04

Melhorando as requisições HTTP e evoluindo com o Vuex

Transcrição

Agora que já definimos o Vuex disponível dentro da nossa aplicação e não mais utilizamos o local storage para controlar o estado do usuário, precisamos terminar de refatorar nosso sistema e remover os outros locais que ainda apontam para localstore.

Começaremos por `index.js`, ao invés de realizar o `localStore.getItem('token')`, faremos o import do `provedor` escreveremos:

```
http.interceptors.request.use(function (config){
    const token = provedor.state.token
    if (token) {
        config.headers.Authorization = `Bearer ${token}`
    }
})
```

COPIAR CÓDIGO

Em `BarraNavegacao.vue` realizaremos a modificação em `usuarioestaLogado()`:

```
computed: {
    usuarioestaLogado () {
        return Boolean(this.$store.state.token)

    }
}
```

COPIAR CÓDIGO

Faremos modificações ainda em `BarraNavegacaoQuandoLogado` e `BarraNavegacaoQuandoDeslogado`, utilizaremos como exemplo o segundo caso em que definiremos uma mutation para `DESLOGAR_USUARIO`:

```
<script>
    export default {
        methods: {
            efetuarLogout () {
                this.$store.commit('DESLOGAR_USUARIO')
                this.$router.push({ name: 'login' })
            }
        }
    }
</script>
```

COPIAR CÓDIGO

Precisaremos definir esta nova mutação, afinal ela ainda não existe em `provedor`.

```
const mutations = {
    DEFINIR_USUARIO_LOGADO (state, { token, usuario }) {
        state.token = token
        state.usuario = usuario
    },
    DESLOGAR_USUARIO (state) {
        state.token = null
        state.usuario = {}
    }
}
```

COPIAR CÓDIGO

Feitas as alterações, testaremos a aplicação no navegador para verificar se tudo ocorre como o esperado. A barra de navegação está dinâmica, e o token é coletado com sucesso.

Terminada esta etapa, continuaremos a evoluir nossa aplicação. Nosso componente de login é responsável por fazer a requisição http. Em projetos que utilizam Vuex todas as ações http ficam encapsuladas dentro da store, pois o Vuex além de disponibilizar mutações, há também as ações.

Em `provedor.js` implementaremos justamente nossa ação de login. É interessante assinalar que as mutações em Veux são escritas em caixa alta e as ações não, isso é uma convenção dos desenvolvedores para facilitar a identificação desses elementos no código.

Importaremos o `http` . Criaremos um `const action` e definiremos a ação `efetuarLogin()` . Primeiramente injetaremos o `commit` , além disso receberemos o `usuario` . Feito isso, retornaremos uma `Promise()` e então escreveremos a requisição `post()` . Em caso de sucesso realizaremos o `commit` . Em caso de erro realizaremos um `catch()` .

```
const action = {
    efetuarLogin ({ commit }, usuario) {
        return new Promise( (resolve, reject) => {
            http.post('auth/login', usuario)
            .then(response => {
                commit('DEFINIR_USUARIO_LOGADO', {
                    token: response.data.access_token,
                    usuario: response.data.user
                })
                resolve(response.data)
            })
            .catch(err => {
                console.log(err)
                reject(err)
            })
        })
    }
}
```

COPIAR CÓDIGO

Feito isso, passaremos as ações para dentro do Veux:

```
export default new Vuex.Store({
    state: estado,
    mutations,
    actions
})
```

COPIAR CÓDIGO

Em nosso `Login.vue` , faremos a seguinte modificação em `efetuarLogin()` :

```
methods: {
    efetuarLogin()
        this.$store.dispatch('efetuarLogin', this.usuario)
            .then() => this.$router.push({ name:
'gerentes'})

}
```

COPIAR CÓDIGO

Nossa aplicação não quebrou e continua operando perfeitamente. O que realizamos foi retirar a responsabilidade do componente de realizar uma chamada http, ao invés disso utilizamos uma ação do Vuex.