03

Comportamento do login e logout

Transcrição

Temos nosso componente de login operante, então já podemos redirecionar nosso usuário para esta tela quando ele se cadastrar na aplicação.

Em `NovoUsuario.vue`, em caso de sucesso, além de fazer o `console.log()` que faremos da `resposta`, queremos pedir que o view redirecione o usuário para a tela de login. O view deve acessar o `$router` e então o usuário deverá ser alocado para uma rota específica de `name: 'login'`.

```
//código anterior omitido

methods: {
        enviarFormulario() {
          axios
            .post("http://localhost:8000/auth/register",
this.usuario)
            .then(resposta => {
                console.log(resposta)
                this.$router.push({ name: 'login' })
            })
            .catch(erro => console.log(erro))
        }
```

COPIAR CÓDIGO

Salvaremos e voltaremos ao navegador para preenchermos os campos novamente. A clicarmos em "Logar" para nos cadastramos, veremos no

"Console" do inspetor de código que o `cosole.log()` da `resposta` foi feito, e fomos redirecionados para a tela de login.

Feito isso, já estamos prontos para implementar a lógica de login. Queremos que, quando o formulário for submetido, do mesmo jeito que fazemos no formulário de registro, não queremos que tenha um comportamento padrão. Então ao invés disso, queremos que ele chame o método `"efetuarLogin"` em `@submit.prevent`.

Vamos implementar o `efetuarLogin()` usando `methods:` em `Login.vue`. Para efetuarmos o login do usuário, precisaremos do `axios`, então vamos importá-lo também.

Depois, o configuraremos para fazer o `.post()` para a nossa url, depois enviar os dados do `usuario` com `this`, e por fim queremos que, em caso de sucesso, o `response` será `console.log()` recebendo a `response`. E, em caso de erro, pediremos que o `catch()` pegue o `error` de `console.log()` do `error`.

Salvaremos e rodaremos novamente para testarmos autenticando o usuário na aplicação no navegador.

```
<template>
    <div class="container">
        <h1>Login</h1>
        <form @submit.prevent="efetuarLogin">
            <div class="form.group">
                <label for="email">E-mail</label>
                <input type="email" class="form-control" v-
model="usuario.email">
            </div>
            <div class="form.group">
                <label for="senha">Senha</label>
                <input type="password" class="form-control"
v-model="usuario.senha">
            </div>
```

```
            <button type='submit' class="btn btn-primary
brn-block">
                Logar
            </button>
            <router-link :to="{ name: 'novo.usuario' }">
                Não possui um cadastro, cadastre-se aqui!
            </router-link>
        </form>
    </div>
</template>

<script>
import axios from 'axios'

export default {
    data () {
        return {
            usuario: {}
        }
    }
},
methods: {
    efetuarLogin () {
        axios.post('http://localhost:8000/auth/login',
this.usuario)
             .then(response => console.log(response))
             .catch(erro => console.log(erro))
    }

}
</script>
```

COPIAR CÓDIGO

Efetuado o login, veremos que está fazendo a requisição para o servidor, e conseguiremos notar, na linha de "Network", a requisição de login clicando em "Headers". No "Preview", teremos o retorno do token de acesso, o qual permitirá que façamos outras requisições que estejam protegidas no servido:

Resta armazenamos este token e seguir com o fluxo da aplicação.

Retornaremos ao método `efetuarLogin()`, que além de realizar o `console.log()` irá também possibilitar o armazenamento do token. Quanto ao armazenamento, existem algumas possibilidades de realizar essa ação, mas queremos garantir que mesmo que o usuário recarregue aplicação o token fiquei disposição, pois recarregar a aplicação não necessariamente o invalida.

Para tanto, faremos uso de um recurso chamado de *local storage*, que é o armazenamento que irá guardar o token, mesmo se o usuário recarregar a aplicação.

