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:

Continuamos con JQuery y Ajax.

Desarrollaremos una aplicación completa de ABM (alta/baja/modificación) y consulta utilizando el método $.ajax() para hacer la petición al servidor y el resto de los métodos vistos a lo largo del curso.

Objetivos:

- Integrar los métodos vistos a lo largo del curso junto con el método $.ajax() para las peticiones al servidor, en un ejemplo que reúne todos los conocimientos que han ido adquiriendo de manera integradora.

Temario:

CONSIGNAS PARA EL APRENDIZAJE COLABORATIVO

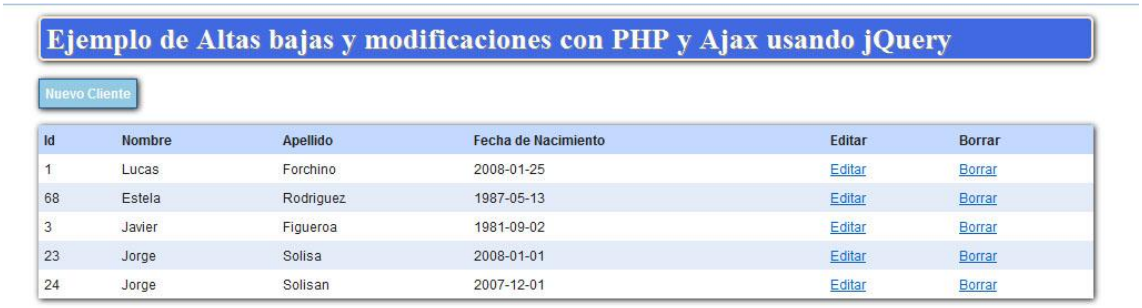
En esta Unidad los participantes se encontrarán con diferentes tipos de consignas que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

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.

EJEMPLO PRACTICO ABM PHP, JQUERY Y AJAX

En esta última unidad vamos a desarrollar un ABM (alta/baja/modificación) y consulta 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.



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.

Como notarán la baja no tiene confirmación los desafío a modificar el script para que muestre los datos seleccionados sin la posibilidad de modificarlos con la posibilidad de Borrar o Cancelar.

EJEMPLO PRACTICO CRUD PHP, PDO Y AJAX

Vamos a ver ahora un ejemplo completo de aplicación CRUD (Create-Read-Update-Delete) utilizando las sentencias preparadas de PDO.

Este ejemplo esta desarrollado con los métodos de Jquery para el uso de Aax y Bootstrap para el diseño de las vistas.

Si siguieron el ejemplo anterior se darán cuenta de que la lógica de este ejemplo es bastante similar al anterior, el cambio radica fundamentalmente en la parte de PHP y el modo de acceder a los datos de la tabla sql pero la parte de Ajax hecha a través de la función $.ajax o de algunos de los métodos derivados de esta como $.get o $.post de Jquery es muy parecida al ejemplo anterior.

En primer lugar una vez que esta listo el DOM se ejecuta la función readRecords():

```
$(document).ready(function () {

   // READ records on page load

   readRecords(); // calling function

});
```

Esta función carga en el div indicado el contenido devuelto por el script read.php

```
// READ records

function readRecords() {

   $.get("ajax/read.php", {}, function (data, status) {

      $(".records_content").html(data);

   });

}
```

El resto de las funciones dentro del archivo script.js responden al click asociado a los botones que se ven en la pagina.

Al hacer click en el botón Add Record se ejecuta la función addRecord() que llama mediante el método $.post al srcipt créate.php

Y lo mismo en el caso del Delete y del Update.

Como les dije anteriormente la lógica es siempre la misma. Es una metodología que se puede aplicar a diferentes datos.

Lo que cambia en este ejemplo con respecto al anterior es la parte del lado del servidor, de la cual voy a hacer una muy breve explicación aunque no es el objetivo del curso. Al que no le interese o no tenga conocimientos previos de PHP puede tomarlo como una caja negra, en la que a uds lo único que les interesa es que ese script php les devuelve una información que tienen que actualizar en su html o una respuesta que también tienen que mostrar en el html.

En el archivo db_connection.php encontraran la conexión a la base de datos utilizando PDO.

