cipsa.net

PHP & MySQL

Anexo 9.- AJAX

DISTRIBUIDO POR:

CENTRO DE INFORMÁTICA PROFESIONAL S.L.

C/ URGELL, 100
08011 BARCELONA
TFNO: 93 426 50 87

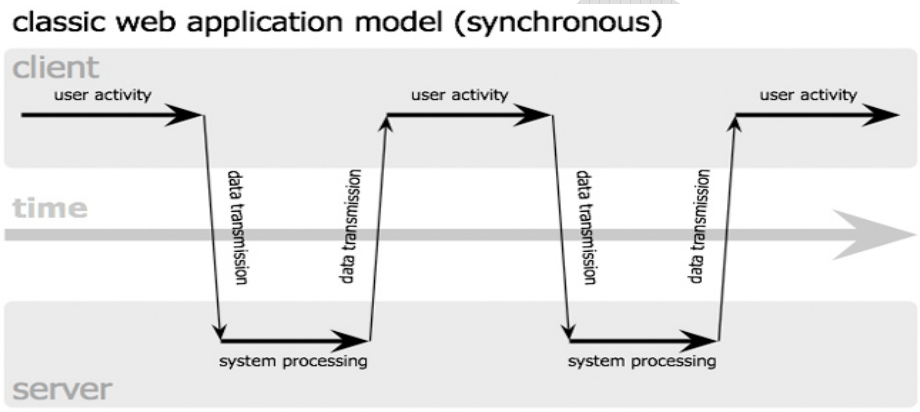
C/ RAFAELA YBARRA, 10
48014 BILBAO
TFNO: 94 448 31 33

www.cipsa.net

AJAX

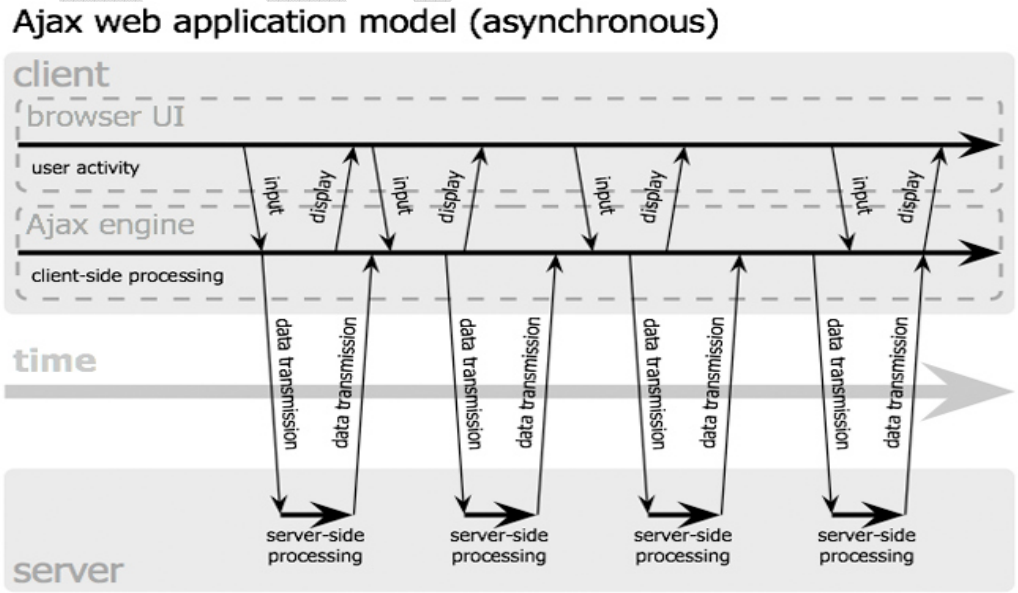
Modelo síncrono / asíncrono

Las aplicaciones web han funcionado tradicionalmente empleando el conocido como *modelo síncrono*. En este modelo tanto el envío de datos al servidor como la actualización de cualquier parte del contenido de una página web (por pequeño que fuese), requería la recarga completa de la página. Esta recarga implicaba realiza una petición al servidor para volver a obtener la página (petición HTTP_GET), o para enviar datos (petición HTTP_POST), y la obtención desde el servidor de la página actualizada al completo con el consiguiente consumo de tiempo y ancho de banda en la red.



Esquema de funcionamiento clásico síncrono

Con ***AJAX*** (*Asynchronous JavaScript and XML*) aparece el *modelo asíncrono*. Este modelo permite tanto el envío de datos al servidor como la actualización de parte del contenido de la página mediante sin necesidad de recargar la página al completo. Se utilizan para ello peticiones al servidor que se ejecutan en el contexto de la propia página web modificando su contenido sin recargarla.



La programación de AJAX se ha realizado tradicionalmente empleando el objeto *XMLHttpRequest*. Este objeto permite enviar una petición a una determinada URL y obtener la respuesta enviada por el servidor. Desgraciadamente la implementación de AJAX varía según el navegador lo que complicaba enormemente el código.

JQuery incluye métodos que implementan las operaciones de AJAX para todos los navegadores, de manera que no es necesario preocuparse por ello simplificando mucho el código necesario.

Ejecución asíncrona.

El concepto de *ejecución asíncrona* significa que cuando se realiza una petición al servidor mediante AJAX, la respuesta NO es inmediata. Por ello, las funciones de JQuery para AJAX requieren una función de retrollamada que se ejecuta cuando se recibe la respuesta del servidor y la procesan.

Ejemplo: La siguiente página muestra un botón que al pulsarse (evento *click*) envía una solicitud al script "*mensaje.php*" y muestra la respuesta enviada en la capa (id="capa"):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
    <title></title>
    <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
    <script>
        $().ready( function() {
            $("#boton").click(function() {
                $.get("mensaje.php", function(mensaje) {
                    $("#capa").html(mensaje);
                });
            });
        });
    </script>
</head>
<body>
<div id="capa">CAPA</div>
<input type="button" id="boton" value="MOSTRAR">
</body>
</html>
```

La función ***$.get()*** de JQuery envía una solicitud *HTTP_GET* a la URL indicada como primer argumento, y ejecuta la función indicada como segundo argumento cuando se obtiene la respuesta.

El script de PHP solicitado "*mensaje.php*" únicamente contiene el siguiente código:

```
<?php
        header('Content-Type: text/html; charset=utf-8');
        echo "MENSAJE DEL SERVIDOR";
?>
```

Funciones de petición AJAX

En AJAX se emplean básicamente dos tipos de peticiones para comunicarse con el servidor asíncronamente: Peticiones GET y POST.

Peticiones *GET*

Las peticiones GET son aquellas *en las que se pide al servidor una información*. Estas se realizan a través de una URL que puede llevar concatenados valores (*querystring*) indicando lo que se solicita.

JQuery dispone del método ***$.get()*** para realizar peticiones GET mediante AJAX y obtener los datos enviados como respuesta desde el servidor. El método consta de los siguientes parámetros:

jQuery.get(url [, data] [, success] [, dataType])

- *url* → Es la URL el script PHP al que va dirigida la petición.
- *data* → Datos para la petición. Puede ser una cadena o un objeto.
- *success* → Función a ejecutarse cuando se reciba la respuesta del servidor. Sus parámetros reciben los datos enviados por el servidor.
- *dataType* → Formato de los datos devueltos. Los valores posibles son *xml*, *text*, *json*, *html*, *script*.

El tipo del parámetro dato recibido en la función de retrollamada depende en gran medida del tipo MIME de la respuesta esperada del servidor y especificada en el parámetro ***data_type*** de las funciones ***$.get()***, ***$.post()***. Las opciones posibles son las siguientes:

- "***xml***" → Los datos son tratados como un documento XML procesable por JQuery.
- "***html***" → Los datos son tratados como texto incluyendo las etiquetas las cuales pueden insertarse como parte del documento de la página actual.
- "***text***" → Los datos son tratados como texto
- "***json***" → Los datos son tratados como JSON y se devuelve un objeto de Javascript.

Petición GET.

El siguiente código muestra una página en la que se visualiza *la fecha y hora actual del servidor* (no la del ordenador del usuario) refrescándose 1 vez por segundo sin recargar la página mediante AJAX.

tiempo.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
<script>
        $().ready(function() {
                setInterval(refrescar, 1000);
        });
        function refrescar() {
                $.get('tiempo.php', function( respuesta ) {
                        $('#capa').html( respuesta );
                });
        }
</script>
</head>
<body>
        <div id='capa'></div>
</body>
</html>
```

En el código se inicializar un temporizador que ejecuta 1 vez por segundo la función *refrescar()*. Esta función envía una petición al script '*tiempo.php*' del servidor. La función de retrollamada recibe por el parámetro *respuesta* el código respondido por el servidor y lo asigna como contenido a la capa *id='capa'* para su visualizar la hora.

El script llamado '*tiempo.php*' envía ***código HTML*** que es el obtenido por la función de retrollamada y añadido al contenido de la capa:

tiempo.php

```
<?php
        header('Content-Type: text/html; charset=utf-8');
        $date = new DateTime();
        list($y, $M, $d, $h, $m, $s) = explode(' ', $date->format("Y m d H i s"));
        echo "<p>Fecha: $d/$M/$y</p>";
        echo "<p>Hora: $h:$m:$s</p>";
```

Petición GET con datos

La función ***$.get()*** también puede enviar peticiones junto con datos concatenados en la URL. Estos parámetros deben indicarse como un objeto definido en JSON:

Ejemplo: Vamos a modificar el código del script *'tiempo.php'* de modo que reciba dos parámetros con el área y la localidad de la que se desea obtener la fecha y hora actuales. La URL de solicitud para obtener la fecha y hora actuales en Madrid sería:

```
tiempo.php?timezone=Europe/Madrid
```

tiempo.php

```php
<?php
	header('Content-Type: text/html; charset=utf-8');
	$tmzone = $_GET['timezone'];
	$date = new DateTime("now", new DateTimeZone($tmzone));
	list($y, $M, $d, $h, $m, $s) = explode(' ', $date->format("Y m d H i s"));
	echo "<p>Fecha: $d/$M/$y</p>";
	echo "<p>Hora: $h:$m:$s</p>";
```

La llamada al método ***$.get()*** debe modificarse indicando valores para los parámetros '*area*' y '*loc*' en la URL de petición:

tiempo.html

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
<script>
	$().ready(function() {
		setInterval(refrescar, 1000);
	});
	function refrescar() {
		$.get('tiempo.php',
			{'timezone': 'Europe/Madrid'},
			function( respuesta ) {
				$('#capa').html( respuesta );
			});
	}
</script>
</head>
<body>
	<div id='capa'></div>
</body>
</html>
```

Petición GET con recepción de datos codificados con *JSON*

En el ejemplo anterior la respuesta dada por el script invocado era directamente código HTML mostrado en una capa de la página web.

Existen ocasiones en las que lo que se desea obtener no es código HTML sino datos. Estos datos procederán del servidor como respuesta a la petición y pueden ser codificados como *texto*, *json*, *xml*… etc. Lo que se haga con éstos datos o cómo se muestren depende del código *Javascript* de la propia página, no del servidor.

Ejemplo 1: Supóngase el siguiente script "*tiempo.php*" que devuelve un objeto con 6 propiedades (***anyo***, ***mes***, ***dia***, ***hora***, ***minuto***, y ***segundo***) y sus valores correspondientes codificado en *JSON*:

tiempo.php

```php
<?php
	header('Content-Type: text/html; charset=utf-8');
	$tmzone = $_GET['timezone'];
	$date = new DateTime("now", new DateTimeZone($tmzone));
	 // Creacion del objeto a enviar como respuesta.
	 $obj = (object) [
				'anyo' => $date->format("Y"),
				'mes' => $date->format("m"),
				'dia' => $date->format("d"),
				'hora' => $date->format("H"),
				'minuto' => $date->format("i"),
				'segundo' => $date->format("s"),
	 ];
	// Envio del objeto codificado con JSON
	echo json_encode($obj);
```

Si ejecutamos directamente el script PHP con la siguiente URL:

tiempo.php?timezone=Europe/Madrid

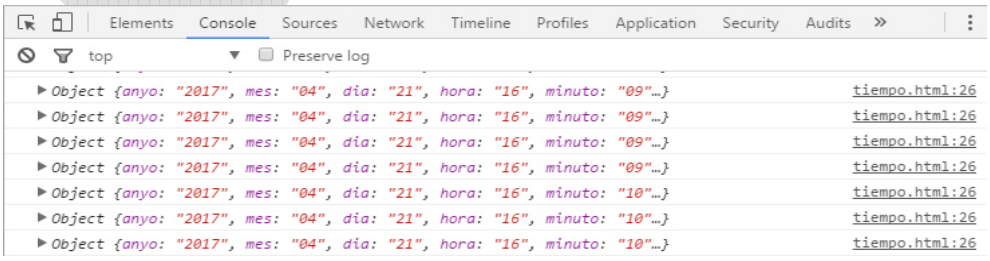
Obtendremos un fichero con el objeto creado en el script expresado en JSON.

```
{"anyo":"2017",
  "mes":"04",
  "dia":"21",
  "hora":"16",
  "minuto":"03",
  "segundo":"35"}
```

Para obtener el objeto correspondiente desde JQuery mediante AJAX debemos añadir el argumento "*json*" al método ***$.get()***.

