Desarrollo de Aplicaciones Web

Desarrollo web en entorno cliente

Tema 11: Formularios y otros aspectos

1. Formularios

En este apartado se va a hacer hincapié en algunos aspectos del uso de formularios en Vue.

1.1. A tener en cuenta

Ya se ha visto, que para tener sincronizado el formulario con los datos del componente se utiliza la directiva v-model. Al igual que los eventos, para v-model también hay modificadores, como, por ejemplo:

- .lazy: para que v-model sincronice al producirse el evento change(cuando el input pierda el foco) y no con cada tecla que se pulse.
- .number: convierte automáticamente el valor introducido a Number.
- .trim: ejecuta trim() sobre el texto introducido.

Además, cuando un campo de formulario está enlazado con v-model, el formulario ignorará los valores iniciales de los atributos value, checked y selected incluidos en la declaración del elemento html. Los valores que se tendrán en cuenta serán los indicados en data(), por lo que si se quieren incluir de primeras, deberán incluirse en el lado javascript de la relación.

1.2. Valores de campos de un formulario

A continuación, se va a ver qué valores se almacenan en la variable del componente relacionada, para cada tipo de elemento de un formulario:

input

Directamente relaciona el valor introducido por el usuario con la variable correspondiente del componente:

```
<input type="text" v-model="libro.titulo">
```

radio button

Se incluirá v-model, en todos los radio button del mismo grupo, relacionando el elemento con la variable correspondiente. El valor que se cargará en dicha variable será el incluido en el atributo value del elemento:

```
<input type="radio" value="V" name="tipo" v-model="libro.tipo">Venta
<input type="radio" value="A" name="tipo" v-model=" libro.tipo ">Alquiler
```

checkbox

Se relacionará el elemento con la variable mediante v-model y, si el elemento tiene atributo value, será este valor el que se almacene, siendo true o false lo almacenado en caso contrario:

```
<input type="checkbox" value="nuevo" v-model="libro.nuevo">Nuevo
<input type="checkbox" v-model="libro.tapaDura">Tapa dura
```

En el primero check el valor de libro.nuevo será “nuevo” en caso de ser marcado. PAra el segundo el valor de libro.tapaDura será true o false.

checkbox múltiple

Es posible almacenar el valor de mas de un checkbox en la misma variable, cuando ésta sea un array. En este caso habrá que incluir atributo value además de v-model:

```
<input type="checkbox" v-model="libro.idiomaDisp" value="eng">Inglés
<input type="checkbox" v-model="libro.categorias" value="esp">Español
<input type="checkbox" v-model="libro.categorias" value="fr">Francés
<input type="checkbox" v-model="libro.categorias" value="it">Italiano
```

En caso de seleccionar mas de uno, en cada posición del array se almacenará el valor del atributo value de los checkboxs seleccionados.

select

v-model se incluye en el <select> y almacenará el valor de la opción seleccionada:

```
<select v-model="libro.categoria">
        <option value="0">Infantil</option>
        <option value="1">Juvenil</option>
        <option value="2">Adulto</option>
</select>
```

Si se tratase de un select de opción múltiple, la variable donde se almacenasen los valores tendría que ser un array.

EJERCICIO 1

- Crea una página que tenga una vista de formulario. El formulario tendrá:
  - Un input para introducir el nombre.
  - Un grupo de checkboxs que se creará a partir de los datos de un array del componente, con aficiones varias.
  - Una lista desplegable que permita seleccionar un color favorito.
  - Una lista desplegable de selección múltiple para elegir ciudades favoritas.
- Los datos introducidos en el formulario, cuando éste se envíe, se almacenarán en un fichero json(preferenciasUsuario.json) en el servidor json-server que tengas arrancado en tu equipo.
- Comprueba que los valores almacenados en preferenciasUsuarios.json son los deseados.

