Criando *front-end* com Vue.js – Parte 2

1 Criando uma página de login

Vamos criar um novo arquivo chamado “LoginView.vue” em “src/views”, de acordo com Código 1. Essa página possui um dois inputs, usuário e senha, e chama uma rota de “login” no back-end, que retorna um token de acesso. Verifique autenticação com token JWT em “Guia Back End com Spring - Parte 4.pdf”.

Código 1 – Arquivo "LoginView.vue"

```
1  <template>
2    <div class="about">
3      <h1 id="hello">Usuários</h1>
4      <p>
5        <label for="nome" id="labelNome">Nome: </label>
6        <input id="nome" type="text" v-model="nome"/>
7      </p>
8      <p>
9        <label for="senha" id="labelSenha">Senha: </label>
10       <input id="senha" type="password" v-model="senha"/>
11     </p>
12     <button @click="login(nome, senha)">Login</button>
13     <p v-if="erro">{{ erro }}</p>
14     <p v-if="token">{{ token }}</p>
15   </div>
16 </template>
17
18 <script setup lang="ts">
19
20 import { ref } from 'vue';
21 import axios from 'axios';
22
23 const nome = ref<string>("Aluno");
24 const senha = ref<string>("123456");
25
26 const token = ref<string>();
27 const erro = ref<string>();
28
29 async function login(nome:string, senha:string) {
30   try {
31       let response = (await axios.post('login',
32               {
33                 nome: nome,
34                 senha: senha
35               }
36           ));
37       token.value = response.data.token;
38       erro.value = '';
39   }
40   catch(ex) {
41       erro.value = (ex as Error).message;
42   }
43 }
44 </script>
```

Para que essa página possa ser acessada, precisamos configurá-la como uma rota no arquivo “src/router/index.ts”, conforme Código 2, onde:

- path – Define a url relativa da página;
- name – Define um identificador para a página. Dentro do Vue.js podemos navegar utilizando tanto o “path” quanto o “name”;
- component – Define o arquivo que contém nossa página. Aqui realizamos a importação diretamente na definição da rota. Isso faz com que todos os arquivos associados a ela sejam baixados durante a primeira navegação.

Código 2 – Arquivo "router/index.ts"

```
1  import { createRouter, createWebHistory } from 'vue-router'
2  import HomeView from '../views/HomeView.vue'
3
4  const router = createRouter({
5    history: createWebHistory(import.meta.env.BASE_URL),
6    routes: [
7      {
8        path: '/',
9        name: 'home',
10       component: HomeView
11     },
12     {
13       path: '/about',
14       name: 'about',
15       // route level code-splitting
16       // this generates a separate chunk (About.[hash].js) for this route
17       // which is lazy-loaded when the route is visited.
18       component: () => import('../views/AboutView.vue')
19     },
20     {
21       path: '/login',
22       name: 'login',
23       component: () => import('../views/LoginView.vue')
24     }
25   ]
26 })
27
28 export default router
```

Finalmente, criamos um link para nossa página no arquivo “App.vue”, conforme Código 3, onde:

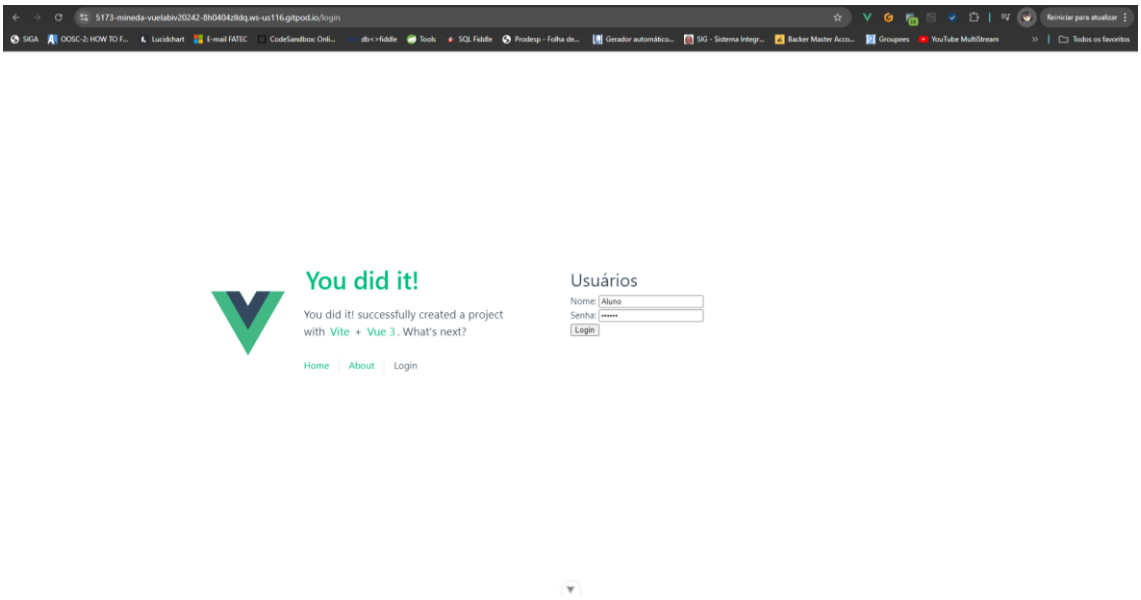
- RouterLink – Um link para nossa página. Utilizamos esse componente no lugar da tag HTML “a”, para evitar que a página seja recarregada ao navegarmos;
- RouterView – Local da página onde o conteúdo definido em “RouterLink” será carregado.

- **Código 3 – Arquivo "App.vue"**

```
1  <script setup lang="ts">
2  import { RouterLink, RouterView } from 'vue-router'
3  import HelloWorld from './components/HelloWorld.vue'
4  </script>
5
6  <template>
7    <header>
8      <img   alt="Vue   logo"   class="logo"   src="@/assets/logo.svg"   width="125"
9  height="125" />
10
11     <div class="wrapper">
12       <HelloWorld msg="You did it!" />
13
14       <nav>
15         <RouterLink to="/">Home</RouterLink>
16         <RouterLink to="/about">About</RouterLink>
17         <RouterLink to="/login">Login</RouterLink>
18       </nav>
19     </div>
20   </header>
21
22   <RouterView />
23 </template>
```

A Figura 1 apresenta o resultado final.

Figura 1 - Tela LoginView.vue



2 Store com Pinia

O token de acesso é algo que gostaríamos de ter disponível em qualquer ponto de minha aplicação e, para isso, podemos criar uma store, utilizando o Pinia, no arquivo "src/stores/usuario.ts". Conforme vemos em Código 4, podemos declarar variáveis e funções em uma store. Esses elementos podem ser acessados em qualquer ponto de nossa aplicação e, assim como variáveis locais, forçam uma re-renderização de qualquer componente que as utilizem. Pontos de interesse:

- O nome da store será "usuarioStore" (linha 5);
- Dentro dela colocamos todas as variáveis e a função de login criadas em "UsuarioView.vue";
- Em "return" (linha 26) indicamos os elemento públicos de nossa store.

Código 4 – Store usuario.ts

```
1  import { ref } from 'vue'
2  import { defineStore } from 'pinia'
3  import axios from 'axios'
4
5  export const usuarioStore = defineStore('usuario', () => {
6    const nomeUsuario = ref<string>()
7    const token = ref<string>()
8    const erro = ref<string>()
9
10   async function login(nome:string, senha:string) {
11     try {
12         let response = (await axios.post('login',
13                 {
14                     nome: nome,
15                     senha: senha
16                 }
17             ));
18         nomeUsuario.value = nome;
19         erro.value = '';
20         token.value = response.data.token;
21     }
22     catch(ex) {
23         erro.value = (ex as Error).message;
24     }
25   }
26   return { nomeUsuario, token, erro, login }}
27 )
```

Como transferimos as variáveis e a função de login para a store, precisamos reescrever “LoginView.vue”, conforme Código 5. Nele, nós importamos a store na linha 22 e a instanciamos na linha 26. A partir daí basta substituir as chamadas para as variáveis e a função de login, incluindo o prefixo “store” (nome que demos à variável, na linha 26).

