

Programación AJAX en JavaScript.

Caso práctico

En estos últimos meses, **Antonio** ha realizado un montón de trabajos en el proyecto, y prácticamente ha terminado todas las tareas. **Juan** le dice que todo el trabajo realizado está muy bien, pero que tendría que actualizar algunos de los procesos para darle un toque de modernidad a la aplicación.

Entre las mejoras que le recomienda Juan, están la de utilizar efectos, más dinamismo, usar AJAX (JavaScript  asíncrono y XML) en las validaciones de los formularios, o en cierto tipo de consultas, etc. El término AJAX le suena muy complicado a **Antonio**, pero **Juan** lo convence rápidamente para que intente hacerlo, ya que no necesita aprender ningún lenguaje nuevo. Utilizando un nuevo objeto de JavaScript, con sus propiedades y métodos va a poder emplear AJAX en sus aplicaciones actuales.

Además, **Juan** lo anima a que se ponga a estudiar rápido el tema de AJAX, ya que al final, le va a dar una pequeña sorpresa, con una librería que le va a facilitar enormemente el programar con AJAX y conseguir dar buenos efectos y mayor dinamismo al proyecto web. Esa librería gratuita tiene el respaldo de grandes compañías a nivel mundial, que la están utilizando actualmente. También cuenta con infinidad de complementos, para que pueda modernizar su web todo lo que quiera, y todo ello programando muy pocas líneas de código y en un tiempo de desarrollo relativamente corto.



Elaboración Propia. ([CC BY-NC](#))

1.- Introducción a AJAX.

Caso práctico



[Didier Vidal Web 2.0 \(CC BY-SA\)](#)

AJAX es una tecnología crucial en lo que se conoce como web 2.0. A **Antonio** le atrae mucho el tema, ya que ha visto que con AJAX se pueden hacer envíos y consultas al servidor, sin tener que recargar las páginas web o cambiar de página, con lo que se consigue que sea todo más interactivo y adaptado a las nuevas tendencias. **Antonio** analiza la tecnología AJAX, sus orígenes, y el objeto que se utiliza para realizar las peticiones al servidor y gestionar las respuestas. Su directora **Ada**, le facilita unas

direcciones muy interesantes con contenidos y ejemplos de algunas aplicaciones AJAX de antiguos proyectos realizados en la empresa.

El término AJAX (JavaScript Asíncrono y XML) es una técnica de desarrollo web, que permite comunicar el navegador del usuario con el servidor, en un segundo plano. De esta forma, se podrían realizar peticiones al servidor sin tener que recargar la página, y podríamos gestionar esas respuestas, que nos permitirían actualizar los contenidos de nuestra página, sin tener que realizar recargas.

El término AJAX se presentó por primera vez en el artículo "A New Approach to Web Applications", publicado por Jesse James Garrett el 18 de febrero de 2005.

AJAX no es una tecnología nueva. Son realmente muchas tecnologías, cada una destacando por su propio mérito, pero que se unen con los siguientes objetivos:

- ✓ Conseguir una presentación basada en estándares, usando XHTML, CSS y un uso amplio de técnicas del DOM, para poder mostrar la información de forma dinámica e interactiva.
- ✓ Intercambio y manipulación de datos, usando XML y XSLT.
- ✓ Recuperación de datos de forma asíncrona, usando el objeto XMLHttpRequest.
- ✓ Uso de JavaScript, para unir todos los componentes.

Las tecnologías que forman AJAX son:

- ✓ XHTML y CSS, para la presentación basada en estándares.
- ✓ DOM, para la interacción y manipulación dinámica de la presentación.
- ✓ XML, XSLT y JSON, para el intercambio y manipulación de información.
- ✓ XMLHttpRequest, para el intercambio asíncrono de información.
- ✓ JavaScript, para unir todos los componentes anteriores.

El modelo clásico de aplicaciones Web funciona de la siguiente forma: la mayoría de las acciones del usuario se producen en la interfaz, disparando solicitudes HTTP al servidor web. El servidor efectúa un proceso (recopila información, realiza las acciones oportunas), y devuelve una página HTML al cliente. Este es un modelo adaptado del uso original de la Web como medio hipertextual, pero a nivel de aplicaciones de software, este tipo de modelo no es necesariamente el más recomendable.

Cada vez que se realiza una petición al servidor, el usuario lo único que puede hacer es esperar, ya que muchas veces la página cambia a otra diferente, y hasta que no reciba todos los datos del servidor, no se mostrará el resultado, con lo que el usuario no podrá interactuar de ninguna manera con el navegador. Con AJAX, lo que se intenta evitar, son esencialmente esas esperas. El cliente podrá hacer solicitudes al servidor, mientras el navegador sigue mostrando la misma página web, y cuando el navegador reciba una respuesta del servidor, la mostrará al cliente y todo ello sin recargar o cambiar de página.

AJAX es utilizado por muchas empresas y productos hoy en día. Por ejemplo, Google utiliza AJAX en aplicaciones como Gmail, Google Suggest, Google Maps.., así como Flickr, Amazon, etc.

Son muchas las razones para usar AJAX:

- ✓ Está basado en estándares abiertos.
- ✓ Su usabilidad.
- ✓ Válido en cualquier plataforma y navegador.
- ✓ Beneficios que aporta a las aplicaciones web.
- ✓ Compatible con Flash.
- ✓ Es la base de la web 2.0.
- ✓ Es independiente del tipo de tecnología de servidor utilizada.
- ✓ Mejora la estética de la web.

1.1.- Requerimientos previos.

A la hora de trabajar con AJAX debemos tener en cuenta una serie de requisitos previos, necesarios para la programación con esta metodología.

Hasta este momento, nuestras aplicaciones de JavaScript no necesitaban de un servidor web para funcionar, salvo en el caso de querer enviar los datos de un formulario y almacenarlos en una base de datos. Es más, todas las aplicaciones de JavaScript que has realizado, las has probado directamente abriéndolas con el navegador o haciendo doble clic sobre el fichero .HTML.

Para la programación con AJAX vamos a necesitar de un servidor web, ya que las peticiones AJAX que hagamos, lasaremos a un servidor. Los componentes que necesitamos son:

- ✓ Servidor web (apache, lightTPD, IIS, etc).
- ✓ Servidor de bases de datos (MySQL, Postgresql, etc).
- ✓ Lenguaje de servidor (PHP, ASP, etc).

Podríamos instalar cada uno de esos componentes por separado, pero muchas veces lo más cómodo es instalar alguna aplicación que los agrupe a todos sin instalarlos de forma individual. Hay varios tipos de aplicaciones de ese tipo, que se pueden categorizar en dos, diferenciadas por el tipo de sistema operativo sobre el que funcionan:

- ✓ servidor LAMP (Linux, Apache, MySQL y PHP).
- ✓ servidor WAMP (Windows, Apache, MySQL y PHP).

Una aplicación de este tipo, muy utilizada, puede ser XAMPP (tanto para Windows, como para Linux). Esta aplicación podrás instalarla incluso en una memoria USB y ejecutarla en cualquier ordenador, con lo que tendrás siempre disponible un servidor web, para programar tus aplicaciones AJAX.

 [Servidor XAMPP \(Apache, MySQL, PHP\).](#)



[Diehl P7170081\(CC BY\)](#)

Hasta hace unos años una extensión gratuita llamada **Firebug** para trabajar con AJAX (entre otras cosas). Su desarrollo se ha abandonado ya que el navegador **Firefox** y en especial **Firefox Developer Edition** lo han incorporado. Va a ser muy útil, ya que con ella podremos detectar errores en las peticiones AJAX, ver los datos que enviamos, en qué formato van, qué resultado obtenemos en la petición y un montón de posibilidades más, como inspeccionar todo el DOM de nuestro documento, las hojas de estilo, detectar errores en la programación con JavaScript, etc.

Existe una versión lite del plugin para Chrome por si quieras tener la interfaz de desarrollo más parecida, pero siempre es más cómodo desarrollar con Firefox que con Chrome en lo referido a AJAX. Probar a abrir una petición directamente en la barra de direcciones de los dos navegadores y veréis la diferencia. Abrid por ejemplo esta página:

- ✓ [https://api.nasa.gov/planetary/apod?
api_key=NNKOjkoul8n1CH18TWA9gwngW1s1SmjESPjNoUFo](https://api.nasa.gov/planetary/apod?api_key=NNKOjkoul8n1CH18TWA9gwngW1s1SmjESPjNoUFo)

Para saber más



[Herramientas de desarrolladores para los distintos navegadores.](#)

1.2.- Comunicación asíncrona.

Como ya te comentábamos en la introducción a AJAX, la mayoría de las aplicaciones web funcionan de la siguiente forma:

1. El usuario solicita algo al servidor.
2. El servidor ejecuta los procesos solicitados (búsqueda de información, consulta a una base de datos, lectura de fichero, cálculos numéricos, etc.).
3. Cuando el servidor termina, devuelve los resultados al cliente.

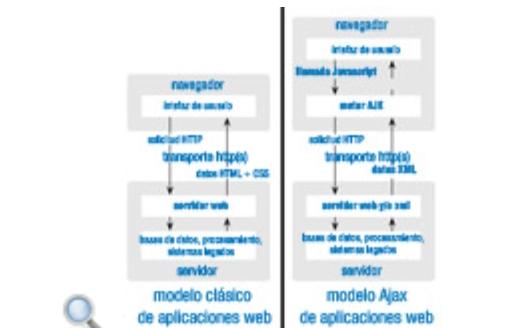


[Julio César Cerletti García Sincronía \(PS\)](#) ([CC BY-SA](#))

En el paso 2, mientras se ejecutan los procesos en el servidor, el cliente lo único que puede hacer es esperar, ya que el navegador está bloqueado en espera de recibir la información con los resultados del servidor.

Una aplicación AJAX, cambia la metodología de funcionamiento de una aplicación web, en el sentido de que, elimina las esperas y los bloqueos que se producen en el cliente. Es decir, el usuario podrá seguir interactuando con la página web, mientras se realiza la petición al servidor. En el momento de tener una respuesta confirmada del servidor, ésta será mostrada al cliente, o bien se ejecutarán las acciones que el programador de la página web haya definido.

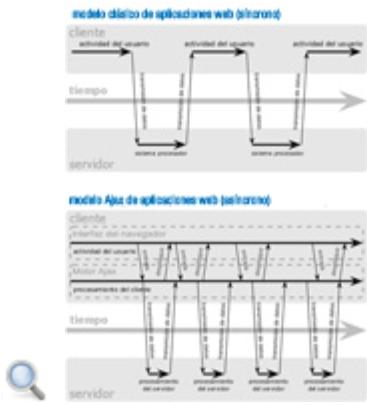
Mira el siguiente gráfico, en el que se comparan los dos modelos de aplicaciones web:



[Rafa Veiga speticiones-modelo-clasico-vs-modelo-ajax](#) ([CC BY-SA](#))

¿Cómo se consigue realizar la petición al servidor sin bloquear el navegador?

Para poder realizar las peticiones al servidor sin que el navegador se quede bloqueado, tendremos que hacer uso del motor AJAX (programado en JavaScript y que antiguamente se encuentra en un frame oculto). Este motor se encarga de gestionar las peticiones AJAX del usuario, y de comunicarse con el servidor. Es justamente este motor, el que permite que la interacción suceda de forma asíncrona (independientemente de la comunicación con el servidor). Así, de esta forma, el usuario no tendrá que estar pendiente del icono de indicador de carga del navegador, o viendo una pantalla en blanco.



Rafa Veiga [sincronia-modelo-clasivo-vs-ajax](#) (CC BY-SA)

Cada acción del usuario, que normalmente generaría una petición HTTP al servidor, se va a convertir en una petición AJAX con esa solicitud, y será este motor, el que se encargará de todo el proceso de comunicación y obtención de datos de forma asíncrona con el servidor, y todo ello sin frenar la interacción del usuario con la aplicación.

¿Te estás liando? Piensa Facebook en la web (no en la aplicación). Cada vez que bajas un rato hacia abajo ves que empiezan a cargarse más y más noticias de tus amigos. Pero algunas veces, sobretodo cuando notas que internet va lento, que ves como se para un segundo y después puedes continuar. Pues eso es una carga asíncrona de contenidos. No dejas parada la web y a ti con ella.

Piensa en un chat web con dos participantes en el que cada uno puede escribir y leer. Estos mensajes aparecen sin bloqueo que puede considerarse AJAX (aunque tenga una serie de mecanismos que permiten la comunicación en tiempo real aún más eficaz).

Autoevaluación

¿Según los gráficos anteriores, en qué modelo de aplicación web la actividad del usuario se ve interrumpida o bloqueada por la espera de las respuestas del servidor?

- Clásico.
- AJAX.

Citas para pensar

"Hablar,es el arte de sofocar e interrumpir el pensamiento."

Carlyle, Thomas.

1.3.- El API XMLHttpRequest.

