# Tema 4. Interactuar con el servidor. jQuery y Ajax.

## Índice del tema



## 1. Objetivos

En este sexto tema se pretenden conseguir los siguientes objetivos:

- Entender el concepto de interfaz de una sola página (SPI).
- Conocer la tecnología Ajax.
- Utilizar los métodos que nos ofrece jQuery para interactuar con el servidor por medio de Ajax.
- Recoger y tratar los datos devueltos por el servidor, tanto en formato JSON, como en XML.
- Utilizar Ajax para mejorar las funcionalidades de la librería jQuery UI.

## 2. Interfaz de una sola página (SPI)

Durante este curso hemos ido viendo cómo podemos, por medio de jQuery, darle cierta interactividad a nuestros sitios web. Este paradigma llevado al extremo, se convierte en una aplicación web que pretende tener todas las ventajas disponibles para las aplicaciones de escritorio.
La idea es crear una única página que va sufriendo cambios parciales a partir de las interacciones con el usuario, de manera que la url base nunca se modifica.

> *Jose María Arranz Santamaría* en su artículo *"manifiesto por una única página web (Single Page Interface)"* nos habla de este tipo de aplicaciones y de cuáles son sus ventajas e inconvenientes.

En las anteriores sesiones, hemos ido construyendo una aplicación web basada en una única página (carro.html). Utilizamos jQuery para definir el comportamiento de la página, creando o eliminando elementos en función de la interacción con el

usuario. Sin embargo, nuestra aplicación tiene un gran problema, todavía no interactúa con el servidor. Cuando una aplicación SPI necesita interactuar con el servidor (obtener datos de la BBDD, enviar datos de un formulario, etc.), utiliza peticiones Ajax. El procedimiento sería el siguiente:

- la aplicación cliente realiza una petición de datos,
- el servidor se encarga de proporcionar estos datos y,
- de nuevo la aplicación cliente, se encarga de procesar esta respuesta para modificar parcialmente la página que tenemos cargada.

Otras operaciones que se deben hacer con el servidor, es ir almacenando el estado de la página, por si se produjera una recarga de la misma, por ejemplo porque el usuario pulsa el botón recargar (F5).
Evidentemente, cuando utilizamos el modelo SPI también existen desventajas, como son:

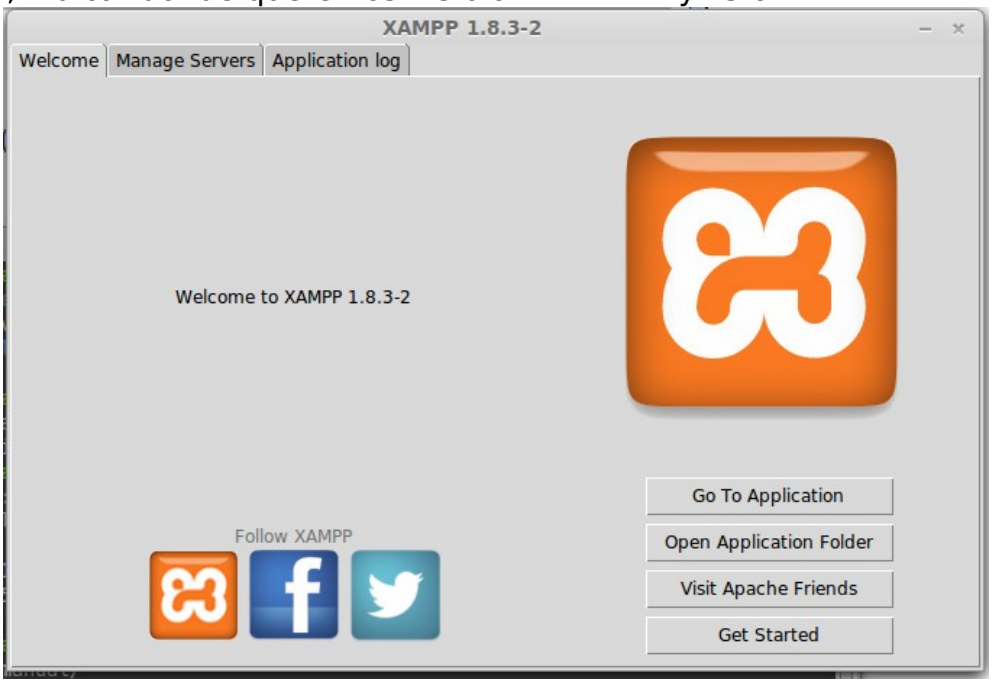
- Memorización de páginas favoritas (bookmarking): cada página web tiene una única URL, dicha URL es, por tanto, "memorizable" como favorito. Como AJAX tiende a cambiar parcialmente las páginas se pierde la posibilidad de guardar favoritos.
- Search Engine Optimization (SEO): la compatibilidad SEO es un deber de cualquier sitio web, cualquiera puede entender esto. Los actuales robots indexadores de páginas ven la web como Web 1.0, es decir, el código JavaScript es totalmente ignorado, de forma que, cualquier cambio parcial de página hecho vía AJAX, cargando datos del servidor, no es conocido por los robots indexadores que atraviesan el sitio web, porque dicha acción no se realiza.
- Servicios basados en visitas de páginas: anuncios y monitorización de visitas, como por ejemplo, Google Analytics, están basados en el número de veces que una página ha sido cargada. Por tanto, los cambios parciales de páginas hechos por AJAX no cuentan.

