Programación AJAX en JavaScript.

Caso práctico

En estos últimos meses, **Antonio** ha realizado un montón de trabajos en el proyecto, y prácticamente ha terminado todas las tareas. **Juan** le dice que todo el trabajo realizado está muy bien, pero que tendría que actualizar algunos de los procesos para darle un toque de modernidad a la aplicación.

Entre las mejoras que le recomienda **Juan**, están la de utilizar efectos, más dinamismo, usar AJAX (JavaScript Asíncrono y XML) en las validaciones de los formularios, o en cierto tipo de consultas, etc. El término AJAX le suena muy complicado a **Antonio**, pero **Juan** lo convence rápidamente para que intente hacerlo, ya que no necesita aprender ningún lenguaje nuevo. Utilizando un nuevo objeto de JavaScript, con sus propiedades y métodos va a poder emplear AJAX en sus aplicaciones actuales.

Además, **Juan** lo anima a que se ponga a estudiar rápido el tema de AJAX, ya que al final, le va a dar una pequeña sorpresa, con una librería que le va a facilitar enormemente el programar con AJAX y conseguir dar buenos efectos y mayor dinamismo al proyecto web. Esa librería gratuita tiene el respaldo de grandes compañías a nivel mundial, que la están utilizando actualmente. También cuenta con infinidad de complementos, para que pueda modernizar su web todo lo que quiera, y todo ello programando muy pocas líneas de código y en un tiempo de desarrollo relativamente corto.

Las **conexiones síncronas** (sincronizadas):
- son aquellas en las que el emisor y el receptor establecen un tipo de comunicación "organizada", en la que por cada solicitud enviada, se espera una respuesta (durante un tiempo máximo o se reenvía de nuevo la petición), y así sucesivamente.

En cambio, en las **conexiones asíncronas**, ésto no es necesariamente así: puede ocurrir que se envíen peticiones sin tener que esperar respuestas y recibir esas respuestas en diferente orden.

En esta unidad de trabajo, se hace una introducción a la **tecnología AJAX**, analizando los requerimientos previos necesarios para su utilización.
- verás lo que es una comunicación asíncrona,
- analizamos el API que nos va a permitir programar con AJAX, empleando el **objeto XMLHttpRequest** de JavaScript.

En la segunda mitad, profundizaremos en cada uno de los **estados de la petición AJAX asíncrona**, así como en los diferentes métodos, de envío de datos y de recepción (texto, XML, JSON).

Por último, en el apartado 3, veremos librerías que nos van a facilitar muchísimo la programación con AJAX, ya que no tendremos que preocuparnos de incompatibilidades entre navegadores, etc. Haremos una introducción a la librería jQuery, y a los métodos utilizados para trabajar con AJAX, con ejemplos en video, a modo aclaratorio.

1.- Introducción a AJAX

Caso práctico

AJAX es una tecnología crucial en lo que se conoce como web 2.0. A **Antonio** le atrae mucho el tema, ya que ha visto que con AJAX se pueden hacer envíos y consultas al servidor, sin tener que recargar las páginas web o cambiar de página, con lo que se consigue que sea todo más interactivo y adaptado a las nuevas tendencias. **Antonio** analiza la tecnología AJAX, sus orígenes, y el objeto que se utiliza para realizar las peticiones al servidor y gestionar las respuestas. Su directora **Ada**, le facilita unas direcciones muy interesantes con contenidos y ejemplos de algunas aplicaciones AJAX de antiguos proyectos realizados en la empresa.

El término **AJAX** (**JavaScript Asíncrono y XML**)

- es una técnica de desarrollo web, que permite comunicar el navegador del usuario con el servidor, en un segundo plano.
- de esta forma, se podrían realizar peticiones al servidor sin tener que **recargar la página**, y podríamos gestionar esas respuestas, que nos permitirían actualizar los contenidos de nuestra página, sin tener que realizar recargas.

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

AJAX no es una tecnología nueva: son realmente muchas tecnologías, cada una destacando por su propio mérito, pero que se unen con los siguientes objetivos:

- conseguir una presentación basada en estándares, usando HTML, CSS y un uso amplio de técnicas del DOM, para poder mostrar la información de forma dinámica e interactiva.
- intercambio y manipulación de datos, usando **XML** y **XSLT**, **JSON**, etc.
- recuperación de datos **de forma asíncrona**, usando el **objeto XMLHTTPRequest**.
- uso de JavaScript, para unir todos los componentes.

Las tecnologías que forman AJAX son:

- HTML 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 unir todos los componentes anteriores.

El modelo clásico de aplicaciones Web funciona de la siguiente forma:

- la mayoría de las acciones del usuario se producen en la interfaz, disparando solicitudes HTTP al servidor web.
- el servidor efectúa un proceso (recopila información, realiza las acciones oportunas), y devuelve una pagina HTML al cliente.
- este es un modelo adaptado del uso original de la Web como medio hipertextual, pero a nivel de aplicaciones de software, este tipo de modelo no es necesariamente el más recomendable.

Cada vez que se realiza una petición al servidor:

- el usuario lo único que puede hacer es esperar, ya que muchas veces la página cambia a otra diferente, y hasta que no reciba todos los datos del servidor, no se mostrará el resultado, con lo que el usuario no podrá interactuar de ninguna manera con el navegador.

Con AJAX, lo que se intenta evitar, son esencialmente esas esperas:

- el cliente podrá hacer solicitudes al servidor, mientras el navegador sigue mostrando la misma página web, y cuando el navegador reciba una respuesta del servidor, la mostrará al cliente y todo ello sin recargar o cambiar de página.

AJAX es utilizado por muchas empresas y productos hoy en día: por ejemplo, Google utiliza AJAX en aplicaciones como Gmail, Google Maps.., así como Flickr, Amazon, etc.

Son muchas las razones para usar AJAX:

- está basado en estándares abiertos.
- su usabilidad.
- válido en cualquier plataforma y navegador.
- beneficios que aporta a las aplicaciones web.
- es la base de la web 2.0.
- es independiente del tipo de tecnología de servidor utilizada.
- mejora la estética de la web.