El corazón de AJAX es una API denominada XMLHttpRequest ([XHR](#)), disponible en los lenguajes de scripting en el lado del cliente, tales como JavaScript. Se utiliza para realizar peticiones, HTTP o HTTPS, directamente al servidor web, y para cargar las respuestas directamente en la página del cliente. Los datos que recibamos desde el servidor se podrán recibir en forma de texto plano o texto XML. Estos datos, podrán ser utilizados para modificar el DOM del documento actual, sin tener que recargar la página, o también podrán ser evaluados con JavaScript, si son recibidos en formato JSON.



Otto Nassar wordle de mi delicious (CC BY-SA)

XMLHttpRequest juega un papel muy importante en la técnica AJAX, ya que sin este objeto, no sería posible realizar las peticiones asíncronas al servidor.

El concepto que está detrás del objeto XMLHttpRequest, surgió gracias a los desarrolladores de Outlook Web Access (de Microsoft), en su desarrollo de Microsoft Exchange Server 2000. La interfaz IXMLHTTPRequest, se desarrolló e implementó en la segunda versión de la librería MSXML, empleando este concepto. Con el navegador Internet Explorer 5.0 en Marzo de 1999, se permitió el acceso a dicha interfaz a través de ActiveX.

Posteriormente la fundación Mozilla, desarrolló e implementó una interfaz llamada nsIXMLHttpRequest, dentro de su motor Gecko. Esta interfaz, se desarrolló adaptándose lo más posible a la interfaz implementada por Microsoft. Mozilla creó un envoltorio para usar esa interfaz, a través de un objeto JavaScript, el cual denominó XMLHttpRequest. El objeto XMLHttpRequest fue accesible en la versión 0.6 de Gecko, en diciembre de 2000, pero no fue completamente funcional, hasta Junio de 2002 con la versión 1.0 de Gecko. El objeto XMLHttpRequest, se convirtió de hecho en un estándar entre múltiples navegadores, como Safari 1.2, Konqueror, Opera 8.0 e iCab 3.0b352 en el año 2005.

El W3C publicó una especificación-borrador para el objeto XMLHttpRequest, el 5 de Abril de 2006. Su objetivo era crear un documento con las especificaciones mínimas de interoperabilidad, basadas en las diferentes implementaciones que había hasta ese momento. La última revisión de este objeto, se realizó en Noviembre de 2009.

Microsoft añadió el objeto XMLHttpRequest a su lenguaje de script, con la versión de Internet Explorer 7.0 en Octubre de 2006.

Con la llegada de las librerías  [cross-browser](#) como jQuery, Prototype, etc, los programadores pueden utilizar toda la funcionalidad de XMLHttpRequest, sin codificar directamente sobre la [API](#), con lo que se acelera muchísimo el desarrollo de aplicaciones AJAX.

En febrero de 2008, la W3C publicó otro borrador denominado "XMLHttpRequest Nivel 2". Este nivel consiste en extender la funcionalidad del objeto XMLHttpRequest, incluyendo, pero no limitando, el soporte para peticiones  cross-site, gestión de  byte streams, progreso de eventos, etc. Esta última revisión de la especificación, se encuentra en un grupo de trabajo (W3C Working Group) cuya última revisión es del 18 de Noviembre de 2014.

Una de las limitaciones de XMLHttpRequest es que, por seguridad, sólo nos deja el mismo Dominio

No es lo mismo 127.0.0.1 que localhost.

Algunas veces cuando estamos desarrollando si por algún motivo no queremos instalar un servidor pero queremos probar con datos de ejemplo se pueden utilizar archivos en el sistema de archivos, pero con una restricción: usa Firefox. Es el único navegador a 2017 que te permite realizar pruebas locales.

Citas para pensar

"Tan sólo por la educación puede el hombre llegar a ser hombre. El hombre no es más que lo que la educación hace de él."

Kant, Immanuel.

1.3.1.- Creación del objeto XMLHttpRequest.

Para poder programar con AJAX, necesitamos crear un objeto del tipo XMLHttpRequest, que va a ser el que nos permitirá realizar las peticiones en segundo plano al servidor web.

Una vez más, nos vamos a encontrar con el problema de Internet Explorer, que, dependiendo de la versión que utilicemos, tendremos que crear el objeto de una manera o de otra. Aquí tienes un ejemplo de una función cross-browser, que devuelve un objeto del tipo XHR (XMLHttpRequest):



[Yusuke Kawasaki](#). *My ajax book in store now!* (CC BY)

```

///////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
/////////////////////////////
function objetoXHR()
{
    if (window.XMLHttpRequest)
    {
        // El navegador implementa la interfaz XHR de forma nativa
        return new XMLHttpRequest();
    }
    else if (window.ActiveXObject)
    {
        var versionesIE = new Array('MsXML2.XMLHTTP.5.0', 'MsXML2.XMLHTTP.4.0',
        'MsXML2.XMLHTTP.3.0', 'MsXML2.XMLHTTP', 'Microsoft.XMLHTTP');

        for (var i = 0; i < versionesIE.length; i++)
        {
            try
            { /* Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado. */
                return new ActiveXObject(versionesIE[i]);
            }
            catch (errorControlado) {}//Capturamos el error,
        }
    }
}

/* Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
Emitimos un mensaje de error usando el objeto Error.

Más información sobre gestión de errores en:
HTTP://www.javascriptkit.com/javatutors/trycatch2.shtml */

throw new Error("No se pudo crear el objeto XMLHttpRequest");
}

// para crear un objeto XHR lo podremos hacer con la siguiente llamada.
var objetoAJAX = new objetoXHR();

```

Sin embargo utilizar este objeto no es obligatorio y aunque te aísla de la complejidad de la conexión también puede dificultar la comprensión del AJAX y por eso habrá ejemplos con y sin esta clase. Ya es cuestión tuya su utilización.

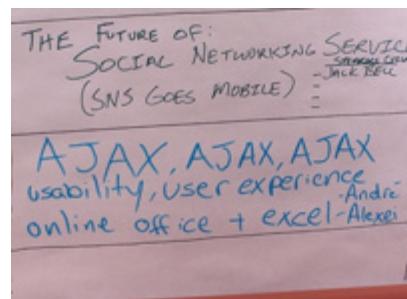
Una opción muy interesante, consiste en hacer una librería llamada, por ejemplo, funciones.js, que contenga el código de tus funciones más interesantes como, crearEvento(), `objetoXHR()`, etc. De esta manera, irás creando tus propios recursos, con el código de JavaScript que más uses en tus aplicaciones.

Citas para pensar

"El éxito no se logra sólo con cualidades especiales, es sobre todo un trabajo de constancia, de método y de organización." **Sargent, J.P.**

1.3.2.- Métodos del objeto XMLHttpRequest.

El objeto XMLHttpRequest dispone de los siguientes métodos, que nos permitirán realizar peticiones asíncronas al servidor:



[Andre Charland AJAX Session at MindCamp2.0 \(CC BY\)](#)

Métodos del objeto XMLHttpRequest.

Metodo	Descripción
abort()	Cancela la solicitud actual.
getAllResponseHeaders()	Devuelve la información completa de la cabecera.
getResponseHeader()	Devuelve la información específica de la cabecera.
open(metodo, url, async, usuario, password)	Especifica el tipo de solicitud, la <u>URL</u> , si la solicitud se debe gestionar de forma asíncrona o no, y otros atributos opcionales de la solicitud. <ul style="list-style-type: none">✓ método: indicamos el tipo de solicitud: GET o POST.✓ url: la dirección del fichero al que le enviamos las peticiones en el servidor.✓ async: true (asíncrona) o false (síncrona).✓ usuario y password: si fuese necesaria la autenticación en el
send (datos)	<ul style="list-style-type: none">✓ send(string) Envía la solicitud al servidor.✓ datos: Se usa en el caso de que estemos utilizando el método POST, como método de envío. Si usamos GET, datos será null.
setRequestHeader()	Añade el par etiqueta/valor a la cabecera de datos que se enviará al servidor.

Para probar el siguiente código, que incluye una petición AJAX, tienes que hacerlo a través del servidor web. Para ello debes extraer el ejemplo, dentro de la raíz del servidor web, arrancar el servidor web e ir a la dirección [HTTP://localhost/web/dwec07132](http://localhost/web/dwec07132)

Puedes utilizar Firebug, para comprobar como se realiza la petición AJAX.



[Descarga el código fuente del ejemplo dwec07132.](#) (0.01 MB)



[Geoff Parsons. AJAX on Rails \(CC BY-SA\)](#)

```

function cargarSync(objeto, url)
{

    if (miXHR)
    {
        alert("Comenzamos la petición AJAX"); // Recordar que alert no es un sistema de co
        // el usuario. Y debería utilizarse en las aplicaciones finales.

        //Si existe el objeto miXHR
        miXHR.open('GET', url, false); //Abrimos la url, false=SINCRONA

        // Hacemos la petición al servidor. Como parámetro del método send:
        // null -> cuando usamos GET.
        // cadena con los datos -> cuando usamos POST
        miXHR.send(null);

        //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
        textoDIV(objeto, miXHR.responseText);

        alert("Terminó la petición AJAX");
    }
}

```

En esta función se realiza una petición AJAX, pero de forma síncrona (comportamiento normal del navegador). En dicha petición se realiza la carga del fichero indicado en la url (debe ser un fichero perteneciente al mismo DOMinio del servidor). La respuesta (responseText), que obtenemos en esa petición, se coloca en un DIV, con la función personalizada textoDIV (su código fuente está en el fichero funciones.js).

En el siguiente código comentado, equivalente al anterior, puede verse una petición de la forma normal de AJAX sin utilizar la clase anteriormente vista:

```
function cargaAJAX() {
    var peticionXML;
    if (window.XMLHttpRequest) { // A partir del IE8 todos los navegadores funcionan
        peticionXML = new XMLHttpRequest();
    } else { // código para IE6 e IE7 pero que se puede obviar en los tiempos modernos
        peticionXML = new ActiveXObject("Microsoft.XMLHTTP");
    }

    // parámetros de open
    // 1º GET o POST.
    // URL del archivo.
    // async. true o false para comunicación asíncrona
    // usuario
    // contraseña

    peticionXML.open("GET", "http://servidor/peticion", true); // Subsustituir el servid
    // Send envia una petición al servidor. Si se utiliza
    // el método post en open, en send se puede enviar información
    // al servidor.

    peticionXML.send();
    // Aquí falta un método que veremos en la siguiente sección que se encarga de mo

}

// En las líneas siguientes estamos indicando que la llamada a la petición AJAX se hará
// cuando se pulse el botón.

window.addEventListener("load", function() {
    document.getElementsByTagName("button")[0].addEventListener("click", cargaAJAX);
});
```

1.3.3.- Propiedades del objeto XMLHttpRequest.

El objeto XMLHttpRequest, dispone de las siguientes propiedades, que nos facilitan información sobre el estado de la petición al servidor, y donde recibiremos los datos de la respuesta devuelta en la petición AJAX:

Propiedades del objeto XMLHttpRequest.

Propiedad	Descripción
onreadystatechange	Almacena una función (o el nombre de una función), que será llamada automáticamente, cada vez que se produzca un cambio en la propiedad readyState.
readyState	Almacena el estado de la petición XMLHttpRequest. Posibles estados, del 0 al 4: <ul style="list-style-type: none">✓ 0: solicitud no inicializada.✓ 1: conexión establecida con el servidor.✓ 2: solicitud recibida.✓ 3: procesando solicitud.✓ 4: solicitud ya terminada y la respuesta está disponible.
responseText	Contiene los datos de respuesta, como una cadena de texto.
responseXML	Contiene los datos de respuesta, en formato XML.
status	Contiene el estado numérico, devuelto en la petición al servidor (por ejemplo: "404" para "No encontrado" o "200" para "OK").
statusText	Contiene el estado en formato texto, devuelto en la petición al servidor (por ejemplo: "Not Found" o "OK").

Para probar el siguiente código, que incluye una petición AJAX, tienes que hacerlo a través del servidor web. Para ello debes extraer el ejemplo dentro de la raíz del servidor web, arrancar el servidor web e ir a la dirección <HTTP://localhost/web/dwec07133>

Puedes utilizar las herramientas de desarrollador, para comprobar como se está realizando la petición AJAX.



Karl Baron. [ajax universal \(CC BY\)](#)

 [Descarga el código fuente del ejemplo dwec07133.](#) (0.01 MB)

```
function cargarAsync(objeto, url)
{
    if (miXHR)
    {
        alert("Comenzamos la petición AJAX"); // Recordar que alert no es un sistema de co
        // el usuario. Y debería utilizarse en las aplicaciones finales.

        //Si existe el objeto miXHR
        miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

        // Hacemos la petición al servidor. Como parámetro:
        // null -> cuando usamos GET.
        // cadena con los datos -> cuando usamos POST
        miXHR.send(null);

        //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
        textoDIV(objeto, miXHR.responseText);

        alert("Terminó la petición AJAX"); // Mejor con un console.log para verlo en la co
    }
}
```

En esta función se realiza una petición AJAX, pero de forma asíncrona. En dicha petición se realiza la carga del fichero indicado en la url (debe ser un fichero perteneciente al mismo DOMinio del servidor). La respuesta (responseText), que obtenemos en esa petición, se coloca en un DIV, con la función personalizada textoDIV (su código fuente está en el fichero funciones.js). Si ejecutas el ejemplo anterior, verás que no se muestra nada, ya que no hemos gestionado correctamente la respuesta recibida de forma asíncrona. En el siguiente apartado 2, de esta unidad, veremos como corregir ese fallo y realizar correctamente esa operación.

