Introducción al desarrollo con Vue Js
Manejo de datos usando Vuex

¿Qué es Vuex?

Vuex es un patrón de gestión de estado - biblioteca para aplicaciones Vue.js. Sirve como una tienda centralizada para todos los componentes de una aplicación, con reglas que aseguran que el estado solo pueda mutarse de manera predecible. También se integra con la extensión de devtools oficial de Vue para proporcionar funciones avanzadas como la depuración de viaje en el tiempo de configuración cero y la exportación / importación de instantáneas de estado.

Nota: *antes de realizar este ejercicio se recomienda leer sobre el nuevo ciclo de vida de un componente con Vue utilizando Vuex.*

https://vuex.vuejs.org

Trabajemos un ejemplo con Vuex

Para ello vamos a recrear el ejemplo de la practica anterior solo que ahora trabajaremos con Vuex en lugar de props. Para esto vamos a crear tres archivos:

- Index.html
- Formulario.js
- Lista.js

Donde formulario va a ser nuestro componente principal y lista será el componente hijo dentro de formulario.

1. Añadiendo los CDN

Primero dentro de nuestro index html dentro del head vamos a agregar primero el cdn de Vue y luego vamos a buscar el google Vuex vamos a buscar el enlace al script en línea lo vamos a copiar y lo colocaremos dentro de una etiqueta script src.

```
9     <!-- Vue CDN -->
10    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
11
12    <!-- Vuex CDN -->
13    <script src="https://unpkg.com/vuex@3.1.2/dist/vuex.js"></script>
14
```

Ahora en la parte baja de nuestro documento vamos a enlazar con los componentes en el siguiente orden. Primero el componente padre y luego el hijo.

```
<!-- Componnte principal -->
<script src="formulario.js"></script>

<!-- Componente secundario -->
<script src="lista.js"></script>

```

2. Definiendo el primer componente.

Vamos a crear un input y un botón en nuestro témplate y vamos a instanciar al componente hijo.

```
Vue.component('formulario', {
    template:
    `
    <div>
        <h5>Registrar Personas con Vuex</h5>
        <input type="text" v-model="nombre" @keyup.enter="agregar">
        <button @click="agregar">Agregar</button>
        <p>
            <lista></lista>
        </p>
    </div>
    `,
```

Este componente va a manejar un dato al que llamaremos nombre.

```
    `,
    data() {
        return {
            nombre: ''
        }
    },
```

3. Generando nuestra instancia de Vuex.

En nuestro documento html justo bajo los enlaces de nuestros componentes vamos a bajar y abrir un par de etiquetas script, vamos a crear una constante llamada store donde crearemos nuestra instancia de Vuex.

Dentro de esta instancia de Vuex vamos a crear una propiedad llamada state (este va a ser el estado de nuestra aplicación de Vue) donde guardaremos los datos que deseamos manejar globalmente entre componentes, aquí crearemos un dato llamado nombres y lo igualaremos a un array vacío.

```
32      //Instancia del almacenamiento con Vuex
33      const store = new Vuex.Store({
34          state: {
35              nombres: []
36          },
```

4. **Generando nuestra instancia de Vue.**

Después de generar nuestra instancia de Vuex debemos asignar esta instancia a nuestra aplicación de Vue para elle, justo debajo de la instancia de Vuex vamos instanciar nuestra aplicación de Vue vamos a pasar la propiedad "*el"* y una nueva propiedad llamada *"store"* y le pasaremos el valor de nuestra variable que almacena la instancia de Vuex.

```
44      //Instancia de Vue
45      new Vue({
46          el: '#app',
47          store: store
48      })
```

5. **Generando nuestras mutaciones.**

Las mutaciones son los métodos (funciones) que realizan cambios en el estado de nuestra aplicación. En este ejemplo solo generaremos una mutación a la que llamaremos guardar.

Nota: *todas las mutaciones deben recibir como parámetro el state y si se desea enviar otro parámetro adicional, solamente se separarán por una como como en un lenguaje de programación convencional.*

Esta mutación recibirá dos parámetros, el state y un parámetro llamado n que será un nombre, y lo que hará esta mutación será añadir este nombre que estaremos recibiendo en el array nombres.

```
32      //Instancia del almacenamiento con Vuex
33      const store = new Vuex.Store({
34          state: {
35              nombres: []
36          },
37          mutations: {
38              guardar (state, n) {
39                  state.nombres.push(n)
40              }
41          }
42      })
43
```

6. Llamando los datos de Vuex a nivel global.

Como mencionamos al principio la utilidad de Vuex es la ventaja de poder llamar a los datos desde cualquier componente sin tanta complicación. Para llamar a los datos de Vuex desde un componente lo haremos con la propiedad computada (*computed*).

Dentro de esta propiedad computada vamos a llamar a la función Vuex.mapState y dentro de los paréntesis de la función abriremos corchetes y entre comillas llamaremos uno a uno a los datos del estado que necesitemos en eses componente.

Ya que nosotros solo tenemos un dato dentro de nuestro estado lo llamaremos solo a él.

Dentro del componente padre.

```
computed: {
    ...Vuex.mapState(['nombres'])
}
```

7. Llamando nuestras mutaciones de Vuex a nivel global.

Al igual que los datos del estado las mutaciones de Vuex las podemos llamar a nivel global a través de la propiedad methods de los componentes, dentro de esta llamamos a la función Vuex.mapMutations y procedemos a llamar una a una a las mutaciones que necesitamos en ese componente. Esta característica nos permite excluir mutaciones o datos del estado que no estaremos usando en ese componente ahorrando de esta manera recursos.

Dentro del componente padre.

```
18      methods: {
19          ...Vuex.mapMutations(['guardar']),
```

8. Creando el método agregar.

En nuestro témplate al botón y al input les indicamos que al presionar el botón o al hacer enter ejecute una función llamada agregar, sin embargo no hemos definido esta función, esta función la definiremos justo debajo de las mutaciones que mandamos a llamar.

```
18      methods: {
19          ...Vuex.mapMutations(['guardar']),
20          agregar() {
21              this.guardar(this.nombre)
22              this.nombre = ''
23          }
24      },
```

9. Definiendo el componente hijo – mapeando los registros.

Ahora solo resta definir nuestro componente hijo al que llamamos lista. Vamos a definir nada más dos propiedades su témplate y sus propiedades computadas para mandar a llamar los datos de estado con Vuex. Al llamar a los datos de Vuex con a través de la propiedad computada y queremos usarlo en nuestro témplate, basta con llamarlo por el nombre.

```
Vue.component('lista', {
    template:
    `
    <div>
        <ul>
            <li v-for="nombre in nombres">{{ nombre }}</li>
        </ul>
    </div>
    `,
    computed: {
        ...Vuex.mapState(['nombres'])
    }
})
```

Nota: *complete la funcionalidad de esta aplicación (editar y eliminar) utilizando el manejo de estados y mutaciones que nos ofrece Vuex.*