TEMA I

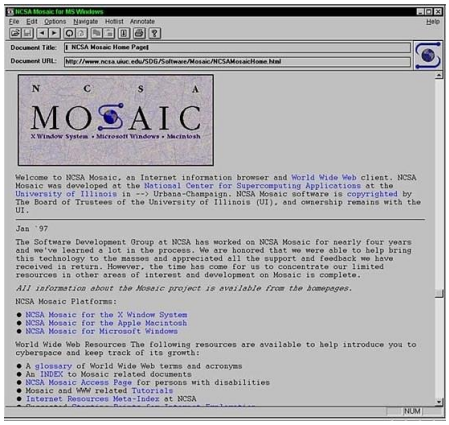
INTRODUCCIÓN AL DESARROLLO WEB *CLIENT-SIDE*

ÍNDICE

I DESARROLLO DE APLICACIONES WEB

La web fue inicialmente concebida y creada por *Tim Berners-Lee*, un especialista del *Laboratorio europeo de partículas* (CERN) en 1989. En sus mismas palabras, había una "*necesidad de una herramienta colaborativa que soportara el conocimiento científico*" en un contexto internacional. Él y su compañero *Robert Cailliau* crearon un prototipo de web para el CERN y lo mostraron a la comunidad para sus pruebas y comentarios.

Dicho prototipo estaba basado en el concepto de *hipertexto*. Como resultado se crearon unos protocolos y especificaciones que han sido adoptados universalmente e incorporados a Internet gracias a aportaciones posteriores como la popular interfaz **Mosaic**, desarrollado por el NCSA (*National Center for Supercomputing Applications*), organismo norteamericano relacionado con la investigación en el campo de la Informática y las telecomunicaciones.



Todos los prototipos y desarrollos posteriores crecieron bajo la guía del consorcio W3C, una organización con base en el MIT de Massachusetts y que se responsabiliza de desarrollar y mantener los estándares web.

El desarrollo web ha sido y sigue estando muy influenciado por múltiples campos como el de las nuevas tecnologías, los avances científicos, el diseño gráfico, la programación, las redes, el diseño de interfaces de usuario, la usabilidad y una variedad de múltiples recursos. De este modo, podemos entender el desarrollo Web como un campo multidisciplinar.

I.1 ÁREAS EN EL DISEÑO WEB

Hay cinco áreas que cubren la mayor parte de las facetas del diseño Web:

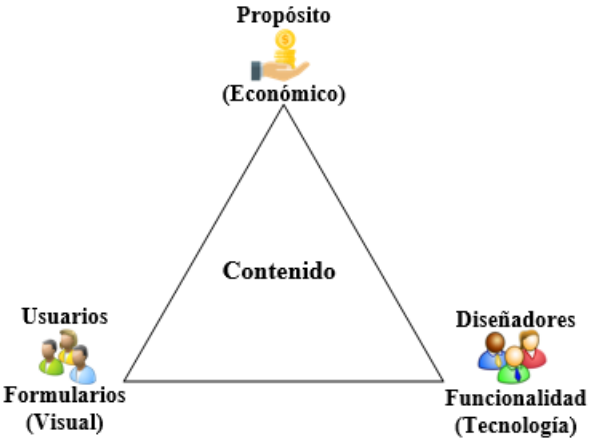
- ✓ **Contenido**: incluye la forma y organización del contenido del sitio. Esto puede abarcar desde cómo se escribe el texto hasta cómo está organizado, presentado y estructurado usando tecnologías de marcas como HTML.

- ✓ **Visual**: hace referencia a la plantilla empleada en un sitio web. Esta plantilla generalmente se genera usando HTML, CSS o incluso Flash (cada vez más en desuso) y puede incluir elementos gráficos para decoración o para navegación.

