CLIENTES WEB

1.1.- EL MODELO CLIENTE-SERVIDOR

Una página desarrollada en Xhtml y CSS decimos que es **estática**. Se trata de un simple documento de texto que viaja por la red utilizando algún protocolo de comunicación (habitualmente el http – hipertext transfer protocol) desde una máquina donde está alojada (a la que llamamos **servidor**) hasta la máquina del usuario que ha solicitado la página (a la que llamamos **cliente**).
En el cliente el documento solicitado se recompone y se carga en una aplicación llamada **navegador web**. El navegador web lee el fichero e interpreta el código Xhtml y aplica, en su caso, los estilos CSS para componer la apariencia de la página. Pero una vez terminado este proceso ya no hay más procesamiento, por lo que la página en sí misma no podrá cambiar o ejecutar tareas. De ahí el hecho de que digamos que es una página estática.
A esta forma simple de funcionamiento de la web se le llama **modelo cliente-servidor**:

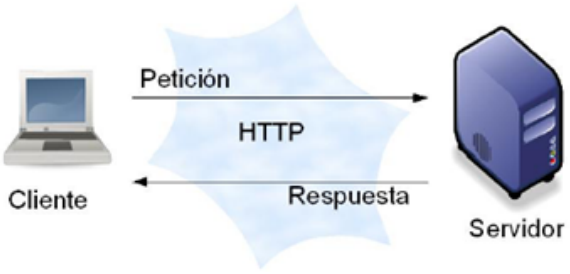


Fig. 1 Modelo cliente-servidor básico.

1.2.- EL NAVEGADOR

El cliente web más habitual es el navegador web (que llamaremos de aquí en adelante **browser o navegador**). Este programa permite que el usuario escriba un localizador universal de recurso (llamado **URL**) y se encarga de generar la petición en internet. Serán los routers y máquinas de la web los encargados de redirigir esta petición al servidor.
El servidor web por su parte contiene un repositorio de los recursos disponibles para peticiones (páginas web, hojas de estilo, imágenes, vídeos, etc.). Cuando recibe una petición envía al cliente el/los recursos solicitados).
En la actualidad los navegadores son clientes web complejos. No sólo interpretan texto sino que cuentan con intérpretes de lenguajes de programación especiales, cuyo código puede insertarse dentro de las páginas web para darles dinamismo. A estos lenguajes se les suele llamar **lenguajes de scripts**.
Según **W3Counter** (compañía que suministra un sistema gratuito de estadísticas para sitios web) la siguiente imagen muestra una estadística del uso de los navegadores en 34.916 sitios web durante el mes de julio de 2017 (https://www.w3counter.com/globalstats.php?year=2017&month=6). Recordar que para Windows 10, Internet Explorer pasa a denominarse Microsoft Edge:

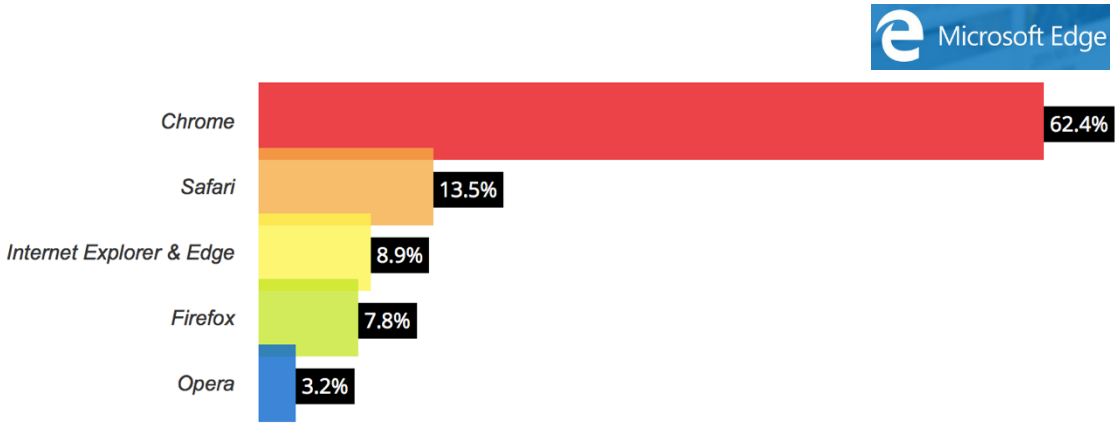


Fig. 2: Popularidad de los navegadores.

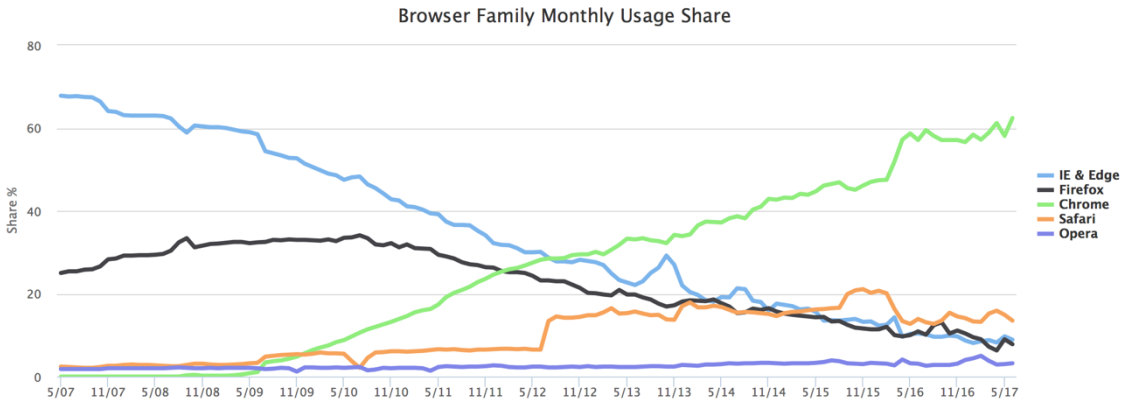


Fig. 3 Evolución de la popularidad de los navegadores desde 2007 (Fuente: https://www.w3counter.com/trends).

