Introducción a las aplicaciones web

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:

- **Aplicaciones sin conexión a Internet**. Este tipo de aplicaciones no necesitan ninguna conexión a Internet o a una red local de ordenadores para funcionar. 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.

- **Aplicaciones con conexión a Internet**. Estas aplicaciones sí necesitan conexión, ya sea a Internet o a una red local. Dentro de este grupo, encontramos varios subtipos:

    - **Aplicaciones P2P** (*peer-to-peer*): todos los elementos conectados a la red tienen el mismo "rango" y comparten información entre ellos. Es el mecanismo en el que se basan varios programas de descarga, como los de archivos tipo *torrent*.
    - **Aplicaciones cliente-servidor**: consisten en que un conjunto de ordenadores (llamados *clientes*) se conectan a uno central (llamado *servidor*), que proporciona la información y los servicios solicitados. Es el caso de los videojuegos online, donde el ordenador del jugador (cliente) se conecta a un servidor central que gestiona el juego. Dentro de las aplicaciones cliente-servidor, hay un subtipo especialmente numeroso, las **aplicaciones web**. 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. Estas aplicaciones interactúan con el servidor para procesar datos y presentar información al usuario a través del navegador.

Sin embargo, el avance experimentado en este sector durante los últimos años ha ampliado este concepto. Así, una **aplicación web** sería aquella realizada a partir de lenguajes y tecnologías de desarrollo web, tales como HTML, CSS, JavaScript, PHP, entre otros. Estas aplicaciones pueden ejecutarse en un navegador convencional o mediante un motor de navegador embebido en otro sistema, como el *webview* de Android o iOS.

- Webview es un componente que permite a las aplicaciones mostrar contenido web sin tener que abrir un navegador. Aunque tiene un uso más limitado que un navegador, esta diseñado para consultas puntuales desde la propia aplicación, lo que permite una carga de páginas más rápida. Aplicaciones como *Facebook* e *Instagram* utilizan este componente para integrar contenido web de manera eficiente.

De este modo, además de las aplicaciones web "tradicionales", también tendrían cabida en esta definición las llamadas **aplicaciones híbridas** (desarrolladas con tecnologías web pero exportadas a formato nativo para diversos dispositivos), y **aplicaciones de escritorio** que emplean tecnologías web, como framework *Electron* de JavaScript. Un ejemplo de estas aplicaciones es *Visual Studio Code*, que aprovecha las ventajas de las tecnologías web para ofrecer experiencias de usuario avanzadas y multiplataforma.

2. Arquitectura de una aplicación web

2.1. ¿Qué es "la web"?

La web es una plataforma global que proporciona acceso a una gran cantidad de recursos como documentos, videojuegos, redes sociales, foros, entre otros. Se popularizó a principios de los años 90 gracias a aplicaciones como el correo electrónico y los chats. Con la llegada de la web 2.0, surgieron nuevas aplicaciones que la potenciaron aún más, como los blogs o las redes sociales. Hoy en día, es común ver vídeos o películas en Internet, algo que hace unos años era impensable.

2.2. Elementos de una aplicación web

En una aplicación web podemos distinguir los siguientes tres elementos principales:

- **Front-end**:
    - Es la parte de la aplicación que se ejecuta en el cliente.
    - Los usuarios interactúan con la aplicación utilizando normalmente un navegador web (Google Chrome, Firefox, etc).
    - Se centra en la experiencia del usuario y la presentación. Se encarga de renderizar la interfaz de usuario, gestionar la interacción del usuario y enviar peticiones al servidor.
    - Las tecnologías que se utilizan son principalmente HTML, CSS y Javascript.
- **Back-end**:
    - Es la parte de la aplicación que se ejecuta en el servidor.
    - Es responsable de gestionar las peticiones que llegan desde los clientes, procesar la lógica de negocio, interactuar con la base de datos y devolver las respuestas adecuadas.
    - Las tecnologías utilizadas son variadas:
        - **Servidores web:** Apache, Nginx, entre otros.
        - **Lenguajes de programación:** Javascript (Node.js), PHP, Python (Django, Flask), etc
        - **Bases de datos:** MySQL, PostgreSQL, MongoDB, Redis, etc.
- **Comunicación entre cliente y servidor**:
    - Asegura que cliente y servidor puedan interactuar de forma eficiente.
    - Para ello, se define algún protocolo de comunicación para que ambas partes intercambien información. Puede ser una API REST donde se envíe la información en formato JSON o algún otro

protocolo donde se intercambie información en XML o protocolos simples que envíen contenido HTML.

2.3. Funcionamiento de una aplicación web

Como hemos comentado, las aplicaciones web funcionan bajo una arquitectura cliente-servidor, que distribuye las tareas entre los servidores (proveedores de recursos y servicios) y los clientes (solicitantes de dichos recursos y servicios).

