
Vue.JS - Básico

Ricardo Ruiz

Full Stack Developer

ricardo.ruiz@padtec.com

.br

(19) 2104-1706

Padtec

O que iremos ver nesse treinamento

- Pré-requisitos
- Ferramentas
- Introdução
- SPA
- JS vs Vue.js
- Data
- Methods
- Computed
- Watch
- Lifecycle Hooks
- Diretivas
 - v-bind
 - v-on
 - v-if
 - v-for
 - v-model
- Components
 - Escopo
 - Props
 - Events
 - Slots

Pré-requisitos

- HTML
- CSS
- Javascript

Ferramentas

- Visual Studio Code
 - <https://code.visualstudio.com/>
- Node.js
 - <https://nodejs.org/en/>
- Vue.js Devtools
 - <https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjimejiggliccpnnnanhbledaibpd>
- Vue CLI
 - <https://cli.vuejs.org/>

Introdução

O que é Vue.js?

Vue é um framework progressivo para a construção de interfaces de usuário. Ao contrário de outros frameworks monolíticos, Vue foi projetado desde sua concepção para ser adotável incrementalmente. A biblioteca principal é focada exclusivamente na camada visual (view layer), sendo fácil adotar e integrar com outras bibliotecas ou projetos existentes. Por outro lado, Vue também é perfeitamente capaz de dar poder a sofisticadas **Single-Page Applications** quando usado em conjunto com ferramentas modernas e bibliotecas de apoio.

Características

- **Reatividade / Sincronia** => Manter a ui sincronizada com o estado. Estado (state) é um dado em uma variável.
- **Conveniência** => Torna mais simples a manipulação do DOM.
- **Componentes** => Facilita a divisão da interface em componentes, facilitando a manutenção.

SPA - Single-Page Application

SPA basicamente significa codificar menos no server-side e mais no client-side, ou seja, a aplicação estará contida toda ou quase toda no cliente (dentro do navegador Web). É praticamente uma aplicação Desktop rodando sob o navegador.

Vantagens:

1. Balanceamento da responsabilidade da execução entre cliente e servidor (agora não é só mais responsabilidade do servidor);
2. Menos código do servidor, e mais responsabilidade no cliente;
3. Melhorar a experiência ao usuário (UX) criando interface com usabilidade moderna e de fácil entendimento do usuário;
4. Menor consumo de banda, pois as cargas de dados são feitas por demanda e por AJAX.

Sintaxe básica (JS)

```
<div>
  <button>Incrementar</button>
  <span></span>
</div>
```

```
const button = document.querySelector("button");
const span = document.querySelector("span");
let total = 0;

function incrementar() {
  span.innerText = total++;
}
button.addEventListener("click", incrementar);
```

Sintaxe básica (Vue.JS)

```
<div id="app">
  <button @click="total++">Incrementar</button>
  <span>{{total}}</span>
</div>
```

```
const vm = new Vue({
  el: "#app",
  data: {
    total: 0
  }
});
```


Reatividade sem Vue

```
<div>
  <p>Camisas - R$ <span class="preco">49</span></p>
  <button class="adicionar">Adicionar</button>
  <button class="remover">Remover</button>
  <span class="total">0</span>
  <p>Total: <span class="precoTotal"></span></p>
</div>
```

```
const dados = {
  preco: 49,
  total: 0
};

const adicionar = document.querySelector(".adicionar");
const remover = document.querySelector(".remover");
const preco = document.querySelector(".preco");
const total = document.querySelector(".total");
const precoTotal = document.querySelector(".precoTotal");

preco.innerText = dados.preco;
total.innerText = dados.total;
precoTotal.innerText = dados.total * dados.preco;

function atualizarUI() {
  total.innerText = dados.total;
  precoTotal.innerText = dados.total * dados.preco;
}
```

Reatividade sem Vue

```
function incrementar() {  
  dados.total++;  
  atualizarUI();  
}  
  
function diminuir() {  
  dados.total--;  
  atualizarUI();  
}  
  
adicionar.addEventListener("click", incrementar);  
remover.addEventListener("click", diminuir);
```

Reatividade com Vue

```
<div id="app">
  <p>Bermudas - R$ <span>{{preco}}</span></p>
  <button @click="total++">Adicionar</button>
  <button @click="total--">Remover</button>
  <span>{{total}}</span>
  <p>Total: <span>{{total * preco}}</span></p>
</div>

<script>
export default {
  data: {
    preco: 49,
    total: 0,
  }
}
</script>
```

Estrutura básica

```
<template>
  <div id="app">
    <p>Hello world</p>
  </div>
</template>
```

```
<script>
export default {
  name: 'app',
  data() {
    return {
    },
  },
  methods: {
  },
  computed: {
  },
  watch: {
  }
}
</script>
```

```
<style>
</style>
```

Criação do projeto

1 - Primeiro vamos instalar o Vue CLI

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

2 - Após a instalação vamos verificar a versão

```
vue --version
```

3 - Agora com o CLI instalado vamos criar o primeiro projeto

```
vue create treinavue-basico
```



```
Vue CLI v3.11.0
? Please pick a preset: (Use arrow keys)
> default (babel, eslint)
  Manually select features
```