Estos requisitos hacen que el uso intensivo de AJAX esté desaconsejado en sitios web de gran público. Sin embargo, la diferencia entre aplicación web y sitio web es cada vez más sutil, de hecho cualquier sitio web es hoy día una aplicación web.
En el artículo de *Jose María Arranz Santamaría* mencionado anteriormente se comentan algunas soluciones a este tipo de problemáticas.

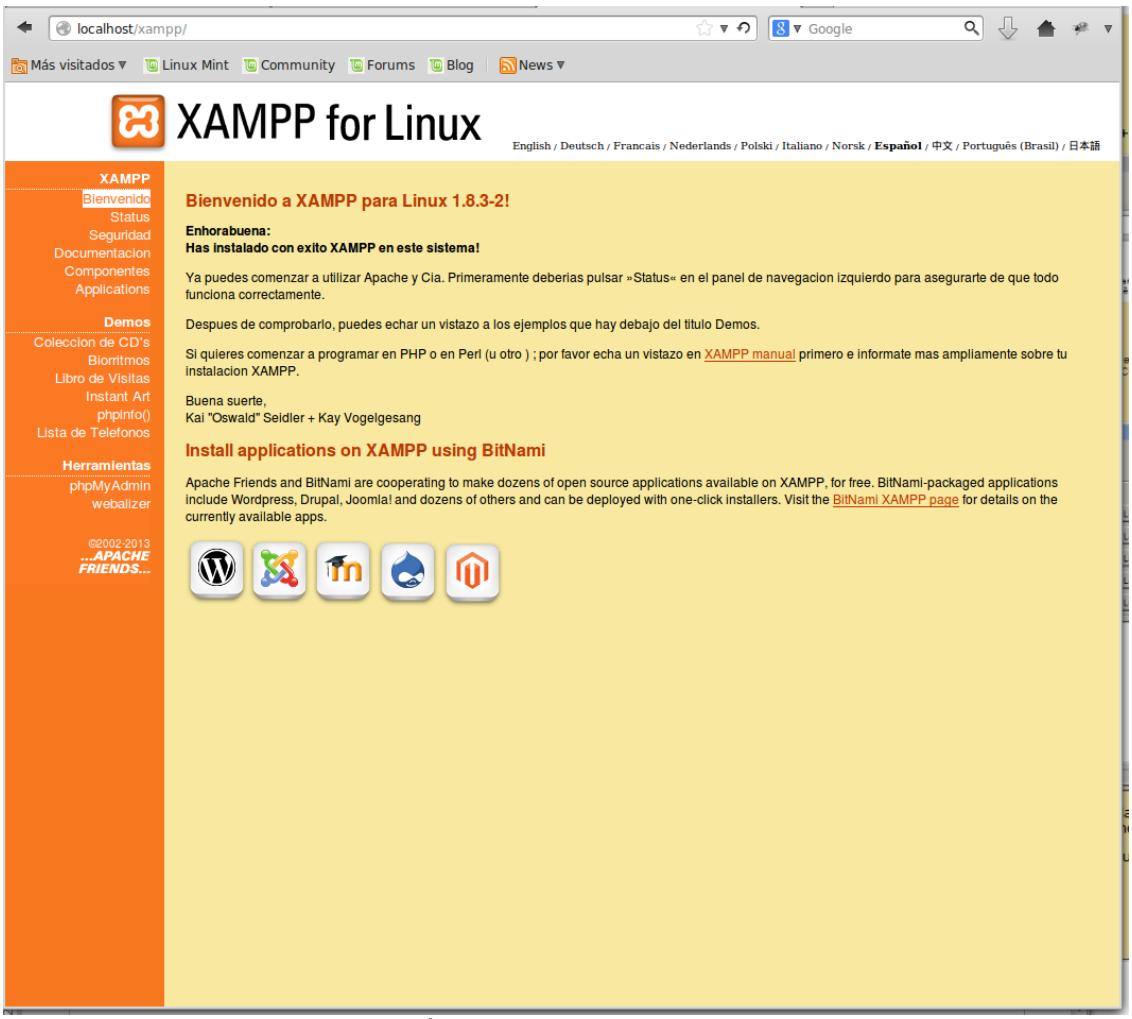
## 3. Preparación del entorno

Como ya hemos comentado, las peticiones Ajax se realizan llamando al servidor web, por lo tanto, en este tema para poder ejecutar los ejemplos y resolver los ejercicios, vamos a necesitar tener disponible un servidor web.
El lenguaje de servidor que se va a utilizar es php, pero, dado que esto es un curso de jQuery, se proporciona toda la funcionalidad php necesaria para resolver los ejercicios.
Lo que sí que vamos a tener que hacer es preparar el entorno. Vamos a necesitar tres cosas, un servidor web (Apache), un módulo de scripting para php conectado con el servidor web y un servidor de BBDD (mysql). Para simplificar la instalación, utilizaremos una plataforma AMP que integra estas tres herramientas en un solo paquete.
En nuestro caso usaremos XAMPP, que nos proporciona un instalador para Windows, Linux y Mac y que se distribuye bajo licencia GNU.
La instalación de este paquete pre configurado es bastante sencilla.
Si accedemos a la página del proyecto XAMPP, dentro del sitio web de ApacheFriends (http://www.apachefriends.org/en/xampp.html), tendremos acceso a las diferentes distribuciones.
Una vez accedemos al apartado de XAMPP para nuestro sistema operativo, en la

página se nos informa de las versiones de los productos incluidas en el paquete, y además, tenemos acceso a la zona de descargas y a una breve explicación de los métodos de instalación disponibles.
El método más sencillo para realizar la instalación es descargar el instalador, ejecutarlo, indicar dónde queremos instalar XAMPP... y listo.




Mediante este panel de control podremos iniciar o finalizar los servicios de nuestro sistema. Pulsaremos el botón start de apache y de mysql, el resto de servicios no nos interesan.
Para comprobar que nuestro servidor está funcionando correctamente, sólo tenemos que acceder a través de nuestro navegador a la url http://localhost.
También podemos acceder a esta página pulsando en el botón Admin de Apache.
Se abrirá una interfaz web con una serie de funcionalidades a las que tenemos acceso. Por ejemplo, si queremos comprobar que el módulo de php está funcionando correctamente, podemos pulsar en el enlace phpinfo de la sección Php.
Además, disponemos de una serie de herramientas, entre las cuales, cabe destacar phpMyAdmin, que es una aplicación web que nos dará acceso a nuestro SGBD Mysql. Otra forma de acceder a la herramienta, es a través de la url http://localhost/phpMyAdmin/.




Si hemos realizado una instalación sobre Linux en el directorio /opt/lamp, la carpeta que utiliza apache para alojar los documentos de nuestras aplicaciones web será /opt/lamp/htdocs.
