UTN.BA FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

EXPERTO UNIVERISTARIO EN MySQL Y PHP NIVEL AVANZADO

www.sceu.frba.utn.edu.ar/e-learning

Módulo II

CONTINUAMOS CON PHP AVANZADO

PHP Y Ajax.

Presentación de la Unidad:

En esta unidad aprenderán el concepto de AJAX, cual es la ventaja de su uso y cual es la diferencia con una aplicación web tradicional.

Aprenderán a a instanciar el objeto Axax. Configurar y abrir una petición al servidor. Enviar la petición al servidor y administrar la respuesta devuelta por éste.

Objetivos:

- Comprender la técnica de desarrollo web utilizando Ajax.
- Aprender a instanciar el objeto Axax. Configurar y abrir una petición al servidor. Enviar la petición al servidor y administrar la respuesta devuelta por éste.

Temario:

CONSIGNAS PARA EL APRENDIZAJE COLABORATIVO

En esta Unidad los participantes se encontrarán con diferentes tipos de consignas que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

1. Los foros asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen las actividades sugeridas y compartan en los foros los resultados obtenidos.

¿QUÉ ES AJAX?

AJAX, acrónimo de ***A****synchronous* ***J****avaScript* ***A****nd* ***X****ML* (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (*Rich Internet Applications*). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante XMLHttpRequest objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado enXML.

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores dado que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

DIFERENCIAS CON LAS APLICACIONES WEB TRADICIONALES

En las aplicaciones web tradicionales los usuarios interactúan mediante formularios, que al enviarse, realizan una petición al servidor web. El servidor se comporta según lo enviado en el formulario y contesta enviando una nueva página web. Se desperdicia mucho ancho de banda, ya que gran parte del HTML enviado en la segunda página web, ya estaba presente en la primera. Además, de esta manera no es posible crear aplicaciones con un grado de interacción similar al de las aplicaciones habituales.

En aplicaciones AJAX se pueden enviar peticiones al servidor web para obtener únicamente la información necesaria, empleando SOAP o algún otro lenguaje para servicios web basado en XML, y usando JavaScript en el cliente para procesar la respuesta del servidor web. Esto redunda en una mayor interacción gracias a la reducción de información intercambiada entre servidor y cliente y a que parte del proceso de la información lo hace el propio cliente, liberando al servidor de ese trabajo. La contrapartida es que la descarga inicial de la página es más lenta al tenerse que bajar todo el código JavaScript

APLICACIONES QUE USAN AJAX

Tradicionalmente se ha considerado la primera aplicación AJAX al cliente Web que tiene la herramienta de trabajo en grupo Microsoft Exchange Server aunque sin lugar a dudas Google es uno de los grandes responsables de la popularización de AJAX, al usarla en varias de sus aplicaciones, entre las que se cuentan Google Groups, Google Suggest, Google Maps y el servicio de correo electrónico gratuito Gmail. Así como también empresas en crecimiento que actualmente están desarrollando aplicaciones basadas en AJAX.

- Uno de los primeros entornos para programar sitios web que permitió a los programadores incorporar AJAX fácilmente fue Ruby on Rails.
- A9, buscador de Amazon
- Flickr. Álbumes de fotos online.
- Oddpost, servicio avanzado de webmail de Yahoo!
- Basecamp, servicio de gestión de proyectos diseñado por 37Signals sobre plataforma Rails.
- 24SevenOffice ERP/CRM
- Panoramio.com Comunidad de fotos sobre Google Maps
- meebo Mensajería Instantánea desde tu navegador

TECNOLOGÍAS INCLUIDAS EN AJAX

Ajax es una combinación de cinco tecnologías ya existentes:

- XHTML (o HTML) y hojas de estilos en cascada (CSS) para el diseño que acompaña a la información.
- Document Object Model (DOM) accedido con un lenguaje de scripting por parte del usuario, especialmente implementaciones ECMAScript como JavaScript yJScript, para mostrar e interactuar dinámicamente con la información presentada.
- El objeto XMLHttpRequest para intercambiar datos de forma asíncrona con el servidor web.
- PHP es un lenguaje de programación de uso general de script del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico también utilizado en el método Ajax.
- XML es el formato usado generalmente para la transferencia de datos solicitados al servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano yJSON.