utilize a opção default

Criação do projeto

```
Vue CLI v3.11.0
🌟 Creating project in /media/rick/Arquivos/Padtec/treinavue-basico.
📁 Initializing git repository...
⚙️ Installing CLI plugins. This might take a while...

> yorkie@2.0.0 install /media/rick/Arquivos/Padtec/treinavue-basico/node_modules/yorkie
> node bin/install.js

setting up Git hooks
done

> core-js@2.6.9 postinstall /media/rick/Arquivos/Padtec/treinavue-basico/node_modules/core-js
> node scripts/postinstall || echo "ignore"

added 1168 packages from 920 contributors and audited 24068 packages in 18.407s
found 0 vulnerabilities

🚀 Invoking generators...
📦 Installing additional dependencies...

added 35 packages from 28 contributors, updated 2 packages, moved 9 packages and audited 24355 packages in 9.278s
found 0 vulnerabilities

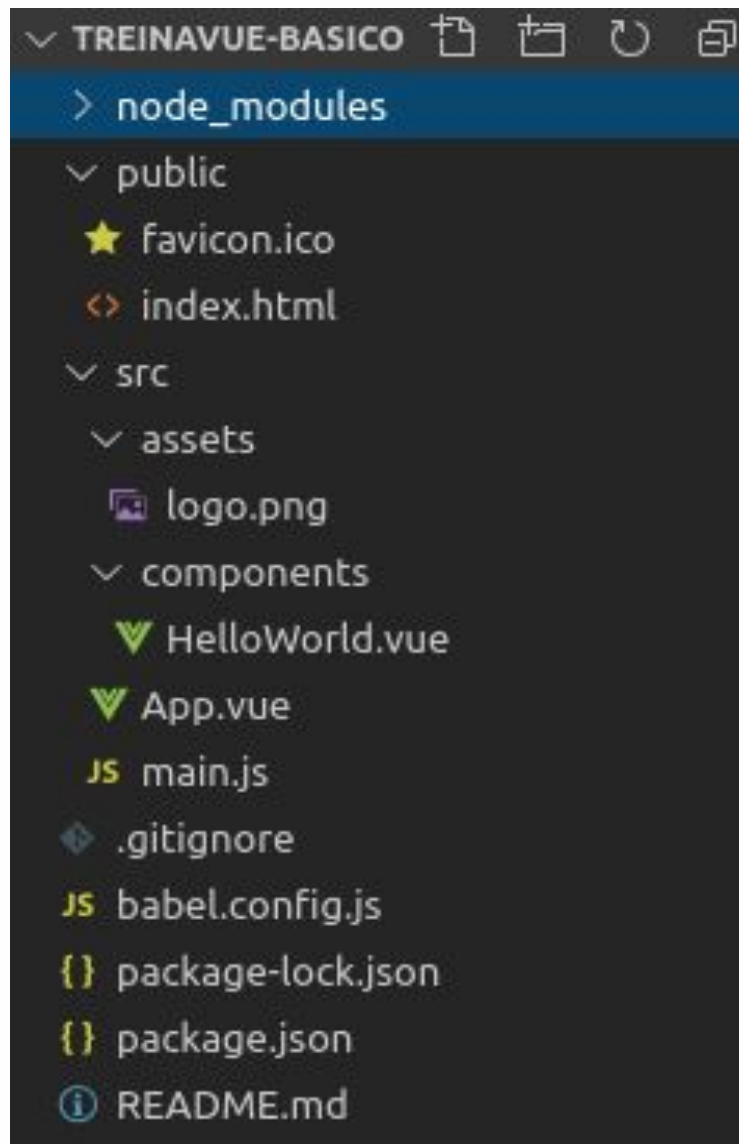
🔗 Running completion hooks...

📄 Generating README.md...

🎉 Successfully created project treinavue-basico.
👉 Get started with the following commands:

$ cd treinavue-basico
$ npm run serve
```

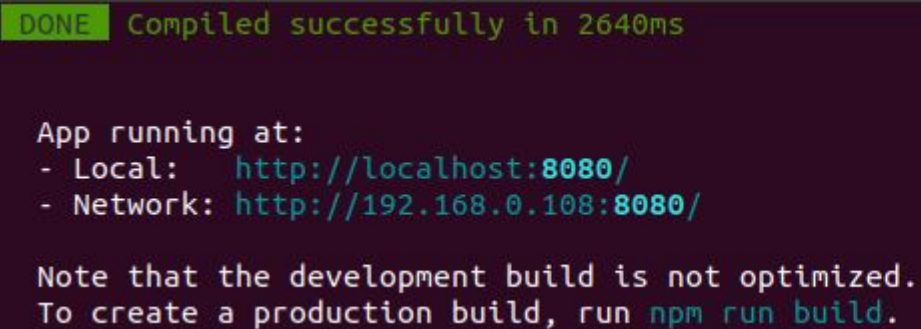
Estrutura do projeto



Executando a aplicação criada

Para executar a nossa aplicação vamos executar o seguinte comando:

npm run serve

A terminal window with a dark purple background. At the top, a green box contains the text 'DONE' in white, followed by 'Compiled successfully in 2640ms' in green. Below this, the text 'App running at:' is shown in white. Underneath, two lines of text are listed: '- Local: http://localhost:8080/' and '- Network: http://192.168.0.108:8080/'. At the bottom, a note in white text states: 'Note that the development build is not optimized. To create a production build, run npm run build.'

```
DONE Compiled successfully in 2640ms

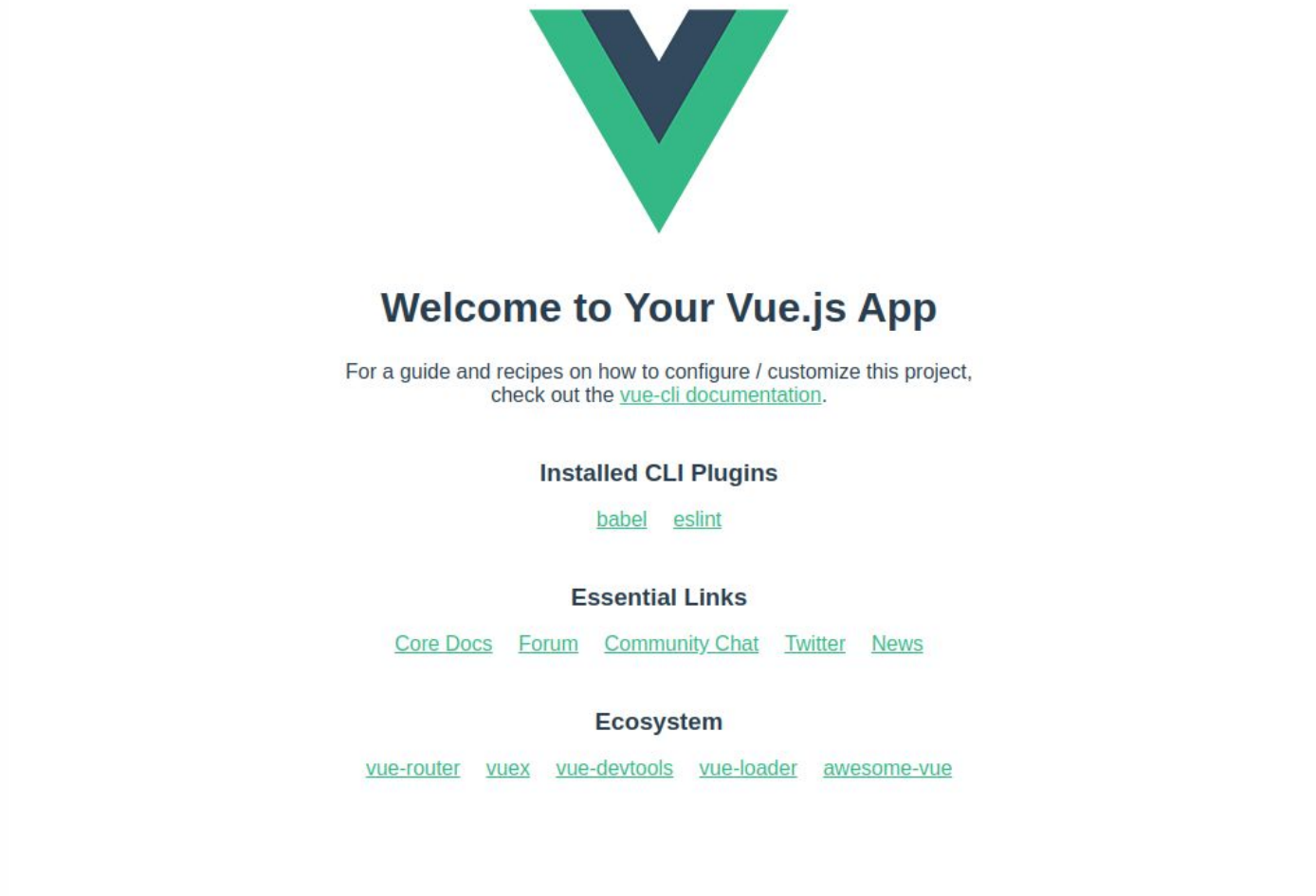
App running at:
- Local: http://localhost:8080/
- Network: http://192.168.0.108:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

Após o fim da compilação veremos a saída acima e a nossa aplicação estará disponível na porta informada.

Executando a aplicação criada

Ao acessar o endereço (<http://localhost:8080>) teremos a seguinte representação:



Exercício 1

1 - Instalar as ferramentas necessárias

Node.js

Vue CLI

Visual Studio Code

2 - Criar o projeto inicial

3 - Executar o projeto

Ao final desse exercício você estará com o seu ambiente de desenvolvimento preparado.

Resultado:

<https://github.com/ricardoarui/treinavue-basico>

Data

A propriedade **data** é responsável por dar a reatividade aos estados.

```
<template>
  <div id="app">
    
    <!-- Data -->
    <h1>{{titulo}}</h1>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      titulo: 'Esse é o título da página'
    }
  }
}
</script>
```

Data

Sempre defina os estados no data, mesmo que você ainda não possua o valor. É o registro no data que garante a reatividade.

```
<template>
  <div id="app">
    
    <!-- Data -->
    <h1>{{titulo}}</h1>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      titulo: ''
    }
  },
  created() {
    setTimeout(() => {
      this.titulo = 'Título da página'
    }, 1500);
  }
}
</script>
```

Data

Qualquer tipo de dado pode ser utilizado em um data.

```
<template>
  <div id="app">
    
    <p>{{frutas[0]}}</p>
    <p>{{frutas[1]}}</p>
    <p>{{objeto.item}}</p>
    <p>{{comprou}} - {{total}}</p>
  </div>
</template>
<script>
export default {
  name: 'app',
  data() {
    return {
      comprou: true,
      total: 49,
      vitalicio: null,
      objeto: {
        item: "Item 1"
      },
      frutas: ["Banana", "Uva"]
    }
  }
}
</script>
```

Exercício 2

Altere o arquivo App.vue de forma que tenha uma propriedade “lado” (que será o lado de um quadrado) em seu “data” e que no template do mesmo sejam exibidos o perímetro e a área de desse quadrado.

Altere o valor do “lado” e veja como o template reage a isso.

Lado do quadrado: 2

Perímetro do quadrado: 8

Área do quadrado: 4

Resultado:

<https://github.com/ricardoarui/treinavue-basico/tree/01-data>

Methods

Métodos são as ações definidas na instância do componente Vue.

```
<template>
  <div id="app">
    <p>{{preco}}</p>
    <button @click="adicionarCupom">Adicionar Cupom</button>
    <p>{{alerta}}</p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      preco: 100,
      alerta: ""
    }
  },
  methods: {
    adicionarCupom() {
      this.preco *= 0.9;
      this.alertaCupom();
    },
    alertaCupom() {
      this.alerta = "Cupom Adicionado";
    }
  }
}
```

Exercício 3

Altere o arquivo App.vue de forma que tenha uma propriedade “quantidade” (de um produto) iniciando com um valor positivo e uma mensagem iniciando com valor em branco.

Crie um botão que ao ser clicado decemente a quantidade e quando o seu valor for menor ou igual a zero, mostre a mensagem que o produto está esgotado.



Resultado:

<https://github.com/ricardoaruiz/treinavue-basico/tree/02-methods>

Diretivas

São atributos html especiais do Vue que permitem a interação entre o código JavaScript e o HTML.

Diretiva v-bind

O v-bind é uma diretiva que permite a utilização de expressões dentro de atributos html.