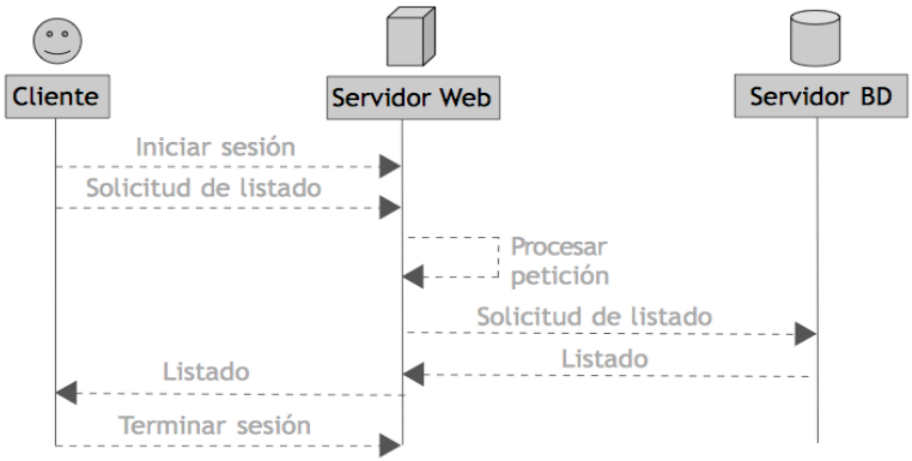
Los pasos típicos en la comunicación cliente-servidor son los siguientes:

1. El cliente inicia sesión en el servidor
2. El cliente solicita al servidor el recurso o servicio (una página web, un documento, subir información, etc.)
3. El servidor recibe la petición, la procesa y determina qué programa debe atenderla, 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 realizar una nueva petición (volver al paso 2) o terminar la sesión.

En muchos casos, la arquitectura del servidor es **multicapa o multinivel**. Esto significa que podemos tener diferentes aplicaciones en distintos equipos (o en el mismo) que ofrecen recursos o servicios. Por ejemplo, un servidor de bases de datos puede estar en una máquina, un servidor web en otra (o en la misma que el servidor de bases de datos), y un servidor de correo electrónico en otra máquina. De esta manera, se distribuyen los procesos y el trabajo, e incluso se podrían configurar opciones de seguridad y rendimiento separadas para cada servidor.

Ejemplo: arquitectura de dos o tres niveles

Consideremos una petición de un cliente para obtener un listado de noticias almacenadas en una base de datos. Este proceso puede ser representado con un diagrama de secuencia como sigue:



1. El cliente envía una solicitud para iniciar sesión al servidor web.
2. Después de iniciar sesión, el cliente solicita un listado de datos al servidor web.

3. El servidor web procesa la petición, lo que puede implicar realizar algún tipo de lógica o validación antes de enviar la solicitud al servidor de base de datos.
4. Una vez procesada la petición, el servidor web envía una solicitud de listado al servidor de base de datos.
5. El servidor de base de datos procesa la solicitud y devuelve el listado de datos solicitados al servidor web.
6. El servidor web prepara la respuesta y la envía al cliente.
7. Finalmente, el cliente envía una solicitud para terminar la sesión al servidor web

En una **arquitectura de tres niveles**, el servidor web y el servidor de bases de datos pueden estar en la misma máquina o en máquinas separadas, cada una con su hardware y control de acceso específicos. Esta configuración es común en aplicaciones web, ya que permiten una mejor gestión y seguridad de los datos.

En una **arquitectura de dos niveles**, el servidor web también actúa como servidor de bases de datos, respondiendo directamente a las peticiones del cliente sin consultar a otros servidores. Esta configuración es menos flexible y segura, y puede ofrecer un rendimiento inferior en sistemas muy congestionados.

2.4. Localización y acceso a aplicaciones web

Hemos visto la arquitectura de una aplicación web y cómo los elementos del front-end y el back-end interactúan mediante la comunicación cliente-servidor. En este apartado vamos a ver cómo se localizan y acceden a estas aplicaciones web en Internet. Aquí es donde entran en juego los conceptos de dominios y URLs.

Hostings y nombres de dominio

El servidor, que se encarga de recibir peticiones de todos los clientes que se conecten a él y enviarles la información que solicitan, necesita estar disponible desde cualquier lugar en Internet para que los usuarios puedan conectarse a él.

Por tanto, en primer lugar, debemos buscar un servidor en Internet en el que alojar nuestro sitio web. Esto puede hacerse de varias formas:

- **Servidor propio:** disponer de un servidor (o servidores) propios y una dirección IP pública fija a la que acceder. Esta opción es menos común hoy en día debido a los costos y la complejidad.
- **Empresa de alojamiento o hosting:** contratar un espacio (o máquina entera, si el proyecto es grande y se dispone de presupuesto suficiente) en una empresa de alojamiento. Esta opción es mucho más habitual, y de ahí que proliferen las empresas que se dedican al hosting como OVH, Hostinger, Ionos, etc.

Además de un espacio de alojamiento, es necesario adquirir un **nombre de dominio** para que los usuarios puedan recordar y utilizar para identificar nuestra web de forma única en Internet. Este dominio se vincula a la dirección IP del servidor, permitiendo que las solicitudes del cliente lleguen al lugar correcto.

Por ejemplo, cuando escribimos la dirección *www.google.es*, un sistema conocido como **DNS** (*Sistema de Nombres de Dominio*) traduce este nombre de dominio a la dirección IP del servidor donde está alojada la aplicación de Google en español. Este proceso, llamado resolución de DNS, es crucial para el funcionamiento de las aplicaciones web, ya que facilita la localización de recursos en la red global.

El **nombre de dominio** se adquiere a través del **registro de dominio**, un proceso gestionado por empresas autorizadas por la **ICANN** (*Internet Corporation for Assigned Names and Numbers*). Para registrar un dominio, se debe contactar con una empresa registradora, comprobar que el dominio deseado está disponible, y seguir el proceso de registro.

URLs

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 secuencia de caracteres que sigue un estándar y que permite denominar recursos dentro de Internet para que puedan ser localizados.

