MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO WEB EN ENTORNO SERVIDOR
**TÉCNICO EN DESARROLLO DE APLICACIONES WEB**

# Generación dinámica de páginas web: mecanismos de generación dinámica

08

# ÍNDICE

# OBJETIVOS

***Definir y utilizar formularios para responder de forma dinámica en los eventos de aplicación Web.***

***Identificar y aplicar los parámetros relativos a la configuración de la aplicación Web.***

***Describir aplicaciones Web con mantenimiento de estado y separación de lógica de negocio***

***Probar y documentar el código implementado***

***Analizar y utilizar los lenguajes de programación JavaScript con Ajax.***

## / 1. Introducción y contextualización práctica

El desarrollo de aplicaciones web permite que la generación de las páginas web de forma dinámica se realice desde el servidor, como se ha visto en temas anteriores. Pero también será posible la creación de páginas web dinámicas desde el cliente, para este objetivo será clave el uso de JavaScript y AJAX.

Sin duda, la tecnología clave para este tipo de desarrollos en la actualidad es AJAX, puesto que permite que el servidor solo devuelva los datos que se han de mostrar, pero será desde el lado del cliente utilizando JavaScript y AJAX que se genere y actualice el resultado que se va a mostrar en el navegador.

Además, gracias a la librería Node.js será posible la interconexión del código JavaScript en páginas HTML. En el tema 1 se expuso cómo crear un servidor utilizando Node.js, ahora, se verán las pautas de programación necesarias para crear los diferentes módulos que constituyen el funcionalmente global de la aplicación.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema, encontrarás su resolución en el apartado Resumen y Resolución del caso práctico.


*Fig. 1. JavaScript. Lenguaje de programación.*

*Audio Intro. "¿Cómo consigo un sitio web más dinámico? ¿Tecnologías?"*
*https://bit.ly/2GhYkRG*

# / 2. Creación de contenidos dinámicos en el lado del cliente

Una **página web dinámica** es aquella que presenta diferentes tipos de contenidos, datos o información en función de distintos parámetros, por ejemplo, en base a las credenciales de un usuario requeridos para la autenticación de este, se mostrará un contenido u otro.

Por lo tanto, podemos decir que las páginas web dinámicas permiten devolver un contenido específico para cada petición, en lugar de hacerlo de forma estática, para todos lo mismo. Para la construcción de páginas web dinámicas será necesario el uso de mecanismos dinámicos, tanto en el lado del cliente como del servidor, en el caso **del lado del cliente hablamos de JavaScript y AJAX**, lenguajes de programación que abordaremos en este tema, mientras que en el **lado del servidor** se suele utilizar entre otras tecnologías, **PHP**, como se ha visto en los capítulos anteriores.

Por el contrario, una **página web estática** no cambia su contenido hasta que el diseñador vuelve a programarlo, por lo tanto, el usuario no podrá actualizar su contenido hasta que no vuelva a hacer una petición de la página web al servidor. Podemos decir, que se basan solo en mostrar información al usuario, pero no interaccionan con él, como sí ocurre con las dinámicas. Este tipo de páginas están creadas utilizando **HTML y CSS**.

Hasta ahora, en el lado del cliente, solo nos hemos centrado en la creación de la capa de presentación utilizando HTML y CSS, las páginas web dinámicas y los datos que se mostraban de forma en el navegador eran generados así desde el servidor, utilizando las tecnologías vistas en los capítulos anteriores.

Desde el lado del cliente uno de los lenguajes más extendidos en cuanto a la creación dinámica de contenido para páginas web es JavaScript, cuya ejecución se lleva a cabo desde el propio navegador.

Recuerda el funcional completo de una página web dinámica que se representa en el siguiente diagrama.


*Fig. 2. Diagrama petición de página web dinámica.*

# / 3. AJAX. ASYNCHRONOUS JavaScript AND XML

La actualización del contenido de una página web utilizando JavaScript implica que para que la información mostrada en un sitio web se modifique, debe cambiar la respuesta del servidor, es decir, será necesario estar realizando peticiones constantes al servidor para actualizar la información y con ello se modifique la interfaz de presentación.

