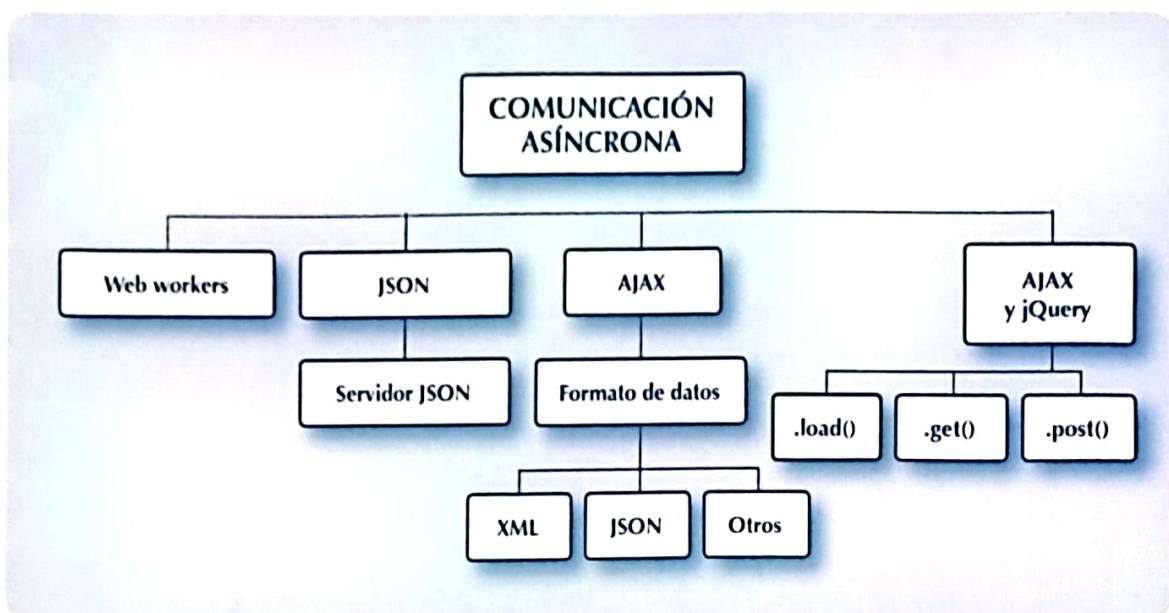


Mecanismos de comunicación asíncrona

Objetivos

- ✓ Entender el concepto de *web worker* y saber cómo se pueden realizar tareas en JavaScript de forma asíncrona.
- ✓ Comprender la sintaxis JSON y valorar el potencial de uso que puede tener para intercambiar datos entre la parte back-end y front-end.
- ✓ Conocer el modo de funcionamiento de AJAX y el tratamiento heterogéneo de datos que se puedan recibir (JSON, XML y otros tipos de datos).
- ✓ Utilizar jQuery y AJAX para realizar peticiones asíncronas a un servidor.

Mapa conceptual



Glosario

Asíncrono. Suceso o proceso que se produce de forma independiente al tiempo de procesamiento.

CORS (Cross Origin Resource Sharing). Mecanismo que, básicamente, consiste en acceder a información o a datos de otro dominio distinto al del origen de la petición.

jQuery. Biblioteca multipropósito que fue creada por John Resig y que ha sido mejorada en sucesivas versiones. Con esta librería, se puede manipular el DOM, crear animaciones, utilizar AJAX para realizar peticiones asíncronas al servidor, etcétera.

JSON (JavaScript Object Notation). Formato para el almacenamiento e intercambio de datos.

5.1. Introducción

La comunicación asíncrona es una de las cualidades que confiere a JavaScript su máxima potencia. Cuando AJAX surgió fue una revolución con respecto a las páginas web clásicas, puesto que permitía una comunicación y un refresco de información en una página web de forma que no había que recargarla de nuevo. Eso tiene todo su sentido, dado que ¿por qué tiene que refrescarse una página completa cuando solamente hay que cambiar el valor de un campo de esta?

Actualmente, no se concibe una aplicación web sin la utilización de este conjunto de técnicas. Hoy en día, el proceso en el lado del cliente ha ganado terreno frente a la lógica situada en el servidor, de tal manera que, en algunos frameworks, el back-end se utiliza solamente para persistencia de datos (guardar y recuperar datos de una base de datos).



CORS es el acrónimo de Cross Origin Resource Sharing y es un mecanismo por el cual una página web pide un permiso para poder acceder a recursos que se encuentran en otro servidor o dominio distinto al que pertenece el primero. Es muy común que una página web haga una petición XMLHttpRequest a otro servidor para cargar, por ejemplo, un JSON.

Los navegadores, por defecto, restringen este tipo de peticiones cruzadas por razones de seguridad. Tanto si el programador utiliza XMLHttpRequest como la API Fetch se va a encontrar con este problema, salvo que se utilicen encabezados CORS en las peticiones.

Si el lector quiere profundizar más en este concepto, puede acceder a las recomendaciones del W3C sobre CORS a través del siguiente código QR:



5.2. Los web workers

Un web worker es un proceso JavaScript que se ejecuta en segundo plano, de tal manera que la página web no se vea penalizada ni ralentizada por tener un proceso ejecutándose de forma permanente.

Se pueden utilizar web workers para analizar los patrones de navegación o para realizar llamadas asíncronas a un servicio de forma transparente al usuario.

Para ejecutar un web worker, el navegador tiene que soportar HTML5 y estar en una versión no excesivamente antigua.

A continuación, se muestra un ejemplo de web worker en el que se ha creado el web worker en un fichero aparte (es una buena práctica realizarlo así) llamado *trabajador.js*. El fichero HTML principal tendrá el siguiente aspecto:

```
<!DOCTYPE html>
<html>
<body>

```

```

        };
    } else {
        document.getElementById("result").innerHTML = "Fallo al ejecutar el web worker";
    }
}
function stopWorker() {
    w.terminate();
    w = undefined;
}
</script>
</body>
</html>

```

Como se puede observar, se ha incluido un botón para iniciar el web worker y otro botón para pararlo. Hay que tener en cuenta los siguientes aspectos del programa:

- El web worker se crea en la variable *w* siempre y cuando *w* tenga el tipo “undefined”.
- Cuando se crea el web worker, existe un modo de comunicación con la web host, que es el paso de mensajes. Se pueden capturar los mensajes mediante un listener “*w.onmessage = function(event).....*”.
- El web worker puede enviar mensajes a la página web host mediante la función *postMessage()*.

