Cruzeiro do Sul
Educacional

Introdução ao Vue.js

Conteudista
Prof. Me. Marco Antonio Sanches Anastacio

Revisão Textual
Esp. Pérola Damasceno

OBJETIVOS DA UNIDADE

- Entender o que é o Vue.js e como utilizá-lo na construção de páginas *web* interativas;
- Compreender os principais conceitos associados ao uso do Vue.js, e as vantagens e desvantagens de sua utilização.

Atenção, estudante! Aqui, reforçamos o acesso ao conteúdo *on-line* para que você assista à videoaula. Será muito importante para o entendimento do conteúdo.

Este arquivo PDF contém o mesmo conteúdo visto *on-line*. Sua disponibilização é para consulta *off-line* e possibilidade de impressão. No entanto, recomendamos que acesse o conteúdo *on-line* para melhor aproveitamento.

Afinal, o que é Vue.js?

A popularidade do *JavaScript* entre os desenvolvedores *web* fez com que surgissem, ao longo do tempo, uma enorme variedade de *frameworks front-end*, destinados aos mais variados propósitos, como os mais utilizados no mercado atualmente: React, Angular e Vue.js.

Nesta Unidade, falaremos sobre o Vue.js, que é um *framework JavaScript* de código aberto (*opens source*) criado por Evan You, em 2014.

Utilizado no desenvolvimento de interfaces de usuário e *SPA* (*Single Page Applications*), o Vue.js foi projetado para ser progressivo, o que implica dizer que o desenvolvedor pode utilizá-lo tanto como uma biblioteca simples, com a finalidade de adicionar funcionalidades em uma página *web*, ou usá-lo como um *framework* completo para construir aplicativos mais completos e complexos.

Leitura

O que são aplicações SPA?

Conhecido por sua simplicidade e baixa curva de aprendizagem, ele se utiliza de uma arquitetura baseada em componentes reutilizáveis e independentes, que podem ser combinados para desenvolver interfaces complexas, e diretivas, que são atributos que adicionam funcionalidades e comportamentos dinâmicos aos elementos HTML.

Como uma comunidade muito ativa e farta documentação oficial (inclusive em Português), o Vue.js é um dos mais populares *frameworks* entre os desenvolvedores, como mostrou uma pesquisa realizada pelo *StackOverFlow* em maio de 2022.

Leitura

Visite a página do *StackOverFlow* e conheça mais detalhes dessa pesquisa.

De modo geral, o Vue.js é uma excelente escolha para projetos *web* que precisam de um *framework* reativo, modular, leve e fácil de aprender. Dentre as principais vantagens de sua utilização, podemos destacar as seguintes:

- **Fácil aprendizagem:** uma aplicação básica em Vue.js requer apenas um conhecimento básico de HTML, CSS e *JavaScript*;
- ***Performance:*** por ser leve e rápido, ele tem bom desempenho para aplicações de pequeno ou grande porte;
- **Flexibilidade:** pode ser integrado facilmente com outras bibliotecas e *frameworks*;
- **Modular e reativo:** os componentes são reutilizáveis de facilmente compartilhados e sua interface é atualizada quando os dados são alterados.

Entretanto, considerando as desvantagens do Vue.js, percebemos que a complexidade do paradigma reativo pode ser um problema no início. Além disso, em comparação com outros *frameworks* SPA, o Vue.js possui recursos um pouco mais limitados.

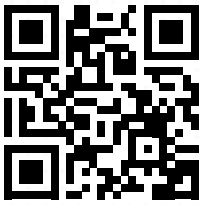
Primeiros Passos

Para começar a utilizar o Vue.js você precisa adicioná-lo ao seu projeto. Existem quatro formas principais para fazer isso, dependendo de suas necessidades:

- Importar a partir de uma rede de distribuição de conteúdo (CDN). Esta opção é recomendada apenas para prototipação ou aprendizagem;
- Baixar o arquivo .js e hospedá-lo em seu servidor *web*, que é a opção recomendada para projetos em produção;
- Instalação a partir do npm, que é recomendado para aplicações de larga escala;
- Utilização da ferramenta CLI oficial, que oferece recursos prontos para o desenvolvimento moderno de *front-end.*

Leitura

O que é npm?



