UNIDAD 7.1 AJAX

https://lenguajejs.com/javascript/peticiones-http/ajax/

Para esta unidad necesitamos un servidor web en algunos ejemplos que ejecutan php en el servidor, podemos montarlo de manera sencilla utilizando el instalador XAMPP(en caso de no usar PHP utilizamos la extensión live server):

Download XAMPP

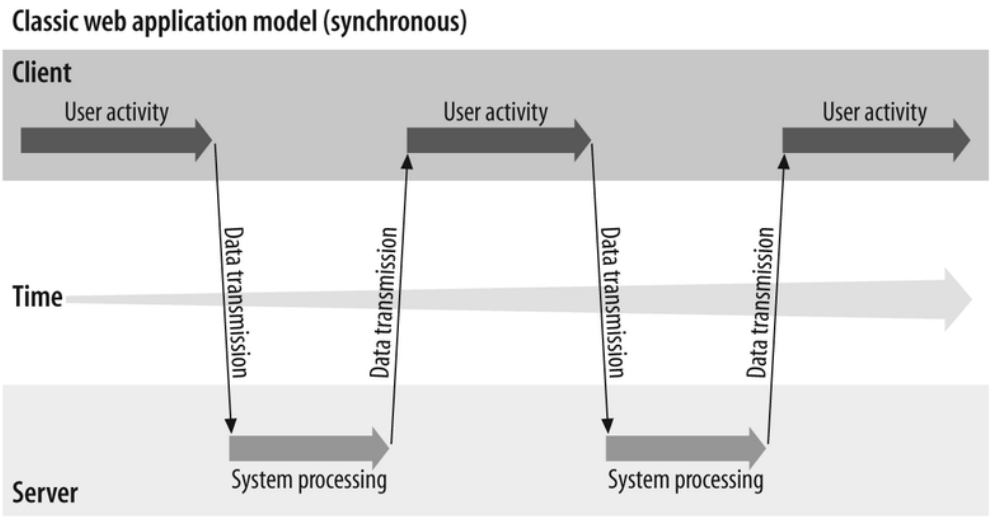
AJAX es el acrónimo de *Asynchronous JavaScript And XML* (JavaScript asíncrono y XML), una técnica de desarrollo web que nos permite crear aplicaciones interactivas que se ejecutan en el lado del cliente (navegador) mientras se realiza una comunicación con el servidor en segundo plano.

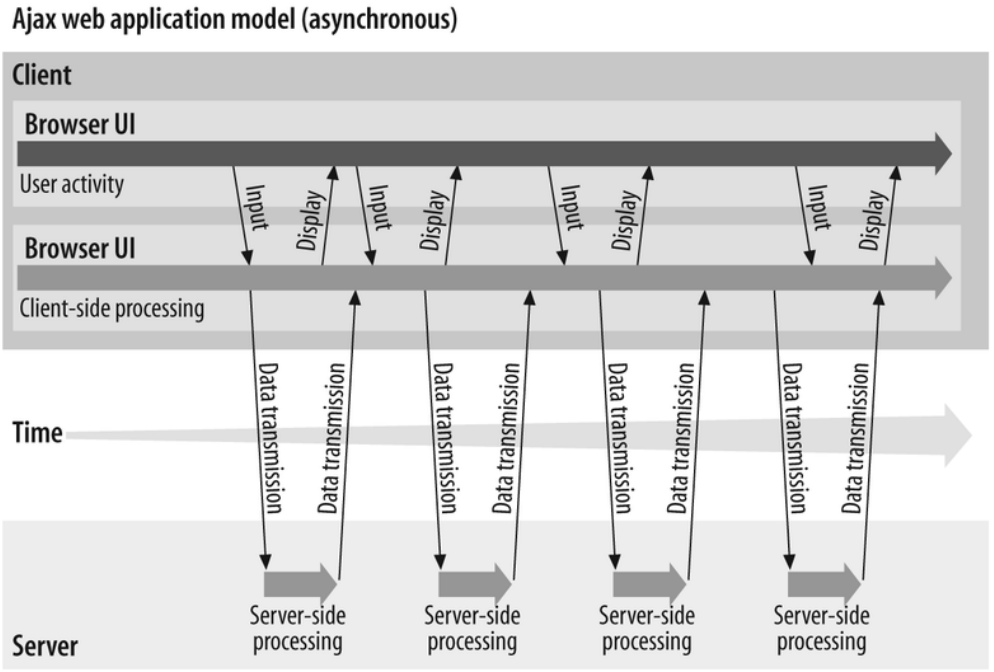
El resultado es una petición al servidor con una obtención de datos sin necesidad de que la página se recargue, es decir, los datos se cargan en una parte de la página mientras que el resto de información se mantiene invariable, de modo que esa carga no interfiere en la visualización ni comportamiento de la página.

Las ventajas de utilizar AJAX son muchas: se reduce la velocidad de carga, se mejora la interactividad y, sobre todo, la usabilidad de las aplicaciones web.

Para trabajar con AJAX necesitamos, evidentemente, utilizar Javascript, que es el lenguaje que realiza las llamadas AJAX mientras que el acceso a los datos se realiza a través de un objeto llamado **XMLHttpRequest**. Sin embargo, y aunque el nombre pueda despistar, los datos no tienen porqué estar formateados en XML; podemos utilizar también archivos de texto, archivos JSON, etc.

