Projeto Nest.js e Vue.js

Alunos: Augusto Castejon, Álvaro Dias, Luiz Felippe e Vinicius Dogonski

1. Fazendo a validação no backend

1.1. Agora vamos criar a funcionalidade "inserir nova chave" e também vamos validar os dados que vão ser inseridos, já que não será permitido inserir uma chave vazia nem com o mesmo nome de outra chave

1.2. No model de chave, já temos a seguinte configuração:

```
import * as mongoose from 'mongoose';

export const ChaveSchema = new mongoose.Schema({
nome : {
   type : String,
   unique : true,
   required: true
},
situacao : {
   type : String,
   required : true
},
status : Boolean

});

export interface Chave extends mongoose.Document {
nome: string;
situacao: string;
status: Boolean;
}
```

- O atributo **unique** é responsável por verificar se já existe alguma chave com o mesmo valor de atributo, caso exista ele irá disparar um erro, e o required vai verificar se o meu campo é obrigatório. Assim, a nível de backend já temos a validação feita.

2. Criando o método

2.1. Com a verificação já feita, vamos criar um método para inserir a chave. Em **chave.service.ts,** adicione o método**:**

```
import { NotFoundException } from '@nestjs/common';
```

```
   async criarChave(nome: string): Promise<Chave> {
       try {
           const chave = new this.chaveModel({ nome,
situacao: "DISPONIVEL", status: true });
           return await chave.save();
       } catch (Error) {
           console.log(Error.message)
           throw new NotFoundException(
               Error.message
           );
       }
   }
```

2.2. Em **chave.controller.ts,** adicione o seguinte código**:**

```
import { Post, Body } from '@nestjs/common';
```

```
   @Post()
   async criarChave(@Body() data: Chave): Promise<Chave> {
     console.log(data)
     return this.chaveService.criarChave(data.nome);
   }
```

Feito isso já temos o nosso backend pronto para receber do frontend os dados da chave a ser inserida no BD, e fazer essa tarefa com sua devida validação.

3. Incluindo a nova funcionalidade no frontend

3.1. Faça o download do ícone:
https://github.com/luiz-felippelb/vuejs-emprestimoChave-daw1/blob/main/src/assets/add.png
Adicione-o em **/emprestimo-chave/frontend/src/assets/**. Este será o ícone do botão para adicionar uma nova chave.

3.2. Em **/emprestimo-chave/frontend/src/components:**

→ mkdir models
→ cd models
→ touch AddKey.vue

3.3. Abra no VSCode o arquivo AddKey.vue e adicione as seguintes linhas de código:

```
<script setup lang="ts">
import { ref } from 'vue';
import axios from 'axios';

const props = defineProps({
  show: Boolean
})

const nomeChave = ref('');
const errorMessage = ref('')

const cadastrarChave = async () => {
  if (nomeChave.value.trim() !== '') {
      try {
      errorMessage.value = ''
      const response = await
axios.post('http://localhost:3000/chave', { nome:
nomeChave.value });

      // Lide com a resposta da API, por exemplo, exibindo
uma mensagem de sucesso
      console.log('Chave cadastrada com sucesso:',
response.data);

      // Limpar o campo de nome da chave após o cadastro
      nomeChave.value = '';
      } catch (error) {
      errorMessage.value = 'Chave já cadastrada!'
      console.error('Erro ao cadastrar a chave:', error);
      }
  }
};

</script>

<template>
  <Transition name="modal">
      <div v-if="show" class="modal-mask">
      <div class="modal-container">

      <div class="modal-header">
```

```
            <h3>Cadastro de Chave</h3>
      </div>

      <div class="modal-body">
            <form @submit.prevent="cadastrarChave">
            <label for="nomeChave">Nome da Chave:</label>
            <input type="text" id="nomeChave"
v-model="nomeChave" required>
            <p v-if="errorMessage != ''">{{ errorMessage
}}</p>
            <button id="button-form"
type="submit">Cadastrar</button>
            </form>
      </div>

      <button class="modal-default-button"
@click="$emit('close')"> X </button>
      </div>
      </div>
  </Transition>
</template>

<style scoped>
input {
  display: block;
  border: 1px solid black;
  margin-top: 0.3rem;
  padding: 0.5rem;
}

p {
  margin-top: 0.2rem;
  font-size: 0.9rem;
  color: red;
}

#button-form {
  font-size: 1rem;
  margin: auto;
  margin-top: 2rem;
  background-color: #42b983;
  border-radius: 2rem;
  padding: 0.5rem 1.5rem;
```

```
  cursor: pointer;
}

.modal-mask {
  position: fixed;
  z-index: 9998;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  transition: opacity 0.3s ease;
}

.modal-container {
  width: 400px;
  margin: auto;
  padding: 3rem;
  background-color: #fff;
  border-radius: 1rem;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.33);
  transition: all 0.3s ease;
}

.modal-header h3 {
  font-size: 1.5rem;
  text-align: center;
  color: #45A452;
}

.modal-body {
  margin-top: 2rem;
}

.modal-default-button {
  cursor: pointer;
  background-color: #B05E31;
  border-radius: 1rem;
  padding: 0.2rem 0.4rem;
  position: absolute;
  top: 15px;
  right: 15px;
```

```
}

.modal-enter-from {
  opacity: 0;
}

.modal-leave-to {
  opacity: 0;
}

.modal-enter-from .modal-container,
.modal-leave-to .modal-container {
  -webkit-transform: scale(1.1);
  transform: scale(1.1);
}
</style>
```

Acabamos de criar um componente que será responsável por exibir um modal na tela com o campo para inserir o nome da nova chave e o botão para cadastrar. A função **cadastrarChave** pega a string digitada no campo de input, retirando os espaços em branco e, caso esta string não seja vazia (aqui temos uma segunda validação, agora a nível de frontend) é feito uma solicitação http do tipo post, passando o nome da chave via URL: **http://localhost:3000/chave** para nosso backend.

3.4. Agora, para que nosso componente funcione de fato, vamos adicioná-lo ao nosso arquivo **Header.vue.** Onde nosso modal será apresentado quando o ícone de “adicionar nova chave” for clicado. Em **Header.vue,** vamos adicionar alguns trechos de código, faremos isso por partes, em suas respectivas tags.

3.4.1. Em **<script setup lang=”ts”></script>:**

```
import AddKey from './modals/AddKey.vue'
const showModal = ref(false)
```

3.4.2. Em **<template><header> … </header></template>:**

```
  <nav>
     <button @click="showModal = true"><img
src="../assets/add.png" alt="Add key icon"
width="60" height="60"></button>
   </nav>
```

3.4.3. Em **<template> … </template>:**

```
 <Teleport to="body">
   <AddKey :show="showModal" @close="showModal =
false"></AddKey>
 </Teleport>
```

3.4.4. Em **<style scoped> … </style>:**

```
nav {
 display: flex;
 padding-top: 3.5em;
 padding-right: 1rem;
}

button {
 margin-right: 1rem;
 cursor: pointer;
 background-color: transparent;
}
```

Aqui, o diferencial está na tag <Teleport> que é uma feature do Vue capaz de “ativar” nosso modal **AddKey.vue**.

Agora conseguimos adicionar quantas chaves for necessário e visualizá-las quando clicarmos no campo responsável por listar as chaves, que foi a primeira funcionalidade desenvolvida.