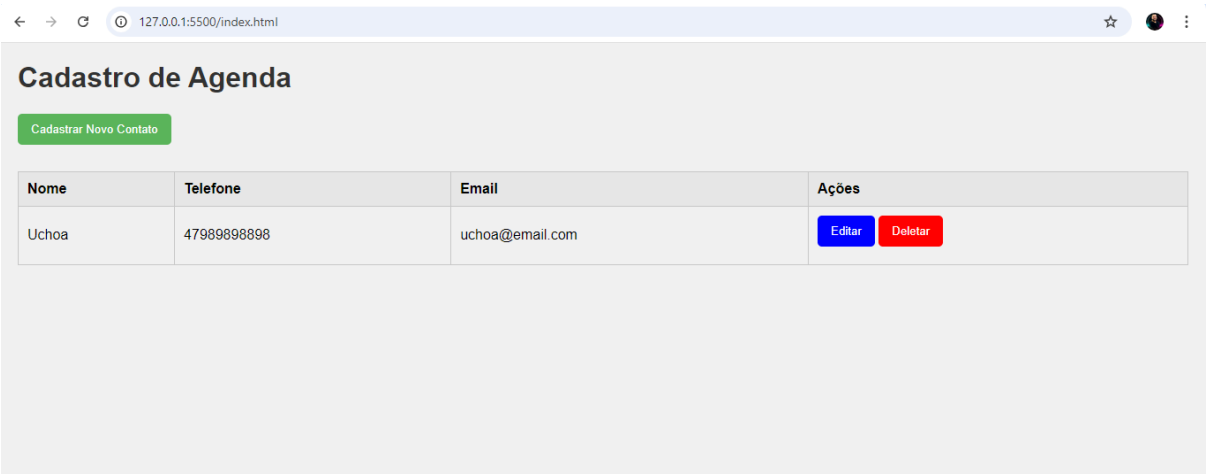
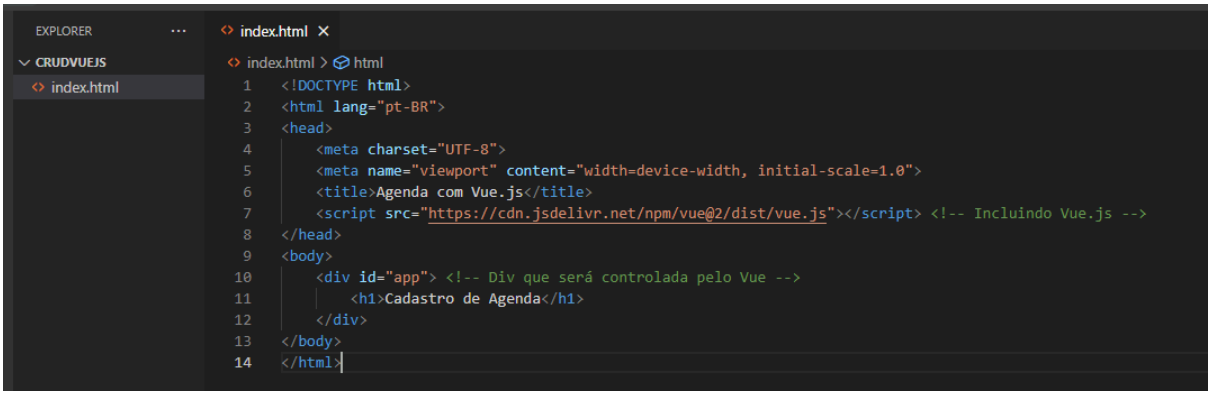
Prática: Crud com VueJS 2



Neste projeto iremos trabalhar as principais funcionalidades apresentadas nas OTs com isso aprimorar nossos conhecimentos com o framework Vue.

Passo 1: Criando a estrutura básica:



`<!DOCTYPE html>`: Declaração do tipo de documento HTML5.

`<html lang="pt-BR">`: Define o idioma do documento como português do Brasil.

`<meta charset="UTF-8">`: Define a codificação de caracteres como UTF-8.