1.1.- Requerimientos previos

A la hora de trabajar con AJAX debemos tener en cuenta una serie de requisitos previos, necesarios para la programación con esta metodología.

Hasta este momento:

- nuestras aplicaciones de JavaScript no necesitaban de un servidor web para funcionar, salvo en el caso de querer enviar los datos de un formulario y almacenarlos en una base de datos.
- es más, todas las aplicaciones de JavaScript que has realizado, las has probado directamente abriéndolas con el navegador o haciendo doble click sobre el fichero .html.

Para la programación con AJAX vamos a necesitar de un servidor web, ya que las peticiones AJAX que hagamos, las haremos a un servidor; los componentes que necesitamos son:

- servidor web (apache, ligHTTPd, etc).
- servidor de bases de datos (MySQL, MariaDB, Postgresql, etc).
- lenguaje de servidor (PHP, ASP, etc).

Debes conocer

Podríamos instalar cada uno de esos componentes por separado, pero muchas veces lo más cómodo es instalar alguna aplicación que los agrupe a todos sin instalarlos de forma individual.

Hay varios tipos de aplicaciones de ese tipo, que se pueden categorizar en dos, diferenciadas por el tipo de sistema operativo sobre el que funcionan:

- servidor LAMP (Linux, Apache, MariaDB y PHP).
- servidor WAMP (Windows, Apache, MariaDB y PHP).

Una aplicación de este tipo, muy utilizada, puede ser XAMPP (tanto para Windows, como para Linux).

- esta aplicación podrás instalarla incluso en una memoria USB y ejecutarla en cualquier ordenador, con lo que tendrás siempre disponible un servidor web, para programar tus aplicaciones AJAX.
- su instalación es muy sencilla, simplemente tendrás que descargar el paquete para tu Sistema Operativo y ejecutarlo.

Servidor XAMPP.

1.2.- Comunicación asíncrona

Como ya te comentábamos en la introducción a AJAX, la mayoría de las aplicaciones web funcionan de la siguiente forma:

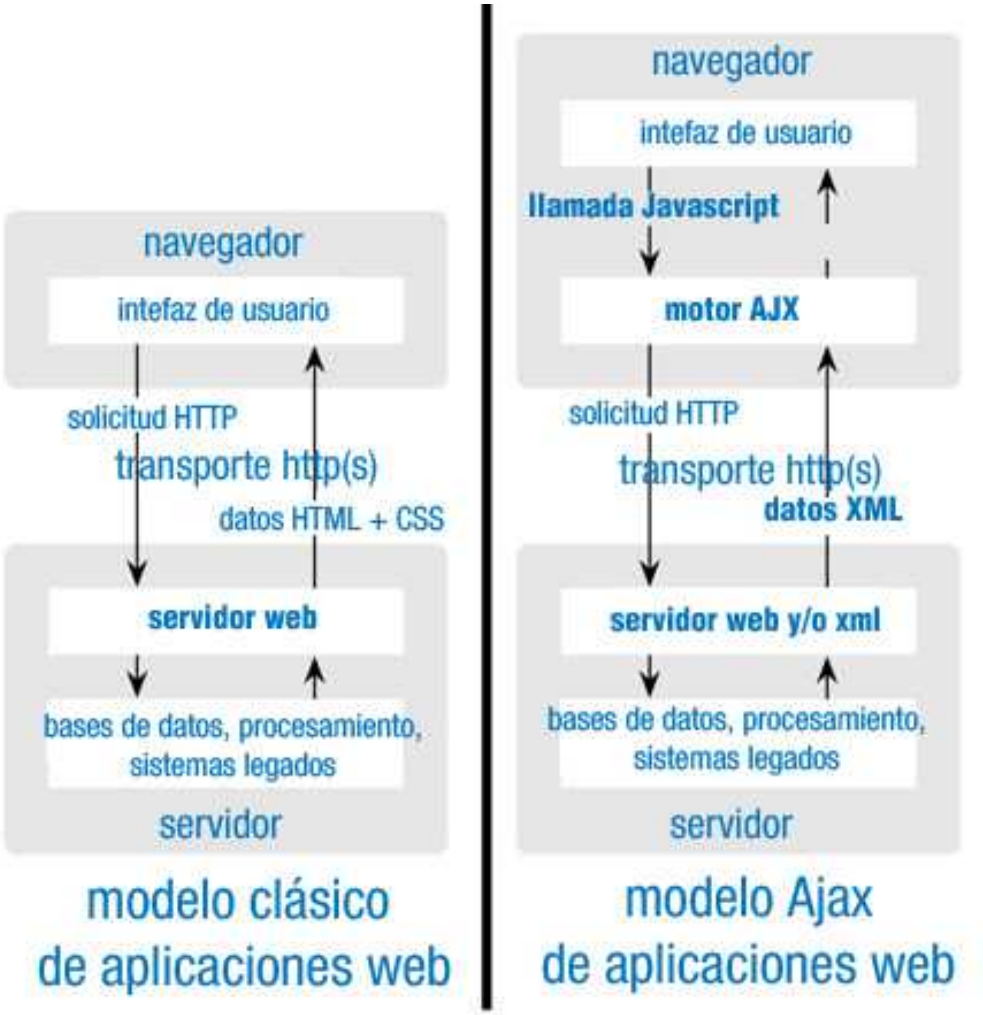
1. El usuario solicita algo al servidor.
2. El servidor ejecuta los procesos solicitados (búsqueda de información, consulta a una base de datos, lectura de fichero, cálculos numéricos, etc.).
3. Cuando el servidor termina, devuelve los resultados al cliente.

En el paso 2, mientras se ejecutan los procesos en el servidor, el cliente lo único que puede hacer es esperar, ya que el navegador está bloqueado en espera de recibir la información con los resultados del servidor.

Una **aplicación AJAX**, cambia la metodología de funcionamiento de una aplicación web, en el sentido de que, elimina las esperas y los **bloqueos** que se producen en el cliente.

