EL PROTOCOLO HTTP

Charly Cimino

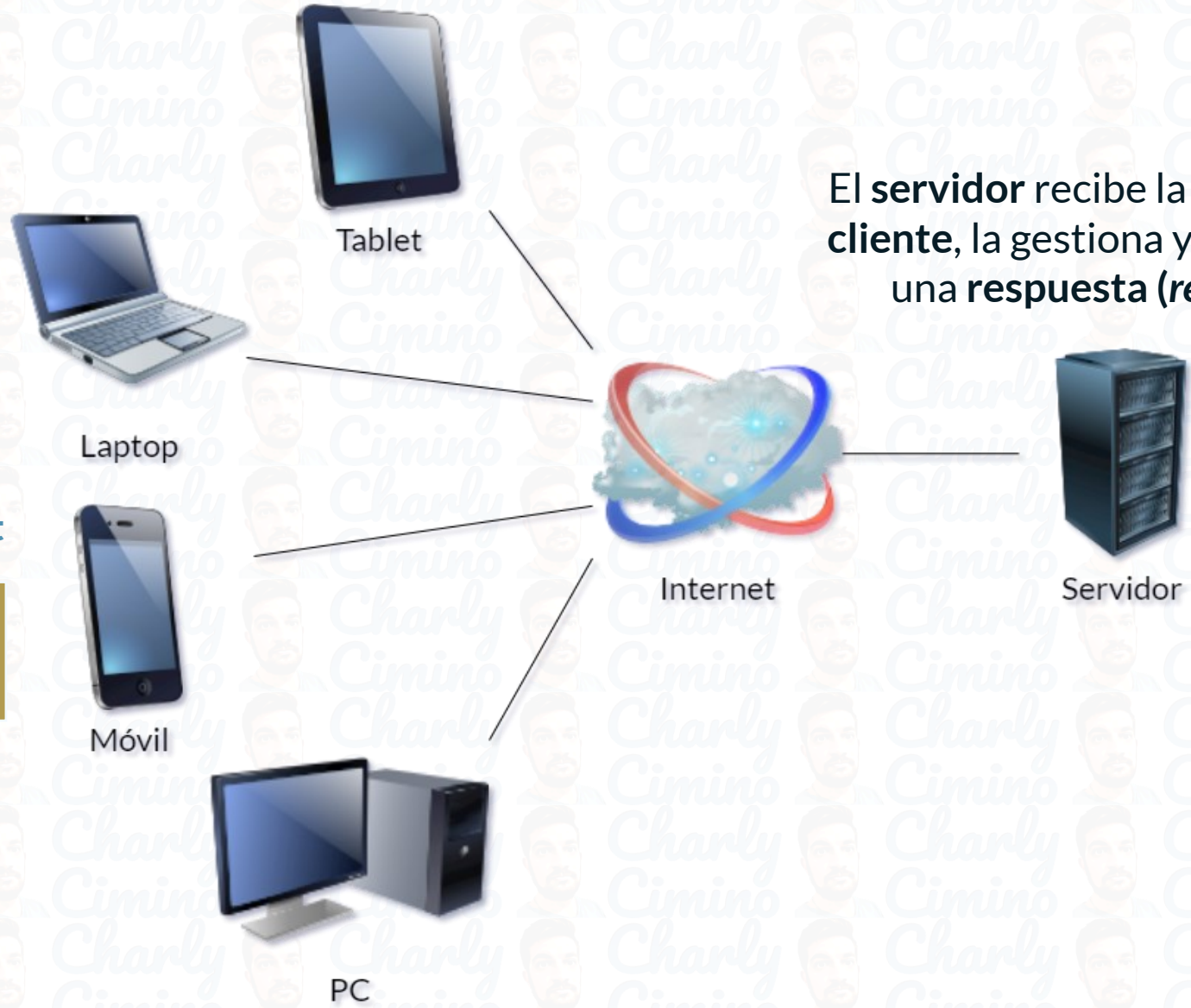
El protocolo HTTP

Charly Cimino

Versión 2023.09.01

Arquitectura Cliente-Servidor

A través de diferentes dispositivos (*tablets*, móviles, *Smart* TVs, *laptops*, computadoras de escritorio, etc) llamados **clientes**, establecemos **peticiones (*requests*)** a un **servidor**.



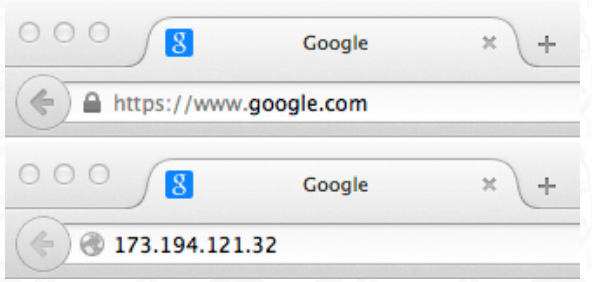
El **servidor** recibe la petición del **cliente**, la gestiona y le devuelve una **respuesta (*response*)**.

Algunos servicios que operan sobre Internet

- *World Wide Web* (WWW)
- Correo Electrónico
- VoIP (Voz sobre IP)
- RSS
- Transferencia de archivos vía FTP

¿Cómo funciona la web?

Se escribe una dirección web (como http://www.google.com) en un navegador o se hace click en un enlace.



En lugar de recordar una dirección IP (por ejemplo. **142.251.35.174**), es más fácil recordar un nombre de dominio (por ejemplo, **google.com**).
Ese es el trabajo que hacen los servidores **DNS** (**Sistema de Nombres de Dominio**): reemplazar el nombre de dominio por la dirección IP donde se aloja el servidor web.

El navegador va al servidor DNS y encuentra la dirección real del servidor donde el sitio web reside.

El navegador envía una petición **HTTP** al servidor web que pide una copia de la página web que se desea visitar.

Esta solicitud y todos los datos enviados entre el cliente y el servidor, se realizan a través de una conexión a Internet usando el protocolo **TCP/IP (Protocolo de control de transmisión/Protocolo de Internet)**.

El servidor evalúa la petición y, si la aprueba, envía una respuesta exitosa en forma de paquetes de datos que conforman la página web.

Los datos se envían a través de la web como miles de trozos pequeños, permitiendo que muchos usuarios pueden descargar la misma página web al mismo tiempo. Si los sitios web fueran enviados como grandes trozos, sólo un usuario podría descargarlos a la vez, lo que volvería a la web muy ineficiente.