Para evitar tener que estar realizando peticiones constantes, aparece **AJAX**. Gracias a esta tecnología **ya no es necesaria la actualización permanente del sitio web por parte del usuario**, sino que a través de la instanciación del objeto **XMLHttpRequest** se mantiene a la espera de cambios y los muestra cuando son recibidos.

**AJAX permite el desarrollo de aplicaciones web dinámicas que funcionan de forma asíncrona**.

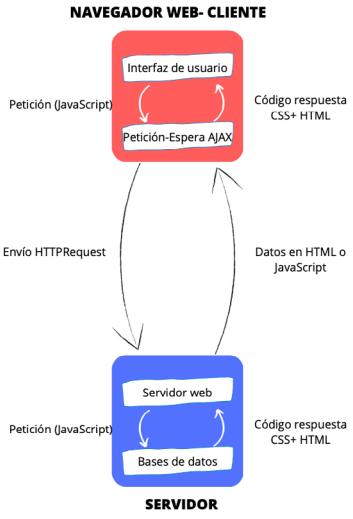


*Fig. 3. Diagrama funcionamiento AJAX.*

El funcionamiento de AJAX se basa en los siguientes pasos o fases:

1. En primer lugar, el navegador realizará una llamada que activa la instanciación de **XMLHttpRequest**.

2. **Instanciación objeto XMLHttpRequest**.

3. **Modelado de la función que va a "esperar" la respuesta** del servidor.

4. **Creación de la función que envía la petición** al servidor.

5. La función de respuesta queda escuchando y **cuando recibe la respuesta del servidor realiza el resto de las instrucciones** descritas en la función de respuesta.

# / 4. DOM. Document Object Model

En primer lugar, conviene conocer la estructura principal de una página HTML, esto es, de aquellos elementos que definen la presentación de la interfaz.

Una de las características que se requieren para poder actuar de forma dinámica sobre los elementos de una página web es el DOM (*Document Objet Model*), que **permite acceder a los diferentes elementos de una página web**, puesto que trata cada uno de éstos como nodos de un árbol, siendo el árbol el documento HTML al completo.

El siguiente código supone la estructura básica de un documento escrito en **HTML**, en el que el nodo raíz será el elemento HTML y los nodos padres **HEAD** y **BODY**.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//
EN" "http:// www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<title>Tema 8. DWES</title>
</head>
<body>
<p>Soy una página de prueba</p>
</body>
</html>
```

*Código 1. Estructura básica DOM.*

**Este modelo permite diferenciar diferentes tipos de nodos** a tener en cuenta puesto que las funciones en JavaScript permiten distinguir entre la selección de un tipo u otro. Por ejemplo, será posible utilizar funciones que nos permitan acceder al contenido de un nodo indicando el tipo o el nombre. Si equiparemos esto con otros lenguajes de programación, acceder a un nodo de un documento, será equivalente a extraer el valor de un atributo en objeto en programación Java.

- ***Document***: Se trata del nodo raíz del que derivan el resto.
- ***Element***: Representa las etiquetas utilizadas en HTML, puede contener atributos, los cuales coinciden con los utilizados en cada etiqueta.
- **Attr**: Representa los atributos de las etiquetas HTML.
- **Text**: Representa el texto que se encuentra entre los delimitados de una etiqueta HTML.
- ***Comment***: Se utilizan para representar los comentarios en una página HTML.

## 4.1. Funciones DOM. Extracción de valores

Como se ha descrito anteriormente, este modelo utiliza un sistema de nodos que quedan identificados de diferentes formas, por ejemplo, a través del nombre de elemento, de un identificador o de un atributo, entre otros. A continuación, veremos las **funciones que permiten acceder a cada uno de estos elementos para consultar o modificar su contenido**.

- **getElementsByTagName(Etiqueta)**: Obtiene todos los elementos de un documento HTML cuya etiqueta sea indicada por parámetro. Esta función se aplica al elemento "document", es decir, a todo el documento HTML.

```
var elemP = document.getElementsByTagName("p");
```

*Código 2. Función acceso a elementos por etiqueta.*

Si existe más de un elemento con estas características, la variable devuelta será un array con todos los elementos "p", para recorrer el array y extraer el valor de cada uno de los elementos se utiliza un bucle for. La función se aplica al elemento "document", es decir, a todo el documento HTML.

