Unidad de Trabajo 7: Programación AJAX en JavaScript

1. Introducción a AJAX

AJAX, que significa **JavaScript Asíncrono y XML**, es una técnica de desarrollo web que facilita la comunicación entre el navegador del usuario y el servidor en segundo plano. Esta tecnología permite realizar solicitudes al servidor sin recargar la página, lo que posibilita la gestión de respuestas y la actualización de contenidos de manera dinámica sin necesidad de recargas completas.

El término AJAX fue introducido por primera vez en el artículo "A New Approach to Web Applications", publicado por Jesse James Carrett el 18 de febrero de 2005.

Aunque AJAX no constituye una tecnología completamente nueva, más bien es la **combinación de diversas tecnologías** que se destacan por sus propios méritos. Estas tecnologías trabajan juntas con los siguientes objetivos:

- Conseguir una presentación basada en estándares mediante el uso de XHTML, CSS y técnicas avanzadas del DOM para mostrar la información de forma dinámica e interactiva.
- Facilitar el intercambio y manipulación de datos utilizando XML y XSLT.
- Recuperar datos de manera asíncrona a través del objeto XMLHTTPRequest.
- Emplear JavaScript como lenguaje principal para unificar todos los componentes.

Las **tecnologías** que conforman AJAX son diversas:

- XHTML y CSS para la presentación basada en estándares.
- DOM para la interacción y manipulación dinámica de la presentación.
- XML, XSLT y JSON para el intercambio y manipulación de información.
- XMLHTTPRequest para el intercambio asíncrono de información.
- JavaScript para integrar y coordinar todas las partes mencionadas anteriormente.

El paradigma clásico de las aplicaciones web sigue un patrón en el cual la mayoría de las interacciones del usuario ocurren en la interfaz, generando solicitudes HTTP hacia el servidor web. El servidor realiza un procesamiento que involucra la recopilación de información y la ejecución de acciones correspondientes, devolviendo finalmente una página HTML al cliente. Este enfoque deriva de la concepción original de la web como un medio hipertextual, pero, a nivel de aplicaciones de software, puede no ser siempre el más eficiente.

En este modelo convencional, cada solicitud al servidor implica que el usuario debe esperar, ya que la página a menudo cambia a una diferente. Hasta que todos los datos del servidor se reciben, el resultado no se muestra, limitando la interactividad del usuario con el navegador. AJAX aborda este problema al intentar eliminar estas esperas. Permite que el cliente realice

solicitudes al servidor mientras el navegador continúa mostrando la misma página web. Cuando el navegador recibe una respuesta del servidor, la presenta al cliente sin necesidad de recargar o cambiar de página.

En la actualidad, AJAX es ampliamente utilizado por numerosas empresas y productos, entre ellos Google (en aplicaciones como Gmail, Google Suggest y Google Maps), así como por servicios como Flickr, Amazon, entre otros. Existen diversas razones para adoptar AJAX:

- Está basado en estándares abiertos.
- Mejora la usabilidad de las aplicaciones web.
- Es compatible con cualquier plataforma y navegador.
- Ofrece beneficios significativos para las aplicaciones web.
- Puede integrarse con tecnologías como Flash.
- Constituye la base de la web 2.0.
- Es independiente del tipo de tecnología de servidor utilizada.
- Contribuye a mejorar la estética y la experiencia visual de la web.

1.1. Requerimientos previos

Cuando trabajamos con AJAX, es esencial tener en cuenta una serie de requisitos previos fundamentales para programar con esta metodología. Hasta este momento, nuestras aplicaciones en JavaScript no requerían un servidor web para funcionar, a menos que quisieras enviar datos de un formulario y almacenarlos en una base de datos. De hecho, todas las aplicaciones en JavaScript que has desarrollado hasta ahora las has probado directamente abriéndolas con el navegador o haciendo doble clic en el archivo HTML.

Sin embargo, al programar con AJAX, necesitaremos un **servidor web**, ya que las solicitudes AJAX que realicemos se dirigirán a un servidor. Los componentes esenciales que necesitaremos son:

- Servidor web (por ejemplo, Apache, ligHTTPd, IIS).
- Servidor de bases de datos (por ejemplo, MySQL, PostgreSQL).
- Lenguaje del lado del servidor (por ejemplo, PHP, ASP).

Aunque podríamos instalar cada uno de estos componentes por separado, en muchas ocasiones **resulta más conveniente utilizar una aplicación que los agrupe** sin necesidad de instalarlos individualmente. Existen varios tipos de aplicaciones de este tipo, que se pueden categorizar en dos grupos según el sistema operativo sobre el cual operan:

- Servidor LAMP (Linux, Apache, MySQL y PHP).
- Servidor WAMP (Windows, Apache, MySQL y PHP).

Una aplicación ampliamente utilizada de este tipo es **XAMPP**, disponible tanto para Windows como para Linux. XAMPP se puede instalar en una memoria USB, lo que permite ejecutarla en cualquier ordenador, proporcionando así un servidor web siempre disponible para programar tus aplicaciones AJAX de manera conveniente. A continuación tienes el enlace para la instalación de XAMPP Apache:

https://www.apachefriends.org/es/index.html

1.2. Comunicación asíncrona

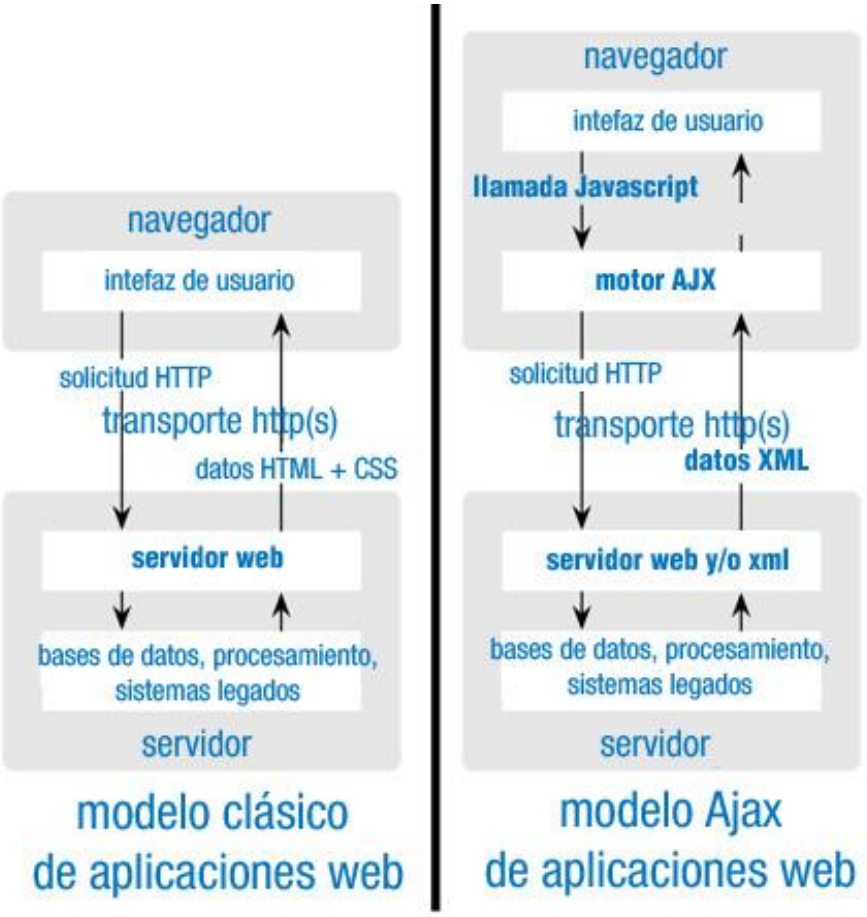
Como mencionamos en la introducción a AJAX, la mayoría de las aplicaciones web siguen el siguiente proceso:

1. El usuario solicita información al servidor.
2. El servidor lleva a cabo los procesos solicitados, que pueden incluir la búsqueda de información, consultas a una base de datos, lectura de archivos, cálculos numéricos, entre otros.
3. Una vez que el servidor ha completado estos procesos, devuelve los resultados al cliente.

