SERVIDORES WEB, DNS Y FTP

DESPLIEGUE DE APLICACIONES WEB

CASTILLEJO LOBATO, JOSÉ ANTONIO

CORREA MORALES, NADIA LIA

COLÁS GIL, CHISELA

MILLÁN DÍAZ, ROMÁN

RUIZ JIMÉNEZ, SANDRA

Tabla de contenido

1. ¿QUÉ ES INTERNET?

Internet es un sistema de redes interconectadas mediante distintos protocolos que ofrece una gran diversidad de servicios y recursos, como, el acceso a archivos de hipertexto a través de la web.

1.1. Protocolos de Internet

El éxito de Internet se basa mucho en el empleo de TCP/IP, el conjunto de protocolos de comunicación que permiten el intercambio de información de forma independiente de los sistemas en que ésta se encuentra almacenada. TCP/IP constituye la solución al problema de heterogeneidad de los sistemas informáticos.

Sirven para la transmisión de datos en Internet.

Existen muchos protocolos al establecer una conexión a internet y según el tipo que se necesite establecer, dichos protocolos varían.

Estos incluyen mecanismos para que los dispositivos se identifiquen y establezcan conexiones entre sí, así como reglas de formato que especifican cómo se forman los paquetes y los datos en los mensajes enviados y recibidos.

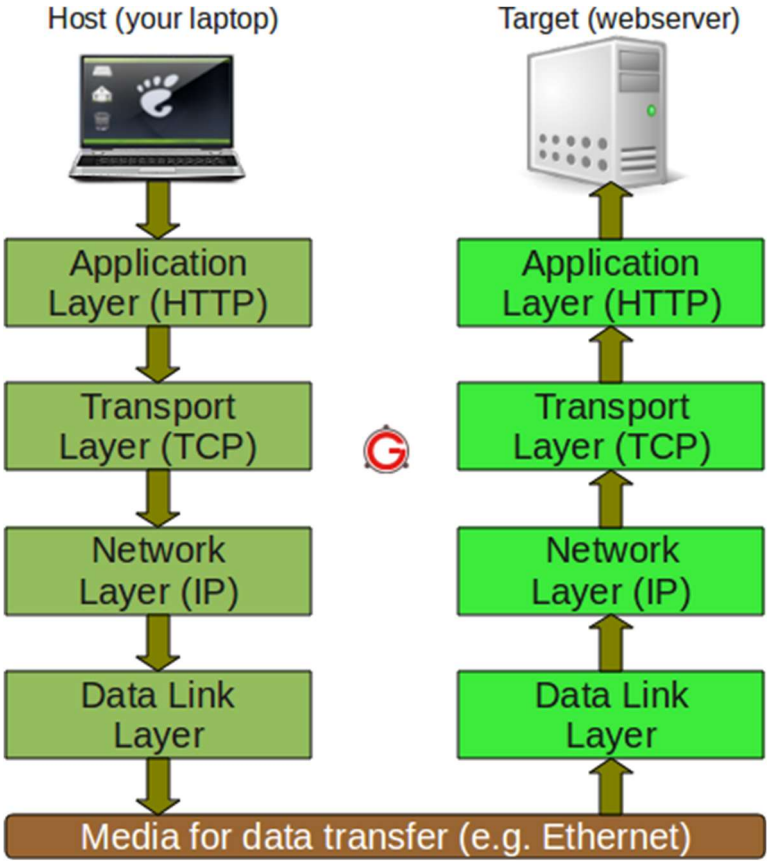
Hex	Protocol Number	Keyword	Protocol
0x00	0	HOPOPT	IPv6 Hop-by-Hop Option
0x01	1	ICMP	Internet Control Message Protocol
0x02	2	IGMP	Internet Group Management Protocol
0x03	3	GGP	Gateway-to-Gateway Protocol
0x04	4	IP-in-IP	IP in IP (encapsulation)
0x05	5	ST	Internet Stream Protocol
0x06	6	TCP	Transmission Control Protocol
0x07	7	CBT	Core-based trees
0x08	8	EGP	Exterior Gateway Protocol
0x09	9	IGP	Interior Gateway Protocol (any private interior gateway, for example Cisco's IGRP)
0x0A	10	BBN-RCC-MON	BBN RCC Monitoring
0x0B	11	NVP-II	Network Voice Protocol
0x0C	12	PUP	Xerox PUP
0x0D	13	ARGUS	ARGUS
0x0E	14	EMCON	EMCON
0x0F	15	XNET	Cross Net Debugger
0x10	16	CHAOS	Chaos
0x11	17	UDP	User Datagram Protocol
0x12	18	MUX	Multiplexing
0x13	19	DCN-MEAS	DCN Measurement Subsystems
0x14	20	HMP	Host Monitoring Protocol
0x15	21	PRM	Packet Radio Measurement

Los protocolos en internet más importantes son TCP (*Transmission-Control-Protocol*) e IP (*Internet Protocol*). De manera conjunta (TCP/IP) podemos enlazar los dispositivos que acceden a la red, algunos otros protocolos de comunicación asociados a internet son POP, SMTP y HTTP.

El conjunto de protocolos TCP/IP está organizado en 4 capas conceptuales que están relacionadas con los niveles OSI. La diferencia reside en que TCP/IP organiza y agrupa las capas de manera diferente, haciendo que sea totalmente adaptable a futuras implementaciones, adaptaciones o normativas.

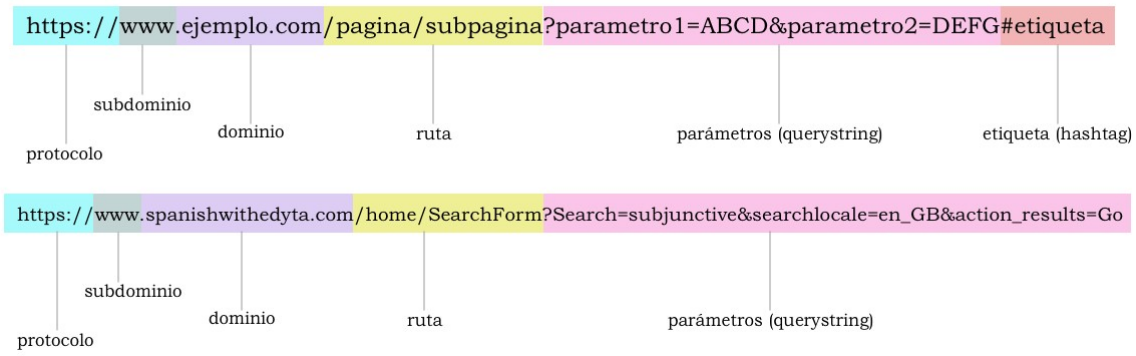
<table>
<tr><th colspan="2">Capas según el modelo OSI</th><th colspan="2">Capas según el modelo TCP/IP</th></tr>
<tr><td>7</td><td>Aplicación
Application</td><td rowspan="3">4</td><td rowspan="3">Aplicación
Process</td></tr>
<tr><td>6</td><td>Presentación
Presentation</td></tr>
<tr><td>5</td><td>sesión
Session</td></tr>
<tr><td>4</td><td>Transporte
Transport</td><td>3</td><td>Transporte
Host-to-Host</td></tr>
<tr><td>3</td><td>Red
Network</td><td>2</td><td>Internet
Network</td></tr>
<tr><td>2</td><td>Enlace de datos
Data Link</td><td rowspan="2">1</td><td rowspan="2">Acceso al medio
Media</td></tr>
<tr><td>1</td><td>Física
Physical</td></tr>
</table>

Existen muchas familias de protocolos asociados a TCP/IP en sus diferentes implementaciones y versiones, de hecho, hay más de 100 protocolos para diferentes funcionalidades.



1.2. URL

URL (*Uniform Resource Locator*) es la dirección única y específica que se asigna a cada uno de los recursos que hay en Internet (sitios web, páginas web, textos, fotos, vídeo…) para que puedan ser localizados por el navegador y visualizados por el usuario.



Protocolo HTTP

Sirve para transferir información entre diferentes actores dentro de Internet.

En teoría, todas las páginas deberían servirse a través del protocolo https://. En práctica, no todos los *webmasters* lo han implementado, http:// no es bueno ni para el posicionamiento, ni para el usuario.

Subdominio

El subdominio de una URL es una extensión del nombre de dominio que se utiliza para organizar diferentes secciones de una web y que funciona de manera independiente a la misma.

Por lo tanto, pueden ser los tres www como en este ejemplo:

https://www.ejemplodesubdominio.com

O un subdominio que tenga que ver con la versión lingüística de una página. Ejejmplo: htttps://es.ejemplodesubdominio.com

Dominio

El dominio está compuesto por dos elementos: el nombre de dominio y la TLD (*Top domain level*).

El dominio es un nombre único que sirve para identificar una página web, lo que significa que el dominio para una web es lo mismo que el número de matrícula para un coche.

Las TLD a veces dan pistas sobre el tipo de página que tenemos delante. Por ejemplo, las páginas .gob o .gov tienen que ver con el gobierno mientras que las .org suelen estar relacionadas con los organismos de educación o las ONGs. Además, la TLD también puede ser internacional (.com), nacional (.es), general (.info, .net), etc.

Ruta

Es lo que viene después de la barra /.

Normalmente indica páginas y subpáginas que podemos encontrar en un sitio web.

Parámetro

En una URL puede haber varios parámetros. Y cuando es el caso, estos se separan con el símbolo ampersand (&).

Los parámetros pueden indicar diferentes cosas. A veces tienen que ver con una búsqueda en el sitio, a veces son parámetros de campañas publicitarias, etc.

Etiqueta

Las etiquetas en una URL aparecen después del hashtag #.