`<meta name="viewport" content="width=device-width, initial-scale=1.0">`: Faz com que o layout se adapte ao tamanho da tela.

`<script src="...">`: Inclui o Vue.js a partir de uma CDN (Content Delivery Network).

`<div id="app">`: A div com id "app" é o ponto de montagem do Vue.

Passo 2: Criando a estrutura básica da Agenda (CRUD)

```
<div id="app">
    <h1>Cadastro de Agenda</h1>

    <form @submit.prevent="addContact"> <!-- Formulário para adicionar contatos -->
        <input type="text" v-model="newContact.name" placeholder="Nome" required> <!-- Campo de entrada para nome -->
        <input type="text" v-model="newContact.phone" placeholder="Telefone" required> <!-- Campo de entrada para telefone -->
        <input type="email" v-model="newContact.email" placeholder="Email" required> <!-- Campo de entrada para email -->
        <button type="submit">Adicionar</button> <!-- Botão para adicionar contato -->
    </form>

    <table>
        <thead>
            <tr>
                <th>Nome</th>
                <th>Telefone</th>
                <th>Email</th>
                <th>Ações</th>
            </tr>
        </thead>
        <tbody>
            <!-- Os contatos serão listados aqui -->
        </tbody>
    </table>
</div>
```

`<form @submit.prevent="addContact">`: O evento `@submit.prevent` evita o envio padrão do formulário e chama o método `addContact` quando o formulário é submetido.

`v-model="newContact.name"`: Liga o valor do campo de entrada à propriedade `name` do objeto `newContact` no Vue.

`<table>`: Estrutura básica de uma tabela com cabeçalhos para as colunas.

Passo 03: Integrando Vue.js e criando a lógica de adição de contatos

```
<script>
    new Vue({
        el: '#app', // Monta a instância do Vue no elemento com id "app"
        data: {
            newContact: { // Objeto para armazenar os dados do novo contato
                name: '',
                phone: '',
                email: ''
            },
            contacts: [] // Array que armazenará todos os contatos
        },
        methods: {
            addContact() { // Método para adicionar um novo contato
                if (this.newContact.name && this.newContact.phone && this.newContact.email) { // Verifica se todos os campos estão preenchidos
                    this.contacts.push({ ...this.newContact }); // Adiciona o novo contato ao array
                    this.newContact = { name: '', phone: '', email: '' }; // Limpa os campos após adicionar
                }
            }
        }
    });
</script>
```

`new Vue({ ... })`: Cria uma nova instância do Vue.

`el: '#app'`: Indica onde a instância do Vue será montada.

`data`: Contém as propriedades reativas do Vue.

`methods`: Contém os métodos que podem ser chamados na aplicação.

Passo 04:Listando contatos na tabela

```
<tbody>
    <tr v-for="(contact, index) in contacts" :key="index"> <!-- Itera sobre os contatos -->
        <td>{{ contact.name }}</td> <!-- Exibe o nome do contato -->
        <td>{{ contact.phone }}</td> <!-- Exibe o telefone do contato -->
        <td>{{ contact.email }}</td> <!-- Exibe o email do contato -->
        <td>
            <button @click="editContact(index)">Editar</button> <!-- Botão para editar o contato -->
            <button @click="deleteContact(index)">Deletar</button> <!-- Botão para deletar o contato -->
        </td>
    </tr>
</tbody>
```

`v-for="(contact, index) in contacts"`: Itera sobre o array `contacts`, criando uma linha da tabela para cada contato.

`:key="index"`: A chave única para cada item na lista, melhora o desempenho da renderização.

`{{ contact.name }}`: Usado para exibir o nome do contato.

