UniSENAI

Trilha 03

Componentes Vue

3 - Componentes Vue

Os **componentes** são o coração do desenvolvimento com Vue.js. Eles permitem criar interfaces modulares e reutilizáveis, promovendo organização e escalabilidade no código. Nesta seção, você aprenderá sobre a estrutura dos arquivos `.vue`, como os componentes se comunicam, o uso de **props**, **eventos**, **slots** e **mixins**, além de boas práticas para organização de projetos.

3.1 Estrutura de Componentes (.vue)

Um componente Vue é geralmente representado por um arquivo `.vue`. Esses arquivos possuem uma estrutura padrão, onde HTML, CSS e JavaScript coexistem no mesmo arquivo, criando uma abordagem modular.

Estrutura Básica de um Arquivo `.vue`

```
<template>
  <div>
    <h1>{{ titulo }}</h1>
    <p>Componente Vue.js básico!</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      titulo: 'Olá, Vue Component!'
    };
  }
};
</script>

<style>
h1 {
  color: blue;
}
</style>
```

Divisões do Arquivo:

1. `<template>`: Contém o código HTML que será renderizado.
2. `<script>`: Define a lógica do componente, como dados e métodos.
3. `<style>`: Contém os estilos CSS específicos do componente. Pode ser global ou scoped (limitado ao componente).

3.2 Comunicação entre Componentes

Em aplicações Vue, os componentes frequentemente precisam trocar informações. Essa comunicação é feita de maneira hierárquica:

1. **De pai para filho:** Através de **props**.
2. **De filho para pai:** Através de **eventos personalizados**.
3. **Entre componentes irmãos ou distantes:** Através de **gerenciadores de estado** (como Vuex) ou por meio de um evento intermediário.

Props: Passando Dados do Pai para o Filho

As **props** são propriedades que permitem enviar dados de um componente pai para um componente filho.

Exemplo:

1. **Componente Pai (App.vue):**

```
<template>
  <div>
    <h1>Componente Pai</h1>
    <Filho :mensagem="mensagemDoPai" />
  </div>
</template>

<script>
import Filho from './Filho.vue';

export default {
  components: { Filho },
  data() {
    return {
      mensagemDoPai: 'Olá, do componente pai!'
    };
  }
};
</script>
```

2. **Componente Filho (Filho.vue):**

```
<template>
  <div>
    <h2>Componente Filho</h2>
    <p>Mensagem recebida: {{ mensagem }}</p>
  </div>
</template>

<script>
export default {
  props: ['mensagem']
};
</script>
```

Explicação:

- O componente pai usa a diretiva `:mensagem` para passar a propriedade `mensagemDoPai` para o componente filho.
- O componente filho acessa essa propriedade por meio de `props`.

Eventos: Enviando Dados do Filho para o Pai

Quando um componente filho precisa enviar informações para o pai, ele utiliza **eventos personalizados**.

Exemplo:

1. **Componente Pai (App.vue):**

```
<template>
  <div>
    <h1>Componente Pai</h1>
    <Filho @enviarMensagem="receberMensagem" />
    <p>Mensagem do Filho: {{ mensagemFilho }}</p>
  </div>
</template>

<script>
import Filho from './Filho.vue';

export default {
  components: { Filho },
  data() {
    return {
      mensagemFilho: ''
    };
  },
  methods: {
    receberMensagem(mensagem) {
      this.mensagemFilho = mensagem;
    }
  }
};
</script>
```

2. **Componente Filho (Filho.vue):**

```
<template>
  <div>
    <h2>Componente Filho</h2>
    <button @click="enviarMensagem">Enviar Mensagem</button>
  </div>
</template>

<script>
export default {
  methods: {
    enviarMensagem() {
      this.$emit('enviarMensagem', 'Olá, do componente filho!');
    }
  }
};
</script>
```

Explicação:

- O filho emite o evento `enviarMensagem` com a mensagem como argumento.
- O pai ouve esse evento e executa o método `receberMensagem`.

Slots: Personalizando Componentes

Os **slots** permitem que o conteúdo de um componente pai seja inserido em um componente filho.

Exemplo:

1. **Componente Pai (App.vue):**

```
<template>
  <div>
    <ComponenteComSlot>
      <h2>Conteúdo Personalizado</h2>
      <p>Este conteúdo foi inserido no slot!</p>
    </ComponenteComSlot>
  </div>
</template>

<script>
import ComponenteComSlot from './ComponenteComSlot.vue';

export default {
  components: { ComponenteComSlot }
};
</script>
```

2. **Componente Filho (ComponenteComSlot.vue):**

```
<template>
  <div>
    <h1>Componente com Slot</h1>
    <slot></slot>
  </div>
</template>
```

Explicação:

- O `<slot>` no componente filho é preenchido com o conteúdo fornecido pelo componente pai.

3.3 Criando e Utilizando Mixins

Os **mixins** são uma maneira de reutilizar lógica entre diferentes componentes. Eles permitem que você extraia métodos, propriedades ou hooks e os reutilize em outros componentes.

