

## JQUERY

martes, 1 de octubre de 2024 14:19

Requisitos necesarios:

- 1) Visual Studio Code
- 2) Cuenta Github
- 3) Entorno de trabajo: Windows

JQUERY es un Framework. Un Framework es una tecnología que está basada en otros lenguajes Y que contiene funcionalidades para hacerlo más cómodo o para ofrecer otro tipo de Desarrollo.

En este ejemplo, este Framework está basado en JavaScript.

Utiliza funciones anónimas, es decir, no tenemos que declarar funciones con nombre dentro de nuestro Código.

Ejemplo de Javascript:

```
<button onclick="mostrarMensaje()">Pulsar</button>
```

JS

```
function mostrarMensaje(){  
    alert("algo");  
}
```

Una de las ventajas de JQUERY es que el código de la petición y del método de la llamada Están en el mismo sitio.

Ejemplo de JQUERY

```
$(“button”).click(){  
    alert("algo");  
}
```

<https://www.w3schools.com/jquery/default.asp>

Sintaxis Jquery:

```
$(selector).evento(function() {  
    //CÓDIGO A REALIZAR  
});
```

También tenemos instrucciones para realizar una petición de una acción.

```
$(selector).accion();
```

Lo más importante son los selectores:

Tenemos selectores básicos:

- ID: #idobjeto

```
<button id="idobjeto">
```

- Tipo de objeto: button, h1

```
<button>  
<h1>
```

- Class: .rojo

```
<style>  
    .rojo { .... }  
</style>
```

```
<button class="rojo">
```

```
$(".rojo").accion();
```

Jquery puede ser llamado mediante un fichero JS externo en url o trabajar en local con El fichero descargado.

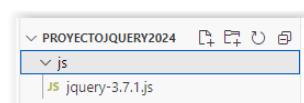
Vamos a trabajar con LOCAL.

```
<script src="jquery.js"></script>  
<script src="https://jquery.com/jquery.js"></script>
```

Metodología de trabajo:

Tendremos un solo proyecto con múltiples páginas. Iremos nombrando las Diferentes páginas mediante la sintaxis [web01primerjquery.html](#)  
Llamamos al proyecto [ProyectoJquery2024](#)

Descargamos Jquery dentro del proyecto y con una carpeta llamada js



```
<!-- COMENTARIOS DENTRO DE ESTE ENTORNO  
| CONTROL + K + C -->  
<!-- DESCOMENTAR UN COMENTARIO: CONTROL + K + U -->
```

```
//LA PRIMERA LINEA DENTRO DE JQUERY SERA
//QUE REALICE LAS ACCIONES AL CARGAR LA
//PAGINA
$(document).ready(function() {
})
```

## Selectores Jquery

Pulsar para algo...

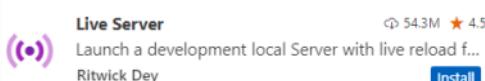
Soy un título azul

Botón azul

### CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .azul {
            color:blue
        }
    </style>
</head>
<body>
    <h1>Selectores Jquery</h1>
    <button id="boton1">Pulsar para algo...</button>
    <h2 class="azul">Soy un título azul</h2>
    <button class="azul">Botón azul</button>
    <p>Aprendiendo Jquery 1</p>
    <p>Aprendiendo Jquery 2</p>
    <p>Aprendiendo Jquery 3</p>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        //AQUI NUESTRO CODIGO JQUERY
        //LA PRIMERA LINEA DENTRO DE JQUERY SERA
        //QUE REALICE LAS ACCIONES AL CARGAR LA
        //PAGINA
        $(document).ready(function() {
            //alert("má primera app!!!");
            //SELECTOR DE ETIQUETA HTML
            // $("#h1").hide();
            // //SELECTOR DE ID
            // $("#boton1").hide();
            //PODEMOS ASOCIAr ACCIONES A OBJETOS
            $("p").click(function() {
                //IMPRESCINDIBLE PARA EL DESARROLLO
                //FRONT ES LANZAR MENSAJES EN CONSOLA
                console.log("Mostrando mensajes en consola Web");
                //TENEMOS UNA PALABRA CLAVE LLAMADA this
                //QUE HACE REFERENCIA AL OBJETO QUE HA
                //REALIZADO LA LLAMADA AL METODO
                $(this).hide();
            })
            $("#boton1").click(function() {
                alert("Soy un botón");
            })
        })
    </script>
</body>
</html>
```

Instalamos una extensión para poder ejecutar páginas en un servidor de forma local  
En nuestro equipo



### ATRIBUTOS HTML/JQUERY

Todas las etiquetas HTML contienen atributos.  
Dependiendo de la etiqueta dichos atributos pueden ser funcionales o no para acceder a ellos.

El atributo class de CSS podemos acceder mediante un método de etiqueta

```
<h1 class="azul">
```

Existen atributos que NO podemos acceder de forma nativa, son atributos de la propia etiqueta

```
<input type="text" id="caja1"/>
```

Para poder acceder a este tipo de atributos necesitamos el método **attr()** de Jquery

```
var tipo = $("#caja1").attr("type"); // "text"
```

También nos sirve para cambiar el atributo de forma dinámica

```
$("#caja1").attr("type", "radio");
```

Es muy útil dependiendo de la etiqueta, por ejemplo, en las imágenes

```

```

```
$("#Imagen1").attr("src", "Imagen2.jpg");
$("#Imagen1").attr("width", "5000");
```

Creamos una carpeta llamada **images** en nuestro proyecto y copiamos un par de imágenes

Creamos una página llamada **web02atributos.html**



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        img {
            width: 150px;
            height: 150px;
        }
    </style>
</head>
<body>
    
    
    
    
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $("img").mouseover(function() {
                $(this).attr("src", "images/joker.png");
            })
            $("img").mouseout(function() {
                $(this).attr("src", "images/batman.jpg");
            })

            $("img").click(function() {
                var imagen = $(this).attr("src");
                console.log("Pulsando sobre " + imagen);
            })
        })
    </script>
</body>
</html>
```

#### ESTILOS CSS JQUERY

Podemos modificar de forma dinámica a los estilos inline dentro de una etiqueta o a Sus estilos generales de un **<style>** o de un **css**.

Ejemplo **INLINE**

```
<h1 style="color:blue"></h1>
```

Se utiliza **CSS**

```
$(“h1”).css(“color”, “red”);
$(“h1”).css(“background-color”, “yellow”);
```

Tenemos la posibilidad de cambiar los estilos a la vez utilizando json.

```
{ “key”: “value”, “key”: “value” }
```

```
$(“h1”).css( { “color”: “red”, “background-color”: “yellow” } );
```

Tenemos la posibilidad de utilizar clases.

```
<style>
    .azul { color:blue; }
</style>
```

```
$(“h1”).addClass(“azul”);
```

```
$("h1").removeClass("azul");
```

#### VALORES ETIQUETAS HTML

Dentro de los controles HTML tenemos que diferenciar dos tipos mostrando datos:

- 1) Etiquetas de contenido: h1, div, p
- 2) Etiquetas de formulario: input, button

Si trabajamos con etiquetas de contenido se utiliza para acceder/modificar el contenido  
Los métodos:

- **html()**: Nos permite injectar contenido HTML dinámicamente
- **text()**: Cambia el texto plano

```
<p id="parrafo">Soy un texto</p>
```

```
var contenido = $("#parrafo").text();
```

Hacen la misma funcionalidad

```
$("#parrafo").text("Nuevo parrafo");
```

```
$("#parrafo").html("Nuevo parrafo");
```

Si utiliza HTML, el código es interpretado

```
$("#parrafo").html("<h2>Nuevo parrafo</h2>");
```

En los controles de formulario, para poder acceder al contenido se utiliza **val()**

```
<input type="text" id="caja"/>
```

```
$("#caja").val("Texto en CAJA");
```

**Nota:** Todo lo que tenemos en cualquier etiqueta es un **string**, aunque sea un número

Creamos una nueva página llamada **web03sumartablanumeros.html**

## Sumar números

Número 1   
Número 2

## La suma es 11

Número tabla multiplicar

7 \* 1 7  
7 \* 2 14  
7 \* 3 21  
7 \* 4 28  
7 \* 5 35  
7 \* 6 42  
7 \* 7 49  
7 \* 8 56  
7 \* 9 63  
7 \* 10 70

#### CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Sumar números</h1>
    <label>Número 1</label>
    <input type="text" id="cajanumero1"/><br/>
    <label>Número 2</label>
    <input type="text" id="cajanumero2"/><br/>
    <button id="botonsumar">Sumar números</button>
    <h1 id="resultado"></h1>
    <hr/>
    <label>Número tabla multiplicar</label>
    <input type="text" id="cajatabla"/>
    <button id="botontabla">Tabla multiplicar</button>
    <table id="tabla"></table>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $("#botontabla").click(function() {
                var num = parseInt($("#cajatabla").val());
                var html = "";
                for (var i = 1; i <= 10; i++){
                    var operacion = num * i;
                    var texto = num + " * " + i;
                    html += "<tr>";
                }
            })
        })
    </script>
```

```

        html += "<td>" + texto + "</td>";
        html += "<td>" + operacion + "</td>";
        html += "</tr>";
    }
    $("#tabla").html(html);
})
$("#botonsumar").click(function() {
    var num1 = $("#cajanumero1").val();
    var num2 = $("#cajanumero2").val();
    var suma = parseInt(num1) + parseInt(num2);
    $("#resultado").text("La suma es " + suma);
})
})
</script>
</body>
</html>

```

#### PRACTICA

Creamos una nueva página llamada **web04evaluarnumeros.html**

Tendremos tres cajas para escribir tres números.

Necesitamos, mediante un botón, mostrar el MAYOR, MENOR e INTERMEDIO de los Tres números y su SUMA.

Sintaxis:

```

if (num1 == 0) {
} else if (num1 == 1) {
} else {
}

```

Operadores relaciones:

- And: &&
- Or: ||
- NOT: !

```

if (num1 == 0 && num2 == 1){
}

```

## Evaluar números

Número 1	<input type="text" value="7"/>
Número 2	<input type="text" value="55"/>
Número 3	<input type="text" value="1"/>

**Evaluar números**

## Mayor: 55, Menor: 1, Intermedio: 7

#### CODIGO HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Evaluar números</h1>
    <label>Número 1</label>
    <input type="text" id="cajanumero1"/><br/>
    <label>Número 2</label>
    <input type="text" id="cajanumero2"/><br/>
    <label>Número 3</label>
    <input type="text" id="cajanumero3"/><br/>
    <button id="botonevaluar">Evaluar números</button>
    <h1 style="color:green" id="resultado"></h1>
    <script src="js/jquery-3.7.1.js"></script>
<script>
    $(document).ready(function() {
        $("#botonevaluar").click(function() {
            var num1 = parseInt($("#cajanumero1").val());
            var num2 = parseInt($("#cajanumero2").val());
            var num3 = parseInt($("#cajanumero3").val());
            var mayor, menor, intermedio;
            if (num1 >= num2 && num1 >= num3){
                mayor = num1;
            } else if (num2 >= num1 && num2 >= num3){
                mayor = num2;
            } else{
                mayor = num3;
            }
            if (num1 <= num2 && num1 <= num3){
                menor = num1;
            } else if (num2 <= num1 && num2 <= num3){
                menor = num2;
            }
            intermedio = num3;
            $("#resultado").text("Mayor: " + mayor + ", Menor: " + menor + ", Intermedio: " + intermedio);
        })
    })
</script>

```

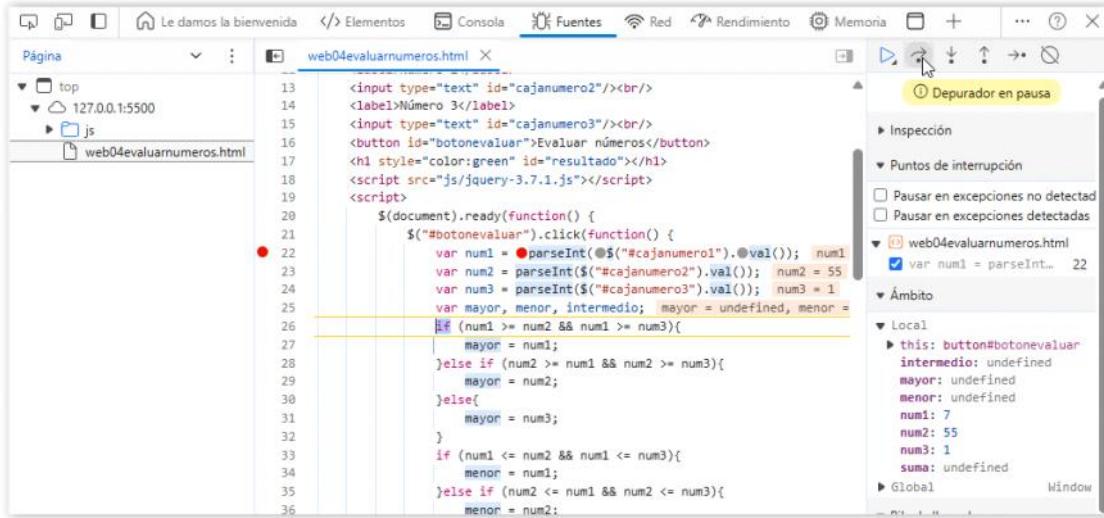
```

        menor = num2;
    }else{
        menor = num3;
    }
    var suma = num1 + num2 + num3;
    intermedio = suma - menor - mayor;
    $("#resultado").text("Mayor: " + mayor + ", Menor: "
        + menor + ", Intermedio: " + intermedio);
})
})
</script>
</body>
</html>

```

#### DEPURAR ENTORNO WEB

Para depurar estos códigos, vamos a utilizar el explorador Web. Con las herramientas del desarrollador podemos ir paso a paso y visualizar lo que estamos haciendo con las variables. La tecla F12 abre Developer tools. Dentro de **Fuentes** tenemos los ficheros de nuestro servidor. Con la tecla F10 podemos ir paso a paso por nuestro código y visualizar los cambios de variables.



Creamos una nueva página llamada **web05dianacimiento.html**

#### CALCULAR DÍA DE NACIMIENTO DE LA SEMANA

Pedir una fecha al usuario para calcular el día de la semana que nació.

Tenemos que tener la tabla de días de la semana para la correspondencia comenzando en sábado:

DÍA	NÚMERO
Sábado	0
Domingo	1
Lunes	2
Martes	3
Miércoles	4
Jueves	5
Viernes	6

Debemos pedir el **día**, el número de **mes** y el **año** que el usuario haya nacido.

A partir de esto datos hay que calcular lo siguiente para averiguar el día de la semana de nacimiento:

Ejemplo → 15/06/1997

Hay que tener en cuenta el mes para realizar el cálculo, si el mes es Enero, el Mes será 13 y restaremos uno al año. Si el Mes es Febrero, el Mes será 14 y restaremos uno al año.

Para poder calcular las el número final de la semana debemos seguir los siguientes pasos:

1. Multiplicar el Mes más 1 por 3 y dividirlo entre 5

$$(6 + 1) * 3 / 5 \rightarrow 4$$

2. Dividir el año entre 4  
1997 / 4 → 499

3. Dividir el año entre 100

$$1997 / 100 \rightarrow 19$$

4. Dividir el año entre 400

$$1997 / 400 \rightarrow 4$$

5. Sumar el dia, el doble del mes, el año, el resultado de la operación 1 , el resultado de la operación 2 , menos el resultado de la operación 3 más la operación 4 más 2.

$$15 + (6 * 2) + 1997 + 4 + 499 - 19 + 4 + 2 \rightarrow 2514$$

6. Dividir el resultado anterior entre 7.

$$2514 / 7 \rightarrow 359$$

7. Restar el número del paso 5 con el número del paso 6 por 7.

$$2514 - (359 * 7) \rightarrow 1$$

8. Miramos la tabla y vemos que el número 1 corresponde a **DOMINGO**

Dia: 8
Mes: 10
Año: 2018
<input type="button" value="Mostrar dia de la semana"/>

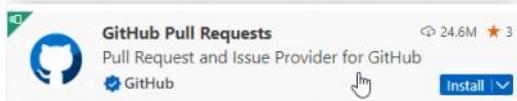
LUNES

CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Día nacimiento semana</h1>
    <label>Día</label>
    <input type="text" id="cajadia"/><br/>
    <label>Mes</label>
    <input type="text" id="cajames"/><br/>
    <label>Año</label>
    <input type="text" id="cajaanyo"/><br/>
    <button id="botonevaluar">Día nacimiento</button>
    <h1 id="resultado"></h1>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $("#botonevaluar").click(function() {
                var dia = parseInt($("#cajadia").val());
                var mes = parseInt($("#cajames").val());
                var anyo = parseInt($("#cajaanyo").val());
                if (mes == 1){
                    mes = 13;
                    anyo--;
                }else if (mes == 2){
                    mes = 14;
                    anyo -= 1;
                }
                var op1 = Math.trunc(((mes + 1) * 3) / 5);
                var op2 = Math.trunc(anyo / 4);
                var op3 = Math.trunc(anyo / 100);
                var op4 = Math.trunc(anyo / 400);
                var op5 = dia + (mes * 2) + anyo + op1 + op2 - op3 +
op4 + 2;
                var op6 = Math.trunc(op5 / 7);
                var resultado = Math.trunc(op5 - (op6 * 7));
                var diaSemana = "";
                if (resultado == 0){
                    diaSemana = "SABADO";
                }else if (resultado == 1){
                    diaSemana = "DOMINGO";
                }else if (resultado == 2){
                    diaSemana = "LUNES";
                }else if (resultado == 3){
                    diaSemana = "MARTES";
                }else if (resultado == 4){
                    diaSemana = "MIERCOLES";
                }else if (resultado == 5){
                    diaSemana = "JUEVES";
                }else if (resultado == 6) {
                    diaSemana = "VIERNES";
                }
                $("#resultado").text(diaSemana);
            })
        })
    </script>
</body>
</html>
```

Instalar esta App que es Git for Windows

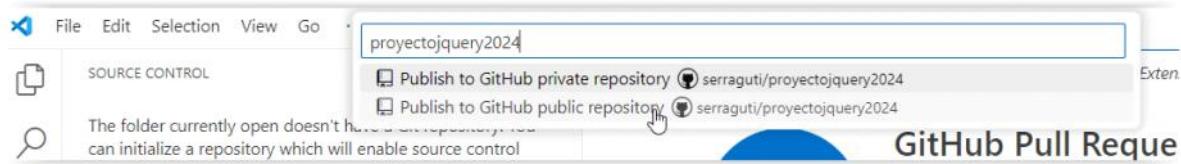
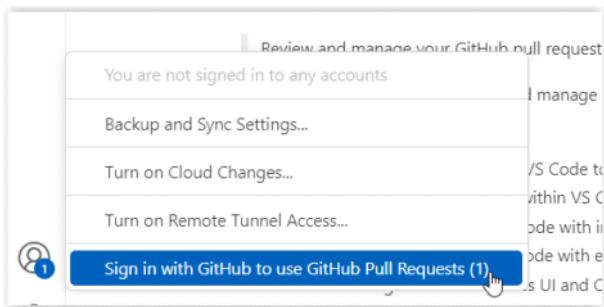
<https://git-scm.com/download/win>



Abrimos CMD

```
C:\>git config --global user.name "USERGITHUB"
```

```
C:\>git config --global user.email "pacoserranox@gmail.com"
```



Mi repositorio para este proyecto es este

[serraguti/proyectojquery2024Octubre \(github.com\)](https://github.com/serraguti/proyectojquery2024Octubre)

Mi nombre de usuario **serraguti**

#### CONTROLES Y EVENTOS DINAMICOS

Los elementos dinámicos no dejan de ser un dibujo pero, en ocasiones, podríamos necesitar que tuvieran acciones/eventos sobre ellos.

Los controles/dibujos podemos generarlos de dos formas:

- 1) html(): El método html genera código dinámico, pero los controles asociados NO tienen Eventos jquery.
- 2) Controles lógicos propios de jquery: Este tipo de dibujos pueden tener asociados los Eventos, cuesta "algo" más generarlos, pero nos permiten jugar con los eventos en jquery

Creamos una página llamada **web06testeventos.html**

#### CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Controles dinámicos</h1>
  <button>Pulsar para hacer algo...</button>
  <div id="prueba"></div>
  <script src="js/jquery-3.7.1.js"></script>
  <script>
    $(document).ready(function() {
      $("button").click(function() {
        alert("boton pulsado");
        $("#prueba").html(
          "<button id='miboton' onclick='mostrarMensaje()'>
          boton NUEVO</button>");
      })
      $("#miboton").click(function() {
        alert("boton NUEVO pulsado!!!");
      })
    })
    function mostrarMensaje() {
      alert("boton NUEVO pulsado!!!");
    }
  </script>
```

```

    </script>
</body>
</html>

```

Un Handler es un manejador que asociamos a un control dinámico de Jquery

```

var parrafo = $("<p>");
parrafo.text("Nuevo parrafo");
parrafo.click(function() {
    //CÓDIGO DEL OBJETO GENERADO
})

```

También podemos crear el control con características en su propia definición.

```

var parrafo = $("<p>", { "text": "Nuevo párrafo", "style": "color:green" });

```

Para poder dibujar dicho control dentro de nuestra página necesitamos un control PADRE

```

<div id="padre"></div>

```

Tenemos dos formas de incluir dicho control

- 1) **append:** Se utiliza Padre-Hijo: \$("#padre").append(parrafo)
- 2) **appendTo:** Se utiliza Hijo-Padre: parrafo.appendTo(\$("#padre"))

# Controles dinámicos

Introduzca un texto  Generar control

TEXTO 1

Pulsado!!!

Pulsado!!!

TEXTO 5

#### CODIGO HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Controles dinámicos</h1>
    <label>Introduzca un texto</label>
    <input type="text" id="cajatexto"/>
    <button id="botongenerar">Generar control</button><br/>
    <div id="contenedor"></div>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $("#botongenerar").click(function() {
                var texto = $("#cajatexto").val();
                //CREAMOS UN PARRAFO POR CADA PULSACION
                //LO HACEMOS CON JQUERY PARA INCLUIR UN LISTENER
                var parrafo =
                    $("<p>", { "text": texto, "style": "color:red" });
                //AÑADIMOS AL CONTENEDOR EL PARRAFO
                $("#contenedor").append(parrafo);
                //INCLUIMOS UN EVENTO A CADA PARRAFO GENERADO
                parrafo.click(function() {
                    $(this).css("color", "blue");
                    $(this).text("Pulsado!!!");
                })
            })
        })
    </script>
</body>
</html>

```

Creamos una nueva página llamada [web07generarimagenes.html](#)

# Generar imágenes



## CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <style>
        img {
            width: 90px;
            height: 90px;
        }
    </style>
</head>
<body>
    <h1>Generar imágenes</h1>
    <input type="range" id="rango" min="1" max="20" value="1"/>
    <span id="valorrango"></span>
    <div id="contenedor"></div>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $("#rango").change(function() {
                var numImagenes = parseInt($(this).val());
                $("#valorrango").text(numImagenes);
                //LIMPIAMOS EL CONTENEDOR DE IMAGENES
                $("#contenedor").text("");
                for (var i = 1; i <= numImagenes; i++){
                    var imagen = $("<img>"
                        , { "src": "images/batman.jpg" });
                    $("#contenedor").append(imagen);
                    imagen.mouseover(function() {
                        $(this).attr("src", "images/joker.png");
                    })
                    imagen.mouseout(function() {
                        $(this).attr("src", "images/batman.jpg");
                    })
                }
            })
        })
    </script>
</body>
</html>
```

## EJEMPLO SUMAR BOTONES

Debemos crear una página para generar botones dinámicamente, 10 botones por ejemplo.

Al iniciar la página dichos botones tendrán un número generado mediante random. Google.

Al pulsar cada botón, lo que haremos es ir sumando su valor dentro de un h1.

# Generar botones

57

Número 6 Número 1 Número 42 Número 43 Número 7

## CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Generar botones</h1>
    <h1 style="color:blue" id="suma">0</h1>
    <div id="contenedor"></div>
```

```

<script src="js/jquery-3.7.1.js"></script>
<script>
    $(document).ready(function() {
        for (var i = 1; i <= 10; i++){
            var aleatorio = parseInt(Math.random() * 100) + 1;
            var boton = $("<button>");
            boton.text("Número " + aleatorio);
            boton.val(aleatorio);
            $("#contenedor").append(boton);
            boton.click(function() {
                var numero = parseInt($(this).val());
                var suma = parseInt($("#suma").text());
                suma += numero;
                $("#suma").text(suma);
            })
        }
    })
</script>
</body>
</html>

```

#### BUCLES DE REFERENCIA

La mayoría de lenguajes tienen bucles de referencia a objeto.

Son súper útiles para recorrer conjuntos de elementos sin necesidad de estar convirtiendo.

Pueden ser conjuntos de números, botones, clases...

```

var botones = [boton1, boton2, boton3]

for (var i = 0; i < botones.length; i++){
    //VARIABLE DE REFERENCIA
    var boton = botones[i];
}

```

Dentro del lenguaje Jquery tenemos un bucle de referencia llamado **each**

Nos permite declarar y utilizar directamente los objetos de un conjunto.

Por ejemplo, este código:

```

$("button").each(function(){
    //CON this HACEMOS REFERENCIA A CADA ELEMENTO DEL CONJUNTO
    $(this).text("Botón...");
})

```

Creamos una nueva página **web09buclestexto.html**



#### CODIGO HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Bucles referencia</h1>
    <p>Parrafo 1</p>
    <p>Parrafo 2</p>
    <p>Parrafo 3</p>
    <p>Parrafo 4</p>
    <p>Parrafo 5</p>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $("p").click(function() {
                //AL PULSAR CUALQUIER <p> CAMBIAMOS
                //CADA PARRAFO CON UN COLOR...
                var colores = ["blue", "yellow", "red", "green",
                "fuchsia"];
                $("p").each(function() {
                    var indice = parseInt(Math.random() *
colores.length);
                    var color = colores[indice];
                    $(this).css("background-color", color);
                })
            })
        })
    </script>

```

```
</script>
</body>
</html>
```

#### ATRIBUTOS SIN VALOR

Dentro de los atributos HTML existen ciertos atributos que NO tienen valor asociado. Dichos atributos no podemos recuperarlos con `attr` tal y como hemos visto.

#### CON VALOR

```

```

#### SIN VALOR

```
<input id="caja" type="text" disabled/>
```

Si necesitamos recuperar el valor true/false de un atributo de este tipo tenemos dos posibilidades

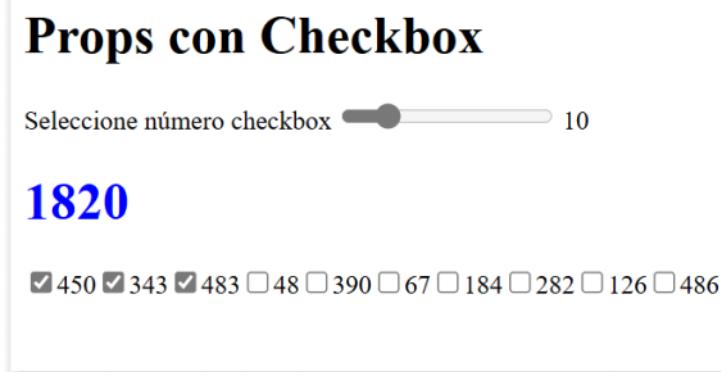
- 1) `prop`
- 2) `:is`

**Nota:** IS necesita los dos puntos ANTES de la propiedad/atributo

```
var valor = $("#caja").prop("disabled")
```

```
var valor = $("#caja").is(":disabled")
```

Creamos una página llamada `web10checkboxprops.html`



#### CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Props con Checkbox</h1>
    <label>Seleccione número checkbox</label>
    <input type="range" id="rango" min="1" max="50" value="1"/>
    <span id="valorrango"></span>
    <h1 style="color:blue" id="resultado">0</h1>
    <div id="contenedor"></div>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $("#rango").change(function() {
                var numeroElementos = parseInt($(this).val());
                $("#valorrango").text(numeroElementos);
                $("#contenedor").html("");
                for (var i = 1; i <= numeroElementos; i++){
                    var aleatorio = parseInt((Math.random() * 500))
+ 1;
                    var chk = $("<input>"
                        , { "value": aleatorio, "type":
"checkbox"});
                    var label = $("<label>", { "text": aleatorio});
                    $("#contenedor").append(chk);
                    $("#contenedor").append(label);
                    chk.change(function() {
                        var resultado =
parseInt($("#resultado").text());
                        var valor = parseInt($(this).val());
                        //DEBEMOS PREGUNTAR POR EL VALUE DEL
CHECKBOX
                        //PARA SUMAR O RESTAR
                        if ($(this).is(":checked") == true){
                            resultado += valor;
                        }else{
                            resultado -= valor;
                        }
                        $("#resultado").text(resultado);
                    })
                }
            })
        })
    </script>
```

```

        </script>
    </body>
</html>

VERSION 2

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Props con Checkbox</h1>
    <label>Seleccione número checkbox</label>
    <input type="range" id="rango" min="1" max="50" value="1"/>
    <span id="valorrango"></span>
    <h1 style="color:blue" id="resultado"></h1>
    <div id="contenedor"></div>
    <script src="js/jquery-3.7.1.js"></script>
<script>
    $(document).ready(function() {
        $("#rango").change(function() {
            var numeroElementos = parseInt($(this).val());
            $("#valorrango").text(numeroElementos);
            $("#contenedor").html("");
            for (var i = 1; i <= numeroElementos; i++) {
                var aleatorio = parseInt((Math.random() * 500)
+ 1;
                var chk = $("<input>"
                    , { "value": aleatorio, "type":
"checkbox"});
                var label = $("<label>", { "text": aleatorio});
                $("#contenedor").append(chk);
                $("#contenedor").append(label);
                chk.change(function() {
                    var resultado = 0;
                    //RECORREMOS TODOS LOS CHECKBOX
                    $("input[type='checkbox']:checked").each(fun
ction() {
                        var valor = parseInt($(this).val());
                        resultado += valor;
                    })
                    $("#resultado").text(resultado);
                })
            }
        })
    })
</script>
</body>
</html>

```

#### ATRIBUTOS DATA-KEY

Esto es una herramienta para los desarrolladores. No sirve para nada de forma visual, es un Atributo que podemos agregar a cualquier etiqueta y almacenar nuestra información.

Podemos realizar su funcionalidad de forma estática dentro del HTML o mediante JQUERY asignar o recuperar algún Key/Value.

Pongamos que tenemos un formulario con datos de un usuario. Necesitamos mostrar Sus datos, pero no representar todos visualmente.

Debemos incluir etiquetas **data-key**

HTML

```
<input type="text" value="Paco Garcia" id="cajausuario"
data-dni="12345678X" data-idusuario="14"/>
```

Para recuperar/modificar una etiqueta dinámicamente:

```
var dni = $("#cajausuario").data("dni");
var id = $("#cajausuario").data("idusuario");
$("#cajausuario").data("dni", "77777777J");
```

```
<input type="text" list="listausuarios"/>
<datalist id="listausuarios">
    <option value="Paco" data-dni="12345555X"/>
    <option value="Ana" data-dni="84565464X"/>
    <option value="Adrian"/>
    <option value="Lucia"/>
    <option value="Antonia"/>
    <option value="Antonio"/>
</datalist>
```

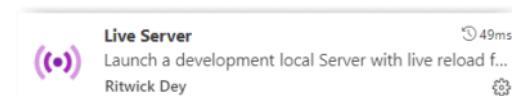
#### CONSUMO ELEMENTOS DE SERVIDOR

Esto es HTML, no existen accesos a datos puros directamente con una base de datos.

Para poder acceder a datos, necesitamos utilizar Servicios en Cloud.

Para poder trabajar con servicios es necesario tener nuestra App dentro de un **server**

NO SE PUEDE HACER EN LOCAL



Un servicio son datos en la nube, cualquier dato y cualquier formato se puede utilizar

Los datos del servicio pueden salir de cualquier lugar, otro servicio, una base de datos...

Un servicio nos puede permitir recuperar datos o modificar/insertar datos.

Los servicios utilizan un concepto muy importante llamado **Serialización**

La serialización implica crear un objeto en un punto determinado y poder recuperar Ese objeto con su forma en otro punto distinto.

Tenemos varios tipos de formato:

- 1) **Json:** Formato con Key:Value

```
{  
    "nombre": "Alumno",  
    "apellidos": "MCSD",  
    "edicion": 2024  
}
```

- 1) **Xml:** Formato de etiquetas como HTML

```
<alumno>  
    <nombre>Alumno</nombre>  
    <apellidos>MCSD</apellidos>  
    <edicion>2024</edicion>  
</alumno>
```

Tenemos dos tipos de servicios:

- 1) **Servicios Api/Rest:** Devuelven la información en un tipo **string**, es decir, un texto  
Y nosotros debemos convertir dicho texto a formato.
- 1) **Servicios WCF:** Devuelven la información en clases y objetos. Guardamos un objeto Dentro de un punto y recuperaremos dicho objeto formateado mediante **binario**

Comenzaremos con formato XML. Son datos formados como un documento HTML, tiene jerarquía.

Vamos a comenzar en LOCAL, pero LOCAL o URL es lo mismo.

Para consumir servicios XML se utiliza un método **get()** de Jquery

```
$get("URL", function(data) {  
    La información viene dentro de data y con formato XML, no con formato string  
});
```

Dentro de los servicios Api existen varios métodos estandar. Todo depende del Servicio que estemos consumiendo.

- **GET:** Sirve para recuperar datos.
- **POST:** Sirve para insertar datos (mentira)
- **PUT:** Sirve para modificar datos
- **DELETE:** Sirve para eliminar datos

Nosotros vamos a comenzar con GET, utilizando el método **get()**

Si quisieramos consumir el resto de métodos, necesitamos hacerlo mediante **ajax()**

En el momento de leer cualquier servicio debemos recordar que es asíncrona la petición. Esto quiere decir que hacemos la petición, pero no sabemos CUANDO nos devolverá Los datos/respuesta.

```
console.log("Antes del servicio");  
$.get("URL", function(data) {  
    console.log("Leyendo servicio");  
})  
console.log("Después del servicio");
```

El orden de los elementos sería el siguiente:

- Antes del servicio
- Después del servicio
- Leyendo servicio

Como son peticiones asíncronas, no es lo mismo hacerlo en la página que en un Fichero JS externo.

Con la variable **data**, al ser una variable de tipo XML, podemos actuar sobre ella mediante Métodos JQUERY. Si tratamos a dicha variable con JQUERY, podemos aplicar métodos Como **find()**

```
$(data).find("nombre")  
$(data).find("nombre").each(function() {  
    $(this).text(); //EL VALOR INTERNO DEL TEXTO DE UN NODO...
```

```
$(this).attr(); //EL VALOR DE ALGUN ATRIBUTO...
```

```
});
```

También tenemos algunos métodos de posicionamiento para conjuntos:

```
.first(), last(), eq(indice)
```

XML diferencia mayúsculas de minúsculas

Sobre nuestro proyecto, creamos una carpeta llamada **documents** y copiamos el primer Documento llamado **empleados.xml**

Creamos una nueva página llamada **web11lecturaxmempleados.html**

The screenshot shows a simple web application titled "Servicio XML Empleados". It features a button labeled "Leer XML" which, when clicked, displays a list of names: JIMENEZ, MARTINEZ, NEGRO, GUTIERREZ. To the right of the main content, a developer tools sidebar is open, showing an error message: "Failed to load resource: the server".

#### CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Servicio XML Empleados</h1>
    <button id="botonleer">Leer XML</button>
    <ul id="listaempleados"></ul>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $("#botonleer").click(function() {
                console.log("Antes del servicio");
                $.get("documents/empleados.xml", function(data){
                    console.log("Leyendo servicio");
                    var html = "";
                    //DEBEMOS BUSCAR TODAS LAS ETIQUETAS <APELIDO>
                    $(data).find("APELIDO").each(function() {
                        //COMO EL CONTENIDO DE LA ETIQUETA ES LO
                        //QUE BUSCAMOS LEEMOS DIRECTAMENTE EL
                })
                var apellido = $(this).text();
                html += "<li>" + apellido + "</li>";
            })
            $("#listaempleados").html(html);
        })
        console.log("Después del servicio");
    })
</script>
</body>
</html>
```

Creamos una página llamada **web12departamentosxml.html**

The screenshot shows a web application titled "Departamentos XML". It features a button labeled "Cargar departamentos" and a list of department codes and names: 10 CONTABILIDAD ELCHE, 20 VENTAS MADRID, 30 INVESTIGACION GRANADA, 40 PRODUCCION SALAMANCA.

#### CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
```

```

<h1>Departamentos XML</h1>
<button id="botoncargar">Cargar departamentos</button>
<table id="tabladedepartamentos"></table>
<script src="js/jquery-3.7.1.js"></script>
<script>
$(document).ready(function() {
    $("#botoncargar").click(function() {
        $.get("documents/departamentos.xml", function(data)
{
            console.log("leyendo...");
            var html = "";
            $(data).find("DEPT").each(function() {
                html += "<tr>";
                var numero = $(this).attr("DEPT_NO");
                var nombre =
$(this).find("DNOMBRE").first().text();
                var localidad =
$(this).find("LOC").first().text();
                html += "<td>" + numero + "</td>";
                html += "<td>" + nombre + "</td>";
                html += "<td>" + localidad + "</td>";
                html += "</tr>";
            })
            $("#tabladedepartamentos").html(html);
        })
    })
})
</script>
</body>
</html>

```

#### FILTRAR DATOS DOCUMENTO XML

No siempre podremos filtrar la información al recuperar datos de un servicio, la cuestión del Filtro es propia del servicio, normalmente nos ofrece los datos ya filtrados, pero a veces no Es así.

En el momento de filtrar tenemos dos opciones con jquery:

- Atributos
- Valor de las etiquetas

```

<EMPRESA ID="1">
    <NOMBRE>MICROSOFT</NOMBRE>
</EMPRESA>
<EMPRESA ID="2">
    <NOMBRE>GOOGLE</NOMBRE>
</EMPRESA>
<EMPRESA ID="3">
    <NOMBRE>ORACLE</NOMBRE>
</EMPRESA>
<EMPRESA ID="4">
    <NOMBRE>AMAZON</NOMBRE>
</EMPRESA>

```

Si filtramos por atributo lo bueno es que podemos utilizar selectores CSS

EMPRESA[ID=2] --> GOOGLE

Filtrar por el valor de un campo se utiliza el método :contains(valor)

EMPRESA:contains(AMAZON)

Todo esto se aplica con el método **find()**, es decir, una vez que ya hemos recuperado los Datos, por ejemplo:

`$(data).find(FILTRO)`

Creamos una página llamada `web13filtrosxml.html`

## Filtros XML

Introduzca ID

Introduzca localidad

# Departamento: CONTABILIDAD, Localidad: ELCHE

- Id: 60, RRHH
- Id: 70, COSTURA

#### CODIGO HTML

```
<!DOCTYPE html>
```

```

<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Filtros XML</h1>
    <label>Introduzca ID</label>
    <input type="text" id="cajaid"/>
    <button id="botonbuscaratributo">Buscar departamento</button>
    <hr/>
    <label>Introduzca localidad</label>
    <input type="text" id="cajalocalidad"/>
    <button id="botonbuscarlocalidad">Buscar localidades</button>
    <hr/>
    <h1 style="color:blue" id="mensaje"></h1>
    <ul id="listadepartamentos"></ul>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $("#botonbuscaratributo").click(function() {
                var id = $("#cajaid").val();
                $.get("documents/departamentos.xml", function(data){
                    console.log("leyendo...");
                    //PARA FILTRAR NECESITAMOS LA SIGUIENTE
                    //SINTAXIS: DEPT[DEPT_NO=10]
                    var filtro = "DEPT[DEPT_NO=" + id + "]";
                    //EL ID ES UNICO, POR LO QUE NO HACEMOS EACH
                    var nododept = $(data).find(filtro).first();
                    var nombre = $(nododept).find("DNOMBRE").text();
                    var localidad = $(nododept).find("LOC").text();
                    $("#mensaje").text(
                        "Departamento: " + nombre + ", Localidad: " +
                        localidad);
                })
            })
            $("#botonbuscarlocalidad").click(function() {
                var localidad = $("#cajalocalidad").val();
                $("#listadepartamentos").html("");
                $.get("documents/departamentos.xml", function(data){
                    //FILTRAR POR CONTENIDO LOCALIDAD
                    //LOC[contains(GIJON)
                    var filtro = "LOC[contains(" + localidad + ")";
                    //SON VARIOS ELEMENTOS, POR LO QUE DEBEMOS
                    //RECORRER
                    var html = "";
                    if ($(data).find(filtro).length == 0){
                        $("#mensaje").text("No existen
departamentos");
                    }else{
                        $(data).find(filtro).each(function() {
                            var nodopadre = $(this).parent();
                            var id = $(nodopadre).attr("DEPT_NO");
                            var nombre =
$(nodopadre).find("DNOMBRE").text();
                            html += "<li>Id: " + id + ", " + nombre
+ "</li>";
                        })
                        $("#listadepartamentos").html(html);
                    }
                })
            })
        })
    </script>
</body>
</html>

```

#### PRACTICA ALUMNOS NOTAS

Queremos mostrar los alumnos que tengan una nota igual o superior a lo que hemos Escrito en la caja.

Alumnos notas XML		
Introduzca nota	<input type="text" value="10"/>	<input type="button" value="Buscar alumnos"/>
Nombre	Apellidos	Nota
Jose Maria Gutierrez		10
Adrian	Ramos Zidane	10

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Alumnos notas XML</h1>
    <label>Introduzca nota</label>
    <input type="text" id="cajanota"/>
    <button id="botonbuscar">Buscar alumnos</button>
    <table border="1" id="tablaalumnos">
        <thead>
            <tr>
                <th>Nombre</th>

```

```

        <th>Apellidos</th>
        <th>Nota</th>
    </tr>
</thead>
<tbody></tbody>
</table>
<script src="js/jquery-3.7.1.js"></script>
<script>
$(document).ready(function() {
    $("#botonbuscar").click(function() {
        $.get("documents/alumnos.xml", function(data){
            console.log("leyendo");
            var nota = parseInt($("#cajanota").val());
            var html = "";
            $(data).find("alumno").each(function() {
                var notaAlumno =
                    parseInt($(this).find("nota").text());
                if (notaAlumno >= nota){
                    var nombre =
$(this).find("nombre").text();
                    var apellidos =
$(this).find("apellidos").text();
                    html += "<tr>";
                    html += "<td>" + nombre + "</td>";
                    html += "<td>" + apellidos + "</td>";
                    html += "<td>" + notaAlumno + "</td>";
                    html += "<tr>";
                }
            })
            $("#tablaalumnos tbody").html(html);
        })
    })
})
</script>
</body>
</html>

```

Vamos a realizar una práctica en la que leeremos un servicio real en la nube  
 Dependiendo del servicio, nos ofrece los datos con etiquetas directamente o nos  
 Ofrece los datos con xmlns que es a lo que se llama namespace XML

```

Response body
<?xml version="1.0" encoding="utf-16"?>
<ArrayOfDepartamento xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Departamento>
<IdDepartamento>10</IdDepartamento>
<Nombre>DISNEYLAND</Nombre>
<Localidad>PARIS</Localidad>
</Departamento>
<Departamento>
<IdDepartamento>20</IdDepartamento>
<Nombre>PARQUE WARNER</Nombre>
<Localidad>MADRID</Localidad>
</Departamento>
<Departamento>
<IdDepartamento>30</IdDepartamento>
<Nombre>PORT AVENTURA</Nombre>
<Localidad>TABARINA</Localidad>
</Departamento>
<Departamento>
<IdDepartamento>40</IdDepartamento>
<Nombre>TERRA MITICA</Nombre>
<Localidad>BENIDORM</Localidad>
</Departamento>
</ArrayOfDepartamento>

```

Con XMLNS

```

<CustomersResponse xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/Northwind.ServiceModel.Types">
<ResponseStatus xmlns:d2p1="http://schemas.servicestack.net/types" i:nil="true"/>
<Results xmlns:d2p1="http://schemas.datacontract.org/2004/07/Northwind.ServiceModel.Types">
<d2p1:Customer>
<d2p1:Address>Obere Str. 57</d2p1:Address>
<d2p1:City>Berlin</d2p1:City>
<d2p1:CompanyName>Alfreds Futterkiste</d2p1:CompanyName>
<d2p1>ContactName>Maria Anders</d2p1>ContactName>
<d2p1>ContactTitle>Sales Representative</d2p1>ContactTitle>
<d2p1:Country>Germany</d2p1:Country>
<d2p1:Fax>030-0076545</d2p1:Fax>
<d2p1:Id>ALFKI</d2p1:Id>
<d2p1:Phone>030-0074321</d2p1:Phone>
<d2p1:PostalCode>12209</d2p1:PostalCode>
<d2p1:Region i:nil="true"/>
</d2p1:Customer>

```

Lo primero de todo, para consumir un servicio API XML es necesario que el propio servicio  
 Nos de permiso, es decir, que tenga CORS habilitado el servicio.  
 CORS es una funcionalidad en los servicios que impiden que se pueda acceder a sus  
 Elementos para recuperarlos.

<https://northwind.netcore.io/customers.xml>

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

Creamos una página llamada **web15customersservice.html**

```

$( "#botoncargarcustomers" ).click(function(){
    var url = "https://northwind.netcore.io/customers.xml";
    $.get(url, function(data){
        console.log("Leyendo servicio");
        var numeroElementos =
            $(data).find("d2p1\\:\\Customer").length;
        $("#mensaje").text("Elementos: " + numeroElementos);
    })
    console.log("Despues servicio");
})

```

Vamos a realizar lo siguiente:

- Cargaremos todos los clientes (nombre) en un desplegable al iniciar la aplicación.
- Al seleccionar un cliente, mostramos sus datos

## Customers XML Service

[Cargar Customers](#) [Antonio Moreno](#) [Cargar departamentos](#)

### Antonio Moreno Taquería

#### CODIGO HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Customers XML Service</h1>
    <button id="botoncargarcustomers">Cargar Customers</button>
    <select id="selectcustomers"></select>
    <button id="botoncargardepartamentos">Cargar
departamentos</button>
    <h1 id="mensaje" style="color:fuchsia"></h1>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $("#botoncargarcustomers").click(function(){
                var url =
"https://northwind.netcore.io/customers.xml";
                $.get(url, function(data){
                    console.log("Leyendo servicio");
                    var numeroElementos =
                        $(data).find("d2p1\\:\\Customer").length;
                    $("#mensaje").text("Elementos: " +
numeroElementos);
                    //RECORREMOS LOS CUSTOMERS
                    $(data).find("d2p1\\:\\Customer").each(function()
{
                        var contactName =
                            $(this).find("d2p1\\:\\ContactName").text();
                        var idCustomer =
                            $(this).find("d2p1\\:\\Id").text();
                        var option = $("<option>" +
                            , {"text": contactName, "value": idCustomer});
                        $("#selectcustomers").append(option);
                    })
                    console.log("Despues servicio");
                })
                $("#selectcustomers").change(function() {
                    var idCliente = $("#selectcustomers").val();
                    var url =
"https://northwind.netcore.io/customers/" +
idCliente + ".xml";
                    $.get(url, function(data){
                        var companyName =
                            $(data).find("d2p1
\\:\\CompanyName").first().text();
                        $("#mensaje").text(companyName);
                    })
                })
            })
            $("#botoncargardepartamentos").click(function(){

```

```

        var url =
"https://apicruddepartamentosxml.azurewebsites.net/api/Departamentos
";
        $.get(url, function(data){
            console.log("Leyendo servicio");
            var numeroElementos =
                $(data).find("Departamento").length;
            $("#mensaje").text("Elementos: " +
numeroElementos);
        })
        console.log("Despues servicio");
    })
})
</script>
</body>
</html>

```

## PRACTICA CLIENTES XML

[Swagger UI \(apiclientesxml.azurewebsites.net\)](#)

Debemos dibujar un botón por cada Cliente del documento.  
Los botones mostrarán el nombre del cliente  
Al pulsar el botón del cliente, mostraremos sus datos.

**Clientes XML**

DELORIAN	RICARDO TUBBS	JOHNNY UTAH	PIN
MADMARTIGAN	JESUS QUINTANA	MICHAEL KNIG	
CUCO	ICE MAN	MIKE DONOVAN	BASTIAN BALTA
IAN MALCOLM	HARRY CALLAHAN	ELLEN RIPLEY	
IVAN DRAGO	CARLTON BANKS	BALKI BARTOKOM	
BIFF TANNEN	AXEL FOLEY		

**RICARDO TUBBS**

miamivice@video.es



## REALIZAR CRUD SOBRE SERVICIOS API

Un CRUD es acrónimo sobre Create, Read, Update y Delete en un servicio.

Vamos a llamar al siguiente servicio:

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

Para probar esta funcionalidad, primero vamos a visualizar cómo Podemos realizar peticiones desde el cliente, para ello utilizaremos Cualquier cliente de servicios Api, por ejemplo, Postman o Insomnia

<https://insomnia.rest/download>

Para realizar peticiones GET podemos seguir haciendo como hasta ahora , es decir, con el método `$.get()`

Si deseamos llamar a otros métodos como POST o DELETE, se realiza Mediante `$.ajax()`

Sintaxis:

```

$.ajax({
    url: URL + REQUEST,
    type: GET | POST | PUT | DELETE,
    contentType: TIPO DATOS A ENVIAR,
    dataType: TIPO DATOS A RECIBIR,
    data: LOS DATOS A ENVIAR AL SERVICIO,
    success: function(data) {
        //CONSUMIMOS EL SERVICIO
    }
})

```

**Normas a seguir:** Cuando trabajamos TODOS con servicios a la vez, Nos lo podemos cargar y tener errores que no son nuestros. Para las pruebas, NO TOCAMOS LOS DATOS EXISTENTES.

Cada uno, seleccionamos nuestros número y jugamos con él.

- Tendremos un método para cargar departamentos al inicio.
- Cuando realicemos consultas de acción, también cargaremos los Datos de nuevo para visualizar los cambios.
- Incluiremos funcionalidad para insertar, modificar y eliminar.

Creamos una nueva página llamada `web17cruddepartamentos.html`

Posibles errores de código HTML

**404: Tenemos mal escrita la dirección HTML, ya sea la URL o el request**

Failed to load resource: <apicruddepartamentos.aspxssssartamentos:1> ↗  
the server responded with a status of 404 (Not Found)

**500: Error de base de datos. Estamos repitiendo un número (ID)  
Que ya existe en el servicio.**

① ▶ POST jquery-3.7.1.js:9940 ⓘ ⓘ  
<https://apicruddepartamentosxml.azurewebsites.net/api/departamentos> 500  
(Internal Server Error)

415: Estamos enviando un contentType incorrecto al servicio

① ▶ POST jquery-3.7.1.js:9940 ⓘ ⓘ  
<https://apicruddepartamentosxml.azurewebsites.net/api/departamentos> 415  
(Unsupported Media Type)

400: Tenemos mal formado el documento XML al enviarlo al server

① ▶ POST jquery-3.7.1.js:9940 ⓘ ⓘ  
<https://apicruddepartamentosxml.azurewebsites.net/api/departamentos> 400  
(Bad Request)

# CRUD departamentos XML

Id departamento

Nombre departamento

Localidad

Id departamento	Nombre	Localidad
4	Silence	is gold
10	DISNEYLAND	PARIS
20	PARQUE WARNER	MADRID
30	PORT AVENTURA	TABARNIA
40	TERRA MITICA	BENIDORM

## CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>CRUD departamentos XML</h1>
    <label>Id departamento</label>
    <input type="text" id="cajaid"/><br/>
    <label>Nombre departamento</label>
    <input type="text" id="cajanombre"/><br/>
    <label>Localidad</label>
    <input type="text" id="cajalocalidad"/><br/>
    <button id="botoninsert">Insertar</button>
    <button id="botonupdate">Update</button>
    <button id="botondelete">Delete</button><hr/>
    <table id="tabladepartamentos" border="1">
        <thead>
            <tr>
                <th>Id departamento</th>
                <th>Nombre</th>
                <th>Localidad</th>
            </tr>
        </thead>
        <tbody></tbody>
    </table>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        var url =
https://apicruddepartamentosxml.azurewebsites.net/;
        $(document).ready(function() {
            loadDepartamentos();
            $("#botoninsert").click(function() {
                var id = $("#cajaid").val();
                var nombre = $("#cajanombre").val();
                var localidad = $("#cajalocalidad").val();
                //NECESITAMOS ALMACENAR LOS DATOS XML EN UNA

```

## VARIABLE

```
//COMO STRING Y SIN ESPACIOS
var dataXML = getDepartamentoXML(id, nombre,
localidad);
var request = "api/departamentos";
$.ajax({
    url: url + request,
    type: "POST",
    contentType: "text/xml",

```

```

        data: dataXML,
        success: function() {
            console.log("Insertado");
            loadDepartamentos();
        }
    })
})
$("#botonupdate").click(function() {
    var id = $("#cajaid").val();
    var nombre = $("#cajanombre").val();
    var localidad = $("#cajalocalidad").val();
    var dataXML = getDepartamentoXML(id, nombre,
localidad);
    var request = "api/departamentos";
    $.ajax({
        url: url + request,
        type: "PUT",
        contentType: "text/xml",
        data: dataXML,
        success: function() {
            console.log("Modificando");
            loadDepartamentos();
        }
    })
})
$("#botonDelete").click(function() {
    var id = $("#cajaid").val();
    var request = "api/departamentos/" + id;
    $.ajax({
        url: url + request,
        type: "DELETE",
        success: function() {
            console.log("eliminado");
            loadDepartamentos();
        }
    })
})
function getDepartamentoXML(id, nombre, localidad){
    var dataXML = "<Departamento>";
    dataXML += "<IdDepartamento>" + id +
"</IdDepartamento>";
    dataXML += "<Nombre>" + nombre + "</Nombre>";
    dataXML += "<Localidad>" + localidad +
"</Localidad>";
    dataXML += "</Departamento>";
    return dataXML;
}
function loadDepartamentos(){
    var request = "api/departamentos";
    $.get(url + request, function(data){
        console.log("Leyendo servicio...");
        var html = "";
        $(data).find("Departamento").each(function() {
            var id = $(this).find("IdDepartamento").text();
            var nombre = $(this).find("Nombre").text();
            var localidad =
$(this).find("Localidad").text();
            html += "<tr>";
            html += "<td>" + id + "</td>";
            html += "<td>" + nombre + "</td>";
            html += "<td>" + localidad + "</td>";
            html += "</tr>";
        })
        $("#tabladepartamentos tbody").html(html);
    })
}
</script>
</body>
</html>
```

#### SERVICIO XML API DOCTORES

Quiero realizar el mismo ejemplo, pero utilizando un servicio nuevo.

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

#### CONSUMIR SERVICIOS FORMATO JSON

Un documento JSON es un formato estandar de los servicios API  
Está muy demandado porque lo utilizan el resto del mundo, excepto  
España

El formato estará compuesto con información de objetos y arrays  
Mediante Key:Value

Definición de un objeto

```
{
    "marca": "Mercedes",
    "modelo": "Clase A"
}
```

La representación de múltiples objetos está compuesta por [ .... ]

```
[  
  {  
    "marca": "Mercedes",  
    "modelo": "Clase A"  
  },  
  {  
    "marca": "Renault",  
    "modelo": "Clio"  
  }  
]
```

Una de las diferencias respecto a XML es que no existen búsquedas Nativas, es decir, al no ser etiquetas HTML no podemos aplicar Métodos **find()**

Para consumir documentos JSON tenemos un método llamado **\$.getJSON()**

```
$.getJSON(URL, function(data) {  
  //CONSUMIMOS LOS DATOS  
})
```

También podemos hacerlo con **\$.ajax()**

```
$.ajax({  
  url: url,  
  type: "GET",  
  dataType: "application/json",  
  success: function(data) {  
  
  }  
})
```

Podemos leer cualquier documento de la misma forma, pero Depende si el documento es simple o compuesto devolverá Unos datos u otros.

- 1) Simple: Nos devuelve en el recorrido KEY/VALUE
- 2) Compuesto: Nos devuelve un índice y el objeto recuperado.

Lo bueno es que los objetos que utilizamos ya están dentro del código, Es decir, el propio lenguaje los crea, no tenemos que declarar nada De forma explícita.

No necesitamos, al consumir, crear el objeto.

```
let coche = new Object();  
coche.marca = "....";
```

Para leer un documento JSON se hace con un bucle de tipo **\$.each()**

OBJETO SIMPLE

```
$.each(OBJETO A LEER, function(key, value) {  
  
})
```

OBJETO COMPUESTO

```
$.each(OBJETO A LEER, function(index, object) {  
  
})
```

Comenzamos leyendo en local y luego ya trabajemos con servicios reales.

Creamos una página llamada **web19jsonsimple.html**

## Documento JSON simple

[Cargar tareas simples](#)



- Key: UNO, Value: Sacar a pasear a la tortuga
- Key: DOS, Value: Comprar cereales
- Key: TRES, Value: Jugar al Fifa
- Key: CUATRO, Value: Partida de Poker
- Key: CINCO, Value: Limpiar el coche
- Key: SEIS, Value: Ganar la Champions (Real Madrid)

CODIGO HTML

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">
```

```

<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>Document</title>
</head>
<body>
<h1>Documento JSON simple</h1>
<button id="botonload">Cargar tareas simples</button>
<ul id="tareas"></ul>
<script src="js/jquery-3.7.1.js"></script>
<script>
$(document).ready(function() {
    $("#botonload").click(function() {
        $.getJSON("documents/tareas.json", function(data){
            console.log("Leyendo");
            //DOCUMENTO SIMPLE DEVUELVE KEY/VALUE
            var html = "";
            $.each(data, function(key, value){
                html += "<li>Key: " + key
                + ", Value: " + value + "</li>";
            })
            $("#tareas").html(html);
        })
    })
})
</script>
</body>
</html>

```

## Documento JSON compuesto

Cargar clientes

- Index: 0 - DELORIAN
- Index: 1 - RICARDO TUBBS
- Index: 2 - JOHNNY UTAH
- Index: 3 - ISCO ALARCON
- Index: 4 - OPTIMUS PRIME
- Index: 5 - MICHAEL SCOFIELD
- Index: 6 - CORONEL BRADDOCK
- Index: 7 - MADMARTIGAN
- Index: 8 - JESUS QUINTANA
- Index: 9 - MICHAEL KNIGHT
- Index: 10 - MARTY MACFLY

### CODIGO HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
<h1>Documento JSON compuesto</h1>
<button id="botonload">Cargar clientes</button>
<ul id="listaclientes"></ul>
<script src="js/jquery-3.7.1.js"></script>
<script>
$(document).ready(function() {
    $("#botonload").click(function() {
        $.ajax({
            url: "documents/clientes.json",
            method: "GET",
            dataType: "json",
            success: function(data){
                console.log("Leyendo...");
                var html = "";
                $.each(data.clientes, function(index,
cliente){
                    var nombre = cliente.nombre;
                    html += "<li>Index: " + index
                    + " - " + nombre + "</li>";
                })
                $("#listaclientes").html(html);
            }
        })
    })
})
</script>
</body>
</html>

```

Búsqueda dentro de un documento JSON

# Maestro detalle Jugadores JSON

## Varane

Posición: Defensa central, Edad: 22



### CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Maestro detalle Jugadores JSON</h1>
    <select id="selectjugador"></select><br/>
    <div id="datosjugador"></div>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $.getJSON("documents/jugadores.json", function(data){
                $.each(data.jugadores, function(index, jugador){
                    var nombre = jugador.nombre;
                    var option = $("<option>", {"text": nombre});
                    $("#selectjugador").append(option);
                })
            })
            $("#selectjugador").change(function() {
                var name = $("#selectjugador").val();
                console.log(name);
                $.ajax({
                    url: "documents/jugadores.json",
                    type: "GET",
                    success: function(data){
                        var html = "";
                        $.each(data.jugadores, function(index,
player){
                            var playerName = player.nombre;
                            console.log(playerName);
                            if (playerName == name){
                                var position = player.posicion;
                                var edad = player.edad;
                                var imagen = player.imagen;
                                html += "<h1 style='color:blue'>" +
name + "</h1>";
                                html += "<h3>Posición: " + position
                                + ", Edad: " + edad + "</h3>";
                                html += "<img src='" + imagen + "' "
                                + "style='width: 150px; height:
150px' />";
                            }
                        })
                    })
                    $("#datosjugador").html(html);
                })
            })
        })
    </script>
</body>
</html>
```

### VERSION 2

Modificamos el ejemplo de la siguiente forma:

- No quiero EACH en la búsqueda, quiero directamente el jugador.