Su función, entre otras cosas, consiste en permitir hacer *scroll* hasta un elemento en concreto. Por ejemplo, si mandamos a alguien una URL que contenga una etiqueta, ésta le dirigirá a la parte exacta de la página en cuestión.

2. EL PROTOCOLO HTTP

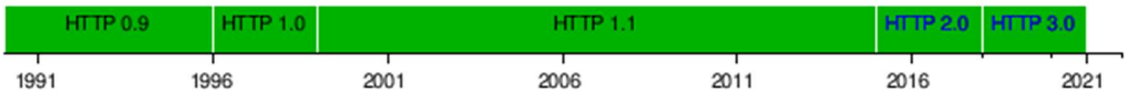
El Protocolo de transferencia de hipertexto (en inglés, *HyperText Transfer Protocol*, abreviado **HTTP**) es el protocolo de comunicación que permite las transferencias de información a través de archivos (XML, HTML…) en la *World Wide Web*.

Cada vez que hacemos clic en un enlace o escribimos una URL y pulsamos *Enter*, el navegador envía una petición al servidor web del sitio al que estamos intentando acceder. El servidor recibe y procesa la solicitud y devuelve los recursos relevantes junto con un encabezado HTTP.

Los códigos de estado de HTTP se entregan al navegador en el encabezado de HTTP. Aunque los códigos de estado se devuelven cada vez que el navegador solicita una página web o un recurso, la mayoría de las veces no son visibles para el usuario.

HTTP es un protocolo sin estado, por lo que no guarda ninguna información sobre conexiones anteriores.

El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las *cookies*, que es información que un servidor puede almacenar en el sistema cliente.



Versiones del protocolo

2.1. Métodos de petición HTTP

HTTP define un conjunto de **métodos de petición** para indicar la acción que se desea realizar para un recurso determinado.

2.1.1. Método GET

El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos, es decir, no deben incluir datos.

Sintaxis: GET /index.html

Petición con cuerpo	No
Respuesta válida con cuerpo	Sí
Seguro	Sí
Idempotente	Sí
Cacheable	Sí
Permitido en HTML forms	Sí

2.1.2. Método HEAD

El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

Sintaxis: HEAD /index.html

Petición con cuerpo	No
Respuesta válida con cuerpo	No
Seguro	Sí
Idempotente	Sí
Cacheable	Sí
Permitido en HTML forms	No

2.1.3. Método POST

El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

El tipo del cuerpo de la solicitud es indicado por la cabecera Content-Type.

Como se describe en la especificación HTTP 1.1, el método POST está diseñado para permitir un método uniforme que cubra las siguientes funciones:

- Modificación de recursos existentes.

- Publicar un mensaje en un tablón de anuncios, grupo de noticias, lista de correos, o grupos similares de artículos;
- Agregar un nuevo usuario a través de un modal de suscripciones;
- Proveer un conjunto de datos, como resultado del envío de un formulario, a un proceso *data-handling*.
- Extender una base de datos a través de una operación de concatenación.

Sintaxis: POST /index.html

Petición con cuerpo	**Sí**
Respuesta válida con cuerpo	Sí
Seguro	No
Idempotente	No
Cacheable	Solo si incluye nueva información
Permitido en HTML forms	Sí

2.1.4. Método PUT

El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

Crea un nuevo elemento o reemplaza una representación del elemento de destino con los datos de la petición.

La diferencia entre el método PUT y el método POST es que PUT es un método idempotente: llamarlo una o más veces de forma sucesiva tiene el mismo efecto (sin efectos secundarios), mientras que una sucesión de peticiones POST idénticas pueden tener efectos adicionales, como enviar una orden varias veces.

Sintaxis: PUT /nuevo.html HTTP/1.1

Petición con cuerpo	**Sí**
Respuesta válida con cuerpo	No
Seguro	No
Idempotente	Sí
Cacheable	No
Permitido en HTML forms	No

2.1.5. Método PATCH

El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.

A diferencia de PUT, el método PATCH no es idempotente, esto quiere decir que peticiones idénticas sucesivas pueden tener efectos diferentes. Sin embargo, es posible emitir peticiones PATCH de tal forma que sean idempotentes.

PATCH (al igual que POST) puede provocar efectos secundarios a otros recursos.

Para averiguar si un servidor soporta PATCH, el servidor puede notificar su compatibilidad al añadirlo a la lista en el *header*: *Allow* o *Access-Control-Allow-Methods* (para CORS).

Otra indicación (implícita) de que las peticiones PATCH son permitidas, es la presencia del *header*: *Accept-Patch*, el cual especifica los formatos de documento patch aceptados por el servidor.

Sintaxis: PATCH /file.txt HTTP/1.1

Petición con cuerpo	**Sí**
Respuesta válida con cuerpo	Sí
Seguro	No
Idempotente	No
Cacheable	No
Permitido en HTML forms	No

2.1.6. Método DELETE

El método DELETE borra un recurso en específico.

Sintaxis: DELETE /file.html HTTP/1.1

Petición con cuerpo	**Quizás**
Respuesta válida con cuerpo	Quizás
Seguro	No
Idempotente	Sí
Cacheable	No
Permitido en HTML forms	No

2.1.7. Método CONNECT

El método CONNECT es un método de salto entre servidores. Establece un túnel hacia el servidor identificado por el recurso.

Inicia la comunicación en dos caminos con la fuente del recurso solicitado. Puede ser usado para abrir una comunicación túnel.

Por ejemplo, el método CONNECT puede ser usado para acceder a sitios web que usan SSL (HTTPS).

Sintaxis: CONNECT www.example.com:443 HTTP/1.1

Petición con cuerpo	**No**
Respuesta válida con cuerpo	Sí
Seguro	No
Idempotente	No
Cacheable	No
Permitido en HTML forms	No

2.1.8. Método OPTIONS

El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.

Solicita opciones de comunicación permitidas para una URL o servidor determinado. Un cliente puede especificar una URL con este método, o un asterisco (*) para referirse a todo el servidor.

Sintaxis: OPTIONS /index.html HTTP/1.1

OPTIONS * HTTP/1.1

Petición con cuerpo	**No**
Respuesta válida con cuerpo	Sí
Seguro	Sí
Idempotente	Sí
Cacheable	No
Permitido en HTML forms	No

2.1.9. Método TRACE

El método TRACE efectúa una prueba de bucle de mensaje por el camino al recurso objetivo proporcionando un útil mecanismo de *debugging*.

El destino final de la petición debería devolver el mensaje recibido, excluyendo algunos de los campos descritos abajo, de vuelta al cliente como el mensaje *body* y una respuesta 200 (OK) con un *Content-Type* de message/http.

El destinatario final es o el servidor de origen o el primer servidor en recibir un Max-Forwards de valor 0 en la petición.

Sintaxis: TRACE /index.html

Petición con cuerpo	No
Respuesta válida con cuerpo	No
Seguro	Sí
Idempotente	Sí
Cacheable	No
Permitido en HTML forms	No

2.2. Códigos de estado HTTP

1xx	**Códigos informativos** que indican que la solicitud iniciada por el navegador continúa.
100	Continue
101	Switching Protocols
102	Processing WebDAV
103	Checkpoint draft POST PUT
122	Reques-t-URI too long IE7

2xx	**Los códigos con éxito** regresaron cuando la solicitud del navegador fue recibida, entendida y procesada por el servidor.
200	OK
201	Created
202	Accepted
203	Non-Au-tho-rit-ative Inform-ation 1.1
204	No Content
205	Reset Content
206	Partial Content

207	Multi--Status [WebDAV 4918]
208	Already Reported [WebDAV 5842]
226	IM Used [3229 GET]

3xx	**Códigos de redireccionamiento** devueltos cuando un nuevo recurso ha sido sustituido por el recurso solicitado.
300	Multiple Choices
301	Moved Perman-ently
302	Found
303	See Other [1.1]
304	Not Modified
305	Use Proxy [1.1]
306	Switch Proxy [unused]
307	Temporary Redirect [1.1]
308	Permanent Redirect [7538]

4xx	**Códigos de error del cliente** que indican que hubo un problema con la solicitud.
400	Bad Request
401	Unauth-orized
402	Payment Required [res]
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authen-tic-ation Required

408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precon-dition Failed
413	Request Entity Too Large
414	Reques-t-URI Too Long
415	Unsupp-orted Media Type
416	Requested Range Not Satisf-iable
417	Expect-ation Failed
418	I'm a teapot [2324]
422	Unproc-essable Entity [WebDAV 4918]
423	Locked [WebDAV 4918]
424	Failed Dependency [WebDAV 4918]
425	Unordered Collection [3648]
426	Upgrade Required [2817]
428	Precon-dition Required [draft]
429	Too Many Requests [draft]
431	Request Header Fields Too Large [draft]
444	No Response [nginx]
449	Retry With [MS]
450	Blocked By Windows Parental Controls [MS]
451	Unavai-lable For Legal Reasons [draft]
499	Client Closed Request [nginx]

5xx	**Códigos de error del servidor** que indican que la solicitud fue aceptada, pero que un error en el servidor impidió que se cumpliera
500	Internal Server Error
501	Not Implem-ented
502	Bad Gateway
503	Service Unavai-lable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates [2295]
507	Insuff-icient Storage [WebDAV 4918]
508	Loop Detected [WebDAV 5842]
509	Bandwidth Limit Exceeded [nostd]
510	Not Extended [2774]
511	Network Authen-tic-ation Required [draft]
598	Network read timeout error [nostd]
599	Network connect timeout error [nostd]

Comentarios

WebDAV	Extensión WebDAV
1.1	HTTP/1.1
GET, POST, PUT, POST	Solo para estos métodos
IE	Extensión IE
MS	Extensión MS
nginx	Extensión nginx
2518, 2817, 2295, 2774, 3229, 4918, 5842	Número RFC
draft	Draft propuesto
nostd	Extensión no estándar

res	Reservado para uso futuro
unused	Ya no se usa, deprecado

2.2.1. Los códigos de estado HTTP y los motores de búsqueda (SEO)

Los robots de los motores de búsqueda ven los códigos de estado HTTP mientras rastrean tu sitio. En algunos casos, estos mensajes pueden influir en el hecho de que las páginas web sean indexadas y en la forma en que los motores de búsqueda perciben la salud del sitio web.

