

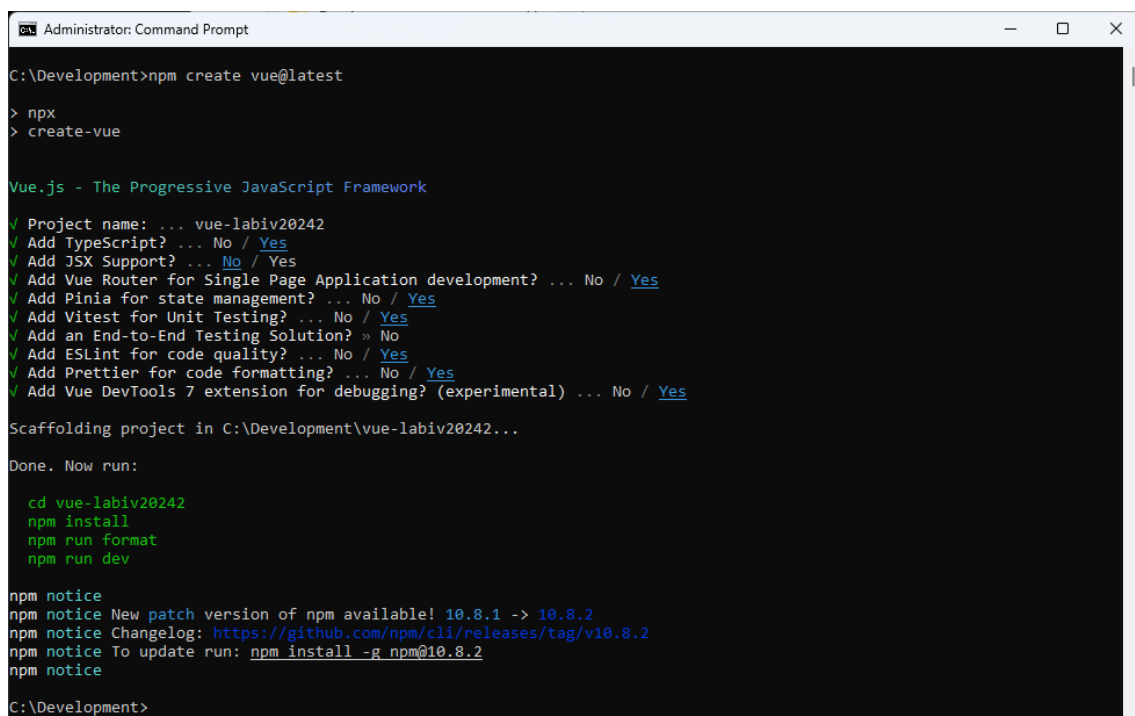
Criando *front-end* com Vue.js – Parte 1

1 Criando o projeto

Iremos utilizar o “npm” (Node Package Manager) para gerenciar nosso projeto. Para utilizá-lo, é necessário instalar o Node.js (<https://nodejs.org/en/>). Para Linux Debian/Ubuntu consulte: <https://deb.nodesource.com/>.

O comando para criação de um novo projeto Vue.js 3 é “npm create vue@latest”, conforme Figura 1. Isso iniciará um processo iterativo, onde a primeira pergunta é o nome do projeto. Para selecionar as opções use as setas e confirme com Enter.