2. Computed

Cuando se han visto las propiedades calculadas, dijimos que éstas son variables cuyo valor se ha de calcular a partir de otras del componente. Su principal ventaja es que se cachean, por lo que si se vuelven renderizar en el DOM no vuelven a evaluarse, a menos que cambie el valor de alguna de las variables reactivas que use.

Por defecto estas propiedades solo son de lectura, pero es posible definir para ellas métodos getter y setter, haciéndolas de lectura-escritura:

```
computed: {
        fullName: {
                get() {
                        return this.name + ' ' + this.surname;
                },
                set(newValue) {
                        const names = newValue.split(' ');
                        this.name = names[0];
                        this.surname = names[names.length - 1];
                }
        }
}
```

Con la sentencia this.fullName = 'Elena Serrano' se asignaría “Elena” a la variable name y “Serrano” a surname.

3. Watchers

Otra funcionalidad que ofrece Vue y no se ha visto hasta el momento, son los watchers. Se trata de una opción para casos concretos en los que es necesario controlar cuándo cambia de valor una variable reactiva para poder ejecutar cierto código en el momento en que dicha variable sufra una modificación.

Es una opción costosa, por lo que no es conveniente abusar de ella. Por ello, solo se utilizará cuando sea indispensable, comúnmente cuando se quiere llevar a cabo operaciones asíncronas o costosas. Por ejemplo, se suelen utilizar cuando se trabaja con rutas.

```
data() {
        return {
                name: 'John',
                surname: 'Doe',
                fullName: 'John Doe',
        }
},
watch: {
        name(newValue, oldValue) {
                this.fullName = newValue + ' ' + this.surname;
        },
        surname(newValue, oldValue) {
                this.fullName = this.name + ' ' + newValue;
        }
}
```

El watcher tendrá el mismo nombre que la variable que se quiere controlar, y recibirá dos parámetros:

- newValue : contendrá el último valor almacenado por la variable.
- oldValue : contiene el nuevo valor que va a tomar la variable.

4. Ciclo de vida del componente

Un componente pasa por distintos estados a lo largo de su ciclo de vida y podemos poner *hooks* para ejecutar una función cuando alcanza ese estado. Los principales *hooks* son:

- beforeCreate: aún no se ha creado el componente (sí la instancia de Vue) por lo que no tenemos acceso a sus variables, etc.
- created: se usa por ejemplo para realizar peticiones a servicios externos lo antes posible.
- beforeMount: ya se ha generado el componente y compilado su template.
- mounted: ahora ya tenemos acceso a todas las propiedades del componente. Es el sitio donde hacer una petición externa si el valor devuelto queremos asignarlo a una variable del componente.
- beforeUpdate: se ha modificado el componente, pero aún no se han renderizado los cambios.
- updated: los cambios ya se han renderizado en la página.
- beforeUnmount: antes de que se destruya el componente (en versiones anteriores a Vue3 beforeDestroy).
- unmounted: ya se ha destruido el componente (en versiones anteriores a Vue3 destroyed).

Componentes asíncronos

En proyectos grandes con centenares de componentes podemos hacer que en cada momento se carguen sólo los componentes necesarios de manera que se ahorra mucho tiempo de carga de la página.

Para que un componente se cargue asíncronamente al registrarlo se hace como un objeto que será una función que importe el componente. Un componente normal (síncrono) se registraría así:

```
import Articulo from './Articulo.vue'
export default {
        components: {
                Articulo
        },
}
```

Si queremos que se cargue asíncronamente no lo importamos hasta se registra:

```
export default {
        components: {
                Articulo: () => import('./ Articulo.vue'),
        },
…
```

5. Entornos y variables de entorno

En Vue hay, normalmente, 3 entornos o modos: **development**, **test** y **production**. Las variables de entorno se almacenarán en uno de los siguientes ficheros:

- **.env**: se cargan en todos los modos.
- **.env.local**: se cargan en todos los modos pero son ignoradas por git.
- **.env.[modo]**: se cargan sólo en todos el modo indicado.
- **.env.[modo].local**: ídem pero son ignoradas por git.