- ✓ **Tecnología**: En este contexto hacemos referencia a los diferentes tipos de elementos interactivos de un sitio web, ya sea con un usuario o con una aplicación, construidos empleando determinadas técnicas de programación. Tenemos dos tipos:
  - Tecnologías orientadas al cliente: Son aquellas que permiten crear interfaces de usuario y establecer comunicación con el servidor basadas en *HTML*, *CSS* y *JavaScript*, en este caso, el traductor (intérprete) suele ser el mismo navegador.
  - Tecnologías orientadas al servidor: Permiten implementar comportamientos de la aplicación web en el servidor, los lenguajes más utilizados son *Java EE*, *lenguajes .NET*, *PHP*, *Ruby on Rails*, *Python*, *Django*, *Groovy*, *Node.js*, etc…
- ✓ **Distribución**: la velocidad y fiabilidad con la que un sitio web se distribuye en Internet o en una red interna corporativa está relacionada con el hardware/software utilizado y el tipo de arquitectura de red utilizada en la conexión.
- ✓ **Propósito**: la razón por la que un sitio web existe, generalmente está relacionada con algún aspecto de tipo económico. Por lo tanto, este elemento debería considerarse en todas las decisiones que tomemos en las diferentes áreas.

El porcentaje de influencia de cada una de estas áreas en un sitio web, puede variar dependiendo del tipo de sitio que se está construyendo. En este sentido podemos suponer que una página web personal generalmente no tiene las consideraciones económicas que tendría una web que va a vender productos en Internet.

Podríamos visualizar este punto en la imagen de la derecha, donde vemos la relación entre usuarios, diseñadores, propósito y el contenido que se pretende mantener:



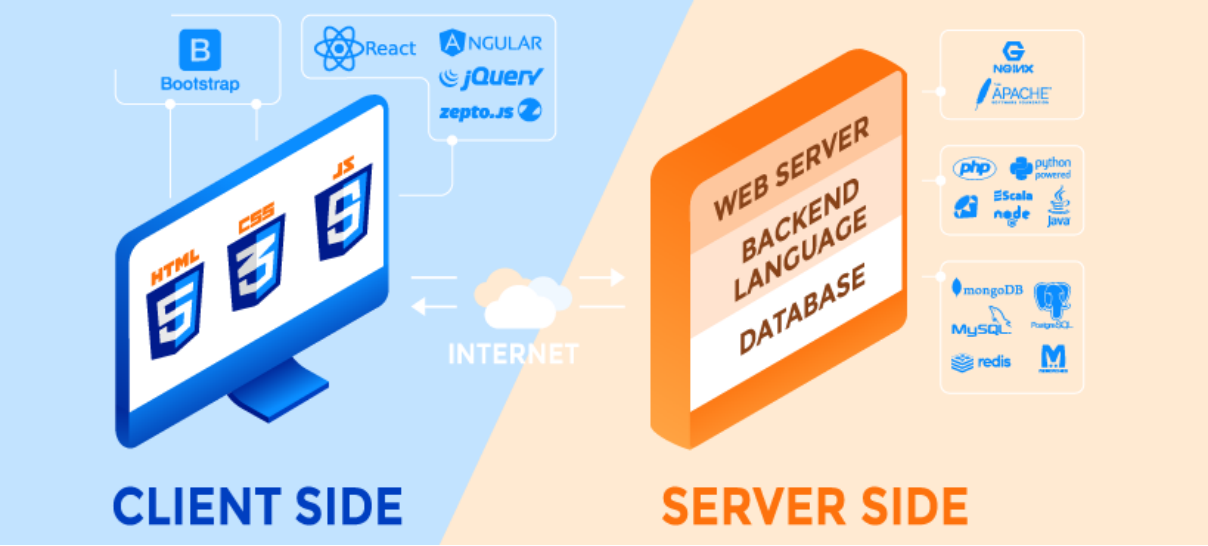
I.2 MODELO WEB CLIENTE-SERVIDOR

Podemos definir una **aplicación web** como *aquella aplicación que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador* (Cliente). Esto la diferencia de las **aplicaciones de escritorio**, escritas en lenguajes como C o C++ para ser usadas en un ordenador y orientadas a los recursos que nos ofrece este. Un ejemplo de aplicación web sería el que nos encontramos en un terminal de reserva de billetes un tren de cercanías o aquella orientada a la *Internet of things*.

Ya estemos ante una aplicación o un sitio web, se va a seguir un modelo basado en la programación cliente-servidor con tres elementos comunes:

- **El lado del cliente** (*client-side*): este elemento hace referencia a los navegadores web y está soportado por tecnologías como HTML, CSS, lenguajes como JavaScript y controles ActiveX, los cuales se utilizan para crear la presentación de la página o proporcionar características interactivas. Es aquí donde tiene lugar la parte del software que interactúa con los usuarios o **front-end**.

