# Tema 6: Comunicación asíncrona cliente-servidor

## ¿Qué aprenderás?

- Desarrollar programas básicos en PHP como lenguaje en entorno servidor.
- Utilizar mecanismos de comunicación asíncrona con el servidor.
- Utilizar distintos formatos para el envío y recepción de información asíncrona.
- Utilizar distintas APIs para la comunicación asíncrona.

## ¿Sabías que…?

- En 2005 Jesse James Garret acuñó el término AJAX.
- Aunque la "x" de Ajax se refiere a XML, hoy en día se utiliza JSON.
- MongoDB acepta consultas escritas en JavaScript que combinadas con Node permiten crear un back-end solo en JavaScript.
- Adobe's Acrobat soporta JavaScript en archivos PDF.

# 6. Comunicación asíncrona cliente-servidor

Hasta ahora solo hemos visto dos técnicas para que el usuario pueda hacer una petición al servidor enviándole parámetros:

- Mediante un link HTML
- Mediante un formulario HML

Son dos técnicas que además no hemos visto directamente en éste curso porque son propias de HTML. En ambas podemos codificar parámetros, con la diferencia que si lo hacemos con un formulario es el usuario quien puede indicar el valor de los parámetros.

Pero ambas se caracterizan porque cuando obtienen una respuesta se borra el contenido actual del navegador que es sustituido por la respuesta obtenida.

Éste paradigma conlleva un problema cuando queremos hacer una petición al servidor para información sobre solo una sección de la web, ya que nos obliga a cargar de nuevo toda una nueva página web (con todas las peticiones que conlleva). Esto conlleva un mayor tiempo de carga, mayor gasto de recursos y sobretodo una peor experiencia para el usuario comparado con si solo pudiera cargar una parte pequeña de la web mientras el resto sigue intacto.

AJAX (Asynchronous JavaScript And XML) es una técnica creada para dar solución a los problemas planteados anteriormente permitiendo enviar consultas a un servidor desde JavaScript con las siguientes características:

- Las peticiones se realizan con JavaScript. Por lo tanto podemos programar peticiones que sean completamente distintas según los parámetros que queramos.
- Las peticiones son asíncronas. Cuando se hace una petición no sabemos el tiempo que el servidor va a tardar en contestarnos. Ajax nos asegura que, a no ser que le digamos lo contrario, el resto de código JavaScript va a seguir ejecutándose mientras el servidor no responda. Y en el momento de recibir la respuesta se ejecutará la función que nosotros le definamos para gestionar ésa respuesta.
- Al recibir una respuesta no se recarga toda la web. Cuando recibíamos la respuesta se ejecutará una función que habremos escrito nosotros para extraer los datos de la respuesta y modificar el DOM según nos convenga.
- Aunque AJAX fue pensado para que el servidor retornara un XML con la información retornada por el servidor (por ello la “X” de AJAX), actualmente el lenguaje de estructuración de datos más utilizado en AJAX es JSON.

Las funciones para realizar consultas AJAX las encontramos en dos objetos (APIs) de JavaScript:

- XMLHttpRequest
- Fetch

Pero antes de ver cómo funcionan y hacer nuestras peticiones, vamos a ver como configurar correctamente el IDE para probar AJAX.

## 6.1. Configuración del PHP

Como AJAX es una técnica para enviar peticiones a un servidor, deberemos configurar uno para poder hacer nuestras pruebas. Por su bajo coste y fácil puesta en marcha utilizaremos un servidor PHP .

El método recomendado y más sencillo de instalar PHP en un entorno de desarrollo es instalando algún paquete de desarrollo en entorno servidor que contenga PHP como puede ser XAMPP. Como XAMPP a veces es complejo de instalar en algún sistema operativo hay que dejar claro que de él solo nos interesa el PHP, ya que utilizaremos el *plugin* de VisualStudioCode llamado PHP Server como servidor.

En cualquier caso, cualquier configuración de un servidor PHP en el que podamos ejecutar nuestro código , es completamente válido.

En el siguiente video podemos ver cómo comprovar si ya tenemos PHP, cómo instalar XAMPP y editar la variable de entorno si fuera necesario y como instalar el servidor PHP en Visual Studo Code.


6_1 Configuración de PHP

## 6.2. Introducción a PHP

PHP es el lenguaje de programación en entorno servidor que utilizaremos como ejemplo para ésta documentación. Por ello es importante hacer una pequeña introducción a los conceptos clave antes de ver como enviar peticiones asíncronas al servidor con JavaScript.

