C.F.G.S.: DESARROLLO DE APLICACIONES WEB
Módulo: DESARROLLO WEB EN ENTORNO CLIENTE

01 AJAX

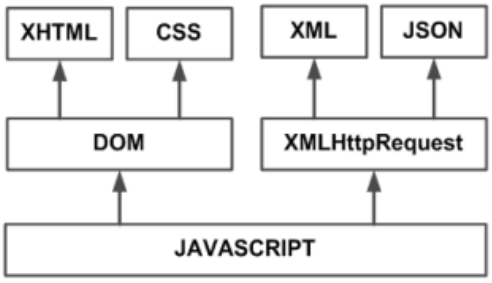
¿QUÉ ES AJAX?

AJAX son las siglas de **A**synchronous **J**avaScript **A**nd **X**ML.

No es un lenguaje de programación sino una **técnica de desarrollo web** para crear aplicaciones interactivas mediante un conjunto de tecnologías que nos permiten realizar cambios sobre las páginas sin necesidad de recargarlas completamente, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

Las tecnologías que forman AJAX son:

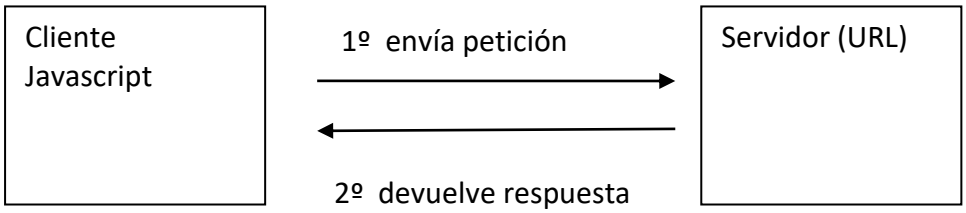
- XHTML y CSS, para crear una presentación basada en estándares.
- DOM, para la interacción y manipulación dinámica de la presentación.
- XML, XSLT y JSON, para el intercambio y la manipulación de información.
- XMLHttpRequest, para el intercambio asíncrono de información.
- JavaScript, para unir todas las demás tecnologías.



Su característica fundamental es que permite actualizar parte de una página con información que se encuentra en el servidor sin tener que refrescar completamente la página. De modo similar podemos enviar información al servidor.

Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma Ajax es una tecnología **asíncrona**, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página.

Las peticiones Ajax son ejecutadas por el código JavaScript, se envía una petición a una URL(al servidor), cuando se recibe una respuesta, se ejecuta una función (de devolución) que recibe como argumento la respuesta del servidor y realiza algo con ella. Debido a que la respuesta es **asíncrona**, el resto del código de la aplicación continúa ejecutándose.



3º Javascript ejecuta una función (de devolución) a la que pasa como argumento la respuesta del servidor

Aunque la definición de Ajax posee la palabra "XML", la mayoría de las aplicaciones no utilizan dicho formato para el transporte de datos, sino que en su lugar se utiliza HTML plano o información en formato JSON (*JavaScript Object Notation*).

Casos prácticos y ejemplos reales

En el mundo real, **usar AJAX en un programa o aplicación se ha convertido en esencial** para numerosas aplicaciones y proyectos web, destacando por su capacidad para mejorar la interactividad y la eficiencia, y la mejor muestra de ellos la encontramos en dos servicios de Google que se encuentran entre los más utilizados.

- **Gmail**: Este popular servicio de correo electrónico de Google es un ejemplo clásico del uso de AJAX. Gmail utiliza AJAX para actualizar dinámicamente la interfaz de usuario, **permitiendo a los usuarios leer, escribir, buscar y organizar correos electrónicos sin la necesidad de recargar** la página completa. Esta implementación de AJAX mejora significativamente la experiencia del usuario, ofreciendo una aplicación web que se muestra más como un programa de escritorio en términos de respuesta y eficiencia.
- **Google Maps**: Otra aplicación de Google que hace un uso extensivo de AJAX es Google Maps. **AJAX permite a los usuarios interactuar con el mapa de manera fluida, cargando y mostrando información geográfica sin recargar la página**. Esto incluye el desplazamiento por el mapa, el zoom y la visualización de detalles de lugares específicos. La capacidad de AJAX para actualizar solo partes de la página en respuesta a las acciones del usuario hace que Google Maps sea increíblemente rápido y receptivo, proporcionando una experiencia de usuario altamente interactiva y eficiente.