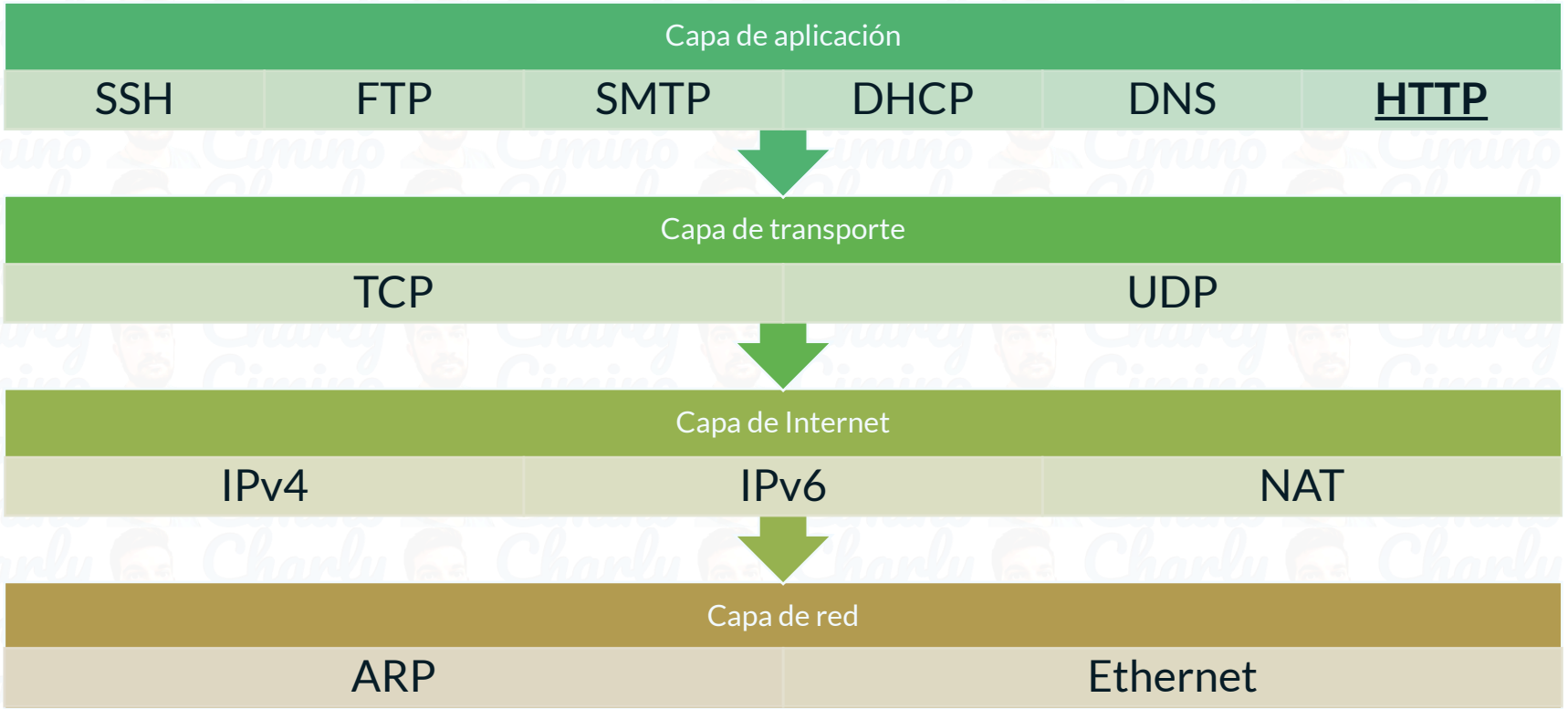
El navegador web recibe esos paquetes y los reagrupa, para formar la página web.

Luego, la muestra.

https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/How_the_Web_works

¿Qué es un protocolo?

En informática, se define como un conjunto de normas que permite la comunicación entre computadoras, estableciendo sus formas de identificación en la red, la manera de transmitir los datos y cómo la información debe procesarse.

