Protocolo HTTP

Objetivos

- Comprender el funcionamiento del protocolo HTTP usado en las aplicaciones web para la comunicación entre clientes (navegadores) y servidores.
- Entender los principales conceptos del esquema de direcciones URI y URL
- Entender los principales elementos de los mensajes y verbos que hacen parte del protocolo HTTP.

Introducción

En esta guía se explican los principales elementos del esquema de direccionamiento URI / URL y el protocolo HTTP usado en las aplicaciones web.

HTTP (hypertext Transfer Protocol) es el protocolo de comunicaciones usado en la Web para intercambiar documentos HTML, archivos CSS, Javascript, imágenes y otros recursos similares. El protocolo HTTP sigue un esquema petición-respuesta en donde (1) un navegador web, el cliente del protocolo, envía un mensaje de petición a un servidor web y, en consecuencia (2) el servidor retorna un mensaje de respuesta.

En HTTP, cada mensaje de petición y de respuesta se compone de un conjunto de líneas de texto. Por ejemplo, el mensaje de petición incluye una primera línea de texto que incluye la operación (conocida como método o verbo) que desea ejecutar el cliente, un conjunto de líneas adicionales con campos de encabezado, una línea vacía que representa el final del encabezado y, opcionalmente, el texto del mensaje. Esta simplicidad en el formato de los mensajes ha facilitado el desarrollo de una gran variedad de clientes y servidores web.

La primera versión del protocolo HTTP fué propuesta por Tim Berners-Lee al crear la World-Wide-Web en el CERN. La versión 0.9 fue la primera con una documentación técnica completa (en 1991). La versión 1.0 fue estandarizada por el HTTP Working Group (HTTP WG) en 1995 como el RFC 1945. La versión 1.1, la más usada en la actualidad, fue presentada por el mismo grupo en 1996 y revisada en 2007. En los últimos años, varias empresas presentaron propuestas de mejoramiento al protocolo. En 2012 Google presentó SPDY como una propuesta que elimina redundancias en los encabezados y

lograba grandes mejoras de rendimiento. Estas propuestas fueron la base para la más reciente versión HTTP/2 aprobada como un estándar en 2015.

Identificador de los Recursos en Internet

Desde la perspectiva de los usuarios, un navegador web es capaz de utilizar protocolos como HTTP para acceder a páginas web, imágenes y otros recursos. Los usuarios usualmente ingresan en el navegador web una dirección que sigue un formato conocido como URL (Universal Resource Locator)[1].

Las URL siguen un formato específico:

```
scheme://[user:password@]host[:port]/path[?queryString][#fragment]
```

scheme:	Define el mecanismo de acceso al recurso. Normalmente corresponde con el protocolo de comunicaciones, por ejemplo http, ftp o https. En algunos casos es un identificador usado por el navegador para determinar que programa o plugin debe procesar el recurso, por ejemplo mailto, tel o skype.
user:	(Opcional) nombre del usuario que accede al recurso o es propietario del mismo.
password:	(Opcional) contraseña del usuario
host:	Nombre completo o dirección IP del servidor que alberga el recurso.
port:	(Opcional) Número del puerto de red usado por el servidor. Si el URL no incluye el número del puerto, usualmente el sistema usa el puerto por defecto del protocolo. Por ejemplo, para el protocolo http el puerto por defecto es el 80.
path:	(Opcional) Dirección del recurso dentro del servidor.
queryString:	(Opcional) Lista de parámetros usadas para acceder al recurso. Los parámetros se componen de pares “nombre=valor” separados por “&”
fragment:	(Opcional) Referencia a un fragmento dentro del recurso. Por ejemplo una sección de una página web.

[1] En Computación, un URI (Universal Resource Identifier) es una cadena de texto que se permite identificar de forma inequívoca algún recurso en particular. Existen varios tipos de URI. Por ejemplo, se conocen como URN (Universal Resource Names) identificadores que se definen en un espacio de nombres particular, como los ISBN e ISSN usados para identificar libros y revistas. URL es el esquema definido para identificar recursos en la web.

Métodos de Acceso

Un navegador web puede realizar una serie de diferentes operaciones sobre los recursos dependiendo del protocolo que se utiliza. El protocolo HTTP define un conjunto operaciones. Todas las peticiones HTTP deben incluir el tipo de operación que se desea realizar y el recurso sobre el cual se desea hacer la operación.

