|  |
| --- |
| **Introducción a AJAX** |

La base de AJAX es conseguir que las aplicaciones web se parezcan a las aplicaciones de escritorio. En definitiva, conseguir que las páginas sean más responsivas.

Google Maps fue uno de los primeros sitios de gran tamaño en incorporar el uso de AJAX para actualizar el contenido de una página sin tener que cargarlo de nuevo. La velocidad de respuesta del sitio se debe a que sólo se modifican los datos del mapa, manteniéndose el resto de elementos de la página, como el campo de búsqueda, la barra lateral de resultados o los controles del mapa, que permanecen incluso cuando se solicita información adicional del mapa.

AJAX trabaja de una forma muy concreta: se utiliza el JavaScript de la página para enviar una petición a un servidor web, que a su vez devuelve ciertos datos, su respuesta. El programa JavaScript toma esta respuesta y hace algo con ella, como desplazar hacia el norte la imagen del mapa de Google Maps si ha pulsado ese botón, actualizando únicamente esa parte del contenido del navegador.

Pero AJAX presenta otras opciones que podemos usar en nuestro sitio web:

* **Mostrar nuevo contenido HTML sin recargar la página**: por ejemplo, una página que muestre titulares de noticias y solo cargue el contenido de la noticia cuando el usuario haga click sobre uno de ellos.
* **Mostrar al instante los resultados del envío de un formulario:** imaginemos un formulario de suscripción a una newsletter en el que, nada más rellenar los datos del formulario y enviarlos, aparece un mensaje confirmando la suscripción.
* **Registrarse en la página sin salir de ella**: se trata de otro uso de JavaScript relacionado con formularios.
* **Complemento para calificar un elemento de una lista**: te habrás fijado que muchas páginas web relacionadas con libros o películas muestran un sistema de calificación con estrellas que indican la valoración de un usuario. Gracias a AJAX, los visitantes pueden enviar sus votos sin tener que salir de la página, simplemente haciendo click en las estrellas. Existe un complemento muy bueno de jQuery para ello; se encuentra en <https://github.com/wbotelhos/raty>.
* **Navegar por la información de una base de datos**: por ejemplo, si buscamos libros en Amazon, obtendremos una lista con 10 libros de un determinado tema con un indicador para ver los 10 siguientes. Con AJAX podemos movernos por los registros de la base de datos sin tener que saltar de una página a otra.

|  |
| --- |
| **Las bases de AJAX** |

Se necesita que JavaScript, la programación de los servidores y el navegador web trabajen en conjunto.

En el modelo de solicitud tradicional tenemos que una página web hace una solicitud de un archivo al servidor web y este envía la respuesta. Por lo que tiene que recargar las páginas continuamente.

En el modelo de solicitud AJAX, el navegador solo solicita la información que se ha modificado. El servidor devuelve los datos solicitados y la apariencia y el contenido de la página se actualizan gracias a la colaboración de JavaScript.

**Las piezas básicas del puzzle**

AJAX no es una tecnología única, sino la mezcla de varias que trabajan en conjunto para mejorar la experiencia del usuario. Existen tres componentes básicos:

* **El navegador**: los navegadores cuentan con un elemento fundamental que hace que AJAX funcione: *el objeto XMLHttpRequest*. Se trata del elemento que permite que JavaScript envíe peticiones a un servidor web y sea capaz de interpretar sus respuestas.
* **JavaScript**: es el que se encarga de enviar la solicitud al servidor web, espera la respuesta, la procesa una vez recibida y actualiza la página añadiendo contenido o modificando, de alguna forma, su presentación. Dependiendo de la aplicación que se esté ejecutando, JavaScript se encargará de enviar la información de un formulario, solicitar registros a la base de datos o, simplemente, enviar un paquete de datos (como la calificación de un libro). Una vez enviados los datos al servidor, la aplicación JavaScript esperará su respuesta, que puede variar entre registros adicionales de la base de datos o un mensaje que diga, por ejemplo, “*Su calificación ha sido registrada*”.