Passo 05: Implementando a edição de contatos

```
    methods: {
        addContact() { // Método para adicionar um novo contato
            if (this.newContact.name && this.newContact.phone && this.newContact.email) { // Verifica se todos os campos estão preenchidos
                this.contacts.push({ ...this.newContact }); // Adiciona o novo contato ao array
                this.newContact = { name: '', phone: '', email: '' }; // Limpa os campos após adicionar
            }
        },
        editContact(index) { // Método para preencher os campos de edição
            this.currentContact = { ...this.contacts[index] }; // Preenche os campos com dados do contato selecionado
            this.editIndex = index; // Armazena o índice do contato a ser editado
        },
        updateContact() { // Método para atualizar o contato
            this.$set(this.contacts, this.editIndex, this.currentContact); // Atualiza o contato no array
            this.currentContact = { name: '', phone: '', email: '' }; // Limpa os campos após a atualização
            this.editIndex = -1; // Reseta o índice de edição
        }
    }
});
```

`this.currentContact`: Objeto que armazenará os dados do contato sendo editado.

`this.$set(...)`: Método do Vue que garante a reatividade ao atualizar um item específico em um array.

Passo 06: Implementando a Exclusão de contatos

```
    methods: {
        addContact() { // Método para adicionar um novo contato
            if (this.newContact.name && this.newContact.phone && this.newContact.em
                this.contacts.push({ ...this.newContact }); // Adiciona o novo cont
                this.newContact = { name: '', phone: '', email: '' }; // Limpa os c
            }
        },
        editContact(index) { // Método para preencher os campos de edição
            this.currentContact = { ...this.contacts[index] }; // Preenche os campo
            this.editIndex = index; // Armazena o índice do contato a ser editado
        },
        updateContact() { // Método para atualizar o contato
            this.$set(this.contacts, this.editIndex, this.currentContact); // Atual
            this.currentContact = { name: '', phone: '', email: '' }; // Limpa os c
            this.editIndex = -1; // Reseta o índice de edição
        },
        deleteContact(index) { // Método para remover um contato
        this.contacts.splice(index, 1); // Remove o contato do array pelo índice
        }
    }
});
</script>
```

`splice(index, 1)`: Remove um item do array na posição especificada.

Nosso projeto está ficando com uma cara boa, agora vamos estilizar o mesmo?

Passo 07: Melhorando a UI e a estilização

```
    </script>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            background-color: #f4f4f4;
            }
        h1 {
            color: #333;
        }
        button {
            padding: 10px 15px;
            border: none;
            border-radius: 5px;
            background-color: #5cb85c;
            color: white;
            cursor: pointer;
            margin-bottom: 10px;
        }
    </style>

</body>
</html>
```

Vamos estilizar os botões para isso criamos duas classes e colocamos os seguintes css:

```
    <button class="edit" @click=
    <button class="delete" @clic
</td>
```

```
    .edit{
        background-color: blue;
        color: white;
    }

    .delete{
        background-color: red;
        color: white;
    }
</style>

/body>
/html>
```

Ao testarmos a aplicação observamos que o editar não está funcionando.

Precisamos ajustar os seguintes locais:

```
<form @submit.prevent="editIndex === -1 ? addContact() : updateContact()"> <!-- Condição para adicionar ou atualizar -->
    <input type="text" v-model="currentContact.name" placeholder="Nome" required> <!-- Ligado a currentContact -->
    <input type="text" v-model="currentContact.phone" placeholder="Telefone" required> <!-- Ligado a currentContact -->
    <input type="email" v-model="currentContact.email" placeholder="Email" required> <!-- Ligado a currentContact -->
    <button type="submit">{{ editIndex === -1 ? 'Adicionar' : 'Atualizar' }}</button> <!-- Texto dinâmico -->
</form>
```

```
<script>
    new Vue({
        el: '#app', // Monta a instância do Vue no elemento com id "app"
        data: {
            currentContact: { // Objeto para armazenar os dados do contato em edição
                name: '',
                phone: '',
                email: ''
            },
            contacts: [], // Array que armazenará todos os contatos
            editIndex: -1 // Índice para saber se estamos editando um contato
        },
```

```
        methods: {
            addContact() { // Método para adicionar um novo contato
                if (this.currentContact.name && this.currentContact.phone && this.currentContact.email) {
                    this.contacts.push({ ...this.currentContact }); // Adiciona o novo contato ao array
                    this.resetForm(); // Limpa os campos após adicionar
                }
            },
            editContact(index) { // Método para preencher os campos de edição
                this.currentContact = { ...this.contacts[index] }; // Preenche os campos com dados do contato selecionado
                this.editIndex = index; // Armazena o índice do contato a ser editado
            },
            updateContact() { // Método para atualizar o contato
                this.$set(this.contacts, this.editIndex, this.currentContact); // Atualiza o contato no array
                this.resetForm(); // Limpa os campos após a atualização
                this.editIndex = -1; // Reseta o índice de edição
            },
            deleteContact(index) { // Método para remover um contato
                this.contacts.splice(index, 1); // Remove o contato do array pelo índice
            },
            resetForm() { // Método para limpar os campos
                this.currentContact = { name: '', phone: '', email: '' }; // Reseta o objeto de contato
            }
        }
    });
```

