de fácil manutenção.

Nota: é também possível declarar estilos que serão utilizados por todos os componentes.

Ciclo de vida de um componente

Como mencionado anteriormente, o Vue disponibiliza *lifecycle hooks* (gatilhos) que podemos utilizar para modificar o estado do componente em diversos momentos da sua criação (de acordo com a nossa necessidade) (Vue Official Documentation, 2020).

Figura 12 - Ciclo de vida da instância do Vue.



Hooks implementados possuem acesso à toda instância do componente a partir do seu contexto *this.* São implementados como funções dentro do <script>. Vamos explorar cada um deles a seguir.

Figura 13 - Exemplo de código de um hook.

```
async created() {
  try {
    this.racas = await this.buscarRacas();
    this.servicos = await this.buscarServicos();
  } catch (error) {
    console.log(error);
  }
},
```

- **beforeCreate() e created()**

Utilizado geralmente para chamadas de API e requisição de dados externos, este *hook* é invocado em seguida da inicialização da instância. Neste momento, a instância não está completamente preparada e não permite alterações no DOM, pois o DOM virtual ainda não foi renderizado na instância do componente.

As chamadas a APIs externas aqui são interessantes e recomendadas, uma vez que os dados recebidos do servidor podem afetar e modificar o DOM que será gerado pelo nosso componente, evitando assim que o mesmo seja renderizado uma única vez e nos poupe recursos no lado do cliente, evitando renderizações desnecessárias.

Importante reforçar que, apesar de recomendada, a chamada a APIs externas não é obrigatória aqui (vai do contexto de cada aplicação) e pode também ser chamada no próximo *hook.*

- **beforeMount() e mounted()**

Chamada após a montagem do DOM, permitindo interações e manipulações diretas a ele (acesso a $el).

Figura 14 - Log do console do browser do $el nos hooks created e mounted.

```
undefined                                          Home.vue?76f2:37
                                                   Home.vue?76f2:40
▼<div class="home">
    <img alt="Vue logo" src="/img/logo.82b9c7a5.png">
  ▶<div class="row">…</div>
    <div class="espaco-abaixo"></div>
    <div class="espaco-abaixo"></div>
    <hr>
  ▶<div data-v-788fbde6>…</div>
  </div>
```

- **beforeUpdate() e updated()**

Chamada após a re-renderização do componente devido a alguma mudança de estado e notificações disparadas aos observadores.

- **beforeDestroy() e destroyed()**

Invocado logo antes ou depois que a instância do componente for destruída. Interessante para fazer algum tipo de limpeza (além daquela que o próprio Vue realiza).

- **errorCaptured()**

Invocado quando algum erro é capturado pela instância. Permite tratamento customizado do erro (página personalizada, mensagem, API de monitoramento).

Sintaxe: errorCaptured(error, component, trace) { }

Propriedades de um componente

O <script> possui uma estrutura pré-definida para podemos construir nosso componente que chamaremos de atributos do componente. Podemos dizer que elas são a sintaxe base da lógica do Vue, cada um com sua finalidade específica, realizando uma operação para registrar demais atributos do componente, modificar o estado do componente ou reagir a eventos do componente, ou que o componente/elementos do componente sejam alvos. Todos os atributos do componente que listaremos aqui são montadas na raiz da instância, portanto podemos acessá-los a partir de outros atributos, via *this.** (diferentemente do template aqui é obrigatório o uso do *this*, caso necessário utilizar algum atributo da instância do componente ou global do Vue).

Figura 15 - Os métodos criados buscarRacas e buscarServicos estão vinculados ao *this*.

```
async created() {
  try {
    this.racas = await this.buscarRacas();
    this.servicos = await this.buscarServicos();
  } catch (error) {
    console.log(error);
  }
},
```

- **props**

São os parâmetros de entrada do componente fornecidos pelos componentes externos (ou via instância global do Vue) que declaram este em seus escopos. Aceitam quaisquer tipos de dados primitivos do Javascript.

Um conceito importante que iremos falar pela primeira vez aqui é o da reatividade (vamos explorar extensivamente no próximo capítulo), mas, em resumo, é a capacidade que o Vue injeta em seus elementos de reagir a uma alteração de seus estados. E um recurso muito poderoso das props é que elas também fazem parte do escopo de reatividade. Ou seja, alterações externas nas props oriundas de outros componentes externos ao contexto deste (que declarou a prop) gatilham a reatividade e consequentemente permitem reações e re-renderizações, modificando automaticamente o estado deste componente. Não se preocupe, vamos aprofundar bastante nos conceitos de reatividade em breve.

Podemos declarar as props em duas sintaxes diferentes:

Figura 16 - Declarando props como um array (tipagem fraca).

```
<script>
export default {
  props: ['clientes'],
```

Figura 17 - Declarando props como um objeto (tipagem forte).

```
<script>
export default {
  props: {
    clientes: { type: Array, default: () => {} }
  },
```

Na declaração de props como um objeto, será lançado um erro de tipagem, caso seja fornecido um valor de tipo diferente para a propriedade (o que não acontece na declaração como *array* – tipagem fraca).

Figura 18 - Exemplo de utilização de uma prop (*this.clientes*) no <script> em outro atributo do componente.

```
totalServicos() {
  if (!Array.isArray(this.clientes)) { return 0; }
  return this.clientes.reduce((total, cliente) => total + cliente.servico.preco, 0);
}
```

- **data()**

Propriedades internas do componente. Aquilo que irá carregar um valor, ser utilizado no template ou em outros atributos do componente, e não oriundo de um componente externo, deve ser declarado como *data*.

Importante! (Vue Official Documentation, 2020) Diferentemente das props e computed, o *data* deve ser declarado como uma função para que cada componente possa manter sua cópia daquela estrutura de dados (caso contrário, todas seriam compartilhadas por referência entre os componentes, e cada alteração de estado em uma causaria uma re-renderização de todos os componentes).