```

$.each(data.jugadores, function(index, player){
    var playerName = player.nombre;
    console.log(playerName);
    if (playerName == name){
        var position = player.posicion;
    }
})

```

#### CODIGO HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Maestro detalle Jugadores JSON</h1>
    <select id="selectjugador"></select><br/>
    <div id="datosjugador"></div>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            $.getJSON("documents/jugadores.json", function(data){
                var i = 0;
                $.each(data.jugadores, function(index, jugador){
                    var nombre = jugador.nombre;
                    var option = $("<option>" +
                        , {"text": nombre, "value": i});
                    $("#selectjugador").append(option);
                    i += 1;
                })
            })
            $("#selectjugador").change(function() {
                //EXISTE UNA PROPIEDAD QUE ACCDE AL INDICE DEL
                //ELEMENTO SELECCIONADO DE UN SELECT selectedIndex
                //ES UNA PROPIEDAD DE JAVASCRIPT DEL CONTROL
                //SI NECESITAMOS UNA PROPIEDAD DE JS DESDE UN
                //CODIGO JQUERY, SE UTILIZA LA PALABRA CLAVE prop
                var index =
                    $("#selectjugador").prop("selectedIndex");
                //var index = $("#selectjugador").val();
                $.ajax({
                    url: "documents/jugadores.json",
                    type: "GET",
                    success: function(data){
                        var html = "";
                        var player = data.jugadores[index];
                        var position = player.posicion;
                        var edad = player.edad;
                        var imagen = player.imagen;
                        html += "<h1 style='color:blue'>" +
                            + name + "</h1>";
                        html += "<h3>Posición: " + position +
                            + ", Edad: " + edad + "</h3>";
                        html += "<img src=' " + imagen + " ' " +
                            + "style='width: 150px; height: 150px' />";
                        $("#datosjugador").html(html);
                    }
                })
            })
        })
    </script>
</body>
</html>

```

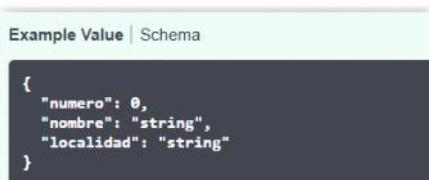
#### CRUD SERVICIO API JSON

Para realizar un CRUD es exactamente igual a el formato XML  
Lo único que cambia es el formato del objeto a enviar, que debe ser  
Exactamente igual al que nos indique el servicio.

Si tenemos swagger, el formato JSON nos dice perfectamente que necesita.

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

Ejemplo de POST



Las peticiones con un cliente Postman o Insomnia también son exactamente iguales.

```

1 = [
2 = {
3 =   "numero": 0,
4 =   "nombre": "string",
5 =   "localidad": "string"
6 = },
7 = {
8 =   "numero": 10,
9 =   "nombre": "DISNEYLAND",
10 =  "localidad": "PARIS"
11 = },
12 = {
13 =   "numero": 20,
14 =   "nombre": "PARQUE WARNER",
15 =   "localidad": "MADRID"
16 = },
17 =

```

Para poder enviar la información con `$.ajax` y con `data` debemos enviar  
Un objeto JSON.  
Dicho objeto debe ser del mismo tipo que estamos recibiendo, incluyendo  
Números y letras.

Existe un método para convertir un objeto de JavaScript a formato JSON y  
Poder ser enviado al servicio.

```

var departamento = new Object();
departamento.numero = 11;
departamento.nombre = "INFORMATICA";

var json = JSON.stringify(departamento);

```

Creamos una nueva página llamada `web22cruddepartamentosjson.html`

## Crud departamentos JSON

Id departamento   
 Nombre   
 Localidad   
Insertar Modificar Eliminar

<b>Id departamento</b>	<b>Nombre</b>	<b>Localidad</b>
10	DISNEYLAND	PARIS
20	PARQUE WARNER	MADRID
30	PORT AVENTURA	TABARNIA
40	TERRA MITICA	BENIDORM

### CODIGO HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Crud departamentos JSON</h1>
  <label>Id departamento</label>
  <input type="text" id="cajanumero"/><br/>
  <label>Nombre</label>
  <input type="text" id="cajanombre"/><br/>
  <label>Localidad</label>
  <input type="text" id="cajalocalidad"/><br/>
  <button id="botoninsert">Insertar</button>
  <button id="botonupdate">Modificar</button>
  <button id="botondelete">Eliminar</button>
  <hr/>
  <table id="tabladedepartamentos" border="1">
    <thead>
      <tr>
        <th>Id departamento</th>
        <th>Nombre</th>
        <th>Localidad</th>
      </tr>
    </thead>
    <tbody></tbody>
  </table>
</body>

```

```

<script src="js/jquery-3.7.1.js"></script>
<script>
    let urlDepartamentos =
    "https://apicruddepartamentoscore.azurewebsites.net/";
    $(document).ready(function() {
        loadDepartamentos();
        $("#botoninsert").click(function() {
            var numero = parseInt($("#cajanumero").val());
            var nombre = $("#cajanombre").val();
            var localidad = $("#cajalocalidad").val();
            //NECESITAMOS CREAR UN OBJETO QUE CONTENGA
            //EL MISMO NOMBRE DE PROPIEDADES DEL OBJETO JSON
            //QUE RECIBIRA EL SERVICIO API
            var departamento = new Object();
            departamento.numero = numero;
            departamento.nombre = nombre;
            departamento.localidad = localidad;
            //CONVERTIMOS EL OBJETO DE JS A FORMATO JSON
            var dataJSON = JSON.stringify(departamento);
            console.log(dataJSON);
            var request = "api/departamentos";
            $.ajax({
                url: urlDepartamentos + request,
                type: "POST",
                contentType: "application/json",
                data: dataJSON,
                success: function() {
                    console.log("Departamento insertado");
                    loadDepartamentos();
                }
            })
        })
        $("#botonupdate").click(function() {
            var numero = parseInt($("#cajanumero").val());
            var nombre = $("#cajanombre").val();
            var localidad = $("#cajalocalidad").val();
            var departamento = new Object();
            departamento.numero = numero;
            departamento.nombre = nombre;
            departamento.localidad = localidad;
            var dataJSON = JSON.stringify(departamento);
            var request = "api/departamentos";
            $.ajax({
                url: urlDepartamentos + request,
                type: "PUT",
                contentType: "application/json",
                data: dataJSON,
                success: function(){
                    console.log("Updated");
                    loadDepartamentos();
                }
            })
        })
        $("#botondelete").click(function() {
            var id = $("#cajanumero").val();
            var request = "api/departamentos/" + id;
            $.ajax({
                url: urlDepartamentos + request,
                type: "DELETE",
                success: function(){
                    console.log("Delete");
                    loadDepartamentos();
                }
            })
        })
    })
    function loadDepartamentos(){
        var request = "api/departamentos";
        $.getJSON(urlDepartamentos + request, function(data){
            console.log("Leyendo");
            var html = "";
            $.each(data, function(index, departamento){
                html += "<tr>";
                html += "<td>" + departamento.numero + "</td>";
                html += "<td>" + departamento.nombre + "</td>";
                html += "<td>" + departamento.localidad +
                "</td>";
                html += "</tr>";
            })
            $("#tabladedepartamentos tbody").html(html);
        })
    }
</script>
</body>
</html>

```

#### MAESTRO DETALLE DEPARTAMENTOS EMPLEADOS

Debemos realizar la siguiente aplicación.  
Al cargar la página, dibujaremos dinámicamente los datos de los departamentos  
Al pulsar sobre **Empleados**, mostraremos los empleados del departamento en una Tabla.

# Maestro detalle empleados departamento

- TERRA MITICA [Empleados](#)
- PORT AVENTURA [Empleados](#)
- PARQUE WARNER [Empleados](#)
- DISNEYLAND [Empleados](#)

Apellido	Oficio	Salario	Departamento
rey	PRESIDENTE	11392	10
cerezo	DIRECTOR	318500	10
muñoz	EMPLEADO	169250	10

Departamentos:

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

Empleados

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

CODIGO HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Maestro detalle Empleados departamentos</h1>
    <ul id="capadedepartamentos"></ul>
    <table id="tablaempleados" border="1">
        <thead>
            <tr>
                <th>Apellido</th>
                <th>Oficio</th>
                <th>Salario</th>
                <th>Departamento</th>
            </tr>
        </thead>
        <tbody></tbody>
    </table>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        $(document).ready(function() {
            var urlDepartamentos =
                "https://apicruddepartamentoscore.azurewebsites.net/";
            var urlEmpleados =
                "https://apiempleadossppg.azurewebsites.net/";
            var requestDepartamentos = "api/departamentos";
            $.getJSON(urlDepartamentos + requestDepartamentos,
function(data){
            $.each(data, function(index, departamento){
                var li = $("<li>", {"text": departamento.nombre});
                var link = $("<a>", {
                    "text": " Empleados",
                    "href": "#",
                    "data-iddepartamento": departamento.numero
                });
                li.append(link);
                $("#capadedepartamentos").append(li);
                link.click(function() {
                    var idDepartamento =
                        $(this).data("iddepartamento");
                    console.log(idDepartamento);
                    var requestEmpleados =
                        "api/empleados/empleadosdepartamento/"
                        + idDepartamento;
                    $.ajax({
                        url: urlEmpleados + requestEmpleados,
                        type: "GET",
                        success: function(data){
                            var html = "";
                            $.each(data, function(index,
                                empleado){
                                html += "<tr>";
                                html += "<td>" + empleado.apellido +
                                    "</td>";
                                html += "<td>" + empleado.oficio +
                                    "</td>";
                                html += "<td>" + empleado.salario +
                                    "</td>";
                                html += "<td>" + empleado.departamento +
                                    "</td>";
                                html += "</tr>";
                            })
                            $("#tablaempleados
tbody").html(html);
                        }
                    })
                })
            })
        })
    
```

```

        })
    </script>
</body>
</html>

```

#### SIGUIENTE EJEMPLO

Al iniciar la página, rellenamos un <select> con los diferentes **oficios** de los empleados. Al seleccionar un oficio, debemos mostrar los datos de los empleados con dicho oficio.

Vamos a incrementar el salario de los Empleados por Oficio

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

The screenshot shows a web application titled "Empleados Oficios". At the top left, there is a dropdown menu labeled "DIRECTOR" with a downward arrow icon. To its right, the text "Incremento salarial:" is followed by a text input field containing the value "1". Below this, a button labeled "Incrementar salarios" is visible. The main content area displays a table with three columns: "Apellido", "Oficio", and "Salario". The data in the table is:

Apellido	Oficio	Salario
negro	DIRECTOR	43764
jimenez	DIRECTOR	408769
cerezo	DIRECTOR	340383
serra	DIRECTOR	416884

#### CODIGO HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Incremento empleados oficios</h1>
    <select id="selectooficios">
    </select>
    <label>Incremento salarial:</label>
    <input type="text" id="cajaincremento"/>
    <button id="botonincrementar">Incrementar salarios</button>
    <table border="1" id="tablaempleados">
        <thead>
            <tr>
                <th>Apellido</th>
                <th>Oficio</th>
                <th>Salario</th>
            </tr>
        </thead>
        <tbody></tbody>
    </table>
    <script src="js/jquery-3.7.1.js"></script>
    <script>
        let urlEmpleados =
        "https://apiempleadosaction.azurewebsites.net/";
        $(document).ready(function(){
            var requestOficios = "api/empleados/getoficios/oficios";
            $.getJSON(urlEmpleados + requestOficios, function(data){
                $.each(data, function(index, oficio){
                    var option = $(<option>
                        , {"text": oficio, "value": oficio});
                    $("#selectooficios").append(option);
                })
            })
            $("#selectooficios").change(function() {
                var oficio = $(this).val();
                //PINTAR LA TABLA
                cargarEmpleadosOficio(oficio);
            })
            $("#botonincrementar").click(function() {
                var incremento = $("#cajaincremento").val();
                var oficio = $("#selectooficios").val();
                var requestUpdate =
                "api/Empleados/IncrementarSalarioOficios/"
                + oficio + "/" + incremento;
                $.ajax({
                    url: urlEmpleados + requestUpdate,
                    type: "PUT",
                    success: function() {
                        console.log("Update");
                        cargarEmpleadosOficio(oficio);
                    }
                })
            })
        })
        function cargarEmpleadosOficio(oficio){

```

```

        var requestEmpleados =
            "api/Empleados/GetEmpleadosOficio/empleadosoficio/"
+ oficio;
        $.ajax({
            url: urlEmpleados + requestEmpleados,
            type: "GET",
            success: function(data){
                var html = "";
                $.each(data, function(index, empleado){
                    html += "<tr>";
                    html += "<td>" + empleado.apellido +
                "

```

#### FRAMEWORK BOOTSTRAP

Es un Framework para nosotros, es decir, creado para los desarrolladores que nos permite Poder incluir estilos de forma sencilla sin pensar mucho.

Lo podemos incluir dentro de cualquier Framework front, como VUE, ANGULAR...

Para incluir bootstrap dentro de nuestro proyecto es casi igual en todas tecnologías.

- 1) Descargar la librería en local e incluir los links para los css y js
- 2) Incluir las carpetas necesarias para el diseño (css y js)

<https://getbootstrap.com/docs/5.3/getting-started/download/>

Nombre	Fecha de modificación	Tipo	Tamaño
bootstrap.bundle.js	09/10/2024 9:09	JSFile	203 KB
bootstrap.bundle.js.map	09/10/2024 9:09	Linker Address Map	435 KB
bootstrap.bundle.min.js	09/10/2024 9:09	JSFile	79 KB
bootstrap.bundle.min.js.map	09/10/2024 9:09	Linker Address Map	325 KB
bootstrap.esm.js	09/10/2024 9:09	JSFile	133 KB
bootstrap.esm.js.map	09/10/2024 9:09	Linker Address Map	299 KB
bootstrap.esm.min.js	09/10/2024 9:09	JSFile	73 KB
bootstrap.esm.min.js.map	09/10/2024 9:09	Linker Address Map	218 KB
bootstrap.js	09/10/2024 9:09	JSFile	142 KB
bootstrap.js.map	09/10/2024 9:09	Linker Address Map	300 KB

Nombre	Fecha de modificación	Tipo	Tamaño
bootstrap.css	09/10/2024 9:09	CSSfile	275 KB
bootstrap.css.map	09/10/2024 9:09	Linker Address Map	664 KB
bootstrap.min.css	09/10/2024 9:09	CSSfile	228 KB
bootstrap.min.css.map	09/10/2024 9:09	Linker Address Map	577 KB

- 1) Buscar una plantilla de Bootstrap y copiar su código y cambiar las referencias A los links y styles de css y js.

Bootstrap funciona con estilos ya predefinidos de CSS

Cada elemento podemos ponerlo con un estilo, dependiendo de su etiqueta HTML

Recuperamos esta plantilla

<https://getbootstrap.com/docs/5.3/examples/navbar-fixed/>

Todas las plantillas son exactamente iguales.

Algunas plantillas contendrán más cositas como, por ejemplo, si una plantilla contiene Un slider, a lo mejor dicho slider está hecho con un Plug in de Jquery y debemos incluir Dicha librería.

- Incluir CSS y JS de Bootstrap
- Incluir las librerías propias de estilos que tengamos en la plantilla
- Los plugin extras que hubiera

#### PRACTICA FINAL JQUERY

Vamos a realizar un CRUD aplicando estilos bootstrap con múltiples páginas.

La página HOME mostrará una tabla con los departamentos.

Al pulsar sobre un Departamento, podremos ir a otra página para visualizar sus detalles

¿Qué necesitamos en la otra página de detalles para trabajar? El ID del Departamento

La información entre páginas Web es un estandar:

<https://www.miservidor.com/pagina1.html?param1=valor1>

Múltiples datos

<https://www.miservidor.com/pagina1.html?param1=valor1&param2=valor2&param3=valor3>

En este ejemplo anterior tenemos una página que recibe tres parámetros

¿Cómo podemos recuperar dichos parámetros mediante JQUERY? Google

```
var params = new window.URLSearchParams(window.location.search);
var data = params.get('NOMBRE PARAMETRO')
```

Este es el servicio CRUD que vamos a utilizar

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

## Servicio CRUD Hospitales

- /webresources/hospitales
- /webresources/hospitales/{id}
- /webresources/hospitales/post
- /webresources/hospitales/put
- /webresources/hospitales/delete/{id}

- 1) Página principal con todos los hospitales en una tabla.
- 2) Página para insertar hospitales
- 3) Al pulsar sobre un determinado hospital (Links) podremos navegar a otra página  
Para editar los datos de un determinado hospital
- 4) Al pulsar sobre eliminar un hospital, pedimos confirmación y eliminamos.
- 5) Utilizar Bootstrap

Todas las páginas serán 26

ID hospital	Nombre	Dirección	Teléfono	Camas	Link
321	mama	melhuevo	923-54110	324	<a href="#">Editar</a> <a href="#">Delete</a>
123	La paz	Rivas Vaciamadrid	23543545	45	<a href="#">Editar</a> <a href="#">Delete</a>
18	General	Atocha s/n	595-3111	987	<a href="#">Editar</a> <a href="#">Delete</a>
22	La Paz	Castellana 1000	923-5411	412	<a href="#">Editar</a> <a href="#">Delete</a>
45	San Carlos	Ciudad Universitaria	597-1500	845	<a href="#">Editar</a> <a href="#">Delete</a>

Comenzamos creando una nueva página llamada **web26indexhospitales.html**

### INDEX.HTML

```
<!doctype html>
<html lang="en" data-bs-theme="auto">
  <head>
    <script src="js/color-modes.js"></script>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
    <meta name="generator" content="Hugo 0.122.0">
    <title>Fixed top navbar example · Bootstrap v5.3</title>
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <meta name="theme-color" content="#712cf9">
    <!-- Custom styles for this template -->
    <link href="css/customstyles.css" rel="stylesheet">
    <link href="css/navbar-fixed.css" rel="stylesheet">
```

```

</head>
<body>
  <svg xmlns="http://www.w3.org/2000/svg" class="d-none">
    <symbol id="check2" viewBox="0 0 16 16">
      <path d="M13.854 3.646a.5.5 0 1 0 .7081-7 7a.5.5 0 0
1-.708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.29316.646-6.647a.5.5 0
0 1 .708 0z"/>
    </symbol>
    <symbol id="circle-half" viewBox="0 0 16 16">
      <path d="M8 15A7 7 0 1 0 8 1v14zm0 1A8 8 0 1 1 8 0a8 8 0 0 1
0 16z"/>
    </symbol>
    <symbol id="moon-stars-fill" viewBox="0 0 16 16">
      <path d="M6 .278a.768.768 0 0 1 .08.858 7.208 7.208 0 0
0-.878 3.46c 0 .4021 3.278 7.277 7.318 7.277.527 0 1.04-.055
1.533-.16a.787.787 0 0 1 .81.316.733.733 0 0 1-.031.893a.849 8.349
0 0 1 8.344 16C3.734 16 0 12.286 0 7.71 0 4.266 2.114 1.312
5.124.86a.752.752 0 0 1 6 .278z"/>
      <path d="M10.794 3.148a.217.217 0 0 1 .412 0l.387
1.162c.173.518.579.924 1.097 1.097l1.162.387a.217.217 0 0 1
0 .412l-1.162.387a1.734 1.734 0 0 1.097 1.097l-.387 1.162a.217.217
0 0 1-.412 0l-.387-1.162a1.734 1.734 0 0 0 9.31
6.593l-1.162-.387a.217.217 0 0 1 0-.412l1.162-.387a1.734 1.734 0 0 0
1.097-1.097l.387-1.162zM13.863.099a.145.145 0 0 1 .274
01.258.774c.115.346.386.617.732.732l.774.258a.145.145 0 0 1
0 .274l-.774.258a1.156 1.156 0 0 0 -.732.732l-.258.774a.145.145 0 0
1-.274 0l-.258-.774a1.156 1.156 0 0 0 -.732-.732l-.774-.258a.145.145
0 0 1 0-.274l.774-.258c.346-.115.617-.386.732-.732L13.863.1z"/>
    </symbol>
    <symbol id="sun-fill" viewBox="0 0 16 16">
      <path d="M8 12a4 4 0 1 0 0-8 4 4 0 0 0 0 8zM8 0a.5.5 0 0
1 .5v2a.5.5 0 0 1-1 0v-2A.5.5 0 0 1 8 0zm0 13a.5.5 0 0
1 .5v2a.5.5 0 0 1-1 0v-2A.5.5 0 0 1 8 13zm8-.5.5 0 0
1-.5.5h-2a.5.5 0 0 1 0-1h2a.5.5 0 0 1 3 8zm10.657-5.657a.5.5 0 0 1
0 .707l-1.414 1.414a.5.5 0 1 1-.707-.708l1.414-1.414a.5.5 0 0 1 .707
0zm-9.193 9.193a.5.5 0 0 1 0 .707L3.05 13.657a.5.5 0 0
1-.707-.707l1.414-1.414a.5.5 0 0 1 .707 0zm9.193 2.121a.5.5 0 0
1-.707 0l-1.414-1.414a.5.5 0 0 1 .707-.707l1.414 1.414a.5.5 0 0 1
0 .707zM4.464 4.465a.5.5 0 0 1-.707 0L2.343 3.05a.5.5 0 1
0 .707-.707l1.414 1.414a.5.5 0 0 1 0 .708z"/>
    </symbol>
  </svg>
  <div class="dropdown position-fixed bottom-0 end-0 mb-3 me-3 bd-mode-toggle">
    <button class="btn btn-bd-primary py-2 dropdown-toggle d-flex align-items-center"
      id="bd-theme"
      type="button"
      aria-expanded="false"
      data-bs-toggle="dropdown"
      aria-label="Toggle theme (auto)">
      <span class="bi my-1 theme-icon-active" width="1em"
        height="1em"><use href="#circle-half"></use></span>
      <span class="visually-hidden" id="bd-theme-text">Toggle
theme</span>
    </button>
    <ul class="dropdown-menu dropdown-menu-end shadow" aria-
labelledby="bd-theme-text">
      <li>
        <button type="button" class="dropdown-item d-flex align-
items-center" data-bs-theme-value="light" aria-pressed="false">
          <svg class="bi me-2 opacity-50" width="1em"
            height="1em"><use href="#sun-fill"></use></svg>
          Light
          <svg class="bi ms-auto d-none" width="1em" height="1em">
<use href="#check2"></use></svg>
        </button>
      </li>
      <li>
        <button type="button" class="dropdown-item d-flex align-
items-center" data-bs-theme-value="dark" aria-pressed="false">
          <svg class="bi me-2 opacity-50" width="1em"
            height="1em"><use href="#moon-stars-fill"></use></svg>
          Dark
          <svg class="bi ms-auto d-none" width="1em" height="1em">
<use href="#check2"></use></svg>
        </button>
      </li>
      <li>
        <button type="button" class="dropdown-item d-flex align-
items-center active" data-bs-theme-value="auto" aria-pressed="true">
          <svg class="bi me-2 opacity-50" width="1em"
            height="1em"><use href="#circle-half"></use></svg>
          Auto
          <svg class="bi ms-auto d-none" width="1em" height="1em">
<use href="#check2"></use></svg>
        </button>
      </li>
    </ul>
  </div>
  <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-
dark">
    <div class="container-fluid">
      <a class="navbar-brand" href="#">Hospitales</a>
      <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarCollapse" aria-
controls="navbarCollapse" aria-expanded="false" aria-label="Toggle
navigation">

```

```

        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
        <ul class="navbar-nav me-auto mb-2 mb-md-0">
            <li class="nav-item">
                <a class="nav-link active" aria-current="page"
                   href="web26indexhospitales.html">Home</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="web26createhospital.html">
New hospital</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link disabled" aria-disabled="true">
Disabled</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
<main class="container">
    <div class="bg-body-tertiary p-5 rounded">
        <h1>Hospitales CRUD</h1>
        <table id="tablahospitales" class="table table-hover">
            <thead>
                <tr>
                    <th>Id hospital</th>
                    <th>Nombre</th>
                    <th>Dirección</th>
                    <th>Teléfono</th>
                    <th>Camas</th>
                    <th>Link</th>
                </tr>
            </thead>
            <tbody></tbody>
        </table>
    </div>
</main>
<script src="js/jquery-3.7.1.js"></script>
<script src="js/bootstrap.bundle.js"></script>
<script>
    let urlHospital =
    "https://apicrudhospital.azurewebsites.net/";
    $(document).ready(function() {
        var request = "/webresources/hospitales";
        $.ajax({
            url: urlHospital + request,
            type: "GET",
            contentType: "application/json",
            success: function(data){
                var html = "";
                $.each(data, function(index, hospital){
                    html += "<tr>";
                    html += "<td>" + hospital.idhospital +
                "</td>";
                    html += "<td>" + hospital.nombre + "</td>";
                    html += "<td>" + hospital.direccion +
                "</td>";
                    html += "<td>" + hospital.telefono +
                "</td>";
                    html += "<td>" + hospital.camas + "</td>";
                    html += "<td>" ;
                    html += "<a href='web26edithospital.html?
idhospital=";
                    html += hospital.idhospital + "'>Editar</a>";
                    html += "<a href='#" +
                onclick='deleteHospital(";
                    html += hospital.idhospital + ")>Delete</a>";
                    html += "</td>";
                    html += "</tr>";
                })
                $("#tablahospitales tbody").html(html);
            }
        })
    })
    function deleteHospital(idHospital){
        var requestDelete = "webresources/hospitales/delete/" +
        idHospital;
        $.ajax({
            url: urlHospital + requestDelete,
            type: "DELETE",
            success: function(){
                console.log("Deleted");
            }
        })
    }
</script>
</body>
</html>

```

A continuación, creamos una nueva página llamada **web26createhospital.html**

## Create hospital

Id hospital

Nombre

Dirección

Teléfono

Camas

**Insertar hospital**

### CODIGO HTML

```
<!doctype html>
<html lang="en" data-bs-theme="auto">
  <head>
    <script src="js/color-modes.js"></script>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and
Bootstrap contributors">
    <meta name="generator" content="Hugo 0.122.0">
    <title>Fixed top navbar example · Bootstrap v5.3</title>
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <meta name="theme-color" content="#712cf9">
    <!-- Custom styles for this template -->
    <link href="css/customstyles.css" rel="stylesheet">
    <link href="css/navbar-fixed.css" rel="stylesheet">
  </head>
  <body>
    <svg xmlns="http://www.w3.org/2000/svg" class="d-none">
      <symbol id="check2" viewBox="0 0 16 16">
        <path d="M13.854 3.646a.5.5 0 1 0 .708l-7 7a.5.5 0 0
1 .708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.29316.646-6.647a.5.5 0
0 1 .708 0z"/>
      </symbol>
      <symbol id="circle-half" viewBox="0 0 16 16">
        <path d="M8 15A7 7 0 1 0 8 1v14zm0 1A8 8 0 1 1 8 0a8 8 0 0 1
0 16z"/>
      </symbol>
      <symbol id="moon-stars-fill" viewBox="0 0 16 16">
        <path d="M6 .278a.768.768 0 0 1 .08.858 7.208 7.208 0 0
0-.878 3.46c0 4.021 3.278 7.277 7.318 7.277.527 0 1.04-.055
1.533-.16a.787.787 0 0 1 .81.316.733.733 0 0 1 -.031.893a8.349 8.349
0 0 1 8.344 16C3.734 16 0 12.286 0 7.71 0 4.266 2.114 1.312
5.124.06A.752.752 0 0 1 6 .278z"/>
        <path d="M10.794 3.148a.217.217 0 0 1 .412 0l.387
1.162c.173.518.579.924 1.097 1.0971.162.387a.217.217 0 0 1
0 .412l-1.162.387a1.734 1.734 0 0 0-1.097 1.097l-.387 1.162a.217.217
0 0 1-.412 0l-.387-1.162a1.734 1.734 0 0 0 9.31
6.593l-1.162-.387a.217.217 0 0 1 -.412l1.162-.387a1.734 1.734 0 0 0
1.097-1.097l.387-1.162m13.863.099a.145.145 0 0 1 .274
01.258.774c.115.346.386.617.732.732l.774.258a.145.145 0 0 1
.024.274 0l-.258-.774a1.156 1.156 0 0 0-.732.732l-.258.774a.145.145 0 0 1
1-.274 0l-.258-.774a1.156 1.156 0 0 0-.732-.732l-.774-.258a.145.145 0 0 1
0 0 1 0-.274l.774-.258c.346-.115.617-.386.732-.732L13.863.1z"/>
      </symbol>
      <symbol id="sun-fill" viewBox="0 0 16 16">
        <path d="M8 12a4 4 0 1 0 -8 4 4 0 0 0 8zM0 0a.5.5 0 0 1
1 .5v2a.5.5 0 0 1-1 0v-2A.5.5 0 0 1 8 0zm0 13a.5.5 0 0 1
1 .5v2a.5.5 0 0 1-1 0v-2A.5.5 0 0 1 8 13zm8-.5a.5.5 0 0 1
1-.5h2a.5.5 0 0 1 1 .5zm8 0a.5.5 0 0 1 1 .5h2a.5.5 0 0 1
1 .5zm-9.193 9.193a.5.5 0 0 1 0 .707l3.05 13.657a.5.5 0 0 1
0 .707l-1.414 1.414a.5.5 0 1 1-.707-.707l1.414-1.414a.5.5 0 0 1
.707l-1.414 1.414a.5.5 0 0 1 .707-.707l1.414 1.414a.5.5 0 0 1
0 .707M4.464 4.465a.5.5 0 0 1-.707 0L2.343 3.05a.5.5 0 1
1 .707-.707l1.414 1.414a.5.5 0 0 1 0 .708z"/>
      </symbol>
    </svg>
    <div class="dropdown position-fixed bottom-0 end-0 mb-3 me-3 bd-
mode-toggle">
```

```

        <button class="btn btn-bd-primary py-2 dropdown-toggle d-flex align-items-center"
            id="bd-theme"
            type="button"
            aria-expanded="false"
            data-bs-toggle="dropdown"
            aria-label="Toggle theme (auto)">
            <svg class="bi my-1 theme-icon-active" width="1em" height="1em"><use href="#circle-half"></use></svg>
            <span class="visually-hidden" id="bd-theme-text">Toggle theme</span>
        </button>
        <ul class="dropdown-menu dropdown-menu-end shadow" aria-labelledby="bd-theme-text">
            <li>
                <button type="button" class="dropdown-item d-flex align-items-center" data-bs-theme-value="light" aria-pressed="false">
                    <svg class="bi me-2 opacity-50" width="1em" height="1em"><use href="#sun-fill"></use></svg>
                    Light
                    <svg class="bi ms-auto d-none" width="1em" height="1em"><use href="#check2"></use></svg>
                </button>
            </li>
            <li>
                <button type="button" class="dropdown-item d-flex align-items-center" data-bs-theme-value="dark" aria-pressed="false">
                    <svg class="bi me-2 opacity-50" width="1em" height="1em"><use href="#moon-stars-fill"></use></svg>
                    Dark
                    <svg class="bi ms-auto d-none" width="1em" height="1em"><use href="#check2"></use></svg>
                </button>
            </li>
            <li>
                <button type="button" class="dropdown-item d-flex align-items-center active" data-bs-theme-value="auto" aria-pressed="true">
                    <svg class="bi me-2 opacity-50" width="1em" height="1em"><use href="#circle-half"></use></svg>
                    Auto
                    <svg class="bi ms-auto d-none" width="1em" height="1em"><use href="#check2"></use></svg>
                </button>
            </li>
        </ul>
    </div>

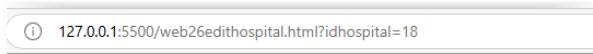
    <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Hospitales</a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarCollapse">
                <ul class="navbar-nav me-auto mb-2 mb-md-0">
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page" href="web26indexhospitales.html">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="web26createhospital.html">
New hospital</a>
                        </li>
                    <li class="nav-item">
                        <a class="nav-link disabled" aria-disabled="true">
Disabled</a>
                        </li>
                </ul>
            </div>
        </div>
    </nav>
<main class="container">
    <div class="bg-body-tertiary p-5 rounded">
        <h1>Create hospital</h1>
        <label>Id hospital</label>
        <input type="text" id="cajaidhospital" class="form-control"/>
        <label>Nombre</label>
        <input type="text" id="cajanombre" class="form-control"/>
        <label>Dirección</label>
        <input type="text" id="cajadireccion" class="form-control"/>
        <label>Teléfono</label>
        <input type="text" id="cajatelefono" class="form-control"/>
        <label>Camas</label>
        <input type="text" id="cajacamas" class="form-control"/><br/>
        <button id="botoninsert" class="btn btn-danger">
            Insertar hospital
        </button>
    </div>
</main>
<script src="js/jquery-3.7.1.js"></script>
<script src="js/bootstrap.bundle.js"></script>
<script>
    let urlHospitales =
    "https://apicrudhospital.azurewebsites.net/";
    $(document).ready(function() {
        $("#botoninsert").click(function() {

```

```
        var id = parseInt($("#cajaidhospital").val());
        var nombre = $("#cajanombre").val();
        var direccion = $("#cajadireccion").val();
        var tlf = $("#cajatelefono").val();
        var camas = parseInt($("#cajacamas").val());
        var hospital = new Object();
        hospital.idhospital = id;
        hospital.nombre = nombre;
        hospital.direccion = direccion;
        hospital.telefono = tlf;
        hospital.camas = camas;
        var hospitalJSON = JSON.stringify(hospital);
        var request = "webratesources/hospitales/post";
        $.ajax({
            url: urlHospitales + request,
            type: "POST",
            contentType: "application/json",
            data: hospitalJSON,
            success: function() {
                console.log("insertado!!!!");
                //CUANDO INSERTEMOS, NOS VAMOS A INDEX
                window.location.href =
                    "web26indexhospitales.html";
            }
        })
    })
}
</script>
</body>
</html>
```

El siguiente paso a realizar es crear una página para modificar un determinado hospital.

Llamamos a la página **[web26edithospital.html](#)**



## **EDIT.HTML**

```
<!doctype html>
<html lang="en" data-bs-theme="auto">
  <head>
    <script src="js/color-modes.js"></script>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
    <meta name="generator" content="Hugo 0.122.0">
    <title>Fixed top navbar example · Bootstrap v5.3</title>
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <meta name="theme-color" content="#712cf9">
    <!-- Custom styles for this template -->
    <link href="css/customstyles.css" rel="stylesheet">
    <link href="css/navbar-fixed.css" rel="stylesheet">
  </head>
  <body>
    <svg xmlns="http://www.w3.org/2000/svg" class="d-none">
      <symbol id="check2" viewBox="0 0 16 16">
        <path d="M13.854 3.646a.5.5 0 0 1 .708l-7 7a.5.5 0 0 1 -.708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.29316.646-6.647a.5.5 0 0 1 .708 0z"/>
      </symbol>
      <symbol id="circle-half" viewBox="0 0 16 16">
        <path d="M8 15A7 7 0 1 0 8 1v14zM0 1A8 8 0 1 1 8 0a8 8 0 0 1 0 16z"/>
      </symbol>
      <symbol id="moon-stars-fill" viewBox="0 0 16 16">
        <path d="M6 .278a.768.768 0 0 1 .08858 7.208 7.208 0 0 0 0-.878 3.46C 4.021 3.278 7.277 7.318 7.277.527 0 1.04-0.55 1.533-.16a.787.787 0 0 1 .81316.733.733 0 0 1 -.031.893A8.349 8.349 0 0 1 8.344 16C3.734 16 0 12.286 0 7.71 0 4.266 2.114 1.312 5.124.06A.752.752 0 0 1 6 .278z"/>
        <path d="M10.794 3.148a.217.217 0 0 1 .412 0l.387 1.162c.173.518.579.924 1.097 1.097l1.162.387a.217.217 0 0 1 0 .412l-1.162.387a1.734 1.734 0 0 0 -1.097 1.097-1.387 1.162a.217.217 0 0 1 0 -.414 0l-.387-1.162A1.734 1.734 0 0 0 0.31 6.5931-1.162-.387a.217.217 0 0 1 0 0 -.412l1.162-.387a1.734 1.734 0 0 0 1.097-1.097l.387-1.162M13.863.099a.145.145 0 0 1 .274 0.258.774c.115.346.386.617.732.732l.774.258a.145.145 0 0 1 0 .274l-.774.258a1.156 1.156 0 0 0 -.732-.732l-.774-.258a.145.145 0 0 1 -.274 0.258-.774a1.156 1.156 0 0 0 0-.732-.732l-.774-.258a.145.145 0 0 1 0 -.274l.774-.258c.346-.115.617-.386.732-.732L13.863.12z"/>
      </symbol>
      <symbol id="sun-fill" viewBox="0 0 16 16">
        <path d="M8 12a4 4 0 1 0 0-8 4 4 0 0 0 0 8zM8 0a.5.5 0 0 1 .5.5v2a.5.5 0 0 1 0-2A.5.5 0 0 1 8 0zm0 13a.5.5 0 0 1 1.5.5v2a.5.5 0 0 1 0-2A.5.5 0 0 1 8 13zm8-5a.5.5 0 0 1 -1.5.5h-2a.5.5 0 0 1 0-1h2a.5.5 0 0 1 1.5.5zm8 0a.5.5 0 0 1 1.5.5h-2a.5.5 0 0 1 0-1h2a.5.5 0 0 1 1.5.5 0 0 1 0 .7071-1.414 1.414a.5.5 0 1 1 .707-.708l1.414-1.414a.5.5 0 0 1 .707 0zm9.193 2.121a.5.5 0 0 1 -1.707-.7071l1.414-1.414a.5.5 0 0 1 .707 0zm9.193 2.121a.5.5 0 0 1 -1.707 0l1.414-1.414a.5.5 0 0 1 .707-.7071l1.414 1.414a.5.5 0 0 1 0 .707zm4.464 4.465a.5.5 0 0 1 -1.707 0L2.343 3.05a.5.5 0 0 1 0 .707-1.7071l1.414 1.414a.5.5 0 0 1 0 .708z"/>
    </symbol>
  </svg>

```

```

        </svg>
    <div class="dropdown position-fixed bottom-0 end-0 mb-3 me-3 bd-mode-toggle">
        <button class="btn btn-bd-primary py-2 dropdown-toggle d-flex align-items-center">
            id="bd-theme"
            type="button"
            aria-expanded="false"
            data-bs-toggle="dropdown"
            aria-label="Toggle theme (auto)">
            <svg class="bi my-1 theme-icon-active" width="1em" height="1em"><use href="#circle-half" href="#"></use></svg>
            <span class="visually-hidden" id="bd-theme-text">Toggle theme</span>
        </button>
        <ul class="dropdown-menu dropdown-menu-end shadow" aria-labelledby="bd-theme-text">
            <li>
                <button type="button" class="dropdown-item d-flex align-items-center" data-bs-theme-value="light" aria-pressed="false">
                    <svg class="bi me-2 opacity-50" width="1em" height="1em"><use href="#sun-fill" href="#"></use></svg>
                    Light
                    <svg class="bi ms-auto d-none" width="1em" height="1em">
                    <use href="#check2" href="#"></use></svg>
                </button>
            </li>
            <li>
                <button type="button" class="dropdown-item d-flex align-items-center" data-bs-theme-value="dark" aria-pressed="false">
                    <svg class="bi me-2 opacity-50" width="1em" height="1em"><use href="#moon-stars-fill" href="#"></use></svg>
                    Dark
                    <svg class="bi ms-auto d-none" width="1em" height="1em">
                    <use href="#check2" href="#"></use></svg>
                </button>
            </li>
            <li>
                <button type="button" class="dropdown-item d-flex align-items-center active" data-bs-theme-value="auto" aria-pressed="true">
                    <svg class="bi me-2 opacity-50" width="1em" height="1em"><use href="#circle-half" href="#"></use></svg>
                    Auto
                    <svg class="bi ms-auto d-none" width="1em" height="1em">
                    <use href="#check2" href="#"></use></svg>
                </button>
            </li>
        </ul>
    </div>

    <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Hospitales

```

```

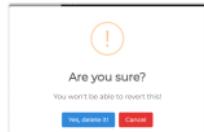
"https://apicrudhospital.azurewebsites.net/";
$(document).ready(function() {
    var params =
        new window.URLSearchParams(window.location.search);
    var idHospitalURL = params.get('idhospital');
    var requestFind = "webresources/hospitales/" +
idHospitalURL;
    //BUSCAMOS UN HOSPITAL Y LO DIBUJAMOS EN LAS CAJAS
    $.ajax({
        url: urlHospitales + requestFind,
        type: "GET",
        success: function(data){
            $("#cajaidhospital").val(data.idhospital);
            $("#cajanombre").val(data.nombre);
            $("#cajadireccion").val(data.direccion);
            $("#cajatelefono").val(data.telefono);
            $("#cajacamas").val(data.camas);
        }
    })
    $("#botonupdate").click(function() {
        var requestPut = "webresources/hospitales/put";
        var idhospital =
parseInt($("#cajaidhospital").val());
        var nombre = $("#cajanombre").val();
        var direccion = $("#cajadireccion").val();
        var telefono = $("#cajatelefono").val();
        var camas = parseInt($("#cajacamas").val());
        var hospital = new Object();
        hospital.idhospital = idhospital;
        hospital.nombre = nombre;
        hospital.direccion = direccion;
        hospital.telefono = telefono;
        hospital.camas = camas;
        var dataJSON = JSON.stringify(hospital);
        $.ajax({
            url: urlHospitales + requestPut,
            type: "PUT",
            contentType: "application/json",
            data: dataJSON,
            success: function(){
                console.log("Update");
                window.location.href =
"web26indexhospitales.html";
            }
        })
    })
})
</script>
</body>
</html>

```

#### Investigación:

- Al eliminar un hospital, recargaremos los datos de los hospitales
- Necesito que, al pulsar sobre eliminar, nos muestre un mensaje de confirmación.  
Si pulsamos SI, eliminaremos el Hospital (window.confirm())
- Una vez que tengamos esto, debemos utilizar la librería SWEETALERT para Mostrar un mensaje "bonito"

<https://sweetalert2.github.io/>



#### VERSION 2

```

<!doctype html>
<html lang="en" data-bs-theme="auto">
<head>
    <script src="js/color-modes.js"></script>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <meta name="description" content="">
    <meta name="author" content="Mark Otto, Jacob Thornton, and
Bootstrap contributors">
    <meta name="generator" content="Hugo 0.122.0">
    <title>Fixed top navbar example · Bootstrap v5.3</title>
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <meta name="theme-color" content="#712cf9">
    <!-- Custom styles for this template -->
    <link href="css/customstyles.css" rel="stylesheet">
    <link href="css/navbar-fixed.css" rel="stylesheet">
    <link href="css/sweetalert2.min.css" rel="stylesheet">
</head>
<body>
    <svg xmlns="http://www.w3.org/2000/svg" class="d-none">
        <symbol id="check2" viewBox="0 0 16 16">
            <path d="M13.854 3.646a.5.5 0 0 1 .708l-7 7a.5.5 0 0
1 -.708 0l-3.5-.5a.5.5 0 0 1 .708-.708L6.5 10.29316.646-6.647a.5.5 0
0 1 .708 0z"/>
        </symbol>
        <symbol id="circle-half" viewBox="0 0 16 16">
            <path d="M8 15A7 7 0 1 0 8 1v14zm0 1A8 8 0 1 1 8 0a8 8 0 0 1
0 16z"/>
        </symbol>
    </svg>

```

```

        <symbol id="moon-stars-fill" viewBox="0 0 16 16">
            <path d="M6 .278a.768.768 0 0 1 .08.858 7.208 7.208 0 0
0-.878 3.46c0 4.021 3.278 7.277 7.318 7.277.527 0 1.04-.055
1.533-.16a.787.787 0 0 1 .81.316.733.733 0 0 1 -.031.893A8.349 8.349
0 0 1 8.344 16C3.734 16 0 12.286 0 7.71 0 4.266 2.114 1.312
5.124.06A.752.752 0 0 1 6 .278z"/>
            <path d="M10.794 3.148a.217.217 0 0 1 .412 0l.387
1.162c.173.518.579.924 1.097 1.0971.162.387a.217.217 0 0 1
0 .4121-1.162.387a1.734 1.734 0 0-1.097 1.0971-.387 1.162a.217.217
0 0 1-.412 0l-.387-1.162A1.734 1.734 0 0 0 9.31
6.5931-1.162-.387a.217.217 0 0 1 0-.4121.162-.387a1.734 1.734 0 0 0
1.097-1.0971.387-1.162M13.863.099a.145.145 0 0 1 .274
01.258.774c.115.346.386.617.732.7321.774.258a.145.145 0 0 1
0 .2741-.774.258a1.156 1.156 0 0 0 -.732.7321-.258.774a.145.145 0 0
1-.274 0l-.258-.774a1.156 1.156 0 0 0 -.732-.7321-.774-.258a.145.145
0 0 1 0-.2741.774-.258c.346-.115.617-.386.732-.732L13.863.1z"/>
        </symbol>
        <symbol id="sun-fill" viewBox="0 0 16 16">
            <path d="M8 12a4 4 0 1 0 -8 4 0 0 0 0 8zM8 0a.5.5 0 0
1 .5.5v2a.5.5 0 0 1-1 0v-2A.5.5 0 0 1 8 0zm0 13a.5.5 0 0
1 .5.5v2a.5.5 0 0 1-1 0v-2A.5.5 0 0 1 8 13zm8-5a.5.5 0 0
1-.5.5h-2a.5.5 0 0 1 0-1h2a.5.5 0 0 1 .5.5zM 8a.5.5 0 0
1-.5.5h-2a.5.5 0 0 1 0-1h2a.5.5 0 0 1 8zm10.657-5.657a.5.5 0 0 1
0 .7071-1.414 1.415a.5.5 0 1 1-.707-.70811.414-1.414a.5.5 0 0 1 .707
0zm-9.193 9.193a.5.5 0 0 1 0 .707L3.05 13.657a.5.5 0 0
1-.707-.70711.414-1.414a.5.5 0 0 1 .707 0zm9.193 2.121a.5.5 0 0
1-.707 0l-1.414-1.414a.5.5 0 0 1 .707-.70711.414 1.414a.5.5 0 0 1
0 .707zM4.464 4.465a.5.5 0 0 1-.707 0L2.343 3.05a.5.5 0 1
1 .707-.70711.414 1.414a.5.5 0 0 1 0 .708z"/>
        </symbol>
    </svg>
    <div class="dropdown position-fixed bottom-0 end-0 mb-3 me-3 bd-mode-toggle">
        <button class="btn btn-bd-primary py-2 dropdown-toggle d-flex align-items-center"
               id="bd-theme"
               type="button"
               aria-expanded="false"
               data-bs-toggle="dropdown"
               aria-label="Toggle theme (auto)">
            <svg class="bi my-1 theme-icon-active" width="1em"
height="1em"><use href="#circle-half"></use></svg>
            <span class="visually-hidden" id="bd-theme-text">Toggle
theme</span>
        </button>
        <ul class="dropdown-menu dropdown-menu-end shadow" aria-
labelledby="bd-theme-text">
            <li>
                <button type="button" class="dropdown-item d-flex align-
items-center" data-bs-theme-value="light" aria-pressed="false">
                    <svg class="bi me-2 opacity-50" width="1em"
height="1em"><use href="#sun-fill"></use></svg>
                    Light
                    <svg class="bi ms-auto d-none" width="1em" height="1em">
<use href="#check2"></use></svg>
                </button>
            </li>
            <li>
                <button type="button" class="dropdown-item d-flex align-
items-center" data-bs-theme-value="dark" aria-pressed="false">
                    <svg class="bi me-2 opacity-50" width="1em"
height="1em"><use href="#moon-stars-fill"></use></svg>
                    Dark
                    <svg class="bi ms-auto d-none" width="1em" height="1em">
<use href="#check2"></use></svg>
                </button>
            </li>
            <li>
                <button type="button" class="dropdown-item d-flex align-
items-center active" data-bs-theme-value="auto" aria-pressed="true">
                    <svg class="bi me-2 opacity-50" width="1em"
height="1em"><use href="#circle-half"></use></svg>
                    Auto
                    <svg class="bi ms-auto d-none" width="1em" height="1em">
<use href="#check2"></use></svg>
                </button>
            </li>
        </ul>
    </div>

    <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-
dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Hospitales</a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarCollapse" aria-
controls="navbarCollapse" aria-expanded="false" aria-label="Toggle
navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarCollapse">
                <ul class="navbar-nav me-auto mb-2 mb-md-0">
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page"
                           href="web26indexhospitales.html">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="web26createhospital.html">
New hospital</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

```

```

        <li class="nav-item">
            <a class="nav-link disabled" aria-disabled="true">
Disabled</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
<main class="container">
<div class="bg-body-tertiary p-5 rounded">
    <h1>Hospitales CRUD</h1>
    <table id="tablahospitales" class="table table-hover">
        <thead>
            <tr>
                <th>Id hospital</th>
                <th>Nombre</th>
                <th>Dirección</th>
                <th>Teléfono</th>
                <th>Camas</th>
                <th>Link</th>
            </tr>
        </thead>
        <tbody></tbody>
    </table>
</div>
</main>
<script src="js/jquery-3.7.1.js"></script>
<script src="js/bootstrap.bundle.js"></script>
<script src="js/sweetalert2.all.min.js"></script>
<script>
    let urlHospital =
"https://apicrudhospital.azurewebsites.net/";
    $(document).ready(function() {
        loadHospitales();
    })
    function loadHospitales(){
        var request = "/webresources/hospitales";
        $.ajax({
            url: urlHospital + request,
            type: "GET",
            contentType: "application/json",
            success: function(data){
                var html = "";
                $.each(data, function(index, hospital){
                    html += "<tr>";
                    html += "<td>" + hospital.idhospital +
"</td>";
                    html += "<td>" + hospital.nombre + "</td>";
                    html += "<td>" + hospital.direccion +
"</td>";
                    html += "<td>" + hospital.telefono +
"</td>";
                    html += "<td>" + hospital.camas + "</td>";
                    html += "<td> ";
                    html +=
"<a href='web26edithospital.html?
idhospital=" + hospital.idhospital + "' class='btn btn-
info'>Editar</a>";
                    html += "<button href='#'
onclick='deleteHospital(" + hospital.idhospital + ")' class='btn btn-
danger'>Delete</button>";
                    html += "</td>";
                    html += "</tr>";
                })
                $("#tablahospitales tbody").html(html);
            }
        })
    }
    function deleteHospital(idHospital){
        Swal.fire({
            title: "Are you sure?",
            text: "You won't be able to revert this!",
            icon: "warning",
            showCancelButton: true,
            confirmButtonColor: "#3085d6",
            cancelButtonColor: "#d33",
            confirmButtonText: "Yes, delete it!"
        }).then((result) => {
            if (result.isConfirmed) {
                var requestDelete = "webresources/hospitales/delete/" +
idHospital;
                $.ajax({
                    url: urlHospital + requestDelete,
                    type: "DELETE",
                    success: function(){
                        console.log("Deleted");
                        loadHospitales();
                    }
                })
            }
        });
    }
</script>
</body>
</html>

```



# REACT

miércoles, 9 de octubre de 2024 13:14

**Nota:** Cuando utilicemos Frameworks SPA NO debemos sincronizar con OneDrive.

## INSTALACION

Los elementos de FRONT que vamos a ver utilizan **NODE.JS** para desplegar sus Características y funcionalidades.

Utilizaremos la consola de comandos para desplegar las aplicaciones.

Todo trabaja bajo servidor.

Podemos utilizar la consola de VS Code para hacer los despliegues.

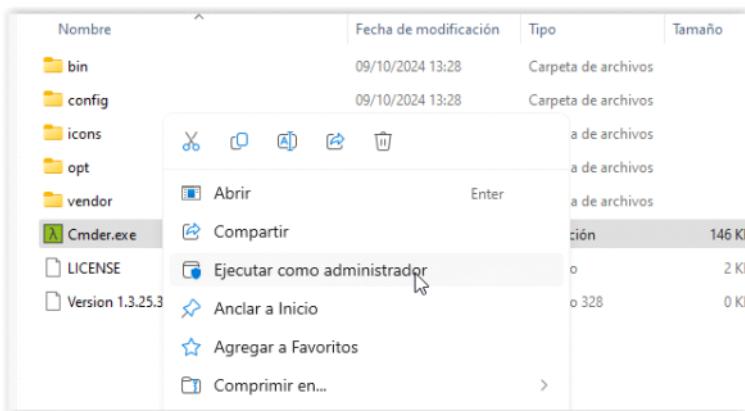
Vamos a instalar el siguiente programa:

- 1) Descargamos el programa.

<https://cmder.app/>

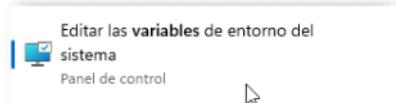
- 2) Sobre Archivos de Programa, creamos una nueva carpeta llamada **Cmder** y copiamos Ahí los ficheros descomprimidos.

- 3) Ejecutamos el fichero **cmd.exe** como administrador



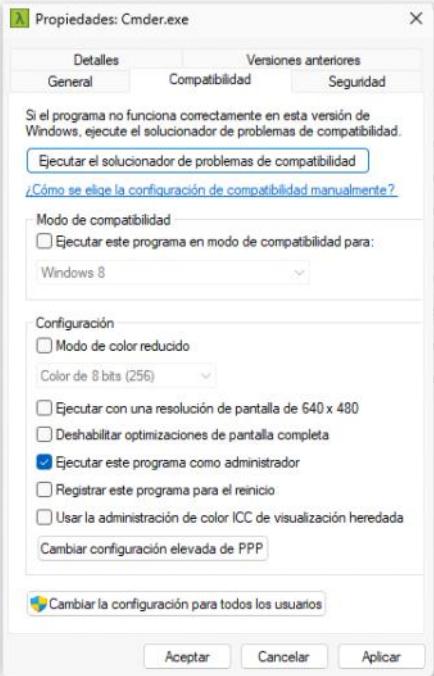
- 4) Pulsamos sobre **Unblock and Continue**

- 5) Añadimos Variables de entorno la ruta del Path de nuestro programa.
- 6) Abrimos Sistema de Windows, Opciones Avanzadas y Variables de entorno



- 7) Copiamos la ruta de nuestra carpeta dentro de Variables de entorno del sistema en **path**

- 8) Configuramos CMDER para que se ejecute siempre como Administrador. Botón derecho sobre **cmd.exe** y Propiedades.



Instalamos NODE.

Run JavaScript Everywhere

Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.

Download Node.js (LTS)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Para comprobar si lo tenemos correctamente instalado, debemos escribir la siguiente instrucción desde la línea de comandos: `node --version`

```
C:\...\Master Desarrollo 2024-2025...\  
λ node --version  
v20.18.0
```

Para las instalaciones se utiliza Node Package Manager que simplemente es un comando que contiene `node.js` y dónde podemos descargar librerías centralizadas para nuestros frameworks. Dichas librerías pueden ser descargadas para un proyecto:

```
$ npm install sweetalert2
```

Pueden ser descargadas para utilización global de las librerías, por ejemplo, otro repositorio de librerías. Vamos a instalar YARN que es un repositorio global para FRONT, React, Angular, Vue

```
npm install --global yarn
```

**REACT** es un Framework FRONT de tipo Single Page Application desarrollado por Facebook.  
Tiene una empresa detrás y las actualizaciones las marca dicha empresa.

Todos son iguales, pero cada uno contendrá "alguna" característica distinta.  
Depende del Framework se utilizará un lenguaje u otro.

Una App de tipo SPA lo que realiza, de forma Visual es tener una "única" página y,  
En el interior de dicha página mostrar diferentes VISTAS.

A dichas vistas, en este tipo de Apps se le llaman **Components**

Para trabajar con este tipo de proyectos se realiza mediante  
Línea de comandos, tanto para generar el proyecto como para  
Arrancar el Server.

También podemos crear el proyecto de CERO e incluir cosas, pero eso  
Nos lleva 10 días.

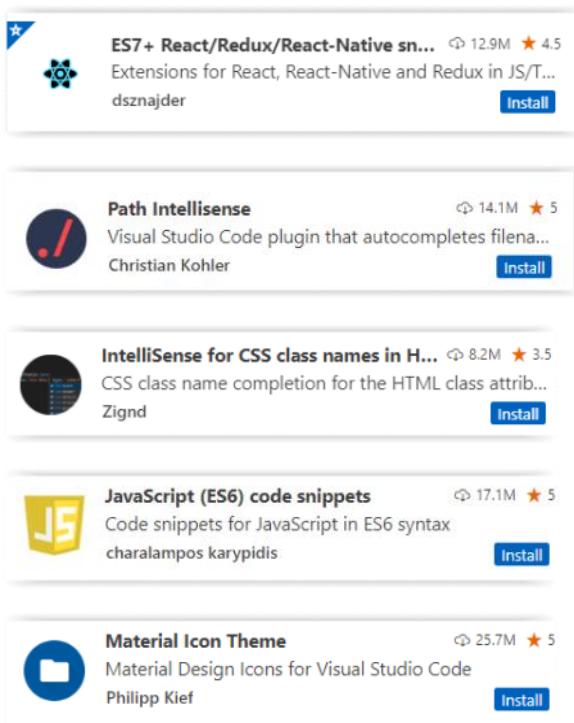
Tendremos una carpeta PRINCIPAL y dentro las librerías.

Las librerías NO se sincronizan con GitHub.

React utiliza un lenguaje llamado JSX que es una mezcla entre HTML y JS

Trabajaremos con Visual Studio. Dentro de Visual tenemos extensiones  
Que nos "ayudan" con diferentes lenguajes.

Vamos a instalar la siguiente extensión dentro de VS Code.



Forma de trabajar:

- Tendremos múltiples carpetas/proyectos
- Cada proyecto será una nueva carpeta.
- Para centralizarlo, yo voy a crear una carpeta llamada **react**

**REACT** es **case sensitive** en el nombre de las carpetas, así que TODO lo  
Que escribamos en minúsculas.

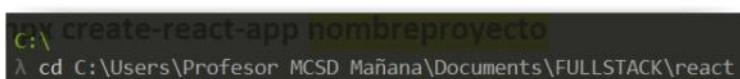
Vamos a crear la carpeta **react** en Documents/FULLSTACK

Para crear proyectos react necesitamos hacerlo mediante línea de comandos.

```
npx create-react-app nombreproyecto
```

Lo haremos desde la línea de comandos y la carpeta react. Abrimos nuestra línea de comandos,  
Entramos dentro de react.