En la siguiente imagen se puede ver un gráfico muy claro en el que se comparan las peticiones síncronas con las peticiones asíncronas utilizando AJAX:





NOTA:

CTRL+SHIFT+R recarga la página de google chrome sin hacer uso de la caché

Podemos también evitar el uso de la caché poniendo en el head los siguientes encabezados en el HEAD:

```
<meta http-equiv="Cache-Control" content="no-cache, no-store, must-revalidate" />
<meta http-equiv="Pragma" content="no-cache" />
<meta http-equiv="Expires" content="0" />
```

Ref. https://cristian.sulea.net/blog/disable-browser-caching-with-meta-html-tags/

Acceso a archivos de texto

Mediante **AJAX** es posible **cargar contenido de un archivo de texto desde el servidor de manera asíncrona** utilizando una serie de métodos y propiedades que debemos conocer y que son propios del objeto ***XMLHttpRequest (XHR)***

Estos **métodos** son los siguientes:

- ***open (method, url, async):*** especifica el tipo de la petición, donde:
  - ***method***: puede ser GET o POST.
  - ***url***: la localización del archivo.
  - ***async***: *true* (asíncrono) o *false* (síncrono).
- ***send()***: envía la petición al servidor (utilizado para GET).
- ***send(cadena)***: envía la petición al servidor (utilizado para POST).

Propiedades importantes:
readyState : Nos interesa sobre todo cuando el valor es 4 (solicitud finalizada ok)
status: Nos interesa sobre todo cuando el valor es 200 (correcto)
responseText: Obtenemos el contenido de un archivo de texto solicitado

Ver métodos y propiedades en:
https://www.w3schools.com/js/js_ajax_http.asp

Veamos un ejemplo, de envío de un archivo de texto:

```
<!DOCTYPE html>
<html>

<body>
    <div id="texto">
        <h1>AJAX</h1>
    </div>
    <button id="cambiaContenido">Cambia el contenido</button>
    <script>
        document.getElementById("cambiaContenido").addEventListener("click",
cambiaContenido);

        function cambiaContenido() {
            var xhr = new XMLHttpRequest();
            xhr.addEventListener("readystatechange", function () {
if (this.readyState == 4 && this.status == 200) {
                    document.getElementById("texto").innerHTML =
this.responseText;
                }
             });
            /* .open: especifica la solicitud
             - GET/POST.
             - Archivo: txt, php, xml, json, etc.
             - true/false: asincrono ó sincrono. */
            xhr.open("GET", "holamundo.txt", true);
            /* .send: envía la solicitud al servidor.
                Si utilizamos POST debemos pasar los datos por parámetro */
            xhr.send();
        }
    </script>

</body>

</html>
```

Podríamos substituir el control en detalle del cambio de estado, añadiendo una función al evento load del objeto XMLHttpRequest, en la función cargada podríamos acceder al this.responseText:

```
xhr.addEventListener("load", cargada);
```

Acceso a archivos XML

Para acceder a un archivo XML que está en el servidor usamos los mismos **métodos y propiedades** que vimos en el párrafo anterior cambiando la propiedad responseText por la propiedad *responseXML*

Veámoslo con un ejemplo:

Tenemos el siguiente archivo XML, alojado en nuestro sitio web

```xml
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Greatest Hits</TITLE>
    <ARTIST>Dolly Parton</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>RCA</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1982</YEAR>
  </CD>
  <CD>
    <TITLE>Still got the blues</TITLE>
    <ARTIST>Gary Moore</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Virgin records</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
    <TITLE>Eros</TITLE>
    <ARTIST>Eros Ramazzotti</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>BMG</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
```

```
</CD>
<CD>
  <TITLE>One night only</TITLE>
  <ARTIST>Bee Gees</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Polydor</COMPANY>
  <PRICE>10.90</PRICE>
  <YEAR>1998</YEAR>
</CD>
<CD>
  <TITLE>Sylvias Mother</TITLE>
  <ARTIST>Dr.Hook</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>CBS</COMPANY>
  <PRICE>8.10</PRICE>
  <YEAR>1973</YEAR>
</CD>
<CD>
  <TITLE>Maggie May</TITLE>
  <ARTIST>Rod Stewart</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Pickwick</COMPANY>
  <PRICE>8.50</PRICE>
  <YEAR>1990</YEAR>
</CD>
<CD>
  <TITLE>Romanza</TITLE>
  <ARTIST>Andrea Bocelli</ARTIST>
  <COUNTRY>EU</COUNTRY>
  <COMPANY>Polydor</COMPANY>
  <PRICE>10.80</PRICE>
  <YEAR>1996</YEAR>
</CD>
<CD>
  <TITLE>When a man loves a woman</TITLE>
  <ARTIST>Percy Sledge</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>Atlantic</COMPANY>
  <PRICE>8.70</PRICE>
  <YEAR>1987</YEAR>
</CD>
<CD>
  <TITLE>Black angel</TITLE>
  <ARTIST>Savage Rose</ARTIST>
  <COUNTRY>EU</COUNTRY>
  <COMPANY>Mega</COMPANY>
  <PRICE>10.90</PRICE>
  <YEAR>1995</YEAR>
</CD>
<CD>
  <TITLE>1999 Grammy Nominees</TITLE>
  <ARTIST>Many</ARTIST>
  <COUNTRY>USA</COUNTRY>
```

```
  <COMPANY>Grammy</COMPANY>
  <PRICE>10.20</PRICE>
  <YEAR>1999</YEAR>
</CD>
<CD>
  <TITLE>For the good times</TITLE>
  <ARTIST>Kenny Rogers</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Mucik Master</COMPANY>
  <PRICE>8.70</PRICE>
  <YEAR>1995</YEAR>
</CD>
<CD>
  <TITLE>Big Willie style</TITLE>
  <ARTIST>Will Smith</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>Columbia</COMPANY>
  <PRICE>9.90</PRICE>
  <YEAR>1997</YEAR>
</CD>
<CD>
  <TITLE>Tupelo Honey</TITLE>
  <ARTIST>Van Morrison</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Polydor</COMPANY>
  <PRICE>8.20</PRICE>
  <YEAR>1971</YEAR>
</CD>
<CD>
  <TITLE>Soulsville</TITLE>
  <ARTIST>Jorn Hoel</ARTIST>
  <COUNTRY>Norway</COUNTRY>
  <COMPANY>WEA</COMPANY>
  <PRICE>7.90</PRICE>
  <YEAR>1996</YEAR>
</CD>
<CD>
  <TITLE>The very best of</TITLE>
  <ARTIST>Cat Stevens</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Island</COMPANY>
  <PRICE>8.90</PRICE>
  <YEAR>1990</YEAR>
</CD>
<CD>
  <TITLE>Stop</TITLE>
  <ARTIST>Sam Brown</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>A and M</COMPANY>
  <PRICE>8.90</PRICE>
  <YEAR>1988</YEAR>
</CD>
<CD>
```

```
  <TITLE>Bridge of Spies</TITLE>
  <ARTIST>T'Pau</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Siren</COMPANY>
  <PRICE>7.90</PRICE>
  <YEAR>1987</YEAR>
</CD>
<CD>
  <TITLE>Private Dancer</TITLE>
  <ARTIST>Tina Turner</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Capitol</COMPANY>
  <PRICE>8.90</PRICE>
  <YEAR>1983</YEAR>
</CD>
<CD>
  <TITLE>Midt om natten</TITLE>
  <ARTIST>Kim Larsen</ARTIST>
  <COUNTRY>EU</COUNTRY>
  <COMPANY>Medley</COMPANY>
  <PRICE>7.80</PRICE>
  <YEAR>1983</YEAR>
</CD>
<CD>
  <TITLE>Pavarotti Gala Concert</TITLE>
  <ARTIST>Luciano Pavarotti</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>DECCA</COMPANY>
  <PRICE>9.90</PRICE>
  <YEAR>1991</YEAR>
</CD>
<CD>
  <TITLE>The dock of the bay</TITLE>
  <ARTIST>Otis Redding</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>Atlantic</COMPANY>
  <PRICE>7.90</PRICE>
  <YEAR>1987</YEAR>
</CD>
<CD>
  <TITLE>Picture book</TITLE>
  <ARTIST>Simply Red</ARTIST>
  <COUNTRY>EU</COUNTRY>
  <COMPANY>Elektra</COMPANY>
  <PRICE>7.20</PRICE>
  <YEAR>1985</YEAR>
</CD>
<CD>
  <TITLE>Red</TITLE>
  <ARTIST>The Communards</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>London</COMPANY>
  <PRICE>7.80</PRICE>
```

```
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Unchain my heart</TITLE>
    <ARTIST>Joe Cocker</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>EMI</COMPANY>
    <PRICE>8.20</PRICE>
    <YEAR>1987</YEAR>
  </CD>
</CATALOG>
```