En el apartado de ejercicios, se explicará cómo prepararemos el entorno para poder ejecutar nuestra aplicación del carrito de la compra.

# 4. Ajax y jQuery

Uno de los problemas que nos encontramos a la hora de realizar peticiones Ajax, es la compatibilidad entre distintos navegadores. Distintos navegadores tienen formas distintas de implementar esta funcionalidad.
Como ya ocurre con otras características del lenguaje javascript, jQuery nos ofrece una serie de métodos que nos van a facilitar mucho las cosas a la hora de realizar estas peticiones asíncronas al servidor. Entre otras cosas, nos ocultará los cambios en la implementación de los distintos navegadores.
A continuación, veremos cómo funcionan estos métodos y que usos podemos darles.

## *a) Añadiendo HTML*

A menudo, lo único que necesitamos cuando hacemos una petición Ajax, es que nos devuelva una porción de código HTML. Esta técnica se conoce como AHAH (Asynchronous HTTP and HTML) y su implementación por medio de jQuery es trivial. Utilizaremos para ello el método .load(). Este método obtienes datos del servidor (normalmente HTML) y los coloca dentro del elemento seleccionado.
El método puede recibir tres parámetros:

- url (este parámetro es obligatorio): Un string que contiene la url del elemento a cargar. Si en este parámetro aparece un espacio en blanco, lo que hay a continuación de este, se interpreta como un selector. Este selector se utiliza para saber que parte del documento queremos que se cargue en el elemento seleccionado.

```
$("#articulo").load("articulo.html");
```

  En este ejemplo, se cargará todo el contenido de la página artículo.html en el elemento con id artículo de la página actual.

```
$("#resultados").load("test.html #contenedor");
```

  En este otro ejemplo, se cargará en el elemento con id resultados de la página actual, el elemento con id contenedor de la página test.html.
- data (este parámetro es opcional): Un string indicando los datos que se le pasan al servidor. También se pueden pasar estos datos utilizando un objeto (normalmente con notación JSON). Si pasamos los datos como un objeto la petición se realiza utilizando el método POST, si no, se enviarán por GET.

```
$("#users").load("users.php", { 'users[]' : ["Juan", "Elena"] });
```

  En este ejemplo, se carga en el elemento con id users de la página actual, el resultado de la ejecución del script users.php, pasándole por POST un array de usuarios.
- complete: Una función que se ejecuta cuando se completa la petición.

```
$("#feeds").load("feeds.php", {limite: 25}, function() {
alert("Se han cargado las últimas 25 entradas del feed");
});
```

  En el ejemplo, se carga en el elemento con id feeds de la página actual, el resultado de la ejecución del script feeds.php, pasándole por POST un objeto que contiene el entero limite. Cuando se haya terminado la carga de los resultados se mostrará un alert. La función se ejecutará una vez para cada elemento seleccionado, al cual podremos acceder a través de $(this).