A continuación, se puede ver el contenido del web worker (fichero trabajador.js):

```

var anterior = 0;
var actual = 1;
var secuencia = "";
function temporizador() {
    if (!anterior){
        secuencia+=" "+actual;
    }else{
        secuencia+=" - "+actual;
    }
    postMessage(secuencia);
    aux = anterior + actual;
    anterior = actual;
    actual = aux;
    setTimeout("temporizador()",500);
}
temporizador();

```

5.3. JSON

JSON es el acrónimo de JavaScript Object Notation y es una sintaxis que se emplea para almacenar e intercambiar datos. Como se podrá verse a continuación, es una alternativa más fácil de usar a XML.

5.3.1. Sintaxis JSON

La sintaxis JSON en JavaScript tiene en cuenta cinco reglas:

1. Es un subconjunto de la sintaxis de JavaScript.
2. Los datos aparecen como pares nombre/valor.
3. Los datos se separan con comas.
4. Las llaves {} contienen objetos.
5. Los corchetes [] contienen arrays.

El siguiente ejemplo de JSON define el objeto empleados con un array que contiene tres registros del tipo empleado, el cual es un objeto:

```
{  
    "empleados": [  
        {"nombre": "Gerardo", "apellidos": "García"},  
        {"nombre": "Maryna", "apellidos": "Hernández"},  
        {"nombre": "Abel", "apellidos": "Pacheco"}  
    ]  
}
```

Entre las ventajas de JSON, cabe citar las siguientes:

- *Formato ligero de intercambio de datos.* Como se puede observar en el ejemplo anterior, se ha prescindido al máximo de elementos que sobrecarguen los datos de información innecesaria.
- *Independiente del lenguaje.* JSON utiliza la sintaxis JavaScript, pero el formato JSON es solo texto, al igual que XML. El texto puede leerse y utilizarse como formato de datos por cualquier lenguaje de programación. En el siguiente ejemplo, se podrá observar cuán fácil es incorporar datos en formato JSON a un código JavaScript.
- *Autodescriptivo y fácil de entender.*

A continuación, se muestra un ejemplo de utilización de datos en formato JSON:

```
<!DOCTYPE html>  
<html>  
<body>  
<h2>Ejemplo de objetos JSON</h2>  
<p id="demo"></p>  
<script>  
var text = '{"nombre":"IES MAR DE ALBORÁN","calle":"Fuente María Gil  
S/N","teléfono":"951 33 56 45"}';  
var obj = JSON.parse(text);  
document.getElementById("demo").innerHTML =  
obj.nombre + "<br>" +  
obj.calle + "<br>" +  
obj.teléfono;  
</script>
```

```
</body>  
</html>
```

Las diferencias entre JSON y XML son las siguientes:

- XML tiene que ser analizado con un analizador XML, mientras que JSON puede ser analizado por una función JavaScript estándar.
- JSON no utiliza etiqueta de fin.
- JSON es más corto.
- JSON es más rápido de leer y escribir.
- JSON puede utilizar matrices.
- JSON es más rápido y sencillo que XML.

Las similitudes entre ambos son:

- JSON y XML son autodescriptivos (legible por humanos).
- JSON y XML son jerárquicos (valores dentro de valores).
- JSON y XML pueden ser analizados y utilizados por muchos lenguajes de programación.
- Tanto JSON como XML se pueden obtener con una llamada XMLHttpRequest.

5.3.2. Práctica guiada: instalación de un servidor JSON

El objetivo de este ejercicio es mostrar cómo se puede instalar un servidor JSON en Node, desde el cual se puedan recuperar datos en este formato para luego ser utilizados en scripts (JavaScript).

Para este ejercicio, hay que tener Node instalado, en Ubuntu preferiblemente. En caso de no saber cómo instalar Node en su sistema operativo, se puede visitar el artículo siguiente: <https://nodejs.org/es/download/package-manager/>

1. Instalar el servidor JSON

El primer paso es instalar el módulo json-server en Node. Para instalar el servidor JSON, habrá que ejecutar el siguiente comando. De esa forma, se instalará de forma global el servidor.

```
sudo npm install json-server -g
```

2. Configurar el servidor JSON

Una vez instalado el módulo, hay que configurar el servidor creando una carpeta con el nombre que se desee (*star-wars*, por ejemplo). En la carpeta, se puede crear un fichero JSON con los datos o se puede descargar el archivo (<http://myfpschool.com/wp-content/uploads/2018/11/starwars.zip>), que contiene una carpeta llamada *public* y un archivo de nombre *db.json* con los datos con los que se va a trabajar en formato JSON. Si se opta por la descarga, la carpeta tendrá el aspecto de la figura 5.1.

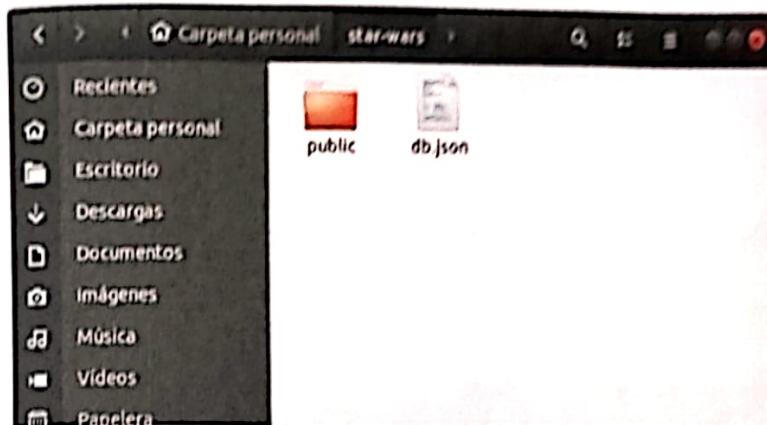


Figura 5.1
Estructura de archivos y carpetas del servidor JSON.