Podemos ver información detallada de sobre los navegadores que soportan objetos HTML5 y CSS3. La página se llama "When can I use" (cuándo puedo usar) y su URL es la siguiente www.caniuse.com
Por ejemplo, he consultado en esta página cuándo se puede usar la capacidad de Drag&Drop (arrastrar y soltar) que se ha definido como uno de los nuevos aspectos del HTML 5. Esta es la comparativa resultante:

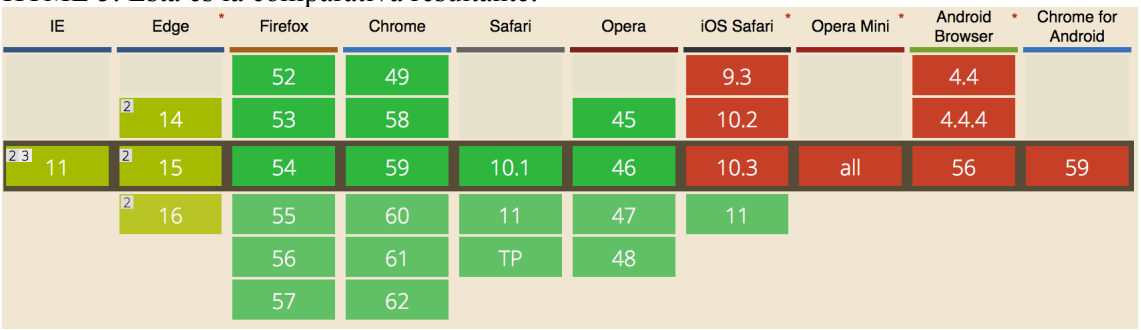


Fig. 4: Soporte de Drag&Drop de HTML5 en los navegadores web

En la imagen anterior el verde indica que la característica de Drag&Drop sí está soportada. El verde oliva indica que se soporta sólo parcialmente y el rojo indica que no está soportada. Así Edge 13 sólo lo soporta parcialmente, las versiones de Safari no.
Puedes utilizar esta página para comprobar coberturas de HTML5, CSS3, JavaScript, etc.

1.3.- CÓMO FUNCIONA EL PROCESO DE CARGA

Cómo funciona el proceso de carga de una página web con el objetivo de realizar un diseño adaptable al dispositivo. A grandes rasgos, las diferentes etapas por las que podemos pasar cuando visitamos una web. Estas son:

- Latencia
- Petición DNS
- Redirecciones
- Peticiones HTTP
- Respuesta HTML
- Descompresión
- Carga del DOM
- Renderizado del HEAD
- Renderizado del BODY
- Eventos

1.3.1.- LATENCIA

La latencia o retardo, es **el tiempo que tardas en conectarte a tu proveedor de internet**. Normalmente en las conexiones de fibra óptica o ADSL este tiempo es mínimo (alrededor de 9ms). Esto es diferente en una red móvil, ya que el smartphone primero debe conectarse a una torre que le proporciona cobertura y en ese momento es cuando se produce la conexión. Dependiendo de las circunstancias, este proceso puede tardar varios segundos antes siquiera de empezar a cargarse la web.

1.3.2.- PETICIÓN AL DNS

Una vez tenemos la conexión establecida, **el navegador envía una petición al servidor de nombres** (DNS), proporcionado por el proveedor de internet. Esto para conocer la dirección IP a la que apunta el dominio que estamos visitando.

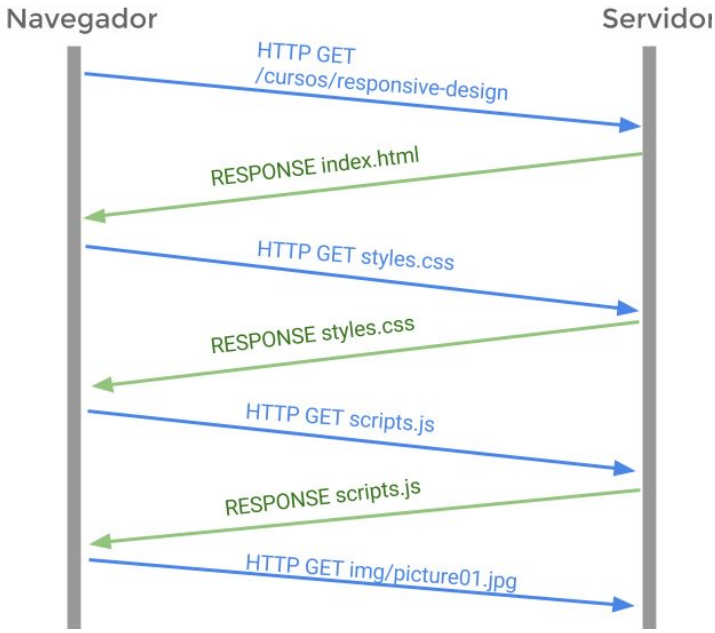
1.3.3.- REDIRECCIONES

No siempre la URL que introducimos en el navegador apunta directamente a una dirección IP, muchas veces apunta a otra URL, por ejemplo: **http://platzi.com/html** redirecciona a **https://platzi.com/cursos/html5-css3/**. **Las redirecciones añaden tiempo al proceso de carga de una web** aunque sean dentro del mismo dominio.

