UniSENAI

# Trilha 02

*Conhecendo o Vue.js*

# 2. Conhecendo o Vue.js

### 2.1 Configurando o jsFiddle para Testes Rápidos

O **jsFiddle** é uma ferramenta online que permite testar, compartilhar e depurar códigos de maneira rápida e eficiente. Para quem está começando com Vue.js, o jsFiddle pode ser uma excelente opção para experimentação e aprendizado, pois não exige configurações complexas de ambiente. Nesta seção, aprenderemos como configurar o jsFiddle para utilizar o Vue.js, além de explorarmos exemplos práticos.

---

## O que é o jsFiddle?

O **jsFiddle** é uma plataforma baseada na web que oferece um ambiente para escrever e executar códigos de **HTML**, **CSS** e **JavaScript** diretamente no navegador. Ele é amplamente usado para:

- Testar snippets de código pequenos.
- Compartilhar exemplos com outros desenvolvedores.
- Depurar problemas rapidamente.

Ao trabalhar com Vue.js, o jsFiddle é útil para testar interações simples e aprender a usar os conceitos básicos do framework.

---

## Por que usar o jsFiddle com Vue.js?

1. **Configuração Rápida**: Não é necessário instalar nada localmente. Basta acessar o site e começar a codar.
2. **Aprendizado Modular**: Permite focar em trechos específicos do código sem se preocupar com estrutura de arquivos.
3. **Compartilhamento Simples**: Gera links para compartilhar os exemplos criados com colegas ou professores.
4. **Ambiente Limpo**: Ideal para isolar problemas e testar soluções sem interferências externas.

---

## Passo a Passo para Configurar o jsFiddle com Vue.js

### Passo 1: Acesse o jsFiddle

1. Acesse o site oficial do jsFiddle: https://jsfiddle.net.
2. Você verá uma interface com quatro seções principais:
    - **HTML**: Para o código HTML.
    - **CSS**: Para o estilo.
    - **JavaScript**: Para o código JavaScript.

- **Resultado (Output)**: Para visualizar a saída.

---

**Passo 2: Inclua a Biblioteca Vue.js**

Para usar o Vue.js no jsFiddle, você precisa adicionar a biblioteca ao ambiente:

1. Clique no menu "Settings" (Configurações) no canto superior direito da seção **JavaScript**.
2. No campo **External Resources** (Recursos Externos), insira o link para a versão do Vue.js que deseja usar:
   - Para Vue 3:

```
https://unpkg.com/vue@3
```

   - Para Vue 2:

```
https://cdn.jsdelivr.net/npm/vue@2
```

3. Clique em **OK** para salvar as configurações.

---

**Passo 3: Estruture o Código**

Agora, você pode começar a escrever o código. No jsFiddle, o Vue.js funciona com base em um elemento de raiz no HTML e um script em JavaScript para inicializar a aplicação.

**Exemplo básico de configuração no jsFiddle:**

- **HTML:**

```
<div id="app">
  <h1>{{ mensagem }}</h1>
  <button @click="mudarMensagem">Clique aqui</button>
</div>
```

- **JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      mensagem: 'Olá, Vue.js no jsFiddle!'
    };
  },
  methods: {
    mudarMensagem() {
      this.mensagem = 'Você clicou no botão!';
    }
  }
});

app.mount('#app');
```

- **CSS (opcional)**

```
h1 {
  color: blue;
  font-family: Arial, sans-serif;
}
button {
  padding: 10px 20px;
  font-size: 16px;
  cursor: pointer;
}
```

- **Resultado:**
  - Quando você clica no botão, a mensagem exibida na tela muda.

## Exemplo Prático 1: Contador Simples

Vamos criar um contador que aumenta e diminui o valor ao clicar em botões.

- **HTML:**

```
<div id="app">
  <h1>Contador: {{ contador }}</h1>
  <button @click="incrementar">+</button>
  <button @click="decrementar">-</button>
</div>
```

- **JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      contador: 0
    };
  },
  methods: {
    incrementar() {
      this.contador++;
    },
    decrementar() {
      this.contador--;
    }
  }
});

app.mount('#app');
```

**Explicação:**

- O `data()` contém o estado inicial do contador (0).
- Os métodos `incrementar` e `decrementar` atualizam o estado.
- O Vue atualiza automaticamente o DOM sempre que o valor de `contador` é alterado.

