

## EXERCISES QUE TRABAJAREMOS EN LA CUE

- EXERCISE 1: CREANDO MODULOS EN VUEX PARTE 1.
- EXERCISE 2: CREANDO MODULOS EN VUEX PARTE 2.

### EXERCISE 1: CREANDO MODULOS EN VUEX PARTE 1

Para realizar el siguiente ejercicio, vamos a crear una nueva carpeta, que en este caso llamaremos "cue14", luego, copiaremos el proyecto "cue13", realizado en el CUE anterior.

Haremos la copia de la siguiente manera:

Vamos al terminal, y buscaremos el directorio que contenga el proyecto "cue13".

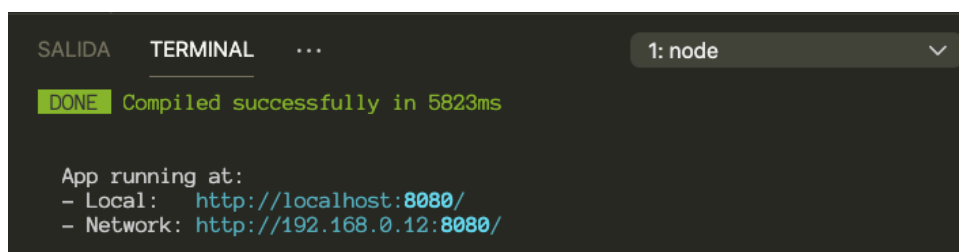
```
~/Documents/cue13
```

Desde esa ubicación, copiaremos el proyecto a la carpeta cue14, con el siguiente comando:

```
~/Documents/cue13 cp -a cue13/. ../cue14/cue14
```

Una vez copiado el proyecto, lo abriremos desde Visual Studio Code, y ahí el terminal para levantarlo.

```
"npm run serve"
```



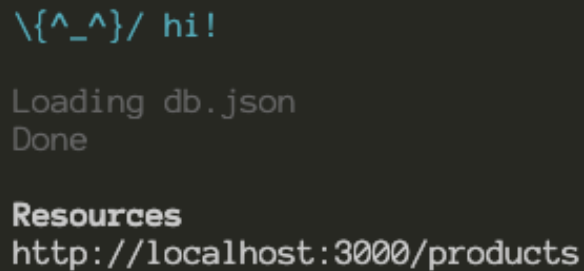
```
SALIDA  TERMINAL  ...  1: node  v
DONE Compiled successfully in 5823ms

App running at:
- Local: http://localhost:8080/
- Network: http://192.168.0.12:8080/
```

Luego, es importante recordar que se debe levantar el servidor **json-server**, para realizar las peticiones.

Para hacerlo, nos dirigiremos a la carpeta `“db_fake”`, y escribiremos el siguiente comando:

```
“json-server -watch db.json”
```



```
\{^_^}/ hi!  
  
Loading db.json  
Done  
  
Resources  
http://localhost:3000/products
```

## CREANDO MÓDULO

Iremos a la carpeta `“/store”`, en la cual solo se encuentra el archivo principal `index.js`. En ella crearemos una nueva llamada `“counter”`, y dentro de ésta, un archivo llamado `“index.js”`.



En el archivo `index.js`, lo primero que debemos hacer, es crear una constante con el nombre `“count”`.

Si lo notamos, la estructura es la misma que en el archivo principal de `vuex`, pues tenemos:

- State.
- Mutations.
- Getters.
- Actions.

```
1 const count={
2   namespace:true,
3   state:{
4     counter:0,
5   },
6   mutations:{
7     ADD:(state)=>{
8       state.counter++;
9     }
10  },
11  getters:{
12  },
13 },
14  actions:{
15    add:({commit})=>{
16      commit('ADD');
17    }
18  },
19  modules:{
20  }
21 }
22 }
23
24 export default count;
```

**“namespace:true”**, nos sirve para indicarle a **vuex**, que invocaremos los datos a través del nombre del módulo.

## IMPORTANDO MÓDULO

Para importar el módulo creado, vamos a abrir el archivo principal de Vuex: **“index.js”**.

```
1 import count from './counter'
```