En el archivo Ajax/lib.php encontraran los métodos que realizan las acciones contra la base de datos para dar de alta, baja, modificar y consultar.

```
<?php

require __DIR__ . '/db_connection.php';

class CRUD
{

   protected $db;

   function __construct()
   {
      $this->db = DB();
   }

   function __destruct()
   {
      $this->db = null;
   }

   /*
    * Add new Record
    *
    * @param $first_name
    * @param $last_name
    * @param $email
    * @return $mixed
    * */
   public function Create($first_name, $last_name, $email)
   {
```

```
    $query = $this->db->prepare("INSERT INTO users(first_name, last_name, email)
VALUES (:first_name,:last_name,:email)");
    $query->bindParam("first_name", $first_name, PDO::PARAM_STR);
    $query->bindParam("last_name", $last_name, PDO::PARAM_STR);
    $query->bindParam("email", $email, PDO::PARAM_STR);
    $query->execute();
    return $this->db->lastInsertId();
  }

  /*
   * Read all records
   *
   * @return $mixed
   * */
  public function Read()
  {
    $query = $this->db->prepare("SELECT * FROM users");
    $query->execute();
    $data = array();
    while ($row = $query->fetch(PDO::FETCH_ASSOC)) {
      $data[] = $row;
    }
    return $data;
  }

  /*
   * Delete Record
   *
   * @param $user_id
   * */
  public function Delete($user_id)
  {
    $query = $this->db->prepare("DELETE FROM users WHERE id = :id");
    $query->bindParam("id", $user_id, PDO::PARAM_STR);
    $query->execute();
  }

  /*
   * Update Record
   *
   * @param $first_name
   * @param $last_name
   * @param $email
   * @return $mixed
   * */
  public function Update($first_name, $last_name, $email, $user_id)
  {
```

```
        $query = $this->db->prepare("UPDATE users SET first_name = :first_name,
last_name = :last_name, email = :email  WHERE id = :id");
        $query->bindParam("first_name", $first_name, PDO::PARAM_STR);
        $query->bindParam("last_name", $last_name, PDO::PARAM_STR);
        $query->bindParam("email", $email, PDO::PARAM_STR);
        $query->bindParam("id", $user_id, PDO::PARAM_STR);
        $query->execute();
    }

    /*
     * Get Details
     *
     * @param $user_id
     * */
    public function Details($user_id)
    {
        $query = $this->db->prepare("SELECT * FROM users WHERE id = :id");
        $query->bindParam("id", $user_id, PDO::PARAM_STR);
        $query->execute();
        return json_encode($query->fetch(PDO::FETCH_ASSOC));
    }
}

?>
```

Estos métodos son llamados cada uno desde su correspondiente archivo:

Create.php: llama al método create(…) de la clase CRUD para dar de alta un registro

Read.php: llama al método read() de la clase CRUD que lee los registros de la tabla.

Update.php: llama al método update(…..) de la clase CRUD modifica un registro en la tabla

Delete.php: llama al método delete(…) de la clase CRUD que da de baja el registroen cuestión

Details.php: llama al método details(…) de la clase CRUD para mostrar los detalles de un registro en particular y lo utiliza para mostrarlo al usuario para que modifique los datos.

Se utilizan sentencias preparadas de la extensión PDO de PHP. Veamos un poco que son y para que sirven las sentencias preparadas.

Una **Sentencia Preparada**, o **Prepared Statement**, es una característica de algunos **sistemas de bases de datos** que permite ejecutar la misma (o similar) **sentencia SQL**

provocando una mejora en la seguridad y en el rendimiento de la aplicación.

Las bases de datos MySQL soportan sentencias preparadas. Una sentencia preparada o una sentencia parametrizada se usa para ejecutar la misma sentencia repetidamente con gran eficiencia.

Flujo de trabajo básico

La ejecución de sentencias preparadas consiste en dos etapas: la preparación y la ejecución. En la etapa de preparación se envía una plantilla de sentencia al servidor de bases de datos. El servidor realiza una comprobación de sintaxis e inicializa los recursos internos del servidor para su uso posterior.

El servidor de MySQL soporta el uso de parámetros de sustitución posicionales anónimos con ?.