3. Arrancar el servidor JSON

Es el momento de arrancar el servidor JSON utilizando el siguiente comando. Esto arrancará el servidor en modo lectura en el puerto 3005.

```
json-server --watch db.json -p 3005
```

RECUERDA

- ✓ Si vas a hacer prácticas en tu equipo, lo mejor es que establezcas un *delay* para que la respuesta a tu aplicación sea más o menos la de un servidor de internet. Para un *delay* de un segundo y medio (-d 1500), como se puede observar, el tiempo se especifica en milisegundos:

```
json-server --watch db.json -p 3005 -d 1500
```

En la figura 5.2, se muestra el aspecto del terminal al ejecutar el comando anterior con el *delay*.

```
morenoperezjc@min1:~/star-wars$ json-server --watch db.json -p 3005 -d 1500
\{^_~}/ h1
Loading db.json
done

Resources
http://localhost:3005/people

Home
http://localhost:3005

Type s + enter at any time to create a snapshot of the database
watching...
```

Figura 5.2
Arrancando el servidor JSON.

4. Comprobar el funcionamiento del servidor JSON

Es el momento de comprobar si todo funciona. Para lo que hay que teclear dentro de un navegador la siguiente dirección:

`http://localhost:3005/people`

Si todo va bien, el resultado será el siguiente:

`testeando el servidor json`

Para comprobar si el servidor es capaz de mostrar imágenes, puede probarse a teclear la siguiente dirección:

`http://localhost:3005/adi_gallia.jpg`

Si el resultado es el que se muestra en la figura 5.3, el servidor estará funcionando correctamente.

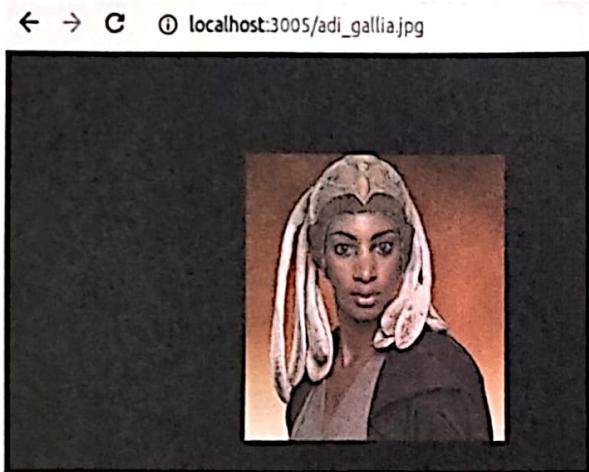


Figura 5.3
Comprobando el servidor JSON.

5. Parar el servidor JSON

Para parar el servidor JSON, bastaría con teclear `ctrl-C` en la ventana del terminal.

5.4. Comunicación asíncrona. AJAX

AJAX es el acrónimo de Asynchronous JavaScript and XML. Con esta técnica, se pueden crear páginas dinámicas de forma rápida que consultan de una forma asíncrona fuentes de datos externas a la página.

Esta tecnología permite que una página se actualice sin tener que enviar información otra vez al servidor (y esperar a que el servidor la responda y envíe información de vuelta).

Como únicamente se intercambian datos, solo se envía lo que se necesita (no toda la página).

TOMA NOTA



Es importante que, a partir de este capítulo, el lector se acostumbre a ejecutar las páginas web desde un servidor local o remoto. Utilizar un servidor local tipo XAMPP o LAMPP simplifica el desarrollo al no tener que subir los archivos al servidor remoto. Además, la ejecución será mucho más rápida.

Las páginas web anticuadas había que recargarlas de forma completa (imagine un formulario con campos desplegables tipo país, provincia, población, etc., que se tienen que recargar cada vez que se hace una selección). El programador tenía dos opciones, o traerse mucha información durante la carga, lo que ralentizaba mucho la navegación, o realizar muchas recargas de la información. Ambas opciones eran poco operativas.

Cuando apareció la comunicación asíncrona, muchas aplicaciones comenzaron a utilizar AJAX como Google Maps, Gmail, YouTube, Facebook Tabs, etc.



SABÍAS QUE...

AJAX fue introducido en Google Suggest por primera vez. Cuando se escribía un texto en el campo de entrada de búsqueda de Google, un script te ofrecía una serie de sugerencias, con lo cual la gente se quedaba sorprendida. Eso no era posible realizarlo con HTML y las páginas normales, obviamente, no lo hacían.

AJAX se basa en el objeto XMLHttpRequest para intercambiar datos con el servidor de forma asíncrona. Existen otros métodos, como la librería o API Fetch, que han mejorado este tipo de acceso con primitivas más optimizadas. Algunos frameworks utilizan un objeto iframe en vez del objeto XMLHttpRequest para intercambiar datos.

Aunque AJAX aconseja XML para intercambio de datos, ya que muchas aplicaciones utilizan JSON por ser un formato más ligero y ofrecer ciertas ventajas frente al primero.

En la figura 5.4, se muestra el esquema de funcionamiento de una web que utilice AJAX.

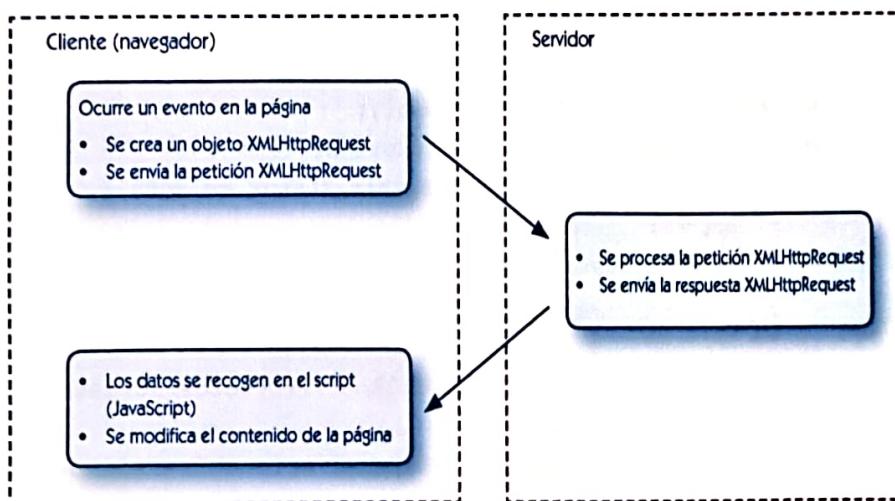


Figura 5.4
El mecanismo de comunicación asíncrona.