- es decir, el usuario podrá seguir interactuando con la página web, mientras se realiza la petición al servidor.
- en el momento de tener una respuesta confirmada del servidor, ésta será mostrada al cliente, o bien se ejecutarán las acciones que el programador de la página web haya definido.

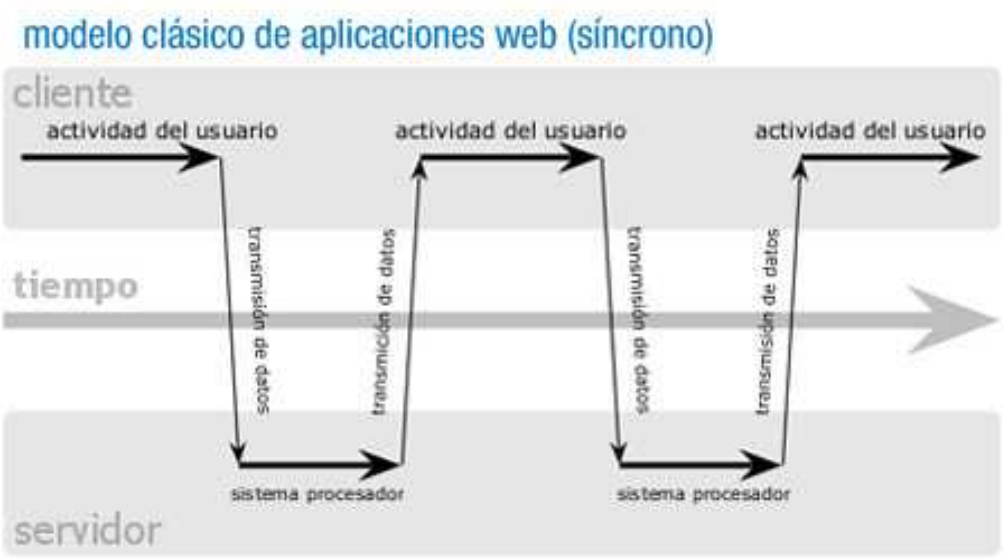
Mira el siguiente gráfico, en el que se comparan los dos modelos de aplicaciones web:

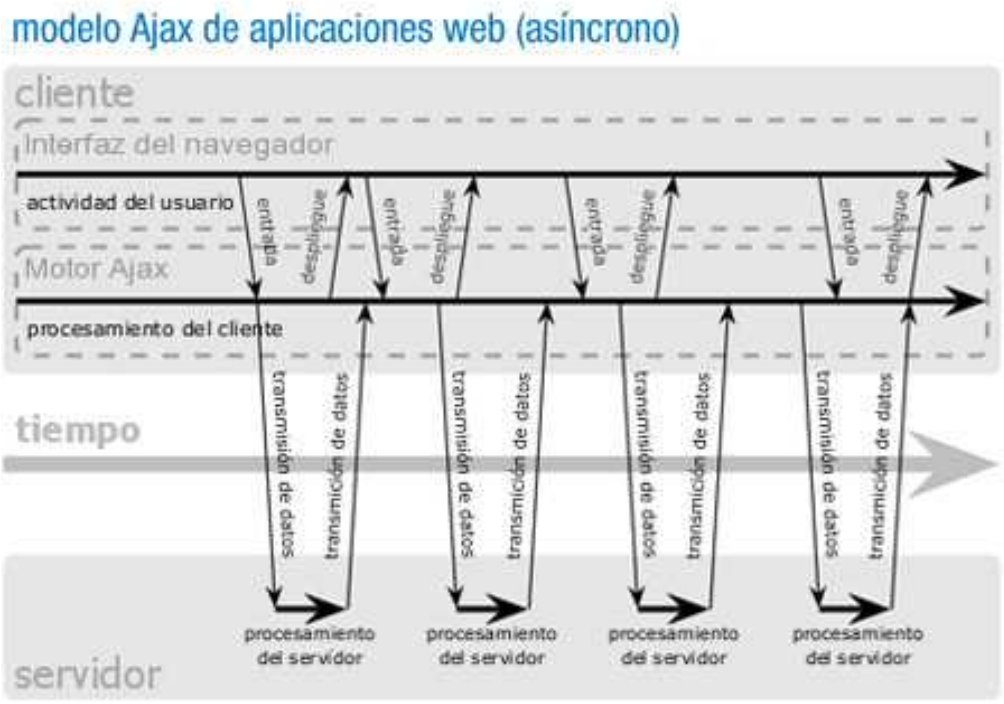


¿Cómo se consigue realizar la petición al servidor sin bloquear el navegador?

Para poder realizar las peticiones al servidor sin que el navegador se quede bloqueado, tendremos que hacer uso del **motor AJAX** (programado en JavaScript).

- este motor se encarga de gestionar las **peticiones AJAX** del usuario, y de comunicarse con el servidor.
- es justamente este motor, el que permite que la interacción suceda de forma asíncrona (independientemente de la comunicación con el servidor).
- así, de esta forma, el usuario no tendrá que estar pendiente del icono de indicador de carga del navegador, o viendo una pantalla en blanco.





Cada acción del usuario, que normalmente generaría una **petición HTTP** al servidor, se va a convertir en una **petición AJAX** con esa solicitud, y será este motor, el que se encargará de todo el proceso de comunicación y obtención de datos de forma asíncrona con el servidor, y todo ello sin frenar la interacción del usuario con la aplicación.

Autoevaluación

Pregunta

¿Según los gráficos anteriores, en qué modelo de aplicación web la actividad del usuario se ve interrumpida o bloqueada por la espera de las respuestas del servidor?

Respuestas

- ○ Clásico.
- ○ AJAX

.

