UD4.1:HTTP

UD4.1 - HTTP

1. ¿Qué es 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 (XHML, HTML...) en la **World Wide Web**. HTTP fue desarrollado por el **World Wide Web Consortium y la Internet Engineering Task Force**, colaboración que culminó en 1999 con la publicación de una serie de RFC, siendo el más importante de ellos el **RFC 2616** que especifica la versión 1.1. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse.

1.1. Terminología básica.

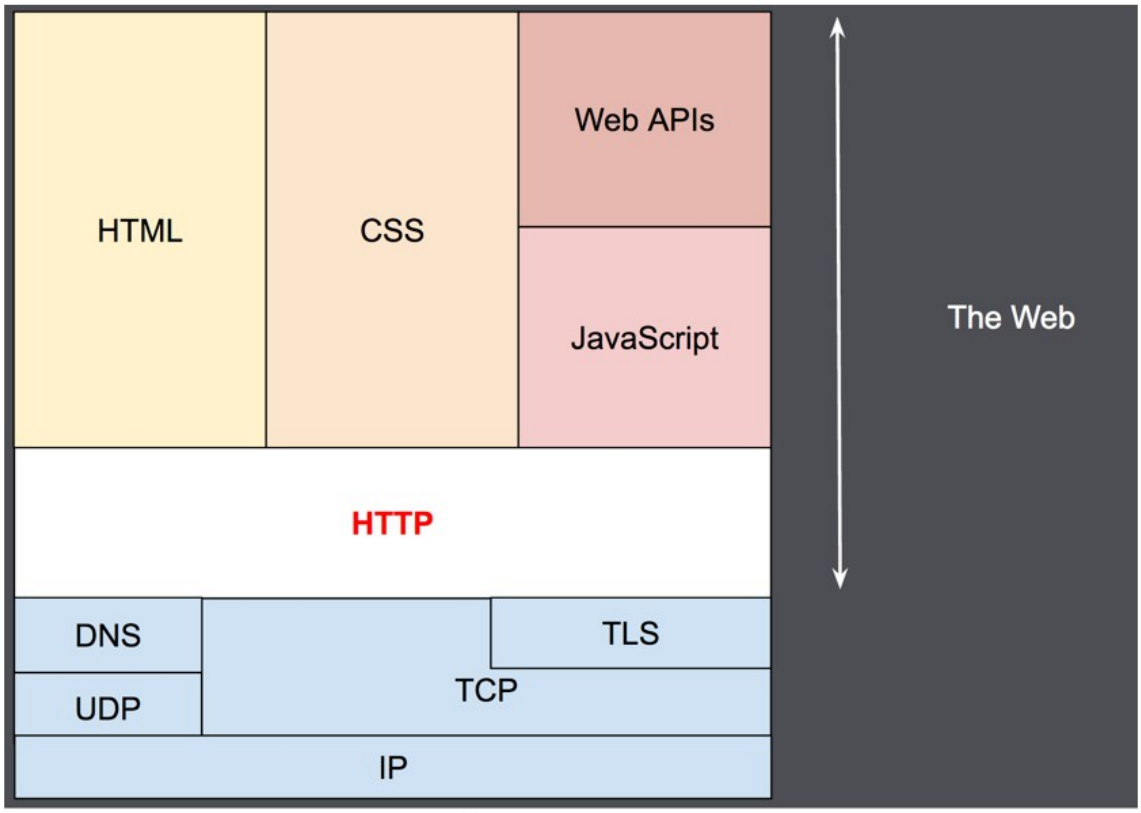
- **WWW:** La World Wide Web (www) es una red informática mundial accesible a través de Internet. Está formada por páginas web interconectadas que ofrecen diversos tipos de contenido textual y multimedia. Su función es ordenar y distribuir la información que existe en internet.
- **URL:** Uniform Resource Locator (Localizador de Recursos Uniforme). Una URL no es más que una dirección que es dada a un recurso único en la Web. En teoría, cada URL valida apunta a un único recurso. Dichos recursos pueden ser páginas HTML, documentos CSS, imágenes, etc.
- **CLIENTE:** también es conocido como agente del usuario, es cualquier herramienta que actué en representación del usuario. Esta función es realizada en la mayor parte de los casos por un navegador Web. Hay excepciones, como el caso de programas específicamente usados por desarrolladores para desarrollar y depurar sus aplicaciones.
- **SERVIDOR WEB:** "sirve" los datos que ha pedido el cliente. Un servidor conceptualmente es una unica entidad, aunque puede estar formado por varios elementos, que se reparten la carga de peticiones, (load balancing), u otros programas, que gestionan otros computadores (como cache, bases de datos, servidores de correo electrónico, ...), y que generan parte o todo el documento que ha sido pedido.

- **PROXIE:** dispositivos intermedios entre Cliente/Servidor que operan procesando la capa de aplicación; existen otros dispositivos intermedios como Routers o switches pero operan en capas más bajas como es conocido.

1.2. Arquitectura HTTP.

HTTP es un protocolo de la capa de **aplicación**, y se transmite sobre el **protocolo TCP**, o el protocolo encriptado **TLS (SSL)**, aunque teóricamente podría usarse cualquier otro protocolo fiable. Gracias a que es un protocolo capaz de ampliarse, se usa no solo para transmitir documentos de hipertexto (HTML), si no que además, se usa para transmitir imágenes o vídeos, o enviar datos o contenido a los servidores, como en el caso de los formularios de datos. HTTP puede incluso ser utilizado para transmitir partes de documentos, y actualizar páginas Web en el acto.

En la siguiente imagen se puede apreciar el orden de capas y protocolos asociados:



1.3. Evolución.

Fue inventado por **Tim Berners-Lee** entre los años 1989-1991, mientras trabajaba en el CERN. En 1989, Tim Berners-Lee escribió una propuesta para desarrollar un sistema de hipertexto sobre Internet. Inicialmente lo llamó: 'Mesh' (malla, en inglés), y posteriormente se renombró como World Wide Web (red mundial), durante su implementación en 1990. Desarrollado sobre los protocolos existentes TCP e IP, está basado en cuatro bloques:

- Un formato de texto para representar documentos de hiper-texto: **HyperText Markup Language (HTML)**.
- Un protocolo sencillo para el intercambio de esos documentos, del inglés: **HypertText Transfer Protocol (HTTP)** : protocolo de transferencia de hiper-texto.
- Un cliente que muestre (e incluso pueda editar) esos documentos, llamado **navegador**.
- Un servidor para dar acceso a los documentos: httpd (http daemon).

Estos cuatro bloques fundamentales se finalizaron para finales de 1990, y los primeros servidores estaban ya funcionando fuera del CERN a principios del 1991. El 6 de Agosto de 1991, el **post** de Tim Berners-Lee, se considera actualmente como el inicio oficial de la Web como proyecto público. La versión del protocolo HTTP usada en aquel momento, era realmente muy sencilla, posteriormente pasó a HTTP/0.9, referido algunas veces, como el protocolo de una sola línea.

HTTP ha pasado por múltiples versiones del protocolo, muchas de las cuales son compatibles con las anteriores. El **RFC 2145** describe el uso de los números de versión de HTTP. En la siguiente imagen se observa la evolución de las versiones de HTTP:

A continuación se muestra una breve descripción de las versiones.

- **0.9 (lanzada en 1991):** Obsoleta. Soporta solo un comando, GET, y además no especifica el número de versión HTTP. No soporta cabeceras.
- **HTTP/1.0 (mayo de 1996):** Esta es la primera revisión del protocolo que especifica su versión en las comunicaciones, y todavía se usa ampliamente, sobre todo en servidores proxy, aunque es muy limitada. Permite los métodos de petición GET, HEAD y POST.
- **HTTP/1.1 (junio de 1999):** Versión más usada actualmente; Las conexiones persistentes están activadas por defecto y funcionan bien con los proxies. También permite al cliente enviar múltiples peticiones a la vez por la misma conexión (pipelining) lo que hace posible eliminar el tiempo de **delay** por cada petición.
- **HTTP/2 (mayo de 2015):** Esta nueva versión no modifica la semántica de aplicación de http (todos los conceptos básicos continúan sin cambios). **Un protocolo para un mayor rendimiento** y sus mejoras se enfocan en como se empaquetan los datos y en el transporte. Por ejemplo, añade el uso de una única conexión y la compresión de cabeceras.
- **HTTP/3 (Octubre de 2018):** HTTP/3 es el sucesor propuesto de HTTP/2, aunque todavía es un borrador ya está en uso en la web; utilizando UDP en lugar de TCP para el protocolo de transporte subyacente. Fue desarrollado inicialmente por Google en el que se utiliza el control de congestión del espacio de usuario.