En el siguiente ejemplo, cargamos un enlace existente dentro del documento enlaces.html en el  con id contenedor. Al finalizar, se muestra un alert, con el código html insertado.

```
<!DOCTYPE html>
<html>
  <head>
    <title>enlaces</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
</head>
<body>
    <ul>
            <li><a id="yahoo" href="http://www.yahoo.es">Yahoo</a></li>
            <li><a id="google" href="http://www.google.es">Google</a></li>
      </ul>
</body>
</html>
```

Enlaces.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>método .load()</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
                    type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
        {
            $("#contenedor").load("enlaces.html #yahoo", function()
            {
                  alert($(this).html());
            });
        });
    </script>
</head>
<body>
    <div id="contenedor"></div>
```

```
</body>
</html>
```
Ejemplo 1: metodoLoad.html

## b) Añadiendo javascript

En ocasiones, no nos interesa obtener todos los ficheros javascript cuando la página se carga, por ejemplo, puede ser que no sepamos que scripts vamos a necesitar, hasta que el usuario seleccione una serie de opciones. Cuando esto suceda, tendremos que descargar un fichero javascript después de cargada la página. Para esto, jQuery dispone del método $.getScript(). Este método recibe como parámetro la url del fichero javascript a descargar y realiza una petición Ajax que solicita el archivo en cuestión. Una vez recibido el archivo, todo el código que contenga será ejecutado.
Además de la url, el método puede recibir también una función callback que se ejecutará cuando la petición finalice.

```
<!DOCTYPE html>
<html>
  <head>
    <title>método .getScript()</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
      <style>
        .block { background-color: blue; width: 150px; height: 70px; margin: 10px; }
      </style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
                    type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
        {
              var url =
"https://raw.github.com/jquery/jquery-color/master/jquery.color.js";
              $("#go").click(function()
              {
                     $.getScript(url, function()
                     {
                            $(".block")
                            .animate( { backgroundColor: "rgb(255, 180, 180)" }, 1000 )
                            .delay(500)
                            .animate( { backgroundColor: "olive" }, 1000 )
                            .delay(500)
                            .animate( { backgroundColor: "#00f" }, 1000 );
                     });
              });
        });
    </script>
</head>
<body>
  <button id="go">&raquo; Run</button>
  <div class="block"></div>
</body>
</html>
```
Ejemplo 2: metodoGetScript.html
En el ejemplo, al pulsar el botón run, se carga el script que contiene la animación de color de jQuery. Al terminar la petición, se realiza una animación utilizando las características del script descargado.

## c) Recibiendo datos

Cuando necesitamos que el servidor nos envíe una serie de datos, tenemos dos opciones para recibirlos: utilizar el lenguaje XML para definir nuestro propio formato de intercambio de datos o utilizar objetos JSON que son más sencillos de manejar desde javascript, y por ende, desde jQuery.

### Recibiendo JSON

Para recibir datos utilizando la notación JSON, jQuery nos ofrece el método $.getJSON(). Este método no sólo obtiene los datos, sino que también los procesa. Cuando los datos llegan desde el servidor, están en forma de cadena de texto con

formato JSON. El método $.getJSON() analiza esta cadena y nos proporciona un objeto javascript con el que podremos trabajar sin ningún problema.
Una cuestión a destacar en este método (y en el resto de métodos que veremos a continuación), es que no es necesario invocarlos a través de un selector u objeto jQuery, en lugar de esto, lo invocamos a través del objeto jQuery global, representado a través del símbolo $.
Al igual que el método .load(), este método puede recibir hasta tres parámetros: la url de la petición Ajax (obligatorio), un objeto data con los parámetros que necesitamos enviar (opcional) y una función success, que se ejecutará cuando la petición haya terminado y los datos hayan sido procesados.
Este método devuelve un objeto jqXHR, que es un objeto jQuery que contiene un subconjunto de las propiedades del objeto XMLHTTPRequest de javascript.
De esta forma, podemos asociar una serie de funciones callback que se ejecutarán cuando se produzcan determinados eventos en la petición. Estas funciones son:

- .done(): se ejecutará cuando la petición se haya procesado correctamente.
- .fail(): se ejecutará cuando se produzca un error al procesar la petición.
- .always(): se ejecutará siempre, se produzca un error o no.

Es muy habitual utilizar encadenamiento, para asociar estas funciones con el objeto devuelto. En el siguiente ejemplo, podemos ver cómo se van ejecutando las distintas funciones, a medida que se van produciendo los eventos.