Figura 19 - Exemplo de declaração do atributo data.

```
export default {
  data() {
    return {
      cliente: new Cachorro(),
      racas: [],
      servicos: []
    }
  },
```

- **computed**

Propriedades computadas como cálculos, concatenação de propriedades etc. São *getters* e *setters* customizados para combinar propriedades de um componente. Propriedades computadas também são reativas, e caso haja alguma alteração de seu estado ou de qualquer propriedade dentro do seu escopo, serão atualizadas causando a re-renderização nos elementos onde estiverem declaradas ou associadas.

Aqui é importante ressaltar que se declararmos uma propriedade computada como uma arrow function o *this* (pela natureza das *arrow functions*, onde o *this* está vinculado ao escopo de onde ela é chamada e não necessariamente ao escopo do objeto em que ela está declarada) em seu escopo não será a instância do Vue. Recomendamos que sempre que se precisar utilizar uma *prop*, data, *method* etc. aqui (ou em qualquer atributo do componente) deve-se declarar com a sintaxe tradicional ES5 (*function () { ... }*) (Vue Official Documentation, 2020)**.**

Figura 20 - Exemplo de uma propriedade computada.

```
computed: {
    totalServicos() {
        if (!Array.isArray(this.clientes)) { return 0; }
        return this.clientes.reduce((total, cliente) => total + cliente.servico.preco, 0);
    }
}
```

- **methods**

Métodos que adicionam funcionalidade ao componente. Utilizados para diversos propósitos, principalmente reação a eventos.

Figura 21 - Exemplo de um método no <script> do componente.

```
cadastrarCliente() {
    if (this.cliente.nome.length === 0 || this.cliente.idade === 0 || this.cliente.raca.length === 0) {
        alert('Favor preencher todas as informações do cadastro do cliente');
    }

    this.$emit('novo-cadastro', this.cliente);
    this.cliente = new Cachorro();
}
}
```

Muitas vezes *methods e computed properties* entregam o mesmo resultado (como por exemplo, concatenar duas propriedades e exibi-las em caixa alta – pode ser alcançado tanto como uma *computed* como um *methods*). Mas devemos explorar o propósito de cada uma e explorar individualmente suas responsabilidades, portanto:

- *Computed* se comportam como propriedades e devem ser utilizadas para este fim. *Computed* não aceitam parâmetros (uma diferença bem relevante dos *methods*);
- *Methods* devem ser utilizados como funções, seja internamente no <script> ou no <template> (reagindo a eventos de clique, hover etc. – mas não limitado somente a isso!).
- **watch**

Cria um observador customizado para uma propriedade que é chamado toda vez que houver uma alteração no estado ou valor daquela propriedade. Qualquer propriedade da instância pode ser associada a um watcher (inclusive *computed properties*).

Figura 22 - Exemplo de um watcher (escondendo o menu em toda mudança de rota).

```
watch: {
  $route(to, from) {
    this.displaySubmenu = false;
  }
}
```

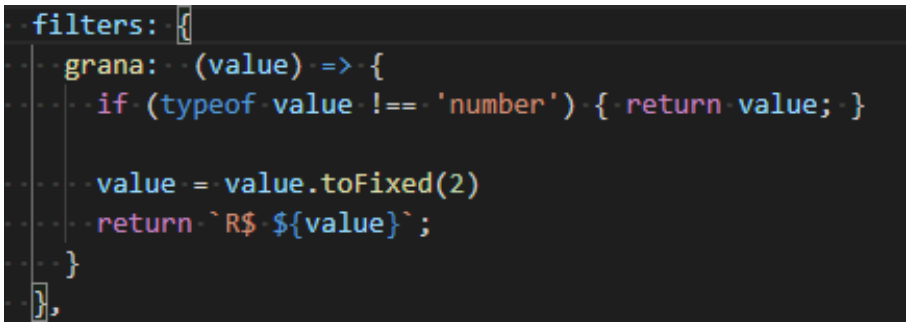
- **filters**

Filters são declarados como funções, recebem um valor (escalar) e retornam esse valor alterado. Utilizados geralmente para formatação de texto. Possui uma sintaxe especial no <template> para ser declarado com "|".

Figura 23 - Utilização de filters no template.

```
<td>Total</td>
<td colspan="5">{{ totalServicos | grana }}</td>
</tr>
```

Figura 24 - Exemplo de filtro (para formação de campos monetários).



```
filters: {
  grana: (value) => {
    if (typeof value !== 'number') { return value; }

    value = value.toFixed(2)
    return `R$ ${value}`;
  }
},
```

- **components**

Registro de componentes externos para serem usados no componente. Aqui vem um registro importante. Como vimos na estrutura base do Vue, importamos um componente (App.vue) em nosso arquivo *main*. Isso pode ser repetido para outros componentes da aplicação. Geralmente, importa-se globalmente componentes que são utilizados com muita frequência (para que não seja necessário importá-los novamente toda vez que for utilizar em um outro determinado componente).

Então, podemos registrar outros componentes localmente em um componente (com a propriedade *components*) ou globalmente. Utilizar o primeiro caso sempre que for um componente que não será reutilizado com frequência (específico para um determinado componente).

- **extends**

Herança entre componentes. Permite omitir as partes principais do componente e sobrescrever somente aquela funcionalidade que precisa. Funciona bem para <script>, mas se declarar <template> ou <style> eles serão substituídos por completo no componente filho.

- **mixins**

Blocos reutilizáveis de código Vue que aceitam declarações de opções do componente como created, data, methods etc. São fundidas com o código do componente e em caso de alguma funcionalidade sobreposta, ela será fundida como

um assign de objetos do Javascript puro, sendo as funcionalidades/chaves declaradas no código do componente tendo precedência sobre o mixins.

- **plugins**

Adiciona funcionalidades globais para o Vue (como diretivas, filtros, transições etc.). A partir da implementação da interface install, você terá acessa a instância do Vue e poderá modificá-la conforme sua necessidade. Deve ser registrado usando Vue.use no arquivo de entrada do vue (main.js) para que os componentes possam acessar suas funcionalidades. Exemplos de plugins: vue-router, vuetify, vuex, vue-sentry, vue-meta, vue-dotenv, vue-carousel, vue-perfect-scrollbar e centenas de outros.

- **slots**

Slots não são propriedades declaradas de um componente (acessadas no <script>), mas são uma parte fundamental para a customização de um componente. O slot (ou slots, caso declare mais de um) são containers de conteúdo que podem ser injetados externamente ao componente (no componente pai que está utilizando o componente).