Dentro del objeto **“modules”**, invocaremos a **count**.

```
1 modules: {
2   count,
3 }
```

## ACCEDIENDO A DATOS DEL MÓDULO 'COUNT'

Para utilizar los datos guardados en el módulo, vamos a dirigirnos a components, ahí crearemos una carpeta llamada "counter", y dentro de ésta, el archivo Counter.vue.

```
1 <template>
2   <div>
3     <h1>Contador: {{counter}}</h1>
4     <button @click="add">Agregar</button>
5   </div>
6 </template>
7
8 <script>
9 import {mapState, mapActions} from 'vuex';
10 export default {
11   name: 'counter-component',
12   // props: {},
13   data: function() {
14     return {}
15   },
16   computed: {
17     ...mapState('count', ['counter']),
18   },
19   methods: {
20     ...mapActions('count', ['add']),
21   },
22   // components: {},
23 }
24 </script>
```

El componente está llamando a un **state** dentro del módulo **"count"**, seguido del llamado a una función llamada **"add"**, que también está dentro de éste.

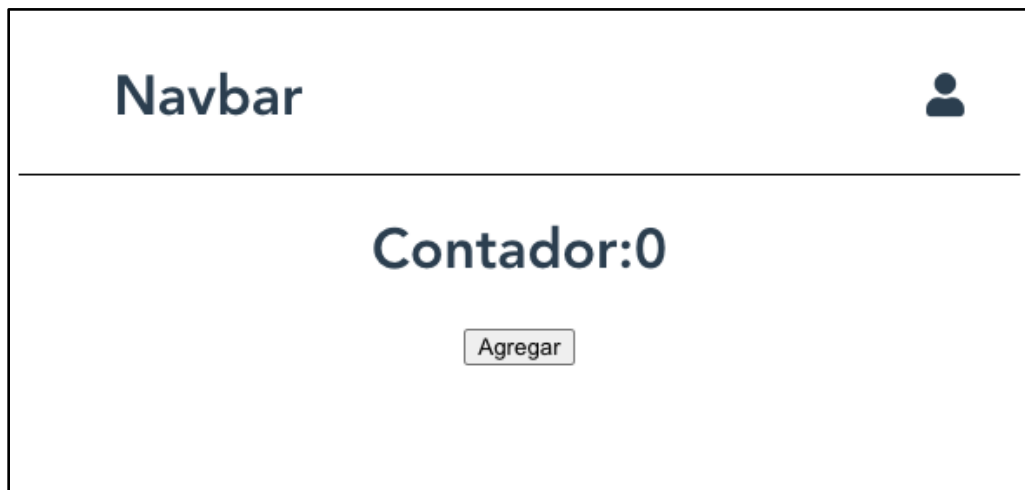
## VINCULANDO ARCHIVO A VISTA

Nos dirigiremos a la vista **Home.vue**, y reemplazaremos el uso del componente **"HelloWorld.vue"**, por el creado.

El archivo **Home.vue**, debe quedar de la siguiente manera:

```
1 <template>
2   <div class="home">
3     <Count></Count>
4
5   </div>
6 </template>
7
8 <script>
9 // @ is an alias to /src
10 import Count from '@components/counter/Counter.vue'
11
12 export default {
13   name: 'Home',
14   components: {
15     Count,
16   }
17 }
18 </script>
```

Si guardamos, y levantamos el proyecto, se verá así:



Al hacer clic en agregar, veremos que el dato vinculado al módulo count, cambiará.



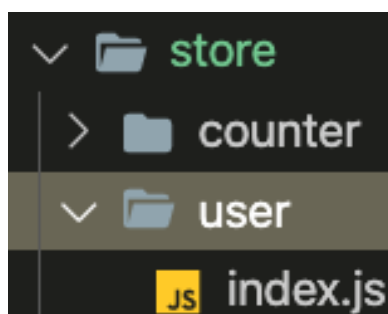
## EXERCISE 2: CREANDO MODULOS EN VUEX PARTE 2

Para realizar el siguiente ejercicio, vamos a continuar trabajando en el proyecto “cue14” creado anteriormente. En este caso, pasaremos a módulos los datos de productos y usuario, que se encuentran mezclados en un solo archivo principal.