Ajax no constituye una tecnología en sí, sino que es un término que engloba a un grupo de éstas que trabajan conjuntamente.

NAVEGADORES QUE PERMITEN AJAX

Ha de tenerse en cuenta que ésta es una lista general, y el soporte de las aplicaciones Ajax dependerá de las características que el navegador permita.

- Navegadores basados en Gecko como Mozilla, Mozilla Firefox, SeaMonkey, Camino, K-Meleon, IceWeasel, Flock, Epiphany, Galeon y Netscape versión 7.1 y superiores
- Navegadores basados en WebKit como Google Chrome de Google o Safari de Apple.
- Microsoft Internet Explorer para Windows versión 5.0 y superiores, y los navegadores basados en él
- Navegadores con el API KHTML versión 3.2 y superiores implementado, incluyendo Konqueror versión 3.2 y superiores y el Web Browser for S60 de Nokia tercera generación y posteriores
- Opera versión 8.0 y superiores, incluyendo Opera Mobile Browser versión 8.0 y superiores.

NAVEGADORES QUE NO PERMITEN AJAX

- Opera 7 y anteriores
- Microsoft Internet Explorer para Windows versión 4.0 y anteriores
- Anteriores a Safari 1.2
- Dillo
- Navegadores basados en texto como Lynx y Links
- Navegadores para personas con capacidades especiales visuales (Braille)
- Algunos navegadores de teléfonos móviles
- Navegador de la PSP

1. PRIMEROS PASOS

El objeto **XHTMLttp** permite realizar una petición al servidor de forma asincrónica y sin cambiar la URL.

Los navegadores competencia de IE (Microsoft) implementaron un clon del objeto "**XMLHttp**" llamado: **XMLHttpRequest**. Este objeto es nativo de JavaScript(no requiere ningún plugin ni permisos especiales)

1.2. PROCEDIMIENTO

Veamos como es el procedimiento

- Instanciar objeto AJAX.
- Configurar y abrir petición
- Definir una función de JavaScript que se encargue de administrar la evolución de la petición (dada su asincronía)
- Enviar la petición y los datos al servidor
- En la función definida antes, manipular el estado de la petición y, en el caso correcto, recibir los datos y actuar en consecuencia con ellos, según lo que hubiera que hacer.

1.2.1. Instanciar objeto AJAX

IE 5.5.y 6.0

La instanciación se realiza de la siguiente manera:

objeto : new ActiveXObject("nombreClase");

El nombre de la clase depende de la versión del sistema operativo.

Otros navegadores y I.E. desde versión 7

objeto : new XMLHttpRequest();

Utilizo la función "**obtenerXHR()**" incluida en el archivo "instanciacion.js" descripta en el apéndice A, la cual se llama desde la página html de la siguiente manera:

var peticion = obtenerXHR();

1.2.2. Configurar y abrir la petición ("open")

El método **"open"**, no abre la conexión con el servidor, sino que sólo configura la petición y la deja lista para enviarla. Sus parámetros son los siguientes:

Método	**GET o POST**
URL	URL que se quiere invocar, si vamos a enviar datos vía GET, debemos adjuntarlos a la URL
Asincronismo	(true) asincrónica (false) sincrónica
Usuario	
Contraseña	

Nota: No tiene sentido utilizar usuario y contraseña, si el código se puede leer desde la

pantalla del navegador!!!

El código a utilizar sería el siguiente:

```
petición.open("GET","url",true);
```

1.2.3. Definir una función JavaScript

1.2.4. Enviar la petición y los datos al servidor ("send")

Éste es el método que finalmente se encarga de enviar una petición al servidor, una vez que se configuró con el método open. Tiene un solo parámetro opcional que representa los datos que enviaremos en la petición, en el caso de enviar datos vía POST. Si bien es opcional, por compatibilidad con algunos navegadores siempre se acostumbra incluir un parámetro, en caso de no enviar dato al servidor o hacerlo vía GET en la URL, se puede usar el valor especial null para definirlo. El código sería:

```
petición.send(null)
```