Figura 25 - Exemplo de slot.

```
<v-btn text>Voltar para loja</v-btn>
```

Na Figura 25 podemos ver que existe um conteúdo fornecido entre as tags do componente <v-btn>. “Voltar para a loja” (ou qualquer texto) é um exemplo de utilização de slots. De acordo com o contexto do componente pai, fornecemos um conteúdo que será posicionado e renderizado pelo componente filho.

Para declarar um slot no componente, utiliza-se a sintaxe <slot></slot>. Caso haja a necessidade de se declarar mais de um slot (vários conteúdos contextualizados, por exemplo, uma janela de diálogo, podemos ter um slot para o cabeçalho, corpo da janela e rodapé, fornecendo o conteúdo destes pelo componente pai, mas deixando a estrutura e posicionamento para o componente da janela de

diálogo). Você deve atribuir um *name* para o slot e utilizar a sintaxe (para injetar conteúdo em cada slot diferente):

```
<template v-slot:nome-do-slot>[aqui vem o conteúdo]</template>
```

O conteúdo de um slot não está limitado a texto simples, podendo receber qualquer código compatível com Vue (HTML, template, outros componentes etc.).

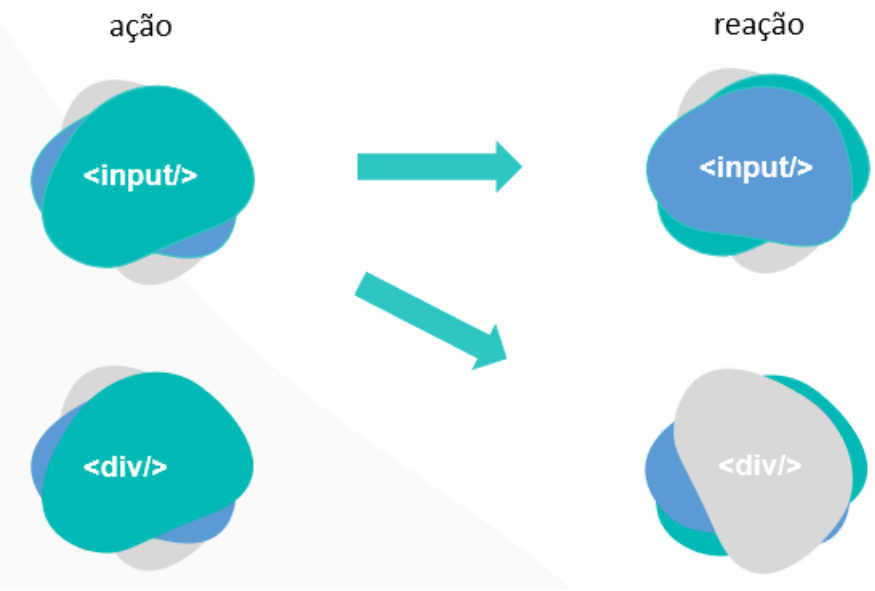
Capítulo 4. Layout dinâmico – Diretivas

Componentes são a base da sintaxe do Vue e formam sua estrutura, enquanto diretivas são as engrenagens que ditam a reatividade do Vue e sua capacidade de renderização automática do layout.

O que é reatividade?

Two-way data binding, ou somente reatividade, é a capacidade de atualização em tempo real de um ou mais elementos a partir da mudança de estado de outro.

Figura 26 - Reatividade resumida.



Uma ação (por exemplo, entrada do usuário), altera o estado do input (ou qualquer outro elemento – primitivo do HTML ou componente do Vue) e todos os demais elementos que estiverem mapeados àquele estado (na Figura 26, por exemplo, alteramos também o estado de uma associada ao conteúdo da <input>).

Quando falamos sobre estado, estamos falando de seu valor interno, ou qualquer atributo, propriedade, evento que esteja tentando alterar seu valor de alguma forma. A alteração do estado está relacionada ao evento “touched”. Exemplos

de alteração de estado: entrada do usuário, hover do mouse, clique de um botão, alteração do HTML interno por alteração de alguma propriedade, mudança de classe ou estilo etc.

Reatividade aprofundada

Vamos ver alguns exemplos simples de “reatividade” que poderiam ser alcançados utilizando Vanilla Javascript (javascript puro):

Figura 27 - Reatividade com Vanilla Javscript.

```
<div>
  <input id=“someTextInput" type="text">
  <div>
    Valor digitado foi: <span id=“inputTextDisplay"></span>
  </div>
</div>
<script type="text/javascript">
  var textInput = document.getElementById("someTextInput")
  textInput.addEventListener("input", function () {
    document.getElementById(“inputTextDisplay").innerHTML =  textInput.value;
  });
</script>
```

Note a quantidade de linhas que precisamos preencher, instalar (nós mesmos) o listener para o campo de input, atribuir o novo valor para o campo de texto. Parece simples para uma variável somente, mas imagina à medida que sua aplicação cresce e você precisa fazer isso para dezenas ou centenas de inputs, divs e demais elementos?

Já com o vue…

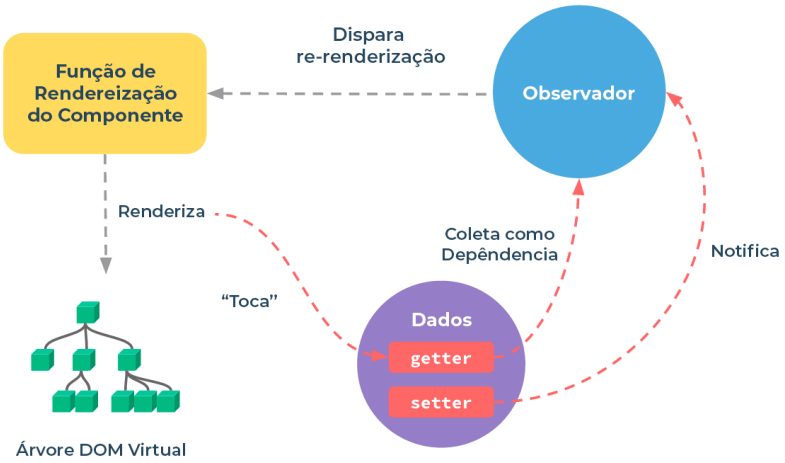
Figura 28 - Reatividade com Vue.

```
<div>
  <input type="text" v-model="textInput"/>
  Valor digitado foi: {{ textInput }}
</div>
```

Com somente uma variável em Vue, alcançamos o mesmo resultado, pois todo o código que escrevemos em Vue recebe observadores que observam e reagem a alteração de seus estados. No exemplo da Figura 28, quando o usuário alterar o

valor do input de texto associado a variável *textInput*, ela será automaticamente atualizada no campo de template ({{ }}).

