TEMA 23

XML / JSON

Introducción

Desde el punto de vista de su funcionalidad hemos visto HTML como lenguaje de marcas orientado a la presentación de la información.

Si observamos los lenguajes de marcas desde el punto de vista del almacenamiento y la transmisión de la información, podemos encontrar, principalmente 2 formatos estándares:

- XML (eXtented Markup Language)

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE matricula SYSTEM "matricula.dtd">
<matricula>
    <personal>
        <dni>99999999G</dni>
        <nombre>Antonio García</nombre>
        <titulacion>GS Desarrollo de aplicaciones</titulacion>
        <curso_academico>2023/2024</curso_academico>
    </personal>
    <pago>
        <tipo_matricula>Matrícula ordinaria</tipo_matricula>
    </pago>
</matricula>
```

- JSON (JavaScript Object Notation)

```json
{
  "matricula": {
    "personal": {
      "dni": "99999999G",
      "nombre": "Antonio García",
      "titulacion": "GS Desarrollo de aplicaciones",
      "curso_academico": "2023/2024"
    },
    "pago": {
      "tipo_matricula": "Matrícula ordinaria"
    }
  }
}
```

Comenzaremos trabajando con JSON:

JSON

Como hemos comentado, representa datos estructurados en la sintaxis de objetos de javascript. Es comúnmente utilizado para transmitir datos en aplicaciones web (Envío de datos servidor/cliente).

JSON actualmente es el formato de transferencia de datos más usado por su menor tamaño en referencia a otros tipos de formato.

El objetivo del uso de JSON será trabajar con datos almacenados en JSON y poder crear objetos propios JSON.

Aunque es muy parecido a la sintaxis de objeto literal de JavaScript, puede ser utilizado independientemente de JavaScript, y muchos entornos de programación poseen la capacidad de leer (convertir, parsear) y generar JSON.

Un archivo JSON contiene cadenas y es útil para cuando se quieren transmitir datos a través de una red. Esta información para poder tratarla debe convertirse a un objeto nativo de JavaScript, pero no es un problema ya que JavaScript posee un objeto global JSON que tiene todos los métodos disponibles para poder operar con dicha información.

Un objeto JSON es solamente un archivo con extensión .json y un mime type (application/json)➔ (Se utiliza para especificar de forma estandarizada la naturaleza y el formato de un documento, archivo o conjunto de datos).

Ejemplos de tipos mime: (tipo/subtipo)

text/plain

text/html

image/jpg

application/json

application/xml

En la siguiente página se lleva el registro oficial de tipos mime:

https://www.iana.org/assignments/media-types/media-types.xhtml

Los tipos de datos que puede contener un archivo json son los siguientes (tipos de datos estándar de javascript):

- Cadenas
- Números
- Booleanos
- Arrays

Ejemplo de una estructura json:

```
{
  "Equipo": "Equipo 1",
  "Ciudad": "Sevilla",
  "creación": 2016,
  "activo": true,
  "miembros": [
    {
      "nombre": "Antonio Sánchez",
      "edad": 29,
      "Dirección": "Oviedo, 12",
      "Idiomas": ["Inglés", "Francés", "Alemán", "Español"]
    },
    {
      "nombre": "Lorena Sáez",
      "edad": 28,
      "Dirección": "Lucas, 32",
      "Idiomas": ["Inglés", "Italiano", "Alemán", "Español"]
    }
  ]
}
```

Para poder validar los archivos json que cream y también poder leer más fácilmente los archivos json que recuperamos de un servicio, utilizamos la siguiente web:

http://json.parser.online.fr/

Vamos a copiar el texto del archivo json anterior y lo pegamos en la dirección anterior para poder analizarlo.

Reglas a tener en cuenta al crear un archivo json:

Se basa en:

- Una colección de duplas (Propiedad/valor) de claves-valor:
  - Siempre comienza por una llave de apertura y termina con una llave de cierre.
  - Cada clave viene separada del valor mediante ":"
  - Las duplas están separadas entre ellas mediante ","
- Una lista ordenada de diferentes valores (a modo de array):
  - Viene precedida por el corchete abierto y posteriormente se cierre con el corchete de cierre.
  - Los distintos valores se separan mediante una ",".

Referente a los valores pueden ser:

- Una cadena de texto entre comillas dobles o simples.

- Un número, incluido decimales.
- Un booleano (true || false).
- Null.
- Un objeto.
- Un array.
- También se puede validar un archivo JSON utilizando el siguiente enlace: JSONLint.

Vamos a practicar un poco algunos ejercicios de json:-

Ejercicios Json:

1. Vamos a crear un JSON para representar la información de una biblioteca, incluyendo, nombre, dirección, teléfono, ciudad, un campo booleano que indique si esta en venta (true/false) y la lista de libros de contiene, insertando para cada libro, la siguiente información: título, autores (una lista de los nombres de los autores), año de publicación, géneros (una lista de los nombres de los géneros) y una lista de comentarios. Cada comentario estará compuesto del nick del usuario y el propio comentario aportado. En este caso lo guardaremos como ***biblioteca.json.***

Y ahora vamos a resolver si siguiente caso práctico:

2. Nuestro cliente “Cinelandia”, quiere vender sus entradas de cine en el famoso portal www.cineentradas.com. Para ello nos pone en contacto con soporte, donde se nos indica que debemos facilitarles la información del cine nombre,

dirección, teléfono y ciudad, las salas, y las películas, donde salas y películas son listas. Para las salas: hay que indicar el nombre de la sala, si está disponible, y también incluyendo el aforo. Para cada película de la cartelera: título, director, genero, duración (en minutos), nombre de la sala donde se proyecta y sesiones. Sesiones es una lista, informando de las distintas horas. Esta información debemos facilitarla en un formato JSON. Crea el fichero ***cinelandia.json*** con la información requerida.

Crear un ejemplo con 3 salas, y 2 películas, cada una con 3 sesiones.

Ahora vamos a ver cómo se accede a la información de un archivo JSON:

Para ello vamos a crear un nuevo archivo en nuestro directorio de trabajo denominado Tema23.html:

```
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <!-- ETIQUETA PARA HACER RESPONSIVE LA PÁGINA, ADAPTACIÓN DE ESCALAS
-->
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Tema 23 html</title>
    <!-- ICONO DE VENTANA -->
    <link rel="icon" type="image/x-icon" href="imagenes/favicon.ico">
```

Podéis cerrar el archivo.

Lo primero que vamos a hacer para probar la lectura del json es usar javascript, para ello vamos a añadir el siguiente fragmento de código JS a nuestra cabecera.

Como ya sabemos los archivos json son objetos Javascript y como tales podemos usarlos en JS tal y como están diseñados:

En el Head de nuestro archivo abrimos la etiqueta de código javascript:

<script></script>

Y entre dichas etiquetas pegamos el siguiente código:

```
var archivojson = {
            "Nombre": "Samaniego",
            "Direccion": "C/ San Jacinto, S/N",
            "Telefono": 954182644,
            "ciudad": "Sevilla",
            "venta": true,
            "libros":
                [
                    {
                        "titulo": "El saber no ocupa lugar",
                        "autores":
                            [
                                { "nombre": "Daniel Unamuno" },
                                { "nombre": "Laura Gallego" }
                            ],
                        "generos":
                            [
                                { "nombre": "Romántico" },
                                { "nombre": "Actualidad" }
                            ],
```

```
                        "comentarios":
                            [
                                { "nick": "Alon23", "texto": "Libro
adecuado para la lectura" },
                                { "nick": "Ana1999", "texto": "Final no
esperado" }
                            ]
                    }
                ]
        };
```

Hemos creado una variable denominada “archivojson” a la cual le hemos asignado el json de la práctica que hemos realizado antes de biblioteca.json.

Ahora crearemos en nuestro body el siguiente código:

```
<body>
    <form>
        <label for="idArea">Datos leídos del json</label><br>
        <textarea id="idArea" name="idArea"
style="width:90%;height:400px;"></textarea>
        <br><br>
        <input type="button" onclick="Borrar()" value="Borrar">
        <input type="button" onclick="lecturaNombre()" value="Nombre">

    </form>
</body>
```

Como podéis observar hemos creado un campo de Área de texto que nos va a servir para poder mostrar los valores que iremos leyendo del json que hemos guardado en la variable anterior.

Creamos 2 botones, un botón borrar, que servirá para limpiar el área de texto y un segundo botón que lo hemos denominado "Nombre" y que llamará a un método JS que se denominará "lecturaNombre()".

Una vez creados los botones y sus llamadas a JS, tenemos que definir esos métodos a los que hemos llamado.

En el HEAD, justo debajo de la declaración de la variable "archivojson", añadimos el siguiente código:

```
 //Borra toda la información del campo textarea:
        function Borrar() {
            document.getElementById("idArea").value = "";
        }
        //Lectura del campo nombre del json:
        function lecturaNombre() {
            document.getElementById("idArea").value =
document.getElementById("idArea").value + archivojson.Nombre + "\n";
        }
```

Como podemos ver son los dos métodos:

1. Borrar: simplemente accede al campo del formulario que tienen nuestra área de texto por ID y vacía su contenido (Iguala el contenido a vacío).
2. lecturaNombre: accede al contenido del área de texto, pero esta vez le asigna el contenido que ya tuviera la

misma, para no borrar nada, más el contenido de la propiedad "Nombre" del archivo json. Al final añade un carácter escape "\n" que se utiliza para saltar de línea.

Ahora solamente vamos a ponerlo un poco más bonito, en el mismo archivo, vamos a añadir a la cabecera la etiqueta de estilos y escribimos:

```
<style>
        textarea {
            padding: 10px;
            background-color: rgb(25, 78, 78);
            color: white;
            border-radius: 10px;
            border: solid 5px rgb(232, 107, 18);
        }

        input[type="button"] {
            padding: 10px;
            background-color: rgb(86, 142, 162);
            color: white;
            border-radius: 10px;
            border: solid 1px rgb(64, 49, 177);
        }

        label {
            color: rgb(44, 123, 160);
            font-size: x-large;
            font-weight: 800;
        }
    </style>
```

Si ahora ejecutamos nuestro archivo HTML, podemos probar el efecto que tiene la página:

(*) Como práctica el alumno tendrá que añadir 3 botones más seguidos, que serán para leer y escribir en el text área, los siguientes campos:

- dirección
- Teléfono
- Venta

Debe quedar algo así:

Ahora vamos a ver como podemos acceder a los arrays y a niveles más profundos del JSON:

Vamos a añadir información del primero libro que tenemos, queremos sacar el título del primero libro:

Para ello añadimos a nuestro archivo:

El botón al final de nuestro body:

```
   <input type="button" onclick="lecturaNombreLibro1()" value="Nombre
libro 1">
```

Ahora el método al final de nuestro código JS:

```
      //Lectura del campo titulo del primer libro:
        function lecturaNombreLibro1() {
            document.getElementById("idArea").value =
document.getElementById("idArea").value + archivojson.libros[0].titulo +
"\n";
        }
```

Como véis, hemos accedido al array de libros, con el índice a 0 para obtener el primer elemento, al igual que en JAVA. Y a continuación volvemos a poner “.” Y el nombre de la propiedad una vez que estamos en el elemento en el que nos hemos posicionado.

(*) Como ejemplo, el alumno debe poner un botón para obtener el título del libro segundo del array. Como no tenemos un segundo libro, lo añadimos rápidamente a nuestro json:

```
"libros":
    [
        {
            "titulo": "El saber no ocupa lugar",
            "autores":
                [
                    { "nombre": "Daniel Unamuno" },
                    { "nombre": "Laura Gallego" }
                ],
            "generos":
                [
                    { "nombre": "Romántico" },
                    { "nombre": "Actualidad" }
                ],
            "comentarios":
                [
                    { "nick": "Alon23", "texto": "Libro adecuado para la lectura" },
                    { "nick": "Ana1999", "texto": "Final no esperado" }
                ]
        },
        {"titulo":"título del segundo libro"}
    ]
;
```

Solamente el título que es lo que nos va a hacer falta ahora.

Ahora vamos a recorrer el primero nombre de autores y el segundo de autores:

En el apartado de html:

```
 <input type="button" onclick="lecturaNombreAutores1Libro1()"
value="Autor 1 de Libro 1">
        <input type="button" onclick="lecturaNombreAutores2Libro1()"
value="Autor 2 de Libro 2">
```

En el apartado JS:

```
 //Lectura del campo nombre del autor 1 del libro 1:
        function lecturaNombreAutores1Libro1() {
            document.getElementById("idArea").value =
document.getElementById("idArea").value +
archivojson.libros[0].autores[0].nombre + "\n";
        }
```

