



EXERCISES QUE TRABAJAREMOS EN LA CUE:

- EXERCISE 1: GETTERS EN VUEX.
- EXERCISE 2: MUTATIONS (ELIMINAR PRODUCTO).
- EXERCISE 3: MUTATIONS.

EXERCISE 1: GETTERS EN VUEX.

INSTRUCCIONES

Lee con atención cada una de las cuestiones que se presentan, y responde de acuerdo a lo requerido.

Para realizar el siguiente ejercicio, copia el proyecto hecho en el CUE anterior, en este caso llamado "cue 11", en una nueva carpeta, que llamaremos "cue12".

Para crear una carpeta vacía, utilizaremos el siguiente comando:

```
~/Documents mkdir cue12
```

Luego, nos dirigiremos a donde está el proyecto cue11, en este caso, tenemos una carpeta llamada cue11, y dentro de ella estará dicho proyecto.

```
~/Documents/cue11 ls
CUE11_text-class-review.docx
CUE11_textclassreview_recuperado.docx
Cue11_ejercicio2.docx
Cue11_ejercicio1_instalacion_intro.docx
Cue11_ejercicio3_mapState.docx
cue11
```

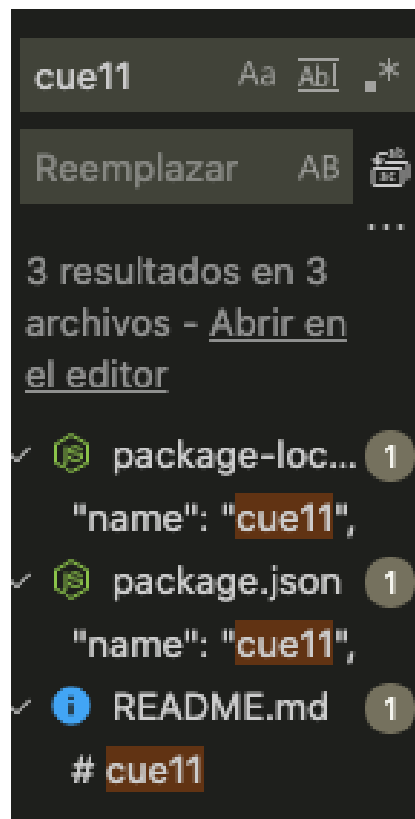
Ahora, teniendo la carpeta cue11 visualizada, vamos a copiarla a la carpeta cue12.

```
~/Documents/cue11 cp -a cue11/. ../cue12/cue12
```

De esta forma, se copiará todo el contenido de la carpeta cue11, en una nueva carpeta dentro de cue12, llamada de igual manera.

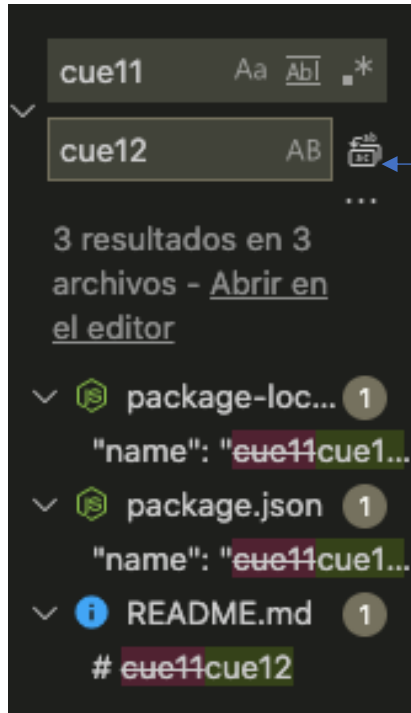
Teniendo esto, ya solo necesitamos abrir el proyecto con Visual Studio Code.

Para cambiar el nombre del proyecto de cue11 a cue12, debemos hacer una búsqueda en el proyecto donde aparezca el primero.