1.3.- El API XMLHttpRequest

El corazón de AJAX

- es una API denominada **XMLHTTPRequest** (**XHR**): disponible en los lenguajes de scripting en el lado del cliente, tales como JavaScript.
- se utiliza para
  - realizar **peticiones**, HTTP o HTTPS, directamente al servidor web,
  - para cargar las **respuestas** directamente en la página del cliente.
- los datos que recibamos desde el servidor se podrán recibir en forma de
  - **texto plano**
  - **texto XML**.
- estos datos, podrán ser utilizados para
  - modificar el DOM del documento actual, sin tener que recargar la página,
  - ser evaluados con JavaScript, si son recibidos en formato JSON.

XMLHTTPRequest juega un papel muy importante en la técnica AJAX, ya que sin este objeto, no sería posible realizar las **peticiones asíncronas** al servidor.

- el concepto que está detrás del objeto XMLHTTPRequest, surgió gracias a los desarrolladores de Outlook Web Access (de Microsoft), en su desarrollo de Microsoft Exchange Server 2000.
  - la interfaz IXMLHTTPRequest, se desarrolló e implementó en la segunda versión de la librería MSXML, empleando este concepto.
  - con el navegador Internet Explorer 5.0 en Marzo de 1999, se permitió el acceso a dicha interfaz a través de ActiveX.
- posteriormente la fundación Mozilla, desarrolló e implementó una interfaz llamada nsIXMLHTTPRequest, dentro de su motor Gecko.
  - esa interfaz, se desarrolló adaptándose lo más posible a la interfaz implementada por Microsoft.
  - Mozilla creó un envoltorio para usar esa interfaz, a través de un objeto JavaScript, el cuál denominó XMLHTTPRequest.
  - el objeto XMLHTTPRequest fue accesible en la versión 0.6 de Gecko, en diciembre de 2000, pero no fue completamente funcional, hasta Junio de 2002 con la versión 1.0 de Gecko.
  - el objeto XMLHTTPRequest, se convirtió de hecho en un estándar entre múltiples navegadores, como Safari 1.2, Konqueror, Opera 8.0 e iCab 3.0b352 en el año 2005.
- el W3C publicó una especificación-borrador para el objeto XMLHTTPRequest, el 5 de Abril de 2006.
  - su objetivo era crear un documento con las especificaciones mínimas de interoperabilidad, basadas en las diferentes implementaciones que había hasta ese momento.
  - la última revisión de este objeto, se realizó en Noviembre de 2009.
  - Microsoft añadió el objeto XMLHTTPRequest a su lenguaje de script, con la versión de Internet Explorer 7.0 en Octubre de 2006.
- con la llegada de las **librerías cross-browser** como **jQuery**, **Prototype**, etc, los programadores pueden utilizar toda la funcionalidad de XMLHTTPRequest, sin codificar directamente sobre la API, con lo que se acelera muchísimo el desarrollo de aplicaciones AJAX.
- en febrero de 2008, la W3C publicó otro borrador denominado "XMLHTTPRequest Nivel 2".
  - este nivel consiste en extender la funcionalidad del objeto XMLHTTPRequest, incluyendo, pero no limitando, el soporte para
    - **peticiones cross-site**
    - gestión de **byte streams**
    - progreso de eventos
    - etc.
  - esta última revisión de la especificación, sigue estando en estado "working draft" (borrador), a septiembre de 2010.

librerías cross-browser: se refiere a la capacidad que una web, aplicación web, construcción HTML o script del lado del cliente tiene y que permite que sea soportada por todos los navegadores, es decir que se pueda mostrar o ejecutar de forma correcta en cualquier navegador.

peticiones cross-site: se refiere a peticiones situadas en diferentes dominios.

1.3.1.- Creación del objeto XMLHttpRequest

Para poder programar con AJAX

- necesitamos crear un objeto del tipo **`XMLHttpRequest`**, que va a ser el que nos permitirá realizar las **peticiones en segundo plano** al servidor web.
- para ello primero crearás en la carpeta pública de tu servidor el archivo **`fecha.php`** que simplemente devuelve la fecha del servidor pasados 2 segundos.
- su contenido sería el siguiente:

```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.");
?>
```

Ahora crearemos el siguiente archivo:

```html
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>DWEC07 .- Ejemplo petición AJAX</title>
  <script type="text/javascript">
   let peticion;
   const iniciar = () => {
    peticion = new XMLHttpRequest();
    peticion.open('GET', "http://localhost/fecha.php");
    peticion.send();
    peticion.addEventListener("load", cargada);
    document.getElementById("estado").classList = [ 'cargando' ];
    document.getElementById("estado").innerText = "Cargando...";
   }

   const cargada = () => {
    document.getElementById("resultados").innerText = peticion.responseText;
    document.getElementById("estado").classList = [ 'cargada' ];
    document.getElementById("estado").innerText = "Cargada.";
   }

   window.addEventListener("load", iniciar, false);
  </script>
  <style>
   #resultados {
    background: yellow;
   }
   .cargando {
    background: red;
   }
   .cargada {
    background: green;
   }
  </style>
 </head>
```

```
  <body>
    <p>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:
    </p>
    <p>
      Esta solicitud tardará 2 segundos aproximadamente,
      que es el tiempo de ejecución de la página PHP en el servidor
    </p>
    <p>
      Contenedor resultados:
      <div id="resultados"></div>
    </p>
    <p>
      Estado de las solicitudes:
      <div id="estado"></div>
    </p>
  </body>
</html>
```

Si abres esta archivo en tu navegador podrás comprobar que te muestra la página, te indica que está cargando la petición AJAX y que cuando ésta termina te muestra su resultado y te indica que ya ha se ha cargado. Te animo a que la pruebes.

Puedes observar que

- primero creamos un objeto **`XMLHttpRequest`** mediante su constructor.
- mediante el método **`open`** indicamos la petición
- mediante el método **`send`** la enviamos.
- registramos el **evento de la petición** **`load`** que llamará a la función cargada cuando haya terminado la carga de nuestra petición.
- luego mediante la propiedad **`responseText`** de la petición accedemos a la respuesta de dicha petición.

1.3.2.- Métodos del objeto XMLHttpRequest

El objeto XMLHTTPRequest dispone de los siguientes métodos, que nos permitirán realizar **peticiones asíncronas** al servidor:

Métodos del objeto XMLHttpRequest.