PHP es un lenguaje diseñado específicamente para usarse en la web para generar código HTML, lo que establece muchas de sus principales características generales:

1. Cualquier archivo de texto llano y lenguaje de marcas puede ser interpretado como código PHP si cumple dos requisitos:
    - El archivo ha de tener extensión .php
    - El código PHP ha de empezar después de la mara " <?php " . Si queremos cerrar el código PHP para añadir otro tipo de código deberemos cerrar el PHP con la marca " ?>"
2. Un archivo puede contener múltiples bloques de códigos PHP abiertos y cerrados por " <?php " y " ?>". En éstos casos las variables y sentencias se interpretaran como su estuviera todo el código en un mismo bloque PHP.
3. Aunque puede mostrar mensajes por consola, en general un bloque de código PHP escribirá sus respuestas en el archivo y posición donde se haya ejecutado el bloque de código.
4. Ejecutando el PHP en el servidor, no podemos pedir datos al usuario.
5. En web, utilizaremos PHP sobre todo para generar código HTML.
6. Los navegadores interpretan las páginas con extensión ".php" igual que si fueran ".html" porque los navegadores no son capaces de interpretar el código PHP.
7. Es un lenguaje "case-sensitve" por lo que las mayúsculas importan.
8. Cada sentencia PHP ha de terminar en un punto y coma ";" y no le importan los espacios en blanco o saltos de línea.

Una vez vistas las características más generales empezaremos a ver las características mas enfocadas a la programación

### 6.2.1. Definición de variables

Las variables se definen escribiendo la palabra clave "$" seguido del nombre de la variable. Igual que en JavaScript , en PHP no debemos indicar el tipo de variable, que puede variar el en transcurso del programa según el valor que le asignemos.

Podemos escribir un texto en la respuesta pasándolo como parámetro al método global "echo()".

En PHP se pueden utilizar tanto comillas simples como dobles para indicar una string, pero las dobles interpretarán las variables que contengan y las simples no. También podemos concadenar distintos valores con el operador “.”

En el siguiente ejemplo podemos ver como se utiliza el punto para concadenar valores y la diferencia entre comillas simples y dobles.

**Código ejemplo.php**

```php
<?php
$plato ="guisantes";
$cantidad=10;
echo "quiero comer $cantidad $plato";
echo "<br />";
echo 'quiero comer $cantidad $plato';
echo "<br />";
echo 'quiero comer'. $cantidad." ".$plato;
```

**Visualización en el navegador**

quiero comer 10 guisantes
quiero comer $cantidad $plato
quiero comer10guisantes

### 6.2.2. Controles de flujo

En PHP tenemos las instrucciones de control de flujo más comunes en programación.

El bloque IF

```php
<?php
$val1=11; $val2=5;
if($val1>$val2){
   echo "$val1 > $val2";
}else{
   echo "$val1 < $val2";
}
```

Tenemos el bloque FOR

```php
<?php
$suma = 0;
for ($i = 0; $i <= 4; $i++) {
   $suma += $i; // forma abreviada de $suma = $suma + $i;
}
echo $suma; // Imprimirà en pantalla 10
```

El bloque WHILE

```php
<?php
$suma = 0;    $i = 0;
while ($i <= 4) {
   $suma += $i;
   $i++;
}
echo $suma; // Imprimirà en pantalla 10
```

El bloque DO WHILE

```php
<?php
$suma = 0;  $i = 0;
do {
   $suma += $i;
   $i++;
} while ($i <= 4);
echo $suma; // Imprimirà en pantalla 10
```

### 6.2.3. **Funciones**

Declaramos una función con la sintaxis “function(){}”. Entre paréntesis indicamos los parámetros que recibiremos. No hemos de indicar el tipo de dato de los parámetros ni el tipo de dato que retornará la función.

En el siguiente ejemplo definimos una función suma que recibe dos parámetros y retorna el resultado de su suma. Luego llamamos a ésa función y escribimos el resultado en el documento.

**Suma.php**

```php
<?php
function suma($param1, $param2)
{
   $suma = $param1 + $param2;
   return $suma;
}
$resultado = suma(5,6);

echo ("El resultado es: $resultado"); //El resultado es: 11
```

### 6.2.4. Llamar a un PHP con parámetros

Hasta ahora hemos visto ejemplos muy estáticos porque las variables siempre tenían el mismo valor. Pero cuando pedimos al servidor un documento .php podemos pasar parámetros a la petición con distintos métodos. El más básico es escribiendo los parámetros en la misma URL a continuación del nombre del archivo con la siguiente sintaxis:

nombreDocumento.php?parametro1=valor&parametro2=valor&parametro3=valor

Éste método lo llamaremos GET, y es quizás el más básico para pasar parámetros. Muy útil si queremos que el usuario pueda copiar la URL manteniendo los parámetros (por ejemplo copiar una búsqueda en Google).

Hay que aclarar que podemos indicar tantos parámetros como nos sea necesario.