Figura 1 - Criação do Projeto



```
Administrator: Command Prompt
C:\Development>npm create vue@latest

> npx
> create-vue

Vue.js - The Progressive JavaScript Framework

✓ Project name: ... vue-labiv20242
✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit Testing? ... No / Yes
✓ Add an End-to-End Testing Solution? » No
✓ Add ESLint for code quality? ... No / Yes
✓ Add Prettier for code formatting? ... No / Yes
✓ Add Vue DevTools 7 extension for debugging? (experimental) ... No / Yes

Scaffolding project in C:\Development\vue-labiv20242...

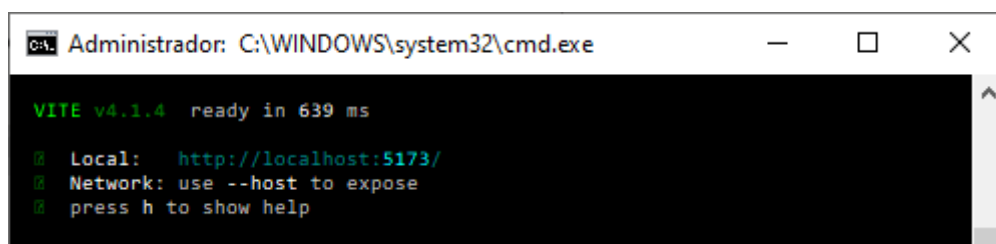
Done. Now run:

  cd vue-labiv20242
  npm install
  npm run format
  npm run dev

npm notice
npm notice New patch version of npm available! 10.8.1 -> 10.8.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.8.2
npm notice To update run: npm install -g npm@10.8.2
npm notice
C:\Development>
```

Ao digitar os comandos sugeridos ao final da criação, as dependências serão baixadas (npm install), o código formatado (npm run format) e a aplicação executada em modo de desenvolvimento (npm run dev). Uma porta disponível será escolhida e a aparência da aplicação varia de acordo com as bibliotecas escolhidas durante a criação.

Figura 2 - Execução



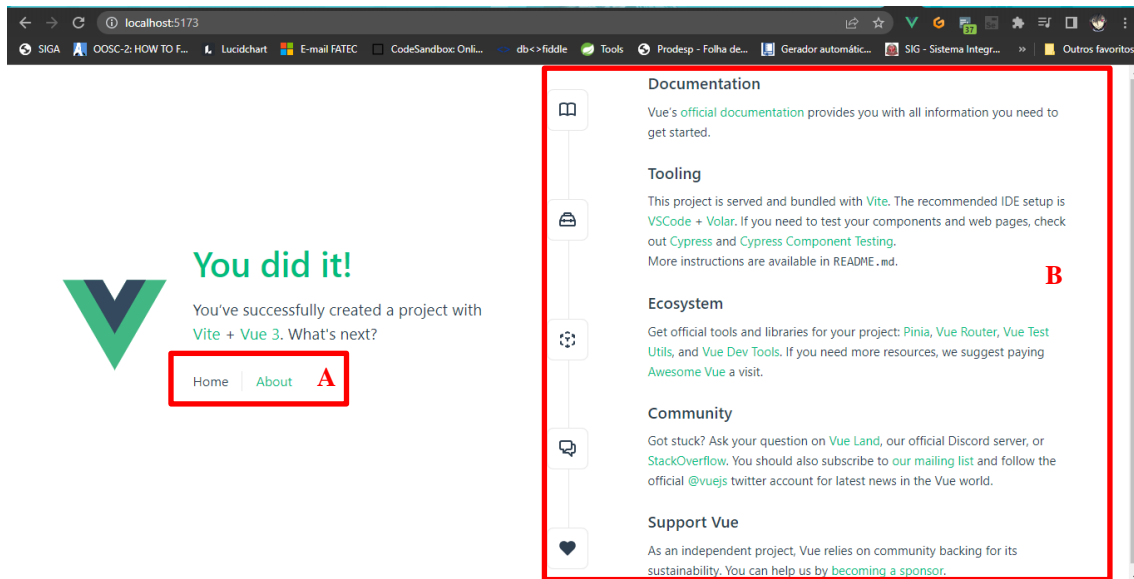
```
Administrator: C:\WINDOWS\system32\cmd.exe

VITE v4.1.4 ready in 639 ms

  Local:   http://localhost:5173/
  Network: use --host to expose
  press h to show help
```

A Figura 3 apresenta uma aplicação criada com as opções mostradas na Figura 1.