1.2.5. Administrar la petición

readyState: Esta propiedad de sólo lectura devuelve un código numérico entre 0 y 4 inclusive, que indica en qué estado se encuentra la petición. Los posibles estados son:

Código	Estado	Descripción
0	Sin inicializar	El requerimiento sólo fue instanciado. Muchos navegadores no manejan este código y utilizan directamente el siguiente.
1	Cargando	El requerimiento se configuró (con open) pero todavía no se envió.
2	Cargado	El requerimiento se envió o se está enviando, aunque todavía no tenemos respuesta alguna del servidor.
3	Interactivo	El servidor ya respondió la petición, ya tenemos disponible la cabecera pero el contenido todavía está descargando.
4	Completo	La petición ya finalizó y el contenido está completo.

status: Devuelve el código HTTP que nos devolvió el servidor. Esta

Código	Descripción
200	La petición se pudo procesar en forma correcta.
404	La URL que peticionamos no existe en el servidor.
500	Error interno del servidor. Puede indicarnos que el servidor está saturado o que hay algún error en el script ejecutado en el servidor.
400	La petición enviada al servidor es errónea. Hay algún inconveniente con las cabeceras o con la información POST enviada.
403	No tenemos permiso de acceder al recurso en el servidor.
405	No se acepta el método. Hay un problema al definir los métodos POST o GET
414	La URL pedida es muy larga. Puede producirse cuando se envían muchos datos por GET. En este caso, se debe cambiar el método a POST.
503	El servidor está temporalmente no disponible

Ver el código de la función "`procesarPeticion()`" en el Apéndice B.

El objeto XMLHttpRequest posee un solo evento estándar, que se puede (y se debe) capturar

para procesar la petición, dado que se está tratando con una petición asincrónica donde el objeto se encarga de avisarnos cuando la petición termina. Este aviso se realiza sobre la base de una función o método que se debe designar como el encargado de procesar la información. La propiedad "onreadystatechange", entonces, debe asignarse a una función que se ejecutará de manera automática cada vez que la propiedad readyState cambie su valor (entre 0 y 4). De esta forma, cuando el estado llegue a 4, ya estaremos listos para leer los datos del servidor y actuar en consecuencia.

```
peticion.onreadystatechange = procesarPeticion;
```

Nota 1: El nombre de la función va sin paréntesis.
Nota 2: En el caso de ser una petición del tipo sincrónica, (valor false en open) no es necesario colocar "onreadystatechange", pues la respuesta llega luego de que todo se cargo.

Apéndice A

```
function obtenerXHR()
{
  var xmlHttp=null;
  if (window.ActiveXObject)
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
  else
    if (window.XMLHttpRequest)
      xmlHttp = new XMLHttpRequest();
  return xmlHttp;
}
```

El objeto XMLHttpRequest es un elemento fundamental para la comunicación asincrónica con el servidor. Este objeto nos permite enviar y recibir información en formato XML y en general en cualquier formato

La creación de un objeto de esta clase varía si se trata del Internet Explorer de Microsoft de las versiones 5.5 y 6.0, ya que este no lo incorpora en JavaScript sino que se trata de una ActiveX:

```
if (window.ActiveXObject)
  xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
```

En cambio en Internet Explorer desde la versión 7, FireFox y otros navegadores lo incorpora JavaScript y procedemos para su creación de la siguiente manera:

```
if (window.XMLHttpRequest)
  xmlHttp = new XMLHttpRequest();
```

Entonces implementamos una función que nos retorne un objeto XMLHttpRequest haciendo transparente el proceso en cuanto a navegador donde se esté ejecutando:

Apéndice B

```
function procesarPeticion() {
        if (canal.readyState == 4) {
                if(canal.status != 200) { alert('Error de conexión al
servidor'); return; }

                document.getElementById('box').innerHTML =
canal.responseText;
        }
}
```

Al escribir el siguiente código:

```
peticion.onreadystatechange = procesarPeticion;
```

Lo que estamos haciendo es ejecutar la función procesarPeticion() cada vez que cambie el readyState.