Gracias a la información recibida, JavaScript será capaz de actualizar la página mostrando nuevas entradas procedentes de la base de datos o informando al visitante de que se ha registrado correctamente. La actualización implica tener que manipular el DOM para añadir, cambiar o eliminar etiquetas HTML y contenido.

* **El servidor web**: el servidor web se encarga de devolver al navegador la respuesta adecuada. Esa respuesta puede ser un texto sin formato, un *código* *HTML*, un *documento* *XML* o *datos* *JSON*. Si, por ejemplo, el servidor recibiese la información de un formulario, podría encargarse de añadirla a una base de datos y enviar después un mensaje de respuesta tal como “*Registro añadido*”. Y si JavaScript enviase una petición para ver 20 entradas más de la base de datos, el servidor respondería enviando la información correspondiente a las 20 entradas.

**Comunicación con el servidor web**

El núcleo de cualquier programa AJAX es el objeto ***XMLHttpRequest***, al que en ocasiones se hace referencia como XHR. Se trata de una función incluida en los navegadores web, que permite que JavaScript envíe peticiones a un servidor web y reciba la información que éste devuelve. Tenemos cinco pasos fundamentales, todos ellos gestionados por JavaScript:

1. Crear una instancia del objeto *XMLHttpRequest*: se trata de informar al navegador web que se quiere enviar cierta información a un servidor web, con el fin de que se prepare para ello. La forma más sencilla es de crear un objeto *XMLHttpRequest* es:

*var xhr = new XMLHttpRequest();*

1. Usar el ***método open()*** de XHR para establecer el método de envío de datos, así como su destino.

Hay varias formas de enviar datos. Las dos más comunes so GET y POST, las mismas que se usan en los formularios HTML. El método GET envía la información al servidor web como parte de la URL: *shop.php?productID=45*. En este ejemplo se envían los datos como parte de una cadena de búsqueda, que es la información que sigue al *símbolo ?*

El método POST envía los datos independientemente de la URL. Lo más normal es utilizar GET para recuperar los datos de un servidor, y POST para enviar información que debe ser almacenada en el servidor.

El método *open()* también se usa para especificar a qué página web del servidor se deben enviar los datos. Normalmente se trata de una página del servidor que utiliza un lenguaje de script PHP/ASP para recuperar la información de una base de datos o llevar a cabo cualquier otra tarea, y que se establece usando su URL. Este código, por ejemplo, indica al objeto XHR qué método (GET) debe usar y la página del servidor a la que realizar la petición:

*xhr.open('GET','compra.php?productID=55');*

|  |
| --- |
| **Nota** |
| La URL que se establezca para el método *open()* debe encontrarse en el mismo sitio web que la página desde la que se realiza la petición, por razones de seguridad de AJAX. JSONP proporciona una forma de evitar este inconveniente. |

1. Crear una función que se encargue de los resultados.

Lo normal es queramos hacer algo con los resultados devueltos por el servidor web. Para ello escribiremos una función de JavaScript que se encargue de trabajar con los resultados. A esta función la denominaremos ***callback***; suele ser el núcleo del programa.

Esta función se encargará de manipular el contenido de la página, es decir, su DOM, eliminando elementos (por ejemplo, un formulario que se ha enviado usando AJAX), añadiendo otros (un mensaje o una nueva tabla HTML de registro para la base de datos) o cambiando los existentes (por ejemplo, resaltando la calificación que el visitante ha establecido para cualquier producto).

1. Enviar la petición.

Para enviar la información al servidor web usaremos el ***método send()*** del objeto XHR. Ahora le diremos al servidor que estamos listos. Si utilizamos el método GET el procedimiento es:

*Xhr.send();*

Este método acepta un argumento con los datos que deben enviarse al servidor. En el caso de una petición GET los datos se enviarán en la URL de esta forma:

*xhr.send('q=javascript');*

Una vez enviada la petición, JavaScript sigue con su ejecución. La “A” de AJAX significa “asíncrono”.

1. Recibir la respuesta.

Tras procesar la petición, el servidor devuelve una respuesta al navegador web. La función *callback* que vimos en el paso 3 se encargará de ella pero, mientras tanto, el objeto XHR recibirá varias partes de la información enviada por el servidor web, incluyendo el estado de la petición, una respuesta de texto y, posiblemente, una respuesta XML.