```
for(var i=0; i< elemP.length; i++) {
 var contenidoParrafo = elemP[i];
}
```

*Código 3. Función acceso a array de elementos.*

- **getElementsByName(Etiqueta)**: Obtiene todos los elementos de un documento HTML cuyo nombre (atributo name) coincida con el que se indica por parámetro. Esta función se aplica al elemento "document", es decir, a todo el documento HTML.

```
var elemName = document.getElementsByName("prueba");
```

*Código 4. Función acceso a elementos por nombre. .*

- **getElementsById(Etiqueta)**: Obtiene todos los elementos de un documento HTML cuyo identificador coincida con el que se pasa por parámetro. Esta función también se aplica al elemento "document", es decir, a todo el documento HTML.

```
var elemId = document.getElementById("id");
```

*Código 5. Función acceso a  elementos por identificador.*

**Es de las funciones más útiles en el desarrollo de aplicaciones web dinámicas.**

```
var idElemento = document.getElementById("id1");
<div id="id1">
        <a href="/" id="logo">...</a>
</div>
```

*Código 6. Función acceso a array de elementos por identificador.*

## 4.2. Funciones DOM. Inserción de elementos

Además del acceso a los nodos ya creados y la modificación de su valor, es posible **crear nuevo contenido utilizando las funciones dedicadas a tal fin**:

- **document.createElement(Etiqueta)**: Crea un elemento del tipo especificado por parámetro, es decir, del tipo de etiqueta indicado.

```
var nuevoElemento =
    document.createElement("p");
```

*Código 7. Creación nuevo elemento etiqueta.*


*Fig. 4. Nuevo elemento.*

- **document.createTextNode("cadena de texto")**: Crea un elemento de texto con el contenido especificado por parámetro.

```
var nuevoElementoTexto =
document.createTextNode("Las
cosas palacio van despacio");
```

*Código 8. Creación nuevo elemento de texto.*

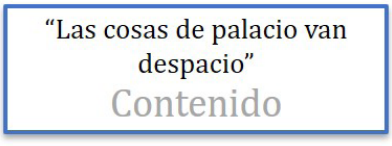


*Fig. 5. Nuevo elemento de texto.*

- **element.appendChild(elemento_hijo)**: Esta función realiza la inserción de un nuevo elemento "elemento_hijo" que dependerá del elemento "element" sobre el que se define acción. Para poder utilizar esta función se debe haber creado previamente el elemento del que depende el elemento hijo pasado por parámetro.

```
nuevoElemento.appendChild(nuevoElementoText);
```

*Código 9. Inserción de un nuevo nodo.*

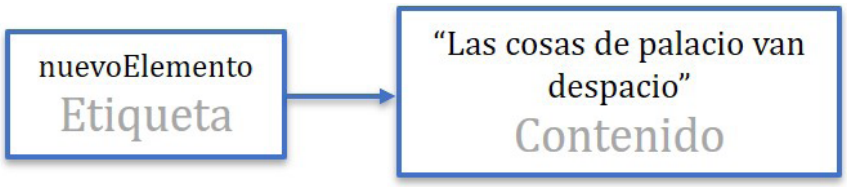


*Fig. 6. Inserción nuevo elemento.*

Para concluir la inclusión de un nuevo nodo, será necesario añadirlo al resto de la página, hasta ahora teníamos el nodo creado, pero no estaba insertado como tal en el documento HTML.

# / 5. XMLHTTPREQUEST

El objeto XMLHttpRequest **modela una interfaz utilizada para la realización de peticiones HTTP y HTTPS**. Por lo tanto, la creación de este elemento es imprescindible puesto que será necesario para establecer la conexión con el servidor.

El código para implementar esta instalación se realiza en función de los navegadores, puesto que en algunos casos obsoletos se utilizará **ActiveXObject**.

```
<if(window.XMLHttpRequest){
peticion = new XMLHttpRequest();
}
else if(window.ActiveXObject) {
// Obsoletos
peticion = new ActiveXObject("Microsoft.XMLHTTP");
}
```

*Código 10. Modelado XMLHttpRequest.*

Tras la instanciación de **XMLHttpRequest** se utiliza la función **onreadystatechange**, que hace que este objeto quede escuchando hasta que se reciba la respuesta del servidor. Finalmente, se indica el nombre de la función que se va a ejecutar cuando se reciba dicha respuesta, en este caso se ha creado con el nombre "respuesta".

