

## VUE

viernes, 24 de octubre de 2025 9:06

VUE es un Framework SPA, es lo mismo que React y Angular.

Veremos los mismos conceptos, pero con otra sintaxis y alguna característica distinta.

Vue es un Framework libre, simplemente la comunidad se encarga de ir sacando mejoras y Nuevas versiones.

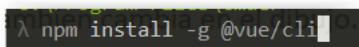
Dentro de VUE tenemos components, comunicación padre e hijo y tenemos props y métodos.

Una de las mayores diferencias está en que no tiene STATE.

Si cambiamos el valor de una variable, también cambia en el dibujo.

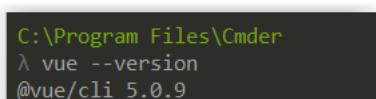
Lo primero es instalar VUE y configurar VS code.

`npm install -g @vue/cli`

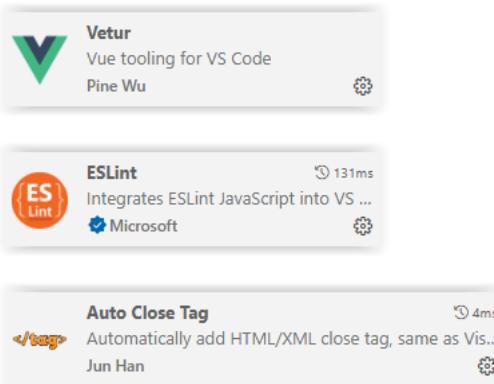


Para comprobar la versión una vez instalado...

`vue --version`



Instalamos las siguientes versiones de VS Code



Vamos a trabajar como hemos hecho en React.

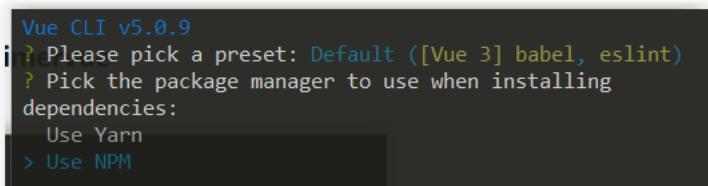
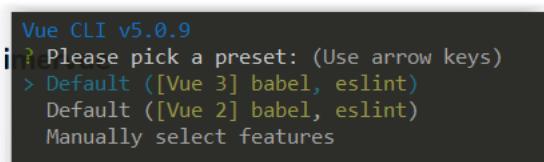
Tendremos una carpeta llamada `vue` y dentro múltiples proyectos

En vue no es imprescindible escribir en minúsculas, pero como no me gusta pensar, pondremos Todo en minúscula.

Para crear proyectos en VUE se realiza la siguiente instrucción:

`vue create NOMBREPROYECTO`

Creamos un nuevo proyecto llamado `primervue`



Una vez creado el proyecto, debemos entrar en la carpeta del proyecto y, para ejecutar el servidor, Utilizaremos el siguiente comando:

**npm run serve**

```
Successfully created project primervue. Ejecutar el servidor con:
  Get started with the following commands:
    $ cd primervue
    $ npm run serve
```

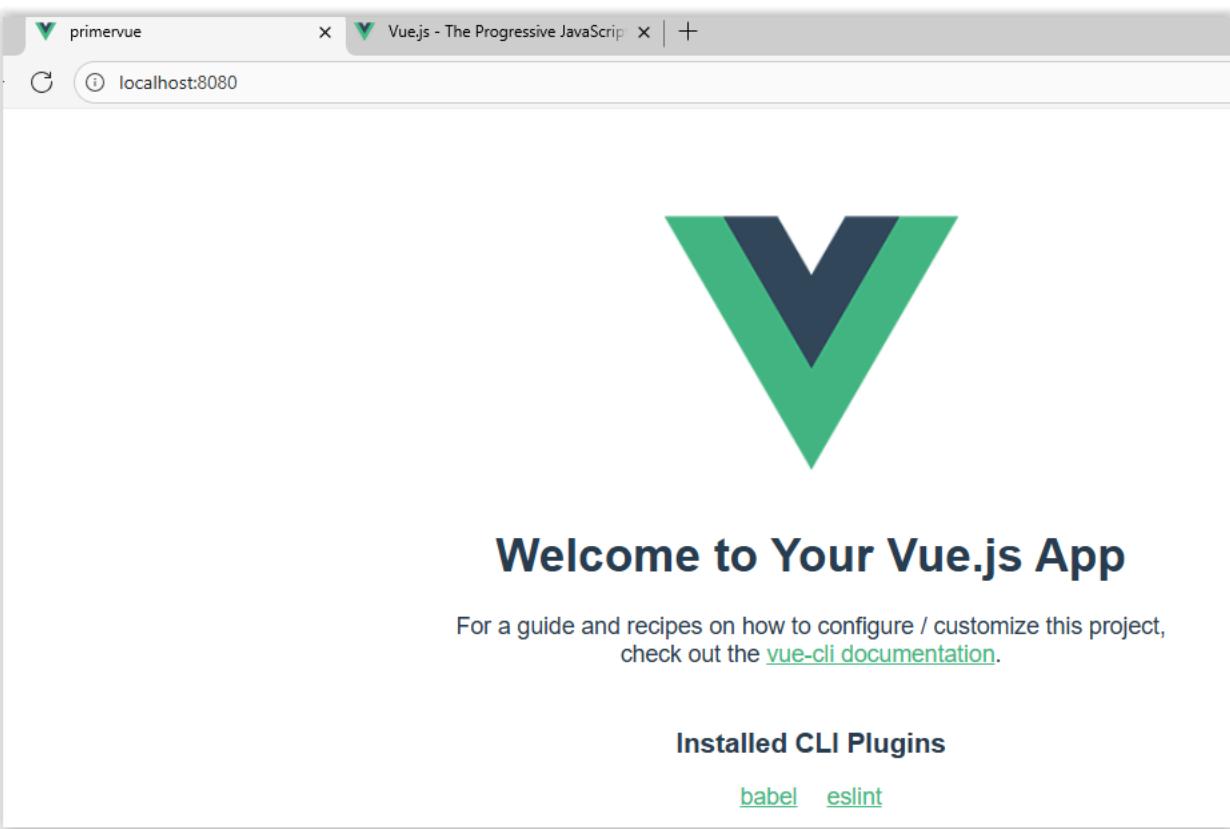
Para abrir VS Code con nuestro proyecto: **code .**

Nuestro servidor estará alojado en la siguiente URL:

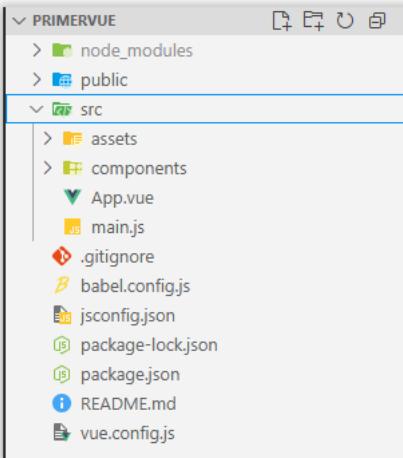
```
App running at:
- Local: http://localhost:8080/
- Network: http://10.203.1.31:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

Y ya tendremos el proyecto funcional.



Nos centraremos primero en la estructura del proyecto.



Si abrimos **public** veremos nuestra página inicial y única **index.html**

```
</head>
<body>
  <noscript>
    <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work without JavaScript</strong>
  </noscript>
  <div id="app"></div>
  <!-- built files will be auto injected -->
</body>
</html>
```

En este Framework, se dibuja el componente mediante la etiqueta `<div id="app"></div>`

Lo que nos interesa es la carpeta **src**

Tendremos una carpeta **assets** para nuestros iconos o cositas del front

En la carpeta **components** crearemos nuestros componentes propios con lógica

La extensión para los componentes es **.vue**

Comenzamos mirando **main.js**

```
import { createApp } from 'vue'
import App from './App.vue'

createApp(App).mount('#app')
```

La clase **MAIN** es el jefe de todo VUE, es el encargado de indicar si tendremos Rutas por ejemplo.

Un componente dentro de VUE está creado en TRES secciones:

1. **template:** Es el dibujo HTML de nuestro componente. Debe tener una etiqueta Parent como, Por ejemplo, un `<div>`

```
<template>
  
  <HelloWorld msg="Welcome to Your Vue.js App"/>
</template>
```

2. **script:** Es la lógica del componente, ahí incluiremos nuestros métodos, variables y **hooks** que son Los ciclos de vida de la página como, **componentDidMount**

```
<script>
import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'App',
  components: {
    HelloWorld
  }
}
</script>
```

3. **style:** Es el estilo opcional CSS de nuestro componente.

```
<style>
#app {
```

```

font-family: Avenir, Helvetica, Arial, sans-serif;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
text-align: center;
color: #2c3e50;
margin-top: 60px;
}
</style>

```

Por ahora, iremos trabajando como base con el componente **App.vue**

Sobre **components**, agregamos un nuevo componente llamado **HolaMundo.vue**

**HOLAMUNDO.VUE**

```

<template>
  <div>
    <h1>Mi primer vue de viernes, que ilusión!!!!</h1>
  </div>
</template>

<script>
  //VUE UTILIZA CODIGO JS (ESPECIAL)
  //CADA COMPONENTE QUE GENEREMOS DEBE TENER UN NOMBRE OBLIGATORIO
  //Y DEBEMOS EXPORTALO CON DICHO NOMBRE. EL name ES LA ETIQUETA QUE
  //UTILIZARAN OTROS COMPONENTES PARA DIBUJAR.
  export default {
    name: "HolaMundo"
  }
</script>
|
<style>
  h1 {color: blue}
</style>

```

El siguiente paso es utilizar este nuevo componente dentro de **App.vue**

Cualquier componente que utilicemos, primero debe ser **importado** dentro de **<script>**

**APP.VUE**

```

<script>
import HolaMundo from './components/HolaMundo.vue';
import HelloWorld from './components/HelloWorld.vue'

```

Cuando visualicemos la página, veremos un error GIGANTE

Compiled with problems:

ERROR

[eslint]

C:\Users\Profesor MCSD Mañana\Documents\FRONT\VUE\primervue\src\App.vue  
7:8 error 'HolaMundo' is defined but never used no-unused-vars

✖ 1 problem (1 error, 0 warnings)

Una característica diferente en VUE es que, si declaramos un elemento y, NO LO UTILIZAMOS, No permite compilar.

**Norma:** Todo lo declarado debe ser utilizado.

```
export default {
  name: 'App',
  components: {
    HelloWorld, HolaMundo
  }
}
</script>
```

Para utilizarlo, igual que en React en la zona de <template>

```
<template>
  

  <HolaMundo/>
</template>
```

Y veremos todo funcional.



## Mi primer vue de viernes, que ilusión!!!!

### VARIABLES DENTRO DE UN COMPONENT

Todo el código lógico siempre estará dentro de <script>

Dentro del script no podemos escribir alegramente, existen "zonas" dónde utilizar cada uno De los elementos con su significado.

La zona para declarar variables se llama **data()** y todo lo que declaremos lo podemos utilizar en el Component.

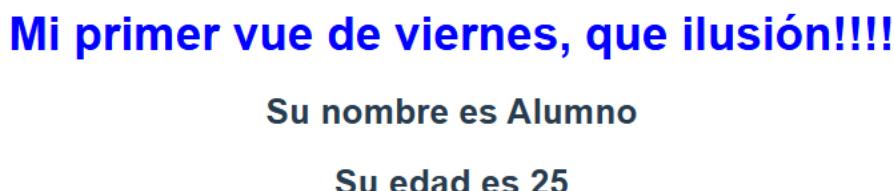
Debemos incluir **return** para utilizar dichas variables en el component.

