EXERCISES QUE TRABAJAREMOS EN LA CUE:

- EXERCISE 1: VUEX, INTRODUCCIÓN Y ESTADOS.
- EXERCISE 2: MISMOS DATOS Y DIFERENTES ENFOQUES EN COMPONENTES.
- EXERCISE 3: VUEX - MAPSTATE.

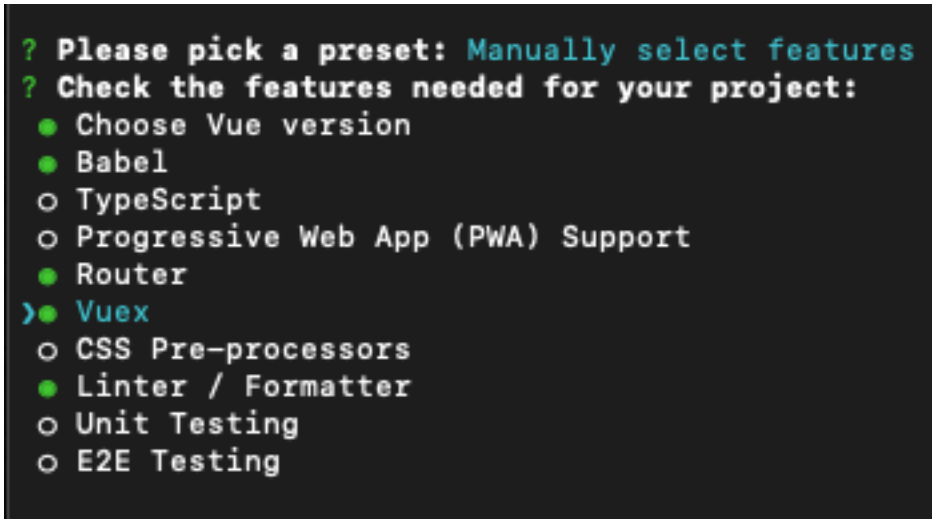
EXERCISE 1: VUEX, INTRODUCCIÓN Y ESTADOS

INSTRUCCIONES

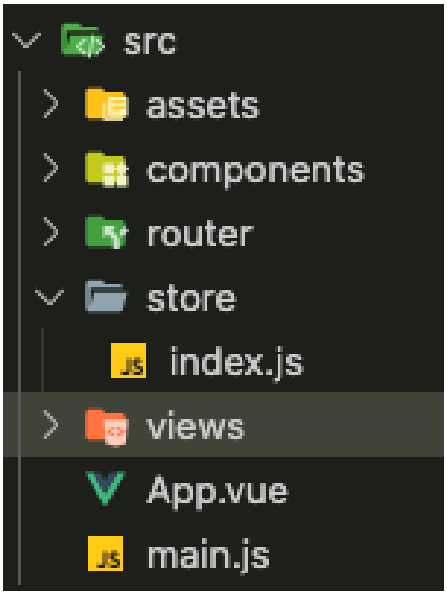
Para realizar el siguiente ejercicio, vamos a crear un nuevo proyecto a través de vue/cli. En este caso, se llamará “cue11”, pero puedes nombrarlo como prefieras. Cabe destacar que, al momento de hacer la instalación, no olvides agregar Vuex.

Instalar Vuex por vue/cli:

Cuando iniciamos un nuevo proyecto a través de vue/cli, nos preguntará si deseamos agregar Vuex. De esta forma, nos creará automáticamente el archivo necesario para trabajar.



Al terminar la instalación, vamos a obtener una nueva carpeta dentro de “src”, llamada “store”.



Al abrir index.js, veremos la estructura vacía de Vuex.

```
1  import Vue from 'vue'
2  import Vuex from 'vuex'
3
4  Vue.use(Vuex)
5
6  export default new Vuex.Store({
7    state: {
8    },
9    mutations: {
10   },
11   actions: {
12   },
13   modules: {
14   }
15 })
```

State:

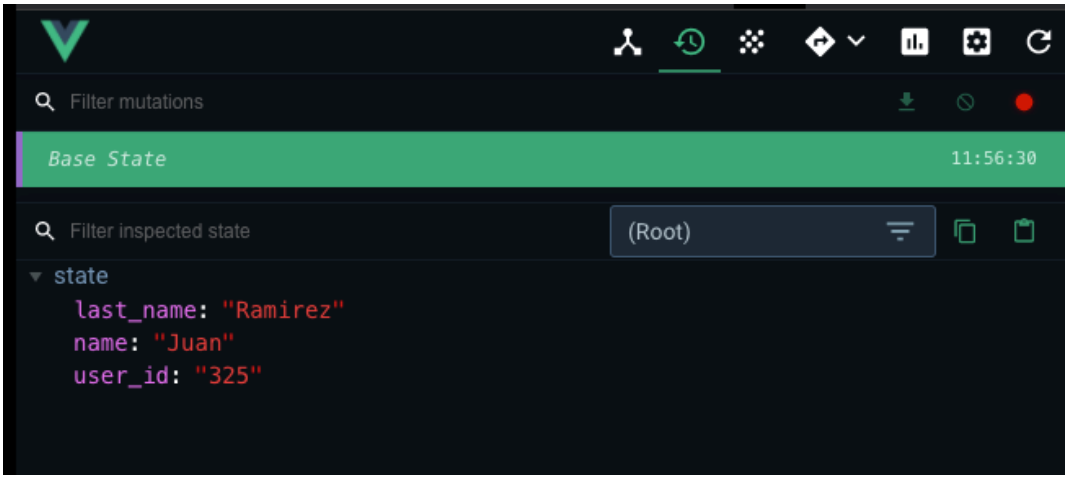
Debemos pensar en él como si fuera data de un componente, ya que podremos utilizar los mismos tipos de datos, con la única diferencia de que va a poder ser accesado por diferentes componentes.

Ejercicio:

Para iniciar este proceso de aprendizaje sobre cómo trabajar con Vuex, crearemos 3 datos dentro de state.

```
state: {
    user_id:"325",
    name: 'Juan',
    last_name: 'Ramirez',
  },
```

Al momento de guardar estos datos en state, podremos visualizarlos a través de “vue-devtools” desde el navegador.



Componente NavBar.vue

El componente que accederá a los datos que creamos en el almacén de Vuex, será NavBar.vue, este archivo lo crearemos en la carpeta “componentes”.

Template NavBar.vue

Iniciemos por el template de nuestro componente navBar.

```
<template>
  <div>
    <div class="container">
      <div class="navbar_title">
        <h1>Navbar</h1>
      </div>
      <div class="dropdown">
        <i class="icon fas fa-user"></i>
        <div class="dropdown-content">
          <div>
            <div class="title">Usuario</div>
            <div class="name">{{userName}}</div>
            <button class="btn_logout">Logout</button>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>
```

Una vez creado el template, pasaremos a la sección de script.

Script NavBar.vue

En esta sección haremos el enlace a los datos almacenados en “Vuex”, la forma de acceder es: **this.$store.state.nombre_propiedad**.

```
<script>
export default {
  name: 'Nav-component',
  computed: {
    userName(){
      return this.$store.state.name + " " +
this.$store.state.last_name;
    }
  },
}
</script>
```

Como nombre y apellido vienen separados, los uniremos en una sola variable llamada: **userName,** la cual retorna el nombre y el apellido.

Sytle NavBar.vue

Por último, agregaremos estilos CSS a nuestro componente.

```
<style scoped>
  .container{
    border-bottom: 1px solid black;
    display:grid;
    grid-template-columns: repeat(10fr);
  }
  .navbar_title{
    grid-row:1/2;
    grid-column:1/8;
    text-align:start;
  }
  .dropdown{
    grid-row: 1/2;
    grid-column: 8/9;
    margin: auto 0;
    padding: 5px;
    text-align:center;
  }
  .dropdown-content{
    display:none;
    position:absolute;
    right:10px;
    background-color:#f9f9f9;
    min-width:160px;
    box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
    z-index: 1;
  }
  .dropdown:hover  .dropdown-content{
    display:block;
    text-align:center;
  }
  .title{
    font-size:20px;
    border-bottom: 1px solid black;
  }
  .name{
    padding: 10px 0;
  }
  .btn_logout{
    display:inline-block;
    background: rgb(74,132,199);
    width:80%;
```

```
    padding: 2px 0;
    margin-bottom:10px;
  }
  .icon{
    font-size:25px;
  }
</style>
```

Enlazando NavBar.vue a nuestra app.

Hasta el momento, solo hemos creado el componente, pero aún no puede ser visualizado. Para hacer esto, iremos al archivo principal llamado “App.vue”.