```
peticion.onreadystatechange = respuesta;
```

*Código 11. Evento "escuchador" de cambios.*

El objeto XMLHttpRequest presenta varias propiedades que serán utilizadas en la implementación del código de conexión y tratamiento de la respuesta.

| Propiedad | Descripción |
|---|---|
| **readyState** | Recoge el valor del estado de la petición (0-1-2-3-4) de forma numérica. |
| **responseText** | Recoge la respuesta del servidor a la petición enviada como una cadena de texto. |
| **responseXML** | Recoge la respuesta del servidor a la petición enviada en formato XML. |
| **status** | Código de estado HTTP:<br>· 200: respuesta correcta<br>· 404: recurso no encontrado<br>· 500: error de servidor |
| **statusText** | Código de estado HTTP en cadena de texto. |

*Tabla 1. Propiedades del objeto XMLHttpRequest.*

*Audio 1. "Métodos de XMLHttpRequest"*
https://bit.ly/2SgggWy

# / 6. Petición al servidor

Para realizar la petición HTTP, se utiliza la función ***open***, la cual **permite indicar el método HTTP que se va a utilizar, la URL a la que se hace la petición, y finalmente, se añade el valor true o false**.

```
peticion.open('get o post', 'url', 'boolean');
```

*Código 12. Modelado petición para el servidor.*

Con el valor del último parámetro pudiendo ser true o false, el primero se utiliza para las **peticiones asíncronas**, mientras que el segundo se utiliza si lo que se pretende es implementar una **petición de tipo síncrono**.

Que una petición sea asíncrona no detiene la ejecución de la aplicación, mientras que, si se define como síncrona, sí se detendrá la ejecución hasta recibir la respuesta del servidor.

Al utilizar AJAX lo que se pretende es la creación de aplicaciones web dinámicas, por lo tanto, se utilizará prácticamente siempre el valor true, lo que ofrece un funcionamiento más dinámico que si detenemos la aplicación esperando a recibir una respuesta.

Finalmente, se realiza el envío de la petición al servidor utilizando la función **send**.

```
peticion.send(null);
```

*Código 13. Envío petición al servidor.*

*Vídeo 1. "Modelado petición AJAX"*
*https://bit.ly/36oZw06*

# / 7. Tratamiento de la respuesta

Cuando se recibe la respuesta se invoca una función, esto se modela en el apartado 3 a través de la función **onreadystatechange**. Esta función podrá **realizar cualquier acción que se implemente con código JavaScript**, ahora bien, de forma genérica se realizan algunas evaluaciones para comprobar si los valores obtenidos están completos o han producido algún error.