Leitura

O que é CDN?

Nos exemplos desta Unidade, utilizaremos a primeira opção, porém, recomendamos que você teste e se familiarize com as demais formas para adicionar o Vue.js ao seu projeto.

Antes de partirmos para nosso primeiro exemplo prático, recomendamos que você instale a extensão Vue.js devtools, disponível para navegadores como o Chrome (Figura 1) e o Firefox, que permitirá inspecionar e realizar *debug* por meio de uma interface mais amigável.

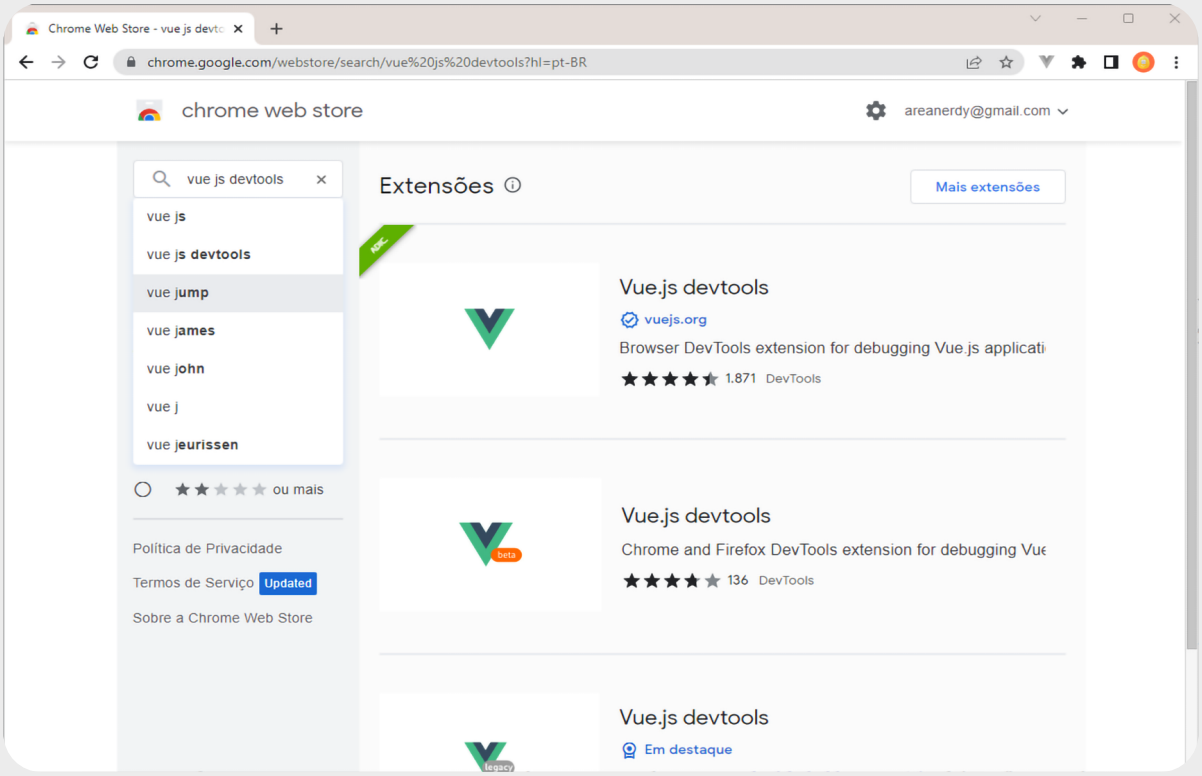


Figura 1 – Vue.js devtools na Chrome *Web* Store

Fonte: Acervo do conteudista

#ParaTodosVerem: imagem do navegador Chrome da Google. Do lado esquerdo, há um *menu* com fundo cinza e, acima, uma caixa de pesquisa com uma lupa à esquerda, que mostra o texto "vue js devtools". No centro, é exibido o título "Extensões" e, logo abaixo, são listados três retângulos com as opções para instalação da ferramenta no navegador. Todos os retângulos têm, à esquerda, o símbolo do Vue.js e o título "Vue.js devtools", seguido de uma breve descrição abaixo. Fim da descrição.