```
<template>
  <div id="app">
    <a v-bind:href="link">Link Google</a>
    <p v-bind:class="cor">Parágrafo</p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      link: "https://www.google.com.br"
      cor: "azul"
    }
  }
}
</script>
```

O valor após os : é considerado o argumento da diretiva.

Diretiva v-bind - Atalho

O uso de dois pontos **:href** na frente de um atributo html, é a mesma coisa que utilizarmos o **v-bind:href**

```
<template>
  <div id="app">
    <a :href="link">Link Google</a>
    <p :class="cor">Parágrafo</p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      link: "https://www.google.com.br",
      cor: "azul"
    }
  }
}
</script>
```

Diretiva v-bind

É possível utilizar os atributos normais junto com as diretivas. Se o atributo permitir apenas um valor, o último será o utilizado.

```
<template>
  <div id="app">
    <a :href="link" href="https://gmail.com.br">Link Google</a>
    <p :class="cor" class="p1" >Parágrafo</p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      link: "https://www.google.com.br"
      cor: "azul"
    }
  }
}
</script>
```

Diretiva v-bind - Expressões

Expressões podem ser utilizados dentro dos valores.

```
<template>
  <div id="app">
    <p :class="comprou ? 'verde' : 'vermelho'">Comprou</p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      comprou: false
    }
  }
}
</script>
```

Exercício 4

Partindo do resultado do exercício 3, altere o arquivo App.vue de forma que tenha uma que quando a quantidade for positiva, o nome do produto apareça em verde e caso contrário apareça em vermelho.



Resultado:

<https://github.com/ricardoarui/treinavue-basico/tree/03-vbind>

Documentação diretivas:

<https://br.vuejs.org/v2/api/index.html#Diretivas>

Diretiva v-on

O **v-on** é uma diretiva que permite observarmos eventos nas tags.

```
<template>
  <div id="app">
    <button v-on:click="handleClick">Clique</a>
  </div>
</template>

<script>
export default {
  name: 'app',
  methods: {
    handleClick(event) {
      console.log(event);
    }
  }
}
</script>
```

Diretiva v-on - Atalho

O uso do arroba **@click** na frente do evento, é a mesma coisa que utilizarmos o **v-on:click**.

```
<template>
  <div id="app">
    <button @click="handleClick">Clique</a>
  </div>
</template>

<script>
export default {
  name: 'app',
  methods: {
    handleClick(event) {
      console.log(event);
    }
  }
}
</script>
```


Diretiva v-on - Expressões

Não precisa ser necessariamente um método, podemos usar expressões dentro dos eventos.

```
<template>
  <div id="app">
    <button @click="ativo = !ativo">Ativar</button>
    <p :class="ativo ? 'verde' : 'vermelho'">Texto de teste.</p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data: {
    ativo: true
  }
}
</script>
```

Diretiva v-on - Modificadores

Algumas diretivas possuem modificadores. Estes são utilizados através do “.”. No caso de eventos, `event.preventDefault()` pode ser adicionado com o `@click.prevent`.

```
<template>
  <div id="app">
    <a @click.prevent="scrollSuave" href="#interno">Prevent</a>
    <a @click.once="scrollSuave" href="#interno">Once</a>
    <a @click.prevent.once="scrollSuave" href="#interno">Once</a>
  </div>
</template>

<script>
export default {
  name: 'app',
  methods: {
    scrollSuave() {
      console.log("Scroll Suave");
    }
  }
}
</script>
```

Para mais informações sobre modificadores: <https://vuejs.org/v2/api/#v-on>

Exercício 5

Altere o template do App.vue adicionando o seguinte conteúdo:

```
<ul>
  <li><a href="https://api.iextrading.com/1.0/stock/aapl/quote">Apple</a></li>
  <li><a href="https://api.iextrading.com/1.0/stock/googl/quote">Google</a></li>
</ul>
```

Adicione um evento de clique a cada <a> do html prevenindo o comportamento padrão. Chame um método / expressão diferente em cada link. Fique a vontade para utilizar o v-on ou seu atalho “@”

Experimente remover a opção que previne o comportamento padrão para ver o que acontece.

Resultado:

<https://github.com/ricardoarui/treinavue-basico/tree/04-von>

Documentação diretivas:

<https://br.vuejs.org/v2/api/index.html#Diretivas>

Diretiva v-if

O **v-if** é uma diretiva que permite utilizarmos condicionais para mostrar ou não um elemento. Podemos utilizar também o **v-else** que deve vir logo após o if para funcionar.

```
<template>
  <div id="app">
    <p v-if="logado">Você está logado.</p>
    <p v-else>Você não está logado.</p>
  </div>
</template>

<script>

export default {
  name: 'app',
  data() {
    return {
      logado: false
    }
  }
}

</script>
```

Diretiva v-if - Grupos

Podemos agrupar um conteúdo com a tag **<template>**

```
<template>
  <div id="app">
    <template v-if="logado">
      <p>Você está logado.</p>
      <button @click="logado = false">Deslogar</button>
    </template>
    <template v-else>
      <p>Você não está logado.</p>
      <button @click="logado = true">Logar</button>
    </template>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      logado: false
    }
  }
}
</script>
```

Diretiva v-if (v-if vs v-show)

O **v-if** remove o elemento, **v-show** apenas adiciona o **display: none;**. **v-show** é preferido se você for mudar constantemente o estado, por ser mais rápido.

```
<template>
  <div id="app">
    <p v-show="logado">Show logado</p>
    <p v-if="logado">If logado</p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      logado: false
    }
  }
}
</script>
```

Exercício 6

Partindo do resultado do exercício 4, altere o arquivo App.vue de forma que a mensagem de produto esgotado seja mostrada somente quando a quantidade for menor ou igual a zero, porém, utilizando o v-if / v-show.



Resultado:

<https://github.com/ricardoarui/treinavue-basico/tree/05-vif>

Documentação diretivas:

<https://br.vuejs.org/v2/api/index.html#Diretivas>

Diretiva v-for

Itera sobre uma lista de itens (array, objetos).