- **El lado del servidor** (*server-side*): incluye el hardware y software del servidor Web así como diferentes elementos de programación y tecnologías incrustadas incluyendo tecnologías de *servidor de bases de datos* que soporten múltiples sitios web. Esta es la parte que procesa la entrada de datos que previamente se realizó en el front-end, y que es conocida como **back-end**. Después de dicho procesamiento construirá y enviará una página de respuesta al cliente.



- **La red**: describe los diferentes elementos de conectividad (definida esta como *la capacidad que tiene un dispositivo para poder conectarse a otros*). Aquí se detallan los diferentes protocolos y material utilizado para poder realizar dicha conexión mostrando el sitio web al usuario.

> Al lado del cliente o Client-Side también se le conoce como front-end, aunque estos dos términos no significan exactamente lo mismo. El Client-Side hace referencia únicamente al lugar en el que se ejecutan los procesos, mientras que el front-end hace referencia a los tipos de procesos que se ejecutan en el lado del cliente. Lo mismo puede aplicarse al Server-Side y al back-end pero en el servidor.

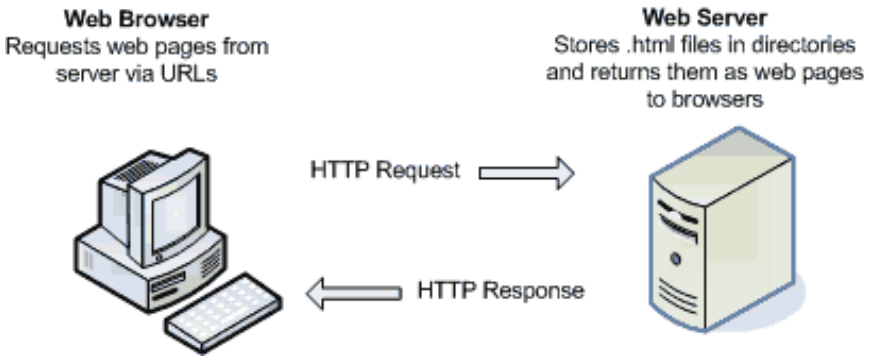
I.2.1 Funcionamiento del modelo web Cliente-servidor

Supongamos que estamos con nuestro smartphone, y consultamos una web de viajes turísticos desde el navegador Chrome. Veamos el camino que hacen los *paquetes IP*:

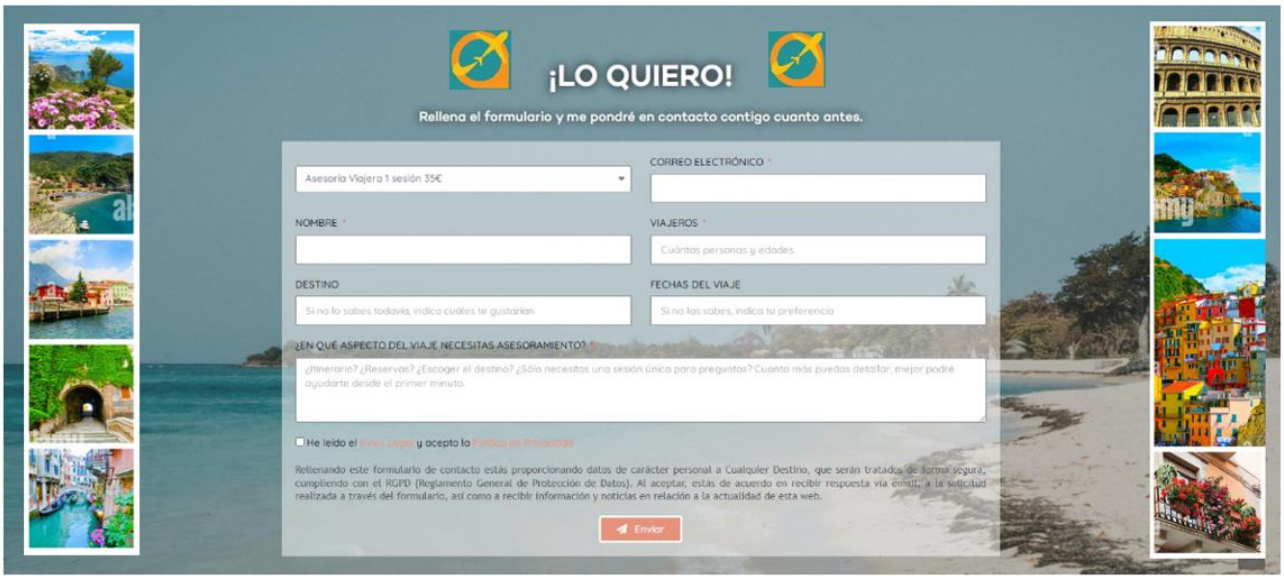
1. Escribimos la dirección del sitio web, p.e. www.travelplanner.com, en la barra del navegador. Esto da lugar a una solicitud a los servidores DNS para la dirección IP de *travelplanner.com*, y los servidores DNS responden a esta solicitud sirviendo la dirección IP al navegador.