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:

protocolo://subdominio.dominio.com/carpeta/pagina?param=valor

- El **protocolo**, define las reglas para la comunicación entre cliente y servidor. En una URL, el protocolo va al principio, hasta los dos puntos y el delimitador //. En nuestro ejemplo, el protocolo sería **http** (*HiperText Transfer Protocol*). Veremos más adelante algunos ejemplos de protocolos.
- El **nombre de dominio**, identifica el servidor y la entidad (empresa o web) a la que se va 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*
- El **subdominio**, que es opcional, es una forma de tener un sitio web relacionado, como anexo, a una web principal. Es habitual que las empresas de hosting permitan registrar subdominios, aunque algunas lo ofrezcan con ciertas restricciones, ya sea por el número de subdominios permitidos o por el servicio que prestan. Por ejemplo, si tenemos el dominio *www.miweb.com* del ejemplo anterior como URL principal, podríamos tener además, *tienda.miweb.com* como subdominio para una tienda asociada a la web.
- 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*
- Unos **parámetros** opcionales, que proporcionan información adicional a la solicitud, útil para el servidor en la entrega de contenidos dinámicos.

> **Ejercicio 1:**
>
> Hemos visto que la ICANN autoriza a diversas empresas para gestionar el registro de nombres de dominio en Internet. Estas empresas, conocidas como agentes registradores de dominio, ofrecen distintos planes de pago para este servicio. Busca en Internet una lista de agentes registradores de dominio oficiales para España y consulta los precios que tienen algunos de ellos.

3. Protocolos más utilizados

Para que los clientes y servidores puedan comunicarse, es necesario establecer un **protocolo** de comunicación. Un protocolo define una serie de reglas que especifican qué tipo de mensajes se van a intercambiar, en qué orden y qué contenido va a tener cada tipo de mensaje, asegurando que ambos 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:

- **Protocolo TCP**, que establece cómo debe estructurarse y fraccionarse la información para ser enviada
- **Protocolo IP**, que define cómo se identifican los equipos en la red, mediante direcciones IP.

Sobre la base de TCP/IP, se establecen diversos protocolos específicos según el tipo de aplicación que se vaya a utilizar (web, correo electrónico, subida de archivos, etc). Para las aplicaciones web, los protocolos de comunicación más utilizados son:

- **HTTP** (*HyperText Transfer Protocol*): protocolo existente desde 1990 que permite la transferencia de archivos, principalmente de archivos HTML. Funciona mediante 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 descrifrar 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 HTTP a HTTPS al conectar con páginas que necesitan ser más seguras (login, datos de pago, etc.). Este cambio puede observarse en la barra de direcciones del navegador, donde se mostrará el protocolo *https* o el icono de un candado. Sin embargo, es necesario configurar el servidor web para aceptar comunicaciones HTTPS cuando sea necesario.

Otros protocolos, aunque menos utilizados en aplicaciones web, son esenciales para otras aplicaciones en Internet. Por ejemplo, para el envío y recepción de correo electrónico se emplean los protocolos **SMTP** y **POP3/IMAP** respectivamente. Para transferir archivos a un servidor remoto, se puede usar el protocolo **FTP**. Este último protocolo es especialmente útil durante el desarrollo de aplicaciones web para subir actualizaciones al servidor.

3.1. Más sobre HTTP/HTTPS

El protocolo fundamental para las aplicaciones web es 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.

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 saber el navegador que se está utilizando en el cliente, el idioma, etc.
- Unos **datos adicionales**, en caso de que sean necesarios. Por ejemplo, la información introducida en un formulario o los bytes de un archivo a subir.

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

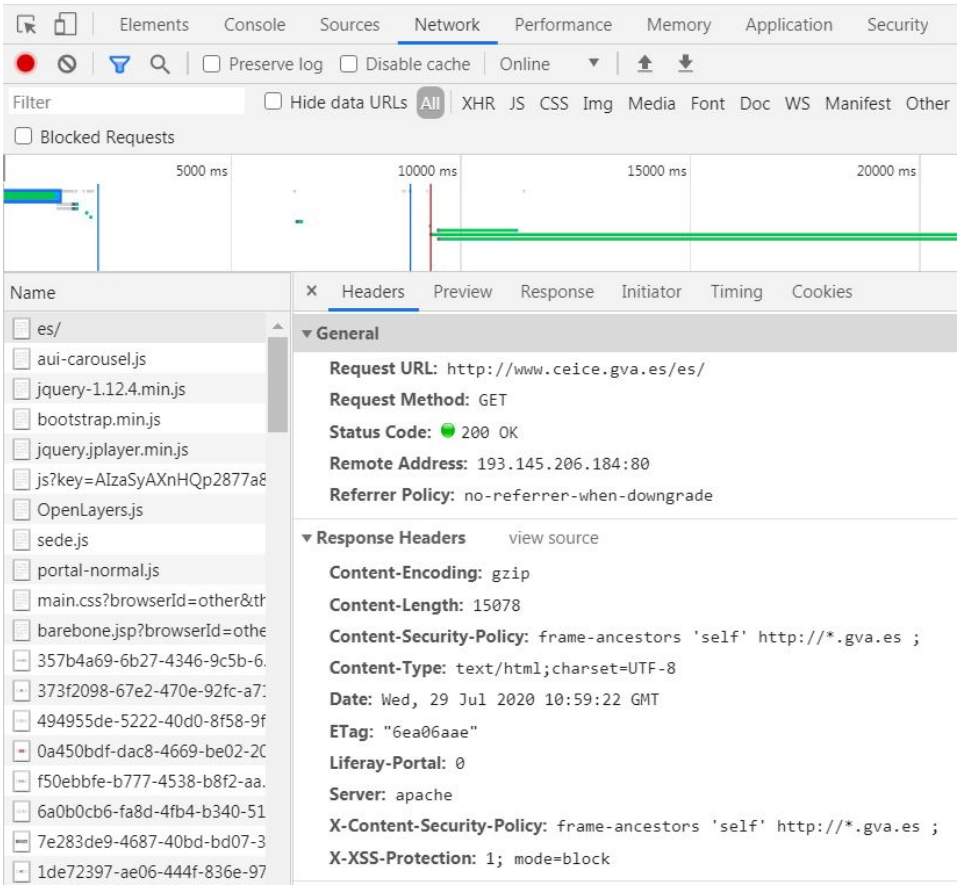
Respuestas HTTP

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.