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <style>
        table,
        th,
        td {
            border: 1px solid black;
            border-collapse: collapse;
        }

        th,
        td {
            padding: 5px;
        }
    </style>
</head>

<body>
    <div id="texto">
        <h1>Colección de CDs</h1>
        <button id="cargaCatalogo">Carga catálogo</button>
    </div>
    <table id="demo"></table>

    <script>
        document.getElementById("cargaCatalogo").addEventListener("click",
cargarCatalogo);

        function cargarCatalogo() {
            var xhr = new XMLHttpRequest();
            xhr.onreadystatechange = function () {
                if (this.readyState == 4 && this.status == 200) {
                    cargarXML(this);
                }
            };
            xhr.open("GET", "catalogo.xml", true);
```

```
            xhr.send();
        }

        function cargarXML(xml) {
            var docXML = xml.responseXML;
            var tabla = "<tr><th>Artista</th><th>Titulo</th></tr>";
            var discos = docXML.getElementsByTagName("CD");
            for (var i = 0; i < discos.length; i++) {
                tabla += "<tr><td>";
                tabla += discos[i].getElementsByTagName("ARTIST")[0].textContent;
                tabla += "</td><td>";
                tabla += discos[i].getElementsByTagName("TITLE")[0].textContent;
                tabla += "</td></tr>";
            }
            document.getElementById("demo").innerHTML = tabla;
        }
    </script>

</body>

</html>
```

Podríamos también acceder a APIS open-source o de cualquier tipo para obtener datos para utilizar en nuestras aplicaciones.

Por ejemplo en la siguiente dirección obtenemos datos aleatorios de personas para poder probar nuestras aplicaciones, indicamos cuantas personas queremos obtener y en qué formato:

https://randomuser.me/api/?results=5&format=XML

poniendo esta línea obtendremos los datos de 5 personas en formato XML para utilizar como queramos:

```
 xhr.open("GET",
"https://randomuser.me/api/?results=5&format=XML", true);
```

AJAX. con PHP y envío con GET

Vamos a ver en esta ocasión cómo **manejar un array desde el lado del servidor con PHP**, y cómo **procesar la respuesta** recibida a través del ***this.responseText*** que viene de la instrucción echo del PHP

En esta ocasión lo haremos mediante una **petición con GET**, por lo que la URL con sus parámetros irá en texto plano en el **método *open()***. que es el que especifica la solicitud(GET o POST, y si será sincrona o asincrona).

En el envío con GET tenemos una limitación de 2000 caracteres y los datos serán envíados en la URL.

Veamos un ejemplo:

```
<html>

<head>
    <meta charset="utf-8" />
    <title>Ajax con PHP</title>
    <script>
        window.addEventListener("load", inicio);

        function inicio() {
            document.getElementById("nombre").addEventListener("keyup",
mostrarNombre);
        }

        function mostrarNombre(e) {
            var cadena = e.target.value;
            //var cadena = document.getElementById("nombre").value;

            if (cadena.length == 0) { //Si al levantar la tecla no hay nada (ej.al
borrar)
                document.getElementById("sugerencia").innerHTML = "";
                return;
            } else {
                var xhr = new XMLHttpRequest();
                xhr.onreadystatechange = function () {
```

```
                    if (this.readyState == 4 && this.status == 200) {
                        document.getElementById("sugerencia").innerHTML =
this.responseText;
                    }
                };
                xhr.open("GET", "arraynombres.php?nombre=" + cadena, true);
                xhr.send();
            }
        }
    </script>
</head>

<body>
    <p><b>Escribe un nombre en el cuadro inferior:</b></p>
    <form>
        Nombre:
        <input type="text" id="nombre" />
    </form>
    <p>Sugerencias: <span id="sugerencia"></span></p>
</body>