Durante el paso 2, mientras los procesos se ejecutan en el servidor, el cliente se encuentra en un estado de espera, ya que el navegador está bloqueado hasta recibir la información con los resultados del servidor.

Una aplicación AJAX transforma la **dinámica de funcionamiento** de una aplicación web al eliminar las esperas y bloqueos que normalmente experimenta el cliente. En otras palabras, el usuario puede seguir interactuando con la página web mientras se realiza la solicitud al servidor. Cuando se recibe una respuesta confirmada del servidor, esta se muestra al cliente o se ejecutan las acciones que el programador de la página web haya definido. Este enfoque mejora significativamente la experiencia del usuario al permitir una interactividad continua sin interrupciones.

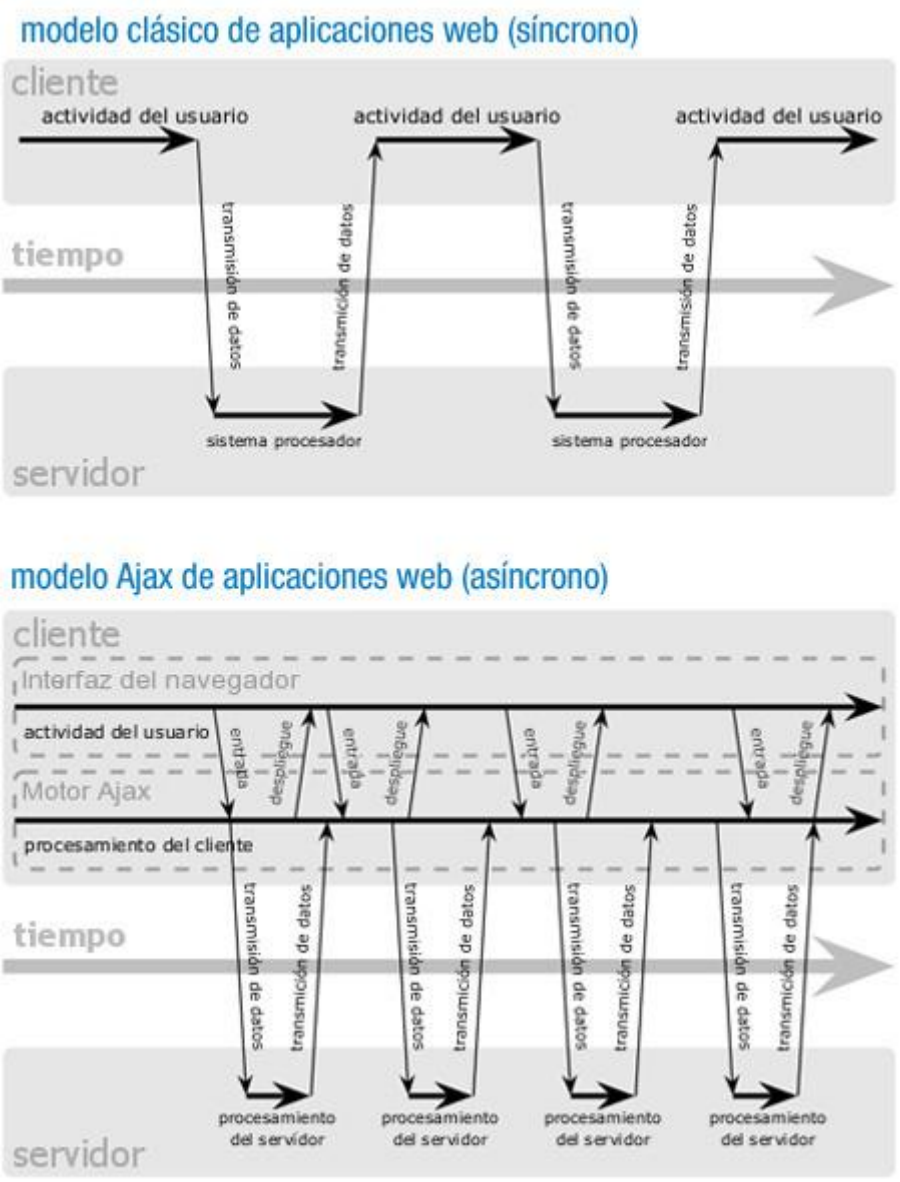
En el siguiente gráfico puedes ver y comparar el modelo de ejecución tradicional frente al modelo AJAX:



Ahora, ¿cómo se consigue realizar esa petición asíncrona sin bloquear el navegador?

Para realizar solicitudes al servidor sin que el navegador se quede bloqueado, es necesario utilizar el **motor AJAX**, programado en JavaScript. Este motor se encarga de gestionar las peticiones AJAX del usuario y de establecer la comunicación con el servidor. Es precisamente este motor el que posibilita que la interacción ocurra de forma asíncrona, independientemente de la comunicación con el servidor.

De esta manera, el usuario no se verá afectado por la carga del navegador ni se encontrará con una pantalla en blanco. Cada acción del usuario, que normalmente generaría una solicitud HTTP al servidor, se transformará en una petición AJAX con esa solicitud. Será este motor el que se encargue de todo el proceso de comunicación y obtención de datos de forma asíncrona con el servidor, permitiendo que la interacción del usuario con la aplicación no se vea afectada de modo que su actividad no deba ser interrumpida tal y como puedes ver en el gráfico mostrado a continuación:



1.3. El API XMLHttpRequest

El núcleo de AJAX se basa en una API conocida como **XMLHTTPRequest (XHR)**, disponible en lenguajes de scripting del lado del cliente, como JavaScript. Esta API se utiliza para realizar solicitudes HTTP o HTTPS directamente al servidor web y cargar las respuestas en la página del cliente. Los datos recibidos del servidor pueden estar en formato de texto plano o XML, y se pueden utilizar para modificar dinámicamente el DOM del documento actual sin necesidad de recargar la página. Además, si los datos se reciben en formato JSON, pueden evaluarse fácilmente con JavaScript.

XMLHTTPRequest desempeña un papel crucial en la técnica AJAX, ya que **permite realizar solicitudes asíncronas al servidor**. El concepto detrás de este objeto se originó en el

desarrollo de Outlook Web Access de Microsoft, específicamente en el desarrollo de Microsoft Exchange Server 2000.