---

## Exemplo Prático 2: Lista Dinâmica

Este exemplo mostra como criar uma lista dinâmica, onde você pode adicionar e remover itens.

- **HTML:**

```
<div id="app">
  <h1>Minha Lista</h1>
  <ul>
    <li v-for="(item, index) in lista" :key="index">
      {{ item }}
      <button @click="removerItem(index)">Remover</button>
    </li>
  </ul>
  <input v-model="novoItem" placeholder="Adicionar item">
  <button @click="adicionarItem">Adicionar</button>
</div>
```

- **JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      lista: ['Item 1', 'Item 2', 'Item 3'],
      novoItem: ''
    };
  },
  methods: {
    adicionarItem() {
      if (this.novoItem.trim() !== '') {
        this.lista.push(this.novoItem);
        this.novoItem = '';
      }
    },
    removerItem(index) {
      this.lista.splice(index, 1);
    }
  }
});

app.mount('#app');
```

**Explicação:**

- A diretiva `v-for` renderiza os itens da lista dinamicamente.
- O método `adicionarItem` adiciona um novo item à lista.
- O método `removerItem` remove um item da lista pelo índice.

## Melhores Práticas ao Usar o jsFiddle com Vue.js

1. **Organize seu Código**: Separe o HTML, CSS e JavaScript para manter o código legível.
2. **Teste Incrementalmente**: Construa e teste seu código em pequenas etapas para identificar erros rapidamente.
3. **Use Recursos Externos**: Além do Vue.js, você pode adicionar outras bibliotecas, como Axios para requisições HTTP, ou Bootstrap para estilização.
4. **Salve e Compartilhe**: Após criar seu exemplo, salve o "fiddle" e compartilhe o link com colegas ou professores para feedback.

---

## Aplicações Reais do jsFiddle com Vue.js

1. **Ensino e Aprendizado**: Professores e alunos podem usar o jsFiddle para compartilhar exemplos durante as aulas.
2. **Prototipagem Rápida**: Desenvolvedores podem testar ideias ou funcionalidades antes de integrá-las ao projeto principal.
3. **Depuração de Problemas**: Útil para isolar problemas em trechos de código sem interferência de outras partes do sistema.

---

### 2.2 Hello World com Vue.js

O primeiro passo em qualquer aprendizado de uma nova tecnologia é criar um exemplo básico, o famoso "Hello World". No Vue.js, isso é feito de maneira simples e clara, destacando uma das principais vantagens do framework: sua facilidade de configuração e uso.

---

## O que é o "Hello World" no Vue.js?

O "Hello World" no Vue.js é o exemplo mais básico de como renderizar informações dinâmicas na tela usando o poder do framework. Ele demonstra como ligar o JavaScript ao HTML por meio de sua reatividade e data binding.

---

## Como Criar um Hello World com Vue.js

O Vue.js pode ser usado diretamente em um arquivo HTML sem a necessidade de configuração avançada. Tudo o que você precisa é incluir a biblioteca Vue.js e criar uma instância Vue para interagir com o DOM.

### Exemplo Básico

1. **Código HTML:**

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Hello World com Vue.js</title>
  <script src="https://cdn.jsdelivr.net/npm/vue@3"></script>
</head>
<body>
  <div id="app">
    <h1>{{ mensagem }}</h1>
  </div>

  <script>
    // Instância Vue
    const app = Vue.createApp({
      data() {
        return {
          mensagem: 'Olá, Mundo! Este é o meu primeiro projeto com Vue.js!'
        };
      }
    });

    // Ligação com o DOM
    app.mount('#app');
  </script>
</body>
</html>
```

### Explicação do Código

1. **Incluir Vue.js:** A biblioteca Vue.js é carregada diretamente de um CDN. Nesse exemplo, usamos a versão 3 do Vue.js.
2. **Elemento de Raiz (`#app`):** O Vue.js se conecta ao elemento com o ID `app`, tornando-o reativo.
3. **Instância Vue:** A função `Vue.createApp()` cria uma nova instância do Vue. O objeto retornado é configurado com um `data()` que define as variáveis usadas na interface.
4. **Interpolação:** A variável `mensagem` é exibida na página por meio de `{{ mensagem }}`.