```
export default {
  name: "HolaMundo",
  data() {
    //LAS VARIABLES SON DECLARADAS EN FORMATO JSON
    //DENTRO DE UN RETURN
    return {
      nombre: "Alumno",
      edad: 25,
      miArray: [],
      objeto: {}
    }
  }
}
```

Dichas variables se escriben con doble llave dentro del Template si queremos dibujarlas

```
<template>
  <div>
    <h1>Mi primer vue de viernes, que ilusión!!!!</h1>
    <h2>Su nombre es {{ nombre }}</h2>
    <h2>Su edad es {{ edad }}</h2>
  </div>
</template>
```

Podremos visualizar el resultado:



Mi primer vue de viernes, que ilusión!!!!  
Su nombre es Alumno  
Su edad es 25

#### UTILIZAR CSS EXTERNOS DESDE <STYLE>

Vamos a crear un CSS dentro de **assets** y veremos cómo utilizarlo en nuestro componente.

Creamos una carpeta llamada **css** dentro de **assets** y un fichero llamado **estilos.css**

```
img {
  width: "150px";
  height: "150px";
}

h2{
  background-color: yellow;
}
```

Para utilizar los CSS dentro de **<style>** se utiliza la instrucción **@import**

```
<style>
  h1 {color: blue}
  @import './assets/css/estilos.css'
</style>
```

Incluimos una imagen que NO existe en nuestro server:

```
<template>
  <div>
    <h1>Mi primer vue de viernes, que ilusión!!!!</h1>
    
    <h2>Su nombre es {{ nombre }}</h2>
    <h2>Su edad es {{ edad }}</h2>
  </div>
</template>
```

```
</uiv>
</template>
```

VUE comprueba todo en nuestro servidor incluidos elementos que NO sean compilables.

## Compiled with problems:

ERROR in ./src/components/HolaMundo.vue?vue&type=template&id=23be7e16 (.node\_modules/vue-loader/lib/index.js??clonedRuleSet-40.use[0]!./node\_modules/vue-loader/dist/templateLoader.js??ruleSet[1].rules[3]!./node\_modules/vue-loader/dist/index.js??ruleSet[0].use[0]!./src/components/HolaMundo.vue?vue&type=template&id=23be7e16) 2:0-55

Module not found: Error: Can't resolve './assets/images/vatman.jpg' in 'C:\Users\Profesor MCSD Mañana\Documents\FRONT\VUE\primervue\src\components'

### ROUTING CON VUE

**Nota:** En Vue los nombres de los componentes deben estar formados por dos palabras.

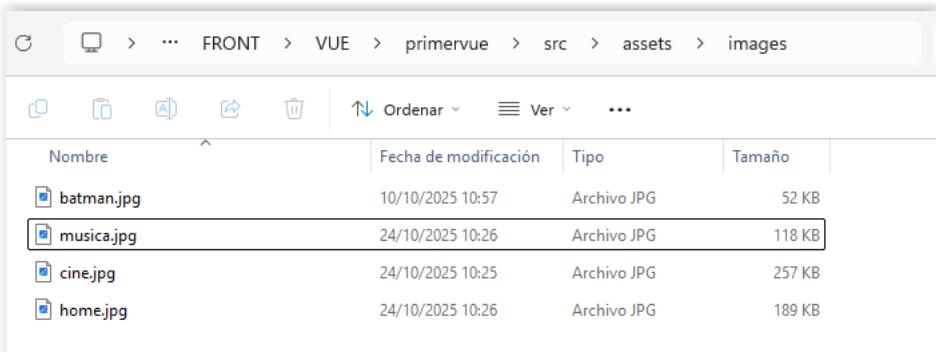
Routing no es nativo dentro de VUE, debemos instalar la librería dentro de nuestro proyecto

La lógica es la misma que hemos utilizado en React, navegar entre components.

Instalamos la librería de Routing para vue.

```
λ npm install vue-router@next --save
```

Incluimos la imágenes de Cine, Música y Home dentro de `src/assets/images`



Vamos a utilizar la palabra **Component** para nombrar a nuestros componentes.

Sobre **components** creamos los siguientes elementos:

CineComponent  
MusicaComponent  
HomeComponent

Los diseñamos con una imagen y un título.

```
<template>
  <div>
    <h1>Home Component</h1>
    
  </div>
</template>

<script>
  export default {
    name: "HomeComponent"
  }

```

```
</script>

<style>
  @import './assets/css/estilos.css'
</style>
```

Para navegar entre componentes es la misma lógica que en React.  
Necesitamos una clase llamada **Router.js**

Debemos realizar los siguientes pasos:

1. Importar cada uno de los components en la navegación

```
import HomeComponent...
import CineComponent...
```

2. Necesitamos un Array con cada ruta y cada component a representar en la navegación

```
const Rutas = [
  { path: "/", component: HomeComponent },
  { path: "/cine", component: CineComponent }
]
```

3. Debemos declarar una variable con las rutas y el tipo de navegación

```
const router = createRouter(history, routes: Rutas)
```

4. Habilitar dentro de main.js la navegación con estas rutas

```
createApp(App).use(router).mount()
```

Sobre la carpeta **src** creamos una nueva clase llamada **Router.js**

#### ROUTER.JS

```
//NECESITAMOS LAS LIBRERIAS DE NAVEGACION QUE HEMOS
//INCLUIDO CON npm install vue-router
import { createRouter, createWebHistory } from "vue-router";
import HomeComponent from "./components/HomeComponent.vue";
import CineComponent from "./components/CineComponent.vue";
import MusicaComponent from "./components/MusicaComponent.vue";
//UN ARRAY CON LAS RUTAS DE NAVEGACION
const myRoutes = [
  { path: "/", component: HomeComponent },
  { path: "/musica", component: MusicaComponent },
  { path: "/cine", component: CineComponent }
]
//CREAMOS UNA VARIABLE PARA EL router INDICANDO EL TIPO DE NAVEGACION Y
//LAS RUTAS
const router = createRouter({
  history: createWebHistory(),
  routes: myRoutes
})
//LA CONSTANTE router ES LA QUE UTILIZARA EL FICHERO main.js
//DEBEMOS EXPORTARLA PARA QUE SEA UTILIZADA
export default router;
```

Debemos indicar que utilizaremos **router** como Rutas dentro de **main.js**

#### MAIN.JS

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './Router'

createApp(App)
  .use(router)
  .mount('#app')
```

Para visualizar los componentes dentro de la navegación se debe hacer mediante la etiqueta  
**<router-view>**

Vamos a incluir un diseño en **App.vue**

#### APP.VUE

```
<template>
  <div>
    <h1 style="color:red">Elemento estático</h1>
    <hr/>
    <router-view></router-view>
  </div>
</template>
```

Por último, nos queda realizar la navegación mediante Links.

En VUE no se utiliza la etiqueta `<a>` para la navegación entre components.

Para navegar con Routing se utiliza la etiqueta `<router-link to="path">`

Creamos un nuevo component llamado `MenuComponent.vue`

`MENUCOMPONENT.VUE`

```
<template>
  <div>
    <ul id="menu">
      <li>
        <router-link to="/">Home</router-link> |
      </li>
      <li>
        <router-link to="/cine">Cine</router-link> |
      </li>
      <li>
        <router-link to="/musica">Música</router-link>
      </li>
    </ul>
  </div>
</template>
<script>
  export default {
    name: "MenuComponent"
  }
</script>
<style>
  ul#menu li {
    display: inline;
  }
</style>
```

Incluimos el menú dentro de `App.vue`

```
<template>
  <div>
    <h1 style="color:red">Elemento estático</h1>
    <MenuComponent/>
    <hr/>
    <router-view></router-view>
  </div>
</template>
<script>
import MenuComponent from './components/MenuComponent.vue';
export default {
  name: 'App',
  components: {
    MenuComponent
  }
}
</script>
<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

Deberíamos visualizar nuestra App funcional:

## Elemento estático

[Home](#) | [Cine](#) | [Música](#)

## Musica Component



Antes de continuar con más teoría, quiero un proyecto llamado **vuepruebas** en el que debemos repetir lo que hemos realizado:

- **Imágenes**
- **Variables data()**
- **CSS**
- **Rutas: 2/3 components y menú**

## HOOKS O CICLOS DE VIDA DE COMPONENTS

Un hook es como se denomina a un ciclo de vida. Tanto en React, como en Angular como en Vue.

Estamos hablando de **componentDidMount()** o **componentDidUpdate()**

Los Hooks no dejan de ser métodos incluidos dentro de nuestro component en la etiqueta **<script>**

```
<script>
  export default {
    name: "MiComponent",
    HOOKS
  }
</script>
```

Tenemos los siguientes métodos de Hooks:

- **mounted()**: Cuando el componente es montado y dibujado
- **unmounted()**: Cuando el componente es eliminado del Parent
- **created()**: Cuando el componente es montado, pero no dibujado todavía
- **updated()**: Cuando cambia nuestro dibujo HTML en el Template

## EVENTOS EN VUE

Para utilizar los eventos (un simple click de botón) tenemos dos formas de llamarlos:

1. **v-on:EVENTO**
2. **@evento**

En código:

```
<button v-on:click="metodo()">
<button @click="metodo()">
```

Dentro de VUE no podemos crear los métodos en cualquier lugar, la organización dentro de **<script>**  
Es una característica

Para poder declarar métodos dentro de VUE utilizamos la "zona": **methods**

Vamos a crear un componente para visualizar que las variables son reactivas y mostramos  
Console.log con los ciclos de vida.

Creamos un nuevo componente llamado **CicloVida.vue**

Console	
Component created	Ci
Component mounted	Ci
Component updated	Ci
Component unmounted	Ci
Component created	Ci
Component mounted	Ci
Component updated	Ci

Mostrar mensaje



## CICLOVIDA.VUE

```
<template>
  <div>
    <h1>Ciclo vida y métodos VUE</h1>
    <h2 style="color:red">{{ mensaje }}</h2>
    <button @click="cambiarMensaje()">
      Mostrar mensaje
    </button>
  </div>
</template>
<script>
  export default {
    name: "CicloVida",
    data() {
      return {
        mensaje: "Hoy es viernes!!!"
      }
    },
    methods: {
      //AQUÍ INCLUIMOS NUESTROS METODOS PERSONALIZADOS
      cambiarMensaje() {
        //PARA ACCEDER A LAS VARIABLES DE data() DEL COMPONENTE
        //DEBEMOS UTILIZAR LA PALABRA this
        this.mensaje = "Hoy se sale!!!";
      }
    },
    mounted() {
      console.log("Component mounted")
    },
    created() {
      console.log("Component created")
    },
    updated() {
      console.log("Component updated")
    },
    unmounted() {
      console.log("Component unmounted")
    }
  }
</script>
```

## DIRECTIVAS VUE

Una directiva es poder escribir código lógico dentro la zona del dibujo HTML del framework.

Por ejemplo, en React:

```
{
  this.state.algo &&
  <h1>
```

Las directivas de VUE se escriben dentro la etiqueta `<template>`

Nos permiten realizar IF y Bucles dentro del código.

La palabra para realizar directivas es: **v-**

IF: **v-if**  
FOR: **v-for**

Ejemplo:

```
<h1 v-if={variable == true}>SOLAMENTE ME VES CON true</h1>
<h1 v-if-else-if={variable == false}>SOLO CON EL IF</h1>
```

Para complementar la teoría vamos a funcionar con formularios.

En un formulario, podemos enlazar variables mediante la directiva **v-model**

Enlazar lo que quiere decir es que será bidireccional el cambio, si modifico una caja, la Variable asociada es modificada también.

Si modifico la variable por código, el dibujo de la caja (variable) es modificado.

```
<input type="text" v-model={modelo}/>
```

A este concepto se le llama **binding**

Creamos un nuevo componente llamado **DirectivasComponent.vue**

Vamos a dibujar todo el contenido de DATA.

Para dibujar todo el contenido, se utiliza el dólar para visualizar elementos de VUE.

```
Onenote > DirectivasComponent.vue > Vue (Official) > template > div
<template>
  <div>
    <h1>Ejemplo directivas</h1>
    <h3>{{ $data }}</h3>
    <label>Introduzca un número</label>
    <input type="text" v-model="numero"/>
  </div>
</template>

<script>
  export default {
    name: "DirectivasComponent",
    data() {
      return {
        numero: 0,
        nombre: "Alumno"
      }
    }
</script>
```

Veremos lo siguiente:

[Home](#) | [Directivas](#) | [Ciclo vida](#) | [Cine](#) | [Música](#)

## Ejemplo directivas

{ "numero": 0, "nombre": "Alumno" }

Introduzca un número

Si cambiamos el valor de la variable mediante **BINDING...**

[Home](#) | [Directivas](#) | [Ciclo vida](#) | [Cine](#) | [Música](#)

## Ejemplo directivas

{ "numero": "09", "nombre": "Alumno" }

Introduzca un número

En VUE son importantes los TYPES de los objetos de formulario si utilizamos binding.  
Como podemos comprobar, hemos puesto type="text" y el valor de la variable ha cambiado a string.

```
<label>Introduzca un número</label>
<input type="number" v-model="numero"/>
```

## Ejemplo directivas

{ "numero": 9, "nombre": "Alumno" }

Introduzca un número

9

POSITIVO

DIRECTIVASCOMPONENT.VUE

```
<template>
  <div>
    <h1>Ejemplo directivas</h1>
    <h3>{{ $data }}</h3>
    <label>Introduzca un número</label>
    <input type="number" v-model="numero"/>
    <h3 v-if="numero > 0" style="color:green">POSITIVO</h3>
    <h3 v-else-if="numero < 0" style="color:red">NEGATIVO</h3>
    <h3 v-else style="color:darkblue">CERO</h3>
  </div>
</template>
<script>
  export default {
    name: "DirectivasComponent",
    data() {
      return {
        numero: 0,
        nombre: "Alumno"
      }
    }
  }
</script>
```

En este mismo ejemplo, vamos a visualizar un Bucle sobre objetos.

Dentro de los bucles, como en React, se recorren los elementos con variables de **Referencia**, como **map** o **un bucle of** de Javascript.