Como se puede observar, el procesamiento se produce una parte en el cliente y otra parte en el servidor.

5.4.1. Práctica guiada: sistema de localización de ciudades

Véase ahora un ejemplo de página web con un script en AJAX. En este ejemplo, se va a crear una pequeña página web en la que se pueda introducir el nombre de una ciudad y el sistema ofrezca sugerencias con ciudades que comiencen por las letras que se vayan introduciendo en un campo de texto. Básicamente, la página web tiene el aspecto que se muestra en la figura 5.5.

Escribe el nombre de alguna ciudad dónde quieras ir:

Ciudad: gij

Sugerencias: Gijón, Gibraltar

Figura 5.5
Ejemplo de la página web de sugerencia de ciudades.

El sistema estará compuesto de dos partes: en una, se tiene el código HTML (index.html) con los script JavaScript integrados y la pertinente llamada XMLHttpRequest y, en otra, un fichero php llamado *getSugerencia.php*, el cual recibe la petición y envía de vuelta el nombre de las ciudades que ha encontrado en una lista creada previamente.

Fichero index.html:

```
<html>
<head>
<script>
function muestraSugerencia(str) {
    if (str.length == 0) {
        document.getElementById("txtSugerencia").innerHTML = "";
        return;
    } else {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("txtSugerencia").innerHTML = this.responseText;
            }
        };
        xmlhttp.open("GET", "http://localhost/Proyectos/JC/getSugerencia.php?param=" + str, true);
        // La dirección anterior corresponde a la ubicación de la página php en el servidor localhost
        // Esta dirección podrá diferir dependiendo de la necesidad del programador.
    }
}
</script>
<body>
<input type="text" id="txtBusqueda" value="" onkeyup="muestraSugerencia(this.value)">
<div id="txtSugerencia"></div>
</body>
```

```

xmlhttp.send();
}

</script>
</head>
<body>

<p><b>Escribe el nombre de alguna ciudad donde quieras ir:</b></p>
<form>
Ciudad: <input type="text" onkeyup="muestraSugerencia(this.value)">
</form>
<p>Sugerencias: <span id="txtSugerencia"></span></p>
</body>
</html>

```

El script anterior va a realizar las llamadas a:

- `var xmlhttp = new XMLHttpRequest()`. Para crear un objeto del tipo XMLHttpRequest.
- `xmlhttp.open()`. Para llamar al servicio o página web donde se van a recibir los datos.
- `xmlhttp.send()`. Para enviar la petición.

Como se va a hacer una petición asíncrona, el programador tiene que decir al sistema qué código tiene que ejecutarse cuando ocurra algo con la petición que ha realizado. Cada vez que ocurra algo en la petición, se ejecutará la función asociada al evento o método `onreadystatechange` del objeto XMLHttpRequest creado previamente:

- `xmlhttp.onreadystatechange = function() {....}`

Una vez que se esté dentro de esta función, se recoge el valor recibido y, en este caso, se le asignará a un elemento HTML creado para tal fin, pero ¿por qué se espera que se cumpla la condición “`this.readyState == 4 && this.status == 200`”?

La respuesta es porque cuando:

- `this.readyState == 4`: significa que la petición ha terminado (completada) y que la respuesta, por lo tanto, está lista.
- `this.status == 200`: significa que la petición se ha completado OK. Otros valores típicos de este atributo pueden ser el famoso 404 (Not found) o el 403 (Forbidden).

TOMA NOTA



Valores del atributo readyState:

- 0 (no inicializada).
- 1 (leyendo).
- 2 (leído).
- 3 (interactiva).
- 4 (completo).

Ahora quedaría la parte servidora, que procesaría la petición AJAX. En este caso, sería un script php colocado en el servidor de nombre getSugerencia.php:

```
<?php  
// Array de nombres de ciudades  
$arr[] = "Casares";  
$arr[] = "Barcelona";  
$arr[] = "Alcorcón";  
$arr[] = "Málaga";  
$arr[] = "Fuengirola";  
$arr[] = "Marbella";  
$arr[] = "Río de Janeiro";  
$arr[] = "Gijón";  
$arr[] = "Oviedo";  
$arr[] = "Cáceres";  
$arr[] = "Mérida";  
$arr[] = "Estepona";  
$arr[] = "París";  
$arr[] = "Londres";  
$arr[] = "Berlín";  
$arr[] = "Moscú";  
$arr[] = "Atenas";  
$arr[] = "Kassel";  
$arr[] = "Praga";  
$arr[] = "Nueva York";  
$arr[] = "Dublín";  
$arr[] = "Manilva";  
$arr[] = "Gibraltar";  
$arr[] = "Ulan Bator";  
$arr[] = "Bombay";  
$arr[] = "Niza";  
$arr[] = "Pekín";  
$arr[] = "Tokio";  
$arr[] = "Vasteras";  
$arr[] = "Washington";  
$arr[] = "México DC";  
$arr[] = "Estocolmo";  
$arr[] = "Stocholm";  
  
// Primero recogemos el parámetro de la URL  
$param = $_REQUEST["param"];  
  
$sugerencia = "";  
  
// Mira si coincide el parámetro con alguna de las ciudades  
if ($param != "") {  
    $param = strtolower($param); //convierte el parámetro recibido a  
    minúscula
```

```

$len=strlen($param); //determinamos la longitud del parámetro recibido
foreach($arr as $nombre) {
// comparamos si el texto recibido coincide con el comienzo de alguna ciudad registrada en nuestro array
if (stristr($param, substr($nombre, 0, $len))) {
    if ($sugerencia === "") {
        $sugerencia = $nombre; // primera sugerencia
    } else {
        $sugerencia .= ", $nombre"; //segunda y siguientes sugerencias
    }
}
}

// En el caso de no encontrar ninguna sugerencia se le hace saber al usuario.
// en caso contrario devuelve las sugerencias encontradas
echo $sugerencia === "" ? "ninguna sugerencia" : $sugerencia;
?>

```

El script php es sencillo de comprender. En un bucle, se va recorriendo el array y seleccionando aquellas posiciones del array cuyas primeras letras coinciden con el texto que el usuario ha tecleado en la página web.