La respuesta con el estado de la petición es un número que indica cómo ha respondido el servidor a la petición.

*404 🡪 no se ha encontrado el archivo.*

*200 o 304 🡪 todo correcto.*

*500 🡪 Internal Server Error (error de procesamiento de la página).*

*403 🡪 Acceso prohibido (fichero protegido por contraseña).*

Podemos ver la lista de los posibles códigos en:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

Además, en la mayoría de los casos recibiremos una respuesta de texto, que se almacenará en la propiedad *responseText* del objeto XHR. Puede tratarse de código HTML, un sencillo mensaje de texto o un grupo completo de datos JSON.

Por último, el servidor enviará una respuesta de texto en un archivo XML, que se almacenará en la propiedad *responseXML* del objeto XHR. Aunque aún se usan las respuestas XML, lo más habitual es que los servidores envíen la respuesta como texto, HTML o datos JSON, por lo que no es probable que tenga que trabajar con respuestas XML.

Sea cual sea el tipo de datos que se envíe, siempre será posible usar la función *callback* para actualizar la página web. Una vez termine esta función, se habrá completado el ciclo AJAX. Aunque siempre es posible tener varias peticiones AJAX funcionando al mismo tiempo.

|  |
| --- |
| **AJAX al estilo jQuery** |

La biblioteca jQuery proporciona varias funciones que simplifican mucho el proceso. En los pasos anteriores vimos que el paso más importante es el paso 3. En este paso nos tenemos que encargar de la respuesta que nos suministra el servidor. JQuery se encargará de simplificar el resto de los pasos para que nos podamos concertar en programar este paso.

**El método load()**

Es el método más sencillo. Se encarga de cargar un archivo HTML en un elemento específico de la página. Supongamos que tenemos una lista de titulares, apareciendo los 5 últimos cuando se carga la página. Pero queremos añadir vínculos para que los usuarios seleccionen el tipo de historias que se mostrarán en esa área: últimas noticias, noticias locales, deportes, economía, etc. Podríamos vincular a otras páginas, pero eso obligaría a que las visitas tuviesen que acceder a otra página, y además, no usaríamos AJAX.

La otra posibilidad sería cargar el tipo de noticias seleccionando en el mismo campo de la página. De esa forma, cada vez que el visitante seleccionase una categoría, el navegador pediría un nuevo archivo HTML al servidor, y situaría el contenido de la respuesta en la misma zona sin tener que abandonar la página.

|  |
| --- |
| **Nota** |
| El principal objetivo de AJAX es permitir que JavaScript se comunique con el servidor y obtenga información de este. Normalmente será recuperar información de una base de datos, enviar un correo electrónico o permitir el registro de un usuario. |

Para utilizar la ***función load()*** deberemos comenzar usando un selector de jQuery para identificar el elemento de la página donde se cargará el HTML solicitado. Después se llama a la función *load()* para pasar la URL de la página que se quiera cargar.

Imaginemos que tenemos una etiqueta div con un ID *lineasCabecera* y queremos cargar en ella el HTML del archivo *noticiasActuales.html*. Podríamos hacerlo así:

*$('lineasCabecera' ).load('noticiasActuales.html');*

Al ejecutarse este código, el navegador solicitará el archivo *noticiasActuales.html* al servidor web. Una vez descargado, sustituirá el contenido de la etiqueta referenciada por el contenido del archivo.

|  |
| --- |
| **Nota** |
| Solo es posible cargar archivos HTML situados en el mismo sitio web. |

Debemos tener cuidado con las rutas de los archivos cuando usemos esta función. En primer lugar, la URL que pasemos a la función deberá estar relacionada con la página actual. Es decir, debemos utilizar la misma ruta que si estuviera vinculando el archivo HTML que queremos cargar.

Además, las rutas del archivo HTML no se reescriben al cargar el nuevo HTML en el documento, por lo que si el documento original tuviese imágenes o vínculos, sus vínculos tendrían que funcionar en relación con la página que utilice la función *load()*.