En general, los códigos de estado HTTP de niveles 100 y 200 no tendrán mucho impacto en tu SEO.

Las respuestas de nivel 400 y 500 pueden evitar que los bots rastreen e indexen las páginas web. Demasiados de estos errores también pueden indicar que el sitio no es de alta calidad, lo que posiblemente baje la clasificación.

2.3. MIME

El tipo Extensiones multipropósito de Correo de Internet (MIME, del inglés *Multipurpose Internet Mail Extensions*) es una forma estandarizada de indicar la naturaleza y el formato de un documento, archivo o conjunto de datos.

Los navegadores suelen usar el tipo MIME, en vez de la extensión de archivo, para determinar cómo procesará un documento. Por eso es importante que los servidores estén configurados correctamente para adjuntar el tipo MIME correcto al encabezado del objeto de respuesta.

Una parte importante del MIME está dedicada a mejorar las posibilidades de transferencia de texto en distintos idiomas y alfabetos.

2.3.1. Estructura

La estructura de un tipo MIME es muy simple; consiste en un tipo y un subtipo, dos cadenas, separadas por un '/'. No se permite espacio. El *tipo* representa la categoría y puede ser de tipo *discreto* o *multiparte*. El *subtipo* es específico para cada tipo.

Un tipo MIME no distingue entre mayúsculas y minúsculas, pero tradicionalmente se escribe todo en minúsculas.

2.3.2. Tipos discretos

Tipo	Descripción	Ejemplo de subtipos típicos
text	Representa cualquier documento que contenga texto y es teóricamente legible por humanos.	text/plain text/html text/css text/javascript
image	Representa cualquier tipo de imagen. Los videos no están incluidos, aunque las imágenes animadas (como el gif animado) se describen con un tipo de imagen.	image/gif image/png image/jpeg image/bmp image/webp
audio	Representa cualquier tipo de archivos de audio.	audio/midi audio/mpeg audio/webm audio/ogg audio/wav
video	Representa cualquier tipo de archivos de video.	video/webm video/ogg
application	Representa cualquier tipo de datos binarios.	application/octet-stream application/pkcs12 application/vnd.mspowerpoint application/xhtml+xml application/xml application/pdf

Para documentos de texto sin subtipo específico, se debe usar text/plain.

De forma similar, para los documentos binarios sin subtipo específico o conocido, se debe usar application/octet-stream.

La mayoría de los servidores web envían recursos de tipo desconocido utilizando el tipo MIME predeterminado application/octet-stream. Por razones de seguridad, la mayoría de los navegadores no permiten establecer una acción predeterminada personalizada para dichos recursos, lo que obliga al usuario a almacenarlo en el disco para usarlo.

2.3.3. Tipos multiparte

Los tipos de multiparte indican una categoría de documento que está rota en distintas partes, a menudo con diferentes tipos de MIME.

Es una forma de representar un documento compuesto.

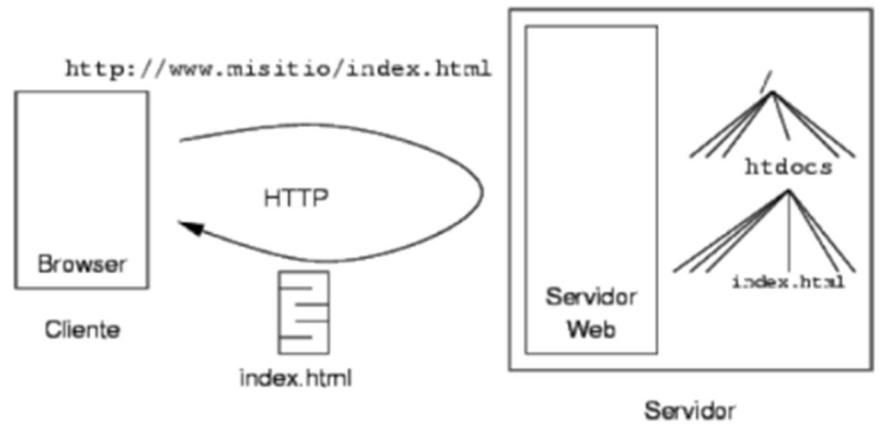
Con la excepción de multipart/form-data, que se utilizan en relación con formularios HTML y el método POST, y multipart/byteranges, que se utilizan junto con el mensaje de estado de Contenido Parcial 206 para enviar solo un subconjunto de un documento completo, HTTP no maneja documentos multiparte de una manera específica: el mensaje simplemente se transmite al navegador, que probablemente propondrá una ventana Guardar como…, sin saber cómo mostrar el documento en línea.

3. ARQUITECTURAS WEB

Una aplicación web es un tipo especial de aplicación cliente/ servidor, donde tanto el cliente como el servidor y el protocolo de comunicación están estandarizados. Esta es proporcionada por un servidor Web y utilizada por usuarios que se conectan desde cualquier punto vía clientes Web como son los browsers o navegadores.

La arquitectura de un Sitio Web tiene tres componentes principales:

- Un servidor Web, es un programa que está esperando permanentemente las solicitudes de conexión por parte de los clientes y distribuye páginas de información formateada. Sus principales funciones son: procesar los requerimientos de la base de datos, formatear los datos para transmitirlos al cliente y procesar la lógica de la aplicación.
- Una conexión de red por la que se realizan los requerimientos mediante el uso del protocolo HTTP.
- Uno o más clientes, es un programa con el que interactúa el usuario para solicitar el envío de recursos al servidor. El cliente web interpreta los recursos como puede ser las páginas HTML, imágenes, sonidos, etc. Sus principales funciones son: administrar la interfaz de usuario, hacer validaciones locales y recibir resultados y formatearlos.



Esquema básico de una aplicación web

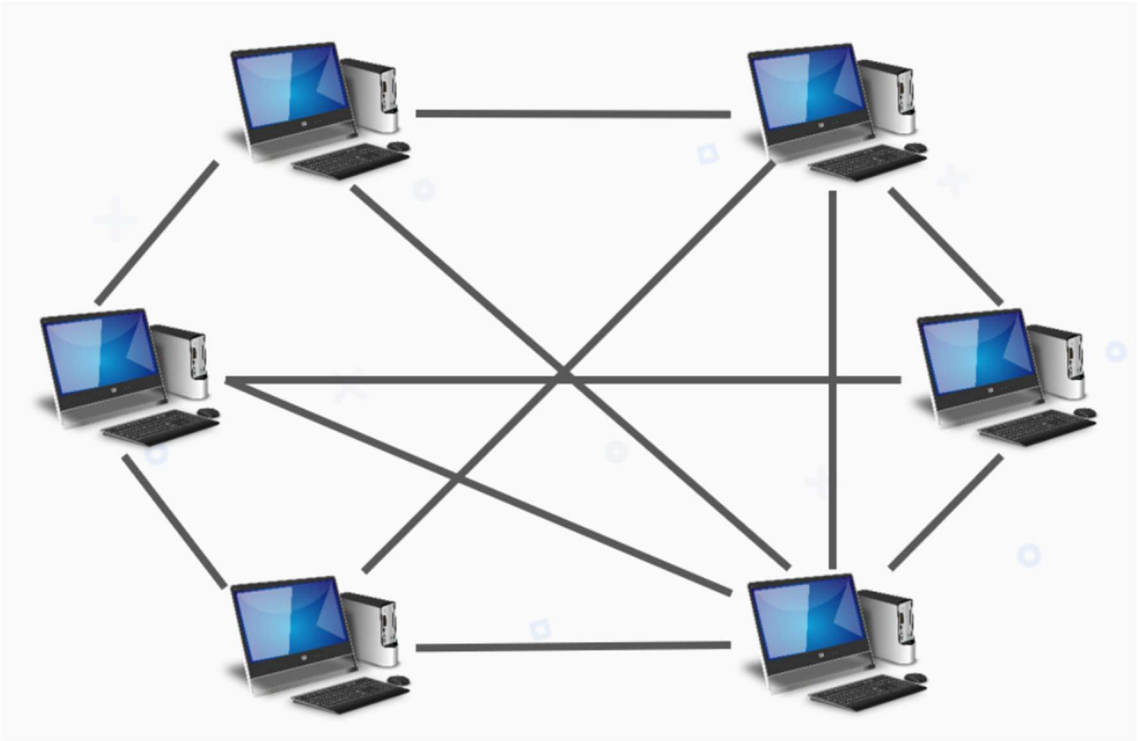
3.1. Tipos de Arquitectura Web

Pese a que las aplicaciones web se basan en una arquitectura cliente/servidor, existen diversas variantes según la implementación de las funcionalidades de la parte servidor:

3.1.1. Arquitectura Peer to Peer (P2P) o Punto a Punto

Esta arquitectura hace referencia a una red de computadoras donde todos los dispositivos conectados a la red actúan como cliente y servidor al mismo tiempo. En esta arquitectura no es necesario un servidor central que administre la red (aunque puede existir), si no que todos los nodos de la red pueden comunicarse entre sí.

