BLOQUE VI. ALMACENAMIENTO DE INFORMACIÓN. AJAX con JSON

JSON (JavaScript Object Notation) es un formato basado en texto para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o viceversa)

1. OBJETOS JSON BIEN FORMADOS

Los objetos JSON son cadenas de texto, aunque debe ser convertido a un objeto nativo de JavaScript para acceder a sus datos. Es posible incluir los mismos tipos de datos básicos dentro de un JSON que en un objeto estándar de JavaScript (cadena, números, arrays, booleanos y otros literales de objeto). Esto permite construir una **jerarquía de datos**.

Un objeto JSON puede ser almacenado en su propio archivo, que es básicamente un archivo de texto con una extension .json, y una MIME type de application/json.

Nota: Convertir una cadena a un objeto nativo se denomina *parsing*, mientras que convertir un objeto nativo a una cadena para que pueda ser transferido se denomina *stringification*.

Debemos considerar algunas cosas para tener un objeto JSON bien formado:

- Se requiere usar comillas dobles para las cadenas y los nombres de propiedades. Las comillas simples no son válidas.
- Una coma o dos puntos mal ubicados pueden producir que un archivo JSON no funcione. Es posible validar JSON utilizando una aplicación como JSONLint.
- A diferencia del código JavaScript en que las propiedades del objeto pueden no estar entre comillas, en JSON, sólo las cadenas entre comillas pueden ser utilizadas como propiedades.

Podemos validar un documento JSON utilizando un validador como: **https://jsonlint.com/**

1.1. JavaScript Object Notation (JSON)

JSON es un formato basado en texto para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o viceversa)

Los objetos JSON son cadenas de texto, aunque debe ser convertido a un objeto nativo de JavaScript para acceder a sus datos. JavaScript posee un objeto global JSON que tiene los métodos disponibles para la conversión.

1.2 JSON y XML

Lo primero que se debe decir es que ambos formatos se pueden utilizar para enviar y recibir información desde y hacia un servidor web (bases de datos, entre sistemas heterogéneos, etc). Por lo tanto, podemos decir que son equivalentes en cuanto a esta utilidad.

Ambos son autodesriptivos, jerárquicos y pueden ser utilizados por multitud de lenguajes de programación. Pero también hay una serie de diferencias entre ambos: JSON no utiliza etiquetas finalizadoras, produce documentos más cortos, son más rápidos de leer y escribir, y se pueden utilizar arrays.

Resumiendo, podemos decir que las principales diferencia son que XML tiene que ser analizado con un analizador XML mientras que JSON se puede analizar mediante una función estándar de JavaScript. Eso lleva a que XML es mucho más difícil de analizar que JSON.

Veamos un ejemplo en ambos formatos (equivalentes):

Ejemplo JSON

```
{"employees":[
   { "firstName":"John", "lastName":"Doe" },
   { "firstName":"Anna", "lastName":"Smith" },
   { "firstName":"Peter", "lastName":"Jones" }
]}
```

Ejemplo XML

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

1.3. Formato JSON

JSON es un formato ligero de intercambio de datos, independiente de los lenguajes de programación y con la particularidad de ser reconocido de forma nativa por JavaScript. Se utiliza de manera frecuente en las aplicaciones de intercambio de datos cliente-servidor como alternativa a XML. Con la ventaja de que la codificación de la información en el servidor es sencilla de realizar y en el lado del cliente no es necesario recurrir a un parser DOM.

Vamos a ver un ejemplo:

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Ejemplo JSON_1</title>
</head>

<body>
    <h1>Ejemplo 1 formato JSON</h1>

    <script type="text/javascript">

        /* Definición de objets JSON */
        var alumno1 = {
            codigo: "V001",
            nombre: "Antonio Flores",
            nota_media: 9
        };
        var alumno2 = {
            codigo: "V002",
            nombre: "Laura Torralba",
            nota_media: 7
        };
        var alumno3 = {
            codigo: "V003",
            nombre: "Joaquín Prieto",
            nota_media: 3
        };

        /* Mostrar las propiedades de los objetos */
        document.write(alumno1.nombre + " -> " + alumno1.nota_media + "<br/>");
        document.write(alumno2.nombre + " -> " + alumno2.nota_media + "<br/>");
        document.write(alumno3.nombre + " -> " + alumno3.nota_media + "<br/>");

        /* Modification de la propiedad nota_media del objeto alumno1 */
        alumno1.nota_media = 6;

        /* Mostrar las propiedades del objeto alumno1 */
        document.write("<hr />");
        document.write(alumno1.nombre + " -> " + alumno1.nota_media+ "<br/>");
```

Podemos ver cómo se han creado los objetos JSON para los alumnos de la manera