```
cd path
```



Creamos un nuevo proyecto llamado **primerreact**

```
C:\Users\Profesor MCSD Mañana\Documents\FULLSTACK\react
λ npx create-react-app primerreact
Need to install the following packages:
create-react-app@5.0.1
Ok to proceed? (y) y
```

Una vez creado, entraremos dentro de la carpeta del proyecto

`cd primerreact`

Ejecutaremos VS Code desde esa ubicación para trabajar con el siguiente comando

`code .`

```
C:\Users\Profesor MCSD Mañana\Documents\FULLSTACK\react
λ cd primerreact\
code .
C:\Users\Profesor MCSD Mañana\Documents\FULLSTACK\react\primerreact (master)
λ code .
```

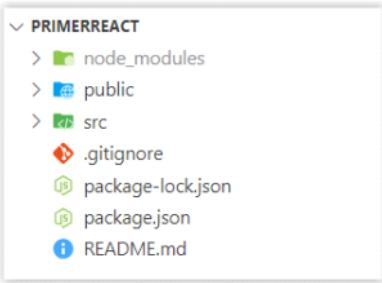
Para ejecutar el proyecto se utiliza la siguiente instrucción que arrancará el server

`npm start`

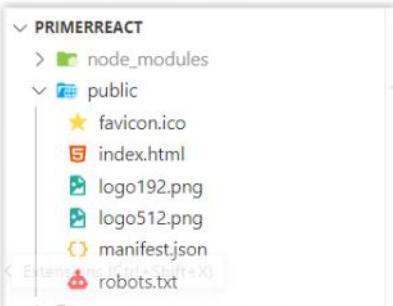
```
You can now view primerreact in the browser.

Local:          http://localhost:3000
On Your Network:  http://10.202.1.26:3000
```

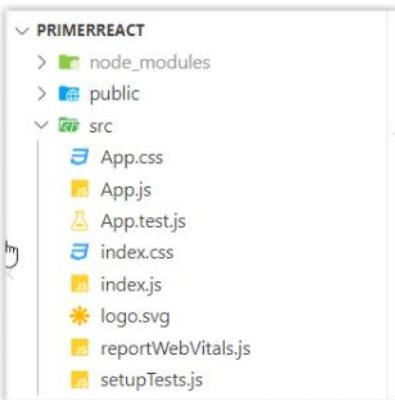
Al abrir el proyecto, tendremos el siguiente sistema de carpetas



- **node\_modules:** Esta carpeta no la tocaremos, forma parte de las librerías que utiliza React para trabajar. Esta carpeta NO se sincroniza con GitHub mediante el fichero **.gitignore**
- **package.json:** Este fichero indica todas las librerías instaladas en nuestro proyecto react. En realidad es un indicador de dependencias.
- **public:** Es la raíz de nuestro proyecto y ahí es donde se encuentra la página inicial Del componente principal.



- **src:** Es nuestra carpeta para desarrollar, dónde más trabajaremos. En esta carpeta irán Los componentes y la lógica de nuestra aplicación. Cada componente es un fichero \*.js



Si nos fijamos, dentro de **index.html** tenemos un `<div id="root">`

Dentro de dicho div es dónde irán incluidos los componentes.

```
const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Dentro de **index.js** está buscando un TAG llamado **root** y dibujando un component llamado **App**  
`<App/>`

Dicho component contiene el dibujo principal de nuestra página

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="Edit <code>src/App.js</code> and save to <a href="#" className="App-link" >App</a>">
      </header>
    </div>
  );
}

export default App;
```

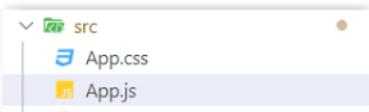
Cada Component tiene un nombre que se establece por **function**

```
function App() {
  return (
```

Para poder utilizar un component, debemos realizar un **import**

```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from '
```

Podemos tener todos los components que deseemos en nuestro proyecto.  
Cada Component, puede tener un CSS opcional.



En el momento de trabajar, podemos tener todos los components por ahí sueltos o ser Organizados.

En el momento de trabajar, todos los Componentes deben estar en una misma carpeta Llamada **components**

Una vez creada la carpeta, tenemos dos posibilidades:

- 1) El component no utiliza recursos CSS. Creamos el component individual sin carpeta
- 2) El component utiliza un recurso CSS, crearemos una subcarpeta con el JS y el CSS

Un Component contiene una mezcla de código HTML y JS dentro de un mismo fichero.  
Un Component SIEMPRE devolverá código HTML como respuesta para su dibujo. **return**

```
function MiComponent() {
    return (<h1>Mi primer componente</h1>)
}

export default MiComponent;
```

Una vez creado el component, posteriormente, para utilizarlo en cualquier otro lugar/component

```
import MiComponent from 'RUTA JS'
```

Dibujamos el component

```
<MiComponent/>
```

Sobre **src** creamos una nueva carpeta llamada **components**

Dentro de **components** creamos una nueva clase JS llamada **Saludo.js**

```
function Saludo(){
    return (<h1>React en Juernes</h1>)
}

export default Saludo;
```

Este component ya podemos utilizarlo en cualquier lugar.

Vamos a probar a dibujarlo dentro de **index.js**

- 1) Importamos nuestro Component dentro de **index.js**

```
import Saludo from './components/Saludo';
```

- 2) Dibujamos nuestro Component con el mismo NAME después del import

```
const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
<React.StrictMode>
    <Saludo />
</React.StrictMode>
);
```

Solamente podemos devolver una etiqueta HTML dentro de los **return** de cada component

Si deseamos devolver más de una etiqueta, tendremos que utilizar algún contenedor estilo **<div>** o etiquetas propias de **React**

```

function Saludo(){
    return (
        <div>
            <h1>React en Juernes</h1>
            <h2>Otro mensaje más por aquí</h2>
        </div>
    )
}

```

## PROPS EN COMPONENTS

Un **Prop** es un atributo/propiedad de los componentes de React.

Es exactamente igual que los atributos HTML.

```

```

Los props nos permiten poder recibir/enviar información dentro de un componente

Podemos incluir tantos props como deseemos.

```
<Saludo nombre="Alumno"/>
```

Dentro de un Componente si utilizamos props, debemos indicarlo en la definición de **FUNCTION**

La variable que recibe dicha función se llamará **props**

Dentro de **props** tendremos la propiedad/es que nos hayan enviado desde un componente Parent

```
function MiComponent(props){
    props.Propiedad
}
```

Esta funcionalidad nos permite tener componentes dinámicos

Podemos utilizar variables dentro de la sintaxis.

Podemos declarar variables exactamente como en JS con **var** o con **let** dentro de la función, pero FUERA del **return (HTML)**

Para poder utilizar variables dentro **return** se denominan mediante **{miVariable}**

Vamos a modificar el código de **Saludo.js** y recibiremos un **nombre** para que el dibujo sea Dinámico.

```

function Saludo(props) {
    //ESTO ES CODIGO JS, PODEMOS DECLARAR MULTIPLES VARIABLES
    var dato = "Mañana es viernes....";
    var nombre = props.nombre;
    return (
        <div>
            <h1>React en Juernes</h1>
            <h2>Dato objetivo: {dato}</h2>
            <h1>Su nombre es {nombre}</h1>
        </div>
    )
}

export default Saludo;

```

Modificamos el código de **index.js** para enviar múltiples componentes **Saludo**

## INDEX.JS

```

root.render(
  <React.StrictMode>
    <Saludo nombre="Lucia" edad="19"/>
    <Saludo nombre="Adrian" edad="25"/>
  </React.StrictMode>
);

```

Y podremos visualizar los resultados y comprobar que los props son opcionales tanto  
Al enviarlos como al recibirlas.

## React en Juernes

**Dato objetivo: Mañana es viernes....**

**Su nombre es Lucia**

## React en Juernes

**Dato objetivo: Mañana es viernes....**

**Su nombre es Adrian**

Podemos capturar las variables de forma individual:

```

function Saludo(props) {
  //ESTO ES CODIGO JS, PODEMOS DECLARAR MULTIPLES VARIABLES
  var dato = "Mañana es viernes....";
  var nombre = props.nombre;
  var edad = props.edad;
  return (
    <div>
      <h1>React en Juernes</h1>
      <h2>Dato objetivo: {dato}</h2>
      <h1>Su nombre es {nombre} y su edad es {edad}</h1>
    </div>
  )
}

```

Tenemos otro tipo de sintaxis para capturar múltiples variables de props a la vez en una sola línea. Dicha sintaxis se llama deconstrucción.

```

const { variable1, variable2, variable3 } = props;

function Saludo(props) {
  //ESTO ES CODIGO JS, PODEMOS DECLARAR MULTIPLES VARIABLES
  var dato = "Mañana es viernes....";
  //<Saludo nombre="Adrian" edad="25"/>
  const { nombre, edad } = props;

```

El nombre de las variables declaradas deben ser las mismas variables que recibimos en props.

### LLAMADAS A METODOS EN COMPONENTES Y SU COMUNICACION

Cuando hablamos de métodos dentro de React ya NO ESTAMOS HABLANDO DE **function**

Si necesitamos declarar métodos para ser llamados de alguna forma, se utiliza una sintaxis

Parecida a **const**  
Utilizan sintaxis **Lambda**

Dichos métodos deben ir declarados dentro de **function** y fuera de **return**

```
const miMetodo = () => {
    //ACCIONES DE NUESTRO METODO
}
```

Para la llamada a nuestro método

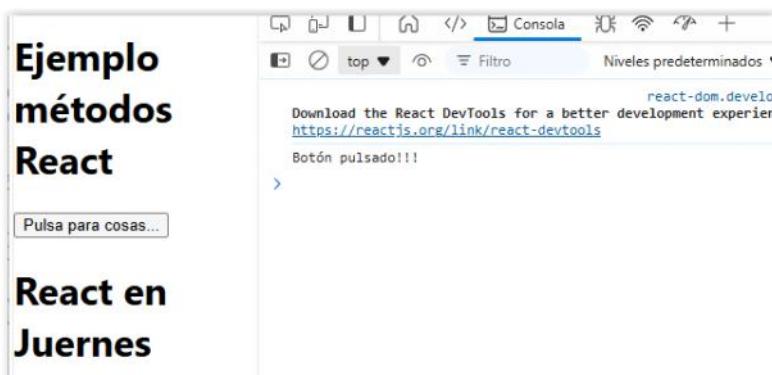
```
<button onClick={() => miMetodo()}>Pulsar para algo</button>
```

Creamos un nuevo componente llamado **Metodos.js**

#### METODOS.JS

```
function Metodos(){
    //NOS DECLARAMOS UN METODO PARA
    //MOSTRAR UN MENSAJE POR CONSOLA
    const mostrarMensaje = () => {
        console.log("Botón pulsado!!!");
    }
    return (<div>
        <h1>Ejemplo métodos React</h1>
        <button onClick={() => mostrarMensaje()}>
            Pulta para cosas...
        </button>
    </div>)
}
export default Metodos;
```

Comprobamos la funcionalidad dentro de **index.js**



También podemos **LLAMAR** a nuestro método dentro del **RENDER** del componente.  
Un **Render** es la ejecución/dibujo de la carga de un componente.

Para poder llamar al método dentro del **render** lo realizamos directamente dentro del **return**

```
const mostrarMensaje = () => {
    console.log("Botón pulsado!!!");
}

return (<div>
    <h1>Ejemplo métodos React</h1>
    {mostrarMensaje()}
```

Por supuesto, podemos recibir parámetros dentro de nuestros métodos

```
const metodoAccion = (param1, param2) => {
    //ACCIONES
}
```

Llamada al método

```
<button onClick={() => metodoAccion('VALOR1', 111)}>Boton con parámetros</button>
```

Vamos a realizar un nuevo componente en el que mostraremos el doble de un número al Pulsar un botón.



Sobre **components** creamos un nuevo componente llamado **DobleNumero.js**

#### DOBLENUMERO.JS

```
function DobleNumero(){
  const numeroDoble = (numero) => {
    var doble = numero * 2;
    console.log(doble);
  }
  return (<div>
    <h1>Ejemplo métodos parámetros</h1>
    <button onClick={() => numeroDoble(7)}>Doble 7</button>
    <button onClick={() => numeroDoble(199)}>Doble
199</button>
  </div>
)
export default DobleNumero;
```

Las variables **NUNCA** son modificadas en el dibujo, es decir, los cambios nunca entran dentro del **RENDER**

```
var ejemplo = "Soy una variable de jueves!!!";

const cambiarVariable = () => {
  console.log("Valor Antes: " + ejemplo);
  ejemplo = "He cambiado de VALOR!!!!!!";
  console.log("Valor Despues: " + ejemplo)
}
```

```
return (<div>
  <h1>Ejemplo métodos parámetros</h1>
  <h2>{ejemplo}</h2>
  <button onClick={() => cambiarVariable()}>
    Cambiar valor Ejemplo
  </button>
)
```

Como podemos comprobar el valor de la variable SI QUE ES MODIFICADO, pero el **DIBUJO NO ES MODIFICADO**

Dentro de React, los dibujos solamente se realizan una vez al iniciar el **Render**.

Aunque el dibujo sea actualizado, el dibujo no cambia porque no se vuelve a realizar el render

Cuando cambiamos de valor las variables.

Para poder realizar cambios dinámicos en los dibujos necesitamos saber los **Estados de la página**



#### ESTILOS CSS DENTRO DE REACT

Un CSS no deja de ser una hoja de estilos. Dentro de React no puedo utilizar los CSS de forma "normal".

Tenemos tres formas de llamar a CSS dentro de React

- 1) Un CSS externo que debemos importar dentro del Component

```
import './App.css';
```

Este CSS no podemos utilizarlo dentro del component con la palabra **class**, en su lugar Utiliza la palabra **className**

```
return <div className="App">
```

- 2) Almacenar los valores de los estilos dentro de una variable JS como objeto JSON

```
var estilo = {  
  color: "blue",  
  backgroundcolor: "yellow"  
}
```

Para utilizar este tipo de variable **INLINE** en las etiquetas, debemos hacerlo mediante **style** y **llaves**

```
<h1 style={estilo}>Ejemplo estilo JS</h1>
```

- 3) Utilizar estilos directamente **INLINE** en la propia etiqueta HTML, pero los estilos Definidos irán con **DOBLE LLAVE**

```
<h2 style={{color:"fuchsia"}}>Estilo de etiquetas</h2>
```

## PRACTICA REACT

Creamos un nuevo proyecto, dentro de la carpeta **react**, llamado **ejemplosreact**

Debemos incluir TODOS los componentes dentro de una carpeta llamada **components** (**App**) y el resto de los componentes con los que trabajemos.

Crearemos una carpeta llamada **src/assets/images** e incluimos una imagen en su interior. Quiero dibujar dicha imagen LOCAL en un component que crearemos a continuación

Necesito un component llamado **SumarNumeros** que recibirá dos números en un método desde Dos botones diferentes en su click y vemos la funcionalidad dentro de **console**. **Lo mismo que DOBLENUMERO, pero enviando dos números al método que crearemos.**

Por último, crearemos un CSS externo llamado **SumarNumeros.css** y aplicamos cualquier Estilo dentro de **SumarNumeros.js**

SUBIR EL PROYECTO A GITHUB

The screenshot shows a React application with the title "Sumar números Component". In the center is a large image of a LEGO Batman minifigure. Below the image are two buttons: "Sumar 5 + 6" and "Sumar 7 + 7". To the right of the application window is a browser's developer tools console. The console has a header with tabs like "Console", "Network", "Elements", etc., and a status bar with "react refresh:6". It displays the message "Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools". Below that, it shows two log entries: "Suma 11" and "Suma 14", both from "SumarNumeros.js:7".

SUMARNUMEROS.JS

```

import batman from './assets/images/batman.jpg';
import './SumarNumeros.css';
function SumarNumeros() {
  const realizarSuma = (num1, num2) => {
    var suma = num1 + num2;
    console.log("Suma " + suma);
  }
  return (<div>
    <h1>Sumar números Component</h1>
    <button onClick={() => realizarSuma(5,6)}>
      Sumar 5 + 6
    </button>
    <button onClick={() => realizarSuma(7,7)}>
      Sumar 7 + 7
    </button>
    <img src={batman}/>
  </div>)
}
export default SumarNumeros;

```

#### SUMARNUMEROS.CSS

```

h1 {
  color:red
}
img {
  width: 150px;
  height: 150px;
}

```

URL de nuestro proyecto:

[serraguti/ejemplosreact2024 \(github.com\)](https://serraguti/ejemplosreact2024)

El siguiente paso que vamos a realizar será:

- 1) Detener el server
- 2) Salimos de línea de comandos
- 3) Cerramos VS Code
- 4) Si lo tenemos en GIT, borramos la carpeta **ejemplosreact** de Windows

Una vez eliminado el proyecto, vamos a descargarlo de Github (en VS Code) y lo volveremos a poner a punto para utilizarlo.

Una vez descargado nuestro proyecto, no podemos ejecutarlo directamente, ya que NO tenemos las dependencias (librerías) de nuestro proyecto.

Deberemos aplicar el comando **npm install** para que instale las librerías antes de **npm start**

#### COMUNICACIÓN ENTRE COMPONENTES

La comunicación entre componentes es algo esencial dentro de cualquier Framework SPA.

Esta comunicación nos permite poder trabajar entre componentes y hacer elementos dinámicos, por Ejemplo, utilizando **props**

La comunicación con **props** es una comunicación **Padre --> Hijo**

```

function Hijo(props) {
  var valor1 = props.propiedad1;
  var valor2 = props.propiedad2;
}

```

#### PADRE

<Hijo propiedad1="Hola" propiedad2="Mundo"/>

Actualmente tenemos un componente llamado **SumarNumeros.js** que suma dos números Que hemos puesto estáticos desde el propio component.

Vamos a modificar el componente para recibir los números desde props.

Entramos dentro del Parent que es **index.js**

```

<React.StrictMode>
  <SumarNumeros numero1="5" numero2="6"/>
  <SumarNumeros numero1="77" numero2="88"/>
</React.StrictMode>

```

Entramos dentro de **SumarNumeros.js (HIJO)** y modificamos el código para recuperar los datos Mediante **props**

#### SUMARNUMEROS.JS

```

function SumarNumeros(props) {
  const realizarSuma = (num1, num2) => {
    //var suma = num1 + num2;
    var suma =
      parseInt(props.numero1) + parseInt(props.numero2);
    console.log("Suma " + suma);
  }
}

```

La comunicación entre Padre Hijo es super simple.  
La funcionalidad la realiza el Hijo.

#### COMUNICACIÓN HIJO A PADRE

La comunicación desde el Hijo al Padre se realiza mediante métodos.  
El padre tendrá un método y el hijo tendrá también otro método.  
El método del hijo llamará al método del padre.

Necesitamos varios elementos

#### PARENT COMPONENT

```
const metodoPadre = () => {  
}
```

Debemos enviar el método dentro de la etiqueta de creación del Hijo

```
<Hijo metodoPadre={ metodoPadre }/>
```

Una vez que tenemos esto configurado en el padre, debemos tener un método en el Hijo que llamará al metodoPadre (VERDE) y ejecutará el ROJO.

No es método cualquiera, es un método que viene en **props** y que debe ejecutarse utilizando Otro tipo de sintaxis.

Para empezar, debemos recibir el método en **props**

#### HIJO COMPONENT

```
var metodoHijo = props.metodoPadre;
```

Cuando lo necesitemos, ejecutamos el método.

```
<button onClick={ () => metodoHijo() }>Ejecutar padre</button>
```

Vamos a realizar un ejemplo sencillo donde tendremos dos componentes:

**SaludoPadre.js**  
**SaludoHijo.js**

#### SALUDOHIJO.JS

```

function SaludoHijo(props) {
  //CAPTURAMOS EN UNA VARIABLE EL METODO DEL PARENT
  //QUE VIENE EN PROPS
  var ejecutarPadre = props.metodoPadre;
  return (<div>
    <h1 style={{color:"blue"}}>
      Saludo HIJO
    </h1>
    <button onClick={ () => ejecutarPadre() }>
      Llamar al Parent
    </button>
  </div>
}
export default SaludoHijo;

```

#### SALUDOPADRE.JS

```

import SaludoHijo from "./SaludoHijo";
function SaludoPadre() {
  //NECESITAMOS UN METODO EN ESTE CODIGO PARA
  //QUE EL HIJO SEA CAPAZ DE EJECUTARLO
  const metodoPadre = () => {
    console.log("Ejecutando método del PARENT");
  }
  return (<div>
    <h1 style={{color:"red"}}>
      Saludo Padre
    </h1>
    {/* DESDE PROPS ENVIAREMOS EL METODO DEL PARENT
    PARA QUE EL HIJO PUEDA REALIZAR LA LLAMADA */}
    <SaludoHijo metodoPadre={metodoPadre}>
  </div>
}

```

```
export default SaludoPadre;
```

Y podremos visualizar el resultado

## Saludo Padre

## Saludo HIJO

Llamar al Parent

Realmente dónde cobra importancia esta funcionalidad es cuando podemos enviar Parámetros desde el Hijo al Padre para que también sea dinámico.

### PADRE

```
const metodoPadre = (VALOR) => {  
}
```

### HIJO

```
<button onClick={ () => ejecutarPadre('SOY UN HIJO') }>Llamar al Parent</button>
```

Vamos a modificar el código del component **SaludoPadre.js** para que reciba un mensaje.

### SALUDOPADRE.JS

```
function SaludoPadre() {  
    //NECESITAMOS UN METODO EN ESTE CODIGO PARA  
    //QUE EL HIJO SEA CAPAZ DE EJECUTARLO  
    const metodoPadre = (nombre) => {  
        console.log("Yo soy tu padre, " + nombre);  
    }  
}
```

### SALUDOHIJO.JS

```
function SaludoHijo(props) {  
    //CAPTURAMOS EN UNA VARIABLE EL METODO DEL PARENT  
    //QUE VIENE EN PROPS  
    var ejecutarPadre = props.metodoPadre;  
    return (<div>  
        <h1 style={{color:"blue"}}>  
            Saludo HIJO  
        </h1>  
        <button onClick={ () => ejecutarPadre('Luke') }>  
            Llamar al Parent  
        </button>  
    </div>)  
}
```

Si necesitamos los valores dinámicos, tendríamos que combinar tanto **props** como métodos del Parent

### SALUDOPADRE.JS

```
<SaludoHijo idhijo="1" metodoPadre={metodoPadre}/>  
<SaludoHijo idhijo="2" metodoPadre={metodoPadre}/>  
<SaludoHijo idhijo="3" metodoPadre={metodoPadre}/>
```

### SALUDOHIJO.JS

```

function SaludoHijo(props) {
  //CAPTURAMOS EN UNA VARIABLE EL METODO DEL PARENT
  //QUE VIENE EN PROPS
  var ejecutarPadre = props.metodoPadre;
  var idhijo = props.idhijo;
  return (<div>
    <h1 style={{color:"blue"}}>
      Saludo HIJO
    </h1>
    <button onClick={() => ejecutarPadre('Luke ' + idhijo)}>
      Llamar al Parent
    </button>
  </div>
}
export default SaludoHijo;

```

Y ya podremos visualizar el código de comunicación dinámico

#### PRACTICA COMUNICACIÓN ENTRE COMPONENTS

Vamos a crear un componente llamado **Matematicas** que contendrá dos botones. Uno de los botones será para mostrar el **doble** de un número y otro de los botones será para mostrar el **triple** de un número. El componente **Matematicas** recibirá un **número** mediante **props** del Parent. (*PadreMatematicas.js*)

<Matematicas numero="7"/>

Hijo: Al pulsar sobre el botón, ejecutaremos un método para mostrar el DOBLE  
Padre: Al pulsar el botón del HIJO, mostramos el TRIPLE

Escribimos el método **dobleNumero()** dentro del HIJO (*Matematicas.js*)  
Escribimos el método **tripleNumero()** en el PARENT (*PadreMatematicas.js*)

#### MATEMATICAS.JS

```

function Matematicas(props) {
  var numero = props.numero;
  var mostrarTriple = props.mostrarTriple;
  const mostrarDoble = () => {
    var resultado = parseInt(numero) * 2;
    console.log(resultado);
  }
}

```

```

    return (<div>
      <h1 style={{color:"fuchsia"}}>Matematicas Hijo: {numero}</h1>
      <button onClick={() => mostrarDoble()}>
        Doble {numero}
      </button>
      <button onClick={() => mostrarTriple(numero)}>
        Triple {numero}
      </button>
    </div>)
  }
export default Matematicas;

```

#### PADRE MATEMATICAS.JS

```

import Matematicas from "./Matematicas";
function PadreMatematicas() {
  const mostrarTriple = (valor) => {
    var resultado = parseInt(valor) * 3;
    console.log(resultado);
  }
  return (<div>
    <h1>Padre Matematicas</h1>
    <Matematicas numero="4" mostrarTriple={mostrarTriple}/>
    <Matematicas numero="99" mostrarTriple={mostrarTriple}/>
  </div>)
}
export default PadreMatematicas;

```

#### ESTADO DE LA PAGINA

El estado de las páginas dentro de React es algo básico. Nos indicar los "cuando" suceden Las acciones en una página, por ejemplo:

- Cargar un Component
- Update un Component
- Eliminar un component

En React, cuando algo en el **render** es modificado, no visualizamos los cambios. Mediante los estados de la página, podemos indicar que dibuje de nuevo nuestras Variables que hayan cambiado.

En react, para poder visualizar variables que sean modificadas en el dibujo, deben ser De tipo **state**

Para poder trabajar con ellas debemos indicar la palabra **useState**. Con dicha Declaración de variables, le indicamos al render que se vea forzado a hacer Seguimiento a dichas variables.

Dentro de estas variables "especiales" tenemos dos atributos:

- 1) **get**: Cuando recuperamos el valor de la variable
- 2) **set**: Cuando asignamos un valor a la variable

Con este tipo de variables se utiliza una sintaxis parecida a desestructurar

```
const {dato1, dato2} = props;
```

La única diferencia es que utilizan **CORCHETES** en lugar de llaves.  
Se deben igualar a **useState**

Sintaxis:

```
const [ numero, setNumero ] = useState(VALOR INICIAL);
```

La primera variable es para dibujar el valor y la segunda variable es para establecer el valor.

Para poder utilizar **useState** debemos hacer un **import** de 'react'

Vamos a realizar un ejemplo sencillo en el que vamos a dibujar un contador cada vez Que pulsemos un botón.  
Creamos un nuevo componente llamado **Contador.js**

# Ejemplo Contador State

## Contador 2

Sumar contador

### CONTADOR.JS

```
//PARA PODER UTILIZAR useState DEBEMOS HACER
//UN IMPORT DE react
import { useState } from "react";
function Contador() {
  //DECLARAMOS UNA VARIABLE DE TIPO STATE Y
  //EN LA CREACION LE INDICAMOS EL TIPO DE DATO
  const [ numero, setNumero ] = useState(0);
  const sumarContador = () => {
    //PARA MODIFICAR EL VALOR DE UNA VARIABLE DE TIPO
    //STATE, SE UTILIZA setVariable();
    setNumero(numero + 1);
  }
  return (<div>
    <h1 style={{color:"blue"}}>
      Ejemplo Contador State
    </h1>
    <h2 style={{color:"red"}}>Contador {numero}</h2>
    <button onClick={() => sumarContador()}>
      Sumar contador
    </button>
    <button onClick={() => {
      setNumero(numero - 1);
    }}>
      Restar contador
    </button>
  </div>
)
export default Contador;
```

### PRACTICA JQUERY

- Realizar la práctica con JQUERY
- El EndPoint es ALUMNOS

## Alumnos

- Al iniciar la aplicación, mostraremos en un desplegable (<select>) los diferentes cursos
- Cuando seleccionemos un Curso, mostraremos el Nombre y Apellidos en una lista De los alumnos de dicho curso

[Swagger UI \(apiejemplos.azurewebsites.net\)](#)

### OBJETOS EN REACT

Debemos recordar que para poder utilizar los dibujos dinámicos necesitamos una Variable con useState y con su GET y su SET

Pongamos que necesitamos almacenar los datos de una Persona.

```
const [nombre, setNombre] = useState("");
const [apellidos, setApellidos] = useState("");
const [edad, setEdad] = useState(0);
```

La forma más óptima para hacer esto es trabajar con objetos.

Un objeto se declara con atributos/propiedades en formato key:value como en JSON.

```
var objeto = {
  propiedad1:valor1, propiedad2: valor2
}
```

Posteriormente, simplemente se accede a la propiedad: **objeto.propiedad1**

Dentro de los métodos de React, también podemos devolver código HTML.

```
const mostrarMensaje = () => {
  return (<h1>Soy CODIGO HTML</h1>);
}
```

Vamos a realizar una aplicación para trabajar con objetos y con state y props  
El objeto que vamos a utilizar será un Coche, dicho coche tendrá una velocidad que recibiremos mediante **props**, una marca, un modelo y las acciones.

Creamos un nuevo componente llamado **Car.js**

## Component Car: Audi, Q3

Velocidad actual: 125 km/h

### Arrancado

Arrancar/Detener Acelerar coche

## Component Car: Pontiac, Firebird

Velocidad actual: 225 km/h

### Arrancado

Arrancar/Detener Acelerar coche

### INDEX.JS

```
<React.StrictMode>
  <Car marca="Audi" modelo="Q3" aceleracion="25" velocidadmaxima="240"/>
  <Car marca="Pontiac" modelo="Firebird" aceleracion="45" velocidadmaxima="340"/>
</React.StrictMode>
);
```

### CAR.JS

```
import { useState } from "react";
function Car(props) {
  //VAMOS A CREAR UNA VARIABLE QUE NOS PERMITIRA
  //VISUALIZAR EL ESTADO DEL COCHE
  const [estado, setEstado] = useState(false);
  //VAMOS A TENER UNA VARIABLE PARA VISUALIZAR LA VELOCIDAD ACTUAL DEL
  //COCHE
  const [velocidad, setVelocidad] = useState(0);
  //CREAMOS UN OBJETO CUYAS PROPIEDADES VENDRAN DEFINIDAS EN
  //LA ETIQUETA DEL PARENT
  var coche = {
    marca: props.marca,
    modelo: props.modelo,
    velocidadMaxima: parseInt(props.velocidadmaxima),
    aceleracion: parseInt(props.aceleracion)
  }
  //INCLUIREMOS UN METODO PARA COMPROBAR EL ESTADO DEL COCHE
  //DEPENDIENDO DE DICHO ESTADO, LO QUE HAREMOS SERA DEVOLVER CODIGO
  //HTML
  const comprobarEstado = () => {
    if (estado == true){
      return (<h1 style={{color: "green"}}>Arrancado</h1>)
    }else{
      return (<h1 style={{color: "red"}}>Detenido</h1>)
    }
  }
  //CREAMOS UN METODO PARA CAMBIAR LA ACCELERACION DEL COCHE
  const acelerarCoche = () => {
    if (estado == false){
      alert("El coche esta detenido");
      setVelocidad(0);
    }else{
      if (velocidad >= coche.velocidadMaxima){
        //PONEMOS LA VELOCIDAD MAXIMA
        setVelocidad(coche.velocidadMaxima);
      }else{
        //CAMBIAMOS LA VELOCIDAD JUNTO A SU ACCELERACION
        setVelocidad(velocidad + coche.aceleracion);
      }
    }
    return (<div>
      <h1 style={{color:"blue"}}>
        Component Car: {coche.marca}, {coche.modelo}
      </h1>
      <h2 style={{color:"fuchsia"}}>
        Velocidad actual: {velocidad} km/h
      </h2>
      <div>
        {comprobarEstado()}
      </div>
      <button onClick={() => {
        //MODIFICAR EL ESTADO
        setEstado(!estado);
      }}>
        Arrancar/Detener
      </button>
      <button onClick={() => acelerarCoche()}>

```

```

        Acelerar coche
    </button>
</div>
}
export default Car;

```

Mientras voy contando teoría, vamos a cerrar el proyecto actual y creamos un nuevo Proyecto llamado **reactclases**

## REACT CLASES JSX ES6

Ahora que ya somos máquinas en React, vamos a cambiar la sintaxis.

Dentro de React tenemos dos tipos de Sintaxis para crear components, todo funciona igual, podemos combinar las dos, pero NO dentro de un mismo component. Hemos utilizado un tipo de sintaxis de JavaScript. Ahora vamos a utilizar una sintaxis más parecida a Java o C#, con clases.

Ejemplo Component:

```

class MiComponent extends Component {
    //LOS METODOS CAMBIAN SU SINTAXIS
    miMetodo = () => {
        //ACCION DEL METODO
    }

    //EL DIBUJO VA DENTRO DE LA PALABRA render()
    render() {
        return (<h1>Soy Component Class</h1>)
    }
}

export default MiComponent;

```

Al estar hablando de clases tenemos más control sobre las acciones, es decir, no Todo el código está ahí escrito dentro de function. También nos permitirá poder incluir más código, por ejemplo, **constructores** de clase. Un constructor es el primer código que se ejecuta en una clase.

La palabra **extends** indica la herencia de una clase y también podremos utilizar **Props** y **state** de forma nativa, sin necesidad de estar declarando variable con **useState**

Se utiliza la palabra **this** para acceder a los elementos y el propio component dentro de la Clase

Por ejemplo, cuando digo "nativo" es que no es necesario declarar ni indicar nada En nuestro código, ya viene por la herencia de extends:

```
<MiComponente propiedad="valor"/>

class MiComponente extends Component {
    this.props.propiedad
}
```

El funcionamiento de **useState** es el mismo si queremos que los dibujos sean dinámicos.

En el component tenemos una propiedad llamada **state** que nos permite declarar en su Interior todos los elementos que deseamos que sean dinámicos en el dibujo del **render**

La variable **state** se utiliza como objeto.

```
class MiComponent extends Component {
    state = {
        velocidad: 150,
        estado: false,
        coche: {
            marca: "AUDI",
            modelo: "Q7"
        }
    }
}
```

En nuestro dibujo, si deseamos acceder a cualquier elemento de state

```
{ this.state.velocidad }
{ this.state.coche.marca }
```

Las variables son de SOLO lectura, tenemos un método llamado **setState** para modificar El valor de una o varias variables de estado.

```
this.setState({velocidad: 110});
```

Si deseamos hacerlo en un método

```
acelerarCoche = () => {
    console.log("Acelerando!!!");
    this.setState({velocidad: 50});
}
```

Vamos a comenzar realizando el ejemplo del **Contador** pero aplicando la sintaxis **JSX**

# Class Component Contador

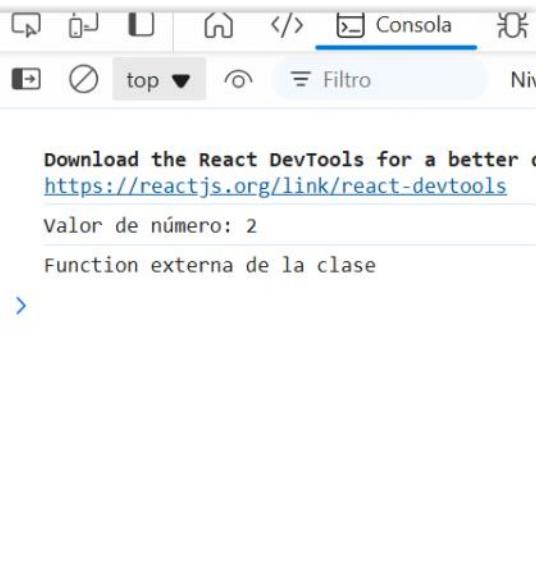
Inicio del contador: 9

Valor del state: 12

# Class Component Contador

Inicio del contador: 14

Valor del state: 14



## CONTADOR.JS

```
<React.StrictMode>
  <Contador inicio="9"/>
  <Contador inicio="14"/>
</React.StrictMode>

//DEBEMOS IMPORTAR Component PARA LA HERENCIA
import { Component } from "react";
//PODEMOS DECLARAR METODOS FUERA DE LA CLASE
//DICHOS METODOS NO PUEDEN UTILIZAR NADA DEL Component
//Y SE DECLARAN CON function
function metodoExterno() {
  console.log("Function externa de la clase");
}

class Contador extends Component {
  //LAS VARIABLES Y LOS METODOS SE DECLARAN FUERA DEL RENDER
  //NO SE UTILIZAN PALABRAS CLAVE var, let o const
  numero = 1;
  //LOS METODOS SE DECLARAN DIRECTAMENTE AQUI
  incrementarNumero = () => {
    //PARA PODER ACCEDER A LAS VARIABLES DE LA CLASE
    //DEBEMOS UTILIZAR LA PALABRA CLAVE this
    this.numero = this.numero + 1;
    console.log("Valor de número: " + this.numero);
  }

  //VAMOS A DECLARAR UNA VARIABLE DE ESTADO QUE TENDRA
  //EL VALOR DE PROPS
  state = {
    valor: parseInt(this.props.inicio)
  }
  //CREAMOS UN METODO PARA CAMBIAR EL VALOR DEL STATE
  incrementarValorState = () => {
    //PARA MODIFICAR LOS ELEMENTOS QUE TENGAMOS DENTRO DE STATE
    //SE UTILIZA setState CON UN JSON DEL OBJETO CON LAS VARIABLES
    //QUE DESEEMOS MODIFICAR
    //LAS VARIABLES QUE NO MODIFIQUEMOS NO CAMBIARAN
    this.setState({
      valor: this.state.valor + 1
    });
  }
  render() {
    return (<div>
      <h1>Class Component Contador</h1>
      <h2 style={{color: "blue"}}>
        Inicio del contador: {this.props.inicio}
      </h2>
      <h2 style={{color: "red"}}>
        Valor del state: {this.state.valor}
      </h2>
      /* LA LLAMADA A LOS METODOS ES MAS SENCILLA, NO NECESITAMOS
      LAMBDA Y TAMPOCO SE UTILIZAN PARENTESIS*/
      <button onClick={this.incrementarValorState}>
        Incrementar state
      </button>
      /* VAMOS A UTILIZAR UNA FUNCION ANONIMA PARA LLAMAR A UN METODO */
      <button onClick={() => {
        //SI DESEAMOS LLAMAR A UN METODO DE LA CLASE, SE UTILIZA LA
        //PALABRA this
        this.incrementarNumero();
        //SI DESEAMOS LLAMAR A UN METODO EXTERNO DEL COMPONENT
        //NO UTILIZAMOS this
        //metodoExterno();
      }}>
        Incrementar número
      </button>
    </div>
  }
}

export default Contador;
```

## DIBUJOS COMPLEJOS HTML

Hace un rato, hemos podido realizar dibujos dinámicos dentro del render

```
const comprobarEstado = () => {
  if (estado == true){
    return (<h1 style={{color: "green"}}>Arrancado</h1>)
  }else{
    return (<h1 style={{color: "red"}}>Detenido</h1>)
  }
}
```

Solamente podemos utilizar un **return**

¿Qué sucede si deseamos realizar algún dibujo más complejo?  
Por ejemplo, si deseamos dibujar una tabla (**<tr><td>**) o una lista (**<li>**)

Tenemos dos formas de devolver dibujos más complejos:

- 1) Método que devuelva el código HTML utilizando código JS, por ejemplo, Almacenando el código a devolver dentro de un Array y devolviendo el valor Del Array.

```
var titulos = [];
titulos.push(<h1>Titulo 1</h1>);
titulos.push(<h1>Titulo 2</h1>);
titulos.push(<h1>Titulo 3</h1>);
return titulos;
```

- 2) Utilizando el código **render** de react

Vamos a crear una nueva clase llamada **DibujosComplejos.js**

## Dibujos complejos HTML

- 11
- 69
- 28
- 10
- 11
- 49
- 6

### DIBUJOSCOMPLEJOS.JS

```
import { Component } from "react";
class DibujosComplejos extends Component{
  //VAMOS A DIBUJAR UNA SERIE DE NUMEROS EN FORMATO HTML
  //UTILIZANDO UN ARRAY CON <li>
  dibujarNumeros = () => {
    //DECLARAMOS UN ARRAY PARA ALMACENAR EL DIBUJO
    var lista = [];
    //VAMOS A REALIZAR UN BUCLE PARA RELLENAR NUMEROS DINAMICOS
    for (var i = 1; i <=7; i++) {
      var numero = parseInt(Math.random() * 100) + 1;
      //MEDIANTE EL METODO push DEL ARRAY IREMOS
      //RELLENANDO EL CODIGO HTML
      lista.push(<li>{numero}</li>);
    }
    return lista;
  }
  render() {
    return (<div>
      <h1>Dibujos complejos HTML</h1>
      <ul>
        {this.dibujarNumeros()}
      </ul>
    </div>
  )
}
export default DibujosComplejos;
```

Al ejecutar nuestra App podemos visualizar un error en la consola

# Dibujos complejos HTML

- 11
- 69
- 28
- 10
- 11
- 49
- 6

The screenshot shows a browser's developer tools console. At the top, there are various icons and a dropdown menu labeled "Consola". Below the toolbar, the console output is displayed. A red warning icon is present. The text in the console includes:

- "hot module Download the React DevTools for a better development experien <https://reactjs.org/link/react-devtools>"
- "Warning: Each child in a list [react-jsx-dev-runtime.deve](#) should have a unique "key" prop."
- "Check the render method of `DibujosComplejos`. See <https://reactjs.org/link/warning-keys> for more information."
- "at li at DibujosComplejos (<http://localhost:3000/main.10f1fec...hot-update.js:25:5>)"
- "[NUEVO] Explicar errores de la consola con Copilot en Edge: haga clic [🔗](#) para explicar un error. [Más inform](#) No vo  
ación"

Cuando generamos código complejo dentro de react, necesitamos incluir una Clave (key) única para cada elemento del dibujo.  
Necesita un **KEY UNICO**

// REllenando el código HTML  
lista.push(<li key={i}>{numero}</li>);

El siguiente ejemplo dibujará elementos complejos, pero utilizando el código Del **render**

Dicha lógica debe estar escrita dentro del render y con **LLAVES**

```
render() {
  return (<div>
    <h1>Título del Render</h1>
    {
      //LOGICA REACT
      return (<h1>LOGICA</h1>)
    }
  </div>
}
```

Para poder realizar un bucle dentro de una colección se utiliza **map(objeto, index)**

Para poder trabajar dentro del render no se utiliza un código nativo como, por ejemplo, Acabamos de ver con un método.

Debemos utilizar **map** si deseamos trabajar con objetos múltiples/colección.

```
Array.map(object, index) {
  //ACCESO A CADA OBJETO
}
```

La ventaja es que dentro de Array.map en el render podemos hacer un **return** y continuará La ejecución del bucle.

Vamos a realizar una App en la que tendremos una colección de nombres.

Dibujaremos dichos nombres dentro del **render()**

Tendremos la posibilidad de añadir nuevos nombres.

- 1) En el **render()**, debemos dibujar los nombres al inicio
- 2) Necesitamos un método **push** en el Array para añadir nuevos nombres
- 3) Al pulsar un botón, añadimos un nombre
- 4) El Array debe estar declarado dentro de **state**

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

# Dibujos complejos react Collection

Generar nombre

**Lucia**

**Adrian**

**Antonia**

**Diana**

**Sofia**

**Carlos**

**NUEVO NOMBRE**

## DIBUJOSCOMPLEJOSREACT.JS

```
import { Component } from "react";
class DibujosComplejosReact extends Component{
    //EN STATE TENDREMOS UN CONJUNTO DE NOMBRES
    state = {
        nombres: ["Lucia", "Adrian", "Antonia", "Diana", "Sofia", "Carlos"]
    }
    generarNombre = () => {
        this.state.nombres.push("NUEVO NOMBRE");
        //ACTUALIZAMOS STATE
        this.setState({
            nombres: this.state.nombres
        });
    }
    render(){
        return (<div>
            <h1>Dibujos complejos react Collection</h1>
            <button onClick={() => this.generarNombre()}>
                Generar nombre
            </button>
            {
                //ESTO ES CODIGO REACT, ES DIFERENTE AL CODIGO JS
                //ES CODIGO LOGICO CON SINTAXIS JSX
                //EL CODIGO LOGICO DEBE CONTENER UN RETURN
                this.state.nombres.map((name, index) => {
                    //ESTE CODIGO NUNCA DEBE ESTAR VACIO, SIEMPRE
                    //TIENE QUE DEVOLVER UN return
                    return (<h1 key={index} style={{color: "blue"}}>{name}</h1>
                })
            }
        </div>
    }
    export default DibujosComplejosReact;
```

## COMUNICACIÓN PADRE-HIJO, HIJO-PADRE

La comunicación entre Padre e Hijo no cambia nada, seguimos trabajando con **props** pero  
Con la sintaxis **this.props** y la propiedad a la que deseemos acceder

La llamada de Hijo a Padre se sigue realizando con métodos

Necesitamos un método en el Padre y un método en el Hijo.

Los métodos se envían mediante **props**

### PADRE

```
metodoPadre = () => {
    //ACCIONES EN EL PARENT
}
```

El método es enviado al hijo mediante props

```
<Hijo nombre="Soy tu hijo" callPadre={metodoPadre}/>
```

### HIJO

Necesitamos un método para invocar el props del Parent

```
metodoAccionHijo = () => {
    //DESDE AQUÍ LLAMAMOS AL METODO DEL PADRE CON EL NAME DE PROPS
    this.props.callPadre(VALOR);
```

}

Vamos a realizar la siguiente práctica:

- Tendremos una serie de deportes en un Array
- Recorreremos los deportes con **React**
- Tendremos dos Componentes: **PadreDeportes** y **HijoDeporte**
- El padre dibujará una etiqueta Hijo por cada deporte que contenga en el Array
- Al pulsar un botón en cada hijo, podremos seleccionar nuestro deporte Favorito

## Padre deportes

Su deporte favorito es: Curling

### Petanca

### Curling

### Poker

### Padel

#### HIJODEPORTE.JS

```
import { Component } from "react";
class HijoDeporte extends Component{
    seleccionarFavorito = () => {
        //CAPTURAMOS EL DEPORTE SELECCIONADO EN props
        var deporte = this.props.nombre;
        //REALIZAMOS LA LLAMADA AL PADRE ENVIANDO EL DEPORTE
        this.props.mostrarFavorito(deporte);
    }
    render() {
        return (<div>
            <h2 style={{color: "blue"}}>
                {this.props.nombre}
            </h2>
            <button onClick={this.seleccionarFavorito}>
                Seleccionar favorito
            </button>
        </div>
    }
}
export default HijoDeporte;
```

#### PADREDEPORTES.JS

```
import { Component } from "react";
import HijoDeporte from "./HijoDeporte";
class PadreDeportes extends Component{
    deportes = ["Petanca", "Curling", "Poker", "Padel", "Futbol"]
    //NECESITAMOS UN METODO PARA DIBUJAR EL DEPORTE HIJO
    //RECIBIREMOS EL DEPORTE FAVORITO SELECCIONADO EN DICHO METODO
    mostrarFavorito = (deporteSeleccionado) => {
        //MODIFICAMOS EL DEPORTE FAVORITO DE STATE
        this.setState({
            favorito: deporteSeleccionado
        })
    }
    //CREAMOS UNA VARIABLE STATE PARA MOSTRAR EL DEPORTE SELECCIONADO
    state = {
        favorito: ""
    }
    render() {
        return (<div>
            <h1 style={{color: "red"}}>Padre deportes</h1>
            <h4 style={{backgroundColor:"yellow"}}>
                Su deporte favorito es: {this.state.favorito}
            </h4>
            {
                //RECORREMOS TODOS LOS DEPORTES Y DIBUJAMOS ETIQUETAS
                //HIJO POR CADA UNO
                this.deportes.map((deporte, index) => {
                    return (<HijoDeporte key={index}>
                        nombre={deporte} mostrarFavorito={this.mostrarFavorito}/>
                    )
                })
            }
        </div>
    }
}
```

```
}
```

```
export default PadreDeportes;
```

#### PRACTICA COMUNICACIÓN COMPONENTS

- Realizar una página que dibujará componentes basados en números aleatorios
- Tendremos un component llamado **HijoNúmero.js** y un component llamado **PadreNumeros.js**
- El dibujo de cada **HijoNúmero** será mediante un Array en **PadreNumeros.js**
- Al pulsar un botón **Nuevo Número** en el component **Padre**, generamos un número aleatorio. (Un nuevo Component Hijo)
- En el cada Component **HijoNúmero**, tendremos un botón que al pulsar irá SUMANDO
- La suma la veremos en el PADRE en un triste <h1>

## Padre Números

La suma es 94

Generar número

Número: 5

Sumar 5

Número: 11

Sumar 11

Número: 57

Sumar 57

#### PADRENUMEROS.JS

```
import { Component } from "react";
import HijoNumero from "./HijoNumero";
class PadreNumeros extends Component{
    state = {
        numeros: [5, 11],
        suma: 0
    }
    sumarNumero = (valor) => {
        this.setState({
            suma: this.state.suma + valor
        })
    }
    generarNumeroAleatorio = () => {
        let aleatorio = parseInt(Math.random() * 200) + 1;
        this.state.numeros.push(aleatorio);
        this.setState({
            numeros: this.state.numeros
        });
    }
    render() {
        return (<div>
            <h1>Padre Números</h1>
            <h2 style={{backgroundColor:"yellow"}}>
                La suma es {this.state.suma}
            </h2>
            <button onClick={this.generarNumeroAleatorio}>
                Generar número
            </button>
            {
                this.state.numeros.map((num, index) => {
                    return (<HijoNumero numero={num} key={index}>
                        sumarNumero={this.sumarNumero}</HijoNumero>)
                })
            }
        </div>
    }
}
export default PadreNumeros;
```

#### HIJONUMERO.JS

```
import { Component } from "react";
class HijoNumero extends Component {
    sumarNumeroHijo = () => {
        var num = parseInt(this.props.numero);
        this.props.sumarNumero(num);
    }
    render() {
        return (<div>
            <h1 style={{color:"red"}}>
                Número: {this.props.numero}
            </h1>
            <button onClick={this.sumarNumeroHijo}>
```

```

        Sumar {this.props.numero}
    </button>
</div>
}
export default HijoNumero;

```

## CONDICIONALES REACT

Dentro del lenguaje React (render()) podemos realizar acciones pero con una sintaxis "especial".

Sucede lo mismo con los condicionales, es decir, nuestros tradicionales IF. Dentro de React Dichos IF se escriben con una sintaxis especial.

```

{
  //CONDICIONAL IF SIMPLE
  this.variable == 0 &&
  (<h1>CONDICION POSITIVA</h1>)
}

```

CONDICIONAL CON UN IF ELSE

```

{
  //CONDICIONAL IF ELSE
  this.variable == 0 ?
  (<h1>CONDICION POSITIVA</h1>):
  (<h1>CONDICION NEGATIVA</h1>)
}

```

CONDICIONAL IF Y ELSE IF

```

{
  //CONDICIONAL IF ELSE
  this.variable == 0 ?
  (<h1>CONDICION POSITIVA CERO</h1>):
  this.variable == 1 ?
  (<h1>CONDICION POSITIVA 1</h1>):
  (<h1>CONDICION ELSE</h1>)
}

```

MAS PREGUNTAS ENTRE PARENTESIS

```

{
  //CONDICIONAL IF ELSE
  (this.variable == 0 || this.variable == 5)?
  (<h1>CONDICION POSITIVA CERO</h1>):
  this.variable == 1 ?
  (<h1>CONDICION POSITIVA 1</h1>):
  (<h1>CONDICION ELSE</h1>)
}

```

Vamos a realizar una práctica como la de números, pero trabajando con objetos.  
Tendremos un conjunto de Comics y un Parent y un Hijo.

El padre dibujará múltiples comics y cada comic podrá ser seleccionado o eliminado.

Tendremos un Component llamado **Comic.js** y otro llamado **Comics.js**

## Comics

### Spiderman



### Spiderman

Hombre araña



[Seleccionar Favorito](#) | [Eliminar Comic](#)

### Wolverine

Lobezno



[Seleccionar Favorito](#) | [Eliminar Comic](#)

## COMICS.JS

```
import { Component } from "react";
import Comic from "./Comic";
class Comics extends Component{
  state = {
    comics: [
      {
        titulo: "Spiderman",
        imagen:
          "https://3.bp.blogspot.com/-i70Zu_LAHwI/T290xxduu-I/AAAAAAAALq8/8bXDrdv50o/s1600/spiderman1.jpg",
        descripcion: "Hombre araña"
      },
      {
        titulo: "Wolverine",
        imagen:
          "https://images-na.ssl-images-amazon.com/images/I/51c1Q1IdUBL.SX259_B01_204_203_200_.jpg",
        descripcion: "Lobezno"
      },
      {
        titulo: "Guardianes de la Galaxia",
        imagen:
          "https://cdn.normacomics.com/media/catalog/product/cache/1/thumbna_il/9df78ebab33525d08d6e5fb8d27136e95/g/u/guardianes_galaxia_guardianes_infinito.jpg"
      },
      {
        descripcion: "Yo soy Groot"
      },
      {
        titulo: "Avengers",
        imagen:
          "https://d261pennugtm8s.cloudfront.net/stores/057/977/products/ma_vengers_01_01-891178138c020318f315132687055371-640-0.jpg",
        descripcion: "Los Vengadores"
      },
      {
        titulo: "Spawn",
        imagen:
          "https://i.pinimg.com/originals/e1/d8/ff/e1d8ff4aeab5e567798635008fe98ee1.png",
        descripcion: "Al Simmons"
      },
      {
        titulo: "Batman",
        imagen:
          "https://www.comicverso.com/wp-content/uploads/2020/06/The-Killing-Joke-657x1024.jpg",
        descripcion: "Murcielago"
      }
    ],
    favorito: null
  }
  seleccionarFavorito = (comicFavorito) => {
    console.log("seleccionar favorito");
    this.setState({
      favorito: comicFavorito
    })
  }
  eliminarComic = (index) => {
    //ELIMINAR ALGO DE UN ARRAY
    //TENEMOS UN METODO LLAMADO splice QUE NOS PIDE EL INDEX
    //DEL ELEMENTO A ELIMINAR DEL ARRAY Y EL NUMERO DE ELEMENTOS A
    //ELIMINAR
    console.log("eliminar comic");
    this.state.comics.splice(index, 1);
    this.setState({
      comics: this.state.comics
    })
  }
}
```

```

    render(){
      return (<div>
        <h1>Comics</h1>
        {
          this.state.favorito &&
          (<div style={{backgroundColor: "lightgreen"}}>
            <h2 style={{color:"blue"}}>
              {this.state.favorito.titulo}
            </h2>
            <img src={this.state.favorito.imagen}
              style={{width: "100px", height: "150px"}}/>
          </div>
        )
        {
          this.state.comics.map((objetoComic, index) => {
            return (<Comic key={index}
              index={index}
              comic={objetoComic}
              seleccionarFavorito={this.seleccionarFavorito}
              eliminarComic={this.eliminarComic}/>)
          })
        }
      </div>
    )
  }
  export default Comics;

```

#### CÓDIGO

```

import { Component } from "react";
class Comic extends Component{
  render() {
    return (<div>
      <h1>{this.props.comic.titulo}</h1>
      <p>{this.props.comic.descripcion}</p>
      <img src={this.props.comic.imagen}
        style={{width: "100px", height: "150px"}}/>
      <button onClick={() => {
        this.props.seleccionarFavorito(this.props.comic);
      }}>
        Seleccionar Favorito
      </button>
      <button onClick={() => {
        this.props.eliminarComic(this.props.index);
      }}>
        Eliminar Comic
      </button>
    </div>
  )
}
export default Comic;

```

CREAMOS UN NUEVO PROYECTO LLAMADO **reactformsrutas**

#### RUTAS REACT

Dentro de los Frameworks no tenemos las librerías instaladas directamente para cada Funcionalidad, por ejemplo, si queremos utilizar sweetalert o utilizar bootstrap, necesitamos Acceder a las librerías mediante **npm**  
Si necesitamos acceder a Servicios Api, también debemos instalar librerías, es decir, los Proyectos vienen limpios y, a medida que vamos progresando debemos incluir funcionalidades.

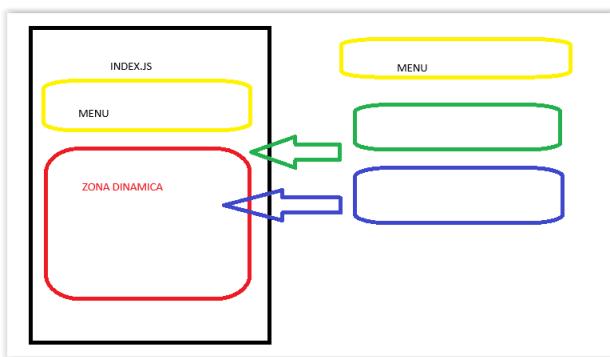
Lo mismo sucede con las rutas en React. Una ruta es poder navegar entre components Utilizando Links.

Para instalar la navegación necesitamos el siguiente comando:

**npm install --save react-router-dom**

Dentro de todos los frameworks SPA no existe navegación tradicional, ya que no existen Páginas sino que estamos hablando de Components  
Necesitamos una serie de elementos, por ejemplo, una Tabla de rutas que indicará las Rutas que debe seguir un Link para cada Component

Tendremos elementos estáticos dentro de la página principal y tendremos elementos Dinámicos al cargar los components con sus links



Este concepto se llama **ROUTING**

Debemos realizar las siguientes acciones para poder aplicar Routing:

- 1) Necesitamos una clase que se encargará de mapear las rutas que nosotros Indiquemos con cada Component.

Dicha clase se llamará siempre **Router.js**  
A partir de esta clase, dibujaremos dentro de **index.js** el "lugar" dinámico  
Para cargar cada Component con cada ruta

- 2) Añadir a **Router** cada ruta para cada Component

```
<Ruta path="/" Component={Home}/>
<Ruta path="/comics" Component={Comics}/>
```

Las rutas irán por URL

<https://localhost:3000/> -> HOME  
<https://localhost:3000/comics> --> COMICS

Para indicar la "zona" dinámica para cada component se utiliza la etiqueta <Router/>

```
root.render(
  <React.StrictMode>
    <h1>TITULO ESTATICO</h1>
    <...><Router/>
    <footer>Pie de página</footer>
  </React.StrictMode>
);
```

El component **Router.js** debe ir dentro de la carpeta **components**

#### ROUTER.JS

```
class Router extends Component {
  render() {
    <BrowserRouter>
      <Routes>
        <Route path="/" component={Home}/>
        <Route path="/comics" component={Comics}/>
      </Routes>
    </BrowserRouter>
  }
}
```

Vamos a tener tres components.  
**Home**, **Cine**, **Musica**

## INDEX PRINCIPAL

### Cine Component



### Pie de página

```

import {Component} from 'react'

class Home extends Component{
    render() {
        return (<div>
            <h1>Home Component</h1>
            
        </div>
    }
}

export default Home;

```

Creación de Components mediante snippets: rcc + TAB

Sobre **components** creamos un nuevo component llamado **Router.js**

```
import {BrowserRouter, Routes, Route} from 'react-router-dom';
```

#### ROUTER.JS

```

import {BrowserRouter, Routes, Route} from 'react-router-dom';
import React, { Component } from 'react';
import Home from './Home';
import Cine from './Cine';
import Música from './Música';
export default class Router extends Component {
    render() {
        return (
            <BrowserRouter>
                <Routes>
                    <Route path="/" element={<Home/>}/>
                    <Route path="/cine" element={<Cine/>}/>
                    <Route path="/música" element={<Música/>}/>
                </Routes>
            </BrowserRouter>
        )
    }
}

```

El siguiente paso será utilizar el Component **Router** dentro de **index.js**

#### INDEX.JS

```

root.render(
    <React.StrictMode>
        <h1>INDEX PRINCIPAL</h1>
        <hr/>
        <Router/>
        <hr/>
        <h2>Pie de página</h2>
    </React.StrictMode>
);

```

El siguiente paso será crearnos un nuevo componente llamado **MenuRutas.js**

#### MENURUTAS

```

import React, { Component } from 'react'
export default class MenuRutas extends Component {
    render() {
        return (
            <div>
                <ul>
                    <li>
                        <a href="/">Home</a>
                    </li>
                    <li>
                        <a href="/cine">Cine</a>
                    </li>
                    <li>
                        <a href="/música">Música</a>
                    </li>
                </ul>
            </div>
        )
    }
}

```

El menú lo incluimos como elemento estático dentro de **index.js**

```
root.render(  
  <React.StrictMode>  
    <MenuRutas/>  
    <hr/>  
    |   <Router/>  
    <hr/>  
    <h2>Pie de página</h2>  
  </React.StrictMode>  
)
```

Y tendremos dibujos estáticos y dinámicos mediante rutas

- [Home](#)
- [Cine](#)
- [Música](#)  


## Musica



## Pie de página

### FORMULARIOS EN REACT

Para trabajar con formularios en React no se realizar de forma tradicional, es decir, buscando un elemento por su ID y trabajando con su value.

Lo primero de todo es que tenemos que incluir, de manera obligatoria, la etiqueta **<form>**

Cuando tenemos una etiqueta **<form>** nuestra página va a intentar realizar un **submit()**

Debemos bloquear dicho submit para capturar la petición en FRONT

Para desactivar el evento, necesitamos recibir el propio evento (event) y aplicar el Método **preventDefault()**: **event.preventDefault()**

Los **<input>** van a ser elementos lógicos dentro de la clase de React, es decir, Debemos crear esos objetos y asociarlos con el código HTML

```
cajaNombre = React.createRef();
```

Ejemplo de una clase con Form

```
class ClaseForm {  
  cajaInput = React.createRef();  
  
  procesarPeticionForm = (evento) => {  
    //ANTES DE REALIZAR NINGUNA ACCION preventDefault  
    evento.preventDefault();  
    //RESTO DE LINEAS DE CODIGO  
  }  
  
  render() {  
    return (<div>  
      <form onSubmit={this.procesarPeticionForm}>  
        <input ref={this.cajaInput}/>  
        <button>Cosas</button>  
      </form>  
    </div>  
  }  
}
```

}

**Nota:** Siempre que trabajemos con FORMS, la petición será sin paréntesis.

Creamos un nuevo componente llamado **FormSimple.js**

# Formulario simple React

## Nombre: Alumno, Edad: 25

Nombre:

Edad:

### FORMSIMPLE.JS

```
import React, { Component } from 'react'
export default class FormSimple extends Component {
    //NECESITAMOS DECLARAR VARIABLES DE REFERENCIA
    //PARA CADA CONTROL DENTRO DE REACT
    cajaNombre = React.createRef();
    cajaEdad = React.createRef();
    state = {
        user: null
    }
    //PARA PROCESAR LA PETICION NECESITAMOS UN METODO
    //QUE RECIBA Event (e)
    peticionForm = (e) => {
        //LA PRIMERA LINEA SIEMPRE SERA DETENER LA PROPAGACION
        //DEL EVENTO (submit)
        e.preventDefault();
        console.log("Petición lista");
        let nombre = this.cajaNombre.current.value;
        let edad = parseInt(this.cajaEdad.current.value);
        this.setState({
            user: {
                nombre: nombre,
                edad: edad
            }
        })
    }
    render() {
        return (
            <div>
                <h1>Formulario simple React</h1>
                {
                    this.state.user &&
                    <h2 style={{color:"blue"}}>
                        Nombre: {this.state.user.nombre}
                        , Edad: {this.state.user.edad}
                    </h2>
                }
                <form onSubmit={this.peticionForm}>
                    <label>Nombre: </label>
                    <input type="text" ref={this.cajaNombre}/><br/>
                    <label>Edad: </label>
                    <input type="text" ref={this.cajaEdad}/><br/>
                    <button>
                        Enviar datos
                    </button>
                </form>
            </div>
        )
    }
}
```

### PRACTICA CONJETURA DE COLLATZ

Todo número positivo siempre será 1 siguiendo estas instrucciones:

- Si el número es PAR, dividimos entre 2
- Si el número es IMPAR, multiplicamos por 3 y sumamos 1

Pongamos como ejemplo el número 6

- 6, 3, 10, 5, 16, 8, 4, 2, 1

Creamos un nuevo Componente llamado **Collatz.js**. Lo ponemos en juego dentro de Nuestro Menú de la aplicación.

Tendremos una caja de texto para pedir el número y al pulsar un botón, mostraremos Una lista (**<li>**) con toda la secuencia de números de Collatz

# Collatz

Introduzca un número

Conjetura Collatz

- 6
- 3
- 10
- 5
- 16
- 8
- 4
- 2
- 1

## COLLATZ.JS

```
import React, { Component } from 'react'
export default class Collatz extends Component {
  //NECESITAMOS UN OBJETO CAJA PARA RECUPERAR EL DATO
  cajaNumero = React.createRef();
  state = {
    numeros: []
  }
  mostrarCollatz = (e) => {
    e.preventDefault();
    let numero = parseInt(this.cajaNumero.current.value);
    let aux = [];
    aux.push(numero);
    this.state.numeros.push(numero);
    while (numero != 1){
      if (numero % 2 == 0){
        numero = numero / 2;
      }else{
        numero = numero * 3 + 1;
      }
      aux.push(numero);
    }
    this.setState({
      numeros: aux
    })
  }
  render() {
    return (
      <div>
        <h1>Collatz</h1>
        <form onSubmit={this.mostrarCollatz}>
          <label>Introduzca un número</label>
          <input type="text" ref={this.cajaNumero}/>
          <button>
            Conjetura Collatz
          </button>
        </form>
        <ul>
          {
            this.state.numeros.map((numero, index) => {
              return (<li key={index}>{numero}</li>)
            })
          }
        </ul>
      </div>
    )
  }
}
```

Realizamos otro ejemplo "parecido", pero con la tabla de multiplicar  
Introducimos un número en una caja y escribimos en una tabla  
La operación y el resultado

5\*1 5  
5\*2 10  
...

Creamos un nuevo componente llamado **TablaMultiplicar.js**

# TablaMultiplicar

Introduzca número  Tabla multiplicar

## Operación Resultado

5 * 1	5
5 * 2	10
5 * 3	15
5 * 4	20
5 * 5	25
5 * 6	30
5 * 7	35
5 * 8	40
5 * 9	45
5 * 10	50

## TABLAMULTIPLICAR.JS

```
import React, { Component } from 'react'
export default class TablaMultiplicar extends Component {
  cajaNumero = React.createRef();
  state = {
    tabla: []
  }
  mostrarTabla = (e) => {
    e.preventDefault();
    let numero = parseInt(this.cajaNumero.current.value);
    console.log(numero);
    var aux=[];
    for (var i = 1; i <= 10; i++) {
      let dato = {
        operacion: numero + " * " + i,
        resultado: (numero * i)
      }
      aux.push(dato);
    }
    this.setState({
      tabla: aux
    })
  }
  render() {
    return (
      <div>
        <h1>TablaMultiplicar</h1>
        <form>
          <label>Introduzca número</label>
          <input type="text" ref={this.cajaNumero}/>
          <button onClick={this.mostrarTabla}>
            Tabla multiplicar
          </button>
        </form>
        <table>
          <thead>
            <tr>
              <th>Operación</th>
              <th>Resultado</th>
            </tr>
          </thead>
          <tbody>
            {
              this.state.tabla.map((fila, index) => {
                return ( <tr key={index}>
                  <td>{fila.operacion}</td>
                  <td>{fila.resultado}</td>
                </tr>
              })
            }
          </tbody>
        </table>
      </div>
    )
  }
}
```

También podemos utilizar, si lo deseamos, los dibujos.