Éstos parámetros son almacenados en nuestro programa en una variable global de tipo array y de nombre $_GET[]. Se almacena el valor según el nombre del parámetro, así que para acceder a un parámetro en concreto utilizaremos la sintaxis:

$valorParametro = $_GET["nombreParametro"];

En el siguiente ejemplo haremos una suma entre dos valores recibidos por parámetro:

**suma.php?num1=5&num2=10**

```php
<?php
$numero1 = $_GET["num1"]; //vale 5
$numero2 = $_GET["num2"]; //vale 10

$resul = $numero1+$numero2;
echo ("El resultado es:".$resul); //en éste caso 15
```

Como hacer que el usuario modifique la URL para enviar valores a un PHP no és muy usable, normalmente se suelen usar los formularios, que permiten generar una petición con parámetros a un archivo PHP pero en donde los valores de los parámetros los puede escribir el usuario.

Los *name* de los inputs se convierten en el nombre de los parámetros y su valor es el escrito por los usuarios.

En el siguiente ejemplo tenemos un formulario que se va a enviar por GET a "suma.php" con los parámetros "num1" y "num2" pero los valores escritos por el usuario. Si el usuario escribiera los valores 3 y 6 se llamaría a suma.php?num1=3num2=6:

**formulario.html**

```
<form action="suma.php" method="get">
  Numero 1:
<input type="numer" name="num1" />
  Numero 2:
<input type="numer" name="num2" />
  <input type="submit">
</form>
```

si el usuario escribe un 3 y un 6 en los *inputs*

**suma.php?num1=3&num2=6**

```
<?php
$numero1 = $_GET["num1"];//3
$numero2 = $_GET["num2"];//6

$resul = $numero1+$numero2;
echo ("El resultado es:".$resul); //9
```

A parte de GET podemos enviar parámetros con otros métodos. El más común es el método POST que oculta los parámetros de la URL. Éstos parámetros son almacenados en nuestro programa en una variable global de tipo array y de nombre $_POST[ ].

Para programar el ejemplo anterior con POST deberíamos cambiar el atributo "method" del formulario y la forma de acceder a los parámetros del PHP. Nos quedaría de la siguiente manera:

**formulario.html**

```
<form action="suma.php" method="post">
  Numero 1:
<input type="numer" name="num1" />
  Numero 2:
<input type="numer" name="num2" />
  <input type="submit">
</form>
```

si el usuario escribe un 3 y un 6 en los *inputs*

**suma.php**

```
<?php
$numero1 = $_POST["num1"];//3
$numero2 = $_POST["num2"];//6

$resul = $numero1+$numero2;
echo ("El resultado es:".$resul); //9
```

Por último, podemos saber en el PHP si un parámetro existe o no con el método "isset()" y pasándole por parámetro el $_GET[] o $_POST[] del parámetro que queramos comprovar. Si no existe y lo intentamos obtener, PHP nos generará un error.

Si en el siguiente ejemplo el usuario no escribe los dos valores de los *input*, entonces se mostrará el texto "faltan parámetros"

**formulario.html**

```
<form action="suma.php" method="post">
  Numero 1:
<input type="numer" name="num1" />
  Numero 2:
<input type="numer" name="num2" />
  <input type="submit">
</form>
```

**suma.php**

```
<?php
if(isset($_GET["num1"]) && isset( $_GET["num2"])){
  $numero1 = $_GET["num1"];
  $numero2 = $_GET["num2"];

  $resul = $numero1+$numero2;
  echo ("El resultado es:".$resul);
}else{
  echo "faltan parametros";
}
```

### 6.2.5. Sesiones

Las sesiones en PHP es lo que nos van a permitir que el servidor reconozca que dos peticiones son realizadas por el mismo usuario (efectivamente suelen usar cookies).

Por ejemplo, si queremos un contador que cuente cuantas veces un mismo usuario ha visitado nuestro PHP, con lo que hemos visto hasta ahora no sería suficiente. Pues si programásemos algo parecido al código que se muestra a continuación, siempre retornaría el valor de 1.

**contador.php**

```php
<?php
$contador=0;
$contador++;
echo "Es tu visita numero".$contador;
```

Como las variables de sesión nos permiten almacenar información entre dos peticiones realizadas por el mismo usuario, utilizándolas vamos a poder contar el número de accesos de un mismo usuario.

Siempre que queramos utilizar variables de sesión debemos escribir en la primera línea del documento la sentencia PHP : “session_start();” .

Una vez inicializada la sesión , podemos acceder a las variables de sesión para modificar o leer su valor con la sintaxis:

$_SESSION["nombre_variable_session"]

Podemos saber si una variable de sesión ya existía con el método “isset()” visto anteriormente con los parámetros.

**contador.php**

```php
<?php
session_start();
//si no exite la variable de sesión "cuenta", $contador será igual 0
if(!isset($_SESSION["cuenta"])){
 $contador=0;
}else{
//si existe la variable de session "cuenta", $contador tendrá su valor
   $contador= $_SESSION["cuenta"];
}
//Actualizamos el contador
$contador++;
//y almacenamos su valor en la variable de session
$_SESSION["cuenta"]=$contador;

echo "Es tu visita numero".$contador;
```

### 6.2.6. Arrays y JSON

En PHP podemos crear *arrays* asociativas o numéricas y fácilmente obtener su nomenclatura JSON y viceversa, pasar de JSON a un *array*.