2. Características.

- **SENCILLO:** Incluso con el incremento de complejidad, que se produjo en el desarrollo de la versión del protocolo HTTP/2, en la que se encapsularon los mensajes, HTTP esta pensado y desarrollado para ser leído y fácilmente interpretado por las personas, haciendo de esta manera más fácil la depuración de errores, y reduciendo la curva de aprendizaje para las personan que empieza a trabajar con él.
- **EXTENSIBLE:** Presentadas en la versión HTTP/1.0, las cabeceras de HTTP, han hecho que este protocolo sea fácil de ampliar y de experimentar con él. Funcionalidades nuevas pueden desarrollarse, sin más que un cliente y su servidor, comprendan la misma semántica sobre las cabeceras de HTTP.
- **SIN ESTADOS:** HTTP es un protocolo sin estado, es decir: no guarda ningún dato entre dos peticiones en la misma sesión. Esto crea problemáticas, en caso de que los usuarios requieran interactuar con determinadas páginas Web de forma ordenada y coherente, por ejemplo, para el uso de "cestas de la compra" en páginas que utilizan en comercio electrónico. **SOLUCIONES**: El uso de HTTP cookies, si permite guardar datos con respecto a la sesión de comunicación. 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. Esto le permite a las aplicaciones web instituir la noción de **sesión**, y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.
- **HTTP Y CONEXIONES:** Una conexión se gestiona al nivel de la capa de trasporte, y por tanto queda fuera del alcance del protocolo HTTP. Aún con este factor, HTTP no necesita que el protocolo que lo sustenta mantenga una conexión continua entre los participantes en la comunicación, solamente necesita que sea un protocolo fiable o que no pierda mensajes (como mínimo, en todo caso, un protocolo que sea capaz de detectar que se ha pedido un mensaje y reporte un error).

3. Funciones de Control.

La característica del protocolo HTTP de ser ampliable, ha permitido que durante su desarrollo se hayan implementado más funciones de control y funcionalidad sobre la Web: caché o métodos de identificación o autentificación fueron temas que se abordaron pronto en su historia. Al contrario la relajación de la restricción de origen solo se ha abordado en los años de la década de 2010.

Se presenta a continuación los elementos más destacables que se pueden controlar con HTTP:

- **CACHE:** El como se almacenan los documentos en la caché, puede ser especificado por HTTP. El servidor puede indicar a los proxies y clientes, que quiere almacenar y durante cuanto tiempo. Aunque el cliente, también puede indicar a los proxies de caché intermedios que ignoren el documento almacenado.
- **AUTENTIFICACIÓN:** Hay páginas Web, que pueden estar protegidas, de manera que solo los usuarios autorizados puedan acceder. HTTP provee de servicios básicos de autentificación, por ejemplo mediante el uso de cabeceras como: WWW-Authenticate, o estableciendo una sesión especifica mediante el uso de HTTP cookies. Actualmente esta característica es menos usada, ya que es realizada mediante código PHP o Phyton.
- **PROXIES Y TUNNELING:** Servidores y/o clientes pueden estar en intranets y esconder así su verdadera dirección IP a otros. Las peticiones HTTP utilizan los proxies para acceder a ellos.
- **SESIONES:** El uso de HTTP cookies permite relacionar peticiones con el estado del servidor. Esto define las sesiones, a pesar de que por definición el protocolo HTTP es un protocolo sin estado. Esto es muy útil no sólo para aplicaciones de comercio electrónico, sino también para cualquier sitio que permita configuración al usuario.

4. Funcionamiento HTTP.

Cuando el cliente quiere comunicarse con el servidor, tanto si es directamente con él, o a través de un proxy intermedio, realiza los siguientes pasos:

1. Abre una conexión TCP: la conexión TCP se usará para hacer una petición, o varias, y recibir la respuesta. El cliente pude abrir una conexión nueva, reusar una existente, o abrir varias a la vez hacia el servidor.
2. Hacer una petición HTTP: Los mensajes HTTP (previos a HTTP/2) son legibles en texto plano. A partir de la versión del protocolo HTTP/2, los mensajes se encapsulan en franjas, haciendo que no sean directamente interpretables, aunque el principio de operación es el mismo.