Figura 3 - Aplicação Vue.js



Principais arquivos da aplicação:

- package.json – Contém a definição dos scripts de execução (dev, build, format, etc) e as dependências do projeto. Alterado automaticamente quando da instalação ou desinstalação de dependência com o comando "npm";
- index.html – Página principal (e única) da aplicação. Possui uma "div" com id "app" e importa o arquivo "main.ts";
- src/main.ts – Arquivo principal da aplicação. Instancia o Vue.js em conjunto com os principais componentes (Pinia, Router, etc) e associa à "div" contida no arquivo "index.html", por meio do id ("app");
- src/App.vue – Layout principal da aplicação. Possui uma barra de navegação (Figura 3.A) com links internos, definidos pela tag <RouterLink>, e uma área onde as "páginas" são carregadas (Figura 3.B), definida pela tag <RouterView>;
- src/router/index.ts – Arquivo onde são definidos os links internos da aplicação. Cada URL (rota) interna é associada a um componente (arquivo com extensão "vue"). O conteúdo dos componentes é aberto na área demarcada pela tag <RouterView> (Figura 3.B). O componente pode ser importado ao início do arquivo ou durante a definição da rota. Componentes importados no início do arquivo serão baixados assim que a aplicação for aberta, enquanto componentes importados na definição da rota somente serão baixados quando o usuário navegar para a URL associada.

2 Algumas diretivas do Vue.js

Para iniciar, vamos alterar o arquivo "src/views/AboutView.vue", de acordo com o apresentado em Código 1. Um arquivo Vue.js representa um componente e é composto por três partes: template, script e style.

Em template, linhas de 1 a 5, colocamos a parte HTML do componente, os elementos visíveis. Aqui exibimos dinamicamente, na linha 3, o conteúdo da variável "nome", com a construção "{{ nome }}".

Em script, linhas de 7 a 11, o código JavaScript ou TypeScript responsável por gerenciar o funcionamento do componente é inserido. Neste exemplo, nós apenas usamos "ref", importado na linha 8, para declarar um atributo reativo "nome", linha 10. Ao alterar qualquer atributo declarado por meio de "ref", qualquer elemento associado em template será automaticamente atualizado.

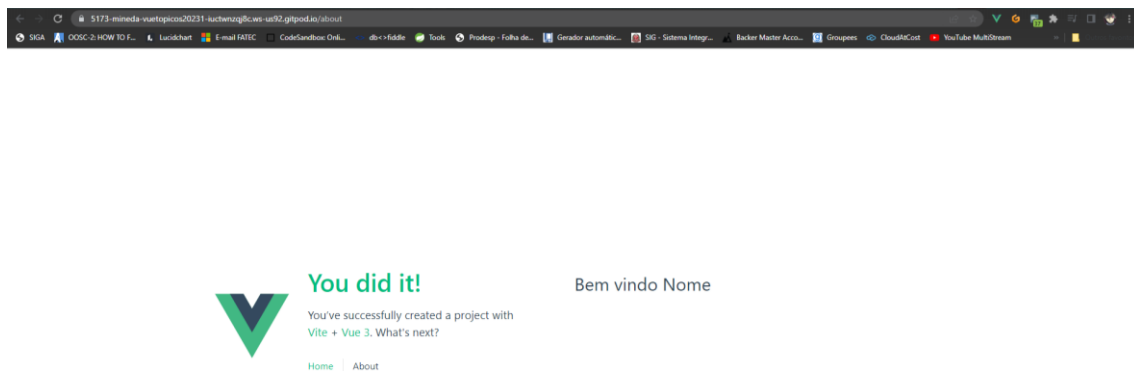
Em style podemos definir estilos css para uso no componente. Esta seção é opcional e deve sempre ser colocada após as outras duas.