```
<template>
  <div id="app">
    <ul>
      <li v-for="(curso, index) in cursos">{{curso}}</li>
    </ul>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      cursos: ["HTML", "CSS", "JavaScript", "PHP", "WordPress"]
    }
  }
}
</script>
```


Diretiva v-for (objetos)

Aqui um exemplo utilizando objeto

```
<template>
  <div id="app">
    <div v-for="estado in estados">
      <p>{{estado.nome}}</p>
      <p>{{estado.populacao}}</p>
    </div>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
    return {
      estados: {
        sp: { populacao: "45 milhões", nome: "São Paulo" },
        mg: { populacao: "21 milhões", nome: "Minas Gerais" },
        rj: { populacao: "17 milhões", nome: "Rio de Janeiro" }
      }
    }
  }
}
</script>
```

Diretiva v-for (parâmetros)

**Para arrays você pode utilizar o valor e o index de cada item.
Para objetos é possível utilizar, valor, chave e index.**

```
<template>
  <div id="app">
    <div v-for="(value, index) in lista">
      <p>{{index}}: {{value}}</p>
    </div>
    <div v-for="(value, key, index) in melancia">
      <p>{{index}} {{key}}: {{value}}</p>
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      lista: ["Item 1", "Item 2", "Item 3"],
      melancia: { cor: "Verde", peso: "10kg", preco: "R$ 15" }
    }
  }
}
</script>
```

Diretiva v-for (propriedade :key)

Cria uma identificação única para o item.

Sem o key, bugs podem ocorrer principalmente em componentes mais complexos.

```
<template>
  <div id="app">
    <div v-for="fruta in frutas" :key="fruta.id">
      <p>{{fruta.nome}}</p>
      <p>{{fruta.cor}}</p>
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      frutas: [
        { id: "banana-1", nome: "Banana", cor: "Amarela" },
        { id: "melancia-1", nome: "Melancia", cor: "Verde" }
      ]
    }
  }
}
</script>
```

Diretiva v-for (numero in)

É possível fazer um laço definido por um número de repetições conhecido.

```
<template>
  <div id="app">
    <span v-for="numero in 5">{{numero}}</span>
  </div>
</template>
```

Diretiva v-for (reatividade)

Modificar diretamente o valor de uma array, não irá acionar a reatividade do Vue.

```
<template>
  <div id="app">
    <ul>
      <li v-for="item in lista">{{item}}</li>
    </ul>
    <button @click="removerItem">Remover</button>
    <button @click="mudarItem">Mudar</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      lista: ["Item 1", "Item 2", "Item 3"]
    }
  },
  methods: {
    removerItem() {
      this.lista.pop(); // Aciona reatividade
    },
    mudarItem() {
      this.lista[0] = "Banana"; // Não aciona reatividade
    }
  }
}
</script>
```

Exercício 7

Altere o arquivo App.vue de forma que exiba uma lista do conteúdo apresentadas nesse curso:

```
data
methods
diretivas
computed
watch
```

Resultado:

<https://github.com/ricardoarui/treinavue-basico/tree/06-vfor>

Documentação diretivas:

<https://br.vuejs.org/v2/api/index.html#Diretivas>

Computed

Quando precisamos modificar um valor para ser exibido por exemplo, podemos utilizar uma propriedade dentro de computed.

```
<template>
  <div id="app">
    <p>{{total}}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      preco: 59,
      desconto: 10
    }
  },
  computed: {
    total() {
      return "R$ " + (this.preco - this.desconto);
    }
  }
}
</script>
```

É possível acionar uma função toda vez que um dado reativo é modificado. Para isso usamos uma propriedade dentro de watch..

```
<template>
  <div id="app">
    <p>{{contador}}</p>
    <button @click="contador++">Adicionar</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      contador: 0
    }
  },
  watch: {
    contador() {
      console.log(this.contador);
    }
  }
}
</script>
```


Watch assíncrono

O watch se diferencia do computed principalmente pela sua capacidade de receber eventos assíncronos.

```
<template>
  <div id="app">
    <input type="text" v-model="cep" placeholder="Cep" maxlength="8" />
    <ul>
      <li v-for="(valor, chave) in endereco">{{chave}}: {{valor}}</li>
    </ul>
  </div>
</template>

<script>
export default {
  data() {
    return { cep: "", endereco: {} }
  },
  watch: {
    cep(valor) {
      if (valor.length === 8) {
        fetch(`https://viacep.com.br/ws/${valor}/json/`).then(r => r.json()).then(r => { this.endereco = r; });
      }
    }
  }
}
</script>
```

Exercício 8

Altere o arquivo App.vue

Ter o nome e sobrenome do usuário em data e criar uma propriedade computed que retorne o nome completo mostrando na tela

```
Nome: Alfredo  
Sobre nome: Barbosa  
Nome completo: Alfredo Barbosa
```

Resultado:

<https://github.com/ricardoaruiz/treinavue-basico/tree/07-computed-watch>

**É utilizado para tornar reativo o conteúdo de formulários.
Chamado de two-way data binding.**

```
<template>
  <div id="app">
    <div>
      <input v-model="nome" />
      <p>{{nome}}</p>
    </div>
    <div>
      <textarea v-model="mensagem"></textarea>
      <p>{{mensagem}}</p>
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      nome: "",
      mensagem: ""
    }
  }
}
</script>
```

two-way VS one-way

Por padrão todo o conteúdo do Vue.js é one-way, isso significa que mudança no JavaScript criam mudanças no DOM. Já no two-way, tanto mudanças no dom como no JavaScript, mudam o conteúdo.