5.4.2. Ejemplo con AJAX y JSON

En este apartado, se explicará cómo se trata la información recibida en formato JSON por un script JavaScript. A continuación, se muestra un ejemplo sencillo en el que se tiene un archivo HTML llamado *json.html* y un archivo de texto *mywebs.txt* con información en formato JSON, más concretamente, un array de tres objetos en los que se almacena tanto el nombre como la dirección de tres páginas web.

A continuación, se muestra el contenido del fichero *json.html*:

```

<html>
<head>
</head>
<body>
<div id="lasWebs"></div>
<script>

var xmlhttp = new XMLHttpRequest();
var url = "mywebs.txt";
xmlhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
    var listaWebs = JSON.parse(this.responseText);
    getWebs(listaWebs);
}
};

xmlhttp.open("GET", url, true);
xmlhttp.send();

```

```

        function getWebs(arr) {
            var out = "";
            var i;
            for(i = 0; i < arr.length; i++) {
                out += '<a href=' + arr[i].url + '>' +
                    arr[i].display + '</a><br>';
            }
            document.getElementById("lasWebs").innerHTML = out;
        }
    </script>
</body>
</html>

```

Como se puede observar, al recibir los datos, se utiliza el método parse del objeto JSON para interpretar la información obtenida:

```
var listaWebs = JSON.parse(this.responseText);
```

Esta llamada creará un array de objetos, el cual se envía como parámetro a la función getWebs(), que lo único que hace es recorrerlo y mostrar el contenido en el elemento <div> de la página web con el ID igual a "lasWebs".

A continuación, se muestra el contenido del fichero mywebs.txt:

```
[
    {
        "display": "Web de MyFPSchool",
        "url": "http://myfpschool.com"
    },
    {
        "display": "IES Mar de alborán",
        "url": "http://www.maralboran.es"
    },
    {
        "display": "SomeBooks",
        "url": "http://www.somebooks.es"
    }
]
```

En este fichero se puede observar lo explicado anteriormente. Es un array de tres posiciones en las que tiene tres objetos, cada uno con dos campos: display y url.

Actividad propuesta 5.1



Verifica si el código expuesto en este apartado funciona correctamente en un navegador. Como característica especial, en vez de páginas web, utiliza razas de animales.

5.4.3. Ejemplo con AJAX y XML

Según el W3C, “el lenguaje XML (Extensible Markup Language) es un formato de texto simple y muy flexible derivado de SGML (ISO 8879). Se diseñó originalmente para la publicación electrónica a gran escala y en la actualidad desempeña un papel cada vez más importante en el intercambio de una amplia variedad de datos en la Web y en otros lugares”.

Primero se creará en el servidor un documento XML al que puedan realizarse consultas. El archivo al cual consultar sería el siguiente:

```
<concesionario>
<coche combustible="D">
    <marca>Volkswagen</marca>
    <modelo>Passat</modelo>
    <color>Azul</color>
    <año>2005</año>
    <precio>7900.00</precio>
</coche>
<coche combustible="G">
    <marca>Volkswagen</marca>
    <modelo>Touran</modelo>
    <color>Azul</color>
    <año>2007</año>
    <precio>8200.00</precio>
</coche>
<coche combustible="D">
    <marca>Opel</marca>
    <modelo>Astra</modelo>
    <color>Negro</color>
    <año>2013</año>
    <precio>8490.00</precio>
</coche>
</concesionario>
```

Este fichero se denominará *coches.xml*, se situará en el mismo directorio que el archivo HTML y será el fichero que servirá para realizar la consulta.

El siguiente paso es crear un archivo *coches.html* que contenga un script que lleve a cabo un filtrado sobre un determinado tipo de nodos del fichero *coches.xml*.

```
<!DOCTYPE html>
<html>
<body>

<p id="listado"></p>

<script>

var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
```

```

        if (this.readyState == 4 && this.status == 200) {
            recuperaDatos(this);
        }
    };
    xhttp.open("GET", "coches.xml", true);
    xhttp.send();

    function recuperaDatos(xml) {
        var datos = "";
        var xmlDatos = xml.responseXML;
        array = xmlDatos.getElementsByTagName("marca");
        for (i = 0; i < array.length; i++) {
            datos = datos + "marca: " + array[i].childNodes[0].nodeValue + "<br>";
        }
        document.getElementById("listado").innerHTML = datos;
    }

</script>
</body>
</html>

```

Como se puede observar, en las líneas de código:

```
var xmlDatos = xml.responseXML;
```

Se utiliza responseXML y no responseText, dado que el formato de los datos es XML.

Nótese que la variable array contiene los nodos que van a ser seleccionados, para lo cual se utiliza el método getElementsByTagName seleccionando solamente los nodos tipo “marca”. Al iterar sobre el array, se irá seleccionado el valor de los nodos “marca” del archivo XML.

Actividad propuesta 5.2



Basándote en el código expuesto en este apartado, carga los datos de un fichero XML con la temática que deseas.

5.5. Comunicación asíncrona con el servidor. AJAX y jQuery

Como ya se ha visto, AJAX es la forma que tiene JavaScript de comunicarse de forma asíncrona con un servidor. Utilizar jQuery es otra manera más de realizar peticiones por GET o POST a un servidor e incorporar el resultado a un campo de una página web.