Para crear un *array* con posiciones numéricas en PHP utilizaremos el constructor array() pasando como parámetro los distintos elementos que queremos que tenga el array. Por ejemplo:

*$miArray = array("valor posición 0", "valor posición 1", ect…);*

Para obtener su representación en JSON utilizaremos la función *json_encode()* pasando como parámetro la variable que tiene el array. Por ejemplo:

*$miArrayJSON = json_encode($miArray);*

Si queremos crear un array asociativo utilizaremos igualmente el constructor *array*() pero entre paréntesis escribiremos la clave y el valor con la sintaxis : *clave=>valor*. Por ejemplo:

*$miArrayAssoc=array("dni"=>"3382839X","edad"=>33);*

Si tenemos una string con una estructura JSON la podemos pasar a array con la función *json_encode(stringJSON)*. Por ejemplo:

*$miArrayAssoc=json_encode({"dni":"3382839X","edad":33});*

A continuación podemos ver cómo realizar las tres operaciones anteriores:

```php
<?php
$nombres= array("Merce","Josefa","Mortadelo","Filemon");//definimos el array
$nombresJSON = json_encode($nombres); //lo pasamos a JSON
//imprimimos el JSON
echo($nombresJSON);  //["Merce","Josefa","Mortadelo","Filemon"]

//definimos array assoc
$pastel=array("nombre"=>"tiramisu", "kcal"=>200,"sabor"=>"rico rico");
$pastelJSON = json_encode($pastel); //lo pasamos a JSON
//lo imprimimos
echo ($pastelJSON);  //{"nombre":"tiramisu","kcal":200,"sabor":"rico rico"}

$jsonReloj='{"marca":"horas","mecanica":"japonesa","precio":"60"}'; //definimos el JSON
$arrayReloj=json_encode($jsonReloj); //generamos el array
//imprimios el objeto
var_dump($arrayReloj); //string(67) ""{\"marca\":\"horas\",\"mecanica\":\"japonesa\",\"precio\":
\"60\"}""
```

Para imprimir el contenido de un array en PHP, si lo intentamos hacer con echo nos dará un error, debemos utilizar la función *var_dump()*. También se puede utilizar para imprimir cualquier objeto en PHP.

## 6.3. API XMLHttpRequest

XMLHttpRequest es la primera API que ofreció un conjunto de métodos para realizar operaciones AJAX. Para poder utilizarla basta con crear un objeto con el constructor XMLHttpRequest .Cada uno de éstos objetos representarán una petición AJAX, y será a través de ellos que configuraremos, enviaremos y recibiremos dicha petición.

Para configurar una petición Ajax con XMLHttpRequest hay que seguir los siguientes pasos:

9. Declarar el objeto XMLHttpRequest. Por ejemplo:

   *let xmlHttp = new XMLHttpRequest();*

10. Configurar la petición con su método *open()* y sus tres parámetros:
    - El método de la petición (GET, POST, etc..)
    - La URL de la petición con sus parámetros GET
    - Un valor opcional *true* o *false* para indicar si queremos que sea asíncrona o no (por defecto y recomendado, *true*).

    Por ejemplo:

    *xmlHttp.open("GET","respuesta.php?edad=51",true);*

11. Establecer con el método *setRequestHEader()* el HEADER de la petición si es necesario . Por ejemplo:

    *xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");*

12. Configurar la función que se va a encargar de controlar cada cambio de estado en la petición. Para comprender la importancia de éste función, debemos saber que cualquier objeto XMLHttpRequest tiene una propiedad *readyState* que nos indica con un valor numérico del 0 al 4 el estado de la comunicación. Su significado es:
    - 0: aún no se ha enviado la petición
    - 1: se ha enviado la petición
    - 2: se han recibido las cabeceras de la respuesta
    - 3: se está cargando parte del contenido de la respuesta
    - 4: se ha terminado de cargar toda la respuesta.

    Con el evento *onreadystatechange* podemos asignar una función encargada de gestionar cada cambio que se produzca en la propiedad *readyState*. Por ejemplo:

```
xmlHttp.onreadystatechange = function(){
        if(xmlHttp.readyState ==4){
                console.log("respuesta recibida!");
        }
}
```

Pero que hayamos obtenido una respuesta del servidor, no implica que se haya encontrado el recurso pedido. Por ello además del estado de la comunicación debemos conocer el estado de la respuesta. Eso lo podremos conocer con la propiedad *status* de cualquier objeto XMLHttpRequest, y en general nos retornará dos posibles valores:

- 200 : indica que se ha encontrado y obtenido el recurso pedido.
- 404 : indica que la URL del recurso no se ha encontrado.