```
for (var REFERENCIA of CONJUNTO){
```

```
}
```

```
this.state.CONJUNTO.map((referencia, index) => {
```

```
)}
```

En la declaración de la directiva debemos declarar la variable de referencia.

Para indicar la variable de referencia se utiliza un atributo de VUE que es mediante los dos puntos llamado **key**

Ejemplo:

```
<h1 v-for="CONJUNTO" :key="variableReferencia">
  {{variableReferencia}}
</h1>
```

Incluimos un Array dentro de **data()**

```
name: "DirectivasComponent",
data() {
  return {
    series: ["Futurama", "Mindhunter", "The Boys", "Stranger Things"],
```

```

        numero: 0,
        nombre: "Alumno"
    }
}

```

Realizamos un dibujo en template

```

<h1>Ejemplo directivas</h1>
<h3>{{ $data }}</h3>
<ul v-if="series.length > 0">
    <li v-for="serie in series" :key="serie">
        {{ serie }}
    </li>
</ul>

```

Y podremos visualizar el dibujo con el recorrido

## Ejemplo directivas

```
[ "Futurama", "Mindhunter", "The Boys", "Stranger Things", "numero": 0, "nombre": "Alumno" ]
```

Futurama  
Mindhunter  
The Boys  
Stranger Things

Vamos a incluir una nueva caja para intentar añadir una nueva serie mediante un método y un botón

## Ejemplo directivas

```
[ "Futurama", "Mindhunter", "The Boys", "Stranger Things", "Eureka!!!" ], "numero": 0, "nombre": "Alumno", "series": [ "Eureka!!!" ]
```

Introduzca serie  Nueva serie???

Futurama  
Mindhunter  
The Boys  
Stranger Things  
House???  
Eureka!!!

DIRECTIVASCOMPONENT.VUE

```
<template>
<div>
    <h1>Ejemplo directivas</h1>
```

```

<h3>{{ $data }}</h3>
<label>Introduzca serie</label>
<input type="text" v-model="serieNueva"/>
<button @click="nuevaSerie()">
    Nueva serie???
</button>
<ul v-if="series.length > 0">
    <li v-for="serie in series" :key="serie">
        {{ serie }}
    </li>
</ul>
<label>Introduzca un número</label>
<input type="number" v-model="numero"/>
<h3 v-if="numero > 0" style="color:green">POSITIVO</h3>
<h3 v-else-if="numero < 0" style="color:red">NEGATIVO</h3>
<h3 v-else style="color:darkblue">CERO</h3>
</div>
</template>
<script>
    export default {
        name: "DirectivasComponent",
        data() {
            return {
                series: ["Futurama", "Mindhunter", "The Boys", "Stranger Things"],
                numero: 0,
                nombre: "Alumno",
                serieNueva: ""
            }
        },
        methods: {
            nuevaSerie() {
                this.series.push(this.serieNueva);
            }
        }
    }
</script>

```

Ejemplo Número Par/Impar

- Necesitamos un nuevo componente llamado **ParImpar.vue**
- Tendremos la posibilidad de escribir un número en una caja de texto
- Tendremos dos funcionalidades:
  - Mediante un botón, generamos un random y debería decirnos si es Par o Impar
  - Cambiando el valor de la caja, la funcionalidad será la misma



**REAL MADRID**

# Al Clásico en BMV

# Par Impar

Introduzca número 36

**PAR**

Generar número

## PARIMPAR.VUE

```
<template>
  <div>
    <h1>Par Impar</h1>
    <label>Introduzca número</label>
    <input type="number" v-model="numero"/>
    <h1 v-if="numero % 2 == 0">PAR</h1>
    <h1 v-else>IMPAR</h1>
    <button v-on:click="generarRandom()">
      Generar número
    </button>
  </div>
</template>

<script>
  export default {
    name: "ParImpar",
    data() {
      return {
        numero: 4
      }
    },
    methods: {
      generarRandom() {
        this.numero = parseInt(Math.random() * 50) + 1
      }
    }
  }
</script>
```

## PROPIEDADES CONMUTADAS

Este concepto no existe en React.

Es un método que se utiliza como propiedad y puede devolver algún valor.

Es una herramienta dentro de VUE que nos permitirá generar código HTML dinámico en un método

Parece un método, pero es una propiedad. Se escribe sin paréntesis

No pueden recibir parámetros

Sirve para combinar código con directivas y poder realizar dibujos más complejos

Método con código HTML

```
evaluarNumero() {
  var resultado = "";
  if (this.numero > 0) {
    resultado = "<h1>POSITIVO</h1>";
  } else {
    resultado = "<h1>NEGATIVO</h1>";
  }
  return resultado;
}
```

Podemos invocar a este código, pero cómo lo dibujamos??

```
<button @click="evaluarNumero()>
```

Tenemos una directiva que nos permite dibujar código HTML y su traducción.

```
<div v-html="resultado">
```

Una propiedad conmutada nos permite generar código HTML y evaluar algo con un **return**, quitando código de directivas dentro del template.

Pongamos que tenemos un bucle:

```
<div v-for="num in numeros" key="num">
```

```
<div v-for="num in numeros" :key="num">
    <h1 v-if="num > 0">POSITIVO</h1>
    <h1 v-else-if="num < 0">NEGATIVO</h1>
    <h1 v-else>CERO</h1>
</div>
```

Ejemplo con propiedad comutada:

```
<div v-for="num in numeros" :key="num">
    <span evaluarNumero></span>
</div>
```

Una propiedad comutada SIEMPRE debe devolver un **return**

Vamos a crear un ejemplo llamado **PropiedadComutada.vue**

# Propiedad comutada

- Parchis
- Canicas
- Luz verde, luz roja
- La galleta
- <span style='color:red'>Parchis</span>
- <span style='color:red'>Canicas</span>
- <span style='color:red'>Luz verde, luz roja</span>
- <span style='color:red'>La galleta</span>

PROPIEDADCONMUTADA.VUE

```
<template>
<div>
    <h1>Propiedad comutada</h1>
    <ul>
        <li v-for="game in juegosRojos" :key="game" v-html="game">
        </li>
    </ul>
    <ul>
        <li v-for="game in juegos" :key="game">
            {{ game }}
        </li>
    </ul>
</div>
</template>
<script>
    export default {
        name: "PropiedadComutada",
        computed: {
            //SE ESCRIBE COMO UN METODO, PERO SE UTILIZA COMO UNA PROPIEDAD
            juegosRojos() {
                var datos = this.juegos;
                var i = 0;
                for (var game of this.juegos){
                    var juegoRojo = "<span style='color:red'>" +
                        game + "</span>";
                    datos[i] = juegoRojo;
                    i++;
                }
                return datos;
            },
            data() {
                return {
                    juegos: ["Parchis", "Canicas", "Luz verde, luz roja", "La galleta"]
                }
            }
        }
    }
</script>
```

&lt;/script&gt;

Dentro del mismo ejemplo vamos a realizar una propiedad conmutada para evaluar si un número es Par o Impar.

# Propiedad conmutada

Introduzca un número

**Par: 56**

PROPIEDADCONMUTADA.VUE

```
<template>
  <div>
    <h1>Propiedad conmutada</h1>
    <label>Introduzca un número</label>
    <input type="number" v-model="numero"/>
    <div v-html="evaluarNumero"></div>
    <ul>
      <li v-for="game in juegosRojos" :key="game" v-html="game">
      </li>
    </ul>
    <ul>
      <li v-for="game in juegos" :key="game">
        {{ game }}
      </li>
    </ul>
  </div>
</template>
<script>
  export default {
    name: "PropiedadConmutada",
    computed: {
      evaluarNumero() {
        let data = "";
        if (this.numero % 2 == 0) {
          data = "<h4 style='color:green'>Par: " + this.numero + "</h4>";
        } else {
          data = "<h4 style='color:red'>Impar: " + this.numero + "</h4>";
        }
        return data;
      },
      //SE ESCRIBE COMO UN METODO, PERO SE UTILIZA COMO UNA PROPIEDAD
      juegosRojos() {
        var datos = this.juegos;
        var i = 0;
        for (var game of this.juegos) {
          var juegoRojo = "<span style='color:red'>" +
            game + "</span>";
          datos[i] = juegoRojo;
          i++;
        }
        return datos;
      }
    },
    data() {
      return {
        numero: 0,
        juegos: ["Parchis", "Canicas", "Luz verde, luz roja", "La galleta"]
      }
    }
  }
</script>
```

## METODOS FILTERS GLOBALES

Esta funcionalidad nos permite tener métodos que podemos reutilizar en múltiples components.

Por ejemplo, acabamos de visualizar las propiedades conmutadas, dichas propiedades tienen que Estar dentro del propio component.

Si necesitamos aplicar, por ejemplo, Par/Impar en otros components, necesitamos "copiar" el Código de **computed**

La solución está en utilizar filtros globales, que no dejan de ser métodos globales

La llamada a dichos métodos podemos hacerla desde el <template>

Dichos métodos se declaran dentro de **main.js**

```
js main.js
1 import { createApp } from 'vue'
2 import App from './App.vue'
3 import router from './Router'
4
5 createApp(App)
6 .use(router)
7 .mount('#app')
```

Para declarar los métodos necesitamos una "zona" declarada como Global filters

```
app.config.globalProperties.$filters = {
    METODOS GLOBALES
}
```

Para utilizar dichos métodos dentro de un template:

```
{{ $filters.miMetodoGlobal() }}
```

Vamos a realizar un nuevo componente llamado **MetodosFilters.vue**

Vamos a realizar algo tan sencillo como poner un texto a mayúsculas y devolver el Doble de un número

## Métodos filters

Número: 55, Doble: 110  
 Número: 21, Doble: 42  
 Número: 33, Doble: 66  
 Número: 44, Doble: 88  
 Número: 88, Doble: 176

LUCIA  
 DIANA  
 ANTONIA  
 CARLOS

Comenzamos escribiendo un par de métodos dentro de **main.js**

**MAIN.JS**

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './Router'

var app = createApp(App)
app.config.globalProperties.$filters = {
    //CREAMOS DOS METODOS QUE RECIBIRAN "ALGO"
    //Y DEVOLVERAN UN RESULTADO
    mayuscula(dato) {
        return dato.toUpperCase();
    },
    getNumeroDoble(numero){
        return numero * 2;
    }
}
```

}

```
app.use(router).mount('#app')
```

#### METODOSFILTERS.VUE

```
<template>
  <div>
    <h1>Métodos filters</h1>
    <ul>
      <li v-for="num in numeros" :key="num">
        Número: {{ num }},
        Doble: {{ $filters.getNumeroDoble(num) }}
      </li>
    </ul>
    <ul>
      <li v-for="name in nombres" :key="name">
        {{ $filters.mayuscula(name) }}
      </li>
    </ul>
  </div>
</template>
<script>
  export default {
    name: "MetodosFilters",
    data() {
      return {
        numeros: [55,21,33,44,88],
        nombres: ["Lucia", "Diana", "Antonia", "Carlos"]
      }
    }
</script>
```

Creamos un nuevo proyecto llamado **vuelunes**

Necesito un **HomeComponent** y habilitar la navegación

Necesitamos un **Router** y un **MenuComponent**

Tendremos otro componente llamado **CollatzComponent** y realizamos la conjetura de Collatz  
Con una caja y un botón

"Todo número positivo será siempre cero siguiendo estas instrucciones"

- Si el número es par, dividimos entre dos
- Si el número es impar, multiplicamos por tres y sumamos uno

#### Versión 2:

Mediante un **FILTER GLOBAL**, nos devolverá un dibujo rojo o verde dependiendo si el número  
De Collatz es PAR o IMPAR.