Figura 29 - Reatividade no Vue.



A reatividade é nativa para elementos primitivos: *string, integers, booleans*, Arrays (somente push ou atualização completa do array) e modificação de objetos (propriedades de primeiro nível). Existem algumas limitações para a reatividade em Arrays e objetos:

- Remover um elemento em um Array;
- Modificar uma propriedade de segundo ou maior nível em um objeto (exemplo: obj.propPai.propFilha).

Nestes casos, mesmo havendo alteração do estado, o Vue não irá disparar a reatividade. Isso pode ser endereçado ativando manualmente a reatividade nestes elementos. Para isso, o Vue disponibiliza duas funções: *Vue.set()* e *Vue.delete()* (ou utilizando this quando declarando em um componente).

Diretivas

Vue nos oferece uma série de diretivas que podem ser injetadas tanto em elementos HTML, quanto componentes declarados em um <template>, que nos

permite alterar o comportamento daquele elemento, alterar seu estilo, reagir ou ativar eventos, renderizá-lo condicionalmente e diversas outras possibilidades. Vamos aqui explorar os principais.

- **v-model**

Atribuição a elementos de formulário (input, select, textarea, …). Cria um vínculo *two-way data binding* que atualiza o estado do campo automaticamente de acordo com o seu tipo (“text”, “checkbox”, “radio”, etc.) a partir da entrada de dados pelo usuário.

Figura 30 - Reatividade no Vue.

```
<div>
  <input type="text" v-model="nomeCachorro"/>
  Nome do cliente: {{ nomeCachorro }}
</div>
```

- **v-if / v-else-if / v-else**

Controle de visibilidade de elementos (renderizar ou não o elemento).

Figura 31 - Exemplo de utilização do v-if.

```
<div v-if="filaBanho.length > 0">
  Próximo cliente na fila {{ filaBanho[0] }}
</div>
<div v-else>
  Aguardando próximo cliente
</div>
```

Caso a condição atribuída ao *v-if* seja falsa, o elemento em questão não será renderizado (diferentemente de *v-show* que veremos a seguir).

- **v-show**

Também responsável por controlar a visibilidade de elementos, mas aqui, ao contrário do *v-if*, o elemento é renderizado e seu *display* é modificado de acordo com o valor atribuído a *v-show.*

- **v-for**

Loop em uma estrutura de dados (array ou objeto) e renderização de listas.

Figura 32 - Exemplo de renderização de lista utilizando v-for.

```
<table>
  <tr>
    <th>Nome</th>
    <th>Raça</th>
    <th>Idade</th>
    <th>Sintomas</th>
  </tr>
  <tr v-for="(cliente, index) in filaVeterinario" :key="index">
    <td>{{ cliente.nome }} </td>
    <td>{{ cliente.raca }} </td>
    <td>{{ cliente.idade }} </td>
    <td>{{ cliente.sintomas }} </td>
  </tr>
</table>
```

Assim como as demais diretivas, no v-for, a reatividade é mais visível e extremamente útil, caso a estrutura de dados seja atualizada, ela é automaticamente re-renderizada!

Sempre que utilizarmos o v-for, o Vue pede que seja fornecido um atributo *key* para que possa utilizar para renderizar os elementos da lista. V-for pode ser utilizado com qualquer elemento HTML.

- **v-bind**

Atribui aos atributos (class, style, required, disabled, href, etc.) de elementos do DOM uma variável que vai controlar seu valor.

Figura 33 - Sintaxes para v-bind.

```
<img v-bind:src="cliente.urlFoto">
<img :src="cliente.urlFoto"
<span :class="[classA, classB, …]">
<span :class="{ statusLiberado: cliente.banho.finalizado }">
```

Como v-bind é uma das diretivas mais utilizadas, Vue permite que o próprio v-bind seja omitido, escrevendo somente :class, :style, etc. (todas estas declarações serão associadas ao v-bind). Props de componentes quando associadas a uma variável e não a uma string, também utilizam a diretiva *v-bind*.

Figura 34 - Exemplo de declaração de prop utilizando v-bind.

```
<veterinario :clientes="clientesVeterinario" @realizado="finalizarConsulta"></veterinario>
```

Na Figura 34 "*:clientes*" é o mesmo que "*v-bind:clientes"*.

- **v-on**

Gatilho para eventos (click, change, input, blur, focus) de elementos HTML ou eventos customizados em componentes. Uma vez declarado um gatilho de evento via v-on, deve-se declarar uma função em *methods* para receber o resultado do evento. Na própria Figura 34, temos um exemplo de gatilhos de eventos. Aqui declaramos um evento customizado chamado "realizado" e associamos seu resultado (sempre que for chamado) para um método *finalizarConsulta.* Assim como v-bind, o v-on, por ser bastante utilizado pode ser utilizado com a abreviação "@" que substitui "v-on:nome-do-evento".

Figura 35 - Exemplo de sintaxe de v-on.

```
<button v-on:click="darBanho">Dar banho</button>
<button @click="darBanho">Dar banho</button>
```

Em um componente para se disparar um evento customizado, utilizamos a seguinte sintaxe:

Figura 36 - Exemplo de emissão de eventos.

```
methods: {
    finalizaConsulta(cliente) {
        this.$emit('realizado', cliente);
    }
}
```

No Vue, para diferenciar as funções da instância global para as do componente, precedemos as globais com “$”.

Podemos também customizar o disparo ou comportamento de nossos eventos, utilizando **modificadores de eventos** do Vue.

