

## exemplo vue 3 vite bootstrap tooltip

🕒 Thought for 19 seconds ▾

Vamos criar um exemplo de uso do Bootstrap Tooltip em um projeto Vue 3 com Vite.

Como o Bootstrap não é automaticamente integrado com Vue 3 (especialmente os componentes que precisam de JavaScript, como tooltips), temos algumas opções.

Uma maneira é usar o Bootstrap via CDN, mas isso pode não ser ideal para o Vite porque o Vite é um build tool moderno.

Outra maneira é usar o Bootstrap como um módulo, mas note que o Bootstrap 5 não depende do jQuery, então podemos usar o pacote oficial do Bootstrap.

No entanto, para usar os tooltips do Bootstrap, precisamos inicializá-los manualmente porque o Bootstrap não sabe sobre a existência dos elementos criados dinamicamente pelo Vue.

Passos:

1. Criar um projeto Vue 3 com Vite.
2. Instalar o Bootstrap.
3. Importar o CSS do Bootstrap.
4. Inicializar os tooltips.

Vamos optar por usar o Bootstrap via npm e inicializar os tooltips usando a API do Bootstrap, mas de uma forma que funcione com o Vue.

Como o Vue manipula o DOM virtualmente, podemos usar o ciclo de vida do componente para inicializar os tooltips.

Vamos criar um exemplo de um botão com tooltip.

No entanto, note que o Bootstrap Tooltip requer a inicialização via JavaScript. Para isso, podemos usar o `createPopper` (que já é uma dependência do Bootstrap 5) ou simplesmente chamar a função `tooltip` do Bootstrap.

Mas o Bootstrap 5 não tem uma função `tooltip` global como o Bootstrap 4 com jQuery. Em vez disso, usamos a classe `Tooltip` do Bootstrap.

Vamos fazer:

1. Instalar o Bootstrap:

```
npm install bootstrap
```

2. No componente, vamos importar o CSS do Bootstrap.

3. No template, adicionar um elemento com o atributo `data-bs-toggle="tooltip"` e o título.

4. No código Vue, vamos inicializar o tooltip no momento adequado (quando o componente for montado).
5. Também é importante limpar o tooltip quando o componente for desmontado.

Vamos criar um componente chamado `TooltipExample.vue`.

No entanto, se tivermos muitos tooltips, podemos criar uma diretiva personalizada para facilitar.

Mas para o exemplo, vamos fazer de duas formas:

- a) Inicializando o tooltip manualmente no `onMounted` e limpando no `onUnmounted`.
- b) Criando uma diretiva personalizada.

Primeiro, a forma manual.

Em um componente Vue 3 com Composition API:

Vou mostrar um exemplo completo de como usar Bootstrap tooltips em Vue 3 com Vite. Aqui está uma implementação passo a passo:

## 1. Instalar as dependências

bash

```
npm install bootstrap @popperjs/core
```

## 2. Configurar Bootstrap no projeto

`main.js` ou `main.ts`:

javascript

```
import { createApp } from 'vue'
import App from './App.vue'

// Importar CSS do Bootstrap
import 'bootstrap/dist/css/bootstrap.min.css'
// Bootstrap JS (necessário para tooltips)
import 'bootstrap'

createApp(App).mount('#app')
```

## 3. Exemplo de componente usando tooltips

## App.vue:

vue

```
<template>
  <div class="container mt-5">
    <h1 class="mb-4">Vue 3 + Bootstrap Tooltips</h1>

    <div class="row">
      <!-- Tooltip básico -->
      <div class="col-md-4 mb-3">
        <button
          type="button"
          class="btn btn-primary"
          data-bs-toggle="tooltip"
          data-bs-placement="top"
          :title="tooltipText"
          ref="tooltip1"
        >
          Tooltip no topo
        </button>
      </div>

      <!-- Tooltip com conteúdo dinâmico -->
      <div class="col-md-4 mb-3">
        <button
          type="button"
          class="btn btn-success"
          data-bs-toggle="tooltip"
          data-bs-placement="right"
          :data-bs-title="dynamicTooltip"
          ref="tooltip2"
        >
          Tooltip dinâmico
        </button>
        <div class="mt-2">
          <input
            v-model="dynamicTooltip"
            class="form-control"
            placeholder="Digite o texto do tooltip"
          />
        </div>
      </div>

      <!-- Tooltip customizado -->
      <div class="col-md-4 mb-3">
        <button
```

```
        type="button"
        class="btn btn-warning"
        data-bs-toggle="tooltip"
        data-bs-placement="bottom"
        data-bs-custom-class="custom-tooltip"
        data-bs-title="Este é um tooltip customizado!"
        ref="tooltip3"
    >
    Tooltip customizado
</button>
</div>

<!-- Tooltip em elementos não-button -->
<div class="col-md-4 mb-3">
    <span
        class="badge bg-info"
        data-bs-toggle="tooltip"
        data-bs-placement="left"
        data-bs-title="Tooltip em um span"
        ref="tooltip4"
    >
        Passe o mouse aqui
    </span>
</div>

<!-- Tooltip programático -->
<div class="col-md-4 mb-3">
    <button
        type="button"
        class="btn btn-danger"
        ref="programmaticTooltip"
        @click="toggleTooltip"
    >
        {{ isTooltipVisible ? 'Ocultar' : 'Mostrar' }} Tooltip
    </button>
</div>
</div>
</template>

<script setup>
import { ref, onMounted, onUnmounted, nextTick } from 'vue'
import { Tooltip } from 'bootstrap'

// Refs para os elementos
const tooltip1 = ref(null)
const tooltip2 = ref(null)
```

```
const tooltip3 = ref(null)
const tooltip4 = ref(null)
const programmaticTooltip = ref(null)

// Dados reativos
const tooltipText = ref('Este é um tooltip básico!')
const dynamicTooltip = ref('Texto dinâmico do tooltip')
const isTooltipVisible = ref(false)

// Instâncias dos tooltips
let tooltipInstances = []

// Inicializar tooltips quando o componente montar
onMounted(() => {
    // Inicializar tooltips com data-bs-toggle
    initializeTooltips()

    // Tooltip programático
    const programmaticInstance = new Tooltip(programmaticTooltip.value, {
        title: 'Tooltip controlado programaticamente',
        placement: 'bottom'
    })
    tooltipInstances.push(programmaticInstance)
})

// Limpar tooltips quando o componente desmontar
onUnmounted(() => {
    tooltipInstances.forEach(instance => {
        if (instance && instance.dispose) {
            instance.dispose()
        }
    })
})

// Inicializar todos os tooltips
const initializeTooltips = () => {
    // Usar nextTick para garantir que o DOM esteja renderizado
    nextTick(() => {
        // Selecionar todos os elementos com data-bs-toggle="tooltip"
        const tooltipElements = [tooltip1, tooltip2, tooltip3, tooltip4]

        tooltipElements.forEach(element => {
            if (element.value) {
                const tooltip = new Tooltip(element.value)
                tooltipInstances.push(tooltip)
            }
        })
    })
}
```

```
        })
    }

// Controlar tooltip programaticamente
const toggleTooltip = () => {
  const instance = tooltipInstances.find(
    inst => inst._element === programmaticTooltip.value
  )

  if (instance) {
    if (isTooltipVisible.value) {
      instance.hide()
    } else {
      instance.show()
    }
    isTooltipVisible.value = !isTooltipVisible.value
  }
}

// Atualizar tooltip dinâmico
watch(dynamicTooltip, (newValue) => {
  const instance = tooltipInstances.find(
    inst => inst._element === tooltip2.value
  )

  if (instance) {
    instance.setContent({ '.tooltip-inner': newValue })
  }
})
</script>

<style>
/* Estilos customizados para tooltips */
.custom-tooltip .tooltip-inner {
  background-color: #ff9900;
  color: #000;
  font-weight: bold;
}

.custom-tooltip .tooltip-arrow {
  border-top-color: #ff9900 !important;
}

.container {
  max-width: 900px;
}
```

```

    }
</style>

```

## 4. Usando uma diretiva customizada (opcional)

Crie um arquivo **directives/tooltip.js**:

javascript

```

import { Tooltip } from 'bootstrap'

export default {
  mounted(el, binding) {
    const options = {
      title: binding.value?.title || binding.value,
      placement: binding.value?.placement || 'top',
      trigger: binding.value?.trigger || 'hover focus',
      customClass: binding.value?.customClass || ''
    }

    // Armazenar a instância no elemento
    el._tooltip = new Tooltip(el, options)

    // Atualizar tooltip se o conteúdo mudar
    if (binding.value?.title || binding.value) {
      el.setAttribute('data-bs-title', options.title)
    }
  },
  updated(el, binding) {
    if (el._tooltip) {
      const newTitle = binding.value?.title || binding.value
      el._tooltip.setContent({ '.tooltip-inner': newTitle })
    }
  },
  beforeUnmount(el) {
    if (el._tooltip) {
      el._tooltip.dispose()
      delete el._tooltip
    }
  }
}

```

**main.js** (registrar a diretiva):

javascript

```
import tooltipDirective from './directives/tooltip'

const app = createApp(App)
app.directive('tooltip', tooltipDirective)
app.mount('#app')
```

## Uso da diretiva:

vue

```
<template>
  <button
    v-tooltip="'Tooltip com diretiva'"
    class="btn btn-primary"
  >
    Botão com diretiva
  </button>

  <button
    v-tooltip="{
      title: 'Tooltip customizado',
      placement: 'right',
      customClass: 'custom-tooltip'
    }"
    class="btn btn-secondary"
  >
    Botão com opções
  </button>
</template>
```

## 5. Configuração do Vite (vite.config.js)

javascript

```
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

export default defineConfig({
  plugins: [vue()],
  // Configurações opcionais para melhor performance
  optimizeDeps: {
    include: ['bootstrap', '@popperjs/core']
  }
})
```

## Principais pontos:

1. **Importação:** É necessário importar tanto o CSS quanto o JS do Bootstrap
2. **Inicialização:** Tooltips precisam ser inicializados manualmente no Vue
3. **Gerenciamento de ciclo de vida:** Sempre descarte as instâncias no `onUnmounted`
4. **Conteúdo dinâmico:** Use `setContent()` para atualizar tooltips dinamicamente
5. **Placements:** `top`, `bottom`, `left`, `right`

Este exemplo mostra tanto o uso básico quanto técnicas avançadas como diretivas customizadas e controle programático dos tooltips.

Uncaught ReferenceError: watch is not defined

⌚ Thought for 41 seconds ▾

Vamos corrigir o erro. No exemplo fornecido, usamos `watch` sem importá-lo do Vue. No Vue 3, precisamos importar as funções que usamos, como `watch`.

Além disso, note que o exemplo está usando a sintaxe `<script setup>`, que é uma sintaxe mais recente do Vue 3.

Vamos corrigir o componente `App.vue` importando `watch` e ajustando o código para inicializar os tooltips corretamente.

