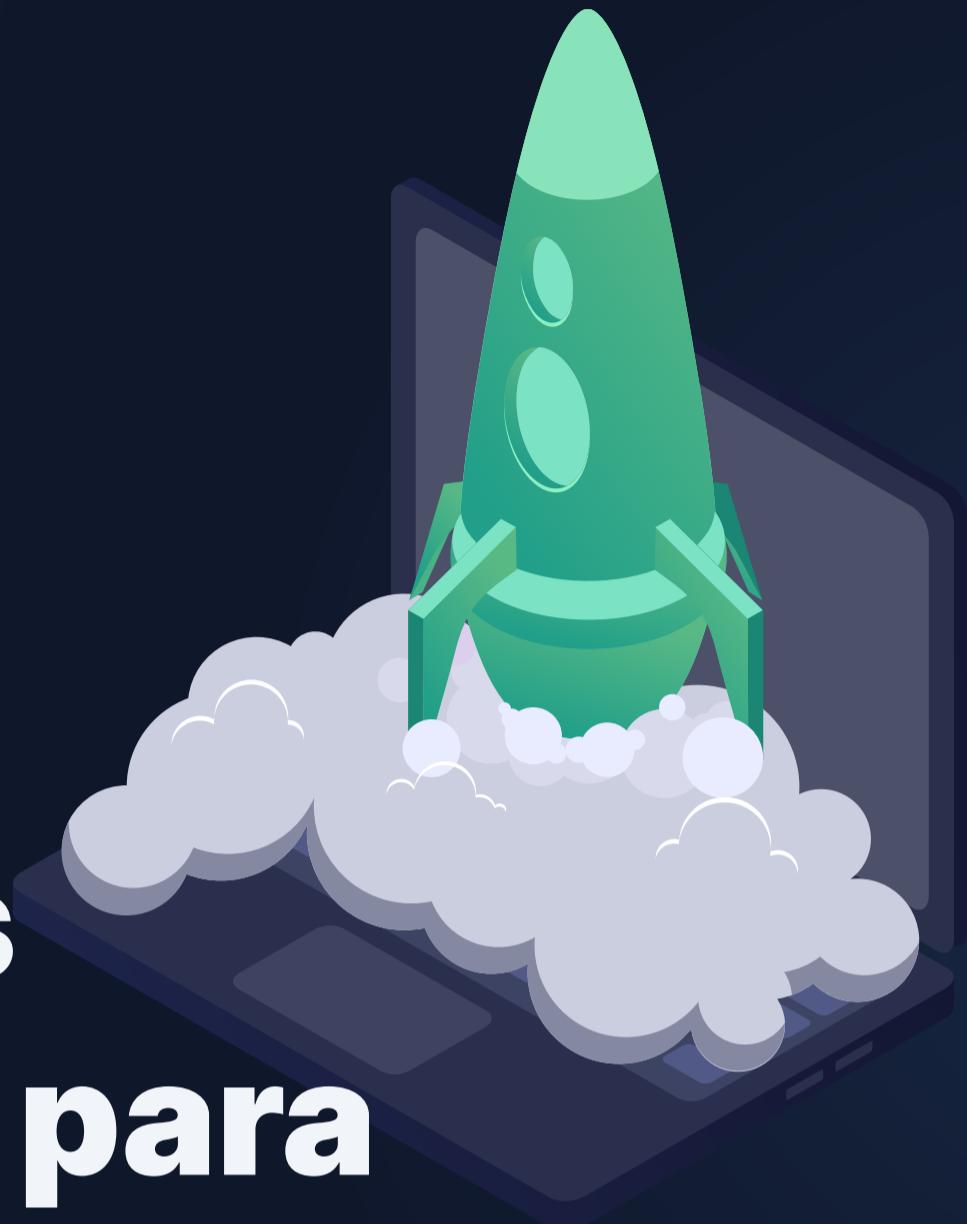




# 7

técnicas  
masters para  
você decolar  
com  
**Vue.js**



**10 BÔNUS PARA SE TORNAR UM  
MELHOR DESENVOLVEDOR VUE.JS!**

# Introdução

O Vue.js é um framework javascript de criação de interfaces criado em 2014 por Evan You. Com adeptos no mundo todo, virou o favorito dos desenvolvedores - inclusive o meu, claro!