Hello World, Vue.js

Nossa primeira aplicação prática no mundo do Vue.js será um *Hello World*, cujo código é apresentado no exemplo a seguir:

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.  <title>Meu primeiro app Vue</title>
5.  <script src="https://unpkg.com/vue@3"></script>
6. </head>
7. <body>
8.  <div id="app">
9.    {{ message }}
10.  </div>
11.  <script>
12.    const app = Vue.createApp({
13.      data() {
14.        return {
15.        message: 'Hello World, Vue.js!'
16.      }}
17.    })
18.    app.mount('#app')
19.  </script>
20. </body>
21. </html>
```

Neste exemplo, criamos uma instância Vue simples que renderiza a mensagem "*Hello World*, Vue.js!" em um elemento HTML. Não se preocupe se o código pareceu um pouco confuso para você, vamos comentar cada parte no Quadro 1:

Quadro 1 – Exemplo comentado

Linha(s)	Descrição
Linha 5	Importamos o Vue.js a partir de um CDN
Linha 8	Definimos um elemento HTML com **id = "app"**, que renderiza a mensagem com interpolação de dados do Vue.js
Linha 12	Criamos uma instância Vue.js
Linha 13-16	Especificamos o modelo de dados contendo uma propriedade chamada **message**, que atribuímos o valor "Hello World, Vue.js!"
Linha 18	O método **mount** é utilizado para instanciar o Vue.js no elemento HTML que possui o **id = "app"**

Fonte: Elaborado pelo conteudista

Em nosso exemplo, o elemento HTML que será utilizado como destino é uma **div** cujo **id** é **app**. A interpolação de dados irá renderizar a mensagem nesse elemento e, para isso, utilizamos chaves duplas **{{}}** que contêm a propriedade que se deseja exibir, neste caso **mensagem**.

Este é um exemplo simples, mas que já mostra um pouco das características do *framework*. Nosso próximo passo será expandir os conceitos trabalhados aqui para que possamos criar aplicativos mais complexos e, para isso, falaremos um pouco sobre as diretivas e componentes.

Conhecendo o Vue.js

A arquitetura do Vue.js nos permite construir aplicações que renderize dados no *DOM* (*Document Object Model*) utilizando-se uma sintaxe de *template* bastante simples. O *DOM* é uma representação dos estilos, elementos e conteúdo de uma página HTML, que mudam na medida em que há uma interação do usuário, o que exige uma atualização constante do conteúdo pelo navegador.

Na manipulação de dados o Vue.js utiliza-se de diretivas integradas ao HTML, que oferecem flexibilidade aos elementos. Já no conceito de *template* são utilizados os componentes, que permitem a reutilização de modelos que podem ser vinculados à página.

Além disso, como vimos no exemplo anterior, a interpolação é uma forma simples, porém, poderosa de exibir dados dinamicamente. Veremos, a seguir, como esses conceitos se integram à construção de páginas *web*.

Criando uma Instância de Aplicação Vue

Uma aplicação Vue começa com a criação de uma nova instância, que é feita com a função **createApp**, seguida de um conjunto de propriedades que contém informações necessárias à configuração do componente raiz da aplicação. Esse componente constitui o ponto de partida para a renderização da aplicação, que é montada em um elemento HTML do DOM, por meio do método **mount**, que retorna a instância do componente raiz:

```
<script>
  const app = Vue.createApp({
    /* opções */
  })
  app.mount("#app")
</script>
```

Existem várias opções que adicionam propriedades à instância do componente, tais como:

- ***data***: retorna um objeto que contém dados da instância;
- ***methods***: configura um objeto que contém métodos referentes à instância do Vue;
- ***computed***: calcula valores baseados em outros dados;
- ***watch***: observa as alterações em dados específicos definidos na propriedade *data*;
- ***template***: define a estrutura e aparência do componente.

Toda instância Vue possui um ciclo de vida que passa por uma série de etapas desde sua criação até sua destruição. Desta forma, existem propriedades opcionais que caracterizam os diferentes estados de vida de uma instância Vue, que são:

- **beforeCreate:** a instância ainda não foi criada e, portanto, as propriedades de dados e métodos ainda não estão disponíveis;
- **created:** a instância foi criada, porém, ainda não está pronta para ser montada. As propriedades de dados e métodos já estão disponíveis;
- **beforeMount:** momento antes da instância ser montada no elemento HTML;
- **mounted:** a instância foi montada no DOM e está pronta para ser utilizada;
- **beforeUpdate:** a instância foi atualizada, porém, as mudanças ainda não foram refletidas no DOM;
- **updated:** a instância foi atualizada e agora as mudanças já foram refletidas no DOM;
- **beforeDestroy:** momento antes da instância ser destruída.
- **destroyed:** momento em que a instância é destruída e não está mais disponível.

Veja, a seguir, um exemplo simples do ciclo de vida de uma instância do Vue.js:

```
<html>
<head>
    <title>Ciclo de vida Vue</title>
    <script src="https://unpkg.com/vue@3"></script>
</head>

<body>
    <div id="app">
        <p>{{ message }}</p>
        <input v-model="message" />
    </div>
    <script>
        const app = Vue.createApp({
            data() {
                return {
                    message: 'Olá mundo Vue.js!'
                }
            },
            created() {
                console.log('Instância criada')
            },
            mounted() {
                console.log('Instância montada')
            },
            updated() {
                console.log('Instância atualizada')
            },
            unmounted() {
                console.log('Instância desmontada')
            }
        })
        app.mount('#app')
    </script>
```

```
</body>
</html></body>
</html>
```

No *menu* do navegador, procure por "Ferramentas do desenvolvedor" e observe os estados de vida do Vue.js. Perceba que o estado **updated()** somente é disparado quando alteramos o campo **input** para "Olá mundo!", como mostra a Figura 2.

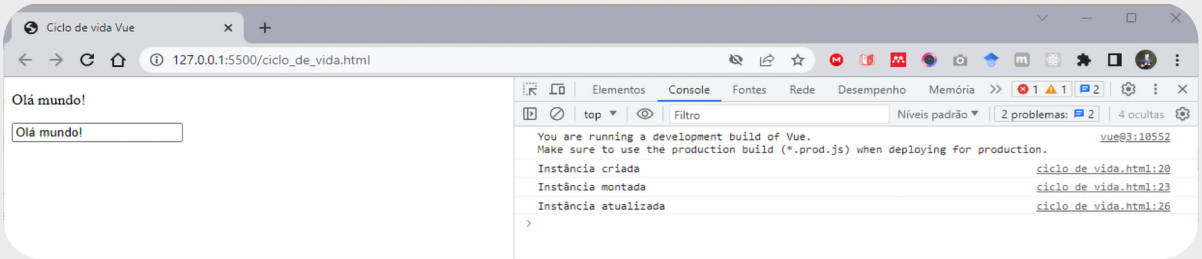


Figura 2 – Ciclo de vida Vue.js

Fonte: Acervo do conteudista

#ParaTodosVerem: imagem do navegador Chrome, da Google, com fundo na cor branca. Ao lado esquerdo, há um título com o texto "Olá mundo!" e, logo abaixo, um campo *input* com o texto "Atualizar". À direita, é exibida a aba de ferramentas do desenvolvedor, com a guia "Console" selecionada com um traço na cor azul na parte inferior. Logo abaixo, uma tabela exibe, do lado esquerdo, os estados do Vue.js e à direita o nome da aplicação (ciclo_de_vida.js) seguido pela linha do código que está sendo executada. Fim da descrição.

Leitura

O que são as ferramentas de desenvolvimento do navegador?

Diretivas do Vue.js

Sinteticamente, podemos definir as diretivas como sendo atributos adicionados a um elemento HTML que estendem seu comportamento e funcionalidade. Na prática, teremos um comportamento que reage a interações do usuário, atualizando a interface dinamicamente.

No Vue.js as diretivas são precedidas pelo prefixo **"v-"**, são aplicadas aos elementos HTML e possuem comportamentos distintos, que são configuráveis por meio de opções, como veremos mais adiante. As principais diretivas Vue.js são:

- **v-model:** é utilizada no conceito *"two-way-databind"*, ou seja, permite criar um vínculo bidirecional de dados entre um elemento HTML (como o **input**, por exemplo) e um componente Vue.js. A diretiva **v-model** permite atualização automática do componente à medida que um campo de formulário recebe informações. O *two-way-databind* é um dos conceitos mais relevantes para o desenvolvimento de aplicações web interativas e é uma das principais vantagens do Vue.js em relação a outros *frameworks*.

Vejamos um exemplo simples, que permite ao Vue observar o valor do campo **input** e atualizar a variável **nome**.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.    <title>Two-way-data-bind Vue.js</title>
5.    <script src="https://unpkg.com/vue@3"></script>
6. </head>
7. <body>
8.    <div id="app">
9.       <label for="nome">Nome:</label>
10.       <input type="text" v-model="nome">
11.       <p>Olá {{ nome }}! Bem-vindo ao Vue</p>
12.    </div>
13.    <script>
14.       const app = Vue.createApp({
15.          data() {
16.             return {
17.                nome: ''
18.             }
19.          }
20.       })
21.       app.mount('#app')
22.    </script>
23. </body>
24. </html>
```