RECUERDA

- ✓ A veces, AJAX puede cambiar, dependiendo del navegador en el que se esté utilizando. En ocasiones, el programador tiene que comprobar el tipo de navegador antes de ejecutar un código u otro.

Puedes encontrar más información en la página web de jQuery, accediendo a través del siguiente código QR:



5.5.1. Carga simple de datos load()

La función `.load` de jQuery permite cargar datos directamente del servidor en un campo HTML que el programador decida. El formato de esta función es el siguiente:

```
jQuery.load( url [, datos ] [, la_función ] )
```

Donde:

- *url* es un string que contiene la página web que se va a invocar.
- *datos* es un objeto o un string con la información que se quiere enviar al servidor.
- *la_función* es una función que se desea invocar cuando la petición se complete.

Este es el método más sencillo de obtener datos de un servidor utilizando jQuery. Una vez que la petición se completa, la función `load()` cargará los datos recibidos en el campo que el programador decida. En el siguiente ejemplo, los datos se cargan en un campo con identificador `id=list`:

```
$("#list").load("datos.html");
```

Como se puede ver en la orden anterior, los datos se reciben de la página web `datos.html`.

A continuación, se muestra un ejemplo de llamada vía AJAX-jQuery para recuperar los datos de una página web:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#btnDatos").click(function(){
        $("#list").load("datos.html");
    });
});
</script>
</head>
<body>
<h3>Asignaturas</h3>
<ul id="list"></ul>

<button id="btnDatos">Carga las asignaturas</button>

</body>
</html>
```

El fichero datos.html contendrá el siguiente código HTML:

```
<li>Desarrollo web en entorno cliente</li>
<li>Desarrollo web en entorno servidor</li>
<li>Programación</li>
<li>Entornos de desarrollo</li>
<li>Sistemas informáticos</li>
```

El aspecto de esta página web tras pulsar el botón será el que se observa en la figura 5.6.

Asignaturas

- Desarrollo web en entorno cliente
- Desarrollo web en entorno servidor
- Programación
- Entornos de desarrollo
- Sistemas informáticos

Carga las asignaturas

Figura 5.6
Aspecto del ejemplo
en el navegador.

Actividad propuesta 5.3



Crea un programa basado en el código expuesto en este apartado y verifica que funciona correctamente en un navegador.

5.5.2. Llamada asíncrona utilizando POST

A continuación se va a mostrar un ejemplo de llamada utilizando POST. Como todos los programadores saben, las llamadas por GET son similares. La función POST tiene el siguiente formato:

```
jQuery.post( url [, datos ] [, la_función ] [, dataType ] )
```

Donde:

- *url* es un string que contiene la página web que se va a invocar.
- *datos* es un objeto o un string con la información que se quiere enviar al servidor.
- *la_función* es una función que se desea invocar cuando la petición se complete.
- *dataType* es el formato de datos que se espera recibir del servidor (XML, JSON, script, text o HTML).

En el ejemplo siguiente, la llamada es:

```
$.post("suma.php",
        $("#formulario").serialize(),
        function(data,status){
            $('#respSuma').html(data);
        });

```

Se puede comprobar en el código anterior que se hace una llamada a la página web suma.php, a la cual se le envían los datos del formulario. Se utiliza el método serialize() para crear un string en formato URL-encoded con los datos del formulario y que pueda recibirlas correctamente el script en el lado del servidor.

Véase el ejemplo de página web que realiza la suma de dos números:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8" />
    <title> Suma dos números con AJAX y JQuery </title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script>
        $(document).ready(function(){
            $("#btnSuma").click(function(){
                $.post("suma.php",
                    $("#formulario").serialize(),
                    function(data,status){
                        $('#respSuma').html(data);
                    });
            });
        });
    </script>
</head>
<body>
    <form method="post" id="formulario">
        Introduzca dos valores para calcular la suma:<br>
        <input type="number" name="valor1" placeholder="valor1" autofocus/>
        <input type="number" name="valor2" placeholder="valor2"/>
        <input type="button" id="btnSuma" value="Suma" />
    </form>
    <br><br>
    <div id="respSuma"></div>
</body>
</html>
```

La página php (suma.php), como se puede apreciar en el código siguiente, es sumamente simple. La única función que realiza es la suma de los dos parámetros recibidos por POST:

```
<?php
    $valor1 = $_POST['valor1'];
    $valor2 = $_POST['valor2'];
    $suma = $valor1+$valor2;
    echo "La suma es: ".$suma;
?>
```

En la figura 5.7, el aspecto de la página web una vez ejecutada en el navegador.

Figura 5.7
Aspecto del ejemplo anterior en un navegador.

RECUERDA

- ✓ Load(), post() y get() son funciones reducidas de la llamada jQuery ajax(). La llamada post() del ejercicio expuesto en este apartado se podría haber sustituido por el siguiente código:

```
$.ajax({
    type: «POST»,
    url: url,
    data: $("#formulario").serialize(),
    success: function(data)
    {
        $('#respSuma').html(data);
    }
});
```

Como se puede observar, la llamada post() utilizada en el ejercicio es más intuitiva y genera menos líneas de código.

Actividad propuesta 5.4



¿Serías capaz de conseguir que el ejercicio expuesto en este apartado pudiese realizar la suma o la resta?

5.6. Práctica guiada: encuestas interactivas con AJAX

En este ejemplo, se va a crear una encuesta interactiva utilizando AJAX. Este tipo de encuestas es muy común en muchas páginas web. Esta encuesta hará una pregunta al usuario y mostrará el resultado total de todas las preguntas realizadas hasta el momento.

La página web constará de tres partes:

- Un fichero de texto (*resultados.txt*). Donde se almacenan los resultados.
- Una página html (*encuesta.html*). Donde se genera la encuesta.
- Una página php (*encuesta_voto.php*). Recoge el voto del usuario y calcula los tantos por ciento teniendo en cuenta los datos anteriores, además de publicar el resultado.