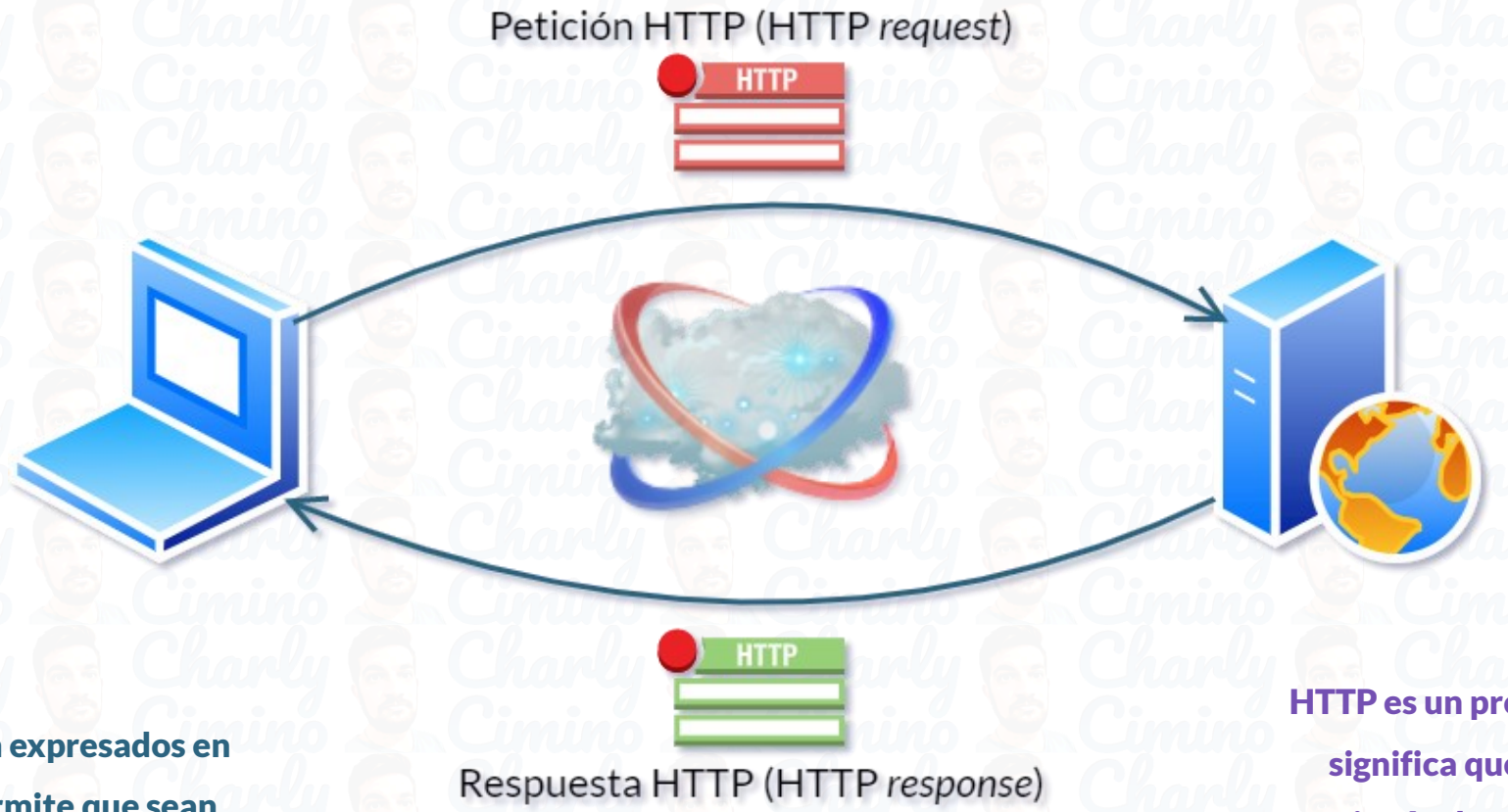


Algunos de los protocolos existentes para el modelo de red de 4 capas TCP/IP

https://es.wikipedia.org/wiki/Protocolo_de_comunicaciones

¿Qué es HTTP?

***HyperText Transfer Protocol* (HTTP)**, es un protocolo que permite la comunicación entre clientes y servidores, utilizando mensajes individuales para la transferencia de recursos web.



Los mensajes que envía el cliente, normalmente un navegador web (*browser*), se llaman peticiones (*requests*).

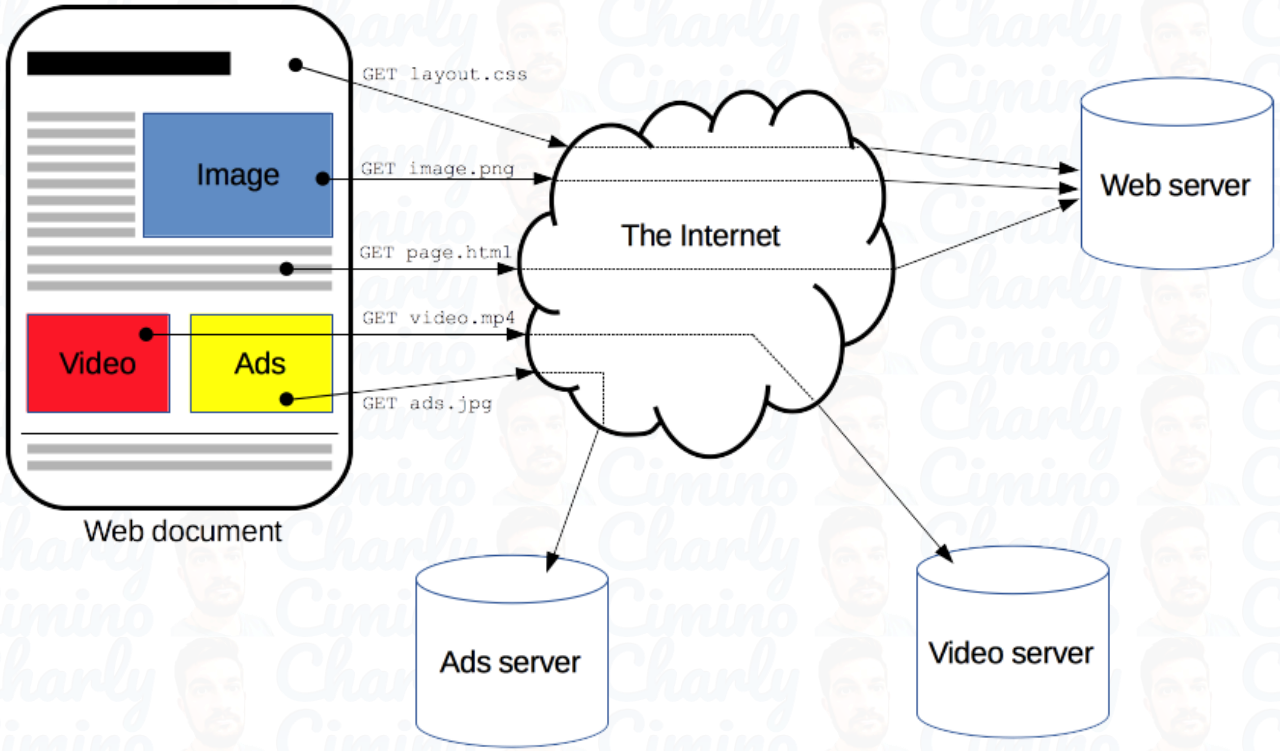
Los mensajes que envía el servidor web (*web server*) se llaman respuestas (*responses*).

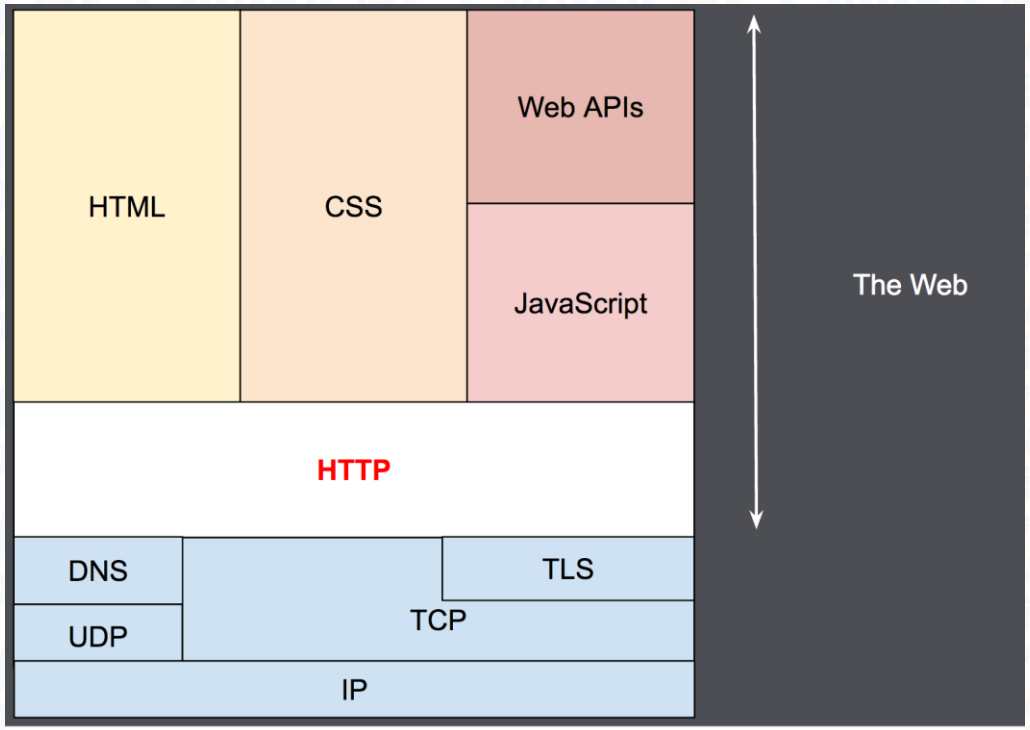
Los mensajes HTTP son expresados en texto plano, lo que permite que sean más legibles y fáciles de depurar.