#### MAIN.JS

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './Router'
var app = createApp(App);
```

```
app.config.globalProperties.$filters = {
    evaluarNumero(num) {
        if (num % 2 == 0){
            return "<span style='color:green'>" + num + "</span>";
        }else{
            return "<span style='color:red'>" + num + "</span>";
        }
    }
}
app.use(router).mount('#app')
```

## COLLATZCOMPONENT.VUE

```
<template>
<div>
    <h1>Collatz</h1>
    <label>Introduzca número</label>
    <input type="number" v-model="numero"/>
    <button @click="generarCollatz()">
        Generar Collatz
    </button>
    <ul>
        <li v-for="num in numerosCollatz" :key="num"
            v-html="$filters.evaluarNumero(num)">
        </li>
    </ul>
</div>
</template>
<script>
    export default {
        name: "CollatzComponent",
        data() {
            return {
                numero: 0,
                numerosCollatz: []
            }
        },
        methods: {
            generarCollatz() {
                let aux = [];
                while (this.numero != 1){
                    if (this.numero % 2 == 0){
                        this.numero = this.numero / 2;
                    }else{
                        this.numero = this.numero * 3 + 1;
                    }
                    aux.push(this.numero);
                }
                this.numerosCollatz = aux;
            }
        }
    }
</script>
```

El siguiente ejemplo seguimos con un clásico.

Creamos un componente llamado **TablaMultiplicar.vue**

Debemos dibujar una tabla con la operación y el resultado.

Realizamos el ejemplo de varias formas:

1. Un botón y un método que genere el código HTML y dibuje la tabla.
2. Generamos otra tabla mediante solo directivas, bucle contador y dibujo.
3. Generamos la tabla utilizando Filters, un filtro que sea `getOperacion()` y otro que sea `getResultado` y nos devuelva el resultado de la multiplicación para dibujarlo en la tabla

# Tabla multiplicar

Tabla multiplicar

Operación	Resultado
80*1	80
80*2	160
80*3	240
80*4	320
80*5	400
80*6	480

## MAIN.JS

```

import { createApp } from 'vue'
import App from './App.vue'
import router from './Router'
var app = createApp(App);
app.config.globalProperties.$filters = {
    evaluarNumero(num) {
        if (num % 2 == 0){
            return "<span style='color:green'>" + num + "</span>";
        }else{
            return "<span style='color:red'>" + num + "</span>";
        }
    },
    getResultado(numero, multi) {
        return numero * multi;
    },
    getOperacion(numero, multi) {
        return numero + " * " + multi;
    }
}
app.use(router).mount('#app')

```

## TABLAMULTIPLICAR.VUE

```

<template>
    <div>
        <h1>Tabla multiplicar</h1>
        <label>Introduzca un número</label>
        <input type="number" v-model="numero"/>
        <button v-on:click="generarTabla()">
            Tabla multiplicar
        </button>
        <table border="1">
            <thead>
                <tr>
                    <th>Operación</th>
                    <th>Resultado</th>
                </tr>
            </thead>
            <tbody v-html="html"></tbody>
        </table>
        <table border="1">
            <thead>
                <tr>
                    <th>Operación</th>
                    <th>Resultado</th>
                </tr>
            </thead>
            <tbody v-if="numero != 0">
                <tr v-for="contador in 10" :key="contador">
                    <td>
                        {{ numero + " * " + contador }}
                    </td>
                    <td>
                        {{ numero * contador }}
                    </td>
                </tr>
            </tbody>
        </table>
        <table border="1" style="background-color: lightgreen;">
            <thead>
                <tr>
                    <th>Operación</th>
                    <th>Resultado</th>
                </tr>
            </thead>
            <tbody v-if="numero != 0">
                <tr v-for="contador in 10" :key="contador">
                    <td>{{ $filters.getOperacion(numero, contador) }}</td>
                    <td>{{ $filters.getResultado(numero, contador) }}</td>
                </tr>
            </tbody>
        </table>
    </div>
</template>
<script>
    export default {
        name: "TablaMultiplicar",
        data() {
            return {
                numero: 0,
                html: ""
            }
        },
        methods: {
            generarTabla() {
                let aux = "";
                for (var i = 1; i <= 10; i++){
                    aux += "<tr>";
                    aux += "<td>" + this.numero + "*" + i + "</td>";
                    aux += "<td>" + (this.numero * i) + "</td>";
                    aux += "</tr>";
                }
            }
        }
    }
</script>

```

```

        }
        this.html = aux;
    }
}
</script>

```

## COMUNICACION COMPONENTES

La comunicación entre componentes es igual que **React**

De padre a hijo se utiliza **props**

Para poder enviar props desde un padre a un hijo debemos indicarlo mediante dos puntos  
Previos en el atributo. (No siempre)

- Si props es estático, es decir, no es una variable, no utilizamos dos puntos.
- Si props es dinámico, es decir, una variable enlazada, debemos utilizar los dos puntos.

Props se captura, en el hijo, dentro de **export default**

Con una sintaxis parecida a desestructurar.

### PADRE

```
<Hijo :dato1="valor1" :dato2="valor2"/>
```

### HIJO

```

<scripts>
    export default {
        name: "SoyUnHijo",
        props: [dato1, dato2]
    }
</scripts>

```

```

<template>
    <h1>{{ dato1 }}</h1>
</template>

```

Creamos un nuevo proyecto llamado **vuecomunicacioncomponents**

Vamos a tener dos components:

1. **PadreDeportes**: Será el componente padre que contendrá los hijos y enviará **props**
2. **HijoDeporte**: Será el componente hijo y dibujará los **props** que reciba del padre.

# Padre deportes

Curling

Peonza

Canicas

Bolos

### HIJODEPORTE

```

<template>
    <div>
        <h4 style="color:red">{{ nombredeporte }}</h4>
    </div>
</template>
<script>
    export default {
        name: "HijoDeporte",
        props: ["nombredeporte"],
        mounted() {
            console.log(this.nombredeporte)
        }
    }
</script>

```

### PADREDEPORTES

```

<template>
  <div>
    <h1 style="color:blue">Padre deportes</h1>
    <ul>
      <li v-for="sport in deportes" :key="sport">
        <HijoDeporte :nombredeporte="sport"/>
      </li>
    </ul>
  </div>
</template>
<script>
import HijoDeporte from './HijoDeporte.vue';
export default {
  name: "PadreDeportes",
  components: {
    HijoDeporte
  },
  data() {
    return {
      deportes: ["Curling", "Peonza", "Canicas", "Bolos"]
    }
  }
}
</script>

```

## COMUNICACIÓN HIJO PARENT

Esta funcionalidad se realiza mediante métodos.

Tendremos un método en el padre y un método en el hijo

Para enviar el método desde el padre al hijo es necesario hacerlo mediante **props**

La funcionalidad es la siguiente:

1. Método en el Parent

metodoPadre()

1. Método en el Hijo

metodoHijo()

Para realizar la llamada desde el Hijo al Parent debemos hacerlo con la palabra clave **\$emit**

```

metodoHijo() {
  $emit(metodoPadre())
}

```

Desde el padre, debemos enviar, mediante props el método a invocar

## PADRECOMPONENT

<HijoComponent :propiedad="valor" v-on:metodoPadre="metodoPadre"/>

## HIJOCOMPONENT

```

<script>
  export default {
    name: "HijoComponent",
    methods: {
      metodoHijo() {
        this.$emit(metodoPadre())
      }
    }
  }
</script>

```

Vamos a implementar la funcionalidad dentro del proyecto de comunicación.

Haremos un botón en el HijoDeporte para seleccionar un favorito.

Enviremos al padre el propio deporte seleccionado.

# Padre deportes

Su deporte favorito es Canicas

Curling



#### PADREDEPORTES

```
<template>
  <div>
    <h1 style="color:blue">Padre deportes</h1>
    <h2 style="background-color: yellow;" v-if="favorito">
      Su deporte favorito es {{ favorito }}
    </h2>
    <ul>
      <li v-for="sport in deportes" :key="sport">
        <HijoDeporte :nombredeporte="sport"
          v-on:seleccionarFavoritoParent="seleccionarFavoritoParent"/>
      </li>
    </ul>
  </div>
</template>
<script>
import HijoDeporte from './HijoDeporte.vue';
export default {
  name: "PadreDeportes",
  components: {
    HijoDeporte
  },
  data() {
    return {
      deportes: ["Curling", "Peonza", "Canicas", "Bolos"],
      favorito: null
    }
  },
  methods: {
    seleccionarFavoritoParent(deporteFavorito) {
      console.log("Yo soy tu parent")
      this.favorito = deporteFavorito;
    }
  }
}
</script>
```

#### HIJODEPORTE

```
<template>
  <div>
    <h4 style="color:red">{{ nombredeporte }}</h4>
    <button @click="seleccionarFavorito">
      Seleccionar favorito
    </button>
  </div>
</template>
<script>
export default {
  name: "HijoDeporte",
  props: ["nombredeporte"]
  , mounted() {
    console.log(this.nombredeporte)
  },
  methods: {
    seleccionarFavorito() {
      console.log("Soy el hijo " + this.nombredeporte);
      //PARA ENVIAR INFORMACION SE REALIZA A PARTIR DE LA LLAMADA
      //CON CADA PARAMETRO SEPARADO CON COMAS, DENTRO DEL METODO
      // $emit
      this.$emit("seleccionarFavoritoParent", this.nombredeporte);
    }
  }
}
</script>
```

#### PRACTICA COMUNICACIÓN PADRE HIJO

- Realizar un component **PadreNumeros** que dibujará components **HijoNumero** con números en un Array.
- Tendremos la posibilidad de generar nuevos números aleatorios dentro del Parent.
- En el component **HijoNumero** tendremos un botón **Sumar Número** en el que iremos Sumando el número pulsado.
- El dibujo de la suma lo veremos en el Parent



## PADRENUMEROS.VUE

```
<template>
  <div>
    <h1>Padre números</h1>
    <h3 style="color:red">
      La suma es {{ suma }}
    </h3>
    <button @click="generarNumero()">
      Nuevo número
    </button>
    <div v-for="num in numeros" :key="num">
      <HijoNumero :numero="num" v-on:sumarNumeros="sumarNumeros"/>
    </div>
  </div>
</template>
<script>
import HijoNumero from './HijoNumero.vue';
export default {
  name: "PadreNumeros",
  components: {
    HijoNumero
  },
  data() {
    return {
      numeros: [22, 33, 14],
      suma: 0
    }
  },
  methods: {
    sumarNumeros(numero) {
      console.log(numero);
      this.suma += numero;
    },
    generarNumero() {
      let random = parseInt(Math.random() * 500) + 1;
      this.numeros.push(random);
    }
  }
}
</script>
```

## HIJONUMERO.VUE

```
<template>
  <div>
    <h5 style="color:blue">Número hijo: {{ numero }}</h5>
    <button @click="callSumarNumeros()">
      Sumar {{ numero }}
    </button>
  </div>
</template>
<script>
export default {
  name: "HijoNumero",
  props: ["numero"],
  methods: {
    callSumarNumeros() {
      this.$emit("sumarNumeros", this.numero)
    }
  }
}
</script>
```

## DIRECTIVAS DE ESTILO

Una directiva puede tener un condicional con v-if  
También podemos aplicar directivas directamente sobre los estilos de un determinado CSS

### CSS

```
.rojo {
    color:red
}

.verde {
    color: green
}
```

Las directivas nos permiten, de forma dinámica, modificar estilos dependiendo de un valor

Podríamos preguntar, en el código template por una variable y aplicar la directiva con una clase  
En dicha variable.

Ejemplo:

```
<p class="{
    rojo: CONDICION TRUE
    verde: CONDICION false
}">
    Mi párrafo
</p>
```

## FORMULARIOS SUBMIT

Hasta ahora, hemos estado enviando información sin utilizar un Form, es funcional, pero  
No es buena praxis, ya que sin Form, dependiendo del servidor, no funcionará en producción.

Siempre que enviamos información al servidor, debemos hacerlo mediante etiquetas <form>

El problema que tenemos ahora mismo con este teoría es que nos sucede lo mismo que en React.  
Cuando enviamos la información con un botón, enviará la información mediante un submit  
Hacia el servidor. Debemos aplicar event.preventDefault()

La única diferencia está en la sintaxis, event no es recibido en los métodos, sino que se incluye en  
Las peticiones de cada elemento HTML.

La sintaxis debe realizarse mediante v-on:TIPOEVENTO

```
<form v-on:submit.prevent="enviarInformacion()">
    <button>
        Enviar información
    </button>
</form>
```

Vamos a realizar la práctica de Comics.

Comenzamos creando un nuevo componente llamado **ComicComponent.vue**

Creamos otro componente llamado **ComicComponent.vue**

Dentro de Comics, mostramos el componente hijo de Comic.

Sobre **assets**, creamos una nueva carpeta llamada **css** y dentro un fichero llamado **comics.css**

```
.rojo {
    background-color: red;
    color: white
}

.verde {
    background-color: lightgreen;
}

img {
    width: 150px;
    height: 250px;
}

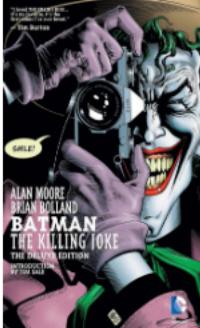
div#comics {
    float: left;
```



Sobre **ComicsComponent**, en la zona de **data()**, incluimos nuestros comics.

## Padre comics

Tituloo	Asterix	Imagen	<a href="https://www.brunolibros.es/in">https://www.brunolibros.es/in</a>	Descripción	y Obelix	Año	1996
---------	---------	--------	---	-------------	----------	-----	------

<b>Spiderman</b>	<b>Wolverine</b>	<b>Spawn</b>	<b>Batman</b>	<b>Ast</b>				
Seleccionar favorito	<b>Delete comic</b>	Seleccionar favorito	<b>Delete comic</b>	Seleccionar favorito	<b>Delete comic</b>	Seleccionar favorito	<b>Delete comic</b>	Seleccionar favorito
Hombre araña	Lobezno	Al Simmons	Murcielago	y Ot				
								
Year: 1997	Year: 2003	Year: 2000	Year: 2001	Year:				

### COMICS COMPONENT.VUE

```
<template>
<div>
  <h2>Padre comics</h2>
  <div>
    <form v-on:submit.prevent="createComic()">
      <label>Tituloo</label>
      <input type="text" v-model="comicForm.titulo"/>
      <label>Imagen</label>
      <input type="text" v-model="comicForm.imagen"/>
      <label>Descripción</label>
      <input type="text" v-model="comicForm.descripcion"/>
      <label>Año</label>
      <input type="number" v-model="comicForm.year"/>
      <button>
        Nuevo comic
      </button>
    </form>
  </div>
  <div v-if="comicFavorito" style="background-color: lightgray;">
    <h6>{{ comicFavorito.titulo }}</h6>
    