Código 5 – Arquivo "LoginView.vue"

```
1  <template>
2    <div class="about">
3      <h1 id="hello">Usuários</h1>
4      <p>
5        <label for="nome" id="labelNome">Nome: </label>
6        <input id="nome" type="text" v-model="nome"/>
7      </p>
8      <p>
9        <label for="senha" id="labelSenha">Senha: </label>
10       <input id="senha" type="password" v-model="senha"/>
11     </p>
12     <button @click="store.login(nome, senha)">Login</button>
13     <p v-if="store.erro">{{ store.erro }}</p>
14     <p v-if="store.token">{{ store.token }}</p>
15   </div>
16 </template>
17
18 <script setup lang="ts">
19
20 import { ref } from 'vue';
21 import axios from 'axios';
22 import { usuarioStore } from '@/stores/usuario';
23
24 const nome = ref<string>("Aluno");
25 const senha = ref<string>("123456");
26 const store = usuarioStore();
27
28 </script>
```

Agora vamos fazer um exemplo prático de uso, incluindo o token de autenticação que armazenamos na store em todas as requisições que realizarmos com o axios. Para isso vamos utilizar a funcionalidade interceptor do axios para executar um código personalizado em toda requisição. O Código 6 apresenta essa configuração, que realizamos no arquivo “src/main.ts”. Pontos de interesse:

- Utilizamos “axios.interceptors.request” (linha 15) para incluir um código em todas as requisições. Também podemos incluir um código em todas as respostas em “axios.interceptors.response”;
- Nosso código verifica se existe um token na store (linha 17) e, em caso afirmativo, o inclui no header “Authorization” (linha 18), seguindo o padrão de autenticação por token JWT.

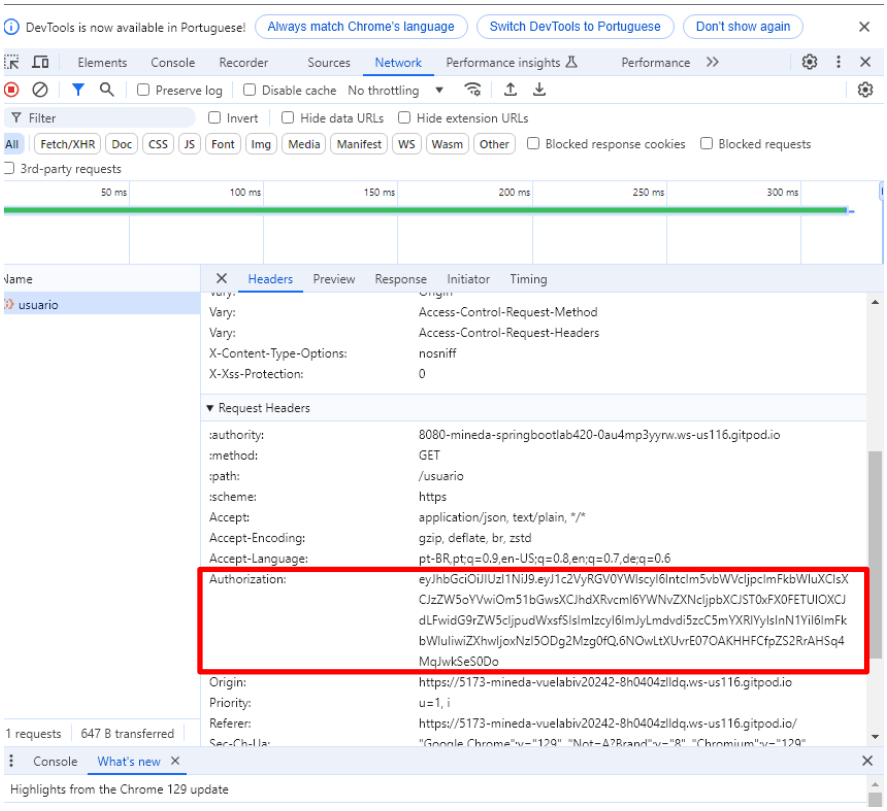
Código 6 – Configuração de um interceptor

```
1  import './assets/main.css'
2
3  import { createApp } from 'vue'
4  import { createPinia } from 'pinia'
5
6  import App from './App.vue'
7  import router from './router'
8
9  import axios from 'axios'
10 import { usuarioStore } from '@/stores/usuario';
11
12 axios.defaults.baseURL    =    'https://8080-mineda-springbootlab420-0au4mp3yyrw.ws-
13 us116.gitpod.io/'
14
15 axios.interceptors.request.use(config => {
16     const store = usuarioStore();
17     if(store.token) {
18       config.headers.Authorization = store.token
19     }
```

```
    return config
  })

const app = createApp(App)

app.use(createPinia())
app.use(router)

app.mount('#app')
</script>
```