2. A continuación, usando la dirección IP, el navegador del usuario realiza una solicitud a los servidores del sitio web (p.e. un Apache Server que aloje el script *index.php*) para el contenido que aparece en la página, como imágenes en miniatura de pueblos o ciudades, el logotipo de *travelplanner* y la barra de búsqueda de destino turístico.

3. Este script se ejecuta (se procesa la parte de PHP y el resultado se junta con la parte de HTML), y finalmente el resultado (el sitio web) se sirve al cliente haciendo el camino inverso.



4. Así, los servidores de *travelplanner* lo entregan al navegador y este se dispone a cargar la página en el dispositivo del cliente. Por ejemplo:



5. En este momento, el HTML, CSS y JavaScript que dictan cómo aparece la página principal de *travelplanner* para el usuario son interpretados por el navegador en el lado del cliente para mostrar la página al usuario.

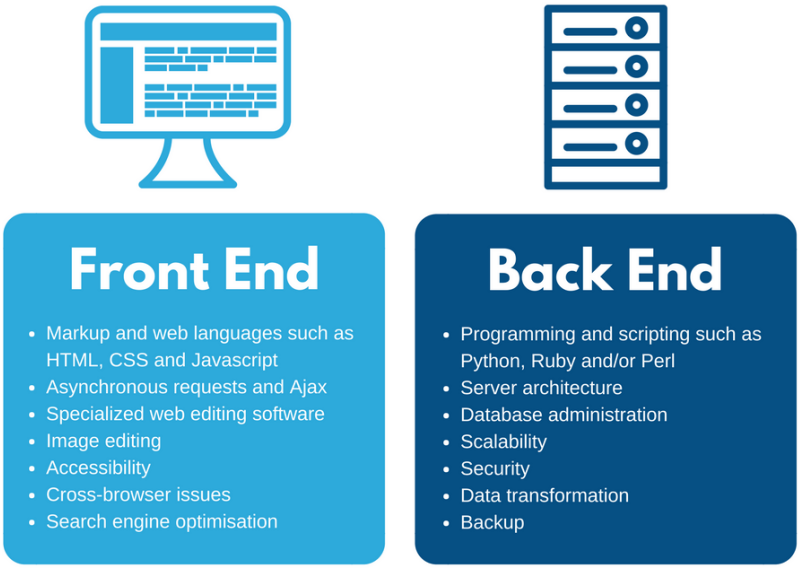
6. Dicha página también puede responder a "eventos": por ejemplo, si el ratón del usuario pasa por encima de una de las imágenes en miniatura de los destinos turísticos, la imagen se expande y las miniaturas adyacentes se mueven ligeramente hacia un lado para dejar espacio a la imagen más grande. Este es un ejemplo de proceso del lado del cliente; el código en la propia página web responde al ratón del usuario e inicia esta acción sin comunicarse con el servidor.

I.3 TÉCNICO EN PROGRAMACIÓN FRONT-END Y BACK-END

La separación del sistema en **front-end** y **back-end** es una abstracción que ayuda a mantener las diferentes partes del sistema cliente-servidor separadas, algo deseable en cualquier desarrollo software:

- **Back-end**. Parte no visible de la web donde encontramos las bases de datos o los scripts que se ejecutan en el servidor. Los técnicos de back-end se encargan del todo el proceso de desarrollo software en el server-side, como el acceso a la base de datos (*MySQL*, *MaríaDB*, *PostgreSQL*, *MongoDB*, Oracle, etc.), creación de servicios, etc. Sus técnicos programarán en lenguajes como *PHP*, *Ruby on Rails*, *Django*, *Node.js*, *.NET*, etc.