Una petición completa y comentada en el código sería:

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

```

<head>
    <meta charset="utf-8">
    <title>Ajax</title>
    <script>

        function cargaXML() {
            var peticionXML;
            if (window.XMLHttpRequest) { // A partir del IE8 todos los navegadores funcionan
                peticionXML = new XMLHttpRequest();
            } else { // código para IE6 e IE7
                peticionXML = new ActiveXObject("Microsoft.XMLHTTP"); // Ya se pueda obviar
            }
            // La función siguiente se va a llamar cada vez
            // que readyState cambie.
            peticionXML.onreadystatechange = function() {
                // Los posibles status son
                // 200: Comunicación correcta.
                // 404: Pues que no hay comunicación.
                // Los posibles readyState son:
                // 0: Solicitud no inicializada
                // 1: Conexión establecida con el servidor.
                // 2: Solicitud recibida.
                // 3: Procesando solicitud
                // 4: Solicitud ya terminada y la respuesta está disponible.
                if (peticionXML.readyState == 4 && peticionXML.status == 200) {
                    document.getElementById("cambia").innerHTML = peticionXML.responseText;
                    // respuesta se muestra en el div con id cambia.
                    console.log(peticionXML.responseText);
                }
            } // fin de la función anónima
            // parámetros de open
            // 1º GET o POST.
            // URL del archivo.
            // async. true o false para comunicación asíncrona
            // usuario
            // contraseña

            peticionXML.open("GET", "http://servidor/peticion.php", true);
            // Send envia una petición al servidor. Si se utiliza
            // El método post en open, en send se puede enviar información
            // Al servidor.
            //() el tercer parámetro indica asíncrono si es true o síncrono si es false.

            peticionXML.send();
            // Además se pueden utilizar
            // -> setRequestHeader() es el otro método que sirve para
            // añadir el par etiqueta/valor a la cabecera de datos
            // que se encuentra en el servidor.
            // -> abort()
            // que cancela la petición actual.
            // -> getAllResponseHeaders() que devuelve toda la información

```

```
// de la cabecera.  
// -> getResponseHeader() que devuelve la información específica  
// de la cabecera.  
  
}  
window.addEventListener("load", function() {  
    document.getElementsByTagName("button")[0].addEventListener("click", cargaXML);  
});  
</script>  
</head>  
  
<body>  
  
    <button type="button">Muestra contenido AJAX</button>  
    <div id="cambia">  
        <h2>Cambia el contenido</h2>  
    </div>  
</body>  
  
</html>
```

2.- Envío y recepción de datos de forma asíncrona.

Caso práctico



[Jeremy Keith. Black Ajax \(CC BY\)](#)

Ahora que **Antonio** ya conoce el objeto XMLHttpRequest, con sus propiedades y métodos, se centra en cómo se realiza la petición al servidor de forma asíncrona, y cómo se gestionan los estados y las respuestas que nos devuelve. También estudia qué formatos tiene para enviar datos al servidor, y en qué formatos va a recibir esos resultados.

De entre todos los formatos de recepción de datos, **Juan** recomienda a **Antonio** uno de ellos: el formato JSON.

Dicho formato, utiliza la nomenclatura de JavaScript, para enviar los resultados. De esta forma puede utilizar dichos resultados directamente en la aplicación JavaScript, sin tener que realizar prácticamente ningún tipo de conversiones intermedias.

En el apartado 1.3.3., hemos visto un ejemplo de una petición asíncrona al servidor, pero vimos que éramos incapaces de mostrar correctamente el resultado en el contenedor resultados.

```

function cargarAsync(objeto, url)
{
    if (miXHR)
    {
        alert("Comenzamos la petición AJAX"); // Recordar que alert no es un sistema de co
        //con el usuario. Y NO debería utilizarse en las aplicaciones finales.

        //Si existe el objeto miXHR
        miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

        // Hacemos la petición al servidor. Como parámetro:
        // null -> cuando usamos GET.
        // cadena con los datos -> cuando usamos POST
        miXHR.send(null);

        //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
        textoDIV(objeto, miXHR.responseText);

        alert("Terminó la petición AJAX");
    }
}

```

Si ejecutas el ejemplo del apartado 1.3.3., verás que no se muestra nada en el contenedor resultados, y la razón es por que, cuando accedemos a la propiedad `miXHR.responseText`, ésta no contiene nada. Eso es debido a la solicitud asíncrona. Si recuerdas, cuando hicimos el ejemplo con una solicitud síncrona, se mostró la alerta de "Comenzamos la petición AJAX", aceptaste el mensaje, y justo 2 segundos después recibimos la alerta de "Terminó la petición AJAX". En el modo **síncrono**, el navegador cuando hace la petición al servidor, con el método `miXHR.send()`, se queda esperando hasta que termine la solicitud (y por lo tanto nosotros no podemos hacer otra cosa más que esperar ya que el navegador está bloqueado). Cuanto termina la solicitud, pasa a la siguiente línea: `textoDIV(objeto,...)`, y por tanto ya puede mostrar el contenido de `responseText`.

En el modo asíncrono, cuando aceptamos la primera alerta, prácticamente al instante se nos muestra la siguiente alerta. Ésto es así, por que el navegador en una petición asíncrona, no espera a que termine esa solicitud, y continúa ejecutando las siguientes instrucciones. Por eso, cuando se hace la llamada con el método `miXHR.send()`, dentro de la función `cargarAsync()`, el navegador sigue ejecutando las dos instrucciones siguientes, sin esperar a que termine la solicitud al servidor. Y es por eso que no se muestra nada, ya que la propiedad `responseText`, no contiene ningún resultado todavía, puesto que la petición al servidor está todavía en ejecución. **¿Cuál será entonces la solución?**

Una primera solución que se nos podría ocurrir, sería la de poner un tiempo de espera o retardo, antes de ejecutar la instrucción `textoDIV`. Ésto, lo único que va a hacer, es bloquear todavía más nuestro navegador, y además, tampoco sabemos exactamente lo que va a tardar el servidor web en procesar nuestra solicitud.

La segunda solución, consistiría en detectar cuándo se ha terminado la petición AJAX, y es entonces en ese momento, cuando accederemos a la propiedad `responseText` para coger los resultados. Ésta será la solución, que adoptaremos y que veremos en el apartado 2.1 de esta unidad.

En los siguientes ejemplos veremos usar el objeto XMLHttpRequest de varias formas distintas y en el apartado 2.8 información de REST y como podemos simular una petición AJAX con un archivo sin necesidad de un servidor (algo muy útil para probar nuestro conocimiento).

2.1.- Estados de una solicitud asíncrona (parte I).

Cuando se realiza una petición asíncrona, dicha petición va a pasar por diferentes estados (del 0 al 4 - propiedad **readyState** del objeto

XHR), independientemente de si el fichero solicitado al servidor, se encuentre o no. Atención: dependiendo del navegador utilizado, habrá algunos estados que no son devueltos.

Cuando dicha petición termina, tendremos que comprobar cómo lo hizo, y para ello evaluamos la propiedad **status** que contiene el estado devuelto por el servidor: 200: *OK*, 404: *Not Found*, etc. Si *status* fue *OK* ya podremos comprobar, en la propiedad **responseText** o **responseXML** del objeto XHR, los datos devueltos por la petición.



Roberto Zingales . Message error 404 (CC BY)



[Descarga el código fuente del ejemplo dwec0721.](#) (0.01 MB)

```

///////////////////////////////
// Función cargarAsync: carga el contenido de la url usando una petición AJAX de forma ASINC
/////////////////////////////
function cargarAsync(url)
{
    if (miXHR)
    {
        alert("Comenzamos la petición AJAX");

        //Si existe el objeto miXHR
        miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

        // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
        miXHR.onreadystatechange = estadoPeticion;

        // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por
        miXHR.send(null);
    }
}

/////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
/////////////////////////////
function estadoPeticion()
{
    switch(this.readyState) // Evaluamos el estado de la petición AJAX
    {
        // Vamos mostrando el valor actual de readyState en cada llamada.
        case 0: document.getElementById('estado').innerHTML += "0 - Sin iniciar.<br/>";
        break;
        case 1: document.getElementById('estado').innerHTML += "1 - Cargando.<br/>";
        break;
        case 2: document.getElementById('estado').innerHTML += "2 - Cargado.<br/>";
        break;
        case 3: document.getElementById('estado').innerHTML += "3 - Interactivo.<br/>";
        break;
        case 4: document.getElementById('estado').innerHTML += "4 - Completado.";
            if (this.status == 200)      // Si el servidor ha devuelto un OK
            {
                // Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
                textoDIV(document.getElementById("resultados"), this.responseText);
                alert("Terminó la petición AJAX");
            }
        break;
    }
}

```

En la forma habitual de Javascript se haría uso de una función anónima (o no si quieras que no lo sea que se encargaría de mostrar los valores.

```
// El código completo puedes verlo en el apartado 1.3.3
```

```
peticionXML.onreadystatechange = function() {
    switch(peticionXML.readyState) // Evaluamos el estado de la petición AJAX
    {      // Vamos mostrando el valor actual de readyState en cada llamada.
        case 0: document.getElementById('estado').innerHTML += "0 - Sin iniciar.<br/>";
        break;
        case 1: document.getElementById('estado').innerHTML += "1 - Cargando.<br/>";
        break;
        case 2: document.getElementById('estado').innerHTML += "2 - Cargado.<br/>";
        break;
        case 3: document.getElementById('estado').innerHTML += "3 - Interactivo.<br/>";
        break;
        case 4: document.getElementById('estado').innerHTML += "4 - Completado.";
            if (this.status == 200)      // Si el servidor ha devuelto un OK
            {
                // Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
                textoDIV(document.getElementById("resultados"), this.responseText);
                alert("Terminó la petición AJAX");
            }
        break;
    }
}
```

2.2.- Estados de una solicitud asíncrona (parte II).



[Andrew Mager. Google | 200 OK \(CC BY-SA\)](#)

El código del apartado 2.1, fue programado con fines didácticos para mostrar los 4 estados de la petición. El ejemplo completo, con una imagen animada indicadora de la actividad AJAX, se muestra a continuación:

 [Descarga el código fuente del ejemplo dwec0722.](#) (0.01 MB)

```

///////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
/////////////////////////////
function cargarAsync(url)
{
    if (miXHR)
    {
        // Activamos el indicador AJAX.
        document.getElementById("indicador").innerHTML=<img src='AJAX-loader.gif' />;

        //Si existe el objeto miXHR
        miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

        // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
        miXHR.onreadystatechange = estadoPeticion;

        // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por
        miXHR.send(null);
    }
}

/////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
/////////////////////////////
function estadoPeticion()
{
    // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
    // y por último se comprueba el status devuelto por el servidor==200.
    if ( this.readyState==4 && this.status == 200 )
    {
        // Desactivamos el indicador AJAX.
        document.getElementById("indicador").innerHTML="";

        // Metemos en el contenedor resultados las respuestas de la petición AJAX.
        textoDIV(document.getElementById("resultados"), this.responseText);
    }
}

```

Y ahora de otra forma equivalente, distinta al código que os descargáis.

```
// La función siguiente se va a llamar cada vez
// que readyState cambie.
peticionXML.onreadystatechange = function() {

    if (peticionXML.readyState == 4 && peticionXML.status == 200) {
        document.getElementById("cambia").innerHTML = peticionXML.responseText;
        // Si os fijáis es un texto, txt básico. Nada de XML o AJAX.
        console.log(peticionXML.responseText);
    }
} // fin de la función anónima
```

Si os fijáis también se puede recibir texto básico. Nada de XML o AJAX.

2.3.- Envío de datos usando método GET.



[Paul Downey. 418 I'm a Teapot \(CC BY\)](#)

Vamos a ver un ejemplo, en el que se realiza una petición AJAX a la página procesar.php, pasando dos parámetros: nombre y apellidos, usando el método GET.

 [Descarga el código fuente del ejemplo dwec0723.](#) (0.01 MB)


```

function iniciar()

{
    // Creamos un objeto XHR.
    miXHR = new XMLHttpRequest();

    // Cargamos de forma asíncrona el texto que nos devuelve la página
    // procesar.php con los parámetros indicados en la URL
    cargarAsync("procesar.php?nombre=Teresa&apellidos=Blanco Ferreiro");
}

///////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
/////////////////////////////
function cargarAsync(url)
{
    if (miXHR)
    {
        // Activamos el indicador AJAX antes de realizar la petición.
        document.getElementById("indicador").innerHTML=<img src='AJAX-loader.gif' />

        //Si existe el objeto miXHR
        miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

        // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
        miXHR.onreadystatechange = estadoPeticion;

        // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por
        miXHR.send(null);
    }
}
/////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
/////////////////////////////
function estadoPeticion()
{
    // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
    // y por último se comprueba el status devuelto por el servidor==200.
    if ( this.readyState==4 && this.status == 200 )
    {
        // Desactivamos el indicador AJAX.
        document.getElementById("indicador").innerHTML="";

        // Metemos en el contenedor resultados las respuestas de la petición AJAX.
        textoDIV(document.getElementById("resultados"), this.responseText);
    }
}

```

En la petición GET, los parámetros que pasemos en la solicitud, se enviarán formando parte de la URL. Por ejemplo: `procesar.php?nombre=Teresa&apellidos=Blanco Ferreiro`. Cuando realizamos la petición por el método GET, te recordamos una vez más, que pondremos `null` como parámetro del método `send`, ya que los datos son enviados a la página `procesar.php`, formando parte de la URL: `send(null)`.