Metodo	Descripción
`abort()`	Cancela la solicitud actual.
`getAllResponseHeaders()`	Devuelve la información completa de la cabecera.
`getResponseHeader()`	Devuelve la información específica de la cabecera.
`open``(`*`metodo, 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. • *`metodo`*: 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 el servidor.
`send``(`*`datos`*`)`	• *`datos`*: se usa en el caso de que estemos utilizando el método POST, como método de envío. Si usamos GET, lo dejaremos vacío.
`setRequestHeader()`	Añade el par etiqueta/valor a la cabecera de datos que se enviará al servidor.

1.3.3.- Propiedades del objeto XMLHttpRequest

El objeto `XMLHTTPRequest`, dispone de las siguientes propiedades, que nos facilitan información sobre el estado de la petición al servidor, y donde recibiremos los datos de la respuesta devuelta en la petición AJAX:

Propiedades del objeto XMLHttpRequest.

Propiedad	Descripción
`onreadystatechange`	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

Caso práctico

Ahora que **Antonio** ya conoce el objeto XMLHTTPRequest, con sus propiedades y métodos, se centra en cómo se realiza la petición al servidor de forma asíncrona, y cómo se gestionan los estados y las respuestas que nos devuelve. También estudia qué formatos tiene para enviar datos al servidor, y en qué formatos va a recibir esos resultados.

De entre todos los formatos de recepción de datos, **Juan** recomienda a **Antonio** uno de ellos: el formato JSON. Dicho formato, utiliza la nomenclatura de JavaScript, para enviar los resultados. De esta forma puede utilizar dichos resultados directamente en la aplicación JavaScript, sin tener que realizar prácticamente ningún tipo de conversiones intermedias.

En el apartado anterior, hemos visto un ejemplo sencillo de una petición asíncrona al servidor.
En este apartado nos centraremos en ver cómo podemos realizar diferentes tipos de peticiones y cómo gestionar los datos devueltos por las mismas.

Autoevaluación

Pregunta 1

En JavaScript una petición AJAX no puede ser síncrona. ¿Verdadero o falso?

- ○ Verdadero
- ○ Falso

2.1.- Envío de datos usando método GET

Hemos implementado el siguiente archivo PHP que simplemente muestra los datos recibidos -bien sea por una petición GET o POST- (nombre y apellidos) y la hora del servidor.

- este archivo llamado **`procesar.php`** deberás situarlo en la carpeta htdocs de tu servidor ya que será al que realicemos la petición.

```php
<?php
    // Imprimimos un mensaje con los textos recibidos
    if (isset($_GET['nombre']))
        echo "<p>Saludos desde el servidor: hola {$_GET['nombre']} {$_GET['apellidos']}.</p>";
    else if (isset($_POST['nombre']))
        echo "Saludos desde el servidor: hola {$_POST['nombre']} {$_POST['apellidos']}. ";

    // Mostramos la fecha y hora del servidor web.
    echo "<p>La fecha y hora del Servidor Web: ";
    echo date("j/n/Y G:i:s");
    echo "</p>";
?>
```

En la **petición GET**, los parámetros que pasemos en la solicitud, se enviarán formando parte de la URL.

- por ejemplo: ***`http://localhost/procesar.php?nombre=Bob&apellidos=Esponja`***.
- cuando realizamos la petición por el método GET, llamamos al método **`send`**, sin argumentos, ya que los datos son enviados a la página **`procesar.php`**, formando parte de la URL: **send()**.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>DWEC07.- Ejemplo método GET</title>
    <script type="text/javascript">
      let peticion;
```

```
    const iniciar = () => {
        peticion = new XMLHttpRequest();
        peticion.open('GET', "http://127.0.0.1/procesar.php?nombre=Bob&apellidos=Esponja");
        peticion.send();
        peticion.addEventListener("load", cargada);
    }

    const cargada = () => {
        document.getElementById("resultados").innerHTML = peticion.responseText;
    }

    window.addEventListener("load", iniciar, false);
  </script>
  <style>
    #resultados{
        background: yellow;
    }
  </style>
 </head>
 <body>
  <p>
    A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:
  </p>
  <p>
    Contenedor resultados:
    <div id="resultados"></div>
  </p>
 </body>
</html>
```

2.2.- Envío de datos usando método POST

Vamos a ver un ejemplo en el que se realiza una petición AJAX, a la página procesar.php pasando dos párametros: nombre y apellidos, usando el método POST.

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>DWEC07.- Ejemplo método POST</title>
    <script type="text/javascript">
      let peticion;
      const iniciar = () => {
          peticion = new XMLHttpRequest();
          peticion.open('POST', "http://127.0.0.1/procesar.php");
          peticion.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
          peticion.send('nombre=Bob&apellidos=Esponja');
          peticion.addEventListener("load", cargada);
      }

      const cargada = () => {
          document.getElementById("resultados").innerHTML = peticion.responseText;
      }

      window.addEventListener("load", iniciar, false);
    </script>
    <style>
      #resultados{
          background: yellow;
      }
    </style>
  </head>
  <body>
    <p>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:
    </p>
    <p>
      Contenedor resultados:
      <div id="resultados"></div>
    </p>
  </body>
</html>
```

En este ejemplo, tuvimos que realizar los siguientes cambios para adaptarlo al método POST:

- el método `open` se modifica para indicar que la **petición es tipo POST** y simplemente indicamos la URL y no los parámetros: `peticion.open('POST', "http://localhost/procesar.php");`
- tenemos que crear una **cabecera** con el tipo de contenido que vamos a enviar, justo antes de enviar la petición con el método send:
  `peticion.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");`
- en el método send pasamos los parámetros (nombre=Teresa&apellidos=Blanco Ferreiro) que serán enviados por el método POST: `peticion.send('nombre=Bob&apellidos=Esponja');`

2.3.- Recepción de datos en formato XML

Cuando realizamos una petición AJAX, que nos devuelve la respuesta en **formato XML**, dichos datos los tendremos que consultar en la propiedad **`responseXML`** del objeto XMLHttpRequest.