Existen una gran variedad de métodos de las peticiones HTTP (request methods). Los métodos más comunmente utilizados incluyen opciones para obtener, grabar o borrar recursos (por ejemplo archivos). Estos comandos, usados por ejemplo como parte de la arquitectura REST[2], se describen a continuación:

GET:	Retorna la información (en forma de entidad) asociada al recurso identificado con la URI solicitada.
POST:	Éste método es usado para que el servidor acepte la entidad enviada como parte de la petición, como un nuevo elemento del recurso asociado a la URI solicitada.
PUT:	Éste método es usado para que la entidad enviada como parte del request sea guardada bajo la URI solicitada. Si la entidad se refiere a un recurso ya existente, se procesa como una entidad actualizada.
DELETE:	Este método indica al servidor que el recurso identificado con la URI solicitada debe ser eliminado.

Existen otros métodos de las peticiones http que pueden ser usados por navegadores y servidores para coordinar su funcionamiento. Algunos de estos métodos se presentan a continuación:

OPTIONS:	Representa una petición sobre las opciones de comunicación disponibles en la cadena petición/respuesta identificada por la URI solicitada. Éste método permite al cliente identificar las opciones asociadas a un recurso o capacidades de un servidor, sin iniciar una acción sobre el recurso.
HEAD:	Su funcionamiento es idéntico al de GET, con la excepción que no retorna el cuerpo de la respuesta. Solo se reciben los datos del encabezado.
TRACE:	Éste método es usado para que el cliente pueda obtener información sobre el flujo de mensajes con el servidor.
CONNECT:	La especificación reserva éste método para ser usado con un proxy que puede cambiar dinámicamente para ser un tunel.

[2] https://sistemas.uniandes.edu.co/~recursos-csw/dokuwiki/doku.php?id=4capas:tecnologias:servicios:rest

Estructura de una Petición HTTP

Para poder realizar cualquiera de las operaciones sobre los recursos, el navegador web debe envia un mensaje de solicitud. Este mensaje debe contener como mínimo el método que desea ejecutar, el recurso y la versión del protocolo que se está usando. Opcionalmente el mensaje incluye un conjunto de encabezados y/o un contenido. A continuación se describe el formato de este tipo de mensajes.

Línea inicial:	`<Método> <urn (también se acepta URI)> <HTTP/version>`
Encabezado:	`<llave>: <valor>` `[<llave>: <valor>]`
Fin Encabezado:	
Cuerpo:	(Usado en métodos como POST y PUT)

Estructura de una Respuesta HTTP

A su vez, ante cualquier petición, el servidor web genera un mensaje de respuesta que es enviado de regreso al navegador. Este mensaje incluye, como mínimo, un código de respuesta y un mensaje (por ejemplo, un mensaje de error). Cuando la petición ha sido exitosa, normalmente la respuesta incluye un conjunto de encabezados y un contenido. A continuación se describe el formato de estos mensajes:

Línea inicial:	`HTTP/<version> <código de respuesta> <mensaje de código>`
Encabezado:	`<llave>: <valor>` `[<llave>: <valor>]`
Fin Encabezado:	
Cuerpo:	(Contenido del recurso, usado en métodos como GET)

Códigos de Respuesta

A continuación se detallan los códigos de respuesta generales con algunos de los más comunes que pueden encontrarse en aplicaciones web.

- 1XX Información: No es usado frecuentemente, solo proveen información.
    - 100 Continue
    - 101 Switching Protocols

- 2XX Exitoso: Usado para indicar que el request ha sido aceptado exitosamente.
    - 200 OK
    - 201 Created
    - 202 Accepted
- 3XX Redirección: Normalmente indica que se debe tomar alguna otra acción.
    - 300 Multiple Choices
    - 301 Moved Permanently
    - 302 Found
- 4XX Error de Cliente: Se usa cuando se identifica un error del cliente.
    - 400 Bad Request
    - 401 Unauthorized
    - 402 Payment Required
    - 403 Forbidden
- 5XX Error de Servidor: Se usa cuando se puede identificar que el servidor falla al procesar el request.
    - 500 Internal server error
    - 501 Not implemented
    - 502 Bad Gateway
    - 503 Service Unavailable

Tipos de contenido (MIME-Types)

Los MIME-types son identificadores estándar de los contenidos de internet, se usan para que los navegadores o clientes de correo puedan identificar el tipo de contenido que estan recibiendo. La estructura general contiene un tipo, un subtipo y parámetros opcionales de la siguiente forma: “<Tipo>/<Subtipo> ; <parámetros opcionales>”.