  </div>
  <div id="comics" v-for="(comic, index) in comics" :key="comic">
    <ComicComponent :comic="comic" :index="index"
      v-on:seleccionarFavorito="seleccionarFavorito"
      v-on:deleteComic="deleteComic"/>
  </div>
  </div>
</template>
<script>
import ComicComponent from './ComicComponent.vue';
export default {
  name: "ComicsComponent",
  components: {
    ComicComponent
  },
  methods: {
    seleccionarFavorito(comic) {
      console.log(comic.titulo);
      this.comicFavorito = comic;
    },
    createComic() {
      this.comics.push(this.comicForm);
    },
    deleteComic(index) {
      this.comics.splice(index, 1);
    }
  },
  data() {
    return {
      comicForm: {
        titulo: "",
        imagen: "",
        descripcion: "",
        year: 0
      }
    }
  }
}
</script>
```

```

        },
        comicFavorito: null,
        comics: [
            {
                titulo: "Spiderman",
                imagen:
                    "https://3.bp.blogspot.com/-i70Zu\_LAHwI/T290xxduu-I/AAAAAAAALq8/8bXDrdvW50o/s1600/spiderman1.jpg",
                    descripcion: "Hombre araña"
                    , year: 1997
            },
            {
                titulo: "Wolverine",
                imagen:
                    "https://images-na.ssl-images-amazon.com/images/I/51c101TdUBL.\_SX259\_BO1,204,203,200\_.jpg",
                    descripcion: "Lobezno"
                    , year: 2003
            },
            {
                titulo: "Guardianes de la Galaxia",
                imagen:
                    "https://comicstores.es/imagenes/9772938/977293843900900006.JPG",
                    descripcion: "Yo soy Groot"
                    , year: 2006
            },
            {
                titulo: "Avengers",
                imagen:
                    "https://d26lpennugtm8s.cloudfront.net/stores/057/977/products/ma\_avengers\_01\_01-891178138c020318f315132687055371-640-0.jpg",
                    descripcion: "Los Vengadores"
                    , year: 1993
            },
            {
                titulo: "Spawn",
                imagen:
                    "https://i.pinimg.com/originals/e1/d8/ff/e1d8ff4aeab5e567798635008fe98ee1.png",
                    descripcion: "Al Simmons"
                    , year: 2000
            },
            {
                titulo: "Batman",
                imagen:
                    "https://www.comicverso.com/wp-content/uploads/2020/06/The-Killing-Joke-657x1024.jpg",
                    descripcion: "Murcielago"
                    , year: 2001
            }
        ]
    }
}
</script>

```

## COMICCOMPONENT.VUE

```

<template>
<div>
    <h2 style="color:blue">{{ comic.titulo }}</h2>
    <button @click="seleccionarFavorito()">
        Seleccionar favorito
    </button>
    <button @click="deleteComic()" class="rojo">
        Delete comic
    </button>
    <p>{{ comic.descripcion }}</p>
    
    <h4 :class="{>
        rojo: comic.year <= 2000,>
        verde: comic.year > 2000
    }">
        Year: {{ comic.year }}</h4>
</div>
</template>
<script>
    export default {
        name: "ComicComponent",
        props: ["comic", "index"],
        methods: {
            seleccionarFavorito() {
                console.log(this.comic);
                this.$emit("seleccionarFavorito", this.comic)
            },
            deleteComic(){
                this.$emit("deleteComic", this.index);
            }
        }
    }
</script>

```

```
<style>
  @import './assets/css/comics.css'
</style>
```

## PRACTICA

Necesito un componente que nos muestre los elementos seleccionados de un control <select>, tal y como hicimos con React.  
Los elementos los dibujamos con un Array  
Al pulsar el botón, mostraremos los seleccionados.

1. Hacerlo solo con VUE
2. Instalar librerías de VUE para selección múltiple

Creamos un nuevo componente llamado **SeleccionMultiple.vue**

Tendremos un botón para generar N checkbox con un número aleatorio cada uno  
Al seleccionar un Checkbox, sumamos y al deseleccionar, restamos.

Creamos un componente llamado **SumaCheckbox.vue**

**Nota:** Una vez terminado, necesito tener **Router** dentro de nuestro proyecto

```
npm install vue-router@next --save
```

Creamos una navegación con **Router.js** dentro del proyecto y **src**

```
import PadreDeportes from "./components/PadreDeportes.vue";
import ComicsComponent from "./components/ComicsComponent.vue";
import PadreNumeros from "./components/PadreNumeros.vue";

const myRoutes = [
  { path: "/", component: PadreDeportes },
  { path: "/comics", component: ComicsComponent },
  { path: "/numeros", component: PadreNumeros }
]

const router = createRouter({
  history: createWebHistory(),
  routes: myRoutes
})
```

```
export default router;
```

Habilitamos la navegación en **main.js**

#### MAIN.JS

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './Router'

createApp(App).use(router).mount('#app')
```

Incluimos un componente llamado **MenuComponent.vue** con la navegación de **<router-link>** y Ponemos, en **App**, la etiqueta **<router-view>** y el componente **MenuComponent**

#### RUTAS CON PARAMETROS

Las rutas con parámetros con iguales, en la declaración, a como lo hicimos en React, es decir, Mediante los dos puntos separamos los diferentes parámetros y los declaramos

Se declaran en **path** de la ruta:

```
{ path: "/comics/:id", component: ComicsComponent}
```

Dentro de una misma ruta, también tenemos la posibilidad de declarar parámetros opcionales. Estos siempre irán en el último lugar

```
{ path: "/comics/:id?", component: ComicsComponent}.
```

Para enviar los parámetros en la ruta, simplemente utilizamos las etiquetas **<router-link>**

```
<router-link to="/comics/14">Comics</router-link> |
```

La recuperación de los parámetros dentro del componente (ejemplo comics) se realiza con una Sintaxis "parecida" a **\$emit**

La recepción de los parámetros debemos hacerla siempre dentro del método **mounted()**

Para recuperarlos tenemos el objeto **\$route**

**this.\$route.params.NOMBREPARAMETRO**

Para comprobar esta funcionalidad vamos a crear un componente llamado **Numerodoble.vue**

Dicho componente recibirá un parámetro de un número (lo haremos opcional) y mostramos su doble.

```
const myRoutes = [
  { path: "/", component: PadreDeportes},
  { path: "/numerodoble/:numero?", component: NumeroDoble},
  { path: "/comics", component: ComicsComponent},
  { path: "/numeros", component: PadreNumeros},
]
```

En el **MenuComponent** incluimos dos **<router-link>**, uno sin parámetro y otro con parámetro

```
<li>
  <router-link to="/numerodoble">Doble SIN</router-link>
</li>
<li>
  <router-link to="/numerodoble/77">Doble 77</router-link>
</li>
```

El siguiente paso es recibir los parámetros dentro de **mounted()**

Necesitamos **data()** para el dibujo y **mounted()** para capturar los parámetros

[Deportes](#) | [Comics](#) | [Números](#) | [Doble SIN](#) | [Doble 77](#)

## Número doble

Parámetro recibido: 77

NUMERODOBLE.VUE

```
<template>
  <div>
    <h1>Número doble</h1>
    <h2 style="color:red">{{ mensaje }}</h2>
  </div>
</template>
<script>
  export default {
    name: "NumeroDoble",
    data() {
      return {
        mensaje: ""
      }
    },
    mounted() {
      console.log("Param: " + this.$route.params.numero);
      //LOS PARAMETROS SIEMPRE SON STRING, NO IMPORTA SI SON NUMERICOS
      let numero = this.$route.params.numero;
      if (numero == ""){
        this.mensaje = "Sin parámetros en Routing";
      }else{
        this.mensaje = "Parámetro recibido: " + numero;
      }
    }
  }
</script>
```

Una vez que hemos visto que el componente no se "redibuja", sucede lo mismo que pasaba en React, si el componente ya está dibujado, no vuelve a realizar un render().

Debemos hacer lo mismo que React, utilizar **componentDidUpdate()**, pero con sintaxis Vue

Tenemos una nueva "zona" llamada **watch** que es la encargada de supervisar los cambios De parámetros.

```
watch: {
  someParamsRouting(nextVal, oldVal) {
    nextVal: El valor nuevo que ha tomado nuestro parámetro
    oldVal: El valor anterior del parámetro
  }
}
```

Por último, para indicar lo que vamos a supervisar, NO se utiliza la palabra **this**

También debemos indicar el nombre del parámetro a supervisar entre comillas.

```
watch : {
  '$route.params.IDPARAMETRO' (nextVal, oldVal) {
    }
}
```

Aplicamos la nueva funcionalidad dentro de nuestro componente **NumeroDoble.vue**

```
}, watch: {
  '$route.params.numero' (nextVal, oldVal){
    this.mensaje = "Esto ha cambiado: "
    + this.$route.params.numero;
    this.doble = parseInt(this.$route.params.numero);
    console.log("Next: " + nextVal);
    console.log("Old: " + oldVal);
  }
}
```

**Importante:** SIEMPRE debemos comparar los valores de nuestro **watch** para no tener bucles infinitos.

```
}, watch: [
    '$route.params.numero' (nextVal, oldVal){
        if (nextVal != oldVal){
            this.mensaje = "Esto ha cambiado: "
            + this.$route.params.numero;
            this.doble = parseInt(this.$route.params.numero) * 2;
            console.log("Next: " + nextVal);
            console.log("Old: " + oldVal);
        }
    }
]
```

El siguiente paso es visualizar cómo podemos **Navegar** por código, es decir, realizar la etiqueta `<Navigate/>` de React, pero en VUE.

Vamos a incluir un botón y un método para navegar a **Home**

La sintaxis es la siguiente:

`this.$router.push(PATH)`

```
redirectToHome() {
    this.$router.push("/");
}
```

#### NUMERODOBLE.VUE

```
<template>
    <div>
        <h1>Número doble: {{ doble }}</h1>
        <h2 style="color:red">{{ mensaje }}</h2>
        <button @click="redirectToHome()">
            Go to Home
        </button>
    </div>
</template>
<script>
    export default {
        name: "NumeroDoble",
        methods: {
            redirectToHome() {
                this.$router.push("/");
            }
        },
        data() {
            return {
                mensaje: "",
                doble: 0
            }
        },
        mounted() {
            console.log("Param: " + this.$route.params.numero);
            //LOS PARAMETROS SIEMPRE SON STRING, NO IMPORTA SI SON NUMERICOS
            let numero = this.$route.params.numero;
            if (numero == ""){
                this.mensaje = "Sin parámetros en Routing";
            }else{
                this.mensaje = "Parámetro recibido: " + numero;
                this.doble = parseInt(numero) * 2;
            }
        },
        watch: {
            '$route.params.numero' (nextVal, oldVal){
                if (nextVal != oldVal){
                    this.mensaje = "Esto ha cambiado: "
                    + this.$route.params.numero;
                    this.doble = parseInt(this.$route.params.numero) * 2;
                    console.log("Next: " + nextVal);
                    console.log("Old: " + oldVal);
                }
            }
        }
    }

```

```

    }
</script>

```

El siguiente paso es generar dinámicamente Links con parámetros.

Vamos a tener un Array con números dentro de **MenuComponent**.

Dibujaremos los **<router-link>** mediante un **v-for** y enviando el parámetro con dichos números Del Array.

```

<li v-for>
  <router-link to="/numerodoble/77">Doble 77</router-link>
</li>

```

[Deportes](#) | [Comics](#) | [Números](#) | [Doble SIN](#) | [Doble 77](#) | [Doble 33](#) | [Doble 22](#) |



## Número doble: 44

**Esto ha cambiado: 22**

[Go to Home](#)

### MENUCOMPONENT.VUE

```

<template>
  <div>
    <ul id="menu">
      <li>
        <router-link to="/">Deportes</router-link> |
      </li>
      <li>
        <router-link to="/comics">Comics</router-link> |
      </li>
      <li>
        <router-link to="/numeros">Números</router-link> |
      </li>
      <li>
        <router-link to="/numerodoble">Doble SIN</router-link> |
      </li>
      <li v-for="num in numeros" :key="num">
        <router-link :to="'/numerodoble/' + num">Doble {{ num }}</router-link> |
      </li>
    </ul>
  </div>
</template>
<script>
  export default {
    name: "MenuComponent",
    data() {
      return {
        numeros: [77,33,22]
      }
    }
  }
</script>
<style>
  ul#menu li {
    display: inline;
  }
</style>

```

### TABLA MULTIPLICAR CON ROUTING

- Tendremos un menú para dibujar números aleatorios que serán parte de los Links Para la tabla de multiplicar, es decir, un component llamado **MenuTablaMultiplicar**
- Al seleccionar un número en un Link, dibujaremos una tabla de multiplicar capturando El parámetro del Routing, es decir, otro component llamado **TablaMultiplicar**

[Tabla 70](#) | [Tabla 1](#) | [Tabla 5](#) | [Tabla 20](#) | [Tabla 10](#) |

## Tabla multiplicar

Operación	Resultado
5*1	5
5*2	10
5*3	15
5*4	20
5*5	25
5*6	30

ROUTER.JS

```
{ path: "/tabla/:valor", component: TablaMultiplicar}
```

MENUTABLAMULTIPLICAR.VUE

```
<template>
  <div>
    <ul id="menu">
      <li v-for="num in numeros" :key="num">
        <router-link :to="/tabla/" + num>
          Tabla {{ num }} |
        </router-link>
      </li>
    </ul>
  </div>
</template>
<script>
  export default {
    name: "MenuTablaMultiplicar",
    data() {
      return{
        numeros: []
      }
    },
    mounted() {
      for (var i = 1; i <= 5; i++){
        let random = parseInt(Math.random() * 100) + 1;
        this.numeros.push(random);
      }
    }
  }
</script>
<style>
  ul#menu li {
    display: inline;
  }
</style>
```

TABLAMULTIPLICAR.VUE

```
<template>
  <div>
    <h1>Tabla multiplicar</h1>
    <table border="1">
      <thead>
        <tr>
          <th>Operación</th>
          <th>Resultado</th>
        </tr>
      </thead>
      <tbody v-html="html">
      </tbody>
    </table>
  </div>
</template>
<script>
  export default {
    name: "TablaMultiplicar",
    data() {
      return {
        html: "",
        numero: 0
      }
    },
    watch: {
      '$route.params.valor' (nextVal, oldVal){
        if (nextVal != oldVal){
          this.generarTabla();
        }
      }
    },
    methods: {
```

```

generarTabla() {
    let aux = "";
    this.numero = parseInt(this.$route.params.valor);
    for (var i = 1; i <= 10; i++){
        aux += "<tr>";
        aux += "<td>" + this.numero + "*" + i + "</td>";
        aux += "<td>" + (this.numero * i) + "</td>";
        aux += "</tr>";
    }
    this.html = aux;
},
mounted() {
    this.generarTabla();
}
}
</script>

```

## SERVICIOS VUE

Vue no tiene una forma nativa de leer servicios (librería)  
Necesitamos instalar una librería para trabajar.