Monitorización con Google Chrome

Podemos usar el navegador Google Chrome para 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, *ceice.gva.es*), y podemos comprobar la información enviada y recibida:



> **Ejercicio 2:**
>
> Utiliza Google Chrome (opción de *Herramientas para desarrolladores*, pestaña *Network*) para ver el esquema de petición y respuesta HTTP hacia alguna web conocida, como por ejemplo *stackoverflow.com*. Identifica el código de estado, las cabeceras de respuesta (trata de identificar algunas de ellas) y el contenido.

4. Sistema de nombres de dominio o DNS

En las comunicaciones en redes TCP/IP, cuando un equipo envía un paquete de datos a otro, es necesario identificar tanto el origen como el destino. Las aplicaciones que inician la transmisión deben incorporar valores obligatorios tales como: dirección MAC de origen y de destino, dirección IP de origen y de destino y puerto de origen y de destino. Estos campos siempre deben tener un valor asignado. Esto significa que, cuando se quiere visitar una página web que está alojada en un equipo (por ejemplo, el equipo con el nombre *www.google.es*), el cliente debe saber cuál es la dirección IP de destino para incorporarla en el paquete.

De hecho, cuando se abre un navegador web es posible acceder a un recurso tanto por su nombre de dominio como por su dirección IP. Siguiendo con el ejemplo anterior, si accedemos a una ventana de terminal y ejecutamos el comando *ping www.google.es*, obtendremos en la primera línea su dirección IP. Si se coloca esa misma dirección IP en un navegador web, se puede comprobar que devuelve la misma página web de inicio. Sin embargo, como usuarios, nos resulta más fácil dirigirnos a un equipo (host, servidor web, servidor de correo, etc.) utilizando su nombre de dominio que recordar su dirección IP correspondiente.

El **DNS** (*Domain Name System*) ó **Sistema de Nombres de Dominio** es el encargado de proporcionar un mecanismo eficaz para la traducción de direcciones IP de recursos de red a nombres fácilmente legibles y memorizables por las personas, y viceversa. A esta acción se la conoce como **resolución DNS**. Para ello, utiliza una base de datos distribuida y jerárquica que asocia direcciones IP de hosts, servicios o cualquier recurso conectado a internet o red privada con información de diverso tipo.

- **Jerárquica** porque se organiza en una estructura de dominios, los cuales se componen de subdominios, que a su vez se pueden dividir en otros subdominios y así hasta 127 niveles.
- **Distribuida** porque la información de la base de datos no está toda en un solo servidor central, sino que la información se encuentra repartida por partes en diferentes servidores DNS de Internet.

El DNS soporta tanto IPv4 como IPv6, y la información se almacena en forma de **registros de recursos**, en inglés *Resource Records* (RR), de distintos tipos, los cuales pueden almacenar direcciones IP u otro tipo de información. Esta información se agrupa en zonas, que corresponden a un espacio de nombres o dominio concretos y que son mantenidas por el servidor DNS autoritativo de la misma. A continuación, profundizaremos un poco más en estos y otros conceptos.

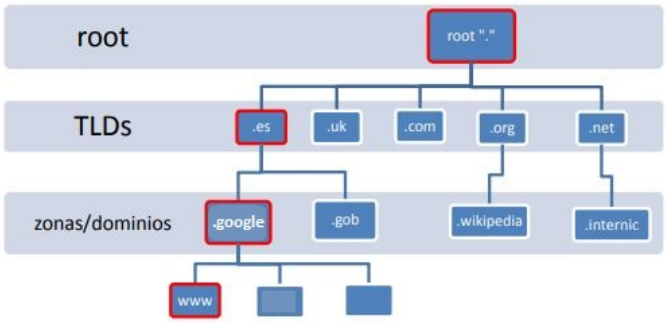
4.1. Elementos integrantes del DNS

El DNS se estructura en tres componentes principales: **espacio de nombres de dominio**, **servidores de nombres** y **resolvers**.

Espacio de nombres de dominio

El espacio de nombres de dominio es una estructura jerárquica de árbol invertido donde cada nodo contiene **registros de recursos** (*RR*) que proporcionan información sobre los componentes hardware y software que respaldan dicho dominio, como por ejemplo, los hosts, los servidores de nombres, los servidores web, los servidores de correo, etc.

Del nodo raíz, situado en el nivel más alto, parten las ramas que conforman las zonas. Éstas, a su vez, pueden contener uno o más nodos o dominios que a su vez pueden dividirse en subdominios según se baja en la jerarquía.