</html>
```

A continuación el archivo arraynombres.php

```
<?php
 //Array de nombres
$a = array("Sara","Imanol","Dani","Antonio","David","Igor", "Naroa", "Christian",
"Joseba", "Angel", "Alex", "Dumitru", "Mikel", "Ivan", "Martin");

//Tomamos el valor del input procedente de la URL, utilizando el array $_REQUEST nos
// servirá tanto para GET como para POST
$nombre = $_REQUEST["nombre"];
$sugerencia = "";

if ($nombre!==""){
    $nombre = strtolower($nombre); //Pasamos el nombre a minúsculas
    $long = strlen($nombre);

    foreach($a as $nom){//Cada elemento del array lo pasa a $nom en cada iteración
        if(stristr($nombre, substr($nom, 0, $long))){ //Si coincide la cadena pasada
con los primeros caracteres de algún elemento del array
            if($sugerencia === ""){ //Si no hay texto en sugerencia
                $sugerencia = $nom;
            }else{
                $sugerencia = "$sugerencia, $nom";
            }
        }
    }
```

```
}
/*if ($sugerencia === "") echo "No hay sugerencias";
else echo $sugerencia;*/
echo ($sugerencia ===""") ? "No hay sugerencias" : $sugerencia;
?>
```

AJAX. con PHP y envío con POST

La diferencia con el caso anterior es que necesitamos añadir una cabecera HTTP mediante el método ***setRequestHeader()*** para indicar el tipo de datos a enviar y especificar los datos en el método ***send()***.

El cambio con respecto al envío con GET sería el siguiente:

```
//xhr.open("GET","arraynombres.php?nombre="+cadena,true);
  xhr.open("POST", "arraynombres.php", true);
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
//xhr.send(); //En caso de que utilizáramos GET
  xhr.send("nombre=" + cadena);
```

Sintaxis de JSON

Cuando trabajamos con mucha cantidad de información, se puede volver necesario separar nuestro código de programación de los datos que aparecen en él. De esta forma, podemos guardar la información en un archivo independiente, separado del archivo donde tenemos el código de nuestro programa. Así, si necesitamos actualizar o modificar datos, no tenemos que tocar el código de nuestro programa.

¿Qué es JSON?

JSON son las siglas de JavaScript Object Notation, y no es más que un formato ligero de datos, con una estructura (notación) específica, que es totalmente compatible de forma nativa con Javascript. Como su propio nombre indica, JSON se basa en la sintaxis que tiene Javascript para crear objetos.

Un archivo JSON mínimo suele tener la siguiente sintaxis:

```
{
}
```

Sin embargo, su contenido puede ser simplemente un array , un number , un string , un boolean o incluso un null , sin embargo, lo más habitual es que parta siendo un objecto o un array .

Puedes comprobar en JSONLint si algo concreto es un JSON válido o no.

Este ejemplo anterior simplemente es un objeto vacío {}. Un archivo JSON, suele contener mucha información almacenada. Vamos a modificar ese objeto vacío para que contenga más datos para ejemplificarlo:

```
{
  "name": "Manz",
  "life": 3,
  "totalLife": 6
  "power": 10,
  "dead": false,
  "props": ["invisibility", "coding", "happymood"],
  "senses": {
    "vision": 50,
    "audition": 75,
    "taste": 40,
    "touch": 80
  }
}
```

Si comparamos un JSON con un objeto Javascript, aparecen algunas ligeras diferencias y matices:

- Las propiedades del objeto deben estar entrecomilladas con «comillas dobles»
- Los textos deben estar entrecomillados con «comillas dobles»
- Sólo se puede almacenar tipos como array , number , string , boolean o null.
- Tipos de datos como function , Date, Regexp u otros, no es posible almacenarlos en un JSON.
- Tampoco es posible añadir comentarios en un JSON.

Hay formatos derivados de JSON que sí permiten comentarios, como es el caso de JSON5.
Mucho cuidado con las comillas mal cerradas o las comas sobrantes (antes de un cierre de llaves, por ejemplo). Suelen ser motivos de error de sintaxis frecuentemente. Existe una cómoda extensión para Visual Code llamada Fix JSON que te corrige los errores de formato de un JSON.

¿Cómo utilizar JSON?
Como hemos mencionado, si analizamos bien la sintaxis de un JSON, nos habremos dado cuenta que es muy similar a un objeto declarado Javascript, pero con ciertas diferencias:

```
const user = {
  name: "Manz",
  life: 99,
};
```

En Javascript tenemos una serie de métodos que nos facilitan la tarea de pasar de objeto de Javascript a JSON y viceversa, pudiendo trabajar con contenido de tipo string (que contenga un JSON) y objetos Javascript según interese.

Dichos métodos son los siguientes:

Método	Descripción
Parseo (De string a objeto)	
JSON.parse(str)	Convierte el texto str (si es un JSON válido) a un objeto javascript y lo devuelve.
Convertir a texto (De objeto a string)	

JSON.stringify(obj)	Convierte un objeto obj a su representación JSON y la devuelve.
JSON.stringify(obj, props)	Idem al anterior, pero filtra y mantiene solo las propiedades de props.
JSON.stringify(obj, props, spaces)	Idem al anterior, pero identa el JSON a spaces espacios.

El primero de ellos, el método .parse() nos va a permitir pasar el contenido de texto de un JSON a un objeto . En contrapartida, el método .stringify() nos va a permitir pasar de objeto de Javascript a contenido de texto con el JSON en cuestión.

Convertir JSON a objeto

La acción de convertir JSON a objeto Javascript se le suele denominar parsear. Es una acción que analiza un string que contiene un JSON válido y devuelve un objeto Javascript con dicha información correctamente estructurada. Para ello, utilizaremos el mencionado método JSON.parse():