La interfaz IXMLHTTPRequest se implementó inicialmente en la segunda versión de la librería MSXML, y se permitió el acceso a través de ActiveX en Internet Explorer 5.0 en marzo de 1999. Mozilla también desarrolló su propia interfaz llamada nslXMLHTTPRequest para su motor Gecko, creando un objeto JavaScript llamado XMLHTTPRequest para su uso. Este objeto se convirtió en un estándar entre varios navegadores, como Safari, Konqueror, Opera e iCab.

El W3C publicó un borrador de especificación para XMLHTTPRequest en abril de 2006, con el objetivo de establecer mínimos de interoperabilidad basados en las diversas implementaciones existentes. La última revisión de esta especificación se realizó en noviembre de 2009. Microsoft incorporó el objeto XMLHTTPRequest a su lenguaje de script con Internet Explorer 7.0 en octubre de 2006.

Finalmente, con la aparición de **librerías cross-browser** como jQuery y Prototype, los programadores pueden aprovechar la funcionalidad de XMLHTTPRequest sin necesidad de codificar directamente sobre la API, lo que acelera significativamente el desarrollo de aplicaciones AJAX.

1.3.1. Creación del objeto XMLHttpRequest. Métodos y propiedades

En primer lugar, para poder programar con AJAX el primer paso será instanciar un objeto del tipo **XMLHTTPRequest**, que va a ser el que nos permitirá realizar las peticiones en segundo plano al servidor web. Por convenio, se suele denominar al mismo como **xhr**. De ese modo, la primera línea de comando que deberás incluir en tu código para programar AJAX será:

```
let xhr = new XMLHttpRequest();
```

Este objeto dispone de **métodos** que serán necesarios para poder manejar las peticiones realizadas al servidor. En la siguiente tabla se resumen los más importantes:

Método	Descripción
abort()	Cancela la solicitud en curso.
getAllResponseHeaders()	Devuelve la información completa de la cabecera.
getResponseHeader()	Devuelve la información específica de la cabecera.

open(método,url,async,usuario,password)	Especifica el tipo de solicitud, la URL, si la solicitud se debe gestionar de forma asíncrona o no, y otros atributos opcionales de la solicitud: **método:** indicamos el tipo de solicitud: GET o POST. **url:** la dirección del fichero al que le enviamos las peticiones en el servidor. **async:** true (asíncrona) o false (síncrona). **usuario y password:** si fuese necesaria la autenticación en él
send(datos)	Envía la solicitud al servidor. **datos:** Se usa en el caso de que estemos utilizando el método POST, como método de envío. Si usamos GET, datos será null.
setRequestHeader()	Añade el par etiqueta/valor a la cabecera de datos que se enviará al servidor.

A continuación, te muestro un ejemplo de cómo se emplea este objeto. En este caso, se va a realizar de forma síncrona (funcionamiento normal del navegador) aunque empleando para ello el objeto XMLHttpRequest. Este ejemplo, consta de 3 archivos: index.html, index.js y fecha.php. Este último archivo es el encargado de gestionar en el servidor la petición AJAX realizada desde el navegador mediante JavaScript. Recuerda que Para probar el siguiente código, que incluye una petición AJAX, tienes que hacerlo a través del servidor web. Para ello debes copiar los ficheros del ejemplo, dentro de la raíz del servidor web, en una carpeta a la que llamaremos, por ejemplo, web/dwec07132. Arrancaremos el servidor web e iremos a la dirección **HTTP://localhost/web/dwec07132**

Ejemplo 1

Archivo index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejemplo dwec07 - 1.3.2 - AJAX SÍNCRONO</title>
<script src="index.js"></script>
<style>
  #resultados{
    background: yellow;
  }
</style>
```

```
</head>
<body>
<h2>A continuación se cargarán por AJAX los datos recibidos en la solicitud SÍNCRONA:</h2>
<p>Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la página PHP en el servidor.</p>
<h4>Contenedor resultados:</h4><br>
<div id="resultados"></div>
</body>
</html>
```

Archivo index.js

```
document.addEventListener('DOMContentLoaded',function(){
  let xhr = new XMLHttpRequest();//Creamos el objeto para manejo AJAX.

  xhr.open('GET','fecha.php',false);//Abrimos el canal con el servidor.
  xhr.send();//Enviamos la petición.

  //Utilizamos los métodos del DOM para añadir el resultado.
  let nodo =document.getElementById('resultados');
  nodo.appendChild(document.createTextNode(xhr.responseText));

  //También se podía haber incluido como:
  //document.getElementById('resultados').innerHTML = xhr.responseText;
});
```

Archivo fecha.php

```
<?php
// retrasamos 2 segundos la ejecución de esta página PHP.
sleep(2);
// Mostramos la fecha y hora del servidor web.
echo "La fecha y hora del Servidor Web: ";
echo date("j/n/Y G:i:s.");
?>
```

Esta función efectúa una solicitud AJAX de manera **síncrona**, siguiendo el comportamiento normal del navegador. En esta petición, se carga el archivo especificado en la URL, que debe pertenecer al mismo dominio del servidor. La respuesta obtenida, identificada como **responseText**, se inserta en un DIV utilizando los métodos del DOM vistos hasta ahora.

Asimismo, las propiedades de este objeto XMLHttpRequest van a permitir manejar las solicitudes AJAX y respuestas del servidor. Entre los métodos más importantes se encuentran los siguientes:

Propiedad	Descripción
onreadystatechange	Almacena una función (o el nombre de una función), que será llamada automáticamente, cada vez que se produzca un cambio en la propiedad **readyState.**
readyState	Almacena el estado de la petición XMLHTTPRequest. Posibles estados, del 0 al 4: 0: solicitud no inicializada. 1: conexión establecida con el servidor. 2: solicitud recibida. 3: procesando solicitud. 4: solicitud ya terminada y la respuesta está disponible.
responseText	Contiene los datos de respuesta, como una cadena de texto.
responseXML	Contiene los datos de respuesta, en formato XML.
status	Contiene el estado numérico, devuelto en la petición al servidor (por ejemplo: "404" para "No encontrado" o "200" para "OK").
statusText	Contiene el estado en formato texto, devuelto en la petición al servidor (por ejemplo: "Not Found" o "OK").

2. Envío y recepción de datos de forma asíncrona

Tal y como viste en el ejemplo anterior, una vez creado el objeto XMLHttpRequest se debe abrir un canal de comunicación con el servidor empleando el método **open().** Una vez hecho ésto, el método **send()** se encarga de enviar la solicitud al servidor (y los parámetros si se emplea el método POST, tal y como veremos más adelante). Ahora bien, en el caso anterior la petición se realizó de modo síncrono. Debemos, por tanto, dar un paso más en nuestro desarrollo de aplicaciones web y comenzar a gestionar la sincronía de la página. Para ello, implementaremos el uso de las propiedades asociadas a la API XMLHttpRequest.

2.1. Estados de una solicitud asíncrona

Durante la ejecución de una solicitud asíncrona, la petición atraviesa diversos **estados**, numerados del 0 al 4, según la propiedad readyState del objeto XHR. Este proceso ocurre independientemente de si el archivo solicitado al servidor se encuentra disponible o no. Cuando dicha petición termina, tendremos que comprobar cómo lo hizo, y para ello evaluamos la propiedad status que contiene el estado devuelto por el servidor: 200: OK,

404: Not Found, etc. Si el status fue OK ya podremos comprobar, en la propiedad responseText o responseXML del objeto XHR, los datos devueltos por la petición. A continuación, vamos a volver a ver el ejemplo 1 pero, en este caso, gestionando la asincronía de la petición al servidor. Para ello, se emplea la propiedad **onreadystatechange** que nos permite asignar una función al objeto **xhr** y que se va a ejecutar **cada vez que cambie** el estado de la petición. Asimismo, mediante las propiedades **status** y **readyState** podemos definir el comportamiento de la web en función del estado/resultado de la petición. Cabe destacar que, en estos ejemplos, estamos programando la respuesta de forma sencilla para que sean didácticos. No obstante, en proyectos reales deberás tener en cuenta que en función de otros estados/respuestas del servidor el usuario deberá recibir información sobre la petición si, por ejemplo, la misma no fue exitosa, etc.

Ejemplo 2

Archivo index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejemplo dwec07 - 2.1 - AJAX ASÍNCRONO</title>
<script src="index.js"></script>
<style>
  #resultados{
    background: yellow;
  }
</style>
</head>
<body>
<h2>A continuación se cargarán por AJAX los datos recibidos en la solicitud ASÍNCRONA:</h2>
<p>Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la página PHP en el servidor.</p>
<H4>Estado de las solicitudes:</H4><br>
<div id="estado"></div>

<h4>Contenedor resultados:</h4><br>
<div id="resultados"></div>

</body>
</html>
```