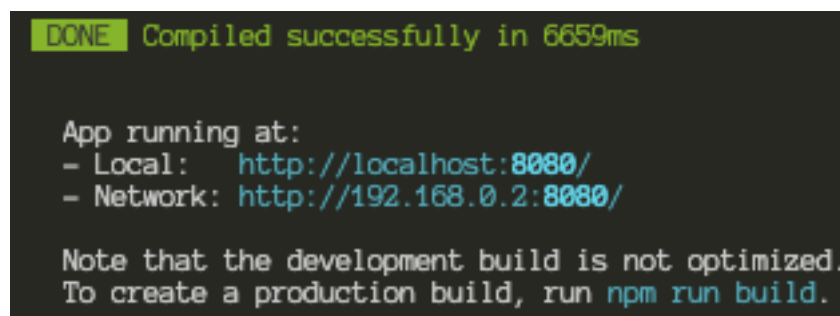
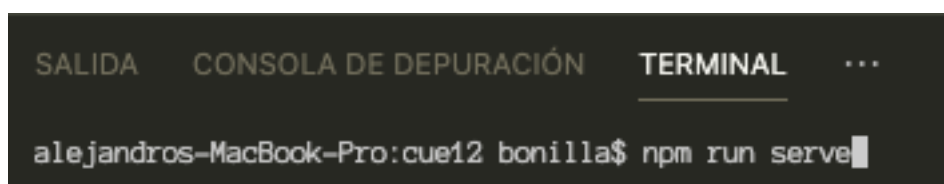
Si lo notamos, aparece en 3 archivos, por lo que haremos el reemplazo de cue11 por cue12 en todos.

De la siguiente forma.



Luego presionamos el botón “reemplazar todo”.

Si lo copiamos bien, ya solo deberíamos poder correr la aplicación desde el terminal con `npm run serve`.



GETTERS

Cuando trabajamos con States en Vuex, a menudo es necesario poder generar datos derivados, como lo hacemos en un componente con las propiedades computadas. Es por esta razón, que en Vuex se crea algo similar a dichas propiedades computadas, llamado Getters, que nos permitirán acceder a los States, y poder retornar datos nuevos.

Si observamos el archivo ProductCount.vue, en la carpeta products dentro de components.

Veremos lo siguiente:

```
computed: {  
  ...mapState(['products']),  
  count(){  
    return this.products.length;  
  }  
},
```

Teniendo solo mapState, debemos generar otra propiedad computada adicional, llamada **count**, para poder realizar la lógica de obtener el largo del Array.

Si necesitamos contar los elementos, desde otro componente, tendríamos que repetir código. Lo que no es aconsejable.

EJERCICIO: CREANDO GETTER COUNTPRODUCTS

Para poder trabajar con getters, debemos crearlos dentro del store. Entonces, generaremos un nuevo objeto llamado getters: {}.

```
getters: {  
  countProducts: state => {  
    return state.products.length;  
  },  
}
```

Cada getter debe tener un nombre, en este caso le llamaremos "countProducts".

Al entrar como primer argumento state a la función, podemos acceder a todos los state de Vuex. En este caso, accedemos a products para obtener el largo del Array.

Una vez teniendo creado el getter, lo podremos invocar desde el componente "ProductCount.vue", a través de un mapGetters.

```
1 <template>
2   <h1> Productos {{countProducts}}</h1>
3 </template>
4
5 <script>
6 import {mapGetters} from 'vuex';
7 export default {
8   name: 'Product-count',
9   // props: {},
10  data: function(){
11    return {}
12  },
13  computed: {
14    ...mapGetters(['countProducts']),
15  },
16  methods: {
17    // -- Metodos
18  },
19  // components: {},
20 }
21 </script>
22
23 <style scoped>
24
25 </style>
```

CREANDO GETTER TOTALPRODUCTS:

Vamos a copiar el contenido de la propiedad computada total del componente ProductTotal.vue.

```
computed: {
  ...mapState(['products']),
  total(){
    return this.products.reduce((total,prod)=>{
      return total + (prod.quantity * prod.price)
    },0)
  }
},
```

Y creamos un nuevo getter, llamado "totalProducts".

```
1 totalProducts: state=>{
2   return state.products.reduce((total,prod)=>{
3     return total + (prod.quantity * prod.price)
4   },0)
5 }
```

Llamaremos a esta propiedad en ProductTotal.vue, a través de mapGetters.

```
1 <template>
2   <div class="container">
3     <div>Total: ${{totalProducts}}</div>
4   </div>
5 </template>
6
7 <script>
8 import {mapGetters} from 'vuex'
9 export default {
10   name: 'ProductTotal',
11   // props: {},
12   data: function() {
13     return {}
14   },
15   computed: {
16     ...mapGetters(['totalProducts']),
17   },
18   methods: {
19     // -- Metodos
20   },
21   // components: {},
22 }
23 </script>
24
25
26 <style scoped>
27   .container{
28     font-size:25px;
29     background:grey;
30     color:white;
31     padding-top:10px;
32     text-align: start;
33   }
34 </style>
```