1. El fichero de texto (*resultados.txt*)

Es donde se van a almacenar los resultados de la encuesta. Se va a llamar *resultados.txt* y se va a iniciar con la siguiente línea de texto:

0||0||0||0

Cada uno de esos campos separados por || serán los votos que obtendrá cada una de las opciones. Conforme los usuarios vayan votando, el contenido del fichero irá cambiando:

19||3||6||0

2. La página web (*encuesta.html*)

La página web para la encuesta es sumamente sencilla. Consta de cuatro input de tipo radio que, al ser seleccionados, ejecutarán la función *getvoto()*. Esta función realiza una llamada AJAX a la página web php *encuesta_voto.php* pasándole como parámetro por GET el valor del voto emitido por el usuario.

```
<!DOCTYPE html>
<html>
<head>
<script>
function getvoto(int) {
    xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function() {
        if (this.readyState==4 && this.status==200) {
            document.getElementById("encuesta").innerHTML=this.responseText;
        }
    }
    xmlhttp.open("GET","encuesta_voto.php?voto="+int,true);
    xmlhttp.send();
}
</script>
</head>
<body>
<div id="encuesta">
<h3>¿Qué equipo crees que va a ganar la liga este año?</h3>
<form>
```

```

Real Madrid:
<input type="radio" name="voto" value="0" onclick="getvoto(this.
value)">
<br>
Barcelona:
<input type="radio" name="voto" value="1" onclick="getvoto(this.
value)">
<br>
Atlé&acute;tico de Madrid:
<input type="radio" name="voto" value="2" onclick="getvoto(this.
value)">
<br>
Sevilla:
<input type="radio" name="voto" value="3" onclick="getvoto(this.
value)">
</form>
</div>
</body>
</html>

```

El aspecto de la página web anterior será el que se muestra en la figura 5.8.

¿Qué equipo crees que va a ganar la liga este año?

- Real Madrid:
- Barcelona:
- Atlético de Madrid:
- Sevilla:

Figura 5.8

Aspecto de la página web anterior.

3. La página php (*encuesta_voto.php*)

En esta página php, se reciben los votos, se calculan los resultados y se le presentan al usuario. El código está comentado para una mejor comprensión.

```

<?php
    //Se abre el fichero de donde están almacenados los datos
    $fichero = "resultados.txt";
    $contenido = file($fichero);
    //colocamos el contenido en un array y lo almacenamos en cuatro va-
    riables por equipos
    $array = explode("||", $contenido[0]);
    $real = $array[0];
    $bar = $array[1];
    $atl = $array[2];
    $sev = $array[3];

```

```

//extraemos el voto de los participantes
$voto = $_GET['voto'];

//actualizamos los votos añadiendo el recibido a los anteriores
if ($voto == 0) {
    $real = $real + 1;
}
if ($voto == 1) {
    $bar = $bar + 1;
}
if ($voto == 2) {
    $atl = $atl + 1;
}
if ($voto == 3) {
    $sev = $sev + 1;
}

//creamos la cadena que se va a insertar en el fichero
$insertvoto = $real."||".$bar."||".$atl."||".$sev;
//se abre el fichero como escritura y se escriben los votos actualizados
$fp = fopen($fichero,"w");
fputs($fp,$insertvoto);
fclose($fp);

// se calcula el % del voto de cada uno de los equipos
$denominador=(int)$real+(int)$bar+(int)$atl+(int)$sev;
$tantoMadrid=100*round($real/$denominador,2);
$tantoBarcelona=100*round($bar/$denominador,2);
$tantoAtletico=100*round($atl/$denominador,2);
$tantoSevilla=100*round($sev/$denominador,2);

?>
<h2>Resultado:</h2>
<table>
    <tr>
        <td>Real Madrid:</td>
        <td>
            ' hei-
            ght='20'> <?php echo($tantoMadrid); ?>%
        </td>
    </tr>
    <tr>
        <td>Barcelona:</td>
        <td>
            ' 
            height='20'> <?php echo($tantoBarcelona); ?>%
        </td>
    </tr>
    <tr>
        <td>Atlético de Madrid:</td>

```

```

<td>
' hei-
ght='20'> <?php echo($tantoAtletico); ?>%
</td>
</tr>
<tr>
<td>Sevilla:</td>
<td>
' hei-
ght='20'> <?php echo($tantoSevilla); ?>%
</td>
</tr>
</table>

```

En la figura 5.9, se observa el aspecto del resultado una vez emitido el voto por parte del usuario.

Resultado:

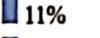
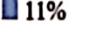
| | |
|---------------------|---|
| Real Madrid: |  68% |
| Barcelona: |  11% |
| Atlético de Madrid: |  11% |
| Sevilla: |  11% |

Figura 5.9
Aspecto de la página web
una vez emitido el voto.

Para mostrar el resultado con una barra, se ha utilizado una imagen de 30 píxeles de ancho y 200 píxeles de alto de color azul. Esa imagen tendrá la anchura del tanto por ciento de votos obtenido por cada equipo. De esta manera, el resultado quedará más vistoso.

Actividad propuesta 5.5



Añade un equipo a la encuesta y verifica que su programa funciona correctamente en un navegador.

Resumen

- Los web worker aparecen en HTML5.
- Para crear un web worker, se utiliza la sentencia `w = new Worker("trabajador.js");` donde `trabajador.js` es el fichero JavaScript que ejecutará el web worker.