```

        for (var i = 1; i <= 10; i++)
    {
        var operacion = numero + " * " + i;
        var resultado = numero * i;
        //ALMACENAMOS EL DIBUJO DIRECTAMENTE
        aux.push(<tr>
            <td>{operacion}</td>
            <td>{resultado}</td>
        </tr>)
    }
    this.setState({
        tabla: aux
    })
}

```

```

<tbody>
{
    this.state.tabla.map((fila, index) => {
        return (fila)
    })
}
</tbody>

```

#### CODIGO CON DIBUJO HTML

```

import React, { Component } from 'react'
export default class TablaMultiplicar extends Component {
    cajaNumero = React.createRef();
    state = {
        tabla: []
    }
    mostrarTabla = (e) => {
        e.preventDefault();
        let numero = parseInt(this.cajaNumero.current.value);
        console.log(numero);
        var aux=[];
        for (var i = 1; i <= 10; i++)
        {
            var operacion = numero + " * " + i;
            var resultado = numero * i;
            //ALMACENAMOS EL DIBUJO DIRECTAMENTE
            aux.push(<tr>
                <td>{operacion}</td>
                <td>{resultado}</td>
            </tr>)
        }
        this.setState({
            tabla: aux
        })
    }
    render() {
        return (
            <div>
                <h1>TablaMultiplicar</h1>
                <form>
                    <label>Introduzca número</label>
                    <input type="text" ref={this.cajaNumero}/>
                    <button onClick={this.mostrarTabla}>
                        Tabla multiplicar
                    </button>
                </form>
                <table>
                    <thead>
                        <tr>
                            <th>Operación</th>
                            <th>Resultado</th>
                        </tr>
                    </thead>
                    <tbody>
                        {this.state.tabla}
                    </tbody>
                </table>
            </div>
        )
    }
}

```

Versión 2

- En lugar de dibujar una caja de texto, quiero un desplegable con los Números. <select>
- Generamos una serie de números aleatorios y, al iniciar la página, los ponemos Dentro del <select>

```

cajaNumero = React.createRef();
<input/>

```

```
this.cajaNumero.current.value
```

SELECT:

```
selectNumero = React.createRef();
<select>
this.selectNumero.current.value
```

# TablaMultiplicar

Introduzca número

## Operación Resultado

23 * 1	23
23 * 2	46
23 * 3	69
23 * 4	92
23 * 5	115
23 * 6	138

### CODIGO COMPONENT

```
import React, { Component } from 'react'
export default class TablaMultiplicar extends Component {
  selectNumero = React.createRef();
  state = {
    tabla: []
  }
  mostrarTabla = (e) => {
    e.preventDefault();
    let numero = parseInt(this.selectNumero.current.value);
    console.log(numero);
    var aux = [];
    for (var i = 1; i <= 10; i++) {
      let dato = {
        operacion: numero + " * " + i,
        resultado: (numero * i)
      }
      aux.push(dato);
    }
    this.setState({
      tabla: aux
    })
  }
  generarOptions = () => {
    console.log("Loading");
    //VAMOS A GENERAR DINAMICAMENTE TODO EL DIBUJO HTML
    //QUE NECESITAMOS
    var options = [];
    for (var i = 1; i <= 5; i++){
      var random = parseInt(Math.random() * 50) + 1;
      options.push(<option key={i} value={random}>
        {random}
      </option>);
    }
    return options;
  }
  render() {
    return (
      <div>
        <h1>TablaMultiplicar</h1>
        <form>
          <label>Introduzca número</label>
          <select ref={this.selectNumero}>
            {this.generarOptions()}
          </select>
          <button onClick={this.mostrarTabla}>
            Tabla multiplicar
          </button>
        </form>
        <table>
          <thead>
            <tr>
              <th>Operación</th>
              <th>Resultado</th>
            </tr>
          </thead>
          <tbody>
            {
              this.state.tabla.map((fila, index) => {
                return ( <tr key={index}>
                  <td>{fila.operacion}</td>
                  <td>{fila.resultado}</td>
                </tr>
              })
            }
          </tbody>
        </table>
      </div>
    )
  }
}
```

```

        </table>
    </div>
)
}

```

#### SELECCIÓN MULTIPLE DE ELEMENTOS

Vamos a visualizar en una práctica cómo podemos recuperar múltiples Elementos de un <select> con selección múltiple

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

## Selección Múltiple Forms

### Elemento 1 Elemento 5



#### SELECCIONMULTIPLE.JS

```

import React, { Component } from 'react'
export default class SeleccionMultiple extends Component {
  selectMultiple = React.createRef();
  state = {
    seleccionados: ""
  }
  mostrarMultiples = (e) => {
    e.preventDefault();
    //DENTRO DE UN SELECT MULTIPLE TENEMOS UN ARRAY CON TODAS LOS
    //OPTIONS
    let options = this.selectMultiple.current.options;
    let datos = "";
    //CADA OBJETO OPTION DENTRO DEL ARRAY CONTIENE UN ATRIBUTO PARA
    //SABER SI ESTA SELECCIONADO: selected
    //value, text
    for (var option of options){
      if (option.selected == true){
        datos += option.value + " ";
      }
    }
    //MODIFICAMOS EL STATE DEL RENDER()
    this.setState({
      seleccionados: datos
    })
  }
  render() {
    return (
      <div>
        <h1>Selección Múltiple Forms</h1>
        <h2 style={{color:"blue"}}>{this.state.seleccionados}</h2>
        <form onSubmit={this.mostrarMultiples}>
          <select size="10" multiple ref={this.selectMultiple}>
            <option>Elemento 1</option>
            <option>Elemento 2</option>
            <option>Elemento 3</option>
            <option>Elemento 4</option>
            <option>Elemento 5</option>
            <option>Elemento 6</option>
            <option>Elemento 7</option>
          </select>
          <button>
            Mostrar seleccionados
          </button>
        </form>
      </div>
    )
  }
}

```

#### CICLO DE VIDA DE LA PAGINA

Mientras os voy contando algo de teoría, creamos un nuevo Proyecto llamado **reactservicios**

El ciclo de vida de una página contiene elementos dinámicos y estáticos. Por ejemplo, el **render()** es un elemento dinámico que puede cambiar

Por el state de la página.

En cambio, una variable declarada a nivel de Clase es estática para el dibujo.

Como hemos podido comprobar, podemos realizar dibujos dinámicos, pero NO tenemos un control absoluto de "CUANDO" poder hacer determinadas Acciones.

Por ejemplo, al cargar el Component hacer algo y SOLO AL CARGAR no al Cambiar el render()

Tenemos una serie de métodos en el Component (ya definidos) para Capturar los diferentes ciclos de vida del Component.

- componentDidMount(): Se ejecuta ANTES de dibujar el component
- componentWillMount(): Se ejecuta DESPUES del dibujo
- componentDidUpdate(): Se ejecuta cada vez que cambia el **render()**
- componentWillUnmount(): Se ejecuta después de eliminar el component

También tenemos un constructor, que es el primer método que se Ejecuta en cualquier clase Component. Nos permitiría inicializar variables. Por ejemplo, ahora mismo tenemos Nuestra variables por ahí tiradas.

Se suelen inicializar, por organización más bien, las asignaciones De los elementos de nuestra App dentro de un constructor

```
class Component {  
    constructor(props) {  
        //PODEMOS HACER CUALQUIER CODIGO LOGICO AQUÍ  
        state = { variable: props.ALGO }  
    }  
}
```

## SERVICIOS REACT

Los servicios api/restful ya sabemos lo que son. Pueden tener formato

Tanto XML/JSON

El acceso a servicios dentro de React NO es nativo, por lo que necesitamos

Alguna librería para poder llamar a los servicios Api

Hasta podríamos incluir la librería de Jquery y llamarlo con dicha librería.

Tenemos múltiples librerías de lectura de servicios Api

En nuestro caso vamos a utilizar una librería llamada **axios**

Esta librería nos permite leer tanto XML como JSON de una forma sencilla.

Las peticiones de cualquier librería de acceso a Api son asíncronas.

Dentro de los Frameworks se utilizan **promesas** para poder leer los

Servicios.

Una promesa significa asociar una respuesta al método de la llamada.

```
import axios from 'axios';  
  
axios.get(URL).then((response) => {  
    //DENTRO DE response TENEMOS LA INFORMACION  
    //MEDIANTE UNA VARIABLE LLAMADA data  
    //QUE CONTIENE LA INFORMACION DEL get  
})
```

Lo primero que vamos a realizar es agregar la librería axios dentro de

Nuestro proyecto mediante **npm**

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

Vamos a comenzar leyendo un clásico

<https://northwind.netcore.io/>

# Servicio Api Customers

Cargar clientes Api

Maria Anders

Ana Trujillo

Antonio Moreno

Thomas Hardy

Creamos un nuevo componente llamado **ServicioCustomers.js**

SERVICIOPRODUCTOS.JS

```
import React, { Component } from 'react'
//LO PRIMERO ES UTILIZAR LA LIBRERIA DE axios
import axios from 'axios';
export default class ServicioCustomers extends Component {
    //NECESITAMOS LA URI Y EL REQUEST DE ACCESO AL SERVICIO API
    urlCustomers = "https://northwind.netcore.io/customers.json";
    //NECESITAMOS UNA VARIABLE EN state PARA ALMACENAR
    //LOS CLIENTES
    state = {
        customers: []
    }
    //NECESITAREMOS RECUPERAR LOS CLIENTES CON axios
    //LA PREGUNTA ES CUANDO QUEREMOS HACERLO???
    loadCustomers = () => {
        console.log("Antes del servicio");
        axios.get(this.urlCustomers).then(response => {
            console.log("Leyendo servicio");
            this.setState({
                customers: response.data.results
            })
        })
        console.log("Despues del servicio");
    }
    //VAMOS A CARGAR LOS DATOS EN EL METODO INICIAL DE LA
    //PAGINA
    //LO HAREMOS SOLAMENTE UNA VEZ QUE SERA AL INICIAR EL COMPONENTE
    componentDidMount = () => {
        console.log("Creando component");
        this.loadCustomers();
    }
    render() {
        return (
            <div>
                <h1>Servicio Api Customers</h1>
                <button onClick={this.loadCustomers}>
                    Cargar clientes Api
                </button>
                {
                    this.state.customers.map((cliente, index) => {
                        return (<h3 style={{color:"blue"}} key={index}>
                            {cliente.contactName}
                        </h3>)
                    })
                }
            </div>
        )
    }
}
```

Como podemos comprobar, las peticiones que ponemos al inicio  
De cualquier Component se ejecutan dos veces.

# Servicio Api Customers

Cargar clientes Api

Esta funcionalidad sucede solamente en nuestro servidor Developer.  
Si publicamos nuestra App en un server no se ejecutan dos veces.  
Sirve para los testing de desarrollo.

Dentro de **index.js** tenemos las etiquetas **React.StrictMode** que realizan  
La petición dos veces

```
root.render(  
  <React.StrictMode>  
    <ServicioCustomers />  
  </React.StrictMode>  
)
```

Quitando dichas etiquetas ya podemos jugar

```
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(  
  <ServicioCustomers />  
)
```

## VARIABLES GLOBALES EN REACT

Dentro de la tecnología React no existe el concepto de variable Global,  
Es decir, tener una variable en todo el proyecto y acceder a ella de forma  
Nativa.  
En React, las variables solamente existen en cada Component.  
No hay ninguna definición establecida para variables globales.

Una variable global nos puede servir para compartir datos entre  
Diferentes components.

En nuestro caso nos puede venir bien para tener las URLs centralizadas  
Y realizar simplemente los Request.

Para declarar variables Globales debemos hacerlo en una clase que  
NO SEA UN COMPONENT

Dicho fichero lo almacenaremos dentro de **src**

Ejemplo sintaxis **Global.js**:

```
var Global = {  
  //INCLUIREMOS TODAS LAS VARIABLES/OBJETOS DESEEMOS  
  urlApi: "https://miservicio.com...",  
  contador: 5  
}  
export default Global;
```

Sobre la carpeta **src** agregamos una nueva clase llamada **Global.js**

## GLOBAL.JS

```
var Global = {  
  urlApiCustomers: "https://northwind.netcore.io/"  
}  
  
export default Global;
```

Modificamos el ejemplo de **ServiceCustomers.js** para aplicar la variable  
Global y solamente utilizar **request**

## SERVICECUSTOMERS.JS

Download the React DevTools  
<https://reactjs.org/link/react-devtools>

Creando component

Creando component

>

```

import Global from '../Global';

export default class ServicioCustomers extends Component {
    //NECESITAMOS LA URL Y EL REQUEST DE ACCESO AL SERVICIO API
    urlCustomers = Global.urlApiCustomers;

    loadCustomers = () => {
        console.log("Antes del servicio");
        let request = "customers.json";
        axios.get(this.urlCustomers + request).then(response => {
            console.log("Leyendo servicio");
        })
    }
}

```

Vamos a realizar un buscador de Clientes por su ID  
Tendremos una caja de texto dónde pondremos el ID del Customer y  
Mostraremos sus datos con un botón.

Creamos un nuevo component llamado **BuscadorCustomer.js**

## Buscador Api Customer

ID Customer:  Buscar Customer

- Cliente: Maria Anders
- Empresa: Alfreds Futterkiste
- Puesto: Sales Representative
- Ciudad: Berlin

### BUSCADORCUSTOMER.JS

```

import React, { Component } from 'react';
import axios from 'axios';
import Global from '../Global';
export default class BuscadorCustomer extends Component {
    urlApi = Global.urlApiCustomers;
    cajaId = React.createRef();
    state = {
        customer: null
    }
    buscarCustomer = (e) => {
        e.preventDefault();
        //RECUPERAMOS EL VALOR DE LA CAJA
        let idCustomer = this.cajaId.current.value;
        //customers/ALFKI.json
        let request = "customers/" + idCustomer + ".json";
        axios.get(this.urlApi + request).then(response => {
            console.log("Leyendo servicio");
            this.setState({
                customer: response.data.customer
            })
        })
    }
    render() {
        return (
            <div>
                <h1>Buscador Api Customer</h1>
                <form>
                    <label>ID Customer:</label>
                    <input type="text" ref={this.cajaId}/>
                    <button onClick={this.buscarCustomer}>
                        Buscar Customer
                    </button>
                </form>
                {
                    this.state.customer &&
                    <ul>
                        <li>Cliente: {this.state.customer.contactName}</li>
                        <li>Empresa: {this.state.customer.companyName}</li>
                        <li>Puesto: {this.state.customer.contactTitle}</li>
                        <li>Ciudad: {this.state.customer.city}</li>
                    </ul>
                }
            </div>
        )
    }
}

```

### PRACTICA SERVICIOS API

Realizar una práctica para buscar coches por **Marca** dentro de un Servicio API

Al iniciar la página, mostraremos todos los coches.

Al pulsar sobre un botón de **Filtrar**, mostraremos los coches que hemos filtrado  
Por su Marca mediante una caja de texto.

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

## BuscadorCoches

Introduzca marca <input type="text" value="BATMOVIL"/>		Buscar coches
Coche	Conductor	Imagen
BATMOVIL FIREBALL	BRUCE WAYNE	
BATMOVIL DARK KNIGHT	BRUCE WAYNE	

BUSCADORCOCHES.JS

```
import React, { Component } from 'react'
import axios from 'axios';
import Global from '../Global';
export default class BuscadorCoches extends
Component {
  cajaMarca = React.createRef();
  buscarCoches = (e) => {
    e.preventDefault();
    let marca = this.cajaMarca.current.value;
    // var cochesFiltrados = [];
    // for (var car of this.state.coches){
    //   if (marca == car.marca){
    //     //TENEMOS COCHE FILTRADO
    //     cochesFiltrados.push(car);
    //     console.log(car);
    //   }
    // }
    // // TENEMOS UN METODO DENTRO DE ARRAY QUE
    // NOS PERMITE BUSCAR POR ALGUNA
    // //PROPIEDAD DE LOS OBJETOS UTILIZANDO
    LAMBDA
    //Array.filter(objetoArray =>
objetoArray.Propiedad == valor);
    var cochesFiltrados =
      this.cochesAll.filter
      (car =>
car.marca.toLowerCase().includes(marca.toLowerCase()));
    this.setState({
      cochesDibujo: cochesFiltrados
    })
  }
  //TODOS LOS COCHES AL LEER DEL SERVICIO
  cochesAll = [];
  state = {
    cochesDibujo: []
  }
  reloadDibujo = (e) => {
    e.preventDefault();
    this.setState({
      cochesDibujo: this.cochesAll
    })
  }
}
```

```

        })
    }
loadCoches = () => {
    var request = "/webresources/coches";
    var url = Global.urlApiCoches + request;
    axios.get(url).then(response => {
        this.cochesAll = response.data;
        this.setState({
            cochesDibujo: this.cochesAll
        })
    })
}
componentDidMount = () => {
    this.loadCoches();
}
render() {
    return (
        <div>
            <h1>Buscador Coches</h1>
            <form>
                <label>Introduzca marca</label>
                <input type="text"
ref={this.cajaMarca}/>
                <button onClick={this.buscarCoches}>
                    Buscar coches
                </button>
                <button onClick={this.reloadDibujo}>
                    Recargar coches
                </button>
            </form>
            <table border="1">
                <thead>
                    <tr>
                        <th>Coches</th>
                        <th>Conductor</th>
                        <th>Imagen</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        this.state.cochesDibujo.map((
car, index) => {
                            return (<tr key={index}>
                                <td>{car.marca}
{car.modelo}</td>
                                <td>{car.conductor}
</td>
                                <td>
                                    <img
src={car.imagen}
style={{width:
"150px", height: "150px"}}/>
                                </td>
                            </tr>
                        ))
                    }
                </tbody>
            </table>
        </div>
    )
}
}

```

Vamos a realizar una práctica para mostrar departamentos y empleados.

Al cargar la página, tendremos un desplegable con los departamentos.

Al seleccionar un departamento determinado, mostraremos los empleados.

<https://apidepartamentospgs.azurewebsites.net/api/departamentos>

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

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

Creamos un nuevo component llamado **DepartamentosEmpleados.js**

## Departamentos Empleados

Seleccione un departamento

- CASALES - EMPLEADO
- ALCALA - EMPLEADO
- CEREZO - DIRECTOR
- REY - PRESIDENTE
- MUÑOZ - EMPLEADO

### DEPARTAMENTOSEMPLEADOS.JS

```
import React, { Component } from 'react'
import axios from 'axios';
import Global from '../Global';
export default class DepartamentosEmpleados extends Component {
  selectDepartamento = React.createRef();
  buscarEmpleados = (e) => {
    e.preventDefault();
    let idDepartamento = this.selectDepartamento.current.value;
    let request = "api/empleados/empleadosdepartamento/" + idDepartamento;
    var url = Global.urlApiEmpleados + request;
    axios.get(url).then(response => {
      console.log(response.data);
      this.setState({
        empleados: response.data
      })
    })
  }
  state = {
    empleados: [],
    departamentos: []
  }
  loadDepartamentos = () =>{
    var request = "api/departamentos";
    var url = Global.urlApiDepartamentos + request;
    axios.get(url).then(response => {
      console.log(response.data);
      this.setState({
        departamentos: response.data
      })
    })
  }
  componentDidMount = () => {
    this.loadDepartamentos();
  }
  render() {
    return (
      <div>
        <h1>Departamentos Empleados</h1>
        <form>
          <label>Seleccione un departamento</label>
          <select ref={this.selectDepartamento}>
            {
              this.state.departamentos.map((departamento, index) => {
                return (<option key={index} value={departamento.Numero}>
                  {departamento.Nombre}
                </option>)
              })
            }
          </select>
          <button onClick={this.buscarEmpleados}>
            Buscar empleados
          </button>
        </form>
        <ul>
          {
            this.state.empleados.map((empleado, index) => {
              return (<li key={index}>
                {empleado.apellido} - {empleado.oficio}
              </li>)
            })
          }
        </ul>
      </div>
    )
  }
}
```

### PRACTICA EMPLEADOS OFICIOS

Realizar una práctica en la que cargaremos en un desplegable los oficios de los empleados  
Al seleccionar un determinado oficio (botón), mostraremos los empleados en una lista.

Creamos un component llamado **EmpleadosOficios.js**

# Empleados Oficios

Seleccione un oficio   Buscar empleados

- GUTIERREZ
- GIL
- FERNANDEZ
- SANTIUSTE

## EMPLEADOSOFICIOS.JS

```
import React, { Component } from 'react'
import axios from 'axios'
import Global from '../Global';
export default class EmpleadosOficios extends Component {
  selectOficios = React.createRef();
  state = {
    oficios: [],
    empleados: []
  }
  buscarEmpleados = (e) => {
    e.preventDefault();
    let oficioSeleccionado = this.selectOficios.current.value;
    var request = "api/empleados/getempleadosoficio/empleadosoficio/"
    + oficioSeleccionado;
    var url = Global.urlApiEmpleados + request;
    axios.get(url).then(response => {
      console.log(response.data);
      this.setState({
        empleados: response.data
      })
    })
  }
  loadOficios = () => {
    var request = "api/empleados/getOficios/oficios";
    var url = Global.urlApiEmpleados + request;
    axios.get(url).then(response => {
      //AQUI TENEIS QUE RECORRER LOS EMPLEADOS DE response.data
      //Y DE ALGUNA FORMA, QUEDARNOS SOLO CON LOS OFICIOS DIFERENTES (IF)
      this.setState({
        oficios: response.data
      })
    })
  }
  componentDidMount = () => {
    this.loadOficios();
  }
  render() {
    return (
      <div>
        <h1>Empleados Oficios</h1>
        <form>
          <label>Seleccione un oficio</label>
          <select ref={this.selectOficios}>
            {
              this.state.oficios.map((oficio, index) => {
                return (<option key={index}>{oficio}</option>)
              })
            }
          </select>
          <button onClick={this.buscarEmpleados}>
            Buscar empleados
          </button>
        </form>
        <ul>
          {
            this.state.empleados.map((empleado, index) => {
              return (<li key={index}>{empleado.apellido}</li>)
            })
          }
        </ul>
      </div>
    )
  }
}
```

## COMUNICACIÓN COMPONENTES CON SERVICIOS

Vamos a realizar una práctica que ya hemos hecho esta mañana, la de Empleados Y Departamentos.

- Cargaremos los departamentos en un desplegable.
- Al pulsar un botón, mostraremos los Empleados del departamento

Tendremos dos componentes separados, por un lado, un component  
<Departamento>

Seleccione un departamento   Buscar empleados

y, por otro lado, un component <Empleados iddepartamento="10"/>

- CASALES - EMPLEADO
- ALCALA - EMPLEADO
- CEREZO - DIRECTOR
- REY - PRESIDENTE
- MUÑOZ - EMPLEADO

Para esta nueva práctica, vamos a tener una carpeta llamada **MaestroDetalle** con los dos componentes que os acabo de comentar.

Utilizamos este servicio para buscar empleados por departamento

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

Como podemos comprobar, una vez montada la práctica, no refresca nuestro componente, es decir, no lee el método **componentDidMount()** porque solamente lo lee una vez.

Podemos visualizar también que el dibujo SI que cambia, es decir, recibir **props** lo estamos haciendo bien, pero comunicarnos con el componente para "refrescar" no lo estamos logrando.

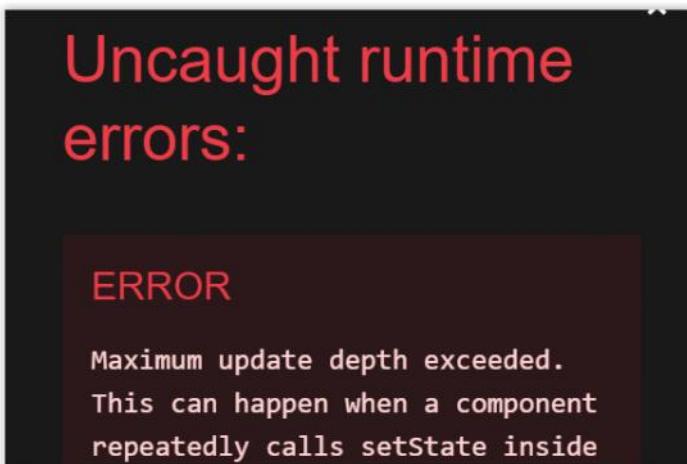
**Nota:** No llamaremos a los métodos del API hasta finalizar la práctica y entender el funcionamiento

Tenemos un método en el ciclo de vida que nos permite "capturar" el dibujo de la página cuando es cambiado. Dicho método se ejecuta cuando el componente ya se ha montado y cuando

**El render() es ejecutado.** El evento se llama **componentDidUpdate()**

Si realizamos el siguiente código, podremos visualizar cómo nos da una excepción

```
componentDidUpdate = () => {
  console.log("Dibujando componente " + this.props.iddepartamento);
  this.setState({
    ...state,
    texto: "Update: " + this.props.iddepartamento
  })
}
```



Estamos cambiando el **state** dentro de `componentDidUpdate`, lo que hace es ejecutar el `Render()` y volver a leer el dibujo de forma infinita.

En el método `componentDidUpdate` podemos recibir **props** y esas props que recibimos son las ANTERIORES que teníamos ANTES de cambiar el dibujo.

Mediante un **IF** podemos controlar el dibujo de nuestro componente

```

//RECIBIMOS LAS ANTIGUAS PROPS (10)
componentDidUpdate = (oldProps) => {
    //SI COMPARAMOS, PODEMOS ACTUALIZAR EL DIBUJO SOLAMENTE CUANDO
    //HA CAMBIADO props (20)
    console.log("Old props: " + oldProps.iddepartamento);
    console.log("Current props: " + this.props.iddepartamento);
    //SOLAMENTE ACTUALIZAREMOS CUANDO PROPS HA CAMBIADO DE VALOR
    if (oldProps.iddepartamento != this.props.iddepartamento){
        this.setState({
            texto: "Update: " + this.props.iddepartamento
        })
    }
}

```

#### DEPARTAMENTOS.JS

```

import React, { Component } from 'react'
import Empleados from './Empleados'
import axios from 'axios'
import Global from '../Global'
export default class Departamentos extends Component {
    selectDepartamentos = React.createRef();
    state = {
        departamentos: [],
        idDepartamento: 0
    }
    buscarEmpleados = (e) => {
        e.preventDefault();
        //CAPTURAMOS EL ID DEL DEPARTAMENTO
        let idDepartamento = this.selectDepartamentos.current.value;
        this.setState({
            idDepartamento: idDepartamento
        })
    }
    loadDepartamentos = () => {
        var request = "api/departamentos";
        var url = Global.urlApiDepartamentos + request;
        axios.get(url).then(response => {
            console.log(response.data);
            this.setState({
                departamentos: response.data
            })
        })
    }
    componentDidMount = () => {
        this.loadDepartamentos();
    }
    render() {
        return (
            <div>
                <h1>Departamentos Component</h1>
                <form>
                    <select ref={this.selectDepartamentos}>
                        {
                            this.state.departamentos.map((departamento, index) => {
                                return (<option key={index} value={departamento.Numero}>
                                    {departamento.Nombre}
                                </option>
                            ))
                        }
                    </select>
                    <button onClick={this.buscarEmpleados}>
                        Buscar empleados
                    </button>
                </form>
                <h2 style={{color:"blue"}}>Id departamento {this.state.idDepartamento}</h2>
            </div>
        )
    }
}

```

#### EMPLEADOS.JS

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from '../Global'
export default class Empleados extends Component {
    state = {
        empleados: [],
        texto: ""
    }
    loadEmpleados = () => {
        let idDepartamento = this.props.iddepartamento;
        var request = "api/empleados/empleadosdepartamento/" + idDepartamento;
        var url = Global.urlApiEmpleados2 + request;
        console.log("Props Id: " + this.props.iddepartamento);
        axios.get(url).then(response => {
            console.log(response.data);
            this.setState({

```

```

        empleados: response.data
    })
}
componentDidMount = () => {
    this.loadEmpleados();
}
//RECIBIMOS LAS ANTIGUAS PROPS (10)
componentDidUpdate = (oldProps) => {
    //SI COMPARAMOS, PODEMOS ACTUALIZAR EL DIBUJO SOLAMENTE CUANDO
    //HA CAMBIADO props (20)
    console.log("Old props: " + oldProps.iddepartamento);
    console.log("Current props: " + this.props.iddepartamento);
    //SOLAMENTE ACTUALIZAREMOS CUANDO PROPS HA CAMBIADO DE VALOR
    if (oldProps.iddepartamento != this.props.iddepartamento){
        this.loadEmpleados();
    }
}
render() {
    return (
        <div>
            <h3 style={{color:"red"}}>
                Empleados Component {this.props.iddepartamento}
            </h3>
            <table border="1">
                <thead>
                    <tr>
                        <th>Apellido</th>
                        <th>Oficio</th>
                        <th>Departamento</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        this.state.empleados.map((empleado, index) => {
                            return (<tr key={index}>
                                <td>{empleado.apellido}</td>
                                <td>{empleado.oficio}</td>
                                <td>{empleado.departamento}</td>
                            </tr>
                        ))
                    }
                </tbody>
            </table>
        </div>
    )
}

```

## RUTAS CON PARAMETROS

Instalamos **router-dom** en el proyecto.

**npm install --save react-router-dom**

Hemos trabajado con rutas sin parámetros dentro de los components

<https://localhost:3000/> --> HOME  
<https://localhost:3000/cine> --> CINE

```

export default class Router extends Component {
    render() {
        return (
            <BrowserRouter>
                <Routes>
                    <Route path="/" element={<Home/>}/>
                    <Route path="/cine" element={<Cine/>}/>
                    <Route path="/musica" element={<Musica/>}/>
                </Routes>
            </BrowserRouter>
        )
    }
}

```

Si necesitamos recibir algún parámetro dentro de la ruta, no sabemos hacerlo Todavía.

Por ejemplo, un Component <Cine/> que recibe, mediante **props** un valor:

<Cine idcine="12"/>

Si estamos dentro de una ruta, debemos indicar que, dicha ruta recibirá un **parámetro**

Necesitamos, en la ruta URL enviar un parámetro:

<https://localhost:3000/cine?idcine=12>

Los parámetros dentro de React NO se mandan como elementos "clásicos" en URL.

<https://localhost:3000/cine/12>

Dichos parámetros que nosotros enviamos deben ser representados con **DOS PUNTOS**

Dentro de la declaración de rutas en <Route/>

<Route path="/cine/:idcine" element={<Cine/>}/>

Podemos tener múltiples parámetros en la ruta:

<Route path="/cine/:idcine/:parametro2" element={<Cine/>}/>

<https://localhost:3000/cine/12/VALORPARAM2>

No podemos enviar NUNCA un **props** dentro de **element**

```
<Route path="/cine/:idcine" element={<Cine idcine="12"/>}/>
```

Para enviar datos a nuestro component, seguimos utilizando `<a href="/cine/12">Cine 12</a>`

Lo que nos interesa es recuperar el parámetro de **idcine** que viene con el valor de **12**

Y poder enviar ese parámetro a nuestro Component Cine `<Cine idcine="12"/>`

Para poder capturar el parámetro dentro de la ruta, tenemos que hacerlo en el Component **Router.js** mediante un objeto llamado **useParams**

Mediante **useParams** nos permitirá capturar los parámetros que vengan en una URL

Los métodos de captura de rutas vienen dentro del método **render()** del Component **Router.js** y van escritos con **function()**

```
function GetParametroComponentCine() {
    //CAPTURAMOS LA VARIABLE O VARIABLES DE RUTA
    //MEDIANTE DESESTRUCTURAR
    let { idcine, otroparametro } = useParams();
    //UNA VEZ QUE TENEMOS LOS VALORES DE LA RUTA
    //YA PODEMOS DIBUJAR LA ETIQUETA DEL COMPONENT CINE
    //CON SUS PROPS
    return <Cine idcine={idcine} otroprops={otroparametro}>
}
```

Para poder llamar a esta función, lo realizamos desde el propio elemento `<Route/>`

```
<Route path="/cine/:idcine/:otroparametro" element={<GetParametroComponentCine/>}>
```

No vamos a utilizar Servicios Api.

Vamos a realizar un clásico, como la Tabla de Multiplicar. Tendremos un component Llamado **TablaMultiplicar.js** que recibirá un número para mostrar su Tabla de multiplicar Mediante **props**

```
<TablaMultiplicar numero="4"/>
```

Comenzamos creando un nuevo component llamado **TablaMultiplicar.js**

**TABLAMULTIPLICAR.JS**

```
import React, { Component } from 'react'
export default class TablaMultiplicar extends Component {
    state = {
        tabla: []
    }
    generarTablaMultiplicar = () => {
        let aux = [];
        let num = parseInt(this.props.numero);
        for (var i = 1; i <= 10; i++){
            var operacion = num * i;
            aux.push(operacion);
        }
        this.setState({
            tabla: aux
        })
    }
    componentDidMount = () => {
        this.generarTablaMultiplicar();
    }
    render() {
        return (
            <div>
                <h1>Tabla Multiplicar Rutas</h1>
                <h3 style={{color:"blue"}}>
                    Número: {this.props.numero}
                </h3>
                <ul>
                    {
                        this.state.tabla.map((numero, index) => {
                            return (<li key={index}>{numero}</li>)
                        })
                    }
                </ul>
            </div>
        )
    }
}
```

Vamos a poner en funcionamiento el component con **Router.js**, por ahora estático.

Vamos a incluir un component llamado **Home.js** y otro component **NotFound.js**

Sobre la carpeta **Components** incluimos un nuevo component llamado **Router.js**

**ROUTER.JS**

```
import React, { Component } from 'react'
```

```

import { BrowserRouter, Route, Routes } from 'react-router-dom'
import TablaMultiplicar from './TablaMultiplicar'
import Home from './Home'
import NotFound from './NotFound'
export default class Router extends Component {
  render() {
    return (
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<Home/>}/>
          <Route path="/tabla"
            element={<TablaMultiplicar numero="19"/>}/>
          {/* PARA LAS RUTAS QUE NO EXISTEN DEBEMOS UTILIZAR
            UN ASTERISCO DENTRO DEL PATH Y DEBE SER
            LA ULTIMA ETIQUETA DE <Routes> */}
          <Route path="*" element={<NotFound/>}/>
        </Routes>
      </BrowserRouter>
    )
  }
}

```

Creamos un componente llamado **MenuRutas.js**

#### MENURUTAS.JS

```

import React, { Component } from 'react'
export default class MenuRutas extends Component {
  render() {
    return (
      <div>
        <ul id="menurutas">
          <li>
            <a href="/">Home</a> |
          </li>
          <li>
            <a href="/tabla">Tabla multiplicar</a> |
          </li>
          <li>
            <a href="/noexisto">Sin ruta</a> |
          </li>
        </ul>
      </div>
    )
  }
}

```

Incluimos dentro de **index.js** el menú y el Router

#### INDEX.JS

```

root.render(
  <div>
    <MenuRutas/>
    <Router/>
  </div>
);

```

Es el momento de centrarnos en los parámetros dinámicos en las rutas.

Abrimos **Router.js** para importar **useParams**

```
import { useParams } from 'react-router-dom'
```

Dentro de **render()** debemos incluir una función para capturar los parámetros DINAMICOS  
Con dichos parámetros, devolveremos el Component TablaMultiplicar con sus **props**

```

export default class Router extends Component {
  render() {
    function TablaMultiplicarElement() {
      //ESTA FUNCION NOS SERVIRA PARA CAPTURAR LOS
      //PARAMETROS EN UNA RUTA.
      //PARA SEPARAR PROPS DE PARAMS VOY A LLAMAR A NUESTRO
      //PARAMETRO EN RUTA minumero
      var { minumero } = useParams();
      //DEVOLVEMOS EL COMPONENT TABLA MULTIPLICAR CON SU PROPS
      //DE LA VARIABLE numero
      return <TablaMultiplicar numero={minumero}>
    }
  }
}

```

Modificamos la etiqueta **<Route/>** para enviar una parámetro llamado **minumero** y utilizar  
La función dentro de **element**

```
<Route path="/tabla/:minumero"
      element={<TablaMultiplicarElement/>}/>
```

Modificamos nuestro componente **MenuRutas.js** y le agregamos nuevas rutas con parámetros

#### MENURUTAS.JS

```
<ul id="menurutas">
  <li>
    <a href="/">Home</a> |
  </li>
  <li>
    <a href="/tabla/21">Tabla multiplicar 21</a> |
  </li>
  <li>
    <a href="/tabla/66">Tabla multiplicar 66</a> |
  </li>
  <li>
    <a href="/tabla/55">Tabla multiplicar 55</a> |
  </li>
  <li>
    <a href="/noexisto">Sin ruta</a> |
  </li>
```

Y ya podremos visualizar el resultado

[Home](#) | [Tabla multiplicar 21](#) | [Tabla multiplicar 66](#) | [Tabla multiplicar 55](#) | [Sin ruta](#) |

## Tabla Multiplicar Rutas

### Número: 21

- 21
- 42
- 63
- 84
- 105
- 126
- 147
- 168
- 189
- 210

#### PRACTICA CONJETURA DE COLLATZ

Necesito crear un proyecto nuevo llamado **practicacollatzreact** y lo que haremos será implementar La funcionalidad de navegación que acabamos de hacer en el proyecto de **reactservicios**

Necesitamos un Component Home, NotFound, Collatz y enviar parámetros dinámicos en ruta  
Al component Collatz.

El Component Collatz recibirá, mediante **props** un número.  
Tendremos un **MenuRutas.js** para pintar los links.

Todo número positivo siempre será 1 siguiendo estos dos pasos

- Si el número es par, dividimos entre 2
- Si el número es impar, multiplicamos por 3 y sumamos 1
  
- 6,3,10,16,8,4,2,1

PARA LA VUELTA, CREAR UN PROYECTO LLAMADO **reactminipracticalNICIALES**

La práctica finaliza a las **13:45**.

Si se termina antes, se avisa al Profesor para la corrección **ANTES DE DETENER EL VIDEO**

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

Equipos

Jugadores

## Mini Practica React

1. Al iniciar el Component, cargar los Equipos
2. Buscaremos jugadores por equipo seleccionado
3. Buscaremos jugadores por su nombre o parte de su nombre
4. No mostraremos la TABLA hasta que no tengamos JUGADORES

The screenshot shows a search interface with a text input 'Nombre jugador' containing 'Jude' and a button 'Buscar por NOMBRE'. Below it, a dropdown menu shows 'Real Madrid Club de Fútbol'. A table displays player data:

Imagen	Nombre	Posición	País	Fecha nacimiento
	Jude Bellingham	Mediapunta	Inglaterra	29/06/2003

### INCLUIR BOOTSTRAP EN PROYECTOS REACT

Para poder incluir Bootstrap tenemos dos formas posibles

#### 1) Objetos propios de React

HTML: <Button> Botón con diseño </Button>

```
import Button from 'react-bootstrap/Button';

function TypesExample() {
  return (
    <>
      <Button variant="primary">Primary</Button>{' '}
      <Button variant="secondary">Secondary</Button>{' '}
    </>
  );
}
```

#### 2) Trabajar con Bootstrap independiente al Framework

Instalar Bootstrap como hemos hecho en Jquery y utilizarlo de forma clásica  
<button className="primary">

Para utilizar bootstrap necesitamos ejecutar una serie de comandos desde **npm**

**npm install --save bootstrap**

**npm install --save jquery popper.js**

Una vez que tenemos las librerías, simplemente debemos hacer los correspondientes imports dentro de **index.js** para que sea aplicable a todas las páginas

#### INDEX.JS

```
import "bootstrap/dist/css/bootstrap.min.css";
import $ from 'jquery';
import Popper from 'popper.js';
import "bootstrap/dist/js/bootstrap.bundle";
```

Podemos visualizar la versión de Bootstrap que tenemos mirando dentro de **package.json**

```

index.js M package.json M TablaMultiplicar.js Router.js M Global.js M MenuRutas.js M NotFound.js M
package.json > {} dependencies
1 {
2   "name": "reactservicios",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "@testing-library/jest-dom": "^5.17.0",
7     "@testing-library/react": "^13.4.0",
8     "@testing-library/user-event": "^13.5.0",
9     "axios": "^1.7.7",
10    "bootstrap": "^5.3.3",
11    "jquery": "^3.7.1",
12    "popper.js": "^1.16.1",
13    "react": "^18.3.1",
14    "react-dom": "^18.3.1",
15    "react-router-dom": "^6.27.0",
16    "react-scripts": "5.0.1",
17    "web-vitals": "^2.1.4"
18  }
19

```

Ya podríamos aplicar los estilos dentro de nuestro proyecto

## RUTAS CON NavLink

Actualmente, tenemos un menú que nos ofrece las rutas con un número para cada Tabla de multiplicar.

Nuestra página NO funciona como debería.  
Si nos fijamos, estamos saltando una norma de SPA, que es recargar la página cada Vez que pulsamos sobre un Component.

Los enlaces a Components dentro de un **Router.js** se deben realizar con una etiqueta llamada **<NavLink>**

**<NavLink to="/">Ir a Home</NavLink>**

El siguiente paso es ir al código de **MenuRutas.js** y modificar los links.

## MENURUTAS.JS

```

<li>
  <NavLink to="/">Home</NavLink> |
</li>

```

Una vez que hemos cargado la página, podremos visualizar que nos ofrece un Error.



El Component **NavLink** solamente puede utilizarse dentro de **Router.js**

Dentro del component Router, tenemos una etiqueta llamada **<BrowserRouter>** donde Podemos dibujar nuestras etiquetas estáticas como, por ejemplo, un **<h1>**

Dentro del Router debemos incluir siempre nuestro **MenuRutas.js**

## ROUTER.JS

```

return (
  <BrowserRouter>
    <MenuRutas/>
      <Routes>

```

El siguiente problema tiene que ver con el propio dibujo de la tabla de multiplicar.  
 Nuestro componente ya no refresca la tabla, sino que, como el dibujo lo estamos  
 Realizando dentro de **componentDidMount** y el componente ya está dibujado, no  
 Entra en ese método. Debemos utilizar **componentDidUpdate()**

#### TABLA MULTIPLICAR

```

componentDidUpdate = (oldProps) => {
  if (oldProps.numero != this.props.numero){
    this.generarTablaMultiplicar();
  }
}

```

#### PRACTICA HOSPITALES Y DOCTORES

[Swagger UI \(apidoctoresroutes2023.azurewebsites.net\)](#)

[TODO supply a title \(apicrudhospital.azurewebsites.net\)](#)

Vamos a realizar una App que contendrá varias características.  
 Al iniciar nuestra App, cargaremos los Hospitales en un menú dinámico.  
 Tendremos un componente para dibujar los doctores que vengan con un hospital  
 Seleccionado.



The screenshot shows a table titled "Doctores del hospital:2". The table has columns for Apellido, Especial, Salario, and Id Hospital. There are four rows of data. A dropdown menu is open over the "Especial" column of the second row, listing "Gregorio marañon", "Provincial", "General", and "La Paz". "La Paz" is highlighted with a blue background. The table data is as follows:

Apellido	Especial	Salario	Id Hospital
Cabeza D.	Psiquiatr	152000	22
Best D.	Urología	225000	22
Galo D.	Pediatria	145222	22

Necesitamos una plantilla de bootstrap.

<https://getbootstrap.com/docs/4.0/examples/starter-template/>

Creamos un nuevo proyecto llamado **reactdoctoreshospital**

Instalamos todas las librerías necesarias

**npm install --save axios**

```
λ npm install --save react-router-dom
```

```
λ npm install --save bootstrap
```

```
λ npm install --save jquery popper.js
```

Hacemos las referencias a bootstrap dentro de **index.js**

#### INDEX.JS

```

import 'bootstrap/dist/css/bootstrap.min.css'
import $ from 'jquery'
import Popper from 'popper.js'
import 'bootstrap/dist/js/bootstrap.bundle'

```

#### GLOBAL.JS

```
var Global = {
```

```

        apiHospitales: "https://apicrudhospital.azurewebsites.net/",
        apiDoctores: "https://apidoctoreroutes2023.azurewebsites.net/"
    }
export default Global;

```

Creamos un component llamado **Home.js**

Vamos a comenzar creando un **Menú** para los hospitales.

Creamos un nuevo component llamado **MenuHospitales.js**

#### PLANTILLA MENU BOOTSTRAP 5.3

```

<nav className="navbar navbar-expand-sm navbar-dark bg-dark" aria-label="Third
navbar example">
  <div className="container-fluid">
    <a className="navbar-brand" href="#">Expand at sm</a>
    <button className="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarsExample03" aria-controls="navbarsExample03" aria-
expanded="false" aria-label="Toggle navigation">
      <span className="navbar-toggler-icon"></span>
    </button>
    <div className="collapse navbar-collapse" id="navbarsExample03">
      <ul className="navbar-nav me-auto mb-2 mb-sm-0">
        <li className="nav-item">
          <a className="nav-link active" aria-current="page" href="#">Home</a>
        </li>
        <li className="nav-item">
          <a className="nav-link" href="#">Link</a>
        </li>
        <li className="nav-item">
          <a className="nav-link disabled" aria-disabled="true">Disabled</a>
        </li>
        <li className="nav-item dropdown">
          <a className="nav-link dropdown-toggle" href="#" data-bs-
toggle="dropdown" aria-expanded="false">Dropdown</a>
          <ul className="dropdown-menu">
            <li><a className="dropdown-item" href="#">Action</a></li>
            <li><a className="dropdown-item" href="#">Another action</a></li>
            <li><a className="dropdown-item" href="#">Something else here</a>
          </ul>
        </li>
      </ul>
    </div>
  </div>
</nav>

```

#### MENUHOSPITALES.JS

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from './Global'
import { NavLink } from 'react-router-dom'
export default class MenuHospitales extends Component {
  state = {
    hospitales: []
  }
  loadHospitales = () => {
    var request = "webresources/hospitales";
    var url = Global.apiHospitales + request;
    axios.get(url).then(response => {
      console.log("Leyendo servicio...")
      this.setState({
        hospitales: response.data
      })
    })
  }
  componentDidMount = () => {
    this.loadHospitales();
  }
  render() {
    return (
      <div>
        <nav className="navbar navbar-expand-sm navbar-dark bg-dark" aria-label="Third
navbar example">
          <div className="container-fluid">
            <a className="navbar-brand" href="#">Expand at sm</a>
            <button className="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarsExample03" aria-controls="navbarsExample03" aria-
expanded="false" aria-label="Toggle navigation">
              <span className="navbar-toggler-icon"></span>
            </button>
            <div className="collapse navbar-collapse" id="navbarsExample03">
              <ul className="navbar-nav me-auto mb-2 mb-sm-0">
                <li className="nav-item">
                  <a className="nav-link active" aria-current="page" href="#">Home</a>
                </li>
                <li className="nav-item">
                  <a className="nav-link" href="#">Link</a>
                </li>
                <li className="nav-item">
                  <a className="nav-link disabled" aria-disabled="true">Disabled</a>
                </li>
                <li className="nav-item dropdown">
                  <a className="nav-link dropdown-toggle" href="#" data-bs-
toggle="dropdown" aria-expanded="false">Dropdown</a>
                  <ul className="dropdown-menu">
                    {
                      this.state.hospitales.map((hospital, index) => {
                        return (<li key={index}>
                          <NavLink
                            to={`/doctores/${ hospital.idhospital}`}
                            className="dropdown-item"
                            {hospital.nombre}
                          </NavLink>
                        </li>
                      ))
                    }
                  </ul>
                </li>
              </ul>
            </div>
          </div>
        </nav>
      </div>
    )
  }
}

```

```

        }
      </ul>
    </li>
  </ul>
</div>
</div>
</nav>
)
}
}

```

Creamos nuestro component Router.js

#### ROUTER.JS

```

import React, { Component } from 'react'
import { BrowserRouter, Routes, Route, useParams } from 'react-router-dom'
import MenuHospitales from './MenuHospitales'
import Home from './Home'
import Doctores from './Doctores';
export default class Router extends Component {
  render() {
    function DoctoresElement() {
      var {idhospital} = useParams();
      return <Doctores idhospital={idhospital}/>
    }
    return (
      <BrowserRouter>
        <MenuHospitales/>
        <Routes>
          <Route path="/" element={<Home/>}/>
          <Route path="/doctores/:idhospital"
            element={<DoctoresElement/>}/>
        </Routes>
      </BrowserRouter>
    )
  }
}

```

El siguiente paso será crearnos un nuevo component llamado Doctores.js

#### DOCTORES.JS

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from './Global'
export default class Doctores extends Component {
  state = {
    doctores: []
  }
  loadDoctores = () => {
    var idHospital = this.props.idhospital;
    var request = "api/doctores/doctoreshospital/" + idHospital;
    var url = Global.apiDoctores + request;
    axios.get(url).then(response => {
      console.log("Leyendo servicio doctores");
      this.setState({
        doctores: response.data
      })
    })
  }
  componentDidMount = () => {
    this.loadDoctores();
  }
  componentDidUpdate = (oldProps) => {
    //NUNCA LLAMAREMOS A NADA SI NO TENEMOS AQUI UN IF
    if (this.props.idhospital != oldProps.idhospital){
      this.loadDoctores();
    }
  }
  render() {
    return (
      <div>
        <h2>Doctores del hospital:
          <span style={{color: "red"}}>
            {this.props.idhospital}
          </span>
        </h2>
        <table className='table table-bordered'>
          <thead>
            <tr>
              <th>Apellido</th>
              <th>Especialidad</th>
              <th>Salario</th>
              <th>Id Hospital</th>
            </tr>
          </thead>
          <tbody>
            {
              this.state.doctores.map((doc, index) => {
                return (<tr key={index}>
                  <td>{doc.apellido}</td>
                  <td>{doc.especialidad}</td>
                  <td>{doc.salario}</td>
                  <td>{doc.idHospital}</td>
                </tr>
              })
            }
          </tbody>
        </table>
      </div>
    )
  }
}

```

Sobre el componente <Doctores/>

Necesito un botón para cada doctor y visualizar sus Detalles al apretar dicho botón.  
Los detalles del Doctor debemos verlos en <Doctores/>

Componente DetallesDoctor

The screenshot shows a dark-themed application interface. At the top, there is a navigation bar with a logo, followed by links for "Home", "Link", "Disabled", and "Hospitales". Below the navigation bar, the title "Doctores del hospital:18" is displayed. A table lists two doctors: "Miller G." and "Cajal R.". Each doctor entry includes a "Detalles" button. The "Cajal R." row is expanded, showing detailed information: "Cardiología", "151500", and "18".

Apellido	Detalles
Miller G.	<button>Detalles</button>
Cajal R.	<button>Detalles</button>

Cajal R.

Cardiología

151500

18

#### DETALLESDOCTOR.JS

```
import React, { Component } from 'react'
import Global from './Global'
import axios from 'axios'
export default class DetallesDoctor extends Component {
  state = {
    doctor: null
  }
  loadDoctor = () => {
    let idDoctor = this.props.iddoctor;
    let request = "api/doctores/" + idDoctor;
    let url = Global.apiDoctores + request;
    axios.get(url).then(response => {
      this.setState({
        doctor: response.data
      })
    })
  }
  componentDidMount = () => {
    this.loadDoctor();
  }
  componentDidUpdate = (oldProps) => {
    if (this.props.iddoctor != oldProps.iddoctor){
      this.loadDoctor();
    }
  }
  render() {
    return (
      <div>
        {
          this.state.doctor &&
          (<ul className='list-group'>
            <li className='list-group-item active'>
              {this.state.doctor.apellido}
            </li>
            <li className='list-group-item'>
              {this.state.doctor.especialidad}
            </li>
            <li className='list-group-item'>
              {this.state.doctor.salario}
            </li>
            <li className='list-group-item'>
              {this.state.doctor.idHospital}
            </li>
          </ul>
        )
      </div>
    )
  }
}
```

#### DOCTORES.JS

```
import React, { Component } from 'react'
import axios from 'axios'
import Global from './Global'
import DetallesDoctor from './DetallesDoctor'
export default class Doctores extends Component {
  state = {
    doctores: [],
    idDoctor: -1
  }
}
```

```

    }
    loadDoctores = () => {
      var idHospital = this.props.idhospital;
      var request = "api/doctores/doctoreshospital/" + idHospital;
      var url = Global.apiDoctores + request;
      axios.get(url).then(response => {
        console.log("Leyendo servicio doctores");
        this.setState({
          doctores: response.data,
          idDoctor: -1
        })
      })
    }
    componentDidMount = () => {
      this.loadDoctores();
    }
    componentDidUpdate = (oldProps) => {
      //NUNCA LLAMAREMOS A NADA SI NO TENEMOS AQUI UN IF
      if (this.props.idhospital != oldProps.idhospital){
        this.loadDoctores();
      }
    }
    mostrarDetalleDoctor = (idDoctor) => {
      this.setState({
        idDoctor: idDoctor
      })
    }
  }
  render() {
    return (
      <div>
        <h2>Doctores del hospital:</h2>
        <span style={{color: "red"}}>
          {this.props.idhospital}
        </span>
        </h2>
        <table className='table table-bordered'>
          <thead>
            <tr>
              <th>Apellido</th>
              <th>Detalles</th>
            </tr>
          </thead>
          <tbody>
            {
              this.state.doctores.map((doc, index) => {
                return (<tr key={index}>
                  <td>{doc.apellido}</td>
                  <td>
                    <button
                      onClick={() => {
                        this.mostrarDetalleDoctor(doc.idDoctor)
                      }}
                    >Detalles</button>
                  </td>
                </tr>
              ))
            }
          </tbody>
        </table>
      {
        this.state.idDoctor != -1 &&
        (<DetallesDoctor iddoctor={this.state.idDoctor}/>)
      }
    )
  }
}

```

#### POST CON AXIOS REACT

Necesitamos un formulario para los datos.