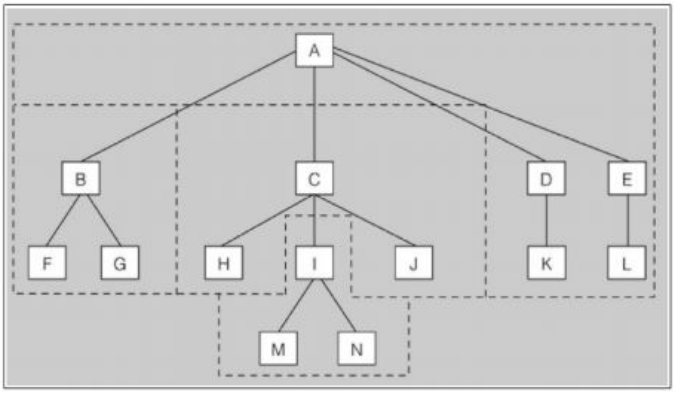


- Cada nodo del árbol se llama dominio y recibe una etiqueta o nombre. El nombre de dominio de un nodo se crea mediante la concatenación de todas las etiquetas, comenzando por dicho nodo y terminando en el nodo raíz. Por tanto, todo dominio completo termina en un punto final "." que indica el final del espacio en la zona raíz.

- El nivel más alto de toda la jerarquía es el **dominio raíz** o **root**, y se representa por "." (punto). En el sistema DNS un nodo puede tener un nombre de hasta 63 caracteres. La profundidad de nodos está limitada a 127 niveles.
- Justo un nivel por debajo se encuentran los **Top Level Domains** o **TLDs**, los cuales están alojados en lo que se conoce como **servidores raíz**. Existen varios servidores raíz ubicados en diferentes continentes. Se identifican con las letras del alfabeto, y varios de ellos se encuentran divididos físicamente y dispersos geográficamente con el fin de aumentar su rendimiento. Sus nombres son de la forma *letra.root-servers.org*. Son gestionados por organizaciones independientes. Por ejemplo, *a.root-servers.org* está gestionado por *VeriSign, Inc*. Puedes encontrar más información sobre estos servidores en *https://root-servers.org/*.
- Respecto a los TLDs, existen 3 tipos:
  - **Dominios de nivel superior geográficos (*ccTLD*)**: son aquellos TLDs que representan a países. Utilizan los códigos de país de dos letras definidos en estándar ISO-31663. Por ejemplo, *.es* para España, *.fr* para Francia, *.pt* para Portugal, etc.
  - **Dominios de nivel superior genéricos (*gTLD*)**: aquellos TLD que están abiertos a cualquier persona u organización. Están formados por tres o más caracteres. Por ejemplo, *.biz* para negocios, *.com* para propósitos comerciales, *.info* para información general, etc.
  - **Dominios de nivel superior de infraestructura (*uTLD*)**: esta categoría tiene un único TLD, *.arpa*, que se utiliza exclusivamente para la gestión de la infraestructura.

- Los TLDs, a su vez, son nodos padre de otros niveles inferiores que se conocen como **TLDs de segundo nivel**.
- La jerarquía continúa hasta llegar a un nodo final que representa un recurso. Por ejemplo "*www.miweb.com*" realmente es "*www.miweb.com.*", donde el punto final más a la derecha representa la zona raíz, aunque éste último normalmente no se muestra. El nombre formado por toda la cadena se conoce como **Fully Qualified Domain Name** (*FQDN*).

Es importante no confundir los conceptos de **zona** y **dominio**. Un dominio se divide en subdominios para facilitar su administración, y cada parte administrada por uno o más servidores DNS es una zona. El dominio es el árbol del espacio de nombres y la zona es la parte del árbol administrada por un servidor de nombres de dominio concreto.



En la figura anterior hay tantos dominios como recuadros de letras agrupados en 4 zonas delimitadas por las líneas discontinuas. El nombre de dominio correspondiente a cada zona (se nombran según su nodo superior) es A, B.A, C.A, I.C.A. Cada una de estas cuatro zonas tendrá uno o más servidores DNS para gestionarla.

Servidores de nombres

Los servidores de nombres son servidores encargados de mantener y proporcionar información del espacio de nombres o dominios.

- **Servidores autoritativos**: almacenan información completa para una o varias zonas de las cuales son responsables. Proporcionan respuestas sobre los dominios para los que han sido configurados. El estándar que define el DNS establece que toda zona debe tener, al menos, dos servidores autoritativos:
    - El **primario**, que guarda y administra las versiones definitivas de los registros de recursos de la zona
    - El **secundario**, que guarda una copia que se actualiza cada vez que se produce un cambio, a través de un proceso conocido como **transferencia de zona**. Esto proporciona un mecanismo de redundancia, robustez, rendimiento y copia de seguridad, ya que si el servidor primario falla, la red sigue operativa gracias al servidor secundario.
- **Servidores caché:** almacenan conjuntos de registros de distintas zonas obtenidos consultando a los correspondientes servidores autoritativos (búsquedas recursivas). Esta información se almacena localmente de forma temporal (caché) y se renueva periódicamente. Los servidores caché ayudan a descongestionar servidores que reciben grandes cantidades de peticiones.

Con esta organización de servidores de nombres, y su intercomunicación, se consigue la distribución y redundancia del espacio de dominios.

Resolvers

Un resolver es una rutina del sistema operativo que funciona como intermediaria entre las aplicaciones, el sistema operativo y los servidores DNS. Cuando una aplicación necesita una resolución, ya sea de un nombre o de una dirección IP, llama a esta rutina devolviendo la información deseada de forma compatible para que el host la entienda.