Lo habitual es que cuando se está procesando una petición AJAX se tenga un gif animado que sea substituido cuando se realice la carga. Hay muchos gifs por internet, pero hay una web que permite personalizarlos:

 <http://www.ajaxload.info/>

Además tienen esta  [licencia](#) que te permite usarlos como te de la gana. Por ejemplo yo he hecho este gif animado.



Elaboración Propia. (Dominio público)

O uno más habitual:



*Elaboración Propia.
(Dominio público)*

2.4.- Envío de datos usando método POST.



[Post Box, DH6 96: George V, Ludworth Post Office, Co Durham \(CC BY\)](#)

Vamos a ver un ejemplo en el que se realiza una petición AJAX, a la página procesar.php pasando dos parámetros: nombre y apellidos, usando el método POST.

 [Descarga el código fuente del ejemplo dwec0724.](#) (0.01 MB)

```

function iniciar()
{
    // Creamos un objeto XHR.
    miXHR = new XMLHttpRequest();

    // Cargamos de forma asíncrona el texto que nos devuelve la página
    // procesar.php
    // En este caso sólo ponemos los parámetros que pasaremos a la página procesar.php
    cargarAsync("nombre=Teresa&apellidos=Blanco Ferreiro");
}

///////////////////////////////
// Función cargarASync: carga el contenido con los parametros
// que se le van a pasar a la petición AJAX de forma ASINCRONA.
/////////////////////////////
function cargarAsync(parametros)
{
    if (miXHR)
    {
        // Activamos el indicador AJAX antes de realizar la petición.
        document.getElementById("indicador").innerHTML=<img src='AJAX-loader.gif' />

        // Abrimos la conexión al servidor usando método POST y a la página procesar.php
        miXHR.open('POST', "procesar.php", true); // Abrimos la url, true=ASINCRONA

        // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
        miXHR.onreadystatechange = estadoPeticion;

        // En las peticiones POST tenemos que enviar en la cabecera el Content-Type
        // ya que los datos se envían formando parte de la cabecera.
        miXHR.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

        // Hacemos la petición al servidor con los parámetros: nombre=Teresa&apellidos=Bla
        miXHR.send(parametros);
    }
}

```

En este ejemplo, tuvimos que realizar los siguientes cambios para adaptarlo al método POST:

- ✓ La función cargarAsync(), recibirá los parámetros por POST en lugar de GET.
- ✓ El método .open se modifica por:
miXHR.open('POST', "procesar.php", true);
- ✓ Tenemos que crear una cabecera con el tipo de contenido que vamos a enviar, justo antes de enviar la petición con el método .send():
miXHR.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
- ✓ En el método .send() escribiremos los parámetros (nombre=Teresa&apellidos=Blanco Ferreiro) que serán enviados por el método POST:
miXHR.send(parametros).

2.5.- Recepción de datos en formato XML.

Cuando realizamos una petición AJAX, que nos devuelve las respuestas en formato XML, dichos datos los tendremos que consultar en la propiedad **responseXML** del objeto XHR.

 [Descarga el código fuente del ejemplo dwec0725.](#) (0.01 MB)

En la función iniciar(), le hemos dicho que cargue de forma asíncrona, empleando el método GET, el fichero datosXML.php. Esta aplicación PHP, nos devolverá un fichero XML, con una lista de Cd de música con el artista, país, compañía, etc.



Gideon Burton. [XML Connects](#) (CC BY-SA)

Instrucción de carga del fichero datosXML.php:
cargarAsync("datosXML.php");

Si queremos cargar directamente un fichero XML, y conocemos su nombre, podremos escribir directamente:

```
cargarAsync("catalogo.XML");
```

En la función de *estadoPeticion()*, cuando *readyState* es 4 y el *status* es OK (200), accedemos a los resultados de la petición AJAX, en la propiedad **responseXML**. Para gestionar los datos XML, tendremos que recorrerlos empleando los métodos del DOM, ya que un fichero XML comparte la estructura en árbol de un documento HTML, y podemos utilizar, por tanto, los mismos métodos que empleamos para recorrer el DOM HTML.

En nuestro caso, lo primero que vamos a hacer es recorrer los elementos , que son los que contienen toda la información referente a los cd's de música:

```
// Almacenamos el fichero XML en la variable resultados.  
resultados=this.responseXML;  
  
// Tenemos que recorrer el fichero XML empleando los métodos del DOM  
// Array que contiene todos los CD's del fichero XML  
CDs= resultados.documentElement.getElementsByTagName("CD");
```

Haremos un bucle para recorrer todos los cd's del catálogo, y dentro de cada uno, imprimiremos los datos que nos interesen:

```
// Hacemos un bucle para recorrer todos los elementos CD.  
for (i=0;i<CDs.length;i++)  
{  
....
```

Dentro de cada CD, accederemos al elemento que nos interese e imprimiremos su contenido. Para imprimir el contenido de cada nodo, tendremos que hacerlo con el comando `try {} catch {}`, ya que si intentamos acceder a un nodo que no tenga contenido, nos dará un error de JavaScript, puesto que el elemento hijo no existe, y entonces se detendrá la ejecución de JavaScript y no imprimirá nuestro listado.

```
// Para cada CD leemos el título
titulos=CDs[i].getElementsByTagName("TITLE");

try      // Intentamos acceder al contenido de ese elemento
{
    salida+=" " + titulos[0].firstChild.nodeValue + " |";
}
catch (er)      // En el caso de que no tenga contenido ese elemento imprimimos un espacio en
{
    salida+="  |";
}
```

Aquí para pasarlo al sistema normal sería poniendo dentro de la función anónima, pero esta vez calculando la media de precios de los CD al final. No se ha utilizado try catch, pero podéis hacerlo muy fácil.

```
peticionXML.onreadystatechange = function() {
    var texto="";
    var precio, preciomedio=0;

    if (peticionXML.readyState == 4 && peticionXML.status == 200) {
        elementoXML=peticionXML.responseXML; // Carga un objeto XML
        artistas=elementoXML.getElementsByTagName("ARTIST");
        for (var i=0; i<artistas.length; i++)
        {
            texto=texto + artistas[i].childNodes[0].nodeValue + "<br>"; // Otra
        }
        precio=elementoXML.getElementsByTagName("PRICE");
        // Ahora calculo el precio medio
        for (var i=0; i<precio.length; i++)
        {
            preciomedio=preciomedio + parseFloat(precio[i].childNodes[0].nodeValue);
            console.log(precio[i].childNodes[0].nodeValue);
        }

        preciomedio=preciomedio/precio.length;
        document.getElementById("cambia").innerHTML=texto+"<br />"+preciomedio;
        console.log(peticionXML.responseXML);
    }
}
```

2.6.- Recepción de datos en formato JSON (parte I).

Otro formato de intercambio muy utilizado en AJAX, es el formato JSON. JSON es un formato de intercambio de datos, alternativo a XML, mucho mas simple de leer, escribir e interpretar. Significa **Javascript Object Notation**, y consiste en escribir los datos en formato de Javascript. Vamos a hacer un poco de repaso:

Arrays

Se pueden crear con corchetes:

```
var Beatles = ["Paul", "John", "George", "Ringo"];
```

Con `new Array()`:

```
var Beatles = new Array("Paul", "John", "George", "Ringo");<br>
```

O también de la siguiente forma:

```
var Beatles = { "Paul", "John", "George", "Ringo"};<br>
```

Objetos

Un objeto literal se puede crear entre llaves: `{ propiedad1:valor, propiedad2:valor, propiedad3:valor}`

```
var Beatles = {  
    "Country" : "England",  
    "YearFormed" : 1959,  
    "Style" : "Rock'n'Roll"  
}
```

Que será equivalente a:

```
var Beatles = new Object();  
  
Beatles.Country = "England";  
Beatles.YearFormed = 1959;  
Beatles.Style = "Rock'n'Roll";
```



ozzy5836. Beatles (CC BY-SA)

Y para acceder a sus propiedades lo podemos hacer con:

```
alert(Beatles.Style); // Notación de puntos  
alert(Beatles["Style"]); // Notación de corchetes
```

Los objetos también pueden contener arrays literales: [...]

```
var Beatles = {  
    "Country" : "England",  
    "YearFormed" : 1959,  
    "Style" : "Rock'n'Roll",  
    "Members" : ["Paul", "John", "George", "Ringo"]  
}
```

Y los arrays literales podrán contener objetos literales a su vez: {...}, {...}

```
var Rockbands =  
[  
    { "Name" : "Beatles", "Country" : "England", "YearFormed" : 1959, "Style" : "Rock'n'Roll",  
        ["Paul", "John", "George", "Ringo"] }, { "Name" : "Rolling Stones", "Country" : "England",  
        "YearFormed" : 1962, "Style" : "Rock'n'Roll", "Members" : ["Mick", "Keith", "Charlie", "Bill"] }]
```

La sintaxis de JSON es como la sintaxis literal de un objeto, excepto que, esos objetos no pueden ser asignados a una variable. JSON representará los datos en sí mismos. Por lo tanto el objeto *Beatles* que vimos antes se definiría de la siguiente forma:

```
{  
    "Name" : "Beatles",  
    "Country" : "England",  
    "YearFormed" : 1959,  
    "Style" : "Rock'n'Roll",  
    "Members" : ["Paul", "John", "George", "Ringo"]  
}
```

Una cadena JSON es, simplemente, una cadena de texto, y no un objeto en sí misma. Necesita ser convertida a un objeto antes de poder ser utilizada en JavaScript. Ésto se puede hacer con la función eval() de JavaScript y también se pueden usar lo que se conoce como analizadores JSON, que facilitarán esa conversión.

2.7.- Recepción de datos en formato JSON (parte II).

Veamos un ejemplo completo en el que recibimos, con AJAX, los datos procedentes de un listado de una tabla MySQL en formato JSON.

 [Descarga el código fuente del ejemplo dwec0727.](#) (0.01 MB)

Si quieras probar el ejemplo, necesitas un servidor web, PHP y MySQL. Puedes instalarte por ejemplo XAMPP en cualquiera de sus versiones para Windows o Linux.



[Dion Hinchcliffe How mashups work using Google Maps as an example \(CC BY\)](#)

En este ejemplo, se realiza una consulta a una tabla (se adjunta código SQL de creación de la base de datos, usuario, contraseña y la tabla con los datos). La página PHP, devolverá los resultados en una cadena de texto, que contiene un array de objetos literales en formato JSON.

Código de PHP del fichero *datosJSON.php*:

```
...
// Consulta SQL para obtener los datos de los centros.
$sql="select * from centros order by nombrecentro";
$resultados=mysql_query($sql,$conexion) or die(mysql_error());

while ( $fila = mysql_fetch_array($resultados, MYSQL_ASSOC))
{
    // Almacenamos en un array cada una de las filas que vamos leyendo del recordset.
    $datos[]=$fila;
}

echo JSON_encode($datos); // función de PHP que convierte a formato JSON el array.
...
```

Nuestra aplicación de JavaScript recibe, por AJAX, esos datos, en la propiedad `responseText`. Para poder utilizar directamente esos datos en JavaScript, lo que tenemos que hacer es evaluar la expresión (cadena de texto, que recibimos de la página `datosJSON.php`). La expresión JSON contenía un array de objetos literales, por lo que tenemos que crear una variable, para asignar ese array y poder recorrerlo.

Para evaluar la expresión lo hacemos con la función `eval()` de JavaScript:

```
// Asignamos a la variable resultados la evaluación de responseText
var resultados=eval( '(' +this.responseText+')');

// Hacemos un bucle para recorrer todos los objetos literales recibidos en el array resultados
for (var i=0; i < resultados.length; i++)
{
    objeto = resultados[i];
    texto+="|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| " +objeto.nombrecentro+" | " + +objeto.localidad+" | " +objeto.provincia+" | " +objeto.telefono+" | " +obje     objeto.numvisitantes+ " |
";
}
}
```

Para saber más

 [Para depurar en Firefox AJAX.](#)

 [Para depurar en Chrome AJAX.](#)

Para saber más

Si haces alguna vez un formato de AJAX y tienes errores extraños lo mejor es comprobarlo en algún validador/formateador como:

 <http://jsonlint.com/>

 <https://jsonformatter.curiousconcept.com/>

Citas para pensar

"Sivivir es durar, prefiero una canción de los Beatles a un Long Play de los Boston Pops. - Mafalda." **Quino, Joaquín Salvador Lavado.**

2.8.- API REST y más.