1.3.4.- PETICIONES HTTP

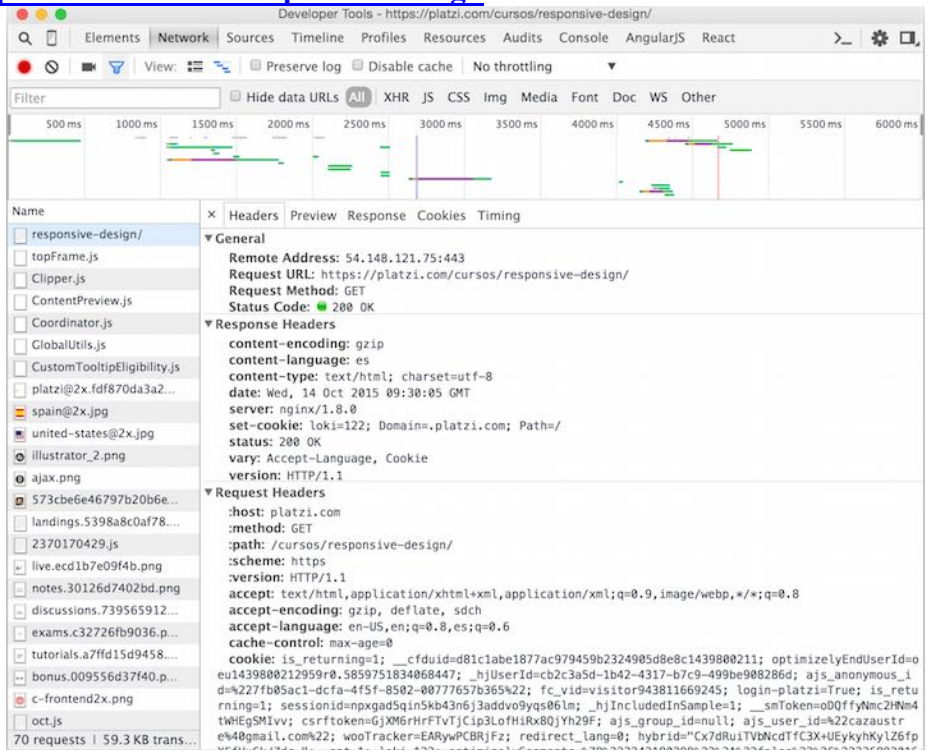
Ya por fin el navegador tiene la dirección IP. En ese momento solicita una petición HTTP al servidor para que éste le envíe la página html relativa a la URL introducida.

Las peticiones HTTP incluyen también **cabeceras**, para complementar la información de la petición. Una de estas cabeceras es "USER-AGENT", donde se guarda la información del navegador utilizado y el sistema operativo. Esto es útil si queremos desde el servidor **envíar una versión diferente** al cliente si este se conecta desde un smartphone o desde un laptop.



1.3.5.- ENVÍO DEL FICHERO HTML

Después de que el servidor reciba la petición HTTP, éste devuelve al navegador la página HTML correspondiente. La respuesta también tiene una cabecera, y en ella puede haber información sobre si se permite "CACHEAR" el recurso y el tiempo que puede ser. **Esta información la podemos configurar en el servidor.** Las cabeceras HTTP se pueden visualizar en las "CHROME DEVELOPER TOOLS" en la pestaña de NETWORK. En la imagen vemos las cabeceras para la petición y respuesta HTTP a la URL **https://platzi.com/cursos/responsive-design**



1.3.6.- DESCOMPRESIÓN

Los ficheros HTML, JavaScript y CSS pueden enviarse comprimidos desde el servidor usando **GZIP.** Esto hace que pesen menos y su tiempo de descarga se reduzca. El navegador al enviar la petición HTTP, lleva en sus cabeceras información sobre esto. Le indica al servidor **si acepta ficheros comprimidos**. De ser así, el servidor los mandará de esta manera, sino los envía sin comprimir lo que afectará al tiempo de carga. Por tanto es una buena práctica habilitar la compresión GZIP en nuestros servidores. El navegador al recibirlos se encargará de descomprimirlos.

1.3.7.- CARGA DEL DOM

En el siguiente paso el navegador se encargará de **formar el DOM (Document Object Model)** a partir del contenido HTML que recibe. El DOM no es más que una representación en forma de árbol de nodos de la estructura de nuestra página HTML.

1.3.8.- RENDERIZADO DE LA ETIQUETA HEAD

Cuando el DOM está listo, el navegador comienza a renderizar el HTML, elemento a elemento, empezando por el contenido de la etiqueta **<head>**. Cada vez que el navegador encuentre un enlace a un fichero (como puede ser un fichero CSS, JavaScript, imagen, etc.) se realizará una petición HTTP. Si el navegador tiene algunos recursos cacheados de una anterior visita, no necesitará pedirlos nuevamente. De esta manera **la primera visita puede llevar más tiempo de carga que las posteriores**.

EJECUCIÓN SINGLE-THREAD

Tenemos que tener cuidado con los ficheros JavaScript que carguemos. JavaScript es SINGLE-THREAD, que significa el navegador **sólo puede ejecutar un fichero a la vez**. Cada etiqueta **<script>** que encontremos en el **<head>** del fichero HTML será ejecuta en el momento que llegue el renderizado. Tanto scripts internos como enlaces a ficheros JavaScript externos. Por tanto, si tenemos varios **<scripts>** podemos ver cómo la página sigue cargando pero en el navegador aún tenemos un fondo blanco ya que e**l renderizado no ha llegado todavía al contenido del HTML.**

1.3.9.- RENDERIZADO DEL <BODY>

Cuando finaliza el renderizado del **<head>** comienza la parte del contenido. De igual manera que en el **<head>** va pintando uno a uno los elementos que va encontrando. También se van aplicando las reglas CSS que tenga cada elemento para poder colocarlo en la pantalla y ver cuál será su aspecto final.