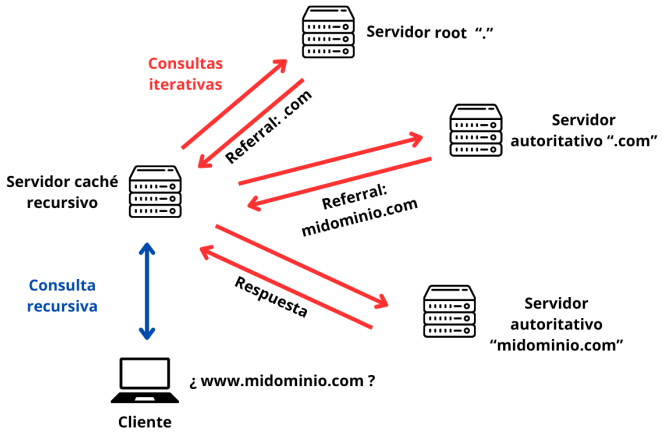
4.2 Proceso de resolución DNS

La principal función de un servidor DNS es responder a consultas de clientes o de otros servidores DNS. Existen dos tipos principales de consultas:

- **Consulta Recursiva:**
    - El *resolver* (cliente) pide a su servidor DNS local una respuesta basada en sus archivos de zona o caché.
    - Si el servidor no tiene la información solicitada, responde con el puntero al servidor raíz ".".
    - El *resolver* consulta al servidor raíz, luego los servidores de dominio de nivel superior (TLD), y así sucesivamente, hasta encontrar el servidor autoritativo que tiene la respuesta correcta.
    - Normalmente, el *resolver* hace una consulta recursiva a un servidor DNS intermedio que actúa como caché recursivo, evitando que el cliente tenga que realizar múltiples consultas.
- **Consulta Iterativa:**

- El *resolver* solicita al servidor DNS una respuesta o un error si el recurso no existe.
- El servidor DNS actúa como intermediario, realizando consultas iterativas a otros servidores DNS hasta obtener la respuesta o un error.

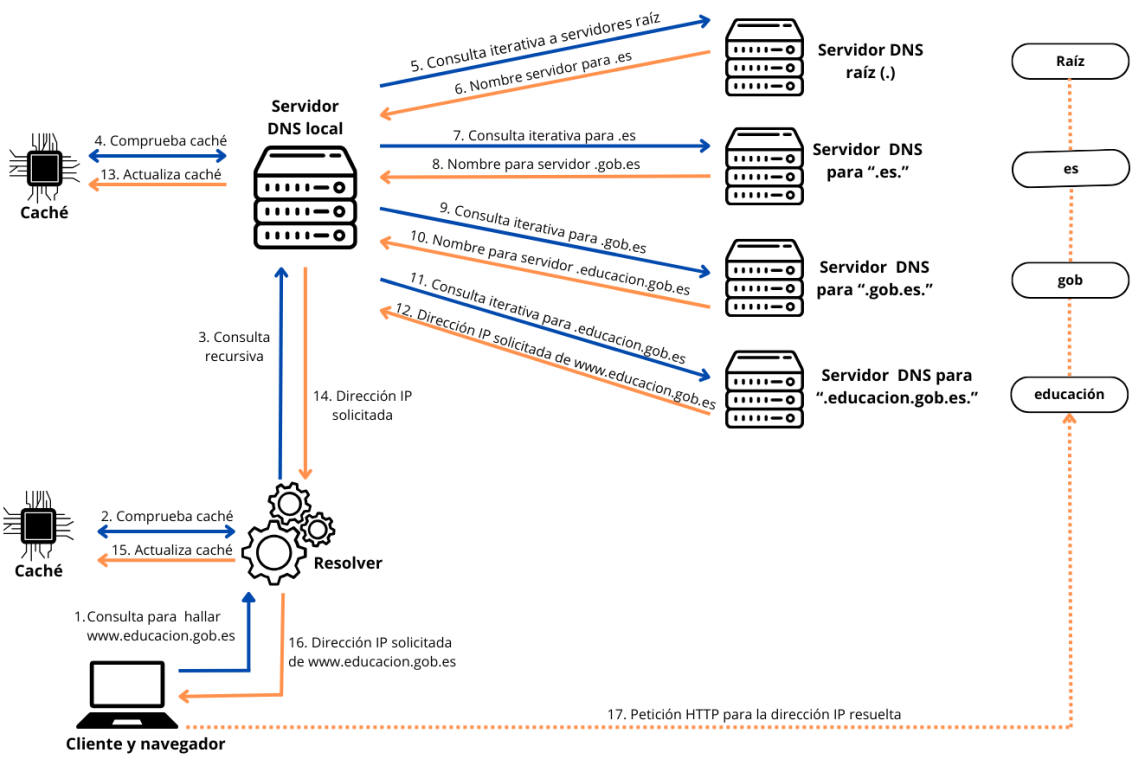
En resumen, las consultas recursivas son generalmente generadas por los clientes DNS, mientras que las consultas iterativas son creadas por los servidores DNS al consultar a otros servidores.



En términos generales, el proceso que se sigue en una resolución DNS es el siguiente.

- El cliente (*resolver*) envía la consulta a su servidor DNS.
- Si el servidor DNS tiene autoridad sobre el dominio consultado, devuelve la respuesta usando sus propios registros.
- Si el servidor no tiene autoridad y no es recursivo, informa al *resolver* dónde debe dirigir su consulta.
- Si el servidor no es autoritativo pero es recursivo, realiza consultas iterativas para encontrar el servidor autoritativo y devuelve la respuesta al cliente, almacenando la información en caché para futuras consultas.