```
        //Lectura del campo nombre del autor 2 del libro 1:
        function lecturaNombreAutores2Libro1() {
            document.getElementById("idArea").value =
document.getElementById("idArea").value +
archivojson.libros[0].autores[1].nombre + "\n";
        }
```

El resultado de nuestra pantalla será:

(*) Para finalizar las prácticas con archivos json el alumno debe crear 4 botones más para mostrar: los dos géneros del primer libro y los dos comentarios del primer libro (En este caso podemos concatener el Nick con el texto).

Debe quedar algo así para los géneros:

Y ahora si añadimos los nicks y comentarios, debería quedar así:

XML

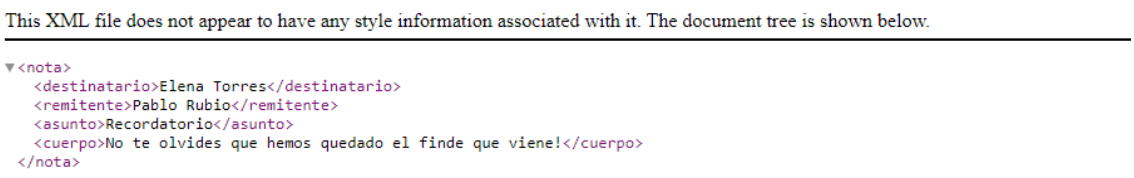
XML es un lenguaje de marcas que se inventó antes que JSON, pero al igual que JSON, sirven para almacenar y transportar información en la red, entre sistemas.

Veamos el siguiente ejemplo de un xml sencillo:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<nota>
  <destinatario>Elena Torres</destinatario>
  <remitente>Pablo Rubio</remitente>
  <asunto>Recordatorio</asunto>
  <cuerpo>No te olvides que hemos quedado el finde que viene!</cuerpo>
</nota>
```

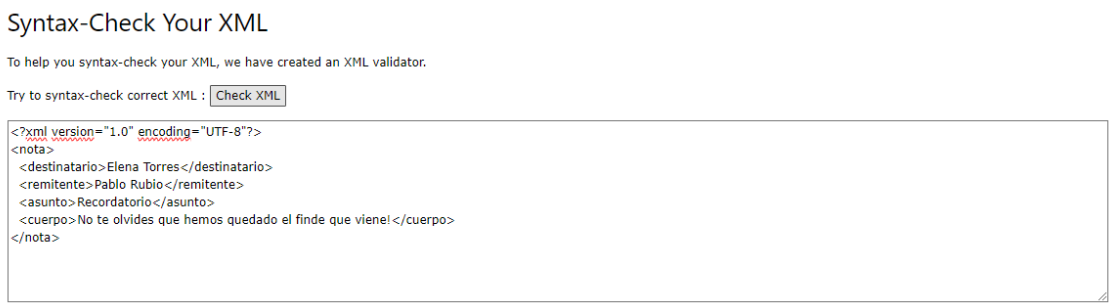
Vamos a crear un archivo denominado correo.xml en nuestro área de trabajo y que contenga la información anterior.

Si ejecutamos el mismo podemos ver que el navegador lo sabe interpretar sin problema:



Podemos validar el código del mismo en la siguiente página:

https://www.w3schools.com/xml/xml_validator.asp



Copiaremos el código dentro del área de texto y pulsaremos en el botón de chequeo, nos saldrá un mensaje indicando si existiera algún error.

Puedes probar a añadir algún carácter al final del texto y verás como sale el error al chequear el mismo.

El XML está diseñado para separar datos de presentación, como hemos visto lo único que tenemos es una estructura de datos. EN JSON también vimos que no había presentación alguna, en ese caso tendríamos que haberlo presentado en HTML. Ya veremos más adelante como se presenta la información de un XML en HTML de una forma más visible y atractiva.

Imaginemos que queremos un archivo xml con el tiempo atmosférico que hace en Sevilla, tendríamos el siguiente archivo denominado: "tiempo.xml":

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tiempo_atomosferico>

    <informacion>Agencia Estatal de Meteorología - AEMET</informacion>
    <credit_URL>https://www.aemet.es/</credit_URL>

    <image>
        <url>https://www.aemet.es/es/imagen-logo1</url>
        <titulo>AEMET</titulo>
        <link>https://www.aemet.es/</link>
    </image>

    <localizacion>Sevilla</localizacion>
    <estacion>Carmona</estacion>
    <latitud>36.43</latitud>
    <longitud>-7.38</longitud>
    <fecha_observacion>20/04/2024</fecha_observacion>

    <tiempo>Nublado</tiempo>
    <temperatura_max>27</temperatura_max>
    <temperatura_min>13</temperatura_min>
    <temperatura_med>21</temperatura_med>

</tiempo_atomosferico>
```

Pruébalo y comprueba que no da error al chequear el mismo.

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
▼<tiempo_atomosferico>
   <informacion>Agencia Estatal de Meteorología - AEMET</informacion>
   <credit_URL>https://www.aemet.es/</credit_URL>
  ▼<image>
     <url>https://www.aemet.es/es/imagen-logo1</url>
     <titulo>AEMET</titulo>
     <link>https://www.aemet.es/</link>
   </image>
   <localizacion>Sevilla</localizacion>
   <estacion>Carmona</estacion>
   <latitud>36.43</latitud>
   <longitud>-7.38</longitud>
   <fecha_observacion>20/04/2024</fecha_observacion>
   <tiempo>Nublado</tiempo>
   <temperatura_max>27</temperatura_max>
   <temperatura_min>13</temperatura_min>
   <temperatura_med>21</temperatura_med>
 </tiempo_atomosferico>
```


Podéis observar que tiene ya 2 niveles de anidamiento, el principal y uno más.