HTTP es un protocolo sin estado, lo que significa que el servidor no guarda ningún dato entre dos solicitudes.
(Para ello se usan *cookies*)

https://developer.mozilla.org/es/docs/Web/HTTP

La dinámica de HTTP





Una página web completa resulta de la unión de distintos subdocumentos recibidos, como por ejemplo: los estilos de la página web (CSS), las imágenes, vídeos, archivos, *scripts* (JS), etc...

Al hablar del diseño y la programación web, nos concentramos en **cómo procesar** los datos.
Nos abstraemos de **cómo se transportan** entre los dispositivos a través de la red.

https://developer.mozilla.org/es/docs/Web/HTTP/Overview

Una típica sesión de HTTP

Un código de estado exitoso podría ser el `200 - OK`.
Un código de estado erróneo podría ser el `404 - Not Found`.

Más detalle sobre códigos de estado HTTP

En TCP, el puerto por defecto para un servidor HTTP en una computadora es el **80**.
Se pueden usar otros puertos como el **8000** o el **8080**.

A partir del protocolo HTTP/1.1, la conexión no se cierra al finalizar la tercera fase. El cliente puede continuar realizando peticiones, o sea, que la segunda y tercera fase pueden repetirse cualquier número de veces.

https://developer.mozilla.org/es/docs/Web/HTTP/Session

Recursos y URL

El objetivo de una solicitud HTTP se denomina **recurso**: puede ser un documento, una foto, un binario, etc.

Cada recurso es identificado por HTTP mediante un **Identificador Uniforme de Recursos** (**URI**).

La forma más común de **URI** es la **URL** (**Localizador Uniforme de Recursos**), que se conoce como la **dirección web**.

```
http://www.ejemplo.com
http://www.ejemplo.com:80
http://192.168.0.10:80
http://www.ejemplo.com/contacto.html
http://www.ejemplo.com/es/about
http://www.ejemplo.com/es/buscar?q=programar
http://www.ejemplo.com/es/tutoriales/html#etiquetas
http://www.ejemplo.com/es/empleados?nom=Juan%20Carlos&ape=Pérez
```

Ejemplos de URLs

https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web#urls_and_urns

Partes de una URL

```
http://www.ejemplo.com:80/ruta/hacia/archivo.html?clave1=valor1&clave2=valor2#AlgunMarcador
```

Una URL esta compuesta de diferentes partes, algunas obligatorias y otras opcionales.

El protocolo a utilizar por el *browser*. Por lo general es `http://` o `https://` (versión segura).

Otros (no web) podrían ser `mailto:` (para abrir un cliente de mail), `ftp:` (para transferir archivos) o `file:///` (para abrir archivos localmente).

Nombre de dominio o dirección IP (menos frecuente) a la cual se desea acceder.

Puerto utilizado por el servidor para permitir el acceso a los recursos.

Si utiliza los estándares (80 para HTTP y 443 para HTTPS) se puede omitir, de lo contrario, es obligatorio.

```
http://ejemplo.com:80
```

- Si no se especifica el protocolo al colocar el link en la barra de direcciones del *browser*, asumirá que se trata de `http://`
- Si no se especifica el protocolo al colocar el link en el atributo **`href`** de una etiqueta `<a></a>` en HTML, se asumirá que se trata del mismo protocolo que la página actual.
- Durante mucho tiempo los dominios requerían empezar con `www.` Ya no, pero aún continúa habiendo dominios con `www.` por herencia.
- Generalmente, las empresas compran ambos dominios (`empresa.com` y `www.empresa.com`), haciendo que un *link* redirija automáticamente al otro.

Probá los siguientes links en tu *browser*:

```
w3schools.com
www.w3schools.com
https://www.w3schools.com
https://www.w3schools.com:443
```

https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web#sintaxis_de_identificador_uniforme_de_recursos_(uris)

Partes de una URL

Ruta de acceso

```
http://ejemplo.com/carpeta1/carpeta2/pagina.html
```

Ruta de acceso al recurso en el servidor web.

En los primeros días de la Web, una ruta como esta presentaba la ubicación física (real) del recurso en el servidor. Actualmente, es una mera abstracción manejada por los servidores web que casi nunca revelan la ubicación física.

Probá los siguientes links en tu *browser*:

```
https://w3schools.com/
https://w3schools.com/html/
https://w3schools.com/html/html_basic.asp
https://w3schools.com/html/pic_trulli.jpg
```

https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web#ruta_de_acceso

Partes de una URL

Query String

```
http://ejemplo.com/buscar?nom=Juan&ape=Pérez
```

Parámetros opcionales (*query strings*) proporcionados al servidor web.

Comienza con un `?` seguido por una lista de pares `clave=valor` separados por el símbolo `&`.

El servidor web puede llegar a requerir estos parámetros para procesar la petición que debe retornar al usuario.

Probá los siguientes links en tu *browser*:

```
https://developer.mozilla.org/es/search
https://developer.mozilla.org/es/search?q=http
https://developer.mozilla.org/es/search?q=Qué+es+URL
```

https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web#consulta

Partes de una URL

Fragmento

```
http://ejemplo.com/articulo1.html#Conclusiones
```

Es una referencia a otra parte del mismo recurso.

Representa una especie de "marcador" que indica al *browser* que muestre el contenido en esa referencia señalada.

Luego del `#`, se coloca el identificador del fragmento (suele ser el valor del atributo `id` de un elemento HTML).

El identificador del fragmento nunca se envía al servidor.