Archivo index.js

```
document.addEventListener('DOMContentLoaded',function(){
   let xhr = new XMLHttpRequest();

   xhr.open('GET','fecha.php',true);//true = petición asíncrona.

      //Utilizaremos onreadystatechange para ejecutar una función cada vez que haya un cambio en el estado de la petición asíncrona:
   xhr.onreadystatechange = function(){

      switch(xhr.readyState){
            // Evaluamos el estado de la petición AJAX y vamos mostrando el valor actual de readyState en cada llamada:
         case 0:
            document.getElementById('estado').innerHTML += "0 - Sin iniciar.<br/>";
            break;
         case 1:
            document.getElementById('estado').innerHTML += "1 - Cargando.<br/>";
            break;
         case 2:
            document.getElementById('estado').innerHTML += "2 - Cargado.<br/>";
            break;
         case 3:
            document.getElementById('estado').innerHTML += "3 - Interactivo.<br/>";
            break;
         case 4:
            document.getElementById('estado').innerHTML += "4 - Completado.";
            if(xhr.status === 200){
               document.getElementById('resultados').innerHTML = xhr.responseText;
            };
            break;
      }
   }

   xhr.send();

});
```

Archivo fecha.php

```php
<?php
// Para que el navegador no haga cache de los datos devueltos por la página PHP.
header('Cache-Control: no-cache, must-revalidate');
header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

// retrasamos 2 segundos la ejecución de esta página PHP.
sleep(2);
// Mostramos la fecha y hora del servidor web.
echo "La fecha y hora del Servidor Web: ";
echo date("j/n/Y G:i:s.");
?>
```

En el siguiente ejemplo, te muestro cómo puedes incluir una imagen de carga en tu página para que la misma se ejecute mientras se procesa la respuesta del servidor:

Ejemplo 3

Archivo index.html

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejemplo dwec07 - 2.2 - AJAX ASÍNCRONO</title>
<script src="index.js"></script>
<style>
  #resultados{
    background: yellow;
  }
</style>
</head>
<body>
<h2>A continuación se cargarán por AJAX los datos recibidos en la solicitud ASÍNCRONA:</h2>
<p>Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la página PHP en el servidor.</p>
<h4>Contenedor resultados:</h4><br>
<div id="resultados"></div>
<H4>Estado de las solicitudes:</H4><br>
<div id="indicador"></div>
```

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

Archivo index.js

```
document.addEventListener('DOMContentLoaded', function () {
   let xhr = new XMLHttpRequest();

   // Añadimos un indicador de proceso asíncrono:
   let indicador = document.createElement('img');
   indicador.setAttribute('src', 'state.gif');
   document.getElementById('indicador').appendChild(indicador);

   xhr.open('GET', 'fecha.php', true); // true = petición asíncrona.
    // Utilizaremos onreadystatechange para ejecutar una función cada vez que haya un cambio en el estado de la petición asíncrona:
   xhr.onreadystatechange = function () {

      // Una vez finalizada la solicitud mostramos los resultados:
      if (xhr.readyState === 4 && xhr.status === 200) {
         // Desactivamos el indicador AJAX:
         document.getElementById("indicador").innerHTML = "";
         // Metemos en el contenedor resultados las respuestas de la petición AJAX:
         document.getElementById('resultados').innerHTML = xhr.responseText;
      }
   }

   xhr.send();
});
```

Archivo fecha.php

```
<?php
// Para que el navegador no haga cache de los datos devueltos por la página PHP.
header('Cache-Control: no-cache, must-revalidate');
header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

// retrasamos 2 segundos la ejecución de esta página PHP.
sleep(2);
// Mostramos la fecha y hora del servidor web.
echo "La fecha y hora del Servidor Web: ";
echo date("j/n/Y G:i:s.");
?>
```

2.2. Envío de datos usando GET

A continuación, vamos a ver un ejemplo en el que se realiza una petición AJAX a la página **procesar.php**, pasando dos parámetros: nombre y apellidos usando el método GET. En la petición GET, los parámetros que pasemos en la solicitud, **se enviarán formando parte de la URL separados de la misma por ? y con la sintaxis de clave=valor.** Por ejemplo: **procesar.php?nombre=Teresa&apellidos=Blanco Ferreiro.** Cuando realizamos la petición por el método GET, te recordamos una vez más, que pondremos null como parámetro del método send, ya que los datos son enviados a la página procesar.php, formando parte de la URL: send(null). No obstante, cabe destacar que también puedes dejar el método send sin parámetro alguno:

Ejemplo 4

Archivo index.html

```
<?php
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejemplo dwec07 - 2.3 - AJAX ASÍNCRONO GET - POST</title>
<script src="index.js"></script>
<style>
  #resultados{
  background: yellow;
  }
</style>
</head>
<body>
<h3>A continuación se cargarán por AJAX los datos recibidos en la solicitud ASÍNCRONA:</h3><br/>
<p>Contenedor resultados:</p>
<div id="resultados"></div>
<div id="indicador"></div>
</body>
</html>
?>
```

Archivo index.js

```
document.addEventListener('DOMContentLoaded',function(){
  let xhr = new XMLHttpRequest();

    // Activamos el indicador Ajax antes de realizar la petición. Fíjate que lo puedes incorporar también como:
  document.getElementById('indicador').innerHTML = '<img src="state.gif"/>';