```
function respuesta(){
   if(peticion.readyState == 4){
      if(peticion.status == 200) {
         alert(peticion.responseText);
      }
   }
}
```

*Código 14. Diseño función respuesta.*

Uno de los elementos clave para el tratamiento de la respuesta es **readyState**, se trata de un valor que indica el estado del a petición, los posibles valores que puede tomar son:

- **Objeto no inicializado**, es decir, no se ha invocado al método open.
- **Petición en estado "cargando"**, no se ha invocado el método send.
- **Petición "cargada"**, sin respuesta del servidor.
- **Solo se han recibido algunos datos.**
- **Se han recibido todos los datos.**

Por lo tanto, todo el proceso quedará modelado de la siguiente forma:

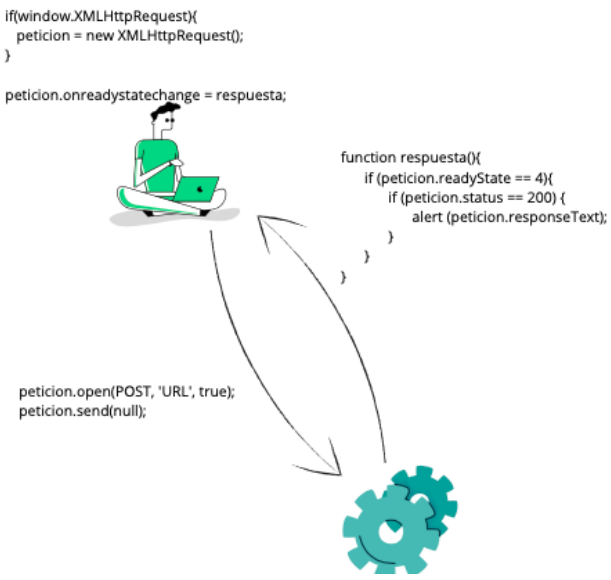


*Fig. 7. Diagrama petición-respuesta AJAX.*

*Vídeo 2. "Modelado respuesta AJAX"*
*https://bit.ly/3lbfQWn*

# / 8. NODE.JS

Cuando se crea una aplicación web PHP y MySQL, será la encargada de la implementación en el servidor, y, además, se necesitan conocimientos en CSS, HTML y JS, para el desarrollo de la parte del cliente. Mientras que **usando Node.js solo será necesario conocer CSS, HTML y JS**.

Como ya se introdujo en el primer capítulo, la arquitectura **Node.js** está basada en el motor de **JavaScript V8** y a través del lenguaje en JavaScript y los módulos Node.js permite el **desarrollo de aplicaciones en el lado del servidor puesto que permite utilizar JavaScript en múltiples entornos donde hasta ahora no era posible**.

La descarga desde paquete se realiza de forma inmediata desde el sitio web.

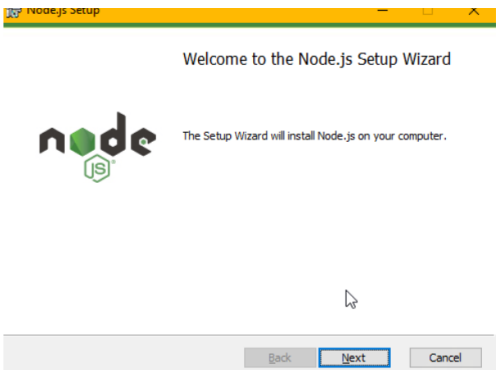


*Fig. 8. Instalación Node.js*

Tras la descarga se ejecuta el instalador descargado y comienza el proceso a través de una asistente de instalación. Basta con pulsar ***Next*** hasta completar la instalación, no es necesaria ninguna configuración específica.

Para comprobar si la instalación se ha realizado de forma correcta se utilizan los comandos:

- **node --version**: Tras realizar el proceso de instalación anterior para confirmar el resultado satisfactorio se ejecuta esta línea por comandos y el resultado mostrará la versión de Node.js instalada.

- **npm --version**: NPM es un gestor de paquetes de JavaScript, si al ejecutarlo se muestra una versión, está instalado. Con la instalación de Node.js se instala también este gestor. Si se desea actualizar la versión basta con ejecutar por línea de comandos **npm install -g npm**

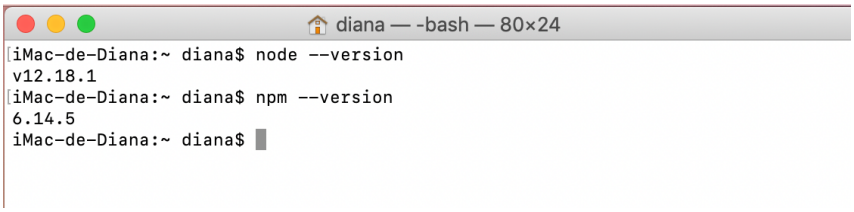


*Fig. 9. Comandos instalación y resultado ejecución para Node.js*

## 8.1. Funcionamiento general

El desarrollo con Node.js se puede realizar desde un entorno de desarrollo más "complejo" como ***Visual Studio Code***, hasta otros muchos más sencillos de usar, es decir, en un editor de texto como **Notepad++**.

El funcionamiento de **Node.js** se basa en la **construcción de diferentes módulos que implementan la funcionalidad completa de cualquier aplicación que se va a desarrollar**.

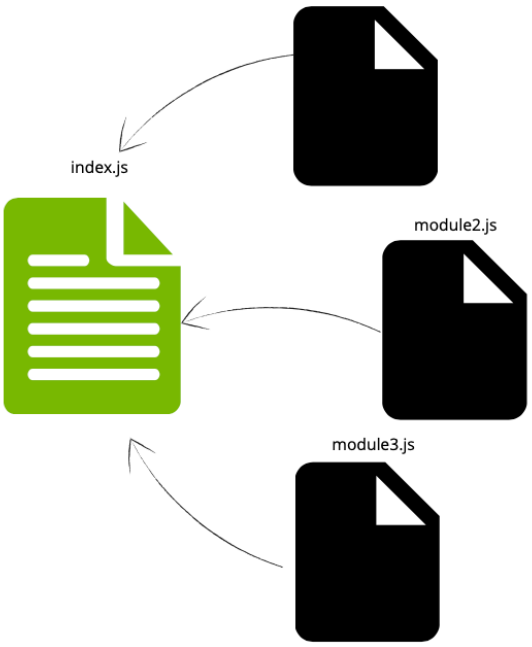


*Fig. 10. Diagrama de ficheros Node.js*

*Audio 2. "Node.js la escalabilidad como clave"*
*https://bit.ly/34jQb7m*

Ahora bien, la implementación de estos ficheros requiere de ciertos aspectos que se deben tener en cuenta en su diseño. En primer lugar, se crean las funciones y demás códigos de aplicación en la distribución de módulos diseñada. Podemos diferenciar entre dos tipos de ficheros, el principal (habitualmente llamado **index.js) y los ficheros que implementan cada uno de los módulos**.

Como se puede observar en la imagen anterior, habitualmente se creará un archivo principal para el proyecto (**index.js**), desde el que serán llamados el resto de los módulos de la aplicación. Todos los ficheros que se van a desarrollar serán de tipo JavaScript, por lo tanto, **todos se almacenan con la extensión .js.**

A continuación, se crean tantos archivos correspondientes a los módulos y se implementa en ellos el código necesario para el funcionamiento de la aplicación.

**Importante**, para ejecutar un fichero creado utilizando Node.js se usa la siguiente instrucción desde el terminal o cmd. Es necesario que nos situemos en la ruta en la que está almacenado el fichero, por ejemplo, si hemos creado el archivo ficheroNode.js en la carpeta "proyectosNode" en primer lugar debemos acceder a esta ruta o ejecutar la instrucción añadiéndola.