Podemos comprobar que los archivos XML siempre tienen estructura de árbol, en la que tienen una rama principal (tiempo_atmosferico: del archivo anterior) y luego pueden tener ramas hijas y ramas subhijas, en este caso hemos puesto una rama hija denominada “image” que tiene dentro 3 elementos hijos (url, titulo, link).

La primera línea de nuestro archivo XML define la versión y la codificación de los caracteres al igual que en los archivos HTML utilizamos “UTF-8”.

Los archivos XML siempre deben tener como mínimo un rama principal que contenga a todas las demás ramas hijas o atributos hijos.

Al igual que en JSON, los XML TAGAS o etiquetas, son sensitivos y siempre deben cerrarse:

<mensaje></mensaje>

<Mensaje></mensaje> --->Daría error, pues no es la misma etiqueta.

Hay algunos caracteres que no se pueden usar como valor en XML, ya que generarían un error al procesar el archivo, estos caracteres especiales son por ejemplo "<", si pusiéramos:

<mensaje>salario < 1000</mensaje> ->Daría error al procesar el fichero.

Estos caracteres especiales hay que reemplazarlos por sus códigos referenciados:

<	<	Menor que
>	>	Mayor que
&	&	Ampersand
'	‘	Apóstrofo
"	“	Comillas dobles

Para poner correctamente el contenido que vimos antes, tendríamos que haber puesto:

<mensaje>salario < 1000</mensaje>

Prueba esto en alguno de los archivos xml que hemos visto anteriormente y chequéalos.

Los comentarios en XML se escriben igual que en html:

<!—Comentario en XML igual que en html →

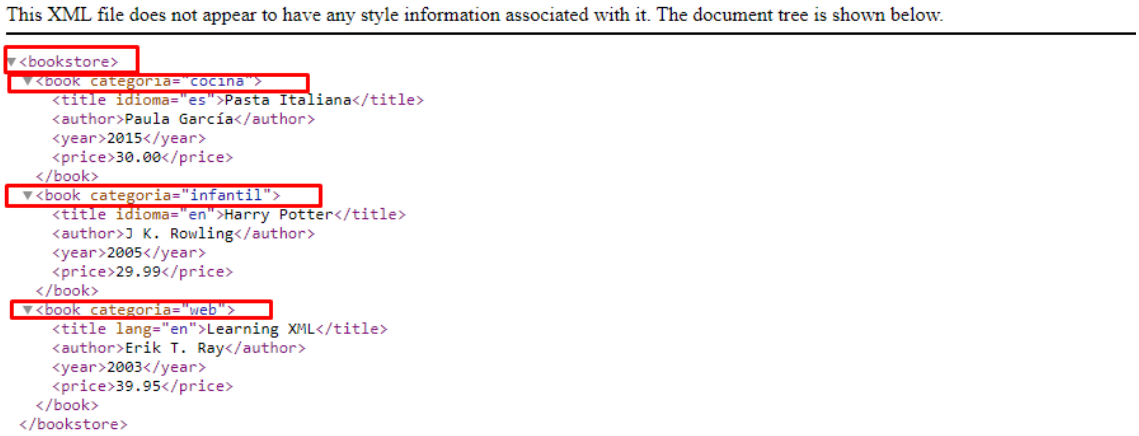
En JSON aunque no lo dijimos, no están permitidos los comentarios.

Ahora vamos a escribir un archivo denominado “libros.xml”, en el que se puede apreciar que existe una rama principal, 3 subramas con su contenido, y también vemos atributos dentro de algunos tags para poder categorizar la información. La información de los atributos siempre debe ir entre comillas dobles “” o simples ‘’.

Creamos un archivo denominado “libros.xml” y pegamos el siguiente código:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book categoria="cocina">
    <title idioma="es">Pasta Italiana</title>
    <author>Paula García</author>
    <year>2015</year>
    <price>30.00</price>
  </book>
  <book categoria="infantil">
    <title idioma="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book categoria="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Puedes probarlo y verificar que se ve correctamente.



Vemos claramente las 3 subramas de libros.

Como ejercicio introduce una subrama más de un libro con una categoría nueva y con los datos que desees.