  Sus objetivos son: un acceso rápido a los datos, una comunicación eficiente con la base de datos y el navegador, control de la seguridad, etc.



- **Front-end**. Parte visible de una web, como las hojas de estilo, el código HTML, los scripts que se ejecutan en el lado del cliente, etc.). Aquí trabajan los diseñadores de la aplicación web y los programadores; los primeros están encargados de determinar lo que se conoce como *experiencia de usuario (UX)* así como el diseño visual de la aplicación, mientras que los segundos serán expertos en los lenguajes *HTML*, *CSS* y *JavaScript*.

  Como objetivos de esta parte del desarrollo: optimizar la presentación y conseguir una interfaz final lo más agradable y funcional posible para los usuarios (*user-friendly interface*).

II LENGUAJES DE PROGRAMACIÓN EN CLIENTES WEB

Uno de los objetivos en la programación web es saber escoger la tecnología correcta para nuestro trabajo. Muchas veces los desarrolladores escogen rápidamente una tecnología favorita, que puede ser *JavaScript*, *ColdFusion* o *PHP* y la usan en todas las situaciones.

La realidad es que cada tecnología tiene sus pros y sus contras. En general las tecnologías client-side y server-side poseen características que las hacen complementarias más que adversarias.

Por ejemplo, cuando añadimos un formulario para recoger información y grabarla en una base de datos, es obvio que tendría más sentido, justo antes de enviar la información a la base de datos del servidor, chequear el formulario en el lado del cliente para asegurarnos que la información introducida es correcta. La programación en el lado del cliente consigue que la validación del formulario sea mucho más efectiva y que el usuario se sienta menos frustrado al cubrir los datos en el formulario. Por otro lado, el almacenar los datos en el servidor estaría mucho mejor gestionado por una tecnología del lado del servidor con un rápido acceso a la base de datos.

Cada tipo general de programación tiene su propio lugar y la mezcla es generalmente la mejor solución.

Cuando hablamos de lenguajes de programación en clientes web, podemos distinguir dos variantes:

- Lenguajes que nos permiten dar formato y estilo a una página web (HTML, CSS)
- Lenguajes que nos permite aportar dinamismo a páginas web (lenguajes de scripting).

En este módulo nos centraremos principalmente en estos últimos, y en particular en el lenguaje *JavaScript* que será el lenguaje que utilizaremos a lo largo de todo este módulo formativo. Y es que *JavaScript* es el lenguaje de script más utilizado en la programación en el lado del cliente, y está soportado mayoritariamente por todas las plataformas o sistemas operativos.

Definimos *Lenguaje de Script* como aquél lenguaje de guiones o de órdenes que se almacena por lo general en archivos de texto plano y que será ejecutado línea a línea por un programa intérprete que, en nuestro caso será el navegador.

Una *página web dinámica* es aquella que no muestra el mismo contenido para todos los usuarios y cambia en función de las entradas de este. Por ejemplo, la página de inicio de sesión de Facebook es, en su mayor parte, estática; pero la que nos encontramos luego de entrar es dinámica (adaptada a cada usuario).

III …Y APARECIÓ JAVASCRIPT

III.1 DESARROLLO

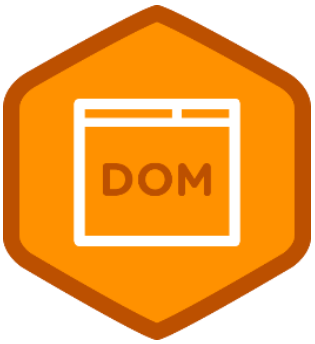
Fue en 1995 cuando JavaScript fue desarrollado por *Brendan Eich* (actualmente CEO de Mozilla) para el navegador *Netscape Navigator* con el nombre de ***Mocha***. Más tarde, se renombró a ***LiveScript*** quedando finalmente como *JavaScript*.