**Saída no Navegador:**

```
Olá, Mundo! Este é o meu primeiro projeto com Vue.js!
```

## Exemplo Interativo - Atualizando a Mensagem

Para tornar o exemplo mais dinâmico, podemos adicionar um botão que atualiza a mensagem na tela.

1. **Código HTML:**

```
<div id="app">
  <h1>{{ mensagem }}</h1>
  <button @click="atualizarMensagem">Clique para alterar a mensagem</button>
</div>
```

2. **Código JavaScript**

```
const app = Vue.createApp({
  data() {
    return {
      mensagem: 'Olá, Mundo!'
    };
  },
  methods: {
    atualizarMensagem() {
      this.mensagem = 'Você clicou no botão!';
    }
  }
});

app.mount('#app');
```

**Explicação Adicional:**

- O atributo `@click` vincula o evento de clique ao método `atualizarMensagem`.
- O método `atualizarMensagem` altera o valor de `mensagem`, e o Vue.js automaticamente atualiza a interface devido à reatividade.

---

### 2.3 Two-Way Data Binding e Reatividade

Uma das características mais poderosas e diferenciadoras do Vue.js é o two-way data binding (ligação bidirecional de dados). Ele permite que os dados entre o modelo JavaScript e o DOM HTML estejam sempre sincronizados. Esse recurso é a base da reatividade do Vue.js, simplificando a criação de interfaces dinâmicas.

---

## O que é Two-Way Data Binding?

**O two-way data binding significa que:**

1. Alterações feitas no modelo de dados (variáveis no `data()`) são refletidas automaticamente na interface (DOM).
2. Alterações feitas pelo usuário na interface (ex.: em inputs) atualizam automaticamente o modelo de dados.

---

## Por que o Two-Way Data Binding é importante?

1. Simplicidade: Elimina a necessidade de manipular manualmente o DOM com JavaScript.
2. Reatividade: A interface responde automaticamente a alterações nos dados, garantindo uma experiência de usuário mais fluida.
3. Produtividade: Reduz a quantidade de código necessário para sincronizar dados.

---

## Como Funciona o Two-Way Data Binding no Vue.js?

O Vue.js implementa o two-way data binding principalmente com a diretiva `v-model`. Essa diretiva é usada para ligar um campo de entrada de dados (ex.: input, textarea) a uma variável no `data()`.

**Exemplo Básico: Campo de Texto Dinâmico**

1. **HTML:**

```
<div id="app">
  <h2>Nome: {{ nome }}</h2>
  <input type="text" v-model="nome">
</div>
```

2. **JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      nome: 'João'
    };
  }
});

app.mount('#app');
```

**Explicação do Funcionamento:**

- A diretiva `v-model` sincroniza o campo de entrada (`input`) com a variável `nome` no `data()`.
- Quando o usuário digita no campo, o valor de `nome` é atualizado automaticamente.
- Da mesma forma, alterações no `data()` (ex.: `this.nome = 'Maria'`) também atualizam o campo de entrada.

---

## Reatividade no Vue.js

O Vue.js utiliza um sistema de reatividade que observa alterações nos dados e atualiza o DOM automaticamente. Isso é feito por meio de um mecanismo chamado Observer.

**Como a Reatividade Funciona?**

1. Quando uma variável no `data()` é declarada, o Vue.js a transforma em reativa.
2. Sempre que o valor dessa variável muda, o Vue.js identifica a alteração e atualiza a interface correspondente.

### Exemplo Prático de Reatividade: Contador Dinâmico

1. **HTML:**

```
<div id="app">
  <h2>Contador: {{ contador }}</h2>
  <button @click="incrementar">+</button>
  <button @click="decrementar">-</button>
</div>
```

2. **JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      contador: 0
    };
  },
  methods: {
    incrementar() {
      this.contador++;
    },
    decrementar() {
      this.contador--;
    }
  }
});

app.mount('#app');
```