```
var alumno1 = {
      codigo: "V001",
      nombre: "Antonio Flores",
      nota_media: 9
  };
```

Al fin y al cabo, creamos objetos. De la misma manera que para acceder a su información, se trata de forma similar a la del resto de objetos:

```
document.write(alumno1.nombre + " -> " + alumno1.nota_media + "<br/>");

alumno1.nota_media = 6;
```

También, sería posible acceder a la información de forma asociativa:

```
document.write(alumno1["nombre"] + " -> " + alumno1["nota_media"] +"<br/>");
```

Es posible la creación de objetos compuestos. Añade el siguiente código después de la definición de los 3 objetos de los alumnos:

```
/* Definición de un objeto composite */
  var alumnosGrupo = {
      grupoAlu1: alumno1,
      grupoAlu2: alumno2
  };
```

Y para mostrar la información, añade antes de </script>:

```
/* Mostrar la propiedad nota_media del objeto
 alumno2 incluido en el objeto alumnosGrupo */
document.write("<hr />");
document.write("Mostrar el objeto alumnosGrupo:" + "<br/>");

 document.write(alumnosGrupo.grupoAlu2.nombre + " -> " +
alumnosGrupo.grupoAlu2.nota_media + "<br/>");
 document.write(alumnosGrupo["grupoAlu2"]["nombre"] + " -> " +
alumnosGrupo["grupoAlu2"]["nota_media"] + "<br/>");
```

Observa las dos formas equivalentes de acceder a la información que se han utilizado.

Para terminar el ejemplo, vamos a construir objetos JSON a partir de tablas. Añade el siguiente código al final del script:

```
        /* Definición de 2 tablas (nombres y notas) */
        var nombres = ["María Beltrán", "Jose Manuel Marín"];
        var notas = [9, 5];

        /* Definición de un objeto JSON basado en las tablas */
        var alumnos = {
            nom: nombres,
            punt: notas
        };

        /* Mostrar las propiedades del segundo alumno */
        document.write("<hr />");
        document.write("Visualización (utilizando las tablas) de las
propiedades del segundo alumno:");
        document.write("<br />");
        document.write(alumnos.nom[1] + " -> " + alumnos.punt[1] );
```

Podemos observar cómo se ha credo dos tablas con los datos y después se han agrupado en un objeto JSON con las propiedades nom y punt, que son propiedades compuestas basadas en las tablas.

La notación es muy intuitiva, basta con indicar el nombre del objeto JSON como un prefijo y después con un punto indicar el nombre de la tabla y el rango del valor entre [].

El resultado final quedaría así:

Ejemplo 1 formato JSON

Antonio Flores -> 9
Laura Torralba -> 7
Joaquín Prieto -> 3

Antonio Flores -> 6

Mostrar el objeto alumnosGrupo:
Laura Torralba -> 7
Laura Torralba -> 7

Visualización (utilizando las tablas) de las propiedades del segundo alumno:
Jose Manuel Marín -> 5

2. LECTURA DE UN ARCHIVO JSON

Vamos a partir del ejemplo del tema anterior (alumnos) y en un archivo de texto llamado alumnos.json ponemos los datos:

```
{
    "alumno1": {
        "codigo": "V001",
        "nombre": "Antonio Flores",
        "nota_media": 9
    },
    "alumno2": {
        "codigo": "V002",
        "nombre": "Laura Torralba",
        "nota_media": 7
    },
    "alumno3": {
        "codigo": "V003",
        "nombre": "Joaquín Prieto",
        "nota_media": 3
    }
}
```

En el apartado body, simplemente creamos un contenedor para los datos y llamamos a una función ajaxJSON(), la cual leerá los datos y los pondrá en el div divResultado.

```
<body>
    <h1>Ejemplo 2 formato JSON</h1>

    <div id="divResultado"></div>   <!-- Capa para mostrar el
resultado -->

    <script type="text/javascript">
        ajaxJSON();     // Llamada de la función ajaxJSON para
cargar los datos
    </script>
</body>
```

El script ajaxJSON() lo definimos en el encabezado del documento (o en un archivo externo, por supuesto):

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
    <title>Ejemplo JSON_1</title>

    <script type="text/javascript">

        /* Función ajaxJSON */
        function ajaxJSON() {
```

Primero, recogemos el enlace de la capa de visualización en una variable:

```
var resultado = document.getElementById("divResultado");
```

Abrimos instanciamos un objeto tipo ActiveX o JavaScript que permite obtener datos en formato JSON.

```
if (window.XMLHttpRequest) {
      //Código para IE7+, Firefox, Chrome, Opera, Safari
      httpRequest = new XMLHttpRequest();

} else {
      // Código para IE6, IE5
      httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
}
```

XMLHttpRequest es un objeto ActiveX o JavaScript que permite obtener datos en formato XML, JSON, HTML, de texto plano o incluso de tipo file o ftp.

Fue diseñado por Microsoft y adoptado por Mozilla, Apple y Google. Actualmente es un estándar de la W3C.

Proporciona una forma fácil de obtener información de una URL sin tener que recargar la página completa. Una página web puede actualizar sólo una parte de la página sin interrumpir lo que el usuario está haciendo. XMLHttpRequest es ampliamente usado en la programación AJAX.

En los navegadores antiguos, es necesario recurrir a un objeto ActiveX.

ActiveX es un entorno para definir componentes de software reusables de forma independiente del lenguaje de programación. Las aplicaciones de software pueden ser diseñadas por uno o más de esos componentes.

Fue presentado en 1996 por Microsoft como una evolución de sus tecnologías Component Object Model (COM) y Object Linking and Embedding (OLE) y se usa generalmente en su sistema operativo Windows, aunque la tecnología como tal no está atada al mismo.

Muchas aplicaciones Microsoft Windows —como puedan ser Internet Explorer, Microsoft Office, Microsoft Visual Studio, etc — usan controles ActiveX para proveer sus juegos de funcionalidades y también encapsular su propia funcionalidad como controles ActiveX que así pueden ser empotrados en otras aplicaciones. Internet Explorer también permite empotrar sus propios controles ActiveX en páginas web.

El actual navegador de Microsoft, Microsoft Edge, no soporta esta tecnología, por lo que se recomienda evitar su uso.

AJAX (Asynchronous JavaScript And 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, mejorando 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, aunque existe la posibilidad de configurar las peticiones como síncronas de tal forma que la interactividad de la página se detiene hasta la espera de la respuesta por parte del servidor.

JavaScript es un lenguaje de programación (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 en XML.

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).

Con la instancia del objeto httpRequest, abrimos el archivo y definimos el tipo de flujo. Hay que tener en cuenta que el nombre del archivo también se puede sustituir por una url donde se encuentre el documento.

```
httpRequest.open("GET", "alumnos.json", true);  // true: modo asíncrono

/* Definición del tipo de flujo */
httpRequest.setRequestHeader("Content-type", "application/json");
```

XMLHttpRequest.open inicializa o reinicializa una solicitud. Al menos se deben poner los dos primeros parámetros:

- **method**. Puede ser:
    - **GET.** Se usa para solicitar datos de un recurso específico, la cadena de consulta (pares de nombre / valor) se envía en la URL de una solicitud (/form.php?name1=value1&name2=value2). Además, se debe tener en cuenta que las solicitudes pueden almacenarse en caché, permanecen en el historial del navegador, tienen restricciones de longitud.
    - **POST**. se usa para enviar datos a un servidor para crear o actualizar un recurso. Los datos enviados al servidor se almacenan en el cuerpo de solicitud de la solicitud HTTP. Las solicitudes no tienen restricciones en la longitud de los datos
    - **PUT.** Se usa para enviar datos a un servidor para crear o actualizar un recurso. La diferencia entre POST y PUT es que las solicitudes PUT son idempotentes. Es decir, llamar a la misma solicitud PUT varias veces siempre producirá el mismo resultado. Por el contrario, llamar a una solicitud POST repetidamente tiene efectos secundarios de crear el mismo recurso varias veces.
    - **DELETE**. Borra un recurso
- **URL**.
- **async**. Admite los valores **true** o **false**. El valor true sirve para una conexión asíncrona y false para una síncrona.
  Preferiblemente es preferible una comunicación **asíncrona** porque mientras se reciben los datos, el navegador seguirá cargando el resto de componentes, mientras que con una comunicación síncrona, se paraliza el navegador hasta que ha terminado de recibir los datos.
- **user**. Usuario para un proceso de autenticación.
- **password**.

En este momento hay que procesar el flujo en el momento que esté disponible. Cuando esté completado el flujo, se procesará convitiendolo en objetos JavaScript (esta pesada operación se puede realizar con un método nativo de JavaScript, JSON.parse).