Muchos la consideran como una variante de la arquitectura Cliente-Servidor mientras otros establecen esto no es así ya que la arquitectura Cliente-Servidor tiene como punto medular la centralización, mientras la arquitectura P2P busca la descentralización.



Arquitectura Peer to Peer (P2P)

Entre las ventajas que tiene esta arquitectura podemos incluir:

- Alta escalabilidad: a mayor número de nodos, más recursos hay disponibles y la carga de trabajo se divide entre todos los participantes.

- Tolerancia a fallos: al tener los recursos distribuidos por varios nodos permite que los recursos estén disponibles sin importar si algunos de los nodos se caen.
- Descentralización: Los sistemas P2P puros tienen una autonomía completa, por lo que pueden funcionar sin un servidor central que organice toda la red.
- Privacidad: Debido al sistema de búsqueda por inundación o tablas hash, es posible tener un alto nivel de privacidad, pues no es tan fácil saber qué nodo fue el que realizó la petición inicial.
- Equilibrio de carga: En una arquitectura P2P, todos los nodos de la red tienen el mismo rol y responsabilidades, por lo que toda la carga de trabajo se equilibra entre todos los nodos de la red.

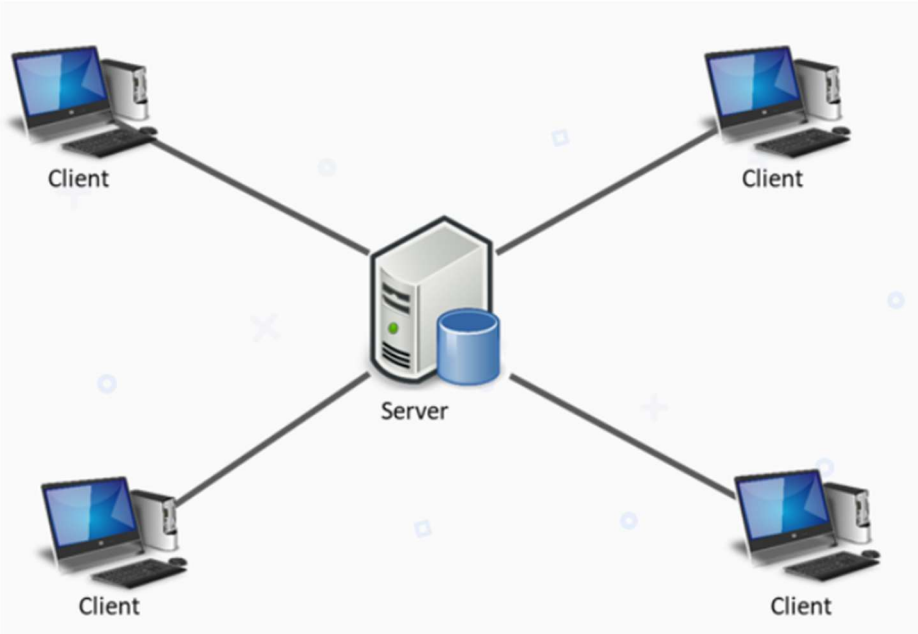
Así mismo esta arquitectura presenta las siguientes desventajas:

- Alta complejidad: tiene una complejidad muy alta para desarrollarse, ya que tenemos que crear aplicaciones que funcionen como cliente y servidor, al mismo tiempo que tenemos que garantizar que las búsquedas de los recursos sean eficientes.
- Control: es muy fácil perder el control sobre el contenido que se puede compartir en la red, a menos que se cuente con un servidor central para administrar los recursos que se buscan y los usuarios que se conectan a la red.
- Seguridad: Esta es una de las características menos implementadas en este tipo de arquitecturas, para evitar la intercepción de las comunicaciones, nodos maliciosos, contenido falso o adulterado o la propagación de virus o programas maliciosos.
- Tráfico: En redes no optimizadas como las no estructuradas, podemos saturar la red con una gran cantidad de tráfico para propagar las peticiones a los nodos adyacentes.

3.1.2. Arquitectura Cliente-Servidor

Una de las arquitecturas más conocidas, que está compuesta por dos componentes, el proveedor, que brinda una serie de servicios o recursos; y el consumidor, quien consume dichos servicios y recursos.

En esta arquitectura existe un servidor y múltiples clientes que se conectan al servidor para recuperar todos los recursos necesarios para funcionar.



Arquitectura Cliente-Servidor

El esquema de funcionamiento básico sería el siguiente:

1. Desde el navegador web (cliente) el usuario solicita la carga de una página web indicando su URL.
2. El servidor recibe la petición de la página web.
3. Busca en su sistema de almacenamiento la página solicitada.
4. Envía el contenido de la página web por el mismo medio por el que recibió la petición.
5. El navegador web recibe el código de la página y lo interpreta mostrando al usuario la página web.

En este sentido, actúa como una aplicación distribuida que particiona tareas o cargas de trabajo entre el servidor, proveedor de un recurso o servicio, y los clientes, solicitantes del servicio. Del mismo modo es considerada una arquitectura distribuida debido a que el servidor y el cliente se encuentran distribuidos en diferentes equipos (aunque podrían estar en la misma máquina) y se comunican únicamente por medio de la red o Internet.

Entre las principales ventajas de este tipo de arquitectura tenemos:

- Centralización del control: tanto los accesos, recursos y la integridad de los datos son controlados por el servidor. El servidor es la única fuente de la verdad, lo que impide que los clientes conserven información desactualizada.
- Seguridad: El servidor por lo general está protegido por firewall o subredes que impiden que los atacantes puedan acceder a la base de datos o los recursos sin

pasar por el servidor. De esta forma un programa cliente defectuoso o no autorizado no puede dañar el sistema.

- Fácil de instalar (cliente): El cliente es por lo general una aplicación simple que no tiene dependencias, por lo que es muy fácil de instalar.
- Separación de responsabilidades: La arquitectura cliente-servidor permite implementar la lógica de negocio de forma separada del cliente.
- Portabilidad: Una de las ventajas de tener dos aplicaciones es que podemos desarrollar cada parte para correr en diferentes plataformas, por ejemplo, el servidor solo en Linux, mientras que el cliente podría ser multiplataforma.

En lo que respecta a las desventajas son las siguientes:

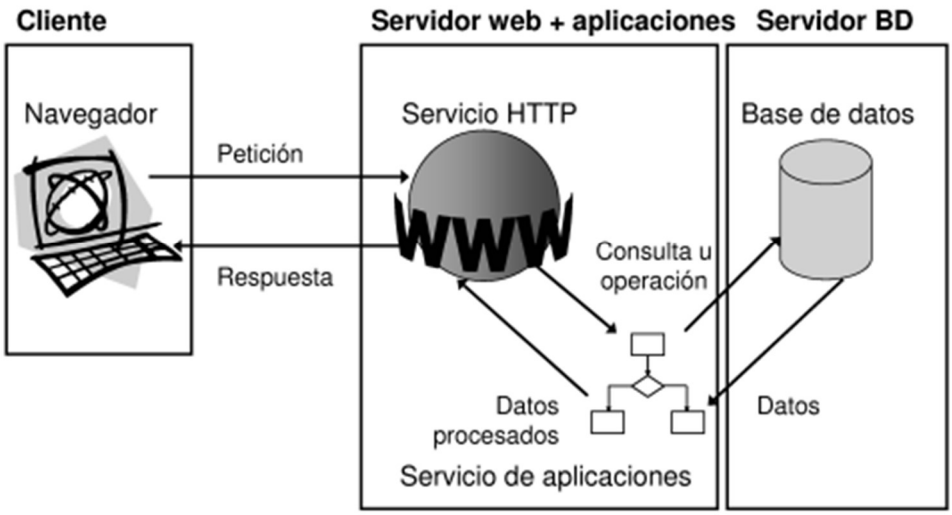
- Actualizaciones (clientes): pueden ser difícil de gestionar ya que puede haber muchos terminales con el cliente instalado y tenemos que asegurar que todas sean actualizadas cuando salga una nueva versión.
- Concurrencia: una gran cantidad de clientes que envían peticiones simultáneas al mismo servidor puede ser un problema para el servidor, quien tendrá que atender todas las peticiones de forma simultánea, aunque se puede mitigar con una estrategia de escalamiento, lo debemos de tener en cuenta.
- Todo o nada: en caso que el servidor se caiga, todos los clientes quedarán totalmente inoperables.
- Protocolos de bajo nivel: para establecer comunicación entre el cliente y el servidor. Se suele emplear Sockets, HTTP, RPC, etc. Lo que puede implicar un reto para los desarrolladores.
- Depuración: debido a que los clientes están distribuidos en diferentes máquinas, incluso, equipos a los cuales no tenemos acceso, se hace complicado recopilar la traza de un error.

3.1.3. Arquitectura con Servidor de Aplicaciones

En este tipo de arquitectura se separa la lógica de datos y los datos a un servidor de base de datos específico. Un servidor de aplicaciones es el corazón de un gran sistema distribuido y proporciona servicios que soportan la ejecución y disponibilidad de las aplicaciones desplegadas.

Al separar lógica de datos y los datos se permite que la función que realiza el servidor de aplicaciones sea diferente debido a que los recursos que va a manipular no son archivos estáticos, sino que contienen el código que tiene que ejecutar.

Con esta arquitectura un servidor web en solitario enviaría al cliente el recurso solicitado tal cual, mientras que el servidor de aplicaciones lo ejecuta y envía al cliente el resultado a través del servidor web.