Agora ao testarmos temos todo o CRUD funcionando e o editar trazendo a alteração de valores e de botões corretamente. Foi preciso criarmos uma **função para resetar** o formulário e invocamos ela após o update para limpar os campos do formulário e também ao adicionar um usuário novo.

Principais Modificações

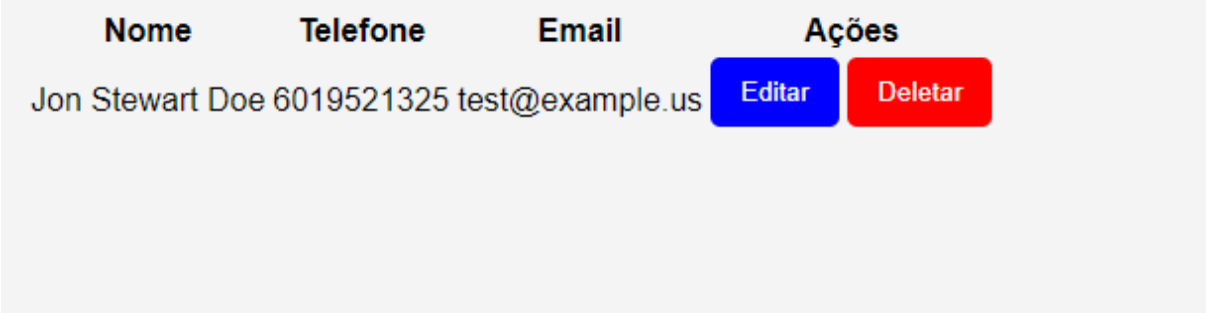
1. **Uso de `currentContact`**:
    - Agora o formulário está vinculado a `currentContact`, permitindo que os campos sejam preenchidos corretamente quando um contato é editado.
2. **Condição no Botão do Formulário**:
    - O botão do formulário muda entre "Adicionar" e "Atualizar" com base no valor de `editIndex`.
3. **Método `resetForm()`**:
    - Esse método limpa os campos após adicionar ou atualizar um contato.

4. **Lógica no `@submit`**:
    - A lógica para decidir entre adicionar ou atualizar um contato foi incorporada diretamente no evento `@submit` do formulário.

Teste

Com essas correções, o recurso de edição deve funcionar corretamente. Você pode testar adicionando contatos e depois clicando em "Editar" para verificar se os campos do formulário são preenchidos com as informações do contato selecionado. Após editar, o botão deve mudar para "Atualizar" e, ao clicar, o contato deve ser atualizado na tabela.

Agora para finalizar vamos dar uma estilizada na tabela e nos inputs?



Basta acrescentar este css que tudo ficará perfeitamente ajustado:

```css
    input {
        padding: 10px;
        margin-right: 10px;
        border: 1px solid #ccc;
        border-radius: 5px;
    }
    table {
        width: 100%;
        border-collapse: collapse;
        margin-top: 20px;
    }
    th, td {
        padding: 10px;
        border: 1px solid #ccc;
        text-align: left;
    }
    th {
        background-color: #e7e7e7;
    }
    tr:hover {
        background-color: #f1f1f1;
    }
</style>
```

Apresentando a versão finalizada: