

Despliegue de aplicaciones web

0. Introducción a las aplicaciones web

Nacho Iborra

IES San Vicente



Esta obra está licenciada bajo la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Para ver una copia de esta licencia, visita <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Índice de contenidos

Despliegue de aplicaciones web.....	1
1. Tipos de aplicaciones.....	3
1.1. ¿Qué es una aplicación web?.....	3
2. Arquitectura de una aplicación web.....	4
2.1. ¿Qué es “la web”?.....	4
2.2. Elementos de una aplicación web.....	4
2.3. Funcionamiento de una aplicación web.....	4
2.4. URLs y dominios.....	5
3. Aplicaciones web vs escritorio.....	7
3.1. Ventajas de las aplicaciones web.....	7
3.2. Inconvenientes de las aplicaciones web.....	8
4. Protocolos más utilizados.....	9
4.1. Más sobre HTTP/HTTPS.....	9
5. Patrones de diseño software.....	12
5.1. El patrón MVC.....	12
5.2. Otras alternativas al MVC.....	13
6. Recursos necesarios.....	15
6.1. Lenguajes.....	15
6.2. Ejemplos de software.....	16

1. Tipos de aplicaciones

Cuando estamos utilizando un ordenador, una tablet o un teléfono móvil, ¿qué tipos de aplicaciones o programas podemos estar utilizando? Básicamente distinguimos dos grandes grupos:

- Aquellas aplicaciones que no necesitan ninguna conexión a Internet o a una red de ordenadores para funcionar. Este tipo de aplicaciones suelen llamarse **aplicaciones de escritorio**, y podemos encontrar ejemplos muy variados: un procesador de textos, un lector de libros electrónicos, un reproductor de música o vídeo, e incluso videojuegos que tengamos instalados.
- Aquellas aplicaciones que sí necesitan conexión, bien sea a Internet o a un ordenador de su red local. En este otro grupo también tenemos ejemplos variados de aplicaciones. Por ejemplo, si compartimos un documento de texto en Google Drive, o si abrimos el navegador para acceder a una plataforma de un curso online, o incluso si jugamos a videojuegos junto con otras personas de otros lugares. Aquí distinguimos varios subtipos de aplicaciones. Algunos de los más habituales son:
 - Las **aplicaciones P2P** (*peer-to-peer*), donde todos los elementos conectados a la red tienen el mismo "rango", por así decirlo, y comparten información entre ellos. Es el mecanismo en el que se basan varios programas de descarga, como los de archivos tipo *torrent*.
 - Las **aplicaciones cliente-servidor**, llamadas así porque consisten en que un conjunto de ordenadores (llamados *clientes*) se conectan a uno central (llamado *servidor*) que es el que les proporciona la información y los servicios que solicitan. En el caso de videojuegos donde nos conectamos a otro lugar para jugar con otras personas, estamos utilizando aplicaciones cliente-servidor, donde nuestro ordenador (uno de los clientes), tiene instalada una parte de la aplicación y el servidor al que se conecta le proporciona la información de los escenarios y del resto de jugadores y personajes.
 - Dentro del tipo de aplicaciones cliente-servidor, las **aplicaciones web** son un subtipo, quizá el más numeroso. Es en este subtipo en el que nos vamos a centrar.

1.1. ¿Qué es una aplicación web?

Podemos encontrar diversas definiciones de aplicación web buscando en Internet. Una de las más habituales hace referencia a aplicaciones que se cargan o ejecutan desde un navegador web, accediendo a un servidor.

Sin embargo, el avance experimentado en este sector durante los últimos años hace que podamos definir un concepto más amplio. Así, una aplicación web sería aquella realizada a partir de lenguajes y tecnologías de desarrollo web, tales como HTML, CSS, Javascript... Pueden ejecutarse en un navegador convencional, o un motor de navegador embebido en otro sistema (como el webview de Android o iOS, por ejemplo). De este modo, además de las aplicaciones web "tradicionales", también tendrían cabida las llamadas *aplicaciones híbridas* (desarrolladas con tecnologías web pero exportadas a formato nativo de diversos dispositivos), y aplicaciones de escritorio que emplean tecnologías web, como por ejemplo el framework Electron de Javascript.

2. Arquitectura de una aplicación web

2.1. ¿Qué es “la web”?

Podemos ver la web como una especie de plataforma mundial donde tenemos disponibles gran cantidad de recursos (documentos, videojuegos, redes sociales, foros, etc.). Se hizo popular a principios de los años 90 gracias a aplicaciones como el correo electrónico, los chats, etc. y con la aparición de la web 2.0 vinieron otra serie de aplicaciones que la potenciaron aún más, como los blogs o las redes sociales. Poco a poco se han ido añadiendo funcionalidades, hasta el punto de que hace pocos años era impensable poder ver vídeos o películas en Internet, y hoy es algo muy habitual.