En otras palabras: si utilizamos rutas relativas al documento para los vínculos y el archivo HTML que se debe cargar se encontrase en otra carpeta del sitio, lo más probable es que los vínculos y las imágenes no funcionen al cargar el nuevo HTML en la página.

Existe una solución: utilizar solo vínculos relativos a la raíz del documento, o asegurarnos de que el archivo que carguemos se encuentre en el mismo directorio que la página que esté usando la función *load().*

La función *load()* permite especificar la parte del archivo HTML que queremos incluir en la página. Supongamos que solicitamos una página que incluye todos los elementos de una página normal del sitio. Lo más probable es que solo necesitemos un área concreta de la página, un div específico con su contenido. En este caso, podemos especificar la parte de la página que deseamos cargar insertando un espacio tras su URL, seguido por un selector de jQuery. En el ejemplo anterior, se trataría de cargar el contenido de un ***div*** con in ***ID*** denominado ***noticias*** que forme parte del archivo *noticiasActuales.html*.

*$('lineasCabecera' ).load('noticiasActuales.html #noticias');*

**Métodos get() y post()**

En el apartado anterior recuperábamos del servidor un contenido HTML. En otras situaciones lo que necesitamos es enviar un mensaje, un código numérico o simplemente datos que será necesario procesar más adelante utilizando jss. Si, por ejemplo, queremos usar AJAX para obtener filas de una base de datos, lo más probable es que el servidor nos devuelva un archivo XML con esos datos, o un objeto JSON. Este tipo de datos deben ser procesados para poder generar el código HTML que necesitamos.

Estas dos funciones de jQuery nos permiten enviar y recuperar datos hacia y desde un servidor. Ya hemos comentado anteriormente que es necesario tratar el *objeto XMLHttpRequest* de una forma distinta cuando usamos el método GET o el método POST. Estas funciones se encargan de solventar sus diferencias.

La estructura básica de estas funciones es:

*$.get(url, data, callback);*

*$.post(url, data, callback);*

A diferencia de la mayoría de las funciones de jQuery, no es necesario añadir *get()* o *post()* a un selector de jQuery. Las dos funciones son independientes, sin conexión con el resto de elementos de la página.

Definición de los parámetros:

* **url**: es una cadena que contiene la ruta de acceso hacia el recurso del servidor.
* **Data**: puede ser una cadena o un literal de un objeto de javaScript que contenga los datos que se quieran enviar al servidor.
* **Callback:** es la función que procesa la información devuelta por el servidor.

Al utilizar una de estas dos funciones, el navegador enviará una petición utilizando el método especificado, a la URL establecida, con los datos indicados. Cuando el servidor devuelva al navegador los datos solicitados, este último los entregará a la función *callback*, que procesará la información y, seguramente, actualizará la página web.

**Dar formato a los datos que se envían al servidor**

Normalmente, debemos enviar datos a un servidor web. Si, por ejemplo, queremos obtener información sobre un determinado producto que tenemos en una tabla de una base de datos, no tenemos más que enviarle el código que lo represente. Cuando el servidor web reciba el número de la solicitud XHR, buscará el producto en la base de datos, recuperará la información del producto y la enviará de vuelta al navegador web. También es posible usar AJAX para enviar un formulario repleto de información, como parte de un pedido en línea, o como parte de un registro para el envío de un boletín de noticias.

En cualquier caso, siempre deberemos dar un formato adecuado a los datos para que las funciones *get()* o *post()* puedan comprenderlos. El segundo de los parámetros enviados a cualquiera de estas funciones contiene los datos que queremos enviar al servidor, y puede tener formato de una cadena de búsqueda o del literal de un objeto de javaScript.

**Cadena de búsqueda**

Como ya conocemos, las cadenas de búsqueda aparecen al final de la URL tras el *símbolo ?*. Contienen dos pares nombre/valor. Por ejemplo, *codProducto=30&idSesion=100*.

Con esta cadena se consigue exactamente lo mismo que creando dos variables, *codProducto* e *idSesion*.