Ejemplo:

```
<bookstore>
  <book categoria="cocina">
    <title idioma="es">Pasta Italiana</title>
    <author>Paula García</author>
    <year>2015</year>
    <price>30.00</price>
  </book>
  <book categoria="infantil">
    <title idioma="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book categoria="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
  <book categoria="Informática">
    <title lang="es">Aprendiendo Lenguaje de Marcas</title>
    <author>Fernando Gallego</author>
    <year>2024</year>
    <price>0</price>
  </book>
</bookstore>
```

Las reglas para la construcción de tags en archivos xml son:

- Son sensibles a mayúsculas y minúsculas.
- Los nombres de los elementos deben comenzar con una letra o un guión bajo.
- Los nombres de los elementos no pueden comenzar con las letras xml (o XML, o Xml, etc.)
- Los nombres de los elementos pueden contener letras, dígitos, guiones, guiones bajos y puntos.

- Los nombres de los elementos no pueden contener espacios.

Elementos vs Atributos

Con atributo:

```
<persona sexo="femenino">
  <nombre>Ana</nombre>
  <Apellido>Ocaña</Apellido>
</persona>
```

Sin atributo:

```
<persona>

 <sexo>femenino</sexo>
 <nombre>Ana</nombre>
 <Apellido>Ocaña</Apellido>
</persona>
```

Hacen la misma función en ambos casos, nos dan la misma información, no hay reglas en como especificar la información de atributos y campos o tags.

Ahora vamos a ver, al igual que hicimos con el formato json, como podemos acceder a los datos de un archivo xml que nos ha devuelto por ejemplo un servicio.

Para ello vamos a crear el siguiente archivo:

Tema23_1.html

Y incluimos el código siguiente:

```
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <!-- ETIQUETA PARA HACER RESPONSIVE LA PÁGINA, ADAPTACIÓN DE ESCALAS
-->
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Tema 23_1 html</title>
    <!-- ICONO DE VENTANA -->
    <link rel="icon" type="image/x-icon" href="imagenes/favicon.ico">
    <script>
        var archivoxml = "<libros><libro>" +
            "<titulo>Mi clase de lenguaje de Marcas</titulo>" +
            "<autor>Fernando Gallego</autor>" +
            "<año>2024</año>" +
            "</libro></libros>";


        //Borra toda la información del campo textarea:
        function Borrar() {
            document.getElementById("idArea").value = "";
        }
        //Lectura del campo titulo del xml:
        function lecturaTitulo() {
            document.getElementById("idArea").value =
xmlDoc.getElementsByTagName("titulo")[0].childNodes[0].nodeValue + "\n";
        }

        var parser = parser = new DOMParser();
        var xmlDoc = parser.parseFromString(archivoxml, "text/xml");
    </script>
    <style>
        textarea {
            padding: 10px;
            background-color: rgb(25, 78, 78);
            color: white;
            border-radius: 10px;
            border: solid 5px rgb(232, 107, 18);
```

```
        }

        input[type="button"] {
            padding: 10px;
            background-color: rgb(86, 142, 162);
            color: white;
            border-radius: 10px;
            border: solid 1px rgb(64, 49, 177);
        }

        label {
            color: rgb(44, 123, 160);
            font-size: x-large;
            font-weight: 800;
        }
    </style>
</head>

<body>
    <form>
        <label for="idArea">Datos leídos del xml</label><br>
        <textarea id="idArea" name="idArea"
style="width:90%;height:400px;"></textarea>
        <br><br>
        <input type="button" onclick="Borrar()" value="Borrar">
        <input type="button" onclick="lecturaTitulo()" value="Titulo">
    </form>
</body>
```

El ejercicio es parecido al que hicimos con la lectura del formato json, hemos aprovechado el mismo esqueleto.

Aquí lo que varía es la forma de tratar el contenido del texto:

Vimos que el json es el formato original de un objeto en JS, pero para poder tratar un contenido en XML, vamos a utilizar un parseador:

```
    var parser = parser = new DOMParser();
    var xmlDoc = parser.parseFromString(archivoxml, "text/xml");
```

Esto lo que hace es convertir el resultado en texto del contenido recibido en un texto xml (acodaros de los mime type).

```
xmlDoc.getElementsByTagName("titulo")[0].childNodes[0].nodeValue + "\n";
```