IMÁGENES

Cuando el navegador encuentra una etiqueta
¡Error! Nombre de archivo no especificado. <img> solicita el recurso y comienza a descargar la imagen. Si la imagen es muy pesada esto puede llevar un tiempo.

IMÁGENES DE BACKGROUND

Si un elemento, debido a su estilo CSS tiene una imagen de fondo, en este momento se empezaría a descargar.

SCRIPTS

Si tenemos código JavaScript dentro de la etiqueta **<body>**, es en este instante cuando se empieza a ejecutar.

1.3.10.- EVENTOS

Cuando finalmente se ha renderizado el documento, el navegador dispara el evento ONLOAD. Si tenemos algún script que escuche ese evento, en este momento se ejecutaría.

1.4.- MODELO DE EJECUCIÓN DE CÓDIGO EN EL CLIENTE

El navegador no deja de ser un programa en la máquina del cliente. El proceso de ejecución se inicia con la solicitud de un recurso web mediante la descripción de su URL.

Formato de las URL: servicio://maquina.dominio:puerto/ruta/recurso

Determinadas máquinas llamadas **servidores DNS** (Domain Name Servers) se encargan de averiguar la dirección IP de la máquina que contiene el recurso solicitado.
Una vez obtenido este dato se genera una petición en la Web a través del protocolo utilizado (que en la URL corresponde a la parte de servicio). Típicamente se trata del protocolo http, aunque pueden ser muchos otros protocolos (ftp, https, etc.).
El siguiente paso pone en marcha la operativa de la Web. Nuestros paquetes http viajarán a través de diferentes máquinas, hubs, switchs, y unas máquinas especiales llamadas routers que lo irán redirigiendo hasta que alcancen el servidor.
El servidor recibe la petición por un puerto de entrada (habitualmente en web utilizamos el puerto 8080). Comprueba si existe /ruta/recurso y en caso afirmativo se vuelve a generar el mismo proceso pero en este caso enviando el recurso hasta la máquina del cliente.
Una vez que el cliente recibe el recurso se carga en el navegador web y aquí se atraviesan varias fases de procesamiento. El navegador se compone de varias partes:

- El **motor de búsquedas** aporta una forma sencilla de comunicación entre el usuario y la máquina. Es la ventana de nuestro navegador, que debe tener una interfaz agradable e intuitiva.
- El **motor de visualización**. También llamado motor de renderizado. Se encargará de mostrar los recursos en la ventana del navegador. Para este cometido tendrá que interpretar y mostrar código HTML o incluso XML (en este caso mostrará el árbol sintáctico). También tendrá que aplicar los estilos CSS. Incorpora por lo tanto un **analizador XML**, un **analizador CSS**, y en su caso distintos analizadores para los lenguajes del cliente (típicamente un **analizador JavaScript**).
- El **motor de comunicaciones**. Es la parte del navegador capaz de trabajar y soportar los distintos protocolos utilizados en el modelo de red. A este modelo se le suele conocer como **modelo TCP/IP** y trabaja con diferentes protocolos como el TCP, UDP, IP, HTTP, HTTPS, FTP, …
- Los **plugins**. Cada fabricante lucha por hacer su navegador más popular. Esto pasa porque el navegador soporte y reconozca el mayor número de tecnologías posible. Para evitar complicar más el programa se suelen añadir a posteriori pequeños módulos que actúan como intérpretes. A cada uno de estos módulos agregados lo llamamos plugin y tiene como misión soportar una tecnología diferente. Por ejemplo podemos tener plugins para soportar otros lenguajes de marcas como VRML, Math-Ml, etc.

Modelo básico de ejecución de código en el cliente:

1. Descargar el recurso web.
2. Si se trata de una página web se interpreta el código xml o html (utilizando su analizador correspondiente).
3. Se interpretan los estilos asociados para producir la visualización (renderizado) final.
4. Se dispara el motor de JavaScript una vez finalizada la carga y análisis del recurso.

La última parte (el motor de JavaScript) corresponde a la ejecución de un verdadero lenguaje de programación en el cliente web. Para ser exactos no tendríamos que hablar sólo de JavaScript sino de **lenguajes de scripting**.

1.5.- LENGUAJES DE SCRIPTING

Script: El término script (que podemos traducirlo al español como **guión**) suele referirse en informática a un pequeño bloque de código que puede insertarse en medio de un documento. Este código suele ser interpretado comando a comando (como un guión) y por lo tanto no es habitual que se escriba en un lenguaje **compilado** sino en un lenguaje **interpretado**.

Para dar dinamismo a la web era necesario dotar a las páginas de un verdadero lenguaje de programación, con sus bucles, sus funciones, sus respuestas a eventos del usuario, etc. Surgieron así los lenguajes de scripting, que se caracterizan por incluir pequeñas piezas de código dentro del documento HTML.

Para que esto funcionara y el lenguaje de scripting pudiera manipular la propia página web fue necesario primero crear una estructura de objetos de la propia página web. Es decir, si en el código HTML de una página hemos incluido una lista ordenada y tres botones necesitaremos un mecanismo para identificar a cada uno de estos elementos de alguna manera. Esto se llamó el **DOM** (Modelo de Objetos del Documento) y nuestro lenguaje de scripting puede utilizarlo para realizar su tarea. El trabajo con el DOM será un tema dentro de esta asignatura.