## Exemplo Avançado: Formulário com Two-Way Data Binding

1. **HTML:**

```
<div id="app">
  <h2>Formulário de Cadastro</h2>
  <form @submit.prevent="cadastrar">
    <label for="nome">Nome:</label>
    <input id="nome" v-model="nome" type="text">

    <label for="email">Email:</label>
    <input id="email" v-model="email" type="email">

    <button type="submit">Cadastrar</button>
  </form>
  <p>Dados:</p>
  <ul>
    <li>Nome: {{ nome }}</li>
    <li>Email: {{ email }}</li>
  </ul>
</div>
```

2. **JavaScript**

```
const app = Vue.createApp({
  data() {
    return {
      nome: '',
      email: ''
    };
  },
  methods: {
    cadastrar() {
      alert(`Usuário cadastrado: ${this.nome}, ${this.email}`);
      this.nome = '';
      this.email = '';
    }
  }
});

app.mount('#app');
```

**Explicação:**

- A diretiva `v-model` é usada para capturar os valores digitados nos campos de entrada.
- O método `cadastrar` exibe um alerta com os dados preenchidos e, em seguida, limpa os campos.

---

## Resumo

- O "Hello World" é o primeiro exemplo básico de Vue.js, mostrando como o framework se conecta ao DOM.
- O **two-way data binding** simplifica o desenvolvimento ao sincronizar automaticamente os dados do modelo com a interface do usuário.
- A **reatividade** é a base do Vue.js, permitindo interfaces dinâmicas e responsivas.

Esses conceitos são fundamentais para começar a criar aplicações poderosas e reativas com Vue.js!

---

### 2.4 Manipulação de Arrays e Objetos no Vue.js

A manipulação de arrays e objetos é uma tarefa comum no desenvolvimento de aplicações. No Vue.js, temos ferramentas poderosas para lidar com essas estruturas de dados de maneira eficiente, especialmente com o uso das diretivas `v-for` e `v-bind:key`, além de métodos úteis como `set` e funcionalidades para remoção de itens.

## Manipulação de Arrays

Os arrays são frequentemente utilizados para armazenar listas de dados que precisam ser renderizados dinamicamente. O Vue.js facilita a iteração sobre arrays e a exibição de seus itens na interface por meio da diretiva `v-for`.

---

### v-for: Renderizando Listas

A diretiva `v-for` permite iterar sobre arrays e objetos para renderizar itens dinamicamente no DOM.

#### Exemplo Básico

**HTML:**

```
<div id="app">
  <h2>Lista de Tarefas</h2>
  <ul>
    <li v-for="(tarefa, index) in tarefas" :key="index">
      {{ tarefa }}
    </li>
  </ul>
</div>
```

**JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      tarefas: ['Estudar Vue.js', 'Praticar exercícios', 'Ler um livro']
    };
  }
});

app.mount('#app');
```

**Explicação:**

- O `v-for` percorre o array `tarefas`.
- Cada item do array é renderizado como um elemento `<li>`.

- A diretiva `:key` é usada para fornecer uma identificação única a cada item, garantindo que o Vue manipule eficientemente a renderização.

---

### Adicionando e Removendo Itens

Adicionar e remover itens em arrays é uma tarefa simples no Vue.js. Podemos usar métodos JavaScript como `push` e `splice`, e o Vue cuidará automaticamente da atualização do DOM.

Exemplo: Adicionar e Remover Itens

**HTML:**

```
<div id="app">
  <h2>Lista de Compras</h2>
  <ul>
    <li v-for="(item, index) in lista" :key="index">
      {{ item }}
      <button @click="removerItem(index)">Remover</button>
    </li>
  </ul>
  <input v-model="novoItem" placeholder="Adicionar item">
  <button @click="adicionarItem">Adicionar</button>
</div>
```

**JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      lista: ['Leite', 'Pão', 'Manteiga'],
      novoItem: ''
    };
  },
  methods: {
    adicionarItem() {
      if (this.novoItem.trim() !== '') {
        this.lista.push(this.novoItem);
        this.novoItem = '';
      }
    },
    removerItem(index) {
      this.lista.splice(index, 1);
    }
  }
});

app.mount('#app');
```

**Explicação:**

- O método `adicionarItem` usa `push` para adicionar um novo item à lista.
- O método `removerItem` utiliza `splice` para remover um item com base no índice.