   //Al emplear el método GET, los parámetros se pasan en la URL del servidor que maneja la petición después de ? separados como nombre=valor:
  xhr.open('GET', 'procesar.php?nombre=Teresa&apellidos=Blanco Ferreiro', true);
  xhr.onreadystatechange = function(){
    if(xhr.readyState === 4 && xhr.status === 200){
      document.getElementById('indicador').innerHTML = "";
      document.getElementById('resultados').innerHTML = xhr.responseText;
    }
  };
  xhr.send();
});
```

Archivo procesar.php

```
<?php
// Para que el navegador no haga cache de los datos devueltos por la página PHP.
header('Cache-Control: no-cache, must-revalidate');
header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

// Imprimimos un mensaje con los parámetros recibidos:
if (isset($_GET['nombre']))
echo "Saludos desde el servidor: hola {$_GET['nombre']} {$_GET['apellidos']}. ";
else if (isset($_POST['nombre']))
echo "Saludos desde el servidor: hola {$_POST['nombre']} {$_POST['apellidos']}. ";
// Mostramos la fecha y hora del servidor web.
echo "La fecha y hora del Servidor Web: ";
echo date("j/n/Y G:i:s");
?>
```

2.3. Envío de datos usando POST

Observemos ahora un ejemplo en el cual se lleva a cabo una solicitud AJAX a la página procesar.php, transmitiendo dos parámetros, nombre y apellidos, mediante el método **POST**.

Ejemplo 5

Archivo index.html

```
<?php
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejemplo dwec07 - 2.3 - AJAX ASÍNCRONO GET - POST</title>
<script src="index.js"></script>
<style>
  #resultados{
  background: yellow;
  }
</style>
</head>
<body>
<h3>A continuación se cargarán por AJAX los datos recibidos en la solicitud ASÍNCRONA:</h3><br/>
<p>Contenedor resultados:</p>
<div id="resultados"></div>
<div id="indicador"></div>
</body>
</html>
?>
```

Archivo index.js

```
document.addEventListener('DOMContentLoaded',function(){
  let xhr = new XMLHttpRequest();

  // Activamos el indicador Ajax antes de realizar la petición.
  document.getElementById('indicador').innerHTML = '<img src="state.gif"/>';

    //Al emplear el método POST, los parámetros se pasan en el método send como nombre=valor:
  xhr.open('POST', 'procesar.php', true);
  xhr.onreadystatechange = function(){
    if(xhr.readyState === 4 && xhr.status === 200){
      document.getElementById('indicador').innerHTML = "";
      document.getElementById('resultados').innerHTML = xhr.responseText;
```

```
    }
  };

   // En las peticiones POST tenemos que enviar en la cabecera el Content-Type ya que los
datos se envían formando parte de la cabecera:
  xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
  xhr.send('nombre=Teresa&apellidos=Blanco Ferreiro');
});
```

Archivo procesar.php

```
<?php
// Para que el navegador no haga cache de los datos devueltos por la página PHP.
header('Cache-Control: no-cache, must-revalidate');
header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

// Imprimimos un mensaje con los parámetros recibidos:
if (isset($_GET['nombre']))
echo "Saludos desde el servidor: hola {$_GET['nombre']} {$_GET['apellidos']}. ";
else if (isset($_POST['nombre']))
echo "Saludos desde el servidor: hola {$_POST['nombre']} {$_POST['apellidos']}. ";
// Mostramos la fecha y hora del servidor web.
echo "La fecha y hora del Servidor Web: ";
echo date("j/n/Y G:i:s");
?>
```

En este caso, la URL en el método open no incluye los parámetros sino que los mismos se envían en el método send, también con un formato **clave = valor**. El siguiente cambio sí resulta más significativo pues tenemos que crear una **cabecera** con el tipo de contenido que vamos a enviar, justo antes de enviar la petición con el método **send.** La cadena **"application/x-www-form-urlencoded"** indica que los datos en el cuerpo de la solicitud están codificados en el formato de datos de formulario URL-encodificados. En este formato, los datos se envían como pares clave-valor separados por "&", y los valores de las claves y los datos se codifican para que sean seguros para ser transmitidos a través de la web.

2.4. Recepción de datos en formato XML

En sus orígenes AJAX se pensó para recibir datos desde el servidor en formato XML. Cuando efectuamos una solicitud AJAX que nos proporciona las respuestas en formato XML, deberemos examinar esos datos en la propiedad **responseXML** del objeto XHR. Dicha respuesta podrá ser procesada a continuación mediante los métodos y propiedades de JavaScript para la manipulación del DOM puesto que los archivos XML contienen los datos

en formato etiqueta. El siguiente ejemplo que te propongo consta de 3 archivos: index.html, index.js y catalogo.xml. La finalidad del script en este caso es solicitar los datos de un catálogo de CDs y mostrarlos a continuación en pantalla:

Ejemplo 6

Archivo index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejemplo dwec07 - 2.3 - AJAX CON DATOS XML</title>
<script src="index.js"></script>
</head>
<body>
A continuación se cargarán por AJAX los datos recibidos en la solicitud ASÍNCRONA:<br/>
Contenedor resultados:<div id="resultados"></div>
</body>
</html>
```

Archivo index.js

```
document.addEventListener('DOMContentLoaded',function(){
  let xhr = new XMLHttpRequest();

  xhr.open('GET','catalogo.xml',true);
  xhr.onreadystatechange = function(){
    if(xhr.readyState === 4 && xhr.status === 200){
      let datos = xhr.responseXML;
      // Tenemos que recorrer el fichero XML empleando los métodos del DOM
      // Array que contiene todos los CD's del fichero XML
      let CDs = datos.getElementsByTagName("CD");

      // En la variable salida compondremos el código HTML de la tabla a imprimir.
                                                    let salida = "<table
border='1'><tr><th>Titulo</th><th>Artista</th><th>Año</th></tr>";

      // Hacemos un bucle para recorrer todos los elementos CD.
      for (i=0;i<CDs.length;i++){
        salida += "<tr>";
        // Para cada CD leemos los datos:
```

```
            let titulos = CDs[i].getElementsByTagName("TITLE");
            salida += "<td>" + titulos[0].firstChild.nodeValue + "</td>";

            let artistas = CDs[i].getElementsByTagName("ARTIST");
            salida += "<td>" + artistas[0].firstChild.nodeValue + "</td>";

            let anhos = CDs[i].getElementsByTagName("YEAR");
            salida += "<td>" + anhos[0].firstChild.nodeValue + "</td>";

            //Se seguiría con el resto de datos/etiquetas.
            // Cerramos la fila.
            salida += "</tr>";
        }
        // Cuando ya no hay más Cd's cerramos la tabla.
        salida += "</table>";
        // Imprimimos la tabla dentro del contenedor resultados.
        document.getElementById("resultados").innerHTML = salida;
    }
  }
  xhr.send();
});
```