```
	function refrescar() {
		$.get('tiempo.php',
				{'timezone', 'Europe/Madrid'},
				function( obj ) {
					console.log(obj);
				}, "json");
	}
```

El resultado mostrado por consola indica que ahora el parámetro *obj* de la función de retrollamada ya no es una cadena con código HTML, sino un objeto:



La página web anterior se modifica ahora parece recibir no HTML sino el objeto creado en el script "*tiempo.php*" con los atributos (*anyo*, *mes*, *dia*, *hora*, *minuto* y *segundo*).

Desde la función de retrollamada del método ***$.get()*** se recupera el objeto y se muestran sus atributos mediante *JQuery* en las celdas de la tabla.

tiempo.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
<style>
        td {    background-color: #ffff00;
                text-align: center}
        th {    width: 100px;
                background-color: #eeeeee}
</style>
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
<script>
        $().ready(function() {
                setInterval(refrescar, 1000);
        });
        function refrescar() {
                $.get('tiempo.php',
                                {'timezone': 'Europe/Madrid'},
                                function( obj ) {
                                        // Obtencion del objeto y visualización de sus atributos
                                                $('#anyo').html(obj.anyo);
                                                $('#mes').html(obj.mes);
                                                $('#dia').html(obj.dia);
                                                $('#hora').html(obj.hora);
                                                $('#minuto').html(obj.minuto);
                                                $('#segundo').html(obj.segundo);
                                }, "json");
        }
</script>
</head>
<body>
        <div id='capa'>
                <table>
                        <tr>
                                <th>AÑO</th>
                                <th>MES</th>
                                <th>DIA</th>
                                <th>HORA</th>
                                <th>MINUTO</th>
                                <th>SEGUNDO</th>
                        </tr>
                        <tr>
                                <td id='anyo'></td>
                                <td id='mes'></td>
                                <td id='dia'></td>
                                <td id='hora'></td>
                                <td id='minuto'></td>
                                <td id='segundo'></td>
                        </tr>
                </table>
        </div>
</body>
</html>
```

JQuery dispone también del método ***$.getJSON(),*** que es específico para solicitar peticiones AJAX en la que se espera una respuesta con datos codificados en JSON.

La función ***refrescar()*** de la página anterior podría codificarse del siguiente modo:

```
        function refrescar() {
                $.getJSON('tiempo.php',
                                {'timezone': 'Europe/Madrid'},
                                function( obj ) {
                                        console.log(obj);
                                        $('#anyo').html(obj.anyo);
                                        $('#mes').html(obj.mes);
                                        $('#dia').html(obj.dia);
                                        $('#hora').html(obj.hora);
                                        $('#minuto').html(obj.minuto);
                                        $('#segundo').html(obj.segundo);
                                });
        }
```

Peticiones *POST* (*$().post()*)

Las peticiones POST *son solicitudes en las que se envía información al servidor*. Esto también puede hacerse en las peticiones GET, pero se considera más apropiado hacerlo con POST cuando hay mucha información o es delicada (claves, datos de altas o modificaciones…, etc).

JQuery dispone del método ***$.post()*** para realizar peticiones POST. Los parámetros son los mismos que los del método *$.get()*:

jQuery.post(url [, data] [, success] [, dataType])

- *url* → Es la URL el script PHP al que va dirigida la petición.
- *data* → Datos conjuntos de la petición. Puede ser un objeto o una cadena.
- *success* → Función a ejecutarse cuando se reciba la respuesta del servidor. Sus parámetros reciben los datos enviados por el servidor.
- *dataType* → Formato de los datos devueltos. Los valores posibles son *xml*, *text*, *json*, *html*, *script*.

Ejemplo: La siguiente función envía una petición POST al script "*usuarios.php*" pasando un parámetro '*nombre'* con valor '*Oscar'*. La respuesta enviada por el script es recibida por el parámetro *datos* de la función de retrollamada.

```
$.post( "usuarios.php", {'nombre': 'Oscar'}, function( datos ) {
          alert( "Exito" );
        });
```

Suele ser habitual emplear las peticiones POST para enviar los valores de un formulario. Esto también suele hacerse con AJAX empleando una serie las funciones de JQuery ***$().serialize()*** y ***$().serializeArray()***. En ambos casos, el valor serializado puede recuperarse desde un script de PHP y ser deserializado para obtener los valores del formulario.

Al igual que en las funciones ***$.get()*** y ***$.getJSON()***; la respuesta enviada por el servidor se recibe a través del primer parámetro de la función de retrollamada.

Petición POST empleando *$().serializeArray()*

Supóngase el siguiente formulario en el que se muestran una caja de texto (*name='caja_texto'*), varios botones de opción (*name='opciones'*), casillas de verificación (*name='casilla1'* y *name='casilla2'*), y dos listas de selección simple (*name='lista_simple'*) y múltiple (*name='lista_multiple*).

El código javascript captura el evento de envío del formulario (*submit*), serializa los datos del formulario mediante la función ***$().serializeArray()***, y los envía al script '*formulario.php*' mediante ***$().post()*** con el identificador '*datos*'. El resultado devuelto por el script es mostrado en la capa *id='vista'*:

Código página: "*formulario.html*"

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
</head>
<script src="scripts/jquery-3.0.0.min.js"></script>
 <script>
        $().ready( function() {
                $( "form" ).on( "submit", function( event ) {
                        event.preventDefault();
                        // Serializacion de datos
                        var datos = $( this ).serializeArray();
                          console.log(datos);
                        // Envio de datos serializados mediante POST
                        $.post('formulario.php', {'datos': datos},
                                        function(respuesta) {
                                                $('#vista').html(respuesta);
                        });
                });
        });
 </script>
<body>
        <div id='vista'>
        </div>
        <form>
                <p>Texto</p>
                <input type='text' name='caja_texto'><br/>
                <p>Optiones:</p>
                opcion 1<input type='radio' name='opciones' value='opcion 1'>
                opcion 2<input type='radio' name='opciones' value='opcion 2'>
                opcion 3<input type='radio' name='opciones' value='opcion 3'>
                <p>Casillas:</p>
                casilla 1<input type='checkbox' name='casilla1'>
                casilla 2<input type='checkbox' name='casilla2'>
                <p>Lista:</p>
                <select name='lista_simple'>
                        <option value='opcion1'>opcion 1</option>
                        <option value='opcion2'>opcion 2</option>
                        <option value='opcion3'>opcion 3</option>
                        <option value='opcion4'>opcion 4</option>
                </select>
                <p>Lista Multiple</p>
                <select name='lista_multiple[]' multiple>
                        <option value='opcion1'>opcion 1</option>
                        <option value='opcion2'>opcion 2</option>
                        <option value='opcion3'>opcion 3</option>
                        <option value='opcion4'>opcion 4</option>
                </select>
                <br/><br/>
                <input type='submit' value='enviar'>
        </form>
</body>
</html>
```

Al pulsar enviar, la sentencia *console.log(datos)* de Javascript muestra en la consola del Google Chrome la matriz generada por la función *$().serializeArray()*:

Cada control del formulario es representado por un objeto con un atributo '*name*', y el valor correspondiente en el atributo '*value*'. Todos los objetos se almacenan a su vez en una matriz convencional.

En un script de PHP pueden deserializarse los datos para convertirlos en una matriz asociativa del mismo tipo que *$_POST*.

Código script "*formulario.php*"