Verificaremos que tanto el proyecto esté compilado `“npm run serve”`, como la base de datos de prueba esté levantada en el puerto 3000 `“json-server -watch db.json”`.

### CREANDO MÓDULO USUARIO

Para dividir a nuestro archivo principal de Vuex, vamos a comenzar por los datos de usuario. Nos dirigiremos a la carpeta store, dentro de ésta crearemos una carpeta `“user”`, y una vez lista, generaremos el archivo `index.js` dentro.



Iremos al archivo principal de **vuex**, y nos fijaremos en todos los datos que están relacionados con usuario. Debemos revisar: **states, getters, mutations y actions**.

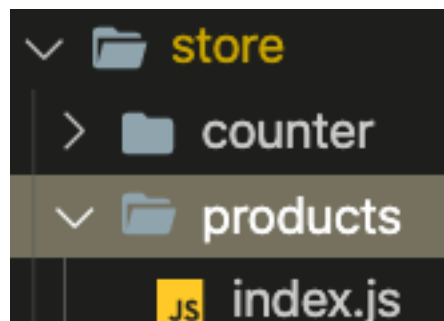
En el caso de usuario, solo tiene states y un **getter** asociado.

El archivo **index.js** debe quedar de la siguiente forma:

```
1 const user = {
2   namespace: true,
3   state: {
4     user_id: "325",
5     name: 'Pedro',
6     last_name: 'Fuentes',
7   },
8   getters: {
9     userName: state => {
10       return state.name + " " + state.last_name;
11     }
12   }
13 }
14 export default user;
```

## CREANDO MÓDULO PARA PRODUCTOS

Dentro de la carpeta **"store"**, vamos a crear una llamada **"products"**, con el archivo **index.js**.



Ahora, debemos ir al archivo principal de **Vuex**, y verificar todos los datos relacionados a productos. Una vez encontrados, los copiaremos a nuestro módulo.

El archivo debe quedar de la siguiente forma.

Es importante que recordemos importar **Vue**, ya que se ocupa en la mutación **"EDIT\_PRODUCT"**, y de importar Axios, el cual se encarga de hacer la peticiones **HTTP**.

```
1 import Vue from 'vue';
2 import axios from 'axios'
3 const products={
4   namespace:true,
5   state:{
6     products:[],
7     id_product_edit:null,
8   },
9   getters:{
10    countProducts: state=>{
11      return state.products.length;
12    },
13    totalProducts: state=>{
14      return state.products.reduce((total,prod)=>{
15        return total + (prod.quantity * prod.price)
16      },0)
17    },
18    getProductByID: (state)=>(id)=>{
19      return state.products.find(prod=>prod.id===id);
20    }
21  },
22  mutations:{
23    REMOVE_PRODUCT: (state,id)=>{
24      let index = state.products.findIndex(prod=>prod.id ===id);
25      state.products.splice(index,1);
26    },
27    ADD_PRODUCT: (state, product)=>{
28      product.id = Math.floor(Math.random() * 1000);
29      state.products.push(product);
30    },
31    SET_ID_PRODUCT_ID: (state,id)=>{
32      state.id_product_edit = id;
33    },
34    EDIT_PRODUCT: (state,product)=>{
35      let index =
36 state.products.findIndex(prod=>prod.id===product.id);
37      Vue.set(state.products,index,product);
38
39      state.id_product_edit=null;
40    },
```





```
41   SET_PRODUCTS: (state, products) => {
42     state.products = products;
43   },
44 },
45 actions: {
46   removeProduct: (context, id) => {
47     axios.delete(`http://localhost:3000/products/${id}`)
48       .then(resp => {
49         console.log(resp)
50         context.commit('REMOVE_PRODUCT', id);
51       })
52       .catch(error => {
53         console.log(error)
54       })
55   },
56   addProduct: ({commit}, product) => {
57     axios.post('http://localhost:3000/products', product)
58       .then(resp => {
59         console.log(resp)
60         commit('ADD_PRODUCT', resp.data);
61       })
62       .catch(error => {
63         console.log(error);
64       })
65   },
66   setIdProductEdit: ({commit}, id) => {
67     commit('SET_ID_PRODUCT_ID', id);
68   },
69   editProduct: ({commit}, product) => {
70     axios.put(`http://localhost:3000/products/${product.id}`,
71 product)
72       .then(resp => {
73         console.log(resp)
74         commit('EDIT_PRODUCT', resp.data);
75       })
76       .catch(error => {
77         console.log(error)
78       })
79   },
80   fetchProducts: ({commit}) => {
81     axios.get('http://localhost:3000/products')
82       .then(resp => {
83         console.log(resp)
84         commit('SET_PRODUCTS', resp.data);
85       })
86       .catch(error => {
87         console.log(error)
```

```
88     })
89   }
90 }
91 export default products;
```