Exemplo de Mixin:

1. **Mixin (meuMixin.js):**

```
export default {
  data() {
    return {
      mensagemMixin: 'Olá, de um mixin!'
    };
  },
  methods: {
    mostrarMensagem() {
      console.log(this.mensagemMixin);
    }
  }
};
```

2. **Componente Vue (App.vue):**

```
<template>
  <div>
    <h1>{{ mensagemMixin }}</h1>
    <button @click="mostrarMensagem">Mostrar Mensagem</button>
  </div>
</template>

<script>
import meuMixin from './meuMixin';

export default {
  mixins: [meuMixin]
};
</script>
```

Explicação:

- O mixin encapsula lógica que pode ser reutilizada em outros componentes.
- O componente importa o mixin e automaticamente obtém seus dados e métodos.

3.4 Organização de Projetos e Estruturação de Pastas

Uma boa organização do projeto facilita a manutenção e escalabilidade. No Vue.js, a estrutura padrão é:

```
src/
├── assets/        # Arquivos estáticos (imagens, fontes, etc.)
├── components/    # Componentes reutilizáveis
│   ├── Header.vue
│   ├── Footer.vue
│   └── Card.vue
├── views/         # Componentes de páginas
│   ├── Home.vue
│   ├── About.vue
│   └── Contact.vue
├── router/        # Configuração de rotas
│   └── index.js
├── store/         # Gerenciamento de estado (Vuex)
│   └── index.js
├── App.vue        # Componente raiz
└── main.js        # Arquivo principal
```

Dicas de Organização:

1. Separe componentes reutilizáveis em uma pasta específica (`components/`).
2. Use a pasta `views/` para componentes que representam páginas.
3. Centralize o gerenciamento de estado em `store/` (se utilizar Vuex).
4. Armazene rotas na pasta `router/`.

Resumo

- **Estrutura de Componentes (.vue):** Divide HTML, CSS e JavaScript em um único arquivo.
- **Comunicação entre Componentes:** Realizada por meio de **props**, **eventos** e **slots**.
- **Mixins:** Promovem a reutilização de lógica em diferentes componentes.
- **Organização do Projeto:** Uma boa estrutura melhora a manutenibilidade e escalabilidade da aplicação.

Lista de Atividades

Questões Teóricas (10 questões)

1. **Explique a estrutura básica de um arquivo `.vue`. Quais são as principais seções e suas finalidades?**
2. **Defina o que são props no Vue.js e como elas são usadas para comunicação entre componentes.**
3. **Explique a diferença entre `props` e eventos personalizados na comunicação entre componentes no Vue.js. Quando usar cada um deles?**
4. **O que são slots no Vue.js e qual a sua principal utilidade? Dê um exemplo de como eles podem ser usados para personalizar componentes.**
5. **Descreva o conceito de mixins no Vue.js. Quais são os benefícios e os possíveis problemas de usá-los em projetos grandes?**
6. **Explique a importância de utilizar o atributo `v-bind:key` ao trabalhar com listas dinâmicas em um componente.**
7. **Quais são as vantagens de organizar um projeto Vue.js utilizando pastas como `components/`, `views/`, `router/` e `store/`?**
8. **O que acontece se dois componentes diferentes utilizarem o mesmo mixin? Como o Vue trata possíveis conflitos de métodos ou dados?**
9. **Qual a diferença entre usar um slot simples (`<slot></slot>`) e um slot nomeado (`<slot name="..."></slot>`) em um componente Vue?**
10. **Descreva como o Vue.js facilita a comunicação de dados entre componentes pais e filhos. Por que isso é importante para aplicações modulares?**

Questões Práticas (10 questões)

1. **Crie um componente básico Vue (`.vue`) que exiba uma mensagem dinâmica vinda de um dado no `data()`. Adicione estilos locais na seção `<style>`.**
2. **Crie um componente pai e um componente filho. O pai deve passar uma mensagem como `prop` para o filho, e o filho deve exibi-la na tela.**
3. **Implemente um sistema onde o componente filho envie um dado para o componente pai por meio de um evento personalizado.**
4. **Crie um componente que utilize um slot simples para renderizar conteúdo personalizado vindo do componente pai.**
5. **Adapte o exercício anterior para utilizar slots nomeados, permitindo renderizar diferentes partes de um layout (ex.: cabeçalho e rodapé).**
6. **Desenvolva um mixin que contenha um método e um dado. Implemente esse mixin em dois componentes diferentes e demonstre o uso de suas funcionalidades.**
7. **Organize um projeto Vue.js criando a seguinte estrutura:**
    - Uma pasta `components/` contendo dois componentes reutilizáveis (ex.: `Header.vue` e `Footer.vue`).

- Uma pasta `views/` contendo duas páginas (`Home.vue` e `About.vue`).
- Um arquivo `router/index.js` para configurar rotas entre as páginas.

8. **Crie um componente que renderize uma lista de itens usando `v-for`. Garanta que cada item tenha uma `key` única com `v-bind:key`.**
9. **Desenvolva um componente pai que contenha um botão. Ao clicar no botão, o pai deve enviar um dado para o componente filho usando `props`.**
10. **Crie um formulário que utilize slots para personalizar seus campos de entrada. O formulário deve conter um cabeçalho, campos de texto e um botão, todos configurados dinamicamente por meio de slots.**

- *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