Estos ejemplos demuestran cómo AJAX puede ser utilizado para crear aplicaciones web ricas y dinámicas, mejorando la experiencia del usuario al proporcionar respuestas rápidas y minimizar los tiempos de carga.

Objeto XMLHttpRequest

El corazón de Ajax es el objeto XMLHttpRequest que nos permite realizar una conexión al servidor y enviarle una petición y recibir la respuesta que procesaremos en nuestro código Javascript.

En los navegadores actuales, un objeto XMLHttpRequest se crea simplemente así:

```
Peticion_http = new XMLHttpRequest();
```

El objeto **XMLHttpRequest** se utiliza para intercambiar datos con un servidor

Métodos del objeto XMLHttpRequest

Para **enviar** una solicitud se utilizan los métodos open() y send() del objeto XMLHttpRequest

open (method, url, async)
- Establece los parámetros de la petición que se realiza al servidor
- method: tipo de conexión GET o POST
- url: url destino, (puede indicarse de forma relativa o absoluta) localización del fichero en el servidor
- async: true (asíncrono, por defecto) o false (síncrono)

send (string)
- Realiza la petición HTTP al servidor
- En modo GET: send()
- En modo POST: send (cadena con datos que enviamos al servidor)

Además existen otros métodos

abort()	Detiene la petición actual
getAllResponseHeaders()	Devuelve una cadena de texto con todas las cabeceras de la respuesta del servidor
getResponseHeader("cabecera")	Devuelve una cadena de texto con el contenido de la cabecera solicitada

Propiedades del objeto XMLHttpRequest

onload

propiedad para indicar la función a ejecutar tras recibir la respuesta del servidor

onreadystatechange

propiedad para indicar la función a ejecutar cada vez que se produce un cambio en el estado de la petición HTTP. Normalmente es una referencia a una función JavaScript

readyState

Estado de la conexión, puede valer desde 0 (no iniciada) hasta 4 (completado).

`0`	No inicializado (objeto creado, pero no se ha invocado el método `open`)
`1`	Cargando (objeto creado, pero no se ha invocado el método `send`)
`2`	Cargado (se ha invocado el método `send`, pero el servidor aún no ha respondido)
`3`	Interactivo (se han recibido algunos datos, aunque no se puede emplear la propiedad `responseText`)
`4`	Completo (se han recibido todos los datos de la respuesta del servidor)

status

Código enviado por el servidor, del tipo 404 (documento no encontrado) o 200 (OK).

statusText

Texto asociado al código del status (p.ej. “OK”, “Not Found”)

responseText

datos devueltos por el servidor en formato string

responseXML

Datos devueltos por el servidor en forma de documento XML que puede ser recorrido mediante las funciones del DOM (getElementsByTagName, etc).

Ver Ejemplo 1 Carga txt

Descarga un archivo del servidor y muestra su contenido sin necesidad de recargar la página.

```
function descargaArchivo() {
// Obtener la instancia del objeto XMLHttpRequest
        peticion_http = new XMLHttpRequest();
// Preparar la funcion de respuesta
        peticion_http.onreadystatechange = muestraContenido;
// Realizar peticion HTTP
        peticion_http.open('GET', 'http://localhost//EJ AJAX 2024//saludo.txt', true);
        peticion_http.send(null);
        function muestraContenido() {
                if(peticion_http.readyState == 4) {
                        if(peticion_http.status == 200) {
                document.getElementById("fichero").innerHTML=peticion_http.responseText;
                                        }
                                }
                        }
                }
```

Se siguen 4 pasos:

instanciar el objeto XMLHttpRequest
preparar la función de respuesta
realizar la petición al servidor
ejecutar la función de respuesta.