### Vue.set: Reatividade em Arrays e Objetos

Por padrão, o Vue.js não detecta alterações em propriedades adicionadas dinamicamente a objetos ou mudanças diretas em índices de arrays. Para isso, utilizamos o método `Vue.set`.

Exemplo com Vue.set

**HTML:**

```
<div id="app">
  <h2>Produtos</h2>
  <ul>
    <li v-for="(produto, index) in produtos" :key="index">
      {{ produto.nome }} - R$ {{ produto.preco }}
    </li>
  </ul>
  <button @click="atualizarPreco(1, 15.99)">Atualizar Preço</button>
</div>
```

**JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      produtos: [
        { nome: 'Produto A', preco: 10.0 },
        { nome: 'Produto B', preco: 20.0 }
      ]
    };
  },
  methods: {
    atualizarPreco(index, novoPreco) {
      this.produtos[index].preco = novoPreco;
    }
  }
});

app.mount('#app');
```

**Explicação:**

- `Vue.set` pode ser usado para garantir a reatividade ao alterar índices ou propriedades inexistentes de objetos.

## Eventos e Métodos no Vue.js

O Vue.js oferece um sistema robusto para lidar com eventos do DOM por meio da diretiva `@event`. Além disso, modificadores de eventos e modificadores de teclas permitem que você gerencie eventos de maneira mais eficiente e elegante.

---

### Modificadores de Evento

Modificadores de evento são sufixos adicionados à diretiva de evento para modificar o comportamento padrão de um evento do DOM.

**Modificadores Comuns:**

1. **.prevent**: Previne o comportamento padrão do navegador.
2. **.stop**: Impede a propagação do evento para elementos pai.
3. **.self**: Garante que o evento seja acionado apenas no elemento atual.

**Exemplo: Usando .prevent**

**HTML:**

```html
<div id="app">
  <form @submit.prevent="enviarFormulario">
    <label for="nome">Nome:</label>
    <input id="nome" v-model="nome" type="text">
    <button type="submit">Enviar</button>
  </form>
  <p>Nome enviado: {{ nome }}</p>
</div>
```

**JavaScript:**

```javascript
const app = Vue.createApp({
  data() {
    return {
      nome: ''
    };
  },
  methods: {
    enviarFormulario() {
      alert(`Formulário enviado com sucesso! Nome: ${this.nome}`);
      this.nome = '';
    }
  }
});

app.mount('#app');
```

**Explicação:**

- O modificador `.prevent` impede o comportamento padrão de envio do formulário, permitindo que o método `enviarFormulario` seja executado sem recarregar a página.

### Modificadores de Teclas

Os modificadores de teclas permitem capturar eventos de teclado específicos, como pressionar a tecla "Enter" ou "Esc".

#### Exemplo: Capturando Enter e Esc

**HTML:**

```html
<div id="app">
        <input v-model="texto" @keyup.enter="salvar" @keyup.esc="limpar" placeholder="Digite algo e pressione Enter ou Esc">
        <p>Texto: {{ texto }}</p>
</div>
```

**JavaScript:**

```javascript
const app = Vue.createApp({
  data() {
    return {
      texto: ''
    };
  },
  methods: {
    salvar() {
      alert(`Texto salvo: ${this.texto}`);
    },
    limpar() {
      this.texto = '';
    }
  }
});