```
<template>
  <div id="app">
    <div>
      Two-way: <input v-model="texto" />
    </div>
    <div>
      One-way: <input :value="texto" />
    </div>
    <p>{{texto}}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      texto: ""
    }
  }
}
</script>
```

Exercício 9

Altere o arquivo App.vue

**Crie um formulário com os campos nome, email e telefone, um botão enviar e um limpar.
Ao pressionar o botão enviar mostrar abaixo mensagem com nome, email e telefone.
Ao pressionar o botão limpar limpar todos os campos e a mensagem exibida**

Nome

E-mail

Telefone

Enviar

Limpar

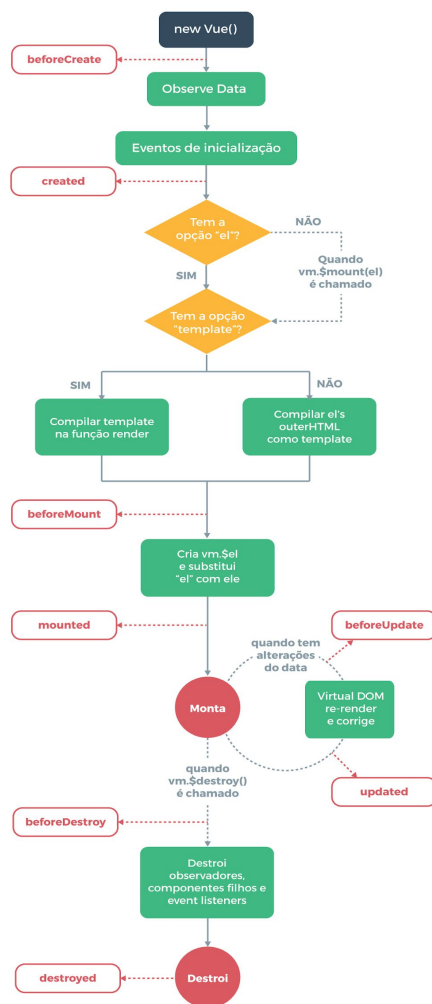
Alfredo Barbosa - abarbosa@email.com.br - (19) 99123-4567

Resultado:

<https://github.com/ricardoarui/treinavue-basico/tree/08-vmodel>

Lifecycle Hooks

Métodos que são ativados durante o ciclo de vida de um componente vue.js. Imagem retirada de: <https://br.vuejs.org/v2/guide/instance.html>



Created

O **beforeCreate** é o primeiro método ativado, ele é ativado antes mesmo das propriedades de data se tornarem reativas. Já no **created**, é possível ter acesso aos dados reativos. Created é o Hook ideal para iniciarmos requisições fetch.

```
<script>
export default {
  data() {
    return { mensagem: "Uma mensagem.", dados: {} }
  },
  methods: {
    chamarApi() {
      fetch("https://api.github.com/users/ricardoarui")
        .then(r => r.json())
        .then(r => {
          this.dados = r;
        });
    }
  },
  beforeCreate() {
    console.log(this.mensagem); // undefined
  },
  created() {
    console.log(this.mensagem); // "Uma mensagem."
    this.chamarApi();
  }
}
</script>
```

Mounted

O **beforeMount** acontece após o created. Em seguida o hook **mounted** acontece, durante essa fase o virtual dom é criado e podemos ter acesso ao **this.\$el**. Ideal para quando queremos modificar o DOM ou adicionar eventos globais (scroll, keyup e outros).

```
<template>
  <div id="app">
    {{mensagem}}
  </div>
</template>

<script>
export default {
  data() {
    return {
      mensagem: "Uma mensagem."
    }
  },
  beforeMount() {
    console.log(this.$el);
    // Template ainda não renderizado
  },
  mounted() {
    console.log(this.$el);
    // Template renderizado
  }
}
</script>
```


Virtual DOM (Document Object Model)

O Virtual DOM é um objeto javascript que simula o dom atual.

Mudanças são primeiramente feitas nesse objeto e em seguida uma verificação é feita no DOM, para sincronizar o mesmo.

Updated

O **beforeUpdate** acontece sempre que houver uma mudança em um dado reativo. Em seguida o hook **updated** acontece, este após o dom ser modificado.

```
<template>
  <div id="app">
    <button @click="contador++">{{contador}}</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      contador: 0
    }
  },
  beforeUpdate() {
    console.log(this.contador);
  },
  updated() {
    console.log(this.contador);
  }
}
</script>
```

Destroyed

O **beforeDestroy** acontece antes do componente ser destruído. Em seguida o hook **destroyed** acontece, este após o componente ser destruído. É muito utilizado quando dividimos a interface em componentes.

```
<template>
  <div id="app">
    <button @click="contador++">{{contador}}</button>
    <button @click="destruir">Destruir</button>
  </div>
</template>
```

```
<script>
export default {
  data() {
    return { contador: 0 }
  },
  methods: {
    destruir() { this.$destroy(); }
  },
  beforeDestroy() {
    console.log("Vai destruir");
  },
  destroyed() {
    console.log("Destruiu");
  }
}
</script>
```

Exercício 10

Altere o arquivo App.vue

**Utilize a api do github para mostrar todos os seus dados na tela.
Faça o fetch das informações utilizando um dos hooks do vue:**

`https://api.github.com/users/seu_usuario` (utilize o seu usuário)

O layout fica a seu gosto

Resultado:

<https://github.com/ricardoarui/treinavue-basico/tree/09-lifecycle-hooks>

Components

O Vue JS é baseado em componentes, então tudo que é feito no final das contas é um componente.

Vamos criar um novo arquivo na pasta `src/components` com o nome de `BotaoContador.vue` que terá o seguinte código:

```
<template>
  <button @click="incrementar">{{contador}}</button>
</template>

<script>
export default {
  data() {
    return {
      contador: 0
    }
  },
  methods: {
    incrementar() {
      this.contador++;
    }
  }
}
</script>
```

Components

Agora no App.vue vamos utilizar o nosso componente. Perceba que importamos o arquivo do componente criado, declaramos ele como um componente em App.vue e simplesmente o referenciamos no template.