Aqui está o `App.vue` corrigido:

```
vue

<template>
  <div class="container mt-5">
    <h1 class="mb-4">Vue 3 + Bootstrap Tooltips</h1>

    <div class="row">
      <!-- Tooltip básico -->
      <div class="col-md-4 mb-3">
        <button
          type="button"
          class="btn btn-primary"
          data-bs-toggle="tooltip"
          data-bs-placement="top"
          :title="tooltipText"
          ref="tooltip1"
        >
          Tooltip no topo
        </button>
```

```
</div>

<!-- Tooltip com conteúdo dinâmico -->
<div class="col-md-4 mb-3">
  <button
    type="button"
    class="btn btn-success"
    data-bs-toggle="tooltip"
    data-bs-placement="right"
    :data-bs-title="dynamicTooltip"
    ref="tooltip2"
  >
    Tooltip dinâmico
  </button>
  <div class="mt-2">
    <input
      v-model="dynamicTooltip"
      class="form-control"
      placeholder="Digite o texto do tooltip"
    />
  </div>
</div>

<!-- Tooltip customizado -->
<div class="col-md-4 mb-3">
  <button
    type="button"
    class="btn btn-warning"
    data-bs-toggle="tooltip"
    data-bs-placement="bottom"
    data-bs-custom-class="custom-tooltip"
    data-bs-title="Este é um tooltip customizado!"
    ref="tooltip3"
  >
    Tooltip customizado
  </button>
</div>

<!-- Tooltip em elementos não-button -->
<div class="col-md-4 mb-3">
  <span
    class="badge bg-info"
    data-bs-toggle="tooltip"
    data-bs-placement="left"
    data-bs-title="Tooltip em um span"
    ref="tooltip4"
  >
```

```
Passe o mouse aqui
</span>
</div>

<!-- Tooltip programático --&gt;
&lt;div class="col-md-4 mb-3"&gt;
  &lt;button
    type="button"
    class="btn btn-danger"
    ref="programmaticTooltip"
    @click="toggleTooltip"
  &gt;
    {{ isTooltipVisible ? 'Ocultar' : 'Mostrar' }} Tooltip
  &lt;/button&gt;
&lt;/div&gt;
&lt;/div&gt;
&lt;/div&gt;
&lt;/template&gt;

&lt;script setup&gt;
import { ref, onMounted, onUnmounted, nextTick, watch } from 'vue'
import { Tooltip } from 'bootstrap'

// Refs para os elementos
const tooltip1 = ref(null)
const tooltip2 = ref(null)
const tooltip3 = ref(null)
const tooltip4 = ref(null)
const programmaticTooltip = ref(null)

// Dados reativos
const tooltipText = ref('Este é um tooltip básico!')
const dynamicTooltip = ref('Texto dinâmico do tooltip')
const isTooltipVisible = ref(false)

// Instâncias dos tooltips
let tooltipInstances = []

// Inicializar tooltips quando o componente montar
onMounted(() =&gt; {
  // Inicializar tooltips com data-bs-toggle
  initializeTooltips()

  // Tooltip programático
  const programmaticInstance = new Tooltip(programmaticTooltip.value, {
    title: 'Tooltip controlado programaticamente',
    placement: 'bottom'
  })
})</pre>
```

```
        })
        tooltipInstances.push(programmaticInstance)
    })

// Limpar tooltips quando o componente desmontar
onUnmounted(() => {
    tooltipInstances.forEach(instance => {
        if (instance && instance.dispose) {
            instance.dispose()
        }
    })
})

// Inicializar todos os tooltips
const initializeTooltips = () => {
    // Usar nextTick para garantir que o DOM esteja renderizado
    nextTick(() => {
        // Selecionar todos os elementos com data-bs-toggle="tooltip"
        const tooltipElements = [tooltip1, tooltip2, tooltip3, tooltip4]

        tooltipElements.forEach(element => {
            if (element.value) {
                const tooltip = new Tooltip(element.value)
                tooltipInstances.push(tooltip)
            }
        })
    })
}

// Controlar tooltip programaticamente
const toggleTooltip = () => {
    const instance = tooltipInstances.find(
        inst => inst._element === programmaticTooltip.value
    )

    if (instance) {
        if (isTooltipVisible.value) {
            instance.hide()
        } else {
            instance.show()
        }
        isTooltipVisible.value = !isTooltipVisible.value
    }
}

// Atualizar tooltip dinâmico
watch(dynamicTooltip, (newValue) => {
```

```

const instance = tooltipInstances.find(
  inst => inst._element === tooltip2.value
)

if (instance) {
  instance.setContent({ '.tooltip-inner': newValue })
}
})

</script>

<style>
/* Estilos customizados para tooltips */
.custom-tooltip .tooltip-inner {
  background-color: #ff9900;
  color: #000;
  font-weight: bold;
}

.custom-tooltip .tooltip-arrow {
  border-top-color: #ff9900 !important;
}

.container {
  max-width: 900px;
}
</style>

```

A principal mudança foi a importação da função `watch` do Vue.

Além disso, note que no template, o primeiro botão tinha um texto "Tooltip no topo" (com um erro de digitação) e foi corrigido para "Tooltip no topo".

Com essa correção, o exemplo deve funcionar sem erros no console.

Você precisa importar a função `watch` do Vue. Aqui está a correção:

## App.vue (versão corrigida)

vue

```

<template>
  <div class="container mt-5">
    <h1 class="mb-4">Vue 3 + Bootstrap Tooltips</h1>

    <div class="row">
      <!-- Tooltip básico -->
      <div class="col-md-4 mb-3">

```

```
<button
    type="button"
    class="btn btn-primary"
    data-bs-toggle="tooltip"
    data-bs-placement="top"
    :title="tooltipText"
    ref="tooltip1"
>
    Tooltip no topo
</button>
</div>

<!-- Tooltip com conteúdo dinâmico -->
<div class="col-md-4 mb-3">
    <button
        type="button"
        class="btn btn-success"
        data-bs-toggle="tooltip"
        data-bs-placement="right"
        :data-bs-title="dynamicTooltip"
        ref="tooltip2"
>
    Tooltip dinâmico
</button>
<div class="mt-2">
    <input
        v-model="dynamicTooltip"
        class="form-control"
        placeholder="Digite o texto do tooltip"
        />
</div>
</div>

<!-- Tooltip customizado -->
<div class="col-md-4 mb-3">
    <button
        type="button"
        class="btn btn-warning"
        data-bs-toggle="tooltip"
        data-bs-placement="bottom"
        data-bs-custom-class="custom-tooltip"
        data-bs-title="Este é um tooltip customizado!"
        ref="tooltip3"
>
    Tooltip customizado
</button>
</div>
```

```

<!-- Tooltip em elementos não-button -->
<div class="col-md-4 mb-3">
  <span
    class="badge bg-info"
    data-bs-toggle="tooltip"
    data-bs-placement="left"
    data-bs-title="Tooltip em um span"
    ref="tooltip4"
  >
    Passe o mouse aqui
  </span>
</div>

<!-- Tooltip programático -->
<div class="col-md-4 mb-3">
  <button
    type="button"
    class="btn btn-danger"
    ref="programmaticTooltip"
    @click="toggleTooltip"
  >
    {{ isTooltipVisible ? 'Ocultar' : 'Mostrar' }} Tooltip
  </button>
</div>
</div>
</div>
</template>

<script setup>
// CORREÇÃO: Importar watch do Vue
import { ref, onMounted, onUnmounted, nextTick, watch } from 'vue'
import { Tooltip } from 'bootstrap'

// Refs para os elementos
const tooltip1 = ref(null)
const tooltip2 = ref(null)
const tooltip3 = ref(null)
const tooltip4 = ref(null)
const programmaticTooltip = ref(null)

// Dados reativos
const tooltipText = ref('Este é um tooltip básico!')
const dynamicTooltip = ref('Texto dinâmico do tooltip')
const isTooltipVisible = ref(false)

// Instâncias dos tooltips

```

```
let tooltipInstances = []

// Inicializar tooltips quando o componente montar
onMounted(() => {
    // Inicializar tooltips com data-bs-toggle
    initializeTooltips()

    // Tooltip programático
    const programmaticInstance = new Tooltip(programmaticTooltip.value, {
        title: 'Tooltip controlado programaticamente',
        placement: 'bottom'
    })
    tooltipInstances.push(programmaticInstance)
})

// Limpar tooltips quando o componente desmontar
onUnmounted(() => {
    tooltipInstances.forEach(instance => {
        if (instance && instance.dispose) {
            instance.dispose()
        }
    })
})

// Inicializar todos os tooltips
const initializeTooltips = () => {
    // Usar nextTick para garantir que o DOM esteja renderizado
    nextTick(() => {
        // Selecionar todos os elementos com data-bs-toggle="tooltip"
        const tooltipElements = [tooltip1, tooltip2, tooltip3, tooltip4]

        tooltipElements.forEach(element => {
            if (element.value) {
                const tooltip = new Tooltip(element.value)
                tooltipInstances.push(tooltip)
            }
        })
    })
}

// Controlar tooltip programaticamente
const toggleTooltip = () => {
    const instance = tooltipInstances.find(
        inst => inst._element === programmaticTooltip.value
    )

    if (instance) {
```

```
if (isTooltipVisible.value) {
    instance.hide()
} else {
    instance.show()
}
isTooltipVisible.value = !isTooltipVisible.value
}

// CORREÇÃO: Watch para atualizar tooltip dinâmico
watch(dynamicTooltip, (newValue) => {
    const instance = tooltipInstances.find(
        inst => inst._element === tooltip2.value
    )

    if (instance) {
        // Método correto para atualizar conteúdo no Bootstrap 5
        instance.setContent({ '.tooltip-inner': newValue })

        // Se o tooltip estiver visível, atualizar imediatamente
        if (instance.tip && instance.tip.classList.contains('show')) {
            instance.update()
        }
    }
})
</script>

<style>
/* Estilos customizados para tooltips */
.custom-tooltip .tooltip-inner {
    background-color: #ff9900;
    color: #000;
    font-weight: bold;
}

.custom-tooltip .tooltip-arrow {
    border-top-color: #ff9900 !important;
}

.container {
    max-width: 900px;
}
</style>
```

## Alternativa mais simples (sem watch)

Se preferir uma abordagem mais simples, você pode usar uma diretiva `v-bind` para atualizar automaticamente:

vue

```
<template>
  <!-- ... resto do template ... -->

  <div class="col-md-4 mb-3">
    <button
      type="button"
      class="btn btn-success"
      data-bs-toggle="tooltip"
      data-bs-placement="right"
      :data-bs-title="dynamicTooltip"
      ref="tooltip2"
      @mouseenter="updateTooltipTitle"
    >
      Tooltip dinâmico
    </button>
    <div class="mt-2">
      <input
        v-model="dynamicTooltip"
        class="form-control"
        placeholder="Digite o texto do tooltip"
        @input="updateTooltipTitle"
      />
    </div>
  </div>
</template>

<script setup>
import { ref, onMounted, onUnmounted, nextTick } from 'vue'
import { Tooltip } from 'bootstrap'

// ... resto do código ...

// Método para atualizar o título do tooltip
const updateTooltipTitle = () => {
  if (tooltip2.value) {
    tooltip2.value.setAttribute('data-bs-title', dynamicTooltip.value)

    // Encontrar e atualizar a instância do tooltip
    const instance = tooltipInstances.find(
      inst => inst._element === tooltip2.value
    )
  }
}
```

```
if (instance) {  
    instance.setContent({ '.tooltip-inner': dynamicTooltip.value })  
}  
}  
}  
</script>
```

