

Lectura y almacenamiento de archivos binarios en tablas de una base de datos.

Lectura desde la base de datos:

Para poder transferir por la WEB datos que están almacenados en forma binaria en un atributo de tipo long blob de una tabla y luego presentarlos del lado del cliente debemos:

- a) Leerlos de la base.
- b) Codificarlos en base64 (6 bits) para asegurarnos de que caracteres ASCII como por ejemplo >, <, !, etc.. no sean transferidos y se confundan con etiquetas HTML por ejemplo. Recordemos que naturalmente el protocolo HTTP transporta datos textuales codificados en binario y no binarios puros. Recordemos que toda codificación en base 64 aumenta la cantidad de datos a transferir en una relación 4/3.
- c) Codificar el resultado en JSON cuidando de que esta codificación no agregue caracteres utf-8 que salgan del rango de codificación base64.
- d) Enviar los datos al navegador remoto.
- e) Parsear el JSON que llega al navegador para convertirlo en una variable de Java script.
- f) Presentar el documento dentro de un iframe HTML.

Codificación base64 (6 bits en lugar de 8).

En lugar de una codificación ASCII de 7 bits o una codificación UNICODE UTF-8 de n grupos de 8 bits. Responde a un array de índice numérico de 64 posiciones (6 bits) donde no hay símbolos que puedan ser usados y confundidos con el etiquetamiento HTML por ejemplo. Solo contiene letras mayúsculas (A-Z) letras minúsculas (a-z), números (0-9) y los símbolos de control + y /

BASE64 INDEX TABLE

0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Lectura de binario del lado Cliente:

```
const data = new URLSearchParams();  
data.append('codArt',argArticulo);
```

Se construye un objeto para almacenar datos de envio en el req. http asincrónico

```
const options = { //opciones de configuracion del fetch  
  method:'POST',  
  headers: {},  
  body: data //es el body del requerimiento http  
}
```

Se construye un objeto para almacenar los atributos del req. http asincrónico

```
fetch('./traeDoc.php', options)
```

Se dispara el req. http asincrónico

```
.then(respuestaDelServer=>{  
  return respuestaDelServer.text();  
})
```

```
.then(datos=>{  
  alert(datos);  
  var objetoDato = JSON.parse(datos);  
  $("#ventanaModalRespuesta").css("visibility","visible");  
  $("#contenidoModalRespuesta").empty();  
  $("#contenidoModalRespuesta").html("<iframe width='100%' height='600px' src='data:application/pdf;base64,'" + objetoDato.documentoPdf + "'></iframe>");  
});
```

Cumplida la cadena de promesas se prepara la ventana donde se mostrara el binario arribado.

Se incrusta el iframe dentro del contenedor de respuestas con la data incluida. Observar que el src del iframe se corresponde con un flujo que proviene de la parte servidora de una aplicación que envía pdf y que está codificado en 64 bits.

Lectura de un binario del lado Servidor:

Aqui se levanta el binario de la tabla de la base

Nombre del atributo que contiene el archivo binario.

```
$sql="select documentoPdf from articulos where codArt = '$codArt'"
```

En PHP leemos la base usando esta sentencia SQL. Usaremos los metodos de preparacion, vinculacion y ejecucion para obtener el resultado de la consulta SQL.

Una vez obtenido el resultado de la lectura de la base. Creamos un objeto articulo con un atributo documentoPdf y le asignamos el resultado de la lectura pero codificado en base64 para transferir por la red evitando errores.

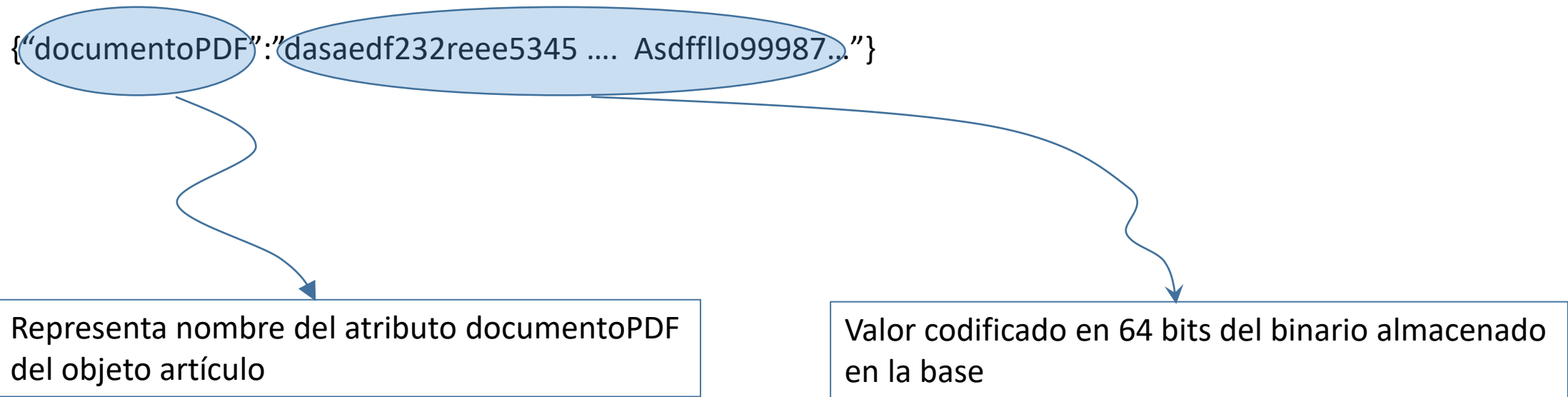
```
$objArticulo->documentoPdf=base64_encode($fila['documentoPdf']);
```

```
$salidaJson = json_encode($objArticulo,JSON_INVALID_UTF8_SUBSTITUTE);
```