El código de la función procesarPeticion() es bien simple. Lo que hace es verificar que el readyState llegue a 4. Cuando esto sucede significa que la petición terminó. En ese caso evalúa el status. Si éste es distinto de 200 quiere decir que ocurrió algún error, sino coloca el contenido devuelto en formato HTML por el script que se ejecutó en el servidor dentro del elemento cuyo id es box a través de la propiedad innerHTML de Javascipt.

EJEMPLOS

En la carpeta de ejemplos encontraran varios. Pasaremos a explicar brevemente cada uno de ellos.

En la carpeta **ej1** encontraran solo la puesta en practica de lo que hemos explicado arriba, como utilizar el objeto xmlHttpRequest, instanciándolo y usando sus métodos para ejecutar un script del lado del servidor que levanta un contenido. Este ejemplo es solamente para mostrar, como les decía, el funcionamiento de una petición Ajax, la instanciación del objeto xmlHttpRequest, el método open, el método send, la función manejadora de la petición, evaluando el readyState y refrescando el contenido en la pagina una vez que el readyState llega al valor 4. Pero no tiene mucho sentido ir hasta el servidor para levantar un archivo html como sucede aquí. Esto tendrá sentido si vamos al servidor y hacemos algo en el servidor que no se pueda hacer en el cliente, por ejemplo vamos y actualizamos algo en una tabla sql, tomamos datos de una tabla sql, enviamos un mail, etc.

Un ejemplo de eso es el que tenemos en la carpeta **ej2**, lo que estamos haciendo en este caso es verificar o no la existencia de un usuario en una tabla sql, y le informamos si el usuario esta o no disponible. El procedimiento utilizado es siempre el mismo.

En la carpeta **ej3-select** mostramos un ejemplo de dos combos dinámicos según la selección del primer combo se muestran determinadas opciones en el segundo combo. Si cambio la selección del primer combo automáticamente cambian las opciones mostradas en el segundo combo. En este caso la diferencia es que trabajamos con responseXML en lugar de tomar los datos de responseText ya que los datos vienen en formato xml.

En el ejemplo **ej4-mysql** las cosas son bastante similares, si bien cambia el objetivo del ejemplo, como se darán cuenta el proceso es siempre el mismo.

El ejemplo **ej5-mysql** se trata de una aplicación CRUD, CRUD es una sigla formada por las iniciales de 4 palabras del idioma ingles (Create-Read-Update-Delete), es decir una aplicación en donde están contempladas las 4 operaciones básicas sobre una tabla sql: alta, baja, modificación y consulta. Todas estas operaciones se llevan a cabo utilizando Ajax.

Como habran podido notar el trabajo con Ajax es una combinación de varias tecnologías, necesitamos Javascript para hacer la conexión con el servidor, html y css para poder actualizar el contenido en nuestra pagina y los scripts del lado del servidor, en este caso en PHP y Mysql.

Esta es la forma de trabajar utilizando Javascript en el cliente para manejar las peticiones al servidor. Existe una forma un poco mas sencilla de hacer este trabajo utilizando un framework Javascript, llamado JQuery, seguramente la mayoría de ustedes lo conoce y habrá utilizado algun componente o plugin alguna vez.

Este framework dispone de algunas funciones que internamente trabajan instanciando el objeto xmlHttpRequest, solo que para nosotros es transparente. Esto nos facilita mucho la tarea ya que todo este trabajo esta enmascarado, por decirlo de alguna manera, y nos limitamos simplemente a utilizar el método que nos provee el framework.

Veremos a continuación un ejemplo CRUD utilizando Ajax con Jquery.

EJEMPLO CRUD CON AJAX Y JQUERY

Como dijimos vamos a desarrollar una aplicación CRUD sobre una tabla de clientes. La idea es mostrar una tabla con los datos de los clientes, la opción de editar los datos para que puedan ser modificados, la opción de borrar los datos correspondientes a un cliente y la opción de dar de alta un cliente nuevo. Todas estas acciones se realizarán mediante el método $.ajax() que llamará a un script en el servidor que realizará el alta, modificación o baja del cliente en la tabla sql según corresponda.

Para poder utilizar el método $.ajax() de Jquery incluimos con una etiqueta script el jquery.min de JQuery.