```
{
	"nombre" : "Alex",
	"resultado" : "true"
}
```

ejemplo.json

```
<!DOCTYPE html>
<html>
  <head>
    <title>callbacks $.getJSON()</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <style>img{ height: 100px; float: left; }</style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
                    type="text/javascript"></script>
    <script type="text/javascript">
    $(function()
      {
             $.getJSON("ejemplo.json", function(datos)
             {
                    console.log( "OK. Nombre="+datos.nombre+"");
             }).done(function(datos)
             {
                    console.log("segundo OK. resultado="+datos.resultado);
             }).fail(function()
             {
                    console.log("error");
             }).always(function(datos)
             {
                    console.log("completado. resultado="+datos.resultado);
             });
      });
    </script>
</head>
<body>
</body>
</html>
```

Ejemplo 3: callbacksGetJSON.html

Como se puede observar en el ejemplo, todos los métodos, excepto .fail(), reciben un parámetro con los datos recibidos en forma de objeto javascript. El acceso a estos datos desde el método .always() tiene un problema, y es que en caso de error también se ejecuta este método, pero el valor del objeto recibido será undefined.
A continuación, veremos un ejemplo más práctico en el que utilizamos el api de flickr para obtener, utilizando el método $.getJSON(), las imágenes relacionadas con el tag que indiquemos. El número máximo de imágenes que podemos obtener es de 20.

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <title>método $.getJSON()</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <style>img{ height: 100px; float: left; }</style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
                    type="text/javascript"></script>
    <script type="text/javascript">
    $(function()
       {
              $("#buscar").click(function ()
              {
                     var textoBusqueda = $("#texto_busqueda").val();
                     var numFotos = parseInt($("#num_fotos").val());
                     if (!isNaN(numFotos))
                     {
                            if (numFotos > 20)
                                   numFotos = 20;
                            var flickerAPI =
"http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?";
                            $.getJSON( flickerAPI,
                            {
                                   tags: textoBusqueda,
                                   tagmode: "any",
                                   format: "json"
                            })
                            .done(function(datos)
                            {
                                   $("#imagenes").empty();
                                   $.each(datos.items, function( i, item )
                                   {
                                          $( "<img/>" ).attr( "src",
item.media.m ).appendTo( "#imagenes" );
                                          if ( i === numFotos-1 )
                                          {
                                                 return false;
                                          }
                                   });
                            });
                     }
              });
       });
    </script>
</head>
<body>
       <label for="texto_busqueda">Tag</label><input type="text" id="texto_busqueda" />
       <label for="num_fotos">Número de fotos (Máx. 20)</label><input type="text"
id="num_fotos" />
       <input type="button" value="Buscar" id="buscar" />
    <div id="imagenes"></div>
</body>
</html>
```

Ejemplo 4: metodoGetJSON.html

## Recibiendo XML

Para recibir datos en formato XML podemos utilizar los métodos $.get() o $.post(). Estos métodos obtienen datos del servidor realizando una petición GET o una petición POST, según cuál de los dos métodos utilicemos. No sólo se utilizan para obtener datos en formato XML, pero son adecuados para ello.
Estos métodos reciben los mismos parámetros que el método .load() y uno adicional, el parámetro dataType. En este parámetro indicaremos el tipo de datos que se espera recibir del servidor (xml, json, script, html). Si no indicamos este parámetro, jQuery intentará deducirlo a partir del tipo mime de la respuesta del servidor.
Al igual que el método .load(), devuelve un objeto jqXHR, que nos permite asociar los callbacks .done(), .fail() y .always().
La función .done() recibirá los datos devueltos en forma de objeto XML, fichero javascript, objeto json, etc. Esto dependerá del dataType indicado.
Como es bien conocido, el lenguaje XHTML es un subconjunto del XML, o mejor dicho, puede ser generado mediante XML. Por lo tanto, si estamos utilizando los

selectores y los métodos de jQuery para poder acceder a la información de los documentos XHTML, también podremos hacerlo para obtener la información de los documentos XML.
Veamos como accedemos a los elementos del siguiente documento, si suponemos que lo hemos obtenido como resultado de una petición Ajax.

```
<?xml version="1.0" encoding="UTF-8"?>
<usuarios>
  <usuario>
      <id>aa</id>
    <nick>alex</nick>
    <nombre>alejandro</nombre>
  </usuario>
  <usuario>
      <id>jj</id>
    <nick>john</nick>
    <nombre>juan</nombre>
  </usuario>
</usuarios>
```

Ejemplo 5: usuarios.xml

La petición nos habrá devuelto un objeto XML. Supongamos que lo tenemos en una variable de nombre data. Para transformarlo en objeto jQuery, sólo tenemos que hacer lo siguiente:
$(data)
Para obtener un objeto jQuery que contenga todos los usuarios del fichero, podemos hacer lo siguiente:
$(data).find("usuario")
Para recorrer cada uno de estos usuarios, utilizaremos el método $.each():
$(data).find("usuario").each(function() {});
Dentro de la función que le pasamos al método $.each(), accederemos a cada uno de los usuarios a través de $(this).
Para acceder a los atributos de un elemento xml utilizaremos el método .attr().
Para acceder a los contenidos utilizaremos el método .text().
Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>método $.get()</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
      <style>
      label { display:block;}
      </style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
                    type="text/javascript"></script>
    <script type="text/javascript">
    $(function()
      {
             $("input").click(function()
             {
                    $.get( "usuarios.xml" )
                    .done(function( data )
                    {
                           $(data).find("usuario").each(function()
                           {
                                  var $usuario = $('<div id="'+$(this).find('id').text()
+'"></div>');
                                  $usuario.append('<label class="nick">nick '+$
(this).find('nick').text()+'</label>');
                                  $usuario.append('<label class="nombre">nombre '+$
(this).find('nombre').text()+'</label>');