Podemos verificar no navegador, conforme Figura 2, que todas as requisições realizadas por nossa aplicação irão conter o token no header “Authorization” depois de realizarmos o login.

Figura 2 – Header “Authorization”



2.1 Persistindo uma store

Como toda variável do Vue.js, as da store também não mantém seus valores se ocorre um reload de página no navegador. Podemos modificar esse comportamento com o Pinia Plugin Persistedstate (documentação disponível em https://github.com/prazdevs/pinia-plugin-persistedstate). Para instalação, basta executar o comando “npm i pinia-plugin-persistedstate” no Terminal. Após isso, configuramos o plugin no arquivo “src/main.ts”, conforme Código 7.

Código 7 – Configuração do pinia-persisted-state

```
import './assets/main.css'

import { createApp } from 'vue'
import { createPinia } from 'pinia'
import piniaPluginPersistedstate from 'pinia-plugin-persistedstate'

import App from './App.vue'
import router from './router'

import axios from 'axios'
```

```
11  import { usuarioStore } from '@/stores/usuario';
12
13  axios.defaults.baseURL    =    'https://8080-mineda-springbootlab420-0au4mp3yyrw.ws-
14  us116.gitpod.io/'
15
16  axios.interceptors.request.use(config => {
17      const store = usuarioStore();
18      if(store.token) {
19        config.headers.Authorization = store.token
20      }
21      return config
22    })
23
24  const app = createApp(App)
25
26  const pinia = createPinia()
27  pinia.use(piniaPluginPersistedstate)
28
29  app.use(pinia)
30  app.use(router)
31
32  app.mount('#app')
```

Precisamos também indicar que desejamos persistir as variáveis de nossa store, conforme Código 8. Por padrão o plugin irá armazenar as variáveis no localStorage, mas isso pode ser alterado via configuração (veja documentação do plugin).

Código 8 – Store usuario.ts

```
1   import { ref } from 'vue'
2   import { defineStore } from 'pinia'
3   import axios from 'axios'
4
5   export const usuarioStore = defineStore('usuario', () => {
6     const nomeUsuario = ref<string>()
7     const token = ref<string>()
8     const erro = ref<string>()
9
10    async function login(nome:string, senha:string) {
11      try {
12          let response = (await axios.post('login',
13                  {
14                      nome: nome,
15                      senha: senha
16                  }
17              ));
18          nomeUsuario.value = nome;
19          erro.value = '';
20          token.value = response.data.token;
21      }
22      catch(ex) {
23          erro.value = (ex as Error).message;
24      }
25    }
26
27    return { nomeUsuario, token, erro, login }},
28  {
29      persist: true
30  }
31  )
```

A Figura 3 mostra que as variáveis da store passam a ser armazenadas automaticamente no localStorage.

Figura 3 - LocalStorage