Vantagens não lhe faltam. **É acessível, versátil e tem excelente desempenho.** Sua arquitetura é enxuta e precisa de poucas configurações no desenvolvimento do projeto, além de ter a possibilidade de integrar facilmente a uma aplicação existente por meio de um tag script.

Desde 2015 eu o utilizo e tenho, inclusive, um [\*\*curso completo gratuito\*\*](#) sobre o Vue.js no Youtube! Com a experiência de uso desse framework, descobri algumas técnicas que facilitam demais o nosso trabalho e quero compartilhar com você!

Criei esse e-book exclusivo, com o objetivo de te passar 7 técnicas masters e mais 10 dicas bônus para fazer você decolar com o Vue.js. Espero que aproveite bastante e aplique essas dicas valiosas a partir de agora.

**Antes de começar, vamos fazer um acordo?!** Se ao final da leitura você achar que este e-book te entregou algum conteúdo de valor, você assume o compromisso de divulgá-lo para amigos programadores para que eles também possam beneficiar-se do mesmo conteúdo que você se beneficiou, ok?

Afinal, conhecimento é tudo, né?! Então vamos compartilhar! Ficarei muito grato! 

**E para compartilhar, basta enviar esta URL para seus amigos:**



<https://pro.tiagomatos.com/curso-vuejs-pro>

Aproveite a leitura e vamos decolar. 

# Quem sou eu?

**Antes de começar, permita me apresentar.**

Me chamo Tiago Matos e iniciei minha carreira profissional aos 16 anos...e lá se vão mais de 20 anos de experiência!

Ao longo desses anos, trabalhei bastante na área de desenvolvimento web, o que me fez adquirir muito conhecimento e aprender vários truques.

Fui sócio e líder de projeto na minha agência digital por 9 anos e, desde 2015, atuo como engenheiro de software no mercado australiano.



The screenshot shows a YouTube channel profile for 'Tiago Matos' with 5,35 mil inscritos. The channel has an 'INSCRITO' button and a notification bell icon. Below the channel info, there's a navigation bar with 'INÍCIO', 'VÍDEOS', 'PLAYLISTS', and a 'REPRODUCIR TODOS' button. A course titled 'Vue.js do jeito ninja' is displayed, featuring two video thumbnails: '01 INSTALAÇÃO E REATIVIDADE' (7:45) and '02 O QUE SÃO DIRETIVAS' (4:13). Below the thumbnails, descriptions for each video are provided, along with the channel's statistics: 57 mil visualizações and 3 anos atrás for the first video, and 20 mil visualizações and 3 anos atrás for the second.

Além disso, meu **curso gratuito de Vue.js do jeito ninja** possui mais de 200 mil visualizações no YouTube - desde o seu lançamento, em 2017, vem ajudando vários programadores a conquistarem vagas de emprego no mercado de trabalho.

**Meu maior objetivo na  
internet é ajudar você a  
*aumentar o seu repertório*  
e se destacar entre os  
programadores!**

Por isso, se surgir alguma dúvida, é só perguntar lá no direct do meu instagram: [@tiagomatosweb](#) ou entrar na nossa comunidade do [Discord](#).

Aproveita também para me seguir e ficar por dentro de várias outras dicas sobre programação web, além de acompanhar os bastidores da minha rotina.

Vamosss 

# Peraí! 🤏

**Antes de continuar, quero te fazer um convite.**

Para uma melhor organização de suporte e atenção a você, eu criei um grupo exclusivo no Discord onde poderemos conversar sobre basicamente tudo de tecnologia.