                                  $("#usuarios").append($usuario);
                           });
                    });
             });
        });
    </script>
</head>
<body>
    <input type="button" value="obtener usuarios" />
      <div id="usuarios"></div>
</body>
```

```
</html>
```
Ejemplo 6: metodoGet.html

Como vemos, en el ejemplo se obtiene un documento de usuarios en xml y se introducen en el div usuarios con el formato y estructura que no interesa.
El método $.post() funciona igual que el método $.get() con la diferencia de que los parámetros, si los hubiera, serían enviados al servidor mediante POST.

### d) El método $.ajax

Todos los métodos vistos hasta este punto son atajos del método $.ajax() de jQuery. Este método es capaz de realizar todas las funcionalidades vistas en los métodos anteriores y muchas otras.
No es el objetivo de este curso ver la tecnología Ajax en profundidad, por lo tanto, con las funcionalidades que nos ofrecen los métodos anteriores, tenemos más que suficiente para tener una visión general sobre las posibilidades que nos ofrece esta tecnología.
Para ampliar conceptos y ver el funcionamiento del método $.ajax() en detalle, se puede consultar la documentación del método en http://api.jquery.com/jQuery.ajax/.

## 5. Ejercicios

### a) Ejercicio 1

Para poder trabajar con nuestra aplicación del carrito de la compra, tendremos que preparar el entorno. Para ello, crearemos una carpeta llamada tema4jquery (dentro del htdocs de nuestro XAMPP). En esta carpeta copiaremos todos los documentos contenidos en el fichero carrito.zip. La carpeta scripts está vacía, por lo tanto, copiaremos en esta carpeta, el fichero carro.js con las soluciones del tema anterior.
En los siguientes ejercicios iremos viendo qué cosas debemos implementar para que nuestra aplicación siga funcionando como nos interesa.
En el fichero carrito.zip tenemos un script (carrito.sql) con una pequeña base de datos que da soporte a nuestra aplicación. Para cargar esta base de datos en nuestro servidor mysql, tenemos que ir a la url http://localhost/phpmyadmin/ y seguir los siguientes pasos:

1. Pulsar el botón importar.
2. Seleccionar el archivo a importar (carrito.sql).
3. Seleccionar el conjunto de caracteres del archivo iso-8859-1.
4. Pulsar continuar.

Para terminar con la configuración debemos editar el fichero configuración.inc que hemos copiado a la carpeta tema4jquery. En este fichero se indican los datos de configuración para conectarnos a mysql. El fichero está preparado para conectarse con el usuario root y el password vacío. Si queremos conectarnos con otro usuario y/u otro password, lo indicaremos en la variable correspondiente.
Una vez realizados estos pasos, ya tenemos preparado nuestro entorno y podemos acceder a la aplicación del carrito de la compra mediante la siguiente url http://localhost/tema4jquery/. Como el elemento item_container está vacío, no aparecerán los artículos disponibles para comprar. Esto lo solucionaremos en el siguiente ejercicio.

### b) Ejercicio 2

Para que aparezcan los artículos en el item_container, vamos a realizar una petición

Ajax que nos devolverá un documento xml con todos los ítems de nuestra base de datos.
La petición la realizaremos utilizando el método $.get() de la siguiente forma:

```
$.get("ajaxCarrito.php", {operacion: "obtenerItems"})
```

Es conveniente que creemos una función para implementar este ejercicio, podemos llamarla, por ejemplo, obtenerItems.
La petición Ajax nos devolverá un documento xml con la siguiente estructura:

```
<?xml version="1.0" encoding="UTF-8"?>
<items>
    <item>
        <id>i1</id>
        <imagen>img/reloj1.jpg</imagen>
        <nombre>Reloj 1</nombre>
        <precio>20.00</precio>
        <stock>82</stock>
    </item>
    <item>
        <id>i10</id>
        <imagen>img/camiseta5.jpg</imagen>
        <nombre>Camiseta 5</nombre>
        <precio>25.00</precio>
        <stock>50</stock>
    </item>
…
</items>
```

Si se produce un error al obtener los ítems, se mostrará un mensaje indicándolo en el item_container.
A partir del xml obtenido, debemos recorrer cada uno de los ítems y generar el código html necesario para aparezcan, en el item_container, los artículos disponibles con las funcionalidades que tenían en el tema anterior.
En el ejemplo 6, se puede ver como recorrer los elementos de un documento XML.
Para generar cada uno de los ítems dentro de la función del método $.each() que recorre los mismos, seguiremos los siguientes pasos:

1. Crearemos un elemento , le pondremos la clase item y el id obtenido en el xml.
2. Dentro del  anterior, crearemos una imagen (<img>) y tres <label>, una para el nombre, otra para el precio y otra para el stock. Los valores que introduciremos en estos campos los obtendremos también del xml.
3. Si el stock es cero deberemos poner la clase agotado.
4. Si el stock es mayor que cero, debemos asociar el evento dblclick al ítem creado, y hacerlo draggable para que el comportamiento sea el mismo que teníamos en el tema anterior.
5. En temas anteriores guardábamos el ancho inicial y la posición inicial del carrito en dos variables para poder consultarlas más tarde. Debemos tener en cuenta que, inicialmente, en la página no existen artículos, por lo tanto, si guardamos el ancho y posición antes de introducir los artículos nos quedaremos con una posición errónea. Así que, debemos actualizar el valor de estas variables al ancho y posición que tiene el carrito después de haber introducido todos los artículos recibidos.

## c) Ejercicio 3

El principal problema que tiene nuestra aplicación del carrito es que cuando se recarga la página se pierde todo lo que teníamos seleccionado en el carrito. Para solucionar este problema tenemos que enviarle al servidor, todas y cada una de las operaciones que realizamos, de esta forma, irá guardando en la sesión los artículos que vamos seleccionando.

Las operaciones que el servidor deberá guardar son: añadir un nuevo ítem al carrito, eliminar un ítem del carrito, incrementar la cantidad comprada de un ítem del carrito y decrementar la cantidad comprada de un ítem del carrito.

### Añadir un nuevo ítem al carrito

Debemos hacer una petición Ajax en el manejador de evento dblclick de los ítems del carrito para que el servidor guarde en la sesión el ítem que estamos añadiendo al carrito. Utilizaremos el método $.getJSON() de la siguiente forma:

- `$.getJSON("ajaxCarrito.php", {operacion: "nuevo", idArticulo: id})`
    - id será una variable que contendrá el id del artículo que se está añadiendo al carrito.

Esta petición Ajax devolverá un booleano en formato JSON indicando si se ha registrado correctamente la operación. En caso de devolver true, se añadirá el artículo al carrito y se actualizarán los campos correspondientes (stock, número de compras, etc.). Si devuelve false o la petición Ajax falla, se mostrará un alert indicándolo y no se añadirá el artículo.
En el servidor ya se controla si el id del artículo enviado ya existía en la sesión, en cuyo caso, incrementará la cantidad, por lo tanto, como al pulsar el botón add se llama al mismo manejador de evento, no hay que hacer nada adicional.

### Eliminar un ítem del carrito

Debemos hacer una petición Ajax en el manejador de evento del botón delete para que el servidor elimine de la sesión el ítem que estamos eliminando del carrito. Utilizaremos el método $.getJSON() de la siguiente forma:

- `$.getJSON("ajaxCarrito.php", {operacion: "elimina", idArticulo: id})`
    - id será una variable que contendrá el id del artículo (sin la "c" inicial) que se está eliminando del carrito.

Esta petición Ajax devolverá un booleano en formato JSON indicando si se ha registrado correctamente la operación. En caso de devolver true, se eliminará el artículo del carrito y se actualizarán los campos correspondientes (stock, número de compras, etc.). Si devuelve false o la petición Ajax falla, se mostrará un alert indicándolo y no se eliminará el artículo.

### Decrementar la cantidad comprada de un ítem del carrito

Debemos hacer una petición Ajax en el manejador de evento del botón minus para que el servidor decremente en la sesión la cantidad del artículo. Utilizaremos el método $.getJSON() de la siguiente forma:

```
$.getJSON("ajaxCarrito.php", {operacion: "decrementa", idArticulo: id})
```

- id será una variable que contendrá el id del artículo (sin la "c" inicial) del que se está decrementando la cantidad.

Esta petición Ajax devolverá un booleano en formato JSON indicando si se ha registrado correctamente la operación. En caso de devolver true, se actualizarán los campos correspondientes (stock, número de compras, etc.). Si devuelve false o la petición Ajax falla, se mostrará un alert indicándolo y no se eliminará el artículo.
Hemos de tener en cuenta que cuando la cantidad vale uno, al pulsar el botón minus se llama al manejador del evento click del botón delete, por lo que no hay que hacer ninguna gestión especial para ese caso.

## *d) Ejercicio 4*

Ahora que ya tenemos almacenados en la sesión los elementos que se han ido añadiendo al carrito, tenemos que realizar la gestión oportuna para que al recargar la página se muestre el carrito, tal y como estaba antes de la recarga. Para ello,

realizaremos una petición Ajax que nos devuelva un array asociativo en formato JSON que contendrá, para cada ítem que hayamos introducido en el carrito, su id y la cantidad comprada. Si no habían ítems en el carrito en el momento de la recarga, se devolverá null. Un ejemplo podría ser el siguiente:

- `{"i5":3,"i6":1}`
    - En este ejemplo, se han seleccionado tres artículos con id "i5" y un artículo con id "i6".

Si la petición Ajax falla, se mostrará un alert indicándolo. En caso contrario, se comprobará que los datos devueltos son distintos de null y, si es así, se recorrerá el array resultante añadiendo cada uno de los artículos al carrito y actualizando los campos correspondientes (stock, número de compras, etc.), es decir, realizaremos las mismas acciones que en el manejador del evento dblclick del artículo correspondiente. En este caso, no podemos hacer un trigger del evento, porque esto provocaría que se aumentara el número de artículos comprados en la sesión del servidor, por lo tanto, lo mejor es tener una función que contenga todas las acciones que se realizan al introducir un artículo al carrito e invocar a la función, tanto aquí, como cuando hacemos doble click sobre el artículo.
Otra cosa a tener en cuenta, es que no podemos hacer esto hasta que esté disponible la lista de artículos, por lo tanto, el lugar idóneo para introducir la petición Ajax que obtiene los artículos que habían en el carrito, es en la función que recibe los datos de la petición Ajax del ejercicio 2, después de haber generado todos los artículos. Es decir, tendremos algo así:

```
$.get("ajaxCarrito.php", {operacion: "obtenerItems"})
.done(function( data )
{
      // aquí recorreremos los resultados y generaremos los ítems
      …
      // aquí actualizaremos el ancho y la posición del carrito
      …
      $.getJSON("ajaxCarrito.php", {operacion: "obtenerComprados"})
      .done(function(data)
      {
      // Aquí recorreremos el array e introduciremos los ítems al carrito
      }
      …
}
…
```

## e) Ejercicio 5

Todavía no hemos hablado de la funcionalidad más importante en un carrito de la compra ¿qué pasa cuando se pulsa el botón comprar?
En nuestro caso, al pulsar el botón comprar se va a mostrar un diálogo de jQuery UI que contendrá un formulario que descargaremos del servidor. El formulario está en el documento formEnviar.html y el elemento que tenemos que mostrar dentro del diálogo tiene el id form_enviar. Utilizaremos el método $.load() para descargar el formulario (Ver ejemplo 7).

- El dialogo tendrá las siguientes características:
- Para mostrar el formulario crearemos un  que añadiremos al body y lo eliminaremos al cerrar el diálogo.
- Debe de ser un Dialogo modal.
- La duración de los efectos de mostrado y ocultación será de 600 ms.
- El efecto de mostrado será clip y el de ocultación será explode.
- El título será "Confirmar compra".
- No se mostrará el botón cerrar.
- No se cerrará con la tecla escape.
- El ancho será de 450px (propiedad width).

- La posición será top (propiedad position).
- Al cerrar el diálogo se eliminará el
- Tendrá un botón Comprar y otro botón Cancelar.
- Si se pulsa el botón Cancelar, se cerrará el diálogo y se cancelará la compra.
- Si se pulsa el botón Comprar seguiremos los siguientes pasos:
    - Comprobaremos que hay algo escrito en todos los campos, si hay algún campo vacío mostraremos un error en el formulario de la siguiente forma:

```
$(this).find("#contenedorError").show();
$(this).find("#error").html("Todos los campos son obligatorios");
```

        - Con $(this) estamos accediendo al diálogo.
    - Si todos los campos están rellenos, realizaremos una petición Ajax al servidor que se encargará de guardar la compra. Utilizaremos el método $.getJSON() de la siguiente forma:

```
$.getJSON("ajaxCarrito.php", {operacion: "comprar"})
```

    - Si la petición Ajax falla, se mostrará un alert indicándolo.
    - Si todo va bien, el servidor se habrá encargado de realizar las operaciones oportunas para actualizar los stocks de los artículos en la base de datos, nosotros tendremos que encargarnos de reinicializar los campos modificados, para ello haremos lo siguiente:
        - Pondremos a cero el número de compras.
        - Pondremos a cero el precio total de la compra.
        - Reinicializaremos el ancho y posición del carrito (elemento cart_items).
        - Ocultaremos los botones de la barra de navegación (Comprar, vaciar, izquierda y derecha).
        - Eliminaremos todos los ítems que haya dentro del carrito.
        - Cerraremos el diálogo. Para poder acceder al diálogo desde dentro de la función de respuesta a la petición Ajax, tendremos que guardarlo en una variable local a la función del botón:

```
$dialogo = $(this);
$.getJSON("ajaxCarrito.php", {operacion: "comprar"})
.done(function(data)
{
        $dialogo.dialog("close");
});
```

        - Esto es así por el comportamiento de las clausuras en javascript. Las clausuras javascript son muy útiles (de hecho en jQuery casi todo son clausuras), pero al principio cuesta un poco entenderlas. Si alguien está interesado en profundizar un poco en este tema, puede consultar la siguiente url: http://digitta.com/2008/03/clausuras-javascript-para-torpes.html . Aquí se explican las clausuras de forma práctica y sin entrar en detalles teóricos.

Como esto habríamos terminado nuestra aplicación del carrito de la compra, es importante observar que la gestión del pedido no está implementada en el servidor, es decir, no se almacena nada sobre las compras realizadas, simplemente se actualizan los stocks de los productos.
Hasta aquí este último tema del curso, para ver cómo debería quedar la aplicación terminada podéis consultar el vídeo de demostración en el aula virtual.