## Versão simplificada (recomendada para iniciantes)

vue

```
<template>  
  <div class="container mt-5">  
    <h1 class="mb-4">Vue 3 + Bootstrap Tooltips - Simplificado</h1>  
  
    <div class="d-flex gap-3">  
      <!-- Tooltip básico (funciona automaticamente) -->  
      <button  
        type="button"  
        class="btn btn-primary"  
        data-bs-toggle="tooltip"  
        data-bs-placement="top"  
        title="Tooltip básico"  
      >  
        Tooltip básico  
      </button>  
  
      <!-- Tooltip com conteúdo Vue reativo -->  
      <button  
        type="button"  
        class="btn btn-success"  
        data-bs-toggle="tooltip"  
        data-bs-placement="right"  
        :title="dynamicTitle"  
      >  
        Tooltip reativo  
      </button>  
  
      <!-- Tooltip inicializado manualmente -->  
      <button  
        type="button"  
        class="btn btn-info"  
        ref="manualTooltip"  
      >  
        Tooltip manual  
      </button>
```

```

</div>

<div class="mt-3">
  <input
    v-model="dynamicTitle"
    class="form-control w-50"
    placeholder="Digite o texto do tooltip"
  />
</div>
</div>
</template>

<script setup>
import { ref, onMounted, onUnmounted } from 'vue'
import { Tooltip } from 'bootstrap'

const dynamicTitle = ref('Texto dinâmico')
const manualTooltip = ref(null)

let tooltipInstance = null

onMounted(() => {
  // Inicializar tooltip manual
  if (manualTooltip.value) {
    tooltipInstance = new Tooltip(manualTooltip.value, {
      title: 'Tooltip inicializado manualmente',
      placement: 'bottom'
    })
  }
})

onUnmounted(() => {
  // Limpar tooltip
  if (tooltipInstance) {
    tooltipInstance.dispose()
  }
})
</script>

```

## Principais pontos corrigidos:

1. Importação do `watch`: Adicionei `watch` na importação do Vue
2. Sintaxe correta: Usei a sintaxe correta para a função `watch`

### 3. Atualização do tooltip: Usei setContent() corretamente para atualizar o conteúdo

A primeira versão corrigida está completa e funcionará corretamente. A versão simplificada é mais fácil de entender se você está começando com Vue 3 e Bootstrap.

exemplo vue 3 vite bootstrap modal

#### 🕒 Thought for 79 seconds ▾

Vamos criar um exemplo de modal do Bootstrap em Vue 3 com Vite.

Passos:

1. Instalar Bootstrap e @popperjs/core (já que o Bootstrap 5 depende do Popper para posicionamento).
2. Importar o CSS do Bootstrap no main.js.
3. Importar o JavaScript do Bootstrap no main.js (ou inicializar os componentes necessários onde for usado).
4. Criar um componente de modal.

No entanto, note que o Bootstrap Modais são componentes que são ativados via JavaScript. Em Vue, podemos controlar a exibição do modal através de uma referência e a instância do modal.

Vamos criar um exemplo que:

- Mostra um botão para abrir o modal.
- O modal terá um título, corpo e rodapé com botões.
- Vamos controlar o modal de forma programática, usando a API do Bootstrap Modal.

Também mostrarei como usar o modal com a abordagem de data-bs-\* (HTML) e como controlar com Vue.

Vamos começar com a abordagem programática, que é mais adequada para Vue.

#### 1. Instalação

bash

```
npm install bootstrap @popperjs/core
```

#### 2. Configurar Bootstrap

main.js

```
javascript
```

```
import { createApp } from 'vue'
import App from './App.vue'
import 'bootstrap/dist/css/bootstrap.min.css'
// Importar o JavaScript do Bootstrap. Nota: estamos importando tudo. Você pode
// otimizar importando apenas o modal se quiser.
import 'bootstrap'

createApp(App).mount('#app')
```

### 3. Criar um componente de modal controlado por Vue

#### App.vue

```
vue
```

```
<template>
  <div class="container mt-5">
    <h1>Vue 3 + Bootstrap Modal</h1>

    <!-- Botão para abrir o modal -->
    <button type="button" class="btn btn-primary mt-3" @click="openModal">
      Abrir Modal
    </button>

    <!-- Modal -->
    <div ref="modalElement" class="modal fade" tabindex="-1" aria-hidden="true">
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title">{{ title }}</h5>
            <button type="button" class="btn-close" @click="closeModal" aria-label="Close"></button>
          </div>
          <div class="modal-body">
            <p>{{ bodyText }}</p>
            <div class="mb-3">
              <label for="message" class="form-label">Mensagem:</label>
              <input v-model="message" type="text" class="form-control" id="message" placeholder="Digite algo">
            </div>
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-secondary" @click="closeModal">Fchar</button>
            <button type="button" class="btn btn-primary" @click="saveChanges">S
```

```
alvar mudanças</button>
      </div>
    </div>
  </div>
</div>
</template>

<script setup>
import { ref, onMounted, onUnmounted } from 'vue'
import { Modal } from 'bootstrap'

// Refs
const modalElement = ref(null)
const title = ref('Título do Modal')
const bodyText = ref('Este é o corpo do modal. Você pode adicionar qualquer conteúdo aqui.')
const message = ref('')

// Instância do modal
let modal = null

// Inicializar o modal quando o componente montar
onMounted(() => {
  modal = new Modal(modalElement.value)
})

// Abrir modal
const openModal = () => {
  if (modal) {
    modal.show()
  }
}

// Fechar modal
const closeModal = () => {
  if (modal) {
    modal.hide()
  }
}

// Salvar mudanças (exemplo de ação)
const saveChanges = () => {
  alert('Mensagem salva: ' + message.value)
  closeModal()
}
```

```
// Limpar a instância do modal quando o componente for desmontado
onUnmounted(() => {
  if (modal) {
    modal.dispose()
  }
})
</script>

<style scoped>
.container {
  max-width: 800px;
}
</style>
```

#### 4. Explicação

- Usamos a ref `modalElement` para obter a referência do elemento DOM do modal.
- No `onMounted`, criamos uma nova instância do Modal do Bootstrap passando o elemento.
- As funções `openModal` e `closeModal` chamam os métodos `show()` e `hide()` da instância do modal, respectivamente.
- No `onUnmounted`, descartamos a instância do modal para evitar vazamentos de memória.
- O modal é controlado completamente por Vue, então podemos usar data binding (`v-model`) no input e métodos para lidar com as ações.

#### 5. Alternativa: Usando o modal com data-bs-\* (sem controle programático)

Se você quiser usar o modal com os atributos `data-bs-*` do Bootstrap, pode fazer assim:

```
vue

<template>
  <div class="container mt-5">
    <h1>Modal com data-bs-*</h1>

    <!-- Botão para acionar o modal -->
    <button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#exampleModal">
      Abrir Modal com data-bs-*
    </button>

    <!-- Modal -->
    <div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModalLabel" aria-hidden="true">
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title" id="exampleModalLabel">Modal com data-bs-*</h5>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>
```

```
h5>
    <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
-label="Close"></button>
</div>
<div class="modal-body">
    <p>Este modal é aberto e fechado usando os atributos data-bs-toggle
e data-bs-dismiss.</p>
</div>
<div class="modal-footer">
    <button type="button" class="btn btn-secondary" data-bs-dismiss="mod
al">Fechar</button>
    <button type="button" class="btn btn-primary">Salvar mudanças</butto
n>
</div>
</div>
</div>
</div>
</template>

<script setup>
// Não é necessário código JavaScript para abrir/fechar, pois o Bootstrap cuida
disso via data attributes.
</script>
```

No entanto, note que nesta abordagem, o modal é controlado pelo Bootstrap através dos data attributes. Isso pode ser mais simples, mas oferece menos controle programático.

## 6. Usando um componente de modal reutilizável

Para um projeto maior, você pode criar um componente de modal reutilizável. Aqui está um exemplo básico:

## components/Modal.vue

```
vue

<template>
  <div ref="modalElement" class="modal fade" tabindex="-1" :aria-labelledby="labelledby" aria-hidden="true">
    <div class="modal-dialog" :class="modalDialogClass">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title" :id="labelledby">{{ title }}</h5>
          <button type="button" class="btn-close" @click="close" aria-label="Close"></button>
        </div>
        <div class="modal-body">
```

```
<slot name="body"></slot>
</div>
<div class="modal-footer">
  <slot name="footer">
    <button type="button" class="btn btn-secondary" @click="close">Fecha
    r</button>
    <button type="button" class="btn btn-primary" @click="save">Salvar</
    button>
  </slot>
</div>
</div>
</div>
</div>
</template>

<script setup>
import { ref, onMounted, onUnmounted } from 'vue'
import { Modal } from 'bootstrap'

const props = defineProps({
  title: {
    type: String,
    default: 'Modal'
  },
  labelledby: {
    type: String,
    default: 'modalLabel'
  },
  modalDialogClass: {
    type: String,
    default: ''
  }
})

const emit = defineEmits(['close', 'save'])

const modalElement = ref(null)
let modal = null

onMounted(() => {
  modal = new Modal(modalElement.value)
})

const open = () => {
  if (modal) {
    modal.show()
  }
}
```

```

        }

        const close = () => {
            if (modal) {
                modal.hide()
            }
            emit('close')
        }

        const save = () => {
            emit('save')
        }

        onUnmounted(() => {
            if (modal) {
                modal.dispose()
            }
        })

        // Exportar métodos para que o componente pai possa abrir/fechar programaticamente
        defineExpose({
            open,
            close
        })
    </script>

```

## App.vue usando o componente Modal

```

vue

<template>
    <div class="container mt-5">
        <h1>Modal Reutilizável</h1>

        <button type="button" class="btn btn-primary" @click="openModal">Abrir Modal
        </button>

        <Modal ref="modalRef" title="Modal Reutilizável" @close="onClose" @save="onS
        ave">
            <template #body>
                <p>Conteúdo do modal passado via slot.</p>
                <div class="mb-3">
                    <label for="input" class="form-label">Campo de exemplo:</label>
                    <input v-model="inputValue" type="text" class="form-control" id="inpu
                    t">
                </div>
            </template>
        </Modal>
    </div>
</template>

```

```
<template #footer>
    <button type="button" class="btn btn-secondary" @click="closeModal">Cancelar</button>
    <button type="button" class="btn btn-primary" @click="saveModal">Salvar</button>
</template>
</Modal>
</div>
</template>

<script setup>
import { ref } from 'vue'
import Modal from './components/Modal.vue'

const modalRef = ref(null)
const inputValue = ref('')

const openModal = () => {
    if (modalRef.value) {
        modalRef.value.open()
    }
}

const closeModal = () => {
    if (modalRef.value) {
        modalRef.value.close()
    }
}

const saveModal = () => {
    alert('Valor salvo: ' + inputValue.value)
    closeModal()
}

const onClose = () => {
    console.log('Modal fechado')
}

const onSave = () => {
    console.log('Salvando...')
}
</script>
```

## Conclusão

Temos três abordagens:

1. Modal controlado programaticamente com a instância do Bootstrap Modal.

2. Modal usando os data attributes do Bootstrap (mais simples, menos controle).

3. Componente de modal reutilizável com slots e métodos expostos.

Escolha a que melhor se adapta ao seu projeto. A abordagem programática é mais poderosa e integrada com o Vue.