Lo primero que hemos creado es un archivo **`datosxml.php`** que devuelve un conjunto de datos sobre nuestros CDs en formato XML.

- como siempre, este fichero deberás localizarlo en la carpeta htdocs de servidor ya que será al que le haremos la petición.

```php
<?php
// Leemos el contenido del fichero XML
// e imprimimos su contenido.
// Muy importante indicar al navegador que va a recibir contenido XML
// eso lo hacemos con la siguiente cabecera:
header("Content-Type: text/xml");

$ficheroxml = "<?xml version=\"1.0\" encoding=\"utf-8\"?>";
$ficheroxml .= "
<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>
</CATALOG>";

echo $ficheroxml;
?>
```

También hemos creado un fichero para procesar una petición asíncrona para estos datos en XML. El contenido del fichero es el siguiente.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>DWEC07.- Ejemplo petición XML</title>
  <script type="text/javascript">
    let peticion;
    const iniciar = () => {
      peticion = new XMLHttpRequest();
      peticion.open('GET', "http://localhost/datosxml.php");
      peticion.send();
      peticion.addEventListener("load", cargada);
    }

    const cargada = () => {
      let resultados = '';
      let cds = peticion.responseXML.documentElement.getElementsByTagName("CD");
      resultados += '<ul>';
      for (let i = 0; i < cds.length; i++) {
        resultados += `<li><b>CD nº ${i + 1}</b><ul>`;
        resultados += `<li><b>Título:</b> ${cds[i].getElementsByTagName('TITLE')[0].innerHTML}</li>`;
        resultados += `<li><b>Artista:</b> ${cds[i].getElementsByTagName('ARTIST')[0].innerHTML}</li>`;
        resultados += `<li><b>País:</b> ${cds[i].getElementsByTagName('COUNTRY')[0].innerHTML}</li>`;
        resultados += `<li><b>Compania:</b> ${cds[i].getElementsByTagName('COMPANY')[0].innerHTML}</li>`;
        resultados += `<li><b>Precio:</b> ${cds[i].getElementsByTagName('PRICE')[0].innerHTML}</li>`;
        resultados += `<li><b>Año:</b> ${cds[i].getElementsByTagName('YEAR')[0].innerHTML}</li>`;
        resultados += `</li></ul>`;
      }
      resultados += '</ul>';

      document.getElementById("resultados").innerHTML = resultados;
    }

    window.addEventListener("load", iniciar, false);
  </script>
  <style>
    #resultados{
        background: yellow;
    }
  </style>
</head>
<body>
  <p>
    A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:
  </p>
  <p>
    Contenedor resultados:
    <div id="resultados"></div>
  </p>
</body>
</html>
```

En la función **iniciar()**, le hemos dicho que cargue de forma asíncrona, empleando el método GET, el fichero localizado en la URL: **http://localhost/datosxml.php**.

- cuando se carga la petición, nos quedamos con los datos XML mediante la propiedad **peticion.responseXML** la cual luego consultamos por sus elementos CD.
- Posteriormente, para cada elemento creamos una lista con cada uno de sus elementos: título, artista, país, etc.

2.4.- Recepción de datos en formato JSON

Otro formato de intercambio muy utilizado en AJAX, es el formato **JSON**.

- es un **formato de intercambio de datos**, alternativo a XML, mucho mas simple de leer, escribir e interpretar.
- significa **Javascript Object Notation**
- consiste en escribir los datos en formato de Javascript.

Para este ejemplo, lo primero que hemos hecho, otra vez, es crear un fichero `datosjson.php` que deberás localizar en la carpeta htdocs de tu servidor y que será al que realizaremos la petición.

- este fichero devolverá un array en JSON con la misma colección que en el apartado anterior, pero en dicho formato.
- su contenido será el siguiente:

```php
<?php
 // Cabecera para indicar que vamos a enviar datos JSON y que no haga caché de los datos.
 header('Content-Type: application/json');

 $datosjson = "[
  {
   \"title\": \"Empire Burlesque\",
   \"artist\": \"Bob Dylan\",
   \"country\": \"USA\",
   \"company\": \"Columbia\",
   \"price\": 10.90,
   \"year\": 1985
  },
  {
   \"title\": \"Hide your heart\",
   \"artist\": \"Bonnie Tyler\",
   \"country\": \"UK\",
   \"company\": \"CBS Records\",
   \"price\": 9.90,
   \"year\": 1988
  },
  {
   \"title\": \"Greatest Hits\",
   \"artist\": \"Dolly Parton\",
   \"country\": \"USA\",
   \"company\": \"RCA\",
   \"price\": 9.90,
   \"year\": 1982
  },
  {
   \"title\": \"Still got the blues\",
   \"artist\": \"Gary Moore\",
   \"country\": \"UK\",
   \"company\": \"Virgin records\",
   \"price\": 10.20,
   \"year\": 1990
  }
 ]";
 echo $datosjson;
?>
```

Al igual que antes, realizamos la petición mediante el método GET a la URL **`http://localhost/datosjson.xml`**.

- ahora nos quedamos con el texto de la respuesta, pero lo pasamos a JSON mediante la instrucción

`JSON.parse(peticion.responseText)`.

- eso nos devolverá un array de objetos que podremos recorrer.
- como puedes ver en el ejemplo que se adjunta, es mucho más sencillo.

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>DWEC07.- Ejemplo petición JSON</title>
    <script type="text/javascript">
      let peticion;
      const iniciar = () => {
        peticion = new XMLHttpRequest();
        peticion.open('GET', "http://localhost/datosjson.php");
        peticion.send();
        peticion.addEventListener("load", cargada);
      }

      const cargada = () => {
        let resultados = '';
        let cds = JSON.parse(peticion.responseText);
        resultados += '<ul>';
        for (let i = 0; i < cds.length; i++) {
            resultados += `<li><b>CD nº ${i + 1}</b><ul>`;
            resultados += `<li><b>Título:</b> ${cds[i].title}</li>`;
            resultados += `<li><b>Artista:</b> ${cds[i].artist}</li>`;
            resultados += `<li><b>País:</b> ${cds[i].country}</li>`;
            resultados += `<li><b>Compania:</b> ${cds[i].company}</li>`;
            resultados += `<li><b>Precio:</b> ${cds[i].price}</li>`;
            resultados += `<li><b>Año:</b> ${cds[i].year}</li>`;
            resultados += `</li></ul>`;
        }
        resultados += '</ul>';

        document.getElementById("resultados").innerHTML = resultados;
      }

      window.addEventListener("load", iniciar, false);
    </script>
    <style>
      #resultados{
          background: yellow;
      }
    </style>
  </head>
  <body>
    <p>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:
    </p>
    <p>
      Contenedor resultados:
      <div id="resultados"></div>
    </p>
```