DHTML: El DHtml o **Html Dinámico** no es, como puede parecer, un lenguaje propio. Consiste en combinar Html, CSS, el DOM y Javascript para construir páginas cuya apariencia y forma pueden cambiar como respuesta a las necesidades de los usuarios. Lo aprenderemos en esta asignatura.

TIPOS DE SCRIPTS EN EL CLIENTE WEB:

Podemos diferenciar dos tipos de scripts: los que se ejecutan cuando se finaliza la carga de la página en el navegador (documento cargado) y los que se ejecutan como respuesta a una acción (llamada **evento**). Ejemplos de eventos son:

- Cuando un elemento recibe el cursor.
- Cuando el ratón entra o sale de un elemento.
- Cuando se hace click del ratón.
- Cuando se pulsa una tecla.

Hay multitud de eventos. Y son los que nos permitirán enganchar y ejecutar el código de nuestros scripts.

1.6.- CARACTERÍSTICAS DE LOS LENGUAJES DE SCRIPTING

- Son **interpretados**: ya sabes que en los lenguajes compilados los programas se convierten a binario (o en casos como Java a bytecodes) antes de poder ejecutarse. Típicamente esto lo hace un programa llamado compilador. En los lenguajes interpretados no se produce esta traducción previa del código. El código fuente se ejecuta directamente utilizando un programa intermedio llamado intérprete que va leyendo secuencia a secuencia. Por este motivo (esto debe hacerse cada vez que se ejecute el mismo código) los lenguajes interpretados suelen ser más lentos. En las páginas web conviene que los lenguajes de script sean interpretados porque así el código viaja al cliente dentro del HTML y es en el navegador donde se interpreta. Por esta razón la gran mayoría de los lenguajes de script son interpretados.
- Son **fáciles de escribir**: dado que escribiremos el código dentro del html de una página (o en ficheros separados que finalmente se anexarán a la página web) estos lenguajes fueron diseñados como versiones pequeñas y simples de otros lenguajes más complejos. Se redujo la complejidad a cambio de perder algo de rendimiento. JavaScript por ejemplo hereda toda la sintaxis del C++ pero es mucho más simple que éste y no soporta una orientación a objetos tan avanzada. VBScript hereda también toda su sintaxis del VisualBasic pero tampoco soporta toda su estructura.
- Se **escriben en texto plano sin formato**: aspecto necesario para poder incluirlos dentro de las páginas web.

Ejemplos de lenguajes de Scripting en un cliente Web:

- **JavaScript:** es actualmente el líder indiscutible en los clientes web. Durante algunos años compitió con VBScript que quedó finalmente desfasado. Como hemos comentado hereda su sintaxis del C++ aunque su nombre induce a muchos a pensar que es una versión de Java (cosa que no es cierta). No obstante, y dado que Java también hereda gran parte de su sintaxis del C++, comparte con Java la forma de escritura.
- **VBScript**: es el lenguaje de scripting desarrollado por Microsoft. Fue el principal competidor de JavaScript pero terminó cayendo en desuso con el auge de otros navegadores Web (Microsoft lo mantenía sólo compatible en sus versiones de Internet Explorer, y los programadores se encontraban con que sus productos no podían ejecutarse en otros navegadores).
- **JScript:** una vez que Microsoft comprendió que había perdido la batalla con su VBScript decidió implementar su propia versión de JavaScript, a la que llamó JScript (cuestión que aporta un poco más de confusión).
- **ECMAScript**: es una norma o estándar de la que derivó el JavaScript. La norma ECMA 262 (también conocida como EcmaScript Tercera Edición) pretende ser un estándar al que todos los lenguajes de script deberían ceñirse.
- **ActionScript:** no es en realidad un lenguaje de scripting para la web. Lo desarrolló Adobe para su programa Flash. Con este lenguaje (bastante similar al JavaScript en sintaxis) se pretende dar interactividad a las películas de Flash. La última versión, la 3.0, sufrió cambios importantes en la gestión de eventos para parecerse más a la forma en que los eventos se gestionan en Java (mediante el uso de listeners).

1.7.- EMAC

ECMA es el nombre de la asociación europea de fabricantes de computadoras (European Computer Manufacturer Association). Se trata de una organización sin ánimo de lucro que se encarga, entre otras cosas, de regular el funcionamiento de muchos estándares de la industria mundial, no solo de Europa. Entre estos estándares se encuentran los de algunos lenguajes de programación, como por ejemplo C# (que es el estándar ECMA-334), las extensiones del lenguaje C++/CLI, el lenguaje de descripción de servicios Web (WSDL) o, por supuesto, nuestro queridísimo ECMAScript que es el estándar ECMA-262.

JavaScript es una de las implementaciones del estándar ECMA-262, en concreto la que se usa en los navegadores. Pero existen otras implementaciones con sus propias extensiones, como por ejemplo ActionScript.

Por eso, cuando en la web se habla de ECMAScript y JavaScript en realidad se está hablando de la misma cosa. El primero es el estándar que define cómo debe funcionar el lenguaje, y el segundo es una implementación concreta de lo que indica la especificación estándar.

1.7.1.- EL PROBLEMA CON EL NOMBRE "ECMASCRIPT 7" O "ES7"

ECMAScriptLa actual versión de ECMAScript es la número 7 (aunque sea la 6ª que sale realmente). El problema es que cuando en la Web buscas información sobre ES7 o ECMAScript 7 realmente de lo que están hablando no es de esta versión.