O nosso Discord é um espaço para você tirar dúvidas, compartilhar conhecimentos e conversar sobre programação em geral. E também mais uma forma de ter contato mais próximo e rápido comigo. 🔥

Ah! Lá no Discord rola alguns eventos interessantes como lives, sessão de tira-dúvidas, sorteios e tudo isso ao vivo comigo.

Não vai perder, né?!



Para entrar, basta clicar no link abaixo:



<https://discord.tiagomatos.com>

Vamos nessa?! Conto com sua presença 😊



# Criando uma situação de exemplo

Para que tenhamos maior proveito nesse e-book, eu preciso criar uma situação de exemplo que represente o mundo real da programação e que as 7 técnicas masters e, ainda, as 10 dicas bônus façam sentido.

Então, para efeito de exemplificação, vamos criar uma página de carrinho de compras (checkout). Para não ficar muito complexo, focaremos apenas na seção da listagem de produtos dentro do carrinho e o cálculo do valor total.

Então, temos os seguintes componentes:

1. Componente `CheckoutProducts`  
para a listagem de produtos

2. Componente `CheckoutCalculator`  
para cálculo do total da compra



```
<script>
export default {
  name: 'CheckoutProducts',
  data() {
    return {
      products: [],
    };
  },
}
</script>
```



```
<script>
export default {
  name: 'CheckoutCalculator',
  data() {
    return {
    };
  },
}
</script>
```

💡 TÉCNICA MASTER 1:

# Use componentes funcionais

No Vue.js, estado é o que determina o comportamento de um componente e diz como ele é renderizado e o quanto é dinâmico. Nesse caso, há dois tipos de componentes: *stateful* com estado e *stateless* sem estado.

Componentes funcionais são aqueles que não possuem estado e nem a sua instância `this`.

Como eles foram criados basicamente para apresentação, não oferecem suporte à reatividade e não podem referenciar a si mesmo com o `this`, portanto, não mantêm ou rastreiam o estado. Trocando em miúdos, eles são componentes "burros" que fornecem apenas funções estáticas.

Os componentes funcionais ainda vão reagir, em alguns casos por exemplo, com novos valores de `props` sendo passados para dentro do componente. Mas, dentro do componente não é possível detectar nenhuma mudança de dados já que não há nenhum gerenciamento de estado.

**Ok Tiago, mas por que devo utilizar?**

É simples: **performance**.

Como não possuem estado, os componentes funcionais geralmente são executados rapidamente, pois não passam pelo processo de inicialização e re-renderização, assim como os componentes *stateful*, ou com estado.

Em apps complexos é bem provável que você note melhorias significativas na performance quando esta técnica é implementada.

## E quando devo utilizá-los? 🤔

Bom, devo dizer que componentes funcionais não são aplicáveis para a maioria dos casos, já que o objetivo de usar Vue.js é ter componentes com propriedades reativas. Além disso, você não consegue usufruir dos benefícios do Vue.js sem a grandeza da reatividade.

Mas, em alguns casos particulares, podemos usar essa técnica para melhorar sensivelmente a performance dos nossos apps. Então, segue uma pequena lista indicativa de potenciais casos de uso para componentes funcionais:

- Buttons;
- Pills;
- Badges;
- Tags;
- Cards;
- Componente para texto;
- Componentes usados para fazer um 'wrap' em componentes internos.

Mas...se ligue! 🧐

Seu app Vue.js versão 2 pode ficar até 3x vezes mais rápido quando aplicada a técnica de componentes funcionais. Todavia, no Vue.js versão 3, há pouca diferença em termos performáticos entre os componentes de estado e os funcionais - tanto que a documentação oficial recomenda não dar tanta importância para isso:

*"Performance gains from 2.x for functional components are now negligible in 3.x, so we recommend just using stateful components"*

## E como devo aplicar esta técnica?

Há duas formas de transformar um componente em componente funcional. Vamos criar o componente `ProductOffTag` para identificar que aquele produto adicionado ao carrinho está em promoção. Assim, vamos exibir uma tag mostrando o valor "10%" que representa, logicamente, o desconto.