Probá los siguientes links en tu *browser*:

```
https://es.wikipedia.org/wiki/HTML
https://es.wikipedia.org/wiki/HTML#Atributos
https://es.wikipedia.org/wiki/HTML#Historia_del_estándar
```

https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web#fragmento

Codificación de las URL

Las direcciones URL solo se pueden enviar a través de Internet utilizando la codificación de caracteres ASCII

Dado que las URL contienen a menudo caracteres fuera del conjunto ASCII y otros dentro de éste pero que son reservados (por ejemplo, los espacios, la **/** o el **&**), deben codificarse:

```
http://www.ejemplo.com/artistas favoritos?nom=Carlos Santana&álbum=Zebop!
```

application/x-www-form-urlencoded

```
http://www.ejemplo.com/artistas%20favoritos?nom=Carlos+Santana&%C3%A1lbum=Zebop%21
```

Los caracteres codificados de reemplazo se muestran subrayados para destacarlos.

El código por ciento reemplaza cada caracter reservado por un **%** seguido del número (en hexadecimal) del caracter en la tabla ASCII.

Los espacios se reemplazan por un **+** (en las *query strings*) o con **%20** (en el resto de la URL).

https://www.w3schools.com/tags/ref_urlencode.asp

El dominio 'localhost'

`localhost` es un nombre reservado, que representa la conexión hacia nosotros mismos (*loopback*), a través de la IP reservada para tal fin (**`127.0.0.1`**).

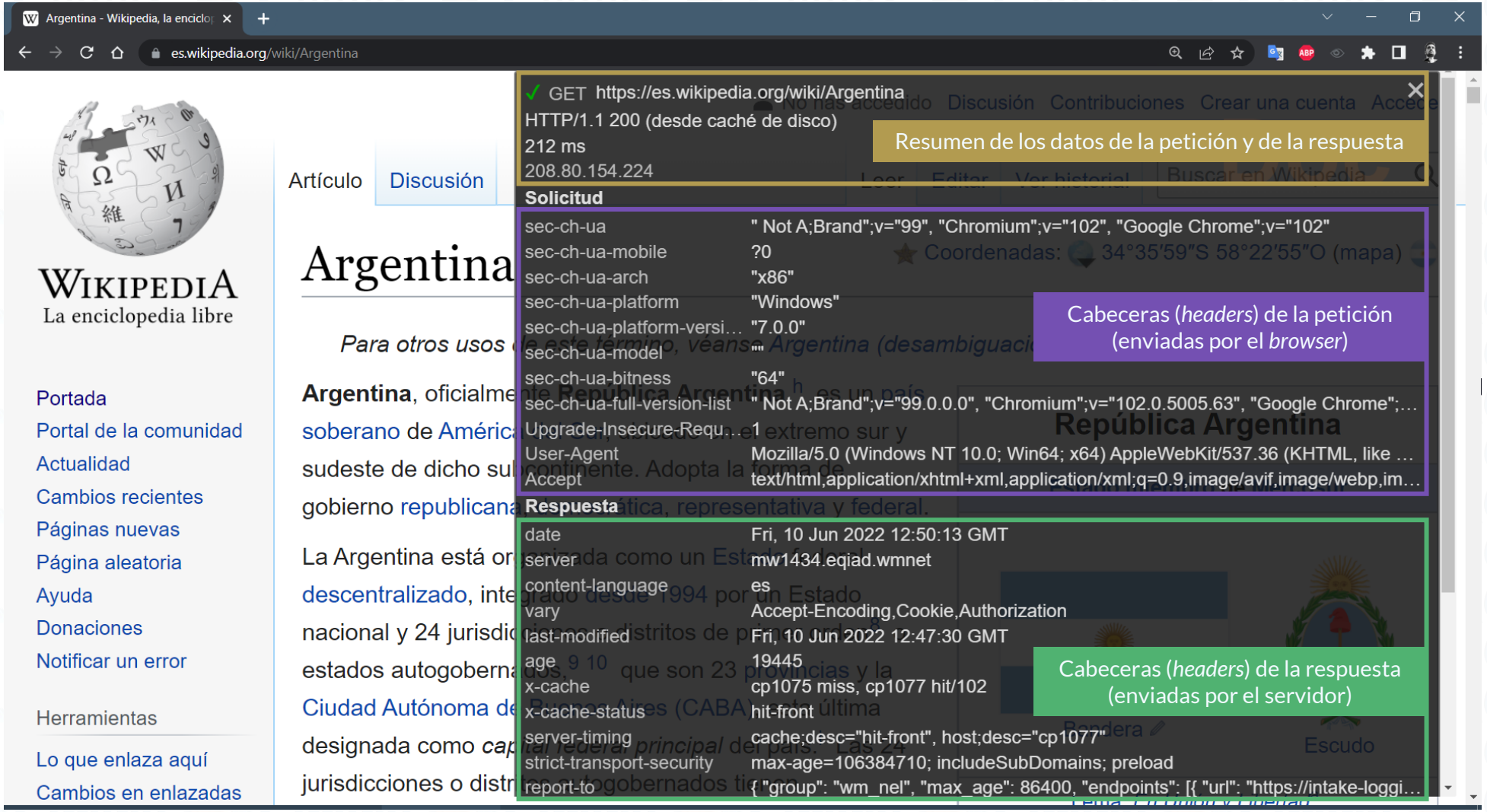
¿Para qué se usa?

- Verificación de software TCP/IP
- Entornos de pruebas offline
- Restricciones de acceso
- Fines educativos

Nuestra computadora hace de cliente y servidor a la vez.

Espiando al *browser*

Con la extensión HTTP Header Spy de Google Chrome se pueden observar los datos (*headers*, *cookies*, códigos, etc.) de las peticiones y respuestas entre el *browser* y un servidor web .



Anatomía de una petición HTTP

La petición HTTP de un cliente consiste simplemente en líneas de texto, separadas mediante saltos de línea, que la dividen en tres partes.

Ejemplo de petición GET

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: es
```

Ejemplo recortado de una petición al servidor de **developer.mozilla.org** para obtener la página principal, preferentemente en español.

La primera parte consiste en un renglón con el nombre del método HTTP (GET, POST, PUT, entre otros...) seguido de la URL completa (sin protocolo ni nombre de dominio) y la versión del protocolo HTTP.

La segunda parte consiste en renglones consecutivos que representan las cabeceras *(headers)* de la petición HTTP, las cuales le brindan información extra al servidor. Este bloque acaba con una línea en blanco.

La última parte es un bloque opcional, que puede contener datos adicionales *(body)*, por ejemplo, para peticiones con el método POST.

Ejemplo de petición POST