Vou mostrar um exemplo completo de como usar Bootstrap modals em Vue 3 com Vite. Aqui estão várias abordagens:

## 1. Instalação

bash

```
npm install bootstrap @popperjs/core
```

## 2. Configuração básica

**main.js:**

javascript

```
import { createApp } from 'vue'
import App from './App.vue'

// Importar Bootstrap
import 'bootstrap/dist/css/bootstrap.min.css'
import 'bootstrap'

createApp(App).mount('#app')
```

## 3. Modal Simples (usando data-bs-\*)

**App.vue:**

vue

```
<template>
  <div class="container mt-5">
    <h1 class="mb-4">Vue 3 + Bootstrap Modals</h1>

    <!-- Modal simples com data attributes -->
    <div class="mb-3">
      <h3>1. Modal com data-bs-*</h3>
```

```
<button
    type="button"
    class="btn btn-primary"
    data-bs-toggle="modal"
    data-bs-target="#simpleModal"
>
    Abrir Modal Simples
</button>
</div>

<!-- Modal Simples -->
<div class="modal fade" id="simpleModal" tabindex="-1" aria-labelledby="simpleModalLabel" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="simpleModalLabel">Modal Simples</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body">
                <p>Este é um modal usando apenas data-bs-* attributes.</p>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Fechar</button>
                <button type="button" class="btn btn-primary">Salvar</button>
            </div>
        </div>
    </div>
</div>

<!-- Modal controlado por Vue -->
<div class="mb-3">
    <h3>2. Modal Controlado por Vue</h3>
    <button
        type="button"
        class="btn btn-success"
        @click="openModal('vueModal')"
>
    Abrir Modal Vue
</button>
</div>

<!-- Modal Vue -->
<div
    class="modal fade"
```

```
    id="vueModal"
    tabindex="-1"
    aria-hidden="true"
    ref="vueModalRef"
  >
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">{{ modalTitle }}</h5>
        <button
          type="button"
          class="btn-close"
          @click="closeModal('vueModal')"
          aria-label="Close"
        ></button>
      </div>
      <div class="modal-body">
        <div class="mb-3">
          <label for="nome" class="form-label">Nome</label>
          <input
            type="text"
            class="form-control"
            id="nome"
            v-model="formData.nome"
            placeholder="Digite seu nome"
          >
        </div>
        <div class="mb-3">
          <label for="email" class="form-label">Email</label>
          <input
            type="email"
            class="form-control"
            id="email"
            v-model="formData.email"
            placeholder="Digite seu email"
          >
        </div>
        <p v-if="modalMessage" class="text-success">{{ modalMessage }}</p>
      </div>
      <div class="modal-footer">
        <button
          type="button"
          class="btn btn-secondary"
          @click="closeModal('vueModal')"
        >
          Cancelar
        </button>
      </div>
    </div>
  </div>
```

```
<button
    type="button"
    class="btn btn-primary"
    @click="submitForm"
>
    Salvar
</button>
</div>
</div>
</div>

<!-- Modal Dinâmico -->
<div class="mb-3">
    <h3>3. Modal Dinâmico</h3>
    <button
        type="button"
        class="btn btn-warning"
        @click="showDynamicModal = true"
    >
        Abrir Modal Dinâmico
    </button>
</div>

<!-- Modal Dinâmico com v-if -->
<div
    v-if="showDynamicModal"
    class="modal fade show d-block"
    tabindex="-1"
    style="background-color: rgba(0,0,0,0.5)"
    @click.self="closeDynamicModal"
>
    <div class="modal-dialog modal-dialog-centered">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Modal Dinâmico</h5>
                <button
                    type="button"
                    class="btn-close"
                    @click="closeDynamicModal"
                ></button>
            </div>
            <div class="modal-body">
                <p>Conteúdo dinâmico do modal. Status: <strong>{{ dynamicStatus }}</strong></p>
                <button
                    class="btn btn-sm btn-info"
                >
                    ...
                </button>
            </div>
        </div>
    </div>
</div>
```

```
        @click="toggleStatus"
      >
        Alternar Status
      </button>
    </div>
    <div class="modal-footer">
      <button
        type="button"
        class="btn btn-secondary"
        @click="closeDynamicModal"
      >
        Fechar
      </button>
    </div>
  </div>
</div>

<!-- Modal com slots (Componente reutilizável) -->
<div class="mb-3">
  <h3>4. Modal Componente Reutilizável</h3>
  <button
    type="button"
    class="btn btn-info"
    @click="openReusableModal"
  >
    Abrir Modal Reutilizável
  </button>
</div>

<!-- Componente Modal Reutilizável -->
<ReusableModal
  v-model:show="showReusableModal"
  :title="reusableModalTitle"
  @confirm="handleConfirm"
  @cancel="handleCancel"
>
  <template #default>
    <p>Conteúdo personalizado usando slots.</p>
    <div class="alert alert-info">
      Este conteúdo é passado através do slot default.
    </div>
  </template>

  <template #footer>
    <button type="button" class="btn btn-outline-secondary" @click="handleCancel"
  </>
```

```
        Cancelar
    </button>
    <button type="button" class="btn btn-danger" @click="handleConfirm">
        Confirmar Ação
    </button>
</template>
</ReusableModal>
</div>
</template>

<script setup>
import { ref, onMounted, onUnmounted } from 'vue'
import { Modal } from 'bootstrap'
import ReusableModal from './components/ReusableModal.vue'

// Refs para modais
const vueModalRef = ref(null)
const showDynamicModal = ref(false)
const showReusableModal = ref(false)

// Dados do formulário
const modalTitle = ref('Formulário Vue Modal')
const formData = ref({
    nome: '',
    email: ''
})
const modalMessage = ref('')
const dynamicStatus = ref('Ativo')
const reusableModalTitle = ref('Modal Reutilizável')

// Instância do modal Bootstrap
let vueModalInstance = null

// Inicializar modal quando componente montar
onMounted(() => {
    if (vueModalRef.value) {
        vueModalInstance = new Modal(vueModalRef.value, {
            backdrop: 'static',
            keyboard: false
        })
    }
})

// Limpar quando desmontar
onUnmounted(() => {
    if (vueModalInstance) {
        vueModalInstance.dispose()
    }
})
```

```
        }
    })

// Métodos para modal Vue
const openModal = (modalId) => {
    if (modalId === 'vueModal' && vueModalInstance) {
        modalMessage.value = ''
        vueModalInstance.show()
    }
}

const closeModal = (modalId) => {
    if (modalId === 'vueModal' && vueModalInstance) {
        vueModalInstance.hide()
    }
}

const submitForm = () => {
    if (!formData.value.nome || !formData.value.email) {
        modalMessage.value = 'Por favor, preencha todos os campos!'
        return
    }

    modalMessage.value = `Dados salvos: ${formData.value.nome} (${formData.value.emai
l})`
}

// Simular API call
setTimeout(() => {
    closeModal('vueModal')
    formData.value = { nome: '', email: '' }
}, 1500)
}

// Métodos para modal dinâmico
const closeDynamicModal = () => {
    showDynamicModal.value = false
}

const toggleStatus = () => {
    dynamicStatus.value = dynamicStatus.value === 'Ativo' ? 'Inativo' : 'Ativo'
}

// Métodos para modal reutilizável
const openReusableModal = () => {
    showReusableModal.value = true
}
```

```

const handleConfirm = () => {
  alert('Ação confirmada!')
  showReusableModal.value = false
}

const handleCancel = () => {
  showReusableModal.value = false
}
</script>

<style scoped>
.modal-backdrop {
  opacity: 0.5;
}

.container {
  max-width: 1200px;
}

.modal-content {
  border-radius: 10px;
}

.modal-header {
  background-color: #f8f9fa;
  border-radius: 10px 10px 0 0;
}
</style>

```

## 4. Componente Modal Reutilizável

**components/ReusableModal.vue:**

```

vue

<template>
  <transition name="modal">
    <div
      v-if="modelValue"
      class="modal-overlay"
      @click.self="closeModal"
    >
      <div
        class="modal-container"
        :class="sizeClass"
      >
    
```

```
<div class="modal-content">
  <!-- Cabeçalho -->
  <div class="modal-header">
    <h5 class="modal-title">
      <slot name="title">
        {{ title }}
      </slot>
    </h5>
    <button
      type="button"
      class="btn-close"
      @click="closeModal"
      aria-label="Close"
    ></button>
  </div>

  <!-- Corpo -->
  <div class="modal-body">
    <slot></slot>
  </div>

  <!-- Rodapé -->
  <div class="modal-footer">
    <slot name="footer">
      <button
        type="button"
        class="btn btn-secondary"
        @click="cancel"
      >
        Cancelar
      </button>
      <button
        type="button"
        class="btn btn-primary"
        @click="confirm"
      >
        Confirmar
      </button>
    </slot>
  </div>
</div>
</div>
</transition>
</template>

<script setup>
```

```
import { computed } from 'vue'

const props = defineProps({
  modelValue: {
    type: Boolean,
    required: true
  },
  title: {
    type: String,
    default: 'Modal'
  },
  size: {
    type: String,
    default: 'md',
    validator: (value) => ['sm', 'md', 'lg', 'xl'].includes(value)
  },
  closeOnBackdrop: {
    type: Boolean,
    default: true
  }
})

const emit = defineEmits([
  'update:modelValue',
  'confirm',
  'cancel'
])

const sizeClass = computed(() => {
  return {
    'sm': 'modal-sm',
    'md': '',
    'lg': 'modal-lg',
    'xl': 'modal-xl'
  }[props.size]
})

const closeModal = () => {
  if (props.closeOnBackdrop) {
    emit('update:modelValue', false)
  }
}

const confirm = () => {
  emit('confirm')
  emit('update:modelValue', false)
}
```

```
const cancel = () => {
  emit('cancel')
  emit('update:modelValue', false)
}

// Fechar com ESC
const handleKeydown = (e) => {
  if (e.key === 'Escape' && props.modelValue) {
    closeModal()
  }
}

// Adicionar/remover event listener
if (typeof window !== 'undefined') {
  window.addEventListener('keydown', handleKeydown)
}
</script>

<style scoped>
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 1050;
  padding: 20px;
}

.modal-container {
  background: white;
  border-radius: 8px;
  box-shadow: 0 4px 20px rgba(0, 0, 0, 0.15);
  width: 100%;
  max-height: 90vh;
  overflow: hidden;
  display: flex;
  flex-direction: column;
}

.modal-container.modal-sm {
  max-width: 400px;
```

```
}
```

```
.modal-container {  
    max-width: 500px;  
}
```

```
.modal-container.modal-lg {  
    max-width: 800px;  
}
```

```
.modal-container.modal-xl {  
    max-width: 1140px;  
}
```

```
.modal-content {  
    display: flex;  
    flex-direction: column;  
    height: 100%;  
}
```

```
.modal-header {  
    padding: 1rem 1.5rem;  
    border-bottom: 1px solid #dee2e6;  
    display: flex;  
    align-items: center;  
    justify-content: space-between;  
}
```

```
.modal-title {  
    margin: 0;  
    font-size: 1.25rem;  
}
```

```
.modal-body {  
    padding: 1.5rem;  
    flex: 1;  
    overflow-y: auto;  
}
```

```
.modal-footer {  
    padding: 1rem 1.5rem;  
    border-top: 1px solid #dee2e6;  
    display: flex;  
    justify-content: flex-end;  
    gap: 0.5rem;  
}
```

```

/* Transições */
.modal-enter-active,
.modal-leave-active {
  transition: opacity 0.3s ease;
}

.modal-enter-from,
.modal-leave-to {
  opacity: 0;
}

.modal-enter-active .modal-container,
.modal-leave-active .modal-container {
  transition: transform 0.3s ease;
}

.modal-enter-from .modal-container,
.modal-leave-to .modal-container {
  transform: scale(0.95);
}
</style>