Primero, antes de continuar con el ejemplo vamos a explicar un poco las funciones principales que dispone Jquery para trabajar con Ajax

Funciones para AJAX

Las funciones y utilidades relacionadas con AJAX son parte fundamental de jQuery. El método principal para realizar peticiones AJAX es $.ajax() (importante no olvidar el punto entre $ y ajax). A partir de esta función básica, se han definido otras funciones relacionadas, de más alto nivel y especializadas en tareas concretas: $.get(), $.post(), $.load(), etc.

La sintaxis de $.ajax() es muy sencilla:

```
$.ajax(opciones);
```

```
$.ajax({
  url: '/ruta/hasta/pagina.php',
  type: 'POST',
  async: true,
  data: 'parametro1=valor1&parametro2=valor2',
  success: procesaRespuesta,
  error: muestraError
});
```

La siguiente tabla muestra las principales opciones que se pueden definir para el método $.ajax():

Opción	Descripción
async	Indica si la petición es asíncrona. Su valor por defecto es true, el habitual para las peticiones AJAX
beforeSend	Permite indicar una función que modifique el objeto XMLHttpRequest antes de realizar la petición. El propio objeto XMLHttpRequest se pasa como único argumento de la función
complete	Permite establecer la función que se ejecuta cuando una petición se ha completado (y después de ejecutar, si se han establecido, las funciones de success o error). La función recibe el objeto XMLHttpRequest como primer parámetro y el resultado de la petición como segundo argumento
contentType	Indica el valor de la cabecera Content-Type utilizada para realizar la petición. Su valor por defecto es application/x-www-form-urlencoded
data	Información que se incluye en la petición. Se utiliza para enviar parámetros al servidor. Si es una cadena de texto, se envía tal cual, por lo que su formato debería ser parametro1=valor1&parametro2=valor2. También se puede indicar un array asociativo de pares clave/valor que se convierten automáticamente en una cadena tipo *query string*
dataType	El tipo de dato que se espera como respuesta. Si no se indica ningún valor, jQuery lo deduce a partir de las cabeceras de la respuesta. Los posibles valores son: xml (se devuelve un documento XML correspondiente al valor responseXML), html (devuelve directamente la respuesta del servidor mediante el valor responseText), script (se evalúa la respuesta como si fuera JavaScript y se devuelve el resultado) y json (se evalúa la respuesta como si fuera JSON y se devuelve el objeto JavaScript generado)
error	Indica la función que se ejecuta cuando se produce un error durante la petición. Esta función recibe el objeto XMLHttpRequest como primer parámetro, una cadena de texto indicando el error como segundo parámetro y un objeto con la excepción producida como tercer parámetro
ifModified	Permite considerar como correcta la petición solamente si la respuesta recibida es diferente de la anterior respuesta. Por defecto su valor es false
processData	Indica si se transforman los datos de la opción data para convertirlos en una cadena de texto. Si se indica un valor de false, no se realiza esta transformación automática
success	Permite establecer la función que se ejecuta cuando una petición se ha completado de forma correcta. La función recibe como primer parámetro los datos recibidos del servidor, previamente formateados según se especifique en la opción dataType
timeout	Indica el tiempo máximo, en milisegundos, que la petición espera la respuesta del servidor antes de anular la petición
type	El tipo de petición que se realiza. Su valor por defecto es GET, aunque también se puede utilizar el método POST

url La URL del servidor a la que se realiza la petición

Además de la función $.ajax() genérica, existen varias funciones relacionadas que son versiones simplificadas y especializadas de esa función. Así, las funciones $.get() y $.post() se utilizan para realizar de forma sencilla peticiones GET y POST:

```
// Petición GET simple
$.get('/ruta/hasta/pagina.php');

// Petición GET con envío de parámetros y función que
// procesa la respuesta
$.get('/ruta/hasta/pagina.php',
 { articulo: '34' },
 function(datos) {
  alert('Respuesta = '+datos);
 });
```

Las peticiones POST se realizan exactamente de la misma forma, por lo que sólo hay que cambiar $.get() por $.post(). La sintaxis de estas funciones son:

```
$.get(url, datos, funcionManejadora);
```

El primer parámerto (url) es el único obligatorio e indica la URL solicitada por la petición. Los otros dos parámetros son opcionales, siendo el segundo (datos) los parámetros que se envían junto con la petición y el tercero (funcionManejadora) el nombre o el código JavaScript de la función que se encarga de procesar la respuesta del servidor.