Si estás matriculado en DWES seguro que ya conoces como funciona y no tendrás problemas en entender y utilizar los ejemplos anteriores. Pero muchos de vosotros sois alumnos que han empezado con segundo directamente y no sabéis PHP. En este capítulo vamos a aprovechar la posibilidad de usar archivos txt donde tu puedas modificar fácilmente el contenido antes de hacer una aplicación PHP para generarlos desde una base de datos.



Paul & Hien Brown. JSON... (CC BY-NC)

Firefox te permite hasta el momento probar ese archivo de texto directamente en el equipo local sin tener que subirlo a un servidor. Chrome y la mayoría de los otros navegadores te obligan a subirlo a un servidor web.

Por ejemplo

```
...
    peticionXML.open("GET", "http://127.0.0.1/catalogo.xml", true); // De esta forma
    // estamos pidiendo la información a un archivo en nuestro servidor web.
// 0 con archivo local. Solamente para pruebas.
    peticionXML.open("GET", "catalogo.xml", true);
...

```

La segunda petición sólo funciona en Firefox. La primera va bien en todos.

Hay otro aspecto muy usado en el mundo aunque no esté nombrado explícitamente en muchos sitios. Para conectarnos a un servidor y realizar AJAX lo que estamos haciendo es usar un protocolo de comunicación basado en JSON. O sea usar el protocolo HTTP, para trasmitir objetos entre dos partes usando JSON.

Así pues, REST es un sistema creado para poder transmitir información a través del protocolo HTTP y que se utiliza en AJAX para transmitir información asíncronamente entre el servidor y tu web. Antes trasmitir información entre dos equipos de distinta arquitectura era algo complicado. Se utilizaban protocolos como SOAP. Ahora es mucho más sencillo ya que los datos se transmiten a través de JSON. De hecho XML es opcional y sólo se usa cuando sea necesario así que el rey de las REST es JSON y no XML. De otra manera, no es necesario ningún tipo de conversión usando nodes, `childnodes`, etc. y es realmente sencillo que cualquiera defina una pequeña arquitectura de este tipo.

Internet está repleto de APIs que son REST. Por ejemplo [Google](#) te permite acceder a los mapas (previo pago en algunos niveles). OpenStreetMaps también te permite usar una API, pero en este caso gratuita y estable en el tiempo.

- ✓  [http://api.openstreetmap.org/api/0.6/notes?
bbox=-0.65094,51.312159,0.374908,51.669148](http://api.openstreetmap.org/api/0.6/notes?bbox=-0.65094,51.312159,0.374908,51.669148)
- ✓  [http://api.openstreetmap.org/api/0.6/notes.json?
bbox=-0.65094,51.312159,0.374908,51.669148](http://api.openstreetmap.org/api/0.6/notes.json?bbox=-0.65094,51.312159,0.374908,51.669148)

Esa dirección se le pasa a la petición y se obtiene un archivo un XML (en el primer ejemplo) y un JSON (en el último) como respuesta dentro de un cuadro de coordenadas (longitud entre mínima y máxima, latitud y igualmente).

Hay muchísimos servicios públicos (y gratuitas) que utilizan REST. Podéis encontrar un listado en el siguiente enlace:

- ✓  [Listado de APIs públicas.](#)

Para saber más



Para saber más. Fetch API.

Hace no mucho se añadió a javaScript el método fetch. Este método va a permitir simplificar las conexiones AJAX a cambio de tener que trabajar con async await en algunos casos. Aún así tienes que conocer el objeto XMLHttpRequest si quieras sacarle partido que es lo que hemos estado viendo en los puntos anteriores.



También tienes la posibilidad de evitar de usar jquery como se ve en el punto 3. Es más sencillo didácticamente, pero puede ser más lento en ejecución. Y es lo que está en el contenido del decreto ya que tenemos que usar librerías o frameworks.

3.- Librerías cross-browser para programación AJAX.

Caso práctico

El estudio del objeto para trabajar con AJAX, está comenzando a dar sus frutos. Lo que más le fastidia a **Antonio**, es que necesita programar bastante código, y aunque puede crear alguna librería para acelerar la programación, también ve que las diferentes incompatibilidades, entre navegadores, no van a ayudar nada en esta labor. Por esta razón está un poco desilusionado, por que le va a suponer bastante trabajo, aunque los resultados merecen la pena.



Elaboración Propia. ([CC BY-NC](#))

En ese momento llega **Juan**, y le da la sorpresa que le había comentado hace unos días. Le facilita un pequeño tutorial sobre la librería jQuery, que le va a permitir hacer peticiones AJAX utilizando prácticamente una línea de código. No tendrá que preocuparse por temas de cross-browsing y, además, la misma librería le facilitará métodos para hacer todo tipo de efectos, animaciones, etc. Esta librería cuenta además con infinidad de complementos gratuitos, que permiten hacer prácticamente cualquier cosa en muy poco tiempo.

La programación con AJAX, es uno de los pilares de lo que se conoce como web 2.0, término que incluye a las aplicaciones web que facilitan el compartir información, la interoperabilidad, el diseño centrado en el usuario y la colaboración web. Ejemplos de la web 2.0, pueden ser las comunidades web, los servicios web, aplicaciones web, redes sociales, servicios de alojamiento de vídeos, wikis, blogs, [mashup](#), etc.

Para saber más



[Más información sobre Mashup \(aplicaciones web híbridas\).](#)

Gracias a las aplicaciones web 2.0, se han desarrollado gran cantidad de utilidades/herramientas/frameworks para el desarrollo web con JavaScript, DHTML (HTML dinámico) y AJAX. La gran ventaja de usar alguna librería o framework para AJAX, es la del ahorro de tiempo y código, en nuestras aplicaciones. Veremos que con algunas librerías vamos a realizar peticiones AJAX, con una simple instrucción de código sin tener que preocuparnos de crear el objeto XHR, ni gestionar el código de respuesta del servidor, los estados de la solicitud, etc.

Otra de las ventajas que nos aportan este tipo de librerías, es la de la compatibilidad entre navegadores (cross-browser). De esta forma tenemos un problema menos, ya que la propia librería será capaz de crear la petición AJAX de una forma u otra, dependiendo del navegador que estemos utilizando.

A principios del año 2008 Google liberó su API de librerías AJAX, como una red de distribución de contenido y arquitectura de carga, para algunos de los frameworks más populares. Mediante esta API se eliminan las dificultades a la hora de desarrollar mashups en JavaScript. Se elimina el problema de alojar las librerías (ya que están centralizadas en Google), configurar las cabeceras de cache, etc. Esta API ofrece acceso a las siguientes librerías Open Source, realizadas con JavaScript:

- ✓ jQuery. Fácil y muy usada.
- ✓ Angular. Tenía mucho futuro y estaba superando en uso en proyectos a jQuery. Pero Google va a utilizar TypeScript, un JavaScript con tipos creado por Microsoft, y van a romper compatibilidad. Así que Angular 2.0 será un proceso de volver aprender, al menos eso dice Google en el momento de la creación de este documento.
- ✓ Backbone.js. No está nada mal.
- ✓ scriptaculous.
- ✓ mootools.
- ✓ Dojo. Siempre muy prometedora.
- ✓ prototype.js (pero ya lleva 11 meses sin actualizaciones).
- ✓ Fuera del objetivo del módulo podéis programar aplicaciones ligeras para dispositivos móviles con Cordova (PhoneGap libre) (Android, ios, Windows Phone, Windows 8, etc..).

Los scripts de estas librerías están accesibles directamente utilizando la URL de descarga, o a través del método `google.load()` del cargador de la API AJAX de Google.

Hay muchísimas librerías que se pueden utilizar para programar AJAX, dependiendo del framework que utilicemos. En el siguiente listado se ven las que crecieron mucho en el 2016.

[Las 40 mejores librerías/frameworks Javascript en 2022.](#)

Nosotros nos vamos a centrar en el uso de la librería jQuery, por ser una de las más utilizadas hoy en día por empresas como Google, DELL, digg, NBC, CBS, NETFLIX, mozilla.org, wordpress, drupal, etc.

Ahora hay una cuestión que preguntarnos sobre jQuery. Tienen tres versiones activas. ¿Cuál utilizar?



[Luis Villa del Campo. AJAX - Super Detergente \(CC BY\)](#)

Hay tres versiones principales. La rama de la versión 1, la rama de la versión 2 y la de la 3. La versión 2 y la 3 cumple los requisitos de cualquier navegador moderno, es bastante más rápida pero tiene un problema. El soporte de las versiones antiguas de los navegadores e Internet Explorer. Algunos estaréis pensando que eso ya no se utiliza. Por desgracia algunas empresas tienen desarrolladas sus aplicaciones para versiones primitivas de Internet Explorer y para esos caso es necesario utilizar jQuery 1.

3.1.- Introducción a jQuery (parte I).



Desconocido. [Logo von jQuery](#) (Dominio público)

jQuery es un framework JavaScript, que nos va a simplificar muchísimo la programación. Como bien sabes, cuando usamos JavaScript tenemos que preocuparnos de hacer scripts compatibles con varios navegadores y, para conseguirlo, tenemos que programar código compatible.

jQuery nos puede ayudar muchísimo a solucionar todos esos problemas, ya que nos ofrece la infraestructura necesaria para crear aplicaciones complejas en el lado del cliente. Basado en la filosofía de "*escribe menos y produce más*", entre las ayudas facilitadas por este framework están: la creación de interfaces de usuario, uso de efectos dinámicos, AJAX, acceso al DOM, eventos, etc. Además esta librería cuenta con infinidad de plugins, que nos permitirán hacer presentaciones con imágenes, validaciones de formularios, menús dinámicos, drag-and-drop, etc.

Esta librería es gratuita, y dispone de licencia para ser utilizada en cualquier tipo de plataforma, personal o comercial. El fichero tiene un tamaño aproximado de 31 KB, y su carga es realmente rápida. Además, una vez cargada la librería, quedará almacenada en caché del navegador, con lo que el resto de páginas que hagan uso de la librería, no necesitarán cargarla de nuevo desde el servidor.

 [Página Oficial de descarga de la librería jQuery.](#)

Para saber más

 [Documentación oficial de la librería jQuery.](#)

Para poder programar con jQuery, lo primero que tenemos que hacer es cargar la librería. Para ello, podemos hacerlo de dos formas:

Cargando la librería directamente desde la propia web de jQuery con la siguiente instrucción:

```
<script src="HTTP://code.jquery.com/jquery-latest.js"></script>
```

De esta forma, siempre nos estaremos descargando la versión más actualizada de la librería. El único inconveniente, es que necesitamos estar conectados a Internet para que la librería pueda descargarse.

Cargando la librería desde nuestro propio servidor:

```
<script src="jquery.js"></script>
```

De esta forma, el fichero de la librería estará almacenado como un fichero más de nuestra aplicación, por lo que no necesitaremos tener conexión a Internet (si trabajamos localmente), para poder usar la librería. Para poder usar este método, necesitaremos descargarnos el fichero de la librería desde la página de jQuery (jquery.com). Disponemos de dos versiones de descarga: la *versión de producción* (comprimida para ocupar menos tamaño), y la *versión de desarrollo* (descomprimida). Generalmente descargaremos la versión de producción, ya que es la que menos tamaño ocupa. La versión de desarrollo tiene como única ventaja que nos permitirá leer, con más claridad, el código fuente de la librería (si es que estamos interesados en modificar algo de la misma).

La clave principal para el uso de jQuery radica en el uso de la función `$()`, que es un alias de `jQuery()`. Esta función se podría comparar con el clásico `document.getElementById()`, pero con una diferencia muy importante, ya que soporta selectores CSS, y puede devolver arrays. Por lo tanto `$()` es una versión mejorada de `document.getElementById()`.

Debes conocer

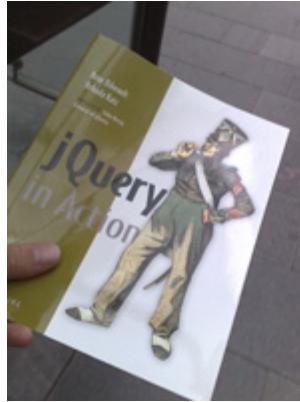
Para usar toda la potencia de jQuery necesitas conocer los selectores CSS para poder mejorar la selección sobre las etiquetas a aplicarle actuaciones.

 [Selectores CSS.](#)

Esta función `$(“selector”)`, acepta como parámetro una cadena de texto, que será un selector CSS, pero también puede aceptar un segundo parámetro, que será el contexto en el cuál se va a hacer la búsqueda del selector citado. Otro uso de la función, puede ser el de `$(function){..})`; equivalente a la instrucción `$(document).ready (function() {...})`; que nos permitirá detectar cuando el DOM está completamente cargado.

Vamos a ver un ejemplo de cómo podemos usar esta función para detectar cuando el DOM

3.2.- Introducción a jQuery (parte II).