Veamos un ejemplo concreto de resolución de nombres para comprender mejor el proceso. Supongamos que un cliente necesita localizar el equipo *www.educacion.gob.es* y envía una petición a su servidor DNS. Asumamos también que su servidor no tiene información sobre ese dominio. Así, observaremos cómo se lleva a cabo la resolución completa



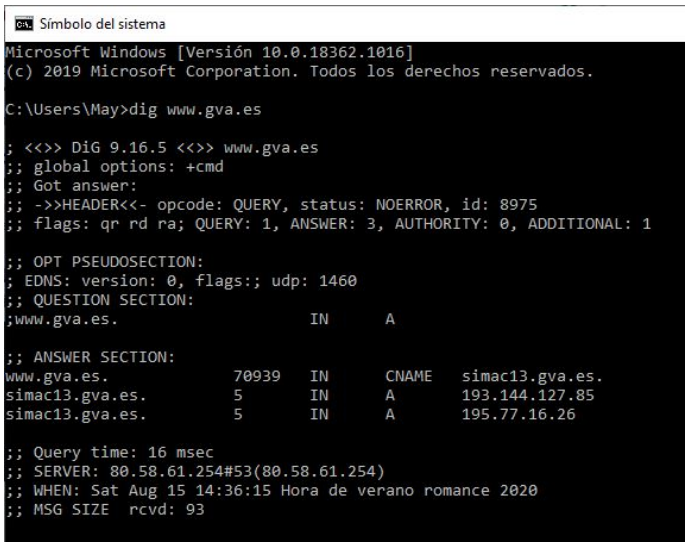
1. **Consulta inicial del cliente:** un cliente, mediante un navegador web, solicita la resolución del nombre de dominio *www.educacion.gob.es*. El *resolver* consulta su caché para ver si tiene la dirección IP almacenada.
2. **Consulta al servidor DNS local:** si la dirección no está en la caché, el *resolver* envía una consulta recursiva al servidor DNS primario o local configurado. Este servidor puede estar dentro de la empresa o en Internet. El servidor DNS local revisa su caché y, si encuentra la información, devuelve la dirección IP correspondiente.
3. **Consultas iterativas del servidor DNS:** si el servidor DNS local no puede resolver la consulta, inicia una serie de consultas iterativas para encontrar el servidor autoritativo del dominio:
    - **Servidor raíz:** consulta al servidor raíz. Si no es autoritativo, el servidor raíz devuelve el nombre del servidor responsable del dominio de primer nivel (*.es*).
    - **Servidor TLD:** consulta al servidor del TLD (*.es*). Si no es autoritativo, devuelve el nombre del servidor responsable del dominio de segundo nivel (*.gob.es*).
    - **Servidor del dominio específico:** consulta al servidor del dominio de segundo nivel (*.gob.es*). Si no es autoritativo, devuelve el nombre del servidor responsable del dominio específico (*educacion.gob.es*).
4. **Respuesta del servidor autoritativo:** finalmente, se llega al servidor autoritativo para el dominio *educacion.gob.es*, que devuelve la dirección IP de *www.educacion.gob.es* al servidor DNS local.
5. **Actualización y respuesta:**
    - El servidor DNS local actualiza su caché con la nueva información.
    - El servidor DNS local envía la respuesta al *resolver* del cliente.
    - El *resolver* del cliente actualiza su caché.
    - La respuesta llega al navegador web en el formato adecuado, completando la solicitud de resolución.

Este proceso consume bastante ancho de banda, por lo que es recomendable implementar una solución DNS interna con caché, especialmente en redes con alto tráfico hacia el exterior, como colegios, universidades, bibliotecas o grandes empresas. Esto permite obtener tiempos de respuesta más rápidos y una mejor experiencia de navegación web.

Herramienta DIG

DIG (*Domain Information Gopher*) es una herramienta de línea de comandos que realiza búsquedas en los registros DNS, a través de los nombres de servidores, y muestra el resultado. Se puede ejecutar tanto en Linux como en Windows aunque, en este último, es posible que sea necesaria su instalación previa.

Veamos un ejemplo de cómo funciona con el comando `dig www.gva.es`:



Explicación de la salida:

- Todas las líneas que inician con ";" son comentarios.
- La primera línea nos devuelve la versión del comando dig que estamos utilizando.
- Luego muestra una **sección de consulta** (*question section*) con nuestra consulta .
- A continuación, aparece la **sección de respuesta** (*answer section*) que contiene la respuesta del servidor DNS.
    - Esta sección puede incluir diferentes tipos de registros de recurso (RR). Por defecto, la respuesta está asociada al registro de recurso de dirección A (RR A) y al registro de recurso de nombre canónico (RR CNAME), pero también se pueden consultar otros registros.
    - El **registro de dirección A** (*RR A*) asocia nombres de dominio (FQDN) a direcciones IPv4. El registro A tiene la estructura siguiente: *NombreDominio IN A IP*.
    - El **registro de recurso nombre canónico** (*RR CNAME*) enlaza el nombre de dominio con un alias, es decir, con otro nombre que también redirige al mismo contenido. El verdadero nombre, por lo tanto, es el que se conecta a través del registro A con la dirección IP. La ventaja de este sistema es que, si la dirección IP cambia, solo es necesario modificar el registro A. Puesto que todos los alias se guían según este registro A. Su estructura es la siguiente: *NombreDominio IN CNAME Nombre canónico o IP*

- **Interprestación de la respuesta:**
    - Primero se muestra el registro CNAME que indica que *www.gva.es* es un alias de *simac13.gva.es*.
    - Luego se muestran dos registros A para *simac13.gva.es*, proporcionando sus direcciones IP.
    - Cada registro comienza con el nombre completo del dominio (incluyendo el punto final), seguido del tiempo en segundos que permanecerá el registro en la memoria caché.
- **Estadísticas de la consulta:**
    - Finalmente, se muestran estadísticas como el tiempo de consulta, el servidor que respondió, la fecha y hora de la consulta y el tamaño del mensaje recibido.