Cada uno de los objetos se pasan a un array para que su posterior procesamiento sea más cómodo. Cada dato recibido es un array que contiene los campos de cada objeto JSON. y por último se colocan los datos recibidos en el contenedor.

```
httpRequest.onreadystatechange = function () {

  /* Prueba si la consulta ha terminado y están en estado OK */
  if (httpRequest.readyState == 4 && httpRequest.status == 200) {

      /* Conversión del flujo JSON en objetos Javascript */
      var datosJSON = JSON.parse(httpRequest.responseText);

      resultado.innerHTML = "";     /* Vaciar el contenedor */

      /* Recorrido de los objetos Javascript */
      for (var objet in datosJSON) {
            resultado.innerHTML += datosJSON[objet].nombre + " -> " +
            datosJSON[objet].nota_media + " <hr />";
      }
  }
  else resultado.innerHTML ="<br/><h2>Error: No se ha podido acceder a
la información</h2>";

}
```

Se informa al servidor que no le enviamos ninguna información.

```
/* Se envía un valor nulo al servidor como respuesta */
httpRequest.send(null);
```

```
/* Mensaje mostrado mientras se espera el procesamiento del archivo
alumnos.json */
results.innerHTML = "Espera de procesamiento JSON ...";
```

Este mensaje normalmente no se verá porque al ser la lectura de un archivo local, la respuesta suele ser inmediata y sin tiempo de espera.

3. LECTURA DE UN ARCHIVO JSON CON UN SCRIPT SERVIDOR PHP

La principal diferencia entre este caso y el del punto anterior es que el flujo JSON se lee por una aplicación del lado del servidor escrita en PHP.

El archivo de datos que contiene las descripciones de los alumnos se mantiene sin ningún cambio (excepto que se guarda junto con el archivo php en el servidor)

El script del lado del cliente es casi idéntico, solamente cambiamos el origen de los datos (habría que poner el DNS completo en un caso real):

```
httpRequest.open("GET", "servidorJSON.php", true);
```

El archivo php (servidorJSON.php) debe contener la definición del tipo de flujo (header), la lectura del archivo JSON y, por último, se asigna sin ninguna transformación a una variable de texto que se envía a la aplicación cliente.

```
<?php

    // Tipo de flujo
    header("Content-Type: application/json");

    // Lectura del archivo JSON
    $alumnosJSON = file_get_contents("alumnos.json");

    // Envío de flujo JSON a la aplicación cliente
    echo $alumnosJSON;

?>
```

4. LECTURA DE UNA TABLA MYSQL CON SERVIDOR PHP Y FLUJO JSON

El objetivo principal de cualquier aplicación real es obtener datos actualizados desde una base de datos, dicha información debe ser accesible desde cualquier lugar que se soliciten los datos. Pero sin olvidarnos nunca de la seguridad para mantener a salvo la información y que solo sea accesible de la forma que se establezca y los datos que nos interese presentar a la aplicación cliente.

Antes de comenzar, creamos una nueva base de datos llamada ‘json’ y una tabla ‘alumnos’ con los campos: ‘codigo’ (Primary Key), ‘nombre’, ‘nota_media’ e introducimos los datos para poder probar la aplicación.

El script del lado del cliente es el mismo que en el caso anterior, poniendo el nombre del archivo php al que pediremos la información:

```
httpRequest.open("GET", "servidorJSON.php", true);
```

En el archivp servidorJSON.php contendrá el siguiente código:

```
<?php

    header("Content-Type: application/json"); // Tipo de flujo

    // Definición de la consulta SQL
    $consulta_sql = "select * from alumnos;";

    // Paramètres SGBD MySQL
    $servidor_mysql = "localhost:3306";
    $usuario_mysql = "usuario";
    $password_mysql = "1234";
    $bd_mysql = "json";

    if ( ( $connexion_mysql = mysqli_connect( $servidor_mysql,
$usuario_mysql, $password_mysql, $bd_mysql ) )   === FALSE )
    {
        echo "";
    }
    else
    {
        $resultado = mysqli_query( $connexion_mysql, $consulta_sql);
        if (mysqli_num_rows($resultado) <1)
        {
            echo "";
        }
        else
        {
            while ($registro=mysqli_fetch_array($resultado,MYSQLI_ASSOC))
            {
                    $filas[] = $registro;
            }

            // Codificación en formato JSON
            $datosJSON = json_encode($filas);

            // Envío del resultado al cliente
            echo $datosJSON;
        }
    }
    mysqli_close($connexion_mysql);
?>
```

Los registros leídos se pasan a un array para que estén todos juntos

Se codifican en formato JSON para ser enviados

El código anterior es muy similar al de los ejemplos del tema 5 para acceder a una base de datos y obtener la información. La diferencia está en que los registros leídos los pasamos a una tabla que contendrá al final todos los datos ($filas[])

Por último, se codifican los datos en formato JSON (json_encode($filas)) y se envían los datos al cliente.

