UniSENAI

Trilha 08

Conceitos Avançados em Vue.js

Parte 8 - Conceitos Avançados no Vue.js

Nesta unidade, vamos aprofundar os conceitos avançados do Vue.js, abordando **gerenciamento de estado com Pinia**, **Composition API**, **boas práticas de desenvolvimento** e ferramentas que ajudam a manter um código limpo e organizado.

Além disso, haverá uma **atividade extra**, onde os acadêmicos deverão pesquisar e preparar uma apresentação sobre **Beautifully Designed para Vue**, explorando técnicas para criar interfaces visuais atraentes.

8.1 Pinia e Composition API

O que é o Pinia?

O **Pinia** é a biblioteca oficial para gerenciamento de estado no Vue.js, substituindo o Vuex. Ele é mais **intuitivo, flexível e performático** do que seu antecessor, permitindo **tipagem mais forte** e integração facilitada com a **Composition API**.

Principais características do Pinia:

✔ **Mais simples que Vuex** – Sintaxe enxuta e menos código boilerplate.
✔ **Melhor suporte à Composition API** – Integração nativa com Vue 3.
✔ **Armazenamento reativo** – O estado é automaticamente reativo.
✔ **Suporte a TypeScript** – Melhor tipagem para desenvolvimento escalável.

8.2 Introdução ao Pinia: Store, State, Mutations, Actions e Getters

Antes de usar o **Pinia**, primeiro precisamos instalá-lo no projeto Vue.js:

```
npm install pinia
```

Agora, importamos e configuramos o Pinia no projeto. No arquivo `main.js` ou `main.ts`, adicionamos:

```
import { createApp } from 'vue';
import { createPinia } from 'pinia';
import App from './App.vue';

const app = createApp(App);
app.use(createPinia());
app.mount('#app');
```

Criando uma Store no Pinia

O Pinia funciona por meio de "stores", que são locais onde o estado é armazenado. Vamos criar um exemplo simples de gerenciamento de usuários.

📌 **Passo 1: Criar a Store**
Crie o arquivo `stores/usuarioStore.js`:

```
import { defineStore } from 'pinia';

export const useUsuarioStore = defineStore('usuario', {
  state: () => ({
    nome: 'João Silva',
    idade: 25
  }),
  getters: {
    saudacao: (state) => `Olá, ${state.nome}!`
  },
  actions: {
    alterarNome(novoNome) {
      this.nome = novoNome;
    }
  }
});
```

Explicação:

- **`state`** → Armazena os dados reativos.
- **`getters`** → São funções que retornam valores computados.
- **`actions`** → Métodos que alteram o estado.

📌 **Passo 2: Utilizar a Store no Componente**
Agora, utilizamos a store dentro de um componente Vue:

Vue Component Example

```vue
<template>
  <div>
    <h2>{{ usuario.saudacao }}</h2>
    <input v-model="novoNome" placeholder="Novo nome" />
    <button @click="usuario.alterarNome(novoNome)">Alterar Nome</button>
  </div>
</template>

<script>
import { useUsuarioStore } from '@/stores/usuarioStore';
import { ref } from 'vue';

export default {
  setup() {
    const usuario = useUsuarioStore();
    const novoNome = ref('');

    return { usuario, novoNome };
  }
};
</script>
```

8.3 Modularização do Pinia

Ao trabalhar em projetos maiores, é recomendado modularizar as stores, separando cada **contexto** em arquivos distintos.

📌 Exemplo de Modularização

Crie a pasta `stores/` e organize as stores:

```
stores/
├── usuarioStore.js
├── produtoStore.js
└── pedidoStore.js
```

No **produtoStore.js**, criamos outra store independente:

```
import { defineStore } from 'pinia';

export const useProdutoStore = defineStore('produto', {
  state: () => ({
    produtos: []
  }),
  actions: {
    adicionarProduto(produto) {
      this.produtos.push(produto);
    }
  }
});
```

Agora, podemos importar e usar as stores independentemente nos componentes Vue.js.

8.4 Gerenciamento de Métodos Assíncronos

O Pinia permite criar **actions assíncronas** para buscar dados de uma API.

📌 **Exemplo: Buscando produtos de uma API**

```
import { defineStore } from 'pinia';
import axios from 'axios';

export const useProdutoStore = defineStore('produto', {
  state: () => ({
    produtos: []
  }),
  actions: {
    async carregarProdutos() {
      try {
        const response = await axios.get('https://fakestoreapi.com/products');
        this.produtos = response.data;
      } catch (error) {
        console.error('Erro ao carregar produtos', error);
      }
    }
  }
});
```

Essa abordagem permite integrar APIs externas de forma simples.

8.5 Monitoramento com Vue DevTools

O **Vue DevTools** é uma extensão que permite inspecionar o estado do Vue.js e do Pinia. Ele facilita o **debugging** e o **monitoramento do estado global da aplicação**.