Así que cuando comprobemos si hemos recibido una respuesta del servidor también deberemos comprobar si era la respuesta que habíamos pedido tal y como se muestra en el siguiente ejemplo:

```
xmlHttp.onreadystatechange = function(){
    if(xmlHttp.readyState ==4){
        console.log("respuesta recibida!");
        if(xmlHttp.status==200){
            console.log("URL de la petición correcta");
            console.log("ahora gestionaremos lo respondido!");
        }
    }
}
```

13. Si ya hemos configurado la petición y controlado sus estados, nos faltará ser capaces de gestionar la respuesta. Dependiendo del formato en el que se haya enviado la respuesta utilizaremos una técnica u otra para gestionarla. Vamos a ver 3 posibles formatos de respuesta:

    - el formato más básico es el texto llano. Los objetos XMLHttpRequest tienen una propiedad llamada *responseText* que contiene el texto que ha retornado el servidor.

      ```
      let textoRespuesta = xmlHttp.responseText;
      ```

    - El formato más utilizado es JSON. Extrayendo primero el texto llano podremos transformar ése texto llano a un objeto JSON con el método *parse* del objeto JSON (como ya vimos en estructuras de datos).

      ```
      let textoRespuesta = xmlHttp.responseText;
      let objetoJSON = JSON.parse(textoRespuesta);
      ```

    - El formato original es XML. Por ello todos los objetos XMLHttpRequest tienen una propiedad *responseXML* que retorna un “document” con su propio DOM y que puede ser por lo tanto recorrido con los métodos del DOM excepto algunos como “getElementById()”.

      ```
      let domXML = xmlHttp.responseXML;
      ```

14. Por último solo queda enviar la petición con el método *send()*. Si la petición AJAX se ha configurado con POST, éste método puede recibir el listado de parámetros a enviar al servidor.

```
xmlHttp.send();
```

A continuación veremos un ejemplo en el que se envían los valores escritos en dos inputs a un servidor. El servidor los sumará y retornará la respuesta que mostraremos dentro del DIV con id “respuesta”. Primero veremos éste caso utilizando una petición tipo GET con una respuesta JSON y luego veremos el mismo caso pero variaremos la petición por POST y la respuesta será en formato XML.

Al enviar la petición por GET los parámetros se añaden en la misma URL y la función “send()” no recibe parámetro.

**Documento HTML**

```
<head>
  <title>Ejemplo AJAX</title>
<script src="cliente.js" defer></script>
</head>
<body>
  <input type="number" id="num1" />
  <input type="number" id="num2" />
  <button onclick="sendAJAX()">
    SEND AJAX
  </button>
  <div id="respuesta"></div>
</body>
```

**Documento respuest.php**

```
<?php
$numero1 = $_GET["n1"];
$numero2 = $_GET["n2"];

$resp=$numero1+$numero2;
echo '{"resp":"'.$resp.'"}';
```

**Documento cliente.js**

```
function sendAJAX() {
  //obtenemos los valores a enviar al servidor
  let num1 = document.getElementById("num1").value;
  let num2 = document.getElementById("num2").value;

  //Declarar el objeto XMLHttpRequest
  let xmlHttp = new XMLHttpRequest();
  //Configurar  la petición pasando los parámetros que queramos
  xmlHttp.open("GET", "respuesta.php?n1=" + num1 + "&n2=" + num2, true);
  //Establecer el HEADER de la petición
  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

  //asignar una función encargada de gestionar cada cambio
  xmlHttp.onreadystatechange = function () {
    if (xmlHttp.readyState == 4) {
      if (xmlHttp.status == 200) {
        //obtenemos la respuesta y la parseamos a JSON
        let textoRespuesta = xmlHttp.responseText;
        let objetoJSON = JSON.parse(textoRespuesta);
        let respuesta = objetoJSON.resp;
        //mostramos la respuesta en el HTML
        document.getElementById("respuesta").innerHTML = respuesta;
      }
    }
  }
    xmlHttp.send();  //enviar la petición
}
```

En el siguiente ejemplo se envían los valores de los inputs por POST añadiéndolos como parámetros en la función “send()”. El servidor retorna el resultado de su suma en XML que finalmente se muestra en el HTML.