A primeira forma é declarar um novo atributo chamado de `functional`, no objeto do componente que possui valor `true`.

```
<template>
  <div>
    {{ value }}
  </div>
</template>

<script>
  export default {
    name: 'ProductOffTag',
    props: ['value'],
    functional: true,
  };
</script>
```

E a segunda forma é adicionar um novo atributo a `tag template`, da seguinte forma:

```
<template functional>
  <div>
    {{ value }}
  </div>
</template>

<script>
  export default {
    name: 'ProductOffTag',
    props: ['value'],
  };
</script>
```

 TÉCNICA MASTER 2:

# “Watcher” imediatamente

Como o nome já indica, **watcher** é um observador que monitora as alterações de uma determinada variável e executa uma dada função, em caso de mudanças.

A questão é que, nativamente, o **watch** não roda de imediato quando o componente é inicializado, mas, em alguns casos, a depender da sua necessidade, é preciso fazer com que ele seja executado junto com a inicialização do componente.

Programadores tendem, naturalmente, a repetir a função que o **watch** executa no hook **created**. Isso não é errado, mas é uma repetição de código desnecessária.

Para exemplificar, vamos criar um **watch** na variável que indica a quantidade de cada produto para fazer o cálculo total do mesmo.

```
<script>
  export default {
    created() {
      this.total();
    },
    watch: {
      qty() {
        this.total();
      },
    },
    methods: {
      total() {
        // Cálculo do total
      },
    };
  }
</script>
```

Então, a dica aqui é transformar em objeto a função criada para o **watch** e adicionar os atributos `immediate` e `handler`. A opção `immediate` recebe um valor booleano `true` e o `handler` uma função. Veja como fica:



```
<script>
  export default {
    watch: {
      qty: {
        immediate: true,
        handler() {
          this.total();
        },
      },
    },
    methods: {
      total() {
        // Cálculo do total
      },
    },
  };
</script>
```

Agora este **watch** será executado uma vez, assim que o componente for inicializado e todas as vezes que a variável `qty` for alterada.

 TÉCNICA MASTER 3:

# Use variáveis locais dentro da computed property

A `computed property` consiste em uma função que depende dos valores de outras funções ou variáveis para ser executada, ou seja, ela é executada toda vez que alguma das suas dependências é alterada, gerando um novo valor.

No contexto do nosso carrinho de compras, certamente queremos apresentar o valor total da compra. Para isso, adicionaremos uma `computed subtotal()` para calcular o valor de cada produto, somado à sua quantidade. Vamos também adicionar uma outra `computed total()` para calcular o valor total da compra, incluindo o imposto representado pela variável `tax`.



```
<script>
  export default {
    name: 'CheckoutCalculator',
    computed: {
      subtotal() {
        // Cálculo subtotal
      },
      total() {
        return this.subtotal + this.tax;
      },
    },
  };
</script>
```

Perceba que o valor do `total()` vai depender sempre do valor do `subtotal()`, somado ao valor do imposto (tax).

Fazendo isso, toda vez que o valor do imposto for alterado devido à localidade do usuário, a função `total()` será executada, retornando um novo valor.

Como essa função também é dependente da função `subtotal()`, o Vue.js vai executar, também, o valor de `subtotal()`. Neste último caso, porém, a execução de `subtotal()` é desnecessária, pois, de fato, o valor final da mesma não foi alterado, ou seja, o subtotal continuou o mesmo.

Em um projeto complexo, esse pode ser um dos grandes fatores para a lentidão do app, já que isso causa grande consumo de memória/processamento, desnecessariamente.

Para melhorar esse desempenho, você pode criar variáveis locais dentro da propriedade `computed` quando tiver referência de uma outra `computed`.