Figura 37 - Exemplos de modificadores de eventos.

```
<form v-on:submit.prevent="cadastrarCliente"></form>

<input type="text" v-on:keyup.13="cadastrarCliente">
```

Na Figura 37, temos dois exemplos de modificadores de eventos, o prevent (equivalente a event.prevent.default()) que impede que o formulário cause uma submissão completa da página e o keyup.13, que faz com que o evento só seja disparado quando o usuário digitar o caractere 13 (*enter)*. Existem outros diversos modificadores de eventos: stop, prevent, capture, self, once, passive etc.; e modificadores de campo de entrada: enter, tab, delete, ctrl, alt etc.; e muitos outros (Vue Official Documentation, 2020).

Capítulo 5. Roteamento

Single Page Applications é um conceito muito difundido nos dias de hoje que representa aplicações que dão a impressão de navegação, mas o conteúdo é todo atualizado sob-demanda, e renderizado em uma única página. Isso traz uma série de vantagens interessantes, como: diminui a carga do servidor (repassa boa parte do processamento para a máquina do cliente), habilidade de fazer requisições assíncronas (natureza do SPA), conteúdo renderizado sob-demanda (e não submissão completa da página) leva normalmente para páginas muito mais rápidas, facilidade de controle do estado da aplicação e também permite criar frontend e backend completamente isolados (até mesmo em linguagens diferentes). Mas também vem com algumas desvantagens, como: problemas de indexação em mecanismos de buscas (o robô não enxerga o conteúdo na hora que ele faz sua requisição), algumas questões extras de segurança devem ser endereçadas (podendo haver redundância devido ao desacoplamento com o servidor) e necessidade de implementação de lógica para endereçar o CORS (*cross-origin resource sharing* (Mozilla, 2020)).

vue-router

O vue-router (Vue Router Documentação Oficial, 2020) é a biblioteca oficial do Vue para roteamento e para criação de SPAs. Seu foco é na modularidade, onde suas views são construídas como componentes, bastante configurável e escalável. Implementa parcialmente o conceito MVVM (Cadu, 2020).

Possui suporte completo de roteamento:

- Query params, mapeamento de parâmetros;
- Nested routes e recursos;
- Match dinâmico de rotas;
- Aplicação de efeitos de transição em trocas de páginas.

Como vimos no em Instalação, a própria CLI do Vue inclui o vue-router como uma das opções de instalação. Também pode ser adicionado no com npm (Vue Router Documentação Oficial, 2020).

Uma vez instalado, vue-router nos fornece uma estrutura base para criação de *views* (páginas) e utilizar rotas na URL. Precisamos incluir o *plugin* do vue-router na instância global do Vue. Isso pode ser feito no arquivo main.js onde inicializamos a instância do Vue (Figura 39).

Figura 38 - Estrutura de pastas do vue-router.

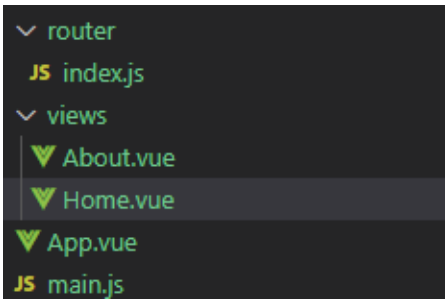


Figura 39 - Importando e incluindo o router na instância do Vue.

```
const vm = new Vue({
  router,
  el: '#app',
  render: h => h(App)
});
```

Figura 40 - Acessando rotas e renderizando views.

```
<template>
  <div id="app">
    <div id="nav">
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link>
    </div>
    <router-view/>
  </div>
</template>
```

Figura 41 - Arquivo exemplo de configuração do vue-router.

```
import Vue from 'vue';
import VueRouter from 'vue-router';
import Home from '../views/Home.vue';
import About from '../views/About.vue';

Vue.use(VueRouter)

const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/about',
    name: 'About',
    component: About
  }
];

const router = new VueRouter({
  mode: 'history',
  base: process.env.BASE_URL,
  routes
});

export default router
```

Configuração das rotas

Nosso arquivo base de configuração do vue-router (por padrão) se encontra em /src/router/index.js. Aqui devemos definir todo o comportamento do router assim como as rotas que nossa aplicação irá utilizar.

Figura 42 - Configuração do vue-router.

```
const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/about',
    name: 'About',
    component: About
  },
  {
    path: '/produtos',
    name: 'produtos',
    component: Products,
    children: [
      { path: '', name: 'lista-de-produtos', component: ProductList },
      { path: ':id', name: 'detalhes-do-produto', component: ProductDetails },
    ]
  },
];

const router = new VueRouter({
  mode: 'history',
  base: process.env.BASE_URL,
  routes
});

export default router
```

Este arquivo nos permite:

- Definição das rotas;
- Agrupamento de rotas (children);
- Uso de regex para match de rotas;
- Página customizada de erro 404 (*{ path: * }*);
- Customizar comportamento entre mudança de rotas (ou reagir a mudanças de rotas).

Navegando

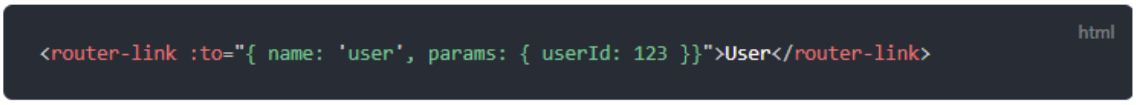
Configuradas as rotas que a aplicação irá utilizar, vue-router disponibiliza alguns componentes que podem ser usados para navegação.

- **<router-link>**

Renderizam elementos para navegar (anchors) e alterar a rota atual da aplicação. Podem ser usados de diversas maneiras:

Figura 43 - Utilizando o componente router-link.