Podemos utilizar las mismas que React:

1. axios
2. fetch

Para los servicios tenemos algo conocido:

```
axios.get(url).then(response => {
})
```

Por supuesto, tendremos el concepto de **Global.js** para incluir nuestras URL de forma general en El proyecto.

Creamos un proyecto nuevo llamado **vueservicios** e instalamos **axios**

```
npm install axios --save
```

Vamos a realizar un componente para dibujar un Api con Coches para probar que todo es igual en VUE con **Axios**

<https://apicochespaco.azurewebsites.net/>

Comenzamos creando un nuevo componente llamado **CochesComponent.vue**

# Api Coches

## PONTIAC FIREBIRD

Driver: MICHAEL KNIGHT



## VOLKSWAGEN ESCARABAJO

Driver: HERBIE

## COCHESCOMPONENT.VUE

```
<template>
<div>
<h1>Api Coches</h1>
<div v-for="car in coches" :key="car">
```

```

<h3 style="color:blue">
    {{ car.marca }} {{ car.modelo }}
</h3>
<p>Driver: {{ car.conductor }}</p>

</div>
</div>
</template>
<script>
import axios from 'axios';
//SI NECESITAMOS VARIABLES PARA TODO EL COMPONENT Y SUS METODOS
//SE DECLARAN AQUI (mounted, methods, create)
let urlApi = "https://apicochespaco.azurewebsites.net/";
export default {
    name: "CochesComponent",
    data() {
        return{
            coches: []
        }
    },
    mounted() {
        let request = "webresources/coches";
        //LAS VARIABLES DECLARADAS FUERA DEL EXPORT NO UTILIZAN this
        let url = urlApi + request;
        axios.get(url).then(response => {
            console.log("Leyendo servicio");
            this.coches = response.data;
        })
    }
}
</script>

```

El siguiente concepto es utilizar **Global.js**

```

var Global = {
    Key: value
}

export default Global;

```

Sobre **src** creamos un nuevo fichero llamado **Global.js**

```

var Global = {
    urlApiCoches: "https://apicochespaco.azurewebsites.net/"
}

export default Global;

```

Utilizamos Global sobre nuestro componente **CochesComponent**

```

import axios from 'axios';
import Global from './.../Global';

//SI NECESITAMOS VARIABLES PARA TODO EL COMPONENT
//SE DECLARAN AQUI (mounted, methods, create)
let urlApi = Global.urlApiCoches;
export default {
    name: "CochesComponent",
    data() {

```

A continuación, quiero una práctica con Customers.

<https://services.odata.org/V4/Northwind/Northwind.svc>

Necesitamos, al cargar la página, cargamos los Customers (ID, Nombre y Compañía) dentro de una Tabla.

Tendremos un botón y una caja para buscar un Customer por su ID y mostramos algunos datos.

## Customers

Id Customer

Buscar cliente

# Sales Representative

UK

120 Hanover Sq.

Id	Nombre	Empresa
ALFKI	Maria Anders	Alfreds Futterkiste
ANATR	Ana Trujillo	Ana Trujillo Emparedados y helados
ANTON	Antonio Moreno	Antonio Moreno Taquería
AROUT	Thomas Hardy	Around the Horn
BERGS	Christina Berglund	Berglunds snabbköp

CUSTOMERSCOMPONENT

```
<template>
<div>
    <h1>Customers</h1>
    <form>
        <label>Id Customer</label>
        <input type="text" v-model="idCustomer"/>
        <button v-on:click.prevent="buscarCustomer()">
            Buscar cliente
        </button>
    </form>
    <div v-if="customer">
        <h3>{{customer.ContactTitle}}</h3>
        <h2>{{ customer.Country }}</h2>
        <h2>{{ customer.Address }}</h2>
    </div>
    <table border="1">
        <thead>
            <tr>
                <th>Id</th>
                <th>Nombre</th>
                <th>Empresa</th>
            </tr>
        </thead>
        <tbody>
            <tr v-for="customer in clientes" :key="customer">
                <td>{{ customer.CustomerID }}</td>
                <td>{{ customer.ContactName }}</td>
                <td>{{ customer.CompanyName }}</td>
            </tr>
        </tbody>
    </table>
</div>
</template>
<script>
import axios from 'axios';
import Global from '../Global';
let urlApi = Global.urlApiCustomers;
export default {
    name: "CustomersComponent",
    data() {
        return {
            clientes: [],
            idCustomer: "",
            customer: null
        }
    },
    mounted() {
        let request = "customers";
        let url = urlApi + request;
        axios.get(url).then(response => {
            console.log("Leyendo customers");
            this.clientes = response.data.value;
        })
    },
    methods: {
        buscarCustomer() {
            //BUCLE RECORRIENDO TODOS LOS CUSTOMERS
            //COMPARAMOS EL idCustomer DE DATA CON
            //CustomerID DEL OBJETO
            for (var cliente of this.clientes){
                if (cliente.CustomerID.toLowerCase() ==
this.idCustomer.toLowerCase()){
                    this.customer = cliente;
                    break;
                }
            }
        }
    }
}
</script>
```

```

        }
    }
}
</script>

```

## UTILIZAR BOOTSTRAP CON VUE

Simplemente tenemos dos formas de realizarlo:

1. Local: Copiamos los ficheros dentro de **src** y **assets** y **style**. Posteriormente, en un componente Principal, pondremos los **@import**

```

@import '~bootstrap/scss/bootstrap.scss';
@import '~bootstrap-vue/src/index.scss';

```

2. Nube: Incluir los Links oficiales de Bootstrap de la nube. En la página **index.html**  
Podremos los Links y ya tendremos aplicado Bootstrap en nuestro proyecto.

Entramos dentro de la página de Bootstrap y copiamos los Links oficiales de CDN.

The screenshot shows a web page titled "Include via CDN". It contains two code snippets for including Bootstrap's compiled CSS and JS files via CDN. The first snippet is for CSS:

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/css/bootstrap.min.css" rel="stylesheet" type="text/css"/>
```

The second snippet is for JS:

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/js/bootstrap.bundle.min.js" integrity="sha384-KF9gAAoDQUzGJZIzW0LwvK40MqB1WZP0XbWQOo9lWvCZD" crossorigin="anonymous">
```

## INDEX.HTML

```

<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>/favicon.ico">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/css/bootstrap.min.css" rel="stylesheet" type="text/css"/>
    <title><%= htmlWebpackPlugin.options.title %></title>
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScript enabled</strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/js/bootstrap.bundle.min.js" integrity="sha384-KF9gAAoDQUzGJZIzW0LwvK40MqB1WZP0XbWQOo9lWvCZD" crossorigin="anonymous"></script>
  </body>
</html>

```

Continuamos trabajando con Apis.

El siguiente ejemplo lo haremos sobre un servicio de empleados

<https://apiempleadosfullstack.azurewebsites.net/>

- Al iniciar la página, cargamos los empleados dentro de un <select>
- Al seleccionar un determinado empleado, mostramos sus detalles



Creamos un nuevo componente llamado `EmpleadoDetails.vue`

```
var Global = {
    urlApiCoches: "https://apicochespaco.azurewebsites.net/",
    urlApiCustomers: "https://services.odata.org/V4/Northwind/Northwind.svc/",
    urlApiEmpleados: "https://apiempleadosfullstack.azurewebsites.net/"
}

export default Global;
```

EMPLEADOSDETAILS.VUE

```
<template>
<div style="width: 50%; margin: auto">
  <h1>Empleados details</h1>
  <form>
    <label>Seleccione un empleado</label>
    <select class="form-control" v-model="idEmpleado">
      <option v-for="emp in empleados" :key="emp.idEmpleado" :value="emp.idEmpleado">
        {{ emp.apellido }}
      </option>
    </select>
    <button v-on:click.prevent="findEmpleado()">
      Detalles
    </button>
  </form>
  <ul class="list-group" v-if="empleado">
    <li class="list-group-item">
      <b>{{ empleado.apellido }}</b>
    </li>
    <li class="list-group-item">
      Oficio {{ empleado.oficio }}
    </li>
    <li class="list-group-item">
      Salario: {{ empleado.salario }}
    </li>
    <li class="list-group-item">
      Departamento: {{ empleado.departamento }}
    </li>
  </ul>
</div>
```

```

        </div>
    </template>
<script>
import Global from './Global';
import axios from 'axios';
let urlApi = Global.urlApiEmpleados;
export default {
    name: "EmpleadosDetails",
    methods: {
        findEmpleado(){
            let request = "api/empleados/" + this.idEmpleado;
            let url = urlApi + request;
            axios.get(url).then(response => {
                console.log("Buscando empleado")
                this.empleado = response.data;
            })
        }
    },
    data() {
        return {
            empleados: [],
            idEmpleado: 0,
            empleado: null
        }
    },
    mounted() {
        let request = "api/empleados";
        let url = urlApi + request;
        axios.get(url).then(response => {
            console.log("Leyendo empleados");
            this.empleados = response.data;
        })
    }
}
</script>

```

Instalamos Routing y creamos **Router.js** dentro de **src** con nuestros components.

```
λ npm install vue-router@next --save
```

#### ROUTER.JS

```

import { createWebHistory, createRouter } from "vue-router";
import HomeComponent from "./components/HomeComponent.vue";
import CochesComponent from "./components/CochesComponent.vue";
import EmpleadosDetails from "./components/EmpleadosDetails.vue";
import CustomersComponent from "./components/CustomersComponent.vue";
const myRoutes = [
    { path: "/", component: HomeComponent},
    { path: "/coches", component: CochesComponent },
    { path: "/empleadosdetails", component: EmpleadosDetails},
    { path: "/customers", component: CustomersComponent}
]
const router = createRouter({
    history: createWebHistory(),
    routes: myRoutes
})
export default router;

```

#### MAIN.JS

```

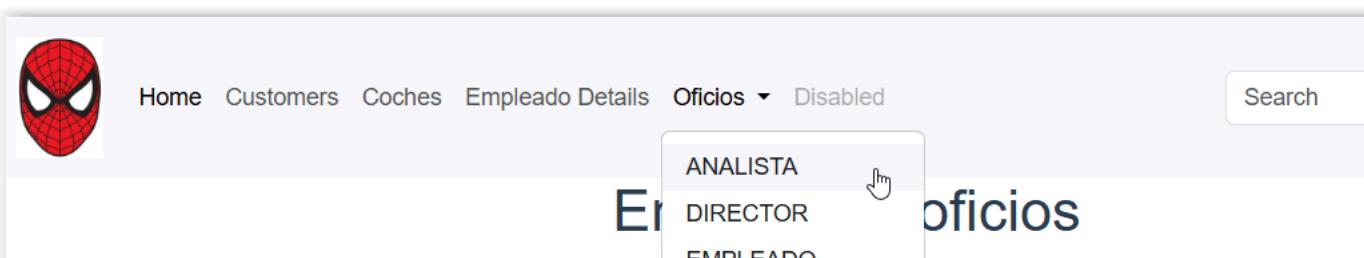
import { createApp } from 'vue'
import App from './App.vue'
import router from './Router'

createApp(App).use(router).mount('#app')

```

Con mucho cuidado, creamos un component llamado **MenuComponent** y buscamos un navbar De Bootstrap.