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

3. Leer la respuesta enviada por el servidor:

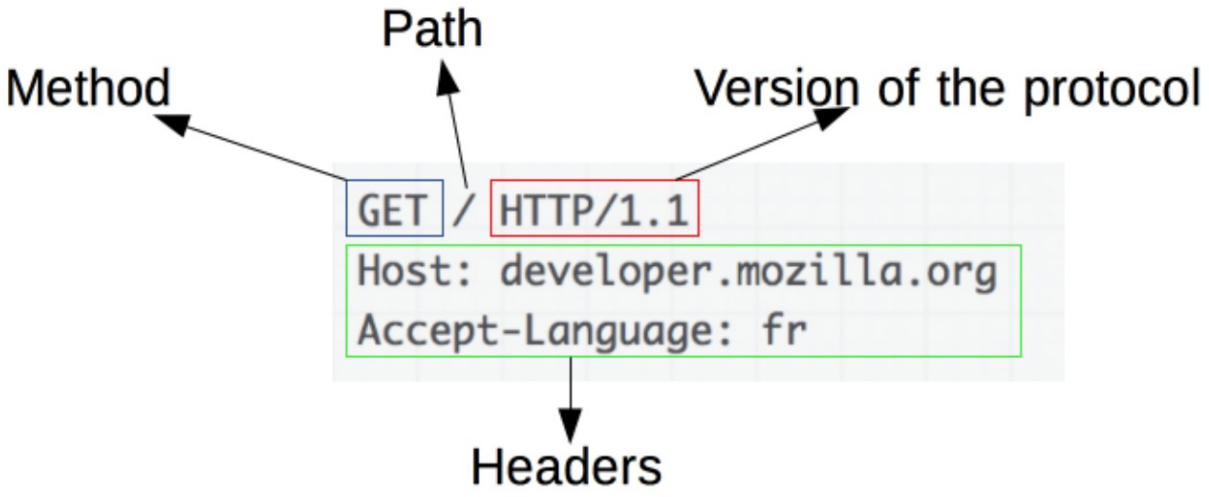
```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the reques
```

4. Cierre de la conexión para futuras peticiones.

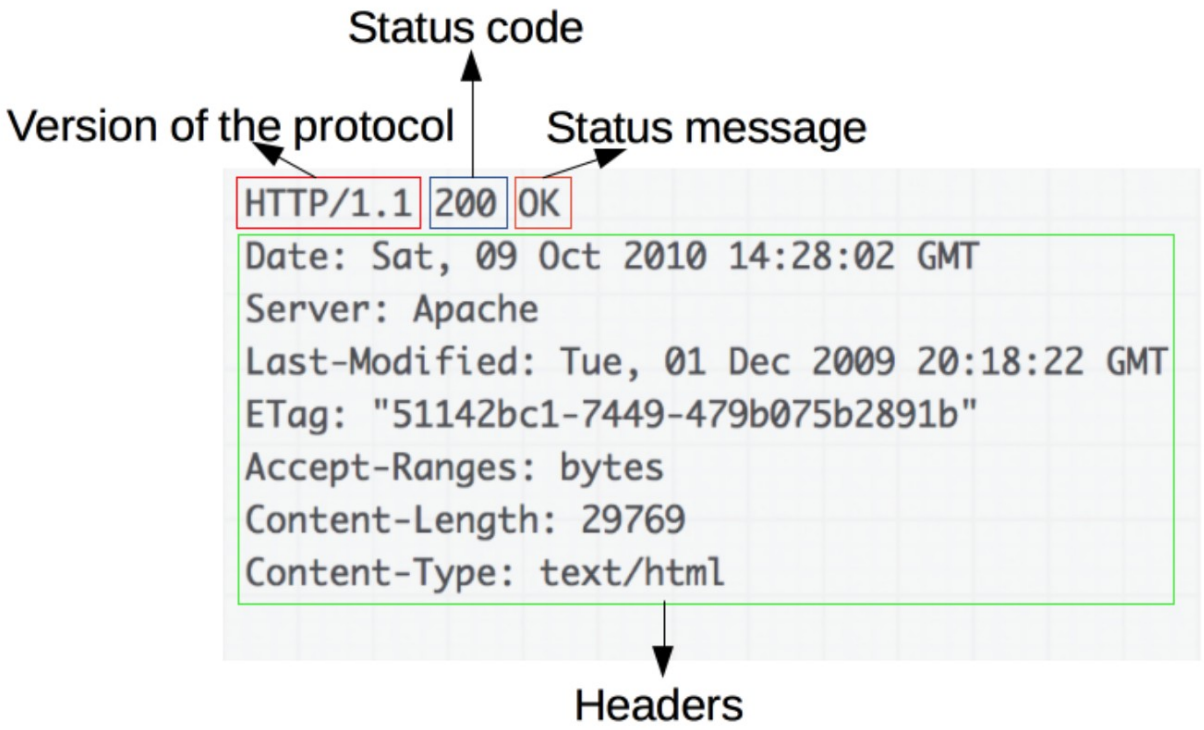
5. Mensajes HTTP.