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

3.- Librerías cross-browser para programación AJAX

Caso práctico

El estudio del objeto para trabajar con AJAX, está comenzando a dar sus frutos. Lo que más le fastidia a **Antonio**, es que necesita programar bastante código, y aunque puede crear alguna librería para acelerar la programación, también ve que las diferentes incompatibilidades, entre navegadores, no van a ayudar nada en esta labor. Por esta razón está un poco desilusionado, por que le va a suponer bastante trabajo, aunque los resultados merecen la pena.

En ese momento llega **Juan**, y le da la sorpresa que le había comentado hace unos días. Le facilita un pequeño tutorial sobre la librería jQuery, que le va a permitir hacer peticiones AJAX utilizando prácticamente una línea de código. No tendrá que preocuparse por temas de cross-browsing y, además, la misma librería le facilitará métodos para hacer todo tipo de efectos, animaciones, etc. Esta librería cuenta además con infinidad de complementos gratuitos, que permiten hacer prácticamente cualquier cosa en muy poco tiempo.

La programación con AJAX,

- es uno de los pilares de lo que se conoce como **web 2.0**, término que incluye a las aplicaciones web que facilitan el compartir información, la interoperabilidad, el diseño centrado en el usuario y la colaboración web.
- ejemplos de la web 2.0, pueden ser
  - las comunidades web
  - los servicios web
  - aplicaciones web
  - redes sociales
  - servicios de alojamiento de vídeos
  - wikis
  - blogs
  - mashup
  - etc.

Es una página web o aplicación que usa y combina datos, presentaciones y funcionalidad procedentes de una o más fuentes para crear nuevos servicios.

- el término implica integración fácil y rápida, usando a menudo **APIs abiertas** y **fuentes de datos** con el objetivo de producir resultados enriquecidos combinando diferentes fuentes.

Para saber más

En el siguiente enlace puedes ampliar información sobre Mashup.

Más información sobre Mashup (aplicaciones web híbridas).

Gracias a las **aplicaciones web 2.0**,

- se han desarrollado gran cantidad de utilidades/herramientas/frameworks para el desarrollo web con JavaScript, DHTML y AJAX.
- la gran ventaja de usar alguna **librería** o **framework** para AJAX, es la del ahorro de tiempo y código, en nuestras aplicaciones.
- veremos que con algunas librerías vamos a realizar **peticiones AJAX**, con una simple instrucción de código sin tener que preocuparnos de crear el objeto XMLHttpRequest, ni gestionar el código de respuesta del servidor, los **estados de la solicitud**, etc.
- otra de las ventajas que nos aportan este tipo de librerías, es la de la compatibilidad entre navegadores (cross-browser).
  - de esta forma tenemos un problema menos, ya que la propia librería será capaz de crear la petición AJAX de una forma u otra, dependiendo del navegador que estemos utilizando.

A principios del año 2008 Google liberó su API de librerías AJAX, como una red de distribución de contenido y arquitectura de carga, para algunos de los frameworks más populares.

- mediante esta API se eliminan las dificultades a la hora de desarrollar **mashups** en JavaScript; se elimina el problema de:
  - alojar las librerías (ya que están centralizadas en Google)
  - configurar las cabeceras de cache, etc.
- esta API ofrece acceso a las siguientes librerías Open Source, realizadas con JavaScript:
  - jQuery.
  - prototype.
  - scriptaculous.

    - mooTools.
    - dojo, swfobject, chrome-frame, webfont, etc.
- los scripts de estas librerías están accesibles directamente utilizando la URL de descarga, o a través del 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: Listado de librerías para JavaScript.

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 un **framework JavaScript**, que nos va a simplificar muchísimo la programación.
- como bien sabes, cuando usamos JavaScript tenemos que preocuparnos de hacer scripts compatibles con varios navegadores y, para conseguirlo, tenemos que programar **código compatible**.
- jQuery nos puede ayudar muchísimo a solucionar todos esos problemas, ya que nos ofrece la infraestructura necesaria para crear aplicaciones complejas en el lado del cliente.
- basado en la filosofía de "*escribe menos y produce más*", entre las ayudas facilitadas por este framework están:
  - la creación de **interfaces de usuario**
  - uso de **efectos dinámicos**
  - AJAX
  - acceso al DOM
  - eventos
  - etc.
- además esta librería cuenta con infinidad de **plugins**, que nos permitirán hacer
  - presentaciones con imágenes
  - validaciones de formularios
  - menús dinámicos
  - drag-and-drop
  - etc.

Esta librería

- es gratuita
- dispone de licencia para ser utilizada en cualquier tipo de plataforma, personal o comercial
- el fichero tiene un tamaño mínimo y su carga es realmente rápida
- además, una vez cargada la librería, quedará almacenada en **caché del navegador**, con lo que el resto de páginas que hagan uso de la librería, no necesitarán cargarla de nuevo desde el servidor.

Página Oficial de descarga de la librería jQuery.

Para saber más

En el siguiente enlace podrás acceder a la documentación oficial de la librería.

Documentación oficial de la librería jQuery.