app.mount('#app');
```

**Explicação:**

- O modificador `.enter` aciona o método `salvar` ao pressionar a tecla Enter.
- O modificador `.esc` limpa o texto ao pressionar a tecla Esc.

---

## Resumo

- **Manipulação de Arrays e Objetos:** `v-for` e `v-bind:key` facilitam a renderização de listas dinâmicas, enquanto métodos como `set` garantem a reatividade.

- **Eventos e Métodos:** Os modificadores de eventos, como `.prevent` e `.stop`, permitem controlar o comportamento padrão do navegador, e os modificadores de teclas simplificam a captura de eventos específicos do teclado.

### 2.5 Design Reativo e Ciclo de Vida dos Componentes

O **design reativo** e o **ciclo de vida dos componentes** são pilares do Vue.js, permitindo que os desenvolvedores criem aplicações dinâmicas e interativas de maneira eficiente. Esses conceitos garantem que as alterações nos dados sejam refletidas automaticamente na interface, enquanto o ciclo de vida dos componentes define como e quando as ações ocorrem dentro de um componente.

---

## Design Reativo no Vue.js

O Vue.js é baseado em um sistema de reatividade que observa alterações nos dados e atualiza automaticamente o DOM (Document Object Model). Esse comportamento elimina a necessidade de manipulação manual do DOM, tornando o desenvolvimento mais simples e produtivo.

---

#### Como Funciona o Design Reativo?

1. **Observação de Dados:** O Vue.js monitora as propriedades declaradas no `data()`.
2. **Atualização Automática:** Quando uma propriedade é alterada, o Vue atualiza todas as partes da interface vinculadas a essa propriedade.
3. **Binding (Ligação):** A ligação entre dados e o DOM é realizada por meio de diretivas, como `v-bind`, `v-model` e interpolação `{{ }}`.

---

#### Exemplo Prático: Contador Reativo

**HTML:**

```
<div id="app">
  <h2>Contador Reativo</h2>
  <p>Valor do contador: {{ contador }}</p>
  <button @click="incrementar">Incrementar</button>
  <button @click="decrementar">Decrementar</button>
</div>
```

**JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      contador: 0
    };
  },
  methods: {
    incrementar() {
      this.contador++;
    },
    decrementar() {
      this.contador--;
    }
  }
});

app.mount('#app');
```

**Explicação:**

- A propriedade `contador` no `data()` é monitorada pelo Vue.
- Quando `incrementar` ou `decrementar` é acionado, o Vue detecta a mudança em `contador` e atualiza o DOM automaticamente.

---

**Vantagens do Design Reativo**

1. **Simplifica o código:** Você não precisa manipular o DOM diretamente.
2. **Alta performance:** O Vue utiliza um sistema eficiente de comparação (Virtual DOM) para minimizar alterações no DOM real.
3. **Manutenção facilitada:** O código é mais legível e modular.

---

## Ciclo de Vida dos Componentes

Cada componente no Vue.js possui um **ciclo de vida**, que consiste em uma série de etapas desde sua criação até sua destruição. Durante essas etapas, o Vue oferece **hooks de ciclo de vida** (funções) que podem ser usados para executar lógica personalizada.

---

**Fases do Ciclo de Vida**

1. **Criação:**
   - O componente é inicializado e suas propriedades e eventos são configurados.
   - Hooks: `beforeCreate`, `created`.
2. **Montagem:**
   - O componente é inserido no DOM.
   - Hooks: `beforeMount`, `mounted`.
3. **Atualização:**
   - O componente é atualizado devido a mudanças em seus dados ou propriedades.
   - Hooks: `beforeUpdate`, `updated`.
4. **Destruição:**
   - O componente é removido do DOM.
   - Hooks: `beforeUnmount`, `unmounted`.

---

**Exemplo Prático: Utilizando Hooks de Ciclo de Vida**

**HTML:**

```html
<div id="app">
  <h2>Exemplo de Ciclo de Vida</h2>
  <p>Mensagem: {{ mensagem }}</p>
</div>
```

**JavaScript:**

```javascript
const app = Vue.createApp({
  data() {
    return {
      mensagem: 'Olá, Vue.js!'
    };
  },
  beforeCreate() {
    console.log('beforeCreate: Componente está sendo criado...');
  },
  created() {
    console.log('created: Componente criado com sucesso!');
  },
  beforeMount() {
    console.log('beforeMount: Preparando para montar no DOM...');
  },
  mounted() {
    console.log('mounted: Componente montado no DOM!');
  },
  beforeUnmount() {
    console.log('beforeUnmount: Preparando para desmontar...');
  },
  unmounted() {
    console.log('unmounted: Componente desmontado.');
  }
});

app.mount('#app');
```

**Explicação:**

- Os logs mostram em qual fase o componente está durante sua execução.
- O hook `mounted` é útil para inicializar plugins ou realizar requisições após o componente estar no DOM.

---

---

### 2.6 Criação de Formulários Dinâmicos

Os formulários dinâmicos no Vue.js são criados usando diretivas como `v-model` para ligar os campos de entrada aos dados no `data()`. Além disso, é possível criar interatividade com elementos como **checkboxes**, **radios** e **selects** de forma simples e reativa.