```
>> cd proyectosNode
>> node ficheroNode.js
```

*Código 15. Ejecución Node.js*

## 8.2. Conexión entre módulos

Para realizar la conexión entre los módulos y con el fichero principal, el llamamiento entre **archivos *js*** no se realiza de la forma habitual, es decir, como cuando se desarrolla para el navegador, sino que ahora requiere de una implementación diferente, estamos trabajando desde el servidor.

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

*Código 16. Inclusión ficheros JavaScript desde navegador.*

En Node.js se utiliza la instrucción ***require***, seguida del nombre completo del fichero *.js* del que se van a tomar las funciones implementadas.

Es importante indicar dónde está el fichero, es decir, la ruta exacta del mismo. Por ejemplo, si queremos conectar con un fichero que se encuentra en la misma ruta que el fichero desde el que es llamado se utiliza "**./**" delante del nombre del fichero.

```
require('./fichero.js');
```

*Código 17. Instrucción require en Node.js*

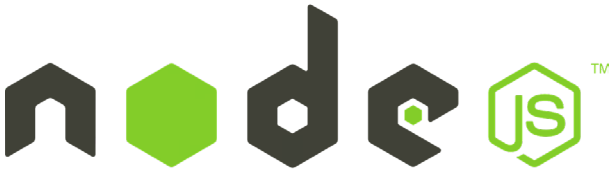

*Fig. 11. Imagen ilustrativa Node.js*

Finalmente, el resultado de la llamada a la función ***require*** se almacena en una constante, puesto que para poder acceder a las funciones contenidas en el fichero .js será necesario utilizar esta constante de acceso.