A tener en cuenta:

- Las variables se incluirán en formato pares clave=valor:
  NOMBRE=Proyecto Vue CLI
  VUE_APP_API=https://localhost/api
- Si el nombre de la variable comienza por VUE_APP_, ésta será accesible desde el código con **process.env.nombreVariable:**
  let urlAPI = process.env.VUE_APP_API;
- Es posible saber en qué entorno se está ejecutando la aplicación consultando el valor de la variable **process.env.NODE_ENV:**

```
if (process.env.NODE_ENV === "development") {
        // Estamos en modo desarrollo
        console.log("Modo desarrollo");
} else {
        // En producción
        console.log("Modo producción");
}
```

6. Autenticación con JWT

6.1. Introducción

Como ya sabemos, HTTP no ofrece opciones de mantener sesiones abiertas entre cliente y servidor. Por ello, se han de implementar mecanismos para poder mantener comunicación cliente-servidor sin que el usuario tenga que introducir sus credenciales a cada paso que dé.

Para añadir seguridad a un sitio web, la parte servidora será la encargada, a grandes rasgos, de:

- Comprobar que las credenciales del usuario son válidas, validándolas contra la base de datos, LDAP, etc.
- Asignar a la sesión, que en ese momento se abra, el o los datos necesarios para mantenerla abierta en el tiempo.
- Mantener los datos de sesión de forma segura.
- Comprobar, en cada petición realizada por el cliente, que efectivamente quien está realizando la petición es quien se autenticó en primer lugar.

En esta unidad vamos a ver como implementar la seguridad, en una aplicación Vue, con JWT (Json Web Token). Vamos a ver JWT para nuestras SPAs creadas con Vue porque es uno de los casos en los que JWT está recomendado.

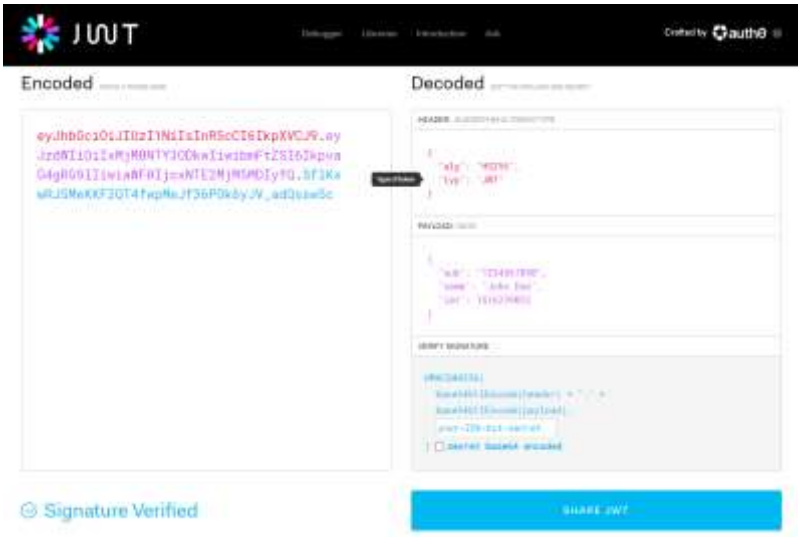
6.2. Autenticación con JWT

JWT (JSon Web Token) es un estándar abierto que permite compartir información de forma segura entre dos partes(cliente-servidor). Cada JWT, contendrá objetos JSon codificados, incluyendo una serie de aserciones o privilegios(claims). Los JWTs se firman mediante un algoritmo criptográfico para que dichos claims no puedan ser modificados tras la generación del token. Serán estos claims los que contengan información sobre la sesión establecida, como por ejemplo el periodo de validez del token, que permisos se han concedido al usuario, etc.