## Como assim? 🤔

O Vue.js, nativamente, coloca em `cache` todo o resultado de uma propriedade `computed` justamente para melhorar a performance dos seus componentes. Quando alguma dependência é alterada, o Vue.js vai executar a função `computed`, e então "cachear" o resultado novamente.

No entanto, se uma das dependências for uma outra propriedade `computed`, o Vue.js não tem como colocar em cache o seu valor e por isso sempre solicitará a sua execução. Neste caso, a solução é justamente criar uma variável local dentro da `computed`, para receber o valor da propriedade `computed` dependente. Assim, o Vue.js vai "cachear" o valor nesta variável.

Vamos ao exemplo para clarear:

```
<script>
  export default {
    name: 'CheckoutCalculator',
    computed: {
      subtotal() {
        // Cálculo subtotal
      },
      total() {
        const subtotal = this.subtotal;
        return subtotal + this.tax;
      },
    },
  };
</script>
```

Veja que o resultado da função dependente `subtotal()` ficará na memória do Vue.js dentro de `total()` e não precisará executar `subtotal()`, melhorando sensivelmente a performance do seu app.

 TÉCNICA MASTER 4:

# Evitando memory leak

Memory leak significa, em tradução livre, “vazamento de memória”. Esse fenômeno ocorre quando um programa usa mais memória do que deve. Assim, esse excesso “vaza” e, normalmente, o app ou até a própria máquina trava. Em geral, a causa mais comum para isso acontecer no Vue.js é a utilização de bibliotecas externas de forma incorreta.

Programadores, principalmente iniciantes, costumam inicializar uma biblioteca externa no **hook** `mounted()`, mas esquecem de destruí-las após o uso.

Neste outro exemplo, vamos usar uma biblioteca figurativa chamada `NumberFormat` para poder formatar o valor total da compra e exibir da seguinte maneira: "R\$100,00".

```
<script>
  export default {
    mounted() {
      this.numberFormat = new NumberFormat();
    },
  };
</script>
```

**Sabe qual o problema nesse caso?** Essa ação faz com que o Vue.js injete uma instância da biblioteca na memória a cada inicialização deste componente, que ocasionará a "explosão" da memória, uma vez que não existe instrução alguma para a remoção desta biblioteca quando o componente é destruído. Exemplo: se você inicializar e destruir este componente 100 vezes, serão 100 instâncias da biblioteca `NumberFormat` na memória.

**Então qual é a dica?** Destruir a biblioteca todas as vezes que ela for instanciada. O ideal é que você aproveite a fase de destruição do componente utilizando o `hook beforeDestroy()` e injete o método de autodestruição da biblioteca. Esse método vai depender do tipo de biblioteca que você está utilizando, de acordo com sua forma de autodestruição. Na maioria das vezes, o método utilizado é o `destroy()`.

```
<script>
  export default {
    mounted() {
      this.numberFormat = new NumberFormat();
    },
    beforeDestroy() {
      this.numberFormat.destroy();
    },
  };
</script>
```

Quer uma dica extra? Caso sinta que criar sempre dois hooks - o `mounted()` e o `beforeDestroy()` seja muito verboso, você pode fazer um atalho usando o `listener $on`:

```
<script>
  export default {
    mounted() {
      this.numberFormat = new NumberFormat();
      this.$on('hook:beforeDestroy', () => {
        this.numberFormat.destroy();
      });
    },
  };
</script>
```

Ahhh! E no Vue.js versão 3, o hook `beforeDestroy()` foi renomeado para `beforeUnmount()`. 😊

 TÉCNICA MASTER 5:

# Entenda o ciclo de vida Vue.js

É essencial entender o ciclo de vida dos componentes Vue.js pois facilita muito na hora de decidir onde cada bloco de código do seu app deve ser inserido.

O Vue.js fornece alguns métodos que são chamados de **hooks** para que seja possível você interagir e criar funcionalidades customizadas, durante todo o ciclo de vida de um componente.