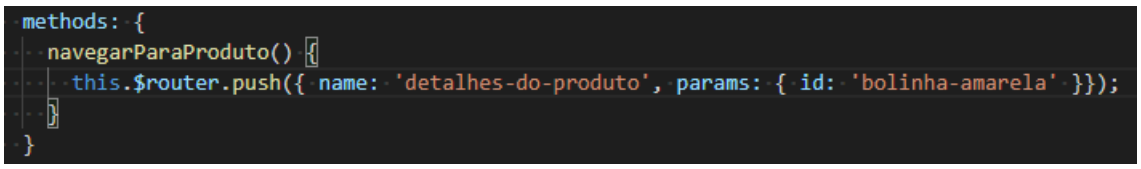
Figura 44 - Sintaxe alternativa para definir o destino da rota.



O parâmetro “to” indica (via endereço ou nome) a rota a qual desejamos navegar e ainda permite em seu slot customizar o conteúdo da área clicável.

Uma terceira forma de se navegar para outras views utilizando o vue-router, é programaticamente no <script>. Como injetamos o plugin na instância do Vue, ele fica disponível a partir do *this.$router* para utilizarmos sua interface. this.$router.*push* serve da mesma forma para navegação entre views (veja que possui sintaxe semelhante ao “to”).

Figura 45 - Alterando a rota no <script>.



Outras sintaxes suportadas:

- *push(‘rota’);*
- *push({ path: ‘rota’ });*
- *push({ name: ‘nome-da-rota’, params: { id: id, … });*

- *push({ path: 'rota', query: { chave: valor }}); (// usar query string ?chave=valor).*

Rotas privadas

Muitas vezes precisamos proteger o acesso a determinadas páginas (como, por exemplo, página somente para usuários logados). Para isso, o vue-router disponibiliza os guards. São funções utilizadas para controlar o acesso a rotas em sua aplicação. Podem ser usadas de forma global via funções: *beforeEach* e *afterEach* no objeto router (arquivo de configuração do vue-router /src/router/index), por rota específica, com *beforeEnter*, ou nos view components usando *beforeRouteEnter, beforeRouteUpdate, beforeRouteLeave* (Vue Router Documentação Oficial, 2020).

Figura 46 - Exemplo de guard usando beforeEach.

```
router.beforeEach((to, from, next) => {
  if (!app.$auth.loggedIn) {
    return redirect('/login')
  }

  next();
})
```

Além das configurações de rotas e guards, o vue-router possui uma série de outras funcionalidades para customização da navegação de nossa aplicação:

- *Redirect* (301 ou 307);
- *Alias:*
  - *A*pelidos para rotas – serão substituídos pelo apelido quando acessar determinada rota;
- Meta:
  - Adicionar propriedades a rota que pode ser acessado por guards ou pelo próprio componente;

- Transitions: Adicionam efeitos ou suavização na troca de rotas;
  - Pode ser única para toda troca de rota;
  - Diferente para cada rota;
  - Pode-se adicionar no *meta* para que o componente escolha a transição;
  - Buscar dados no servidor;
  - Permite exibir barras de loading;
  - Chamar uma API externar e coletar dados de acordo com a rota;
  - Aguardar a execução da API para completar a renderização da rota.

Capítulo 6. Estilo

Vue possui diversas bibliotecas que traduzem as principais recomendações de design e estilo no mercado. Uma das mais poderosas é o Vuetify que é baseado no material design da Google (Material Design, 2020).

Material design foi criado pela Google em 2014 para servir como guia de experiência do usuário para web e dispositivos móveis. Inspirado na física do “mundo real” – tenta reproduzir texturas, incidência de luz e sombras (de acordo com o posicionamento relativos dos elementos) e diversas outras características. Baseado em componentes (assim como o Vue para adaptação total ou parcial em projetos novos ou existentes), possui documentação abrangente de temas, medição, espaçamento, cores, tipografia, componentes, formas, navegação, animação e muito mais, além dos passos para sua adequação nos projetos.

Um conceito importante do Material design é sua unidade básica chamada dp (“dips” ou density independente pixels). É uma forma de manter as proporções da aplicação uniformes independentes da densidade de pixels do aparelho (que podem e variam bastante). Observe que no iOS a unidade que representa os dps são os pts (ou simplesmente points (iOS Device Compatibility Reference, 2017)).

dp = largura (ou altura) em pixels / largura (ou altura) em polegadas;

Assim como demais recomendações para design de aplicações web e mobile, o material design define seu grid de layout:

Figura 47 - Layout material design (desktop).

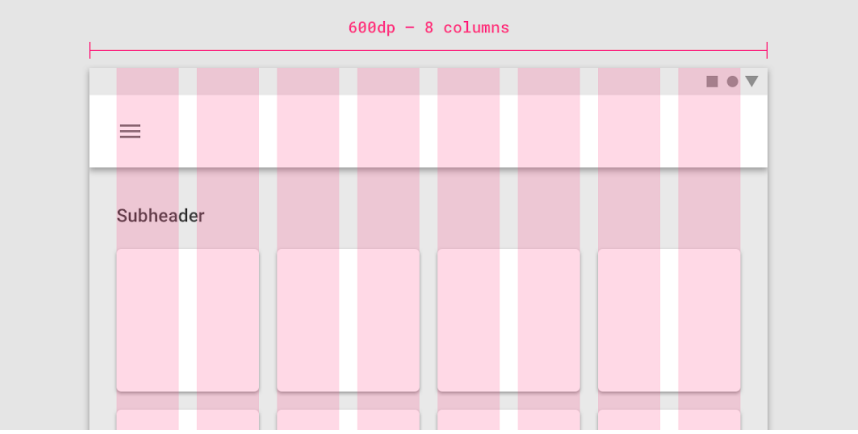


Figura 48 - Layout material design (tablets).