```php
<?php
// Funcion que deserializa la matriz generada
// por la función $().serializeArray() de JQuery
function JQUnserializeArray( $datos ) {
        $matriz = Array();
        foreach( $datos as $dato ) {
                $clave = $dato['name'];
                $valor = $dato['value'];
                if ( substr($clave, -2) === "[]" ) {
                        if ( !isset($matriz[$clave])) {
                                $matriz[$clave]= array();
                        }
                        array_push($matriz[$clave], $valor);
                } else {
                        $matriz[$clave] = $valor;
                }
        }
        return $matriz;
}

header('Content-Type: text/html; charset=utf-8');
// Obtencion de los datos enviados serializados
$datos_serializados = $_POST['datos'];
// Llamada a funcion deserializadora.
// Obtencion de matriz asociativa equivalente.
$matriz = JQUnserializeArray($datos_serializados);
var_dump($matriz);
```

La matriz generada por la función ***JQunserializeArray()*** sería equivalente a la que tendría $_POST en caso de haberse enviados los datos del formulario sin AJAX:

```
C:\xampp7\htdocs\prueba\ajax\formulario.php:5:
array (size=5)
  'caja_texto' => string 'cadena' (length=6)
  'opciones' => string 'opcion 2' (length=8)
  'casilla1' => string 'on' (length=2)
  'lista_simple' => string 'opcion2' (length=7)
  'lista_multiple[]' =>
    array (size=2)
      0 => string 'opcion2' (length=7)
      1 => string 'opcion3' (length=7)
```

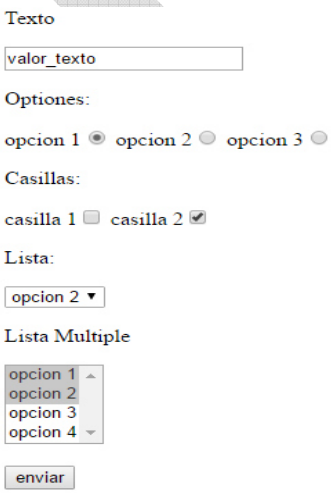
var_dump() de la matriz retornada por la función JQunserializeArray() del script de PHP

Petición POST empleando *$().serialize()*

Modificamos el código javascript de la página '*formulario.html*' anterior para utilizar la función *$().serialize()* para serializar los datos del formulario y enviarlos con el identificador '*datos*' al script '*formulario.php*' mediante *$().post()*:

```
<script>
        $().ready( function() {
                $( "form" ).on( "submit", function( event ) {
                        event.preventDefault();
                         // Serializacion de datos
                         var datos = $( this ).serialize();
                        console.log(datos);
                        // Envio de datos serializados
                        $.post('formulario.php', {'datos': datos},
                                        function(datos) {$('#vista').html(datos);});
                });
        });
</script>
```

Al pulsar enviar, la sentencia *console.log(datos)* de Javascript muestra en la consola del Google Chrome la matriz generada por la función *$().serialize()*:




```
caja_texto=valor_texto&opciones=opcion%201&casilla2=on&lista_s   formulario.html:14
imple=opcion2&lista_multiple%5B%5D=opcion1&lista_multiple%5B%5D=opcion2
```

En el caso de la función de JQuery ***$().serialize()***, los valores de los campos del formulario se serializan en una cadena de texto separados por &. El valor de cada control se expresa indicando su *name* seguido de '=' y el valor correspondiente.

En un script de PHP pueden deserializarse los datos para convertirlos en una matriz asociativa del mismo tipo que *$_POST* empleando la función ***JQUnserialize():***

```php
<?php
// Funcion que deserializa la matriz generada
// por la función $().serializeArray() de JQuery
// Decodifica la URL en una matriz asociativa de parámetros
function JQUnserialize( $cadena_datos ) {
        $matriz = array();
        foreach (explode('&', $cadena_datos ) as $dato) {
                $param = explode("=", $dato);
                if ($param) {
                        $clave = urldecode($param[0]);
                        $valor = urldecode($param[1]);
                        if ( substr($clave, -2) === "[]" ) {
                                if ( !isset($matriz[$clave])) {
                                        $matriz[$clave]= array();
                                }
                                array_push($matriz[$clave], $valor);
                        } else {
                                $matriz[$clave] = $valor;
                        }
                }
        }
        return $matriz;
}

header('Content-Type: text/html; charset=utf-8');
// Obtencion de los datos enviados serializados
$datos_serializados = $_POST['datos'];
// Llamada a funcion deserializadora.
// Obtencion de matriz asociativa equivalente.
$matriz = JQUnserialize($datos_serializados);
var_dump($matriz);
```

La matriz devuelta por la función *JQunserialize()* sería equivalente a la que tendría *$_POST* en caso de haberse enviados los datos del formulario sin AJAX:

```
C:\xampp7\htdocs\prueba\ajax\formulario.php:34:
array (size=5)
  'caja_texto' => string 'valor_texto' (length=11)
  'opciones' => string 'opcion 1' (length=8)
  'casilla2' => string 'on' (length=2)
  'lista_simple' => string 'opcion2' (length=7)
  'lista_multiple[]' =>
    array (size=2)
      0 => string 'opcion1' (length=7)
      1 => string 'opcion2' (length=7)
```

var_dump() de la matriz retornada por la función JQunserialize() del script de PHP

La función *$().load()*

Esta función de JQuery realiza una petición asíncrona al servidor del mismo modo que *$.get()*. Sin embargo, ésta función *se invoca a partir de una selección* para incluir el código HTML enviado por el servidor en el/los elementos seleccionados.

jQuery.post(url [, data] [, success])

- *url* → Es la URL al que va dirigida la petición.
- *data* → Datos conjuntos de la petición.
- *success* → Función a ejecutarse cuando se reciba la respuesta del servidor. Sus parámetros reciben los datos enviados por el servidor.

Ejemplo: El siguiente código mostraría en la capa con *id='resultado'* el contenido de la página *prueba.html*.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
<script src="scripts/jquery-3.0.0.min.js"></script>
<script>
	$().ready( function() {
		$('#resultado').load("load.html", function() {
			alert('cargado');
		});
	});
</script>
</head>
<body>
<div id="resultado"></div>
</body>
</html>
```

Si se incluye una función de retrollamada, ésta se ejecuta una vez por cada elemento seleccionado tras insertarle el código HTML empleando la propiedad ***innerHTML***. Se puede emplear ***this*** en la función para obtener al elemento rellenado en cada ejecución.

El tipo de petición enviada por esta función puede ser *GET* o *POST* según el tipo de dato indicado como parámetro. Si se indica como parámetro *data* un *objeto* se emplea una petición *POST*. Si se indica una *cadena* o se omite se emplea una petición *GET*.

Restricción de origen

Todas las funciones de AJAX están limitadas por las restricciones de seguridad impuestas por los navegadores. Esto impide el envío de peticiones hacia URLs en diferentes dominios, subdominios, puestos o protocolos.

Tratamiento de errores en peticiones AJAX

Existen situaciones en las que una petición AJAX puede fallar y recibir un código de error desde el servidor como respuesta. Estas situaciones es necesario detectarlas y para ello pueden emplearse dos mecanismos.