CREANDO GETTER USERNAME

En el componente NavBar.vue, en la carpeta componentes, tenemos:

```
computed: {  
  userName(){  
    return this.$store.state.name + " " + this.$store.state.last_name;  
  }  
},
```

Esto lo pasaremos a un getter, también llamado userName.

```
1 userName: state=>{  
2   return state.name + " " + state.last_name;  
3 },
```

Luego, en el componente NavBar.vue, lo invocaremos con la ayuda de mapGetters.

```
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 <script>  
21 import {mapGetters} from 'vuex'  
22 export default {  
23   name: 'Nav-component',  
24   // props: {},  
25   data: function() {  
26     return {}  
27   },  
28   computed: {
```



```
29     ...mapGetters(['userName']),
30   },
31 }
32 </script>
33 <style scoped>
34   .container{
35     border-bottom: 1px solid black;
36     display:grid;
37     grid-template-columns: repeat(10fr);
38   }
39   .navbar_title{
40     text-align: start;
41     grid-row:1/2;
42     grid-column:2/8;
43   }
44   .dropdown{
45     grid-row: 1/2;
46     grid-column: 8/9;
47     margin: auto 0;
48     padding: 5px;
49     text-align:center;
50   }
51   .dropdown-content{
52     display:none;
53     position:absolute;
54     right:10px;
55     background-color:#f9f9f9;
56     min-width:160px;
57     box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
58     z-index: 1;
59   }
60   .dropdown:hover .dropdown-content{
61     display:block;
62     text-align:center;
63   }
64   .title{
65     font-size:20px;
66     border-bottom: 1px solid black;
67   }
68   .name{
69     padding: 10px 0;
70   }
71   .btn_logout{
72     display:inline-block;
73     background: rgb(74,132,199);
74     width:80%;
75     padding: 2px 0;
76     margin-bottom:10px;
77   }
78   .icon{
79     font-size:25px;
```



```
80   }
81 </style>
```

Si guardamos y visualizamos nuestra aplicación en el navegador, veremos que nada ha cambiado, solo hemos mejorado el orden, y la no repetición de código a través de getters.

Navbar			
Productos 3			
Código	Producto	Cantidad	Precio
23452323	Funda papel	2	100
23443323	Bandeja Carton	2	200
24452329	Papel de regalo	2	500
Total: \$1600			

EXERCISE 2: MUTATIONS (ELIMINAR PRODUCTO).

INSTRUCCIONES:

Lee cuidadosamente cada una de las cuestiones que se te presentan a continuación y responde de acuerdo con lo requerido.

Para realizar este ejercicio continuaremos trabajando en el proyecto al cual llamé "cue12".

MUTATIONS:

Las mutaciones son la única forma de poder modificar valores en el State de vuex a través de ellas podremos acceder a ellos y realizar lógica para modificarlos.

Para utilizar mutaciones lo haremos en el objeto mutations: {}

```
mutations: {
  },
```

EJERCICIO ELIMINAR PRODUCTO:

Teniendo el Array `products` en los States de `vuex`, la forma de poder eliminar un elemento es conociendo su `id`, ya que esta es única, nos permitirá encontrar el producto y eliminarlo.

La mutación se llamará `"REMOVE_PRODUCT"`, el primer argumento en entrar a la función será `state`, el cual nos permitirá acceder a `products`, el segundo parámetro será la `id` del producto a eliminar. De esta forma podremos crear una función para eliminar un elemento asociado a una `id`.

```
mutations: {  
  REMOVE_PRODUCT: (state, id) => {  
    let index = state.products.findIndex(prod => prod.id === id);  
    state.products.splice(index, 1);  
  }  
}
```

MODIFICANDO PRODUCTLIST.VUE, PARA UTILIZAR LA MUTACIÓN.

Lo primero que haremos será modificar el template, ya que agregaremos una nueva columna. Llamada "Acción".