Vamos a mostrar, en nuestro menú, todos los oficios en un desplegable Dropdown. Posteriormente, al seleccionar un oficio, mediante Routing, cargaremos los empleados de dicho oficio.



Apellido		Salario
GUTIERREZ	PRESIDENTE	219000
GIL	VENDEDOR	390000
FERNANDEZ	Another action	390000
SANTIUSTE	Something else here	390000
	ANALISTA	225000

## MENUCOMPONENT.VUE

```

<template>
  <nav class="navbar navbar-expand-lg bg-body-tertiary">
    <div class="container-fluid">
      <a class="navbar-brand" href="#">Navbar</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
        data-bs-target="#navbarSupportedContent" aria-
        controls="navbarSupportedContent" aria-expanded="false"
        aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
          <li class="nav-item">
            <router-link class="nav-link active" aria-current="page"
to="/">Home</router-link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" to="/customers">Customers</router-
link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" to="/coches">Coches</router-link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" to="/empleadosdetails">Empleado
Details</router-link>
          </li>
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" href="#" role="button" data-
            bs-toggle="dropdown"
              aria-expanded="false">
              Oficios
            </a>
            <ul class="dropdown-menu">
              <li v-for="oficio in oficios" :key="oficio">
                <router-link
                  class="dropdown-item"
                  :to="'/empleadosoficios/' + oficio">
                  {{ oficio }}
                </router-link>
              </li>
              <li><a class="dropdown-item" href="#">Another action</a>
              </li>
              <li><hr class="dropdown-divider">
              </li>
              <li><a class="dropdown-item" href="#">Something else here</a>
              </li>
            </ul>
          </li>
          <li class="nav-item">
            <a class="nav-link disabled" aria-disabled="true">Disabled</a>
          </li>
        </ul>
      </div>
      <form class="d-flex" role="search">
        <input class="form-control me-2" type="search" placeholder="Search"
aria-label="Search" />
        <button class="btn btn-outline-success" type="submit">Search</button>
      </form>
    </div>
  </nav>
</template>
<script>
import axios from 'axios';
import Global from './Global'
export default {
  name: "MenuComponent",
  data() {
    return {
      oficios: []
    }
  },
  mounted() {
    let request = "api/empleados/oficios";
    axios.get(request)
      .then(response => {
        this.oficios = response.data;
      })
      .catch(error => {
        console.log(error);
      });
  }
}
</script>

```

```

        let url = Global.urlApiEmpleados + request;
        axios.get(url).then(response => {
            console.log("Leyendo oficios")
            this.oficios = response.data
        })
    }
</script>

```

El siguiente paso es crear un componente llamado **EmpleadosOficios.vue** y lo ponemos en la ruta de **Router.js**

```

const myRoutes = [
    { path: "/", component: HomeComponent },
    { path: "/coches", component: CochesComponent },
    { path: "/empleadosdetails", component: EmpleadosDetails },
    { path: "/customers", component: CustomersComponent },
    { path: "/empleadosoficios/:oficio", component: EmpleadosOficios }
]

```

#### EMPLEADOSOFICIOS.VUE

```

<template>
    <div>
        <h1>Empleados oficios</h1>
        <table class="table table-warning">
            <thead>
                <tr>
                    <th>Apellido</th>
                    <th>Oficio</th>
                    <th>Salario</th>
                    <th>Departamento</th>
                </tr>
            </thead>
            <tbody>
                <tr v-for="emp in empleados" :key="emp">
                    <td>{{ emp.apellido }}</td>
                    <td>{{ emp.oficio }}</td>
                    <td>{{ emp.salario }}</td>
                    <td>{{ emp.departamento }}</td>
                </tr>
            </tbody>
        </table>
    </div>
</template>
<script>
import axios from 'axios';
import Global from '../Global'
export default {
    name: "EmpleadosOficios",
    data() {
        return {
            empleados: []
        }
    },
    methods: {
        loadEmpleados() {
            let oficio = this.$route.params.oficio;
            let request = "api/empleados/empleadosoficio/" + oficio;
            let url = Global.urlApiEmpleados + request;
            axios.get(url).then(response => {
                console.log("Leyendo empleados oficio");
                this.empleados = response.data;
            })
        }
    },
    mounted() {
        this.loadEmpleados();
    },
    watch: {
        '$route.params.oficio' (nextVal, oldVal) {
            if (nextVal != oldVal){
                this.loadEmpleados();
            }
        }
    }
}
</script>

```

#### SERVICIOS VUE

Un Servicio es una clase que es capaz de administrar peticiones.

Nos permite tener, en un mismo lugar, distintas peticiones para poder reutilizar las llamadas.

Actualmente, tenemos las peticiones separadas por components, si deseamos modificar alguna petición, Debemos "buscar" el componente que tiene dicha petición.

Un Servicio es una clase con métodos **return** que nos devolverán la información una vez que hayan finalizado

La petición.

#### EJEMPLO DE CLASE SERVICE

```
export default class ServiceEjemplo {
    miMetodo1() {
        //ACCIONES
        return data;
    }
    miMetodo2() {
        //ACCIONES
        return data;
    }
}
```

Los servicios deben crearse dentro de la carpeta **src** y una propia carpeta llamada **services** y con extensión **JS**

Creamos una nueva clase llamada **ServiceEjemplo.js**

```
export default class ServiceEjemplo {
    getSaludo(nombre){
        return "Bienvenido a tu viernes, " + nombre
    }
}
```

Para probar la petición al servicio, lo hacemos sobre **HomeComponent**

#### HOMECOMPONENT

```
<template>
<div>
    <h1>Servicios Api VUE</h1>
    <h2 style="color:blue">{{ mensaje }}</h2>
</div>
</template>
<script>
import ServiceEjemplo from './services/ServiceEjemplo';
//PARA PODER UTILIZAR EL SERVICIO NECESITAMOS UNA INSTANCIA (VARIABLE/OBJETO)
//DE LA CLASE SERVICIO.
const service = new ServiceEjemplo();
export default {
    name: "HomeComponent",
    data() {
        return {
            mensaje: ""
        }
    },
    mounted() {
        this.mensaje = service.getSaludo("Alumno");
    }
}
</script>
```

El siguiente paso es migrar nuestro component **CochesComponent** a servicios.

Creamos un nuevo servicio llamado **ServiceCoches.js**

#### SERVICECOCHES.JS

```
import axios from "axios";
import Global from './Global'
export default class ServiceCoches {
    getCoches() {
        let coches = [];
        let request = "webresources/coches";
        let url = Global.urlApiCoches + request;
        axios.get(url).then(response => {
            console.log("Coches servicio");
            coches = response.data;
        })
        return coches;
    }
}
```

Modificamos el código de **CochesComponent**

#### COCHESCOMPONENT

```
<script>
import ServiceCoches from './services/ServiceCoches';
const service = new ServiceCoches();
export default {
```

```

        name: "CochesComponent",
        data() {
            return{
                coches: []
            }
        }, mounted() {
            this.coches = service.getCoches();
        }
    }
</script>

```

Como podemos comprobar, NO devuelve los Coches/data del servicio.

El problema está en el **CUANDO**, no importa si lo hacemos dentro de axios.get o lo que se nos ocurra, la Cuestión es que tenemos que esperar hasta que el servicio nos devuelva los datos.

Cuando hablamos de Servicios api, nosotros realizamos **promesas** a la espera de recibir los datos.

Una promesa es un **then** de **axios**

Debemos crear los métodos con apis asíncronos con **promesas** de forma explícita.

Una promesa puede recibir dos parámetros:

1. **Reject**: El código devuelve un error si ha ido mal la petición en la promesa
2. **Resolve**: todo ha ido estupendo y nos devuelve los datos

#### EJEMPLO DE PROMESA

```

getPromesa = new Promise(function(resolve, reject) {
    //TENEMOS DOS POSIBILIDADES, ERROR O DATOS
    let num = 0;
    if (num == 0){
        //DEVOLVEMOS CORRECTO
        resolve("Todo Ok, Jose Luis")
    }else{
        //DEVOLVEMOS ERROR
        reject("Error de alguna acción")
    }
})

```

También podemos hacerlo sin **reject**

```

getPromesa = new Promise(function(resolve) {
    resolve("Todo Ok, Jose Luis")
})

```

La petición será de una forma conocida:

```

service.getPromesa().then(response => {
    //DENTRO DE RESPONSE, TENEMOS LOS DATOS DE resolve
    console.log(response);
})

```

Por último, debemos devolver objetos dentro de la promesa entre clases y components, por lo que Necesitamos instalar un paquete para poder enviar dichos objetos.

**npm install core-js --save**

Modificamos el código de **ServiceCoches** con promesas.

#### SERVICECOCHES

```
import axios from "axios";
import Global from './Global';
export default class ServiceCoches {
    getCoches = new Promise(function(resolve) {
        let coches = [];
        let request = "webresources/coches";
        let url = Global.urlApiCoches + request;
        axios.get(url).then(response => {
            console.log("Coches servicio");
            coches = response.data;
            resolve(coches);
        })
    })
}
```

El siguiente paso es realizar la petición correcta dentro de **CochesComponent**

#### COCHESCOMPONENT

```
, mounted() {
    //UNA PROMESA NO ES UN METODO, ES UN OBJETO
    service.getCoches.then(result => {
        this.coches = result;
    })
}
```

Si una promesa no es un método, cómo le pasamos parámetros????

Si queremos pasar parámetros, necesitamos modificar la sintaxis del método del Service para que sea un método devolviendo una promesa en lugar de un objeto.

```
getPromesaParametros(param1) {
    //DENTRO DEL METODO, DEVOLVEMOS LA PROMESA
    return new Promise(function(resolve) {
        resolve("Todo ok, " + param1)
    })
}
```

Posteriormente, llamamos como un método:

```
service.getPromesaParametros("Jose Luis").then(result => {
    console.log(result);
})
```

#### SERVICECOCHES

```
import axios from "axios";
import Global from './Global';
export default class ServiceCoches {
    getCoches() {
        return new Promise(function(resolve) {
            let coches = [];
            let request = "webresources/coches";
            let url = Global.urlApiCoches + request;
            axios.get(url).then(response => {
```

```
        console.log("Coches servicio");
        coches = response.data;
        resolve(coches);
    })
}
}
```

## **COCHESCOMPONENT**

```
    }, mounted() {
        service.getCoches().then(result => {
            this.coches = result;
        })
    }
}
```

Por sintaxis, vamos a llevar toda la lógica de Empleados a un Servicio.

Creamos un nuevo servicio **ServiceEmpleados.js** y modificamos nuestro proyecto para que todo siga funcional.

## Métodos:

- `getEmpleados()`
  - `findEmpleado(id)`
  - `getOficios()`
  - `getEmpleadosOficio(oficio)`

SERVICEFIMPI FADOS.IS

```
import axios from "axios"
import Global from './Global';
export default class ServiceEmpleados {
    getEmpleados() {
        return new Promise(function(resolve) {
            let request = "api/empleados";
            let url = Global.urlApiEmpleados + request;
            axios.get(url).then(response => {
                resolve(response.data);
            })
        })
    }
    findEmpleado(idEmpleado) {
        return new Promise(function(resolve) {
            let request = "api/empleados/" + idEmpleado;
            let url = Global.urlApiEmpleados + request;
            axios.get(url).then(response => {
                resolve(response.data);
            })
        })
    }
    getEmpleadosOficio(oficio) {
        return new Promise(function(resolve) {
            let request = "api/empleados/empleadosoficio/" + oficio;
            let url = Global.urlApiEmpleados + request;
            axios.get(url).then(response => {
                resolve(response.data);
            })
        })
    }
    getOficios() {
        return new Promise(function(resolve) {
            let request = "api/empleados/oficios";
            let url = Global.urlApiEmpleados + request;
            axios.get(url).then(response => {
                resolve(response.data);
            })
        })
    }
}
```

El siguiente paso es un clásico, hacemos un CBUJD sobre Departamentos

Utilizamos esta URI:

<https://apicruddepartamentoscore.azurewebsites.net/index.html>

Creamos un nuevo proyecto llamado **vuecruddepartamentos**

Instalamos rutas y axios

`npm install axios --save`

`npm install vue-router@next --save`

Incluimos bootstrap dentro de `public/index.html`

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title><%= htmlWebpackPlugin.options.title %></title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/css/bootstrap.min.css" rel="stylesheet">
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work pr
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/js/bootstrap.bundle.js" integrity="sha384-KnHtCJWt9Jn4ZtTs4uXwEeBhuiqRzIbdyV1GqYlQbPZLqyfD6Mzq0Z8+Cg=="></script>
  </body>
</html>
```

Creamos un nuevo componente llamado **DepartamentosComponent.vue**

Sobre src, creamos **Router.js** e incluimos la navegación para **DepartamentosComponent** en root

```
import { createWebHistory, createRouter } from "vue-router";
import DepartamentosComponent from "./components/DepartamentosComponent.vue";
const myRoutes = [
  {path: "/", component: DepartamentosComponent}
]

const router = createRouter({
  history: createWebHistory(),
  routes: myRoutes
})

export default router;
```

Habilitamos **router** dentro de **main.js**

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './Router'

createApp(App).use(router).mount('#app')
```