**Documento HTML**

```
<head>
   <title>Ejemplo AJAX</title>
<script src="cliente.js" defer></script>
</head>
<body>
   <input type="number" id="num1" />
   <input type="number" id="num2" />
   <button onclick="sendAJAX()">
      SEND AJAX
   </button>
   <div id="respuesta"></div>
</body>
```

**Documento respuest.php**

```
<?php
$numero1 = $_POST["n1"];
$numero2 = $_POST["n2"];
$resul=$numero1+$numero2;
//Tipo de archivo a retornar:
@header("Content-type: text/xml");
$xml='<?xml version="1.0" encoding="utf-8"?>';
$xml.='<respuesta>';
$xml.='<resul>'.$resul.'</resul>';
$xml.='</respuesta>';
echo $xml;
```

**Documento cliente.js**

```
function sendAJAX() {
   //obtenemos los valores a enviar al servidor
   let num1 = document.getElementById("num1").value;
   let num2 = document.getElementById("num2").value;

   //Declarar el objeto XMLHttpRequest
   let xmlHttp = new XMLHttpRequest();
   xmlHttp.open("POST", "respuesta.php", true);
   //Establecer el HEADER de la petición
   xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

   //asignar una función encargada de gestionar cada cambio
   xmlHttp.onreadystatechange = function () {
      if (xmlHttp.readyState == 4) {
         console.log("respuesta recibida!");
         if (xmlHttp.status == 200) {
            //obtenemos la respuesta y extraemos el XML
            let respuestaXML = xmlHttp.responseXML;
            //extraemos la respuesta moviéndonos por el DOM
            let resultado = respuestaXML.firstElementChild.firstElementChild.innerHTML;
           //mostramos la respuesta en el HTML
            document.getElementById("respuesta")
               .innerHTML = resultado;
         }
      }
   }
   xmlHttp.send("n1=" + num1 + "&n2=" + num2);    //enviar la petición con parámetros
}
```

## 6.4. API Fetch

La API Fetch es una nueva API creada con la intención de simplificar las peticiones AJAX. La API Fetch añade el método "fetch()" como un método global a través del cual configuraremos cada petición que queramos realizar.

Si con *XMLHttpRequest* el asincronismo lo conseguíamos con el evento "onreadystatechange", con el método "fetch()" el asincronismo lo conseguimos porque retrnará un objeto del tipo *Promise*.

Los objeto de tipo Promise tienen un método "then()" que recibe como parámetro una función que se ejecutarán cuando se lance algún tipo de evento.

En el caso de "fetch()"su método "then()" se ejecuta cuando el servidor retorna algún valor.

Para configurar una petición Ajax con "fetch()" hay que seguir los siguientes pasos:

1. Configurar la petición creando un objeto "fetch()" con tres parámetros:
   - method: el método de la petición (GET, POST, etc..)
   - body: los parámetros de la petición si se envía por POST
   - headers: las cabeceras de la petición

   Por ejemplo:

```
let configFetch = {
      method: "POST",
     body: "n1="+num1+"&n2="+num2,
      headers:{'Content-Type':'application/x-www-form-urlencoded'}
   };
```

2. lanzar la petición Ajax con el método global *fletch()* pasando como parámetro la URL de la petición y el objeto JSON con la configuración. Éste método generará un objeto del tipo "Promise" a través del cual gestionaremos la respuesta. Por ejemplo:

   let promesa = fetch("respuesta.php",config);

3. Configurar la función que se va a encargar de recibir la respuesta. El objeto tipo "Promise" que retorna *fetch()* tiene definido un método "then()" que ejecutará la función pasada por parámetro cuando el servidor responda. La función que se ejecutará recibirá por parámetro un objeto del tipo "Response" con información de la petición. Por ejemplo:

```
promesa.then( function(respuesta){
        console.log("respuesta recibida! );
        console.log("estado:"+respuesta.status); //200 ok!
});
```

Pero también debemos de ser capaces de controlar si se ha produce un error en la red. Para ello utilizaremos el método *catch* después de *then. Por ejemplo:*

```
promesa.then( function(respuesta){
        //respuesta recibida!
        console.log("estado:"+respuesta.status); //200 ok!
}) .catch(function (error) {
        console.log('Error con la petición:' + error.message);
});
```

4. Para gestionarla respuesta utilizaremos una técnica u otra para gestionarla dependiendo del formato en el que se haya enviado la respuesta. A diferencia de XMLHttpRequest fetch no tiene ningún método para parsear la respuesta a XML y sí tiene para parsear a JSON. Por ello son centraremos solo en respuestas de tipo texto y JSON en fetch. Vamos a ver los 2 posibles formatos de respuesta:
   - el formato más básico es el texto llano. Los objetos "Response" tienen un método llamado "*.text()*" que retorna otro objeto "Promise" que en su "then()" retorna el texto leído. Por ejemplo:

```
promesa.then( function(respuesta){
        console.log("estado:"+respuesta.status); //200 ok!
        response.json().then(
                function (respuestaJSON) {
                        console.log(respuestaJSON);
                }
        );
}) .catch(function (error) {
        console.log('Error con la petición:' + error.message);
});
```

   - JSON es el formato más utilizado. De forma parecida a las respuestas en formato texto , lo objetos "Response" tienen un método llamado Extrayendo primero el texto llano podremos transformar ése texto llano a un objeto JSON con el método *parse* del objeto JSON (como ya vimos en estructuras de datos).

```
let textoRespuesta = xmlHttp.responseText;
let objetoJSON = JSON.parse(textoRespuesta);
```

- XML no está contemplado en “fetch”. Aunque hay librerías externas para hacerlo, no las veremos aquí.