El JSON a enviar es el siguiente. Debe de ser exacto, es decir, los números debemos indicarlos para que no lleven Comilla.



```

JSON ▾
1  {
2    "idhospital": 14,
3    "nombre": "lunes",
4    "direccion": "Felipe X12",
5    "telefono": "55555",
6    "camas": 22
7  }

```

Axios trabaja directamente con el formato JSON, por lo que no necesitamos convertir Un Object a JSON, con declarar el objeto nos bastaría con la sintaxis de React

#### JQUERY

```

let departamento = new Object();
departamento.numero = 22;
departamento.nombre = "INFORMATICA";

let jsonDepartamento = JSON.stringify(departamento);

```

#### REACT

```

let departamento = {

```

```

        numero: 25,
        nombre: "I+D"
    }
}

```

El método **POST** de axios necesita dos parámetros:

- 1) URL de acceso al Servicio Api
- 2) Objeto a enviar al POST (data)

Comenzamos creando un nuevo componente llamado **CreateHospital.js**

#### **CREATEHOSPITAL.JS**

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from './Global'
export default class CreateHospital extends Component {
    cajaId = React.createRef();
    cajaNombre = React.createRef();
    cajaDireccion = React.createRef();
    cajaTelefono = React.createRef();
    cajaCamas = React.createRef();
    insertHospital = (e) => {
        e.preventDefault();
        let request = "webresources/hospitales/post";
        let url = Global.apiHospitales + request;
        //DEBEMOS RESPETAR LOS TIPOS DE DATO DEL SERVICIO
        let id = parseInt(this.cajaId.current.value);
        let nombre = this.cajaNombre.current.value;
        let direccion = this.cajaDireccion.current.value;
        let telefono = this.cajaTelefono.current.value;
        let camas = parseInt(this.cajaCamas.current.value);
        //NECESITAMOS UN OBJETO React CON EL MISMO NOMBRE DE
        //PROPIEDADES QUE EL SERVICIO.
        let hospital = {
            idhospital: id,
            nombre: nombre,
            direccion: direccion,
            telefono: telefono,
            camas: camas
        }
        axios.post(url, hospital).then(response => {
            this.setState({
                mensaje: "Hospital insertado: " + nombre
            })
        })
    }
    state = {
        mensaje: ""
    }
    render() {
        return (
            <div>
                <h1>New Hospital</h1>
                <h3 style={{color:"blue"}}>{this.state.mensaje}</h3>
                <form>
                    <label>Id hospital</label>
                    <input type="text" ref={this.cajaId} className='form-control' />
                    <label>Nombre</label>
                    <input type="text" ref={this.cajaNombre} className='form-control' />
                    <label>Dirección</label>
                    <input type="text" ref={this.cajaDireccion} className='form-control' />
                    <label>Teléfono</label>
                    <input type="text" ref={this.cajaTelefono} className='form-control' />
                    <label>Camas</label>
                    <input type="text" ref={this.cajaCamas} className='form-control' />
                    <button onClick={this.insertHospital} className='btn btn-warning'>
                        Insert
                    </button>
                </form>
            </div>
        )
    }
}

```

El siguiente paso lógico es crear un Component para mostrar los hospitales.

No lo vamos a llamar de forma explícita, sino que lo que haremos será **Navegar**

Cuando insertemos, iremos mediante código a otro **Component**

Para ello, se utiliza una etiqueta llamada **<Navigate>**

Creamos un nuevo componente llamado **Hospitales.js**

#### **HOSPITALES.JS**

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from './Global'
export default class Hospitales extends Component {
    state = {
        hospitales: []
    }
    loadHospitales = () => {
        let request = "webresources/hospitales";
        let url = Global.apiHospitales + request;
        axios.get(url).then(response => {
            this.setState({
                hospitales: response.data
            })
        })
    }
    componentDidMount = () => {
        this.loadHospitales();
    }
}

```

```

        }
        render() {
            return (
                <div>
                    <h1>Hospitales</h1>
                    <table className='table table-light'>
                        <thead>
                            <tr>
                                <th>Id</th>
                                <th>Nombre</th>
                                <th>Dirección</th>
                                <th>Teléfono</th>
                                <th>Camas</th>
                            </tr>
                        </thead>
                        <tbody>
                            {
                                this.state.hospitales.map((hospital, index) => {
                                    return (<tr key={index}>
                                        <td>{hospital.idhospital}</td>
                                        <td>{hospital.nombre}</td>
                                        <td>{hospital.direccion}</td>
                                        <td>{hospital.telefono}</td>
                                        <td>{hospital.camas}</td>
                                    </tr>
                                ))
                            }
                        </tbody>
                    </table>
                </div>
            )
        }
    }
}

```

La etiqueta <Navigate to="/" /> nos lleva a una URL que tengamos mapeada dentro De Router.js

Vamos a llevar a los hospitales después de Insertar.  
La etiqueta Navigate no se utiliza en los métodos, sino que se utiliza dentro del código Del render()

<Navigate to="/" />

Necesitamos en el código HTML saber cuándo ha insertado.  
Se suele utilizar una variable **status** para saber cuándo ha realizado la inserción.

Modificamos el código de CREATEHOSPITAL.JS

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from './Global'
import { Navigate } from 'react-router-dom';
export default class CreateHospital extends Component {
    cajaId = React.createRef();
    cajaNombre = React.createRef();
    cajaDireccion = React.createRef();
    cajaTelefono = React.createRef();
    cajaCamas = React.createRef();
    insertHospital = (e) => {
        e.preventDefault();
        let request = "webresources/hospitales/post";
        let url = Global.apiHospitales + request;
        //DEBEMOS RESPETAR LOS TIPOS DE DATO DEL SERVICIO
        let id = parseInt(this.cajaId.current.value);
        let nombre = this.cajaNombre.current.value;
        let direccion = this.cajaDireccion.current.value;
        let telefono = this.cajaTelefono.current.value;
        let camas = parseInt(this.cajaCamas.current.value);
        //NECESITAMOS UN OBJETO React CON EL MISMO NOMBRE DE
        //PROPIEDADES QUE EL SERVICIO.
        let hospital = {
            idhospital: id,
            nombre: nombre,
            dirección: dirección,
            telefono: telefono,
            camas: camas
        }
        axios.post(url, hospital).then(response => {
            this.setState({
                mensaje: "Hospital insertado: " + nombre,
                status: true
            })
        })
    }
    state = {
        mensaje: "",
        status: false
    }
    render() {
        return (
            <div>
                {
                    this.state.status == true &&
                    (<Navigate to="/hospitales"/>)
                }
                <h1>New Hospital</h1>
                <h3 style={{color:"blue"}}>{this.state.mensaje}</h3>
                <form>
                    <label>Id hospital</label>
                    <input type="text" ref={this.cajaId} className='form-control' />
                    <label>Nombre</label>
                    <input type="text" ref={this.cajaNombre} className='form-control' />
                    <label>Dirección</label>
                    <input type="text" ref={this.cajaDireccion} className='form-control' />
                    <label>Teléfono</label>
                </form>
            </div>
        )
    }
}

```

```

        <input type="text" ref={this.cajaTelefono} className='form-control'/>
        <label>Camas</label>
        <input type="text" ref={this.cajaCamas} className='form-control'/>
        <button onClick={this.insertHospital} className='btn btn-warning'>
          Insert
        </button>
      </form>
    </div>
  )
}

```

#### CRUD DEPARTAMENTOS

Vamos a realizar una aplicación React consumiendo un servicio de Departamentos  
Tendremos una lista de departamentos, insertar, modificar y eliminar departamentos.  
Aplicación con rutas y con bootstrap

Id departamento	Nombre	Localidad	Actions
10	DISNEYLAND	PARIS	<button>Delete</button> <button>Detalles</button> <button>Update</button>
20	PARQUE WARNER	MADRID	<button>Delete</button> <button>Detalles</button> <button>Update</button>
30	PORT AVENTURA	TABARNIA	<button>Delete</button> <button>Detalles</button> <button>Update</button>
40	TERRA MITICA	BENIDORM	<button>Delete</button> <button>Detalles</button> <button>Update</button>

Comenzamos creando nuestra nueva App llamada **reactcruddepartamentos**  
y le incluimos todas las librerías

```

npm install --save axios
npm install --save bootstrap
npm install --save react-router-dom
npm install --save jquery popper.js

```

Vamos a consumir esta URL

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

Abrimos el proyecto, creamos una carpeta llamada **components** dentro de **src**

Creamos otra carpeta llamada **assets/images** dentro de **src** y copiamos ahí una imagen de **loading**

Importamos las clases de diseño dentro de **index.js**

#### **INDEX.JS**

```

index.js M ×
src > index.js > ...
1 import 'bootstrap/dist/css/bootstrap.min.css'
2 import $ from 'jquery'
3 import Popper from 'popper.js'
4 import 'bootstrap/dist/js/bootstrap.bundle'
5

```

Creamos un component llamado **MenuDepartamentos.js** y otro llamado **HomeDepartamentos.js**

Creamos un component llamado **Router.js** y comprobamos que es funcional por ahora

#### **ROUTER.JS**

```

import React, { Component } from 'react'
import { BrowserRouter, Routes, Route } from 'react-router-dom'
import MenuDepartamentos from './MenuDepartamentos'
import HomeDepartamentos from './HomeDepartamentos'
export default class Router extends Component {
  render() {
    return (
      <BrowserRouter>
        <MenuDepartamentos/>
        <Routes>
          <Route path="/" element={<HomeDepartamentos/>}/>
        </Routes>
      </BrowserRouter>
    )
  }
}

```

```
    }
}
```

Sobre **index.js**, incluimos nuestro menú de rutas y quitamos `<React.StrictMode>`

Incluimos una nueva clase llamada **Global.js**

#### GLOBAL.JS

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

export default Global;
```

Sobre **HomeDepartamentos**, dibujaremos los departamentos en una tabla.

Mientras que los departamentos están siendo recuperados del servicio, vamos a dibujar La imagen de "Loading"...

Tenemos dos opciones para poder realizar esta acción:

##### 1) Código Render con sintaxis React:

```
render() {
  return (<div>
    {
      this.state.status == true ??
      (<h1>TRUE</h1>):
      (<h1>FALSE</h1>)
    }
  </div>
}
```

##### 2) Dentro del **render()** pero fuera del **return**. Este código es código JS puro, es decir, Un IF es como en la mayoría de lenguajes.

```
render() {
  if (this.state.status == true){
    return (<h1>TRUE</h1>)
  }else{
    return (<h1>FALSE</h1>)
  }
}
```

```
render() {
  if (this.state.status == false){
    return (<img src={loadingImage}/>)
  }else{
    return (
      <div>
        <h1>Home Departamentos</h1>
      </div>
    )
  }
}
```

#### HOMEDEPARTAMENTOS.JS

```
import React, { Component } from 'react'
import axios from 'axios'
import Global from './Global'
import loadingImage from '../assets/images/loading.gif'
export default class HomeDepartamentos extends Component {
  state = {
    status: false,
    departamentos: []
  }
  loadDepartamentos = () => {
    let request = "api/departamentos";
    let url = Global.apiUrlDepartamentos + request;
    axios.get(url).then(response => {
      console.log("Leyendo departamentos");
      this.setState({
        departamentos: response.data,
        status: true
      })
    })
  }
  componentDidMount = () => {
    this.loadDepartamentos();
  }
}
```

```

        }
        render() {
            if (this.state.status == false){
                return (<img src={loadingImage}/>)
            }else{
                return (
                    <div>
                        <h1>Home Departamentos</h1>
                        <table className='table table-warning'>
                            <thead>
                                <tr>
                                    <th>Id departamento</th>
                                    <th>Nombre</th>
                                    <th>Localidad</th>
                                </tr>
                            </thead>
                            <tbody>
                                {
                                    this.state.departamentos.map((departamento, index) => {
                                        return (<tr key={index}>
                                            <td>{departamento.numero}</td>
                                            <td>{departamento.nombre}</td>
                                            <td>{departamento.localidad}</td>
                                        </tr>
                                    ))
                                }
                            </tbody>
                        </table>
                    </div>
                )
            }
        }
    }
}

```

A continuación, creamos un nuevo componente llamado **CreateDepartamentos.js**  
Y lo ponemos dentro de **Router.js**

```

return (
    <BrowserRouter>
        <MenuDepartamentos/>
        <Routes>
            <Route path="/" element={<HomeDepartamentos/>}/>
            <Route path="/create" element={<CreateDepartamentos/>}/>
        </Routes>
    </BrowserRouter>
)

```

#### CRAEDEPARTAMENTOS

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from './Global'
import { Navigate } from 'react-router-dom'
export default class CreateDepartamentos extends Component {
    cajaId = React.createRef();
    cajaNombre = React.createRef();
    cajaLocalidad = React.createRef();
    state = {
        status: false
    }
    insertarDepartamento = (e) => {
        e.preventDefault();
        let id = parseInt(this.cajaId.current.value);
        let nombre = this.cajaNombre.current.value;
        let localidad = this.cajaLocalidad.current.value;
        let departamento = {
            numero: id,
            nombre: nombre,
            localidad: localidad
        }
        let request = "api/departamentos";
        let url = Global.apiUrlDepartamentos + request;
        axios.post(url, departamento).then(response => {
            console.log("Insertado");
            this.setState({
                status: true
            })
        })
    }
    render() {
        if (this.state.status == true){
            return (<Navigate to="/" />)
        }else{
            return (<div>
                <h1>Nuevo departamento</h1>
                <form>
                    <label>Id departamento</label>
                    <input type="text" ref={this.cajaId} className='form-control' />
                    <label>Nombre</label>
                    <input type="text" ref={this.cajaNombre} className='form-
control' />
                    <label>Localidad</label>
                    <input type="text" ref={this.cajaLocalidad} className='form-
control' />
                    <button className='btn btn-info' onClick={this.insertarDepartamento}>
                        Insertar departamento
                    </button>
                </form>
            </div>
        )
    }
}

```

```

        </div>
    }
}

```

Creamos un nuevo componente llamado **DetalleDepartamento.js** y recibiremos parámetros

Este componente recibirá datos mediante la navegación (**path**)

El componente también tendrá **props** para recibir el id

```
<DetalleDepartamento id="10"/>
```

```
PATH: <Route path="/detalles/:id" />
```

Lo primero que vamos a realizar será poner **<NavLink>** dentro de cada departamento  
Para poder visualizar sus detalles.

Dicho NavLink enviará los datos del ID del departamento dentro de su **Path**

```
<NavLink to="/detalles/10">
```

Modificamos el código de **HomeDepartamentos**

**HOMEDEPARTAMENTOS**

```

this.state.departamentos.map((departamento, index) => {
  return (<tr key={index}>
    <td>{departamento.numero}</td>
    <td>{departamento.nombre}</td>
    <td>{departamento.localidad}</td>
    <td>
      <NavLink to={"/detalles/" + departamento.numero}>
        Detalles
      </NavLink>
    </td>
  </tr>)
}

```

Tenemos que resolver dicha ruta dentro del Component **Router** y devolver el componente  
Detalles con **props**

```
<DetalleDepartamento id="10"/>
```

**ROUTER.JS**

```

import React, { Component } from 'react'
import { BrowserRouter, Routes, Route, useParams } from 'react-router-dom'
import MenuDepartamentos from './MenuDepartamentos'
import HomeDepartamentos from './HomeDepartamentos'
import CreateDepartamentos from './CreateDepartamentos'
import DetalleDepartamento from './DetalleDepartamento'
export default class Router extends Component {
  render() {
    function DetalleDepartamentoElement() {
      let {iddepartamento} = useParams();
      return (<DetalleDepartamento id={iddepartamento}/>)
    }
    return (
      <BrowserRouter>
        <MenuDepartamentos/>
        <Routes>
          <Route path="/" element={<HomeDepartamentos/>}/>
          <Route path="/create" element={<CreateDepartamentos/>}/>
          <Route path="/detalles/:iddepartamento"
            element={<DetalleDepartamentoElement/>}/>
        </Routes>
      </BrowserRouter>
    )
  }
}

```

**DETALLEDEPARTAMENTO.JS**

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from './Global'
import loadingImage from '../assets/images/loading.gif'
import { NavLink } from 'react-router-dom'
export default class DetalleDepartamento extends Component {
  state = {
    departamento: null
  }
  findDepartamento = () => {
    let request = "api/departamentos/" + this.props.id;
    let url = Global.apiUrlDepartamentos + request;
    axios.get(url).then(response => {
      console.log("Detalles departamento")
      this.setState({
        departamento: response.data
      })
    })
  }
}

```

```

componentDidMount = () => {
  this.findDepartamento();
}
render() {
  return (
    <div>
      <NavLink to="/">Back to List</NavLink>
      {
        this.state.departamento ?
          (<ul className='list-group'>
            <li className='list-group-item'>
              Número: {this.state.departamento.numero}
            </li>
            <li className='list-group-item'>
              Nombre: {this.state.departamento.nombre}
            </li>
            <li className='list-group-item'>
              Localidad: {this.state.departamento.localidad}
            </li>
          </ul>) :
          (<img src={loadingImage}/>)
      )
    </div>
  )
}

```

El siguiente componente que vamos a realizar es modificar.

Para modificar, vamos a enviar los parámetros del departamento mediante **props**  
Y con rutas en su **path**

Comenzamos creando un nuevo component llamado **UpdateDepartamento.js**

Recibiremos mediante **props** todos los datos del departamento.

<UpdateDepartamento id="10" nombre="CONTABILIDAD" localidad="ALICANTE"/>

Path: <Route path="/update/:id/:nombre/:localidad" />

Modificamos el código del component **HomeDepartamentos.js**

**HOMEDEPARTAMENTOS.JS**

```

</NavLink>
<NavLink
  to={"/update/" + departamento.numero
  + "/" + departamento.nombre + "/" + departamento.localidad} className="btn btn-success">
  Update
</NavLink>

```

Debemos modificar **Router.js** para recibir los parámetros mediante un **path**

**ROUTER.JS**

```

<Route path="/update/:id/:nombre/:localidad"
  element={<UpdateDepartamentoElement/>}/>

```

```

function UpdateDepartamentoElement() {
  let { id, nombre, localidad } = useParams();
  return (<UpdateDepartamento id={id} nombre={nombre} localidad={localidad}>)
}

```

Dentro de React, cuando escribimos sobre una caja algún valor dinámico para Poder editarlo, no podemos utilizar **value** ya que lo convierte en **Read Only**

En lugar de utilizar **value**, se utiliza **defaultValue**

**UPDATEDEPARTAMENTO.JS**

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from './Global'
import { Navigate, NavLink } from 'react-router-dom'
export default class UpdateDepartamento extends Component {
  cajaId = React.createRef();
  cajaNombre = React.createRef();
  cajaLocalidad = React.createRef();
  updateDepartamento = (e) => {
    e.preventDefault();
    let id = parseInt(this.cajaId.current.value);
    let nombre = this.cajaNombre.current.value;
    let localidad = this.cajaLocalidad.current.value;
    let departamento = {
      numero: id,
      nombre: nombre,
      localidad: localidad
    }
    let request = "api/departamentos";

```

```

let url = Global.apiUrlDepartamentos + request;
axios.put(url, departamento).then(response => {
  console.log("Update...");
  this.setState({
    status: true
  })
})
state = {
  status: false
}
render() {
  return (
    <div>
      {
        this.state.status == true &&
        (<Navigate to="/" />)
      }
      <h1>Update Departamento</h1>
      <NavLink to="/">Back to Index</NavLink>
      <form>
        <label>Id</label>
        <input type="text" ref={this.cajaId} defaultValue={this.props.id}
          disabled
          className='form-control' />
        <label>Nombre</label>
        <input type="text" ref={this.cajaNombre}
          defaultValue={this.props.nombre}
          className='form-control' />
        <label>Localidad</label>
        <input type="text" ref={this.cajaLocalidad}
          defaultValue={this.props.localidad}
          className='form-control' />
        <button onClick={this.updateDepartamento} className='btn btn-dark'>
          Update
        </button>
      </form>
    </div>
  )
}
}

```

El último paso es realizar un Delete sobre departamentos.

Podemos realizarlo de múltiples formas:

- Component que nos pida confirmación.
- Incluir la acción dentro de la página **HomeDepartamentos**

Vamos a realizar la funcionalidad sin rutas.

Haremos la acción dentro del **HomeDepartamentos**

Utilizaremos un botón para realizar la acción del Delete

#### HOMEDEPARTAMENTOS

```

<button className='btn btn-danger'
  onClick={() => {
    this.deleteDepartamento(departamento.numero);
  }}>
  Delete
</button>

```

```

deleteDepartamento = (idDepartamento) => {
  let request = "api/departamentos/" + idDepartamento;
  let url = Global.apiUrlDepartamentos + request;
  axios.delete(url).then(response => {
    console.log("Delete...");
    this.loadDepartamentos();
  })
}

```

Quiero hacer eliminar, pero desde otro Component.  
Debemos hacerlo mediante Routes y parámetros.

A ser posible, que me pida confirmación.

Abrimos INSOMNIA para probar una nueva petición.

Utilizaremos el siguiente servicio

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

Vamos a probar la siguiente petición

**GET** /api/Trabajadores/TrabajadoresHospitales Obtiene el conjunto de Trabajadores de varios hospitales.

Esta petición enviará múltiples ids de Hospital al servicio y devolverá los trabajadores que tengan dichos Ids de Hospital

Para poder enviar información a este servicio, debemos construir nuestro Request

```
https://apiejemplos.azurewebsites.net/api/trabajadores/trabajadoreshospitales?idhospital=17&idhospital=22
```

#### TRABAJADORES CON HOSPITALES MULTIPLES

Vamos a realizar una práctica en la que construiremos la petición para mostrar los trabajadores de un Hospital, mediante la selección múltiple de un <select>

Tendremos dos componentes: **Trabajadores y Hospitales**

```
<Trabajadores idhospitales={hospitalesSeleccionados}/>
```

- 1) Cargaremos los Hospitales en una lista de selección múltiple
- 2) Un componente que nos permitirá cargar los trabajadores de los hospitales seleccionados

Utilizaremos el proyecto **reactservicios**

Comenzaremos incluyendo, dentro de **Global** nuestra URL para acceso al servicio

#### GLOBAL.JS

```
var Global = {  
    urlApiCustomers: "https://northwind.netcore.io/",  
    urlApiCoches: "https://apicochespaco.azurewebsites.net/",  
    urlApiDepartamentos: "https://apidepartamentospgs.azurewebsites.net/",  
    urlApiEmpleados: "https://apiempleadosaction.azurewebsites.net/",  
    urlApiEmpleados2: "https://apiempleadosspgs.azurewebsites.net/",  
    urlPractica: "https://apiejemplos.azurewebsites.net/",  
    urlEjemplos: "https://apiejemplos.azurewebsites.net/"  
}  
  
export default Global;
```

Comenzamos creando un componente llamado **Trabajadores** y otro componente llamado **HospitalesMultiple**

## Hospitales Múltiple

Servicio Cloud  
Provincial Cloud  
General Cloud  
La Paz Cloud  
San Carlos Cloud

Mostrar trabajadores

## Trabajadores

López A., 19

Miller G., 18

Cajal R., 18

Hernández J., 19

#### HOSPITALESMULTIPLE

```
import React, { Component } from 'react'  
import Trabajadores from './Trabajadores'  
import axios from 'axios'  
import Global from '../Global'  
export default class HospitalesMultiple extends Component {  
    selectHospital = React.createRef();  
    state = {  
        hospitales: [],  
        hospitalesSeleccionados: []  
    }
```

```

loadHospitales = () => {
  let request = "api/hospitales";
  let url = Global.urlEjemplos + request;
  axios.get(url).then(response => {
    console.log("Leyendo hospitales");
    this.setState({
      hospitales: response.data
    })
  })
}
componentDidMount = () => {
  this.loadHospitales();
}
getHospitalesSeleccionados = (e) => {
  e.preventDefault();
  let aux = [];
  let options = this.selectHospital.current.options;
  for (var opt of options){
    if (opt.selected == true){
      aux.push(opt.value);
    }
  }
  this.setState({
    hospitalesSeleccionados: aux
  })
}
render() {
  return (
    <div>
      <h1>Hospitales Múltiple</h1>
      <form>
        <select ref={this.selectHospital} className='form-control'
          size="8" multiple>
          {
            this.state.hospitales.map((hospital, index) => {
              return (<option key={index}
                value={hospital.idHospital}>
                  {hospital.nombre}
                </option>
              )
            })
          }
        </select>
        <button onClick={this.getHospitalesSeleccionados} className='btn btn-
info'>
          Mostrar trabajadores
        </button>
      </form>
      {
        this.state.hospitalesSeleccionados.length != 0 &&
        (<Trabajadores idhospitales={this.state.hospitalesSeleccionados}/>)
      }
    </div>
  )
}

```

## TRABAJADORES

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from '../Global'
export default class Trabajadores extends Component {
  state = {
    trabajadores: [],
    mensaje: ""
  }
  loadTrabajadores = () => {
    //RECUPERAR TODOS LOS IDS DE HOSPITALES
    let idsHospitales = this.props.idhospitales;
    if (idsHospitales.length != 0){
      //idhospital=17&idhospital=22&idhospital=14
      let data = "";
      for (var id of idsHospitales){
        data += "idhospital=" + id + "&";
      }
      //ELIMINAMOS EL ULTIMO CARACTER DEL STRING
      //idhospital=25&idhospital=22&
      data = data.substring(0, data.length - 1);
      this.setState({
        mensaje: data
      })
      //PODEMOS REALIZAR LA PETICION AL SERVICIO
      let request = "api/trabajadores/trabajadoresthospitales?" + data;
      let url = Global.urlEjemplos + request;
      axios.get(url).then(response => {
        console.log("Leyendo Trabajadores");
        this.setState({
          trabajadores: response.data
        })
      })
    }
  }
  componentDidMount = () => {
    this.loadTrabajadores();
  }
  componentDidUpdate = (oldProps) => {
    if (oldProps.idhospitales != this.props.idhospitales){
      this.loadTrabajadores();
    }
  }
  render() {
    return (
      <div>
        <h1 style={{color:"fuchsia"}}>Trabajadores</h1>
        <ul className='list-group'>
          {
            this.state.trabajadores.map((trabajador, index) => {
              return (<li key={index} className='list-group-item'>
                {trabajador.apellido}, {trabajador.idHospital}
              )
            })
          }
        </ul>
      </div>
    )
  }
}

```

```

        </li>
    })
}
</ul>
<h5 style={{color:"blue"}}>
    {this.state.mensaje}
</h5>
</div>
)
}
}

```

VERSION 2: Necesito poder incrementar el salario de los trabajadores por hospital Seleccionados.

Cambiamos el dibujo a una tabla para poder ver los cambios.

**PUT**

/api/Trabajadores/UpdateSalarioTrabajadoresHospitales Incrementa el salario de Trabajadores por múltiples HOSPITALES.

Petición PUT

#### URL PREVIEW

https://apiejemplos.azurewebsites.net/api/trabajadore s/updateSalariotrabajadoreshospitales?incremento=1 &idhospital=17&idhospital=22



## Hospitales Múltiple

Servicio Cloud

Provincial Cloud

General Cloud

La Paz Cloud

San Carlos Cloud

Incremento salarial

1

Incrementar salarios

Mostrar trabajadores

## Trabajadores

Apellido	Salario	Hospital
López A.	427177	19
Miller G.	652698	18

#### HOSPITALESMULTIPLES.JS

```

import React, { Component } from 'react'
import Trabajadores from './Trabajadores'
import axios from 'axios'
import Global from '../Global'
export default class HospitalesMultiple extends Component {
    selectHospital = React.createRef();
    cajaIncremento = React.createRef();
    state = {
        hospitales: [],
        hospitalesSeleccionados: []
    }
    loadHospitales = () => {
        let request = "api/hospitales";
        let url = Global.urlEjemplos + request;
        axios.get(url).then(response => {
            console.log("Leyendo hospitales");
            this.setState({
                hospitales: response.data
            })
        })
    }
    componentDidMount = () => {
        this.loadHospitales();
    }
    getHospitalesSeleccionados = (e) => {
        e.preventDefault();
        let aux = this.getSeleccion();
        this.setState({
            hospitalesSeleccionados: aux
        })
    }
    getSeleccion = () => {
        let aux = [];
        let options = this.selectHospital.current.options;
        for (var opt of options){

```

```

        if (opt.selected == true){
            aux.push(opt.value);
        }
    }
    return aux;
}
incrementarSalarios = (e) => {
    e.preventDefault();
    let hospitales = this.getSeleccion();
    let incremento = parseInt(this.cajaIncremento.current.value);
    let data = "";
    for (var id of hospitales){
        data += "idhospital=" + id + "&";
    }
    //ELIMINAMOS EL ULTIMO CARACTER DEL STRING
    //idhospital=25&idhospital=22
    data = data.substring(0, data.length - 1);
    let request = "api/trabajadores/updatestrialotrabajadoreshospitales?";
    incremento=""
    + incremento + "&" + data;
    let url = Global.urlEjemplos + request;
    axios.put(url).then(response => {
        this.setState({
            hospitalesSeleccionados: hospitales
        })
    })
}
render() {
    return (
        <div>
            <h1>Hospitales Múltiple</h1>
            <form>
                <select ref={this.selectHospital} className='form-control' size="8" multiple>
                    {
                        this.state.hospitales.map((hospital, index) => {
                            return <option key={index} value={hospital.idHospital}> {hospital.nombre}</option>
                        })
                    }
                </select>
                <label>Incremento salarial</label>
                <input type="text" ref={this.cajaIncremento} className='form-control' />
                <button onClick={this.incrementarSalarios} className='btn btn-warning'> Incrementar salarios </button>
                <button onClick={this.getHospitalesSeleccionados} className='btn btn-info'> Mostrar trabajadores </button>
            </form>
            {
                this.state.hospitalesSeleccionados.length != 0 &&
                (<Trabajadores idhospitales={this.state.hospitalesSeleccionados}/>)
            }
        </div>
    )
}

```

#### TRABAJADORES CON CODIGO SOLO A LOS QUE VEMOS EN EL COMPONENT

```

import React, { Component } from 'react'
import axios from 'axios'
import Global from '../Global'
export default class Trabajadores extends Component {
    cajaIncremento = React.createRef();
    state = {
        trabajadores: [],
        mensaje: ""
    }
    loadTrabajadores = () => {
        //RECUPERAR TODOS LOS IDS DE HOSPITA
        let idshospitales = this.props.idhospitales;
        if (idshospitales.length != 0){
            //idhospital=17&idhospital=22&idhospital=14
            let data = "";
            for (var id of idshospitales){
                data += "idhospital=" + id + "&";
            }
            //ELIMINAMOS EL ULTIMO CARACTER DEL STRING
            //idhospital=25&idhospital=22
            data = data.substring(0, data.length - 1);
            this.setState({
                mensaje: data
            })
            //PODEMOS REALIZAR LA PETICION AL SERVICIO
            let request = "api/trabajadores/trabajadoreshospitales?" + data;
            let url = Global.urlEjemplos + request;
            axios.get(url).then(response => {
                console.log("Leyendo Trabajadores");
                this.setState({
                    trabajadores: response.data
                })
            })
        }
    }
    componentDidMount = () => {
        this.loadTrabajadores();
    }
    componentDidUpdate = (oldProps) => {
        if (oldProps.idhospitales != this.props.idhospitales){
            this.loadTrabajadores();
        }
    }
}

```

```

incrementoSalarialSoloEstos = (e) => {
  e.preventDefault();
  let incremento = parseInt(this.cajaIncremento.current.value);
  let idshospitales = this.props.idhospitales;
  if (idshospitales.length != 0){
    //idhospital=17&idhospital=22&idhospital=14
    let data = "";
    for (var id of idshospitales){
      data += "idhospital=" + id + "&";
    }
    //ELIMINAMOS EL ULTIMO CARACTER DEL STRING
    //idhospital=25&idhospital=22&
    data = data.substring(0, data.length - 1);
    this.setState({
      mensaje: data
    })
    let request = "api/trabajadores/updatesalariotrabajadoresthospitales?
incremento="
    + incremento + "&" + data;
    let url = Global.urlEjemplos + request;
    axios.put(url).then(response => {
      this.loadTrabajadores();
    })
  }
}
render() {
  return (
    <div>
      <h1 style={{color:"fuchsia"}}>Trabajadores</h1>
      <form>
        <label>Incremento trabajadores</label>
        <input type="text" ref={this.cajaIncremento} className='form-control' />
        <button onClick={this.incrementoSalarialSoloEstos} className='btn btn-outline-danger'>
          Incrementar trabajadores
        </button>
      </form>
      <table className='table table-bordered'>
        <thead>
          <tr>
            <th>Apellido</th>
            <th>Salario</th>
            <th>Hospital</th>
          </tr>
        </thead>
        <tbody>
          {
            this.state.trabajadores.map((trabajador, index) => {
              return (<tr key={index}>
                <td>{trabajador.apellido}</td>
                <td>{trabajador.salario}</td>
                <td>{trabajador.idHospital}</td>
              </tr>
            ))
          }
        </tbody>
      </table>
      <h5 style={{color:"blue"}}>
        {this.state.mensaje}
      </h5>
    </div>
  )
}
}

```

Necesito una práctica de CRUD

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

Creamos un nuevo proyecto llamado **reactcrudcoches** y realizamos lo mismo que ya Hicimos con **Departamentos**

Coches	
GET	/api/Coches
GET	/api/Coches/FindCoches/{id}
POST	/api/Coches/InsertCoches
PUT	/api/Coches/UpdateCoches
DELETE	/api/Coches/DeleteCoches/{id}

PARA FINALIZAR EL MODULO

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

Tenemos dentro del Api dos elementos que son Equipos y Apuestas.

Lo que necesito es lo siguiente:

- Al iniciar nuestra página, cargaremos en el Menú los equipos.
- Al seleccionar un equipo del Menú, veremos sus detalles.
- Tendremos otro componente para visualizar Las apuestas y dentro de

Dicho componente podremos crear nuevas apuestas

Creamos un proyecto nuevo llamado **reactapuestascasico**

# VUE

jueves, 24 de octubre de 2024 13:10

VUE es un Framework de tipo SPA (Single Page Application).

Es de código libre. La comunidad se encarga de crear las mejoras y las diferentes versiones

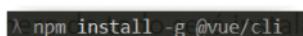
Nos dará igual un Framework que otro, simplemente tenemos que saber las diferencias.

Dentro de VUE tenemos Components. Tenemos comunicación Padre e Hijo y tenemos  
También **props**

Una de las mayores diferencias desde React, radica en que VUE no utiliza **STATE**, es decir,  
Si cambiamos el valor a una variable, el dibujo cambiará.

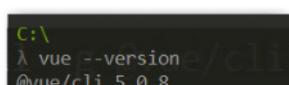
Lo primero de todo será instalar VUE.

```
npm install -g @vue/cli
```



Para comprobar la versión de VUE en la que estamos

```
vue --version
```

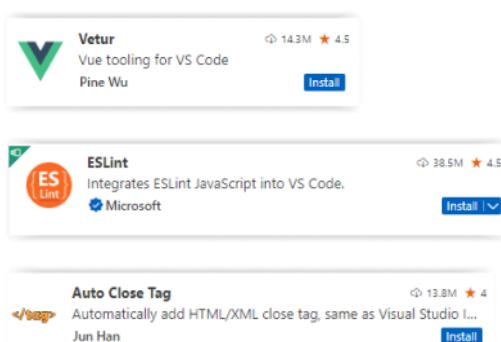


Vamos a trabajar igual que con React

Tendremos una carpeta raíz llamada **vue** y dentro múltiples proyectos con sus funcionalidades

En VUE no es imprescindible trabajar con minúsculas. Pero como no me gusta pensar, trabajamos  
En minúsculas.

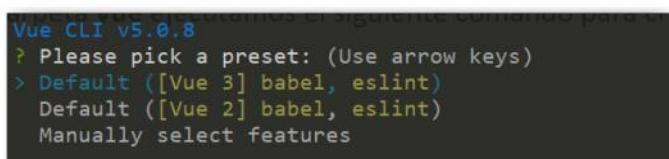
Abrimos VS Code e instalamos una serie de Extensiones.



Para crear proyectos, se utiliza el siguiente comando

```
vue create NOMBREPROYECTO
```

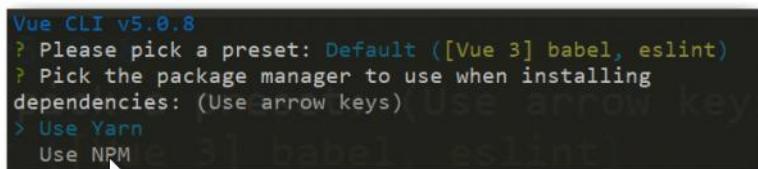
Dentro de nuestra carpeta **vue** ejecutamos el siguiente comando para crear un proyecto llamado **primervue**



The terminal shows the creation of a new Vue project named 'primervue'. It asks for a preset configuration:

```
Vue CLI v5.0.8
? Please pick a preset: (Use arrow keys)
> Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
  Manually select features
```

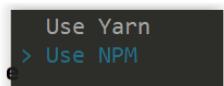
Lo único que tenemos que pensar es que nos pedirá entre YARN o NPM



The terminal shows the choice of package manager:

```
Vue CLI v5.0.8
? Please pick a preset: Default ([Vue 3] babel, eslint)
? Pick the package manager to use when installing dependencies: (Use arrow keys)
> Use Yarn
  Use NPM
```

Nosotros con quedamos con **NPM**



The terminal shows the final confirmation of using NPM:

```
Use Yarn
> Use NPM
```

Una vez el proyecto está creado, nos indica cómo entrar y como arrancar nuestro server.

```

Generating README.md...
Successfully created project primervue.
Get started with the following commands:

$ cd primervue
$ npm run serve

```

Esto sigue siendo igual, simplemente entramos en la carpeta del nuevo proyecto y escribimos `code`.

Para arrancar el servidor `npm run serve`

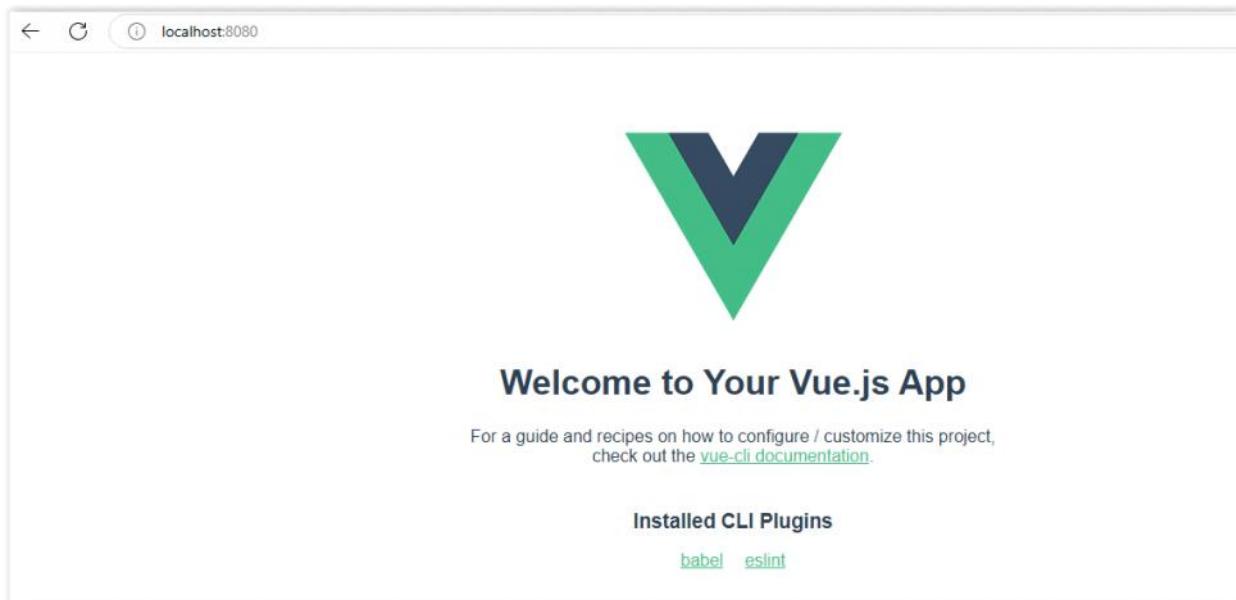
Y veremos que nuestra app arrancará dentro de <http://localhost:8080>

```

DONE Compiled successfully in 3726ms
13:43:5
App running at:
- Local: http://localhost:8080/
- Network: http://10.202.1.26:8080/
Note that the development build is not optimized.
To create a production build, run npm run build.

```

Y veremos el proyecto ya listo y funcional.



Nos centraremos en investigar el proyecto de VUE y dónde están los elementos

Dentro de la carpeta `public` tendremos nuestra página inicial

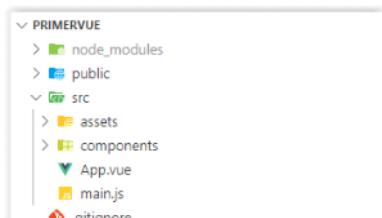
```

<body>
<noscript>
  <strong>We're sorry but <%= htmlWebpackPlugin.options.title
</noscript>
<div id="app"></div>
<!-- built files will be auto injected -->
</body>

```

Tendremos la etiqueta `<div id="app">` que es elemento principal donde vue genera los componentes.

Nosotros estaremos interesados en `src`



Tendremos una carpeta llamada **assets** para trabajar con elementos locales, como imágenes, CSS o JS

Dentro de la carpeta **components** estarán nuestros componentes personalizados

Los components dentro de VUE finalizan con la extensión **.vue**

Lo primero de todo es visualizar **main.js**



```
main.js
src > main.js
1 import { createApp } from 'vue'
2 import App from './App.vue'
3
4 createApp(App).mount('#app')
5
```

La clase **MAIN** es el JEFE de VUE. Será el encargado de habilitar características dentro De VUE, como por ejemplo, **RUTAS**

Un componente está compuesto por **TRES CAPAS**:

- 1) **Template:** Es la plantilla HTML del dibujo del Component Siempre debemos tener una etiqueta Parent, es decir, un <div></div>

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

- 2) **Script:** Es la lógica del componente, dónde incluiremos nuestros métodos, Variables o la declaración del uso de otros components También tendremos algo llamado **Hooks** que son los ciclos de vida del component

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

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

- 3) **Style:** Son los estilos CSS de nuestro component.

```
<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>
```

Estamos viendo el Component **App.vue** que es actualmente el component principal de nuestra App.

Vamos a comenzar trabajando con dicho Component.

Nos vamos a crear nuestro propio component y lo dibujaremos dentro de **App.vue**  
Sobre **components**, creamos un nuevo component llamado **HolaMundo.vue**

**HOLAMUNDO.VUE**

```

<template>
  <div>
    <h1>Mi primera App de Vue, que ilusion!!!</h1>
  </div>
</template>

<script>
  //ESTO ES CODIGO JS (ESPECIAL)
  //CADA COMPONENT QUE CREEMOS SIEMPRE DEBEMOS EXPORTARLO
  //CON UN name
  export default {
    name: "HolaMundo"
  }
</script>

<style>
  h1 {color: blue}
</style>

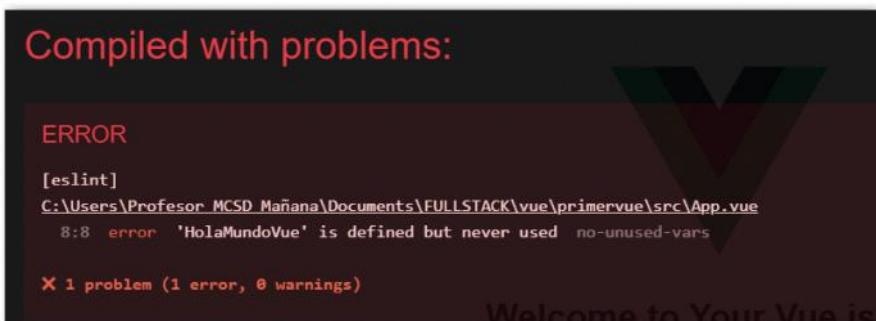
```

Lo siguiente será integrar nuestro component dentro de **App.vue**

En cualquier component que deseemos integra, debemos hacer un **import**, igual que React. Dicho import irá dentro de **script**

```
import HolaMundoVue from './components/HolaMundo.vue'
```

Si abrimos la página, veremos un error GIGANTE



VUE contiene una particularidad que indica que **NO PODEMOS DECLARAR ELEMENTOS Y NO UTILIZARLOS**

Debemos indicar que utilizaremos el component **HolaMundo** dentro **export default**

Se declara dentro de **components**

```
export default {
  name: 'App',
  components: {
    HelloWorld, HolaMundo
  }
}
```

Para utilizarlo, es exactamente igual que **React: <Component/>**

```

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

```

Y veremos nuestra App con el component integrado

# Mi primera App de Vue, que ilusión!!!



Si vemos que nos está dando algún error por las extensiones

```
<template>  Parsing error: No Babel config file detected for C:\Users\Alumnos MCSO Mañana\Documents\FULLSTACK\Jorge\vue\primervue\src\App.vue. Either disable config file checking or
<HolaMundo/>

<HelloWorld msg="Welcome to Your Vue.js App"/>  [vue/no-multiple-template-root] The template root requires exactly one element.
</template>
```

Dentro de `package.json` incluimos la siguiente línea

```
  ],
  "parserOptions": {
    "parser": "@babel/eslint-parser",
    "requireConfigFile": false
  },
```

También tenemos que recordar que SOLO abriremos el proyecto de VUE actual.

## VARIABLES DENTRO DE UN COMPONENTE

Dentro de cualquier componente y dentro de su zona `<script>` existen diferentes lugares para declarar y ejecutar acciones.  
Dependiendo de la "zona" dónde declaremos dicha variable, tendrá una funcionalidad o ámbito distinto.

Tenemos una zona llamada `data()` que nos permite declarar variables para el dibujo del Componente.

Si deseamos tener variables y utilizarlas dentro del componente debemos utilizar la Palabra `return`

```
export default {
  name: "HolaMundo",
  data() {
    //AQUÍ DEBEMOS DECLARAR LAS VARIABLES PARA EL DIBUJO
    //LAS VARIABLES SON DECLARADAS CON FORMATO JSON DENTRO DE
    //UN return
    return {
      nombre: "Alumno",
      numero: 14,
      miArray: [],
      objeto: {}
    }
  }
}
```

Si deseamos dibujar dichas variables, lo haremos dentro del `<template>` y con doble llave

```
<template>
  <div>
    <h1>Mi primera App de Vue, que ilusión!!!</h1>
    <h2>Su nombre es {{nombre}}</h2>
    <h2>El número es {{numero}}</h2>
  </div>
</template>
```

Y podremos visualizar el resultado

# Mi primera App de Vue, que ilusion!!!

Su nombre es Alumno

El número es 14



## UTILIZAR CSS EXTERNO DESDE ASSETS

Dentro de la carpeta `assets` incluimos una nueva carpeta llamada `css`

Creamos un estilo llamado `estilos.css`

```
img {  
  width: "150px";  
  height: "150px";  
}  
  
h2 {  
  background-color: yellow;  
}
```

Los estilos se utilizan dentro de la zona `<style>` en un Component, utilizando la palabra `@import`

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

VUE comprueba todo antes de compilar, incluida su sintaxis y elementos que no lleguen a existir dentro de nuestro servidor.

```
<template>  
  <div>  
    <h1>Mi primera App de Vue, que ilusion!!!</h1>  
    <h2>Su nombre es {{nombre}}</h2>  
    <h2>El número es {{numero}}</h2>  
      
  </div>  
</template>
```

Podremos comprobar que NO solamente los elementos compilados entran dentro del Compilador de VUE

## Compiled with problems:

### Mi primera App de Vue, que ilusion!!!

```
ERROR in ./src/components/HolaMundo.vue?vue&type=template&id=23be7e16 (.node_modules/babel-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/batman.png' in 'C:\Users\Profesor MCSD Mañana\Documents\FULLSTACK\vue\primervue\src\components'
```

Por ejemplo, vamos a incluir una imagen (local) dentro de `assets/images` y vamos a dibujar.

## ROUTING CON VUE

Routing NO es nativo de VUE. Debemos instalar las librerías necesarias para poder realizar la navegación entre Componentes.

Instalamos el paquete de Routing de vue.

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

Buscamos imágenes para **Cine**, **Musica** y otra para **Home**

Ponemos las imágenes dentro de **assets/images**

Vamos a utilizar la palabra **Component** al final de las clases de VUE para denominar a nuestros Components.

Creamos los siguientes components:

HomeComponent  
MusicaComponent  
CineComponent

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

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

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

Dibujamos los components en App para comprobar que son funcionales

El trabajo de Routing es exactamente igual que en React, navegar entre Components.

Necesitamos una clase llamada **Router.js** dentro de la carpeta **src**

Dentro de la clase Router debemos realizar una serie de pasos:

- 1) Importar cada uno de los Components que vayamos a utilizar

```
import HomeComponent...
import MusicaComponent
```

- 2) Necesitamos un Array que contenga todas las rutas que vayamos a necesitar dentro De nuestro Routing.

```
const Rutas = [
  path: "/", component: HomeComponent,
  path: "/musica", component: MusicaComponent
]
```

- 3) Debemos indicar que las rutas irán como integración en nuestra App

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

- 4) Dentro de la clase **MAIN.JS** habilitamos las rutas para nuestra App

```
createApp(app).use(Router).mount(#app)
```

Sobre la carpeta **src** incluimos un nuevo fichero llamado **Router.js**

#### ROUTER.JS

```
import { createRouter, createWebHistory } from "vue-router";
import HomeComponent from './components/HomeComponent.vue';
import CineComponent from './components/CineComponent.vue';
import MusicaComponent from './components/MusicaComponent.vue';
//CREAMOS UNA CONSTANTE PARA LAS RUTAS
const myRoutes = [
  {
    path: "/", component: HomeComponent
  },
  {
    path: "/musica", component: MusicaComponent
  },
  {
    path: "/cine", component: CineComponent
  }
]
//CREAMOS UNA CONSTANTE PARA EL HISTORIAL E INCLUIR EL ARRAY DE RUTAS
//Dicho nombre de constante sera el que utilizaremos dentro de Main.js
const router = createRouter({
  history: createWebHistory(),
  routes: myRoutes
})
//POR ULTIMO, EXPORTAMOS LA CONSTANTE router PARA SER UTILIZADA
//EN App
export default router;
```

El siguiente paso es habilitar 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 de la navegación necesitamos una etiqueta llamada `<router-view>`.

Dónde dicha etiqueta esté ubicada, veremos la integración del Component en la navegación

APP.VUE

```

<template>
  <div>
    <h1 style="color: blue">
      Título estático de App
    </h1>
    <hr/>
    <router-view></router-view>
    <hr/>
  </div>
</template>

```

Por último, incluiremos un menú con los Links a cada uno de nuestros components.

En VUE no se utiliza la etiqueta `<a>` para los Links

Para navegar entre components utilizamos la etiqueta `<router-link>`

```
<router-link to="/">Navegar</router-link>
```

Vamos a crear un nuevo component llamado **MenuComponent**

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>

```

Por último, dibujamos nuestro component Menu dentro de App

APP.VUE

```

<template>
  <div>
    <h1 style="color: blue">
      Título estático de App
    </h1>
    <MenuComponent/>
    <hr/>
    <router-view></router-view>
    <hr/>
  </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>

```

# Título estático de App

Home | Cine | Música

## Musica component



### HOOKS O CICLOS DE VIDA EN COMPONENTS

Un Hook o ciclo de vida indica "cuando" sucede un momento determinado en un Component. Estamos hablando de `componentDidMount()` o `componentDidUpdate()` de React

Para incluir los ciclos de vida dentro de nuestro component debemos hacerlo en `<script>`

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

Tenemos los siguientes Hooks dentro de VUE.

- `mounted()`: Cuando el component es montado y dibujado
- `created()`: Se ejecuta cuando el component es montado, pero no dibujado
- `updated()`: Se ejecuta cuando el component cambia su dibujo dentro de `<template>`
- `unmounted()`: Se ejecuta cuando el component es eliminado de un Parent

### EVENTOS EN VUE

Cuando hablamos de eventos, estamos hablando de los Clicks típicos de los botones

Los eventos se pueden invocar de dos formas diferentes:

- 1) `v-on:EVENTO`
- 2) `@evento`

En código:

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

No podemos declarar los métodos de forma tan alegre por nuestro código, necesitamos una "zona" dentro de `<script>` para declarar los métodos

La zona se llama **methods**

Dentro de **methods**, para acceder a las variables (**data**) ahí sí que utilizaremos la palabra clave **this**

Vamos a comprobar que las variables son **reactivas** dentro de `data()` y con ciclos de vida y Eventos.

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

Lo primero de todo, lo creamos, ponemos un mensaje `<h1>` y lo ponemos dentro de la Navegación.

### CICLOVIDA.VUE

```
<template>
  <div>
    <h1>Ciclo Vida Events</h1>
    <button @click="cambiarSaludo()">
      Cambiar saludo
    </button>
    <h2 style="color:blue">{{saludo}}</h2>
  </div>
</template>
<script>
  export default {
    name: "CicloVida",
    data() {
      return {
        saludo: "Hoy es lunes..."
      }
    },
    methods: {
      //EN ESTA ZONA INCLUIREMOS NUESTROS METODOS PERSONALIZADOS
      cambiarSaludo() {
        //SE UTILIZA LA PALABRA CLAVE this PARA LLAMAR A LAS
      }
    }
  }
</script>
```

```

    //VARIABLES DE data()
    this.saludo = "Y mañana martes...";
}
, mounted() {
  console.log("Component Montado");
}, unmounted() {
  console.log("Component destruido");
}, updated() {
  console.log("Component modificado");
}, created() {
  console.log("Component Creado");
}
}
</script>

```

Y podremos comprobar el resultado

## DIRECTIVAS VUE

Una directiva permite escribir código lógico dentro del `<template>` de Vue.

Es lo mismo que hicimos con el lenguaje de llaves en el `render()` de React

Nos permiten crear condicionales IF o bucles FOR dentro del código HTML

Las directivas van incluidas dentro de las propias etiquetas HTML

Existe una palabra clave para incluir directivas de VUE: `v-`

Tenemos directivas IF: `v-if`

Tenemos directivas FOR: `v-for`

Ejemplo:

```
<h1 v-if="{variable==true}>SOLAMENTE ME DIBUJO CON TRUE</h1>
<h1 v-if-else-if="{variable==false}>ME DIBUJO CON FALSE</h1>
```

Vamos a comenzar a trabajar con formularios con este concepto.

En un formulario, podemos enlazar variables mediante `v-model`

Un modelo nos permite modificar de forma bidireccional una variable que tengamos Enlazada con un control HTML de FORMS.

Ejemplo:

Si modificamos la caja, modificamos el Modelo.

Si modificamos el modelo, se modifica el dibujo de la caja

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

Vamos a realizar un ejemplo para comprobar si un número es Par o es impar.  
Enlazaremos las variables mediante `binding`

Creamos un nuevo component llamado `DirectivasComponent.vue`

## DIRECTIVASCOMPONENT.VUE

```

<template>
  <div>
    <h1>Ejemplo directivas VUE</h1>
    <h1>
      <!-- EL SIMBOLO DOLAR NOS PERMITE ACCEDER A ELEMENTOS DENTRO DE
VUE --&gt;
      {{$data}}
    &lt;/h1&gt;
    &lt;label&gt;Introduzca número&lt;/label&gt;
    &lt;input type="number" v-model="numero"/&gt;
    &lt;h3 v-if="numero &gt; 0" style="color:green"&gt;
      POSITIVO
    &lt;/h3&gt;
    &lt;h3 v-else-if="numero &lt; 0" style="color:red"&gt;
      NEGATIVO
    &lt;/h3&gt;
    &lt;h3 v-else style="color:blue"&gt;
      CERO
    &lt;/h3&gt;
  &lt;/div&gt;
&lt;/template&gt;
&lt;script&gt;
  export default {
    name: "DirectivasComponent",
    data() {
      return {
</pre>

```

```

        numero: 0,
        nombre: "Alumno",
        edad: 25
    }
}
</script>

```

## Título estático de App

[Home](#) | [Cine](#) | [Música](#) | [Hooks](#) | [Directivas](#)

### Ejemplo directivas VUE

Introduzca número

**NEGATIVO**

Debemos tener cuidado con los tipados de los Binding...  
Para visualizarlo, vamos ver en el dibujo todo lo que contenga DATA.

Mediante \$ nos permite acceder a elementos clave dentro de VUE.  
Por ejemplo, podría dibujar todo el contenido de data con `${{data}}`

Cuando hacemos un BINDING, el **type** de los controles prevalece sobre el tipo de variable.  
Es decir, si ponemos un `<input type="text">`, tendremos un TEXT

### Ejemplo directivas VUE

{ "numero": "10", "nombre": "Alumno", "edad": 25 }

Debemos tener en cuenta el Type y utilizarlo de forma conveniente cuando tengamos un  
**number**

```

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

```

Y veremos que el binding ya tiene tipado

{ "numero": 10, "nombre": "Alumno",

Introduzca número

**POSITIVO**

Sobre este mismo ejemplo, vamos a realizar un bucle con la directiva **v-for**

Dentro de los bucles, los elementos que recorremos son mediante **Referencia**, igual  
Que en **React** con un **map** o igual que JS con un bucle **of**

Debemos indicar la variable que utilizaremos dentro del recorrido

Para indicar variables **lógicas** dentro de las directivas se utilizan **los dos puntos** : y  
La palabra **key**

```

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

for (variableReferencia of CONJUNTO) {}

Incluimos un Array dentro de data()

```

```

export default {
  name: "DirectivasComponent",
  data() {
    return {
      series: ["Futurama", "Suits", "The Boys", "Stranger Things"],
      numero: 0,
      nombre: "Alumno",
      edad: 25
    }
  }
}

```

<template>

```

<ul v-if="series.length > 0">
  <li v-for="serie in series" :key="serie">
    {{serie}}
  </li>
</ul>

```

Y podremos comprobar el dibujo

The screenshot shows a user interface with a text input field containing '0' and a list of series below it. The list includes 'Futurama', 'Suits', 'The Boys', and 'Stranger Things'. Above the list, the word 'CERO' is displayed in large blue capital letters.

## PROPIEDADES CONMUTADAS

Una propiedad conmutada es un método que se utiliza como una Propiedad. Es una herramienta que tenemos dentro de VUE y que permite devolver código HTML Dinámico mediante un método.

No se utiliza con paréntesis y se suele utilizar en recorridos para los dibujos dinámicos Solamente una vez.

Sirve para facilitar código en directivas, en lugar de tener todo el código dentro de un <template> podemos tener dicho código en una propiedad conmutada y que aplique Dicho código en un recorrido.

```

//METODO QUE DEVUELVE HTML
evaluarNumero(){
  var resultado = "";
  if (this.numero > 0) {
    resultado = "<h1 style='color:green'>POSITIVO {{this.numero}}</h1>";
  }else{
    resultado = "<h1 style='color:red'>NEGATIVO{{this.numero}}</h1>";
  }
  return resultado;
}

```

Podriamos dibujar el código utilizando una directiva que "traduce" texto a código HTML

<div v-html="resultado"/>

¿Qué sucede si estamos en un recorrido de múltiples números con un v-for?