```
const json = `{
  "name": "Manz",
  "life": 99
}`;
```

```
const user = JSON.parse(json);

user.name;  // "Manz"
user.life;  // 99
```

Como se puede ver, user es un objeto generado a partir del JSON almacenado en la variable json y podemos consultar sus propiedades y trabajar con ellas sin problemas.

Convertir objeto a JSON

La operación inversa, convertir un objeto Javascript a JSON también se puede realizar fácilmente haciendo uso del método JSON.stringify(). Este

método se puede utilizar para transformar un objeto de Javascript a JSON rápidamente:

```
const user = {
  name: "Manz",
  life: 99,
  talk: function () {
    return "Hola!";
  },
};

JSON.stringify(user);       // '{"name":"Manz","life":99}'
```

Observa que, como habíamos dicho, las funciones no están soportadas por JSON, por lo que si intentamos convertir un objeto que contiene métodos o funciones, JSON.stringify() no fallará, pero simplemente devolverá un string omitiendo las propiedades que contengan funciones (u otros tipos de datos no soportados).

Además, se le puede pasar un segundo parámetro al método .stringify(), que será un que actuará de filtro a la hora de generar el objeto. Observa el siguiente ejemplo:

```
const user = {
  name: "Manz",
  life: 99,
  power: 10,
};

JSON.stringify(user, ["life"])            // '{"life":99}'
JSON.stringify(user, ["name", "power"])   //
'{"name":"Manz","power":10}'
JSON.stringify(user, [])   // '{}'
JSON.stringify(user, null) //
'{"name":"Manz","life":99,"power":10}'
```

Observa que en el penúltimo caso, no se conserva ninguna propiedad, mientras que el último, se conserva todo.

Por último, también podemos añadir un tercer parámetro en el método .stringify() que indicará el número de espacios que quieres usar al crear el

del JSON resultante. Observa que hasta ahora, el JSON está minificado y aparece todo junto en la misma línea.
Observa lo que ocurre en los siguientes casos:

```
const user = {
  name: "Manz",
  life: 99
};

JSON.stringify(user, null, 2);
// {
//   "name": "Manz",
//   "life": 99
// }

JSON.stringify(user, null, 4);
// {
//     "name": "Manz",
//     "life": 99
// }

JSON.stringify(user, ["name"], 1);
// {
//  "name": "Manz"
// }
```

En el primer caso, json2, el resultado se genera indentado a 2 espacios. En el segundo caso, json4, el resultado se genera indentado a 4 espacios. En el tercer y último caso, json1, se filtran las propiedades, dejando sólo "name" y se genera indentando a 1 espacio.

Leyendo JSON externo

Ten en cuenta que los ejemplos realizados hasta ahora, estamos convirtiendo un string a objeto y viceversa. Normalmente los contenidos JSON suelen estar almacenados en un archivo externo, que habría que leer desde nuestro código Javascript.

Para ello, hoy en día se suele utilizar la función fetch() para hacer peticiones a sitios que devuelven contenido JSON. También se podría leer ficheros locales con contenido .json.

https://lenguajejs.com/javascript/objetos/json/

Ejemplos de objetos JSON:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
</head>

<body>
    <div id="demo" />
    <!-- JSON (Javascript Object Notation)
        - Sintaxis para guardar e intercambiar datos.
        - Texto escrito en notación de objetos de Javascript.
        - JSON es más corto que XML, más rápido de leer y escribir y puede usar
arrays. -->
    <script>
        /* SINTAXIS:
              - Datos: "nombre": "valor" //Siempre el nombre va entre comillas
              - Valor: número, booleano y null sin comillas; cadena (comillas);
               objetos (llaves);
               array (corchetes).
               No podemos enviar funciones, fechas ni undefined.
        Ej. var ada = {"nombre":"Ada", "nacimiento":1815, "pais":"Reino Unido"} */

        //OBJETOS: nombre del objeto y entre llaves, par nombre:valor
        var objeto1 = {
            "nombre": "Ada",
            "nacimiento": 1815,
            "pais": "Reino Unido"
        };

        //Acceso: utilizamos la notación punto o corchetes
        alert(objeto1.nombre + " : " + objeto1.nacimiento + " : " + objeto1.pais);
        //alert(objeto1["nombre"]+" : "+objeto1["nacimiento"]+" :
"+objeto1["pais"]);

        //Recorrer los nombres de un objeto
        for (var x in objeto1) {
            document.getElementById("demo").innerHTML += x + ":" + objeto1[x] +
"<br/>";
        }

        //Objetos que contienen otros objetos
        var objeto2 = {
            "nombre": "Ada",
            "nacimiento": 1815,
            "pais": "Reino Unido",
            "hijos": {
                "hijo1": "Anne Blunt",
                "hijo2": "Byron King-Noel",
                "hijo3": "Ralph King-Milbanke"
            }
        }
```

```
        //Acceso a un objeto dentro de otro objeto
        var y = objeto2.hijos.hijo1; //objeto2.hijos["hijo1"];
        alert("Hijo 1 es " + y);

        //ARRAYS: nombre del array, y entre corchetes los valores
        var hijos = {
            "hijos": ["Anne Blunt", "Byron King-Noel", "Ralph King-Milbanke"]
        }
        var objeto3 = {
                "nombre": "Ada",
                "nacimiento": 1815,
                "pais": "Reino Unido",
                "hijos": ["Anne Blunt", "Byron King-Noel", "Ralph King-Milbanke"]
            }
            //Acceso
        var z = objeto3.hijos[1];
        alert("Hijo 2 es " + z);

        //Recorrer los elementos de un array
        var z1 = "";
        for (var i in objeto3.hijos) {
            z1 += objeto3.hijos[i] + ", ";
        }
        document.getElementById("demo").innerHTML += "Todos los hijos: " + z1 +
"<br/>";

        var z2 = "";
        for (var i = 0; i < objeto3.hijos.length; i++) {
            z2 += objeto3.hijos[i];
        }
        document.getElementById("demo").innerHTML += "Todos los hijos: " + z2 +
"<br/>";

        //Si queremos borrar elementos de un objeto utilizamos "delete"
    </script>

</body>

</html>
```

Revisar distintos tipos de bucles for (for in y for of):

https://www.w3schools.com/js/js_loop_for.asp

Más información en:

https://desarrolloweb.com/home/json

https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON

NOTA:

En JavaScript, los nombres de las propiedades de un objeto pueden ir con comillas o sin comillas, pero hay algunas consideraciones importantes:

Sin comillas: Si el nombre de la propiedad es un identificador válido en JavaScript (una palabra clave o una combinación de letras, números y guiones bajos que comienza con una letra o un guión bajo), entonces no necesitas ponerlo entre comillas.

```
var persona = {
 nombre: "Ejemplo",
 edad: 25,
 ciudad: "Ejemplo City"
};
```

En este caso, nombre, edad y ciudad son identificadores válidos y no requieren comillas.