Por ejemplo, una respuesta HTTP puede contener en el encabezado una línea de texto indicando que el contenido es un texto, con el subtipo HTML:

```
Content-Type: text/html ; charset=UTF-8
```

Algunos tipos de contenido son los siguientes:

- text/html
- image/png
- application/json

Ejemplos de Peticiones y Respuestas HTTP

Los siguientes son algunos ejemplos de peticiones y respuestas:

1.

Servidor:	google.com
Puerto:	80
Petición:	GET /search?q=Uniandes HTTP/1.1
Respuesta:	HTTP/1.1 200 OK Date: Wed, 27 May 2015 14:02:34 GMT Expires: -1 Cache-Control: private, max-age=0 Content-Type: text/html; charset=ISO-8859-1 Set-Cookie: PREF=ID=e599f478e510c54f:FF=0:TM=1432735354:LM=1432735354:S=7q3FOyJWvZts-yL2; expires=Fri, 26-May-2017 14:02:34 GMT; path=/; domain=.google.com Set-Cookie: NID=67=VhWwrx5t9loAP1….FUg4HUBjUuWKcBiGvgtJ9c; expires=Thu, 26-Nov-2015 14:02:34 GMT; path=/; domain=.google.com; HttpOnly P3P: CP="This is not a P3P policy! See http://www.google.com/support/accounts/bin/answer.py?hl=en&answer=151657 for more info." Server: gws X-XSS-Protection: 1; mode=block X-Frame-Options: SAMEORIGIN Alternate-Protocol: 80:quic,p=0 Accept-Ranges: none Vary: Accept-Encoding Transfer-Encoding: chunked 8000 <!doctype html><html itemscope="" itemtype="http://schema.org/SearchResultsPage" lang="en"> …..

2.

Servidor:	youtube.com
Puerto:	80
Petición:	GET /search?q=Uniandes HTTP/1.1
Respuesta:	HTTP/1.1 200 OK Date: Wed, 27 May 2015 17:28:59 GMT Expires: -1 Cache-Control: private, max-age=0 Content-Type: text/html; charset=ISO-8859-1 Set-Cookie: PREF=ID=da90a3608ef954bd:FF=0:TM=1432747739:LM=1432747739:S=nmJDWxlWFYUxU5-C; expires=Fri, 26-May-2017 17:28:59 GMT; path=/; domain=.google.com Set-Cookie: NID=67=r6Lvg_XiYHC0ydLY0rYFeGB8ALm1SwpyUC22Nn9KCXbGpbKqRA8T91i6h9REtERb06qZZ cWTzr-A_rQhhKBdlY3FClOBLnfFVJWqQW6fdaUBVPGz3vGPNPHGPCvo8QRL; expires=Thu, 26- Nov-2015 17:28:59 GMT; path=/; domain=.google.com; HttpOnly P3P: CP="This is not a P3P policy! See http://www.google.com/support/accounts/bin/answer.py?hl=en&answer=151657 for more info." Server: gws X-XSS-Protection: 1; mode=block X-Frame-Options: SAMEORIGIN Alternate-Protocol: 80:quic,p=0 Accept-Ranges: none Vary: Accept-Encoding Transfer-Encoding: chunked

	8000 <!doctype html> …..

NOTA: Cuando un servidor está detrás de un proxy o un virtual host, es necesario indicar en el encabezado de la petición el host:

3.

