Módulo 1: Introdução ao Vue.js

- Vuejs componentes reutilizáveis
- props
- emit

Os **componentes reutilizáveis** são uma das principais funcionalidades e vantagens do Vue.js. Eles permitem que você divida sua aplicação em pequenos blocos modulares, cada um com sua lógica e template, que podem ser reutilizados em diferentes partes do seu projeto, facilitando a manutenção e a escalabilidade.

O que é um Componente Reutilizável?

Um **componente** é essencialmente uma instância Vue que encapsula sua própria lógica (dados, métodos e propriedades computadas) e uma interface de usuário específica (template HTML). Ao criar componentes reutilizáveis, você pode reutilizá-los em várias partes do seu projeto sem duplicar código.

Vantagens de Componentes Reutilizáveis:

- **Modularidade**: Quebra sua aplicação em blocos menores e independentes, facilitando o desenvolvimento e a manutenção.
- **Reuso**: Componentes podem ser usados várias vezes em diferentes contextos, reduzindo a duplicação de código.
- **Manutenibilidade**: Alterações no componente se propagam para todos os locais em que ele é usado, facilitando a atualização.
- **Legibilidade**: Ao dividir a interface em componentes menores, o código se torna mais legível e fácil de entender.

Criando um Componente Reutilizável

Exemplo Simples de Componente:

Aqui está um exemplo básico de um componente reutilizável chamado `ButtonComponent`, que renderiza um botão e exibe um texto personalizado.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue Component</title>
</head>
<body>
  <div id="app">
    <!-- Usando o componente reutilizável -->
    <button-component text="Clique Aqui"></button-component>
    <button-component text="Enviar"></button-component>
    <button-component text="Cancelar"></button-component>
  </div>
```

```html
  <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.js"></script>
  <script>
    // Definindo um componente reutilizável chamado 'button-component'
    Vue.component('button-component', {
      props: ['text'],  // Propriedade 'text' que será passada para o componente
      template: '<button>{{ text }}</button>'  // Template HTML do componente
    });

    // Instância principal Vue
    new Vue({
      el: '#app'
    });
  </script>
</body>
</html>
```

Explicação:

1. **Definindo um Componente**:
   - O componente `button-component` é registrado usando `Vue.component`.

    - O componente tem uma propriedade chamada `text` que é passada como um prop para o componente, permitindo a personalização de cada botão.
    - O template do componente contém apenas um botão HTML que exibe o texto passado como prop.
2. **Usando o Componente**:
    - No template principal do Vue (`#app`), usamos o componente várias vezes, passando diferentes textos para o prop `text`. Isso torna o componente reutilizável e flexível.

Como Funciona a Reutilização de Componentes?

Componentes reutilizáveis seguem uma estrutura de dados e métodos encapsulados, o que significa que eles podem ter seus próprios dados, propriedades e lógica, mas esses dados não afetam outros componentes. Cada vez que um componente é usado, ele gera uma nova instância com seu próprio escopo.

Comunicação Entre Componentes

Para criar componentes realmente reutilizáveis, é fundamental entender como a comunicação entre eles ocorre. Vue.js utiliza duas principais maneiras de comunicação entre componentes:

1. **Props (Propriedades)**: Permitem que o **componente pai** passe dados para o **componente filho**.
2. **Eventos Personalizados (Custom Events)**: Permitem que o **componente filho** envie informações de volta para o **componente pai**.

1. Usando Props para Passar Dados:

Props são parâmetros que o componente pai passa para o filho. Vamos melhorar o exemplo anterior e adicionar uma cor ao botão.

```
<template>
  <div id="app">
    <button-component text="Clique Aqui" color="green"></button-component>
    <button-component text="Enviar" color="blue"></button-component>
    <button-component text="Cancelar" color="red"></button-component>
  </div>
</template>

<script>
  Vue.component('button-component', {
    props: ['text', 'color'], // Props recebidas
    template: '<button :style="{ backgroundColor: color }">{{ text }}</button>' // Usando
  });

  new Vue({
    el: '#app'
  });
</script>
```

Explicação:

- Agora o componente `button-component` recebe duas props: `text` e `color`.
- A cor do botão é aplicada dinamicamente usando a prop `color`, o que permite ainda mais personalização e reuso.

2. Emitindo Eventos para o Componente Pai:

Muitas vezes, um componente filho precisa se comunicar com o componente pai, como quando um botão é clicado e uma ação precisa ocorrer no pai. Isso pode ser feito emitindo eventos personalizados.