El archivo App.vue quedará de la siguiente manera.

```
<template>
  <div id="app">
    <NavBar></NavBar>
    <router-view/>
  </div>
</template>

<script>
import NavBar from '@/components/NavBar.vue'

export default {
  components:{
    NavBar
  }
}
</script>

<style>
@import
"https://use.fontawesome.com/releases/v5.2.0/css/all.css";

#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
}

#nav {
  padding: 30px;
```

```
}

#nav a {
  font-weight: bold;
  color: #2c3e50;
}

#nav a.router-link-exact-active {
  color: #42b983;
}
</style>
```

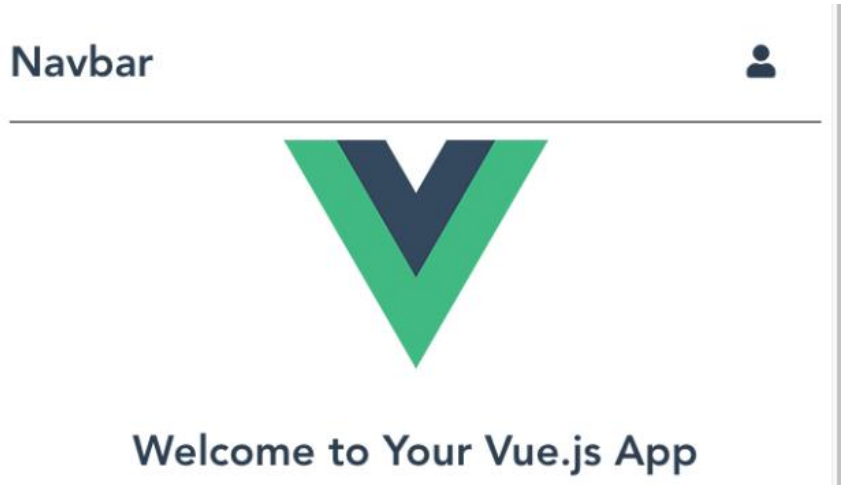
Iconos fontAwesome:

Si nos fijamos en la sección de style de App.vue, notaremos que hicimos un import de la librería a través de cdn.

```
@import "https://use.fontawesome.com/releases/v5.2.0/css/all.css";
```

Agregando estos íconos en el archivo principal, podremos acceder a él desde cualquier otro componente.

Si guardamos y levantamos nuestro proyecto, podremos observar.



Si nos posicionamos en el ícono, en la sección de navbar, podremos ver los datos del usuario asociado, que se encuentran en el almacén de Vuex.

EXERCISE 2: MISMOS DATOS Y DIFERENTES ENFOQUES EN COMPONENTES

INSTRUCCIONES

Lee con atención cada elemento que se presenta, y responde de acuerdo con lo solicitado. Para realizar el ejercicio, continuaremos trabajando en el proyecto creado anteriormente, llamado “cue11”.

Ejercicio

Para comenzar, vamos a agregar datos dentro del almacén de Vuex, éstos se llamarán “products”, y serán de tipo Array.

```
products:[
      {id:1, code:'23452323', product: "Funda papel" , quantity:2,
price:100},
      {id:2, code:'23443323', product: "Bandeja Carton" ,
quantity:2, price:200},
      {id:3, code:'24452329', product: "Papel de regalo" ,
quantity:2, price:500},
    ]
```

El archivo index.js de Vuex final, quedará de la siguiente manera.

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    user_id:"325",
    name: 'Juan',
    last_name: 'Ramirez',
    products:[
      {id:1, code:'23452323', product: "Funda papel" , quantity:2,
price:100},
      {id:2, code:'23443323', product: "Bandeja Carton" ,
quantity:2, price:200},
      {id:3, code:'24452329', product: "Papel de regalo" ,
quantity:2, price:500},
    ]
```

```
  },
  mutations: {
  },
  actions: {
  },
  modules: {
  }
})
```

Nuevos componentes

Para continuar, crearemos una nueva carpeta en components llamada “products”, y dentro de ella incluiremos 3 archivos:

- ProductCount.vue.
- ProductList.vue.
- ProductTotal.vue.

ProductCount.vue

Partiremos por el primer componente, el cual nos permitirá saber cuántos productos tenemos.

```
<template>
  <h1> Productos {{count}}</h1>
</template>

<script>
export default {
  name: 'Product-count',
```

```
  computed: {
    count(){
      return this.$store.state.products.length;
    }
  },
}
</script>

<style scoped>

</style>
```

ProductList.vue

Este componente será el encargado de mostrar los datos, a través de una pequeña grilla.

```
<template>
  <div>
    <div class="container">
      <div class="tab">Código</div>
      <div class="tab">Producto</div>
      <div class="tab">Cantidad</div>
      <div class="tab">Precio</div>
    </div>
    <div class="container" v-for="product in products"
:key="product.id">
      <div class="tab_content">{{product.code}}</div>
      <div class="tab_content">{{product.product}}</div>
      <div class="tab_content">{{product.quantity}}</div>
      <div class="tab_content">{{product.price}}</div>
    </div>
  </div>
</template>

<script>
export default {
  name: 'Product-list',
  // props: {},
  data: function(){
    return {}
  },
  computed: {
    products(){
      return this.$store.state.products
    }
  },
  methods: {
```

```
    // -- Metodos
  },
  // components: {},
}
</script>

<style scoped>
  .container{
    display:grid;
    grid-template-columns: 1fr 4fr 1fr 2fr;
  }
  .tab_content{
    background:orange;
  }
  .tab{
    background:grey;
    color:white;
  }
</style>
```

ProductTotal.vue

El ultimo componente, nos servirá para realizar la sumatoria general de los precios de los productos, multiplicados por su cantidad.