El nombre de JavaScript se debió a que Netscape añadió compatibilidad con Java en su navegador (en este tiempo Java empezó a hacerse tremendamente popular). Además, Netscape fue adquirida por *Sun Microsystems*, propietaria de la marca Java.

En este sentido se suele dar la confusión de pensar que Java y JavaScript son lo mismo o considerar a este último una extensión del primero, lo cual no es cierto. JavaScript se diseñó con una sintaxis similar al *lenguaje C* y aunque adopta nombres y convenciones del lenguaje *Java*, éste último no tiene relación con *JavaScript* ya que tienen semánticas y propósitos diferentes.

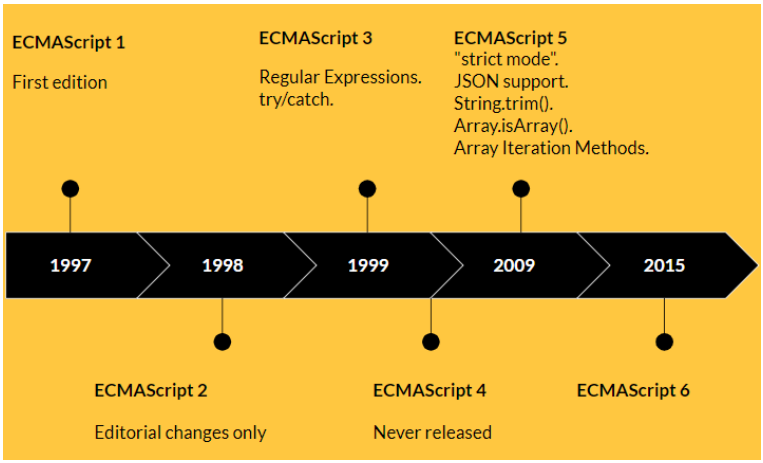
Por aquél entonces Microsoft ya estaba usando su propio JavaScript, al que llamo *JScript* para evitar problemas relacionados con la marca, pero no pudo evitar otros problemas surgidos por las incompatibilidades que su versión de JavaScript tenía con otros navegadores diferentes al Explorer.

Para evitar esas incompatibilidades, en 1997 el W3C promovió la creación del comité **TC39** por la **ECMA** (*European Computer Manufacturers Association*), el cual se encargará de gestionar las especificaciones de este lenguaje de script (da igual el nombre que reciba) en el documento ECMA-262. Desde este momento, tanto *JavaScript* como *JScript* deberán ser compatibles con dicho documento.



A raíz de esto se diseñará el estándar del DOM (*Document Object Model*) que incorporaron los browsers *Internet Explorer 6*, *Netscape Navigator*, *Opera 7* y *Mozilla Firefox* desde su primera versión para, de esta manera, evitar incompatibilidades entre los navegadores.

Es a partir de entonces cuando los estándares de JavaScript se regirán por el ECMAScript. En 1999 se estandariza la versión 3 de JavaScript que se mantuvo vigente hasta hace relativamente poco.



Hubo algunos intentos de lanzar una versión 4, pero la que finalmente se estandarizó hasta ahora es la versión 5 de ECMAScript aprobada en 2011 (respetado incluso por Microsoft con su *JScript*).

Finalmente, en el año 2015 apareció la norma ECMAScript6 (conocida como ES6 o ECMAScript 2015 o ES2015) que mejoró mucho el código JavaScript al dotarlo de elementos avanzados de otros lenguajes.

Han aparecido, hasta el momento de escribir estos apuntes, las versiones siete (o ECMAScript 2016), ocho (o ECMAScript 2017), nueve (o ECMAScript 2018) y diez (o ECMAScript 2019) y acaba de ser liberada la 11 o ECMAScript 2020, todos ellos con ligeros cambios sobre el ES6 (https://es.wikipedia.org/wiki/ECMAScript) y no totalmente soportados por todos los navegadores.

Queda ya poco de la idea original de JavaScript como un lenguaje para añadir efectos y animaciones a los sitios web; desde que en 2005 apareció Gmail y su uso de la tecnología AJAX, la popularidad de JavaScript no ha dejado de crecer haciendo que este evolucione hasta convertirse en un lenguaje multipropósito.