Na prática

Compare este exemplo com os anteriores e observe a função do **v-model** que adicionamos ao elemento **input** na linha 10.

- **v-bind:** é utilizada para vincular, dinamicamente, um valor ao atributo de um elemento HTML como o **src** de uma imagem. Veja o exemplo a seguir:

```
<img v-bind:src="imagemUrl" alt="Imagem">
```

Na prática

Neste exemplo, adicionamos ao modelo de dados uma propriedade chamada *imagemUrl*, que atribuímos o valor **https://bit.ly/3RijRf9**

- **v-if:** é utilizada para renderizar um elemento HTML com base em uma expressão condicional. Caso o resultado da expressão seja **true** (verdadeiro) o elemento HTML será renderizado; caso contrário, não. Veja o seguinte exemplo:

```
<div v-if="mostrarImagem">
  <img v-bind:src="imagemUrl" alt="Imagem">
</div>
```

Agora, a imagem será exibida somente se a propriedade **mostrarImagem** for **true** (verdadeira). Para que o exemplo funcione adequadamente, essa propriedade foi adicionada ao modelo de dados com o valor **true**.

- **v-on:** é utilizada para anexar eventos a um elemento HTML, permitindo executar um método a partir da interação do usuário, como o clique de um botão.

Para nosso próximo exemplo, criaremos um botão para ocultar/exibir a imagem, utilizando as diretivas **v-if** e **v-on** conjuntamente. O código HTML ficou assim:

```
<div id="app">
  <h1>{{ titulo }}</h1>
  <p v-if="mostrarImagem">
      <img v-bind:src="imagemUrl" alt="Imagem">
  </p>
  <p>Clique no botão para ocultar / exibir a imagem</p>
  <button v-on:click="alternarImagem">Clique-me</button>
</div>
```

A diretiva **v-on** adiciona um ouvinte de evento de clique ao botão e, quando clicamos nele, o método **"alternarImagem"** é chamado, alterando o valor da propriedade **mostrarImagem**, do modelo de dados entre **true** e **false**. Observe o script Vue como ficou:

```
<script>
   const app = Vue.createApp({
     data() {
       return {
          titulo: 'Página web dinâmica',
          imagemUrl: 'https://files.readme.io/86176b7-vuejs.png',
          mostrarImagem: true
         }
},
     methods: {
       alternarImagem() {
       this.mostrarImagem = !this.mostrarImagem;
       }
}
   })
   app.mount('#app')
</script>
```

No exemplo, ***methods*** refere-se à propriedade do Vue.js, que contém todos os métodos que podem ser chamados em resposta a eventos, ou para executar alguma tarefa especificamente criada pelo desenvolvedor da aplicação.

- **v-for:** é utilizada para renderizar dinamicamente uma lista de elementos, iterando cada item de base dados de acordo com um template. Essa diretiva tem uma sintaxe na forma ***item in items***, na qual **items** corresponde à base de dados, e **item** é um elemento dessa base que está sendo iterado.