```
POST /form_contacto.php HTTP/1.1
Host: misitio.com
Content-Length: 64
Content-Type: application/x-www-form-urlencoded

name=Juan%20Pérez&mensaje=Hola
```

Una petición al servidor de **misitio.com** para enviar los datos de un formulario a un script del lado del servidor, ubicado en **/form_contacto.php**

https://developer.mozilla.org/es/docs/Web/HTTP/Session#mandando_una_petici%C3%B3n

Métodos para peticiones HTTP

HTTP define un conjunto de métodos para peticiones en los que se indica, de forma semántica, el tipo de acción que se desea llevar a cabo en el servidor web.

Son denominados a veces como 'verbos' de HTTP, siendo los más comunes **GET** y **POST**.

GET	HEAD	POST	PUT	DELETE
• El más común. Solicita un recurso específico. Sólo debe recuperar datos. Si bien también puede enviar datos al servidor, presenta limitaciones (vistas en la siguiente diapositiva).	• Idéntico a **GET**, pero el servidor no devuelve el contenido la respuesta, solo el código y las cabeceras.	• El más usado para enviar datos desde un formulario. Envía información a un recurso específico, causando un cambio de estado o efectos secundarios en el servidor.	• Utilizado normalmente para actualizar contenidos en el servidor, aunque también puede crearlos.	• Borra un recurso específico del servidor.

Otros métodos no tan frecuentemente utilizados: **CONNECT**, **OPTIONS**, **TRACE**, **PATCH**.

https://developer.mozilla.org/es/docs/Web/HTTP/Methods

Comparativa entre los dos métodos más comunes

	GET	POST
Uso	Para solicitar un recurso al servidor web	Para enviar datos a un servidor (crear/actualizar un recurso)
Dónde viajan los datos	En la URL de la petición	En el cuerpo (*body*) de la solicitud HTTP
Seguridad	**GET** es menos seguro en comparación con **POST** porque los datos enviados son parte de la URL. **¡Nunca usarlo para enviar contraseñas, datos bancarios u otra información sensible!**	**POST** es más seguro que **GET** porque los parámetros no se almacenan en el historial del *browser* ni en los registros del servidor web.
Restricciones de longitud	Sí, al enviar datos, el método **GET** los agrega como parte de la URL, cuya longitud es de 2048 caracteres como máximo.	No
Restricciones en el tipo de datos enviados	Solo se permiten caracteres ASCII	Sin restricciones. También se permiten datos binarios.
Codificación	**application/x-www-form-urlencoded**	**application/x-www-form-urlencoded** Para datos binarios: **multipart/form-data**
Botón "Atrás" y "Actualizar" del *browser*	Sin efectos	Los datos se volverán a enviar (el *browser* suele alertar al usuario)
Pueden guardarse en favoritos	Sí	No
Se pueden almacenar en caché	Sí	No
Permanecen en el historial del *browser*	Sí	No

https://www.w3schools.com/tags/ref_httpmethods.asp

¿Qué es un *header*?

Las **cabeceras HTTP** (*HTTP headers*) permiten al cliente y al servidor enviar información adicional junto a una petición o respuesta. El formato es `nombre: valor`

https://developer.mozilla.org/es/docs/Web/HTTP/Headers

Headers más comunes en las peticiones HTTP

Nombre	Uso	Valores típicos / Ejemplos
`Host`	Especifica el nombre de dominio del servidor. Debe enviarse obligatoriamente en todas las solicitudes HTTP/1.1, de lo contrario, se obtendrá un error **`400-Bad Request`**. (Ver más detalle)	`Host: developer.mozilla.org` `Host: w3schools.com`
`User-Agent`	Especifica diversa información sobre el cliente, como por ejemplo, la versión y el nombre del *browser*, el idioma, el sistema operativo, etc. (Ver más detalle)	`User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0` `User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows Phone OS 7.5; Trident/5.0; IEMobile/9.0)`

`Accept-Language`	Especifica el idioma predeterminado del cliente, para poder obtener, si es posible, una versión de la web que corresponda a tal idioma. (Ver más detalle)	`Accept-Language: es`
		`Accept-Language: de-CH`
		`Accept-Language: en-us, en;q=0.5`
		`Accept-Language: fr-CH, fr;q=0.9, en;q=0.8, *;q=0.5`

`Accept-Encoding`	Especifica los nombres de uno o más algoritmos de compresión para aplicar sobre la respuesta, de manera tal de ahorrar ancho de banda y tiempo. La mayoría de los *browsers* ya soportan el método **gzip**, uno de los más usados. (Ver más detalle)	`Accept-Encoding: gzip`
		`Accept-Encoding: gzip, deflate`
		`Accept-Encoding: gzip, deflate, compress`

`If-Modified-Since`	Especifica una fecha y hora para que el servidor revise si el recurso ha cambiado desde ese momento. Si ha cambiado, el servidor retorna una respuesta con los nuevos datos, de lo contrario, envía una respuesta de redirección **`304-Not Modified`**, lo que indicará al *browser* que cargue el contenido desde la caché, permitiendo ahorrar tiempo y ancho de banda. (Ver más detalle)	`If-Modified-Since: Sat, 28 Nov 2009 06:38:19 GMT`
`Cookie`	Especifica las *cookies* HTTP almacenadas en el cliente, las cuales han sido enviadas previamente por el servidor en el *header* **`Set-Cookie`**. Se profundizará en el tema *Cookies* más adelante. (Ver más detalle)	`Cookie: PHPSESSID=298zf09hf012fh2; csrftoken=u32t4o3tb3gg43; _gat=1;`
`Referer`	Especifica la URL de referencia, es decir, desde donde se hizo la petición. Permite a los servidores identificar desde dónde los visitan las personas, pudiendo usar estos datos para elaborar análisis, registros, limitaciones, etc. (Ver más detalle)	`Referer: https://developer.mozilla.org/es/docs/Web` `Referer: http://w3schools.com`
`Authorization`	Especifica las credenciales para autenticar a un usuario en un servidor, usualmente luego de que el servidor haya respondido con un estado **`401-Unauthorized`** y el *header* **`WWW-Authenticate`**. Los datos se envían codificados en Base64, el cual es perfectamente reversible (no es seguro). (Ver más detalle)	`Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l`

https://code.tutsplus.com/es/tutorials/http-headers-for-dummies--net-8039

Anatomía de una respuesta HTTP

La respuesta HTTP de un servidor web, consiste, al igual que antes, simplemente en líneas de texto, separadas mediante saltos de línea, que la dividen en tres partes.

Ejemplo de respuesta exitosa

```
HTTP/1.1 200 OK
Date: Wed, 08 Jun 2022 18:50:29 GMT
Server: Apache
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html><html><head><title>Hola</title></head><body><h1>Hola mundo</h1></body></html>
```

La primera parte consiste en un renglón con la versión del protocolo HTTP seguido por el estado de la petición, que consta de un código y una descripción legible.

La segunda parte consiste en renglones consecutivos que representan las cabeceras *(headers)* de la respuesta HTTP, las cuales le brindan información extra al cliente. Este bloque acaba con una línea en blanco.

La última parte representa los datos recibidos por el servidor web.

Ejemplo de respuesta errónea

