MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO WEB EN ENTORNO CLIENTE
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

El desarrollo asíncrono

12

ÍNDICE

/ 1. Introducción y contextualización práctica 3

/ 2. Introducción a la programación asíncrona 4

/ 3. AJAX 4

3.1. Funcionamiento de AJAX 5

3.2. Objeto 'XMLHttpRequest' y métodos 5

3.3. Propiedades del objeto 6

3.4. AJAX: GET o POST 6

3.5. Ejemplo de peticiones 6

3.6. Cambio de estado en la petición 7

3.7. Peticiones síncronas con AJAX 8

3.8. AJAX: respuesta del servidor 8

/ 4. Limitaciones de AJAX 9

/ 5. Caso práctico 1: "Realizar carga asíncrona" 10

/ 6. Caso práctico 2: "Optimizar código" 10

/ 7. Resumen y resolución del caso práctico de la unidad 10

/ 8. Bibliografía 11

OBJETIVOS

Comprender las comunicaciones asíncronas.

Saber utilizar comunicaciones asíncronas.

Diferencia comunicaciones síncronas.

Saber crear peticiones asíncronas.

Conocer los mecanismos de procesamiento de peticiones asíncronas.

Saber procesar los resultados de peticiones asíncronas.

/ 1. Introducción y contextualización práctica

Una vez somos capaces de crear un contenido web, es necesario que nos centremos en la **interacción con el usuario.** Cuando realizamos una **aplicación web** que **intercambia datos con un servidor**, es muy importante que este intercambio de datos no suponga una gran latencia. Antiguamente, cuando se realizaba una petición web, era preciso volver a recargar toda la página para actualizar un campo determinado, lo cual generaba un gran tráfico de datos adicional, pues, por pequeño que fuese el cambio, la web tenía que recargarse completamente. Gracias a la comunicación asíncrona mediante AJAX, es posible recargar una página web de manera parcial. Con los conocimientos de manipulación de DOM, JavaScript y HTML, podremos generar contenido dinámico en una página web de manera asíncrona.

Fig. 1. La comunicación asíncrona ha supuesto una mejora indiscutible en el desarrollo web.

Escucha el siguiente audio que describe el caso práctico que resolveremos a lo largo de la unidad.

Audio intro. "Comunicaciones asíncronas en JavaScript"
https://bit.ly/2WhhnQJ

/ 2. Introducción a la programación asíncrona

En cualquier solución informática hay que considerar **dos posibles aproximaciones: síncrona y asíncrona.** Teniendo en cuenta que las máquinas en la actualidad normalmente disponen de un procesador, es importante gestionar correctamente su uso. Una mala gestión de Este puede producir un tiempo de respuesta muy elevado. Entendemos que el tiempo de respuesta es aquel que transcurre desde que el usuario solicita una acción hasta que obtiene el resultado.

Fig. 2. La programación asíncrona mejora el tiempo de respuesta.

En el desarrollo de interfaces de usuario, es **muy importante** que **el tiempo de respuesta** se adapte a las necesidades de este, y que sea el menor posible. En el caso de que el tiempo de respuesta sea muy elevado, es recomendable mostrar un mensaje informativo al usuario, por ejemplo, utilizando un cuadro de diálogo indicando «Espere, por favor».

En un **modelo de programación síncrono**, se ejecuta una tarea en cada momento. Cuando se invoca una función que realiza una acción que requiere mucho tiempo, solo se devuelve el resultado cuando la tarea finaliza. En este contexto, el resto del programa se para hasta que la acción finaliza. En un modelo de ejecución asíncrono, se pueden ejecutar varias tareas a la vez, siendo posible comenzar una acción mientras que el programa sigue realizando otras tareas. Cuando la acción iniciada finaliza, es posible tratar el resultado. Cuando se utiliza una solución asíncrona, es necesario utilizar el concepto de hilos de ejecución para poder llevar acabo diferentes tareas y que el programa principal no espere por ellas.

En la **programación en entorno del cliente**, necesitamos ser capaces de generar soluciones asíncronas que faciliten la interacción con el usuario. De esta forma, una web puede consultar un valor en el servidor mientras el usuario recibe datos de otra petición.

/ 3. AJAX

Como hemos visto anteriormente, en el entorno web hay que considerar la comunicación con el servidor a la hora de realizar cualquier tarea por parte del usuario.

AJAX *(Asynchronous JavaScript and XML)* es un **conjunto de tecnologías de desarrollo** que se utiliza en el lado del cliente para crear comunicaciones asíncronas. Las aplicaciones creadas con **AJAX** son capaces de **enviar** y **recibir** peticiones asíncronas al servidor, de forma que se evita bloquear la interfaz gráfica del usuario, es decir, la página web. Es posible implementar esta funcionalidad, porque existe una separación entre la capa de comunicación y presentación. Además, **AJAX permite cargar dinámicamente** el contenido de una web sin tener que recargar de nuevo la web. En las soluciones más modernas de AJAX, se sustituye el XML por la representación en JSON. Al igual que XML, JSON introduce un formato para el intercambio de datos entre el cliente y servidor en formato texto. JavaScript permite convertir cualquier objeto a formato JSON y enviarlo al servidor (lo mismo sucede al recibir datos).

AJAX no es una tecnología independiente, sino la unión de diferentes tecnologías. Normalmente, como estamos en un entorno web, se utilizan HTML y CSS para el marcado y el estilo, respectivamente. Utilizando el DOM de JavaScript es posible modificar el contenido de la web cuando se reciben de manera asíncrona los nuevos datos. Para poder utilizar AJAX, es necesario crear un objeto nuevo dentro del DOM de JavaScript: *XMLHttpRequest*. Por lo tanto, hay que considerar lo siguiente cuando hablamos de AJAX:

- **No** es un lenguaje de programación.
- **Es la combinación** de diferentes técnicas.

- **Utiliza un nuevo objeto** para solicitar y recibir datos del servidor.
- **Utiliza el DOM** de HTML y JavaScript para representar los datos.

Audio 1. "Seguridad en AJAX"

https://bit.ly/3j3W5jg

3.1. Funcionamiento de AJAX

Podemos decir que el funcionamiento de AJAX se divide en diferentes pasos, desde que se solicita la petición, hasta que se recibe la respuesta:

1. Se produce un evento en la página web, por ejemplo, el usuario selecciona 'ciudad' y se solicitan las provincias al servidor.
2. Se **crea un objeto** *'XMLHttpRequest'* en JavaScript para atender a la solicitud.
3. A través del objeto, se **envía la petición** al servidor.
4. Una vez el servidor recibe la petición, la **procesa y genera** una respuesta que envía al cliente.
5. **El cliente lee la respuesta** a través de JavaScript y, utilizando el DOM HTML y JavaScript, visualiza el contenido.

Como se puede observar en el flujo anterior, no existe ninguna tecnología adicional cuando se utiliza AJAX, simplemente se incluye un objeto nuevo.

3.2. Objeto 'XMLHttpRequest' y métodos

En la actualidad, los navegadores **son capaces de soportar el objeto** *'XMLHttpRequest'* para dar cabida a la implementación de AJAX. Sin embargo, en navegadores no tan antiguos, la utilización del objeto dependía del navegador que se estuviese utilizando. Por ejemplo, IE5 e IE6 requerían un tratamiento especial para utilizar este objeto.

MÉTODO	DESCRIPCIÓN
open()	Permite especificar la petición indicando el método de conexión (GET o POST) y la URL a la que conectar, entre otros.
send()	Permite enviar la petición al servidor.
setRequestHeader()	Permite añadir valores a la cabecera de la petición.
getResponseHeader()	Permite recoger información en de la respuesta.

Tabla 1. Métodos relacionados.

Vídeo 1. "Carga asíncrona de datos con AJAX"

https://bit.ly/3en2gLP

3.3. Propiedades del objeto

Para **poder utilizar correctamente AJAX**, es necesario conocer las **propiedades del objeto que lo implementa**. A continuación, se muestran alguna de estas propiedades:

PROPIEDAD	DESCRIPCIÓN
onreadystatechange	Define la función a llamar cuando cambia el estado del objeto.
readyState	Guarda el estado del objeto.: 0: petición no inicializada 1: conexión establecida con el servidor 2: petición recibida 3: procesando la petición 4: petición finalizada y respuesta lista
responseText	Respuesta del servidor en formato texto.
responseXML	Respuesta del servidor en formato XML.
status	Devuelve el estado de la petición HTTP.
statusText	Muestra el estado de la petición en texto.

Tabla 2. Propiedades de XMLHttpRequest.

3.4. AJAX: GET o POST

A la hora de **utilizar AJAX**. cabe preguntarse qué tipo de método utilizaremos para enviar al servidor. Generalmente, las **peticiones GET** son más rápidas y se pueden utilizar en la mayoría de los casos. Sin embargo, es recomendable utilizar el método POST en los siguientes casos:

- **Al actualizar** un archivo o la base datos en el servidor.
- **En el envío de gran cantidad de datos.** Recordemos que las peticiones POST no están limitadas en tamaño.
- En el caso del **envío de datos sensibles como contraseñas.** En general, aquellos datos que no puedan ser visualizados en la interfaz gráfica.

Fig. 3. AJAX puede ser considerado de bajo nivel en JavaScript.

3.5. Ejemplo de peticiones

En el siguiente ejemplo, se muestra una inicialización para realizar una **petición AJAX** pasando por parámetro el nombre del usuario. En este caso, el nombre de usuario está definido como un literal. Como se puede observar, la creación del objeto AJAX se realiza igual que cualquier otro objeto nativo.

```
var xhttp = new XMLHttpRequest();
//Envío de la petición al servidor
xhttp.open("GET", "test.asp?user=fran", true);
xhttp.send();
```

Código 1. Inicialización del objeto para la petición AJAX.

En el siguiente ejemplo, se muestra **cómo se crea un objeto** que realiza una petición al servidor utilizando el método POST. En este caso, los parámetros que se van a enviar al servidor no se incluyen en la petición, es decir, solo se incluyen directamente en el método send().

```
var xhttp = new XMLHttpRequest();
//Envío de la petición al servidor
xhttp.open("POST", "test.asp", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("user=fran");
```

Código 2. Petición con POST

Cabe destacar que ambos fragmentos de código **realizan la misma tarea**, lo único que **cambia es la forma** de completar el paso de parámetros y el método utilizado.

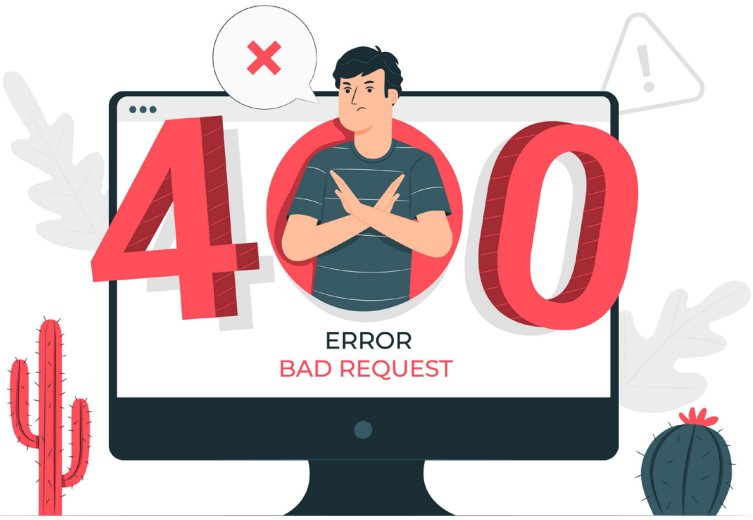

Fig. 4. Las peticiones en AJAX también pueden producir errores.

3.6. Cambio de estado en la petición

Para poder **implementar correctamente una solución AJAX** y que tenga efecto sobre la interfaz web, es necesario definir qué sucede cuando cambia el estado de la petición, pues será en este momento cuando los datos estén listos desde el servidor, es decir, cuando se obtiene la respuesta a la solicitud realizada.

Para llevar a cabo esta tarea, hay que implementar correctamente el método *onreadystatechange* disponible en el objeto XMLHttpRequest. Veamos un ejemplo de funcionamiento:

```
//Supongamos que ya existe el objeto y
definimos el comportamiento de la función
xhttp.onreadystatechange = function() {
 if (this.readyState == 4 && this.status ==
200) {
   document.getElementById("test").
innerHTML = this.responseText;
 }
};
xhttp.open("GET", "text.php?user=fran",
true);
xhttp.send();
```

Código 3. Método onreadystatechange.

Como se puede observar en el ejemplo anterior, la definición del método a ejecutar para procesar la respuesta utiliza una **función anónima** en la que se tiene en cuenta el estado de la petición. Si el estado es 4 y, además, la petición HTTP se ha ejecutado correctamente (código 200), se procesan los datos. En este caso, el procesamiento de la respuesta implica actualizar el documento web asignando al elemento con el identificador 'test', el valor en formato texto de la respuesta recibida del servidor.

Esta función es lanzada cuatro veces (1-4), pues se producen cuatro cambios de estado en el atributo asociado; por ello, hay que establecer una condición de qué realizar cuando la petición ha terminado correctamente.

Fig. 5. AJAX ofrece estados en las peticiones que hay que consultar.

3.7. Peticiones síncronas con AJAX

Hay que considerar la posibilidad de utilizar **AJAX mediante peticiones síncronas**, pues el API permite realizar dicha tarea. Sin embargo, no es recomendable, porque JavaScript detendrá la ejecución hasta recibir la respuesta del servidor. En el supuesto caso de que el servidor esté muy ocupado o su flujo de ejecución vaya más lento de lo normal, la aplicación web quedará suspendida a la espera del resultado. En este escenario, es muy normal que el usuario cierre el navegador e interrumpa la ejecución.

3.8. AJAX: respuesta del servidor

Como hemos visto anteriormente, el procesado de la **respuesta del servidor utiliza una función callback**, que es invocada al r**ecibir el resultado del servidor**. Una función *callback* es aquella que se invoca cuando tiene lugar un evento asíncrono.

La respuesta del servidor posee diferentes atributos que pueden ser consultados dependiendo de cómo se quieran representar o manipular los datos recibidos.

Por un lado, la **respuesta del servidor** ofrece una **representación de los datos** en formato cadena que puede ser utilizado directamente en el documento HTML. Estos datos se proporcionan a través de la propiedad *responseText*. Por otro lado, también se ofrece la respuesta en formato XML, siendo necesario conocer el DOM de XML para manejar correctamente los nodos del documento desde JavaScript.

```
xmlResponse = xhttp.responseXML;
txt = "";
x = xmlDoc.
getElementsByTagName("users");
for (i = 0; i < x.length; i++) {
 txt += x[i].childNodes[0].nodeValue +
"<br>";
}
document.getElementById("user").
innerHTML = txt;
xhttp.open("GET", "users.xml", true);
xhttp.send();
```

Código 4. Procesamiento de una respuesta en XML.

En el ejemplo anterior, se muestra el procesamiento de la respuesta en XML recibida desde una petición AJAX. Como se puede observar, es necesario manejar los nodos del documento XML.

/ 4. Limitaciones de AJAX

Aunque **AJAX introduce grandes ventajas** a la hora de cargar e interactuar con una página web, es cierto que existen algunas limitaciones cuando se utiliza:

Fig. 6. AJAX tiene algunas limitaciones para las peticiones.

- **Si el usuario dispone** de un navegador que no soporta JavaScript o no tiene implementado el objeto de AJAX, esta funcionalidad estará deshabilitada y, por lo tanto, el comportamiento no será el esperado. Si esto sucede, es necesario proporcionar métodos alternativos a la aplicación para poder llevar a cabo las tareas necesarias.

- **Desde el punto de vista** de la usabilidad y accesibilidad, algunas webs que están implementadas en AJAX no pueden ser adaptadas para personas con alguna discapacidad. Esto es importante tenerlo en cuenta para hacer la página accesible a todas las personas.

- **Por motivos de seguridad**, AJAX incorpora técnicas para permitir la conexión solo bajo el mismo dominio. En caso de querer comunicar con otro dominio, hay que incluir específicamente un valor en la cabecera antes de realizar la petición. Esto se debe a que pueden estar realizándose conexiones asíncronas a diferentes dominios sin que el usuario sea consciente de ello, poniendo en riesgo su dispositivo.

Vídeo 2. "Respuesta del servidor con AJAX"
https://bit.ly/2ZrGQsY

/ 5. Caso práctico 1: "Realizar carga asíncrona"

Planteamiento. En uno de los mayores proyectos que tenemos en nuestro equipo, se nos pide realizar una carga asíncrona en una página. El mayor reto es que solo se puede cargar una parte de la página, sin realizar la carga total.

Nudo. ¿Crees que es posible realizar esto? En caso afirmativo, ¿cómo lo harías?

Desenlace. Como hemos visto anteriormente, las comunicaciones asíncronas a través de JavaScript no solo permiten realizar una carga en otro hilo, sino que también proporcionan mecanismos para poder actualizar el resultado obtenido en una porción de la web. Gracias a este funcionamiento, es posible cargar una parte de la web sin recurrir a la carga total. Para poder implementar este comportamiento, hay que introducir los identificadores necesarios en HTML y desarrollar las *callbacks* requeridas en JavaScript. Estas *callbacks* serán lanzadas cuando la petición asíncrona finalice. Nótese que tenemos que considerar también los casos erróneos en nuestra implementación.

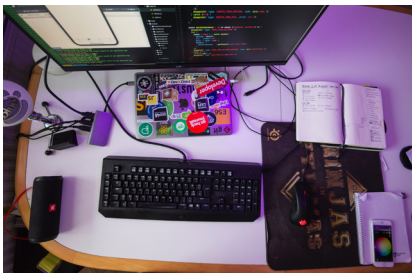

Fig. 7. Buscar soluciones para la carga asíncrona puede requerir un rediseño de la aplicación web.

/ 6. Caso práctico 2: "Optimizar código"

Planteamiento. Al realizar una auditoría de nuestro código, observamos que todas las *callbacks* que se utilizan en AJAX se implementan en la llamada de cada objeto asíncrono. Por ello, el número de líneas de nuestra solución es demasiado grande.

Nudo. ¿Cómo podrías mejorar esto?

Desenlace. Como hemos visto en el manejo de AJAX, las *callbacks* son ejecutas cuando se produce un cambio de estado en el objeto. En muchas ocasiones, los programadores sin experiencia ubican el código de la *callback* en el mismo objeto. Sin embargo, cuando el número de *callbacks* crece, puede ser posible que necesitemos reutilizar alguna de ellas. Para evitar tener código duplicado, una buena práctica es crear funciones que serán invocadas a través de la *callback*. De esta forma, no es necesario duplicar código innecesario, mejorando el grado de reutilización.

Fig. 8. Buscar la mejor solución software debe tener un compromiso entre la calidad y el tiempo requerido para ello.

/ 7. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema hemos presentado diferentes mecanismos de comunicación con el servidor. **Hemos diferenciado entre una comunicación asíncrona y una comunicación síncrona.** Desde el punto de vista web, hemos visto que hay que evitar, en la medida de lo posible, la comunicación síncrona, puesto que supone una espera indeterminada para el usuario.

Como mecanismo de comunicación asíncrono, **hemos presentado AJAX**. Como hemos visto, más que una tecnología nueva, se trata de un conjunto de tecnologías trabajando por una solución. De esta forma, para implementar una solución correcta en AJAX, es necesario conocer tanto JavaScript como HTML.

El mecanismo de funcionamiento de AJAX requiere de un objeto especial que en la actualidad es común para todos los navegadores. Este objeto funciona a través de unas funciones *callbacks*, que son definidas por el usuario, para implementar el comportamiento del objeto cuando se obtiene la respuesta del servidor o se produce un error. Para poder implementar AJAX correctamente, hay que tener en cuenta diferentes aspectos de seguridad para garantizar la integridad de los datos del usuario.

Resolución del caso práctico de la unidad

Como hemos visto en este tema, **las comunicaciones asíncronas** con JavaScript hacen uso de AJAX a través del objeto *XMLHttpRequest*. Durante una petición AJAX, el estado de la petición pasa por **4 estados diferentes**. Cuando la comunicación ha finalizado correctamente es cuando podemos actualizar la web. Hay que tener en cuenta en el código que el estado interno del objeto de la petición debe ser 4, y la respuesta de la petición 200. En nuestra situación, no estamos considerando los estados de la petición para actualizar la web. Para solucionarlo habría que filtrar el resultado cuando nos llega del servidor.

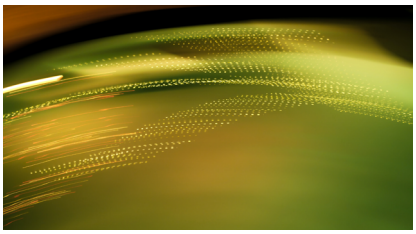

Fig. 9. La carga asíncrona de una web puede mejorar la velocidad de interacción con el usuario.

/ 8. Bibliografía

Paxton, j. (2020). *Javascript cookbook: programming the web.* Sebastopol: O'Reilly Media Inc.

Refsnes, H. (2010). *Learn JavaScript and AJAX with W3Schools.* Hoboken: Wiley.