[Lachlan Hardy](#) picked up a copy
of [\(CC BY\)](#)

Vamos a ver en este apartado, un ejemplo programado por el método tradicional, y su equivalencia, usando la librería jQuery:

 [Descarga el código fuente del ejemplo dwec0732.](#) (0.01 MB)

Ejemplo usando el método tradicional con JavaScript:

```

... Aquí irán las cabeceras y clase .colorido

<script type="text/javascript" src="funciones.js"></script>
<script type="text/javascript">
///////////////////////////////
// Cuando el documento esté cargado completamente llamamos a la función iniciar().
/////////////////////////////
crearEvento(window,"load",iniciar);
/////////////////////////////

function iniciar()
{
    var tabla=document.getElementById("mitabla"); // Seleccionamos la tabla.
    var filas= tabla.getElementsByTagName("tr"); // Seleccionamos las filas de la tabla.

    for (var i=0; i<filas.length; i++)
    {
        if (i%2==1)      // Es una fila impar
        {
            // Aplicamos la clase .colorido a esas filas.
            filas[i].setAttribute('class','colorido');
        }
    }
}
</script>
</head>
<body>
    .. Aquí irá la tabla ...
</body>
</HTML>

```

Ejemplo equivalente al anterior, programado usando la librería jQuery:

... Aquí irán las cabeceras y clase .colorido

```
<script type="text/javascript" src="HTTP://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
// También podríamos poner $(function() {...})
$(document).ready(function()      // Cuando el documento esté preparado se ejecuta esta función
{
    // Seleccionamos las filas impares contenidas dentro de mitabla y le aplicamos la clase
    $("#mitabla tr:nth-child(even)").addClass("colorido");
});
</script>
</head>
<body>
    .. Aquí irá la tabla ...
</body>
</HTML>
```

Como puedes observar, la reducción de código es considerable. Con 2 instrucciones, hemos conseguido lo mismo, que hicimos en el ejemplo anterior con 7 (sin contar el código de crearEvento del fichero funciones.js).

3.3.- Introducción a jQuery (parte III).



[simplu27 - https://pixabay.com/es/c%C3%B3digo-jquery-dise%C3%B1o-web-583073/ \(CC0\)](https://pixabay.com/es/c%C3%B3digo-jquery-dise%C3%B1o-web-583073/)

En la unidad anterior se aprendió como gestiona JavaScript y w3c los eventos y vimos una serie de problemas con los eventos y la compatibilidad entre navegadores. jQuery surgió entre otras cosas para ayudarnos a evitar estos problemas.

Veamos por ejemplo como se haría una asignación de una función a un evento click en un botón:

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Básico de JQuery</title>
    <script type="text/javascript" src="HTTP://code.jquery.com/jquery-latest.js"></script>
</head>

<body>
    <h1> Botón Fácil</h1>
    <button>Mi botón</button>
    <script lang="text/javascript">
        $("button").click(function() {
            $("body").append("<br />Se ha pulsado el botón");
        });
    </script>
</body>
</html>
```

Si os fijáis en el método click se le pasa un parámetro que es una función anónima. Es algo común en muchos códigos, pero no recomendable en muchas circunstancias. Es mejor que creéis una función y la llamamos de esta forma:

```

<script>
    function pulsado() {
        $("body").append("<br />Se ha pulsado el botón");
    };
    $("button").click(pulsado);
</script>

```

Para una gran mayoría de eventos w3c, jQuery dispone de su método para el evento. Por ejemplo como `mouseenter`, `mouseleave`, `resize`, `focus`, `scroll` y un largo etcétera. Se pueden consultar en uno de los enlaces que se encuentra más adelante.

Sin embargo hay uno que es muy interesante denominado ready. Este evento equivalente al [on]load que hemos estado utilizando hasta ahora en el curso para asegurarnos que el DOM está completamente cargado. Por ejemplo, si introducimos el anterior código en el head (o en otro archivo externo) para asignar el evento click a la función pulsado() hay que realizar la asignación del evento dentro de un ready tal y como veremos en el siguiente ejemplo.

```

<!doctype html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Básico de JQuery</title>
    <script type="text/javascript" src="HTTP://code.jquery.com/jquery-latest.js"></script>
    <script lang="text/javascript">
        function pulsado() {
            $("body").append("<br />Se ha pulsado el botón");
        };
        // La llamada al siguiente método es equivalente al onload y al load
        $(document).ready(function() {
            // Aquí dentro se ejecuta lo que se quiera que se haga
            // al cargar el DOM.
            $("button").click(pulsado);
        });
    </script>
</head>

<body>
    <h1> Alguna cosas </h1>
    <button>Mi botón</button>
</body>
</html>

```

La llamada a ready se puede substituir por la siguiente:

```

$(function() { // Sustituye al ready()
    $("button").click(pulsado);
});

```


3.4.- Introducción a jQuery (parte IV).

Para terminar esta introducción vamos a ver como se gestionan los eventos de una forma más moderna con el uso del método on.

El método on te permite trabajar con los eventos w3c de forma similar a lo que se hace con addEventListener. Veamos el ejemplo anterior actualizado:

```
<script>
    function pulsado() {
        $("body").append("<br />Se ha pulsado el botón");
    };
    $(function() {
        $("button").on("click",pulsado);
    });
</script>
```

Es una forma más elegante y creo que comprensible que en los métodos visto en el apartado 3.3.



Pexels - <https://pixabay.com/es/primer-plano-código-codificación-2178341/> (CC0)

Otra ventaja es la posibilidad de añadir distintos eventos y elementos para ser procesados por la misma función. Estos argumentos deben estar dentro de un array literal tal y como se ve en el siguiente ejemplo:

```

<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Gestión de Eventos con JQuery</title>
    <script type="text/javascript" src="HTTP://code.jquery.com/jquery-latest.js"></script>
    <script>
        $(function() {
            console.log("hace algo");
            $("#nombre,#apellidos").on({ //on ha sustituido a bind a partir del jquery 1.7
                "focus": miGestorEventos,
                "blur": miGestorEventos
            });
        });

        function miGestorEventos(evento) {
            console.log(evento);
            if (evento.type == "focus") { // Se ha producido el evento focus
                $(this).css("backgroundColor", "blue");
            } else { // Si se ha producido el evento blur
                $(this).css("backgroundColor", "white");
            }
        }
    </script>
</head>
<body>
    <form>
        <label for="nombre">Nombre</label>
        <input type="text" id="nombre" name="entrada">
        <br />
        <label for="apellidos">Apellidos</label>
        <input type="text" id="apellidos" name="entrada">
    </form>
    <br />
</body>
</html>

```

Con esto sólo se ha visto un poco de jQuery. Hay mucho más que aprender ya que es una librería excepcionalmente útil. Os recomiendo encarecidamente que os leáis los enlaces del siguiente para que jQuery os facilite la vida tanto como a mí.

Para saber más

 [Eventos en jQuery](#)

 [El método on](#)

 [Cambiar el texto de elementos](#)

 [Cambiar el html de elementos](#)

3.5.- Función \$.ajax() en jQuery.

La principal función para realizar peticiones AJAX en jQuery es \$.AJAX() (importante no olvidar el punto entre \$ y AJAX()). Ésta es una función de bajo nivel, lo que quiere decir que disponemos de la posibilidad de configurar, prácticamente todos los parámetros de la petición AJAX, y será, por tanto, equivalente a los métodos clásicos que usamos en la programación tradicional.

La sintaxis de uso es: \$.AJAX(opciones)

En principio, esta instrucción parece muy simple, pero el número de opciones disponibles, es relativamente extenso. Ésta es la estructura básica:

```
$.AJAX({  
    url: [URL],  
    type: [GET/POST],  
    success: [function callback exito(data)],  
    error: [function callback error],  
    complete: [function callback error],  
    ifModified: [bool comprobar E-Tag],  
    data: [mapa datos GET/POST],  
    async: [bool que indica sincronía/asincronia]  
});
```



[daveynin. Reading AJAX For Dummies \(CC BY\)](#)

Por ejemplo:

```
$.AJAX({  
    url: '/ruta/pagina.php',  
    type: 'POST',  
    async: true,  
    data: 'parametro1=valor1&parametro2=valor2',  
    success: function (respuesta)  
    {  
        alert(respuesta);  
    },  
    error: mostrarError  
});
```

Veamos algunas propiedades de la función `$.AJAX()` de jQuery:

Propiedades de la función `$.ajax()` de jQuery.

Nombre	Tipo	Descripción
url	String	La URL a la que se le hace la petición AJAX.
type	String	El método HTTP a utilizar para enviar los datos: POST o GET. Si se omite se usa GET por defecto.
data	Object	Un objeto en el que se especifican parámetros que se enviarán en la solicitud. Si es de tipo GET, los parámetros irán en la URL. Si es POST, los datos se enviarán en las cabeceras. Es muy útil usar la función <code>serialize()</code> , para construir la cadena de datos.
dataType	String	Indica el tipo de datos que se espera que se devuelvan en la respuesta: XML, HTML, JSON, JSONp, script, text (valor por defecto en el caso de omitir dataType).
success	Function	Función que será llamada, si la respuesta a la solicitud terminó con éxito.
error	Function	Función que será llamada, si la respuesta a la solicitud devolvió algún tipo de error.
complete	Function	Función que será llamada, cuando la solicitud fue completada.

Por último veamos un ejemplo en el que se utilice JSON. Para acceder a un archivo (en este caso ya está creado en el servidor), de películas de cine con para que podáis ver como funciona:

Contenido de ajaxpelis.json

```
[  
  { "titulo":"Terminator","genero":"Ciencia Ficción"},  
  { "titulo":"Alien","genero":"Terror"},  
  { "titulo":"La Salchica Peleona","genero":"comedia"}  
]
```

Archivo html:

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Jquery método ajax con JSON.</title>
    <script src="jquery-1.11.1.js"></script>

    <script>
        $(function () {
            $("button").click(function () { // Se se pulsa el boton
                var cadena = "";
                $("#cambia").html("<img src='ajax-loader.gif'>"); // Mientras se carga
                // El método ajax recibe un objeto literal
                $.ajax({
                    url: "ajaxpelis.json",
                    // data: { edadrecomendad: 16}, // El post al servidor No hace falta par
                    type: "GET",
                    dataType: "json",
                    // Si se produce correctamente
                    success: function (datos_devueltos) {
                        $("#cambia").html("<br />");
                        for (var i = 0; i < datos_devueltos.length; i++) {
                            $("#cambia").append("Título:" + datos_devueltos[i].titulo + "<br
}
                },
                // Si la petición falla
                error: function (xhr, estado, error_producido) {
                    console.log("Error producido: " + error_producido);
                    console.log("Estado: " + estado);

                },
                //Tanto si falla como si funciona como sino funciona.
                complete: function (xhr, estado) {
                    console.log("Petición completa");
                }
            });

            });
        });
    </script>
</head>
<body>
    <button type="button">JQuery Ajax JSON.</button>
    <div id="cambia">
        <h2>Cambia el contenido</h2>
    </div>
</body>
</html>

```

Si no os funciona, aseguraros que estáis el archivo html se llama desde el mismo dominio de el que se llama el html, sino os va a dar errores de permisos. Y buscar una imagen AJAX de carga gif para que los usuarios sepan que se está procesando una petición AJAX.

Para saber más



[Todos los parámetros de uso de la función \\$.AJAX\(\) de jQuery.](#)

3.6.- El método .load() y las funciones \$.post() , \$.get() y \$.getJSON() en jQuery.

La función \$.AJAX() es una función muy completa, y resulta bastante pesada de usar. Su uso es recomendable, para casos muy concretos, en los que tengamos que llevar un control exhaustivo de la petición AJAX. Para facilitarnos el trabajo, se crearon 3 funciones adicionales de alto nivel, que permiten realizar peticiones y gestionar las respuestas obtenidas del servidor:

El método .load()

Este método, es la forma más sencilla de obtener datos desde el servidor, ya que de forma predeterminada, los datos obtenidos son cargados en el objeto al cuál le estamos aplicando el método.

Su sintaxis es: `.load(url, [datos], [callback])`

La función callback es opcional, y es ahí donde pondremos la función de retorno, que será llamada una vez terminada la petición. En esa función realizaremos tareas adicionales, ya que la acción por defecto de cargar en un objeto el contenido devuelto en la petición, la realiza el propio método *load()*.

Ejemplos:

```
$("#noticias").load("feeds.HTML");
// carga en el contenedor con id noticias lo que devuelve la página feeds.HTML.

$("#objectID").load("test.php", { 'personas[]': ["Juan", "Susana"] } );
// Pasa un array de datos al servidor con el nombre de dos personas.
```

Cuando se envían datos en este método, se usará el método POST. Si no se envían datos en la petición, se usará el método GET.

Debes conocer



[Más información sobre el método .load\(\) en jQuery.](#)

La función `$.post()`

Nos permite realizar peticiones AJAX al servidor, empleando el método POST. Su sintaxis es la siguiente:

```
$.post( url, [datos], [callback], [tipo] )
```

Ejemplos:

```
$.post("test.php");
```

```
$.post("test.php", { nombre: "Juana", hora: "11am" } );
```

```
$.post("test.php", function(resultados) {
    alert("Datos Cargados: " + resultados);
});
```



Tim Green. Post Office Letter Box (CC BY)

Debes conocer



[Más información sobre el método `.post\(\)` en jQuery.](#)

La función `$.get()` y `$.getJSON()`

Hacen prácticamente lo mismo que POST, y tienen los mismos parámetros, pero usan el método GET para enviar los datos al servidor. Si recibimos los datos en formato JSON, podemos emplear `$.getJSON()` en su lugar.

```
$.get( url, [datos], [callback], [tipo] ) | $.getJSON( url, [datos], [callback] )
```

Debes conocer



[Más información sobre el método `.get\(\)` en jQuery.](#)

3.7.- Herramientas adicionales en programación AJAX.