```
HTTP/1.1 404 Not Found
Date: Wed, 08 Jun 2022 21:58:15 GMT
Server: cloudflare
Content-Type: application/json

{status: "error",message: "No route found for "GET /api/breeds/image/fake" with code: 0",code: 404}
```

https://developer.mozilla.org/es/docs/Web/HTTP/Session#mandando_una_petici%C3%B3n

Headers más comunes en las respuestas HTTP

Nombre	Uso	Valores típicos / Ejemplos
`Cache-Control`	Especifica cómo manejar el almacenamiento en caché en las peticiones y respuestas. Permite reducir el consumo de memoria en el servidor y los tiempos de carga. (Ver más detalle)	`Cache-Control: max-age=3600, public` `Cache-Control: no-cache`

`Content-Disposition`	Especifica si el contenido debe mostrarse en el *browser* o descargarse como un archivo adjunto. (Ver más detalle)	`Content-Disposition: inline`
		`Content-Disposition: attachment`
		`Content-Disposition: attachment; filename="arch.ext"`

`Content-Length`	Especifica el tamaño (en bytes) del contenido que va a ser transmitido al *browser*. Útil para descargas de archivos, para que el *browser* puede determinar el progreso de la descarga. (Ver más detalle)	`Content-Length: 12345`

`Content-Type`	Especifica el tipo MIME del recurso recibido, de manera tal que el *browser* decida cómo interpretarlo. (Ver más detalle)	`Content-Type: text/html; charset=utf-8`
		`Content-Type: image/gif`
		`Content-Type: application/pdf`

`Content-Encoding`	Especifica el algoritmo de compresión del contenido, si éste es comprimido. (Ver más detalle)	`Content-Encoding: gzip` `Content-Encoding: deflate`
`Last-Modified`	Especifica la ultima fecha de modificación del documento. (Ver más detalle)	`Last-Modified: Wed, 21 Oct 2015 07:28:00 GMT`

`Location`	Especifica la URL hacia la cual redirigirse. Obligatorio cuando la respuesta tiene código **3xx o 201**. (Ver más detalle)	`Location: misitio.com`
		`Location: /otroLado`
		`Location: /`

`Set-Cookie`	Especifica una *cookie* HTTP que el servidor pretende crear/actualizar en el cliente. almacenadas en el cliente. Se profundizará en el tema *Cookies* más adelante. (Ver más detalle)	`Set-Cookie: <nombre>=<valor>` `Set-Cookie: <nombre>=<valor>; Expires=<fecha>`
`WWW-Authenticate`	Especifica la forma de autenticación a ser utilizado para acceder a cierto recurso. Suele ser enviado en una respuesta **`401-Unauthorized`**. (Ver más detalle)	`WWW-Authenticate: Basic` `WWW-Authenticate: OAuth realm="Zona premium"`

https://code.tutsplus.com/es/tutorials/http-headers-for-dummies--net-8039

Códigos de estado de respuesta HTTP

Indican si se ha completado satisfactoriamente una solicitud HTTP.
Son números enteros de tres dígitos agrupados en cinco categorías:

1xx
Respuestas informativas

2xx
Respuestas exitosas

3xx
Redirecciones

4xx
Errores del cliente

5xx
Errores del servidor

https://developer.mozilla.org/es/docs/Web/HTTP/Status

Códigos de estado de respuesta HTTP (1xx-2xx-3xx)

Nro.	Detalle	Significado
100	Continue	Todo hasta ahora está bien. El cliente debe continuar con la petición o ignorarla si ya está terminada.
101	Switching Protocols	El servidor acepta el cambio de protocolo propuesto por el *browser* (por ejemplo un cambio de HTTP 1.0 a HTTP 1.1).
102	Processing	El servidor ha recibido la petición y aún se encuentra procesándola, por lo que el *browser* debe esperar.
103	Checkpoint	Se va a reanudar una petición POST o PUT que fue abortada previamente.
200	OK	Petición exitosa
201	Created	Petición exitosa. Se ha creado un nuevo recurso como resultado de ello.
202	Accepted	La petición ha sido recibida pero aún no se ha procesado (eventualmente puede no ser satisfactoria).
203	Non-Authoritative Information	La petición se ha completado con éxito, pero su contenido no se ha obtenido de la fuente originalmente solicitada, sino que se recoge de una copia local o de un tercero.
204	No Content	La petición se ha completado con éxito pero su respuesta no tiene ningún contenido, aunque los encabezados pueden ser útiles.
205	Reset Content	La petición se ha completado con éxito, pero su respuesta no tiene contenidos y además, el *browser* debe recargar la página desde la que se realizó la petición. Es útil, por ejemplo, para páginas con formularios cuyo contenido debe borrarse después de que el usuario lo envíe.
206	Partial Content	La petición servirá parcialmente el contenido solicitado. Esta característica es utilizada por herramientas de descarga (como **wget**) para continuar la transferencia de descargas anteriormente interrumpidas o para dividirlas y procesar sus partes simultáneamente.
207	Multi-Status	Se devuelve un archivo XML que contiene varias respuestas diferentes, en función de las peticiones realizadas.

Nro.	Detalle	Significado
300	Multiple Choice	Esta petición tiene más de una posible respuesta. Se debe elegir una de ellas. Se usa, por ejemplo, para presentar distintas opciones de formato para video, listar archivos con distintas extensiones o desambiguar palabras (como hace Wikipedia).
301	Moved Permanently	La URI del recurso solicitado ha cambiado. Probablemente una nueva URI sea devuelta en la respuesta.
302	Found	El código de redirección más popular, pero los navegadores populares lo implementaron como **303 See Other**. HTTP/1.1 añadió códigos de estado **303** y **307** para eliminar la ambigüedad pero la mayoría de apps web y aún utilizan el código **302** como si fuera el **303**.
303	See Other	El servidor envía esta respuesta para dirigir al cliente a un nuevo recurso solicitado a otra dirección usando una petición **GET**.
304	Not Modified	La URL no ha sido modificada desde que fue requerida por última vez. El cliente suele proveer un *header* como `If-Modified-Since` que indica una fecha y hora con la que el servidor pueda comparar. Este *header* ahorra ancho de banda: el cliente puede continuar usando la misma versión almacenada en su caché.
305	Use Proxy	(Obsoleto) Fue definida en una versión previa de la especificación de HTTP para indicar que una respuesta solicitada debía ser accedida desde un *proxy*. Ha quedado obsoleta por razones de seguridad.
306	Switch Proxy	(Obsoleto) Ya no se utiliza este código pero es reservado para casos futuros.
307	Temporary Redirect	Misma semántica que el **302**, pero con la diferencia de que el cliente debe obtener el recurso solicitado a otra URI con el mismo método que se usó en la petición anterior.