```
<template>
  <div class="container">
    <div>Total: ${{total}}</div>
  </div>
</template>
<script>
export default {
  name: 'ProductTotal',
  computed: {
    total(){
      return this.$store.state.products.reduce((total,prod)=>{
        return total + (prod.quantity * prod.price)
      },0)
    }
  },
}
</script>
<style scoped>
  .container{
    font-size:25px;
    background:grey;
```

```
    color:white;
    padding-top:10px;
    text-align: start;
  }
</style>

```

Vista Products.vue

Una vez creados los 3 componentes, es momento de unirlos en una sola vista. Para ello, crearemos un archivo llamado Products.vue, en la carpeta views.

Al estar listo, procederemos a importar los 3 componentes creados.

```
<template>
  <div>
    <ProductCount></ProductCount>
    <ProductList></ProductList>
    <ProductTotal></ProductTotal>
  </div>
</template>

<script>
import ProductCount from
'@/components/products/ProductCount.vue'
import ProductList from '@/components/products/ProductList.vue'
import ProductTotal from
'@/components/products/ProductTotal.vue'
export default {
  name: 'Product-view',
  components: {
    ProductCount,
```

```
    ProductList,
    ProductTotal
  },
}
</script>

<style scoped>

</style>
```

Ruta para Products.vue

Una vez creado el archivo Products.vue, lo que nos falta será crear una ruta para que éste pueda ser visualizado.

```
import Products from '@/views/Products.vue'
```

Dentro del Array routes, agregaremos un objeto.

```
{
    path:'/products',
    name:'Product',
    component:Products,
  },
```

Si guardamos nuestra aplicación, y vamos al navegador, visitaremos "*/products*".

EXERCISE 3: VUEX - MAPSTATE

INSTRUCCIONES

Para realizar el siguiente ejercicio, continuaremos trabajando con el mismo proyecto que hemos utilizado hasta ahora: “cue11”.

Ejercicio:

Una forma diferente de poder acceder a los datos de Vuex, es a través de mapState, éste nos provee una manera sencilla de invocar a los datos que necesitamos mostrar.

Si abrimos el archivo ProductList.vue, veremos que estamos accediendo al dato products a través de $store, una forma más sencilla de hacerlo sería importar mapState de Vuex, y luego utilizar el spreedOperator para adjuntar los datos.

Editando ProductList.vue

```
 <script>
import {mapState} from 'vuex'
export default {
  name: 'Product-list',
  // props: {},
  data: function(){
    return {}
  },
  computed: {
    ...mapState(['products'])
  },
  methods: {
    // -- Metodos
  },
  // components: {},
}
</script>
```

Con **mapState([‘nombre_estado’])**, podemos acceder a cualquier dato de Vuex.

Editando ProductCount.vue

```
<script>
import {mapState} from 'vuex';
export default {
  name: 'Product-count',
  // props: {},
  data: function(){
    return {}
  },
  computed: {
    ...mapState(['products']),
    count(){
      return this.products.length;
    }
  },
}
</script>
```

En este caso, no nos basta con el dato de Vuex, necesitamos obtener el largo del Array. Una de las formas en que podríamos hacerlo, es invocando a products con mapState, y luego crear una propiedad computada nueva para acceder al tamaño del Array.

Editando ProductTotal.vue

```
<script>
import {mapState} from 'vuex'
export default {
  name: 'ProductTotal',
  // props: {},
  data: function(){
    return {}
  },
  computed: {
    ...mapState(['products']),
    total(){
      return this.products.reduce((total,prod)=>{
        return total + (prod.quantity * prod.price)
      },0)
    }
  },
  methods: {// -- Metodos},
  // components: {},
}
</script>
```

En el caso de productTotal, necesitamos reducir los precios y la cantidad en un solo valor, para ello ocuparemos la función “reduce”.

Primero, obtenemos Products a través de masState, luego, en la segunda propiedad “count”, y la invocamos para utilizar el reduce.

Si guardamos y levantamos nuestra aplicación, veremos el mismo resultado obtenido en el EXERCISE 2.

Productos 3

Código	Producto	Cantidad	Precio
23452323	Funda papel	2	100
23443323	Bandeja Carton	2	200
24452329	Papel de regalo	2	500

Total: $1600