Una vez explicado esto, continuemos con el ejemplo.

Comencemos a analizar el código de nuestro ejemplo.

En el <body> de nuestro archivo index.php tenemos el siguiente código:

```
<div id="popupbox" style="left: 450px;">
        <form name="clientx" id="clientx" method="POST">
                <input type="hidden" name="id" id="id" value="">
                <div>
                        <label>Nombre</label>
                        <input type="text" name="nombre" id="nombre" value = "">
                </div>
                <div>
                        <label>Apellido</label>
                        <input type="text" name="apellido" id="apellido"value = "">
                </div>
                <div>
                        <label>Fecha</label>
```

```
                    <input type="text" name="fecha" id="fecha" value = "">
                </div>
                <div class="buttonsBar">
                    <input id="cancel" type="button" value ="Cancelar"
onclick="$('#popupbox').hide();$('#clientx').reset();"/>
                    <input id="agregar" type="button" name="submit" value
="Guardar" onclick="guardar();"/>
                                    <input id="modificar" type="button"
name="submit" value ="Modificar" onclick="modificarr();"/>
                </div>
        </form>
        </div>
                <div class="container">
                    <h1 class="title">Ejemplo de Altas bajas y modificaciones con
PHP y Ajax usando jQuery</h1>
                    <div id="content">
                    </div>
                </div>
```

El div id=”popupbox” tiene como atributo css display:none, lo pueden ver en el archivo css/style.css con lo cual no aparece visible. Lo que se ve es el div que sigue que es el que contiene un h1 y un div con id content que se cargará con el contenido traído por Ajax.

En nuestro archivo abmc.js observamos que una vez que está listo el DOM de la página se ejecuta la función cargar_all()

```
$(document).ready(function(){
                cargar_all();
        })
```

La función cargar_all utiliza el método $.ajax() para llamar a php-pdo/mostrar.php que es el script que buscará los datos en la tabla de clientes y devolverá el resultado que será cargado en el div id=”content” con el método .html()

```
function cargar_all(){
                $.ajax({
                 type: "POST",
                 url: "php-pdo/mostrar.php",
                 "success":function(data){
                  $('#content').html(data);
                 }
                 });
        }
```

Una vez que tenemos los datos de la consulta en pantalla tenemos tres posibilidades: dar del alta un cliente nuevo , editar los datos de un cliente para modificarlo o dar de baja un cliente.

Comencemos por el alta de un nuevo cliente, el botón para dar de alta un nuevo cliente se genera en el script mostrar.php en la siguiente línea:

```
<div class="bar">
<a id="new" class="button" style="cursor:pointer" onclick="agregar();">Nuevo Cliente</a>
</div>
```

En la que se genera el botón Nuevo cliente y se le asocia el evento onclick de javascript al cual se le asocia la función agregar();

La función agregar() la tenemos definida en nuestro abmc.js

```
function agregar(){
                $('#modificar').hide();
                $('#agregar').show();
                $('#popupbox').show();
        }
```

Lo que hace esta función es ocultar el input con id=”modificar” y mostrar el input id=”agregar”. Por último muestra con el efecto show() el popup para ingresar los datos.

Como se ve en el código de nuestr index.php tenemos un div en el cual están los 3 botones que se verán en el popup, en el caso de Alta se verá el botón Guardar y el botón Cancelar. Y si es una modificación se verá el botón Modificar y el botón Cancelar.

```
<div class="buttonsBar">
        <input id="cancel" type="button" value ="Cancelar" onclick="$('#popupbox').hide();$('#clientx').reset();"/>
        <input id="agregar" type="button" name="submit" value ="Guardar" onclick="guardar();"/>
        <input id="modificar" type="button" name="submit" value ="Modificar" onclick="modificarr();"/>
</div>
```