Parámetros adicionales en la función de retrollamada

La función de retrollamada de funciones como ***$.get()***, ***$.getJSON()***, ***$.post()***, y ***$.load()***; admite dos parámetros adicionales al ya visto con la respuesta devuelta por el servidor:

function(data, textStatus, jqxhr)

- ***data*** → Es el valor devuelto por el servidor. (el tipo varía)
- ***textStatus*** → Cadena de texto con el estado de ejecución de la petición.
- ***jqXHR*** → Objeto que representa el objeto *XMLHttpRequest* empleado de manera nativa por el navegador para la ejecución de las peticiones AJAX. Este objeto dispone de la propiedad ***status*** y ***statusText*** que contienen respectivamente el código HTTP del paquete de respuesta enviado por el servidor y un mensaje informativo sobre el mismo.

La función de retrollamada sólo se ejecuta cuando la petición es satisfactoria en las funciones como ***$.get()***, ***$.post()***, ***$.getJSON()***, pero no en la función ***$().load()*** donde también se ejecuta en caso de fallo permitiendo obtener indicaciones.

Ejemplo: Uso de la función ***$().load()*** para cargar en ejecución el contenido de la página '*load.html*' en la capa (*id='resultado'*)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
<script src="scripts/jquery-3.0.0.min.js"></script>
 <script>
        $().ready( function() {
                $('#resultado').load("load.html",
                        function(data, text, XHR) {
                                if ( XHR.status == 200) {
                                        // Código HTTP 200 → OK
                                        alert("OK");
                                } else {
                                        alert("Error: " + XHR.statusText);
                                }
                });
        });
 </script>
</head>
<body>
<div id="resultado"></div>
</body>
</html>
```

Objeto retornado *jqXHR*

Las funciones AJAX de JQuery (excepto ***$().load()***) devuelven un objeto ***jqXHR*** que representa el objeto *XMLHttpRequest* empleado por el navegador para enviar la petición al servidor. Este objeto provee una serie de *funciones diferidas* a las que puede invocarse pasando como argumento funciones de retrollamada para su ejecución en ciertas situaciones:

- ***jqXHR.done()*** → Se ejecuta cuando la petición AJAX se resuelve satisfactoriamente.
- ***jqXHR.fail()*** → Se ejecuta en caso de producirse un error.
- ***jqXHR.always()*** → Se ejecuta en ambos casos.

Las funciones pueden encadenarse para indicar funciones de retrollamada para todas ellas en la propia llamada a la función AJAX y recibir los siguientes parámetros:

function(jqXHR, textStatus)

- ***textStatus*** → Cadena de texto con el estado de ejecución de la petición.
- ***jqXHR*** → Objeto que representa el objeto *XMLHttpRequest* empleado de manera nativa por el navegador para la ejecución de las peticiones AJAX. Este objeto dispone de la propiedad ***status*** y ***statusText*** que contienen respectivamente el código HTTP del paquete de respuesta enviado por el servidor y un mensaje informativo sobre el mismo.

Ejemplo: El siguiente código muestra el uso de la función ***$.get()*** para obtener el código HTML de la página '*load.html*' e insertarla en la capa (*id='resultado'*).

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
<script src="scripts/jquery-3.0.0.min.js"></script>
 <script>
        $().ready( function() {
                var jqxhr = $.get("load.html", function(result) {
                          $('#resultado').html(result);
                        }, "html")
                          .done(function() {
                            alert( "success" );     // Se ejecuta si la petición OK
                          })
                          .fail(function(jqXHR, textStatus) {
                            alert("Error: " + jqXHR.statusText);   // Error
                          })
                          .always(function() {
                            alert( "finished" );    // Se ejecuta siempre tras petición
                          });
        })
 </script>
</head>
<body>
<div id="resultado"></div>
</body>
</html>
```

Orden de ejecución de funciones.

Si la petición se ejecuta correctamente, el orden de ejecución de las funciones de retrollamada es el siguiente:

1. ***$.get()***
2. ***jqXHR.done()***
3. ***jqXHR.always()***

Si por el contrario la petición sufre un error, el orden de ejecución es el siguiente:

1. ***jqXHR.error()***
2. ***jqXHR.always()***

Para más información sobre el tratamiento de errores en peticiones AJAX mediante funciones JQuery, puede accederse a la siguiente web:

https://cybmeta.com/manejo-de-errores-en-jquery-ajax

Ejemplos resueltos

Ejemplo 1:

Supóngase que deseamos modificar el ejemplo de obtención de la fecha del servidor mediante AJAX de modo que el usuario pueda seleccionar el área y localidad de la que quiere conocer la hora actual. Para ello podemos colocar dos listas:

Areas: Elige un área ▾ Localidades: Elegir ▾

AÑO	MES	DIA	HORA	MINUTO	SEGUNDO
2017	05	02	13	40	36

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
<style>
        td {    background-color: #ffff00;  text-align: center }
        th {    width: 100px; background-color: #eeeeee}
</style>
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
<script>
</script>
</head>
<body>
        Areas: <select id="area">
                <option value='-1'>Elegir</option>
                <option value='2'>A</option>
                <option value='3'>Elegir</option>
        </select>
        Localidades: <select id="localidad">
                <option value='-1'>Elegir</option>
        </select>
        <div id='capa'>
                <table>
                        <tr>
                                <th>AÑO</th>
                                <th>MES</th>
                                <th>DIA</th>
                                <th>HORA</th>
                                <th>MINUTO</th>
                                <th>SEGUNDO</th>
                        </tr>
                        <tr>
                                <td id='anyo'></td>
                                <td id='mes'></td>
                                <td id='dia'></td>
                                <td id='hora'></td>
                                <td id='minuto'></td>
                                <td id='segundo'></td>
                        </tr>
                </table>
        </div>
</body>
</html>
```

La lista de *Áreas* muestra las áreas del planeta geográficas del planeta, mientras que la lista *Localidades* muestra las localidades correspondientes al área seleccionada. Ambas podemos rellenarlas empleando AJAX y la función ***$.getJSON()***:

Para obtener la lista de Áreas se define un script de PHP "*areas.php*" que devuelve una matriz de asociativa cadenas con todas las áreas seleccionables y sus respectivos valores identificativos codificadas en JSON.

Código: *areas.php*

```php
<?php
	header('Content-Type: application/json; charset=utf-8');
	if (!isset($_GET['area'])) {
		$areas[0] = 'Elige un área';
		$areas[1] = 'Africa';
		$areas[2] = 'America';
		$areas[4] = 'Antarctica';
		$areas[8] = 'Arctic';
		$areas[16] = 'Asia';
		$areas[32] = 'Atlantic';
		$areas[64] = 'Australia';
		$areas[128] = 'Europe';
		$areas[256] = 'Indian';
		$areas[512] = 'Pacific';
		echo json_encode($areas);
	}
?>
```

Código JSON enviado:

```json
{
   0: "Elige un área",
   1: "Africa",
   2: "America",
   4: "Antarctica",
   8: "Arctic",
   16: "Asia",
   32: "Atlantic",
   64: "Australia",
   128: "Europe",
   256: "Indian",
   512: "Pacific"
}
```

Para cargar la lista de Áreas se lanza una petición GET al script '*areas.php*' mediante la función ***$.getJSON()*** de JQuery. Esta recibe la matriz de cadenas enviada por el script PHP y la pasa a la función ***cargarAreas*** que las añade a la lista (*id='area'*)

Código: javascript en *tiempo.html*