```

## 5. Modal com Formulário (Exemplo Avançado)

**components/FormModal.vue:**

vue

```

<template>
  <div
    v-if="visible"
    class="modal-backdrop fade show"
    @click.self="onBackdropClick"
  >
    <div class="modal fade show d-block" tabindex="-1">
      <div class="modal-dialog" :class="modalSize">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title">{{ title }}</h5>
            <button
              type="button"
              class="btn-close"
              @click="close"
              aria-label="Close"
            ></button>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>
<script>
  ...
</script>
<style>
  ...
</style>

```

```
<form @submit.prevent="submit">
  <div class="modal-body">
    <slot :form-data="formData" :errors="errors"></slot>
  </div>

  <div class="modal-footer">
    <slot name="footer" :close="close" :submit="submit">
      <button
        type="button"
        class="btn btn-secondary"
        @click="close"
      >
        Cancelar
      </button>
      <button
        type="submit"
        class="btn btn-primary"
        :disabled="loading"
      >
        <span v-if="loading" class="spinner-border spinner-border-sm me-1"></span>
        {{ submitText }}
      </button>
    </slot>
  </div>
</form>
</div>
</div>
</div>
</template>

<script setup>
import { ref, watch } from 'vue'

const props = defineProps({
  visible: {
    type: Boolean,
    required: true
  },
  title: {
    type: String,
    default: 'Formulário'
  },
  size: {
    type: String,
  }
})
```

```
default: '',
validator: (value) => ['', 'modal-sm', 'modal-lg', 'modal-xl'].includes(value),
},
submitText: {
  type: String,
  default: 'Salvar'
},
closeOnBackdrop: {
  type: Boolean,
  default: true
},
initialData: {
  type: Object,
  default: () => ({})
}
})

const emit = defineEmits([
  'update:visible',
  'submit',
  'close'
])

const loading = ref(false)
const formData = ref({ ...props.initialData })
const errors = ref({})

const modalSize = props.size

watch(() => props.visible, (newVal) => {
  if (newVal) {
    formData.value = { ...props.initialData }
    errors.value = {}
    document.body.style.overflow = 'hidden'
  } else {
    document.body.style.overflow = ''
  }
})

const close = () => {
  emit('update:visible', false)
  emit('close')
}

const onBackdropClick = () => {
  if (props.closeOnBackdrop) {
    close()
  }
}
```

```
        }
    }

const submit = async () => {
    loading.value = true
    errors.value = {}

    try {
        await emit('submit', formData.value)
        close()
    } catch (error) {
        if (error.response?.data?.errors) {
            errors.value = error.response.data.errors
        }
    } finally {
        loading.value = false
    }
}

// Fechar com ESC
const handleEscape = (e) => {
    if (e.key === 'Escape' && props.visible) {
        close()
    }
}

// Adicionar event listeners
if (typeof window !== 'undefined') {
    window.addEventListener('keydown', handleEscape)
}
</script>

<style scoped>
.modal-backdrop {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: rgba(0, 0, 0, 0.5);
    z-index: 1040;
}

.modal {
    z-index: 1050;
}
```

```
.modal-content {
    animation: modalShow 0.3s ease-out;
}

@keyframes modalShow {
    from {
        opacity: 0;
        transform: translateY(-50px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}
</style>
```

## 6. Uso do FormModal

vue

```
<template>
    <button @click="showFormModal = true" class="btn btn-primary">
        Abrir Form Modal
    </button>

    <FormModal
        v-model:visible="showFormModal"
        title="Cadastro de Usuário"
        :initial-data="userData"
        @submit="handleSubmit"
    >
        <template #default="{ formData, errors }">
            <div class="mb-3">
                <label class="form-label">Nome</label>
                <input
                    v-model="formData.name"
                    type="text"
                    class="form-control"
                    :class="{'is-invalid': errors.name }"
                >
                <div v-if="errors.name" class="invalid-feedback">
                    {{ errors.name[0] }}
                </div>
            </div>
            <div class="mb-3">
```

```

<label class="form-label">Email</label>
<input
    v-model="formData.email"
    type="email"
    class="form-control"
    :class="{ 'is-invalid': errors.email }"
>
<div v-if="errors.email" class="invalid-feedback">
    {{ errors.email[0] }}
</div>
</div>
</template>
</FormModal>
</template>

<script setup>
import { ref } from 'vue'
import FormModal from './components/FormModal.vue'

const showFormModal = ref(false)
const userData = ref({
    name: '',
    email: ''
})

const handleSubmit = async (formData) => {
    // Simular API call
    console.log('Dados enviados:', formData)

    // Simular erro
    // throw new Error('Erro de validação')
}
</script>

```

## Principais pontos:

### 1. Três abordagens:

- Modal com data-bs-\* (simples, menos controle)
- Modal controlado por Vue (com instância Bootstrap)
- Modal com v-if (totalmente controlado por Vue)

### 2. Componente reutilizável: Crie modais genéricos com slots

### 3. Eventos: Trate eventos como submit, close, cancel

#### 4. Transições: Adicione animações com CSS transitions

#### 5. Acessibilidade: Use atributos ARIA e suporte a teclado (ESC)

Escolha a abordagem que melhor se adapta às suas necessidades!

hook.js:608 [Vue warn]: Missing required prop: "modelValue"

🕒 Thought for 34 seconds ▾

O erro "Missing required prop: 'modelValue'" indica que o componente ReusableModal está esperando uma prop chamada `modelValue` que é obrigatória, mas não foi fornecida.

No exemplo fornecido, o componente ReusableModal foi definido com uma prop `modelValue` obrigatória, e estamos usando `v-model:show` para passar o valor. No entanto, em Vue 3, a diretiva `v-model` sem argumento usa `modelValue` como prop padrão. Mas no nosso caso, usamos `v-model:show`, o que significa que a prop esperada é `show`, não `modelValue`.

Portanto, há uma inconsistência: o componente define uma prop obrigatória chamada `modelValue`, mas o uso do componente está tentando passar uma prop chamada `show`.

Para corrigir, temos duas opções:

1. Alterar a definição do componente ReusableModal para usar a prop `show` em vez de `modelValue`.
2. Ou alterar o uso do componente para usar `v-model` sem argumento (que usaria `modelValue`).

Vou ajustar o exemplo para a primeira opção, pois no template usamos `v-model:show`. Vamos modificar o componente ReusableModal para esperar uma prop `show` e emitir `update:show`.

Aqui está a correção:

**components/ReusableModal.vue (corrigido):**

```
vue

<template>
  <transition name="modal">
    <div
      v-if="show"
      class="modal-overlay"
      @click.self="closeModal"
    >
      <div
        class="modal-container"
        :class="sizeClass"
      >
        <div class="modal-content">
```

```
<!-- Cabeçalho -->
<div class="modal-header">
  <h5 class="modal-title">
    <slot name="title">
      {{ title }}
    </slot>
  </h5>
  <button
    type="button"
    class="btn-close"
    @click="closeModal"
    aria-label="Close"
  ></button>
</div>

<!-- Corpo -->
<div class="modal-body">
  <slot></slot>
</div>

<!-- Rodapé -->
<div class="modal-footer">
  <slot name="footer">
    <button
      type="button"
      class="btn btn-secondary"
      @click="cancel"
    >
      Cancelar
    </button>
    <button
      type="button"
      class="btn btn-primary"
      @click="confirm"
    >
      Confirmar
    </button>
  </slot>
</div>
</div>
</div>
</transition>
</template>

<script setup>
import { computed } from 'vue'
```

```
const props = defineProps({
    show: { // Alterado de modelValue para show
        type: Boolean,
        required: true
    },
    title: {
        type: String,
        default: 'Modal'
    },
    size: {
        type: String,
        default: 'md',
        validator: (value) => ['sm', 'md', 'lg', 'xl'].includes(value)
    },
    closeOnBackdrop: {
        type: Boolean,
        default: true
    }
})

// Emitir eventos para atualizar a prop show
const emit = defineEmits([
    'update:show', // Alterado para update:show
    'confirm',
    'cancel'
])

const sizeClass = computed(() => {
    return {
        'sm': 'modal-sm',
        'md': '',
        'lg': 'modal-lg',
        'xl': 'modal-xl'
    }[props.size]
})

const closeModal = () => {
    if (props.closeOnBackdrop) {
        emit('update:show', false) // Alterado para update:show
    }
}

const confirm = () => {
    emit('confirm')
    emit('update:show', false) // Alterado para update:show
}
```

```

const cancel = () => {
  emit('cancel')
  emit('update:show', false) // Alterado para update:show
}

// Fechar com ESC
const handleKeydown = (e) => {
  if (e.key === 'Escape' && props.show) { // Alterado para props.show
    closeModal()
  }
}

// Adicionar/remover event listener
if (typeof window !== 'undefined') {
  window.addEventListener('keydown', handleKeydown)
}
</script>

<style scoped>
/* ... (estilos permanecem iguais) ... */
</style>

```

No componente pai (App.vue), o uso do componente ReusableModal já está correto, pois estamos usando `v-model:show`.

Se preferir a segunda opção (usar `modelValue`), você teria que alterar o uso do componente em App.vue:

```

vue

<ReusableModal
  v-model="showReusableModal"
  :title="reusableModalTitle"
  @confirm="handleConfirm"
  @cancel="handleCancel"
>
  ...
</ReusableModal>

```

E manter a prop como `modelValue` no componente ReusableModal.

Escolhi a primeira opção porque no exemplo original foi usado `v-model:show`, o que é uma sintaxe para `v-model` com argumento. Ajustando o componente para usar a prop `show` e emitir `update:show` resolve o problema.

Portanto, faça a alteração no componente `ReusableModal` conforme acima e o erro deve ser resolvido.