Existen dos tipos de mensajes HTTP: peticiones y respuestas, cada uno sigue su propio formato.

- **PETICIONES:**Una petición de HTTP, está formado por los siguientes campos:
  - **Un método HTTP**, normalmente pueden ser un verbo, como: GET, POST o un nombre como: OPTIONS o HEAD, que defina la operación que el cliente quiera realizar. El objetivo de un cliente, suele ser una petición de recursos, usando GET, o presentar un valor de un formulario HTML, usando POST, aunque en otras ocasiones puede hacer otros tipos de peticiones.
  - **La dirección del recurso pedido**; la URL del recurso, sin los elementos obvios por el contexto, como pueden ser: sin el protocolo (http://), el dominio (aquí developer.mozilla.org), o el puerto TCP (aquí el 80).
  - **La versión del protocolo HTTP**.
  - **Cabeceras HTTP opcionales**, que pueden aportar información adicional a los servidores.
  - **Cuerpo de mensaje**, en algún método, como puede ser POST, en el cual envía la información para el servidor.



- **RESPUESTAS:** Las respuestas están formadas por los siguentes campos:
  - La versión del protocolo HTTP que están usando.
  - Un código de estado, indicando si la petición ha sido exitosa, o no, y debido a que.
  - Un mensaje de estado, una breve descripción del código de estado.
  - Cabeceras HTTP, como las de las peticiones.
  - Opcionalmente, el recurso que se ha pedido.



5.1. Métodos de petición.

HTTP define una serie predefinida de métodos de petición (algunas veces referido como "verbos") que pueden utilizarse. El protocolo tiene flexibilidad para ir añadiendo nuevos métodos y para así añadir nuevas funcionalidades. El número de métodos de petición se ha ido aumentando según se avanzaba en las versiones.

A continuación se describen los métodos más destacados:

- **GET:** El método GET solicita una representación del recurso especificado. Las solicitudes que usan GET solo deben recuperar datos y no deben tener ningún otro efecto. (Esto también es cierto para algunos otros métodos HTTP.)
- **HEAD:** Pide una respuesta idéntica a la que correspondería a una petición GET, pero en la respuesta no se devuelve el cuerpo. Esto es útil para poder recuperar los metadatos de los encabezados de respuesta, sin tener que transportar todo el contenido.
- **POST:** Envía datos para que sean procesados por el recurso identificado en la URI de la línea petición. Los datos se incluirán en el cuerpo de la petición. A nivel semántico está orientado a crear un nuevo recurso, cuya naturaleza vendrá especificada por la cabecera Content-Type. Ejemplos:
  - Para datos formularios codificados como una URL (aunque viajan en el cuerpo de la petición, no en la URL): application/x-www-form-urlencoded
  - Para bloques a subir, ej. ficheros: multipart/form-data
  - Además de los anteriores, no hay un estándar obligatorio y también podría ser otros como text/plain, application/json, application/octet-stream,...
- **PUT:** Envía datos al servidor, pero a diferencia del método POST la URI de la línea de petición no hace referencia al recurso que los procesará, sino que identifica al los propios datos (ver explicación detallada en el RFC). Otra diferencia con POST es semántica,: mientras que POST está orientado a la creación de nuevos contenidos, **PUT está más orientado a la actualización de los mismos (aunque también podría crearlos)**.
- **DELETE:** Borra el recurso especificado.

5.2. Códigos de respuesta.

El código de respuesta o retorno es un número que indica que ha pasado con la petición. El resto del contenido de la respuesta dependerá del valor de este código. El sistema es flexible y de hecho la lista de códigos ha ido aumentando para así adaptarse a los cambios e identificar nuevas situaciones. Cada código tiene un significado concreto. Sin embargo el número de los códigos están elegidos de tal forma que según si pertenece a una centena u otra se pueda identificar el tipo de respuesta que ha dado el servidor:

- **Códigos con formato 1xx:** Respuestas informativas. Indica que la petición ha sido recibida y se está procesando.
- **Códigos con formato 2xx:** Respuestas correctas. Indica que la petición ha sido procesada correctamente.
- **Códigos con formato 3xx:** Respuestas de redirección. Indica que el cliente necesita realizar más acciones para finalizar la petición.
- **Códigos con formato 4xx:** Errores causados por el cliente. Indica que ha habido un error en el procesado de la petición a causa de que el cliente ha hecho algo mal.
- **Códigos con formato 5xx:** Errores causados por el servidor. Indica que ha habido un error en el procesado de la petición a causa de un fallo en el servidor.