```
<template>
  <div id="app">
    <BotaoContador />
  </div>
</template>

<script>
import BotaoContador from '@components/BotaoContador.vue'

export default {
  components: {
    BotaoContador
  }
}
</script>
```

Desta forma isolamos toda a parte de exibição e comportamento do botão em um componente ganhando assim o poder da reusabilidade sem contar a organização que isso traz para nosso código.

Components (global vs local)

Local

No exemplo anterior além de importar o `BotaoContador.vue` em `App.vue`, também o declaramos na propriedade `components` da instância de `App.vue`. Esse tipo de declaração é o que se chama de local, ou seja, só `App.vue` está apto a utilizar o botão.

```
<template>
  <div id="app">
    <BotaoContador />
  </div>
</template>

<script>
import BotaoContador from '@components/BotaoContador.vue'

export default {
  components: {
    BotaoContador
  }
}
</script>
```

Components (global vs local)

Global

Para declarar um componente como global devemos utilizar o método component da instância de Vue da seguinte forma:

No arquivo main.js:

```
import Vue from 'vue'
import App from './App.vue'
import BotaoContador from '@components/BotaoContador.vue';

Vue.config.productionTip = false

Vue.component("BotaoContador", BotaoContador);

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

Repare que importamos o BotaoContador.vue em main.js, utilizando o método component da instância de Vue declarando-o globalmente. Desta forma qualquer componente de nossa aplicação poderá utilizá-lo sem ter que importar e declarar.

Components (global vs local)

Veja como fica em App.vue agora com o componente registrado globalmente, basta utilizar o mesmo no template que já estará funcionando.

```
<template>
  <div id="app">
    <BotaoContador />
  </div>
</template>

<script>
export default {

}
</script>
```

Exercício 11

Crie 2 componentes para utilizar em App.vue.

1 - Mostre o tempo do dia usando a API:

<https://api.hgbrasil.com/weather?format=json-cors&key=8d9f3d5a> (Código de São Paulo)

2 - Mostre a relação dólar/real

<https://api.exchangeratesapi.io/latest?base=USD>

Crie os componentes em arquivos separados.

O componente 1 deve ser registrado globalmente

O componente 2 deve ser registrado localmente da instância Vue.js

Resultado:

<https://github.com/ricardoariz/treinavue-basico/tree/10-components>

Components (props)

Devemos utilizar **props** para passar dados de um componente pai para um filho. O dado é passado como valor de um atributo html.

App.vue

```
<template>
  <div id="app">
    <ListaProdutos :produtos="produtos" />
  </div>
</template>

<script>
import ListaProdutos from './components/ListaProdutos.vue'

export default {
  name: 'app',
  components: {
    ListaProdutos
  },
  data() {
    return {
      produtos: ['Produto 01', 'Produto 02', 'Produto 03']
    }
  }
}
</script>
```

Components (props)

ListaProdutos.vue

```
<template>
  <div>
    <ul>
      <li v-for="produto in produtos" :key="produto">{{produto}}</li>
    </ul>
  </div>
</template>

<script>
export default {
  name: 'ListaProdutos',
  props: ['produtos']
}
</script>
```

Repare que no componente ListaProdutos.vue existe a propriedade “props” que é um array com um elemento “produtos”. Esse elemento é um parâmetro esperado pelo componente e que está sendo utilizado no template

O App.vue passou esse parâmetro na utilização do ListaProdutos mandando um array com os produtos a serem exibidos.

Components (props estáticas e dinâmicas)

É possível passar qualquer tipo de dados JavaScript, como array's, objetos, boolean, números e mais. Para isso utilize sempre a dinâmica com **v-bind: ou :**.

No exemplo abaixo a prop “titulo” passada é um literal e não uma variável do componente pai, por isso não foi utilizado o **v-bind: ou :**

```
<div id="app">
  <blog-post
    titulo="Esse título é estático"
    :texto="blog.texto"
    :likes="blog.likes"
    :tags="['Frutas', 'Legumes', 'Cozinha']"
  ></blog-post>
</div>
```

Components (props validação)

É sempre recomendado validar o tipo de dado, para que o desenvolvedor não quebre a aplicação passando uma prop com um dado diferente do esperado.

```
props: {  
  titulo: String,  
  likes: Number,  
  tags: {  
    type: Array,  
    required: true  
  },  
  thumb: {  
    type: String,  
    default: "./img/img.png"  
  }  
}
```

Documentação

<https://br.vuejs.org/v2/guide/components-props.html>

Components (props one-way data flow)

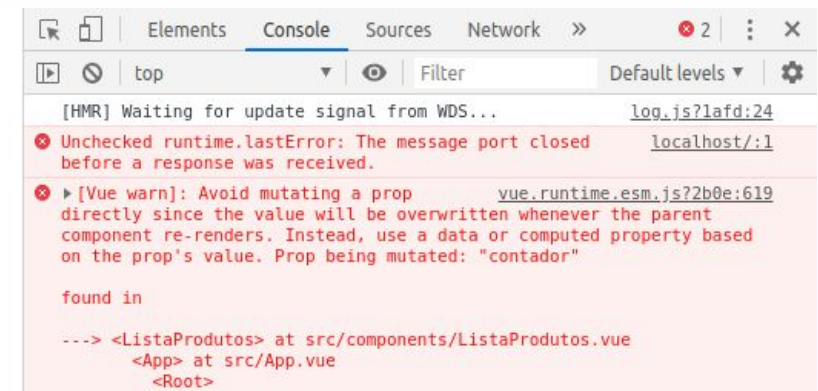
Não é recomendado mudar a propriedade no elemento filho. Se você modificar, ela não será refletida no elemento pai. Propriedades devem ser readonly.

```
<blog-post :contador="contador"></blog-post>
```

```
<article>
  <button @click="contador++">Adicionar {{contador}}</button>
</article>
```

Não recomendado, vue indicará um erro

Adicionar 1 1



Documentação

<https://br.vuejs.org/v2/guide/components-props.html>

Exercício 12

Altere o componente de cotação para que receba como parâmetro a moeda base e mostre a relação da moeda base com o Real

Cotação	Cotação
Mostrar cotação agora	Mostrar cotação agora
1 USD = R\$ 4.1662242563	1 EUR = R\$ 4.5516
USD	EUR

Resultado:

<https://github.com/ricardoaruiz/treinavue-basico/tree/11-components-props>

Components (events)

Como vimos anteriormente, props são para permitir a comunicação de um componente pai com um filho, já os eventos permitem a comunicação no sentido inverso. Lembre-se que não é boa prática alterar as props de um componente.