Cuando programamos en AJAX, uno de los inconvenientes que nos solemos encontrar, es el de la detección de errores. Estos errores pueden venir provocados por fallos de programación en JavaScript, fallos en la aplicación que se ejecuta en el servidor, etc.

Para poder detectar estos errores, necesitamos herramientas que nos ayuden a encontrarlos. En la programación con JavaScript, los errores los podemos detectar con el propio navegador. Por ejemplo, en el navegador Firefox para abrir la consola de errores, lo podemos hacer desde el menú Herramientas, o bien pulsando las teclas **CTRL+MayúsC.+J** (en Windows) o F12 si se dispone de una versión de firebug moderna. En Chrome también se utiliza con F12. En la consola, se nos mostrarán todos los errores que se ha encontrado durante la ejecución de la aplicación. En Internet Explorer versión 9 y posteriores, podemos abrir la **Herramienta de Desarrollo**, pulsando la tecla F12. Desde esta herramienta se pueden consultar los errores de JavaScript, activar los diferentes modos de compatibilidad entre versiones de este navegador, deshabilitar CSS, JavaScript, etc.

Para la detección de errores en AJAX, necesitamos herramientas adicionales o complementos. En Firefox disponemos de un complemento propio. Este complemento nos va a permitir hacer infinidad de cosas: detectar errores de JavaScript, depurar código, analizar todo el DOM del documento en detalle, ver y modificar el código CSS, analizar la velocidad de carga de las páginas, etc. Además, también incorpora en su consola, la posibilidad de ver las peticiones AJAX que se están realizando al servidor: se pueden ver los datos enviados, su formato, los datos recibidos, los errores, etc. Si por ejemplo se produce algún tipo de error en la petición al servidor, en la consola podremos verlo y así poder solucionar ese fallo.



[Resumen textual alternativo](#)

Para saber más



- [Tres extensiones para el desarrollo web para Firefox y Chrome.](#)
- [Las mejores 10 extensiones para el desarrollo Web en Chrome.](#)
- [Las 10 mejores extensiones para el desarrollo web con Firefox.](#)

Para saber más. Otras librerías AJAX fuera de jQuery y fetch API.

Existen otras librerías que permiten realizar conexiones AJAX con librerías de forma sencilla.

Además de jQuery que es muy sencilla y se sigue usando para conexiones AJAX, tenemos axios (que se instala en entorno node que se toca en el punto 4.1 o como aparece en el enlace siguiente).



[Axios client-side](#)

También podéis usar superAgent



[Documentación superAgent](#)

3.8.- Plugins jQuery.

La librería jQuery, incorpora funciones que nos van a ayudar muchísimo, en la programación de nuestras aplicaciones. Además de todo lo que nos aporta la librería, disponemos de plugins o añadidos que aportan funcionalidades avanzadas.

Vamos a encontrar plugins en un montón de categorías: AJAX, animación y efectos, DOM, eventos, formularios, integración, media, navegación, tablas, utilidades, etc.

Para saber más

 [Efectos con jQuery.](#)

 [Documentación oficial de jQuery.](#)

Antes de poder usar cualquier plugin de jQuery, será necesario cargar primero la librería de jQuery, y a continuación la librería del plugin que deseemos, por ejemplo:

```
<script src="http://code.jquery.com/jquery-latest.js"></script>
```

```
<script src="ejemploplugin.js"></script>
```

Todos los plugins contienen documentación, en la que se explica como usar el plugin.

Desde la web oficial de  [jQuery](#), puedes hojear todos los plugins disponibles para jQuery:



[Sean MacEntee. wp plugins \(CC BY\)](#)



También es muy común el encontrar páginas, en las que se muestran rankings de los mejores plugins para jQuery. Algunos ejemplos pueden ser:



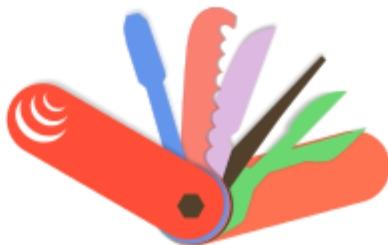
3.9.- Ejemplos en vídeo, de AJAX con jQuery.

Y para terminar, te vamos a poner unos enlaces a unos vídeos hospedados en Youtube.com

 [Descarga el código fuente JavaScript de todos los videos: dwec0737.](#) (0.21 MB)

[Resumen textual alternativo](#)

[Resumen textual alternativo](#)



[Ulrike. Swiss army knife \(CC0\)](#)

Para saber más

[Resumen textual alternativo](#)

[Resumen textual alternativo](#)

[Resumen textual alternativo](#)

4.- Frameworks JavaScript.

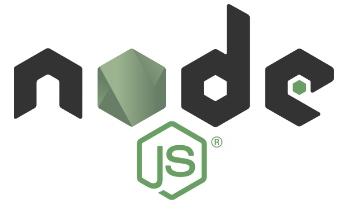
Caso práctico



Antonio y Juan están muy cerca de finalizar el curso, pero hay algo que les llama mucho la atención. Node. Todo el mundo habla de node y usa una cosa llamada npm para realizar el desarrollo de aplicaciones web. Están dispuestos a solucionarlo. Y para ello van a instalar node, bower y babeljs. Van a ponerse en la cresta de la ola tecnológica en JavaScript.

Elaboración Propia. ([CC BY-NC](#))

En este curso hemos usado JavaScript en el lado del servidor. Sin embargo en estos últimos años JavaScript se está utilizando bastante a través del framework Nodejs. Aunque no es objeto de estudio en módulo orientado al lado cliente si que lo es las herramientas y el ecosistema que se utiliza ya que la mayoría de las librerías y frameworks del mundo orientados a JavaScript están ahí.



[node.js authors. Node.js logo](#)

Así pues Node.js es además de una librería, un entorno de ejecución de E/S que se ejecuta sobre el intérprete de JavaScript creado por Google llamado V8. Y aunque es un framework de servidor no es Apache Tomcat o PHP. Funciona de una forma dirigida a eventos.

A modo de ejemplo hola mundo con el que se empieza a aprender nodejs. Se puede comprobar que el código es similar aunque es distinto.

```
// Carga el módulo http
var http = require('http');

// Configura nuestro servidor HTTP para responder con Hola Mundo a todas las peticiones.
var server = http.createServer(function (request, response) {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.end("Hola Mundo \n"); // Cada navegador que se conecte a la web verá ese mensaje.
});

// Escucha en el puerto 8000, IP pasa por defecto a 127.0.0.1
server.listen(8000);

// Muestra un mensaje en la consola
console.log("Server running at http://127.0.0.1:8000/");
```

Ya solo faltaría abrir un navegador con el siguiente comando.

```
node holamundo.js
```

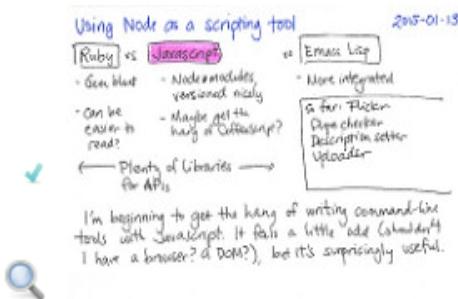
Por supuesto hay que instalar antes node, algo que veremos en el siguiente punto.

Para saber más

Hay un curso gratuito en español para aprender nodejs.

 [Curso gratuito de Node.](#)

4.1.- Instalación de Node.



Sacha Chua [Using Node as a scripting tool](#) (CC BY)

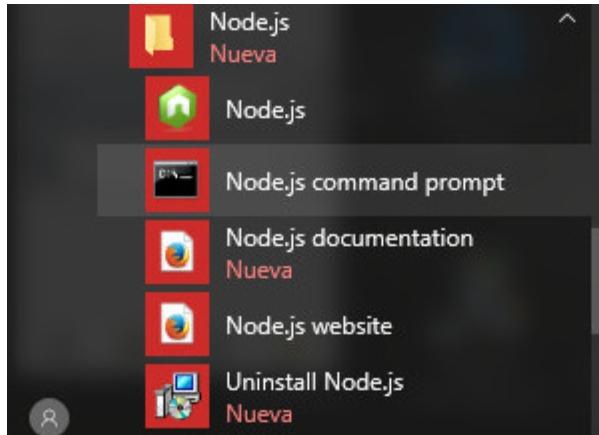
y la versión LTS es la que aunque no tiene lo último de lo último es versión más estable y que va a tener soporte durante más tiempo.

En opinión del redactor de este documento es mejor usar siempre las versiones LTS salvo que vayan a dejar de tener soporte en breve (un par de meses). Cuando una versión LTS deja de tener soporte siempre hacen una nueva versión que normalmente se parece más a la última versión actual que a la última LTS.

Si usas una versión Linux tienes que saber que normalmente no se tiene la última versión estable en el repositorio. Pues para asegurarse de que la dispones puedes seguir las instrucciones indicadas en el siguiente enlace:

✓ [Gestor de paquetes Node.](#)

La instalación por defecto funciona correctamente. Una vez lo tienes instalado puedes proceder al uso yendo al menú de inicio.



Elaboración Propia.

Para saber más

En Windows 10 puedes instalar Ubuntu shell que te permitirá trabajar con las mismas ordenes que utilizarías en MacOS o Linux. En el primer enlace te explica como configurar Windows 10 y en el segundo como instalar la versión 6 LTS.



[Instalar Ubuntu Bash en Windows 10 sin tener que instalar Linux.](#)

4.2.- Gestión de paquetes con npm. Bower.

Node tiene un gestor de paquetes llamado **npm** y es el software que permite instalar paquetes con software para desarrollar con Javascript tanto en backend como en frontend.

Funciona en modo texto como un instalador de paquetes de Linux. Tanto en la forma similar a apt-get de Ubuntu, como en su capacidad de gestionar las dependencias.

¿Qué son las dependencias? Pensad que para realizar un programa puedes necesitar una serie de librerías. Pues para poder utilizar una librería puedes necesitar esas librerías bases. **npm** se encarga de descargar esos otros frameworks de los que depende el que estamos utilizando.

La web que contiene los paquetes con descripciones de los mismos y como instalarlos es la siguiente.

✓  [La web de npmpjs.](#)

Para la instalación podemos usar:

```
npm install -g paquete
```

El parámetro **-g** nos va a servir para que se instalarlo en el path del usuario.

Si estás en Linux y se te produce un mensaje de error ERR! deberías ejecutar el comando así:

```
sudo npm install -g paquete
```

Para la desinstalación podemos :

```
npm uninstall paquete
```

```
npm ls
```

muestra el contenido de paquetes instalados en tu sistema.

Para actualizar a la última versión de npm

```
npm install -g npm@latest
```



Antranias. Cenador. (CC0)

Algunos paquetes npm pueden actualizarse a su última versión usando @latest al final.

Algunos conoceréis bower, es una especie de npm pero para el frontend solamente. Lo vamos a utilizar como ejemplo para el aprendizaje.

Para instalar bower:

```
npm install -g bower
```

Una vez instalado bower podemos instalar jquery directamente con el siguiente comando.

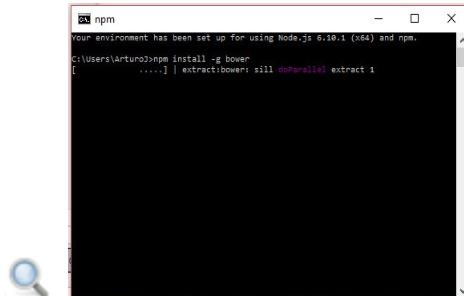
```
bower install jquery
```

Y se habrá descargado bower_components/jquery/dist/ si estás en Linux y MacOS y cambiando las barras se encontraría en Windows.

Ya podrías utilizarlo si tienes el archivo incluido en tu html así:

```
<script src="bower_components/jquery/dist/jquery.min.js"></script>
```

Otra cosa importante es que el directorio de trabajo es en Linux el **home** del usuario y en Windows está en el directorio **users**. Alguna vez puede haber problemas que se pueden solucionar yéndose al directorio donde se abre la shell o el cmd.



Elaboración Propia.

Para saber más

Puedes aprender bower de una forma muy sencilla en su web.

 [Tutorial de uso de Bower.](#)

 [Pagina de Bower.](#)

4.3.- Babeljs.