Para guardarmos um item, inseriremos `localStorage` em `.then` no arquivo `Login.vue`, e pediremos que guarde o `'token'` dentro de `.setItem()`. Dentro do nosso `methods:`, escreveremos:

```
//código anterior omitido

    methods: {
        efetuarLogin () {
            axios.post('http://localhost:8000/auth/login',
this.usuario)
                .then(response => {
                    console.log(response)
                    localStorage.setItem('token',
response.data.access_token)
                })
                .catch(erro => console.log(erro))
        }
    }
]
```

COPIAR CÓDIGO

Se confirmarmos no "Console" no navegador fazendo o login, veremos que estamos logando o `response`, então precisamos acessar `access_token`.

Portanto, queremos `response.data.access_token` em `setItem()` também.

Conseguimos acessar os dados o local storage diretamente no navegador. Ao inspecionarmos a página, clicaremos em "Application > Storage > Local Storage" e clicaremos no nosso endereço "http://localhost:8080" (http://localhost:8080").

Se tentarmos logar novamente inserindo os dados nos campos da página, notaremos que os dados ficarão armazenados na local storage. Mas se recarregarmos a aplicação de novo, ainda ficará disponível, e o usuário não precisará logar mais uma vez.

Uma vez que isso acontece, precisamos direcioná-lo para que possa seguir o fluxo normal da aplicação. Já sabemos como fazer, escrevendo `this.$router.push()` , pedindo que coloque o usuário na rota com o nome `'gerentes'` :

```
//código anterior omitido

    methods: {
        efetuarLogin () {
            axios.post('http://localhost:8000/auth/login',
this.usuario)
                .then(response => {
                    console.log(response)
                    localStorage.setItem('token',
response.data.access_token)
                    this.$router.push({ name: 'gerentes' })
                })
                .catch(erro => console.log(erro))
        }
    }
]
```

COPIAR CÓDIGO

De volta ao navegador, preencheremos os campos e clicaremos em "Logar" novamente, e seremos redirecionados para a tela de "Gerentes".

Sabemos que o login está operante, mas precisaremos criar uma maneira do usuário conseguir fazer o logout. Em `App.vue`, no local em que temos nossa barra, adicionaremos uma nova `<li>` que é o item de lista, a qual será de classe `"nav-item"`, e dentro teremos um `<a>` com `href` igual a `"#"`, e, da mesma forma que fizemos no formulário, pediremos que quando o `<a>` for clicado, não queremos que tenha o comportamento padrão de enviar o usuário para a outra página inserindo `@click.prevent igual a "efetuarLogout" ao invés disso. A classe do Bootstrap será "nav-link"`.

Com isso, o usuário irá clicar e não teremos o comportamento padrão, efetuando o logout de fato.

```
//código anterior omitido
<li class="nav-item">
    <a
     href="#"
     class="nav-link"
     @click.prevent="efetuarLogout">Logout</a>
</li>

//código posterior omitido
```

COPIAR CÓDIGO

Feito isso, implementaremos esse processo em uma tag `<script>` exportando um objeto padrão, implementando os métodos do componente `efetuarLogout()`, o qual fará o inverso do login, indo até o `localStorage` utilizando `removeItem()` com nome de `'token'`.

Já que estamos fazendo o logout, o redicionaremos para a página de login novamente com `this.$router.push()` para a rota de nome `'login'`.

```
//código anterior omitido

<script>
    export default {
        methods: {
            efetuarLogout() {
                localStorage.removeItem('token')
                this.$router.push({name: 'login'})
            }
        }
    }
</script>

//código posterior omitido
```

COPIAR CÓDIGO

Vamos testar o método `efetuarLogout()` na página do navegador. Então faz exatamente o contrário do processo que implementamos anteriormente, ao invés de armazenar informação no local storage, ele exclui essa informação. Por fim, o usuário será redirecionado para a página de login novamente.

Portanto, já implementamos os comportamentos de login e logout do usuário, e tudo opera como o esperado.