```javascript
	$().ready(function() {
		// Cargar areas
		$.getJSON('areas.php', cargarAreas);
	});
	// Refresca lista de áreas
	function cargarAreas( areas ) {
		// Obtencion lista HTML id='area'
		var select = $('#area');
		// Obtencion opciones
		var options = select.prop('options');
		// Agregado de areas como opciones de la lista
		$.each(areas, function(indice, valor) {
		    options[options.length] = new Option(valor, indice);
		});
	}
```

Al seleccionar una determinada área geográfica, debe refrescarse la lista de localidades con las correspondientes. Para ello creamos un script PHP que reciba mediante una petición GET el identificador de un área, y retorna una matriz escalar de cadenas con las localidades correspondientes codificada en JSON.

Código: *areas.php*

```php
<?php
	header('Content-Type: application/json; charset=utf-8');
	if (!isset($_GET['area'])) {
		$areas[0] = 'Elige un área';
		$areas[1] = 'Africa';
		$areas[2] = 'America';
		$areas[4] = 'Antarctica';
		$areas[8] = 'Arctic';
		$areas[16] = 'Asia';
		$areas[32] = 'Atlantic';
		$areas[64] = 'Australia';
		$areas[128] = 'Europe';
		$areas[256] = 'Indian';
		$areas[512] = 'Pacific';
		echo json_encode($areas);
	} else {
		// Obtencion del área seleccionada
		$area = $_GET['area'];
		// Obtencion de las localidades del área indicada
		$localidades = DateTimeZone::listIdentifiers($area);
		echo json_encode($localidades);
	}
?>
```

Código JSON enviado por el script '*area.php*' cuando se selecciona el área geográfica de Europa (*area.php?area=128*)

```
    "Europe/Amsterdam",
    "Europe/Andorra",
    "Europe/Astrakhan",
    "Europe/Athens",
    "Europe/Belgrade",
    "Europe/Berlin",
    "Europe/Bratislava",
    "Europe/Brussels",
    "Europe/Bucharest",
```

Cuando se selecciona un elemento en la lista de Áreas (evento ***change***), se ejecuta la función ***$.getJSON()*** llamando al script '*areas.php*' con el valor del área seleccionada. El resultado es enviado a la función ***refrescarLocalidades(),*** que muestra las localidades en la lista eliminando primero las existentes.

Código: javascript en "*tiempo.html*"

```javascript
	$().ready(function() {
		$.getJSON('areas.php', cargarAreas);
		// Evento de seleccion de area y carga de localidades
		$('#area').change(function() {
			area = $('#area :selected').val();
			$.getJSON("areas.php", {'area' : area}, cargaLocalidades);
		});
	});
	// Refresca lista de localidades
	function cargaLocalidades( localidades ) {
		var select = $('#localidad');
		var options = select.prop('options');
		// Eliminación de opciones existentes
		$('option', select).remove();
		// Agregado de localidad como opciones de la lista
		$.each(localidades, function(indice, valor) {
		    options[options.length] = new Option(valor, indice);
		});
	}
```

Cuando se selecciona un elemento en la lista de localidades (evento *change*), se modifica la variable de localidad seleccionada (*tmzone*). Esta es la que se emplea la función ***refrescar()*** para solicitar la hora al servidor una vez por segundo.

Código completo.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
<style>
        td {background-color: #ffff00;text-align: center}
        th {width: 100px;  background-color: #eeeeee}
</style>
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
 <script>
         var area;
         var tmzone = "Europe/Madrid";   // Localidad por defecto
         $().ready(function() {
                  // Cargar areas
                  $.getJSON('areas.php', cargarAreas);
                  // Evento de seleccion de area y carga de localidades
                  $('#area').change(function() {
                           area = $('#area :selected').val();
                           $.getJSON("areas.php", { 'area' : area}, cargaLocalidades);
                  });
                 // Evento de localidad
                  $('#localidad').change(function() {
                           tmzone = $('#localidad :selected').text();
                  });
                  setInterval(refrescar, 1000);
         });
         // Refresca lista de áreas
         function cargarAreas( areas ) {
                  // Obtencion lista HTML areas
                  var select = $('#area');
                  // Obtencion opciones
                  var options = select.prop('options');
                  // Agregado de areas como opciones de la lista
                  $.each(areas, function(indice, valor) {
                      options[options.length] = new Option(valor, indice);
                  });
         }
         // Refresca lista de localidades
         function cargaLocalidades( localidades ) {
                  // Obtención lista HTML localidades
                  var select = $('#localidad');
                  // Obtenbcion opciones
                  var options = select.prop('options');
                  // Eliminación de opciones existentes
                  $('option', select).remove();
                  // Agregado de localidad como opciones de la lista
                  $.each(localidades, function(indice, valor) {
                      options[options.length] = new Option(valor, indice);
                  });
         }
         // Actualiza la hora actual conforme a la localidad seleccionada
         function refrescar() {
                  $.getJSON('tiempo.php',
                                    {'timezone': tmzone},
                                    function( obj ) {
                                             console.log(obj);
                                              $('#anyo').html(obj.anyo);
                                              $('#mes').html(obj.mes);
                                              $('#dia').html(obj.dia);
                                              $('#hora').html(obj.hora);
                                              $('#minuto').html(obj.minuto);
                                              $('#segundo').html(obj.segundo);
                                    });
         }
 </script>
</head>
<body>
        Areas: <select id="area">
                 <option value='-1'>Elegir</option>
                 <option value='2'>A</option>
                 <option value='3'>Elegir</option>
        </select>
        Localidades: <select id="localidad">
                 <option value='-1'>Elegir</option>
        </select>

        <div id='capa'>
                 <table>
                          <tr>
                                    <th>AÑO</th>
                                    <th>MES</th>
                                    <th>DIA</th>
```

```
                                    <th>HORA</th>
                                    <th>MINUTO</th>
                                    <th>SEGUNDO</th>
                            </tr>
                            <tr>
                                    <td id='anyo'></td>
                                    <td id='mes'></td>
                                    <td id='dia'></td>
                                    <td id='hora'></td>
                                    <td id='minuto'></td>
                                    <td id='segundo'></td>
                            </tr>
                    </table>
            </div>
</body>
</html>
```

Ejemplo 2:

Supóngase que una página en la que deseamos gestionar los usuarios autorizados para acceder a la aplicación bancaria de ejemplos anteriores. Para ello empleamos la tabla "***USUARIOS***" de la base de datos ***"banco"***.