```
const c = require('./fichero.js');
```

*Código 18. Instrucción require ejemplo en Node.js*

Por tanto, la llamada a las funciones contenidas en los ficheros de módulo quedaría:

```
c.nombreFunción(parámetros);
```

*Código 19. Llamada función con parámetros en Node.js*

# / 9. Caso práctico 1: "Creación de una aplicación con AJAX. Implementación de la petición"

**Planteamiento**: Se va a diseñar una aplicación utilizando AJAX y HTML. En este caso práctico 1 se va a modelar las funciones relativas a la petición, el caso práctico 2 se verá en detalle cómo realizar el montaje completo del fichero HTML con AJAX. Para ello se pide implementar una función completa que al ser invocada realice todo el proceso necesario la construcción y envío de la petición.

**Nudo**: Para el modelado de las funciones que generan la petición AJAX se tiene en cuenta los siguientes supuestos:

- La petición será de tipo POST.
- Será de tipo asíncrona.
- Se implementa una función para la instanciación del XML.

**Desenlace**: En primer lugar, cuando se produce la carga de la página se ejecuta una primera función que es la encargada de inicializar la petición.

```
window.onload = acceso;
```

La función *acceso* quedaría de la forma:

```
function acceso() {
peticion("http://localhost/holamundo.txt", "POST", respuesta);
}
```

*Código 20. Código AJAX de acceso.*

La función *petición* recibe por parámetro la URL a la que se accede, el tipo de método para el protocolo de transporte utilizado, y EL booleano indicador del sistema asíncrono.

```
function peticion(url, metodo, funcion) {
peticion_http = inicializaXMLHttpRequest();

if(peticion_http) {
peticion_http.onreadystatechange = funcion;
peticion_http.open(metodo, url, true);
peticion_http.send(null);
}
}
function inicializaXMLHttpRequest() {
if(window.XMLHttpRequest) {
return new XMLHttpRequest();
}
else if(window.ActiveXObject) {
return new ActiveXObject("Microsoft.XMLHTTP");
 }
}
```

*Código 21. Código AJAX de petición.*

# / 10. Caso práctico 2: "Creación de una aplicación con AJAX. Implementación de la respuesta"

**Planteamiento**: Se va a diseñar una aplicación utilizando AJAX y HTML. En este caso se va a modelar la función de la respuesta y como resultado final se incorpora el montaje completo del fichero HTML con AJAX.

**Nudo**: Cuando se recibe la respuesta desde el servidor se invoca una función que será la encargada del tratamiento de los datos y el diseño de la parte de presentación de la interfaz. Como se puede ver en el siguiente fragmento de código se comprueba el valor del estado de la petición y del envío HTTP.

```
function respuesta() {
if(peticion_http.readyState == READY_STATE_COMPLETE) {
if(peticion_http.status == 200) {
    alert(peticion_http.responseText);
  }
 }
}
```

*Código 22. Código función respuesta*

**Desenlace**: La implementación del fichero completo quedaría de la siguiente forma:

```
<html><head>
var READY_STATE_UNINITIALIZED=0;
var READY_STATE_OPENED =1;
var READY_STATE_HEADERS_RECEIVED =2;
var READY_STATE_ LOADING =3;
var READY_STATE_COMPLETE=4;

var peticion_http;
function peticion(url, metodo,funcion){
peticion_http = inicializaXMLHttpRequest();
if(peticion_http) {
peticion_http.onreadystatechange = funcion;
peticion_http.open(metodo, url, true);
peticion_http.send(null);
 }
}
function inicializaXMLHttpRequest() {
 if(window.XMLHttpRequest) {
   return new XMLHttpRequest();
 }
}
function acceso() {
peticion("http://localhost/holamundo.txt", "POST", respuesta);
}
window.onload = acceso;
</script>
</head>
<body></body>
</html>
```

*Código 23. Código completo petición-respuesta AJAX.*

# / 11. Exportación de contenido y LOGS

**a. Exportar el contenido de los módulos**

**Los módulos son aquellos ficheros** que nos permiten **dividir la aplicación en múltiples partes** y así resultará mucho más sencillo implementarla y mantenerla por separado.

Para concluir la implementación de los módulos correctamente, tras haber desarrollado su contenido, es necesario indicar en los ficheros de cada módulo qué funciones van a ser exportadas y con qué nombre identificativo. Para ello se utiliza la palabra "***exports***" seguida de un punto (.) y de un nombre identificativo para alguna de las funciones implementadas en el archivo. Finalmente, se indica con cuál de las funciones del fichero quedará asociado el identificador que se ha escogido.