Con comillas: Si el nombre de la propiedad no es un identificador válido o si quieres usar una cadena que no sigue las reglas de identificación de JavaScript, debes poner el nombre de la propiedad entre comillas.

```
var persona = {
 "nombre completo": "Ejemplo",
 "año de nacimiento": 1999
};
```

> Aquí, las propiedades "nombre completo" y "año de nacimiento" contienen espacios y no son identificadores válidos, por lo que se colocan entre comillas.

Ambas formas son válidas, pero el uso común es evitar comillas si es posible y usarlas sólo cuando sea necesario, por ejemplo, cuando el nombre de la propiedad contiene espacios o caracteres especiales.

Extensión para ver la estructura del json de manera gráfica: JSON Crack

AJAX. con objetos en JSON

De esta manera, veremos cómo pueden **interactuar cliente y servidor con AJAX** mediante el uso de **objetos en JSON**.

```
<html>

<head>
    <meta charset="utf-8" />
    <title>Ajax con PHP y JSON</title>
    <script>
        window.addEventListener("load", inicio);

        function inicio() {
            document.getElementById("mostrar").addEventListener("click", mostrar);
        }

        function mostrar() {
            var xhr = new XMLHttpRequest();
            xhr.onreadystatechange = function () {
                if (this.readyState == 4 && this.status == 200) {
                    //Al hacer parse nos devuelve un objeto
                    var objeto = JSON.parse(this.responseText);

                    //Mostramos los datos
                    document.getElementById("parrafo").innerHTML = objeto.nombre + "
nacio en " + objeto.nacimiento + " en " + objeto.pais + "<br/>";

                    //Si queremos convertir un objeto Javascript en una cadena
```

```
                    var cadena = JSON.stringify(objeto);
                    document.getElementById("parrafo").innerHTML += "El objeto " +
objeto + " en modo cadena es " + cadena;
                }
            }
            xhr.open("GET", "datos.json", true);
            xhr.send();
        }
    </script>
</head>

<body>
    <button id="mostrar">Mostrar</button>
    <p id="parrafo"></p>
</body>

</html>
```

Ajax_datos.json

```
{
  "nombre": "Ana",
  "nacimiento": "4-9-1985",
  "pais": "España"
}
```

A su vez podemos acceder a los datos que nos devuelva una API open-source en formato JSON

A continuación podemos ver una API que devuelve datos aleatorios de personas, en este caso 5 personas en formato json: https://randomuser.me/api/?results=5&format=json

Cambiando esta línea, obtendríamos los datos aleatorios de 5 personas en en la variable objeto.

```
xhr.open("GET",
"https://randomuser.me/api/?results=5&format=json", true);
```

En el siguiente enlace nos muestran un relación de algunas API´s REST de uso libre que podemos utilizar para nuestros proyectos: https://desarrolloweb.com/colecciones/api-rest-uso-publico-libre

Información sobre JSON.parse()
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse
https://ichi.pro/es/una-lista-seleccionada-de-100-api-publicas-geniales-y-divertidas-para-inspirar-su-proximo-proyecto-8109114517939

AJAX. con arrays en JSON

Haremos algo similar, pero en lugar de trabajar con un **objeto** lo haremos con un **array (un json puede ser un objeto o un array)**

```
<html>

<head>
    <meta charset="utf-8" />
    <title>Ajax con PHP y JSON</title>
    <script>
        window.addEventListener("load", inicio);

        function inicio() {
            document.getElementById("mostrar").addEventListener("click", mostrar)
        }

        function mostrar() {
            var xhr = new XMLHttpRequest();
            xhr.onreadystatechange = function () {
                if (this.readyState == 4 && this.status == 200) {
                    //Al hacer parse de un array recibimos un array
                    var alumnos = JSON.parse(this.responseText);

                    //Recorremos el array
                    for (var i = 0; i < alumnos.length; i++) {
                        document.getElementById("parrafo").innerHTML += alumnos[i] +
"<br/>";
                    }

                    //Para convertir un array Javascript en cadena JSON usamos
stringfy
                    var cadena = JSON.stringify(alumnos);
                    document.getElementById("parrafo").innerHTML += "El array " +
alumnos + "en modo cadena es " + cadena + "<br/>";
                }
            }
            xhr.open("GET", "JSON_array.json", true);
            xhr.send();
        }
    </script>
</head>
```

```
<body>
    <button id="mostrar">Mostrar</button>
    <p id="parrafo"></p>
</body>

</html>
```

JSON_array.json

```
["Sara","Imanol","Dani","Antonio","David","Igor", "Naroa", "Christian", "Joseba",
"Angel", "Alex", "Dumitru", "Mikel", "Ivan", "Martin"]
```

También podemos acceder poniendo json en la propiedad responseType del XMLHttprequest antes de hacer el send:

```
xhr.responseType = 'json';
xhr.send();
```

Así obtendremos el objeto javascript directamente en la propiedad response:

```
var objetoJson = xhr.response;
```

AJAX. con acceso a base de datos y JSON

Ejercicio en el que, a partir de un valor introducido en un campo de texto, mostrará todos los alumn@s que han obtenido una puntuación superior y cuya información extraeremos de una base de datos.