También es posible enviar datos al servidor usando AJAX. Pongamos, por ejemplo, que hemos creado una página web que permite que los usuarios califiquen las películas haciendo click en el número de estrellas. Un click en 4 estrellas envía su calificación al servidor como, por ejemplo, *ratio=4*. Vamos a suponer que la página que procesa las calificaciones es *ratioPelicula.php*. El código que se enviaría al servidor usando AJAX es:

*$.get('ratioPelicula.php','ratio=4');*

o

*$.post('ratioPelicula.php','ratio=4');*

|  |
| --- |
| **Nota** |
| Las funciones *get()* y *post()* de jQuery no exigen definir los datos ni la función *callback*. Solo es necesario proporcionar la URL de la página del servidor web. Pero, lo más habitual es proporcionar también estos datos. Por ejemplo, en el código *$.get('ratioPelicula.php','ratio=4');* solo se proporciona la URL y los datos, sin especificar la función *callback*. En este caso, el usuario no hace más que enviar la calificación, sin que se espere una respuesta por parte del servidor con datos que deban ser procesados con la función *callback*. |

|  |
| --- |
| **¿GET o POST?** |
| Los dos métodos para enviar datos a un servidor web son ***GET*** y ***POST***.  **DEPENDE**.  En algunos casos no existe la opción. Supongamos, por ejemplo, que estamos enviando la información a un script que ya está instalado y funcionando en el servidor. Es decir, que la programación ya está hecha, y solo debemos utilizar javaScript para hablar con el script. En este caso, deberemos utilizar el método que está esperando ese script del servidor. Lo más probable es que el programador lo haya configurado para aceptar tanto de *GET* como de *POST*, por lo que deberíamos utilizar cualquiera de los dos. Pero lo más habitual es hablar con el programador para que nos diga qué método ha utilizado.  Si el código del servidor no se ha escrito todavía, tendremos que escoger el método que se va a usar.  El *método GET* es adecuado para peticiones que no afectan al estado de la base de datos o de los archivos que hay en el servidor web. Es decir, cuando se quiere obtener información sobre el precio de un producto o la lista de los productos favoritos.  El *método POST* se usa cuando se quiere enviar datos que van a modificar la información que hay en el servidor, como una petición para eliminar un archivo, o para actualizar una tabla de una base de datos.  En realidad, siempre podemos usar cualquiera de los dos métodos. En ocasiones se usan al revés de lo que hemos comentado anteriormente. Solo hay una situación en la que es obligatorio usar *POST*: si vamos a enviar una gran cantidad de datos a un servidor. El método *GET* tiene una limitación en la cantidad de datos que es capaz de enviar, que varía en función del navegador. Por este motivo, los desarrolladores normalmente utilizan el método *POST* en los formularios que puedan incluir algo más que meras cajas de texto. |

Si necesitamos enviar algo más que un par nombre/valor, insertaremos el *símbolo* *&* entre cada uno de los pares:

*$.post('ratioPelicula.php','ratio=4&usuario=pepe');*

Debemos tener cuidado al usar este método, ya que varios caracteres tienen un significado especial cuando formen parte de una cadena de búsqueda. El *símbolo &*, por ejemplo, se usa para incluir pares de nombre/valor adicionales a la cadena, mientras que el *símbolo =* asigna un nombre a un valor. Por ejemplo, la siguiente cadena no es válida:

*'nombre= pepe & pepa');*

En realidad, queremos asignar *pepe & pepa* a la variable, pero se interpreta como un segundo par nombre/valor. Si queremos utilizar caracteres especiales como parte del nombre o del valor en una pareja, deberemos codificarlos de forma que no se puedan confundir con un carácter que tenga un significado especial. Por ejemplo, el *carácter espacio* se representa con *%20*, el *símbolo* *&* con *%26* y el *símbolo =* con *&3D*. La línea anterior la reescribiríamos para que sea correcta:

***'nombre= pepe%20%26%20pepa');***

JavaScript proporciona un método denominado *encodeURIComponent()* para evitar los problemas con las cadenas:. Si lo presentamos formando parte de una cadena, devolverá la cadena correcta:

***var stringConsulta = 'nombre=' + encodeURIComponent('pepe & pepa');***

**Literal de un objeto**

El método de búsqueda funciona bien para datos sencillos, que no incluyen ningún símbolo de puntuación. Pero *get()* y *post()* admiten un método más eficaz para almacenar los datos: el literal de un objeto.