A continuación veremos un ejemplo en el que se envían por GET los valores escritos en dos inputs a un servidor. El servidor los sumará y retornará un JSON con la respuesta que mostraremos dentro del DIV con id “respuesta”.

**Documento HTML**

```
<head>
  <title>Ejemplo AJAX</title>
<script src="cliente.js" defer></script>
</head>
<body>
  <input type="number" id="num1" />
  <input type="number" id="num2" />
  <button onclick="sendAJAX()">
    SEND AJAX
  </button>
  <div id="respuesta"></div>
</body>
```

**Documento respuest.php**

```
<?php
$numero1 = $_GET["n1"];
$numero2 = $_GET["n2"];

$resp=$numero1+$numero2;
echo '{"resp":"'.$resp.'"}';
```

**Documento cliente.js**

```
function sendAJAX() {
  //obtenemos los valores a enviar al servidor
  let num1 = document.getElementById("num1").value;
  let num2 = document.getElementById("num2").value;
  //fetch retorna un obeto del tipo Promise.

  let configFetch = {
    method: "GET",
   // no tiene body pal ser del tipo GET
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' }
  };
  let promesa = fetch("respuesta.php?"+"n1=" + num1 + "&n2=" + num2, configFetch);
  promesa.then(function (response) {
    //la propiedad ok retorna true si se ha realizado correctamente
    if (response.ok) {
      console.log("REspuesta OK");
    }
    response.json().then(
      function (objetoJSON) {
        let respuesta = objetoJSON.resp;

        //mostramos la respuesta enel HTML
        document.getElementById("respuesta").innerHTML = respuesta;
      });
  }).catch(function (error) {
    console.log('Error con la petición:' + error.message);
  });
}
```

A continuación veremos un ejemplo en el que se envían por POST los valores escritos en dos inputs a un servidor. El servidor los sumará y retornará un JSON con la respuesta que mostraremos dentro del DIV con id “respuesta”.

**Documento HTML**

```
<head>
  <title>Ejemplo AJAX</title>
<script src="cliente.js" defer></script>
</head>
<body>
  <input type="number" id="num1" />
  <input type="number" id="num2" />
  <button onclick="sendAJAX()">
    SEND AJAX
  </button>
  <div id="respuesta"></div>
</body>
```

**Documento respuest.php**

```
<?php
$numero1 = $_POST["n1"];
$numero2 = $_POST["n2"];


$resp=$numero1+$numero2;
echo '{"resp":"'.$resp.'"}';
```

**Documento cliente.js**

```
function sendAJAX() {
  //obtenemos los valores a enviar al servidor
  let num1 = document.getElementById("num1").value;
  let num2 = document.getElementById("num2").value;
  //fetch retorna un obeto del tipo Promise.
  let configFetch = {
    method: "POST",
    body: "n1=" + num1 + "&n2=" + num2,
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' }
  };
  let promesa = fetch("respuesta.php", configFetch);
  promesa.then(function (response) {
    //la propiedad ok retorna true si se ha realizado correctamente
    if (response.ok) {
      console.log("REspuesta OK");
    }
    response.json().then(
      function (objetoJSON) {
        let respuesta = objetoJSON.resp;

        //mostramos la respuesta enel HTML
        document.getElementById("respuesta").innerHTML = respuesta;
      });
  }).catch(function (error) {
    console.log('Error con la petición:' + error.message);
  });
}
```

## 6.5. Depurar código AJAX

Como AJAX es una tecnología en la que interviene el cliente y el servidor, su depuración es un poco más compleja. Ahora no solo vamos a poder tener errores en el mismo código AJAX sino que los errores se van a poder producir también en la misma comunicación , en el servidor o en su respuesta.

Por todo ello para depurar los programas desarrollados con AJAX es esencial utilizar el inspector de red en las herramientas de desarrollo del navegador.

En el siguiente video podemos ver con ejemplos de comunicaciones AJAX cómo utilizar el inspector de red y depurar algunos de los errores que nos van a surgir.


6_5 Depuración AJAX

# Test de autoevaluación

¿Qué es AJAX?

a) Un lenguaje de programación en entorno cliente
b) Un lenguaje de programación en entorno servidor.
c) Un lenguaje de programación que engloba el entorno cliente y el entorno servidor.
d) Una tecnología para enviar datos al servidor desde JS

¿Actualmente qué lenguaje de estructuración de datos es el más utilizado para enviar datos con AJAX?

a) JSON
b) XML
c) HTML
d) PHP

¿Qué tipo de objeto mantiene el estado de una comunicación AJAX?

a) AJAX
b) Document
c) XMLHttpRequest
d) AjaxRequest

## Recursos y enlaces

- Página de instalación de PHP

- Página de XAMPP

- Documentación en W3Schools sobre XMLHttpRequest

- Documentación en MDN sobre Fetch

- Documentación en MDN sobre el objeto JavaScript  Response

## Conceptos clave

- **Comunicación Asíncrona**: se produce cuando después de que el emisor envíe una petición al receptor, el emisor puede seguir trabajando sin tener que quedarse bloqueado esperando la respuesta del receptor.
- **XMLHttpRequest**: la primera API creada para realizar consultas AJAX con JS.
- **FETCH:** la nueva API creada para facilitar las consultas AJAX.