2.2. Elementos de una aplicación web

En una aplicación web podemos distinguir en primer lugar dos grandes lados: el **cliente**, donde está el usuario, que utiliza un navegador web (Google Chrome, Firefox, etc.) para acceder a la aplicación, y el **servidor**, donde está ubicada la aplicación (el foro, la red social, el blog, el curso online, etc.), y que se encarga de atender las peticiones de los clientes y proporcionarles la información que solicitan.

2.3. Funcionamiento de una aplicación web

Como hemos comentado, las aplicaciones web son un tipo de aplicaciones cliente-servidor. Este tipo de arquitecturas distribuyen las tareas entre quienes prestan los recursos y servicios (los servidores) y quienes los solicitan (los clientes).

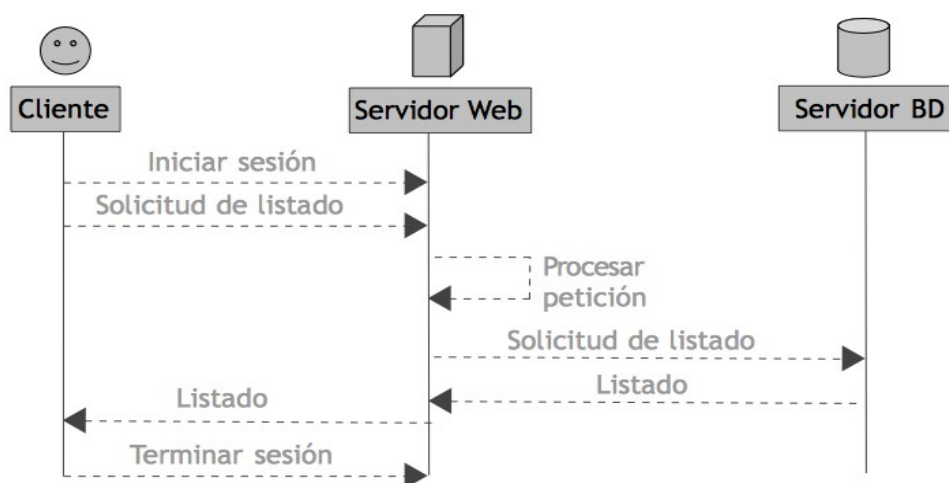
Los pasos que se siguen en la comunicación cliente-servidor, son, en general:

1. El cliente inicia sesión en el servidor
2. El cliente solicita al servidor el recurso o servicio que quiere utilizar (una página web, un documento, subir información, etc.)
3. El servidor recibe la respuesta del cliente, la procesa y decide qué programa debe darle servicio, enviando la petición a dicho programa.
4. El programa responsable procesa la petición, prepara la respuesta y la entrega al servidor.
5. El servidor envía la respuesta al cliente
6. El cliente puede volver al paso 2 y realizar una nueva petición, o bien
7. El cliente termina la sesión en el servidor.

En general, el servidor no tiene por qué ejecutarse solo, sino que podemos tener diferentes aplicaciones en diferentes equipos (o en el mismo), lo que se conoce como arquitectura **multicapa o multinivel**. Por ejemplo, un servidor de bases de datos en una máquina, un servidor web en otra (o en la misma que el de bases de datos), un servidor de correo electrónico... y así distribuir los procesos y el trabajo a realizar, e incluso configurar opciones de seguridad y rendimiento separadas para cada servidor.

2.3.1. Ejemplo: arquitectura de dos o tres niveles

Por ejemplo, si el cliente conectara con el servidor para pedir un listado de noticias almacenadas en una base de datos, expresado como un diagrama de secuencia, el funcionamiento básico de esta petición (y de la arquitectura cliente-servidor en general) puede verse como algo así:



En este ejemplo, el servidor web y el servidor de bases de datos podrían estar instalados en la misma máquina o en máquinas separadas, cada una con su hardware específico y control de acceso de usuarios específico. En cualquier caso, estamos hablando de una arquitectura de **tres niveles** (cliente, servidor web y servidor de base de datos), que es algo bastante habitual en las aplicaciones web, pues casi todas cuentan con una base de datos con información que consultar y modificar.

Sin el servidor de base de datos, estaríamos ante una arquitectura de **dos niveles**, donde el servidor es polivalente, y puede responder directamente a las peticiones de los clientes sin consultar con otros servidores o aplicaciones. Esta opción es menos flexible, menos segura y puede ofrecer peor rendimiento en sistemas congestionados, al no poder dividir el trabajo entre distintos tipos de servidores.

2.4. URLs y dominios

2.4.1. Ubicación de los servidores. Hostings y nombres de dominio

El servidor es el componente de una aplicación web que se encarga de recibir peticiones de todos los clientes que se conecten a él y enviarles la información que solicitan. Para poder hacer esto, el servidor debe estar accesible en un lugar conocido, para que los usuarios puedan conectarse a él. Por ejemplo, cuando escribimos la dirección www.iessanvicente.com, de alguna forma hay "algo" en Internet que sabe dónde está el servidor para la web del IES San Vicente, y envía la petición allí.

En primer lugar, debemos localizar nuestro servidor en Internet. Esto puede hacerse de varias formas. Por ejemplo:

- Disponer de un servidor (o servidores) propios y una dirección IP pública fija a la que acceder. Esta opción es poco habitual hoy en día
- Contratando un espacio (o una máquina entera, si el proyecto es grande y se dispone de presupuesto suficiente) en una empresa de alojamiento o **hosting**. Esta

opción es mucho más habitual, y de ahí que proliferen las empresas que se dedican al *hosting*, tales como OVH, Hostinger, Hostalia, 1&1, etc.

Además, debemos reservar (comprar) un **nombre de dominio** para nuestra empresa o web. El nombre de dominio es lo que escribimos en la barra del navegador, justo después de *www*. En el caso de la web del instituto, dicho dominio es *iessanvicente.com*. El precio de mantener dicho dominio puede variar, pero ronda los 10 o 15 euros al año.

2.4.2. Acceder a los servidores. URLs y DNS

Hemos visto que, en el esquema de funcionamiento de una aplicación web, el cliente solicita recursos al servidor. La forma en que los solicita es mediante **URLs**. Una URL (*Uniform Resource Locator*) es una manera de identificar y localizar cada recurso de una web. Por ejemplo, cuando escribimos en un navegador una dirección como *http://www.miweb.com/paginas/pagina.html*, estamos introduciendo una URL para localizar un recurso (en este caso, una página HTML). Una URL se compone de:

- El **protocolo**, que indica las reglas que se van a seguir para comunicarse cliente y servidor. Veremos más adelante algunos ejemplos de protocolos, pero para lo que nos importa, en una URL el protocolo va al principio, hasta los dos puntos y el delimitador //. En nuestro ejemplo, el protocolo sería *http://*
- El **nombre de dominio**, que ya hemos explicado anteriormente. Identifica al servidor y la empresa/web a la que vamos a conectar. Va justo detrás del protocolo, hasta la siguiente barra. Normalmente termina en *.com*, *.es*, *.net*, etc. En nuestro ejemplo sería *www.miweb.com*
- La **ruta hacia el recurso**, que comprende todas las carpetas y subcarpetas (si las hay) y el nombre de archivo que queremos obtener. En nuestro ejemplo, la ruta sería */paginas/pagina.html*

El navegador obtiene la URL que ha escrito el usuario, y transforma el nombre de dominio en una dirección IP gracias al **servicio DNS** (*Domain Name System*, sistema de nombres de dominio). Este servicio se encarga de traducir los nombres de dominio en direcciones IP, que son las que utilizan los routers para saber dónde encaminar los mensajes. Cada servidor en Internet tiene asignada una (o varias) direcciones IP, y existen diversos servidores DNS disponibles que se van "repartiendo" qué IP corresponde a cada dominio, para saber dónde enviar las peticiones.

Podemos comprobar qué dirección IP tiene asignada un determinado dominio mediante comandos de terminal como *nslookup* o *ping*.

- Si escribimos *nslookup www.iessanvicente.com* obtendremos la dirección IP asociada a la web del IES San Vicente
- Si escribimos *ping www.iessanvicente.com* intentaremos comunicarnos con dicho servidor, y además, en el terminal nos mostrará la dirección IP con la que está intentando comunicar (que coincidirá con la obtenida con *nslookup*).

3. Aplicaciones web vs escritorio

Una vez hemos visto qué arquitectura tienen las aplicaciones web, ¿qué ventajas e inconvenientes podemos ver que tienen frente al otro gran grupo de aplicaciones, las de escritorio?

3.1. Ventajas de las aplicaciones web

- Una de las principales ventajas que tienen es que el cliente no tiene que **instalar** nada en su equipo, o casi nada. Basta con tener instalado un navegador web, y como mucho en algunos casos, algún plugin especial (Flash, Java, etc.). El servidor es el que necesita tener instalado el software para hacer funcionar la aplicación (servidor web, base de datos, archivos necesarios, etc.)
- Otra ventaja es que el cliente apenas tiene **carga de trabajo**. El servidor lleva gran parte del peso del procesamiento (almacenar datos, enviar archivos, etc.). Esto en ocasiones puede suponer una desventaja, si el servidor se satura. Veremos que hay mecanismos para que el cliente haga parte del trabajo y libere así al servidor, con lenguajes como Javascript, y tecnologías como AJAX.
- Además, para utilizar una misma aplicación varias personas, no necesitamos instalarla en todos los equipos donde se vaya a utilizar, ya que todos se conectarán con el navegador al servidor web. Por otra parte, si hacemos un cambio en la aplicación (por ejemplo, cambiamos los colores de los menús, o añadimos alguna funcionalidad más), se hará en el servidor y automáticamente lo verán actualizado todos los usuarios. Esto hace que las aplicaciones web sean más fáciles de **mantener y actualizar**
- La **compatibilidad** es otro factor importante. Muchas aplicaciones de escritorio sólo funcionan para ciertos sistemas operativos, o funcionan diferente dependiendo del sistema. Una aplicación web, en general, funciona de la misma forma independientemente de la plataforma. Eso sí, para dispositivos móviles suelen tener una versión adaptada, sin tanto texto y con funcionalidades más reducidas o localizadas, para poderse manejar mejor en una pantalla pequeña.
- El **control de usuarios y la seguridad** también están centralizados en el servidor, de forma que desde él podemos ver y dar permisos a quien intenta acceder. En una aplicación de escritorio es más difícil controlar que no se entre de forma ilegal en alguno de los equipos donde esté instalada.
- La **movilidad**, ya que al estar la aplicación instalada en un servidor remoto, si disponemos de conexión a Internet podemos utilizar la aplicación desde cualquier lugar.
- La **escalabilidad** del sistema, ya que la mayor parte del trabajo recae en el servidor, se pueden siempre mejorar sus prestaciones y ampliar la capacidad de número de clientes mejorando el hardware, añadiendo más servidores, etc.