Arquitectura con Servidor de Aplicaciones

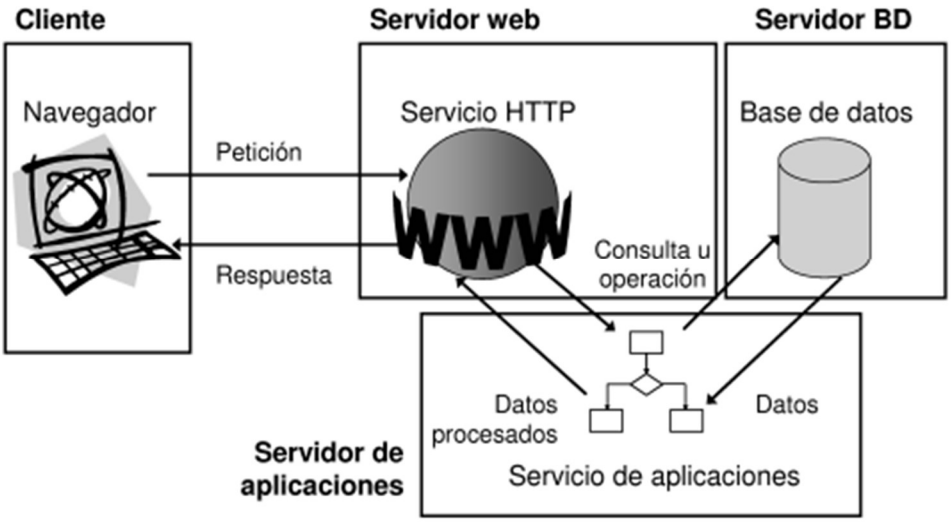
Tal y como podemos ver en la imagen, el servidor web y el servidor de aplicaciones pueden residir en una misma máquina. El servidor web recibe la petición de un recurso, y si éste corresponde a un recurso dinámico, transfiere la parte correspondiente al servidor de aplicaciones el cual devolverá de nuevo al servidor web el recurso ejecutado. Finalmente será el servidor web el que envíe al cliente el resultado final.

Es muy frecuente que el servidor de aplicaciones deba conectarse con una base de datos para obtener los datos solicitados durante la ejecución del código. Dicha base de datos puede residir en la misma máquina que el servidor web o en otro host conectado en red.

3.1.4. Arquitectura con Servidor de Aplicaciones Externo

En este tipo de arquitectura las tres funcionalidades básicas del servidor web se separan en tres servidores específicos, por lo que la parte correspondiente al servidor web suele tener menos carga de trabajo que el servidor de aplicaciones.

Un redirector es el encargado de transferir al servidor de aplicaciones los elementos que necesitan ser ejecutados, y esté será el que se comunique con la base de datos.

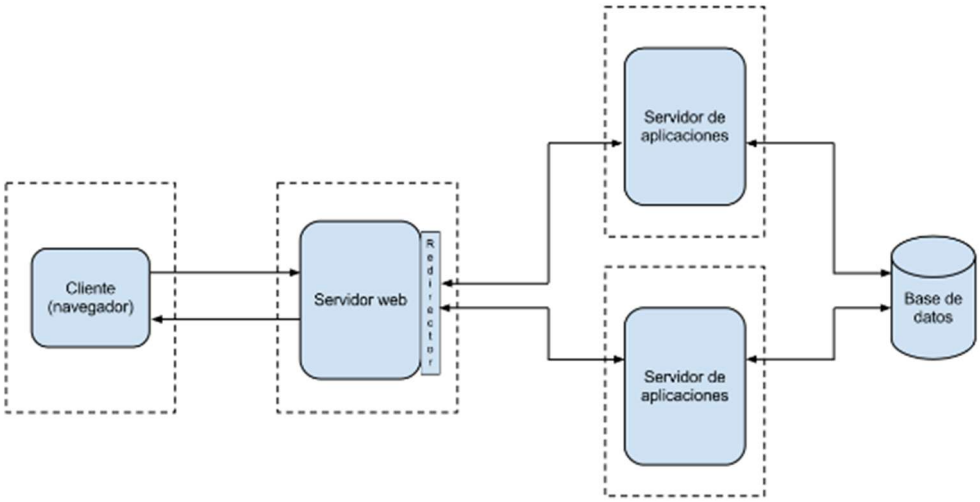


Arquitectura con Servidor de Aplicaciones Externo

3.1.5 Arquitectura con Varios Servidores de Aplicaciones

Este tipo de arquitectura es usada cuando se estima que la carga de trabajo del servidor de aplicaciones será elevada. Por ello se suele implantar un sistema con varios servidores de aplicaciones unidos a un mismo servidor web que requiere menos rendimiento.

La conexión se realizará a través de un redirector, que además realizará las funciones de balanceador de carga para determinar en un determinado momento qué servidor de aplicaciones debe ejecutar el código en función de la carga de trabajo que tenga cada uno. En este caso, todos los servidores de aplicaciones deben ser iguales.



Arquitectura con Varios Servidores de Aplicaciones

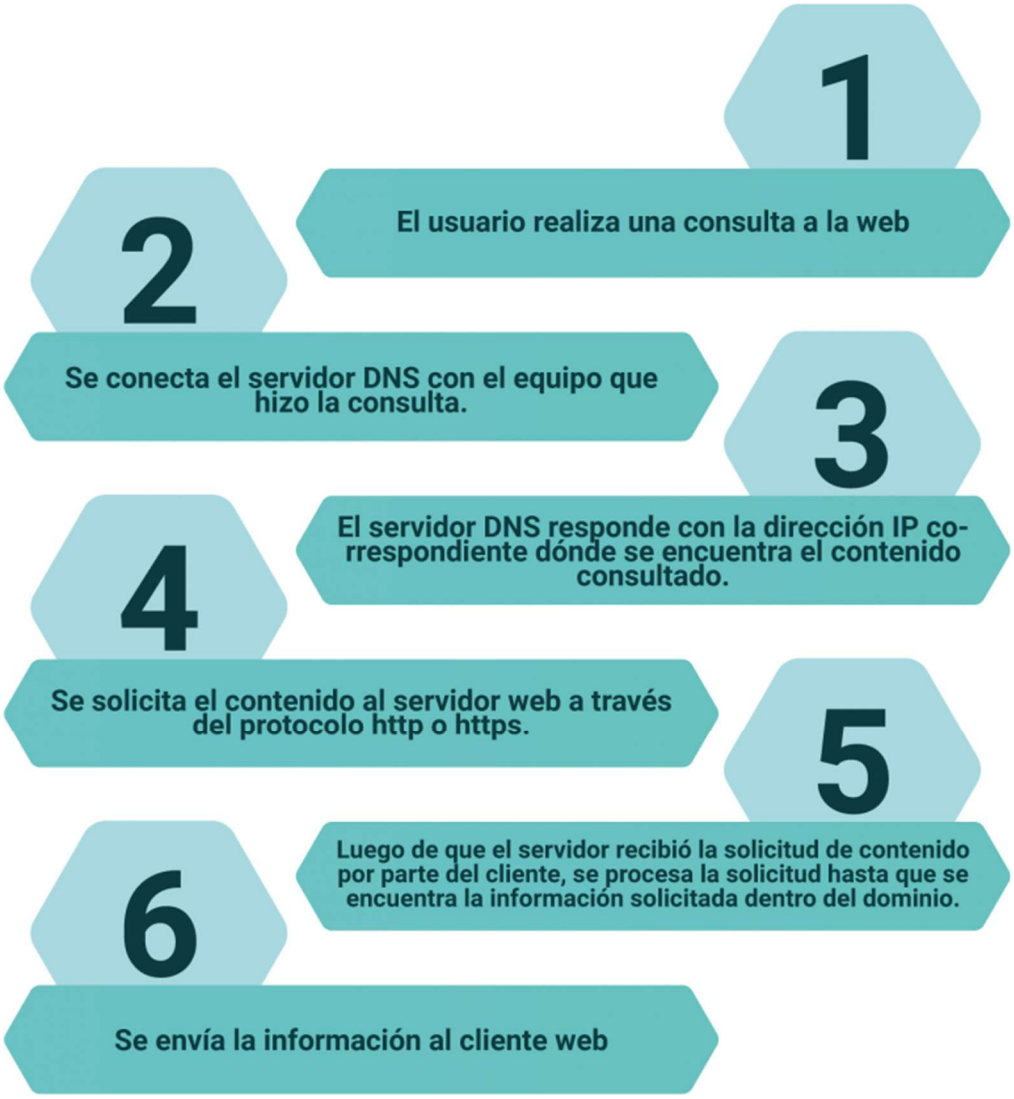
4. SERVIDORES WEB

Un servidor web o servidor HTTP es un software que forma parte del navegador.

Este tipo de servidores son utilizados para la distribución y entrega de contenido web, sea en redes internas o directamente en Internet. Su función es mostrar la información (páginas web) cuando los usuarios la solicitan, es decir, permiten que los internautas visualicen una página web desde su navegador.

4.1. ¿Cómo funcionan los servidores web?

Después de delimitar el protocolo de comunicación (HTTP o HTTPS) y el tipo de petición (GET o POST), el proceso de búsqueda-resultado es el siguiente:



4.2. Servidores web populares

Al principio el servidor web más popular fue Apache, pero con el paso del tiempo llegaron opciones más rápidas y con mayor capacidad, por lo cual es posible elegir entre toda gama de servidores web, cada uno con ciertas características y propiedades, por lo cual es recomendable elegir el que se adapte mejor a las necesidades que se tiene.

4.2.1. Apache HTTP

- Es uno de los servidores más usados.
- Software de código abierto y gratuito.
- Funciones incorporadas para autenticación y validación de usuarios.
- Arquitectura basada en módulos.
- Es multiplataforma.
- Es compatible con apps.
- Es flexible y personalizable, por lo cual se pueden activar o desactivar sus funcionalidades.
- Fácil de configurar e instalar.
- Ofrece seguridad por los módulos de autorización y autenticación.