As ações dos métodos de ciclo de vida podem ser divididas em quatro categorias:

1. Criação
2. Montagem
3. Atualização
4. Destrução

Os métodos ou **hooks** usados durante o processo de criação servem para executar ações em seu componente, antes de adicioná-lo ao DOM - aqui é uma boa hora para configurar todo o seu componente durante a renderização, tanto do lado do cliente como do lado do servidor, como por exemplo, fazer uma conexão AJAX em busca de dados. Mas lembre-se, nessa fase, você ainda não tem acesso ao DOM.

Os métodos de criação são implementados através dos hooks `beforeCreate()` e `created()`.



```
<script>
  export default {
    created() {
      fetch('/api/products')
        .then(response => response.json())
        .then(json => {
          this.products = json;
        });
    }
  }
</script>
```

Os métodos de montagem permitem que você acesse seu componente imediatamente, antes e depois de ser renderizado pela primeira vez. Este é o ponto certo para fazer interações e manipulações diretas com o DOM.

O processo de montagem não deve ser usado para buscar dados para o componente, como por exemplo uma conexão AJAX, visto que os **hooks** dessa fase não funcionam durante a renderização do lado do servidor. Para isso, utilize o **hook** `created()` como mencionado anteriormente.

Os métodos de montagem são implementados através dos hooks `beforeMount()` e `mounted()`.



```
<script>
  export default {
    mounted() {
      this.$el.querySelector('.products');
    }
  }
</script>
```

Já os métodos de atualização são executados sempre que uma propriedade reativa usada por seu componente é alterada ou renderizada novamente. É aqui onde geralmente é feito o processo de "debugagem" desse componente - evite criar ações específicas dependente desses **hooks** para atualizar seu estado. Neste caso é mais recomendado utilizar `computed` ou `watch`.

Os métodos de atualização são implementados através dos hooks `beforeUpdate()` e `updated()`.

Os métodos de destruição são usados nos momentos finais do ciclo de vida de um componente. Eles permitem que você execute ações de limpeza, como vimos claramente na técnica master 4, quando removemos o uso de alguma biblioteca externa para evitar memory leak.

Os métodos de destruição são implementados através dos **hooks** `beforeDestroy()` e `destroyed()`.

A versão 3 do Vue.js introduziu alguns novos **hooks** no seu ciclo de vida, mas ainda assim, os que acabei de citar continuam sendo indispensáveis para você criar componentes com mais confiança.

 TÉCNICA MASTER 6:

# Utilize a técnica lazy load

**Lazy load** ou “carregamento lento”, em português, serve para adiar o carregamento de um objeto até o momento em que ele é necessário.

Como há diversos componentes em um projeto e muitos deles só são necessários em horas específicas, não há necessidade de carregar tudo de uma única vez. Até porque, isso pesaria demais o seu app já no primeiro carregamento.

Em nosso contexto do checkout, seria absolutamente desnecessário carregar os componentes `CheckoutProducts` e `CheckoutCalculator`, por exemplo, na página inicial do projeto. Basta imaginar que nessa página inicial do app não tem a listagem de produtos adicionados ao carrinho, nem tampouco o cálculo do valor total. Concorda?

Então, para melhorar a performance do seu app é altamente recomendado que você utilize a técnica do *lazy load* para importar componentes do Vue.js em geral e, mais especificamente, componentes para as rotas.

A técnica se resume em simplesmente utilizar a `arrow function` para retornar a declaração `import` do próprio javascript.

Vamos aos exemplos:

## Carregamento de rotas

```
routes: [
  {
    path: '/checkout',
    component: () => import('../pages/checkout.vue'),
    name: 'checkout',
  },
],
```

## Carregamento de componentes

```
<script>
  // /page/checkout.vue
  const CheckoutProducts = () => import('CheckoutProducts.vue');
  const CheckoutCalculator = () => import('CheckoutCalculator.vue');

  export default {
    name: 'checkout',
    components: {
      CheckoutProducts,
      CheckoutCalculator,
    },
  };
</script>
```

 TÉCNICA MASTER 7:

# Porque alterar props é anti padrão

Em resumo, passamos sempre dados pela árvore de componentes utilizando o atributo `props`, no Vue.js.