3.2. Inconvenientes de las aplicaciones web

Entre los inconvenientes de las aplicaciones web, existen algunos que no lo son tanto, pero los comentamos a continuación porque han sido durante mucho tiempo puntos flacos de estas aplicaciones, hasta que los recientes avances han conseguido anularlos:

- Uno de los inconvenientes que podemos encontrar en una aplicación web es su **riqueza gráfica**, si la comparamos con aplicaciones de escritorio. Existen algunos tipos de efectos que pueden parecer difíciles de conseguir en una aplicación web, tales como animaciones, o efectos 3D. Sin embargo, poco a poco han ido apareciendo herramientas para paliar esto, como Flash (ya obsoleto), librerías Javascript, CSS3, etc, y hoy en día no es tanta la diferencia entre una aplicación de escritorio y una web en este aspecto. Incluso en muchos casos resulta más sencillo implementar algunos efectos desde tecnologías web.
- Otro inconveniente evidente es la **necesidad de conexión** a la red para poder utilizar la aplicación, aunque algunas aplicaciones como Dropbox o Google Drive permiten trabajar con un documento sin conexión y actualizar los cambios en el servidor cuando exista conexión. También existen otros mecanismos como LocalStorage o SQLite para que los desarrolladores puedan almacenar de forma local los datos hasta encontrar conexión para enviarlos al servidor.

No obstante, sí existen otros inconvenientes evidentes hoy en día, que (de momento) no han conseguido paliarse:

- El **rendimiento** es un hándicap importante aún en las aplicaciones web. Incluso una aplicación híbrida para Android, por ejemplo, ofrecerá un rendimiento menor que la misma aplicación desarrollada de forma nativa con, por ejemplo, Android Studio. Sin embargo, la facilidad que ofrecen los frameworks de aplicaciones híbridas puede compensar el tiempo de desarrollo y la curva de aprendizaje necesarias para dominar el desarrollo en nativo.
 - Relacionado con este factor, otro inconveniente es el **tiempo de respuesta**. Al ser aplicaciones cliente/servidor, puede pasar un tiempo considerable desde que, por ejemplo, enviamos un formulario hasta que el servidor nos avise de que los datos se han guardado correctamente. Estos tiempos son mucho menores en aplicaciones de escritorio, al no necesitar conexión.
- El **tráfico generado** en la red para acceder al servidor también puede ser un factor importante a considerar. Si hay varios clientes accediendo a la aplicación y solicitando datos, la cantidad de información que se envía por Internet o que se solicita al servidor puede llegar a colapsarlo, y que deje de dar servicio a todos los clientes hasta que se recupere.
 - Esto no es un problema en otro tipo de arquitecturas, como por ejemplo las redes P2P, donde el número de clientes mejora el rendimiento del sistema, al haber más lugares desde donde compartir y cargar/descargar los recursos.
- También podemos citar entre los inconvenientes la **apariencia**. A pesar de que las aplicaciones web son más compatibles, su apariencia final depende del navegador que utilice el cliente para visualizarlas. Así, la misma aplicación puede verse de forma diferente según si estamos utilizando Chrome, Internet Explorer, Firefox, etc. Conseguir una homogeneidad en dicha apariencia es una de las tareas más tediosas a las que se enfrentan los diseñadores web.

4. Protocolos más utilizados

A la hora de comunicar clientes y servidores, es necesario establecer un **protocolo** de comunicación, es decir, una serie de reglas que indiquen qué tipo de mensajes se van a intercambiar, en qué orden y qué contenido va a tener cada tipo de mensaje, de forma que los dos extremos de la comunicación (cliente y servidor) puedan entenderse.

Todas las comunicaciones en una red (o en Internet) se basan en el protocolo **TCP/IP** para funcionar. Este protocolo está basado en dos partes: el protocolo *TCP* (que establece cómo debe estructurarse y fraccionarse la información para ser enviada) y el protocolo *IP* (que establece cómo se identifican los equipos en la red, mediante direcciones IP).