En los `tab_content`, agregaremos un botón el cual tendrá asociado un método llamado `remove` el cual recibirá como argumento la `id` del producto.

```
1 <template>  
2   <div>  
3     <div class="container">  
4       <div class="tab">Código</div>  
5       <div class="tab">Producto</div>  
6       <div class="tab">Cantidad</div>  
7       <div class="tab">Precio</div>  
8       <div class="tab">Acción</div>  
9     </div>  
10    <div class="container" v-for="product in products"  
11      :key="product.id">  
12      <div class="tab_content">{{product.code}}</div>  
13      <div class="tab_content">{{product.product}}</div>  
14      <div class="tab_content">{{product.quantity}}</div>  
15      <div class="tab_content">{{product.price}}</div>  
16      <div class="tab_content"><button @click="remove(product.id)">
```

```
17     <i class="fas fa-trash"></i>
18   </button></div>
19 </div>
20 </div>
21 </template>
```

CREANDO METODO REMOVE:

En el objeto methods crearemos el método **remove**:

La forma de invocar la mutación será con **`$store.commit(nombre_mutacion)`**

```
1 methods: {
2   remove(id) {
3     let response = confirm("¿Estas seguro de Eliminar el
4 producto?");
5     if(response) {
6       this.$store.commit('REMOVE_PRODUCT', id);
7     }
8   }
9 },
```

Si nos damos cuenta la mutación REMOVE_PRODUCT, necesita un argumento de entrada en este caso es la id.

De esta forma al momento que el usuario, presione sobre el botón eliminar, se eliminara automáticamente el producto, recalculando los getters asociados en otros componentes.

ProductList.vue final quedaría:

```
1 <template>
2 <div>
3   <div class="container">
4     <div class="tab">Código</div>
5     <div class="tab">Producto</div>
6     <div class="tab">Cantidad</div>
7     <div class="tab">Precio</div>
8     <div class="tab">Acción</div>
9   </div>
10   <div class="container" v-for="product in products"
11 :key="product.id">
12     <div class="tab_content">{{product.code}}</div>
13     <div class="tab_content">{{product.product}}</div>
14     <div class="tab_content">{{product.quantity}}</div>
15     <div class="tab_content">{{product.price}}</div>
16   </div>
```

```

17     <div class="tab_content"><button
18 @click="remove(product.id)">
19     <i class="fas fa-trash"></i>
20     </button></div>
21 </div>
22 </div>
23 </template>
24
25 <script>
26 import {mapState} from 'vuex'
27 export default {
28   name: 'Product-list',
29   // props: {},
30   data: function() {
31     return {}
32   },
33   computed: {
34     ...mapState(['products'])
35   },
36   methods: {
37     remove(id) {
38       let response = confirm("¿Estas seguro de Eliminar el
39 producto?");
40       if(response) {
41         this.$store.commit('REMOVE_PRODUCT',id);
42       }
43     }
44   },
45   // components: {},
46 }
47 </script>
48
49 <style scoped>
50   .container{
51     display:grid;
52     grid-template-columns: 1fr 4fr 1fr 2fr 1fr;
53   }
54   .tab_content{
55     background:orange;
56   }
57   .tab{
58     background:grey;
59     color:white;
60   }
61 </style>



```

Si guardamos y nos dirigimos a “/products”

Navbar



Productos 3

Código	Producto	Cantidad	Precio	Acción
23452323	Funda papel	2	100	
23443323	Bandeja Carton	2	200	
24452329	Papel de regalo	2	500	
Total: \$1600				

Si presionamos en alguno de los productos, nos saldrá un confirm:


 Apl

localhost:8080 dice
 ¿Estas seguro de Eliminar el producto?

Cancelar
 Aceptar


favoritos

Productos 3

Código	Producto	Cantidad	Precio	Acción
23452323	Funda papel	2	100	
23443323	Bandeja Carton	2	200	
24452329	Papel de regalo	2	500	
Total: \$1600				

Luego de Eliminar el producto veremos, que los valores de los getters asociados, vuelven a ser calculados automáticamente.

Productos 2

Código	Producto	Cantidad	Precio	Acción
23452323	Funda papel	2	100	
23443323	Bandeja Carton	2	200	
Total: \$600				

EXERCISE 3: MUTATIONS.

INSTRUCCIONES

Lee con atención los elementos que se presentan a continuación, y responde de acuerdo a lo solicitado.

Para realizar este ejercicio, vamos a continuar trabajando en el proyecto que hemos utilizado hasta ahora ("cue12").

EJERCICIO:

En el ejercicio anterior, vimos como a través de una mutación, se puede acceder al state y eliminar el producto seleccionado. En este ejemplo, se agregará un nuevo producto a través de una mutación.

CREANDO MUTACIÓN:

Para agregar un nuevo producto al state de Vuex, la única forma es a través de una mutación, por lo que iremos al archivo index.js de la carpeta "store".