## IMPORTANDO MÓDULOS A ARCHIVO PRINCIPAL

Vamos a dirigirnos al archivo principal de `vuex index.js`, ubicado en `“store”`. Importaremos los módulos `“user”` y `“product”`, y también borraremos los datos y funciones importadas a los módulos. El archivo debe quedar de la siguiente manera:

```
1 import Vue from 'vue'
2 import Vuex from 'vuex'
3 import count from './counter'
4 import user from './user'
5 import products from './products'
6 Vue.use(Vuex)
7
8 export default new Vuex.Store({
9   state: {
10
11   },
12   getters: {
13
14   },
15   mutations: {
16
17   },
18   actions: {
19
20   },
21   modules: {
22     count,
23     user,
24     products
25   }
26 })
```

## OBTENIENDO DATOS DE MÓDULOS IMPORTADOS

En este momento, todos los componentes están obteniendo datos del archivo principal de **vuex**. Por lo que ninguno de ellos los está mostrando correctamente.

## EDITANDO NAVBAR.VUE

Vamos a comenzar, editando el archivo **Navbar.vue**, ubicado en la carpeta **"components"**.

Editaremos el archivo donde estamos obteniendo los datos de **vuex**, en este caso, la propiedad computada.

Solo se está ocupando la propiedad **getters**, por lo que, en primer lugar, indicaremos el nombre del módulo, seguido del nombre del **getter** a obtener.

```
1 computed: {  
2   ...mapGetters('user', ['userName']),  
3  
4 },
```

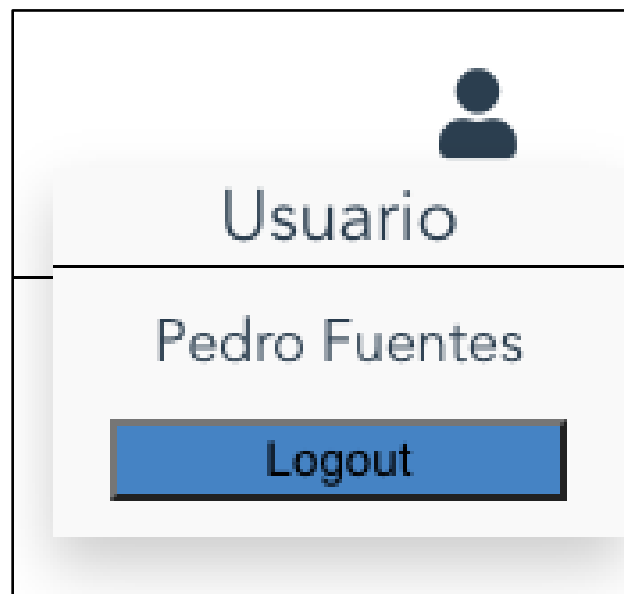
El archivo **Navbar.vue** debe quedar de la siguiente forma:

```
1 <template>  
2   <div>  
3     <div class="container">  
4       <div class="navbar_title">  
5         <h1>Navbar</h1>  
6       </div>  
7       <div class="dropdown">  
8         <i class="icon fas fa-user"></i>  
9         <div class="dropdown-content">  
10          <div>  
11            <div class="title">Usuario</div>  
12            <div class="name">{{userName}}</div>  
13            <button class="btn_logout">Logout</button>  
14          </div>  
15        </div>  
16      </div>  
17    </div>  
18  </div>  
19 </template>  
20  
21 <script>
```



```
22 import {mapGetters} from 'vuex'
23 export default {
24   name: 'Nav-component',
25   // props: {},
26   data: function(){
27     return {}
28   },
29   computed: {
30     ...mapGetters('user', ['userName']),
31   },
32 },
33 }
34 </script>
35
36 <style scoped>
37   .container{
38     border-bottom: 1px solid black;
39     display: grid;
40     grid-template-columns: repeat(10fr);
41   }
42   .navbar_title{
43     text-align: start;
44     grid-row: 1/2;
45     grid-column: 2/8;
46   }
47   .dropdown{
48     grid-row: 1/2;
49     grid-column: 8/9;
50     margin: auto 0;
51     padding: 5px;
52     text-align: center;
53   }
54   .dropdown-content{
55     display: none;
56     position: absolute;
57     right: 10px;
58     background-color: #f9f9f9;
59     min-width: 160px;
60     box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
61     z-index: 1;
62   }
63   .dropdown:hover .dropdown-content{
64     display: block;
65     text-align: center;
66   }
67   .title{
68     font-size: 20px;
```

```
69   border-bottom: 1px solid black;
70 }
71 .name{
72   padding: 10px 0;
73 }
74 .btn_logout{
75   display:inline-block;
76   background: rgb(74,132,199);
77   width:80%;
78   padding: 2px 0;
79   margin-bottom:10px;
80 }
81 .icon{
82   font-size:25px;
83 }
84 </style>
```