Se suponía que para esta última versión que acaba de salir se iban a incluir muchas nuevas funcionalidades en el lenguaje. Sin embargo no dio tiempo material y se ha lanzado una actualización muy descafeinada, con solo un par de cositas. La verdadera versión "grande" con enormes cambios fue la del año pasado, ES6 o ES2015. La más reciente, ES2016, es una actualización muy floja.

Así que cuando en Internet leas algo sobre ES7 en realidad están hablando de algo que no existe todavía y que de momento no se sabe cuándo va a estar disponible. Es posible que muchas de las nuevas características estén ya en ECMAScript 2017 (que sería realmente ES8), pero puede que tarden mucho más y sea finalmente en 2019 o 2020 cuando veamos algunas de ellas. Así que mucho cuidado con lo que te digan sobre ES7: probablemente no estén hablando de lo que tú crees.

1.7.2.- ECMASCRIPT EN LOS NAVEGADORES

Además cada navegador tiene su propia implementación de ECMAScript, es decir, su propio motor de JavaScript. Como todos se han creado siguiendo lo que indica ECMA, en principio son 100% compatibles, pero podemos encontrar pequeñas diferencias entre implementaciones, en especial en aquellas partes del estándar que no dejan 100% claro cómo se debe actuar. Del mismo modo, diferentes implementaciones del motor del lenguaje pueden estar más o menos optimizadas.

1.8.- RESTRICCIONES EN LOS LENGUAJES DE SCRIPTING

A los lenguajes de scripting se les impusieron fuertes restricciones de seguridad, de forma que **no pueden acceder** libremente a los recursos de la máquina del cliente. Esto se hizo de forma deliberada.

Imagina la situación: eres un usuario de la web, te conectas y descargas una página. En medio de esa página hay código programado y ese código accede tranquilamente a tu disco duro y borra archivos. Si esto fuera posible dejaríamos un importante hueco de seguridad y nadie navegaría tranquilo.
Para evitar esto se impuso a los diseñadores de lenguajes de scripting esta restricción: el acceso a la máquina del cliente debe estar limitada y ser muy restringida. Sí se puede, por ejemplo, escribir pequeños ficheros de texto con información del servidor (lo que se conoce como **cookie**).

Resumiendo en los lenguajes de scripting para clientes Web hay dos tipos de restricciones de seguridad:

- 1ª El código de JavaScript sólo podrá realizar procesos relacionados con la página web donde se inserta. No está permitido el acceso libre a los recursos de la máquina del cliente. No se puede, por ejemplo, ejecutar una aplicación en el equipo del cliente descargada de internet. No se puede tampoco escribir directamente ficheros en la máquina del servidor.

- 2ª Los scripts siguen una política llamada **mismo origen**. Esto implica que los scripts de un sitio web no tendrán acceso a los datos de otros sitios web (envío de información en formularios, contraseñas, cookies). Las páginas web de distintos dominios no son accesibles entre sí.

TECNOLOGÍAS Y HERRAMIENTAS EN EL CLIENTE WEB.

2.1.- OTRAS TECNOLOGÍAS QUE NO USAN EL SCRIPTING

Ya hemos descrito los lenguajes de scripting. Aunque son ampliamente utilizados no es la única manera de contar con un lenguaje de programación dentro de una página web. Existen otras tecnologías que funcionan de forma diferente.
Estas otras tecnologías permiten escribir programas (e incluso compilarlos) y luego los adaptan para que todo el programa se descargue junto con la página y se visualice dentro de ella. Ejemplos:

1.- Los **Applets** de Java. Se trata de hacer programas Java que luego se incrustarán en las páginas web. Hay que mencionar que los applets tienen las mismas restricciones de acceso local que los lenguajes de scripting. Pero además están cayendo en desuso porque tienen restricciones adicionales:

- ~ Todo el programa compilado debe descargarse desde el servidor hasta el cliente, lo que ralentiza la velocidad de la página. En los lenguajes de scripting lo que viaja es texto plano y se interpretan en el navegador. En el caso de los applets viaja todo el programa con sus clases.

- ~ El programa se ejecuta en su propio espacio. Por así decirlo es como si se ejecutara en un pequeño marco flotante. Por ese motivo el programa no puede acceder al resto de la página.

Aun así, a menudo, es frecuente encontrarse páginas con applets que se han desarrollado para cuestiones que de otra forma son más complicadas como la generación de gráficos, estadísticas, etc.

2.- Las páginas realizadas en **Flash**. Se trata de un conocido programa de animación vectorial de la casa Adobe. Durante unos años tuvo un gran auge y surgieron multitud de sitios web hechos en Flash. Las ventajas que ofrece eran muy claras de cara al diseño. Cosas como Drag&Drop, animaciones de movimiento, animaciones de forma, menús contextuales, eran muy simples de crear en Flash y muy complejas a la hora de hacerlo en JavaScript. Actualmente las nuevas características de HTML5 y CSS3, junto a librerías hechas con JavaScript (como es el caso de **JQuery**) permiten hacer esto mismo con mucho menos coste, con lo que no sabemos cuál será el futuro de Flash. La principal desventaja de Flash era que las páginas ocupaban mucho espacio con lo que la descarga en internet se volvía lenta y pesada. La ventaja es que Adobe desarrolló ActionScript y Flash alcanzando y alto grado de nivel de programación.

3.- **AJAX**. Son las siglas de **A**sinchronous **J**avaScript **A**nd **X**ml. Se trata de una tecnología muy de moda que ha revolucionado el desarrollo web. De hecho se ha incluido dentro de la librería JQuery. La idea es poder traer al cliente información sin necesidad de pedir toda una página web completa. Esto se llama comunicación asíncrona. Tendremos tiempo de profundizar en este concepto porque AJAX será un tema de esta asignatura.