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado
☐	1	USUARIO	varchar(10)			No	*Ninguna*
☐	2	CLAVE	varchar(10)			No	*Ninguna*
☐	3	ADMINISTRADOR	tinyint(1)			No	0

La página muestra una tabla con el nombre y clave de todos los usuarios registrados junto con un formulario para dar de alta un usuario junto con su contraseña:

USUARIO	CLAVE
roger	1234
yuri	9875

Usuario:
Clave:
GUARDAR

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<style>
        td {background-color: #ffff00;text-align: center}
        th {width: 100px;       background-color: #eeeeee}
</style>
<title>Insert title here</title>
</head>
<body>
        <div id='capa'>
        <table id='tbl_usuarios'>
                <tr><th>USUARIO</th><th>CLAVE</th></tr>
        </table>
        </div>
        <form>
                Usuario: <input type='text' name='usuario'><br/>
                Clave: <input type='password' name='clave'><br/>
                <input type='submit' value='GUARDAR'>
        </form>
</body>
</html>
```

Para obtener una lista con el nombre de usuario y contraseña de todos los usuarios registrados empleamos el script "*listusuarios.php*". Este retorna una matriz de objetos con el nombre de usuario y contraseña de cada usuario registrado codificado en JSON.

Código: *listusuarios.php*

```
<?php
        require_once 'classes/db.class.php';
        header('Content-Type: application/json; charset=utf-8');
        // Devuelve matriz de objetos usuario con todos los usuarios.
        function obtenerUsuarios() {
                $resultados = DB::run('SELECT USUARIO, CLAVE FROM USUARIOS');
                return $resultados->fetchAll(PDO::FETCH_OBJ);
        }
        $usuarios = obtenerUsuarios();
        echo json_encode($usuarios);
?>
```

La página envía una petición GET mediante la función ***$.getJSON()*** al script "*listusuarios.php*" y pasa la matriz de objetos recibida a la función ***refrescar()***.

Esta función elimina las filas presentes en la tabla (*id='tbl_usuarios'*), y a continuación crea y añade filas con el nombre de usuario y contraseña de cada usuario registrado. Si la función recibe como valor un objeto en vez de una matriz se interpreta que ser ha producido un error cuya causa está en el atributo '*error*':

Código: javascript en "*usuarios.html*".

```
<script>
        $().ready( function() {
                $.getJSON('listusuarios.php', refrescar);
        });
        function refrescar( usuarios ){
                // Se ha devuelto una matriz?
                if ( !$.isArray(usuarios)) {
                        alert("Se produjo un error", usuarios.error);
                        return;
                }
                // Eliminar filas existentes
                $('#tbl_usuarios tbody tr').not(':first').remove();
                // Generacion de filas por cada usuario
                $(usuarios).each(function( index, usuario ) {
                        $("#tbl_usuarios").find('tbody')
                    .append($('<tr>')
                        .append($('<td>')
                                .html(usuario.USUARIO))
                         .append($('<td>')
                                .html(usuario.CLAVE))
                    );
                });
        }
</script>
```

Cuando el usuario introduce un nombre de usuario y una contraseña en el formulario y pulsa el botón "*GUARDAR*" se desea que se dé de alta el nuevo usuario y se actualice la tabla de usuarios.

Para ello se captura el evento de envío del formulario (*submit*), se comprueba las cajas de texto y se envían los valores del formulario codificados en una cadena mediante la función ***$().serialize()*** al script "*usuarios.php*" mediante POST con la clave '*datos*':

```
<script>
        $().ready( function() {
                $.getJSON('usuarios.php', refrescar);
                $( "form" ).on( "submit", function( event ) {
                        event.preventDefault();
                        // Validación de campos. No pueden estar vacios
                        if (!$("input[name='usuario']").val())
                                { alert("Usuario vacio"); return; }
                        if (!$("input[name='clave']").val())
                                { alert("Contraseña vacia"); return; }
                         // Serializacion de campos de texto del formulario
                         var datos = $( this ).serialize();
                         // Envio de datos serializados mediante POST y obtención actuales.
                         $.post('usuarios.php', {'datos': datos}, refrescar, 'json');
                        // Limpieza de campos
                        $("input[name='usuario']").val('');
                        $("input[name='clave']").val('');
                });
        });
</script>
```

La función ***refrescar()*** se invoca pasando la respuesta enviada por el script para mostrar la lista actualizada de usuarios o un mensaje de error.

El script "*usuarios.php*" realiza dos funciones:

1.- Recibe la cadena codificada con la función ***$().serialize()*** de JQuery enviada desde la página, y obtiene el nombre de usuario y contraseña indicados. Dicha función codifica los valores del formulario devolviendo una cadena con la forma:

"usuario=roger&clave=1234"

La función de PHP *decodificarURL()*devuelve una matriz asociativa con las claves '*usuario*' y '*clave*' de los valores indicados por el usuario.

```
        // Decodifica la URL en una matriz asociativa de parámetros
        function decodificarURL( $cadena_datos ) {
                $datos = array();
                foreach (explode('&', $cadena_datos ) as $dato) {
                        $param = explode("=", $dato);
                        if ($param) {
                                $clave = urldecode($param[0]);
                                $valor = urldecode($param[1]);
                                $datos[$clave] = $valor;
                        }
                }
                return $datos;
        }

        // POST con datos --> Alta de usuario nuevo.
        $cadena_datos = $_POST['datos'];
        $datos = decodificarURL($cadena_datos);
```

2.- Inserta en la base de datos un nuevo usuario con el nombre y contraseña dados. Si la inserción es correcta se llama a la función ***obtenerUsuario()*** que devuelve una matriz de objetos con todos los usuarios registrados codificados en JSON. En caso de error envía un objeto con un atributo '*error*' y el mensaje de error correspondiente en JSON.

```
        // Devuelve matriz de objetos usuario con todos los usuarios.
        function obtenerUsuarios() {
                $resultados = DB::run('SELECT USUARIO, CLAVE FROM USUARIOS');
                return $resultados->fetchAll(PDO::FETCH_OBJ);
        }


        // POST con datos --> Alta de usuario nuevo.
        $cadena_datos = $_POST['datos'];
        $datos = decodificarURL($cadena_datos);
        try {
                // Alta de usuario nuevo
                $resultados = DB::run('INSERT INTO USUAROS(USUARIO, CLAVE, ADMINISTRADOR)
VALUES (:usuario, :clave, 0)',[':usuario' => $datos['usuario'],
                                ':clave' => $datos['clave']
                               ]);
                // Obtencion de lista actualizada de usuarios registrados
                $usuarios = obtenerUsuarios();
                echo json_encode($usuarios);
        } catch( Exception $ex) {
                // Envio de código de error de alta
                echo json_encode((object)['error' => $ex->getMessage()]);

        }
```

Código completo: "*usuarios.php*"