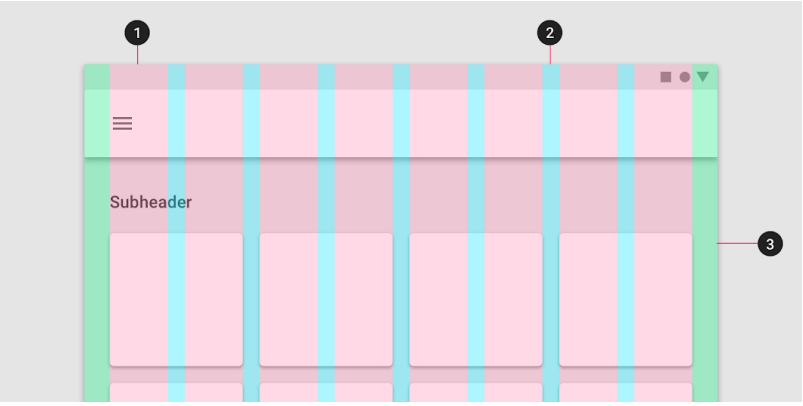
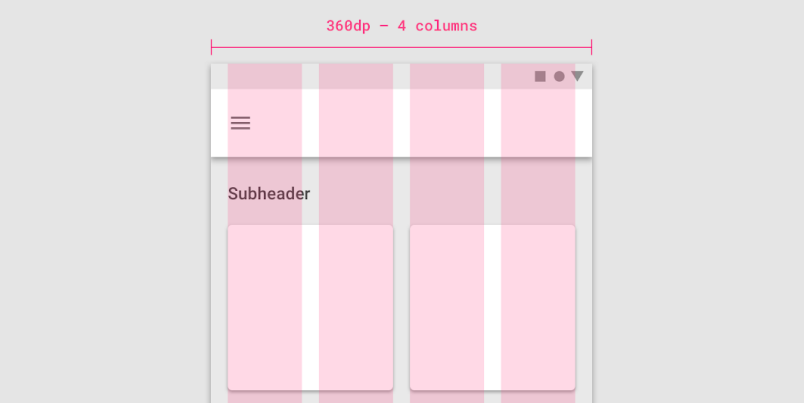


Figura 49 - Layout material design (celulares).



Vuetify

A biblioteca do Vuetify pode ser instalada via CLI ou npm. Note que usando a CLI ela irá fazer algumas alterações na estrutura base do seu projeto (como substituir o App.vue), então recomendamos fazer um backup do projeto ou criar uma nova *branch* antes de instalar a biblioteca.

```
> vue add vuetify
```

Figura 50 - Importação do plugin do vuetify na instância do Vue.

```
new Vue({
  vuetify,
  render: h => h(App)
}).$mount('#app')
```

Figura 51 - Arquivo de configuração do Vuetify.

```
import Vue from 'vue';
import Vuetify from 'vuetify/lib';

Vue.use(Vuetify);

export default new Vuetify({

});
```

Figura 52 - Componente exemplo gerado após a instalação do Vuetify.

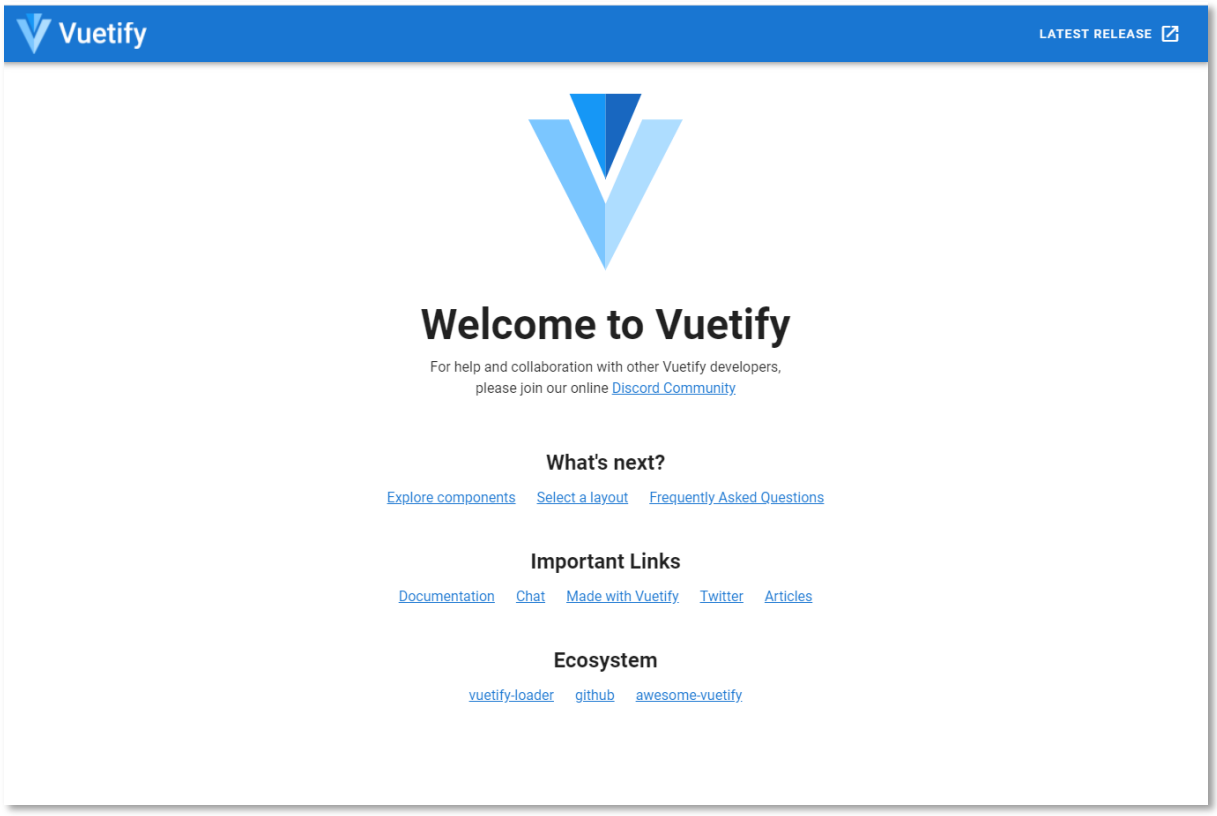
```
<v-app>
  <v-app-bar
    app
    color="primary"
    dark
  >
    <div class="d-flex align-center">
      <v-img
        alt="Vuetify Logo"
        class="shrink mr-2"
        contain
        src="https://cdn.vuetifyjs.com/images/logos/vuetify-logo-dark.png"
        transition="scale-transition"
        width="40"
      />

      <v-img
        alt="Vuetify Name"
        class="shrink mt-1 hidden-sm-and-down"
        contain
        min-width="100"
        src="https://cdn.vuetifyjs.com/images/logos/vuetify-name-dark.png"
        width="100"
      />
    </div>

    <v-spacer></v-spacer>

    <v-btn
      href="https://github.com/vuetifyjs/vuetify/releases/latest"
      target="_blank"
      text
```

Figura 53 - Página resultante utilizado os componentes do vuetify.