---

## Checkboxes

Os checkboxes permitem selecionar múltiplos valores e podem ser controlados por meio de arrays ou booleanos.

**Exemplo: Checkbox com Vários Valores**

**HTML:**

```
<div id="app">
  <h2>Selecione seus interesses:</h2>
  <label>
    <input type="checkbox" value="Esportes" v-model="interesses"> Esportes
  </label>
  <label>
    <input type="checkbox" value="Música" v-model="interesses"> Música
  </label>
  <label>
    <input type="checkbox" value="Tecnologia" v-model="interesses"> Tecnologia
  </label>
  <p>Interesses selecionados: {{ interesses }}</p>
</div>
```

**JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      interesses: []
    };
  }
});

app.mount('#app');
```

### Radios

Os botões de rádio são usados para selecionar um único valor entre várias opções.

**Exemplo: Seleção de Gênero**

**HTML:**

```
<div id="app">
  <h2>Selecione seu gênero:</h2>
  <label>
    <input type="radio" value="Masculino" v-model="genero"> Masculino
  </label>
  <label>
    <input type="radio" value="Feminino" v-model="genero"> Feminino
  </label>
  <label>
    <input type="radio" value="Outro" v-model="genero"> Outro
  </label>
  <p>Gênero selecionado: {{ genero }}</p>
</div>
```

**JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      genero: ''
    };
  }
});

app.mount('#app');
```

## Select

Os elementos `select` permitem criar listas suspensas, e o Vue facilita a ligação dinâmica de dados.

**Exemplo: Seleção de Cidades**

**HTML:**

```
<div id="app">
  <h2>Selecione uma cidade:</h2>
  <select v-model="cidade">
    <option disabled value="">Escolha uma cidade</option>
    <option>São Paulo</option>
    <option>Rio de Janeiro</option>
    <option>Belo Horizonte</option>
  </select>
  <p>Cidade selecionada: {{ cidade }}</p>
</div>
```

**JavaScript:**

```
const app = Vue.createApp({
  data() {
    return {
      cidade: ''
    };
  }
});

app.mount('#app');
```

## Formulário Completo

### HTML:

```
<div id="app">
  <h2>Formulário de Cadastro</h2>
  <form @submit.prevent="cadastrar">
    <label>Nome:</label>
    <input v-model="nome" type="text">

    <label>Gênero:</label>
    <input type="radio" value="Masculino" v-model="genero"> Masculino
    <input type="radio" value="Feminino" v-model="genero"> Feminino

    <label>Interesses:</label>
    <input type="checkbox" value="Esportes" v-model="interesses"> Esportes
    <input type="checkbox" value="Música" v-model="interesses"> Música

    <label>Cidade:</label>
    <select v-model="cidade">
      <option>São Paulo</option>
      <option>Rio de Janeiro</option>
    </select>

    <button type="submit">Cadastrar</button>
  </form>
  <p>Dados: {{ nome }}, {{ genero }}, {{ interesses }}, {{ cidade }}</p>
</div>
```

**JavaScript:**

```
const app = Vue.createApp({
        data() {
                return {
                        nome: '',
                        genero: '',
                        interesses: [ ],
                        cidade: ''
                        };
                },
                methods: {
                        cadastrar() {
                        alert(`Usuário cadastrado:
                                ${this.nome},
                                ${this.genero},
                                ${this.interesses.join(', ')},
                                ${this.cidade}`);
                        }
                }
});

app.mount('#app');
```

## Resumo

- **Design Reativo:** Torna a aplicação dinâmica, atualizando automaticamente o DOM.
- **Ciclo de Vida:** Define quando e como as ações acontecem nos componentes.
- **Formulários Dinâmicos:** Permitem capturar dados de usuários de forma eficiente, usando `v-model` para sincronização.

Esses conceitos são essenciais para o desenvolvimento de interfaces modernas com Vue.js.