III.2 USO DE JAVASCRIPT

JavaScript está orientado a ofrecer las siguientes soluciones al desarrollador:

- ✓ Conseguir que nuestra página web responda o reaccione directamente (eventos) a la interacción del usuario con elementos de formulario y enlaces hipertexto.
- ✓ Validación de datos introducidos por el usuario en formularios.
- ✓ Modificación de la estructura de un documento HTML según datos introducidos por el usuario.
- ✓ Controlar múltiples ventanas o marcos de navegación, plug-ins, o applets de Java basados en las elecciones que ha hecho el usuario en el documento HTML.
- ✓ Preprocesar datos en el cliente antes de enviarlos al servidor.
- ✓ Modificar estilos y contenido en los navegadores de forma dinámica e instantáneamente, en respuesta a interacciones del usuario.
- ✓ Solicitar ficheros del servidor, y enviar peticiones de lectura y escritura a los lenguajes de servidor.
- ✓ Realizar aplicaciones en el Server-Side gracias al entorno de ejecución *Node.js*
- ✓ Interacción con bases de datos no relacionales *MongoDB* gracias a las librerías *Mongoose* y *Expresjs*.
- ✓ Realización de interfaces gráficas gracias a librerías como ***react.js***

- ✓ Realización eficiente de aplicaciones web gracias a la gran cantidad de APIs a disposición del desarrollador.
- ✓ Gestión de operaciones asíncronas en nuestra página gracias a AJAX.

Para concluir, decir que hoy en día JavaScript se utiliza en múltiples entornos: *Frontend*, *Backend*, aplicaciones isomórficas (aquella que comparte todo (o casi todo) su código entre el cliente y el servidor), microcontroladores, *Internet of Things*, *apps wearables*, etc... convirtiéndose en el lenguaje de programación del presente.

Toda la documentación y referencia sobre JavaScript se puede encontrar en el sitio web de desarrolladores de Mozilla (https://developer.mozilla.org/es/docs/Web/JavaScript), muy recomendable de visitar cuando se tienen dudas sobre cómo se usa o implementa una función u objeto determinado.

Actualmente JavaScript es una marca registrada de *Oracle Corporation*, y es usado con licencia por los productos creados por *Netscape Communications* y entidades actuales, como la *Mozilla Fundation*.

III.3 CARACTERÍSTICAS DE JAVASCRIPT

III.3.1 Compatibilidades

JavaScript es un lenguaje interpretado por el navegador, así que dependerá de este para su correcto funcionamiento. Actualmente la gran mayoría de clientes o navegadores soportan JavaScript, incluyendo *Firefox*, *Google Chrome*, *Safari*, *Opera*, *Internet Explorer*, etc. Aun así, cuando escribimos un script en nuestra página web, tenemos que estar seguros de que será interpretado por diferentes navegadores y que aportará la misma funcionalidad y características en cada uno de ellos.

Cada tipo de navegador da soporte a diferentes características del JavaScript y además también añaden sus propios **bugs** o **fallos**. Algunos de estos fallos son específicos de la plataforma sobre la que se ejecuta ese navegador, mientras que otros son específicos del propio navegador en sí.

Para saber si una etiqueta, atributo, valor o palabra reservada es soportada por nuestro navegador o por otros, podremos recurrir a la web *http://www.caniuse.com. Esta página incluso nos recomendará si debemos seguir usando o no alguna palabra reservada.*

A veces las incompatibilidades entre navegadores al interpretar el código de JavaScript no vienen dadas por el propio código en sí, sino que su origen proviene del código fuente HTML. Por lo tanto, es muy importante que tu código HTML siga las especificaciones del estándar W3C y para ello dispones de herramientas como el validador HTML W3C:

http://validator.w3.org/

Además, debemos tener precaución con los siguientes problemas que podemos tener en el uso de JavaScript:

- ✓ No todos los navegadores soportan lenguajes de script (en especial JavaScript) en el lado del cliente.
- ✓ Algunos dispositivos móviles tampoco podrán ejecutar JavaScript.
- ✓ Incluso las implementaciones más importantes de JavaScript en los diferentes navegadores no son totalmente compatibles entre ellas, como ejemplo tenemos las diferentes incompatibilidades entre Firefox e Internet Explorer.
- ✓ La ejecución de código JavaScript en el cliente podría ser desactivada por el usuario de forma manual, con lo que no podremos tener una confianza ciega en que se vaya a ejecutar siempre tu código de JavaScript.
- ✓ Algunos navegadores con accesibilidad por voz, no interpretan el código de JavaScript.