Todas las aplicaciones realizadas con técnicas de AJAX deben instanciar en primer lugar el objeto XMLHttpRequest, que es el objeto clave que permite realizar comunicaciones con el servidor en segundo plano, sin necesidad de recargar las páginas.

Una vez obtenida la instancia del objeto XMLHttpRequest, se prepara la función que se encarga de procesar la respuesta del servidor. La propiedad onreadystatechange del objeto XMLHttpRequest permite indicar esta función directamente incluyendo su código mediante una función anónima o indicando una referencia a una función independiente. En el ejemplo anterior se indica directamente el nombre de la función:

peticion_http.onreadystatechange = muestraContenido;

El código anterior indica que cuando la aplicación reciba la respuesta del servidor, se debe ejecutar la función muestraContenido(). La referencia a la función se indica mediante su nombre sin paréntesis, ya que de otro modo se estaría ejecutando la función y almacenando el valor devuelto en la propiedad onreadystatechange.

Después de preparar la aplicación para la respuesta del servidor, se realiza la petición HTTP al servidor:

```
peticion_http.open('GET', 'http://localhost//EJ AJAX 2021//saludo.txt', true);
peticion_http.send(null);
```

Las instrucciones anteriores realizan el tipo de petición más sencillo que se puede enviar al servidor. En concreto, se trata de una petición de tipo GET simple que no envía ningún parámetro al servidor. La petición HTTP se crea mediante el método open(), en el que se incluye el tipo de petición (GET), la URL solicitada y un tercer parámetro que vale true (false indica que se actúa en modo síncrono)

Una vez creada la petición HTTP, se envía al servidor mediante el método send(). Este método incluye un parámetro que en el ejemplo anterior vale null.

Por último, cuando se recibe la respuesta del servidor, la aplicación ejecuta de forma automática la función establecida anteriormente.

```
function muestraContenido() {
  if(peticion_http.readyState == 4) {
    if(peticion_http.status == 200) {
      alert(peticion_http.responseText);
    }
  }
}
```

La función muestraContenido() comprueba en primer lugar que se ha recibido la respuesta del servidor (mediante el valor de la propiedad readyState). Si se ha recibido alguna respuesta, se comprueba que sea válida y correcta (comprobando si el código de estado HTTP devuelto es igual a 200). Una vez realizadas las comprobaciones, simplemente se muestra por pantalla el contenido de la respuesta del servidor (en este caso, el contenido del archivo solicitado) mediante la propiedad responseText

Ver Ejemplo 2 Carga XML, Ejemplo 3 Carga XML

Ver Ejemplo con asp de w3shools

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

Ver Ejemplo con php y acceso a base de datos

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

Ver Ejemplo con Python

https://recursospython.com/guias-y-manuales/actualizar-contenido-via-ajax/

AJAX Y JSON

JSON (JavaScript Object Notation)

JSON es un formato ligero para el intercambio de datos.

JSON es un subconjunto del lenguaje javascript que se basa en la construcción de una lista ordenada de valores, listas de objetos con una colección de pares nombre/valor.

Se puede usar como alternativa a XML en el intercambio de información vía Ajax si el volumen de datos que manejamos excede de lo razonable: Yahoo y Google lo usan en sus clientes de correo web para aligerar el peso de los documentos de intercambio

La sintaxis de JSON es relativamente sencilla

El elemento base de la sintaxis es un object. Está formado por un conjunto de pares nombre/valor. Un objeto comienza con una llave de apertura y finaliza con una llave de cierre. Cada nombre es seguido por dos puntos seguido de su valor, estando los pares nombre/valor separados por una coma.

```
{
"nombre":"Pepe",
"edad":34,
"domicilio":"calle Alcalá, 1",
"estudios”:[ "primario","secundario","universitario"]
}
```

```
{"empleados":[
   {"nombre":"John", "apellido":"Doe"},
   {"nombre":"Anna", "apellido":"Smith"},
   {"nombre":"Peter", "apellido":"Jones"}
]}
```