O fluxo natural de troca de dados entre componentes é: o componente pai passa os dados para seus componentes filhos através da `props` e os filhos sucedem a passagem de dados para as outras camadas.

Do modo contrário, usamos eventos `$emit` para passar dados do componente filho para o componente pai. Dessa forma, garantimos que cada componente fique separado e desacoplado um do outro e que apenas um componente altere seu próprio estado e somente o pai tenha a autorização de mudar os valores da `props`.

Tentar mudar esse fluxo pode dificultar a manutenção do seu sistema. Por exemplo, se notarmos algum comportamento fora do comum, ter certeza de onde as mudanças de dados estão acontecendo torna muito mais fácil o processo de "debugagem".

Além disso, os valores da `props` são substituídos quando há uma nova renderização do componente pai. Mesmo que você altere o valor de uma `props` no componente filho, este valor será substituído pelo Vue.js quando o componente pai for re-renderizado, perdendo assim as suas alterações.

**Então, qual é a dica?** Mantenha sempre este padrão de fluxo de dados entre componentes pais e filhos para facilitar a manutenção do seu sistema.

**E assim termino aqui minhas 7 técnicas masters para voar com o Vue.js. Mas...conforme prometido, tenho um bônus para você com mais 10 dicas. Continua comigo?!**



## DICAS BÔNUS:

# 10 dicas rápidas de Vue.js que você provavelmente não sabe!



### DICA BÔNUS 1

#### Desabilite a herança de atributos em componentes compartilhados.

Por padrão, todos os atributos de um componente que não forem reconhecidos como `props` serão automaticamente aplicados no elemento raiz deste mesmo componente como atributos HTML normais.

Para desativar essa funcionalidade, basta declarar como `false` a opção `inheritAttrs`, dentro do seu componente.



```
<script>
  export default {
    name: 'Product',
    inheritAttrs: false,
    data() {
      return {
        };
    },
  };
</script>
```

 DICA BÔNUS 2

Oculte interpolação não compiladas até que a instância Vue.js esteja pronta.

Utilize a diretiva `v-cloak`, que permanecerá no elemento como um atributo até que a instância Vue.js associada termine a compilação.

Com uma simples técnica CSS, podemos ocultar este elemento enquanto a diretiva estiver presente no elemento:



```
<div v-cloak>  
  {{ product.name }}  
</div>
```



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

 DICA BÔNUS 3

### Renderize um componente apenas uma vez.

Para renderizar um componente apenas uma vez, utilize a diretiva `v-once`. O Vue.js manterá este componente em cache e o mesmo será tratado como conteúdo estático e não será renderizado novamente, até mesmo em caso de troca de rotas.

Essa diretiva pode ser usada como uma estratégia para melhoria de desempenho.



```
<ul>
  <li
    v-for="product in products"
    v-once
  >
    {{ product.name }}
  </li>
</ul>
```



## DICA BÔNUS 4

### Ignore a interpolação do Vue.js

Seria meio estranho, mas se você quisesse exibir um título que tivesse o mesmo formato da interpolação do Vue.js como por exemplo "{{ Carrinho de compras }}", logicamente o Vue.js iria tentar interpolar isso e certamente exibiria um erro já que não existe nenhuma variável chamada "Carrinho de compras".

Mas, existe uma forma de fazer com que o Vue.js ignore o sinal de interpolação - basta usar a diretiva `v-pre`.