Si se ejecuta DIG especificando la opción **+trace**, se muestra una traza de los servidores por los que pasa la consulta hasta llegar al servidor autoritativo, permitiendo ver la lista completa de nodos y pasos de resolución de un nombre. Esto es muy útil para entender cómo funciona el sistema de nombres de dominio.

5. Patrones de diseño software

Después de revisar los tipos de aplicaciones web, la arquitectura básica, los protocolos más comunes y el papel del DNS, llegamos a un componente esencial en el desarrollo de aplicaciones web robustas y eficientes: los patrones de diseño.

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:

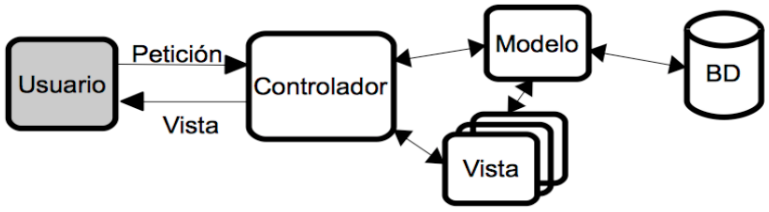
- El **modelo**, que podríamos resumir como el conjunto de todos los datos o información que maneja la aplicación. Típicamente serán variables u objetos extraídos de una base de datos o cualquier otro sistema de almacenamiento, por lo que el código del modelo normalmente estará formado por instrucciones

para conectar con la base de datos, recuperar información de ella y almacenarla en algunas variables o clases 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:



5.2. Otras alternativas a 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, *Movel-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.

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 Angular.

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 operación determinada. Así, 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.

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, típicamente, un navegador web instalado (aunque también podría tratarse de una aplicación híbrida para móvil, o de escritorio, en cuyo caso haría falta la aplicación en sí).
- 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 *servidor de producción*), y otro en algún ordenador local donde irla desarrollando y probando hasta terminarla (*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 Angular, React, Vue... Además, existe algún lenguaje alternativo, como **TypeScript**, similar a JavaScript pero con unas características más restrictivas y estructuradas en cuanto a la escritura de código.
- 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 del framework Node.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 vamos a utilizar en este curso?

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... Posiblemente los dos primeros (Chrome y Firefox) sean las mejores opciones.

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. En nuestro caso, emplearemos lenguaje PHP, y también JavaScript con el framework Node.js. Para el primero, emplearemos el servidor **Apache**, que instalaremos a través de algún sistema *XAMPP* o similar, que integra Apache con un servidor MySQL/MariaDB y el lenguaje PHP preinstalados. Para lo segundo, instalaremos el framework **Node.js** para construir con él el servidor, y algunos módulos adicionales para facilitar la tarea del desarrollo de la aplicación en sí, como veremos en unidades posteriores.

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 optamos por 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.

Para el **servidor de base de datos**, podemos emplear diferentes alternativas, como MySQL/MariaDB, PostreSQL, Oracle, o SQL Server. También podemos optar por sistemas No-SQL, como MongoDB. 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. En nuestro caso, optaremos por un servidor **MySQL/MariaDB** para la comunicación con PHP y Apache (a través del mencionado XAMPP), y por un servidor **MongoDB** para las aplicaciones que hagamos con Node.js.

IDEs

Para el desarrollo de aplicaciones es necesario contar con un buen entorno de desarrollo o IDE con el que editar el código, depurar y probar las aplicaciones.

Existen multitud de opciones disponibles, la mayoría de ellas gratuitas. Desde entornos de propósito general válidos para muchos lenguajes (Atom, Sublime, Visual Studio Code...) a otros más específicos y orientados a algún lenguaje en concreto, como PhpStorm.

En nuestro caso, optaremos por **Visual Studio Code**, por su versatilidad y popularidad creciente. Permite instalar multitud de extensiones para trabajar con distintos tipos de lenguajes y frameworks, y nos permite también una fácil integración con PHP y Node.js.

Algunos frameworks útiles

También es conveniente conocer algunos frameworks populares y realmente útiles para el desarrollo de aplicaciones web. Estos frameworks se pueden clasificar dependiendo de si se emplean para la parte del cliente o la del servidor. En el primer caso, podemos hablar de Angular, React, Vue y algunos otros (aunque éstos son hoy por hoy los más populares), pero para el apartado que nos interesa, que son los frameworks en el lado del servidor, nos centraremos en:

- Un framework para desarrollo de aplicaciones PHP. En este apartado, los más relevantes son **Laravel** y **Symfony** hoy en día.
- Un framework para el desarrollo de aplicaciones JavaScript. Actualmente sólo existe el framework **Node.js** que, a su vez, permite instalar otros frameworks sobre él para facilitar el desarrollo de aplicaciones. Por ejemplo, utilizaremos el framework **Express**.

6.3. Webs de interés

A continuación se enumeran algunas webs donde consultar o descargar los recursos indicados en este tema, para el desarrollo de aplicaciones en entorno servidor.

- XAMPP - https://www.apachefriends.org/es/index.html
- Apache - https://www.apache.org
- Node.js - https://nodejs.org
- MongoDB - https://www.mongodb.com

- Express - https://expressjs.com
- Laravel - https://laravel.com
- Symfony - https://symfony.com

> **Ejercicio 3:**
>
> Utiliza la herramienta Google Trends para buscar los términos de *Laravel*, *Symfony* y *Node.js*. Deduce a partir de las búsquedas cuál de ellos crees que es más popular actualmente. Después, acude a la web de InfoJobs y busca ofertas de trabajo con estos tres frameworks, para determinar cuál es el más demandado en la actualidad.