A partir disso, Vuetify disponibiliza dezenas de componentes que encapsulam todas as definições do Material. Como tal, são componentes do Vue e têm os mesmos comportamentos (reatividade, declaração de propriedades, eventos etc.). Iremos listar aqui os principais componentes, e recomendamos leitura aprofundada da API do Vuetify para encontrar os componentes que melhor se adequam as necessidades de seu projeto (Vuetify - Documentação Oficial, 2020).

- **v-app** (https://vuetifyjs.com/en/components/application/)

Componente utilizado como container da aplicação. Aplica dimensões, tamanhos para responsividade, margens e fontes.

- **v-container** (https://vuetifyjs.com/en/api/v-container/#v-container-api)

Define margens padrão para o conteúdo (adaptadas responsivamente).

- **v-alert** (https://vuetifyjs.com/en/components/alerts/)

Mensagem de alerta para o visitante. Implementa as cores e estilo padrão: success, error, warning, info, default, primary.

- **v-avatar** (https://vuetifyjs.com/en/components/avatars/)

Container arredondado para imagem geralmente utilizado para exibir fotos de usuário.

- **v-app-bar** (https://vuetifyjs.com/en/components/app-bars/)

Componente completo com definição para menus de topo e laterais. Implementa scroll, posicionamento, cores, estilo, posicionamento do conteúdo internamente.

- **v-btn** (https://vuetifyjs.com/en/components/buttons/)

Botões com estilo segundo material design, efeitos de clique, trigger de eventos e conteúdo.

- **v-calendar** (https://vuetifyjs.com/en/components/calendars/)

Componente completo que implementa um calendário, com troca de dados, cadastro de eventos, mudança de visualização (dia/semana/mês) e diversas outras funcionalidades.

- **v-card** (https://vuetifyjs.com/en/components/cards/)

Container de conteúdo, responsivo com definição de posicionamento de elementos, cores, sombreamento.

- **v-carousel** (https://vuetifyjs.com/en/components/carousels/)

Slides de conteúdo. Implementa e customiza o conteúdo, navegação e paginação.

- **v-dialog** (https://vuetifyjs.com/en/components/dialogs/)

  Componente completo que implementa caixas de diálogo (ou modais).

- **v-form, v-input, v-text-field, v-select** (https://vuetifyjs.com/en/components/forms/)

  Componentes para gestão de formulários, caixas de texto, dropdowns, auto-complete, seleção, estilo, labels, etc.

- **v-icon** (https://vuetifyjs.com/en/components/icons/)

  Componente para ícones.

- **v-img** (https://vuetifyjs.com/en/components/images/)

  Componente para imagens, encapsula todas as definições do material design para imagens.

- **v-table** (https://vuetifyjs.com/en/components/simple-tables/)

  Componente de tabela. Configura estilo de colunas, rodapés, hover, paginação, navegação, ordenação de itens da tabela/lista.

Referências

APPLE. *iOS Device Compatibility Reference*. 2017. Disponível em: <https://developer.apple.com/library/archive/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/Displays/Displays.html>. Acesso em: 26 out. 2020.

ARMIN & MELANIE. *10 Enterprises You Didn't Know Were Using Vue.js*. Made With Vue.js, 2019. Disponível em: <https://madewithvuejs.com/blog/enterprises-you-didnt-know-were-using-vue-js>. Acesso em: 26 out. 2020.

CADU. *Entendendo o Pattern Model View ViewModel MVVM*. DevMedia, 2010. Disponível em: <https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>. Acesso em: 26 out. 2020.

CYPRESS. Disponível em: <https://www.cypress.io/>. Acesso em: 26 out. 2020.

DAITYARI, Shaumik. *Angular vs React vs Vue:* Which Framework to choose in 2020. Codeinwp, 2020. Disponível em: <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/>. Acesso em: 26 out. 2020.

GITHUB. *awesome-vue*. Disponível em: <https://github.com/vuejs/awesome-vue> Acesso em: 26 out. 2020.

GITHUB. *Mustache*. 2020. Disponível em: <https://mustache.github.io/>. Acesso em: 26 out. 2020.

GITHUB. *Vue Official Documentation*. 2020. Disponível em: <https://github.com/vuejs/vue>. Acesso em: 26 out. 2020.

MADE WITH VUE. Disponível em: <https://madewithvuejs.com/>. Acesso em: 26 out. 2020.

Material Design. 2020. Disponível em: <https://material.io/design/>. Acesso em: 26 out. 2020.

MOZILLA. *CORS*. 2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Controle_Acesso_CORS>. Acesso em: 26 out. 2020.

NPM trends. *angular vs react vs vue*. 2020. Disponível em: <https://www.npmtrends.com/angular-vs-react-vs-vue>. Acesso em: 26 out. 2020.

NUXTJS. Disponível em: <https://nuxtjs.org/>. Acesso em: 26 out. 2020.

Repositório do Vue no github. Disponível em: <https://github.com/vuejs/>. Acesso em: 26 out. 2020.

ROUTER. *Vue Router Documentação Oficial*. 2020. Disponível em: <https://router.vuejs.org/>. Acesso em: 26 out. 2020.

SPEZZANO, L. *Google, Apple and Other Users of Vue.js*. Medium, 2019. Disponível em: <https://medium.com/notonlycss/google-apple-and-other-users-of-vue-js-e4505359e5d5>. Acesso em: 26 out. 2020.

StackOverflow. 2020. Disponível em: <https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages>. Acesso em: 26 out. 2020.

VUE. Documentação da versão 3.0 do Vue. Disponível em: <https://v3.vuejs.org/>. Acesso em: 26 out. 2020.

VUETIFY. *Documentação Oficial*. 2020. Disponível em: <https://vuetifyjs.com/en/>. Acesso em: 26 out. 2020.

VUEX. Disponível em: <https://vuex.vuejs.org/>. Acesso em: 26 out. 2020.