📌 **Passo 1: Instalar Vue DevTools**
Baixe e instale a extensão no Chrome ou Firefox:
🔗 Vue DevTools

📌 **Passo 2: Habilitar suporte ao Pinia**
Adicione esta configuração ao projeto:

```
import { createPinia } from 'pinia';

const pinia = createPinia();
pinia.use(({ store }) => {
  console.log(`Store ${store.$id} foi inicializada.`);
});
```

Agora, ao inspecionar o Vue DevTools, podemos visualizar todas as **stores** registradas no Pinia.

8.6 Boas Práticas no Desenvolvimento Vue.js

Princípios de Clean Code

✔ **DRY (Don't Repeat Yourself)** – Evite código duplicado.
✔ **KISS (Keep It Simple, Stupid)** – Mantenha o código o mais simples possível.
✔ **SOLID** – Princípios de design para tornar o código modular e escalável.

📌 **Exemplo de Código Ruim (Violando DRY)**

```
function calcularPrecoComImposto(preco) {
  return preco + (preco * 0.2);
}

function calcularPrecoComDesconto(preco) {
  return preco - (preco * 0.1);
}
```

📌 **Código Melhorado (Seguindo DRY)**

```
function calcularPreco(preco, taxa) {
  return preco + (preco * taxa);
}
```

8.7 Estruturação de Projetos Complexos

📌 **Padrão de estrutura recomendado para Vue.js:**

```
src/
|— components/
|— views/
|— stores/
|— services/
|— router/
|— assets/
|— main.js
```

Isso facilita a **escalabilidade** e **manutenção** do projeto.

8.8 Utilização de Ferramentas: ESLint e Prettier

ESLint

O **ESLint** ajuda a manter um código limpo e padronizado, evitando erros comuns.

📌 **Instalar ESLint no projeto:**

```
npm install eslint --save-dev
```

📌 **Criar um arquivo de configuração `.eslintrc.js`:**

```
module.exports = {
  extends: ['eslint:recommended', 'plugin:vue/vue3-recommended'],
  rules: {
    'vue/no-unused-vars': 'error'
  }
};
```

Atividade Extra - Beautifully Designed para Vue

Os acadêmicos deverão pesquisar e preparar uma apresentação sobre **Beautifully Designed para Vue**, explorando **melhores práticas para criar interfaces visuais atraentes**, incluindo:

✔ **Uso de bibliotecas como Vuetify e Bootstrap Vue**
✔ **Padrões de Design no Vue.js**
✔ **Animações e transições visuais**
✔ **Boas práticas na experiência do usuário (UX/UI)**

Essa atividade ajudará a entender **como criar interfaces elegantes e eficientes** no Vue.js. 🚀

Lista de Exercícios - Conceitos Avançados no Vue.js

Questões Teóricas (10 questões)

1. **O que é o Pinia e quais são suas principais vantagens em relação ao Vuex?**
2. **Explique os conceitos de `state`, `actions`, `mutations` e `getters` no Pinia. Como cada um deles é utilizado?**
3. **Qual a importância da modularização do Pinia? Como ela pode ser implementada em um projeto Vue.js?**
4. **Por que é importante gerenciar métodos assíncronos no Pinia? Dê um exemplo de como isso pode ser feito.**
5. **O que é o Vue DevTools e como ele pode ser utilizado para monitorar o estado global do Pinia?**
6. **Explique os princípios de Clean Code: DRY, KISS e SOLID. Como aplicá-los no desenvolvimento Vue.js?**
7. **Como deve ser a estrutura ideal de um projeto Vue.js grande e escalável? Liste as principais pastas e sua finalidade.**
8. **O que são ESLint e Prettier? Como essas ferramentas ajudam no desenvolvimento Vue.js?**
9. **Por que é importante padronizar código em projetos colaborativos? Como ferramentas como Prettier auxiliam nisso?**
10. **O que é a Composition API e como ela se relaciona com o Pinia no Vue.js?**

Questões Práticas (10 questões)

1. **Instale e configure o Pinia em um projeto Vue.js e crie uma store chamada `authStore` para gerenciar autenticação do usuário (armazenando `token` e `usuario`).**

2. **Crie uma store chamada `produtoStore` e implemente um `state` que armazene uma lista de produtos e uma `action` que busque esses produtos de uma API externa.**
3. **No Vue.js, crie um componente `ListaProdutos.vue` que consuma os dados da `produtoStore` e exiba os produtos em uma lista.**
4. **Crie uma funcionalidade no Pinia que permita adicionar e remover produtos de um carrinho de compras (`carrinhoStore`).**
5. **Implemente um `getter` dentro da `carrinhoStore` que calcule o valor total da compra com base nos produtos adicionados.**
6. **Habilite o Vue DevTools no seu projeto Vue.js e visualize os estados das stores do Pinia em tempo real.**
7. **Utilizando ESLint e Prettier, configure o projeto para seguir um padrão de código. Corrija erros e estilize automaticamente o código.**
8. **Ajuste a estrutura do seu projeto Vue.js para seguir boas práticas de organização, modularizando components, stores e serviços.**
9. **Implemente um middleware de autenticação no Pinia, garantindo que certas rotas da aplicação só sejam acessadas por usuários logados.**
10. **Crie uma pequena apresentação sobre Beautifully Designed para Vue, explicando como criar interfaces visuais atraentes com boas práticas de design no Vue.js.**

- *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