Servidor:	**uniandes.edu.co**
Puerto:	**80**
Petición:	GET /institucional/informacion-general/la-universidad HTTP/1.1
Respuesta:	HTTP/1.1 404 Not Found Server: Apache/2.2 Content-Type: text/html; charset=iso-8859-1 Date: Wed, 27 May 2015 17:21:08 GMT Connection: Keep-Alive Set-Cookie: X-Mapping-feojnbhb=40B486778BB1B3158926C133D77814FB; path=/ Content-Length: 313 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN"> <html><head> <title>404 Not Found</title> </head><body> Not Found The requested URL /institucional/informacion-general/la-universidad was not found on this server. <hr> <address>Apache/2.2 Server at 184.106.55.80 Port 80</address> </body></html>

4.

Servidor:	**uniandes.edu.co**
Puerto:	**80**
Petición:	GET /institucional/informacion-general/la-universidad HTTP/1.1 Host: uniandes.edu.co
Respuesta:	HTTP/1.1 200 OK Server: Apache/2.2 Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0 Content-Type: text/html; charset=utf-8 P3P: CP="NOI ADM DEV PSAi COM NAV OUR OTRo STP IND DEM" Date: Wed, 27 May 2015 17:33:07 GMT Status: 200 OK Expires: Mon, 1 Jan 2001 00:00:00 GMT Pragma: no-cache Transfer-Encoding: chunked Connection: Keep-Alive Set-Cookie: X-Mapping-feojnbhb=93A016B6EEBCFDAFBA3FBE76A772C607; path=/ Set-Cookie: cac5c64b455631a9dfaf45c637182793=32q70uho8c7jpgbv8thj15br57; path=/ Last-Modified: Wed, 27 May 2015 17:33:09 GMT 3AE <?xml version="1.0" encoding="utf-8"?> …

5.

Servidor:	**uniandes.edu.co**
Puerto:	**80**
Petición:	GET /images/stories/logo_uniandes.gif HTTP/1.0 Host: uniandes.edu.co
Respuesta:	HTTP/1.1 200 OK Server: Apache/2.2 Content-Type: image/gif Date: Thu, 28 May 2015 17:44:52 GMT Accept-Ranges: bytes Cneonction: Keep-Alive Set-Cookie: X-Mapping-feojnbhb=A20399BE76FE616E824D5633C43FEB1E; path=/ Last-Modified: Thu, 16 Dec 2010 14:36:14 GMT X-Cache-Info: caching Cteonnt-Length: 5526 GIF89a?K??????????????¸??ebc???EBB???$!!???*MJKURSljk=9:0,,??????JE??TO??!-)*r???,?H???*\Ò?Ç#J?H??ŋ3j?õ?Ĭ C?I??ƀ(S?\ŋ?-0cнI??ʻ8s??ŋ?ϟ@????B?m??]?s?I …

Evolución del Protocolo

Como se mencionó al comienzo de la guía, el protocolo HTTP ha evolucionado con los años. Los métodos, el contenido de los mensajes y el funcionamiento mismo del protocolo han sido modificado para atender los requerimientos que han ido surgiendo con la evolución de la web.

Versión 0.9:	● El servidor acepta únicamente el método GET, y la conexión se hace por TCP-IP. ● La respuesta se espera que sea únicamente un texto HTML. ● la conexión se cierra una vez el documento ha sido entregado.
Version 1.0:	● Introduce mensajes tipo MIME para soportar otros tipos de contenido. ● Introduce métodos HEAD y POST ● El mensaje es más parecido al definido anteriormente ● La conexión se cierra una vez se entrega la respuesta completa ● Introduce los códigos de respuesta
Versión 1.1:	● Se introducen todos los métodos detallados en este documento ● Se hace un mejor manejo de la conexión, la conexión puede quedar abierta y mejora latencia. ● Maneja el concepto de chaching para mejorar rendimiento. ● Extiende las definiciones de códigos de respuesta y agrega un campo de advertencia en el encabezado.
Versión 2.0:	● Basado en las propuestas de mejoras realizadas por Google en su propuesta de SPDY. ● Cuando se transmiten varios recursos en la misma conexión “keep-alive”, permite omitir datos redundantes. Por ejemplo, evita enviar dos veces los mismos encabezados.

Bibliografía

- W3C. Protocolo HTTP 1.0. http://www.w3.org/Protocols/HTTP/1.0/spec.html#Status-Line
- W3C. Protocolo HTTP 1.1: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
- Google. SPDY: https://www.chromium.org/spdy
- W3C. Protocolo HTTP/2: https://http2.github.io