CustomButton.vue

```
<template>
  <button @click="emitirEvento">Emitir</button>
</template>

<script>
export default {
  name: 'CustomButton',
  methods: {
    emitirEvento() {
      this.$emit("meu-evento", "Minha Mensagem");
    }
  }
}
</script>
```

Components (events)

App.vue

```
<template>
  <div id="app">
    <p>{{mensagem}}</p>
    <custom-button @meu-evento="receberEvento"></custom-button>
  </div>
</template>

<script>
import CustomButton from './components/CustomButton.vue'

export default {
  name: 'app',
  components: {
    CustomButton
  },
  data() {
    return {
      mensagem: ''
    }
  },
  methods: {
    receberEvento(mensagem) {
      this.mensagem = mensagem;
    }
  }
}
</script>
```

Components (events.sync)

É comum utilizarmos eventos para enviarmos a mutação de uma prop feita no elemento filho, diretamente para o elemento pai. Podemos utilizar o atalho **.sync**.

Documentação:

<https://br.vuejs.org/v2/guide/components-custom-events.html#Modificador-sync>

CustomButton.vue

```
<template>
  <button @click="emitirEvento">Emitir</button>
</template>

<script>
export default {
  props: ['contador'],
  data() { return { contadorComponente: this.contador } },
  methods: {
    emitirEvento() {
      this.contadorComponente++;
      this.$emit("update:contador", this.contadorComponente);
    }
  }
}
</script>
```

Components (events.sync)

App.vue

```
<template>
  <div id="app">
    <p>{{contador}}</p>
    <custom-button @update:contador="contador = $event" :contador="contador"></custom-button>

    <!-- Faz a mesma coisa que a linha de cima -->
    <custom-button :contador.sync="contador"></custom-button>
  </div>
</template>

<script>
import CustomButton from './components/CustomButton.vue'

export default {
  name: 'app',
  components: {
    CustomButton
  },
  data() {
    return {
      contador: 0
    }
  }
}
</script>
```

Exercício 13

Altere o componente de cotação de forma que ele não mostre mais a mensagem do valor da cotação e sim emita para o componente pai que será responsável por mostrá-la.

Cotação

Mostrar cotação agora

USD

1 USD = R\$ 4.1662242563

Cotação

Mostrar cotação agora

XXX

Moeda base não encontrada

Resultado:

<https://github.com/ricardoarui/treinavue-basico/tree/12-components-events>

Components (slots)

Podemos utilizar slots quando precisamos de um conteúdo com estrutura dinâmica dentro de um componente.

ModalAlerta.vue

```
<template>
  <div class="modal">
    <button>Fechar</button>
    <slot></slot>
  </div>
</template>

<script>
export default {
  name: 'ModalAlerta',
}
</script>
```

Components (slots)

App.vue

```
<template>
  <div id="app">
    <modal-alerta>
      <p>{{mensagem}}</p>
      <button>Comprar</button>
    </modal-alerta>
  </div>
</template>

<script>
import ModalAlerta from '../components/ModalAlerta.vue'

export default {
  name: 'app',
  components: {
    ModalAlerta
  },
  data() {
    return {
      mensagem: "Isso vai aparecer no slot."
    };
  }
}
</script>
```

Components (named slots)

Podemos utilizar diversos slots em um componente. Para isso precisamos nomear os slots `<slot name="header"></slot>`, e para utilizar `<template v-slot:header></template>`.
ModalAlerta.vue

```
<template>
  <div class="modal">
    <header>
      <slot name="header"></slot>
    </header>
    <section>
      <slot>Fallback se não tiver conteúdo.</slot>
    </section>
    <footer>
      <slot name="footer"></slot>
    </footer>
  </div>
</template>

<script>
export default {
  name: 'ModalAlerta',
}
</script>

<style>
  header { border-bottom: 2px solid; }
  section { margin: 20px; }
  footer { border-top: 2px solid; }
</style>
```


Components (named slots)

App.vue

```
<template>
  <div id="app">
    <modal-alerta>
      <template v-slot:header>
        <h1>Header do Slot</h1>
      </template>
      <button>Comprar</button>
      <template v-slot:footer>
        <p>Esse é o footer do slot.</p>
      </template>
    </modal-alerta>
  </div>
</template>

<script>
import ModalAlerta from './components/ModalAlerta.vue'

export default {
  name: 'app',
  components: {
    ModalAlerta
  }
}
</script>
```

Exercício 14

Altere o componente de cotação de forma que o campo da moeda base e a mensagem fiquem cada um em um slot.

Cotação

USD

Mostrar cotação agora

1 USD = R\$ 4.1662242563

Cotação

XXX

Mostrar cotação agora

Moeda base não encontrada

Resultado:

<https://github.com/ricardoaruiz/treinavue-basico/tree/13-componentes-slots>

Conclusão

Esse foi um treinamento onde foram abordados os princípios e funcionalidades básicas do Vue.JS. Ainda temos mais assuntos a serem explorados que ficarão para uma segunda etapa do curso.

Para maiores informações sobre a biblioteca:

<https://vuejs.org/v2/guide/>

Dúvidas, sugestões, perguntas...

Obrigado

Padtec



padtec.com