```

<div v-for="num in misNumeros" :key="num">
  <h3 v-if="num > 0" style="color:green">
    POSITIVO
  </h3>
  <h3 v-else-if="num < 0" style="color:red">
    NEGATIVO
  </h3>
  <h3 v-else style="color:blue">
    CERO
  </h3>
</div>

```

Tenemos la posibilidad de indicar, mediante una propiedad conmutada (evaluarNumero()) Y que llame al método y haga el dibujo por mí.

```
<div v-for="num in misNumeros" :key="num">
  <h3 evaluarnumero></h3>
</div>
```

Los métodos/propiedades conmutadas se incluyen en una nueva zona llamada **computed**  
Y siempre deben devolver un valor **return**

Vamos a realizar un nuevo ejemplo en el que veremos las propiedades conmutadas

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

## Propiedades conmutadas

```
Parchis
Canicas
Luz roja, luz verde
La cuerda
La galleta
```

PROPIEDADCONMUTADA.VUE

```
<template>
  <div>
    <h1 style="color:blue">
      Propiedades conmutadas
    </h1>
    <ul>
      <li v-for="game in juegosRojos" :key="game" v-html="game">
      </li>
    </ul>
  </div>
</template>
<script>
  export default {
    name: "PropiedadConmutada",
    computed: {
      //UNA PROPIEDAD CONMUTADA SIEMPRE DEVUELVE VALORES
      //Y NO PUEDE RECIBIR PARAMETROS
      juegosRojos() {
        //VAMOS DEVOLVER TODOS LOS JUEGOS CON
        //UN FORMATO DE COLOR ROJO
        var datos = this.juegos;
        //RECORREMOS CADA JUEGO Y MODIFICAMOS SU DIBUJO A COLOR ROJO
        for (var i = 0; i < datos.length; i++){
          var juego = datos[i];
          juego = "<span style='color:red'>" + juego + "</span>";
          datos[i] = juego;
        }
        return datos;
      },
      data() {
        return {
          juegos: ["Parchis", "Canicas"
            , "Luz roja, luz verde", "La cuerda", "La galleta"]
        }
      }
    }
  </script>
```

Podríamos realizar el mismo ejemplo, pero con un método?

- Un botón, un método, variable con código HTML dinámico y dibujar los valores

Modificamos el ejemplo anterior e implementamos lo mismo pero con métodos.

## Propiedades conmutadas

Juegos azules

Parchis

Canicas

Luz roja, luz verde

PROPIEDADCONMUTADA.VUE

```
<template>
  <div>
    <h1 style="color:blue">
      Propiedades conmutadas
    </h1>
    <button @click="getJuegosAzules()">
      Juegos azules
    </button>
    <div v-html="html"></div>
    <!-- <ul>
      <li v-for="game in juegosRojos" :key="game" v-html="game">
```

```

        </li>
    </ul> -->
</div>
</template>
<script>
export default {
    name: "PropiedadCommutada",
    methods: {
        getJuegosAzules() {
            //EL METODO NO DEVUELVE NADA, NECESITAMOS TENER UN DIBUJO
            // this.html = "";
            for (var juego of this.juegos){
                this.html += "<h4 style='color:blue'>" + juego + "</h4>";
            }
        }
    },
    computed: {
        //UNA PROPIEDAD CONMUTADA SIEMPRE DEVUELVE VALORES
        //Y NO PUEDE RECIBIR PARAMETROS
        juegosRojos() {
            //VAMOS DEVOLVER TODOS LOS JUEGOS CON
            //UN FORMATO DE COLOR ROJO
            var datos = this.juegos;
            //RECORREMOS CADA JUEGO Y MODIFICAMOS SU DIBUJO A COLOR ROJO
            for (var i = 0; i < datos.length; i++){
                var juego = datos[i];
                juego = "<span style='color:red'>" + juego + "</span>";
                datos[i] = juego;
            }
            return datos;
        },
        data() {
            return {
                html: "",
                juegos: ["Parchis", "Canicas"
                    , "Luz roja, luz verde", "La cuerda", "La galleta"]
            }
        }
    }
}
</script>

```

Como hemos podido comprobar, una propiedad commutada y un método en realidad  
Son lo mismo.

La gran diferencia entre ellos está en que una propiedad commutada no admite  
Parámetros en el método, simplemente se realiza la llamada en el dibujo.

#### PRACTICA PAR IMPAR

Necesito un componente que se encargará de evaluar si un número es par o impar.

Tendremos una caja para poder evaluar el número y modificar su valor.

Necesito realizar la práctica de varias formas:

- 1) Propiedad commutada que nos devuelva un `<h1>` con colores si el número  
Es par o es Impar
- 2) El mismo ejemplo en el mismo componente, pero con un botón y método
- 3) Intentarlo con directivas.

Creamos un nuevo componente llamado `NumeroParImpar.vue`



#### NUMEROPARIMPAR

```

<template>
<div>
    <h1>Número Par Impar</h1>
    <label>Número:</label>
    <input type="number" v-model="numero"/>
    <h3 style="color:blue" v-if="numero % 2 == 0">PAR</h3>
    <h3 v-else style="color:red">IMPAR</h3>
    <button @click="getEvaluarNumero()">Evaluar número</button>
    <div v-html="html"></div>
    <div v-html="evaluarNumero"></div>
</div>
</template>
<script>
export default {
    name: "NumeroParImpar",
    computed: {
        evaluarNumero() {
            let data = "";
            if (this.numero % 2 == 0){

```

```

        data = "<h2 style='color:blue'>Par: " + this.numero + "</h2>";
    }else{
        data = "<h2 style='color:green'>Impar: " + this.numero +
    "</h2>";
    }
    return data;
},
methods: {
    getEvaluarNumero() {
        if (this.numero % 2 == 0){
            this.html = "<h3 style='color:green'>Par " + this.numero +
        "</h3>";
        }else{
            this.html += "<h3 style='color:orange'>Impar " + this.numero +
        "</h3>";
        }
    },
    data() {
        return {
            numero: 0,
            html: ""
        }
    }
}
</script>

```

## METODOS FILTERS GLOBALES

Esto es una herramienta para poder reutilizar códigos que tengamos.  
 Hemos visto como podemos utilizar métodos/conmutadas para poder cambiar dibujos  
 Dinámicamente.  
 Cada método o propiedad conmutada siempre estará en cada componente.  
 Si necesitamos aplicar "algo" a varios components, nos toca CTROL + c CONTROL + V

Estos métodos llamados Filters podemos escribirlos solamente UNA VEZ y llamarlos  
 Desde cualquier componente.  
 Los métodos Filters se escriben dentro de la clase **MAIN.JS**

Podemos realizar la llamada desde **<template>** a dichos métodos

Lo primero, será incluir dentro de **main.js** un método Filter Global

```

main.js
import { createApp } from 'vue'
import App from './App.vue'
import router from './Router'
I
createApp(App)
.use(router)
.mount('#app')

```

Necesitamos crear una zona llamada **Global Filters** y, dentro de dicha zona, incluir nuestros  
 Métodos Globales

```

app.config.globalProperties.$filters = {
    //EN ESTE CÓDIGO INCLUIREMOS NUESTROS METODOS GLOBALES
    miMetodoGlobal() {
    }
}

```

Para poder utilizar los métodos en cualquier Component en **<template>**

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

Para probar esta funcionalidad creamos un nuevo component llamado **MetodosFilters.vue**

Vamos a realizar algo tan sencillo como, mediante un método FILTERS, ponemos textos  
 En mayúsculas.  
 Tendremos otro método para dibujar el DOBLE de cualquier número.

Comenzamos escribiendo los dos métodos dentro de **main.js**

**MAIN.JS**

```
//COMENZAMOS SEPARANDO NUESTRA CREACION DE APLICACION EN UNA
//VARIABLE.
var app = createApp(App)
app.config.globalProperties.$filters = {
    //CREAMOS DOS METODOS QUE RECIBIRAN PARAMETROS
    //Y DEVOLVERAN UN CODIGO
    mayuscula(dato){
        return dato.toUpperCase();
    },
    getNumeroDoble(numero){
        return numero * 2;
    }
}

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

#### METODOSFILTERS.VUE

Creamos dos Arrays, una con NOMBRES y otra con números

```
<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: [2,6,55,77,88,99,102],
        nombres: ["Adrian", "Lucia", "Diana", "Carlos", "Antonia"]
      }
    }
</script>
```

Y veremos el resultado

## Métodos Filters

Número: 2, Doble: 4  
 Número: 6, Doble: 12  
 Número: 55, Doble: 110  
 Número: 77, Doble: 154  
 Número: 88, Doble: 176  
 Número: 99, Doble: 198  
 Número: 102, Doble: 204

ADRIAN  
 LUCIA  
 DIANA  
 CARLOS  
 ANTONIA

Creamos un nuevo proyecto llamado **vuepracticashunes**

Necesito un Component **HomeComponent** y habilitar la navegación

Necesitamos un Menú, un Router.js y HomeComponent

Creamos otro component llamado **CollatzComponent** y hacemos la conjetura de Collatz  
 Con una caja y un botón.

Mediante un FILTER, nos devuelva si el número es par o impar y dibujar VERDE: PAR y  
 ROJO: IMPAR

Directamente el Filter nos devuelve el dibujo.

Todo número positivo entero será 1 siguiendo dos pasos:

- 1) Si el número es par, dividimos entre 2
- 2) Si el número es impar, multiplicamos por 3 y sumamos 1

# Collatz Component

Introduzca número  Generar Collatz

```
6  
3  
10  
5  
16  
8  
4  
2  
1
```

## ROUTER.JS

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

## MENUCOMPONENT

```
<template>
  <div>
    <ul id="menu">
      <li>
        <a href="/">Home</a> |
      </li>
      <li>
        <a href="/collatz">Collatz</a>
      </li>
    </ul>
  </div>
</template>
<script>
  export default {
    name: "MenuComponent"
  }
</script>
<style>
  ul#menu li {
    display: inline
  }
</style>
```

## COLLATZCOMPONENT

```
<template>
  <div>
    <h1 style="color:blue">Collatz Component</h1>
    <label>Introduzca número:</label>
    <input type="number" v-model="numero"/>
    <button @click="generarCollatz()">
      Generar Collatz
    </button>
    <div v-if="numerosCollatz.length > 0">
      <ul>
        <li v-for="num in numerosCollatz" :key="num"
          v-html="$filters.evaluarNumero(num)">
        </li>
      </ul>
    </div>
  </div>
</template>
<script>
  export default {
    name: "CollatzComponent",
    data() {
      return {
        numero: -1,
        numerosCollatz: []
      }
    }, methods: {
```

```

generarCollatz() {
    let aux = [];
    aux.push(this.numero);
    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 será la Tabla de multiplicar.

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

Debemos dibujar una tabla con Operación y Resultado

Realizamos el ejemplo de varias formas:

- 1) Un botón y un método que genera código HTML y dibuja nuestra tabla.
- 2) Generamos otra tabla mediante directivas.
- 3) Generamos la tabla utilizando Filters. Método que sea getOperacion y otro getMultiplicacion

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

#### 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>";
        }
    },
    getOperacion(numero, multi) {
        return numero + " * " + multi;
    },
    getMultiplicacion(numero, multi){
        return numero * multi;
    }
}
app.use(router).mount('#app')

```

#### TABLAMULTIPLICAR.VUE

```

<template>
<div>
    <h1 style="color:red">Tabla Multiplicar</h1>
    <label>Introduzca número</label>
    <input type="number" v-model="numero"/>
    <button @click="generarTabla()">
        Generar tabla
    </button>
    <table border="1">
        <thead>
            <tr>
                <th>Operación</th>
                <th>Resultado</th>
            </tr>
        </thead>
        <tbody v-html="html"></tbody>
    </table>
    <table v-if="numero != 0" style="background-color:lightblue">
        <thead>
            <tr>
                <th>Operación</th>
                <th>Resultado</th>
            </tr>
        </thead>
        <tbody>
            <tr v-for="contador in 10" :key="contador">
                <td> {{ $filters.getOperacion(numero, contador)}} </td>
                <td> {{ $filters.getMultiplicacion(numero, contador)}} </td>
            </tr>
        </tbody>
    </table>
    <table v-if="numero != 0" style="background-color:lightgreen">
        <thead>
            <tr>
                <th>Operación</th>
                <th>Resultado</th>
            </tr>
        </thead>

```

```

        </thead>
        <tbody>
            <tr v-for="contador in 10" :key="contador">
                <td>{{ numero + " * " + contador }}</td>
                <td>{{ 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++){
                    var operacion = this.numero + " * " + i;
                    var resultado = this.numero * i;
                    aux += "<tr>";
                    aux += "<td>" + operacion + "</td>";
                    aux += "<td>" + resultado + "</td>";
                    aux += "</tr>";
                }
                this.html = aux;
            }
        }
    }
</script>

```

## COMUNICACIÓN ENTRE COMPONENTES

La comunicación entre componentes es igual que en React.  
De Padre a Hijo se utiliza **props**

Para poder enviar **props** desde un padre a un hijo, debemos indicarlo mediante Los dos puntos. (No siempre)

- Si props es un valor estático, no utilizamos los dos puntos
- Si props es un valor dinámico (una variable) si utilizamos los dos puntos

Props se captura en el Hijo dentro de la zona **export default**

Tiene una sintaxis parecida a desestructurar

### PARENT

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

### Hijo

```

<script>
    export default {
        name: "Hijo",
        props: [ dato1, dato2 ]
    }
</script>
<template>
    <h1>{{ dato1 }}</h1>
</template>

```

Creamos un nuevo proyecto llamado **vuecomunicacioncomponents**

Tendremos dos componentes:

- 1) **PadreDeportes**: Será el componente padre y contendrá una colección con un conjunto De deportes
- 2) **HijoDeporte**: Será el componente hijo y, mediante **props**, recibirá el deporte del padre Para ser dibujado.



### HIJODEPORTE

```

<template>
    <div>
        <li style="color:red">
            {{nombredeporte}}
        </li>
    </div>
</template>
<script>

```

```

export default {
  name: "HijoDeporte",
  props: [ "nombredel deporte" ],
  mounted() {
    console.log(this.nombredel deporte)
  }
}
</script>


## PADREDEPORTES


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

```

### COMUNICACIÓN HIJO A PARENT

Esta funcionalidad se realiza mediante métodos.

Tendremos un método en el Parent y un método en el Hijo.

El Hijo debe **activar** el método del Padre (Seleccionar favorito y dibujar algo en el Parent)

La funcionalidad es la siguiente:

- 1) Método en el padre

```
metodoPadre()
```

- 2) Método en el Hijo

```
metodoHijo()
```

Para llamar al método Parent, se realiza mediante la instrucción **\$emit**

### Hijo Component

```

metodoHijo() {
  this.$emit('metodoPadre')
}

```

- 3) Necesitamos enviar, mediante **props** el método Padre para que pueda ser invocado.

### PADRE COMPONENT

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

### HIJO

```
<button @click="ejecutarHijo()">Ejecutar método para el Parent</button>
```

```

export default {
  name: "HijoComponent",
  methods: {
    ejecutarHijo() {
      this.$emit('metodoPadre')
    }
}

```

Vamos a implementar esta funcionalidad sobre deportes.

Incluimos un botón en cada hijo que permitirá seleccionar favorito.

Dibujaremos, el PadreDeportes el hijo seleccionado.

Enviremos al parent el Deporte seleccionado en el método.

# Padre deportes

## Su deporte favorito es Surf

- Surf Seleccionar favorito
- Curling Seleccionar favorito
- Bolos Seleccionar favorito
- Cartas Seleccionar favorito
- Petanca Seleccionar favorito
- Peonza Seleccionar favorito

### 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 v-for="sport in deportes" :key="sport">
      <HijoDeporte :nombredeporte="sport"
        v-on:seleccionarFavoritoParent="seleccionarFavoritoParent"/>
    </ul>
  </div>
</template>
<script>
import HijoDeporte from './HijoDeporte.vue'
export default {
  name: "PadreDeportes",
  methods: {
    //CREAMOS UN METODO PARA QUE SEA LLAMADO DESDE EL HIJO
    seleccionarFavoritoParent(deporteFavorito) {
      console.log("Yo soy tu padre");
      this.favorito = deporteFavorito;
    }
  },
  components: {
    HijoDeporte
  },
  data() {
    return {
      deportes: ["Surf", "Curling", "Bolos", "Cartas", "Petanca",
      "Peonza"],
      favorito: null
    }
  }
}
</script>
```

### HIJODEPORTE

```
<template>
  <div>
    <li style="color:red">
      {{nombredeporte}}
      <button @click="seleccionarFavorito()">
        Seleccionar favorito
      </button>
    </li>
  </div>
</template>
<script>
export default {
  name: "HijoDeporte",
  props: [ "nombredeporte" ],
  methods: {
    seleccionarFavorito() {
      console.log("Soy el hijo");
      //DEBEMOS REALIZAR LA LLAMADA AL PADRE
      //DICHA LLAMADA SE HACE MEDIANTE string "nombreMetodoParent"
      //PARA ENVIAR INFORMACION, SE REALIZA CON COMAS POR CADA PARAMETRO
      //QUE TENGAMOS EN LA RECEPCION
      this.$emit("seleccionarFavoritoParent", this.nombredeporte);
    },
    mounted() {
      console.log(this.nombredeporte)
    }
}
</script>
```

### PRACTICA COMUNICACIÓN PADRE HIJO

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

## Numeros Padre

La suma es: 171

Nuevo Número

### Número hijo: 14

Sumar 14

### Número hijo: 22

Sumar 22

#### PADRENUMEROS.VUE

```
<template>
<div>
  <h1 style="color:red">Padre números</h1>
  <h3 style="color:fuchsia">
    Las 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: [7, 22, 55],
      suma: 0
    }
  },
  methods: {
    generarNumero() {
      let random = parseInt(Math.random() * 100) + 1;
      this.numeros.push(random);
    },
    sumarNumeros(numero) {
      this.suma += numero;
    }
  }
}
</script>
```

#### HIJONUMERO.VUE

```
<template>
<div>
  <h1 style="color:blue">Hijo número {{numero}}</h1>
  <button @click="callSumarNumerosParent()">
    Sumar {{numero}}
  </button>
</div>
</template>
<script>
export default {
  name: "HijoNumero",
  props: [ "numero" ],
  methods: {
    callSumarNumerosParent() {
      this.$emit("sumarNumeros", this.numero);
    }
  }
}
</script>
```

#### FORMULARIOS SUBMIT

Hasta ahora, hemos enviado información a nuestro componente mediante un modelo Asociado a los controles. Esto no va a cambiar.

Debemos seguir las normas HTML aunque estemos dentro de SPA  
Para enviar información, siempre debemos hacerlo de etiquetas `<form>`

El problema que tenemos es el mismo que en React, es decir, si pulsamos un botón Dentro de un Form, lo que hará será enviar la información al servidor, es decir, un submit Debemos indicar que no realice la acción (`e.preventDefault()`)

La única diferencia está en la sintaxis. Vamos a cambiar la llamada en el formulario Indicando que la acción irá con preventDefault

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

#### DIRECTIVAS DE ESTILO

Una directiva puede tener un condicional `v-if` o bucles `v-for`. Podemos aplicar directivas también sobre el CSS que tengamos.

## CSS

```
.rojo {  
    color: red  
}  
  
.verde {  
    color: green  
}
```

Las directivas nos permite, de forma dinámica aplicar una clase dependiendo de un valor que Tengamos.

Podríamos preguntar sobre una variable y aplicar diferentes estilos dependiendo de la respuesta

Debemos utilizar los dos puntos si queremos hacer directivas de estilos.

```
<p :class="{"  
    rojo: CONDICION ROJO,  
    verde: CONDICION VERDE  
}">  
    Mi párrafo  
</p>
```

Vamos a realizar una práctica que ya hicimos con Comics.

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

Generamos otro component llamado **ComicComponent.vue**

Dentro de Comics, mostramos un mensaje del hijo Comic.

Sobre **assets** creamos una nueva carpeta llamada **css** y un nuevo css llamado **estilocomic.css**

```
✓ .rojo {  
    background-color: red;  
    color: white  
}  
  
✓ .verde {  
    background-color: lightgreen;  
}  
  
✓ img {  
    width: 150px;  
    height: 250px;  
}  
  
✓ div#comics {  
    float: left;  
}
```

Sobre **ComicComponent** vamos a incluir, dentro de **data()** los comics de VUE.

## Comics Component

Título	Descripción	Imagen	Año	Opciones
Spiderman	Selecciónar favorito		1997	<a href="#">Delete</a>
Wolverine	Selecciónar favorito		2003	<a href="#">Delete</a>
Spawn	Selecciónar favorito		2000	<a href="#">Delete</a>
Batman	Selecciónar favorito		2001	<a href="#">Delete</a>

## COMICS COMPONENT

```
<template>
<div>
  <h1 style="color:blue">Comics Component</h1>
  <form v-on:submit.prevent="createComic()">
    <label>Título</label>
    <input type="text" v-model="comicForm.titulo"/>
    <label>Descripción</label>
    <input type="text" v-model="comicForm.descripcion"/>
    <label>Imagen</label>
    <input type="text" v-model="comicForm.imagen"/>
    <label>Año</label>
    <input type="number" v-model="comicForm.year"/>
    <button>
      Crear comic
    </button>
  </form>
  <div v-if="comicFavorito" class="verde">
    <h2 style="color:blue">{{comicFavorito.titulo}}</h2>
    <h2 style="color:yellow">{{comicFavorito.descripcion}}</h2>
  </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: {
    createComic() {
      this.comics.push(this.comicForm);
    },
    seleccionarFavorito(comic){
      console.log(comic.titulo);
      this.comicFavorito = comic;
    },
    deleteComic(index) {
      this.comics.splice(index, 1);
    }
  },
  data() {
    return {
      comicFavorito: null,
      comicForm: {
        titulo: "",
        imagen: "",
        descripcion: "",
        year: 0
      },
      comics: [
        {
          titulo: "Spiderman",
          imagen:
            "https://3.bp.blogspot.com/-i70Zu_LAHwI/T290xxduu-I/AAAAAAAALo8/8bXDrdvw5Bo/s1600/spiderman1.jpg",
          descripcion: "Hombre araña"
        },
        {
          titulo: "Wolverine",
          imagen:
            "https://images-na.ssl-images-amazon.com/images/I/51c1Q1IdUBL._SX259_BO1,204,203,200_.jpg",
          descripcion: "Lobezno"
        },
        {
          titulo: "Guardianes de la Galaxia",
          imagen:
            "https://tomosygrapas.wordpress.com/wp-content/uploads/2015/09/guardians-of-the-galaxy-1-cover-08a2a.jpg",
          descripcion: "Yo soy Groot"
        },
        {
          titulo: "Avengers",
          imagen:
            "https://d26lpennugtm8s.cloudfront.net/stores/057/977/products/ma_avengers_01_01-891178138c020318f315132687055371-640-0.jpg",
          descripcion: "Los Vengadores"
        },
        {
          titulo: "Spawn",
          imagen:
            "https://i.pinimg.com/originals/e1/d8/ff/e1d8fff4aeab5e5677_98635008fe98ee1.png",
          descripcion: "Al Simmons"
        },
        {
          titulo: "Batman",
          imagen:
            "https://www.comicverso.com/wp-content/uploads/2020/06/The-Killing-Joke-657x1024.jpg",
          descripcion: "Murcielago"
        }
      ]
    }
  }
</script>
```

## COMICCOMPONENT

```
<template>
<div>
  <h2>{{comic.titulo}}</h2>
  <button @click="seleccionarFavorito()">
    Seleccionar favorito
  </button>
</div>
```

```

<button class="rojo" @click="deleteComic()">
  Delete
</button>
<p>{{comic.descripcion}}</p>

<h4 :class="{>
  rojo: comic.year <= 2000,>
  verde: comic.year > 2000>
}">
  Año: {{comic.year}}
</h4>
</div>
</template>
<script>
  export default {
    name: "ComicComponent",
    props: [ "comic", "index" ],
    methods: {
      seleccionarFavorito() {
        this.$emit("seleccionarFavorito", this.comic)
      },
      deleteComic() {
        this.$emit("deleteComic", this.index)
      }
    }
  }
</script>
<style>
@import '../assets/css/estilocomic.css';
</style>

```

#### PRACTICA

Necesito un componente para poder mostrar los elementos seleccionados dentro de Un <select>, es decir, selección múltiple como vimos en React  
Tendremos una colección con los datos (string) que deseemos dibujar dentro del Select y, Al pulsar un botón, dibujaremos los elementos seleccionados.

- 1) Hacerlo solo con VUE
- 2) Instalar librerías de VUE

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



#### SELECCIONMULTIPLE.VUE

```

<template>
  <div>
    <h1>Selección múltiple</h1>
    <form v-on:submit.prevent="mostrarSeleccionados()">
      <select size="8" multiple v-model="opcionesSeleccionadas">
        <option v-for="opt in opciones" :key="opt">
          {{opt}}
        </option>
      </select><br/>
      <button>
        Mostrar seleccionados
      </button>
      <h6 style="color:blue">{{seleccionados}}</h6>
    </form>
  </div>
</template>
<script>
  export default {
    name: "SeleccionMultiple",
    methods: {
      mostrarSeleccionados() {
        let aux = "";
        for (let option of this.opcionesSeleccionadas){
          aux += option + ", ";
        }
        this.seleccionados = aux;
      }
    },
    data() {
      return {
        opcionesSeleccionadas: []
        , opciones: []
        , seleccionados: ""
      }
    },
    created(){
      for (var i = 1; i <= 10; i++){
        this.opciones.push("Elemento " + i);
      }
    }
  }
</script>

```

```

        }
    }
</script>

```

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



Implementamos Routing dentro del proyecto actual

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

Creamos un **HomeComponent** y navegamos a cualquier ejemplo que tengamos mediante un **MenuComponent**

#### ROUTER.JS

```

import { createWebHistory, createRouter } from "vue-router";
import HomeComponent from './components/HomeComponent.vue';
import PadreDeportes from './components/PadreDeportes.vue';
const myRoutes = [
  {
    path: "/",
    component: HomeComponent
  },
  {
    path: "/deportes",
    component: PadreDeportes
  }
]
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'

var app = createApp(App)
app.use(router).mount("#app")

```

#### RUTAS CON PARAMETROS

Las rutas con parámetros son igual que en React, es decir, enviar información mediante URL a un componente.

En el **path** de la ruta se declaran las posibles variables mediante **dos puntos**

```
{
  path: "/deportes/:id",
  component: PadreDeportes
}
```

Dentro de una misma ruta, podríamos tener también parámetros opcionales, es decir, rutas Con parámetros y sin parámetros para un mismo componente.  
Se utiliza la interrogación para indicar rutas con parámetros opcionales.

```
{
  path: "/deportes/:id?", 
  component: PadreDeportes
}
```

Para enviar parámetros en la ruta, debemos hacerlo mediante **<router-link>**

```
<router-link to="/deportes/14">Deportes</router-link>
```

La recuperación de los parámetros es muy parecida a la sintaxis con **\$emit**

Para recuperar parámetro/s se debe realizar dentro del método **mounted()** del componente

Para recuperar el valor de un parámetro utilizamos **\$route**

```
this.$route.params.NOMBREPARAMETRO
```

Para comprobar el ejemplo, vamos a crear un nuevo componente llamado **NumeroDoble.vue**

Dicho componente recibirá, en algún momento, número o NO (opcional).

Incluimos, dentro de **Router.js** una ruta con un parámetro opcional para nuestro Component

```

const myRoutes = [
  {
    path: "/", component: HomeComponent
  },
  {
    path: "/numerodoble/:numero?", component: NumeroDoble
  },
]

```

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

```

<li>
  <router-link to="/numerodoble">Número doble SIN</router-link> |
</li>
<li>
  <router-link to="/numerodoble/77">Doble 77</router-link> |
</li>

```

Una vez que lo tenemos montado, vamos a recibir parámetros dentro de **mounted()**

Dichos parámetros serán opcionales en este ejemplo

Necesitamos **data()** para el dibujo y necesitamos **mounted()** para recuperar el parámetro

#### NUMERODOBLE.VUE

```

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

```

Una vez que hemos visto que todo funciona, el component no se "redibuja" si no

Cambiamos entre Components. Lo mismo que sucedía en React.

Debemos de hacer algo al estilo **componentDidUpdate()**

Debemos de supervisar los cambios que se realizan en mi página.

Para ello, tenemos otra nueva "ZONA" llamada **watch**

Dicha zona nos permite controlar los cambios en la página una vez que el component ya está presente en el dibujo.

Debemos de indicar que supervisará un parámetro.

```

watch: {
  someParamRouting(nextVal, oldVal) {
    nextVal: El valor que ha tomado nuevo nuestro parámetro
    oldVal: El valor anterior que tenía el parámetro
  }
}

```

Para indicar que vamos a supervisar el valor de un parámetro, NO se indica la palabra **this**

Y debemos escribir entre comilla simple nuestro nombre de parámetro.

```

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

```

```

watch: {
  '$route.params.numero' (nextVal, oldVal){
    //SOLAMENTE DEBEMOS REALIZAR LAS ACCIONES
    //CUANDO NUESTRO PARAMETRO CAMBIE
    if (nextVal != oldVal){
      this.mensaje = "Esto ha cambiado!!! " +
      this.$route.params.numero;
      let valor = parseInt(this.$route.params.numero);
      this.doble = valor * 2;
    }
  }
},

```

[Home](#) | [Número doble SIN](#) | [Doble 77](#) | [Deportes](#)

## Número doble Routes

Doble: 154

Esto ha cambiado!!! 77



Por último, vamos a visualizar cómo podemos realizar la navegación por código dentro de un Component. Lo mismo que <Navigate/> en React.

Vamos a incluir un botón y un método para navegar a home.

La sintaxis es la siguiente:

```
this.$router.push(URL)
```

```

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

```

### NUMERODOBLE.VUE

```

<template>
  <div>
    <h1>Número doble Routes</h1>
    <button @click="redirectToHome()">
      Go to Home
    </button>
    <h3 style="color:red">
      Doble: {{doble}}
    </h3>
    <h2 style="color:blue">
      {{mensaje}}
    </h2>
  </div>
</template>
<script>
  export default {
    name: "NumeroDoble",
    methods: {
      redirectToHome() {
        this.$router.push("/");
      }
    },
    watch: {
      '$route.params.numero' (nextVal, oldVal){
        //SOLAMENTE DEBEMOS REALIZAR LAS ACCIONES
        //CUANDO NUESTRO PARAMETRO CAMBIE
        if (nextVal != oldVal){
          this.mensaje = "Esto ha cambiado!!! " +
          this.$route.params.numero;
          let valor = parseInt(this.$route.params.numero);
          this.doble = valor * 2;
        }
      }
    },
    data() {
      return {
        mensaje: "",
        doble: 0
      }
    },
    mounted() {
      console.log("Param: " + this.$route.params.numero);
      //LOS PARAMETROS SON STRING, NO IMPORTA SI SON NUMERICOS
      let paramNumero = this.$route.params.numero;
      if (paramNumero == ""){
        console.log("Sin parámetros");
        this.mensaje = "Sin parámetros en Routing";
      }else {
        this.mensaje = "Parámetro recibido: " + paramNumero;
        let valor = parseInt(this.$route.params.numero);
      }
    }
  }

```

```

        this.doble = valor * 2;
    }
}
</script>

```

En el component Menu creamos los Links dinámicos.

Ese valor 55 lo recuperamos de un Array que declaremos en el propio Component Menú

```

<li v-for=">
    <router-link to="/numerodoble/55">Doble 55</router-link> |
</li>

```

Si necesitamos enviar parámetros

```
<router-link :to="'/numerodoble/' + num">Doble {{num}}</router-link> |
```

#### TABLA MULTIPLICAR ROUTING

- Tendremos un Menú para dibujar los números aleatorios que serán parte de Links de la tabla de multiplicar, es decir, un component llamado **MenuTablaMultiplicar**
- Al seleccionar un Link, lo que haremos será mostrar la tabla de multiplicar en Otro component llamado **TablaMultiplicar** que recibirá el número mediante **Routing**

The screenshot shows a web page with a navigation bar at the top containing links to Home, Tabla 14, Tabla 100, Tabla 46, Tabla 40, and Tabla 68. Below the navigation bar, the main content area has a title "Tabla multiplicar 14". Underneath the title is a table with two columns: "Operación" and "Resultado". The table contains the following data:

Operación	Resultado
14 * 1	14
14 * 2	28
14 * 3	42
14 * 4	56
14 * 5	70

#### MENUTABLAMULTIPLICAR.VUE

```

<template>
  <div>
    <ul id="menu">
      <li>
        <router-link to="/">Home</router-link> |
      </li>
      <li v-for="num in numeros" :key="num">
        <router-link to="/tablamultiplicar/" + 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>

```

#### ROUTER.JS

```

import { createWebHistory, createRouter } from "vue-router";
import HomeComponent from './components/HomeComponent.vue';
import PadreDeportes from './components/PadreDeportes.vue';
import NumeroDoble from './components/NumeroDoble.vue'
import TablaMultiplicar from './components/TablaMultiplicar.vue'
const myRoutes = [
  {
    path: "/", component: HomeComponent
  },
  {
    path: "/numerodoble/:numero?", component: NumeroDoble
  },
  {
    path: "/deportes/:id?", component: PadreDeportes
  },
]

```

```

    {
      path: "/tablamultiplicar/:numero", component: TablaMultiplicar
    }
  ]
const router = createRouter({
  history: createWebHistory(),
  routes: myRoutes
})
export default router;

```

**TABLAMULTIPLICAR.VUE**

```

<template>
  <div>
    <h1>Tabla multiplicar {{numero}}</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",
    watch: {
      '$route.params.numero'(nextVal, oldVal){
        if (oldVal != nextVal){
          this.generarTabla();
        }
      }
    },
    data() {
      return {
        html: "",
        numero: 0
      }
    },
    methods: {
      generarTabla() {
        this.numero = parseInt(this.$route.params.numero);
        let aux = "";
        for (var i = 1; i <= 10; i++){
          var op = this.numero + " * " + i;
          var res = this.numero * i;
          aux += "<tr>";
          aux += "<td>" + op + "</td>";
          aux += "<td>" + res + "</td>";
          aux += "</tr>";
        }
        this.html = aux;
      },
      mounted() {
        this.generarTabla();
      }
    }
  }
</script>

```

## SERVICIOS VUE

VUE no tiene una forma nativa de leer los servicios API.  
Necesitamos una librería para leer dichos servicios.

Las dos más utilizadas:

- 1) Axios
- 2) fetch

Para los servicios tenemos algo conocido

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

También vamos a incluir el concepto de **Global.js**

Comenzamos creando un nuevo proyecto llamado **vueservicios**

Instalamos la librería de **axios**

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

Comenzamos leyendo un clásico

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

Comenzamos creando un nuevo componente llamado **CochesComponent**

# Servicio Api Coches

## PONTIAC 1 FIREBIRD 1

Conductor: MICHAEL KNIGHT 1



### COCHESCOMPONENT

```
<template>
  <div>
    <h1>Servicio Api Coches</h1>
    <div v-for="car in coches" :key="car">
      <h2 style="color:blue">
        {{car.marca}} {{car.modelo}}
      </h2>
      <p>Conductor: {{car.conductor}}</p>
      
    </div>
  </div>
</template>
<script>
import axios from 'axios';
//SI NECESITAMOS VARIABLES DECLARADAS PARA UTILIZARLAS EN TODOS
//LOS METODOS (mounted, created, methods) LA DECLARAMOS AQUI
let urlApiCoches = "https://apicochespaco.azurewebsites.net/";
export default {
  name: "CochesComponent",
  data() {
    return {
      coches: []
    }
  },
  mounted() {
    let request = "webresources/coches";
    //LAS VARIABLES DECLARADAS POR ENCIMA DE export default
    //NO UTILIZAN LA PALABRA this
    let url = urlApiCoches + request;
    axios.get(url).then(response => {
      console.log("Leyendo servicio");
      this.coches = response.data;
    })
  }
}
</script>
```

A continuación, vamos a implementar la funcionalidad de **Global.js**

```
var Global = {
  Key:value
}
```

```
export default Global;
```

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

### GLOBAL.JS

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

export default Global;
```

Utilizamos global dentro de **CochesComponent**

```

<script>
import axios from 'axios';
import Global from '../../Global';

export default {
  name: "CochesComponent",
  data() {
    return {
      coches: []
    }
  },
  mounted() {
    let request = "webresources/coches";
    //LAS VARIABLES DECLARADAS POR ENCIMA DE export default
    //NO UTILIZAN LA PALABRA this
    let url = Global.urlApiCoches + request;
    axios.get(url).then(response => {

```

Quiero leer a continuación los Customers

<https://northwind.netcore.io/>

Al cargar la página, mostramos el nombre y compañía de todos los Customers.

Tendremos una caja de texto para buscar un Customer y ver sus detalles.

CustomersComponent.vue

## Customers Component

Id Cliente  Buscar Customer

Nombre: Ana Trujillo

Compañía: Ana Trujillo Emparedados y helados

Título: Owner

Nombre	Empresa
Maria Anders	Alfreds Futterkiste
Ana Trujillo	Ana Trujillo Emparedados y helados
Antonio Moreno	Antonio Moreno Taquería
Thomas Hardy	Around the Horn
Christina Berglund	Röd lund snabbköp

CUSTOMERSCOMPONENT.VUE

```

<template>
  <div>
    <h1>Customers Component</h1>
    <form v-on:submit.prevent="buscarCustomer()">
      <label>Id Cliente</label>
      <input type="text" v-model="idCustomer"/>
      <button>
        Buscar Customer
      </button>
    </form>
    <div v-if="customer">
      <h2>Nombre: {{ customer.contactName }}</h2>
      <h2>Compañía: {{customer.companyName}}</h2>
      <h2>Título: {{customer.contactTitle}}</h2>
    </div>
    <table border="1">
      <thead>
        <tr>
          <th>Nombre</th>
          <th>Empresa</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="customer in customers" :key="customer">
          <td>{{customer.contactName}}</td>
          <td>{{ customer.companyName }}</td>
        </tr>
      </tbody>
    </table>
  </div>
</template>
<script>
import axios from 'axios';
import Global from '../../Global';
export default {
  name: "CustomersComponent",
  data() {

```

```

        return {
          customers: [],
          idCustomer: "",
          customer: null
        }
      },
      mounted() {
        let request = "customers.json";
        let url = Global.urlApiCustomers + request;
        axios.get(url).then(response => {
          console.log("leyendo");
          this.customers = response.data.results;
        })
      },
      methods: {
        buscarCustomer() {
          let request = "customers/" + this.idCustomer + ".json";
          let url = Global.urlApiCustomers + request;
          axios.get(url).then(response => {
            console.log("Buscando customer");
            this.customer = response.data.customer;
          })
        }
      }
    }
  
```

## UTILIZAR BOOTSTRAP CON VUE

Simplemente debemos utilizar las librerías de bootstrap.

Tenemos dos formas, o trabajamos en LOCAL o en la NUBE.

En local, almacenamos las librerías JS y CSS de bootstrap y hacemos referencia a ellas dentro De un componente principal dentro de style.

```

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

```

Vamos a utilizar los Links propios de Bootstrap en la nube y, dichos enlaces se incluyen En la página HTML principal, que en nuestro ejemplo se llama **index.html**.

Vamos a la página de Bootstrap y copiamos los Links de acceso del Framework

The screenshot shows the Bootstrap documentation page with the heading 'Include via CDN'. Below it, there is explanatory text about including Bootstrap's compiled CSS or JS via CDN, mentioning jsDelivr and quick\_start. Two code snippets are provided:

```

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/...>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/...

```

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@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScript</strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
  </body>
</html>

```

Vamos a continuar con otro servicio de Empleados.

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

Al iniciar la página, tendremos un control <select> dónde cargaremos los apellidos de los Empleados.

Al seleccionar un determinado Empleado, mostraremos sus detalles

```

var Global = {
  urlApiCoches: "https://apicochespaco.azurewebsites.net/",
  urlApiCustomers: "https://northwind.netcore.io/",
  urlApiEmpleados: "https://apiempleadosfullstack.azurewebsites.net/"
}

export default Global;

```

Creamos un nuevo componente llamado `EmpleadosDetalle.vue`

The screenshot shows a user interface for viewing employee details. At the top, it says "Empleados details". Below that is a placeholder "Seleccione empleado". A dropdown menu is open, showing "GUTIERREZ" as the selected item. A blue button labeled "Detalles empleado" is visible. To the right, the employee's details are listed: "Oficio: ANALISTA", "Salario: 219003", and "Departamento: 20".

EMPLEADOSDETALLE.VUE

```

<template>
  <div>
    <h1>Empleados details</h1>
    <form v-on:submit.prevent="buscarEmpleado()">
      <label>Seleccione empleado</label>
      <select class="form-control" v-model="idEmpleado">
        <option v-for="emp in empleados" :key="emp.id">
          {{emp.apellido}}
        </option>
      </select>
    </form>
  </div>

```

```

</select>
<button class="btn btn-info">
    Detalles empleado
</button>
</form>
<hr/>
<div v-if="empleado">
    <div>
        <dt>Oficio: </dt>
        <div>{{empleado.oficio}}</div>
        <dt>Salario: </dt>
        <div>{{empleado.salario}}</div>
        <dt>Departamento: </dt>
        <div>{{empleado.departamento}}</div>
    </div>
</div>
</div>
</template>
<script>
import Global from '../Global';
import axios from 'axios';
export default {
    name: "EmpleadosDetalle",
    methods: {
        buscarEmpleado() {
            let request = "api/empleados/" + this.idEmpleado;
            let url = Global.urlApiEmpleados + 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 = Global.urlApiEmpleados + request;
            axios.get(url).then(response => {
                console.log("Leyendo empleados");
                this.empleados = response.data;
            })
        }
    }
}
</script>

```

El siguiente ejemplo que vamos a realizar será sobre el mismo Servicio Api. Mediante un menú de bootstrap, lo que haremos será cargar todos los oficios y Mostrarímos los empleados mediante Routing.

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

The screenshot shows a Vue.js application interface. At the top, there is a dark navigation bar with links: "Expand at sm", "Home", "Coches", "Customers", "Empleados detalle", and "Oficios". The "Oficios" link is highlighted with a mouse cursor. Below the navigation bar is a table titled "Empleados o". The table has columns: "Apellido", "Oficio", "Salario", and "Departamento". There are five rows of data. A context menu is open over the fifth row (SERRA, DIRECTOR, 390039, 20). The menu items are: ANALISTA, DIRECTOR, EMPLEADO, PRESIDENTE, and VENDEDOR.

Apellido	Oficio	Salario	Departamento
JIMENEZ	DIRECTOR	386500	20
NEGRO	DIRECTOR	370000	30
CEREZO	DIRECTOR	318539	10
SERRA	DIRECTOR	390039	20

Comenzamos creando un nuevo componente llamado **MenuComponent** y le ponemos un Navbar

EJEMPLO MENU

```

<nav class="navbar navbar-expand-sm navbar-dark bg-dark"
aria-label="Third navbar"
example">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">Expand at sm</a>
        <button class="navbar-toggler" type="button" data-bs-
        toggle="collapse" data-bs-target="#navbarsExample03" aria-
        controls="navbarsExample03" aria-expanded="false" aria-
        label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarsExample03">
            <ul class="navbar-nav me-auto mb-2 mb-sm-0">
                <li class="nav-item">
                    <a class="nav-link active" aria-current="page" href="#">
                        Home</a>
                </li>
            </ul>
        </div>
    </div>

```

```

        </li>
    <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
    </li>
    <li class="nav-item">
        <a class="nav-link disabled" aria-disabled="true">Disabled</a>
    </li>
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" data-bs-
        toggle="dropdown" aria-expanded="false">Dropdown</a>
        <ul class="dropdown-menu">
            <li><a class="dropdown-item" href="#">Action</a></li>
            <li><a class="dropdown-item" href="#">Another action</a></li>
            <li><a class="dropdown-item" href="#">Something else
                here</a></li>
        </ul>
    </li>
</ul>
<form role="search">
    <input class="form-control" type="search"
    placeholder="Search" aria-label="Search">
</form>
</div>
</div>
</nav>

```

Sobre src, creamos una nueva clase llamada Router.js

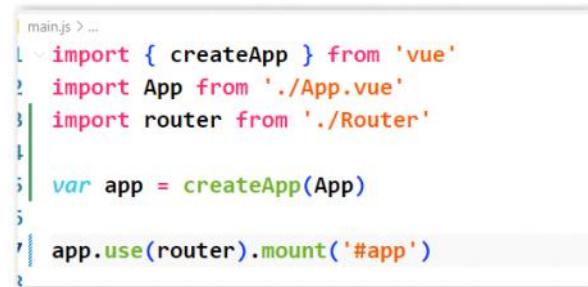
#### ROUTER.JS

```

import { createRouter, createWebHistory } from "vue-router";
import CochesComponent from './components/CochesComponent.vue'
import CustomersComponent from './components/CustomersComponent.vue'
import EmpleadosDetalle from './components/EmpleadosDetalle.vue'
import HomeComponent from './components/HomeComponent.vue'
import EmpleadosOficios from './components/EmpleadosOficios.vue'
const myRoutes = [
    {
        path: "/",
        component: HomeComponent
    },
    {
        path: "/empleadosoficios/:oficio",
        component: EmpleadosOficios
    },
    {
        path: "/coche",
        component: CochesComponent
    },
    {
        path: "/customers",
        component: CustomersComponent
    },
    {
        path: "/empleadosdetalle",
        component: EmpleadosDetalle
    }
]
const router = createRouter({
    history: createWebHistory(),
    routes: myRoutes
})
export default router;

```

#### MAIN.JS



```

main.js > ...
1 import { createApp } from 'vue'
2 import App from './App.vue'
3 import router from './Router'
4
5 var app = createApp(App)
6
7 app.use(router).mount('#app')
8

```

#### MENUCOMPONENT.VUE

```

<template>
    <div>
        <nav class="navbar navbar-expand-sm navbar-dark bg-dark" aria-
label="Third navbar example">
            <div class="container-fluid">
                <a class="navbar-brand" href="#">Expand at sm</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarsExample03" aria-controls="navbarsExample03" aria-
expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="collapse navbar-collapse" id="navbarsExample03">
                    <ul class="navbar-nav me-auto mb-2 mb-sm-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="/coches">Coches</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="/empleadosdetalle">Empleados
        detalle</router-link>
        </li>
        <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" href="#" data-bs-toggle="dropdown" aria-expanded="false">
        Oficios</a>
            <ul class="dropdown-menu">
                <li v-for="ofi in oficios" :key="ofi">
                    <router-link class="dropdown-item" :to="'/empleadosoficios/' + ofi">
                        {{ofi}}
                    </router-link>
                </li>
            </ul>
        </li>
    </ul>
</div>
</template>
<script>
import Global from '../Global'
import axios from 'axios'
export default {
    name: "MenuComponent",
    data() {
        return {
            oficios: []
        }
    },
    mounted() {
        let request = "api/empleados/oficios";
        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**

#### 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 Global from '../Global'
import axios from 'axios'
export default {
    name: "EmpleadosOficios",
    methods: {
        loadEmpleados() {
            let oficio = this.$route.params.oficio;
            let request = "api/empleados/empleadosoficio/" + oficio;
            let url = Global.urlApiEmpleados + request;
            axios.get(url).then(response => {
                console.log("Empleados oficio");
                this.empleados = response.data;
            })
        }
    },
    data() {
        return {
            empleados: []
        }
    },
    mounted() {
        this.loadEmpleados();
    },
    watch: {
        '$route.params.oficio'(nextVal, oldVal){
            if (nextVal != oldVal){
                this.loadEmpleados();
            }
        }
    }
}
</script>

```

#### SERVICIOS CON VUE

Un servicio es una clase que es capaz de administrar peticiones.  
Nos permite tener, en un mismo lugar todas las peticiones de nuestra aplicación

Actualmente, tenemos cada una de las peticiones separadas en Components.  
Si necesitamos reutilizar alguna petición, debemos volver a escribirla.

Un servicio no deja de ser una clase con métodos **return** que nos devolverán la información  
De los Apis.

Ejemplo de una clase Service:

```

export default class ServiceEjemplo {
    miMetodo1(){
        ...ACCIONES
    }
}

```

```

        return data;
    }

    miMetodo2() {
        ...ACCIONES
        return data;
    }
}

```

Los servicios se crean dentro de la carpeta **src** y crearemos una carpeta nueva llamada **services**  
Tienen una extensión **.js**

Para probar la funcionalidad, vamos a crear una carpeta llamada **services** y dentro una  
Clase llamada **ServiceEjemplo.js**

#### SERVICEEJEMPLO.JS

```

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

```

Vamos a visualizar, sobre **HomeComponent** cómo podemos hacer la llamada a un Service.

#### HOMECOMPONENT

```

<template>
    <div>
        <h1 style="color:blue">
            Home Component
        </h1>
        <h2 style="color:red">{{mensaje}}</h2>
    </div>
</template>
<script>
import ServiceEjemplo from './services/ServiceEjemplo';
//PARA PODER UTILIZAR EL SERVICIO, NECESITAMOS UNA CONSTANTE
//INSTANCIADA CON LA CLASE DEL SERVICIO NECESARIO
const service = new ServiceEjemplo();
export default {
    name: "HomeComponent",
    mounted() {
        this.mensaje = service.getSaludo("Alumno");
    },
    data() {
        return {
            mensaje: ""
        }
    }
}
</script>

```

El siguiente paso que vamos a realizar es migrar la lógica de **CochesComponent** a un Servicio.

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

#### SERVICECOCHES.JS

```

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

```

Modificamos el código de **CochesComponent.vue**

#### COCHESCOMPONENT.VUE

```

<template>
    <div>
        <h1>Servicio Api Coches</h1>
        <div v-for="car in coches" :key="car">
            <h2 style="color:blue">
                {{car.marca}} {{car.modelo}}
            </h2>
            <p>Conductor: {{car.conductor}}</p>
            
        </div>
    </div>
</template>
<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 tenemos coches.

Esto sucede por el **CUANDO** de nuestra petición.

Estamos llamando a un servicio asíncrono mediante un método síncrono (**getCoches()**)

Necesitamos crear un método asíncrono dentro de los servicios cuando llamamos a los Apis.  
Para ello, se utilizan **Promesas**. Una promesa es un método en espera de una respuesta como,  
Por ejemplo, realiza axios con **then**

Las promesas pueden recibir dos parámetros:

- 1) **Reject**: El código que devuelve un error en la petición a la promesa.
- 2) **Resolve**: Todo ha ido correcto y devuelve una respuesta OK

EJEMPLO DE PROMESA

```
getPromesa = new Promise(function(resolve, reject) {  
    //TENEMOS DOS POSIBILIDADES: CORRECTO O ERROR  
    let num = 0;  
    if (num == 0){  
        //DEVOLVEMOS CORRECTO  
        resolve("Ok, todo perfecto");  
    }else{  
        reject("Error en la promesa");  
    }  
})
```

Podemos realizar una promesa simple, sin error reject

```
getPromesa = new Promise(function(resolve) {  
    resolve("Todo ha ido perfecto");  
})
```

La petición se realiza de una forma muy conocida...

```
service.getPromesa().then(response => {  
    //DENTRO DE RESPONSE VIENE LA RESPUESTA QUE  
    //HEMOS INDICADO DENTRO DE (resolve)  
    console.log(response);  
})
```

Por último, debemos devolver objetos dentro de una promesa.  
Necesitamos un último paquete para instalar en el que nos permitirá devolver objetos desde una Promesa.

npm install core-js --save

Modificamos el código de **ServiceCoches** y creamos una promesa.

SERVICECOCHES

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

Modificamos la llamada 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 podemos enviar parámetros???

Si necesitamos una promesa como método, debemos modificar su sintaxis.  
O escribir siempre la misma sintaxis...

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

Posteriormente, en la petición, se llama como un método con parámetros...

```
service.getMetodoParametros("Hola mundo").then(result => {  
    console.log(result);  
})
```

Vamos a llevar todo el contenido de empleados a Promesas y servicios.

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

Sobre services, creamos una nueva clase llamada **ServiceEmpleados.js**

#### SERVICEEMPLEADOS

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

Vamos a por un clásico.

Realizaremos un CRUD sobre departamentos.

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

Creamos un nuevo proyecto llamado **vuecruddepartamentos**

Instalamos los siguientes paquetes:

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

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

Incluimos **bootstrap** dentro de **index.html**

```

index.html > <html> > <body> > <div>app</div>
<!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@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly w
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.j
  </body>
</html>

```

Creamos un nuevo componente llamado **DepartamentosComponent**

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

#### ROUTER.JS

```

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 Routing dentro de **main.js**

```

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

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

```

Creamos un Menú llamado **MenuComponent** y lo incluimos dentro de **App.vue**

El siguiente paso es crear, sobre **src** una clase llamada **Global.js**

#### GLOBAL.JS

```

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

export default Global;

```

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

#### SERVICEDEPARTAMENTOS.JS

```

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

```

Sobre `src/assets`, creamos una carpeta llamada `images`  
e incluimos una imagen para hacer el efecto óptico de loading

Implementamos la llamada al servicio dentro de `DepartamentosComponent.vue`

#### DEPARTAMENTOSCOMPONENT.VUE

```
<template>
  <div>
    <h1>Departamentos Component</h1>
    
    <table v-else class="table table-bordered">
      <thead>
        <tr>
          <th>Id departamento</th>
          <th>Nombre</th>
          <th>Localidad</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="dept in departamentos" :key="dept">
          <td>{{ dept.idDepartamento }}</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",
  mounted() {
    service.getDepartamentos().then(result => {
      this.status = true;
      this.departamentos = result;
    })
  },
  data() {
    return {
      departamentos: [],
      status: false
    }
  }
}
</script>
```

Creamos un nuevo componente llamado `CreateDepartamento.vue` y lo ponemos a jugar con las rutas y el menú.

Dentro del servicio, podría recibir el Id, nombre y localidad directamente como parámetros.  
En lugar de dicha acción, voy a recibir un objeto `departamento` en el método `POST`.

#### 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);
    })
  })
}
```

A continuación, escribimos la llamada dentro de `CreateDepartamento.vue`

#### CREATEDEPARTAMENTO.VUE

```
<template>
  <div>
    <h1>Create departamento</h1>
    <h2 style="color: blue">{{mensaje}}</h2>
    <form v-on:submit.prevent="insertDepartamento()">
      <label>Id departamento</label>
      <input type="number" v-model="departamento.idDepartamento"
        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>
  </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 correctamente " + result;
        this.$router.push("/");
      })
    }
  },
  data() {
    return {
      mensaje: "",
      departamento: {
        idDepartamento: 0,
        nombre: "",
        localidad: ""
      }
    }
  }
}
</script>
```

En el siguiente paso, no voy a utilizar el Servicio Api.  
Vamos a implementar Detalles, pero enviando los parámetros por la url con Rutas.