O erro ocorre porque o componente `ReusableModal` está esperando uma prop chamada `modelValue` (padrão do `v-model`) mas você está passando como `show`. Vou corrigir o exemplo:

## App.vue (corrigido)

vue

```
<template>
  <div class="container mt-5">
    <h1 class="mb-4">Vue 3 + Bootstrap Modals</h1>

    <!-- Modal simples com data attributes -->
    <div class="mb-3">
      <h3>1. Modal com data-bs-*</h3>
      <button
        type="button"
        class="btn btn-primary"
        data-bs-toggle="modal"
        data-bs-target="#simpleModal"
      >
        Abrir Modal Simples
      </button>
    </div>

    <!-- Modal Simples -->
    <div class="modal fade" id="simpleModal" tabindex="-1" aria-labelledby="simpleModalLabel" aria-hidden="true">
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title" id="simpleModalLabel">Modal Simples</h5>
            <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
          </div>
          <div class="modal-body">
            <p>Este é um modal usando apenas data-bs-* attributes.</p>
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Fechar</button>
            <button type="button" class="btn btn-primary">Salvar</button>
          </div>
        </div>
      </div>
    </div>
```

```
</div>
</div>
</div>

<!-- Modal controlado por Vue -->
<div class="mb-3">
  <h3>2. Modal Controlado por Vue</h3>
  <button
    type="button"
    class="btn btn-success"
    @click="openModal('vueModal')"
  >
    Abrir Modal Vue
  </button>
</div>

<!-- Modal Vue -->
<div
  class="modal fade"
  id="vueModal"
  tabindex="-1"
  aria-hidden="true"
  ref="vueModalRef"
>
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">{{ modalTitle }}</h5>
        <button
          type="button"
          class="btn-close"
          @click="closeModal('vueModal')"
          aria-label="Close"
        ></button>
      </div>
      <div class="modal-body">
        <div class="mb-3">
          <label for="nome" class="form-label">Nome</label>
          <input
            type="text"
            class="form-control"
            id="nome"
            v-model="formData.nome"
            placeholder="Digite seu nome"
          >
        </div>
        <div class="mb-3">

```

```
<label for="email" class="form-label">Email</label>
<input
    type="email"
    class="form-control"
    id="email"
    v-model="formData.email"
    placeholder="Digite seu email"
    >
</div>
<p v-if="modalMessage" class="text-success">{{ modalMessage }}</p>
</div>
<div class="modal-footer">
    <button
        type="button"
        class="btn btn-secondary"
        @click="closeModal('vueModal')"
        >
            Cancelar
    </button>
    <button
        type="button"
        class="btn btn-primary"
        @click="submitForm"
        >
            Salvar
    </button>
</div>
</div>

<!-- Modal Dinâmico -->
<div class="mb-3">
    <h3>3. Modal Dinâmico</h3>
    <button
        type="button"
        class="btn btn-warning"
        @click="showDynamicModal = true"
        >
            Abrir Modal Dinâmico
    </button>
</div>

<!-- Modal Dinâmico com v-if -->
<div
    v-if="showDynamicModal"
    class="modal fade show d-block"
    >
```

```
tabindex="-1"
style="background-color: rgba(0,0,0,0.5)"
@click.self="closeDynamicModal"
>
<div class="modal-dialog modal-dialog-centered">
  <div class="modal-content">
    <div class="modal-header">
      <h5 class="modal-title">Modal Dinâmico</h5>
      <button
        type="button"
        class="btn-close"
        @click="closeDynamicModal"
      ></button>
    </div>
    <div class="modal-body">
      <p>Conteúdo dinâmico do modal. Status: <strong>{{ dynamicStatus }}</strong></p>
      <button
        class="btn btn-sm btn-info"
        @click="toggleStatus"
      >
        Alternar Status
      </button>
    </div>
    <div class="modal-footer">
      <button
        type="button"
        class="btn btn-secondary"
        @click="closeDynamicModal"
      >
        Fechar
      </button>
    </div>
  </div>
</div>

<!-- Modal com slots (Componente reutilizável) -->
<div class="mb-3">
  <h3>4. Modal Componente Reutilizável</h3>
  <button
    type="button"
    class="btn btn-info"
    @click="showReusableModal = true"
  >
    Abrir Modal Reutilizável
  </button>
```

```
</div>

<!-- Componente Modal Reutilizável - CORRIGIDO -->
<ReusableModal
  v-model="showReusableModal"
  :title="reusableModalTitle"
  @confirm="handleConfirm"
  @cancel="handleCancel"
>
  <template #default>
    <p>Conteúdo personalizado usando slots.</p>
    <div class="alert alert-info">
      Este conteúdo é passado através do slot default.
    </div>
  </template>

  <template #footer>
    <button type="button" class="btn btn-outline-secondary" @click="handleCancel">
      Cancelar
    </button>
    <button type="button" class="btn btn-danger" @click="handleConfirm">
      Confirmar Ação
    </button>
  </template>
</ReusableModal>
</div>
</template>

<script setup>
import { ref, onMounted, onUnmounted } from 'vue'
import { Modal } from 'bootstrap'
import ReusableModal from './components/ReusableModal.vue'

// Refs para modais
const vueModalRef = ref(null)
const showDynamicModal = ref(false)
const showReusableModal = ref(false)

// Dados do formulário
const modalTitle = ref('Formulário Vue Modal')
const formData = ref({
  nome: '',
  email: ''
})
const modalMessage = ref('')
const dynamicStatus = ref('Ativo')
```

```
const reusableModalTitle = ref('Modal Reutilizável')

// Instância do modal Bootstrap
let vueModalInstance = null

// Inicializar modal quando componente montar
onMounted(() => {
  if (vueModalRef.value) {
    vueModalInstance = new Modal(vueModalRef.value, {
      backdrop: 'static',
      keyboard: false
    })
  }
})

// Limpar quando desmontar
onUnmounted(() => {
  if (vueModalInstance) {
    vueModalInstance.dispose()
  }
})

// Métodos para modal Vue
const openModal = (modalId) => {
  if (modalId === 'vueModal' && vueModalInstance) {
    modalMessage.value = ''
    vueModalInstance.show()
  }
}

const closeModal = (modalId) => {
  if (modalId === 'vueModal' && vueModalInstance) {
    vueModalInstance.hide()
  }
}

const submitForm = () => {
  if (!formData.value.nome || !formData.value.email) {
    modalMessage.value = 'Por favor, preencha todos os campos!'
    return
  }

  modalMessage.value = `Dados salvos: ${formData.value.nome} (${formData.value.emai
l})`

  // Simular API call
  setTimeout(() => {
    // Lógica de envio para API
  }, 1000)
}

// Exemplo de uso
<button @click="openModal('vueModal')">Abrir Modal</button>
```

```
closeModal('vueModal')
formData.value = { nome: '', email: '' }
}, 1500)
}

// Métodos para modal dinâmico
const closeDynamicModal = () => {
  showDynamicModal.value = false
}

const toggleStatus = () => {
  dynamicStatus.value = dynamicStatus.value === 'Ativo' ? 'Inativo' : 'Ativo'
}

// Métodos para modal reutilizável
const handleConfirm = () => {
  alert('Ação confirmada!')
  showReusableModal.value = false
}

const handleCancel = () => {
  showReusableModal.value = false
}
</script>

<style scoped>
.modal-backdrop {
  opacity: 0.5;
}

.container {
  max-width: 1200px;
}

.modal-content {
  border-radius: 10px;
}

.modal-header {
  background-color: #f8f9fa;
  border-radius: 10px 10px 0 0;
}
</style>
```

## components/ReusableModal.vue (corrigido)

vue

```
<template>
  <transition name="modal">
    <div
      v-if="modelValue"
      class="modal-overlay"
      @click.self="closeModal"
    >
      <div
        class="modal-container"
        :class="sizeClass"
      >
        <div class="modal-content">
          <!-- Cabeçalho -->
          <div class="modal-header">
            <h5 class="modal-title">
              <slot name="title">
                {{ title }}
              </slot>
            </h5>
            <button
              type="button"
              class="btn-close"
              @click="closeModal"
              aria-label="Close"
            ></button>
          </div>

          <!-- Corpo -->
          <div class="modal-body">
            <slot></slot>
          </div>

          <!-- Rodapé -->
          <div class="modal-footer">
            <slot name="footer">
              <button
                type="button"
                class="btn btn-secondary"
                @click="cancel"
              >
                Cancelar
              </button>
              <button
                type="button"
                class="btn btn-primary"
              >
                <slot name="confirm">
                  OK
                </slot>
              </button>
            </slot>
          </div>
        </div>
      </div>
    </div>
  </transition>
</template>
```

```
@click="confirm"
>
    Confirmar
    </button>
</slot>
</div>
</div>
</div>
</transition>
</template>

<script setup>
import { computed } from 'vue'

const props = defineProps({
  modelValue: { // CORRIGIDO: usando modelValue (padrão do v-model)
    type: Boolean,
    required: true
  },
  title: {
    type: String,
    default: 'Modal'
  },
  size: {
    type: String,
    default: 'md',
    validator: (value) => ['sm', 'md', 'lg', 'xl'].includes(value)
  },
  closeOnBackdrop: {
    type: Boolean,
    default: true
  }
})

const emit = defineEmits([
  'update:modelValue', // CORRIGIDO: emit padrão do v-model
  'confirm',
  'cancel'
])

const sizeClass = computed(() => {
  return {
    'sm': 'modal-sm',
    'md': '',
    'lg': 'modal-lg',
    'xl': 'modal-xl'
  }
})
```

```
}[props.size]
})

const closeModal = () => {
  if (props.closeOnBackdrop) {
    emit('update:modelValue', false)
  }
}

const confirm = () => {
  emit('confirm')
  emit('update:modelValue', false)
}

const cancel = () => {
  emit('cancel')
  emit('update:modelValue', false)
}

// Fechar com ESC
const handleKeydown = (e) => {
  if (e.key === 'Escape' && props.modelValue) {
    closeModal()
  }
}

// Adicionar/remover event listener
if (typeof window !== 'undefined') {
  window.addEventListener('keydown', handleKeydown)
}
</script>

<style scoped>
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 1050;
  padding: 20px;
}
}
```

```
.modal-container {  
    background: white;  
    border-radius: 8px;  
    box-shadow: 0 4px 20px rgba(0, 0, 0, 0.15);  
    width: 100%;  
    max-height: 90vh;  
    overflow: hidden;  
    display: flex;  
    flex-direction: column;  
}  
  
.modal-container.modal-sm {  
    max-width: 400px;  
}  
  
.modal-container {  
    max-width: 500px;  
}  
  
.modal-container.modal-lg {  
    max-width: 800px;  
}  
  
.modal-container.modal-xl {  
    max-width: 1140px;  
}  
  
.modal-content {  
    display: flex;  
    flex-direction: column;  
    height: 100%;  
}  
  
.modal-header {  
    padding: 1rem 1.5rem;  
    border-bottom: 1px solid #dee2e6;  
    display: flex;  
    align-items: center;  
    justify-content: space-between;  
}  
  
.modal-title {  
    margin: 0;  
    font-size: 1.25rem;  
}  
  
.modal-body {
```

```

padding: 1.5rem;
flex: 1;
overflow-y: auto;
}

.modal-footer {
  padding: 1rem 1.5rem;
  border-top: 1px solid #dee2e6;
  display: flex;
  justify-content: flex-end;
  gap: 0.5rem;
}

/* Transições */
.modal-enter-active,
.modal-leave-active {
  transition: opacity 0.3s ease;
}

.modal-enter-from,
.modal-leave-to {
  opacity: 0;
}

.modal-enter-active .modal-container,
.modal-leave-active .modal-container {
  transition: transform 0.3s ease;
}

.modal-enter-from .modal-container,
.modal-leave-to .modal-container {
  transform: scale(0.95);
}
</style>
```