III.3.2 Agujeros de Seguridad

Lamentablemente, JavaScript proporciona un gran potencial para diseñadores maliciosos que quieran distribuir sus scripts a través de la web. Para evitar esto, los navegadores web cliente aplican dos tipos de restricciones:

- ✓ Por razones de seguridad cuando se ejecuta código JavaScript éste lo hace en un **espacio seguro de ejecución** (*sandbox*) en el cual solo podrá realizar tareas relacionadas con la web, nada de tareas genéricas de programación como creación de ficheros, etc.

- ✓ Además, los scripts están restringidos por la política de **mismo origen**: la cual quiere decir que los scripts de una web no tendrán acceso a información enviada desde otra web, tal como usuarios, contraseñas, o cookies.

La mayor parte de los agujeros de seguridad son infracciones tanto de la política de "mismo origen" como de la política de "espacio seguro de ejecución".

III.3.3 Limitaciones de JavaScript

Es importante entender las limitaciones que tiene JavaScript y que, en parte, refuerzan sus capacidades de seguridad. Así, no podrá realizar ninguna de las siguientes tareas:

- Modificar o acceder a las preferencias del navegador del cliente, las características de apariencia de la ventana principal de navegación (aún se permite, pero cada vez es menos permitido por los navegadores), capacidades de impresión, botones de acciones del navegador …
- Lanzar la ejecución de una aplicación en el ordenador del cliente.
- Leer o escribir ficheros en el ordenador del cliente (con la excepción de las *cookies*).
- Escribir directamente ficheros en el servidor.
- Capturar los datos procedentes de una transmisión en streaming de un servidor, para su retransmisión.

- Enviar e-mails a nosotros mismos de forma invisible sobre los visitantes a nuestra página web (aunque sí que podría enviar datos a una aplicación en el lado del servidor capaz de enviar correos).
- Interactuar directamente con los lenguajes de servidor.
- Las páginas web almacenadas en diferentes dominios no pueden ser accesibles por JavaScript (política del mismo origen visto en el apartado anterior).
- JavaScript es incapaz de proteger el origen de las imágenes de nuestra página.
- Implementar multiprocesamiento o multitarea.

III.3.4 Motor de ejecución

El motor de ejecución (o intérprete) de JavaScript es el encargado de ejecutar el código JavaScript en el navegador y, por lo tanto, es en él dónde recaerá la implementación de la seguridad. Entre otros, podemos citar varios ejemplos de motores de *JavaScript*:

- ✓ *ActiveScript* de Microsoft: tecnología que soporta *JScript* como lenguaje de scripting. A menudo se considera compatible con JavaScript, pero Microsoft emplea múltiples características que no siguen los estándares *ECMA* (aunque deberían).

- ✓ El kit de herramientas *Qt C++* también incluye un módulo intérprete de JavaScript.

- ✓ El lenguaje de programación Java en su versión *JDK 1.6* introduce un paquete denominado *javax.script* que permite la ejecución de JavaScript.

- ✓ Y por supuesto todos los motores implementados por los navegadores web como *Mozilla*, *Google*, *Opera*, *Safari*, etc. Cada uno de ellos da soporte a alguna de las diferentes versiones de JavaScript.

Hoy en día las características que más resaltan y que permiten diferenciar a unos navegadores de otros son:

- ✓ La rapidez con la que sus motores de JavaScript pueden ejecutar las aplicaciones.
- ✓ La seguridad y aislamiento que ofrecen en la ejecución de las aplicaciones en diferentes ventanas o pestañas de navegación.

IV BIBLIOGRAFÍA CONSULTADA Y RECOMENDADA

- *Desarrollo web en entorno cliente.* Juan Carlos Moreno Pérez, Edit. Síntesis
- *Desarrollo web en entorno cliente con JavaScript.* Jorge Sánchez Asenjo, Edit. Garceta
- *https://platzi.com/* (Blogs)
- *https://developer.mozilla.org/es/docs/Web/JavaScript*
- *https://www.w3schools.com/*
- *http://es.wikipedia.org*