O *template* para o próximo exemplo é dado:

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

A diretiva **v-for** itera sobre o **array** de nomes para gerar uma lista na qual para cada nome é gerado um elemento **li**. O **array** de nomes foi criado dentro do objeto **data**:

```
<script>
  const app = Vue.createApp({
    data() {
    return {
        nomes: ['Marco', 'Mariana', 'Ana', 'Alexandre', 'Suzana']
    }}
  })
  app.mount('#app')
</script>
```

- **v-show:** é utilizada de forma semelhante ao **v-if** para controlar a exibição condicional de elementos com base em uma expressão. Vejamos um exemplo completo:

```
<html>
<head>
  <script src="https://unpkg.com/vue@3"></script>
</head>

<body>
  <div id="app">
     <p v-show="mostrarMsg"> {{ message }} </p>
     <button v-on:click="mostrarMsg = !mostrarMsg">
      Alterar exibição</button>
  </div>
<script>
    const app = Vue.createApp({
      data() {
        return {
            message: "Esta mensagem será ocultada / exibida",
            mostrarMsg: true
      }}
    })
    app.mount('#app')
</script>
</body>
</html>
```

Neste exemplo, um botão alterna a visibilidade do elemento HTML **<p>**, ocultando ou exibindo a mensagem. Observe que, diferentemente do exemplo com **v-if**, aqui optamos por não utilizar um método. Assim, alteramos o valor da propriedade **mostrarMsg** na própria diretiva **v-on**. Apesar de não ser uma prática recomendada, podemos utilizar dessa forma sempre que a implementação for suficientemente simples para isso.

A despeito do resultado na prática ser o mesmo, a principal diferença entre o **v-if** e o **v-show** está na forma como cada um altera a visibilidade de um elemento utilizando o CSS. Enquanto o **v-if** insere ou remove elementos no DOM, o **v-show** somente altera a propriedade **display** para **none**, ou seja, o elemento ainda existe no DOM, somente sua visibilidade foi alterada.

O **v-show** é mais leve e rápido, sendo recomendado quando não precisamos de grandes mudanças na estrutura da página, apenas alterar a visibilidade de um elemento por meio do CSS. Já o **v-if**, devido à *performance*, deve ser utilizado quando a renderização condicional é menos frequente.

Na prática

Em seu navegador favorito, abra "Ferramentas do desenvolvedor" e, na inspeção de "Elementos", compare o resultado do uso do **v-if** e do **v-show**.

Utilizando Componentes

Os componentes do Vue constituem um dos principais recursos do *framework* e permitem criar elementos HTML personalizados, independentes e frequentemente reutilizáveis, para construção de interfaces de usuário.

Um componente Vue é essencialmente uma instância com opções pré-definidas, onde registramos o componente a partir da criação de um objeto composto por um bloco de códigos com uma estrutura HTML. Essa instância segue a mesma configuração utilizada exemplos anteriores com o objeto app.

Quando criamos um componente, podemos definir propriedades e eventos próprios, que podem ser compartilhados com outros componentes, possibilitando a construção de aplicações complexas e modulares, compostas de pequenos componentes independentes.

A seguir, vamos implementar um componente que lista o nome de carros armazenados em um *array*:

```
1.<html>
2. <head>
3.  <script src="https://unpkg.com/vue@3"></script>
4. </head>
5. <body>
6.  <div id="app">
7.     <h1>{{ titulo }}</h1>
8.     <carros-list
9.        v-for="carro in carros"
10.        v-bind:carro = "carro"
11.     ></carros-list>
12. </div>
13.<script>
14.  const app = Vue.createApp({
15.     data() {
16.        return {
17.           titulo: "Utilizando componentes",
18.           carros: ["Fusca", "Kombi", "Brasília", "Uno", "Opala"]
19.     }}
20.   })
21.   app.component("carros-list", {
22.        props: ["carro"],
23.        template: "<li> {{ carro }} </li>"
24.   })
25.   app.mount("#app")
26.</script>
27.</body>
28.</html>
```

Neste exemplo, criamos uma instância do Vue na linha 14 (**Vue.createApp**), que é montada no elemento HTML ** ** (linhas 6 a 12). O componente **"carros-list"** é definido nas linhas 21 a 24, com propriedade carro (**props: ["carro"]**). O componente recebe *array* como propriedade (**props**) e o utiliza em o *template* renderizar a lista de carros com a diretiva **v-for** (linhas 8 a 11).