308	Permanent Redirect	Misma semántica que el **301**, pero con la diferencia de que el cliente debe obtener el recurso solicitado a otra URI con el mismo método que se usó en la petición anterior.

https://developer.mozilla.org/es/docs/Web/HTTP/Status

Códigos de estado de respuesta HTTP (4xx)

Nro.	Detalle	Significado
400	Bad Request	El servidor no pudo interpretar la solicitud dada una sintaxis inválida.
401	Unauthorized	Es necesario autenticarse para obtener la respuesta solicitada. Similar a **403**, pero en este caso, la autenticación es posible.
402	Payment Required	Reservado para futuros usos, con el objetivo de ser utilizado en sistemas digitales de pagos.
403	Forbidden	El cliente no posee los permisos necesarios para cierto contenido. Similar a **401**, pero en este caso, autenticarse previamente no va a modificar la respuesta.
404	Not Found	El servidor no pudo encontrar el contenido solicitado. Es uno de los más famosos dada su alta ocurrencia en la web.
405	Method Not Allowed	Método de solicitud no soportado para la URI dada, por ejemplo, cuando se utiliza **GET** en un formulario que requiere que los datos sean presentados vía **POST**, aunque los dos métodos obligatorios, **GET** y **HEAD**, nunca deberían retornar este código de error.
406	Not Acceptable	El servidor no es capaz de devolver los datos en ninguno de los formatos aceptados por el cliente (indicados el *header* **`Accept`** de la petición).
407	Proxy Authentication Required	Similar a **401**, pero la autenticación debe realizarse a partir de un *proxy*.
408	Request Timeout	El *browser* ha tardado tanto tiempo en realizar su petición que el servidor ya no la espera.
409	Conflict	Cuando una petición tiene conflicto con el estado actual del servidor.
410	Gone	Cuando el recurso solicitado ha sido borrado permanentemente del servidor.
411	Length Required	Se rechaza la petición porque no se incluyó el *header* **`Content-Length`**.
412	Precondition Failed	La petición impone condiciones en sus *headers* que el servidor no puede cumplir.

Nro.	Detalle	Significado
413	Payload Too Large	La petición es demasiado larga y por ello el servidor no la procesa.
414	URI Too Long	La URI solicitada por el cliente es más larga de lo que el servidor está dispuesto a interpretar.
415	Unsupported Media Type	El formato multimedia de los datos solicitados no es soportado por el servidor.
416	Requested Range Not Satisfiable	El rango especificado por el *header* **`Range`** en la petición está fuera del tamaño de los datos objetivo del URI.
417	Expectation Failed	La expectativa indicada por el *header* **`Expect`** solicitada no puede ser cumplida por el servidor.
418	I'm a teapot	"Soy una tetera". Fue definido en 1998 como una inocentada. No se espera que los servidores web lo implementen realmente
421	Misdirected Request	La petición fue dirigida a un servidor que no es capaz de producir una respuesta.
422	Unprocessable Entity	La petición no se pudo continuar debido a errores de semántica.
423	Locked	El recurso al que se está teniendo acceso está bloqueado.
424	Failed Dependency	La petición falló debido a una falla en la petición anterior.
426	Upgrade Required	El servidor se rehúsa a aplicar la petición usando el protocolo actual, pero puede estar dispuesto a hacerlo después, si el cliente lo actualiza.
428	Precondition Required	El servidor requiere que la petición sea condicional (evita problemas al modificar con **PUT** un recurso que ha sido cambiado por otra parte)
429	Too Many Requests	El cliente ha enviado demasiadas peticiones en muy poco tiempo.
431	Request Header Fields Too Large	Los *headers* son demasiado largos.
451	Unavailable for Legal Reasons	Se solicitó un recurso ilegal, censurado por algún gobierno o tribunal.

https://developer.mozilla.org/es/docs/Web/HTTP/Status

Códigos de estado de respuesta HTTP (5xx)

Nro.	Detalle	Significado
500	Internal Server Error	El servidor ha encontrado una situación que no sabe cómo manejar.
501	Not Implemented	El método solicitado no está soportado por el servidor y no puede ser manejado. Los métodos **GET** y **HEAD** nunca deberían retornar este error.
502	Bad Gateway	El servidor, oficiando de *gateway* o *proxy*, obtuvo una respuesta inválida.
503	Service Unavailable	El servidor está temporalmente no disponible, por mantenimiento o por sobrecarga.
504	Gateway Timeout	El servidor, oficiando de *gateway* o *proxy*, no obtuvo una respuesta a tiempo.
505	HTTP Version Not Supported	La versión de HTTP usada en la petición no está soportada por el servidor.
506	Variant Also Negotiates	Indica un error de configuración del servidor en el que la variante elegida está configurada para participar en la negociación de contenido, por lo que no es un *endpoint* adecuado (resulta en una referencia circular).
507	Insufficient Storage	El servidor no puede almacenar, por falta de espacio, la representación necesaria para completar con éxito la petición.
508	Loop Detected	El servidor detectó un ciclo infinito mientras procesaba la petición.
510	Not Extended	Son requeridas extensiones adicionales para la petición, para que el servidor las cumpla.
511	Network Authentication Required	Indica que el cliente necesita autenticarse para obtener acceso a la red. Este error no es generado por los servidores de origen, sino por la interceptación de *proxies* que controlan el acceso a la red.

https://developer.mozilla.org/es/docs/Web/HTTP/Status

Tipos MIME más comunes

En el *header* **`Content-Type`** se especifica el formato del recurso recibido, según el tipo **MIME (Extensiones multipropósito de Correo de Internet)**, cuyo estándar es responsabilidad de la **IANA (Autoridad de Números Asignados de Internet)**.

El formato es **`tipo/subtipo`**

Tipo MIME	Significado	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, inclusive las animadas (como los gifs)	**`image/gif`, `image/png`, `image/jpeg`, `image/bmp`, `image/svg+xml`**
`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`, `video/avi`, `video/mpg`,**
`application`	Representa cualquier tipo de datos binarios.	**`application/octet-stream`, `application/xml`, `application/pdf`, `application/msword`, `application/excel`, `application/zip`**

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

Para los documentos binarios sin subtipo específico o conocido, se debe usar **`application/octet-stream`**.

https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/MIME_types

FIN DE LA PRESENTACIÓN

Encontrá más como estas en mi sitio web.