5.3. Cabeceras.

Son los metadatos que se envían en las peticiones o respuesta HTTP para proporcionar información esencial sobre la transacción en curso. Cada cabecera es especificada por un nombre de cabecera seguido por dos puntos, un espacio en blanco y el valor de dicha cabecera seguida por un retorno de carro seguido por un salto de línea. Se usa una línea en blanco para indicar el final de las cabeceras. Si no hay cabeceras la línea en blanco debe permanecer.

Las cabeceras le dan gran flexibilidad al protocolo permitiendo añadir nuevas funcionalidades sin tener que cambiar la base. Por eso según han ido sucediendo las versiones de HTTP se han ido añadiendo más y más cabeceras permitidas.

Las cabeceras pueden tener metadatos que tienen que ser procesados por el cliente, **por ejemplo** el servidor en respuesta a petición del cliente se puede indicar el tipo del contenido que contiene, y por parte del cliente los tipos de representaciones aceptables; por el cliente del contenido que pide o por los intermediarios como gestionar el cacheo por parte de los proxys.

Dependiendo del tipo de mensaje en el que puede ir una cabecera las podemos clasificar en **cabeceras de petición, cabeceras de respuesta y cabeceras que pueden ir tanto en una petición como en una respuesta**.

Además se pueden clasificar las cabeceras según su función:

- Cabeceras para **describir la comunicación**: **Host** (indica máquina destino del mensaje), **Connection** (indica como establecer la conexión)
- Cabeceras que indican **las capacidades aceptadas** por el que envía el mensaje: **Accept** (indica el MIME aceptado), **Accept-Charset** (indica el código de caracteres aceptado), **Accept-Encoding** (indica el método de compresión aceptado), **Accept-Language** (indica el idioma aceptado), **User-Agent** (para describir al cliente), **Server** (indica el tipo de servidor), **Allow** (métodos permitidos para el recurso).
- Cabeceras que **describen el contenido**: **Content-Type** (indica el MIME del contenido), **Content-Length** (longitud del mensaje), **Content-Range, Content-Encoding, Content-Language, Content-Location**.
- **Cabeceras que hacen referencias a URIs**: **Location** (indica donde está el contenido), **Referer** (Indica el origen de la petición).
- Cabeceras que permiten **ahorrar transmisiones**: **Date** (fecha de creación), **If-Modified-Since**, **If-Unmodified-Since, If-Match, If-None-Match, If-Range, Expires, Last-Modified, Cache-Control, Via, Pragma, Etag, Age, Retry-After.**
- Cabeceras para **control de cookies**: **Set-Cookie, Cookie**
- Cabeceras para **autentificación**: **Authorization, WW-Authenticate**

6. Gestión de la conexión.

Crear y mantener una conexión tiene una gran influencia en el rendimiento de las páginas Web y las aplicaciones Web. En la versión HTTP/1.x, hay modos de conexión: conexiones breves, conexiones persistentes, y 'pipelining'.

- **CONEXIONES BREVES**: se crea conexión sobre TCP una cada vez que una petición necesitaba ser transmitida, y se cerraba, una vez que la respuesta se había recibido. Este sencillo modelo tenía una limitación intrínseca en su rendimiento: abrir una conexión TCP es una operación costosa en recursos. Se necesitan intercambiar varios mensajes entre el cliente y el servidor para hacerlo. Y la latencia de la red y su ancho de banda, se veían afectados cuando se realizaba una petición. Las páginas web actuales, necesitan varias peticiones (una docena o más) para tener la información necesaria, de manera que este primer modelo es ineficiente para ellas.
- **CONEXIÓN PERSISTENTE:** mantiene las conexiones abiertas, entre peticiones sucesivas, eliminando así el tiempo necesario para abrir nuevas conexiones.
- **MODELO "PIPELINING":** va un paso más allá, y envía varias peticiones sucesivas, sin esperar por la respuesta, reduciendo significativamente la latencia en la red.