## Alternativa: Versão mais simples (sem v-model)

Se preferir uma abordagem mais simples:

**components/SimpleModal.vue:**

vue

```
<template>
  <div v-if="isOpen" class="modal-overlay" @click.self="onOverlayClick">
    <div class="modal-content" :class="size">
```

```
<div class="modal-header">
  <h5 class="modal-title">{{ title }}</h5>
  <button type="button" class="btn-close" @click="close"></button>
</div>
<div class="modal-body">
  <slot></slot>
</div>
<div class="modal-footer">
  <slot name="footer" :close="close">
    <button type="button" class="btn btn-secondary" @click="close">
      Fechar
    </button>
  </slot>
</div>
</div>
</div>

<script setup>
defineProps({
  isOpen: {
    type: Boolean,
    required: true
  },
  title: {
    type: String,
    default: 'Modal'
  },
  size: {
    type: String,
    default: 'md',
    validator: (value) => ['sm', 'md', 'lg', 'xl'].includes(value)
  },
  closeOnOverlay: {
    type: Boolean,
    default: true
  }
})
const emit = defineEmits(['close'])

const close = () => {
  emit('close')
}

const onOverlayClick = () => {
  if (closeOnOverlay) {
```

```
        close()
    }
}

// Fechar com ESC
window.addEventListener('keydown', (e) => {
    if (e.key === 'Escape' && isOpen) {
        close()
    }
})
</script>
```

```
<style scoped>
.modal-overlay {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background: rgba(0, 0, 0, 0.5);
    display: flex;
    align-items: center;
    justify-content: center;
    z-index: 1050;
    padding: 20px;
}

.modal-content {
    background: white;
    border-radius: 8px;
    max-height: 90vh;
    overflow-y: auto;
    display: flex;
    flex-direction: column;
}

.modal-content.sm {
    max-width: 400px;
    width: 100%;
}

.modal-content.md {
    max-width: 500px;
    width: 100%;
}

.modal-content.lg {
```

```

    max-width: 800px;
    width: 100%;

}

.modal-content.xl {
    max-width: 1140px;
    width: 100%;
}

.modal-header {
    padding: 1rem 1.5rem;
    border-bottom: 1px solid #dee2e6;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

.modal-body {
    padding: 1.5rem;
    flex: 1;
}

.modal-footer {
    padding: 1rem 1.5rem;
    border-top: 1px solid #dee2e6;
    display: flex;
    justify-content: flex-end;
    gap: 0.5rem;
}

```

</style>

## Uso do SimpleModal:

vue

```

<template>
    <button @click="isModalOpen = true" class="btn btn-primary">
        Abrir Modal Simples
    </button>

    <SimpleModal
        :is-open="isModalOpen"
        title="Modal Simples"
        @close="isModalOpen = false"
    >
        <p>Conteúdo do modal</p>

```

```

<template #footer="{ close }">
  <button @click="close" class="btn btn-secondary">Cancelar</button>
  <button @click="save" class="btn btn-primary">Salvar</button>
</template>
</SimpleModal>
</template>

<script setup>
import { ref } from 'vue'
import SimpleModal from './components/SimpleModal.vue'

const isModalOpen = ref(false)

const save = () => {
  console.log('Salvando...')
  isModalOpen.value = false
}
</script>

```

## Principais correções:

1. **Prop** `modelValue` : No Vue 3 Composition API com `<script setup>`, o v-model padrão usa `modelValue` como prop e `update:modelValue` como evento

2. **Sintaxe correta:**

- Use `v-model="showReusableModal"` (sem `:show`)
- Ou use  
`:model-value="showReusableModal" @update:model-value="showReusableModal = $event"`

3. **Emit correto:** O evento deve ser `update:modelValue` para funcionar com v-model

A versão corrigida acima deve funcionar sem erros. Escolha a abordagem que preferir:

- **Com v-model:** Mais idiomático no Vue 3
- **Com prop/event explícitos:** Mais explícito e fácil de entender para iniciantes

regular diminuir a opacidade

 Thought for 7 seconds ▾

Vamos criar um exemplo onde podemos diminuir a opacidade de forma controlada. Existem várias maneiras de fazer isso, mas vou criar um exemplo que utiliza uma variável reativa para controlar a opacidade de um elemento, e botões para aumentar e diminuir a opacidade.

Além disso, vou incluir um exemplo de como aplicar essa opacidade em um elemento e como usar transições para suavizar a mudança.

Vou criar um componente Vue 3 com Composition API e `<script setup>`.

Passos:

1. Criar uma referência reativa para a opacidade (inicialmente 1).
2. Criar métodos para aumentar e diminuir a opacidade (com limites entre 0 e 1).
3. Aplicar a opacidade a um elemento (por exemplo, uma div) via binding de estilo.
4. Adicionar botões para controlar a opacidade.
5. Adicionar uma transição CSS para suavizar as mudanças.

Vamos ao código:

Vou criar um exemplo completo de como diminuir/controlar a opacidade em Vue 3 com Vite, usando diferentes abordagens:

## 1. Controle de Opacidade Simples

**App.vue:**

```
vue

<template>
  <div class="container mt-5">
    <h1 class="mb-4">Controle de Opacidade em Vue 3</h1>

    <div class="row">
      <!-- Exemplo 1: Opacidade com v-model -->
      <div class="col-md-6 mb-5">
        <h3>1. Controle de Opacidade</h3>

        <div class="card">
          <div class="card-body">
            <div class="mb-3">
              <label class="form-label">Opacidade: {{ opacityValue }}%</label>
              <input
                type="range"
                class="form-range"
                min="0"
                max="100"
                v-model.number="opacityValue">
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>

<script setup>
  import { ref } from 'vue'

  const opacityValue = ref(100)
</script>
```

```
>
</div>

<div
  class="opacity-demo-box p-4 rounded"
  :style="{ opacity: opacityValue / 100 }"
>
  <p class="mb-2">Caixa com opacidade controlada</p>
  <small>Opacity: {{ (opacityValue / 100).toFixed(2) }}</small>
</div>

<div class="mt-3 d-flex gap-2">
  <button
    v-for="preset in opacityPresets"
    :key="preset"
    class="btn btn-sm btn-outline-secondary"
    @click="opacityValue = preset"
  >
    {{ preset }}%
  </button>
</div>
</div>
</div>

<!-- Exemplo 2: Fade In/Out -->
<div class="col-md-6 mb-5">
  <h3>2. Fade In/Out</h3>

  <div class="card">
    <div class="card-body">
      <button
        class="btn btn-primary mb-3"
        @click="toggleFade"
      >
        {{ isVisible ? 'Fade Out' : 'Fade In' }}
      </button>

      <transition name="fade">
        <div v-if="isVisible" class="alert alert-success">
          Elemento com transição de opacidade
        </div>
      </transition>

      <transition name="fade">
        <div v-if="isVisible" class="mt-3 p-3 bg-light rounded">
          <p>Outro elemento com fade</p>
        </div>
      </transition>
    </div>
  </div>
</div>
```

```
<p class="mb-0">Transição suave de opacidade</p>
</div>
</transition>
</div>
</div>
</div>

<!-- Exemplo 3: Overlay com opacidade -->
<div class="col-md-6 mb-5">
  <h3>3. Overlay/Loading</h3>

  <div class="card">
    <div class="card-body">
      <button
        class="btn btn-warning mb-3"
        @click="showOverlay = !showOverlay"
      >
        {{ showOverlay ? 'Ocultar' : 'Mostrar' }} Overlay
      </button>

      <div
        class="position-relative border rounded p-4"
        style="height: 200px;"
      >
        <div class="content">
          <p>Conteúdo principal da área</p>
          <button class="btn btn-sm btn-outline-primary">
            Botão normal
          </button>
        </div>

        <!-- Overlay -->
        <div
          v-if="showOverlay"
          class="overlay"
          :style="{ opacity: overlayOpacity / 100 }"
        >
          <div class="overlay-content">
            <div class="spinner-border text-primary" role="status"></div>
            <p class="mt-2">Carregando...</p>
          </div>
        </div>
      </div>

      <div class="mt-3">
        <label class="form-label">Opacidade do Overlay: {{ overlayOpacity }}%
      </label>
    </div>
  </div>
</div>
```

```
<input
    type="range"
    class="form-range"
    min="10"
    max="90"
    v-model.number="overlayOpacity"
>
</div>
</div>
</div>
</div>

<!-- Exemplo 4: Galeria com hover -->
<div class="col-md-6 mb-5">
    <h3>4. Efeitos Hover</h3>

    <div class="card">
        <div class="card-body">
            <div class="row g-3">
                <div
                    v-for="item in galleryItems"
                    :key="item.id"
                    class="col-6"
                    @mouseenter="item.active = true"
                    @mouseleave="item.active = false"
                >
                    <div class="gallery-item position-relative">
                        
                        <div
                            class="gallery-overlay"
                            :class="{'active': item.active }"
                        >
                            <h6 class="text-white mb-1">{{ item.title }}</h6>
                            <p class="text-white small mb-0">{{ item.description }}</p>
                        </div>
                    </div>
                </div>
            </div>
        <div class="mt-3">
            <div class="form-check form-switch">
                <input
```

```
        class="form-check-input"
        type="checkbox"
        v-model="enableHover"
        id="hoverSwitch"
      >
    <label class="form-check-label" for="hoverSwitch">
      Ativar efeitos hover
    </label>
  </div>
</div>
</div>
</div>
</div>
</div>

<!-- Exemplo 5: Opacidade condicional -->
<div class="col-md-12 mb-5">
  <h3>5. Opacidade Condisional</h3>

  <div class="card">
    <div class="card-body">
      <div class="row align-items-center mb-4">
        <div class="col-md-4">
          <div class="mb-3">
            <label class="form-label">Nível de prioridade:</label>
            <select v-model="priorityLevel" class="form-select">
              <option value="high">Alta</option>
              <option value="medium">Média</option>
              <option value="low">Baixa</option>
            </select>
          </div>
        </div>
      </div>
      <div class="col-md-8">
        <div class="priority-list">
          <div
            v-for="task in tasks"
            :key="task.id"
            class="priority-item p-3 mb-2 rounded"
            :class="[
              `priority-${task.priority}`,
              { 'current-priority': task.priority === priorityLevel }
            ]"
            :style="{
              opacity: getTaskOpacity(task.priority)
            }"
          >
            <div class="d-flex justify-content-between align-items-center">
```

```
<span>{{ task.name }}</span>
<span class="badge" :class="`bg-${getPriorityColor(task.priority)}-text`">
    {{ task.priority === 'high' ? 'Alta' : task.priority === 'medium' ? 'Média' : 'Baixa' }}
</span>
</div>
</div>
</div>
</div>
</div>

<div class="alert alert-info">
    <p class="mb-0">Tarefas com prioridade diferente da selecionada têm opacidade reduzida.</p>
    </div>
    </div>
</div>
</div>
</div>
</div>
</div>
</template>

<script setup>
import { ref, computed } from 'vue'

// Exemplo 1
const opacityValue = ref(70)
const opacityPresets = [20, 40, 60, 80, 100]

// Exemplo 2
const isVisible = ref(true)
const toggleFade = () => {
    isVisible.value = !isVisible.value
}

// Exemplo 3
const showOverlay = ref(false)
const overlayOpacity = ref(70)

// Exemplo 4
const enableHover = ref(true)
const galleryItems = ref([
{
    id: 1,
    title: 'Imagem 1',
    description: 'Descrição da imagem 1',
    thumbnailUrl: 'https://via.placeholder.com/150x100'
}
])
```

```
        image: 'https://picsum.photos/300/200?random=1',
        active: false
    },
    {
        id: 2,
        title: 'Imagen 2',
        description: 'Descrição da imagem 2',
        image: 'https://picsum.photos/300/200?random=2',
        active: false
    },
    {
        id: 3,
        title: 'Imagen 3',
        description: 'Descrição da imagem 3',
        image: 'https://picsum.photos/300/200?random=3',
        active: false
    },
    {
        id: 4,
        title: 'Imagen 4',
        description: 'Descrição da imagem 4',
        image: 'https://picsum.photos/300/200?random=4',
        active: false
    }
])
])

// Exemplo 5
const priorityLevel = ref('high')
const tasks = ref([
    { id: 1, name: 'Tarefa Urgente', priority: 'high' },
    { id: 2, name: 'Reunião com equipe', priority: 'high' },
    { id: 3, name: 'Documentação do projeto', priority: 'medium' },
    { id: 4, name: 'Atualizar dependências', priority: 'medium' },
    { id: 5, name: 'Organizar arquivos', priority: 'low' },
    { id: 6, name: 'Ler artigos', priority: 'low' }
])

const getTaskOpacity = (taskPriority) => {
    if (priorityLevel.value === taskPriority) return 1
    if (priorityLevel.value === 'high' && taskPriority === 'medium') return 0.6
    if (priorityLevel.value === 'high' && taskPriority === 'low') return 0.3
    if (priorityLevel.value === 'medium' && taskPriority === 'high') return 0.8
    if (priorityLevel.value === 'medium' && taskPriority === 'low') return 0.4
    if (priorityLevel.value === 'low' && taskPriority === 'high') return 0.7
    if (priorityLevel.value === 'low' && taskPriority === 'medium') return 0.5
    return 1
}
```

```
const getPriorityColor = (priority) => {
  return {
    high: 'danger',
    medium: 'warning',
    low: 'success'
  }[priority]
}
</script>

<style scoped>
/* Estilos para o Exemplo 1 */
.opacity-demo-box {
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  transition: opacity 0.3s ease;
  height: 150px;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

/* Estilos para o Exemplo 2 */
.fade-enter-active,
.fade-leave-active {
  transition: opacity 0.5s ease;
}

.fade-enter-from,
.fade-leave-to {
  opacity: 0;
}

/* Estilos para o Exemplo 3 */
.position-relative {
  position: relative;
}

.overlay {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: rgba(0, 0, 0, 0.7);
  display: flex;
```

```
justify-content: center;
align-items: center;
transition: opacity 0.3s ease;
}

.overlay-content {
  text-align: center;
  color: white;
}

.content {
  z-index: 1;
  position: relative;
}

/* Estilos para o Exemplo 4 */
.gallery-item {
  cursor: pointer;
  overflow: hidden;
  border-radius: 8px;
}

.gallery-item img {
  transition: all 0.3s ease;
  width: 100%;
  height: 150px;
  object-fit: cover;
}

.gallery-item:hover img {
  transform: scale(1.05);
}

.img-hover {
  opacity: 0.7;
}

.gallery-overlay {
  position: absolute;
  bottom: 0;
  left: 0;
  right: 0;
  background: rgba(0, 0, 0, 0.7);
  color: white;
  padding: 10px;
  transform: translateY(100%);
  transition: transform 0.3s ease, opacity 0.3s ease;
```

```
    opacity: 0;
}

.gallery-overlay.active {
    transform: translateY(0);
    opacity: 1;
}

/* Estilos para o Exemplo 5 */
.priority-list {
    min-height: 200px;
}

.priority-item {
    transition: all 0.3s ease;
    cursor: pointer;
}

.priority-item:hover {
    transform: translateX(5px);
}

.priority-high {
    background-color: rgba(220, 53, 69, 0.1);
    border-left: 4px solid #dc3545;
}

.priority-medium {
    background-color: rgba(255, 193, 7, 0.1);
    border-left: 4px solid #ffc107;
}

.priority-low {
    background-color: rgba(25, 135, 84, 0.1);
    border-left: 4px solid #198754;
}

.current-priority {
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
}

/* Estilos gerais */
.card {
    transition: transform 0.2s;
}

.card:hover {
```

```

        transform: translateY(-2px);
    }

.form-range::-webkit-slider-thumb {
    background-color: #0d6efd;
}

.form-range::-moz-range-thumb {
    background-color: #0d6efd;
}
</style>
```