Es posible pasar el literal del objeto directamente a las funciones *get()* o *post().* En este ejemplo utilizamos el método de la cadena de búsqueda:

*$.post('ratioPelicula.php','ratio=4');*

Para usar el literal del objeto deberíamos reescribir el código:

*$.post('ratioPelicula.php',{ratio : 4});*

Puedo pasar el literal del objeto directamente a las funciones *get()* o *post()*, o bien almacenarlo antes en una variable y pasar luego esa a *get()* o a *post()*.

*var datos = {ratio: 4};*

*$.post('ratioPelicula.php', datos);*

Podemos incluir cualquier cantidad de pares nombre/valor en el objeto que se pase a las funciones *get()* o *post()*:

*var datos = {*

*ratio: 4,*

*usuario: 'pepe'*

*};*

*$.post('ratioPelicula.php', datos);*

O bien, podemos pasar directamente el objeto literal a *$.post()*.

*$.post('ratioPelicula.php',*

*{*

*ratio: 4,*

*usuario: 'pepe'*

*};*

*); // fin de post*

**Función serialize() de jQuery**

Crear una cadena de búsqueda o un objeto para todos los pares de nombre/valor de un formulario puede ser una tarea costosa. Debemos obtener el nombre y el valor de cada elemento y combinarlos para crear una cadena de búsqueda o el literal de un único objeto de javaScript. JQuery proporciona una función que facilita la conversión de la información de un formulario en datos que las funciones *get()* y *post()* pueden usar directamente.

La función ***serialize()*** se puede aplicar a cualquier formulario, o a una sección del mismo, para crear una cadena de búsqueda. Para usarla, primero deberemos crear una selección de jQuery que contenga el formulario, y después asignarle la función ***serialize()***.

Supongamos, por ejemplo, que tenemos un formulario con un ID denominado *login*. Si deseamos crear una cadena de búsqueda a partir del formulario haremos:

*var datosFormulario = $( '#login').serialize;*

El método ***serialize()*** recopila los nombres de todos los campos y los valores correspondientes, y crea una cadena de búsqueda única.

Para usar este sistema con las funciones *get()* y *post()* solo tenemos que pasar los resultados a la función como parte del segundo parámetro. Si, por ejemplo, queremos enviar el contenido de un formulario de registro a una página denominada login.php haremos:

*var datosFormulario = $( '#login').serialize;*

*$.get('login.php',datosFormulario,resultadosLogin);*

Este código envía los datos introducidos por el visitante en el formulario *login.php*, usando el método *GET*. El último parámetro es la función *callback* que se encarga de hacer algo con los datos recuperados desde el servidor web.

**Procesar los datos del servidor**

AJAX es un sistema de dos sentidos: el programa javaScript envía datos al servidor y el servidor devuelve datos al programa javaScript, para poder utilizarlos en la actualización del contenido de la página. Vamos a conocer cómo recibir y procesar la respuesta del servidor web.

Una vez que el navegador envía una petición al servidor, usando el *objeto* *XMLHttpRequest*, se mantiene a la escucha esperando una respuesta del servidor. Cuando llegue esta respuesta, una *función* *callback* se encargará de ella. Para ello, será necesario asignar varios parámetros a la función, el primero de los cuáles (y el más importante) son los datos devueltos por el servidor. Los datos devueltos por el servidor pueden encontrarse en diferentes formatos. El script del servidor puede devolver un número, una palabra, un párrafo de texto o una página web completa. Si necesitamos enviar una gran cantidad de información (como un conjunto de filas de una tabla de una base de datos), el servidor usará XML o JSON, como veremos.

El segundo parámetro de la *función* *callback* es una cadena que indica el estado de la respuesta. Casi siempre será ***success***; indica que el servidor ha procesado correctamente la petición y que ha devuelto los datos. Pero las peticiones podrían fallar, como es el caso de solicitar un archivo que no existe o si tenemos un error en el programa de servidor. En ese caso la función devuelve el mensaje ***error***.