Sobre esa base de TCP/IP, se establecen una serie de protocolos que se emplean según el tipo de aplicación que se vaya a utilizar (web, correo electrónico, subida de archivos, etc). Al trabajar con aplicaciones web, los protocolos de comunicación más empleados son:

- **HTTP** (*HyperText Transfer Protocol*), un protocolo existente desde 1990 y que permite la transferencia de archivos en general, aunque principalmente de archivos HTML (es decir, documentos web). Se sigue un esquema de peticiones y respuestas entre cliente y servidor como el visto anteriormente.
- **HTTPS**, versión segura del protocolo anterior, donde los datos de las peticiones y las respuestas se envían encriptados, para que nadie que intercepte la comunicación pueda descifrar el contenido de la misma. Este tipo de protocolos se suele utilizar en sistemas bancarios, plataformas de pago (Paypal, por ejemplo), y otras aplicaciones que manejen información delicada (DNIs, números de tarjetas de crédito, etc.).

Normalmente, los navegadores web cambian automáticamente del protocolo “normal” HTTP a HTTPS al conectar con páginas que necesitan ser más seguras (login, datos de pago, etc.). Se puede comprobar el cambio mirando la barra de dirección del navegador: al acceder al protocolo seguro se mostrará el protocolo *https* en la barra, o bien el icono de un candado. Sin embargo, sí deberemos configurar nuestro servidor web para aceptar comunicaciones HTTPS, si fuese el caso.

Otros protocolos son menos utilizados a la hora de trabajar con aplicaciones web, pero sí se utilizan igualmente en otras aplicaciones que requieran de Internet. Por ejemplo, para el envío y recepción de correo electrónico se emplean los protocolos **SMTP** o **POP3/IMAP**, respectivamente. Para enviar archivos a un servidor remoto se puede emplear (además del propio protocolo HTTP) el protocolo **FTP**. Este último protocolo también es habitual a la hora de trabajar con aplicaciones web, especialmente cuando las estamos desarrollando, para subir las actualizaciones de la aplicación al servidor.

4.1. Más sobre HTTP/HTTPS

El protocolo sobre el que se basan las aplicaciones web para funcionar es, por tanto, HTTP (o su versión segura, HTTPS). En ambos casos, se trata de un protocolo para aplicaciones cliente-servidor, donde los datos que se envían uno y otro tienen un formato determinado.

4.1.1. Peticiones HTTP

Por un lado, están los datos que el cliente envía al servidor, y que se denominan **peticiones** (en inglés, *requests*). Estas peticiones se componen, a grandes rasgos, de:

- La **URL** del recurso solicitado
- Unas **cabeceras de petición** que dan información sobre el recurso solicitado y el cliente que lo solicita. Por ejemplo, podemos obtener el navegador que se está utilizando en el cliente, el idioma, etc.
- Unos **datos** adicionales, en caso de que sean necesarios. Por ejemplo, en el caso de enviar un formulario o subir un fichero al servidor, estos datos pueden consistir en la información introducida en el formulario, o en los propios bytes del fichero a subir, respectivamente.

Todos estos datos, a bajo nivel, se encapsulan en paquetes y se fragmentan de acuerdo al protocolo TCP para ser enviados al servidor.