```php
<?php
	require_once 'classes/db.class.php';
	header('Content-Type: application/json; charset=utf-8');
	 // Decodifica la URL en una matriz asociativa de parámetros
	 function JQUnserialize( $cadena_datos ) {
		$datos = array();
		foreach (explode('&', $cadena_datos ) as $dato) {
			$param = explode("=", $dato);
			if ($param) {
				$clave = urldecode($param[0]);
				$valor = urldecode($param[1]);
				$datos[$clave] = $valor;
			}
		}
		return $datos;
	 }
	 // Devuelve matriz de objetos usuario con todos los usuarios.
	 function obtenerUsuarios() {
		$resultados = DB::run('SELECT USUARIO, CLAVE FROM USUARIOS');
		return $resultados->fetchAll(PDO::FETCH_OBJ);
	 }
	// POST con datos --> Alta de usuario nuevo.
	$cadena_datos = $_POST['datos'];
	$datos = JQUnserialize($cadena_datos);
	try {
		// Alta de usuario nuevo
		$resultados = DB::run('INSERT INTO USUAROS(USUARIO, CLAVE, ADMINISTRADOR)
 VALUES (:usuario, :clave, 0)',[':usuario' => $datos['usuario'],
				':clave' => $datos['clave']
			]);
		// Obtencion de lista actualizada de usuarios registrados
		$usuarios = obtenerUsuarios();
		echo json_encode($usuarios);
	} catch( Exception $ex) {
		// Envio de código de error de alta
		echo json_encode((object)['error' => $ex->getMessage()]);

	}
?>
```

Código completo: "*usuarios.html*"

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<style>
	td {background-color: #ffff00;text-align: center}
	th {width: 100px;	background-color: #eeeeee}
</style>
<title>Insert title here</title>
<script src="scripts/jquery-3.0.0.min.js"></script>
 <script>
	$().ready( function() {
		$.getJSON('listusuarios.php', refrescar);
		$( "form" ).on( "submit", function( event ) {
			event.preventDefault();
			// Validación de campos. No pueden estar vacios
			if (!$("input[name='usuario']").val())
				{ alert("Usuario vacio"); return; }
			if (!$("input[name='clave']").val())
				{ alert("Contraseña vacia"); return; }
			// Serializacion de datos
			var datos = $( this ).serialize();
			// Envio de datos serializados
			$.post('usuarios.php', {'datos': datos}, refrescar, 'json');
			// Limpieza de campos
			$("input[name='usuario']").val('');
			$("input[name='clave']").val('');
		});
	});
```

```
        function refrescar( usuarios ){
                // Se ha devuelto una matriz?
                if ( !$.isArray(usuarios)) {
                        alert(usuarios.error);
                        return;
                }
                // Eliminar filas existentes
                $('#tbl_usuarios tbody tr').not(':first').remove();
                // Generacion de filas por cada usuario
                $(usuarios).each(function( index, usuario ) {
                        $("#tbl_usuarios").find('tbody')
                    .append($('<tr>')
                        .append($('<td>')
                                .html(usuario.USUARIO))
                         .append($('<td>')
                                .html(usuario.CLAVE))
                    );
                });
        }
 </script>
</head>
<body>
        <div id='capa'>
        <table id='tbl_usuarios'>
                <tr><th>USUARIO</th><th>CLAVE</th></tr>
        </table>
        </div>
        <form>
                Usuario: <input type='text' name='usuario'><br/>
                Clave: <input type='password' name='clave'><br/>
                <input type='submit' value='GUARDAR'>
        </form>
</body>
</html>
```

Ejercicio Propuesto

Se pide crear una aplicación web de gestión bancaria empleando AJAX y la base de datos ***Banco*** vista en ejemplos anteriores.

Página inicial (*home.html*)

La página inicial (*home.html*) debe permitir la búsqueda de cuentas bancaria introduciendo el nombre, apellido, o parte de alguno de los dos del nombre del titular.

Los datos de las cuentas se almacenan en la tabla ***CUENTAS*** de la base de datos ***Banco***:

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado
☐	1	NUMERO_CUENTA 🔑	char(8)			No	*Ninguna*
☐	2	TITULAR	varchar(50)			No	*Ninguna*
☐	3	SALDO	decimal(10,2)			Sí	0.00
☐	4	INTERES	decimal(3,2)			Sí	0.00

Los datos de las cuentas deben solicitarse al servidor mediante una petición AJAX y mostrarse en una tabla indicando por cada cuenta los siguientes campos: ***NCUENTA***, ***TITULAR***, ***SALDO***, ***INTERES***.

Titular: artyom Obtener

NCUENTE	TITULAR	SALDO	INTERES
AC059603	ARTYOM DEMIDOVICH	1000.00	0.04
AC059609	ARTYOM DEMIDOVICH	850.00	0.04

El campo NCUENTA debe ser un hipervínculo a la página de detalles pasando el identificador de la cuenta como parte de la URL. (*detalles.html?cuenta=AC059609*)

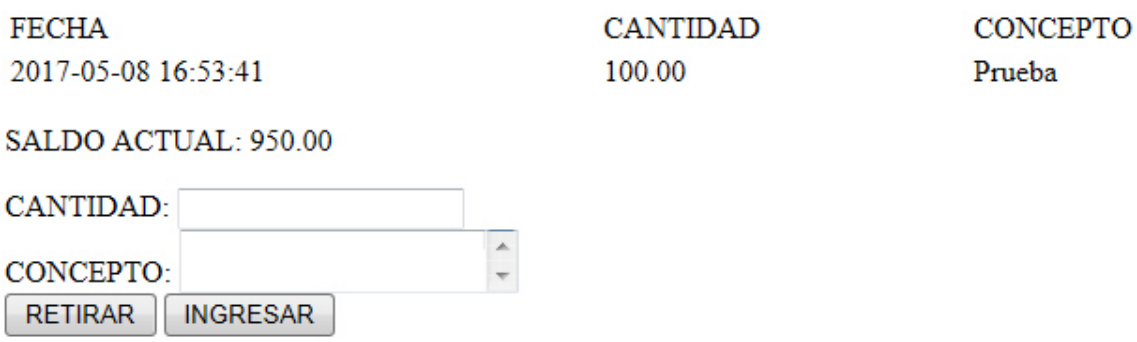
Página de detalles (*detalles.html*)

La página de detalles (*detalles.html*) muestra el saldo actual de la cuenta seleccionada en la página anterior junto con todos los movimientos registrados en la misma obtenidos mediante una petición AJAX.

Los datos de los movimientos se almacenan en la tabla ***MOVIMIENTOS*** de la base de datos ***Banco***:

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
☐	1	ID_MOVIMIENTO 🔑	int(11)			No	*Ninguna*	AUTO_INCREMENT
☐	2	CANTIDAD	decimal(10,2)			No	*Ninguna*	
☐	3	CONCEPTO	varchar(255)			No	*Ninguna*	
☐	4	CUENTA 🔗	char(8)			No	*Ninguna*	
☐	5	FECHA	datetime			No	*Ninguna*	

Los movimientos deben mostrarse en una tabla indicando la ***FECHA***, ***CANTIDAD*** y ***CONCEPTO*** de cada movimiento:



Debe mostrarse además un formulario para registrar nuevos movimientos indicando la cantidad y el concepto. Deben incluirse además dos botones para indicar si se trata de un ingreso o retirada de capital. Al pulsar uno de estos botones debe enviarse una petición AJAX de tipo *POST* al servidor para dar de alta el nuevo movimiento, y actualizar el saldo actual y los movimientos de la cuenta.

Observaciones

Se recomienda emplear peticiones AJAX con envío de datos mediante objetos/matrices codificados en JSON.