Un JWT es una cadena, formada por tres partes separadas por puntos(.), y codificada mediante base64:

- La cabecera (**header**): La cabecera contendrá, como mínimo, el tipo de token (JWT en este caso) y el algoritmo de firma.
- El cuerpo del mensaje (**payload**): En el cuerpo van los datos de usuario y los claims, información que habitualmente es utilizada por el servidor para verificar que el usuario tiene permisos para llevar a cabo la acción que solicita.
- La firma (**signature**): Elemento que garantiza que el token no ha sido modificado.

En la imagen que se incluye a continuación, es posible ver la decodificación de un token para visualizar la información que contiene. Esta imagen se ha sacado de la página jwt.io, en la cual se puede incluir un token y lo decodificará. En ella se pueden ver las tres partes del token separadas por puntos y la decodificación de las mismas en formato JSon: header, payload y signature.



6.3. Firma de un JWT

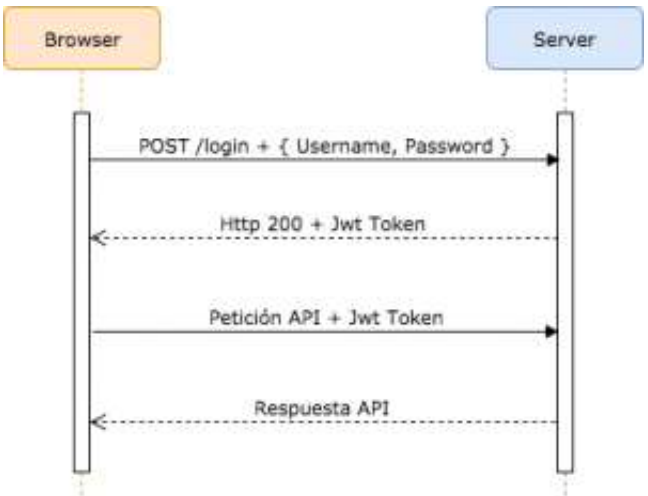
Teniendo en cuenta que, con herramientas como la anterior, es posible decodificar un token y ver su información, es la firma la que marca la diferencia, ya que se construye de tal forma que se podrá verificar que el cliente es quien dice ser y que el mensaje no ha sido modificado por terceros.

La función de codificación se aplica a:

- Codificación en Base64 de header.
- Codificación en Base64 de payload.
- Un secreto, establecido por la aplicación.

De esta forma, si alguien modifica el token inyectando claims o datos maliciosos, entonces el lado servidor podrá verificar que la comprobación de firma no es correcta y denegar la solicitud de recursos realizada, ya sea para obtener datos o modificarlos.

A mayores, y para cualquier sistema de seguridad no solo JWT, siempre debería utilizarse HTTPS y un servidor con certificado para hacer el despliegue de aplicaciones. La comunicación entre partes debería realizarse con HTTPS para encriptar el tráfico, de forma que, si alguien lo interceptara, el propio tráfico a través de HTTP sobre SSL, cifrara toda la comunicación, la del token y todo lo demás, y así añadir esa posibilidad de seguridad.



6.4. Introducción a Vuex

Vuex es una librería de gestión de estado para aplicaciones Vue. Su función principal es la de contener datos y funcione que vayan a ser compartidos entre los distintos componentes de la aplicación, como por ejemplo datos de sesión o autenticación.

En el Aula Virtual encontrarás un proyecto de ejemplo “Autenticación con JWT y Vuex”. A continuación, se van a detallar algunos de los elementos que vas a encontrar en el ejemplo:

- El elemento state es un estado para almacenar datos (en nuestro caso el token).
- El elemento mutation es el que se encargará de cambiar el token de nuestro estado, con el que recogemos del endpoint.
- Como tercer elemento, el action servirá para detallar la acción que llevará al cambio de valor de algunos datos del estado (en nuestro caso, de nuevo, el token).

Y, aunque en el ejemplo no está incluido, el token se suele almacenar haciendo uso de una de las opciones siguientes: localStorage, sessionStorage o Cookies.