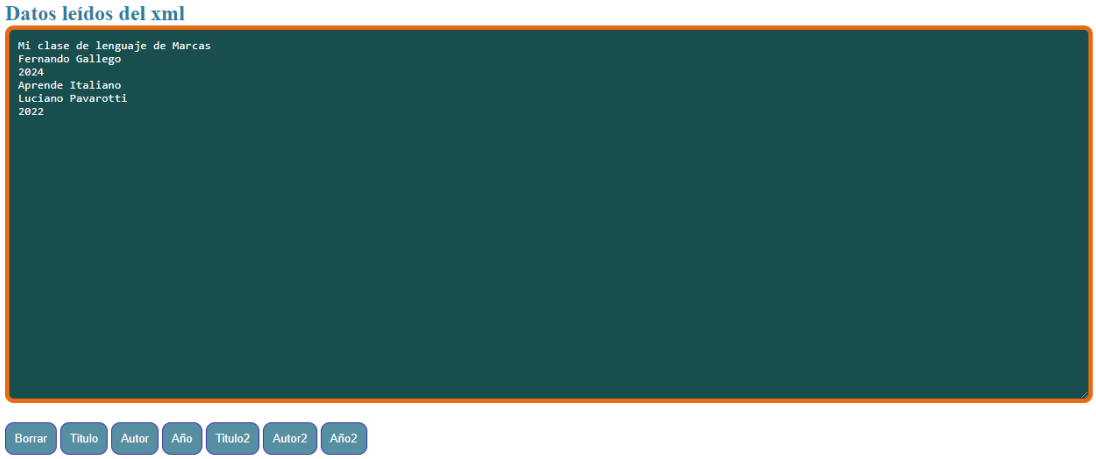
Una vez parseado el dato y guardado en la variable “xmlDoc”, la forma de acceder a la estructura de árbol del xml, es la siguiente:

getElementsByTagName accede por nombre del TAG, a continuación se le pone el índice, ya que puede haber varios tags con el mismo nombre (array de tags), al poner [0] nos trae el primer tag que encuentra, como no tiene hijos, a continuación aparece childNodes[0] y por último la propiedad “nodeValue” que devuelve el valor que tiene el tag seleccionado.

Como ejercicio el alumno debe de añadir dos botones más “autor” y “titulo” para mostrarlos en la pizarra, y que no se borren los datos, hasta que le das a borrar. Debería quedar de la siguiente forma:

Como avance del ejercicio anterior, el alumno debe de introducir un libro nuevo en el xml del ejercicio y poner 3 botones más para leer el título, autor y año del segundo libro.

Quedaría algo así:



Como último ejercicio el alumno debe añadir el siguiente libro al xml e intentar obtener los 3 datos de la fecha por separado, debería quedar algo así:

La información añadida al xml sería:

```
var archivoxml = "<libros><libro>" +
            "<titulo>Mi clase de lenguaje de Marcas</titulo>" +
            "<autor>Fernando Gallego</autor>" +
            "<año>2024</año>" +
            "</libro><libro>" +
            "<titulo>Aprende Italiano</titulo>" +
```

```
            "<autor>Luciano Pavarotti</autor>" +
            "<año>2022</año>" +
            "</libro><libro>"+
            "<titulo>XML y JSON</titulo>" +
            "<autor>Antonio Vans</autor>" +
            "<año><year>2022</year><month>04</month><day>01</day></año>"
+
            "</libro></libros>";
```

Por último vamos a ver las plantillas “XSL” para poder pintar los XML, algo así como los CSS para los HTML, sus hojas de estilo.

Por problemas de restricciones en los navegadores, vamos a probar más rápidamente como se ve un XML con su hoja de estilo en la siguiente URL:

Abre la siguiente url en el navegador:

https://xslttest.appspot.com/

Ahora en el archivo xml, vamos a poner el siguiente código:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="breakfast_menu.xsl"?>