## EDITANDO PRODUCTOS

Si nos dirigimos a la ruta `"/products"`, veremos que todos los componentes están caídos, pues la referencia a los datos está errónea.

- ProductCount.vue

El primer archivo a editar, será **ProductCount.vue**, dentro de **"components>products"**.

La sección editaremos será en las propiedades computadas.

En este caso, debemos poner el nombre del módulo **"productos"**, y seguido, entre corchetes, los getters a utilizar. En este caso: **"countProducts"**.

```
1 computed: {  
2   ...mapGetters('products', ['countProducts']),  
3 },
```

- ProductList.vue

Será el segundo archivo a editar, en éste tenemos 3 elementos:

- Propiedad computada:

```
1 computed: {  
2   ...mapState('products', ['products'])  
3 },
```

- Methods:

Debemos editar el **mapActions** y el **dispatch**.

```
1 methods: {  
2   ...mapActions('products', ['setIdProductEdit']),  
3   remove(id) {  
4     let response = confirm("¿Estas seguro de Eliminar el  
5     producto?");  
6     if(response) {  
7       this.$store.dispatch('products/removeProduct', id);  
8     } }, },
```

- ProductTotal.vue

Aquí vamos a editar la sección de las propiedades computadas.

```
1 computed: {  
2   ...mapGetters('products', ['totalProducts']),  
3 },
```

```
4 },
```

- ProductNew.vue

Vamos a editar la sección de **methods**, específicamente el método **"add"**.

```
1 add() {  
2     if(  
3         this.form.code !== ""  
4         && this.form.product !== ""  
5         && this.form.quantity !== ""  
6         && this.form.price !== ""  
7     ) {  
8         let data = {...this.form}  
9         this.$store.dispatch('products/addProduct', data);  
10        this.clean();  
11    }  
12 },
```

- ProductEdit.vue

Aquí vamos a editar dos secciones:

La primera, serán las propiedades computadas.

```
1 computed: {  
2     ...mapGetters('products', ['getProductByID']),  
3     ...mapState('products', ['id_product_edit'])  
4 },
```

Y la segunda, será **methods**, específicamente el **mapAction**, en el cual debemos referenciar el nombre del módulo.

```
1 ...mapActions('products', ['editProduct']),
```

Si visitamos **"/products"**, revisaremos que se compartan correctamente.

De esta manera, podemos mantener nuestros datos modularizados, según el contexto o tarea que ellos realicen.

## Productos 4

Código	Producto	Cantidad	Precio	Acción
23452323	Funda papel 2	2	100	 
24452329	Papel de regalo	5	1000	 
643454	Hojas de Papel	100	10	 
324345	Tela	10	100	 
Total: \$7200				

### Nuevo Producto

Código

Producto

Cantidad

Precio

 Agregar