Y creamos un **MenuComponent** con un **navbar** y lo ponemos a jugar en **App.vue**

Sobre **src** creamos un nuevo fichero llamado **Global.js**

#### GLOBALJS

```
var Global = {
    urlApiDepartamentos: "https://apicruddepartamentoscore.azurewebsites.net/"
}

export default Global;
```

Sobre **src** creamos una carpeta llamada **services** y una clase llamada **ServiceDepartamentos.js**

#### SERVICEDEPARTAMENTOS.JS

```
import axios from "axios";
import Global from "@/Global";
export default class ServiceDepartamentos {
    getDepartamentos() {
        return new Promise(function(resolve) {
            let request = "api/departamentos";
            let url = Global.urlApiDepartamentos + request;
            axios.get(url).then(response => {
                console.log("Leyendo departamentos");
                resolve(response.data);
            })
        })
    }
}
```

El siguiente paso es implementar la petición en **DepartamentosComponent**

Sobre **src/assets** creamos una carpeta llamada **images** y copiamos una imagen para el efecto Óptico de Loading

#### DEPARTAMENTOSCOMPONENT

```
<template>
<div>
    <h1>Departamentos</h1>
    
    <table v-else class="table table-bordered">
        <thead>
            <tr>
                <th>Id</th>
                <th>Nombre</th>
                <th>Localidad</th>
            </tr>
        </thead>
        <tbody>
            <tr v-for="dept in departamentos" :key="dept">
                <td>{{ dept.numero }}</td>
                <td>{{ dept.nombre }}</td>
                <td>{{ dept.localidad }}</td>
            </tr>
        </tbody>
    </table>
</div>
</template>
<script>
import ServiceDepartamentos from '@/services/ServiceDepartamentos';
const service = new ServiceDepartamentos();
export default {
    name: "DepartamentosComponent",
    data() {
        return {
            departamentos: [],
            status: false
        }
    },
    mounted() {
        service.getDepartamentos().then(result => {
            this.departamentos = result;
            this.status = true;
        })
    }
}
</script>
```

El siguiente paso es crear un componente llamado **CreateDepartamento.vue** y lo ponemos a jugar en las rutas y El menú.

Sobre el servicio, creamos un método para insertar un departamento.

Dicho método debe recibir un ID, nombre y localidad, en lugar de eso, voy a enviar un objeto departamento.

#### SERVICEDEPARTAMENTOS.JS

```
insertDepartamento(departamento) {
    return new Promise(function(resolve) {
        let request = "api/departamentos";
        let url = Global.urlApiDepartamentos + request;
        axios.post(url, departamento).then(response => {
            resolve(response)
        })
    })
}
```

Implementamos la petición dentro de **CreateDepartamento**

#### CREATEDEPARTAMENTO

```
<template>
  <div>
    <h3>Create</h3>
    <form v-on:submit.prevent="insertDepartamento()">
      <label>Id departamento</label>
      <input type="number" v-model="departamento.numero" class="form-control"/>
      <label>Nombre</label>
      <input type="text" v-model="departamento.nombre" class="form-control"/>
      <label>Localidad</label>
      <input type="text" v-model="departamento.localidad" class="form-control"/>
      <button class="btn btn-info">Create</button>
    </form>
    <h4 style="color:red">{{ mensaje }}</h4>
  </div>
</template>
<script>
import ServiceDepartamentos from '@/services/ServiceDepartamentos';
const service = new ServiceDepartamentos();
export default {
  name: "CreateDepartamento",
  methods: {
    insertDepartamento() {
      service.insertDepartamento(this.departamento).then(result => {
        this.mensaje = "Insertado!! " + result;
        this.$router.push("/")
      })
    },
    data() {
      return {
        departamento: {
          numero: 0,
          nombre: "",
          localidad: ""
        },
        mensaje: ""
      }
    }
  }
</script>
```

En el siguiente paso, no vamos a utilizar el servicio Api para mostrar detalles, enviaremos los datos por URL (mala praxis)

Creamos un nuevo componente llamado **DetailsDepartamento.vue**

#### ROUTER.JS

```
const myRoutes = [
  { path: "/", component: DepartamentosComponent },
  { path: "/create", component: CreateDepartamento},
  { path: "/details/:id/:nombre/:localidad", component: DetailsDepartamento}
]
```

Implementamos la navegación dentro de **DepartamentosComponent**

#### DEPARTAMENTOSCOMPONENT

```
<router-link class="btn btn-warning">
```

```
:to="'/details/' + dept.numero + '/' + dept.nombre + '/' + dept.localidad">
  Details
</router-link>
```

Implementamos la recuperación de los parámetros y el dibujo dentro de **DetailsDepartamento**

#### DETAILSDEPARTAMENTO

```
<template>
  <div>
    <h1>Details</h1>
    <router-link to="/">Back to list</router-link>
    <ul class="List-group">
      <li class="List-group-item">
        Id: {{ $route.params.id }}
      </li>
      <li class="List-group-item">
        Nombre: {{ $route.params.nombre }}
      </li>
      <li class="List-group-item">
        Localidad: {{ $route.params.localidad }}
      </li>
    </ul>
  </div>
</template>
<script>
  export default {
    name: "DetailsDepartamento"
  }
</script>
```

El siguiente paso es realizar Update. Para ello, vamos a mapear con Routing enviando el ID, buscando el Departamento dentro del componente y mostrando los datos después de cargar.

Creamos un nuevo componente **UpdateDepartamento**

#### ROUTER.JS

```
{ path: "/update/:id", component: UpdateDepartamento }
```

Incluimos un <router-link> dentro de **DepartamentosComponent**

#### DEPARTAMENTOSCOMPONENT

```
<router-link class="btn btn-info"
:to="'/update/' + dept.numero">
  Edit
</router-link>
```

Implementamos el método **findDepartamento()** dentro de **ServiceDepartamentos**

#### SERVICEDEPARTAMENTOS

```
findDepartamento(idDepartamento) {
  return new Promise(function(resolve) {
    let request = "api/departamentos/" + idDepartamento;
    let url = Global.urlApiDepartamentos + request;
    let departamento = {}
    axios.get(url).then(response => {
      console.log("Find departamento");
      departamento = response.data;
      resolve(departamento)
    })
  })
}
```

Implementamos la petición dentro de **UpdateDepartamento.vue**

## UPDATEDEPARTAMENTO

```
<template>
  <div>
    <h1>Update</h1>
    <router-link to="/">Back to index</router-link>
    <form v-on:submit.prevent="updateDepartamento()">
      <v-if="departamento">
        <label>Id</label>
        <input type="number" class="form-control"
          v-model="departamento.numero" disabled/>
        <label>Nombre</label>
        <input type="text" v-model="departamento.nombre" class="form-control"/>
        <label>Localidad</label>
        <input type="text" v-model="departamento.localidad" class="form-control"/>
        <button class="btn btn-info">
          Update
        </button>
      </v-if>
    </form>
  </div>
</template>
<script>
import ServiceDepartamentos from '@/services/ServiceDepartamentos';
const service = new ServiceDepartamentos();
export default {
  name: "UpdateDepartamento",
  methods: {
    updateDepartamento() {
    }
  },
  mounted() {
    let id = this.$route.params.id;
    service.findDepartamento(id).then(result => {
      this.departamento = result
    })
  },
  data() {
    return {
      departamento: null
    }
  }
}
</script>
```

Implementamos la funcionalidad de PUT dentro de **ServiceDepartamentos**

## SERVICEDEPARTAMENTOS

```
updateDepartamento(departamento) {
  return new Promise(function(resolve){
    let request = "api/departamentos";
    let url = Global.urlApiDepartamentos + request;
    axios.put(url, departamento).then(response => {
      resolve(response);
    })
  })
}
```

Realizamos la llamada desde **UpdateDepartamento**

## UPDATEDEPARTAMENTO

```
methods: {
  updateDepartamento() {
    service.updateDepartamento(this.departamento).then(() => {
      console.log("Updated");
      this.$router.push("/details/" +
        + this.departamento.numero +
        + "/" + this.departamento.nombre +
        + "/" + this.departamento.localidad
      )
    })
  }
},
```

El último paso es implementar Eliminar.

Tenemos dos opciones:

1. Creando un componente para la confirmación.
2. Realizar la acción dentro de Home (DepartamentosComponent)

Implementáis Delete.

Vamos a realizarlo con un componente que recibirá el ID y eliminará.

Mapeamos el componente en router.

#### ROUTER.JS

```
const myRoutes = [
  { path: "/", component: DepartamentosComponent },
  { path: "/create", component: CreateDepartamento },
  { path: "/details/:id/:nombre/:localidad", component: DetailsDepartamento },
  { path: "/update/:id", component: UpdateDepartamento },
  { path: "/delete/:id", component: DeleteDepartamento }
]
```

Incluimos un Link por cada departamento dentro de DepartamentosComponent

#### DEPARTAMENTOSCOMPONENT

```
<router-link class="btn btn-danger"
:to="'/delete/' + dept.numero">
  Delete
</router-link>
```

Implementamos en el servicio la funcionalidad para eliminar

#### SERVICEDEPARTAMENTOS

```
deleteDepartamento(idDepartamento) {
  return new Promise(function(resolve) {
    let request = "api/departamentos/" + idDepartamento;
    let url = Global.urlApiDepartamentos + request;
    axios.delete(url).then(response => {
      resolve(response);
    })
  })
}
```

Por último, utilizamos el componente para eliminar directamente.

Versión 2.0, podríamos pedir confirmación.

#### DELETEDEPARTAMENTO

```
<template>
  <div>
    <h1>Delete</h1>
  </div>
</template>
<script>
import ServiceDepartamentos from '@/services/ServiceDepartamentos';
const service = new ServiceDepartamentos();
export default {
  name: "DeleteDepartamento",
  mounted() {
    let id = this.$route.params.id;
    service.deleteDepartamento(id).then(() => {
      this.$router.push("/");
    })
  }
}
</script>
```

## PRACTICA FULL STACK

- Al cambiar un personaje de serie, debemos pedir confirmación con SweetAlert
- Al menos, algún método del Service lo quiero con la librería FETCH de acceso a Apis.
- Vamos a realizar una aplicación que consumirá un Web Api alojado en Azure.
- Dicha aplicación se encargará de poder gestionar Series de televisión y personajes televisivos.
- El diseño se realizará con Bootstrap.
- No tendremos una página Home, simplemente estará en blanco o con una imagen estática, no tiene funcionalidad el componente, como si no lo implementáramos.

El servicio para consumir se encuentra en la siguiente dirección URL:

<https://apiseriespersonajes.azurewebsites.net/index.html>

Funcionalidad de la aplicación:

En el menú de nuestra aplicación, tendremos que cargar las series para poder navegar a un componente Serie donde se nos mostrará un detalle de la serie y sus personajes.

- Menú principal con series: Cargar series para navegar

The screenshot shows a component titled "Serie" for the TV show "Narcos". It features a large thumbnail image of the show's title card. Below the thumbnail, the series name "Narcos" and its IMDB rating "IMDB: 8.8" are displayed. At the bottom is a green button labeled "Personajes". To the right of the main content, there is a sidebar menu listing other series: Juego de tronos, The Mandalorian, Stranger Things, Narcos (with a hand cursor icon indicating it is selected), The Boys, The Big Bang Theory, Campeones, Como conocí a vuestra madre, and La casa de papel.

The screenshot shows a component titled "Serie" for the TV show "The Boys". It features a large thumbnail image of the show's title card. Below the thumbnail, the series name "The Boys" and its IMDB rating "IMDB: 8.7" are displayed. At the bottom is a green button labeled "Personajes". To the right of the main content, there is a sidebar menu listing other series: Juego de tronos, The Mandalorian, Stranger Things, Narcos, The Boys (with a hand cursor icon indicating it is selected), The Big Bang Theory, Campeones, Como conocí a vuestra madre, and La casa de papel.

En el componente de **Serie** podremos visualizar los personajes de una determinada serie. También podemos volver a la serie.

The screenshot shows a component titled "Personajes" for the TV show "The Boys". It features a table with two columns: "Personaje" and "Imagen". The table contains two rows: one for "Patriota" (with an image of the character) and one for "Luz Estelar" (with an image of the character). At the top left of the table is a red button labeled "Volver".

Tendremos la posibilidad de incluir nuevos personajes.

- Insertar nuevo personaje: **Nota: El Id no importa, se genera solo en el servicio, pero debemos pasarlo en el JSON.**

The screenshot shows a component titled "Nuevo personaje". It has a text input field labeled "Nombre:" where the value "Patriota" is typed. Below the input field is a small preview image of the character.

Profundo

Imagen:  
<https://www.ecartelera.com/images/sets/44900/44983.jpg>

Serie:  
The Boys  
**Insertar personaje**

Tendremos la posibilidad de cambiar a un personaje de serie:

- Modificar serie personaje:

**STRANGER THINGS** Home Nuevo personaje Modificar personajes Series ▾

## Personajes y series

Seleccione una serie:  
The Mandalorian

Seleccione un Personaje:  
Koothrappali

**Guardar cambios**

**The Mandalorian**



**Koothrappali**