La función procesa la información y después actualiza la página de alguna manera: reemplazando un formulario con los datos enviados por el servidor o, simplemente, mostrando un mensaje de éxito. Este proceso de actualización se lleva a cabo mediante las *funciones html() y text() de jQuery* (ya vistas).

Para hacernos una idea del ciclo completo de petición/respuesta, vamos a exponer el sistema de calificación de una película. El visitante puede hacer click en uno de los cinco vínculos para calificar una película, cada uno de los cuáles implica una calificación diferente. Al hacer click en el vínculo, se envía una petición a un programa alojado en el servidor, que añade la calificación a una tabla de la base de datos y devuelve la calificación media de la película. Esta calificación media se mostrará en la página principal.

Para que esta página funcione sin javaScript, cada uno de los vínculos de la página enlazará con una página dinámica alojada en el servidor web, que es capaz de procesar la calificación otorgada por el visitante. Por ejemplo, la calificación de tres estrellas podría ser:

*ratio.php?ratio&película=321.*

Podemos utilizar javaScript para interceptar cualquier click de estos vínculos y trasladarlos en una llamada AJAX al servidor web:

1. *$('#mensaje a').click(function() {*
2. *var href=$(this).attr('href');*
3. *var stringConsulta=href.slice(href.indexOf('? ')+1);*
4. *$.get('ratio.php', stringConsulta, procesarRespuesta);*
5. *return false; // No sigas el vínculo*
6. *}); //fin de click*

Línea 1: seleccionamos todos los vínculos que se encuentran dentro de otra etiqueta con id=”mensaje”.

Línea 2: extraemos el atributo ***href*** del vínculo.

Línea 3: extraemos únicamente la parte de la URL que se encuentra detrás del carácter ***?***.

Línea 4: contiene la petición AJAX. Utiliza el método *GET* para enviar una petición de búsqueda para el enlace al recurso *ratio.php* del servidor web. Los resultados los recibirá la función *callback* denominada ***procesarRespuesta***.

Línea 5: se limita a detener el comportamiento normal del enlace, evitando que el navegador salga de la página actual y cargue la página enlazada.

El último paso va a ser crear la función *callback*. Esta función recibirá los datos del servidor, junto a una cadena con el estado de la respuesta (*success* si el servidor envía la respuesta a la petición). En nuestro caso la función se denomina ***procesarRespuesta:***

*function procesarRespuesta(data) {*

*var nuevoHTML;*

*nuevoHTML = '<h2>Tu voto ha sido registrado</h2>';*

*nuevoHTML += '<p>La media de valoración para esta película es ';*

*nuevoHTML += data + '.</p>';*

*$('#mensaje').html(nuevoHTML); // fin de procesarRespuesta*

*}*

**Solucionar los errores**

Podemos tener problemas en el servidor tales como que esté offline en algún momento, o que la conexión a Internet de un usuario falle en un momento dado. Si pasa algo, tanto el método ***$.get()*** como el *método* ***$.post()*** fallarán sin que lo sepa el visitante de la página.

Aunque no es un problema habitual, siempre conviene estar preparado y avisar al usuario de que algo ha fallado, para que vuelva a cargar la página, intentarlo de nuevo o probar dentro de un rato.

Para responder un error tenemos que añadir la función ***.error()*** al final de las funciones ***$.get()*** o ***$.post()***. Con este método, la línea 4 del código anterior se modificaría sustituyéndola por:

***$.get('ratio.php', stringConsulta, procesarRespuesta).error(respuestaError);***

Lógicamente, deberíamos crear una nueva función, llamada *respuestaError*, que se encargue de notificar al visitante de la existencia de cualquier problema. Por ejemplo:

*function* ***respuestaError*** *() {*

*var mensajeError = “Su voto no puede ser procesado en este momento”;*

*mensajeError = “Por favor, Inténtelo de nuevo”;*

*$('#mensaje').html(mensajeError); // fin de respuestaError*

*}*

|  |
| --- |
| **Ejemplos Sitio Ejercicios5** |
| **Noticias.php**  **Login.php** |