Crearemos una mutación llamada ADD_PRODUCT, que recibirá como segundo argumento un objeto, el cual contendrá los datos para agregar un nuevo producto.

```
1 ADD_PRODUCT:(state, product)=>{  
2   product.id = Math.floor(Math.random() * 1000);  
3   state.products.push(product);  
4 }
```

Como cada producto debe tener una id, le crearemos una de manera aleatoria.

```
1 product.id = Math.floor(Math.random() * 1000);
```

El hecho de asignar una id desde el front es solo para crear el ejemplo, regularmente ésta es creada por la base de datos de una aplicación, de forma automática verificando que sea única.

Para agregar el producto, utilizaremos la función de Array llamada **push**, la cual nos permite agregar un elemento al final de éste.

```
1 state.products.push(product);
```

CREANDO COMPONENTE:

Iremos a la carpeta "@/components/products", y crearemos un nuevo archivo llamado ProductNew.vue.

Partiremos creando la sección de template, que contendrá un pequeño formulario para agregar productos.

```
1 <template>
2   <div class="container">
3     <div class="title"><h2>Nuevo Producto</h2></div>
4     <div class="code">
5       <label for="">Código</label>
6       <input type="text" v-model="form.code">
7     </div>
8     <div class="product">
9       <label for="">Producto</label>
10      <input type="text" v-model="form.product">
11    </div>
12    <div class="quantity">
13      <label for="">Cantidad</label>
14      <input type="text" v-model="form.quantity">
15    </div>
16    <div class="price">
17      <label for="">Precio</label>
18      <input type="text" v-model="form.price">
19    </div>
20    <div class="add">
21      <button @click="add">
22        <i class="fas fa-plus-circle"></i>
23        Agregar
```

```

24     </button>
25   </div>
26 </div>
27 </template>
  
```

Vamos a crear la sección de script, la cual contendrá en data, un objeto llamado form, el cual agrupará todos los datos que se asociaron con v-model, en la sección de template.

En methods, tendremos 2 métodos: uno llamado **add()**, para agregar un nuevo producto a Vuex; y el método **clean()**, que servirá para limpiar el formulario luego de agregar los datos.

La forma de llamar a la mutación es haciendo un commit con el nombre de la mutación a invocar, y si ésta tuviera payload, debemos agregarlo como en el caso del ejercicio.

```

1 <script>
2 export default {
3   name: 'ProductNEW',
4   // props: {},
5   data: function() {
6     return {
7       form: {
8         code: "",
9         product: "",
10        quantity: "",
11        price: "",
12      }
13    }
14  },
15  // computed: {},
16  methods: {
17    add() {
18      if(
19        this.form.code !== ""
20        && this.form.product !== ""
21        && this.form.quantity !== ""
22        && this.form.price !== ""
23      ) {
24        let data = {...this.form}
25        this.$store.commit('ADD_PRODUCT', data);
26        this.clean();
27      }
28    },
29    clean() {
30      this.form.code = ""
31      this.form.product = ""
32      this.form.quantity = ""
33      this.form.price = ""
34    }
  }
}
  
```



```
35 },
36 // components: {},
37 }
38 </script>
```

Por último, para darle presentación al componente, escribimos algunas reglas CSS.

```
1 <style scoped>
2   .container{
3     display:grid;
4     grid-template-columns: repeat(5,1fr);
5     text-align:start;
6     padding:10px;
7     width:500px;
8     margin: 0 auto;
9     background:rgb(66,163,9);
10    margin-top:10px;
11  }
12  .title{
13    grid-column:1/4;
14    grid-row:1/2;
15  }
16  .code{
17    grid-column:1/3;
18    grid-row:2/3;
19  }
20  .product{
21    grid-column:1/4;
22    grid-row:3/4;
23  }
24  .quantity{
25    grid-column:1/2;
26    grid-row:4/5;
27  }
28  .price{
29    grid-column:2/4;
30    grid-row:4/5;
31  }
32  .add{
33    grid-column:1/5;
34    grid-row:5/6;
35    margin:auto 0;
36  }
37  }
38  input{
39    width: 90%;
40  }
41  h2{
42    color:white;
```

```

43   }
44   label{
45     color:white;
46   }
47 </style>