Código 1 – Arquivo "AboutView.vue"

```
1 <template>
2   <div class="about">
3     <h1>Bem vindo {{ nome }}</h1>
4   </div>
5 </template>
6
7 <script setup lang="ts">
8   import { ref } from 'vue';
9
10  const nome = ref("Nome");
11 </script>
```

A Figura 4 apresenta o resultado final, onde {{ nome }} é substituído pelo conteúdo do atributo.

Figura 4 - Tela AboutView.vue



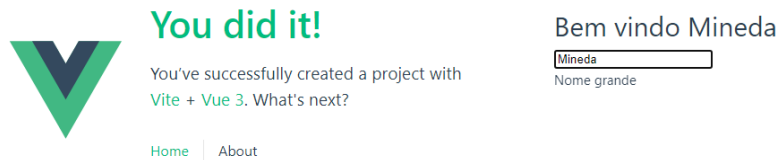
Vamos agora incluir novos elementos em template, de acordo com Código 2. Aqui nós incluímos um input, linha 4, e o associamos ao atributo "nome", por meio da diretiva "v-model". Ao alterar o conteúdo do input, o valor do atributo é atualizado automaticamente. Também incluímos dois parágrafos, linhas 5 e 6, renderizados com base na condição apresentada na diretiva "v-if". O parágrafo da linha 5 somente é renderizado se o tamanho do texto contido em "nome" for superior a 5. O parágrafo da linha 6 somente é renderizado quando a condição do "v-if" é falsa, por conta da diretiva "v-else".

Código 2 – Diretivas "v-model", "v-if" e "v-else"

```
1 <template>
2   <div class="about">
3     <h1>Bem vindo {{ nome }}</h1>
4     <input type="text" v-model="nome" />
5     <p v-if="nome.length > 5">Nome grande</p>
6     <p v-else>Nome pequeno</p>
7   </div>
8 </template>
9
10 <script setup lang="ts">
11   import { ref } from 'vue';
12
13   const nome = ref("Nome");
14 </script>
```

A Figura 5 apresenta o resultado de nossas alterações. O texto apresentado após "Bem vindo" reage automaticamente a qualquer alteração no input. O texto abaixo do input, por outro lado, somente é modificado quando o tamanho do texto digitado ultrapassa 5 caracteres.

Figura 5 - Input e parágrafos condicionais



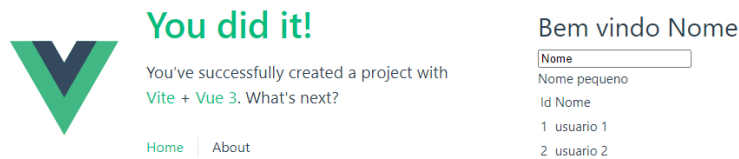
Agora veremos como lidar com listas, utilizando a diretiva "v-for". No Código 3, declaramos um atributo "usuarios", linhas de 24 a 27, contendo uma lista de objetos com dois elementos. No template mostramos a lista em uma tabela, linhas de 7 a 16. Cada linha da tabela (tag "tr") é criada automaticamente por meio da diretiva "v-for", linha 12. Cada elemento da lista é carregado em uma variável temporária "usuario" e seus atributos podem ser apresentados, conforme linhas 13 e 14. Cada linha precisa de um identificador, configurado no argumento "key", e utilizamos o id do usuário ("usuario.id"). Para carregarmos uma variável do Vue.js no argumento "key", estamos utilizando a diretiva "v-bind" em sua forma abreviada, simbolizada por ":" ("::key" é equivalente a "v-bind:key").