1. Cómo funcionan las sentencias preparadas

- **Prepare**. Primero una **plantilla de la sentencia SQL** se crea y se envía a la base de datos. Algunos valores se dejan sin especificar, llamados **parámetros** y representados por un **interrogante** "**?**":

INSERT INTO Clientes VALUES (?, ?, ?, ?);

- Después, la base de datos analiza, compila y realiza la optimización de la consulta sobre la sentencia SQL, y **guarda el resultado sin ejecutarlo**.
- **Execute**. Por último, **la aplicación enlaza valores con los parámetros**, y la base de datos ejecuta la sentencia. La aplicación puede entonces ejecutar la sentencia tantas veces como quiera con valores diferentes.

2. Ventajas de las Sentencias Preparadas

- Las **Sentencias Preparadas** reducen en **tiempo de análisis** ya que la preparación de la consulta de realiza sólo una vez (aunque la sentencia se ejecute las veces necesarias).
- Los **parámetros** enlazados minimizan el ancho de banda consumida del servidor ya que sólo necesitas enviar los parámetros cada vez, no la consulta entera.
- Las **Sentencias Preparadas** son muy últiles frente a Inyecciones SQL, ya que los valores de los parámetros, que son transmitidos después usando un protocolo diferente, no necesitan ser escapados. Si la plantilla de la sentencia original no es derivada de un **input externo**, los **ataques SQL Injection** no pueden ocurrir.

Vamos a ver un ejemplo de Sentencias Preparadas en PHP con PDO.

```
$server = "localhost";
$user = "usuario";
$password = "password";
$dbname = "ejemplo";
try {
// Conectar
$db = new PDO("mysql:host=$server;dbname=$dbname", $user, $password);
// Establecer el nivel de errores a EXCEPTION
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e){
    echo "Error: " . $e->getMessage();
}
```

Una vez establecida la conexión, podemos usarla donde la necesitemos:

```
// Preparar
$stmt = $db->prepare("INSERT INTO Clientes (nombre, ciudad, contacto)
VALUES (:nombre, :ciudad, :contacto)");
$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':ciudad', $ciudad);
$stmt->bindParam(':contacto', $contacto);
// Establecer parámetros y ejecutar
$nombre = "Donald Trump";
$ciudad = "Madrid";
$contacto = 4124124;
$stmt->execute();
$nombre = "Hillary Clinton";
$ciudad = "Barcelona";
$contacto = 4665767;
$stmt->execute();
// Mensaje de éxito en la inserción
echo "Se han creado las entradas exitosamente";
// Cerrar conexiones
$db = null;
```

Los parámetros también pueden enlazarse sin utilizar *bindParam()*, directamente con un array en *execute()*:

```
// Preparar
$stmt = $db->prepare("INSERT INTO Clientes (nombre, ciudad, contacto)
VALUES (:nombre, :ciudad, :contacto)");
// Establecer parámetros y ejecutar
$nombre = "Donald Trump";
$ciudad = "Madrid";
$contacto = 4124124;
$stmt->execute(array(':nombre' => $nombre, ':ciudad' => $ciudad,
':contacto' => $contacto));
// Mensaje de éxito en la inserción
echo "Se han creado las entradas exitosamente";
// Cerrar conexiones
$db = null;
```

Hemos asignado los parámetros con nombre de variable, pero también se puede hacer con números:

```
$stmt = $db->prepare("INSERT INTO Clientes (nombre, ciudad, contacto)
VALUES (?, ?, ?)");
```

```
$stmt->bindParam(1, $nombre);
$stmt->bindParam(2, $ciudad);
$stmt->bindParam(3, $contacto);
```

En caso de pasar los parámetros a través de un array con *execute()*, se haría de la misma forma que antes:

```
// Preparar
$stmt = $db->prepare("INSERT INTO Clientes (nombre, ciudad, contacto)
VALUES (?, ?, ?)");
// Establecer parámetros y ejecutar
$nombre = "Donald Trump";
$ciudad = "Madrid";
$contacto = 4124124;
$stmt->execute(array($nombre, $ciudad, $contacto));
// Mensaje de éxito en la inserción
echo "Se han creado las entradas exitosamente";
// Cerrar conexiones
$db = null;
```