Archivo catalogo.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>
```

2.5. Recepción de datos en formato JSON

Otro formato de intercambio de datos ampliamente utilizado en AJAX es JSON (JavaScript Object Notation). Se trata de un formato de intercambio de datos alternativo a XML, que se caracteriza por ser mucho más sencillo de leer, escribir e interpretar. En JSON, los datos se expresan en formato JavaScript. La sintaxis de JSON es como la sintaxis **literal de un objeto en javaScript**, excepto que, esos objetos no pueden ser asignados a una variable. **JSON representará los datos en sí mismos.** Por ejemplo:

```
{
"Name" : "Beatles",
"Country" : "England",
"YearFormed" : 1959,
"Style" : "Rock'n'Roll",
"Members" : ["Paul","John","George","Ringo"]
}
```

Cabe destacar que una cadena JSON es simplemente un **texto** y no constituye un objeto en sí mismo. Antes de poder ser utilizada en JavaScript, es necesario realizar su conversión a un objeto. Para llevar a cabo esta conversión, se emplea el método **JSON.parse()**. Una vez realizada dicha conversión, y asignando ahora el objeto a una variable, podemos manipular el mismo con los métodos y propiedades ya vistas en JavaScript.

Veamos a continuación un ejemplo de cómo se manejan mediante javaScript los datos JSON. En el siguiente ejemplo, se van a cargar datos de diversos centros educativos y, una vez convertidos en un objeto de javaScript, finalmente se representarán en una tabla en pantalla:

Ejemplo 7

Archivo index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejemplo dwec07 - 2.7 - JSON Ajax asíncrono</title>
<script src="index.js"></script>
</head>
<body>
<h3>A continuación se cargarán por AJAX los datos recibidos en la solicitud ASÍNCRONA:</h3>
<p>Contenedor resultados:</p><br/>
<div id="resultados"></div>
</body>
</html>
```

Archivo index.js

```
document.addEventListener('DOMContentLoaded', function () {
  let xhr = new XMLHttpRequest();

  xhr.open('GET', 'datos.json', true);
  xhr.onreadystatechange = function () {
    if (xhr.readyState == 4 && xhr.status == 200) {
      // Utiliza JSON.parse() para convertir la cadena JSON en un objeto:
      let resultados = JSON.parse(xhr.responseText);

                                        texto = "<table border=1><tr><th>Nombre
Centro</th><th>Localidad</th><th>Provincia</th><th>Telefono</th><th>Fecha
Visita</th><th>Numero Visitantes</th></tr>";

        // Hacemos un bucle para recorrer todos los objetos literales recibidos en el array
resultados y mostrar su contenido:
      for (var i=0; i < resultados.length; i++){
        let objeto = resultados[i];
                                  texto += "<tr><td>"+objeto.nombrecentro+"</td><td>"+
objeto.localidad+"</td><td>"+objeto.provincia+"</td><td>"+
objeto.telefono+"</td><td>"+objeto.fechavisita+"</td><td>"+
objeto.numvisitantes+"</td></tr>";
      }
      document.getElementById('resultados').innerHTML = texto;
    }
  }
  xhr.send();
});
```

Archivo datos.json

```
[
 {
  "id": 1,
  "nombrecentro": "IES Ramon Mª Aller Ulloa",
  "localidad": "Lalin",
  "provincia": "Pontevedra",
  "telefono": "986780114",
  "fechavisita": "2010-11-26",
  "numvisitantes": 90
 },
 {
```

```
  "id": 2,
  "nombrecentro": "IES A Piringalla",
  "localidad": "Lugo",
  "provincia": "Lugo",
  "telefono": "982212010",
  "fechavisita": "2010-11-26",
  "numvisitantes": 85
 },
 {
  "id": 3,
  "nombrecentro": "IES San Clemente",
  "localidad": "Santiago de Compostela",
  "provincia": "A Coruña",
  "telefono": "981580496",
  "fechavisita": "2010-11-26",
  "numvisitantes": 60
 },
 {
  "id": 4,
  "nombrecentro": "IES de Teis",
  "localidad": "Vigo",
  "provincia": "Pontevedra",
  "telefono": "986373811",
  "fechavisita": "2010-11-27",
  "numvisitantes": 72
 },
 {
  "id": 5,
  "nombrecentro": "IES Leliadoura",
  "localidad": "Ribeira",
  "provincia": "A Coruña",
  "telefono": "981874633",
  "fechavisita": "2010-11-25",
  "numvisitantes": 0
 },
 {
  "id": 6,
  "nombrecentro": "IES Cruceiro Baleares",
  "localidad": "Culleredo",
  "provincia": "A Coruña",
  "telefono": "981660700",
```

```
  "fechavisita": "2010-11-26",
  "numvisitantes": 30
 },
 {
  "id": 7,
  "nombrecentro": "IES Leliadoura",
  "localidad": "Ribeira",
  "provincia": "A Coruña",
  "telefono": "981874633",
  "fechavisita": "2010-11-25",
  "numvisitantes": 50
 },
 {
  "id": 8,
  "nombrecentro": "IES Cruceiro Baleares",
  "localidad": "Culleredo",
  "provincia": "A Coruña",
  "telefono": "981660700",
  "fechavisita": "2010-11-26",
  "numvisitantes": 30
 },
 {
  "id": 9,
  "nombrecentro": "IES As Lagoas",
  "localidad": "Ourense",
  "provincia": "Ourense",
  "telefono": "988391325",
  "fechavisita": "2010-11-26",
  "numvisitantes": 35
 },
 {
  "id": 10,
  "nombrecentro": "IES As Fontiñas",
  "localidad": "Santiago de Compostela",
  "provincia": "A Coruña",
  "telefono": "981573440",
  "fechavisita": "2010-11-27",
  "numvisitantes": 64
 }
]
```

3. Librerías cross-browser para programación AJAX

La programación con AJAX constituye uno de los fundamentos de lo que conocemos como web 2.0, un término que engloba aplicaciones web que facilitan el intercambio de información, la interoperabilidad, un diseño centrado en el usuario y la colaboración en línea. Ejemplos de la web 2.0 abarcan comunidades web, servicios web, aplicaciones web, redes sociales, plataformas de alojamiento de vídeos, wikis, blogs, mashups (páginas web o aplicaciones que combinan datos, presentaciones y funcionalidades de una o más fuentes para crear servicios nuevos), entre otros.

Las aplicaciones web 2.0 han generado numerosas utilidades, herramientas y frameworks para el desarrollo web con JavaScript, DHTML (HTML dinámico) y AJAX. Utilizar una **librería o un framework** para AJAX ofrece la ventaja significativa de ahorrar tiempo y reducir la cantidad de código en nuestras aplicaciones. Con algunas librerías, podemos realizar peticiones AJAX con una simple instrucción de código, sin tener que preocuparnos por crear el objeto XHR ni gestionar el código de respuesta del servidor o los estados de la solicitud.

Otro beneficio importante de estas librerías es su capacidad para **garantizar la compatibilidad entre navegadores (cross-browser)**. Esto significa que la librería se encarga de crear la petición AJAX de manera adecuada según el navegador utilizado, eliminando así la preocupación por las diferencias entre navegadores.