Código 3 – Diretiva "v-for"

```
1 <template>
2   <div class="about">
3     <h1>Bem vindo {{ nome }}</h1>
4     <input type="text" v-model="nome" />
5     <p v-if="nome.length > 5">Nome grande</p>
6     <p v-else>Nome pequeno</p>
7     <table>
8       <thead>
9         <tr>
10          <td>Id</td>
11          <td>Nome</td>
12        </tr>
13      <tbody>
14        <tr v-for="usuario in usuarios" :key="usuario.id">
15          <td>{{ usuario.id }}</td>
16          <td>{{ usuario.nome }}</td>
17        </tr>
18      </tbody>
19    </table>
20  </div>
21 </template>
22
23 <script setup lang="ts">
24   import { ref } from 'vue';
25
26   const nome = ref("Nome");
27   const usuarios = ref([
28     { id: 1, nome: "usuario 1", senha: "123" },
29     { id: 2, nome: "usuario 2", senha: "456" }
30   ]);
31 </script>
```

A Figura 6 apresenta o resultado final, com a tabela contendo os ids e nomes de todos os usuários.

Figura 6 - Tabela

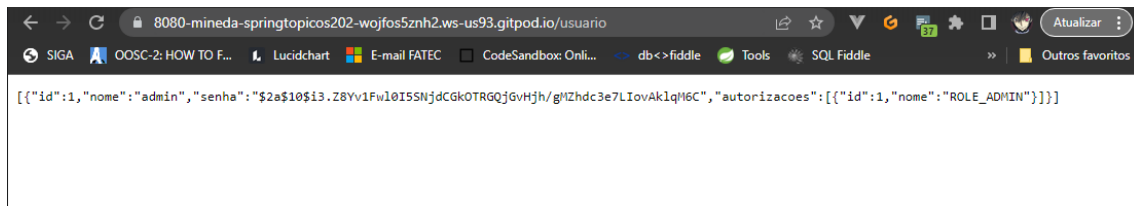


3 Recuperando dados da aplicação back-end

Para realizar a comunicação com a aplicação back-end (veja "Guia Back End com Spring") utilizaremos a biblioteca "axios", que facilita realizar automaticamente a conversão objeto/JSON. A instalação é realizada por meio do comando "npm i axios", executado na raiz do projeto.

A aplicação back-end utilizada possui um serviço rodando na URI "/usuario" com cadastro no método POST e consulta de todos os registros no método GET. A Figura 7 apresenta o resultado de uma requisição GET ao serviço e podemos ver que o objeto retornado é compatível com nossa lista existente de usuários.

Figura 7 - Requisição GET ao serviço "/usuario"



O Código 4 apresenta uma forma de utilizarmos o axios para recuperar a lista de usuários a partir da aplicação back-end. Para se realizar uma requisição, precisamos inicialmente importar o axios (linha 3). Deixaremos a lista "usuarios" (linha 6) em branco, para que possamos verificar o resultado. Para recuperar os resultados, criaremos uma função assíncrona "atualizar" (linhas de 8 a 10) que cuidam de realizar a chamada e carregar os resultados na variável "usuarios". Alguns pontos importantes:

- Para se alterar o valor de uma variável via código, precisamos acessar o atributo "value", como em "usuarios.value" (linha 9);
- A requisição GET com o axios é realizada por meio da função "axios.get" que recebe como parâmetro a URL do serviço (substitua pela URL correspondente do seu serviço);
- O retorno de uma requisição contém vários elementos. Desejamos apenas os dados (lista de usuários), portanto precisamos acessar o atributo "data", em "(await axios.get("https://8080-mineda.gitpod.io/usuario")).data". Reparem que, apesar do retorno do serviço ser originalmente um JSON, podemos carregá-lo diretamente em uma variável (a conversão é realizada automaticamente);
- Precisamos utilizar o operador "await" na linha 9 para aguardar o retorno da chamada ao serviço. Por padrão, ao realizar qualquer requisição com o axios (uma Promise) o código seguiria adiante, sem esperar pelo retorno. O operador "await" pausa a execução, esperando o resultado e permitindo seu carregamento em uma variável. Esse operador somente funciona em funções assíncronas e, por esse motivo, adicionamos a palavra "async" na definição da função (linha 8);
- "onMounted" é um "hook" do Vue.js, executado assim que todos os elementos do componente e seus componentes filhos foram criados e inseridos no DOM. Neste exemplo (linhas de 12 a 14),