Para poder programar con jQuery, lo primero que tenemos que hacer es cargar la librería. Para ello, podemos hacerlo de dos formas:

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

```
<script type="text/javascript"
        src="https://code.jquery.com/jquery-3.5.1.min.js">
</script>
```

- 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.

Cargando la librería desde nuestro propio servidor:

```
<script type="text/javascript" src="jquery-3.5.1.min.js"></script>
```

- de esta forma, el fichero de la librería estará almacenado como un fichero más de nuestra aplicación, por lo que no necesitaremos tener conexión a Internet (si trabajamos localmente), para poder usar la librería.
- para poder usar este método, necesitaremos descargarnos el fichero de la librería desde la página de jQuery .

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()`.

Esta función `$("selector")`,
- acepta como parámetro una cadena de texto, que será un selector CSS
- también puede aceptar un segundo parámetro, que será el contexto en el cuál se va a hacer la búsqueda del selector citado.

Otro uso de la función, puede ser el de `$(function(){..});`
- equivalente a la instrucción `$(document).ready (function() {...});`
- que nos permitirá detectar cuando el DOM está completamente cargado.

Aquí podemos ver un pequeño ejemplo de cómo utilizar la librería y lo que fácil que es su uso.

```
<!DOCTYPE HTML>
<html>
 <head>
  <meta charset="utf-8">
  <title>DWEC07.- Ejemplo con jQuery</title>
  <style type="text/css">
   .colorido{
    background-color:#99FF33;
   }
  </style>

  <script type="text/javascript" src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
  <script type="text/javascript">
   $().ready(() => {
    $("#mitabla tr:nth-child(even)").addClass("colorido");
   });
  </script>
 </head>
 <body>
  <table width="200" border="1" align="center" id="mitabla">
   <tr>
    <td align="center"><b>País</b></td>
    <td align="center"><b>Habitantes</b></td>
    <td align="center"><b>Renta</b></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 principal función para realizar peticiones AJAX en jQuery es **`$.ajax()`**

- es una **función de bajo nivel**, lo que quiere decir que disponemos de la posibilidad de configurar, prácticamente todos los parámetros de la petición AJAX, y será, por tanto, equivalente a los métodos clásicos que usamos en la programación tradicional.

La sintaxis de uso es: **`$.ajax(`***opciones*`)`**

- en principio, esta instrucción parece muy simple, pero el número de opciones disponibles, es relativamente extenso.
- esta es la estructura básica:

```
$.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]
});
```

Veamos algunas propiedades de la función **`$.ajax()`** de jQuery:

Propiedades de la función `$.ajax()` de jQuery.

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, 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.

Veamos un ejemplo, que ya hicimos pero utilizando el objeto XMLHttpRequest, pero ahora utilizando el método ajax de jQuery.

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <title>DWEC07.- Ejemplo jQuery método ajax</title>
  <script type="text/javascript" src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
  <script type="text/javascript">
   $( () => {
    $.ajax({
     url: 'http://localhost/fecha.php',
     type: 'GET',
     async: true,
     success: (respuesta) => {
         $("#resultados").text(respuesta);
         $("#estado").toggleClass('cargada');
```

```
        $("#estado").text("Cargada.");
     }
    });
   });
  </script>
  <style>
   #resultados {
    background: yellow;
   }
   .cargando {
    background: red;
   }
   .cargada {
    background: green;
   }
  </style>
 </head>
 <body>
 <p>
 A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:
 </p>
 <p>
 Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la página PHP en el servidor
 </p>
 <p>
 Contenedor resultados:
 <div id="resultados"></div>
 </p>
 <p>
 Estado de las solicitudes:
 <div id="estado"></div>
 </p>
 </body>
</html>
```

Para saber más

En la documentación oficial de jQuery puedes consultar todos los parámetros de este método.

Todos los parámetros de uso de la función $.ajax() de jQuery.

3.3.- Los métodos .load(), $.post() , $.get() y $.getJSON() en jQuery

La función `$.ajax()` es una función muy completa, y resulta bastante pesada de usar.

- su uso es recomendable, para casos muy concretos, en los que tengamos que llevar un control exhaustivo de la petición AJAX.
- para facilitarnos el trabajo, se crearon 3 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()`**.

Por tanto, podemos realizar lo mismo que en el ejemplo anterior pero utilizando este método:

```
$( () => {
 $("#resultados").load(
  'http://localhost/fecha.php',
  (respuesta) => {
     $("#estado").toggleClass('cargada');
     $("#estado").text("Cargada.");
  }
 );
});
```

Debes conocer
En el siguiente enlace accederás a la documentación oficial con más información sobre este método.

Más información sobre el método .load() en jQuery

El método `$.post()`

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

Su sintaxis es la siguiente:

`$.post( url, [datos], [callback], [tipo] )`

Por tanto, podemos modificar nuestro ejemplo que hacia una llamada de tipo por éste otro:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <title>DWEC07.- Ejemplo jQuery método ajax - load</title>
  <script type="text/javascript" src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
  <script type="text/javascript">
   $( () => {
    $.post(
     'http://127.0.0.1/procesar.php',
```

```
    'nombre=Bob&apellidos=Esponja',
    (respuesta) => {
     $("#resultados").html(respuesta);
    }
   );
  });
 </script>
 <style>
  #resultados {
   background: yellow;
  }
  .cargando {
   background: red;
  }
  .cargada {
   background: green;
  }
 </style>
</head>
<body>
<p>
A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:
</p>
<p>
Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la página PHP en el servidor
</p>
<p>
Contenedor resultados:
<div id="resultados"></div>
</p>
<p>
Estado de las solicitudes:
<div id="estado"></div>
</p>
</body>
</html>
```

Debes conocer
En el siguiente enlace podrás acceder a la documentación oficial sobre éste método para ampliar información.
Más información sobre el método .post() en jQuery

Los métodos `$.get()` y `$.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.

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

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

Debes conocer

En los siguientes enlaces podrás acceder a la documentación oficial con más información sobre estos métodos.

Más información sobre el método .get() en jQuery

Más información sobre el método .getJSON() en jQuery