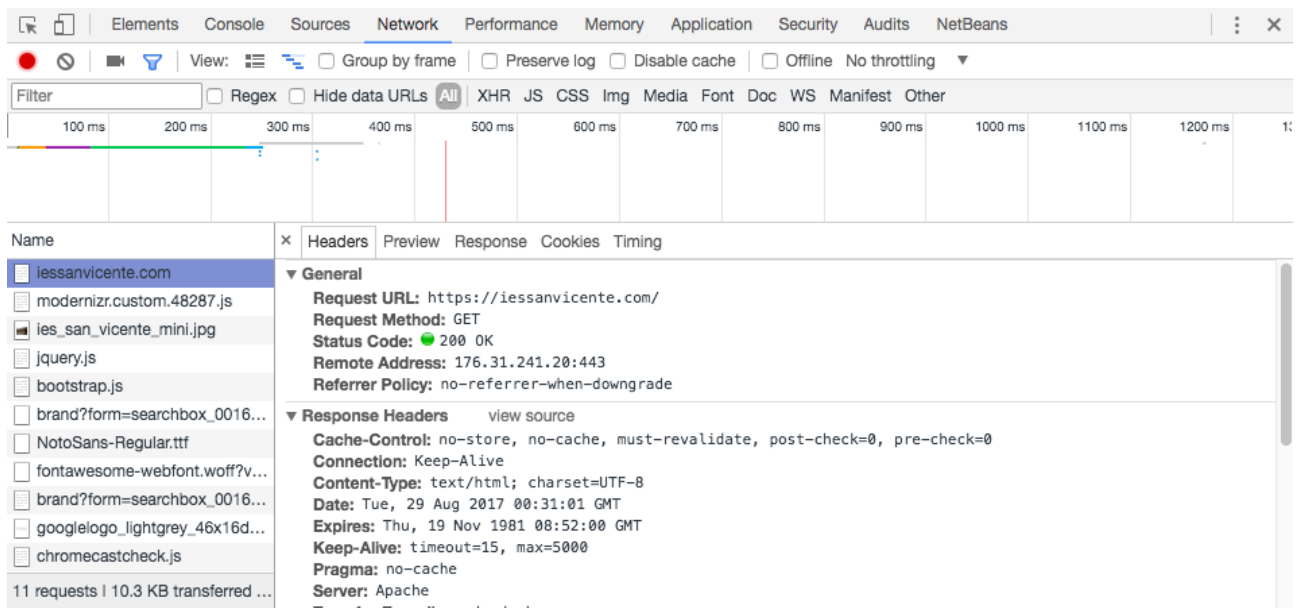
4.1.2. Respuestas HTTP

Por su parte, el servidor, cuando recibe una petición de un cliente, emite una **respuesta** (en inglés, *response*) con la información solicitada, o con algún código de error en caso de que hubiera sucedido alguno. Las respuestas se componen de estos elementos:

- Un **código de estado**, que indica si se ha podido atender correctamente la petición o no. Estos códigos están agrupados en categorías, de forma que, por ejemplo:
 - Los códigos 2xx indican una respuesta satisfactoria. Lo normal es recibir un código 200 si todo ha ido bien
 - Los códigos 3xx indican que ha habido algún tipo de redirección: el recurso solicitado estaba en otra URL y se nos ha redirigido a ella
 - Los códigos 4xx indican un error por parte del cliente. Por ejemplo, el error 404 es muy típico, e indica que la URL indicada no existe. El error 403 es también típico, e indica que el cliente no tiene permiso para acceder al recurso solicitado.
 - Los códigos 5xx indican un error por parte del servidor. Por ejemplo, que esté colapsado y exceda el tiempo de espera para atender la petición.
- Unas **cabeceras de respuesta**, que dan información sobre la respuesta que se envía. Por ejemplo, el tamaño de la respuesta, el tipo de contenido (si es un documento web, un archivo ZIP, etc... es lo que se conoce como *tipo MIME*), la última fecha de modificación, etc.
- El **contenido** solicitado, si no ha habido error al tramitar la petición. Por ejemplo, el contenido de la web que se ha solicitado, o de un archivo que se ha pedido descargar.

4.1.3. Monitorización con Google Chrome

Si tenemos a mano el navegador Google Chrome, podemos comprobar lo que cliente (navegador) y servidor se envían en un proceso HTTP. Para ello, vamos al menú de *Herramientas para desarrolladores*, y más concretamente a la sección *Network*. Desde ahí, accedemos a una web conocida (por ejemplo, iessanvicente.com), y podemos comprobar la información enviada y recibida:



5. Patrones de diseño software

Un patrón de diseño o arquitectura software comprende un conjunto de pautas a seguir, elementos a desarrollar, jerarquías y orden que dotan a una aplicación de una estructura preestablecida, que la hace más propicia para funcionar como debe. Sus principales objetivos son, por un lado, estandarizar la forma en que se desarrollan las aplicaciones, y por otro, elaborar elementos o componentes reutilizables entre diversas aplicaciones, al ajustarse todos a un mismo patrón.

En el ámbito de las aplicaciones web, existen patrones de diseño específicos que nos guían a la hora de estructurar, diseñar y programar estas aplicaciones. Uno de los más utilizados (o quizá el más utilizado) es el patrón MVC, que comentaremos a continuación, pero también han surgido otros (muchos a partir de éste), que han querido dar una vuelta de tuerca más, o adaptarse a las necesidades de aplicaciones web más específicas o concretas. Veremos también algunos de estos patrones en este apartado.

5.1. El patrón MVC

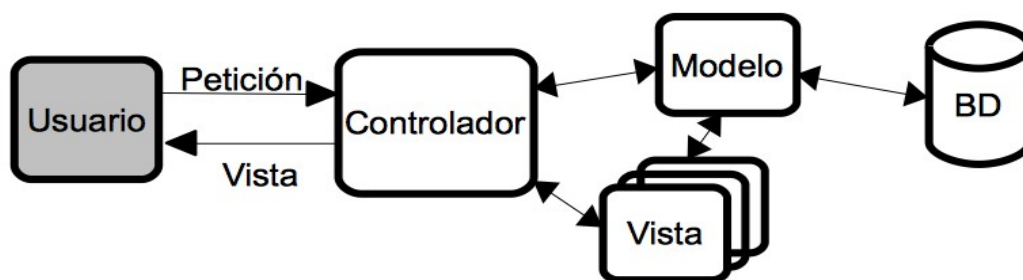
MVC son las siglas de *Modelo-Vista-Controlador* (o en inglés, *Model-View-Controller*), y es, como decíamos antes, el patrón por excelencia ahora mismo en el mundo de las aplicaciones web, e incluso muchas aplicaciones de escritorio.