Site

Navegue na documentação oficial do Vue.js e conheça um pouco mais sobre os componentes.

A reutilização de componentes é um recurso precioso do Vue.js, pois permite a criação de componentes genéricos, que podem ser utilizados diversas vezes em nossas aplicações.

Nosso último exemplo desta Unidade explora essa característica do Vue:

```
<html>
<head>
  <script src="https://unpkg.com/vue@3"></script>
</head>

<body>
  <div id="app">
     <h1>Reutilizando componentes</h1>
    <button-counter></button-counter>
    <button-counter></button-counter>
    <button-counter></button-counter>
  </div>
<script>
    const app = Vue.createApp({})
    app.component("button-counter", {
        data(){
            return {
                count: 0
        }},
        template:`<button v-on:click="count++">
                Você clicou {{ count }} vezes.
                </button>`
    })
```

```
    app.mount('#app')
</script>
</body>
</html>
```

A ideia desse exemplo foi criar um componente do tipo **button** cujo evento **click** adiciona uma unidade a um contador (**count**). Perceba que implementamos o componente **button-counter** uma única vez, mas, reutilizamos o *template* criado três vezes, sem a necessidade de replicar o código utilizado.

Os componentes são a base da estrutura de uma aplicação no Vue.js e podem ser reutilizados em diferentes contextos, como objetos JavaScript que possuem propriedades como data, **methods** e **computed**. As instâncias de um componente possuem seu escopo isolado, o que significa que não interferem com outros componentes, e ciclo de vida próprio, que inclui etapas como **created**, **mounted**, **updated** e **destroyed.**

Em Síntese

Como vimos nesta Unidade, o Vue.js é uma biblioteca JavaScript moderna e progressiva para a construção de interfaces de usuário interativas e reativas, que, por sua simplicidade, eficiência e flexibilidade, permite aos desenvolvedores criarem aplicativos *web* complexos de forma mais ágil. Os componentes facilitam a organização e manutenção do código e as diretivas simplificam a manipulação dinâmica do DOM, conferindo ao Vue.js uma *performance* eficiente, com uma renderização rápida e otimizada, que garante ótima experiência do usuário.

MATERIAL COMPLEMENTAR

Sites

Codesandbox
É uma plataforma em nuvem que permite e o desenvolvimento e compartilhamento *on-line* de soluções *web*, de forma rápida e eficiente.
https://bit.ly/3RhidKJ

Vue.js – O *Framework* JavaScript Progressivo
Visite e explore a página oficial do Vue e aprofunde seus estudos.
https://bit.ly/3Pz0ux6

learnvue.co
É uma plataforma com conteúdos para aprendizagem do Vue.js.
https://bit.ly/44HrEpS

Vue.js Brasil
Conheça a página da comunidade brasileira dedicada ao Vue.js, explore e enriqueça seus conhecimentos com diversos artigos em Português.
https://bit.ly/3resgFF

REFERÊNCIAS BIBLIOGRÁFICAS

CLARK, R. *et al.* **Introdução ao HTML5 e CSS3:** a evolução da web. Rio de Janeiro: Alta *Books*, 2014.

SILVA, M. S. **jQuery**: A biblioteca do programador JavaScript. São Paulo: Novatec, 2014.

SILVA, M. S. **CSS3:** Desenvolva aplicações ***web*** profissionais com uso dos poderosos recursos de estilização das CSS3. São Paulo: Novatec, 2012.

TERUEL, E. C. **HTML 5:** guia prático. 2. ed. São Paulo: Erica, 2013.

W3C. **CSS/treinamento.** 23/10/2015. Disponível em: <https://www.w3.org/community/webed/wiki/CSS/Training>. Acesso em: 04/07/2023.