- **Las llaves determinan objetos**
- **Los corchetes determinan arrays**
- **JSON utiliza comillas dobles**

Los valores de JSON pueden ser:

- Un número (entero o de coma flotante)
- Una cadena (entre comillas dobles)
- Un booleano (true o false)

- Un array (entre corchetes)
- Un objeto (entre llaves)
- null

El tipo de archivo para los archivos JSON es ".json"

JSON.parse (Función de JavaScript)

Convierte una cadena de texto que sigue la notación de objetos de JavaScript (JSON) en un objeto

```
<!DOCTYPE html>
<html>
<body>
<h2>Crear JSON en JavaScript</h2>
<p id="demo"></p>
<script>
var text ='{"nombre":"Pepe",
            "edad":34,
            "domicilio":"calle Alcala, 1",
            "estudios":[ "primario","secundario","universitario"]}';
var obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
obj.nombre + "<br>" +
obj.edad + "<br>" +
obj.domicilio + "<br>"  +
obj.estudios[2];
</script>
</body>
</html>
```

```
var text ='{"empleados":
        [{"nombre":"John", "apellido":"Doe"},
        {"nombre":"Anna", "apellido":"Smith"},
        {"nombre":"Peter", "apellido":"Jones"}]}';
var obj = JSON.parse(text);
document.getElementById("demo").innerHTML='';
for (i=0;i<obj.empleados.length;i++){
document.getElementById("demo").innerHTML+=
        obj.empleados[i].nombre+' '+obj.empleados[i].apellido+'<br>';
}
```

JSON vs XML

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

Los siguientes ejemplos JSON y XML contienen la misma información, los datos de 3 empleados:

JSON Example

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

XML Example

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

Tanto JSON como XML se utilizan para recibir datos de un servidor web.

- Tanto JSON y XML son autodescriptivos
- Tanto JSON y XML son jerárquicos (valores dentro de los valores)
- Tanto JSON y XML pueden ser analizados y utilizados por una gran cantidad de lenguajes de programación
- Tanto JSON y XML se pueden recuperar con un XMLHttpRequest

Se diferencian:

- JSON no utiliza etiqueta final

- JSON es más corto
- JSON es más rápido para leer y escribir
- JSON puede utilizar matrices

La mayor diferencia es que XML tiene que ser analizado con un analizador XML mientras que JSON se puede analizar mediante una función de JavaScript estándar.

Todo esto hace que para aplicaciones AJAX, JSON sea más rápido y más fácil que XML.

JSON y AJAX

Uno de los usos más comunes de JSON es obtener datos de un servidor y mostrarlos en una página web.

EJEMPLOS JSON y AJAX: Ejemplo 3 Carga JSON, Ejemplo 4 Carga JSON:

Creamos el archivo enlaces.txt

```
[
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp"
}
]
```

Y utilizamos AJAX para cargarlo desde el servidor y transformar su contenido en un objeto mediante JSON.parse()

```
function descargaArchivo() {
// Obtener la instancia del objeto XMLHttpRequest
peticion_http = new XMLHttpRequest();
/ Preparar la funcion de respuesta
peticion_http.onreadystatechange = muestraContenido;
// Realizar peticion HTTP
peticion_http.open('GET', 'enlaces.txt', true);
peticion_http.send(null);
```

```
function muestraContenido() {
        if(peticion_http.readyState == 4) {
                if(peticion_http.status == 200) {
                        var arr = JSON.parse(peticion_http.responseText);
                        var out = "";
                        for(i = 0; i < arr.length; i++) {
                                out += '<a href="' + arr[i].url + '">' + arr[i].display +
'</a><br>';
                                }
                                document.getElementById("id1").innerHTML = out;
                        }
                }
        }
}
```

Otra manera de hacer peticiones HTTP: con la API fetch

https://programacionymas.com/blog/ajax-fetch-api-ejemplo

En sus orígenes, hacer peticiones Ajax no era muy sencillo. Se tenía que usar un objeto `XMLHttpRequest`.