Pieter Brueghel el Viejo. *La Torre de Babel*. (Dominio público)

¿Qué es Babeljs? Es un compilador de JavaScript a JavaScript, con la gran ventaja de que te permite utilizar las últimas características del lenguaje que aún no se han implementado en el navegador

Para instalar babel puedes seguir las siguientes instrucciones:

```
npm install --save-dev babel-cli babel-preset-env
```

Babel tiene una serie de plugins. Para añadirlos se necesita saber el nombre:

✓ [Plugins para Babel](#)

Si queremos instalar el soporte completo del EcmaScript 6 que en el momento de escribir esta documentación era bastante pobre en los navegadores, en 2017. Apenas algunas cosas sencillas sobre clases, pero que no funcionarán nunca en los navegadores de los móviles ni en Internet Explorer.

```
npm install --save-dev babel-plugin-transform-es2015-classes
```

Veamos un código autoexplicativo sobre como funciona las clases.

```

// Archivo mascotas.js
// Las clases de JavaScript se parecen bastante más a las de Java y son mucho más sencillas
class Mascota {
    constructor(nombre) {
        this._nombre = nombre;
    }
    get nombre() {
        return (this._nombre);
    }
    set nombre(nuevoNombre) {
        this._nombre=nuevoNombre;
    }
    toString() {
        return '(' + this._nombre + ')';
    }
}

class Gato extends Mascota {
    constructor(nombre,nuevocolor) {
        super(nombre); // Se llama al constructor del parent.
        this._color = nuevocolor; // Fijate que es _color.
    }
    set color(nuevoColor) {
        this._color=nuevoColor;
    }
    get color() {
        return (this._color);
    }
    toString() {
        return super.toString() + ' es un gato de color' + this._color;
    }
}

let ungato = new Gato('Pamplinas', "negro"); // Creamos ungato como variable local al usar
// Veamos como funciona los setters y getters nuevos.
// Los setters y los getters son ya sistemas en los que os aseguramos que nadie acceda
// directamente al valor de la variable, haciéndolo a más segura.
ungato.color="Gris"; // Lo que hemos hecho aquí es llamar al set de forma transparente.
                    // Fíjate que aunque hemos definido _color para almacenar el color
                    // estamos modificándolo con color.
console.log(ungato.nombre); // Estamos llamando al get de forma indirecta.
document.write(ungato);

```

Como ya sabrás los setters y getters son distintos. Fíjate en el código que usa las propiedades de la clase. Ninguno se llama igual que setter o getter. `_color` almacena el valor pero se accede con `color`. Un consejo es que a partir de ahora los métodos deben realizarse de esa forma y ejecutar un código `this.color=nuevocolor` dará el fallo siguiente *InternalError: too much recursion* en Firefox o *Uncaught RangeError: Maximum call stack size exceeded* en Chrome. Si lo ves cambia los nombres a las variables.

Una vez se tiene este código se tiene que incluir dentro de un html que cargue el preprocesador de babel.

```
<!-- Se incluye el compilador que convertirá los scripts que tengan type="text/babel" -->
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
<script type="text/babel" src="mascotas.js">
// A parte de meter el código anterior directamente en vez de introducirlo en otro archivo si te hace falta.
</script>
```

Sin embargo la solución anterior no es la mejor ya que la conversión es lenta ya que cada vez que se carga . Para ello se va a compilar a JavaScript antiguo y no se tendrá recompilar.

```
babel --plugins babel-plugin-transform-es2015-classes mascotas.js -d directorio_salida
```

Es muy importante que se pongan los dos guiones antes de plugin ya que sino no va a funcionar y puede que no sepas el porqué. Otro error es no tener en tu carpeta el archivo que quieras comentar. Si usas rutas puedes tener problemas.

En el directorio_salida habrá un archivo llamado `clases.js` con algunas similitudes, sobretodo en el uso de la clase. Ya no hace falta incluir el babel ni usar `text/babel` en el tipo de script. Por eso cuando vayamos a usar el código convertido sería el momento de incluirlo de una forma similar a la siguiente.

```
<script src="mascotas.js">
// Además de meter el código anterior directamente en vez de introducirlo en otro archivo si te hace falta.
</script>
```

Para terminar necesitas estar atentos a los cambios anuales de JavaScript y de los nombres de los plugins que necesitas para poder desarrollar con las propiedades nuevas (y por desgracia no tan nuevas) que no se han implementado. Sobretodo en los navegadores antiguos y de Microsoft.

Para saber más

También tienes la opción de convertir en el navegador web en este enlace.



[Babel en la web. Editor y convertidor sin línea de comandos.](#)

4.4.- Nuevas tendencias.

En la actualidad JavaScript está en una completa evolución para poder sobrevivir a los tiempos actuales. Un ejemplo es que al final se pueden utilizar clases mucho más completas. Sin embargo para proyectos grandes esto puede ser insuficiente. Para ello Microsoft, en colaboración con Google, ha desarrollado un lenguaje similar a JavaScript. Sin entrar en mucho detalle las principales diferencias entre JavaScript y Typscript son:

- Typscript no se soporta directamente por los navegadores. Hay que transformarlo en Javascript.
- Es un lenguaje tipado. O sea que hay tipos declarados desde el principio.
- Los métodos deben especificar el tipo de datos que devuelven.
- Dispone de Interfaces al estilo de Java o C#. Se puede utilizar implements.
- Genéricos similares a C# o Java o plantillas en C++.



Ryan Dahl. [Ryan Dahl \(CC BY\)](#)

Al igual que babel, **tsc** convierte el código desde el lenguaje origen a JavaScript.

✓ [Hogar del desarrollo en TypeScript](#)

Sin querer entrar en muchos detalles cuando instalas con npm typescript instalas un servidor y un compilador que transforma el código TypeScript a código JavaScript para que puedas utilizarlo en tu desarrollo web: tsc.

```
tsc archivo.ts archivo.js
```

Puedes seguir un esquema similar a babel para añadir tu código a tus proyectos, usarlo en Visual Studio, brackets, atom, o cualquier otro editor actualizado que se encargue de traducirlo todo, o utilizar otras herramientas incluidas.

Cordova, con **v**, es una tecnología que te va a permitir desarrollar aplicaciones móviles partiendo de una aplicación web. Sabiendo algunos eventos más necesarios (como deviceready) puedes crear muy rápidamente aplicaciones para móvil tanto en android como para mac.

✓ [Desarrollo móvil con Cordova](#)

Una aplicación cordova permite usar HTML y JavaScript con muy poco cambio en el código.

Cordova permite utilizar casi cualquier otro framework con el desarrollo que hagas. Por ejemplo puedes usar jQuery dentro del código.

Si no te gusta porque los iconos y temas no están adecuados a los móviles puedes usar [Ionic](#). Por debajo tendrás Angular o React.

Angular. Angular es el Framework de todo. Es algo inestable ya que suelen cambiarlo a menudo. Te permite desarrollar tanto con Javascript como con Typescript, pero es mejor hacerlo con la segunda opción. Para desarrollar en Angular hay que tener node y npm.

- ✓  [Página oficial de Angular](#)
 - ✓  [Guía de instalación de Angular](#)
-

React. Es el framework de Facebook. Reactivo como Angular.

- ✓  [Página oficial de react](#)

Dentro de la web puedes encontrar cursos introductorios. Como algo malo, no está tan bien estructurado y ajustado a patrones de diseño con Angular, pero por eso mismo es más sencillo aprenderlo de primeras.

vue.js.

Mas sencillo que los anteriores y te puede servir como introducción a muchos de los conceptos de Angular o React, pero presentados de forma mucho más sencilla.

- ✓  [Web de vue.js](#)
-

knockout es tu alternativa en el aspecto de diseño de interfaces a Angular y los demás, además de que es muy sencilla, y se puede instalar desde bower.

- ✓  [Página oficial de knockout](#)

Otras tecnologías que utilizan Javascript en la actualidad y son importantes son Reactjs e Ionic, etc. Ambas han inspirado o utilizado los llamados componentes web, stencyl.

- ✓  [Pagina oficial de stencyl.](#)

Ionic es muy interesante para desarrollar aplicaciones web, pero es más difícil de primeras que Cordova. Pero si aprendes Angular y aprendes a crear componentes web (o al menos a utilizarlos) puedes desarrollar aplicaciones muy interesantes.

Dicho esto, esto el curso se acaba. Estudia cualquier otra cosa en especial cualquiera de las nuevas que hemos visto y no te quedes parado ya que JavaScript y los lenguajes similares están en alza. En el punto 5 verás incluso más enlaces útiles.

Para saber más

Tutorial TypeScript:

-  [Tutorial en castellano de TypeScript.](#)

5.- Enlaces de refuerzo y ampliación.

JavaScript está siempre en constante desarrollo evolucionando. Y es imposible tratarlo todo e incluso a veces nombrarlo. Pero hay muchas. Aquí algunas.

- ✓ Desarrollo Android con Ionic.  [Comenzando a programar en Ionic/](#)
- ✓ Desarrollo de componentes web en castellano. [Crear web components nunca fue tan fácil](#)
- ✓ Librería de desarrollo de interfaces en JavaScript React.  [React](#)
- ✓ Desarrollo de JavaScript de aplicaciones de escritorio: [Tu primera aplicación con Electron](#)
- ✓ Desarrollo de aplicaciones de IU DOJO.  [Toolkit Dojo](#)
- ✓ Desarrollo de aplicaciones móviles con Meteor. [Meteor](#)
- ✓ Desarrollo de aplaciones con vue.js.  [Vue.js](#)
- ✓ Tanto JavaScript como TypeScript han añadido Lambdas (=>). Se utilizan cada vez más y lo mismo necesitarías aprender.
 - ◆ En JavaScript, si no funcionan en tu navegador, recuerda usar Babeljs. [Tutorial sobre Lambdas.](#)
 - ◆ En TypeScript  [Lambdas en TypeScript.](#)
- ✓ Después tenemos que hay muchos cursos masivos de JavaScript, TypeScript, Node y más tecnologías que además os pueden servir hasta para enlazarlos a vuestro currículum.
 - ◆  [TypeScript en EDX\(MIT/Harvard\) gratuito por parte de Microsoft.](#)
 - ◆  [Curso de Node de Microsoft en EDX.](#)
 - ◆  [Curso de Angular por parte de Microsoft en EDX.](#)
 - ◆ En  [udemy](#) se pueden encontrar cursos de pago, en mi opinion mejores que los anteriores, pero hay que estar atentos a que bajen de precio al 90%. En
 - ◆ Curso de  [Vue.js 2](#) en español. Gratuito pero puedes patrocinar a su creador.
 - ◆ Fundamentos de  [React](#) en castellano.

Y para finalizar un curso en Youtube para entender vue.js.

Condiciones y términos de uso de los materiales

Materiales desarrollados inicialmente por el Ministerio de Educación, Cultura y Deporte y actualizados por el profesorado de la Junta de Andalucía bajo licencia Creative Commons BY-NC-SA.



MINISTERIO
DE EDUCACIÓN
Y FORMACIÓN PROFESIONAL



Antes de cualquier uso leer detenidamente el siguiente [Aviso legal](#)

Historial de actualizaciones

Versión: 02.00.08	Fecha de actualización: 25/03/23	
Actualización de materiales y correcciones menores.		
Versión: 02.00.00	Fecha de actualización: 31/03/17	Autoría: Arturo José Puentes Castellanos
<p>Ubicación: A lo largo del documento.</p> <p>Mejora (tipo 1): Cambios en el material</p> <ul style="list-style-type: none">- Se han puesto todas las imágenes al formato nuevo de autoría.- Se han añadido imágenes nuevas a algún apartado de 3.x que no disponía.- En el punto 1.1 se ha actualizado el párrafo referido a Firebug.-En el punto 1.2 se han añadido dos párrafos nuevos intentando explicar ejemplos de la vida real que usan AJAX para que los alumnos puedan entender más como funciona.- En el punto 1.3.2, 1.3.3, 2.1, 2,2,2,5, etc... añadido un código ejemplo distinto.- En el 2.3 se habla de los gifs animados que se incluyen en la web para indicar que se está haciendo una operación con AJAX.- En los códigos que he visto con un alert he incluido un comentario indicando que no es bueno que se utilice con usuarios finales.-Se ha añadido el punto 2.8 para que los alumnos sepan como se llaman los API REST que se utilizan en la actualidad que utilizan JSON.- Punto nuevo 4. Frameworks JavaScript.- Se ha añadido el punto 5. de Enlace de refuerzo y ampliación. <p>Ubicación: Primera mitad.</p> <p>Mejora (tipo 3): Rehacer la parte primera de AJAX para que sea más sencilla y más detallada y didáctica.</p>		
Versión: 01.01.00	Fecha de actualización: 31/03/15	Autoría: Arturo José Puentes Castellanos
<p>Ubicación: 3.2 y 3.3</p> <p>Mejora (tipo 2): La explicación de jQuery era demasiado breve para poder ser útil para los alumnos. Lo he explicado sin complicarlo para que los alumnos vean la utilidad de la librería.</p> <p>Ubicación: Glosario.</p> <p>Mejora (tipo 2): Debe pasarse el glosario antiguo al nuevo formato.</p> <p>Ubicación: En bastantes sitios.</p> <p>Mejora (tipo 1): Enlaces antiguos y desfasados. Idiomas y códigos marcados, siglas, etc..</p> <p>Ubicación: No especificada.</p> <p>Mejora (Examen online): Debe añadirse alguna nuevas preguntas sobre jQuery.</p>		
Versión: 01.00.00	Fecha de actualización: 30/04/14	
Versión inicial de los materiales.		