Codificamos en JSON agregando el segundo argumento para cuidar de que este proceso de conversión no genere caracteres UTF-8 que estén fuera del rango base64.

¿Cómo sería el string JSON que representa un objeto de dato entregado en la respuesta http?

`{"documentoPDF": "dasaedf232reee5345 Asdffllo99987..."}`



Representa nombre del atributo documentoPDF del objeto artículo

Valor codificado en 64 bits del binario almacenado en la base

Lo podríamos ver en la presentación si ejecutamos un `alert()` dentro del `fetch()` que hace el requerimiento:
`alert("Respuesta del SERVER a mostrar: "+ respuestaDelServer);`

Actualización de la base - LADO CLIENTE

Se trata de hacer un upload de archivo a través de un form html

`<form id="formArticulosModi" method="post" enctype="multipart/form-data">`

El formulario de modi debe tener estos atributos

....

`<label>Documento Pdf: </label>`

`<input type="file" id="formArticulosEntDocumentoPdfModi" name="documentoPdf" />`

El type del input lleva valor "file"

El input debe llevar un name como cualquier otro input

....

`</form>`

Actualización de la base - LADO SERVIDOR:

Tanto en los procesos de Alta como en los de Modificación de una tabla se podría pretender enviar archivos binarios desde el cliente.

Para simplificar esto procederemos a dividir los scripts del lado del servidor en dos partes. Una primera en donde se traten los atributos simples de un registro o fila y otro donde se actualicen los atributos binarios a partir de los files recibidos desde los formularios multipart del requerimiento por parte del cliente.

En nuestro ejemplo estaremos recibiendo por parte del servidor todos los datos de formulario, tanto los simples como los que traen archivos adjuntos en el body del req HTTP.

En cuanto a los datos simples, podremos leerlos de esta manera:

```
$codArt = $_POST['codArt'];  
$familia = $_POST['familia'];  
$descripcion = $_POST['descripcion'];  
$um = $_POST['um'];  
$fechaAlta = $_POST['fechaAlta'];  
$saldoStock = $_POST['saldoStock'];
```

`$_FILES['documentoPdf']`)

Elemento del arreglo global `$_FILES` que almacena el documento arribado en el input que trae el requerimiento HTTP con el name "documentoPdf"

`$_FILES['documentoPdf']['name']`

EL type de `$_FILES['documentoPdf']` no es string.
O sea que no es una variable simple de una dimensión que contiene el nombre del archivo subido desde el input de java script con nombre documentoPdf sino un array (para verlo se puede usar `var_dump()`).
El elemento name en la 2da dimension de `$_FILES` contiene el nombre de archivo original.

`$contenidoPdf = file_get_contents($_FILES['documentoPdf']['tmp_name']);`

El elemento `tmp_name` en la 2da dimensión de `$_FILES` contiene el nombre del archivo temporario que contiene el binario

El contenido del file subido, queda almacenado en un archivo temporario (`tmp_name`) del lado del Servidor y se puede leer con la función de php `file_get_content()`


```
if (empty($_FILES['documentoPdf']['name'])) {  
    $respuesta_estado = $respuesta_estado . "<br />No ha sido seleccionado ningun file para enviar!";  
}  
else {  
  
    Update de la tabla  
  
}  

```

La variable \$contenidoPdf ya contiene como valor el binario del pdf.

Finalmente, a partir de esta variable que lleva el contenido del documento pdf, se abre la conexión con el motor de base de datos y se ejecutan las sentencias de preparación, enlace y ejecución de la consulta sql.

```
$sql="update articulos set documentoPdf=$contenidoPdf where codArt=:codArt;"
```

Ahora si se puede acceder a la base

De datos y hacer el update correspondiente de la tabla.

Recordar que los signos :xxx representan los valores que serán usadas en la clausula WHERE de la sentencia SQL, luego vendrán la preparación, vinculación y ejecución de la sentencia, de la misma manera que en cualquier consulta de update.