En primer lugar, enviaremos un dato a través de un formulario HTML que recibiremos y procesaremos en PHP. Este dato lo decodificaremos mediante *json_decode* y extraeremos su información.

A continuación crearemos una conexión a una base de datos, y realizaremos una consulta a partir de los datos extraídos del objeto anterior.

La información generada en la consulta será codificada a JSON mediante *json_encode* y transferida mediante AJAX para su posterior procesamiento y visualización en la página.

```
<html>

<head>
    <meta charset="utf-8" />
    <title>Ajax con PHP y JSON</title>
    <script>
        window.addEventListener("load", inicio);

        function inicio() {
            document.getElementById("mostrar").addEventListener("click", mostrar);
        }

        function mostrar() {
            var puntos = document.getElementById("puntuacion").value;
            var objeto = {
                "tabla": "alumnos",
                "valor": parseInt(puntos)
            };

            var xhr = new XMLHttpRequest();
            var txt = "";
            xhr.onreadystatechange = function () {
                if (this.readyState == 4 && this.status == 200) {
                    var array = JSON.parse(this.responseText);
                    for (x in array) {
                        txt += array[x].alumno + " : " + array[x].puntuacion +
"<br/>";
                    }
                    document.getElementById("texto").innerHTML = txt;
                }
            }
            var parametros = JSON.stringify(objeto);
            /*xhr.open("GET", "Ajax_JSON_bbdd.php?objeto=" + parametros, true);
            xhr.send();*/
            xhr.open("POST", "Ajax_JSON_bbdd.php", true);
            xhr.setRequestHeader("Content-type",
"application/x-www-form-urlencoded");
            xhr.send("objeto=" + parametros);
        }
    </script>
</head>

<body>
    Puntuacion:
    <input type="number" id="puntuacion" />
    <button id="mostrar">Enviar</button>
    <div id="texto" />
</body>

</html>
```

Ajax_JSON_bbdd.php

```php
<?php
header("Content-Type: application/json; charset=UTF-8");

//$objeto = json_decode($_GET["objeto"],false);
$objeto = json_decode($_POST["objeto"],false);
//$objeto = json_decode('{"tabla":"alumnos","valor":111}');
$servidor = "localhost";
$usuario = "root";
$password = "";
$bbdd = "prueba";

//Crear la conexión
$conexion = new mysqli($servidor, $usuario, $password, $bbdd);

//Comprobamos la conexión
if ($conexion ->connect_error){
    die("Error en la conexion: "+$conexion->connect_error);
}else{
    //Conexión correcta

    //Creamos la consulta SQL
    $sql = "SELECT idAlumno, alumno, puntuacion FROM $objeto->tabla WHERE puntuacion
>= $objeto->valor";

    $resultado = $conexion ->query($sql);

    $salida = array();
    $salida = $resultado->fetch_all(MYSQLI_ASSOC);

    echo json_encode($salida);
}
$conexion->close();

?>
```

Fetch()

https://lenguajejs.com/javascript/peticiones-http/fetch/
https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Using_Fetch
https://es.javascript.info/fetch

La API Fetch es básicamente un reemplazo moderno para XHR; se introdujo en los navegadores en el año **2015-16** para hacer que las solicitudes HTTP asíncronas sean más fáciles de realizar en JavaScript, tanto para desarrolladores como para otras API que se basan en Fetch.

Está basada en el uso de promesas.

Ejemplos con promesas:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Fetch a file to change this text.</p>
<script>

let file = "datos.txt"

fetch (file)
  .then(x => x.text()) //Devuelve una promesa con el texto plano de la
respuesta que procesamos en el siguiente .then. Podemos utilizar también
x.json() que es lo mismo pero devuelve una promesa con los datos en
JSON, lo que equivale a hacer JSON.parse()
  .then(y => console.log(JSON.parse(y)));
//muestra por consola el objeto

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

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

Para pasar a XML utilizamos un objeto **DOMparser**.

```
fetch('http://example.com/movies.XML')
  .then(response => response.text())
  .then(data => {const parser = new DOMParser();
const xmlDoc = parser.parseFromString(data, 'application/xml');}
```

Actualmente se utiliza fetch con async/await que es una forma teóricamente más entendible que el consumo de promesas con .then
Ejemplo con Async/Await:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Fetch a file to change this text.</p>
<script>
async function obtenerDatos(){
     let url = "fetch_info.txt"
     let x= await fetch(url);
    let y= await x.text();
    document.getElementById("demo").innerHTML = y
}
obtenerDatos();
</script>
</body>
</html>
```

muy recomendable la explicación en:
https://lenguajejs.com/javascript/peticiones-http/fetch/

Envío de datos con fetch al servidor

```
const enviarDatos= (event) => {
    event.preventDefault();

    // Objeto con los datos de un supuesto a enviar
    const formData = {
      nombre: nombre,
      correo: correo,
    };

    // Enviar datos al servidor
    fetch('URL_DEL_SERVIDOR', {
```

```
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(formData),
    })
      .then(response => response.json())
      .then(data => {
        // Manejar la respuesta del servidor si es necesario
        console.log('Respuesta del servidor:', data);
      })
      .catch(error => {
        console.error('Error al enviar los datos:', error);
      });
  };
```

'POST' suele ser el método de envío preferido. Si utilizásemos 'GET' añadiremos a la URL_DEL_SERVIDOR?nombre=${nombre}&correo=${correo}

Acceso a archivo JSON sin fetch

Podemos importar un archivo local (de nuestro propio sitio web) con datos en JSON de la siguiente manera utilizando módulos