Hoy en día, `fetch` permite realizar peticiones HTTP de una manera mucho más legible.

Aquí tienes un ejemplo:

```
fetch('https://rickandmortyapi.com/api/character')
    .then((response) => response.json())
    .then((data) => console.log(data));
```

Con este código realizas una **petición GET asíncrona a la API pública de Rick and Morty**, para obtener los personajes de la serie, en formato JSON.

Ejemplo fetch 1

El método fetch() inicia el proceso de obtención de un recurso de un servidor.

El método fetch() devuelve una “promesa”(promise) que se resuelve en un objeto “respuesta” (response)

¿Qué son las promesas en JavaScript y como crearlas?

Principales métodos de una promesa:

.then(): Este método se utiliza para manejar el resultado exitoso de una promesa. Recibe una función que se ejecutará cuando la promesa se resuelva con éxito y puede recibir el resultado como argumento.

.catch(): Se utiliza para manejar errores que puedan ocurrir durante la ejecución de la promesa. Puedes encadenar *.catch()* después de *.then()* para manejar errores específicos.

Usando fetch para obtener un archivo txt:

```
fetch('ruta/al/archivo.txt')
  .then(response => {
    if (!response.ok) {
      throw new Error('Error al obtener el archivo');
    }
    return response.text(); // Convierte la respuesta en texto
  })
  .then(data => {
    console.log(data); // Aquí tienes el contenido del archivo .txt
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

- fetch('ruta/al/archivo.txt'): Realiza la solicitud GET al archivo .txt en el servidor.
- response.text(): Convierte la respuesta a texto.
- data: Aquí es donde obtienes el contenido del archivo .txt.
- .catch(error): Captura cualquier error que ocurra durante la petición.

Es decir:

1. Solicita un archivo de texto ubicado en una URL específica.
2. Verifica si la respuesta es válida.
3. Convierte la respuesta en texto.
4. Imprime el contenido del archivo en la consola.
5. Maneja errores que puedan ocurrir durante el proceso.

Usando fetch para obtener un archivo XML:

```
fetch('ruta/al/archivo.xml')
 .then(response => {
  if (!response.ok) {
   throw new Error('Error al obtener el archivo XML');
  }
  return response.text(); // Convertimos la respuesta a texto
 })
 .then(data => {
  // Parseamos el contenido XML
  const parser = new DOMParser();
  const xmlDoc = parser.parseFromString(data, 'text/xml');
     // Trabajamos con el archivo XML
  console.log(xmlDoc);
 })
 .catch(error => {    console.error('Error:', error);
 });
```

- fetch('ruta/al/archivo.xml'): Realiza una solicitud GET al archivo XML.
- response.text(): Obtiene la respuesta como texto.
- DOMParser(): Usamos DOMParser para convertir el texto a un documento XML.
- parser.parseFromString(data, 'text/xml'): Convierte el texto a un objeto XML.
- Puedes trabajar con xmlDoc para acceder a los nodos y elementos del XML.

Usando fetch para obtener un archivo JSON:

```
fetch('ruta/al/archivo.json')
 .then(response => {
  if (!response.ok) {
   throw new Error('Error al obtener el archivo JSON');
  }
  return response.json(); // Convierte la respuesta en formato JSON
 })
 .then(data => {
  console.log(data); // Aquí tienes el contenido del archivo JSON
 })
 .catch(error => {    console.error('Error:', error);
 });
```

- fetch('ruta/al/archivo.json'): Realiza una solicitud GET para obtener el archivo JSON.
- response.json(): Convierte la respuesta a un objeto JavaScript (parsing del JSON).
- data: Aquí se obtiene el contenido del archivo JSON y puedes trabajarlo como un objeto.

Ejemplo 1b Carga txt con fetch
Ejemplo 2b Carga XML con fetch
Ejemplo 5b Carga JSON con fetch