```
exports.idFunción= nombreFunción;
```

*Código 24. Instrucción exports.*

Por ejemplo, en el siguiente caso, se ha creado una función con el nombre sumar que recibe por parámetro dos números enteros y devuelve la suma de ambos. Cuando se utiliza esta función desde otro fichero, es decir, cuando es llamada, se va a utilizar como nombre identificador, sum:

```
exports.idFunción= nombreFunción;
```

*Código 25. Ejemplo instrucción exports.*

**b. Función LOG**

Node.js implementa diferentes funciones que nos permiten analizar el funcionamiento y desarrollo del contenido de los ficheros. Por ejemplo, una de las más útiles es la función *log*.

La función log, **devuelve el contenido por consola de aquello que se le pasa por parámetro.**

```
console.log(nombreMódulo);
```

*Código 26. Instrucción log.*

Si ahora se ejecuta el fichero en el que se ha implementado la función anterior, el resultado devolverá entre {} el contenido del objeto, el que se ha pasado por parámetro a la función log. **Cada una de las líneas que se muestran entre las llaves son las propiedades del objeto**.

```
$ node nombreFichero.js
{ id1: [Function: function1],
id2: [Function: function2],
...
idN: [Function: functionN],
}
```

*Código 27. Ejemplo instrucción log.*

# / 12. Resumen y resolución del caso práctico de la unidad

En esta unidad, hemos aprendido que una **página web dinámica** es aquella que muestra un contenido distinto en función del usuario o de las peticiones de éste. Para la construcción de páginas web dinámicas será necesario el uso de mecanismos dinámicos, tanto en el lado del cliente, como del servidor, en el caso **del lado del cliente hablamos de JavaScript y AJAX**.

La actualización del contenido en una página web utilizando JavaScript implica que para que la información mostrada en un sitio web se modifique, debe cambiar la respuesta del servidor, es decir, será necesario estar realizando peticiones constantes al servidor para actualizar la información y con ello se modifique la interfaz de presentación.

AJAX permite el desarrollo de aplicaciones web dinámicas que funcionan de forma asíncrona. El funcionamiento de AJAX se basa en la instanciación de **XMLHttpRequest**, el modelado de las funciones de respuesta y el tratamiento del envío de la petición.

El objeto XMLHttpRequest es la pieza clave del funcionamiento de AJAX puesto que modela una interfaz utilizada para la realización de peticiones HTTP y HTTPS. La creación de este elemento es imprescindible puesto que será necesario para establecer la conexión con el servidor.

Finalmente, se ha analizado de manera exhaustiva el diseño y uso de la arquitectura **Node.js** basada en el motor de JavaScript V8.

**Resolución del caso práctico de la unidad**

Como se ha visto a lo largo del tema, en la actualidad la tendencia habitual es crear sitios web menos estáticos y más dinámicos. Para conseguir esto, entre otras tecnologías, está cada vez más presente el uso de AJAX.

AJAX permite que el contenido de un sitio web quede a la espera de la recepción de los datos solicitados a un servidor, pero que además sea posible seguir realizando cambios sobre el sitio web sin necesidad de recargar la página web a cada instante.

Las tecnologías que quedan implicadas en este tipo de diseños serán desde el lado del cliente, un modelado con JavaScript y AJAX, que gracias al envío de peticiones y la escucha activa a través de XMLHttpRequest, quedará a la espera de la respuesta. Habitualmente el fichero implementado en el servidor podrá estar desarrollado en PHP.

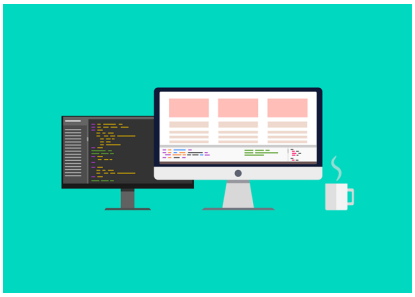

*Fig. 12. Páginas webs dinámicas.*

# / 13. Bibliografía

Ganzabal, X. (2019). *Desarrollo web en entorno servidor* (1.a.ed.). Madrid, España: Síntesis

Vara, J.M. (2012). *Desarrollo en entorno Servidor* (1.a ed.). Madrid, España: Rama.