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

Centro de e-Learning

Javascript, JQuery y JSON avanzado.

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

Módulo 2

JQUERY UI E

INTERACCIÓN CON EL SERVIDOR

JQuery 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 instanciar el objeto Ajax. Configurar y abrir una petición al servidor. Enviar la petición al servidor y administrar la respuesta devuelta por éste.

Cuales son los distintos métodos que nos brinda JQuery para trabajar con Ajax y como se nos simplifica la tarea utilizándolos.

Objetivos:

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

- Métodos $.ajax() $.get() $.post y $.load().

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.

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)

1.1. 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.

1.1.1. 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();
```

1.1.2. 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);
```

1.1.3. Definir una función JavaScript

1.1.4. 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)
```

1.1.5. 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.

En la carpeta ejemplos-ajax pueden encontrar 4 ejemplos sencillos con el uso de Ajax para terminar de comprender su uso.

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.

AJAX Y JQUERY

Ahora que hemos hecho una introducción a lo que es Ajax vamos a ver como podemos simplificar la tarea utilizando Ajax sobre JQuery y cuales son los métodos de que disponemos para tal fin.

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 todas las 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.

La función $.get() dispone a su vez de una versión especializada denominada $.getIfModified(), que también obtiene una respuesta del servidor mediante una petición GET, pero la respuesta sólo está disponible si es diferente de la última respuesta recibida.

jQuery también dispone de la función $.load(), que inserta el contenido de la respuesta del servidor en el elemento de la página que se indica. La forma de indicar ese elemento es la siguiente:

```
<div id="info"></div>
```

```
// Con jQuery
$('#info').load('/ruta/hasta/pagina.php');
```

Al igual que sucedía con la función $.get(), la función $.load() también dispone de una versión específica denominada $.loadIfModified() que carga la respuesta del servidor en el elemento sólo si esa respuesta es diferente a la última recibida.

Por último, jQuery también dispone de las funciones $.getJSON() y $.getScript() que cargan y evalúan/ejecutan respectivamente una respuesta de tipo JSON y una respuesta con código JavaScript.

El método $.getJSON() es una versión abreviada del método $.ajax(). Utilizarlo equivale a utilizar la función $.ajax() con lo siguientes parámetros:

```
$.ajax({
  dataType: "json",
  url: url,
  data: data,
  success: success
});
```

El método $.getScript() es una versión abreviada del método $.ajax(). Utilizarlo equivale a utilizar la función $.ajax() con lo siguientes parámetros:

```
$.ajax({
  url: url,
  dataType: "script",
  success: success
});
```

Veamos algunos ejemplos de la utilización de estos métodos disponibles en el core del framework para simplificar el uso de Ajax y asegurarnos que nuestras aplicaciones funcionarán bien en todos los navegadores.

EJEMPLO 1: *Cargar datos de forma externa a través del método .load()*

El ejemplo se encuentra en la carpeta ej-jquery-load.

El código de este ejemplo es muy sencillo es una muestra del uso de la función .load()

para cargar en un div información recuperada de un script en el servidor.

```
$(document).ready(function(){
  $("#enlaceajax").click(function(evento){
    evento.preventDefault();
    $("#destino").load("contenido-ajax.html");
  });
})
```

Como se ve, el funcionamiento es muy sencillo simplemente se selecciona el div con id destino y se utiliza el método .load para cargar en ese div el contenido devuelto por contenido-ajax.html

EJEMPLO 2: *Modificar datos en una tabla Mysql mediante Ajax*

El ejemplo se encuentra en la carpeta ej-jquery-post.

Al comienzo del script ej1.html tienen la creación de la tabla SQL con los inserts para que el ejemplo funcione correctamente. Para ejecutarlo deberán tipear en la url ej1.html?foto=1 o ej1.html?foto=2 que son las dos fotos que tenemos disponibles en nuestra tabla de fotos.

Repasemos un poco el código para ver que es lo que está haciendo este script.

En nuestro body tenemos lo siguiente:

```
<img src="<?php echo $datos_foto['archivo'] ?>" id="foto" />
        <div id="epigrafe">
        <span id="text_epigrafe"><?php echo $datos_foto['epigrafe'] ?></span>
        <input type="text" id="nuevo_epigrafe" />
        </div>
```

Una etiqueta de imagen y un div id=”epígrafe” que contiene dentro un campo de input.

Cuando cargamos la imagen indicada por GET agregamos el evento click al elemento cuyo id es text-epigrafe, lo que ocasiona ocultar dicho elemento y mostrar a través del efecto show el campo de input id=”nuevo epigrafe” que tenía atributo de css display:none y posicionamos el cursor con $(#nuevo_epigrafe).focus()

```
$("#text_epigrafe").click(function() {
                $(this).hide();
```

```
                $("#nuevo_epigrafe").show();
                $("#nuevo_epigrafe").focus();
        });

        $("#nuevo_epigrafe").blur(function() {
                $(this).hide();
                $("#text_epigrafe").text( $(this).val() );
                $("#text_epigrafe").show();

                //hacemos el ajax para guardar los datos en el servidor
                $.post("guardar.php", { foto: <?php echo $_GET['foto'] ?>, epigrafe:
$(this).val() } ,
                  function(data){
                        alert("respuesta: " + data);
                  });

        });
```

Luego ante el evento blur() del campo de input actualizamos ese dato en el Text-epigrafe y utilizamos la función $.post que henos descripto anteriormente para actualizar los datos en la tabla SQL.

EJEMPLO 3: *Ordenar los datos de una tabla usando plugin sortable y Ajax*

Nota: Para probar los dos ejemplos que siguen deberán importar la base de datos jqueryajax adjuntada. El código se encuentra en la carpeta ordenar_registros-ajax.

Este es un ejemplo similar al que hicimos en una de las unidades anteriores con la diferencia que en nuestro ejemplo anterior ordenábamos todos los datos y una vez que teníamos definido el orden total presionábamos un botón de submit que enviaba los datos a un script php y éste actualizaba los datos en la tabla SQL.

En este caso vamos a hacer lo mismo con la diferencia que el elemento movido y ubicado en otra posición es actualizado en el mismo momento. Cada vez que muevo un elemento lo actualizo en la tabla.

Para ello armamos en el body de nuestro index.php la estructura necesaria para poder utilizar la interacción sortable.

```
<div class="content">
        <h3>Articulos</h3>
```

```
        <ul id="articulos">
        <?php
                while($row = mysql_fetch_array($query))
                {
                        ?>
                        <li id="articulo-<?php echo $row['id_articulo'] ?>"><?php echo
$row['nombre_articulo'] ?></li>
                        <?php
                }
                ?>
   </ul>
   <div class="msg"></div>
</div>
```

Dentro de nuestro $(document).ready definimos la interacción sortable con las propiedades necesarias para poder llevar a cabo la tarea de ordenar los productos en la tabla SQL. Para ello utilizamos la propiedad update para la que definimos una función que será ejecutada una vez que es modificado alguno de los elementos. Y lo que hace esta función es llamar mediante ajax al script order.php utilizando el método $.post() definido anteriormente y que hemos utilizado también en el ejemplo anterior. Ante la respuesta del script mostramos el mensaje devuelto por éste en el div cuya clase es msg a través del método .html() aplicándole el efecto fadeIn() primero y luego el efecto fadeOut().

```
$("ul#articulos").sortable({ placeholder: "ui-state-highlight",opacity: 0.6, cursor:
'move', update: function() {
                var order = $(this).sortable("serialize");
                $.post("order.php", order, function(respuesta){
                        $(".msg").html(respuesta).fadeIn("fast").fadeOut(2500);
                });
        }
        });
```

EJEMPLO 4: *Filtrar datos de una tabla por fecha, email, país utilizando Ajax*

El código de este ejemplo se encuentra dentro de la carpeta filtro_tabla_mysql_ajax.

Veamos un poco de que se trata. En nuestro archivo index.php colocamos un formulario con los campos que me van a servir para acotar la búsqueda de los datos:

```
<form id="frm_filtro" method="post" action="">
```

```
        <ul>
          <li><label>Nacimiento:    del</label>
              <input type="text" name="del" id="del" size="15" class="datepicker" />
            al
            <input type="text" name="al" id="al" size="15" class="datepicker" /></li>
          <li><label>Nombre/Email:</label>  <input  type="text"
name="nombre_email" size="25" /></li>
          <li><label>Pais:</label>
            <select name="pais">
              <option value="0">--</option>
              <!-- Listar Paises -->
              <?php
              $query = mysql_query("SELECT * FROM pais");
              while($row = mysql_fetch_array($query)){
                ?>
                <option value="<?php echo $row['id_pais'] ?>">
                  <?php echo $row['nombre_pais'] ?>
                </option>
                <?php
              }
              ?>
            </select>
          </li>
          <li>
              <button type="button" id="btnfiltrar">Filtrar</button>
          </li>
          <li>
              <a href="javascript:;" id="btncancel">Todos</a>
          </li>
        </ul>
      </form>
```

El campo select lo armamos de manera dinámica accediendo a la tabla de países.

Analicemos un poco el script js.js, que es el que tiene los métodos JQuery que permitirán el funcionamiento del ejemplo.

Primero que nada llamamos a la función filtrar() para cargar la página con todos los datos. La función filtrar utiliza la función $.ajax para ejecutar en el servidor el script ajax.php de la siguiente manera:

```
url: "ajax.php?action=listar"
```

que espera recibir los datos en format JSON:

```
dataType: "json"
```

Si la respuesta es exitosa se ejecutará:

```
success: function(data){
        var html = '';
        if(data.length > 0){
                $.each(data, function(i,item){
                        html += '<tr>'
                        html += '<td>'+item.nacimiento+'</td>'
                        html += '<td>'+item.nombre+'</td>'
                        html += '<td>'+item.email+'</td>'
                        html += '<td>'+item.pais+'</td>'
                        html += '</tr>';
        });
```

que recorrerá a través de la función $.each() los datos recibidos en el JSON y armará las filas de nuestra table.

Cada vez que acotamos la búsqueda en el formulario debemos presionar el botón Filtrar y la búsqueda se ejecutará nuevamente debido a esta línea en nuestr js:

```
$("#btnfiltrar").click(function(){ filtrar() });
```

Por el vínculo Todos ejecutamos:

```
$("#btncancel").click(function(){
                $(".filtro input").val('')
                $(".filtro select").find("option[value='0']").attr("selected",true)
                filtrar()
        });
```

Lo que hace que se blanqueen los campos del formulario y se vuelva a ejecutar la función filtrar() para cargar todos, ya que no existen filtros.

Nos resta ver la porción de código en la cual se establece el ordenamiento ascendente o descendente según la columna donde se clickee. Y es la siguiente:

```
$("#data th span").click(function(){
                var orden = '';
                if($(this).hasClass("desc"))
                {
                        $("#data th span").removeClass("desc").removeClass("asc")
                        $(this).addClass("asc");
                        ordenar = "&orderby="+$(this).attr("title")+" asc"
                }else
                {
                        $("#data th span").removeClass("desc").removeClass("asc")
                        $(this).addClass("desc");
                        ordenar = "&orderby="+$(this).attr("title")+" desc"
```

```
            }
            filtrar()
    });
```

Aquí se establece la variable orederby que será pasada por POST al script ajax.php. Así si yo hago click en el campo nombre y el ordenamiento actual es ascendente me lo ordenará en el sentido inverso pasándole como valor del parámetro $_POST[‘orderby’] nombre desc. Si volvemos a clickear le pasará nombre asc.

Y eso es lo que se evalúa en el archivo ajax.php en las siguientes líneas:

```
if (isset ($_POST['orderby']))
        $vorder = $_POST['orderby'];
else
        $vorder = '';

if($vorder != ''){
        $sql .= " ORDER BY ".$vorder;
```

Si se generó la variable $_POST[‘orderby’] se le concatena al query el ORDER BY.