ROUTER.JS

```
{  
  path: "/details/:id/:nombre/:localidad", component: DetailsComponent  
}
```

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

La petición la realizamos desde la tabla de departamentos. **DepartamentosComponent.vue**

DEPARTAMENTOSCOMPONENT.VUE

```
<router-link class="btn btn-warning"  
 :to="'/details/' + dept.idDepartamento + '/' + dept.nombre + '/' + dept.localidad">  
   Details  
</router-link>
```

DETAILSDEPARTAMENTO.VUE

```
<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 la modificación de los departamentos.  
Vamos a crear una ruta inicial en la que enviaremos el ID.

Posteriormente, dentro del componente de **Update** lo que haremos será buscar el Departamento que corresponda a dicho ID y lo dibujaremos en las cajas.

Creamos un nuevo component llamado **UpdateDepartamento.vue**

ROUTER.JS

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

En **DepartamentosComponent** creamos la ruta **<router-link>** para comprobar si llegan los parámetros

DEPARTAMENTOSCOMPONENT

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

El siguiente paso es crear un método dentro del Service para llamar al Api y buscar por el ID del departamento que recibamos.

SERVICEDEPARTAMENTOS.JS

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

Implementamos la llamada dentro de **UpdateDepartamento** y dibujaremos el departamento encontrado en el Api con un Model.

UPDATEDEPARTAMENTO.VUE

```
<template>  
  <div>
```

```

<h1>Update departamento</h1>
<form v-if="departamento"
style="width: 500px; margin: 0 auto">
  <input type="hidden" v-model="departamento.idDepartamento"/>
  <label>Nombre</label>
  <input type="text" v-model="departamento.nombre"
  classe="form-control"/>
  <label>localidad</label>
  <input type="text" v-model="departamento.localidad"
  classe="form-control"/>
  <button class="btn btn-warning">Update</button>
</form>
</div>
</template>
<script>
import ServiceDepartamentos from '@services/ServiceDepartamentos'
const service = new ServiceDepartamentos();
export default {
  name: "UpdateDepartamento",
  mounted() {
    let id = this.$route.params.id;
    service.findDepartamento(id).then(result => {
      this.departamento = result;
    })
  },
  data() {
    return {
      departamento: null
    }
  }
}
</script>
<style>
</style>

```

El siguiente paso es realizar la acción del PUT (Update) en el servicio.  
Vamos a redireccionar cuando hagamos el Update.  
Recibiremos el objeto departamento en el Service.

#### SERVICEDEPARTAMENTOS.JS

```

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

```

Implementamos la llamada dentro de **UpdateDepartamento**

#### UPDATEDEPARTAMENTO

```

<template>
<div>
  <h1>Update departamento</h1>
  <form v-if="departamento" v-on:submit.prevent="updateDepartamento()">
    <input type="hidden" v-model="departamento.idDepartamento"/>
    <label>Nombre</label>
    <input type="text" v-model="departamento.nombre"
    classe="form-control"/>
    <label>localidad</label>
    <input type="text" v-model="departamento.localidad"
    classe="form-control"/>
    <button class="btn btn-warning">Update</button>
  </form>
</div>
</template>
<script>
import ServiceDepartamentos from '@services/ServiceDepartamentos'
const service = new ServiceDepartamentos();
export default {
  name: "UpdateDepartamento",
  methods: {
    updateDepartamento() {
      service.updateDepartamento(this.departamento).then(result => {
        console.log(result);
        this.$router.push("/");
      })
    }
  },
  mounted() {
    let id = this.$route.params.id;
    service.findDepartamento(id).then(result => {
      this.departamento = result;
    })
  },
  data() {
    return {
      departamento: null
    }
  }
}
</script>
<style>
</style>

```

Vamos a realizar la funcionalidad de eliminar.  
Por comodidad, vamos a realizar un nuevo componente para recibir el ID del Departamento a eliminar.  
Creamos un nuevo componente llamado **DeleteDepartamento.vue**

#### ROUTER.JS

```
{
  path: "/delete/:id", component: DeleteDepartamento
}
```

#### DEPARTAMENTOSCOMPONENT.VUE

```

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

```

Implementamos la funcionalidad de eliminar en el Service

#### SERVICEDEPARTAMENTOS.JS

```

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

```

Podríamos pedir confirmación dentro del propio componente de Delete.  
Lo que haremos será recuperar el ID de la ruta dentro de `mounted()` y Redireccionar a Home una vez que ha eliminado.

#### DELETEDEPARTAMENTO.VUE

```

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

```

MISMA PRACTICA QUE AYER, PERO CON COCHES

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

Necesito realizar un proyecto llamado `vuecrudcoches` con la misma funcionalidad

De ayer.

- Para eliminar, pedir confirmación con `Sweetalert`
- El servicio lo haremos con la librería `FETCH`, no con `AXIOS`

Coches	
<b>GET</b>	/api/Coches Obtiene el conjunto de COCHES, tabla COCHES.
<b>GET</b>	/api/Coches/FindCoche/{id} Obtiene un Coche por su Id, tabla COCHES.
<b>POST</b>	/api/Coches/InsertCoche Crea un nuevo Coche en la BBDD, tabla COCHES
<b>PUT</b>	/api/Coches/UpdateCoche Modifica un Coche en la BBDD, tabla COCHES
<b>GET</b>	/api/Coches/DeleteCoche/{id} Elimina un Coche en la BBDD mediante su ID. Tabla COCHES

```

fetchData() {
    var request = "api/departamentos";
    var url = Global.urlApiDepartamentos + request;
    fetch(url, {
        method: "GET",
    })
        .then((response) => {
            response.json().then((data) => {
                console.log(data);
            });
        })
        .catch((err) => {
            console.error(err);
        });
}

```

#### PRACTICA FULL STACK

- 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 implementamos.

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



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

STRANGER THINGS	
<a href="#">Home</a> · <a href="#">Nuevo personaje</a> · <a href="#">Modificar personajes</a> · <a href="#">Series</a> · <a href="#">Acerca de</a>	
<a href="#">Volver</a>	
Personaje	Imagen
Patriota	
Luz Estelar	

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.**

STRANGER THINGS	
<a href="#">Home</a> · <a href="#">Nuevo personaje</a> · <a href="#">Modificar personajes</a> · <a href="#">Series</a> · <a href="#">Acerca de</a>	
<h2>Nuevo personaje</h2>	
Nombre:	<input type="text" value="Profundo"/>
Imagen:	<input type="text" value="https://www.ecartelera.com/images/sets/44900/44983.jpg"/>
Serie:	<input type="text" value="The Boys"/> <a href="#">Insertar personaje</a>

Tendremos la posibilidad de cambiar a un personaje de serie:

- Modificar serie personaje:

STRANGER THINGS	
<a href="#">Home</a> · <a href="#">Nuevo personaje</a> · <a href="#">Modificar personajes</a> · <a href="#">Series</a> · <a href="#">Acerca de</a>	
<h2>Personajes y series</h2>	
Seleccione una serie:	<input type="text" value="The Mandalorian"/>
Seleccione un Personaje:	<input type="text" value="Koothrappali"/>
<a href="#">Guardar cambios</a>	
<b>The Mandalorian</b>	
<b>Koothrappali</b>	

# CHARLAS TECNICAS

martes, 5 de noviembre de 2024 9:09

## NO PODEMOS UTILIZAR RECURSOS DE CLASE

Para las charlas, será necesario generar dos elementos:

- 1) **POST:** Es la explicación de nuestro código en WordPress.  
Un paso a paso con lo que vayamos haciendo.  
Necesitaremos un Link a Github para el seguimiento del Código.  
Si tenéis LinkedIn, recomendable que lo pongáis también.
- 2) **VIDEO:** Es una explicación de lo que vayamos a realizar y Cómo. Pero no es toda la práctica completa, para eso está el POST. La duración que no sea mayor a 5/6 minutos.

Esta tarea es para casa.

Para grabar el vídeo, tenemos OBS o tenemos Camtasia.

OBS graba la pantalla, pero no permite ediciones.

Camtasia si permite ediciones y también nos permite efectos.

Para llevaros Camtasia, no podéis hacerlo en la nube, ya que Microsoft Bloquea la medicina.

El camtasia podemos recuperarlo de esta ruta:

<\\T06W26\Camtasia>

Elementos principales

- Definir una charla por cada alumno, al final qué sucede con los Post? Duplicamos el trabajo del alumno?
- Cada alumno podrá proponer su Charla al Profesor
- Las charlas podrán tener un Link, Vídeo y explicación detallada de lo que quieren realizar
- Contabilizar el número de charlas realizadas por alumno
- Pensar en los "descartes" de las charlas que no han salido adelante
- Las charlas deben tener más o menos un tiempo de exposición
- Las charlas aparecerán agrupadas, sin la persona que lo ha propuesto
- Los alumnos podrán darse de alta en la app
- Votación de Charlas anónimas
- Incluir validación para cada alumno.
- Los alumnos podrán votar? Bueno o malo?
- Establecer días para las charlas mediante Calendario
- Como hacemos con los recursos de las charlas??
- Comunicación????
- Admin principal para las funcionalidades extra
- Publicación de charlas???

Funcionalidades extra

- Agrupación de sitios/ordenadores por alumno/calendario
- Buena idea lo de moverse, pero dilatar algo más en el tiempo
- Nuestro problema es GitHub, Teams ya está solucionado más o menos.
- Generar informe de los sitios asignados y guardar las fechas
- Grupos de proyectos

Día ONLINE para dar de alta el módulo de Power Platform.

# ANGULAR

miércoles, 6 de noviembre de 2024 9:04

Angular es el Framework SPA oficial de Google.

Tenemos los mismos conceptos que los Frameworks anteriores. No tiene state y la organización De los componentes es distinta, ya que se separan en ficheros distintos.

Utiliza el lenguaje **TypeScript** que es lo mismo que Javascript, pero con tipado de datos.

Comenzamos instalando Angular en nuestro equipo

**npm install -g @angular/cli@latest**

Y dentro de VS Code, instalamos la siguiente extensión



Comandos para trabajar con los proyectos

- 1) Creación de proyectos **ng new**
- 2) Lanzar el servidor: **ng serve**

Para crear nuestro primer proyecto utilizamos este comando.

**ng new primerangular --standalone=false**

```
C:\Users\Profesor MCSD Mañana\Documents\FULLSTACK\angular
λ ng new primerangular --standalone=false
? Which stylesheet format would you like to use? (Use arrow keys)
  CSS [ https://developer.mozilla.org/docs/Web/CSS
  ]
  Sass (SCSS) [ https://sass-lang.com/documentation/syntax#scss
  ]
  Sass (Indented) [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less [ http://lesscss.org

```

```
λ ng new primerangular --standalone=false
? Which stylesheet format would you like to use? CSS [ https://developer.mozilla.org/docs/Web/CSS
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? (y/N) [
```

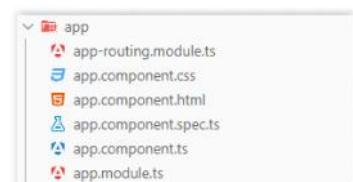
La base de angular son los components. Un component finaliza con la extensión **.ts**

A diferencia de otros frameworks, el component está separado en tres zonas.

- 1) **TS:** El código lógico de los scripts
- 2) **HTML:** Las vistas de los dibujos
- 3) **CSS:** Los estilos de nuestro component

Como son tres ficheros, los components se organizan en carpetas

Dentro de **src/app**



La separación de código es lo importante, abrimos **app.component.ts**

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'
})
export class AppComponent {
  title = 'primerangular';
}
```

Tenemos los siguientes elementos dentro de un component TS:

- **Selector:** Es el nombre del component para ser dibujado
- **templateUrl:** El código del dibujo del component
- **styleUrls:** La hoja/s de estilo CSS

Elementos básicos de Angular: **app.module.ts**

**APP.MODULE.TS:** Es el jefe de Angular. En esta clase es donde debemos declarar todo lo que tengamos dentro de nuestro proyecto, desde cada component, hasta habilitar Routing o Forms.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Nosotros vamos a trabajar igual que con otros Frameworks, es decir, tendremos una carpeta llamada **components** dentro de **src**

Como cada component está compuesto por varios ficheros, tendremos una carpeta a su vez para cada component individual.

Cada component puede contener variables.

Dichas variables con tipados serán declaradas dentro de la zona de **export class**

Dichas variables podemos utilizarlas en nuestros dibujos mediante `{}{ variable }`

Sobre **src**, creamos una nueva carpeta llamada **components**

Dentro de components una carpeta llamada **primercomponent**

Por ahora, solo vamos a tener el propio component, por lo que creamos un fichero llamado **primer.component.ts**.

**PRIMER.COMPONENT.TS**

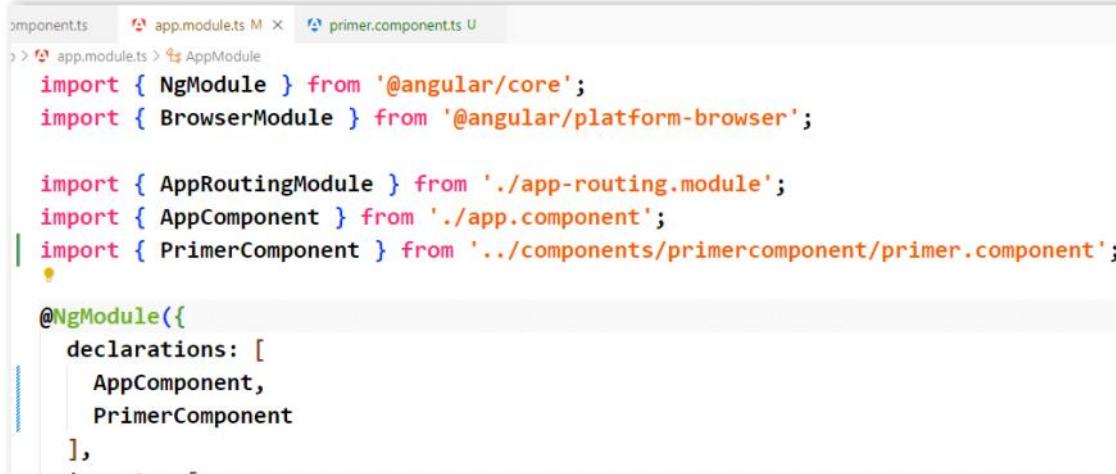
```
import { Component } from "@angular/core";
//UN COMPONENT TENDRA UNA DECLARACION DE SU
CONTENIDO
@Component ({
  //DEBEMOS DE DECLARA EL NOMBRE DEL COMPONENT
  //EN ANGULAR, SE SUELEN LLAMAR MEDIANTE GUION
  selector: "primer-component",
  //COMO NO VAMOS A TENER HTML (VISTA) COMO
  TEMPLATE
  //VAMOS A ESCRIBIR DIRECTAMENTE EL CODIGO
  HTML AQUI
  template: `
    <h1>Soy el primer component de
    Angular!!!</h1>
  `
})
//TODO COMPONENT DEBE SER DECLARADO COMO UNA
CLASE
//Dicho NOMBRE DE CLASE SE UTILIZARA DENTRO DE
app.module.ts
export class PrimerComponent {
  //AQUI DECLARAREMOS LAS VARIABLES QUE
  DESEEMOS
  //DICHAS VARIABLES SON CON TIPADO
  public titulo: string;
  public descripcion: string;
  public anyo: number;
  //EN ANGULAR TENEMOS UN CONSTRUCTOR PARA
```

## INICIAR LAS VARIABLES

```
//O RECUPERARLAS DE ALGUN SITIO
constructor() {
    //AQUI PARA ACCEDER A LAS PROPIEDADES DE
UNA CLASE
    //SE UTILIZA LA PALABRA this
    this.titulo = "Hoy es miércoles";
    this.descripcion = "Hoy no llueve";
    this.anio = 2024;
}
}
```

Para poder utilizar el componente, debemos declararlo dentro de **app.module.ts**

## APP.MODULE.TS



```
component.ts  ❌ app.module.ts M ✘ primer.component.ts U
> ❌ app.module.ts > AppModule
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { PrimerComponent } from '../components/primercomponent/primer.component';

@NgModule({
  declarations: [
    AppComponent,
    PrimerComponent
  ],
  ...
})
```

Para dibujar nuestro componente, por ahora lo vamos a integrar dentro del código de **app.component.html**

```
<h1>Hello, {{ title }}</h1>
<p>Congratulations! Your app is running. 🎉</p>
<primer-component></primer-component>
</div>
```

Por supuesto, podemos dibujar las variables dentro de nuestro **template**

## PRIMER.COMPONENT.TS

```
//VAMOS A ESCRIBIR DIRECTAMENTE EL CODIGO HTML AQUI
template: `
  <h1>Soy el primer componente de Angular!!!</h1>
  <h2 style="color:blue">
    {{ titulo }}, {{ descripcion }}, {{ anio }}
  </h2>
`
```

Y ya podremos visualizar nuestro dibujo con las variables



# Hello, primerangular

Congratulations! Your app is running. 🎉

**Soy el primer component de Angular!!!**

**Hoy es miércoles, Hoy no llueve, 2024**

Vamos a separar el template (vista) del código del Component.

Sobre components/primercomponent creamos un fichero llamado primer.component.html

PRIMER.COMPONENT.HTML

```
<div>
  <h1>Soy el primer component de Angular!!!</h1>
  <h2 style="color: blue">
    {{ titulo }}, {{ descripcion }}, {{ anio }}
  </h2>
</div>
```

Realizamos la llamada a la vista desde el Component

```
@Component ({
  //DEBEMOS DECLARA EL NOMBRE DEL COMPONENT
  //EN ANGULAR, SE SUELEN LLAMAR MEDIANTE GUION
  selector: "primer-component",
  //COMO NO VAMOS A TENER HTML (VISTA) COMO TEMPLATE
  //VAMOS A ESCRIBIR DIRECTAMENTE EL CODIGO HTML AQUI
  templateUrl: "./primer.component.html"
})
```

Por último, también tenemos la posibilidad de indicar CSS para cada component.

Sobre la carpeta components/primercomponent creamos un nuevo fichero llamado primer.component.css

PRIMER.COMPONENT.CSS

```
h1 {
  background-color: lightgreen;
}

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

```
@Component ({
  //DEBEMOS DECLARA EL NOMBRE DEL COMPONENT
  //EN ANGULAR, SE SUELEN LLAMAR MEDIANTE GUION
  selector: "primer-component",
  //COMO NO VAMOS A TENER HTML (VISTA) COMO TEMPLATE
  //VAMOS A ESCRIBIR DIRECTAMENTE EL CODIGO HTML AQUI
  templateUrl: "./primer.component.html",
  styleUrls: ["./primer.component.css"]
})
```

## HOOKS EN ANGULAR

Los Hooks en estos frameworks son los ciclos de vida de nuestros componentes, es decir, Hablamos de `componentDidMount()` de React o `mounted()` de Vue.

Nos permiten controlar los cuandos en las acciones de un componente:

- **ngOnInit:** Es el primer método que se ejecuta después del constructor. Para utilizarlo, debemos implementar `OnInit` en nuestra clase Component. El constructor se utiliza para inicializar las variables. Este método se utilizar para trabajar con dichas variables iniciadas.
- **ngDoCheck:** Se ejecuta cada vez que tenemos un cambio dentro de la vista. Para implementarlo necesitamos utilizar `DoCheck`
- **ngOnDestroy:** Se ejecuta cuando el componente es eliminado del Parent

Vamos a crear una carpeta llamada `hooksangular` y dentro un componente llamado `hooksangular.component.ts`

### HOOKSANGULAR.COMPONENT.TS

```
import { Component, OnInit, DoCheck } from "@angular/core";
@Component({
  selector: "hooks-angular",
  templateUrl: "./hookangular.component.html"
})
export class HooksAngular implements OnInit {
  public mensaje: string;
  constructor() {
    console.log("Soy el constructor de Hooks Angular");
    this.mensaje = "Soy Hooks en Angular";
  }
  cambiarMensaje(): void {
    this.mensaje = "Cambiando en un CLICK!!!";
  }
  ngOnInit(): void {
    console.log("Soy el metodo NgOnInit");
  }
  ngDoCheck(): void {
    console.log("Ejecutando método ngDoCheck");
  }
}
```

### HOOKSANGULAR.COMPONENT.HTML

```
<div>
  <h1 style="color:blue">Hooks Angular</h1>
  <button (click)="cambiarMensaje()">
    Cambiar mensaje
  </button>
  <h2 style="color:red">
    {{mensaje}}
  </h2>
</div>
```

Y veremos los cambios al pulsar el botón

Entrada	Fuente
Soy el constructor de Hooks Angular	hooksangular.component.ts:12
Soy el metodo NgOnInit	hooksangular.component.ts:12
Ejecutando método ngDoCheck	hooksangular.component.ts:21
Angular is running in development mode.	core.js:30060
Ejecutando método ngDoCheck	hooksangular.component.ts:25
Ejecutando método ngDoCheck	hooksangular.component.ts:25
7 Ejecutando método ngDoCheck	hooksangular.component.ts:25

## DIRECTIVAS DE ANGULAR

Una directiva es código lógico dentro de la vista HTML.

- **ngIf:** Condicional
- **ngFor:** Bucle. En la declaración del recorrido debemos utilizar la palabra `let`

### let objeto in objetos

Como en VUE, se utilizan integrados dentro de las propias etiquetas HTML

Para indicar que estamos utilizando una directiva de angular, se pone con un Asterisco antes de la directiva

```
<div *ngIf="condicion">
```

Vamos a realizar un clásico, un componente `deportes`

Dentro de `components` creamos una nueva carpeta llamada `deportes` y dos ficheros `deportes.component.ts` y `deportes.component.html`

#### DEPORTES.COMPONENT.TS

```
import { Component } from "@angular/core";
@Component({
  selector: "app-deportes",
  templateUrl: "./deportes.component.html"
})
export class DeportesComponent {
  public sports: Array<string>;
  constructor() {
    this.sports = ["Canicas", "Padel", "Petanca", "Curling", "Dardos"]
  }
}
```

#### DEPORTES.COMPONENT.HTML

```
<div>
  <h1>Directivas angular</h1>
  <h2>Número de deportes {{ sports.length }}</h2>
  <!-- DIRECTIVA IF -->
  <div *ngIf="sports.length < 6">
    <p style="color:red">
      Tienes menos de 6 deportes
    </p>
  </div>
  <p>El primer deporte es: {{ sports[0] }}</p>
  <ul>
    <li *ngFor="let deporte of sports; let i = index">
      Index: {{ i }}, Deporte: {{deporte}}
    </li>
  </ul>
</div>
```

Y podremos visualizar el resultado:

## Directivas angular

### Número de deportes 5

Tienes menos de 6 deportes

El primer deporte es: Canicas

- Index: 0, Deporte: Canicas
- Index: 1, Deporte: Padel
- Index: 2, Deporte: Petanca
- Index: 3, Deporte: Curling
- Index: 4, Deporte: Dardos

No existe ELSE ni ELSE IF tradicionales como hemos visto en VUE.

Para poder realizar estas acciones tenemos una directiva de plantilla que nos permite separar el código de las condiciones.

Dicha directiva se llama **ngTemplate**

Con un **ID** llamamos a cada plantilla deseada. Los IDs dentro de Angular se establecen mediante #

Ejemplo:

```
<ng-template #templatetrue>
  <h2>TRUE</h2>
<ng-template>
<ng-template #templatefalse>
  <h2>FALSE</h2>
<ng-template>
```

Posteriormente, dentro de una directiva IF, podemos indicar qué plantilla queremos dibujar

```
ngif condicion == true #templatetrue; else #templatefalse
```

Si no tenemos plantilla para el TRUE, el IF utiliza el código que tengamos en la etiqueta HTML

Modificamos el código que tenemos con el IF de length

```
<!-- DIRECTIVA IF -->
<div *ngIf="sports.length < 6; else condicionelse">
  <p style="color: red">
    Tienes menos de 6 deportes
  </p>
</div>
<ng-template #condicionelse>
  <p style="color: blue">
    Tenemos más de 6 deportes!!!
  </p>
</ng-template>
```

Si deseamos realizar `else if`, debemos tener más plantillas y la palabra `then` para cada condición.

`ngif condicion == 1; then PLANTILLA1; condicion == 2; then PLANTILLA2; else PLANTILLAELSE`

```
<!-- DIRECTIVA IF -->
<div *ngIf="sports.length < 6; then condiciontrue; else condicionelse">
</div>
<ng-template #condiciontrue>
  <p style="color: red">
    Tienes menos de 6 deportes
  </p>
</ng-template>
<ng-template #condicionelse>
  <p style="color: blue">
    Tenemos más de 6 deportes!!!
  </p>
</ng-template>
```

También tenemos directivas para los estilos CSS, igual que hicimos con comics en Vue y con El año.

Dicha directiva se llama `ngStyle` y podemos utilizarlo tanto `INLINE` como mediante `CSS`

`INLINE`

```
<h2 [style.color]="condicion ? 'valor true'; 'valor false'">
```

Vamos a realizar un ejemplo en el que tendremos múltiples números en un Array.

Evaluamos dichos números si son Par o Impar y cambiaremos su estilo dependiendo de dicha Pregunta.

Realizamos la declaración de un Array de números para el ejemplo.

```
export class DeportesComponent {
  public numeros: Array<number>;
  public sports: Array<string>;
  constructor() {
    this.numeros = [4,5,6,7,8,9,10,188];
    this.sports = ["Canicas", "Padel", "Peta
  }
}
```

Vamos a dibujar todos los números en una lista y cambiaremos su color, mediante `INLINE`, A Rojo o Verde dependiendo de Par o Impar

## Directivas angular

- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 188

```

<ul>
  <li *ngFor="let num of numeros"
    [style.color]="num % 2 == 0 ? 'green': 'red'">
    {{ num }}
  </li>
</ul>

```

En este ejemplo, la directiva INLINE está muy bien, pero estamos limitados por Cada style con su propiedad.

Podemos utilizar directivas apuntando a CSS.

```

CSS
.estilo1 {
  ...
}

.estilo2 {
  ...
}

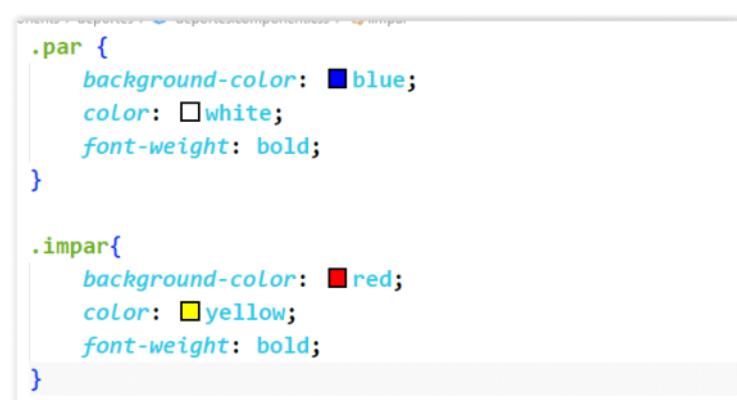
```

Posteriormente, para utilizar estos estilos en directivas se utiliza la sintaxis [class.estilo1]=condicion

```
<h1 [class.estilo1]=condicion1
  [class.estilo2]=condicion2>
```

Vamos a crear un nuevo CSS dentro de deportes llamado deportes.component.css

**DEPORTES.COMPONENT.CSS**



```

.par {
  background-color: blue;
  color: white;
  font-weight: bold;
}

.impar{
  background-color: red;
  color: yellow;
  font-weight: bold;
}

```

Importamos el CSS dentro de nuestro component y aplicamos la directiva de estilos Css dentro de la vista HTML

```

<ul>
  <li *ngFor="let num of numeros"
    [class.par]="num % 2 == 0"
    [class.impar]="num % 2 == 1">
    {{ num }}
  </li>
</ul>

```

Por último, tenemos otra sintaxis para realizar lo mismo  
Dicha sintaxis utiliza **ngClass** e indica, dentro del código, la clase CSS que utilizará

Ejemplo:

```
<h1 [ngClass]="{
  Estilo1: condicion1,
  Estilo2: condicion2
}">
```

```

<ul>
  <li *ngFor="let num of numeros"
    [ngClass]="{
      par: num % 2 == 0,
      impar: num % 2 == 1
    }">
    {{ num }}
  </li>
</ul>

```

El uso de formularios dentro de Angular, está integrado, pero no está activo  
Debemos indicar, dentro de **MODULE** que utilizaremos Forms.

Tenemos dos formas de trabajar con formularios en Angular:

- 1) **Model Binding:** Un modelo es un objeto con propiedades y podemos enlazarlo directamente a un control HTML, igual que con VUE
- 2) **Object Reference:** Es igual a React, declaramos una variable de referencia y Posteriormente la enlazamos en el código HTML.

Todos los controles de formulario en Angular deben tener un ID de Angular de forma obligatoria: **#idAngular**  
También debemos indicar que tendrán un **name**

El formulario donde estemos trabajando con los controles HTML debe tener la directiva **ngForm** para evitar **preventDefault**

**DEBEMOS ACTIVAR FORMS DENTRO DE ANGULAR**

**APP MODULE.TS**

```
app.module.ts M app.component.html M deportes.component.ts
> app.module.ts > ...
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    AppComponent,
    PrimerComponent,
    HooksAngular,
    DeportesComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    AppRoutingModule
  ],
})
```

Vamos a comenzar con **Binding Model**.

# Forms Binding Model

## Datos recibidos

Nombre:

Apellidos:

Edad:

**Nombre: Alumno, Apellidos: Angular, Edad: 26**

Creamos una carpeta llamada **formsbinding** y un componente y vista llamado **formsbinding.component.ts** y **formsbinding.component.html**

**FORMSBINDING.COMPONENT.TS**

```
import { Component } from "@angular/core";
@Component ({
  selector: "app-forms-binding",
  templateUrl: "./formsbinding.component.html"
})
export class FormsBindingComponent {
  // DECLARAMOS UN OBJETO PARA REALIZAR EL
  // BINDING EN UN FORM HTML
  public user: any;
  public mensaje: string;
  constructor(){
    this.mensaje = "";
    this.user = {
      nombre: "",
      apellidos: "",
      edad: 0
    }
  }
  recibirDatos(): void {
```

```

        this.mensaje = "Datos recibidos";
    }

}

FORMS BINDING COMPONENT.HTML

<div>
    <h1>Forms Binding Model</h1>
    <h2>{{mensaje}}</h2>
    <!-- EL FORMULARIO DEBE TENER UN ID DE ANGULAR IGUALADO A ngForm-->
    <form #userForm="ngForm" (ngSubmit)="recibirDatos()">
        <label>Nombre: </label>
        <!-- LOS CONTROLES DE FORMULARIO DEBEN TENER UN ID DE ANGULAR COMO VALOR
        ngModel-->
        <!-- PARA INDICAR EL MODEL BINDING DEBEMOS HACERLO CON LA SIGUIENTE
SINTAXIS
        [(ngModel)]="mi modelo"
        -->
        <input type="text" name="cajanombre"
        #cajanombre="ngModel"
        [(ngModel)]="user.nombre"/><br/>
        <label>Apellidos</label>
        <input type="text" name="cajaapellidos"
        #cajaapellidos="ngModel"
        [(ngModel)]="user.apellidos"/><br/>
        <label>Edad</label>
        <input type="number" #cajaedad="ngModel"
        name="cajaedad"
        [(ngModel)]="user.edad"/><br/>
        <button>
            Enviar datos
        </button>
    </form>
    <h2 style="color:blue">
        Nombre: {{user.nombre}},
        Apellidos: {{user.apellidos}},
        Edad: {{user.edad}}
    </h2>
</div>
```

Cuando incluimos algún tipo de validación HTML5 dentro de los input en Angular, No lo tiene en cuenta. Por ejemplo, si pusieramos **required** a una caja

Para las validaciones de los controles dentro de Angular, podemos incluir código Asociado al ID de angular **#id**

Tenemos una palabra clave llamada **valid** para saber si un control es válido o no.

Debemos combinar la palabra **valid** con **touched** por si el usuario ha incluido el foco en el Control o no.

Para la validación, se utilizan directivas **ngIf**

`ngIf="#idControl.valid == false"`

```

<input type="text" name="cajanombre" required
#cajanombre="ngModel"
[(ngModel)]="user.nombre"/>
<span style="color: red">
*ngIf="cajanombre.valid == false && cajanombre.touched">
    El nombre es obligatorio
</span>
<br/>

<button [disabled]="/>
    userForm.valid">
    Enviar datos
</button>
```

## FORMULARIOS REFERENCIA

Es exactamente igual que con React y su maravilloso **createRef()**

La gran ventaja radica en que no tenemos que escribir tanto código HTML mediante Los **ngModel**

La desventaja está en que si cambiamos el valor de las cajas, no cambia el modelo.

Lo que si es obligatorio es mantener los Ids de Angular

Pongamos que tenemos la siguiente caja en un Form HTML

`<input type="text" name="cajatexto" #cajatexto/>`

Mediante la librería **ViewChild** (tenemos que importarla) podremos enlazar el código HTML de la caja con un objeto declarado en Angular.

En el componente debemos declarar una variable de tipo ViewChild apuntando al ID De Angular.

`@ViewChild("cajatexto") nombreVariable: ElementRef;`

Posteriormente, para recuperar el valor de la caja:

`this.nombreVariable.nativeElement.value`

Vamos a realizar un ejemplo para sumar dos números con formularios de referencia.

# Forms Referencia

Número 1

Número 2

La suma es 11

Tenemos que mantener dos reglas:

- 1) El formulario debe tener **ngForm** para preventDefault
- 2) Cada control HTML debe tener un ID de Angular y un name

Creamos una nueva carpeta llamada **sumarnumeros** y un component **sumarnumeros.component.ts** y **sumarnumeros.component.html**

**SUMARNUMEROS.COMPONENT.TS**

```
import { Component, ViewChild, ElementRef } from "@angular/core";
@Component ({
  selector: "app-sumar-numeros",
  templateUrl: "./sumarnumeros.component.html"
})
export class SumarNumerosComponent {
  //DECLARAMOS VARIABLES PARA HACER REFERENCIA A LAS CAJAS
  //<input/> MEDIANTE SU ID DE ANGULAR
  @ViewChild("cajanumero1") cajaNumero1Ref: ElementRef;
  @ViewChild("cajanumero2") cajaNumero2Ref: ElementRef;
  public suma: number;
  //EN ANGULAR TODAS LAS PROPIEDADES DEBEN SER INSTANCIADAS/INICIADAS
  //LOS OBJETOS DE REFERENCIA DEBEMOS UTILIZAR UNA SINTAXIS
  //MEDIANTE SU CONSTRUCTOR (new) E INDICAR EL VALOR POR DEFECTO QUE
  //DESEAMOS QUE TENGAN LAS CAJAS
  constructor() {
    this.suma = 0;
    this.cajaNumero1Ref = new ElementRef();
    this.cajaNumero2Ref = new ElementRef();
  }
  sumarNumeros(): void {
    let num1 = this.cajaNumero1Ref.nativeElement.value;
    let num2 = this.cajaNumero2Ref.nativeElement.value;
    this.suma = parseInt(num1) + parseInt(num2);
  }
}
```

**SUMARNUMEROS.COMPONENT.HTML**

```
<div>
  <h1>Forms Referencia</h1>
  <form #sumarForm="ngForm">
    <label>Número 1</label>
    <input type="number" name="cajanumero1"
      #cajanumero1/><br/>
    <label>Número 2</label>
    <input type="number" name="cajanumero2"
      #cajanumero2/><br/>
    <button (click)="sumarNumeros()">
      Mostrar suma
    </button>
  </form>
  <h2 style="color:blue">
    La suma es {{suma}}
  </h2>
</div>
```

Como hemos visto, Angular nos obliga a inicializar todas las variables.

Si por cualquier razón no queremos iniciar la variable en el constructor, tenemos la posibilidad de crear el objeto sin necesidad de iniciarla.

La forma de realizarlo es mediante una **ADMIRACION** en el nombre de la variable

```
@ViewChild("cajanumero1") cajaNumero1Ref!: ElementRef;
@ViewChild("cajanumero2") cajaNumero2Ref!: ElementRef;
public suma!: number;
```

## CREACION DE COMPONENTS DINAMICOS

Hemos instalado una extensión que nos permite crear components de forma dinámica.

Debemos de indicar la carpeta dónde crearemos el component y su nombre

**ng g component CARPETA/COMPONENTNAME**

Podemos utilizar el terminal de VS code para generarlos dinámicamente.

```
C:\Users\Profesor MCSD Mañana\Documents\FULLSTACK\angular\primerangular>ng g component components/prueba
CREATE src/app/components/prueba/prueba.component.html (22 bytes)
CREATE src/app/components/prueba/prueba.component.spec.ts (620 bytes)
CREATE src/app/components/prueba/prueba.component.ts (209 bytes)
CREATE src/app/components/prueba/prueba.component.css (0 bytes)
UPDATE src/app/app.module.ts (1124 bytes)
```

Como hemos visto, nos genera el component dentro de la carpeta **app/components**

Tenemos el mismo comando para indicar la ruta exacta dónde deseamos crear un Component.

```
ng g component NOMBRECOMPONENT --path src/components --skip-import
```

#### TABLA MULTIPLICAR PRACTICA

Realizar un nuevo component llamado **tablamultiplicar** donde pediremos un número  
Mediante un Formulario y debemos dibujar una tabla con Operación y Resultado  
De dicho número.

Forms: Al gusto.

- Debemos hacerlo con un Array<number>
- El dibujo será dinámico con un **ngFor**, es decir, el Array solamente tendrá  
El resultado de las operaciones.

```
C:\Users\Profesor MCSD Mañana\Documents\FULLSTACK\angular\primerangular>ng g component tablamultiplicar --path src/components --skip-import
CREATE src/components/tablamultiplicar/tablamultiplicar.component.html (32 bytes)
CREATE src/components/tablamultiplicar/tablamultiplicar.component.spec.ts (600 bytes)
CREATE src/components/tablamultiplicar/tablamultiplicar.component.ts (249 bytes)
CREATE src/components/tablamultiplicar/tablamultiplicar.component.css (0 bytes)
```

## Tabla multiplicar

Número	<input type="text" value="7"/>	<button>Mostrar tabla</button>
Operación	Resultado	
7 * 1	7	
7 * 2	14	
7 * 3	21	
7 * 4	28	
7 * 5	35	
7 * 6	42	
7 * 7	49	
7 * 8	56	
7 * 9	63	
7 * 10	70	

#### TABLAMULTPLICAR.COMPONENT.TS

```
import { Component, ViewChild, ElementRef } from '@angular/core';
@Component({
  selector: 'app-tablamultiplicar',
  templateUrl: './tablamultiplicar.component.html',
  styleUrls: ['./tablamultiplicar.component.css']
})
export class TablamultiplicarComponent {
  @ViewChild("cajanumero") cajaNumeroRef!: ElementRef;
  public numeros: Array<number>;
  public numero: number;
  constructor() {
    this.numeros = new Array<number>();
    this.numero = 0;
  }
  mostrarTabla(): void {
    this.numero = parseInt(this.cajaNumeroRef.nativeElement.value);
    let aux = new Array<number>();
    for (var i = 1; i <= 10; i++) {
      var resultado = this.numero * i;
      aux.push(resultado);
    }
    this.numeros = aux;
  }
}
```

#### TABLAMULTPLICAR.COMPONENT.HTML

```
<div>
  <h1>Tabla multiplicar</h1>
  <form #tablaForm="ngForm">
    <label>Número</label>
    <input type="number" name="cajanumero"
      #cajanumero/>
  </form>

```

```

<button (click)="mostrarTabla()">
    Mostrar tabla
</button>
</form>
<table border="1" *ngIf="numeros.Length > 0">
    <thead>
        <tr>
            <th>Operación</th>
            <th>Resultado</th>
        </tr>
    </thead>
    <tbody>
        <tr *ngFor="let num of numeros; let i = index">
            <td>
                {{ numero }} * {{ i + 1 }}
            </td>
            <td>
                {{ num }}
            </td>
        </tr>
    </tbody>
</table>
</div>

```

## RUTAS CON ANGULAR

Por defecto, Angular genera un proyecto con Routing ya creado y con cosas por ahí.  
No lo queremos, necesitamos aprender lo que tenemos tocado para luego automatizarlo

rutasangular

**ng new rutasangular --standalone=false --routing=false**

En angular ya tenemos predefinido el Routing, pero no lo tenemos activado.

Sucede lo mismo que con Forms, debemos activarlo en el módulo **app.module.ts**

Funciona igual que en React y en Vue, necesitamos de un fichero de rutas  
Apuntando a cada component.

Dicho fichero de rutas lo declaramos en el MODULE y simplemente los components  
Lo utilizarán.

La navegación de los Links se realiza mediante etiquetas **<a>**

Debemos utilizar una sintaxis de Angular:

**<a [routerLink]="/mipath">Mi component</a>**

Necesitaremos un fichero de rutas, que en angular se llama **app.routing.ts**

Para las rutas necesitamos los siguientes elementos:

En **angular** no se incluye la barra en la ruta inicial

Un Array con las rutas y sus paths:

```
Routes = [
    { path: "/", component: MiComponent }
]
```

Para dibujar los componentes en las rutas se utiliza la etiqueta:

**<router-outlet></router-outlet>**

Por comodidad, vamos a trabajar desde **src/app/components** por el comando de generación  
De Componentes

Vamos a realizar la aplicación típica de Home, Musica y Cine.

Sobre **public**, creamos una carpeta llamada **assets** y  
creamos una carpeta llamada **images** e incluimos imágenes para  
Los tres componentes y una imagen **404 Not Found**

Sobre **app** creamos una nueva carpeta llamada **components**

Creamos los tres components:

**ng g component components/home**

**ng g component components/musica**

**ng g component components/cine**

Sobre la carpeta **src/app** creamos un nuevo fichero **app.routing.ts**

**Nota:** Las rutas en Angular NO llevan la barra al inicio

## APP.ROUTING.TS

```

import { HomeComponent } from "./components/home/home.component";
import { CineComponent } from "./components/cine/cine.component";
import { MusicaComponent } from "./components/musica/musica.component";
//NECESITAMOS UNA SERIE DE MODULOS QUE SE ENCUENTRAN DENTRO DE ANGULAR
//PARA LA NAVEGACION DE RUTAS
import { Routes, RouterModule } from "@angular/router";
import { ModuleWithProviders } from "@angular/core";
//NECESITAMOS UN ARRAY CON LAS RUTAS, DICHO ARRAY SERA DEL TIPO Routes
const appRoutes: Routes = [
    { path: "", component: HomeComponent },
    { path: "cine", component: CineComponent },
    { path: "musica", component: MusicaComponent }
]
//DESDE ESTA CLASE DEBEMOS EXPORTAR EL ARRAY DE RUTAS COMO PROVEEDOR
export const appRoutingProvider: any[] = [];
//LAS RUTAS EN SI MISMAS
export const routing: ModuleWithProviders<any> =
RouterModule.forRoot(appRoutes);

```

El siguiente paso será habilitar en `app.module.ts` nuestras rutas y el Proveedor:

#### APP MODULE.TS

```
import { routing, appRoutingProvider} from './app.routing';
```

Dentro de `imports` declaramos `routing`

Dentro de `provider` declaramos `appRoutingProvider`

```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    MusicaComponent,
    CineComponent
  ],
  imports: [
    BrowserModule, routing
  ],
  providers: [appRoutingProvider],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Por último, simplemente dibujamos en el componente `app.component.html` la etiqueta `<router-outlet>`

#### APP COMPONENT.HTML

```
<div>
  <h1>Ejemplo rutas</h1>
  <hr/>
  <router-outlet></router-outlet>
</div>
```

El siguiente paso será probar el menu de rutas.

Creamos un nuevo component llamado `menu`

#### MENU COMPONENT.HTML

```
<div>
  <ul id="menu">
    <li>
      <a [routerLink]="/">Home</a>
    </li>
    <li>
      <a [routerLink]="/cine">Cine</a>
    </li>
    <li>
      <a [routerLink]="/musica">Música</a>
    </li>
  </ul>
</div>
```

A diferencia de VUE y React, para el componente `NotFound` se debe indicar con doble Asterisco.

Creamos un nuevo component `notfound` y dibujamos cualquier html

Modificamos las rutas para incluir dicho component cuando tengamos el error 404

**Nota:** La ruta `NotFound` siempre será la última

```

const appRoutes: Routes = [
  { path: "", component: HomeComponent },
  { path: "cine", component: CineComponent},
  { path: "musica", component: MusicaComponent},
  { path: "**", component: NotfoundComponent }
]

```

## RUTAS CON PARAMETROS

Las rutas son iguales que el resto de Frameworks, los parámetros se declaran mediante los Dos puntos.

Dichas rutas con parámetros estarán declaradas dentro de **path** de Routing

```
{ path: "ruta/:parametro", component: MiComponent }
```

Lo que cambia es cómo recuperamos dichos parámetros.

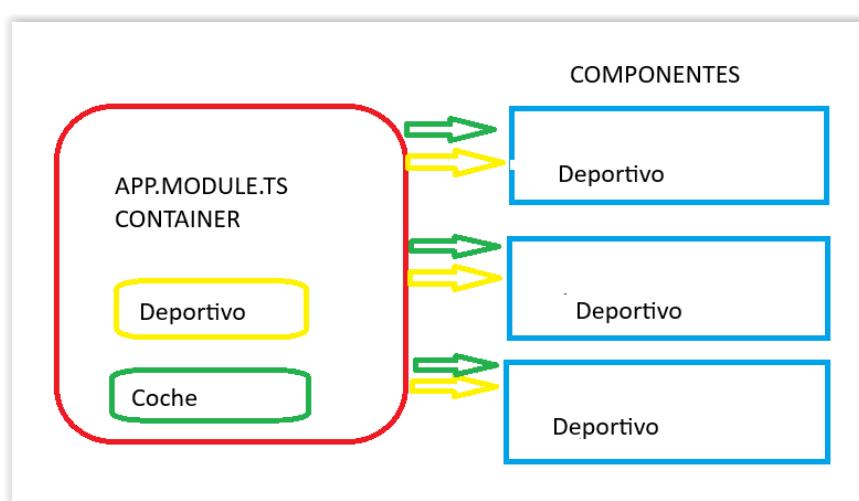
Angular nos proporciona un OBJETO por arte de magia.  
Dicho objeto tendrá los parámetros de las rutas.

Angular tiene un concepto llamado **IoC** y **Dependency Injection** que nos permite Poder recuperar objetos o injectar objetos para que lleguen a las clases en los Constructores por arte de magia.

**IOC:** Inversión de Control. Nos permite injectar nuestras clases desde un Container  
**Dependency Injection:** Nos permite recuperar las clases inyectadas desde un Container

Estos conceptos utilizan el principio SOLID: Principio de Hollywood:  
No nos llame usted, ya le llamamos nosotros

En unos components, recibimos un objeto y otro objeto compatible.  
No los instanciamos, los recibimos.



En nuestro ejemplo, solamente vamos a recuperar objetos inyectados que vienen Creados en el momento en que hemos incluido **Providers** y **Routing**

Nuestros components recibirán objetos Routing para los parámetros inyectados desde El Constructor.

```

export class Component1{
  constructor(Coche){
    ...
  }
}

export class Component1{
  constructor(Coche){
    ...
  }
}

```

El objeto que viene inyectado para las rutas se llama **ActivatedRoute** y nos permite Recuperar los parámetros dentro de Angular.

**Nota:** Por sintaxis, en Angular, los objetos que recibimos en la inyección se declaran Con el guión bajo.

```

constructor(
  _activatedRoute: ActivatedRoute
){
}

```

Posteriormente, una vez que tenemos el objeto inyectado tenemos que suscribirnos Para poder recuperar los parámetros.

Los parámetros serán el nombre que hayamos puesto como :name en path

Dichos parámetros no se pueden recuperar desde el constructor. El constructor Solamente recibe el objeto.

Para recuperar los parámetros utilizamos el método `ngOnInit()`

```
ngOnInit(): void {
  _activatedRoute.params.subscribe((parametros: Params) => {
    //AQUÍ TENEMOS LOS PARAMETROS
  })
}
```

Para comprobar todo esto, vamos a realizar el ejemplo de Número Doble.

[Home](#) | [Cine](#) | [Música](#) | [Número doble 5](#) | [Número doble 77](#) | [Número doble sin params](#)

# Número doble parámetros

## El doble de 5 es 10

Tendremos un componente que recibirá un parámetro llamado :numero

Creamos un nuevo componente llamado `numerodoble`

Vamos a hacer dos rutas, una sin parámetros y otra con parámetros

```
const appRoutes: Routes = [
  { path: "", component: HomeComponent },
  { path: "cine", component: CineComponent },
  { path: "musica", component: MusicaComponent },
  { path: "numerodoble", component: NumeroDobleComponent },
  { path: "numerodoble/:numero", component: NumeroDobleComponent },
  { path: "**", component: NotFoundComponent }
]
```

En el componente `NumeroDoble` tendremos un número dónde recibiremos el dato del Routing.

`NUMERODOBLE.COMPONENT.HTML`

```
<div>
  <h1>Número doble parámetros</h1>
  <div *ngIf="numero">
    <h2 style="color:blue">
      El doble de {{numero}} es {{doble}}
    </h2>
  </div>
  <div *ngIf="!numero">
    <h2 style="color:red">
      No hemos recibido parámetro
    </h2>
  </div>
</div>
```

`NUMERODOBLE.COMPONENT.TS`

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Params } from '@angular/router';
@Component({
  selector: 'app-numero-doble',
  templateUrl: './numero-doble.component.html',
  styleUrls: ['./numero-doble.component.css']
})
export class NumeroDobleComponent implements OnInit {
  //ESTA VARIABLE NADA QUE VER CON EL PARAM
  public numero!: number;
  public doble!: number;
  constructor(private _activeRoute: ActivatedRoute) {}
  ngOnInit(): void {
    //AQUÍ DEBEMOS SUSCRIBIRNOS A LOS PARAMETROS QUE PUEGAN
    //VENIR EN UNA RUTA
    this._activeRoute.params.subscribe((parametros: Params) => {
      //DENTRO DE Params SE RECUPERAN LOS PARAMETROS POR SU NOMBRE
      //CON LA SIGUIENTE SINTAXIS: parametros['PARAMETER NAME']
      //EN ESTE EJEMPLO, EL PARAMETRO SERÁ OPCIONAL, POR LO
      //QUE VAMOS A PREGUNTAR ANTES DE ASIGNAR
      if (parametros['numero'] != null){
        //LOS PARAMETROS SON STRING
        this.numero = parseInt(parametros['numero']);
        this.doble = this.numero * 2;
      }
    })
  }
}
```

En `MenuComponent` creamos rutas con parámetro y sin parámetro

```

<li>
  <a [routerLink]="/numerodoble/77">Número doble 77</a> |
</li>
<li>
  <a [routerLink]="/numerodoble">Número doble sin params</a> |
</li>

```

#### REDIRECCION CON ROUTING

Podemos redirigir a otros componentes mediante código.

Para poder realizar esta acción será necesario recibir otro objeto dentro del Constructor de la clase. Dicho objeto es el objeto Router

```

constructor(
  private _router: Router
) {}

goToHome():void {
  this._router.navigate(["/home"]);
}

```

El método `navigate` contiene una sobrecarga en la que podemos enviar parámetros a una Ruta. Dichos valores estarán establecidos como segundo valor del método

```

goToHome():void {
  this._router.navigate(["/home"], parametros);
}

```

Para probar este nuevo concepto vamos a aplicarlo sobre el componente de Número doble.

Vamos a crear una serie de botones que enviarán un número por código, es decir, enviarán un Parámetro por código mediante Router

## Número doble parámetros

Go to Home Doble 88 Doble 99 Doble 25

**El doble de 99 es 198**

Inyectamos el objeto Router

```

constructor(private _activeRoute: ActivatedRoute
, private _router: Router
) {}

goToHome(): void {
  this._router.navigate(["/"]);
}

```

En el dibujo incluimos un botón para ir a casa.

```

<button (click)="goToHome()">
  Go to Home
</button>

```

El siguiente paso es incluir un método para llamar al mismo componente y enviar un número

```

redirect(num: number): void {
  //NOS VAMOS A LLAMAR A NOSOTROS MISMOS ENVIANDO EN LA RUTA EL
  //PARAMETRO DEL NUMERO RECIBIDO
  this._router.navigate(["/numerodoble", num]);
}

```

Por último, dibujamos los botones con las peticiones.

```
<button (click)="redirect(88)">
  Doble 88
</button>
<button (click)="redirect(99)">
  Doble 99
</button>
<button (click)="redirect(25)">
  Doble 25
</button>
```

Por último, si deseamos hacer dinámicos los Links, es igual a como acabamos de ver  
En el método **navigate**

```
<li *ngFor="let valor of Array">
  <a [routerLink]=[['path', valor]]>
    Component {{valor}}
  </a>
</li>
```

Vamos a realizar el ejemplo de la tabla de multiplicar, pero con rutas.

[Tabla 19](#) | [Tabla 59](#) | [Tabla 3](#) | [Tabla 2](#) | [Tabla 1](#) |

## Dato recibido: 2

Operación	Resultado
2 * 1	2
2 * 2	4
2 * 3	6
2 * 4	8
2 * 5	10
2 * 6	12

Necesito un component **TablaMultiplicarRouting** que recibirá un parámetro  
De un número mostramos la tabla de multiplicar de dicho número.

También necesitamos un **MenuTablaComponent** donde generamos 6  
Números aleatorios y dibujamos sus <a [RouterLink]> Dinámicamente en el menú.

MENUTABLA.COMPONENT.HTML

```
<div>
  <ul id="menu">
    <li *ngFor="let num of numerosRandom">
      <a [routerLink]=[['/tabla', num]]>Tabla {{num}} | </a>
    </li>
  </ul>
</div>
```

MENUTABLA.COMPONENT.TS

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-menu-tabla',
  templateUrl: './menu-tabla.component.html',
  styleUrls: ['./menu-tabla.component.css'
})
export class MenuTablaComponent {
  public numerosRandom: Array<number>;
  constructor() {
    this.numerosRandom = new Array<number>();
    for (var i = 1; i <= 5; i++) {
      let random = Math.floor((Math.random() * 100)) + 1;
      this.numerosRandom.push(random);
    }
  }
}
```

APP.ROUTING.TS

```

    { path: "numero/:numero", component: TablamultiplicarroutingComponent },
    { path: "**", component: NotFoundComponent }

```

#### TABLAMULTPLICARROUTING.COMPONENT.TS

```

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Params } from '@angular/router';
@Component({
  selector: 'app-tablamultiplicarrouting',
  templateUrl: './tablamultiplicarrouting.component.html',
  styleUrls: ['./tablamultiplicarrouting.component.css']
})
export class TablamultiplicarroutingComponent implements OnInit {
  public numero!: number;
  public numeros: Array<number>;
  constructor(private _activatedRoute: ActivatedRoute) {
    this.numeros = new Array<number>();
  }
  generarTabla(): void {
    let aux = new Array<number>();
    for (var i = 1; i <= 10; i++) {
      var resultado = this.numero * i;
      aux.push(resultado);
    }
    console.log(this.numeros);
    this.numeros = aux;
  }
  ngOnInit(): void {
    this._activatedRoute.params.subscribe((params: Params) => {
      this.numero = parseInt(params['numero']);
      this.generarTabla();
    })
  }
}

```

#### TABLAMULTPLICARROUTING.COMPONENT.HTML

```

<div>
  <h1>
    Dato recibido: {{numero}}
  </h1>
  <table border="1">
    <thead>
      <tr>
        <th>Operación</th>
        <th>Resultado</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let num of numeros; let i = index">
        <td>
          {{numero}} * {{i + 1}}
        </td>
        <td>{{num}}</td>
      </tr>
    </tbody>
  </table>
</div>

```

#### MODELOS ANGULAR

Creamos un nuevo proyecto llamado angularcomunicacionmodelos

Un Model representa un objeto que tenemos definido mediante una Serie de propiedades.

Por ejemplo, un Model Departamento.

Por supuesto, podemos utilizar un model en cualquier lugar de nuestra App.

En Angular, al utilizar TypeScript, los objetos están tipados.

La ventaja radica en que si creamos clases Model con sus tipos, tenemos la posibilidad de Recibir y enviar esos modelos entre components y servicios (por ejemplo)

En este ejemplo, podríamos escribir mal y recibir un mensaje 415 diciendo que está mal Formado el objeto del servicio.

```

var departamento = {
  idDepartamento: 10, nombre: "NOMBRE", localidad: "SORIA"
}

```

```

var departamento = {
  idDepartamento: "10", nombre: "NOMBRE", localidad: "SORIA"
}

```

La ventaja que tenemos al utilizar Models está en que solamente tenemos que escribir una Vez la definición. Posteriormente, declaramos variables del tipo de Modelo.

Además, las variables ya tendrán tipos a su vez, que nos controlan lo que hacemos.

Sintaxis:

```

export class Persona {
  public nombre: string;
  public edad: number;

  //PODEMOS UTILIZAR CONSTRUCTORES DENTRO DEL MODEL
  constructor() {
    this.nombre = "Alumno";
  }

  //TAMBIEN PODEMOS TENER SOBRECARGA DE CONSTRUCTORES
  //Y RECIBIR LOS VALORES DIRECTAMENTE
  constructor(name: string, age: number) {
    this.nombre = name;
    this.edad = age;
  }
}

```

}

Podríamos crear, desde otro código un Model utilizando cualquiera de los constructores:

```
public persona: Persona;
this.persona = new Persona();
this.persona.nombre = "Hola";
this.persona = new Persona("Juernes", 25);
```

Vamos a ver un ejemplo simple cargando un JSON de productos y creando un Modelo de Productos.

## Nike Air Jordan, Precio 150



## Nike Air Mag, Precio 1900



Sobre app, creamos una nueva carpeta llamada **models**

Dentro de **models**, una clase llamada **producto.ts**

### PRODUCTO.TS

```
export class Producto {
    //PARA QUE LAS PROPIEDADES SEAN ACCESIBLES, NECESITAMOS
    //QUE SEAN public
    public nombre: string;
    public imagen: string;
    public precio: number;
    //TENDREMOS UN CONSTRUCTOR PARA RECIBIR LOS DATOS
    constructor(name: string, image: string, price: number) {
        this.nombre = name;
        this.imagen = image;
        this.precio = price;
    }
}
```

Sobre app creamos una carpeta **components** y un nuevo component llamado **Listaproductos**.

```
elos>ng g component components/listaProductos
es)
```

### LISTAPRODUCTOS.COMPONENT.TS

```
<div>
    <h1 style="color:blue">
        {{producto.nombre}}, Precio {{producto.precio}}
    </h1>
    <img src={{producto.imagen}}
        style="width: 190px; height: 220px;"/>
    <hr/>
    <div *ngFor="let prod of productos">
        <h2 style="color:red">
            {{prod.nombre}}, Precio {{prod.precio}}
        </h2>
        <img src={{prod.imagen}}
            style="width: 120px; height: 120px;"/>
    </div>
</div>
```

### LISTAPRODUCTOS.COMPONENT.HTML

```
import { Component } from '@angular/core';
import { Producto } from '../../../../../models/producto';
@Component({
    selector: 'app-lista-productos',
    templateUrl: './lista-productos.component.html',
    styleUrls: ['./lista-productos.component.css']
})
export class ListaProductosComponent {
    public producto: Producto;
    public productos: Array<Producto>;
    constructor() {
        this.producto = new Producto("Camiseta", "https://www.webtuti.com/wp-content/uploads/2024/04/camiseta-real-madrid-2024-2025-primeras.webp", 180);
        this.productos = [
            new Producto(
                "Nike Air Jordan",
                "https://images.zapatojios.com/media/2022/07/39253d37.jpg",
                150
            )
        ];
    }
}
```

```

    150
),
new Producto(
  "Nike Air Mag",
  "https://limitedresell.com/img/anblog/b/b-654d14cf06f5-anblog_thumb.jpg",
  1900
),
new Producto(
  "New Balance 998",
  "https://www.sneakers-actus.fr/wp-content/uploads/2023/06/NB-998-U998TE-x-
  Teddy-Santis-Plum-Purple-MIUSA.jpg",
  140
),
new Producto(
  "https://jhayber.com/documents/images/products/63638-850-1.jpg",
  "J-Hayber Olimpo",
  60
),
new Producto(
  "Triple S Balenciaga",
  "https://cdn1.jolicloset.com/img/full/2024/05/1321192-1/plastico-
  zapatillas-balenciaga-triple-s-de-poliuretano-blanco-amarillo.jpg",
  650
)
]
}
}

```

Una vez que lo tenemos, existe otra sintaxis para crear los modelos. Dicha sintaxis nos permite a la vez declarar variables y recibirlas en el constructor.

**Nota:** Si utilizamos esta sintaxis, solamente es para recibir las propiedades en el Constructor. Fuera del constructor no tendríamos variables.

```

export class Producto {
  constructor(
    public nombre: string,
    public imagen: string,
    public precio: number,
  ) {}
}

```

Os he comentado que hagamos el proyecto con Routing ya instalado. Me gustaría incluir routing en este proyecto con las cosas ya montadas.

Necesito un Link en un menú y mostrar los productos al pulsar dicho Link Mediante <router-outlet>

## COMUNICACIÓN COMPONENTES ANGULAR

Por supuesto, podemos realizar comunicación entre componentes en Angular. La comunicación es bidireccional.

Para enviar información desde el Padre al Hijo se realiza mediante **props**, pero Tiene una sintaxis extraña.

Las propiedades a enviar se realizan de la siguiente forma:

```
<padre-component [propiedad]="valor"></padre-component>
```

En el componente Hijo tenemos una librería llamada **@Input** para poder recuperar los **props**

**@Input** nos permite enlazar las propiedades recibidas en un Hijo desde el padre

```

export class Hijo {
  @Input propiedad: tipoDatos;
}

```

Tendremos un componente llamado **HijoCoche** y otro componente llamado **PadreCoches**

## Padre Coches

# Hijo Coche Component

Ford Mustang

Coche encendido!!!

Velocidad: 0 km/h

# Hijo Coche Component

Comenzamos creando un nuevo Modelo llamado `coche.ts`

COCHE.TS

```
export class Coche {
  constructor(
    public marca: string,
    public modelo: string,
    public velocidad: number,
    public aceleracion: number,
    public estado: boolean
  )
}
```

Creamos un nuevo component llamado `HijoCoche`

Dentro de CSS incluimos dos clases para encendido y apagado (rojo y verde)

HIJOCOCHE.COMPONENT.CSS

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

.verde {
  background-color: lightgreen;
}
```

HIJOCOCHE.COMPONENT.TS

```
import { Component } from '@angular/core';
import { Coche } from '../../../../../models/coche';
@Component({
  selector: 'app-hijo-coche',
  templateUrl: './hijo-coche.component.html',
  styleUrls: ['./hijo-coche.component.css'
})
export class HijoCocheComponent {
  public car: Coche;
  public mensaje: string;
  constructor() {
    this.car = new Coche("Pontiac", "Firebird", 220, 20, false);
  }
  comprobarEstado(): boolean {
    if (this.car.estado == false) {
      this.mensaje = "El coche está apagado";
      this.car.velocidad = 0;
      return false;
    } else {
      this.mensaje = "Coche encendido!!!";
      return true;
    }
  }
  encenderCoche() : void {
    this.car.estado = !this.car.estado;
    this.comprobarEstado();
  }
}
```

```

acelerarCoche(): void {
  if (this.comprobarEstado() == false){
    alert("Donde vas??? que está apagado!!!");
  }else{
    this.car.velocidad = this.car.velocidad + this.car.aceleracion;
  }
}

```

#### HIJOCOCHE.COMPONENT.HTML

```

<div>
  <h1>Hijo Coche Component</h1>
  <h2 style="color:blue">
    {{car.marca}} {{car.modelo}}
  </h2>
  <h2 [ngClass]="{
    verde: car.estado == true,
    rojo: car.estado == false
  }">
    {{mensaje}}
  </h2>
  <h2>
    Velocidad: {{car.velocidad}} km/h
  </h2>
  <button (click)="encenderCoche()">
    Start/Stop coche
  </button>
  <button (click)="acelerarCoche()">
    Acelerar!!!
  </button>
</div>

```

El siguiente paso será enviar múltiples coches desde un Parent

Tenemos dos formas de plantearlo:

- 1) Enviar cada Propiedad desde el Padre

```
<div *ngFor="let car of coches">
  <hijo-coche [marca]="car.marca" [modelo]="car.modelo"/>
</div>
```

- 2) Enviar el propio Model con todas sus propiedades

```
<div *ngFor="let car of coches">
  <hijo-coche [coche]="car"/>
</div>
```

Creamos un nuevo component llamado **PadreCoches**

#### PADRECOCHES.COMPONENT.TS

```

import { Component } from '@angular/core';
import { Coche } from '../../../../../models/coche';
@Component({
  selector: 'app-padre-coches',
  templateUrl: './padre-coches.component.html',
  styleUrls: ['./padre-coches.component.css']
})
export class PadreCochesComponent {
  public coches: Array<Coche>;
  constructor() {
    this.coches = [
      new Coche("Ford", "Mustang", 0, 25, false),
      new Coche("Volkswagen", "Escarabajo", 0, 10, false),
      new Coche("Lamborghini", "Diablo", 0, 35, false)
    ]
  }
}

```

#### PADRECOCHES.COMPONENT.HTML

```

<div>
  <h2>Padre Coches</h2>
  <div *ngFor="let coche of coches">
    <app-hijo-coche [car]="coche"></app-hijo-coche>
    <br/>
  </div>
</div>

```

Dentro de Hijo Coche debemos recuperar un **props** llamado **car** y que se debe Declarar mediante **@Input**

#### HIJOCOCHE.COMPONENT.TS

```

> components > hijo-coche > hijo-coche.component.ts > HijoCocheComponent > constructor
import { Component, Input } from '@angular/core';
import { Coche } from '../../models/coche';

@Component({
  selector: 'app-hijo-coche',
  templateUrl: './hijo-coche.component.html',
  styleUrls: ['./hijo-coche.component.css']
})
export class HijoCocheComponent {
  @Input() car!: Coche;
  public mensaje: string;
  constructor() {
    this.mensaje = "";
  }
}

```

#### ENVIAR INFORMACION DE HIJO A PARENT

Para enviar información desde el Hijo al Parent debemos hacerlo mediante métodos.  
(Seleccionar favorito)

Para enviar información debemos utilizar `@Output` desde el component Hijo

La variable `@Output` es de tipo `Emitter`, lo que significa que es un método que Tendrá el Padre declarado.

#### COMPONENT HIJO

```
@Output metodoPadre = new Emitter();
```

El hijo tendrá un método propio:

```
metodoHijo(event): void {
  this.metodoPadre.emit();
}
```

Código HTML del Hijo

```
<button (click)="metodoHijo($event)">Llamar la Parent</button>
```

#### COMPONENT PADRE

El método Padre **debe** recibir un parámetro por si queremos enviar información.