## Lista de Exercícios

---

**Parte 1: Questões Teóricas (15 questões)**

1. **Explique o que é o jsFiddle e qual a sua utilidade no desenvolvimento com Vue.js.**
2. **Descreva o processo para configurar o Vue.js no jsFiddle. Quais os passos necessários para incluir a biblioteca Vue.js?**
3. **O que é o "Hello World" no Vue.js e qual a sua importância no aprendizado do framework?**
4. **Defina o conceito de two-way data binding no Vue.js. Por que ele é uma característica importante?**
5. **Qual a principal função do `v-for` no Vue.js? Dê um exemplo de quando ele é utilizado.**

6. **Explique o propósito da diretiva `v-bind:key` ao trabalhar com listas dinâmicas no Vue.js.**
7. **O que é a reatividade no Vue.js e como ela simplifica o desenvolvimento de interfaces dinâmicas?**
8. **Quais são as principais vantagens de usar o ciclo de vida dos componentes no Vue.js? Cite pelo menos dois hooks e suas finalidades.**
9. **Diferencie os métodos JavaScript `push` e `splice` ao manipular arrays no Vue.js. Como o Vue detecta alterações feitas com esses métodos?**
10. **O que é o método `Vue.set` e em quais cenários ele é necessário para garantir a reatividade?**
11. **Explique o uso dos modificadores de evento `.prevent` e `.stop` no Vue.js. Como eles afetam o comportamento de um evento do DOM?**
12. **Descreva como os modificadores de teclas, como `.enter` e `.esc`, são usados no Vue.js. Cite um exemplo prático de aplicação.**
13. **Quais são as diferenças entre checkboxes, radios e selects em formulários dinâmicos no Vue.js?**
14. **Por que é importante usar o atributo `v-model` ao criar formulários no Vue.js?**
15. **Explique como o Vue.js atualiza o DOM de maneira eficiente ao usar o Virtual DOM. Por que isso é vantajoso?**

---

**Parte 2: Questões Práticas (10 questões)**

1. **Configure um ambiente no jsFiddle para utilizar o Vue.js e crie um exemplo básico que exiba uma mensagem "Olá, Vue.js!". Inclua um botão para alterar a mensagem dinamicamente.**
2. **Crie um contador interativo usando o Vue.js, onde dois botões incrementem e decremente o valor do contador.**
3. **Implemente uma lista de tarefas usando `v-for` para renderizar os itens. Permita que o usuário adicione e remova tarefas da lista.**
4. **Crie um exemplo em que você utiliza `Vue.set` para atualizar dinamicamente o valor de um índice específico de um array, garantindo a reatividade.**
5. **Desenvolva um formulário dinâmico com Vue.js que inclua os seguintes campos:**
    - Nome (input de texto).
    - Gênero (radio buttons).
    - Interesses (checkboxes).
    - Cidade (select).

      Exiba os dados preenchidos abaixo do formulário em tempo real.
6. **Usando modificadores de evento, crie um formulário onde o comportamento padrão do botão de envio seja prevenido e, em vez disso, exiba os dados do formulário em um alerta.**
7. **Crie um exemplo que capture eventos de teclado no Vue.js. Quando o usuário pressionar Enter, o texto digitado deve ser salvo. Caso pressione Esc, o texto deve ser apagado.**

8. **Construa uma aplicação Vue.js que renderize uma lista de produtos com os seguintes dados: Nome, Preço e Quantidade. Permita que o usuário edite o preço de um produto específico dinamicamente.**
9. **Desenvolva uma página que utilize o ciclo de vida dos componentes para:**
    - Mostrar uma mensagem de log no console ao inicializar o componente.
    - Realizar uma requisição (fictícia) simulada quando o componente for montado.
10. **Implemente uma interface que permita ao usuário adicionar itens a um carrinho de compras. Cada item deve ser exibido em uma tabela com a opção de removê-lo. Use `v-for` e `v-bind:key` para lidar com os dados dinamicamente.**

- *Responda todas as questões teóricas e práticas.*
- *Organize as respostas em um arquivo PDF, contendo nome e data.*
- *Envie o PDF aos professores responsáveis, seguindo o padrão de nomenclatura.*
- *Valide o material com um professor antes de prosseguir.*
- *Certifique-se de cumprir o prazo de entrega.*

***Padrão de nomeclatura***
***NomeCompleto_TrilhaX_DataDeEntrega.pdf***
***Exemplo: JoãoSilva_Trilha1_2025-01-30.pdf***