<breakfast_menu>

<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
```

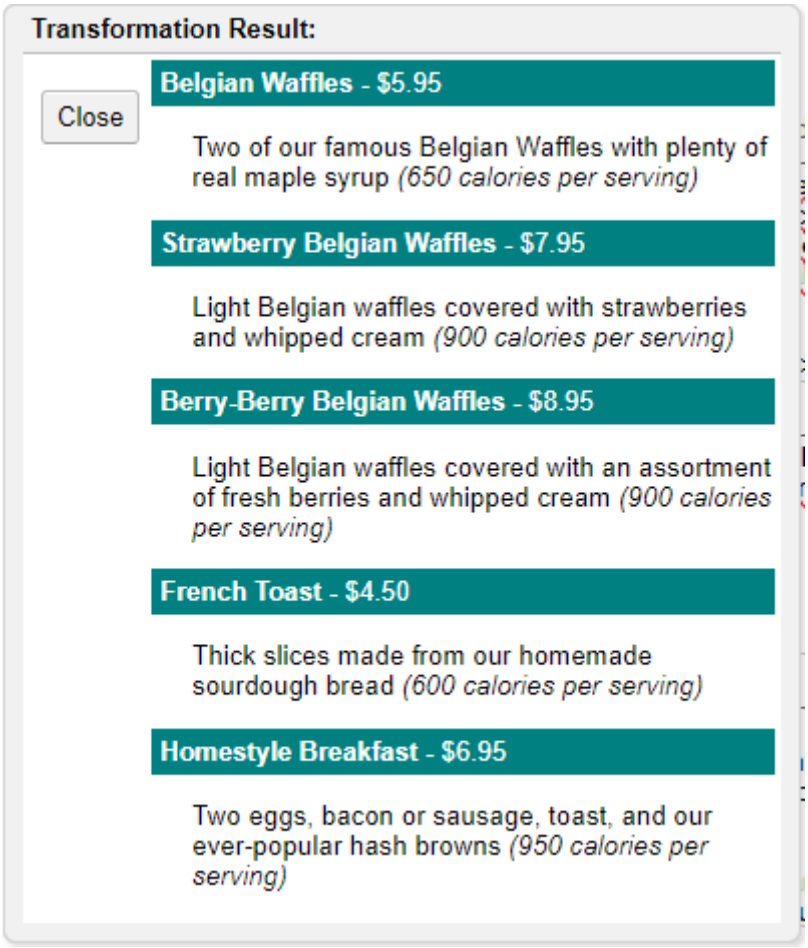
```
<description>Two of our famous Belgian Waffles with plenty of real maple
syrup</description>
<calories>650</calories>
</food>

<food>
<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>Light Belgian waffles covered with strawberries and whipped
cream</description>
<calories>900</calories>
</food>

<food>
<name>Berry-Berry Belgian Waffles</name>
<price>$8.95</price>
<description>Light Belgian waffles covered with an assortment of fresh
berries and whipped cream</description>
<calories>900</calories>
</food>

<food>
<name>French Toast</name>
<price>$4.50</price>
<description>Thick slices made from our homemade sourdough
bread</description>
<calories>600</calories>
</food>

<food>
<name>Homestyle Breakfast</name>
<price>$6.95</price>
<description>Two eggs, bacon or sausage, toast, and our ever-popular hash
browns</description>
<calories>950</calories>
</food>

</breakfast_menu>
```

Y en la parte del fichero XSL vamos a poner el siguiente archivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">
<xsl:for-each select="breakfast_menu/food">
  <div style="background-color:teal;color:white;padding:4px">
```

```
    <span style="font-weight:bold"><xsl:value-of select="name"/> -
</span>
    <xsl:value-of select="price"/>
    </div>
  <div style="margin-left:20px;margin-bottom:1em;font-size:10pt">
    <p>
    <xsl:value-of select="description"/>
    <span style="font-style:italic"> (<xsl:value-of select="calories"/>
calories per serving)</span>
    </p>
  </div>
</xsl:for-each>
</body>
</html>
```

Podemos ejecutar y ver la transformación del archivo XML como se ve utilizando su plantilla XSL.



Como parte última del tema, comentar que el formato JSON se está imponiendo al formato XML más antiguo, sobre todo por las diferencias que vamos a mostrar en la tabla siguiente:

JSON frente a XML

Aquí está la principal diferencia entre JSON y XML

JSON vs XML	
JSON	**XML**
El objeto JSON tiene un tipo	XML los datos no tienen tipo
Tipos JSON: cadena, número, matriz, booleano	Todos los datos XML deben ser cadenas.
Los datos son fácilmente accesibles como objetos JSON.	Es necesario analizar los datos XML.
Los archivos JSON son más humanos readable.	Los archivos XML son menos humanos readable.
JSON es compatible con la mayoría de los navegadores.	El análisis XML entre navegadores puede ser complicado
JSON no tiene capacidades de visualización.	XML proporciona la capacidad de mostrar datos porque es un lenguaje de marcado.
Recuperar valor es fácil	Recuperar valor es difícil
Compatible con muchos kits de herramientas Ajax	No es totalmente compatible con el kit de herramientas Ajax
Una forma totalmente	Los desarrolladores tienen que

JSON vs XML	
automatizada de deserializar/serializar JavaScript.	escribir código JavaScript para serializar/deserializar desde XML
Soporte nativo para objetos.	El objeto tiene que expresarse mediante convenciones, en su mayoría faltando el uso de atributos y elementos.