ACLARACIONES DE LA PARTE FINAL DEL BLOQUE VII

Lo más importante que se debe tener claro en este tema es la forma en que se procesa la información que parte de una base de datos.

Vamos a seguir el código de **json4.html** y **servidorJSONb.php** que se puede descargar y reutilizar.

1. Si observamos el código de **servidorJSONb.PHP**, podemos darnos cuenta de que lo único que cambiará en cualquier proyecto es la consulta SQL y el acceso a la base de datos. El resto de código servirá para casi cualquier proyecto.

Esto es así porque lo único que hace el código es pasar los registros de la consulta a una tabla, transformarla a formato JSON y enviar ese resultado a la página html.

2. En el otro archivo **json4.html**, tampoco hay que cambiar muchas cosas, solamente la presentación de la página final y el procesamiento de los datos recibidos.

Por tanto, en la parte del <body>, hacemos la página como deseemos presentarla (como en cualquier proyecto)
En la función que recibe el código, lo único que debemos cambiar es cómo presentar los datos recibidos y dónde colocarlos.
Por ejemplo, en este archivo la información que se pone es el nombre y nota media de los alumnos.
El código será:

```
resultado.innerHTML = "";

/* Recorrido de los objetos Javascript */ for (var
objet in datosJSON) {
        resultado.innerHTML += datosJSON[objet].nombre + " -> " +
datosJSON[objet].nota_media + " <hr />";
}
```

Pero si queremos poner los datos dentro de una tabla, lo adaptaremos al siguiente código:

```
tabla = "<table border=1 cellpadding=10><tr><th>Nombre</th><th>Nota
Media</th></tr>";
/* Recorrido de los objetos Javascript */ for (var
objet in datosJSON) {
      tabla += "<tr>";
      tabla +="<td>"+datosJSON[objet].nombre+"</td><td>"+
      datosJSON[objet].nota_media + " </td>";
      tabla += "</tr>";
}
tabla += "</table>";
```

```
resultado.innerHTML=tabla;
```

La diferencia está básicamente en las etiquetas que añadimos para dibujar la tabla.

3. Así que, a la hora de abordar los ejercicios de este tema, hay que crear la base de datos (no es necesario crear todos los campos, con 4 o 5 de ejemplo es suficiente), adaptar la consulta del archivo php y adaptar la presentación en el archivo html.

Tara todo esto, podéis reutilizar los ejemplos que he dejado en el aula virtual.

5. EJERCICIOS

Ejercicio 1. Coches. Imaginemos un concesionario de coches que periódicamente debe enviar información sobre los vehículos que tiene en oferta a un portal publicitario de compra-venta de coches. Tenemos la siguiente información como ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<oferta>
	<vehiculo>
		<marca>ford</marca>
		<modelo color="gris">focus</modelo>
		<motor combustible="gasolina">duratorc 1.4</motor>
		<matricula>1234AAA</matricula>
		<kilometros>12500</kilometros>
		<precio_inicial>12000</precio_inicial>
		<precio_oferta>10000</precio_oferta>
		<extra valor="250">pintura metalizada</extra>
		<foto>11325.jpg</foto>
	</vehiculo>
	<vehiculo>
		<marca>ford</marca>
		<modelo color="gris">focus</modelo>
		<motor combustible="diesel">duratorc 2.0</motor>
		<matricula>1235AAA</matricula>
		<kilometros>125000</kilometros>
		<precio_inicial>10000</precio_inicial>
		<precio_oferta>9000</precio_oferta>
		<extra valor="250">pintura metalizada</extra>
		<foto>11328.jpg</foto>
	</vehiculo>
</oferta>
```

A partir de dicha información, crea una base de datos MYSQL llamada coches, en esa BD crea una tabla llamada vehículos e introduce los datos anteriores.

Realiza el código php y html para acceder a los datos de la BD y transferirlos en formato JSON al documento html para poder mostrarlos en forma de tabla.

Ejercicio 2. Cambia la sección de Consultar ALUMNOS del ejercicio 5 del BLOQUE VI Acceso MySQL con PHP, para que en vez de que el link de CONSULTAR llame a un PHP, directamente en la pantalla inicial de LINKS se ponga justo debajo la tabla de resultado del alumno buscado usando AJAX con un JSON. Debes mostrarlo tal cual lo tenías en la miniaplicación del bloque VI pero esta vez que los datos estén dentro de una LISTA HTML que debes crear usando DOM de javascript.