2.2.- HERRAMIENTAS Y LENGUAJES

El mundo de la informática cambia constantemente. Hay multitud de lenguajes, de librerías, de formas de entender un desarrollo. Por eso creo conveniente en este punto mostrar una visión global de tecnologías relacionadas que se están utilizando en los desarrollos Web.
La mayor parte de ellas puede encajarse en algún lugar concreto del desarrollo: bien sea en el cliente, en el servidor, o incluso en una capa intermedia y a menudo distribuida en ambas partes que se denomina la "lógica del negocio". Tendremos tiempo a lo largo de la materia de discutir y aclarar muchos conceptos. Por el momento sólo nombrar que a este paradigma en ocasiones también se le llama **MVC o modelo-vista-controlador**. El modelo se encargaría de los datos (residen en una base de datos en el servidor). El controlador sería la lógica del negocio o lo que es lo mismo toda la operativa de nuestra aplicación. Por último la vista corresponde a cómo se muestra la información (y es habitual que radique en el cliente aunque no es obligatorio).
Veamos entonces qué herramientas y lenguajes se utilizan, tratando de ubicarlos en su sitio más frecuente. No obstante sólo hablamos de eso, de sitio más frecuente. Un lenguaje como JavaScript puede usarse también para programar scripts del lado del servidor. Sin embargo en el lado del servidor son más populares otras tecnologías como PHP, JSP o ASP.
Por otra parte los desarrollos Web comienzan a enfrentarse también a otro tipo de retos. Nos referimos a los dispositivos móviles (terminales telefónicos, tablets, etc.). En la actualidad hay dos plataformas que compiten por la supremacía:
Los Iphones, Ipods y Ipads de Apple: utilizan sistemas operativos IOS y si queremos desarrollar **aplicaciones nativas** los entornos de desarrollo más utilizados son XCode y Cocoa. El lenguaje de programación estrella es el **Objetive-C** y **Swift.**

Vender aplicaciones para Apple: La plataforma más utilizada es el **iTunes**. Los programadores, previo pago de una licencia de desarrollador, publican y venden sus aplicaciones así como las actualizaciones de las mismas.

Los dispositivos **Android**. El Android es un sistema operativo abierto basado en Linux, impulsado por Google. Aunque llegó al panorama más tarde que los dispositivos de Apple está compitiendo fuerte debido a que es gratuito y tiene herramientas de desarrollo abiertas. Para programar estos dispositivos de **forma nativa** se utiliza el lenguaje **Java**. El entorno de desarrollo más utilizado es un simple IDE Eclipse con un plugin gratuito llamado ADT (Android Development Tools). Además hace falta una librería llamada Android-SDK, que también es libre y gratuita.

Vender aplicaciones para Android: La plataforma más utilizada es el **Android Market**. Tiene un funcionamiento muy similar al iTunes pero como es habitual en Android el proceso es gratuito para el desarrollador. Existen otras plataformas de venta y distribución como el Play Store que se distribuye con terminales de Samsung.
Es importante diferenciar entre los conceptos de Aplicación nativa y Aplicación Web:
Aplicación nativa: Se trata de programas que se ejecutan directamente sobre un sistema operativo. Por ejemplo si desarrollamos un programa en Java con las clases adecuadas del Android-SDK se ejecutará directamente en un terminal Android.

Las aplicaciones nativas requieren de conocimientos avanzados en los lenguajes de programación como Objetive-C y Java. Son más rápidas y acceden directamente a los recursos del sistema operativo sobre el que se ejecutan. Sin embargo muchos programadores, especializados en desarrollos WEB, han demandado la posibilidad de crear de forma sencilla e intuitiva aplicaciones WEB para dispositivos móviles.

BREVE INTRODUCCIÓN A JAVASCRIPT

3.1.- BREVE HISTORIA

Con el lanzamiento del navegador Netscape 2.0 se desarrolló un lenguaje de scripting en 1995. Su autor, Brendam Eich decidió en un principio llamarlo **LiveScript**. Uno de los motivos principales que impulsó esta necesidad de contar con un lenguaje en el cliente fue la validación de los datos que un usuario escribe en un formulario. Antes del scripting había que enviar el formulario completo al servidor y allí validar uno a uno el formato de cada campo. Si un campo fallaba (por ejemplo un email mal escrito) todo el formulario era rechazado. Dado que en esa época internet era muy lenta esto suponía unos tiempos de espera poco apropiados. Si en el cliente se dispone de un lenguaje de programación se pueden hacer las validaciones de los campos que escribe el usuario y avisarle de errores sin molestar al servidor. Surgió así LiveScript.

Sin embargo antes del esperado lanzamiento la empresa Netscape se anexiona a Sun Microsystems (gestora del lenguaje Java) que empieza a disfrutar de un enorme éxito en Internet gracias a los **applets**. Esto provoca que se cambie el nombre de LiveScript a **JavaScript 1.0** sólo por razones comerciales (ya que JavaScript, como hemos dicho, hereda su sintaxis original del C++). El lanzamiento de JavaScript fue una revolución. Los programadores que hasta ese momento estaban utilizando InternetExplorer (con su VBScript) consideraron que era preferible programar en JavaScript.

Tras esto se preparó la versión JavaScript 1.1 (para ser utilizada en Netscape 3.0) y por ese mismo tiempo Microsoft comprendió que perdería la batalla y desarrolla su propia versión, llamada **JScript** (para ser utilizada en IExplorer 3.0).