4.2.2. Nginx

- Es uno de los servidores web más usados.
- Software de código abierto y gratuito.
- Arquitectura basada en eventos.
- Alto rendimiento.
- Gestiona sitios web de alto tráfico.
- Funciona bien con PHP–FPM.
- Bajo consumo de batería.
- Multiplataforma.
- Cuenta con una versión de pago.

4.2.3. Microsoft ISS

- Desarrollado por Microsoft, fue creado solamente para el sistema operativo Windows.
- Se ejecuta gracias a la tecnología ISS (*Internet Information Services*).
- Arquitectura modular.
- Compatible con páginas programas en ASP o .net.
- Se puede configurar para utilizar PHP, VBScript, Perl y Java.
- Integración de lenguajes y tecnologías limitada.
- Modelo de proceso único: un solo proceso maneja todas las peticiones.
- Es posible agregar funciones adicionales.
- Cuenta con múltiples características de seguridad y mecanismos de autenticación integrados.

5. DNS

5.1. Estructura, nomenclatura y funcionalidad de DNS

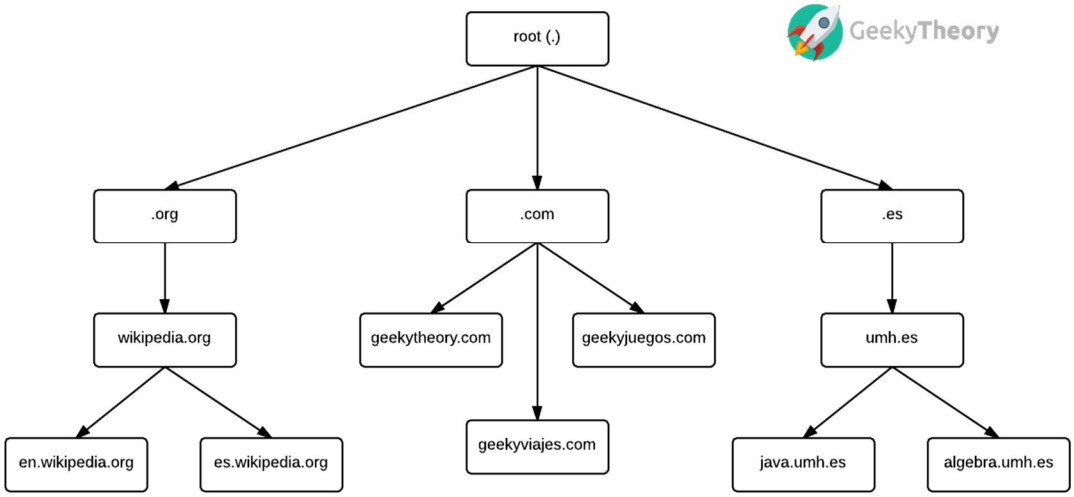
Las personas accedemos a la información en internet a través de nombres de dominio, que son direcciones en lenguaje humano tales como www.google.com, mientras que los navegadores interactúan con la web mediante direcciones IP numéricas como 192.0.2.1. El sistema DNS (Domain Name System) es el encargado de traducir los dominios a sus correspondientes direcciones IP.

Cada dispositivo de internet tiene una dirección IP única que lo identifica del resto. Esta dirección es necesaria para que el navegador identifique quién hace una consulta, entre otras cosas para verificar si tiene acceso, para poder realizarla con éxito.

En DNS encontramos una estructura jerárquica de árbol con nodos enlazados. Cada uno representa un nivel del espacio de nombres.

Siguiendo la imagen, encontramos la raíz (root) que en este caso, se representa con un punto. Debajo del root, nos encontramos los TLDs (Top Level Domains, dominios de nivel superior). Estos son los conocidos .com, .es, .ar, .mx, .org, etc.

A los TLDs les siguen en la estructura los TLDs de segundo nivel (también conocidos como SLD, Second Level Domain o 2LD). Aquí se podría representar el nombre del sitio web, por ejemplo SLD (wikipedia). TLD(org). Esto sucesivamente hasta llegar a un nodo final que representará el recurso. Por ejemplo, drive.google.com cuenta con primer nivel (com), segundo (google) y tercero (drive).



Cada nivel (elemento entre puntos) puede tener como máximo 63 caracteres siendo indiferente el uso de mayúsculas o minúsculas pero sin sobrepasar un total de 255 en el nombre de dominio completo.

5.2. Instalación y configuración de DNS

El servidor DNS de software libre más conocido es DNS BIND. Para su instalación haremos simplemente ejecutamos el siguiente comando:

```
$ apt-get install bind9 bind9utils
```

Después de la instalación podemos comprobar que el servidor bind9 está activo con server bind9 status. Ahora existe en nuestro ordenador la ruta /etc/bind, y un fichero de configuración por defecto /etc/bind/named.conf que especifica los roles que tendrá el servidor donde:

- /etc/bind/named.conf.options: hace referencia al archivo de configuración que posee opciones genéricas.
- /etc/bind/named.conf.local: hace referencia al archivo de configuración para opciones particulares.
- /etc/bind/named.conf.default-zones: hace referencia al archivo de configuración de zonas

Los servidores DNS pueden configurarse de distintas maneras para llevar a cabo uno o varios roles a la vez, como:

- Servidor maestro/primario de una o varias zonas.
- Servidor esclavo/secundario de una o varias zonas.
- Servidor DNS caché
- Servidor de reenvío de servicios.

Una vez instalado DNS BIND, podemos configurarlo de diferentes maneras, dependiendo de qué tipo de servidor queramos tener como hemos visto arriba.

A continuación, vamos a ver tres ejemplos de configuración.

5.2.1 Configuración de servidor caché

Aunque todos los servidores DNS son servidores caché, existe la posibilidad de que un servidor DNS funcione solamente como servidor caché, sin que sea maestro o esclavo.

Para configurarlo haremos lo siguiente:

1. Verificar que el contenido del fichero /etc/bind/named.conf.options, tras la instalación, es el siguiente:

```
options {
 directory "/var/cache/bind";

 // If there is a firewall between you and nameservers you want
 // to talk to, you may need to fix the firewall to allow multiple
 // ports to talk. See http://www.kb.cert.org/vuls/id/800113

 // If your ISP provided one or more IP addresses for stable
 // nameservers, you probably want to use them as forwarders.
 // Uncomment the following block, and insert the addresses replacing
 // the all-0's placeholder.

 //forwarders {
 0.0.0.0;
 };

 auth-nxdomain no; # conform to RFC1035
 listen-on-v6 { any; };
};
```

2. Modificar el fichero /etc/resolv.conf para que solamente tenga activa la siguiente línea:

   nameserver 127.0.0.1

   De tal forma que ahora el servidor DNS activo solamente es el local, que tienes configurado como caché.

3. Una vez efectuados los cambios recargas el servidor con el comando: service bind9 reload ó /etc/init.d/bind9 reload.

5.2.2 Configuración de servidor maestro

Para configurar un servidor DNS como maestro, modificaremos el archivo /etc/bind/named.conf.local realizando el siguiente procedimiento:

1. Configurar el fichero /etc/bind/named.conf.local para indicar: qué zonas son servidas por el servidor, qué zonas son servidas como master y el fichero donde se guarda el contenido de la zona. Por ejemplo:

```
//zonas creadas tipo master
zone "ejemplo.com" {
 type master;
 file "/var/lib/bind/master/db.ejemplo.com.hosts";
};
```

En este ejemplo, el servidor sirve el dominio "ejemplo.com" como master, y la zona se guarda en el fichero /var/lib/bind/master/db.ejemplo.com.hosts.

2. Configurar el fichero /var/lib/bind/master/db.ejemplo.com.hosts para agregar los registros RR a la zona, por ejemplo:

```
;
; BIND Database file for ejemplo.com zone
;

@ IN SOA ejemplo.com. hostmaster.ejemplo.com. (
 2011091601 ; serial number
 3600 ; refresh
 600 ; retry
 1209600 ; expire
 3600 ) ; default TTL
;
 IN NS ns.ejemplo.com.
 IN MX 10 mail.ejemplo.com.
 IN TXT ( "v=spf1 mx ~all" )
;

localhost A 127.0.0.1
ns A 192.168.200.250
mail A 192.168.200.251
www A 192.168.200.252
```

3. Recargas el servidor con el comando: service bind9 reload ó /etc/init.d/bind9 reload.

4. Realizas la siguiente consulta: dig ejemplo.com obteniendo una salida similar a la siguiente:

```
; <<>> DiG 9.7.3 <<>> ejemplo.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25588
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;ejemplo.com. IN A

;; AUTHORITY SECTION:
ejemplo.com. 3600 IN SOA ejemplo.com. hostmaster.ejemplo.com. 2011091601 3600 600 1209600
3600

;; Query time: 3 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Sep 27 11:48:36 2011
;; MSG SIZE rcvd: 76
```

5.2.3 Configuración de servidor secundario

Para configurar un servidor DNS como secundario o esclavo, modificaremos el archivo /etc/bind/named.conf.local realizando el siguiente procedimiento:

1. Configurar el fichero /etc/bind/named.conf.local del servidor esclavo para indicar: qué zonas son servidas por el servidor, qué zonas son servidas como slave, la IP del servidor master (de donde se transferirá la zona cuando se reciba

una notificación de cambio, o se supere el TTL de la zona) y el fichero donde se guarda el contenido de la zona. Por ejemplo:

```
//zonas creadas tipo esclavo
zone "ejemplo.com" {
 type slave;
 masters {
 192.168.200.250;
 };
 file "/var/lib/bind/slave/db.ejemplo.com.hosts";
};
```

En este ejemplo, el servidor sirve el dominio "ejemplo.com" como slave, y la zona se guarda en el fichero /var/lib/bind/slave/db.ejemplo.com.hosts.