# Ponlo en práctica

## Actividad 1

Crea una aplicación web en donde el usuario envíe un número por AJAX a un servidor mediante XMLHttpRequest y POST. El servidor generará un número aleatorio entre 1 y 100 y le retornará en formato XML el resultado de sumar el número aleatorio más el enviado por el usuario. Al recibir la respuesta se mostrará el resultado en el HTML.

**Los solucionarios están disponibles en el aula virtual.**

## Actividad 2

Una aplicación web en donde el usuario pida un listado de clientes y el servidor retorne un array en formato JSON con el listado de nombres de clientes.
Finalmente se han de mostrar todos los clientes en una lista desordenada

**Los solucionarios están disponibles en el aula virtual.**

## Actividad 3

Programa utilizado el API Fetch una aplicación web en donde el usuario pueda indicar el número el cliente del que quiere obtener información según el array de clientes que tenga el servidor.
El servidor retorna el JSON con el nombre del cliente que ha pedido el usuario y se muestra en el HTML.

**Los solucionarios están disponibles en el aula virtual.**

## Actividad 4

Programa utilizado el API Fetch una aplicación web con dos botones y un input. Programa el primer botón para que al clicar pida al servidor que genere un número aleatorio entre 1 y 100 y lo almacene en una variable de sesión. Programa el segundo botón para que al clicar se envíe al servidor el número que el usuario haya introducido dentro del input. El servidor ha de comparar el valor introducido dentro del input con el almacenado en la variable de sesión y indicar si son iguales, o el valor enviado es inferior o superior.

Puedes utilizar un PHP para generar el número y otro para comprobar el número. Recuerda guardar el número en una variable de sesión.
Utiliza el API Fetch con GET para pedir que genere el número y el API Fetch con POST para comprobar el número.

**Los solucionarios están disponibles en el aula virtual.**

# SOLUCIONARIOS

## Test de autoevaluación tema 6

¿Qué es AJAX?

e) Un lenguaje de programación en entorno cliente

f) Un lenguaje de programación en entorno servidor.

g) Un lenguaje de programación que engloba el entorno cliente y el entorno servidor.

**h) Una tecnología para enviar datos al servidor desde JS**

¿Actualmente qué lenguaje de estructuración de datos es el más utilizado para enviar datos con AJAX?

**e) JSON**

f) XML

g) HTML

h) PHP

¿Qué tipo de objeto mantiene el estado de una comunicación AJAX?

e) AJAX

f) Document

**g) XMLHttpRequest**

h) AjaxRequest

# Ponlo en práctica tema 6

## Actividad 1

Crea una aplicación web en donde el usuario envíe un número por AJAX a un servidor mediante XMLHttpRequest y POST. El servidor generará un número aleatorio entre 1 y 100 y le retornará en formato XML el resultado de sumar el número aleatorio más el enviado por el usuario. Al recibir la respuesta se mostrará el resultado en el HTML.

**Los solucionarios están disponibles en el aula virtual.**

## Actividad 2

Una aplicación web en donde el usuario pida un listado de clientes y el servidor retorne un array en formato JSON con el listado de nombres de clientes.
Finalmente se han de mostrar todos los clientes en una lista desordenada

**Los solucionarios están disponibles en el aula virtual.**

## Actividad 3

Programa utilizado el API Fetch una aplicación web en donde el usuario pueda indicar el número el cliente del que quiere obtener información según el array de clientes que tenga el servidor.
El servidor retorna el JSON con el nombre del cliente que ha pedido el usuario y se muestra en el HTML.

**Los solucionarios están disponibles en el aula virtual.**

## Actividad 4

Programa utilizado el API Fetch una aplicación web con dos botones y un input. Programa el primer botón para que al clicar pida al servidor que genere un número aleatorio entre 1 y 100 y lo almacene en una variable de sesión. Programa el segundo botón para que al clicar se envíe al servidor el número que el usuario haya introducido dentro del input. El servidor ha de comparar el valor introducido dentro del input con el almacenado en la variable de sesión y indicar si son iguales, o el valor enviado es inferior o superior.

Puedes utilizar un PHP para generar el número y otro para comprobar el número. Recuerda guardar el número en una variable de sesión.
Utiliza el API Fetch con GET para pedir que genere el número y el API Fetch con POST para comprobar el número.

**Los solucionarios están disponibles en el aula virtual.**