Como su nombre indica, este patrón se basa en dividir el diseño de una aplicación web en tres componentes fundamentales:

- El **modelo**, que podríamos resumir como el conjunto de todos los datos o información que maneja la aplicación. Típicamente serán variables u objetos extraídos de una base de datos o cualquier otro sistema de almacenamiento, por lo que el código del modelo normalmente estará formado por instrucciones para conectar con la base de datos, recuperar información de ella y almacenarla en algunas variables determinadas. Por tanto, no tendrá conocimiento del resto de componentes del sistema.
- La **vista**, que es el intermediario entre la aplicación y el usuario, es decir, lo que el usuario ve en pantalla de la aplicación. Por lo tanto, la vista la compondrán las diferentes páginas, formularios, etc, que la aplicación mostrará al usuario para interactuar con él.
- El **controlador** (o controladores), que son los fragmentos de código encargados de coordinar el funcionamiento general de la aplicación. Ante peticiones de los usuarios, las recogen, las identifican, y acceden al modelo para actualizar o recuperar datos, y a su vez, deciden qué vista mostrarle al usuario a continuación de la acción que acaba de realizar.

Es un patrón de diseño muy conciso y bien estructurado, lo que le ha valido la fama que tiene hoy en día. Entre sus muchas ventajas, permite aislar el código de los tres elementos involucrados (vista, modelo y controlador), de forma que el trabajo es mucho más modular y divisible, pudiendo encargarse de las vistas, por ejemplo, un diseñador web que no tenga mucha idea de programación en el servidor, y del controlador un programador PHP que no tenga muchas nociones de HTML.

En forma de esquema, podríamos representarlo así:



Las peticiones del usuario llegan al controlador, que las identifica, y se comunica con el modelo para obtener los datos necesarios, y con las vistas para decidir qué vista mostrar a continuación y llenarla con los datos del modelo, para después servírsela al usuario como respuesta.

5.2. Otras alternativas al MVC

Como alternativas al patrón MVC, y a raíz de este mismo patrón, surgieron otros algo más específicos. Casi todos ellos tienen como base la parte del *modelo* (es decir, el acceso y gestión de los datos o información de la aplicación) y la parte de la *vista* (es decir, la presentación al usuario). Así, podríamos decir que el único punto "discordante" sería el controlador, que en otros patrones se ha sustituido por otros elementos. De hecho, al conjunto de patrones que siguen esta filosofía (es decir, centrarse en la vista y el modelo, y añadir algo más), se les suele llamar en general MVW (en inglés, *Model-View-Whatever*, en español, *Modelo-Vista-Cualquier cosa*).

La finalidad de esto es, en algunos casos, descomponer el trabajo realizado por los controladores en varios submódulos, y en otros casos, prescindir directamente del controlador. Veamos algunos de los patrones más populares.

5.2.1. El patrón MVVM

El patrón MVVM, como recogen sus siglas, se centra exclusivamente en los componentes del modelo y de la vista (*Modelo-Vista-Vista-Modelo*), y prescinde del controlador. De esta forma, el usuario interactúa directamente con la vista, y las acciones o cambios que introduzca en ella afectan directamente al modelo, y viceversa (los cambios en el modelo se reflejan de forma automática en la vista).

Este patrón está cobrando especial relevancia en las llamadas SPA (*Single Page Applications*), aplicaciones web con una sola página que recarga parcialmente sus contenidos ante las acciones del usuario. En estos casos, no es necesario un controlador que diga qué vista cargar, porque sólo hay una vista principal (que puede estar compuesta por subvistas), y si la estructura es lo suficientemente sencilla, vista y modelo pueden estar intercomunicados sin intermediarios. Algunos frameworks surgidos últimamente han dado aún más peso a este patrón, como es el caso de AngularJS.

5.2.2. El patrón MOVE

El patrón MOVE sustituye el controlador del patrón MVC por dos elementos. Uno que denomina **operaciones** (que sería la O de sus siglas), y que englobaría todo el conjunto de acciones que la aplicación es capaz de realizar, y otro que serían los **eventos** (que sería la E de sus siglas), y que representarían todos aquellos sucesos que desencadenan que se ejecute una acción determinada. Así, por ejemplo, las acciones de los usuarios son eventos sobre la aplicación que provocan que se ejecuten determinadas operaciones.

Estas operaciones, a su vez, pueden acceder al modelo para obtener o actualizar información, y pueden generar o llamar a una vista que mostrar al usuario como respuesta. Se divide así la tarea de los controladores entre los eventos y las operaciones.

5.2.3. El patrón MVP

El patrón MVP sustituye el controlador (o controladores) del MVC por lo que se denominan **presentadores**. Estos presentadores son una especie de intermediarios entre el modelo y la vista, de forma que cada vista tiene el suyo propio, y actúa tras la vista para comunicarse con el modelo, obtener los datos, y cargarlos en ella para mostrarlos al usuario. Se tiene así encapsulado con cada vista su presentador, y la aplicación puede considerarse un conjunto de pares *vista-presentador*, que se encargan de comunicarse con el modelo, que queda por detrás.