2. En el servidor maestro configuras la sección correspondiente al servidor master en el fichero /etc/bind/named.conf.local:

    a. Para indicar qué servidores tienen permitido la transferencia de los ficheros de zona, mediante la directiva allow-transfer. Por ejemplo:

    allow-transfer{192.168.200.100;192.168.210.100;10.10.42.41;10.10.42.42;};

    En este listado deberán estar incluidos todos los servidores slave que tengan configurado a éste como servidor master, y adicionalmente alguna IP que debiera tenerlo permitido por alguna razón.

    b. Mediante la directiva notify-yes se consigue enviar automáticamente una notificación de cambio de zona del maestro, cuando ésta se produce, a los servidores DNS especificados en la zona mediante el registro de recurso NS. Adicionalmente, se puede enviar una notificación de cambio de zona a servidores esclavos que no aparecen en la misma, mediante la directiva also-notify:

    also-notify {192.168.200.100;10.10.42.41;};

    Por ejemplo, una zona tipo master con las directivas anteriores podría ser la siguiente:

```
//zonas creadas tipo master
   zone "ejemplo.com" {
      type master;
      file "/var/lib/bind/master/db.ejemplo.com.hosts";
      allow-transfer{192.168.200.100;192.168.210.100;10.10.42.41;10.10.42.42;};
      notify-yes;
      also-notify {192.168.200.100;10.10.42.41;};
   };
```

6. FTP

6.1. ¿Qué es FTP?

Es el *Protocolo de Transferencia de Archivos,* es un protocolo que permite transferir archivos directamente de un dispositivo a otro. Actualmente, es un protocolo que poco a poco se va abandonando, pero ha estado vigente más de 50 años.

Este protocolo funciona entre ordenadores que estén conectados a Internet, y los archivos se comparten de forma directa y sin ningún intermediario.

6.2. Usuarios y acceso anónimos

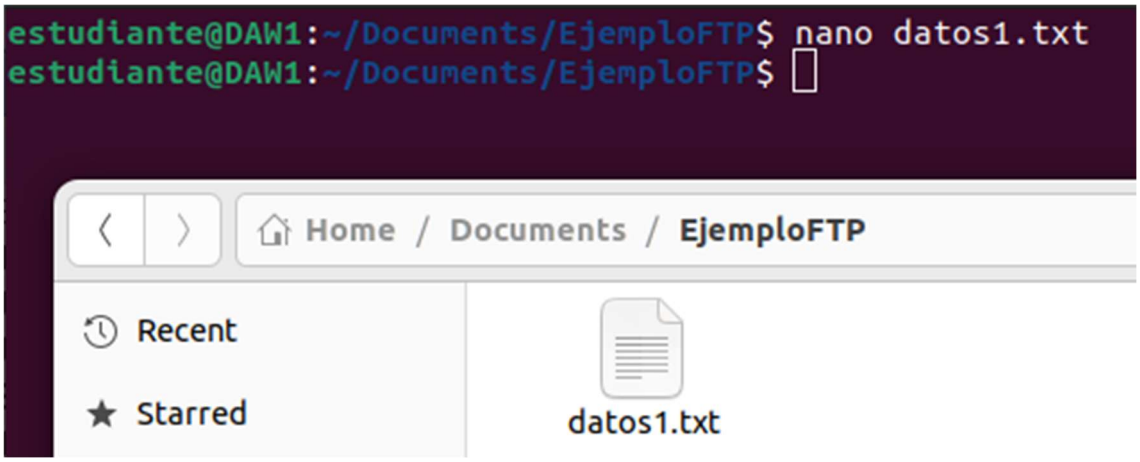
Existen 3 tipos de usuarios:

- **Los usuarios reales:** tienen cuentas que les permiten establecer sesiones en el servidor FTP. Estos usuarios tendrán los permisos establecidos de acceso a archivos y directorios, pero toda la estructura del disco está visible para los usuarios reales.
- **Los usuarios invitados:** también necesitan cuentas para iniciar sesión en el servidor FTP. Cada cuenta de invitado está configurada con un nombre de usuario y una contraseña. Estas cuentas no pueden hacer sesiones de terminales. En el inicio de sesión, el servidor FTP se encarga de restringir la vista de un invitado de la estructura del disco del servidor.
- **Los usuarios anónimos:** inician sesión en el servidor FTP usando ftp o *anonymous* como nombre de usuario. Por convención, los usuarios anónimos indican una dirección de correo electrónico cuando se les solicita una contraseña.
  El FTP se encarga de restringir la vista del usuario anónimo de la estructura del disco del servidor. Una única área del archivo es compartida por todos los usuarios anónimos, a diferencia de las áreas separadas que se pueden crear para cada usuario invitado.

6.3. Clientes gráficos y de línea de comando

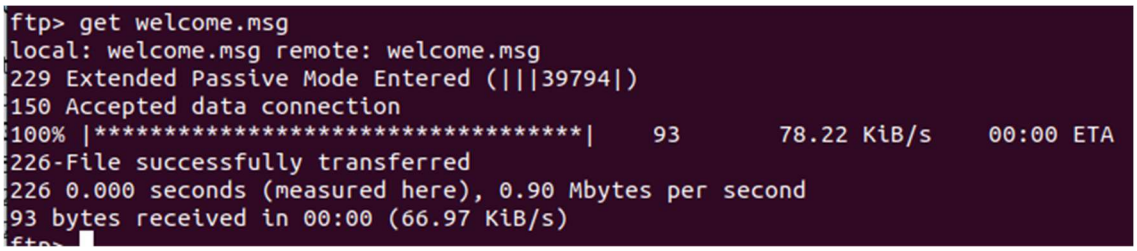
6.3.1. Línea de comando

Creamos un repositorio llamado EjemploFTP y dentro de él creamos un fichero txt con el contenido que queramos.



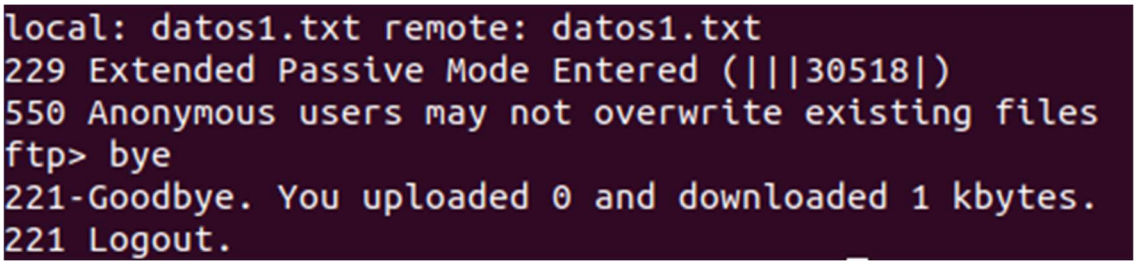
A continuación, establecemos conexión con el servidor ftp de la red iris con los siguientes comandos y nos identificamos con el usuario *anonymous* y sin contraseña.

A continuación, vamos a descargarnos el fichero welcome.msg del servidor



Nos creamos el directorio imágenes

Subimos el archivo datos1.txt con put (como somos *anonymous* no nos deja subirlo al servidor) y establecemos conexión con el comando bye.

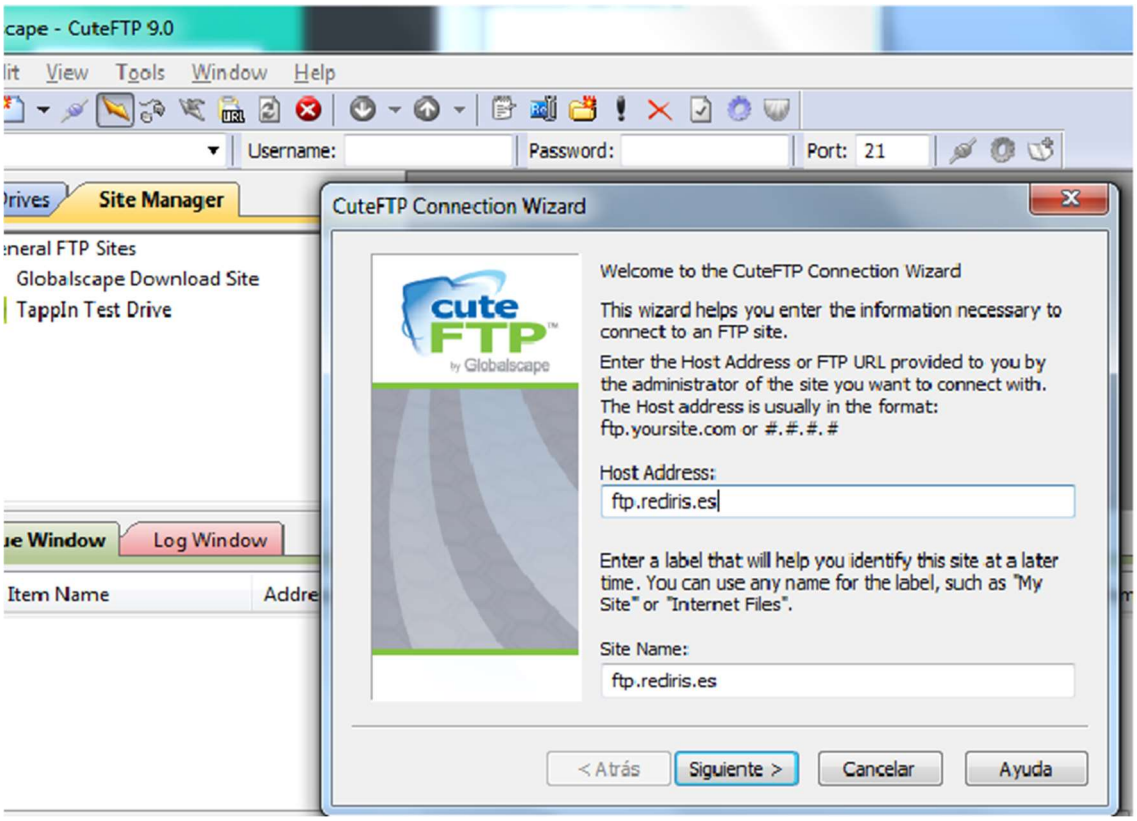


6.3.2. Entorno gráfico

Nos descargamos el cliente cuteFTP y lo instalamos en el equipo.