## 2. Diretiva Customizada para Opacidade

`directives/opacity.js:`

`javascript`

```

export default {
    mounted(el, binding) {
        updateOpacity(el, binding.value)
    },
    updated(el, binding) {
        updateOpacity(el, binding.value)
    }
}

function updateOpacity(el, value) {
    // Se value for um objeto, pode ter mais configurações
    if (typeof value === 'object') {
        const { opacity, duration = 300, easing = 'ease' } = value

        el.style.transition = `opacity ${duration}ms ${easing}`
        el.style.opacity = opacity

        // Se tiver callback após a transição
        if (value.onComplete) {
            el.addEventListener('transitionend', () => value.onComplete(), { once: true })
        }
    } else {
        // Se for apenas número ou string
        el.style.opacity = value
    }
}
```

## main.js (para registrar a diretiva):

```
javascript

import { createApp } from 'vue'
import App from './App.vue'
import opacityDirective from './directives/opacity'

// Importar Bootstrap
import 'bootstrap/dist/css/bootstrap.min.css'
import 'bootstrap'

const app = createApp(App)
app.directive('opacity', opacityDirective)
app.mount('#app')
```

## Uso da diretiva:

vue

```
<template>
<div>
    <!-- Uso simples -->
    <div v-opacity="opacityValue">
        Conteúdo com opacidade
    </div>

    <!-- Uso com opções -->
    <div v-opacity="{ opacity: 0.5, duration: 500 }">
        Com transição suave
    </div>

    <!-- Uso dinâmico -->
    <div v-opacity="isActive ? 1 : 0.3">
        Opacidade condicional
    </div>
</div>
</template>
```

## 3. Composables para Opacidade (Vue 3 Composition API)

### composables/useOpacity.js:

javascript

```
import { ref, computed, watch } from 'vue'

export function useOpacity(initialValue = 1) {
  const opacity = ref(initialValue)

  const opacityStyle = computed(() => ({
    opacity: opacity.value,
    transition: 'opacity 0.3s ease'
  }))

  const opacityPercent = computed(() => Math.round(opacity.value * 100))

  const setOpacity = (value) => {
    // Garantir que o valor esteja entre 0 e 1
    opacity.value = Math.max(0, Math.min(1, value))
  }

  const increaseOpacity = (step = 0.1) => {
    setOpacity(opacity.value + step)
  }

  const decreaseOpacity = (step = 0.1) => {
    setOpacity(opacity.value - step)
  }

  const fadeIn = (duration = 300) => {
    opacity.value = 0
    setTimeout(() => {
      opacity.value = 1
    }, 10)
  }

  const fadeOut = (duration = 300) => {
    opacity.value = 0
  }

  // Watch para ações após mudança
  watch(opacity, (newValue, oldValue) => {
    console.log(`Opacidade alterada de ${oldValue} para ${newValue}`)
  })

  return {
    opacity,
    opacityStyle,
    opacityPercent,
    setOpacity,
```

```
increaseOpacity,  
decreaseOpacity,  
fadeIn,  
fadeOut  
}  
}
```

## Uso do composable:

vue

```
<template>  
  <div class="container">  
    <h2>Composable de Opacidade</h2>  
  
    <div class="mb-3">  
      <label class="form-label">Controle: {{ opacityPercent }}%</label>  
      <input  
        type="range"  
        min="0"  
        max="100"  
        v-model.number="opacityPercent"  
        class="form-range"  
      >  
    </div>  
  
    <div  
      class="demo-box p-4 rounded"  
      :style="opacityStyle"  
    >  
      Caixa com opacidade controlada  
    </div>  
  
    <div class="mt-3 d-flex gap-2">  
      <button @click="increaseOpacity(0.2)" class="btn btn-sm btn-primary">  
        + Opacidade  
      </button>  
      <button @click="decreaseOpacity(0.2)" class="btn btn-sm btn-secondary">  
        - Opacidade  
      </button>  
      <button @click="fadeIn" class="btn btn-sm btn-success">  
        Fade In  
      </button>  
      <button @click="fadeOut" class="btn btn-sm btn-danger">  
        Fade Out  
      </button>  
    </div>  
  </div>
```

```

</div>
</template>

<script setup>
import { watch } from 'vue'
import { useOpacity } from '../composables/useOpacity'

const {
  opacity,
  opacityStyle,
  opacityPercent,
  setOpacity,
  increaseOpacity,
  decreaseOpacity,
  fadeIn,
  fadeOut
} = useOpacity(0.7)

// Watch para sincronizar o range com a opacidade
watch(opacityPercent, (newValue) => {
  setOpacity(newValue / 100)
})
</script>

```

## 4. Componente de Transição Customizado

**components/FadeTransition.vue:**

```

vue

<template>
  <transition
    @before-enter="beforeEnter"
    @enter="enter"
    @leave="leave"
    :css="false"
  >
    <slot v-if="show"></slot>
  </transition>
</template>

<script setup>
import { ref, defineProps, defineEmits } from 'vue'

const props = defineProps({
  show: {

```

```
        type: Boolean,
        required: true
    },
    duration: {
        type: Number,
        default: 300
    },
    initialOpacity: {
        type: Number,
        default: 0
    },
    finalOpacity: {
        type: Number,
        default: 1
    }
})

const emit = defineEmits(['transition-start', 'transition-end'])

const beforeEnter = (el) => {
    el.style.opacity = props.initialOpacity
    emit('transition-start')
}

const enter = (el, done) => {
    let start = null
    const animate = (timestamp) => {
        if (!start) start = timestamp
        const progress = timestamp - start
        const percentage = Math.min(progress / props.duration, 1)

        el.style.opacity = props.initialOpacity +
            (props.finalOpacity - props.initialOpacity) * percentage

        if (progress < props.duration) {
            requestAnimationFrame(animate)
        } else {
            done()
            emit('transition-end')
        }
    }
    requestAnimationFrame(animate)
}

const leave = (el, done) => {
    let start = null
```

```
const animate = (timestamp) => {
  if (!start) start = timestamp
  const progress = timestamp - start
  const percentage = Math.min(progress / props.duration, 1)

  el.style.opacity = props.finalOpacity -
    (props.finalOpacity - props.initialOpacity) * percentage

  if (progress < props.duration) {
    requestAnimationFrame(animate)
  } else {
    done()
    emit('transition-end')
  }
}

requestAnimationFrame(animate)
}
</script>
```