```
metodoPadre(event): void {
}
```

Código HTML del Parent:

```
<app-hijo [propiedad]="valor"
  (metodoPadre)="metodoPadre($event)"/>
```

Vamos a realizar un ejemplo sencillo con los Deportes.

Tendremos un componente llamado `PadreDeportes` y otro componente llamado `HijoDeporte`

## Padre deportes

### Deporte favorito: Petanca

- Petanca
 

Seleccionar favorito
- Curling
 

Seleccionar favorito
- Futbol
 

Seleccionar favorito

Lo que haremos será seleccionar un Deporte favorito desde el hijo enviando la Información al Padre

#### PADREDEPORTES.COMPONENT.TS

```

import { Component } from '@angular/core';
@Component({
  selector: 'app-padre-deportes',
  templateUrl: './padre-deportes.component.html',
  styleUrls: ['./padre-deportes.component.css'
})
export class PadreDeportesComponent {
  public deportes: Array<string>;
  public mensaje!: string;
  //TENDREMOS UN METODO PARA PODER SELECCIONAR UN FAVORITO
  //Y DIBUJARLO
}

```

```

seleccionarFavoritoPadre(event: any): void {
  this.mensaje = "Deporte favorito: " + event;
  console.log("Dato: " + event);
}
constructor() {
  this.deportes = ["Petanca", "Curling", "Futbol", "Dados"]
}

PADREDEPORTES.COMPONENT.HTML

<div>
  <h1>Padre deportes</h1>
  <h1 style="color:red">{{mensaje}}</h1>
  <ul>
    <li *ngFor="let deporte of deportes"
      <app-hijo-deporte [sport]="deporte"
      (seleccionarFavoritoPadre)="seleccionarFavoritoPadre( $event )">
      </app-hijo-deporte>
    </li>
  </ul>
</div>

```

HIJODEPORTE.COMPONENT.TS

```

import { Component, Input, Output, EventEmitter } from '@angular/core';
@Component({
  selector: 'app-hijo-deporte',
  templateUrl: './hijo-deporte.component.html',
  styleUrls: ['./hijo-deporte.component.css']
})
export class HijoDeporteComponent {
  @Input() sport: string;
  @Output() seleccionarFavoritoPadre: EventEmitter<any> =
  new EventEmitter();
  seleccionarFavoritoHijo(): void {
    //REALIZAMOS LA LLAMADA AL METODO PARENT
    this.seleccionarFavoritoPadre.emit( this.sport );
  }
}

```

HIJODEPORTE.COMPONENT.HTML

```

<div>
  <p style="color:blue">{{sport}}</p>
  <button (click)="seleccionarFavoritoHijo()">
    Seleccionar favorito
  </button>
</div>

```

Vamos a realizar una mini práctica para reforzar este nuevo concepto.

Vamos a realizar la práctica de Comics. Tenemos comicsAngular.txt

Tendremos un Model llamado Comic

Tendremos un componente parent llamado Librería

Un componente Hijo llamado Comic

Acciones:

- Dibujar los comics mediante Padre Hijo
- Seleccionar un comic favorito
- Nuevo comic con un Formulario

## Librería

Nombre Peter  
 Imagen <https://m.media-amazon.co>  
 Descripción Porker  
 Nuevo comic



Spiderman	Wolverine	Guardianes de la Galaxia	Avengers
<a href="#">Hombre araña</a>	<a href="#">Lobezno</a>	<a href="#">Yo soy Groot</a>	<a href="#">Los Vengadores</a>
			
<a href="#">Seleccionar favorito</a>	<a href="#">Seleccionar favorito</a>	<a href="#">Seleccionar favorito</a>	<a href="#">Seleccionar favorito</a>

Creamos un nuevo proyecto, llamado serviciosangular

```

export class Comic {
  constructor(
    public nombre: string,
    public imagen: string,
    public descripcion: string
  ){}
}

```

Habilitamos formularios en el proyecto

APP MODULE.TS

```
import { FormsModule } from '@angular/forms';
```

```

@NgModule({
  declarations: [
    AppComponent,
    ComicComponent,
    LibreriaComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
})

```

COMIC COMPONENT.HTML

```

<div style="float:left">
  <h1>{{comic.nombre}}</h1>
  <h2 style="color:blue"> {{comic.descripcion}}</h2>
  
  <button (click)="marcarFavorito()" style="margin-top: 10px;">
    Seleccionar favorito
  </button>
</div>

```

COMIC COMPONENT.TS

```

import { Component, EventEmitter, Input, Output } from '@angular/core';
import { Comic } from '../../../../../models/comic';
@Component({
  selector: 'app-comic',
  templateUrl: './comic.component.html',
  styleUrls: ['./comic.component.css']
})
export class ComicComponent {
  @Input() comic!: Comic;
  @Output() seleccionarFavorito: EventEmitter<any> = new EventEmitter();
  marcarFavorito() {
    this.seleccionarFavorito.emit(this.comic);
  }
}

```

LIBRERIA COMPONENT.TS

```

import { Component, ElementRef, ViewChild } from '@angular/core';
import { Comic } from '../../../../../models/comic';
@Component({
  selector: 'app-libreria',
  templateUrl: './libreria.component.html',
  styleUrls: ['./libreria.component.css']
})
export class LibreriaComponent {
  @ViewChild("cajanombre") cajaNombre!: ElementRef;
  @ViewChild("cajaimagen") cajaImagen!: ElementRef;
  @ViewChild("cajadcripcion") cajaDescripcion!: ElementRef;
  public comics: Array;
  public comicFavorito!: Comic;
  seleccionarFavorito(event: Comic): void {
    this.comicFavorito = event;
  }
  nuevoComic(): void {
    let nombre = this.cajaNombre.nativeElement.value;
    let imagen = this.cajaImagen.nativeElement.value;
    let descripcion = this.cajaDescripcion.nativeElement.value;
    let comicNew = new Comic(nombre, imagen, descripcion);
    this.comics.push(comicNew);
  }
  constructor() {
    this.comics = [
      new Comic(
        "Spiderman",
        "https://images-na.ssl-images-amazon.com/images/I/61AYfL5069L.jpg",
        "Hombre arana"
      ),
      new Comic(
        "Wolverine",
        "https://i.etsystatic.com/9340224/r/il/42f0e1/1667448004/il_570xN.1667448004_sqv0.jpg",
        "Lobezno"
      ),
      new Comic(
        "Guardianes de la Galaxia",
        "https://cdn.normacomics.com/media/catalog/product/cache/1/thumbnail/9df78"
      )
    ];
  }
}

```

```

    "Yo soy Groot"
),
new Comic(
"Avengers",
"https://d261pennugtm8s.cloudfront.net/stores/057/977/products/ma\_avengers\_01\_01-891178138c020318f315132687055371-640-0.jpg",
"Los Vengadores"
),
new Comic(
"Spawn",
"https://i.pinimg.com/originals/e1/d8/ff/e1d8ff4aeab5e567798635008fe98ee1.png",
"Todd MacFarlane"
)
];
}
}

```

#### LIBRERIA.COMPONENT.HTML

```

<div>
  <h1>Librería</h1>
  <form #comicForm="ngForm">
    <label>Nombre:</label>
    <input type="text" #cajanombre name="cajanombre"/><br/>
    <label>Imagen:</label>
    <input type="text" #cajaImagen name="cajaImagen"/><br/>
    <label>Descripción:</label>
    <input type="text" #cajadescpcion name="cajadescpcion"/><br/>
    <button (click)="nuevoComic()">
      Nuevo comic
    </button>
  </form>
  <div style="background-color: yellow;" *ngIf="comicFavorito">
    <img src={{comicFavorito.imagen}}
        style="width: 100px; height: 120px;"/>
  </div>
  <div *ngFor="let comic of comics">
    <app-comic [comic]="comic"
      (seleccionarFavorito)="seleccionarFavorito($event)"></app-comic>
  </div>
</div>

```

#### SERVICIOS ANGULAR

Un servicio es una clase que contiene múltiples métodos, tanto de acción o de recuperar Datos.

Se utilizan de forma distinta que en VUE y debemos declararlos en el Component

Podemos inyectar el servicio desde el módulo.

La declaración de los servicios debemos realizarla dentro de **providers** en nuestro Module

Las clases de servicio, en Angular se denominan: **service.descripcion.ts**

Vamos a trabajar desde la carpeta **src/app**

No podemos utilizar como **providers** cualquier clase. Será necesario utilizar Una denominación de la clase llamada **@Injectable**

Se utiliza como decoración de clase:

```

@Injectable()
export class MiServicio {
}

```

Una vez que podemos inyectar la clase, debemos ponerla en **providers**, ya sea a nivel General (Module) o a nivel Component.

Lo único que vamos a realizar como concepto será llevarnos los comics a un Servicio y crear un método **getComics()** e inyectarlo, tanto en Component como en Module.

Dentro de **src/app** creamos una carpeta llamada **services** y dentro una clase llamada **service.comics.ts**

#### SERVICE.COMICS.TS

```

import { Injectable } from "@angular/core";
import { Comic } from "../models/comic";
@Injectable()
export class ServiceComics {
  getComics(): any {
    let comics = new Array<Comic>();
    comics = [
      new Comic(
        "Spiderman",
        "https://images-na.ssl-images-amazon.com/images/I/61AYFl5069L.jpg",
        "Hombre araña"
      ),
      new Comic(
        "Wolverine",
        "https://i.etsystatic.com/9340224/r/i1/42f0e1/1667448004/i1\_570xN.1667448004\_sqv0.jpg",
        "Lobezno"
      ),
      new Comic(
        "Guardianes de la Galaxia",
        "https://cdn.normacomics.com/media/catalog/product/cache/1/thumbnail/9df78eb33525d08d6e5fb8d27136e95/g/u/guardianes\_galaxia\_guadianes\_infinito.jpg",
        "Yo soy Groot"
      ),
      new Comic(
        "Avengers",
        "https://d261pennugtm8s.cloudfront.net/stores/057/977/products/ma\_aven\_01\_01-891178138c020318f315132687055371-640-0.jpg",
        "Los Vengadores"
      ),
      new Comic(
        "Spawn",
        "https://i.pinimg.com/originals/e1/d8/ff/e1d8ff4aeab5e567798635008fe98"
      )
    ];
  }
}

```

```

        ee1.png",
        "Todd MacFarlane"
    );
}
return comics;
}
}

```

Para poder recuperar los comics, necesitamos **providers**

Vamos a probarlo dentro de un propio Component: Librería

**Nota:** En los constructores podemos recuperar elementos inyectados (servicios o rutas)  
 Sirven para inicializar objetos de la clase/component.  
 Pero los objetos inyectados NO SE UTILIZAN DESDE EL CONSTRUCTOR.  
 Se realizan las acciones siempre desde **ngOnInit**

#### LIBRERIA.COMPONENT.TS

```

import { Component, ElementRef, OnInit, ViewChild } from '@angular/core';
import { Comic } from '../models/comic';
import { ServiceComics } from '../services/service.comics';
@Component({
  selector: 'app-libreria',
  templateUrl: './libreria.component.html',
  styleUrls: ['./libreria.component.css'],
  //DEBEMOS INYECTAR UN SERVICIO PARA PODER RECUPERARLO
  //POSTERIORMENTE EN EL CONSTRUCTOR
  providers: [ServiceComics]
})
export class LibreriaComponent implements OnInit {
  @ViewChild("cajanombre") cajaNombre!: ElementRef;
  @ViewChild("cajaImagen") cajaImagen!: ElementRef;
  @ViewChild("cajadescpcion") cajaDescripcion!: ElementRef;
  public comics!: Array<Comic>;
  public comicFavorito!: Comic;
  constructor(
    private _service: ServiceComics
  ) {}
  ngOnInit(): void {
    this.comics = this._service.getComics();
  }
  seleccionarFavorito(event: Comic): void {
    this.comicFavorito = event;
  }
  nuevoComic(): void {
    let nombre = this.cajaNombre.nativeElement.value;
    let imagen = this.cajaImagen.nativeElement.value;
    let descripcion = this.cajaDescripcion.nativeElement.value;
    let comicNew = new Comic(nombre, imagen, descripcion);
    this.comics.push(comicNew);
  }
}

```

Tal y como lo hemos planteado, tendríamos que estar declarando en cada Component el provider/s a utilizar.

Vamos a centralizar los elementos a inyectar y posteriormente los utilizaremos  
 En cada constructor.  
 Inyectaremos dentro de **Module**

#### APP.MODULE.TS

```

import { ServiceComics } from './services/service.comics';

@NgModule({
  declarations: [
    AppComponent,
    ComicComponent,
    LibreriaComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [ServiceComics],
  bootstrap: [AppComponent]
})

```

#### BOOTSTRAP ANGULAR

Al igual que hemos realizado con otros Frameworks, tenemos Bootstrap propio de Angular o podemos elegir bootstrap estándar.

Existe otro framework de diseño propio para angular que es bastante popular llamado **Material Design Angular**

## Incluir via CDN

When you only need to include Bootstrap's compiled CSS or JS, you can use [jsDelivr](#). See it in action with our simple [quick start](#), or [browse the examples](#) to jumpstart your next project. You can also choose to include Popper and our JS [separately](#).



### INDEX.HTML

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Serviciosangular</title>
<base href="/">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" type="image/x-icon" href="favicon.ico">
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKzyjpEjISvSwarU90FeRpok6YctnYmDrSpNlyT2bRjXh0JNhjY6HW+ALEWIH" crossorigin="anonymous">
</head>
<body>
<app-root></app-root>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-YpcryfotV3LHB60NNkxc59fDVZLESaAA55NDzOxhy9GkcIdsLK1eN7N6j1ehz" crossorigin="anonymous"></script>
</body>
</html>
```

Apuntamos incluir más librerías (Gracias Gabriel)

### SERVICIOS API ANGULAR

Por supuesto, podemos utilizar **axios**, ningún problema, tendríamos que instalarlo.

En este Framework, tenemos peticiones Api integradas directamente.

Para poder realizar peticiones Api, necesitamos incluirlas dentro del Module.

La librería a implementar es **HttpClientModule** incluida dentro de `@angular/common/http`

Dentro de **imports** incluiremos la librería:

```
imports: [
  BrowserModule,
  AppRoutingModule,
  FormsModule,
  HttpClientModule
],
```

Cualquier petición api que realicemos debemos integrarla dentro del concepto de **services**

Cuando realizamos una petición a un servicio Api tenemos que utilizar **promesas**

En lugar de **any** para la devolución de datos, debemos poner **Observable** para Que se haga un seguimiento de los cambios

Comenzamos leyendo un servicio super simple.

<https://servicioapipersonasmvcpgs.azurewebsites.net/api/personas>

Este es el Modelo que vamos a crear

```
{
  "IdPersona": 1,
  "Nombre": "Lucia",
  "Email": "lucia@gmail.com",
  "Edad": 19
},
```

Sobre **models**, creamos una nueva clase llamada **persona.ts**

### PERSONA.TS

```

export class Persona {
    constructor(
        public IdPersona: number,
        public Nombre: string,
        public Email: string,
        public Edad: number
    ){}
}

```

Incluimos la librería HttpClientModule dentro de Module e imports

APP MODULE.TS

```

], [
  providers: [ServiceComics, ServicePersonas, provideHttpClient()],
]

```

Sobre services creamos un nuevo servicio llamado service.personas.ts

SERVICE.PERSONAS.TS

```

import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
@Injectable()
export class ServicePersonas {
    //PARA LAS PETICIONES API VIENE UN OBJETO LLAMADO
    //HttpClient QUE NOS PERMITIRA REALIZAR LAS PETICIONES
    constructor(private _http: HttpClient) {}
    getPersonas(): Observable<any> {
        let urlApiPersonas =
            "https://servicioapipersonasmvcpgs.azurewebsites.net/api/personas";
        //TENEMOS DOS FORMAS DE REALIZAR LA FUNCIONALIDAD DE DEVOLUCION DE
        //DATOS
        //1) IGUAL QUE EN VUE, CREANDO UNA PROMESA POR ENCIMA DE ESTE METODO
        //2) DEVOLVER DIRECTAMENTE LA PETICION PARA QUE SEA EL COMPONENTE
        //QUIEN SE SUSCRIBA
        return this._http.get(urlApiPersonas);
    }
}

```

El siguiente paso es dar de alta a nuestro service dentro de Providers en el Module

APP MODULE.TS

```

], [
  providers: [ServiceComics, ServicePersonas, provideHttpClient()],
]

```

Por último, vamos a crear un component llamado personasapi

PERSONASAPI.COMPONENT.TS

```

import { Component, OnInit } from '@angular/core';
import { ServicePersonas } from '../../../../../services/service.personas';
import { Persona } from '../../../../../models/persona';
@Component({
    selector: 'app-personasapi',
    templateUrl: './personasapi.component.html',
    styleUrls: ['./personasapi.component.css']
})
export class PersonasapiComponent implements OnInit {
    public personas!: Array<Persona>;
    constructor(private _service: ServicePersonas) {}
    ngOnInit(): void {
        this._service.getPersonas().subscribe(response => {
            console.log("leyendo");
            this.personas = response;
        })
    }
}

```

PERSONASAPI.COMPONENT.HTML

```

<div>
    <h1>Personas</h1>
    <ul class="list-group" *ngIf="personas.length > 0">
        <li *ngFor="let persona of personas" class="list-group-item">
            {{persona.Nombre}}
        </li>
    </ul>
</div>

```

Vamos a visualizar cómo podemos llamar al Servicio realizando una Promesa.

SERVICE.PERSONAS.TS

```

getPersonasPromesa(): Promise<any> {
    let urlApiPersonas =
        "https://servicioapipersonasmvcpgs.azurewebsites.net/api/personas";
    let promise = new Promise((resolve) => {
        this._http.get(urlApiPersonas).subscribe((response) => {
            resolve(response);
        })
    })
    return promise;
}

```

Realizamos la petición en el component PersonasApi

```

ngOnInit(): void {
  this._service.getPersonasPromesa().then(response => {
    this.personas = response;
})

```

Quiero visualizar que es un component Standalone.

Simplemente es un componente aislado, la idea radica en poder migrar componentes Entre diferentes proyectos y que el propio componente sea autónomo.

En realidad, significa declarar todo dentro del componente.

Vamos a crear un nuevo componente en este proyecto y lo haremos **standalone**

Copiamos el código de personas API y ver qué es lo que tenemos que tocar para Llevarlo a ser funcional.

Creamos un nuevo componente llamado **personasstandalone**

Vamos a indicar que es aislado en la declaración del propio Component

```

@Component({
  selector: 'app-personasstandalone',
  templateUrl: './personasstandalone.component.html',
  styleUrls: ['./personasstandalone.component.css'],
  standalone: true
})

```

Una vez que es aislado, el componente, en teoría, ya podemos utilizarlo individualmente

**APP.COMPONENT.HTML**

```

<div>
  <app-personasstandalone></app-personasstandalone>
  <app-personasapi></app-personasapi>
</div>

```

En nuestro módulo, nos estará diciendo que ya no es un componente, debemos ponerlo como imports

```

imports: [
  BrowserModule,
  AppRoutingModule,
  FormsModule,
  PersonasstandaloneComponent
],

```

Una vez que hemos visto que lo dibuja, vamos a intentar que llame al servicio de personas.

Copiamos el código de PersonaAPI a este componente

Una vez que tenemos el código, está diciendo que no entiendo ni los IF, ni los FOR

```

src/app/components/personasstandalone/personasstandalone.component.html:4:13:
  4 |       <li *ngFor="let persona of personas" class="list-group-item">
     |~~~~~
Error occurs in the template of component PersonasstandaloneComponent.

src/app/components/personasstandalone/personasstandalone.component.ts:7:15:
  7 |   templateUrl: './personasstandalone.component.html',
     |~~~~~
  personasstandalone.component.spec.ts:10:13:
  personasstandalone.component.ts:10:13:

```

Eso sucede porque este componente es libre y no tiene las librerías del resto de componentes  
Debemos indicar lo que necesitamos en su interior

```

@Component({
  selector: 'app-personasstandalone',
  templateUrl: './personasstandalone.component.html',
  styleUrls: ['./personasstandalone.component.css'],
  standalone: true,
  imports: [NgFor, NgIf]
})
export class PersonasstandaloneComponent implements OnInit

```

```

components > personasstandalone > personasstandalone.component.ts > PersonasstandaloneComponent
@Component({
  selector: 'app-personasstandalone',
  templateUrl: './personasstandalone.component.html',
  styleUrls: ['./personasstandalone.component.css'],
  standalone: true,
  imports: [NgFor, NgIf],
  providers: [ServicePersonas]
})
export class PersonasstandaloneComponent implements OnInit {

```

Hemos visto como realizar los imports y los providers, pero no necesitamos NgIf y NgFor.

Vamos a utilizar otra sintaxis parecida.

Por ejemplo, para un For utilizamos esta sintaxis (Sin let)

```
@for (objeto of objetos; track objeto){
  CODIGO HTML
}
```

PERSONASSTANDALONE.COMPONENT.HTML

```

<div>
  <h1>Personas Standalone</h1>
  <ul class="list-group">
    @for (persona of personas; track persona){
      <li class="list-group-item">
        {{persona.Nombre}}
      </li>
    }
  </ul>
</div>

```

Vamos a crear un nuevo proyecto completamente Standalone, es decir, sin MODULE.

Y nos llevamos la clase Service, el model de personas y el component standalone a ver  
Qué es lo que tenemos que tocar ahí.

Creamos un nuevo proyecto llamado angularstandalone

```
ng new angularstandalone
```

Continuamos en el proyecto de **Servicios Angular**

El siguiente concepto a utilizar está en la clase **Global**

Una variable Global nos permite poder utilizar variables dentro de toda nuestra App.

Tenemos dos opciones dentro de Angular:

- 1) Crear una clase **Global** con sus objetos URLs y utilizarla como hemos realizado  
En el resto de Frameworks

Sobre **src/app** creamos una nueva clase llamada **global.ts**

```

export var Global = {
  urlApiPersonas: "https://servicioapipersonasmvcpgs.azurewebsites.net/"
}

```

Utilizamos dentro del **Service** nuestra clase Global

**SERVICE.PERSONAS.TS**

```

import { Observable } from 'rxjs';
import { Global } from '../global';

@Injectable()
export class ServicePersonas {

  getPersonasPromesa(): Promise<any> {
    let request = "api/personas";
    let url = Global.urlApiPersonas + request;
    let promise = new Promise((resolve) => {
      this._http.get(url).subscribe((response) => {
        resolve(response);
      })
    })
    return promise;
  }
}

```

- 2) Utilizar variables de entorno de Angular. Dichas variables se llaman **environments**  
 Es lo mismo que utilizar Global, es decir, declaramos objetos y podemos utilizarlos  
 En cualquier clase/componente  
 La ventaja es otra, con variables de entorno, podemos tener doble declaración de  
 Variables para trabajar en desarrollo o en producción

Debemos generar los ficheros de entorno

Para ello, ejecutamos el siguiente comando.

ng g environments

```

C:\Users\Profesor MCSD Mañana\Documents\FULLSTACK\angular\serviciosangular>ng g environments
CREATE src/environments/environment.ts (31 bytes)
CREATE src/environments/environment.development.ts (31 bytes)
UPDATE angular.json (3220 bytes)

```

Como podemos comprobar, nos ha creado dos ficheros:

- 1) Entorno de producción: TS
- 2) Entorno de desarrollo: DEVELOPMENT

Cuando estamos trabajando en desarrollo con nuestro Localhost, se utiliza el fichero

Development.

Cuando subimos a un servidor real, se utiliza el fichero TS.

En los dos ficheros debemos tener lo mismo.



```

environment.development.ts
1 export const environment = {
2   urlApiPersonas: "https://www.servidorbroma.com"
3 };
4

environment.ts
1 export const environment = {
2   urlApiPersonas: "https://www.servidorREAL.com"
3 };
4

```

Vamos a incluir nuestro servidor de Personas.

Dentro del Service utilizamos **environment** directamente

```

import { environment } from '../../environments/environment';

```

```

getPersonasPromesa(): Promise<any> {
  let request = "api/personas";
  let url = environment.urlApiPersonas + request;
  let promise = new Promise((resolve) => {
    this._http.get(url).subscribe((response) => {
      resolve(response);
    })
  })
}

```

Dentro del fichero `angular.json` tenemos la información de uso de los ficheros de entorno

```

"development": {
  "optimization": false,
  "extractLicenses": false,
  "sourceMap": true,
  "fileReplacements": [
    {
      "replace": "src/environments/environment.ts",
      "with": "src/environments/environment.development.ts"
    }
  ]
}

```

Vamos a incluir un menú y empezamos a jugar con múltiples componentes.

Creamos un nuevo componente llamado `menu.component` e incluimos un menú de Bootstrap

Vamos a realizar una petición al servicio de Coches

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

Incluimos nuestra URL sobre `environment`

```

export const environment = {
  urlApiPersonas: "https://servicioapipersonasmvcpgs.azurewebsites.net/",
  urlApiCoches: "https://apicochespaco.azurewebsites.net/"
};

```

Sobre `models` creamos una nueva clase llamada `coche.ts`

`COCHE.TS`

```

export class Coche {
  constructor(
    public idcoche: number,
    public marca: string,
    public modelo: string,
    public conductor: string,
    public imagen: string
  ){}
}

```

Sobre services creamos un nuevo servicio llamado service.coches.ts

SERVICE.COCHES.TS

```

import { Coche } from "../models/coche";
import { environment } from "../../environments/environment";
import { HttpClient } from "@angular/common/http";
import { Injectable } from "@angular/core";
import { Observable } from "rxjs";
@Injectable()
export class ServiceCoches {
  constructor(private _http: HttpClient){}
  getCoches(): Observable<any> {
    let request = "webresources/coches";
    let url = environment.urlApiCoches + request;
    return this._http.get(url);
  }
}

```

Declaramos el servicio dentro de los providers del Module.

APP MODULE.TS

```

  ],
  providers: [ServiceComics, ServicePersonas
    , provideHttpClient(), ServiceCoches],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Creamos un nuevo component llamado coches.component.ts

COCHES.COMPONENT.TS

```

import { Component, OnInit } from '@angular/core';
import { ServiceCoches } from '../../services/service.coches';
import { Coche } from '../../../../../models/coche';
@Component({
  selector: 'app-coches',
  templateUrl: './coches.component.html',
  styleUrls: ['./coches.component.css']
})
export class CochesComponent implements OnInit{
  public coches!: Array<Coche>;
  constructor(private _service: ServiceCoches){}
  ngOnInit(): void {
    this._service.getCoches().subscribe(response => {
      this.coches = response;
    })
  }
}

```

COCHES.COMPONENT.HTML

```

<h1>Api coches</h1>
@for (car of coches; track car){
  <div class="card" style="width: 18rem;">
    <img class="card-img-top" src={{car.imagen}} alt="Card image cap">
    <div class="card-body">
      <h5 class="card-title">{{car.marca}} {{car.modelo}}</h5>
      <p class="card-text">Conductor: {{car.conductor}}</p>
    </div>
  </div>
}

```

El siguiente ejemplo será con dos versiones.

La primera versión, vosotros.

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

Necesitamos un component llamado plantillafuncionsimple

# Plantilla funciones simple

Enfermero

Mostrar plantilla

Apellido	Función	Turno	Salario
Hernández J.	Enfermero	T	387234
Bocina G.	Enfermero	M	530435

Necesitamos un **model** llamado **plantilla** y un service llamado **service.plantillas.ts**

- Al iniciar el component, veremos en un desplegable <select> las funciones
- Al seleccionar una función (click) veremos la plantilla que tiene dicha función

GET /api/Plantilla/Funciones

GET /api/Plantilla/PlantillaFuncion/{funcion}

ENVIRONMENT

```
export const environment = {
  urlApiPersonas: "https://servicioapipersonasmvcpgs.azurewebsites.net/",
  urlApiCoches: "https://apicochespaco.azurewebsites.net/",
  urlApiPlantilla: "https://apiplantillacore.azurewebsites.net/"
};
```

PLANTILLA.TS

```
export class Plantilla {
  constructor(
    public idEmpleado: number,
    public idHospital: number,
    public idSala: number,
    public apellido: string,
    public funcion: string,
    public turno: string,
    public salario: number
  ){}
}
```

SERVICE.PLANTILLAS.TS

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { environment } from '../environments/environment';
@Injectable()
export class ServicePlantilla {
  constructor(private _http: HttpClient){}
  getFunciones(): Observable<any> {
    let request = "api/plantilla/funciones";
    let url = environment.urlApiPlantilla + request;
    return this._http.get(url);
  }
  getPlantillaFuncion(funcion: string): Observable<any> {
    let request = "api/plantilla/plantillafuncion/" + funcion;
    let url = environment.urlApiPlantilla + request;
    return this._http.get(url);
  }
}
```

Como vamos a utilizar formularios, damos de alta Forms dentro de Module.  
Además incluimos el **Service** dentro de Providers

APP.MODULE.TS

```
imports: [
  BrowserModule,
  AppRoutingModule,
  FormsModule,
  PersonasstandaloneComponent
],
providers: [ServiceComics, ServicePersonas
  , provideHttpClient(), ServiceCoches
  , ServicePlantilla],
bootstrap: [AppComponent]
}
```

PLANTILLAFUNCIONSIMPLE.COMPONENT.TS

```
import { Component, ElementRef, OnInit, ViewChild } from '@angular/core';
```

```

import { ServicePlantilla } from '../../../../../services/service.plantillas';
import { Plantilla } from '../../../../../models/plantilla';
@Component({
  selector: 'app-plantilla-funcion-simple',
  templateUrl: './plantilla-funcion-simple.component.html',
  styleUrls: ['./plantilla-funcion-simple.component.css']
})
export class PlantillaFuncionSimpleComponent implements OnInit {
  @ViewChild("selectfuncion") selectfuncion!: ElementRef;
  public empleados: Array<Plantilla>;
  public funciones!: Array<string>;
  constructor(private _service: ServicePlantilla) {
    this.empleados = new Array<Plantilla>();
  }
  mostrarPlantilla(): void {
    let funcion = this.selectfuncion.nativeElement.value;
    this._service.getPlantillaFuncion(funcion).subscribe(response => {
      this.empleados = response;
    })
  }
  ngOnInit(): void {
    this._service.getFunciones().subscribe(response => {
      this.funciones = response;
    })
  }
}

```

#### PLANTILLAFUNCIONSIMPLE.COMPONENT.HTML

```

<div>
  <h1>Plantilla funciones simple</h1>
  <form #funcionesForm="ngForm">
    <select name="selectfuncion" #selectfuncion class="form-control">
      <for (funcion of funciones; track funcion)>
        <option value={{funcion}}>{{funcion}}</option>
      </for>
    </select>
    <button (click)="mostrarPlantilla()">
      Mostrar plantilla
    </button>
  </form>
  <table class="table table-bordered">
    <thead>
      <tr>
        <th>Apellido</th>
        <th>Función</th>
        <th>Turno</th>
        <th>Salario</th>
      </tr>
    </thead>
    <tbody>
      <for (emp of empleados; track emp)>
        <tr>
          <td>{{emp.apellido}}</td>
          <td>{{emp.funcion}}</td>
          <td>{{emp.turno}}</td>
          <td>{{emp.salario}}</td>
        </tr>
      </for>
    </tbody>
  </table>
</div>

```

Realizamos la versión 2.

## Plantilla funciones múltiple

Enfermera																											
Enfermero																											
Interino																											
<b>Mostrar plantilla</b>																											
<table border="1"> <thead> <tr> <th>Apellido</th><th>Función</th><th>Turno</th><th>Salario</th></tr> </thead> <tbody> <tr> <td>Higueras D.</td><td>Enfermera</td><td>T</td><td>547135</td></tr> <tr> <td>Amigo R.</td><td>Interino</td><td>N</td><td>222741</td></tr> <tr> <td>Díaz B.</td><td>Enfermera</td><td>T</td><td>338434</td></tr> <tr> <td>Rivera G.</td><td>Enfermera</td><td>N</td><td>509235</td></tr> <tr> <td>Karplus W.</td><td>Interino</td><td>T</td><td>881654</td></tr> </tbody> </table>				Apellido	Función	Turno	Salario	Higueras D.	Enfermera	T	547135	Amigo R.	Interino	N	222741	Díaz B.	Enfermera	T	338434	Rivera G.	Enfermera	N	509235	Karplus W.	Interino	T	881654
Apellido	Función	Turno	Salario																								
Higueras D.	Enfermera	T	547135																								
Amigo R.	Interino	N	222741																								
Díaz B.	Enfermera	T	338434																								
Rivera G.	Enfermera	N	509235																								
Karplus W.	Interino	T	881654																								

Creamos otro component llamado **plantillafuncionmultiple**

Necesito el mismo diseño y el mismo código.

Vamos a utilizar una nueva llamada con selección múltiple

<https://apiplantillacore.azurewebsites.net/api/plantilla/plantillafunciones?funcion=ENFERMERA&funcion=ENFERMERO&funcion=INTERINO>

Como hicimos en React, necesitamos tener múltiples elementos seleccionados y recibirlos  
Dentro del Servicio.

Vamos a recibir un Array de string y lo separaremos para la petición.

#### SERVICE.PLANTILLAS.TS

```

getPlantillaFunciones(funciones: Array<string>): Observable<any> {
  //?funcion=ENFERMERA&funcion=ENFERMERO&funcion=INTERINO
}

```

```

let data = "";
for (var funcion of funciones){
    data += "funcion=" + funcion + "&";
}
//funcion=ENFERMERA&funcion=ENFERMERO&funcion=INTERINO&
//ELIMINAMOS & DE LA ULTIMA POSICION
data = data.substring(0, data.length - 1);
let request = "api/plantilla/plantillafunciones?" + data;
let url = environment.urlApiPlantilla + request;
return this._http.get(url);
}

```

Modificamos el código HTML del Component múltiple

PLANTILLAFUNCIONMULTIPLE.COMPONENT.HTML

```

<h1 style="color: blue">
    Plantilla funciones múltiple
</h1>
<form #funcionesForm="ngForm">
    <select name="selectfunciones" #selectfunciones class="form-control" multiple>
        @for (funcion of funciones; track funcion){
            <option value={{funcion}}>{{funcion}}</option>
        }
    </select>

```

PLANTILLAFUNCIONMULTIPLE.COMPONENT.TS

```

import { Component, ElementRef, OnInit, ViewChild } from '@angular/core';
import { ServicePlantilla } from '../../../../../services/service.plantillas';
import { Plantilla } from '../../../../../models/plantilla';
@Component({
    selector: 'app-plantilla-funcion-multiple',
    templateUrl: './plantilla-funcion-multiple.component.html',
    styleUrls: ['./plantilla-funcion-multiple.component.css']
})
export class PlantillaFuncionMultipleComponent implements OnInit {
    @ViewChild("selectfunciones") selectfunciones!: ElementRef;
    public empleados: Array<Plantilla>;
    public funciones!: Array<string>;
    public funcionesSeleccionadas: Array<string>;
    constructor(private _service: ServicePlantilla) {
        this.empleados = new Array<Plantilla>();
        this.funcionesSeleccionadas = new Array<string>();
    }
    mostrarPlantilla(): void {
        let aux = new Array<string>();
        for (var option of this.selectfunciones.nativeElement.options) {
            if (option.selected == true) {
                aux.push(option.value);
            }
        }
        this.funcionesSeleccionadas = aux;
        this._service.getPlantillaFunciones(this.funcionesSeleccionadas).subscribe(response =>
        {
            this.empleados = response;
        })
    }
    ngOnInit(): void {
        this._service.getFunciones().subscribe(response => {
            this.funciones = response;
        })
    }
}

```

## CRUD DEPARTAMENTOS

Vamos a realizar la típica aplicación de CRUD con sus cosas.



Id departamento	Nombre	Localidad		
1	Ejemplo	Madrid	Details	Details
2	Casa	ALICANTE	Details	Edit
10	DISNEYLAND	PARIS	Details	Delete
20	PARQUE WARNER	MADRID	Details	Edit
30	PORT AVENTURA	TABARNIA	Details	Delete

Comenzamos creando el proyecto llamado **angularcruddepartamentos**

Esta es la URL que utilizaremos

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

Abrimos el proyecto y creamos environments

ng g environments

Incluimos la url de nuestro Api dentro de cada variable de entorno.

```
export const environment = {
  apiUrlDepartamentos: "https://apiejemplos.azurewebsites.net/"
};
```

Sobre index.html agregamos los links de bootstrap

Creamos un component menú y otro home y los ponemos mediante <router-outlet> en la navegación

```
<div>
  <app-menu></app-menu>
  <router-outlet></router-outlet>
</div>
```

Abrimos Module y agregamos la funcionalidad para formularios y servicios

APP MODULE.TS

```
@NgModule({
  declarations: [
    AppComponent,
    MenuComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule, FormsModule
  ],
  providers: [provideHttpClient()],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Sobre src/app creamos una carpeta llamada **models** y una clase llamada **departamento.ts**

```
export class Departamento {
  constructor(
    public idDepartamento: number,
    public nombre: string,
    public localidad: string
  ){}
}
```

Sobre src/app creamos una carpeta llamada **services** y dentro una clase llamada **service.departamentos.ts**

SERVICE.DEPARTAMENTOS.TS

```
import { HttpClient } from "@angular/common/http";
import { Injectable } from "@angular/core";
import { Observable } from "rxjs";
import { environment } from "../environments/environment";
@Injectable()
export class ServiceDepartamentos {
  constructor(private _http: HttpClient){}
  getDepartamentos(): Observable<any> {
    let request = "api/departamentos";
    let url = environment.apiUrlDepartamentos + request;
    return this._http.get(url);
  }
}
```

Incluimos dentro de **providers** de App Module nuestro nuevo Servicio

APP MODULE.TS

```

    AppRoutingModule, FormsModule
  ],
  providers: [provideHttpClient()
    , ServiceDepartamentos
  ],
  bootstrap: [AppComponent]
)
export class AppModule { }

```

Vamos a realizar el dibujo sobre HomeComponent,

#### HOME.COMPONENT.TS

```

import { Component, OnInit } from '@angular/core';
import { ServiceDepartamentos } from './services/service.departamentos';
import { Departamento } from '../models/departamento';
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
  public departamentos!: Array<Departamento>;
  constructor(private _service: ServiceDepartamentos){}
  ngOnInit(): void {
    this._service.getDepartamentos().subscribe(response => {
      this.departamentos = response;
    })
  }
}

```

#### HOME.COMPONENT.HTML

```

<div>
  <h1>Home departamentos</h1>
  <table class="table table-bordered">
    <thead>
      <tr>
        <th>Id departamento</th>
        <th>Nombre</th>
        <th>Localidad</th>
      </tr>
    </thead>
    <tbody>
      <for (dept of departamentos; track dept){>
        <tr>
          <td>{dept.idDepartamento}</td>
          <td>{dept.nombre}</td>
          <td>{dept.localidad}</td>
        </tr>
      </for>
    </tbody>
  </table>
</div>

```

A continuación, vamos a realizar **POST INSERTAR**

Comenzamos creando un componente llamado **create** y lo ponemos en nuestra navegación

Cuando estamos enviando información con **data** a un servicio, es decir, cuando estamos Enviando JSON, como sucede con el método **post** del Api, debemos indicar el tipo De formato que estamos enviando al servicio.

Debemos agregar en la llamada del **POST** otro objeto llamado **HttpHeaders**

#### SERVICE.DEPARTAMENTOS.TS

```

import { HttpClient, HttpHeaders } from "@angular/common/http";

//VOY A RECIBIR DIRECTAMENTE EL OBJETO EN EL METODO DE INSERTAR
insertDepartamento(departamento: Departamento): Observable<any>{
  //ESTO ES COMO JQUERY CONVERTIR UN OBJETO A JSON
  let json = JSON.stringify(departamento);
  //DEBEMOS INDICAR EN LA PETICION QUE TIPO DE FORMATO TIENE EL OBJETO A
ENVIAR
  let header = new HttpHeaders();
  header = header.set("Content-type", "application/json");
  let request = "api/departamentos";
  let url = environment.apiUrlDepartamentos + request;
  return this._http.post(url, json, {headers: header});
}

```

Realizamos el diseño del formulario en el componente **CREATE**

#### CREATE.COMPONENT.HTML

```

<div>
  <form #createForm="ngForm">
    <label>Id departamento</label>
    <input type="number" class="form-control"
    name="cajaid" #cajaid>
    <label>Nombre</label>
    <input type="text" class="form-control"
    name="cajanombre" #cajanombre/>
    <label>Localidad</label>
    <input type="text" class="form-control"
    name="cajalocalidad" #cajalocalidad/>
    <button class="btn btn-info" (click)="insertDepartamento()">
      Crear departamento
    </button>
  </form>
</div>

```

#### CREATE.COMPONENT.TS

```
import { Component, ElementRef, ViewChild } from '@angular/core';
import { ServiceDepartamentos } from '../../../../../services/service.departamentos';
import { Router } from '@angular/router';
import { Departamento } from '../../../../../models/departamento';
@Component({
  selector: 'app-create',
  templateUrl: './create.component.html',
  styleUrls: ['./create.component.css']
})
export class CreateComponent {
  @ViewChild("cajaId") cajaId!: ElementRef;
  @ViewChild("cajanombre") cajaNombre!: ElementRef;
  @ViewChild("cajalocalidad") cajaLocalidad!: ElementRef;
  constructor(
    private _service: ServiceDepartamentos,
    private _router: Router
  ) {}
  insertDepartamento(): void {
    let num = parseInt(this.cajaId.nativeElement.value);
    let nom = this.cajaNombre.nativeElement.value;
    let loc = this.cajaLocalidad.nativeElement.value;
    let newDepartamento = new Departamento(num, nom, loc);
    this._service.insertDepartamento(newDepartamento).subscribe(response => {
      this._router.navigate(['/']);
    })
  }
}
```

Lo siguiente que vamos a realizar será Detalles. Enviaremos los datos al component

Mediante Routing.

Recuperamos los datos y los dibujamos, no tendremos un método en el servicio

Para buscar un departamento.

Creamos un nuevo component llamado **details** y lo ponemos dentro de app routing.

#### APP-ROUTING.TS

```
const routes: Routes = [
  {path: "", component: HomeComponent},
  {path: "create", component: CreateComponent},
  {path: "details/:id/:nombre/:localidad", component: DetailsComponent}
];
```

El siguiente paso será poner un link dentro de Home component enviando los parámetros

Tenemos dos formas:

1) Concatenado

```
<a [routerLink]="/details/" + dept.idDepartamento
+ '/' + dept.nombre + '/' + dept.Localidad" class="btn btn-dark">
  Details
</a>
```

1) Separado por comas

```
<a [routerLink]="/details", dept.idDepartamento, dept.nombre, dept.Localidad"
class="btn btn-outline-dark">
  Details
</a>
```

El paso siguiente es ir al component y recuperar los parámetros

#### DETAILS.COMPONENT.TS

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Params } from '@angular/router';
import { Departamento } from '../../../../../models/departamento';
@Component({
  selector: 'app-details',
  templateUrl: './details.component.html',
  styleUrls: ['./details.component.css']
})
export class DetailsComponent implements OnInit {
  public departamento!: Departamento;
  constructor(
    private _activeRoute: ActivatedRoute
  ) {}
  ngOnInit(): void {
    this._activeRoute.params.subscribe((params: Params) => {
      let id = parseInt(params["id"]);
      let nom = params["nombre"];
      let loc = params["localidad"];
      this.departamento = new Departamento(id, nom, loc);
    })
  }
}
```

#### DETAILS.COMPONENT.HTML

```
<div>
  <h1>Details</h1>
  <p><a [routerLink]="/">Back to Home</a></p>
  <ul class="list-group">
```

```

<li class="list-group-item">Id departamento
{{departamento.idDepartamento}}</li>
<li class="list-group-item">Nombre {{departamento.nombre}}</li>
<li class="list-group-item">Localidad {{departamento.localidad}}</li>
</ul>
</div>

```

Vamos a realizar un component para editar el departamento.  
Esta vez, enviaremos el ID del departamento mediante Routing.

Posteriormente, mediante un método en el Service, recuperamos el Id y buscamos el departamento.

Creamos un nuevo component llamado `edit` y lo ponemos a jugar en Routing mediante Un parámetro llamado `:id`

APP.ROUTING.TS

```

const routes: Routes = [
  {path: "", component: HomeComponent},
  {path: "create", component: CreateComponent},
  {path: "details/:id/:nombre/:localidad", component: DetailsComponent},
  {path: "edit/:id", component: EditComponent}
];

```

Incluimos un RouterLink dentro del component Home para enviar la información del Departamento a editar.

HOME.COMPONENT.HTML

```

<a [routerLink]="['/edit', dept.idDepartamento]" class="btn btn-info">
  Edit
</a>

```

El siguiente paso es crear en el Servicio un nuevo método que recibirá el ID del departamento Y devolveremos el propio departamento con dicho ID.

SERVICE.DEPARTAMENTOS.TS

```

findDepartamento(idDepartamento: string): Observable<any> {
  let request = "api/departamentos/" + idDepartamento;
  let url = environment.apiUrlDepartamentos + request;
  return this._http.get(url);
}

```

```

updateDepartamento(departamento: Departamento): Observable<any> {
  let json = JSON.stringify(departamento);
  let header = new HttpHeaders().set("Content-type", "application/json");
  let request = "api/departamentos";
  let url = environment.apiUrlDepartamentos + request;
  return this._http.put(url, json, {headers: header});
}

```

Sobre el component Edit, recuperamos el Id mediante params y mostramos los datos del Departamento en cajas.

EDITCOMPONENT.HTML

```

<div>
  <h1>Edit</h1>
  <p>
    <a [routerLink]="/">Back to list</a>
  </p>
  <form #editForm="ngForm">
    <input type="hidden" name="cajaid" #cajaid
    value="{{departamento.idDepartamento}}>/>
    <label>Nombre</label>
    <input type="text" class="form-control"
    name="cajanombre" #cajanombre
    value="{{departamento.nombre}}>/>
    <label>Localidad</label>
    <input type="text" class="form-control"
    name="cajalocalidad" #cajalocalidad
    value="{{departamento.localidad}}>/>
    <button class="btn btn-info" (click)="updateDepartamento()">
      Update
    </button>
  </form>
</div>

```

EDITCOMPONENT.TS

```

import { Component, ElementRef, OnInit, ViewChild } from '@angular/core';

```

```

import { ServiceDepartamentos } from '../../../../../services/service.departamentos';
import { ActivatedRoute, Params, Router } from '@angular/router';
import { Departamento } from '../../../../../models/departamento';
@Component({
  selector: 'app-edit',
  templateUrl: './edit.component.html',
  styleUrls: ['./edit.component.css']
})
export class EditComponent implements OnInit {
  @ViewChild("cajaId") cajaId: ElementRef;
  @ViewChild("cajanombre") cajaNombre!: ElementRef;
  @ViewChild("cajalocalidad") cajaLocalidad!: ElementRef;
  public departamento!: Departamento;
  constructor(
    private _service: ServiceDepartamentos,
    private _activeRoute: ActivatedRoute,
    private _router: Router
  ){}
  updateDepartamento(): void{
    let id = parseInt(this.cajaId.nativeElement.value);
    let nom = this.cajaNombre.nativeElement.value;
    let loc = this.cajaLocalidad.nativeElement.value;
    let editDepartamento = new Departamento(id, nom, loc);
    this._service.updateDepartamento(editDepartamento).subscribe(response => {
      this._router.navigate(['/']);
    });
  }
  ngOnInit(): void {
    this._activeRoute.params.subscribe((params: Params) => {
      let id = params["id"];
      this._service.findDepartamento(id).subscribe(response => {
        this.departamento = response;
      });
    });
  }
}

```

El último paso es realizar la acción de eliminar un departamento.

Tenemos dos posibilidades:

- 1) Eliminar mediante un router link. Necesitamos un component apuntando a delete

```
<a [routerLink]="/delete", 10">Delete</a>
```

- 2) Eliminar mediante un botón. No vamos a tener un component, lo que haremos será Directamente eliminar en Home

```
<button (click)="deleteDepartamento(10)">Delete</button>
```

Creamos un método para eliminar dentro de Service

SERVICE.DEPARTAMENTOS.TS

```

deleteDepartamento(idDepartamento: string): Observable<any> {
  let request = "api/departamentos/" + idDepartamento;
  let url = environment.apiUrlDepartamentos + request;
  return this._http.delete(url);
}

```

El siguiente paso es modificar el código de HomeComponent

Necesitamos un método para cargar los departamentos al iniciar nuestra App y También al eliminar.

Creamos un método llamado loadDepartamentos()

HOME.COMPONENT.TS

```

import { Component, OnInit } from '@angular/core';
import { ServiceDepartamentos } from '../../../../../services/service.departamentos';
import { Departamento } from '../../../../../models/departamento';
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
  public departamentos!: Array<Departamento>;
  constructor(private _service: ServiceDepartamentos){}
  loadDepartamentos(): void {
    this._service.getDepartamentos().subscribe(response => {
      this.departamentos = response;
    });
  }
  ngOnInit(): void {
    this.loadDepartamentos();
  }
  deleteDepartamento(id: number): void {
    this._service.deleteDepartamento(id.toString()).subscribe(response => {
      this.loadDepartamentos();
    });
  }
}

```

HOME.COMPONENT.HTML

```

<button (click)="deleteDepartamento(dept.idDepartamento)"
class="btn btn-danger">
  I Delete
</button>

```

El siguiente ejemplo que tenéis que hacer es un reto.

Vamos a realizar una aplicación para dibujar Empleados.

```
200 Response body
{
  "idEmpleado": 7839,
  "apellido": "REY",
  "oficio": "PRESIDENTE",
  "salario": 650000,
  "director": 0
}
```

Tenemos un problema para dibujar los empleados...

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

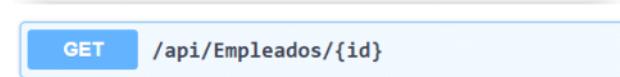
Nuestro Api tiene seguridad.

Dicho Api, mediante un Login necesita obtener un token para realizar las peticiones.

**Nota:** No nos sirve Swagger

Abrimos **Insomnia** y visualizamos cómo podemos realizar peticiones seguras con Usuario/password y token

El único método libre es el de buscar mediante ID



ID: 7839, 7521, 7566

Necesitamos un Usuario y un Password para recuperar el Token

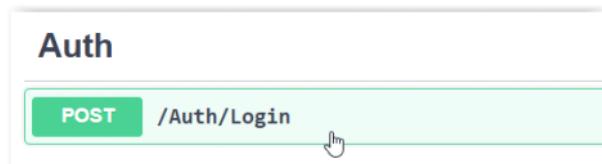
USUARIO: APELLIDO  
PASSWORD: ID EMPLEADO

USUARIO: REY  
PASSWORD: 7839

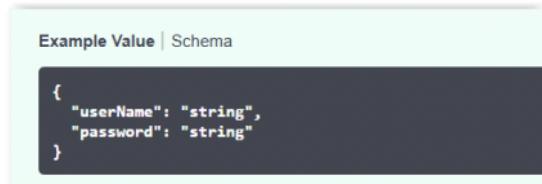
Para poder realizar este tipo de peticiones seguras, debemos hacerlo en dos pasos:

- 1) Enviar una petición al servidor indicando el POST de Auth.
- 2) Nos devolverá un Token
- 3) Con el token, enviar las peticiones a los métodos seguros

Vamos a realizar la prueba con los siguientes endpoints:



La información para este método es la siguiente:



Una vez que hacemos la petición, recibimos el TOKEN

POST https://apiempleadoscoreoauth Send 200 OK 68 ms 382 B Just Now

Params 3 Body 4 Auth Headers 4 Scripts Preview Headers 5 Cookies Tests 0 / 0

JSON Preview

```

1 var {
2   "userName": "REY" ,
3   "password": "7839"
4 }

```

```

1 var {
2   "response":
3     "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e
yJVc2VyRGF0YSI6IntcIk1kRWiwbGVhZG9cIjo3
ODM5LFwiQXB1bGxpZG9cIjpcIlJFWwiLFwit2Z
pY2lvXCI6XCJQUkVTSURFTlRFXCIsXCJTYWxhcm
1vXCI6NjUwMDAwLFwiRGlyZWN0b3IjowfSIsI
mSiIi6MTczMTQwNjUxMywiZXhwIjoxNzNDA3
MTEzLCJpc3MiOiJodHRwczovL2xvY2FsaG9zdDo
0NDM3NS8iLCJhdWQiOijBcG1FbXBsZWfk3NDb3
J1t0F1dGifQ.rxX2ln0FAzSAoXpHrGTMSYRf7T
pXvM6c6RurXeh5XZ0"
3 }

```

Por último, con el Token generado, ya podemos realizar peticiones a los métodos Seguros, enviando en el **HEADER** la siguiente key/value

KEY		TOKEN
-----	--	-------

authorization bearer token

GET https://apiempleadoscoreoauth.a Send 200 OK 159 ms 1952 B Just Now

Params Body Auth Headers 4 Scripts Preview Headers 5 Cookies Tests

+ Add Delete all Description

Accept \*/\*

Host <calculated at runtime>

User-Agent insomnia/10.1.0

Authorization bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e

```

1 var [
2   {
3     "idEmpleado": 7322,
4     "apellido": "FORD",
5     "oficio": "VENDEDOR",
6     "salario": 129010,
7     "director": 7919
8   },
9   {
10    "idEmpleado": 7369,
11    "apellido": "SANCHÁ",
12    "oficio": "EMPLEADO",
13    "salario": 104254,
14    "director": 7902

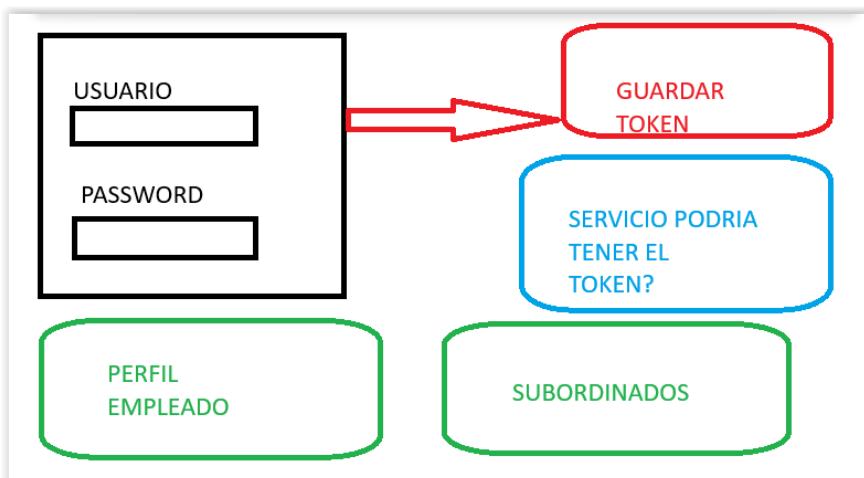
```

Realizar una aplicación llamada **angularempieadosauth** en la que debemos realizar las siguientes acciones:

- Tendremos un componente Login para validar usuario/password
- Tendremos otro componente para visualizar el Perfil del empleado
- Tendremos otro componente para visualizar los subordinados del empleado
- Intentar pedir el Token solamente una vez y almacenarlo en algún sitio para Utilizarlo (localStorage, Global, environment???)

#### USUARIOS:

- SALA, 7521
- REY, 7839
- FORD, 7919
- JIMENEZ, 7566



Solución ejemplo:

<https://github.com/serraguti/angularempleadosauth2024>



Home Perfil Subordinados Login

User

Password



Home Perfil Subordinados Login

## Perfil del empleado

Apellido REY
Oficio: PRESIDENTE
Salario: 650000
Director: 0



Home Perfil Subordinados Login

## Subordinados

Apellido	Oficio	Salario
JIMENEZ	DIRECTOR	386789
NEGRO	DIRECTOR	370539
CEREZO	DIRECTOR	318539
SERRA	DIRECTOR	390039

### EMPLEADO.TS

```
export class Empleado {
  constructor(
    public idEmpleado: number,
    public apellido: string,
    public oficio: string,
    public salario: number,
    public director: number
  ) {}
}
```

### LOGIN.TS

```
export class Login {
  constructor(
    public userName: string,
    public password: string
  ) {}
}
```

### SERVICE.EMPLEADOS.TS

```
import { Injectable } from "@angular/core";
import { Login } from "../models/login";
import { Observable } from "rxjs";
import { HttpClient, HttpHeaders } from "@angular/common/http";
import { environment } from "../../environments/environment";
@Injectable()
export class ServiceEmpleados {
  public token: string;
  constructor(private _http: HttpClient) {
    this.token = "";
  }
  loginEmpleado(user: Login): Observable<any> {
    let json = JSON.stringify(user);
    let header = new HttpHeaders().set("Content-type", "application/json");
    let request = "auth/login";
    let url = environment.apiUrlEmpleados + request;
    return this._http.post(url, json, {headers: header});
  }
  getPerfilEmpleado(): Observable<any> {
    let request = "api/empleados/perfilempleado";
    let url = environment.apiUrlEmpleados + request;
    let header = new HttpHeaders().set("Authorization", "bearer " +
    this.token);
  }
}
```

```
    return this._http.get(url, {headers: header});
}
getSubordinados(): Observable<any> {
  let request = "api/empleados/subordinados";
  let url = environment.apiUrlEmpleados + request;
  let header = new HttpHeaders().set("Authorization", "bearer " +
this.token);
  return this._http.get(url, {headers: header});
}
```