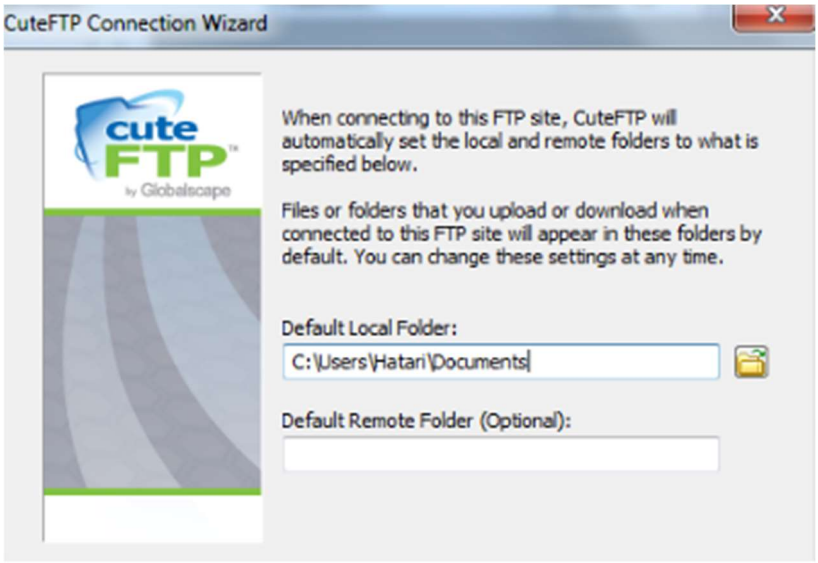
A continuación, nos creamos una carpeta en nuestro directorio personal del equipo llamada EjemploFTP, cuyo contenido será un fichero llamado datos1.txt.

Tendremos que poner la dirección del servidor ftp de la red iris:

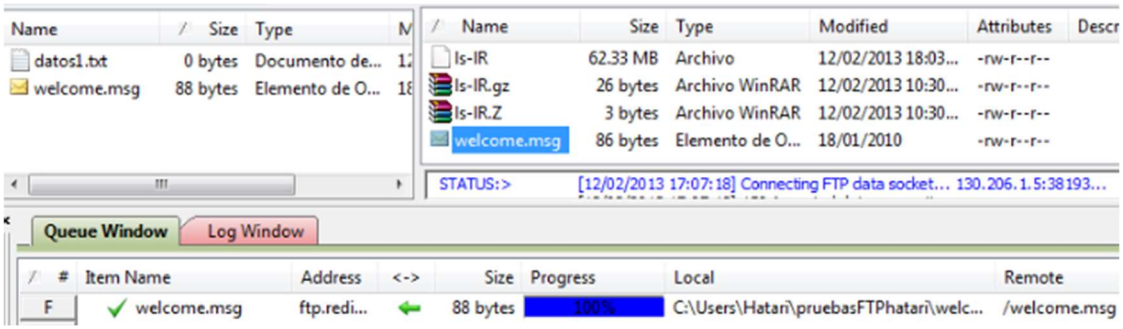


Realizamos la conexión con el usuario anónimo sin contraseña:

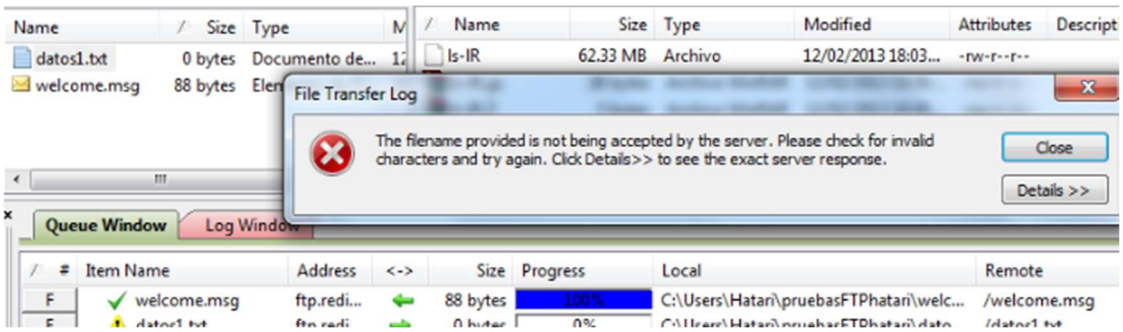
Introducimos nuestro repositorio



Seleccionamos el fichero welcome.msg y nos lo descargamos del servidor:



Como somos anónimo al intentar subir los datos1.txt nos da error:



6.3. FTP Seguro

Con FTPS, una conexión es autenticada utilizando el ID de usuario, la contraseña y los certificados. Los usuarios y las contraseñas para las conexiones serán encriptadas. Cuando se conecte al servidor FTPS de un socio comercial, el cliente primero comprobará que el certificado del servidor sea confiable. Será de confianza cuando el certificado fue firmado por una autoridad certificada.

BIBLIOGRAFÍA

Blancarte, O. (2020). *Introducción a la arquitectura de software*. Disponible en: https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/monolitico. Consultado: 17/11/2022

Cheatography (n.d). *HTTP Status Codes Cheat Sheet*. Disponible en: https://cheatography.com/kstep/cheat-sheets/http-status-codes/. Consultado: 15/11/2022

García Escobedo, J. (2012). *Implantación de arquitecturas web: Modelos de arquitecturas*. Disponible en: https://javiergarciaescobedo.es/despliegue-de-aplicaciones-web/76-arquitecturas-web. Consultado: 15/11/2022

Geeky Theory (2014). *Cómo funciona el DNS*. Disponible en: https://geekytheory.com/como-funciona-el-dns/. Consultado: 18/11/2022

Edytapukocz (s.f.). *Partes de una URL*: *ejemplos básicos*. Disponible en: https://edytapukocz.com/url-partes-ejemplos-facil/. Consultado: 18/11/2022

Goanywhere (s.f.). *SFTP vs. FTPS*. Disponible en: https://www.goanywhere.com/es/blog/sftp-vs-ftps-cual-es-el-mejor-protocolo-para-asegurar-ftp. Consultado: 15/11/2022

Instituto Tecnológico de Matehuala (s.f.). *Programación Web: Arquitectura de aplicaciones web*. Disponible en: https://programacionwebisc.wordpress.com/2-1-arquitectura-de-las-aplicaciones-web/. Consultado: 15/11/2022

Kinsta (2022). *Una guía completa y una lista de códigos de estado HTTP.* Disponible en: https://kinsta.com/es/blog/codigos-de-estado-de-http/. Consultado: 15/11/2022

KIO Networks (s.f.). *Protocolos de comunicación de redes*. Disponible en: https://www.kionetworks.com/blog/data-center/protocolos-de-comunicaci%C3%B3n-de-redes. Consultado: 15/11/2022

Luján Mora, S. (2002). *Programación de Aplicaciones Web: Historia, principios básicos y clientes web*. Disponible en: https://rua.ua.es/dspace/bitstream/10045/16995/1/sergio_lujan-programacion_de_aplicaciones_web.pdf. Consultado: 15/11/2022

Mozilla Corporation (2022). *Métodos de petición HTTP*. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Methods. Consultado: 15/11/2022

Mozilla Corporation (2022). *Tipos MIME*. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/MIME_types. Consultado: 18/11/2022

Mozilla Corporation (2022). *Generalidades del protocolo HTTP*. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Overview. Consultado: 18/11/2022

Oracle (s.f.). *Guía de administración del sistema-servicios de red*. Disponible en: https://docs.oracle.com/cd/E24842_01/html/E22524/wuftp-10.html. Consultado: 15/11/2022

Programación Fácil Blog (2021). *¿Qué es y cómo funciona DNS? Introducción al hacking con Kali Linux – Capítulo 14*. Disponible en: https://programacionfacil.org/blog/fundamentos-de-dns/. Consultado: 15/11/2022

Significados (s.f.). *Qué es el Internet*. Disponible en: https://www.significados.com/internet/. Consultado: 15/11/2022

SitioLibre (s.f.). Tema 5: *Servicios de red implicados en el despliegue de una aplicación web*. Disponible en: https://www.sitiolibre.com/curso/pdf/DAW05.pdf [pdf descargable] Consultado: 15/11/2022

Softwarelab (s.f.). *El formato del url*. Disponible en: https://softwarelab.org/es/url/. Consultado: 18/11/2022

What-When-How (s.f.). *TCP/IP and Network Layers*. Disponible en: https://what-when-how.com/data-communications-and-networking/tcpip-example-data-communications-and-networking/. Consultado: 15/11/2022

Wordpress (s.f.). *Utilización cliente FTP (mediante línea de comandos, entornos gráficos y navegadores/exploradores)*. Disponible en: https://asirhata2.files.wordpress.com/2013/05/utilizacic3b3n-cliente-ftp-mediante-lc3adnea-de-comandos-entornos-grc3a1ficos-y-navegadoresexploradores.pdf. Consultado: 18/11/2022