```
<template>
  <div id="app">
    <button-component text="Clique Aqui" @clicked="handleClick"></button-component>
  </div>
</template>

<script>
  Vue.component('button-component', {
    props: ['text'],
    template: '<button @click="emitClick">{{ text }}</button>',
    methods: {
      emitClick() {
        this.$emit('clicked'); // Emite um evento personalizado chamado 'clicked'
      }
    }
  });

  new Vue({
    el: '#app',
    methods: {
      handleClick() {
        alert('O botão foi clicado!');
      }
    }
  });
</script>
```

Explicação:

- O componente filho (`button-component`) emite um evento chamado `clicked` quando o botão é clicado.
- O componente pai escuta esse evento e chama o método `handleClick`, que exibe um alerta.
- Esse padrão de emitir eventos permite que o componente filho informe o componente pai sobre eventos importantes, mantendo a modularidade e reuso.

Componentes Dinâmicos

Outra funcionalidade poderosa é a capacidade de alternar entre diferentes componentes de forma dinâmica. Vue.js facilita isso com o uso de componentes dinâmicos.

Exemplo de Componentes Dinâmicos:

```
<template>
  <div id="app">
    <button @click="currentComponent = 'component-a'">Componente A</button>
    <button @click="currentComponent = 'component-b'">Componente B</button>

    <component :is="currentComponent"></component>
  </div>
</template>

<script>
  Vue.component('component-a', {
    template: '<div>Este é o Componente A</div>'
  });

  Vue.component('component-b', {
    template: '<div>Este é o Componente B</div>'
  });

  new Vue({
    el: '#app',
    data: {
      currentComponent: 'component-a'
    }
  });
</script>
```

Explicação:

- O `<component>` dinâmico muda entre `component-a` e `component-b` com base no valor de `currentComponent`.
- Isso é útil para criar interfaces que alternam entre diferentes layouts ou modos, sem precisar esconder ou mostrar manualmente elementos no DOM.

Slots: Personalização do Conteúdo de um Componente

Os **slots** permitem que você passe conteúdo personalizado para dentro de um componente. Isso dá ainda mais flexibilidade ao criar componentes reutilizáveis.

```
<template>
  <div id="app">
    <card-component>
      <h1>Título do Cartão</h1>
      <p>Este é o conteúdo do cartão, passado como slot.</p>
    </card-component>
  </div>
</template>

<script>
  Vue.component('card-component', {
    template: `
      <div class="card">
        <slot></slot> <!-- Conteúdo personalizado vai aqui -->
      </div>
    `
  });

  new Vue({
    el: '#app'
  });
</script>
```

Explicação:

- O `slot` no componente `card-component` permite que você insira qualquer conteúdo HTML dentro do componente ao usá-lo.
- Isso torna o `card-component` ainda mais reutilizável, pois o conteúdo do cartão pode variar conforme a necessidade.

Vamos praticar criando nosso próprio componente? Segue a estrutura abaixo:

index.html > html > body > script

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Curso Vue js - components</title>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstr
</head>
<body>
    <div id="app">
        <h1>{{ titulo}}</h1>
        <menu-bar></menu-bar>
        <menu-bar></menu-bar>
        <menu-bar></menu-bar>
    </div>

    <template id="menu">
        <ul>
            <li>Item 1</li>
            <li>Item 2</li>
            <li>Item 3</li>
        </ul>
    </template>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.js"></script>
    <script>
        Vue.component('menu-bar', {
            template: "#menu"
        });
        var app = new Vue({
            el: '#app',
            data: {
                titulo: 'Vue JS',
            }
        });
    </script>
</body>
</html>
```

Como estamos vendo na imagem a seguir temos a reutilização do nosso componente criado, de forma simples e prática, essa é a magia dos componentes.

```
index.html > html > body > script
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Curso Vue js - components</title>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css" int
</head>
<body>
    <div id="app">
        <h1>{{ titulo}}</h1>
        <menu-bar></menu-bar>
        <menu-bar></menu-bar>
        <menu-bar></menu-bar>
    </div>

    <template id="menu">
        <ul>
            <li>Item 1</li>
            <li>Item 2</li>
            <li>Item 3</li>
        </ul>
    </template>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.js"></script>
    <script>
        Vue.component('menu-bar', {
            template: "#menu"
        });
        var app = new Vue({
            el: '#app',
            data: {
                titulo: 'Vue JS',
            }
        });
    </script>
</body>
</html>
```

utilização do componente criado e reutilizando o mesmo

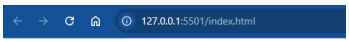
criação do componente

nome do componente

chamada do id do componente criado

titulo com interpolação de variáveis

Resultado:



Vue JS

- Item 1
- Item 2
- Item 3

- Item 1
- Item 2
- Item 3

- Item 1
- Item 2
- Item 3