6. Recursos necesarios

Para implantar una aplicación web y que los clientes puedan utilizarla, necesitamos contar con una serie recursos hardware y software.

En el lado del **cliente**, simplemente habrá que contar con un equipo con el hardware necesario (dependiendo de la aplicación web que sea, podrá ser un móvil, una tablet, portátil, PC...), y un navegador web instalado.

En el lado del **servidor**, normalmente necesitaremos al menos un PC servidor con un hardware relativamente potente (en cuanto a procesador, memoria RAM y capacidad de disco duro). En él, necesitaremos tener instalado:

- Un **servidor web**, o servidor de aplicaciones, donde tendremos alojada nuestra aplicación web, y atenderá las peticiones de los clientes. Normalmente se tendrá un servidor web localizable en Internet donde instalar la aplicación web definitiva (llamado normalmente *servidor de producción*), y otro en algún ordenador local donde irla desarrollando y probando hasta terminarla (llamado normalmente *servidor de pruebas*).
- Adicionalmente, si nuestra aplicación web lo requiere, un **servidor de base de datos** o sistema gestor de bases de datos (SGBD). Esta opción es muy común en las aplicaciones web, pues la mayoría acceden a cierta información que, por lo general, está almacenada en una base de datos.

6.1. Lenguajes

Al hablar de aplicaciones web, es importante determinar el **lenguaje** o lenguajes de programación en que se desarrollan. En el ámbito de las aplicaciones web, distinguimos dos tipos de lenguajes:

- Lenguajes en el entorno cliente o **lenguajes cliente**: son los que permiten que el cliente interactúe con la aplicación web. Para ello, el cliente debe poder ver la aplicación web en el navegador, e interactuar con ella (pinchar en enlaces, rellenar formularios, etc.). En este lado, normalmente se habla de **HTML** y **CSS** para el diseño de las páginas (aunque no son lenguajes de programación propiamente dichos), y de **Javascript** para poder facilitar la interacción entre el usuario y el navegador. También existen otros frameworks y herramientas que facilitan o amplían las posibilidades de desarrollo en el cliente, tales como SASS (un compilador CSS que permite introducir algo de programación en los documentos CSS), y muchos frameworks o librerías Javascript, como jQuery o, más recientemente, Angular.
- Lenguajes en el entorno servidor o **lenguajes servidor**: son los que permiten que el servidor realice ciertas tareas cuando le llegan las peticiones de los clientes, como por ejemplo consultar una base de datos, guardar información en un fichero, o cargar una foto que el usuario está subiendo al servidor. En este otro lado, existen varias familias de lenguajes que podemos elegir, dependiendo del servidor web que queramos utilizar. Por ejemplo, podemos utilizar lenguaje **ASP .NET** (para servidores de Microsoft y entornos Windows), o el lenguaje **JSP** (lenguaje Java para aplicaciones web), o el lenguaje **PHP**, entre otros. También últimamente se ha

hecho un hueco en este grupo el lenguaje **Javascript**, a través de frameworks como Node.js o Backbone.js.

6.2. Ejemplos de software

Hemos visto a grandes rasgos el hardware y software que necesitaremos tanto en los clientes como en el servidor. ¿Qué software concreto podemos instalar por ejemplo?

En el caso del **navegador** para el **cliente**, existen varias opciones dependiendo del sistema operativo del cliente: Mozilla Firefox, Google Chrome, Internet Explorer / Edge (sólo para Windows), Safari (sólo en Windows y Macintosh), Opera...

En el caso del **servidor web**, dependerá del tipo de lenguaje servidor que vayamos a utilizar para hacer nuestra página, y del sistema operativo del servidor.

- Si vamos a emplear lenguaje **PHP**, el servidor más habitual es **Apache**, que además es gratuito y funciona en todo tipo de sistemas. Como alternativa, también se puede emplear algún sistema ***AMPP**, que integra Apache con un servidor MySQL y el lenguaje PHP preinstalados.
- Si vamos a emplear lenguaje **Java** (JSP, servlets, etc.), podemos instalar algún servidor sencillo y gratuito como **Tomcat**, o servidores de aplicaciones más pesados y complejos como **Glassfish**. También funcionan en todo tipo de sistemas
- Si vamos a emplear tecnología **.NET** (ASP.NET), podemos utilizar **IIS** (*Internet Information Services*), un servidor web disponible para Windows, ya que esta tecnología sólo funciona en sistemas operativos Windows.
- En el caso de **Javascript**, existen frameworks como Node.js que actúan a su vez como servidores web (los programamos para que se comporten como servidores), por lo que no es necesario ningún software adicional aparte del propio framework.

Para el **servidor de base de datos**, podemos emplear diferentes alternativas, como **MySQL**, **PostgreSQL**, **Oracle**, o **SQL Server**. Todas ofrecen versiones gratuitas (algunas de ellas con recursos limitados para uso personal) y comerciales (más potentes). MySQL, PostgreSQL y Oracle funcionan en diversos sistemas, pero SQL Server es propiedad de Microsoft, y funciona en sistemas Windows.