- JSON es el acrónimo de JavaScript Object Notation, que es una sintaxis utilizada para almacenar e intercambiar datos.
- La sintaxis JSON en JavaScript sigue cinco reglas:
 1. Es un subconjunto de la sintaxis de JavaScript.
 2. Los datos aparecen como pares nombre/valor.
 3. Los datos se separan con comas.
 4. Las llaves {} contienen objetos.
 5. Los corchetes [] contienen arrays.
- JSON y XML son parecidos, pero JSON es más útil a la hora de programar porque es más corto, más rápido y más sencillo.
- Instalar un servidor JSON permite recuperar de una manera sencilla o incluso insertar datos desde una aplicación web.
- AJAX es el acrónimo de Asynchronous JavaScript and XML. Con esta técnica, se pueden crear páginas dinámicas de forma rápida que consultan de una forma asíncrona fuentes de datos externas a la página.
- AJAX se basa en el objeto XMLHttpRequest para intercambiar datos con el servidor de forma asíncrona.
- Los pasos de una petición AJAX son los siguientes:
 1. `var xmlhttp = new XMLHttpRequest();` Para crear un objeto del tipo XMLHttpRequest.
 2. `xmlhttp.open();` Para llamar al servicio o página web donde se van a recibir los datos.
 3. `xmlhttp.send();` Para enviar la petición.
 4. `xmlhttp.onreadystatechange = function() {....};`
- AJAX se suele utilizar con datos en formato JSON y XML.
- AJAX se integra también en jQuery con las funciones `load()`, `get()` y `post()`.

Ejercicios propuestos



1. Modifica la página web `suma.html` en la que se sumaban dos números utilizando AJAX y jQuery con la función `post()`. Cambia la función `post()` por la función más genérica `ajax()`, aunque el código sea algo más extenso.
2. Repite el ejercicio anterior, pero utiliza GET en vez de POST.
3. Transforma el siguiente código XML en formato JSON:

```
<departamento>
    <trabajador>
        <nombre>Pedro</nombre> <apellidos>Ruiz Aranda</apellidos>
    </trabajador>
    <trabajador>
```

```

<nombre>Isaac</nombre> <apellidos>Sánchez Alcalde</ape-
llidos>
</trabajador>
<trabajador>
<nombre>Lars</nombre> <apellidos>Arnstron</apellidos>
</trabajador>
</departamento>

```

4. Crea un webworker que vaya mostrando en la página web los números pares de forma incremental cada segundo.
5. Elabora una ventana parecida a la siguiente que valide mediante AJAX los datos de usuario y password introducidos. Si se introduce el usuario admin y la password 1234, el sistema dirá USUARIO VÁLIDO y, en caso contrario, responderá USUARIO NO VÁLIDO.

Introduzca usuario y password:

| |
|---------------|
| usuario |
| password |
| Log In |

Figura 5.10
Ejemplo de aplicación.

Introduzca usuario y password:

| |
|---------------|
| admin |
| *** |
| Log In |

USUARIO VALIDO

Figura 5.11
Acceso inválido.

6. Partiendo del ejercicio resuelto de la encuesta del apartado 5.6, prueba que funcionan los script anteriores, pero cambia la pregunta para que tenga 6 equipos.
7. ¿Qué es la API Fetch? Indica si puede ser una alternativa a las formas de comunicación asíncrona vistas hasta el momento.
8. Diseña una página web que calcule la cuota mensual de un préstamo. Para ello, tendrás que solicitar el capital, el interés y el número de meses necesarios para devolver dicho préstamo. El resultado tendrá que ser parecido a la figura 5.13.

Cálculo de la cuota de un préstamo:

| |
|--------------------|
| Capital solicitado |
| interés anual |
| Plazo (meses) |
| Calcular |

Su cuota mensual será de: Euros.

Figura 5.12
Aspecto de la página web
del ejercicio.

Cálculo de la cuota de un préstamo:

| |
|-----------------|
| 100 |
| 5 |
| 10 |
| Calcular |

Su cuota mensual será de: 10.230595941059 Euros.

Figura 5.13
Aspecto de la página web del ejercicio
tras realizar un cálculo.

9. Investiga si existe alguna función que permita cargar en un campo datos codificados en formato JSON utilizando la petición GET HTTP.
10. Teniendo los siguientes datos en formato JSON fichero ciclos.json, cárgalos en una lista en la página web.

Cargando los datos de ciclos:

Pulse para Cargar

Ciclos de informática:

- SMR - Sistemas microinformáticos y redes
- ASIR - Administración de sistemas informáticos y redes
- DAW - Desarrollo de aplicaciones web
- DAM - Desarrollo de aplicaciones multiplataforma

Figura 5.14

Aspecto de la página web pretendida de carga.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cómo se denomina un proceso JavaScript que se ejecuta en segundo plano?:

- a) Proceso sincrónico.
- b) Proceso event-driven.
- c) Web worker.

2. Los web workers funcionan a partir de:

- a) HTML5.
- b) JavaScript ES6.
- c) EmacScript

3. ¿Cuál de las siguientes opciones hay que cumplir para crear un web worker?:

- a) Tener una función startWorker().
- b) Crear un objeto del tipo Worker.
- c) Crear un objeto del tipo webWorker.

4. En JSON, ¿qué contienen las llaves {}?:

- a) Objetos.
- b) Arrays.
- c) Pares nombre/valor.

5. ¿Qué sintaxis utiliza JSON?:

- a) JavaScript.
- b) XML.
- c) Ni JavaScript ni XML.

6. ¿Cuál es el significado del acrónimo AJAX?:

- a) Asynchronous JSON and XML.
- b) Asynchronous JavaScript and XML.
- c) Asynchronous JavaScript and XML.

7. ¿Cuál de las siguientes empresas usó por primera vez AJAX?:

- a) Facebook.
- b) Twitter.
- c) Google.

8. Cuando se hace una XMLHttpRequest(), los datos estarán disponibles si:

- a) this.readyState == 4 || this.status == 200.
- b) this.readyState == 4 && this.status == 200.
- c) this.status == 4 && this.readyState == 200.

9. ¿Qué método se utiliza para interpretar una información en formato JSON?:

- a) JSON.parseJSON().
- b) JSON.parse().
- c) JSON.getData().

10. Cuando se procesa una petición de datos XML, ¿dónde se almacena el contenido?:

- a) XMLHttpRequest.responseXML.
- b) XMLHttpRequest.XMLdata.
- c) XMLHttpRequest.getXML().

SOLUCIONES:

1. **a** **b** **c**

5. **a** **b** **c**

2. **a** **b** **c**

6. **a** **b** **c**

9. **a** **b** **c**

3. **a** **b** **c**

7. **a** **b** **c**

10. **a** **b** **c**

4. **a** **b** **c**

8. **a** **b** **c**