En el inicio de 2008, Google lanzó su API de librerías AJAX, que funciona como una red de distribución de contenido y arquitectura de carga para algunos de los frameworks más populares. Esta API simplifica el desarrollo de mashups en JavaScript al eliminar desafíos como alojar las librerías (ya que están centralizadas en Google) y configurar las cabeceras de caché. La API proporciona acceso a diversas librerías Open Source, realizadas con JavaScript, incluyendo jQuery, prototype, scriptaculous, mooTools, dojo, swfobject, chrome-frame, webfont, entre otras. Los scripts de estas librerías están disponibles directamente a través de la URL de descarga o mediante el método google.load() del cargador de la API AJAX de Google.

Hay muchísimas librerías que se pueden utilizar para programar AJAX, dependiendo del lenguaje que utilicemos. Nosotros nos vamos a centrar en el uso de la librería jQuery, por ser una de las más utilizadas hoy en día por empresas como Google, DELL, digg, NBC, CBS, NETFLIX, mozilla.org, wordpress, drupal, etc.

3.1. Introducción a JQuery

jQuery es una librería de JavaScript que simplifica significativamente la programación. Al utilizar JavaScript, es necesario garantizar la compatibilidad con varios navegadores, lo que implica escribir código compatible. jQuery aborda estos desafíos al proporcionar la infraestructura necesaria para desarrollar aplicaciones complejas en el lado del cliente. Basado en el principio de "escribe menos y produce más", éste ofrece soporte para la creación de interfaces de usuario, efectos dinámicos, AJAX, acceso al DOM, eventos, entre otras funcionalidades.

Además, jQuery cuenta con una amplia variedad de plugins que facilitan la creación de presentaciones con imágenes, validaciones de formularios, menús dinámicos, funcionalidades de arrastrar y soltar, entre otras. Esta librería es gratuita, cuenta con una licencia que permite su uso en cualquier tipo de plataforma, ya sea con fines personales o comerciales. El tamaño del archivo es de aproximadamente 31 KB y su carga es rápida. Una vez cargada, la librería queda almacenada en la caché del navegador, lo que significa que otras páginas que utilicen la misma librería no necesitarán cargarla nuevamente desde el servidor. Aquí te dejo la página principal de JQuery:

https://jquery.com/

En primer lugar, para programar con JQuery se debe incorporar la librería a nuestro proyecto. Ésto se puede hacer de dos modos:

Cargando la librería directamente desde la propia web de jQuery con la siguiente instrucción:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js">
```

De esta forma, siempre nos estaremos descargando la versión más actualizada de la librería. El único inconveniente, es que necesitamos estar conectados a Internet para que la librería pueda descargarse. En la siguiente página puedes encontrar distribuciones de JQuery: https://developers.google.com/speed/libraries?hl=es-419#jquery

Cargando la librería desde nuestro propio servidor:

```
<script type="text/javascript" src="jquery.js"></script>
```

De este modo, el archivo de la librería se guarda como parte integral de nuestra aplicación, lo que significa que no necesitaremos una conexión a Internet (si estamos trabajando localmente) para utilizar la librería. Para adoptar este enfoque, debemos descargar el

archivo de la librería desde la página de jQuery (jquery.com). Hay dos versiones disponibles para descarga: la versión de producción (comprimida para ocupar menos espacio) y la versión de desarrollo (sin comprimir). En general, se descargará la versión de producción, ya que ocupa menos espacio. La versión de desarrollo únicamente ofrece la ventaja de permitir una lectura más clara del código fuente de la librería, en caso de que estemos interesados en realizar modificaciones.

La clave principal para el uso de jQuery radica en el uso de la función **$()**, que es un alias de jQuery(). Esta función se podría comparar con el clásico document.getElementById(), pero con una diferencia muy importante, ya que soporta selectores CSS, y puede devolver arrays. Por lo tanto $() es una **versión mejorada de document.getElementById().**

La función $("selector") toma como parámetro una cadena de texto que representa un selector CSS. También puede aceptar un segundo parámetro que define el contexto en el que se realizará la búsqueda del selector mencionado. Otra aplicación de la función puede ser $(function){..}); equivalente a la instrucción $(document).ready (function() {...});, que nos permite detectar cuando el DOM ha sido completamente cargado.

A continuación vamos a ver el mismo ejemplo desarrollado en primer lugar mediante las técnicas habituales de JavaScript y a continuación mediante el uso de JQuery, de modo que puedas comparar. La función del script en este caso es modificar el estilo de las filas en una tabla para generar una tabla con colores alternos:

Ejemplo 8

Método sin JQuery

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Ejemplo sin jQuery</title>
<style>
   .colorido{
      background-color:#99FF33;
   }
</style>
<script>
document.addEventListener('DOMContentLoaded',function(){
   let tabla = document.getElementById("mitabla"); // Seleccionamos la tabla.
   let filas= tabla.getElementsByTagName("tr"); // Seleccionamos las filas de la tabla.
   for (var i=0; i<filas.length; i++){
```

```
    if (i%2 == 1){// Es una fila impar aplicamos la clase .colorido a esas filas.
      filas[i].setAttribute('class','colorido');
    }
  }
});
</script>
</head>
<body>
<table width="200" border="1" align="center" id="mitabla">
  <tr>
    <td><div align="center"><strong>País</strong></div></td>
    <td><div align="center"><strong>Habitantes</strong></div></td>
    <td><div align="center"><strong>Renta</strong></div></td>
  </tr>
  <tr>
    <td>España</td>
    <td>15600000</td>
    <td>25000</td>
  </tr>
  <tr>
    <td>Italia</td>
    <td>45105500</td>
    <td>45000</td>
  </tr>
  <tr>
    <td>Francia</td>
    <td>58454545</td>
    <td>45645</td>
  </tr>
  <tr>
    <td>Uk</td>
    <td>78799788</td>
    <td>88547</td>
  </tr>
  <tr>
    <td>USA</td>
    <td>98878787</td>
    <td>45124</td>
  </tr>
</table>
</body>
```

```
</html>
```

Método con JQuery

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Ejemplo con jQuery</title>
<style>
   .colorido{
      background-color:#99FF33;
   }
</style>
<!--Primero debemos incluir la librería en nuestro código-->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
<script>
   // Cuando el documento esté preparado se ejecuta esta función:
   $(document).ready(function(){
         // Seleccionamos las filas impares contenidas dentro de mitabla y le aplicamos la clase colorido.
      $("#mitabla tr:nth-child(even)").addClass("colorido");
   });
</script>
</head>
<body>
<table width="200" border="1" align="center" id="mitabla">
   <tr>
      <td><div align="center"><strong>País</strong></div></td>
      <td><div align="center"><strong>Habitantes</strong></div></td>
      <td><div align="center"><strong>Renta</strong></div></td>
   </tr>
   <tr>
      <td>España</td>
      <td>15600000</td>
      <td>25000</td>
   </tr>
   <tr>
      <td>Italia</td>
      <td>45105500</td>
      <td>45000</td>
   </tr>
```