Una vez cargados los datos del cliente si presionamos Guardar como tenemos asociado el evento onclick=”guardar()” a ese botón ejecutaremos la función guardar() definida en el archivo abmc.js:

```
function guardar(){
                $.ajax({
```

```
                type: "POST",
                url: "php-pdo/guardar.php",
                data: {
                nombre : $('#nombre').val(),
                apellido : $('#apellido').val(),
                fecha : $('#fecha').val()
                },
                "success":function(data){
                        $('#clientx').reset();
                        $('#block').hide();
                        $('#popupbox').hide();
                        cargar_all();
                }
               });
        }
```

La función ejecuta el método $.ajax() de JQuery llamando al script php-pdo/guardar.php que es el que dará de alta las datos en la tabla sql, pasándole por post los datos de nombre, apellido y fecha.

Por el éxito reseteamos el formulario y ocultamos el popup. Luego volvemos a llamar a la función cargar_all() que es la que mostrará la información actualizada.

Si en lugar de confirmar el alta la cancelamos, se resetea el formulario y se oculta el popup:

```
<input id="cancel" type="button" value ="Cancelar" onclick="$('#popupbox').hide();$('#clientx').reset();"/>
```

Las otras dos opciones son Editar un cliente de la tabla para modificarlo y eliminarlo. Continuemos con la modificación:

Al armar la tabla con los datos en el script mostrar.php en una de las columnas se coloca el vínculo para editar los datos:

```
<td><a class="edit" style="cursor:pointer"
onclick="editar('.$cl['id'].',\''.$cl['nombre'].'\',\''.$cl['apellido'].'\',\''.$cl['fecha_nac'].'\');
">Editar</a></td>
```

Es decir que se asocia al evento onclick la función editar() la que espera los datos id, nombre, apellido y fecha nacimiento.

Dicha función esconde el botón de Guardar que se utilizaba en el alta, muestra el botón de Modificar, asigna a cada uno de los campos el valor obtenido de la tabla sql y muestra el popup:

```
function editar(id,nombre,apellido,fecha){
```

```
                $('#popupbox').show();
                $('#modificar').show();
                $('#agregar').hide();
                 $('#id').val(id);
                 $('#nombre').val(nombre);
                 $('#apellido').val(apellido);
                 $('#fecha').val(fecha);

        }
```

El botón de Modificar tiene asociado el evento onclick ante lo que ejecutará la función modificarr()

```
<input id="modificar" type="button" name="submit" value ="Modificar" onclick="modificarr();"/>
```

Dicha función utiliza el método $.ajax() de JQuery para llamar al script php-pdo/modificar.php para modificar los datos en la tabla, pasándole los valores tomados del formulario por el método post:

```
function modificarr(){
                $.ajax({
                 type: "POST",
                 url: "php-pdo/modificar.php",
                 data: {
                 nombre : $('#nombre').val(),
                 apellido : $('#apellido').val(),
                 fecha : $('#fecha').val(),
                 id : $('#id').val()
                 },
                 "success":function(data){
                        $('#clientx').reset();
                        $('#block').hide();
                        $('#popupbox').hide();
                        cargar_all();
                 }
                });
        }
```

Por el éxito reseteamos el formulario y ocultamos el popup. Luego volvemos a llamar a la función cargar_all() que es la que mostrará la información actualizada.

Nos queda ver que es lo que estamos haciendo en el caso de la baja. Al armar la tabla con los datos en el script mostrar.php en otra de las columnas se coloca el vínculo para borrar los datos:

```
<td><a class="delete" style="cursor:pointer"
```

```
onclick="borrar('.$cl['id'].')">Borrar</a></td>
```

Se asocia al evento onclick la función borrar() la que espera recibir el id que utilizará para borrar los datos. Vamos a ver que hace la función borrar() definida en nuestro archivo abmc.js:

```
function borrar(id){
                $.ajax({
                 type: "POST",
                 url: "php-pdo/borrar.php",
                 data: {id : id},
                 "success":function(data){
                 cargar_all();
                 }
                 });
        }
```

Llama al script php-pdo/borrar.php utilizando el método $.ajax() pasándole por post el id recibido por parámetro y anteel éxito de la función llama nuevamente a la función cargar_all() para mostrar los datos actualizados.

Con esto finalizamos de describir la funcionalidad de este ejemplo integrador de conceptos.