```
import datos from "./datos.json" assert {type:'json'};
```

datos pasa a ser un objeto Javascript y ahora podremos acceder a las propiedades que contenga, obviamente también podríamos utilizar un fetch sobre el archivo local.

Política Coors

https://lenguajejs.com/javascript/peticiones-http/cors/

Cross Origin (origen cruzado) es la palabra que se utiliza para denominar el tipo de peticiones que se realizan a un dominio diferente del dominio de origen desde donde se realiza la petición. De esta forma, por una parte tenemos las peticiones de origen cruzado (cross-origin) y las peticiones del mismo origen (same-origin).

¿Qué es CORS?

CORS (Cross-Origin Resource Sharing) es un mecanismo o política de seguridad que permite controlar las peticiones HTTP asíncronas que se

pueden realizar desde un navegador a un servidor con un dominio diferente de la página cargada originalmente. Este tipo de peticiones se llaman peticiones de origen cruzado (cross-origin).

Por defecto, los navegadores permiten enlazar hacia documentos situados en todo tipo de dominios si lo hacemos desde el HTML o desde Javascript utilizando la API DOM (que a su vez está construyendo un HTML).

Sin embargo, no ocurre lo mismo cuando se trata de peticiones HTTP asíncronas mediante Javascript (AJAX), sea a través de XMLHttpRequest, de fetch o de librerías similares para el mismo propósito.

Utilizando este tipo de peticiones asíncronas, los recursos situados en dominios diferentes al de la página actual no están permitidos por defecto. Es lo que se suele denominar protección de CORS. Su finalidad es dificultar la posibilidad de añadir recursos ajenos en un sitio determinado.

Si intentamos realizar una petición asíncrona hacia otro dominio diferente, probablemente obtendremos un error de CORS similar al siguiente:

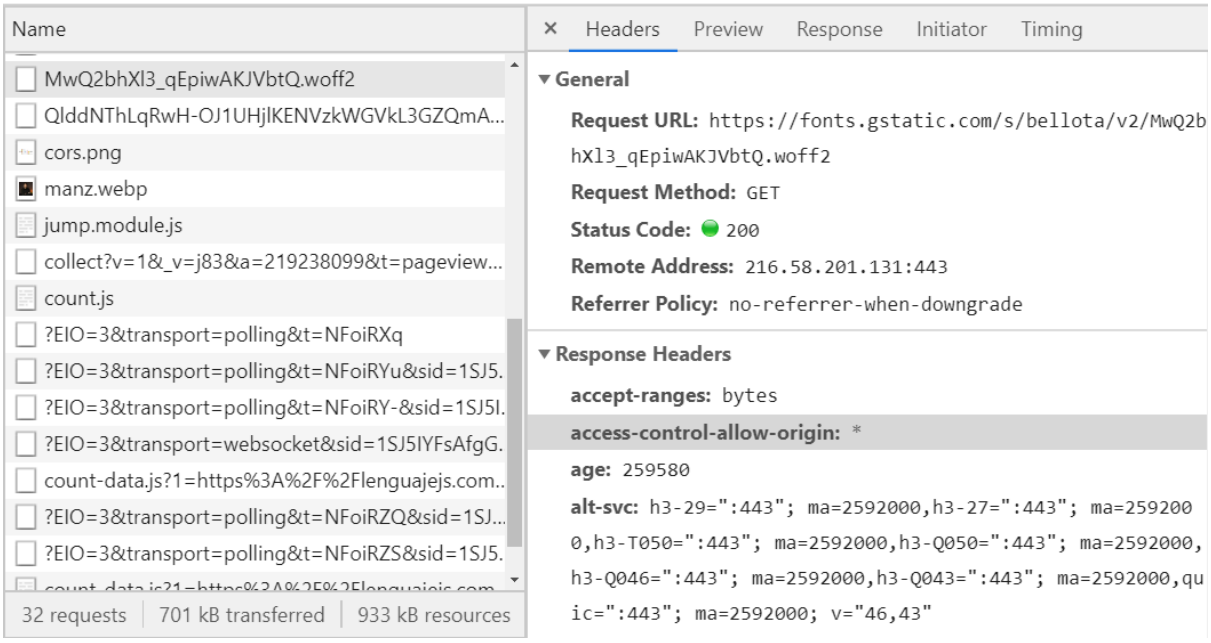
“Access to fetch at 'https://otherdomain.com/file.json' from origin 'https://domain.com/' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.”

Access-Control-Allow-Origin

Como hemos comentado, las peticiones HTTP asíncronas de origen cruzado no están permitidas, pero existen formas de permitirlas. La más básica, probablemente, sea la de incluir una cabecera **Access-Control-Allow-Origin** en la respuesta de la petición (en el servidor), donde debe indicarse el dominio al que se le quiere dar permiso:

Access-Control-Allow-Origin: https://domain.com/

De esta forma, el navegador comprobará dichas cabeceras y si coinciden con el dominio de origen que realizó la petición, esta se permitirá. En el ejemplo anterior, la cabecera tiene el valor https://domain.com/, pero en algunos casos puede interesar indicar el valor *.



El asterisco * indica que se permiten peticiones de origen cruzado a cualquier dominio, algo que puede ser interesante cuando se tienen API públicas a las que quieres permitir el acceso al público en general, casos como los de Google Fonts o JSDelivr, por citar un ejemplo.

Estas cabeceras puedes verlas fácilmente accediendo a la pestaña Network de las herramientas de desarrollador del navegador. En esta sección te aparecerá una lista de peticiones realizadas por el navegador en la página actual. Si seleccionamos una petición y accedemos al apartado de cabeceras (Headers), podremos examinar si existe la cabecera Access-Control-Allow-Origin:

CORS en entornos de desarrollo

Otra opción sencilla y rápida para no tener que lidiar con CORS temporalmente es la de instalar la extensión Allow CORS, disponible tanto Allow CORS para Chrome como Allow CORS para Firefox. Esta extensión deshabilita la política CORS mientras está instalada y activada.

Esta elección es equivalente a que todas las respuestas a las peticiones asíncronas realizadas tengan la mencionada cabecera con el valor *. Obviamente, es importante recalcar que es una opción que sólo nos funcionará en nuestro equipo y navegador, pero puede ser muy práctica para simplificar el trabajo en desarrollo.