OBS: interpolação é o sinal duplo de chaves {{ }}



```
<h2 v-pre>{{ Carrinho de compras }}</h2>
```



## DICA BÔNUS 5

### Sempre valide suas props

Essa não é apenas uma recomendação minha, mas também do guia oficial de boas práticas do Vue.js. O maior objetivo aqui é facilitar a manutenção futura do seu app, já que é muito fácil esquecer o formato exato, tipo e outras peculiaridades que você usou para a `props`. E mais, seus colegas de trabalho não têm bola de cristal, então, facilite a vida dele e deixe claro como usar os seus componentes. 😊



```
<script>
  export default {
    name: 'Product',
    props: {
      tag: {
        type: String,
        required: true,
        validator(value) {
          return [
            'Promoção',
            'Destaque',
            'Últimas unidades',
          ].includes(value);
        },
      },
    },
  };
</script>
```



## DICA BÔNUS 6

### Crie filtros para formatar textos

Vue.js permite definir filtros que podem ser usados para aplicar a formatação de texto comum. Geralmente usamos filtros em dois lugares: dentro da interpolação e também das expressões `v-bind`.

No Vue.js 3 esse recurso foi removido, então, deve ser substituído por `computed` ou `method`.



```
<template>
  <div>
    <h3>{{ product.name | capitalize }}</h3>
  </div>
</template>

<script>
  export default {
    name: 'Product',
    filters: {
      capitalize(value) {
        return value.toUpperCase();
      },
    },
  };
</script>
```



## DICA BÔNUS 7

### Passe todas as props para os componentes filho

Caso você tenha necessidade de passar todas as `props` de um componente pai para um de seus filhos de forma fácil, basta usar a diretiva `v-bind` passando `$props` como argumento.



```
<Product v-bind="$props" />
```



## DICA BÔNUS 8

### Ajude o Vue.js a renderizar mais facilmente suas listas.

Use o atributo especial `key`, principalmente em conjunto da diretiva `v-for`, para facilitar a vida do Vue.js a renderizar o DOM de uma forma mais performática. Sem esse atributo, o Vue.js tem que fazer comparações de tipos de elementos, o que seria muito mais custoso do que simplesmente comparar identificadores únicos passados através do atributo `key`.



```
<ul>
  <li
    v-for="product in products"
    v-key="product.id"
  >
    {{ product.name }}
  </li>
</ul>
```

## DICA BÔNUS 9

### Mantenha seus objetos reativos.

O Vue.js fornece dois métodos para manipular objetos e ao mesmo tempo manter o sistema de reatividade dentro deles. Caso queira adicionar um novo atributo em seu objeto, utilize `this.$set()`. Em caso de querer remover, utilize `this.$delete()`.



```
this.$set(this.product, 'in_checkout', true);
this.$delete(this.product, 'in_checkout');
```

### Vídeo mais aprofundado sobre o tema:



<https://www.youtube.com/watch?v=O8AL5MBiGUQ>



## DICA BÔNUS 10

### Mantenha seus componentes em cache.

`<keep-alive>` é um componente abstrato, ou seja, ele não renderiza um elemento DOM em si e não aparece na cadeia pai do componente. Ele é usado basicamente para envolver componentes dinâmicos com intuito de armazenar em cache a sua instância, fazendo com que sua aplicação ganhe mais performance. Isso evita que o componente seja re-renderizado sem que haja de fato uma necessidade real.



```
<KeepAlive>
```

```
  <CheckoutCalculator />
```

```
</KeepAlive>
```

# Conclusão

**Viu só quanta dica massa  
estava perdendo?**



Comece a colocá-las em prática, e logo você verá a diferença nos seus projetos.

Quando começar a usar essas dicas, compartilhe comigo no meu instagram, **@tiagomatosweb!**

Quero saber se essas táticas estão salvando a sua vida da mesma forma que salvam a minha.

E claro, aproveite para seguir o meu perfil e ficar por dentro de todos os materiais e cursos disponíveis.

Um forte abraço jovem e vamos decolar Three small, colorful rocket ship icons.



Tiago Matos

## Meus links