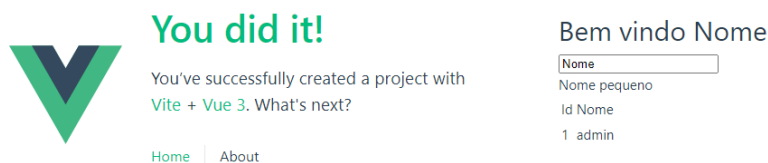
executamos a função "atualizar" nele, o que resultará no preenchimento da lista durante a abertura da página. Para utilizar o "onMounted", precisamos importá-lo (linha 2).

Código 4 - Uso do axios

```
1 <script setup lang="ts">
2 import { onMounted, ref } from 'vue';
3 import axios from 'axios';
4
5 const nome = ref("Nome");
6 const usuarios = ref();
7
8 async function atualizar() {
9   usuarios.value = (await axios.get("https://8080-mineda.gitpod.io/usuario")).data;
10 }
11
12 onMounted(() => {
13   atualizar();
14 });
15 </script>
```

A Figura 8 apresenta o resultado final, com a tabela contendo o usuário "admin".

Figura 8 - Lista preenchida com requisição à aplicação back-end



Como próximo passo, vamos cadastrar um novo usuário, utilizando o método POST do serviço. O Código 5, apresenta o novo template de nosso componente. Foram incluídos:

- Novo componente HTML "input" (linha 7), do tipo "password", ligado à variável "senha" por meio da diretiva "v-model";
- Um componente HTML "button" (linha 8) com label "Cadastrar". Aqui aparece uma nova diretiva do Vue.js, a "v-on", em sua forma abreviada "@". Essa diretiva permite capturar um "evento" de um componente e executar uma ação. Nesse caso, estamos capturando o evento de clique no botão ("@click") e chamando um método "cadastrar", definido em Código 6.

Código 5 – Template do componente

```
1 <template>
2   <div class="about">
3     <h1>Bem vindo {{ nome }}</h1>
4     <label for="nome">Nome: </label>
5     <input id="nome" type="text" v-model="nome" />
6     <label for="senha">Senha: </label>
7     <input id="senha" type="password" v-model="senha" />
8     <button @click="cadastrar">Cadastrar</button>
9     <table>
10      <thead>
11        <td>Id</td>
12        <td>Nome</td>
13      </thead>
14      <tr v-for="usuario in usuarios" :key="usuario.id">
15        <td>{{ usuario.id }}</td>
16        <td>{{ usuario.nome }}</td>
17      </tr>
18    </table>
19  </div>
20 </template>
```

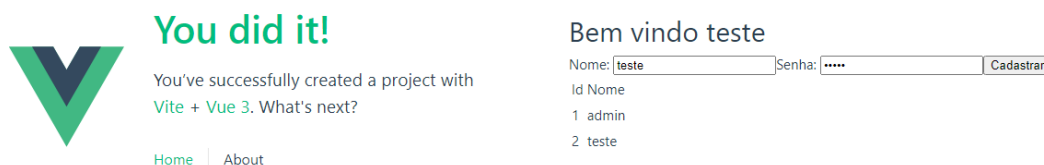
No Código 6 declaramos a variável "senha" (linha 6) e depois criamos o método "cadastrar", que realiza uma requisição POST com "axios.post". O método POST do serviço espera um JSON com a estrutura de um usuário, contendo nome e senha e, por esse motivo, passamos como segundo parâmetro (o primeiro continua sendo a URL do serviço) um novo objeto contendo esses dois atributos, cujos valores são preenchidos com os valores das variáveis locais de mesmo nome ("nome.value" e "senha.value"). Utilizamos o operador "await" para aguardar a resposta da requisição e, somente após sua conclusão, chamamos o método "atualizar" para recarregar a lista.