Aquí comienza una guerra entre estos dos navegadores, que ocasiona muchos perjuicios a los programadores, ya que la implementación del **DOM** en ambos navegadores no es la misma, con lo que los códigos no son 100% portables.

DOM (Modelo de objeto del documento):

Es la jerarquía de objetos existentes, que pueden ser usados desde JavaScript. En ambos navegadores son diferentes.

Por ejemplo para acceder al objeto que representa al propio navegador se utiliza un objeto llamado **navigator:**

- En Netscape 4.x es un objeto independiente: Navigator.
- En IExplorer 4.x es hijo del objeto window y se usa window.navigator

Ya hablaremos del DOM en un tema separado. Debido a este problema entre casas comerciales heredamos esta falta de homogeneidad y uno de los factores que más preocupaciones ha dado a los programadores ha sido la compatibilidad del código.

Netscape se pone en contacto con ECMA (una asociación europea para estándares) en 1997 para solicitar que JavaScript 1.1 sea un estándar. En ECMA trabajan técnicos de Netscape, Sun, Microsoft y otras muchas casas comerciales y se genera un nuevo estándar, el ECMA-262 (que aunque es sólo un estándar a menudo se le conoce como **EcmaScript**). Esta sería la guía a la que todos los navegadores deberían ajustarse en el futuro. Pero cuando el EcmaScript fue aceptado oficialmente como estándar por la **ISO** ya cada casa comercial había desarrollado sus nuevas versiones no compatibles (Netscape desarrolló JavaScript 1.2 y Microsotf llegó a JScript 3.03).

Hoy en día la mayoría de los navegadores incluyen versiones de JavaScript que son completamente compatibles con la norma EcmaScript versión 3. Los únicos problemas de compatibilidad se deben a cuestiones que cada navegador ha mantenido para soportar páginas antiguas y algún aspecto del DOM que sigue sin ser compatible.

Pero sí hay alguna buena noticia: **todos** los navegadores soportan JavaScript y éste es el lenguaje líder indiscutiblemente en la programación de scripts en los clientes web.

3.2.- INTEGRACIÓN DE CÓDIGO EN LAS ETIQUETAS HTML

Al igual que con las hojas de estilo en cascada existen tres formas de integrar el código: en un elemento HTML (evento), para todo un documento HTML o como un fichero externo.

3.2.1.- ENLAZANDO CÓDIGO MEDIANTE UN EVENTO.

Típicamente los eventos son atributos de las etiquetas HTML que comienzan con la palabra **on**. Veamos un ejemplo:

<input type="button" value="saludar" **onclick=**"…. Código JavaScript …"/>

Sólo cuando se produzca el evento se ejecutará el código. Observa que como estamos escribiendo un atributo de una etiqueta HTML todo el código JavaScript debe ir encerrado entre comillas.

Esta forma de enlazar código se usa principalmente para llamar a una función (que debemos programar) cuando ocurre un evento. No es buena idea escribir muchas líneas de código en un onclick="…" por ejemplo, ya que el código JavaScript aparecería muy disperso en la página (y sería difícil de mantener).

3.2.2.- ENLAZANDO CÓDIGO A TODO UN DOCUMENTO HTML

Se utiliza la etiqueta **<script> … </script>** y dentro de esta etiqueta todo el código que se escriba se asumirá que es JavaScript. Esto es así porque en la actualidad el lenguaje de scripting por defecto en todos los navegadores es este. No obstante antes era posible escribir etiquetas como esta:

```
<script language="VBScript">
... código en VBScript ...
</script>
```

Sin embargo sí que es habitual encontrarnos etiquetas como la siguiente:

```
<script type="text/javascript">
... código en JS ...
</script>
```

y de esta forma las usaremos.

La pregunta es: ¿Dónde se pueden colocar los script? Pues en principio en cualquier parte del documento. No obstante a lo largo del curso verás que a menudo te haré la recomendación de incluirlos dentro del <head> siempre que sea posible, de forma que no afecten al body. Recuerda que en el <head> incluimos la parte no visible de una página web (estilos, scripts, metainformación, etc.).

3.2.3.- LLEVANDO EL CÓDIGO JAVASCRIPT FUERA DE LA PÁGINA

De la misma forma que en el caso de las hojas de estilo a menudo tendremos funciones JavaScript que queremos utilizar en múltiples páginas. La opción más sencilla es sacar el código a un fichero externo de texto plano.

Los ficheros JavaScript no tienen que tener una extensión determinada, aunque es usual darles extensión .js. Muchos programadores también los nombran como .inc (de include, ya que serán incluidos dentro de las páginas). Nosotros nos acostumbraremos a la extensión .js.

El llevar el código JavaScript fuera de las páginas aporta muchas ventajas. Los sitios web quedan más estructurados. Un sólo cambio a una función afectará a todas las páginas que la invoquen, con lo que el mantenimiento es mucho más simple.

Para enlazar en nuestra página estos ficheros externos la forma es muy simple:

```
<script type="text/javascript" src="ruta/fichero.js>
... código extra en JS ...
</script>
```

Como ves basta con utilizar el atributo src (source) de la etiqueta script e indicar la ruta al fichero con nuestro código. Ya conoces el caso de las rutas absolutas y relativas. Si lo único que necesitas en un script es incluir un fichero externo, podrás cerrar la etiqueta script en la misma línea.

```
<script type="text/javascript" src="ruta/fichero.js/"></script>
```

ENLACES

- https://www.campusmvp.es/recursos/post/JavaScript-ECMAScript-ES6-Existe-ES7-Aclarando-las-diferentes-versiones-del-lenguaje.aspx
- https://www.ecma-international.org/ecma-262/7.0/
- https://platzi.com/blog/como-carga-una-web/