```
  <tr>
    <td>Francia</td>
    <td>58454545</td>
    <td>45645</td>
  </tr>
  <tr>
    <td>Uk</td>
    <td>78799788</td>
    <td>88547</td>
  </tr>
  <tr>
    <td>USA</td>
    <td>98878787</td>
    <td>45124</td>
  </tr>
</table>
</body>
</html>
```

3.2. Función $.ajax() en JQuery

La función principal para realizar peticiones AJAX en jQuery es **$.ajax()** (es importante no olvidar el punto entre $ y ajax()). Esta función se considera de bajo nivel, lo que significa que proporciona la capacidad de configurar prácticamente todos los parámetros de la petición AJAX. En consecuencia, es equivalente a los métodos clásicos utilizados en la programación tradicional.

La sintaxis básica de uso es la siguiente: **$.ajax(opciones)**. Aunque la instrucción parece simple inicialmente, el número de opciones disponibles es relativamente extenso. La estructura básica es la siguiente:

```
$.AJAX({
    url: [URL],
    type: [GET/POST],
    success: [function callback exito(data)],
    error: [function callback error],
    complete: [function callback error],
    ifModified: [bool comprobar E-Tag],
    data: [mapa datos GET/POST],
    async: [bool que indica sincronía/asincronia]
});
```

Por ejemplo:

```
$.AJAX({
   url: '/ruta/pagina.php',
   type: 'POST',
   async: true,
   data: 'parametro1=valor1&parametro2=valor2',
   success: function (respuesta){
   alert(respuesta);
   },
   error: mostrarError
});
```

Veamos algunas propiedades de la función $.AJAX() de jQuery mediante la siguiente tabla:

Nombre	Tipo	Descripción
url	String	La URL a la que se le hace la petición AJAX.
type	String	El método HTTP a utilizar para enviar los datos: POST o GET. Si se omite se usa GET por defecto.
data	Object	Un objeto en el que se especifican parámetros que se enviarán en la solicitud. Si es de tipo GET, los parámetros irán en la URL. Si es POST, los datos se enviarán en las cabeceras. Es muy útil usar la función serialize(), para construir la cadena de datos.
dataType	String	Indica el tipo de datos que se espera que se devuelvan en la respuesta: XML, HTML, JSON, JSONp, script, text (valor por defecto en el caso de omitir dataType).
success	Function	Función que será llamada, si la respuesta a la solicitud terminó con éxito.
error	Function	Función que será llamada, si la respuesta a la solicitud devolvió algún tipo de error.
complete	Function	Función que será llamada, cuando la solicitud fue completada.

3.3. El método .load() y las funciones $.post(), $.get() y $.getJSON() en JQuery

La función $.ajax() es bastante completa y puede resultar algo compleja de utilizar. Se recomienda su uso en casos específicos en los que sea necesario tener un control exhaustivo de la petición AJAX. Para facilitar el trabajo, se crearon tres funciones adicionales de alto nivel que permiten realizar peticiones y gestionar las respuestas obtenidas del servidor:

El método load()

Este método, es la forma más sencilla de obtener datos desde el servidor, ya que de forma predeterminada, los datos obtenidos son cargados en el objeto al cuál le estamos aplicando el método.

Su sintaxis es: .load(url, [datos], [callback])

La función callback es opcional, y es ahí donde pondremos la función de retorno, que será llamada una vez terminada la petición. En esa función realizaremos tareas adicionales, ya que la acción por defecto de cargar en un objeto el contenido devuelto en la petición, la realiza el propio método load(). Cuando se envían datos en este método, se usará el método POST. Si no se envían datos en la petición, se usará el método GET

Ejemplo:

```
$("#noticias").load("feeds.HTML");
// carga en el contenedor con id noticias lo que devuelve la página feeds.HTML.
$("#objectID").load("test.php", { 'personas[]': ["Juan", "Susana"] } );
// Pasa un array de datos al servidor con el nombre de dos personas.
```

La función $.post()

Nos permite realizar peticiones AJAX al servidor, empleando el método POST.

Su sintaxis es la siguiente: **$.post(url, [datos], [callback], [tipo])**

Ejemplo:

```
$.post("test.php");
$.post("test.php", { nombre: "Juana", hora: "11 am" } );
$.post("test.php", function(resultados) {
alert("Datos Cargados: " + resultados);
});
```

Las funciones $.get() y $.getJSON()

Hacen prácticamente lo mismo que POST, y tienen los mismos parámetros, pero usan el método GET para enviar los datos al servidor. Si recibimos los datos en formato JSON, podemos emplear $.getJSON() en su lugar. Su sintaxis, en cada caso es como se indica a continuación:

$.get(url, [datos], [callback], [tipo])

$.getJSON(url, [datos], [callback], [tipo])

3.4. Herramientas adicionales en programación AJAX

Cuando trabajamos con AJAX, uno de los desafíos comunes es la detección de errores. Estos errores pueden originarse por fallos en la programación JavaScript, problemas en la aplicación ejecutándose en el servidor, entre otros.

Para identificar estos errores, necesitamos herramientas que faciliten su localización. En la programación JavaScript, los errores pueden ser detectados a través del propio navegador. Por ejemplo, en Firefox, podemos abrir la consola de errores desde el menú Herramientas o mediante las teclas CTRL + Mayúsc. + J (en Windows). La consola mostrará todos los errores encontrados durante la ejecución de la aplicación. En Internet Explorer versión 9, podemos abrir la Herramienta de Desarrollo con la tecla F12, desde donde se pueden consultar los errores de JavaScript, activar diferentes modos de compatibilidad, deshabilitar CSS, JavaScript, etc.

Sin embargo, para la detección de errores en las solicitudes AJAX, necesitamos herramientas adicionales o complementos. En Firefox, el complemento Firebug es útil. Proporciona diversas funcionalidades, como la detección de errores en JavaScript, depuración de código, análisis detallado del DOM, visualización y modificación del código CSS, análisis de la velocidad de carga de las páginas, entre otras. Además, Firebug incluye en su consola la capacidad de visualizar las peticiones AJAX realizadas al servidor, mostrando datos enviados, formato, datos recibidos, errores, etc. Así, si se produce algún error en la petición al servidor, podremos identificarlo en la consola y abordar la solución correspondiente.

3.5. Plugins en JQuery

Cabe destacar que, ya de por sí la librería jQuery ofrece funciones que son de gran utilidad en el desarrollo de nuestras aplicaciones. Además de las capacidades intrínsecas de la librería, cuenta con plugins o complementos que proporcionan funcionalidades avanzadas.

Estos plugins abarcan diversas categorías, como AJAX, animación y efectos, DOM, eventos, formularios, integración, medios, navegación, tablas, utilidades, entre otras. Antes de emplear cualquier plugin de jQuery, es necesario cargar primero la librería jQuery y, posteriormente, la librería específica del plugin deseado, tal como se carga la librería en las etiquetas de Script. Aquí te dejo la web de oficial de los plugins en JQuery:

https://plugins.jquery.com/

4. Bibliografía y Webgrafía

[1] A. Garro, "JavaScript", *Arkaitzgarro.com*, 01-ago-2014. [En línea]. Disponible en: https://www.arkaitzgarro.com/javascript/index.html [Consultado: 12-nov-2023].

[2] "Introducción a JavaScript", Uniwebsidad.com. [En línea]. Disponible en: https://uniwebsidad.com/libros/javascript [Consultado: 14-nov-2023].