```

El archivo ProductNew.vue final, debe quedarnos de la siguiente manera.

```

1 <template>
2   <div class="container">
3     <div class="title"><h2>Nuevo Producto</h2></div>
4     <div class="code">
5       <label for="">Código</label>
6       <input type="text" v-model="form.code">
7     </div>
8     <div class="product">
9       <label for="">Producto</label>
10      <input type="text" v-model="form.product">
11    </div>
12    <div class="quantity">
13      <label for="">Cantidad</label>
14      <input type="text" v-model="form.quantity">
15    </div>
16    <div class="price">
17      <label for="">Precio</label>
18      <input type="text" v-model="form.price">
19    </div>
20    <div class="add">
21      <button @click="add">
22        <i class="fas fa-plus-circle"></i>
23        Agregar
24      </button>
25    </div>
26  </div>
27 </template>
28
29 <script>
30 export default {
31   name: 'ProductNEW',
32   // props: {},
33   data: function() {
34     return {
35       form: {
36         code: "",
37         product: "",
38         quantity: "",
39         price: "",
40       }
41     }

```



```

42   },
43   // computed: {},
44   methods: {
45     add() {
46       if(
47         this.form.code !== ""
48         && this.form.product !== ""
49         && this.form.quantity !== ""
50         && this.form.price !== ""
51       ) {
52         let data = {...this.form}
53         this.$store.commit('ADD_PRODUCT', data);
54         this.clean();
55       }
56     },
57     clean() {
58       this.form.code = ""
59       this.form.product = ""
60       this.form.quantity = ""
61       this.form.price = ""
62     }
63   },
64   // components: {},
65 }
66 </script>
67
68 <style scoped>
69   .container{
70     display: grid;
71     grid-template-columns: repeat(5, 1fr);
72     text-align: start;
73     padding: 10px;
74     width: 500px;
75     margin: 0 auto;
76     background: rgb(66, 163, 9);
77     margin-top: 10px;
78   }
79   .title{
80     grid-column: 1/4;
81     grid-row: 1/2;
82   }
83   .code{
84     grid-column: 1/3;
85     grid-row: 2/3;
86   }
87   .product{
88     grid-column: 1/4;
89     grid-row: 3/4;
90   }
91   .quantity{
92     grid-column: 1/2;

```

```
93     grid-row:4/5;
94   }
95   .price{
96     grid-column:2/4;
97     grid-row:4/5;
98   }
99   .add{
100     grid-column:1/5;
101     grid-row:5/6;
102     margin:auto 0;
103   }
104 }
105 input{
106   width: 90%;
107 }
108 h2{
109   color:white;
110 }
111 label{
112   color:white;
113 }
114 </style>
```

IMPORTANDO COMPONENTE A VISTA PRODUCTS

Lo que debemos hacer ahora, es poder referenciar esta vista dentro de Products.vue, que se encuentra en la carpeta views.

Importaremos ProductNew.vue, y lo agregaremos en el objeto componentes.


```
1 <template>
2   <div>
3     <ProductCount></ProductCount>
4     <ProductList></ProductList>
5     <ProductTotal></ProductTotal>
6     <ProductNew></ProductNew>
7   </div>
8 </template>
9
10 <script>
11 import ProductCount from '@components/products/ProductCount.vue'
12 import ProductList from '@components/products/ProductList.vue'
13 import ProductTotal from '@components/products/ProductTotal.vue'
14 import ProductNew from '@components/products/ProductNew.vue'
15 export default {
16   name: 'Product-view',
17   // props: {},
18   data: function() {
```

```

19     return {}
20   },
21   // computed: {},
22   methods: {
23     // -- Metodos
24   },
25   components: {
26     ProductCount,
27     ProductList,
28     ProductTotal,
29     ProductNew
30   },
31 }
32 </script>
33
34 <style scoped>
35
36 </style>
  
```

Si guardamos, y visitamos la ruta /products, lo podemos visualizar así.

Productos 3

Código	Producto	Cantidad	Precio	Acción
23452323	Funda papel	2	100	
23443323	Bandeja Carton	2	200	
24452329	Papel de regalo	2	500	
Total: \$1600				

Nuevo Producto

Código

Producto

Cantidad

Precio

 Agregar

Si agregamos un producto, éste automáticamente se agrega a la lista, modificando los getters asociados.


Nuevo Producto

Código

Producto

Cantidad

Precio

 Agregar

Productos 4

Código	Producto	Cantidad	Precio	Acción
23452323	Funda papel	2	100	
23443323	Bandeja Carton	2	200	
24452329	Papel de regalo	2	500	
23452345	Funda Plastica	4	350	
Total: \$3000				