Código 6 – Script do componente

```
1 <script setup lang="ts">
2 import { onMounted, ref } from 'vue';
3 import axios from 'axios';
4
5 const nome = ref("Nome");
6 const senha = ref("123");
7 const usuarios = ref();
8
9 async function atualizar() {
10   usuarios.value = (await axios.get("https://8080-mineda.gitpod.io/usuario")).data;
11 }
12
13 async function cadastrar() {
14   await axios.post("https://8080-mineda.gitpod.io/usuario",
15     {
16       nome: nome.value,
17       senha: senha.value
18     });
19   atualizar();
20 }
21
22 onMounted(() => {
23   atualizar();
24 });
25 </script>
```

A Figura 9 apresenta o resultado final, contendo uma tela que permite cadastro e listagem de usuários.

Figura 9 – Tela final



Informar a URL completa em cada requisição não é ideal, visto que qualquer alteração no host da aplicação back-end resultariam em alterações em diversos lugares. Para evitar esse problema, podemos configurar o endereço base de nosso serviço no arquivo "main.ts", conforme Código 7 (linha 7).

Código 7 – Configurando URL base

```
1 import { createApp } from 'vue'
2 import { createPinia } from 'pinia'
3 import App from './App.vue'
4 import router from './router'
5 import axios from 'axios'
6
7 axios.defaults.baseURL = 'https://8080-mineda.gitpod.io/'
8
9 import './assets/main.css'
10
11 const app = createApp(App)
12 app.use(createPinia())
13 app.use(router)
14 app.mount('#app')
```

A partir dessa configuração, precisamos informar apenas qual serviço estamos acessando (no nosso exemplo "usuario"), conforme Código 8.

Código 8 – Informando apenas o endereço relativo do serviço

```
1  async function atualizar() {
2    usuarios.value = (await axios.get("usuario")).data;
3  }
4
5  async function cadastrar() {
6    await axios.post("usuario",
7      {
8        nome: nome.value,
9        senha: senha.value
10     });
11    atualizar();
```

Finalmente, nós podemos realizar um tratamento de erro, utilizando try/catch, conforme Código 9. A exceção é carregada na variável "ex". Para acessar seu conteúdo, precisamos indicar seu tipo usando casting para o tipo Error ("ex as Error").

Código 9 – Tratamento de erros

```
1  <template>
2    <div class="about">
3      <h1>Bem vindo {{ nome }}</h1>
4      <label for="nome">Nome: </label>
5      <input id="nome" type="text" v-model="nome" />
6      <label for="senha">Senha: </label>
7      <input id="senha" type="password" v-model="senha" />
8      <button @click="cadastrar">Cadastrar</button>
9      <p v-if="erro">{{ erro }}</p>
10     <table>
11       <thead>
12         <td>Id</td>
13         <td>Nome</td>
14       </thead>
15       <tr v-for="usuario in usuarios" :key="usuario.id">
16         <td>{{ usuario.id }}</td>
17         <td>{{ usuario.nome }}</td>
18       </tr>
19     </table>
20   </div>
21 </template>
22
23 <script setup lang="ts">
24 import { onMounted, ref } from 'vue';
25 import axios from 'axios';
26
27 const nome = ref("Nome");
28 const senha = ref("123");
29 const erro = ref();
30 const usuarios = ref();
31
32 async function atualizar() {
33   try {
34     usuarios.value = (await axios.get("usuario")).data;
35   }
36   catch(ex) {
37     erro.value = (ex as Error).message;
38   }
39 }
40
41 async function cadastrar() {
42   try {
43     await axios.post("usuario",
44       {
45         nome: nome.value,
46         senha: senha.value
47       });
48   }
49   catch(ex) {
50     erro.value = (ex as Error).message;
```



```
51     }  
52     atualizar();  
53 }  
54  
55 onMounted(() => {  
56     atualizar();  
57 });  
58 </script>
```

A Figura 10 apresenta o resultado de uma requisição POST com um usuário já existente (erro 500).

Figura 10 - Tratamento de erros

