HTTP

Manuel Molino Milla

27 de octubre de 2016

Índice

1. Introducción

1.1. ¿Qué es HTTP

- Es acrónimo de *Hypertext Transfer Protocol*.
- Es un protocolo de comunicación que permite las transferencias de información en la World Wide Web
- Fue desarrollado por el *World Wide Web Consortium* y la *Internet Engineering Task Force*
- HTTP es un protocolo sin estado, es decir, 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.
- 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.

1.2. Versiones HTTP

0.9 (lanzada en 1991) Obsoleta. Soporta sólo un comando, GET, y además no especifica el número de versión HTTP. No soporta cabeceras. Como esta versión no soporta POST, el cliente no puede enviarle mucha información al servidor.

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. 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)

HTTP/1.2 Ha quedado como experimental.

HTTP/2 (mayo de 2015) Esta nueva versión no modifica la semántica de aplicación de http (todos los conceptos básicos continúan sin cambios). Sus mejoras se enfocan en como se empaquetan los datos y en el transporte.

1.3. Descripción de HTTP

- Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.
- El cliente (se le suele llamar *user agent*) realiza una petición enviando un mensaje,
- El servidor (se le suele llamar un servidor web) le envía un mensaje de respuesta
- Ejemplos de cliente son los navegadores web y los spider.
- Los mensajes HTTP son en texto plano lo que lo hace más legible y fácil de depurar.

1.4. Petciones HTTP

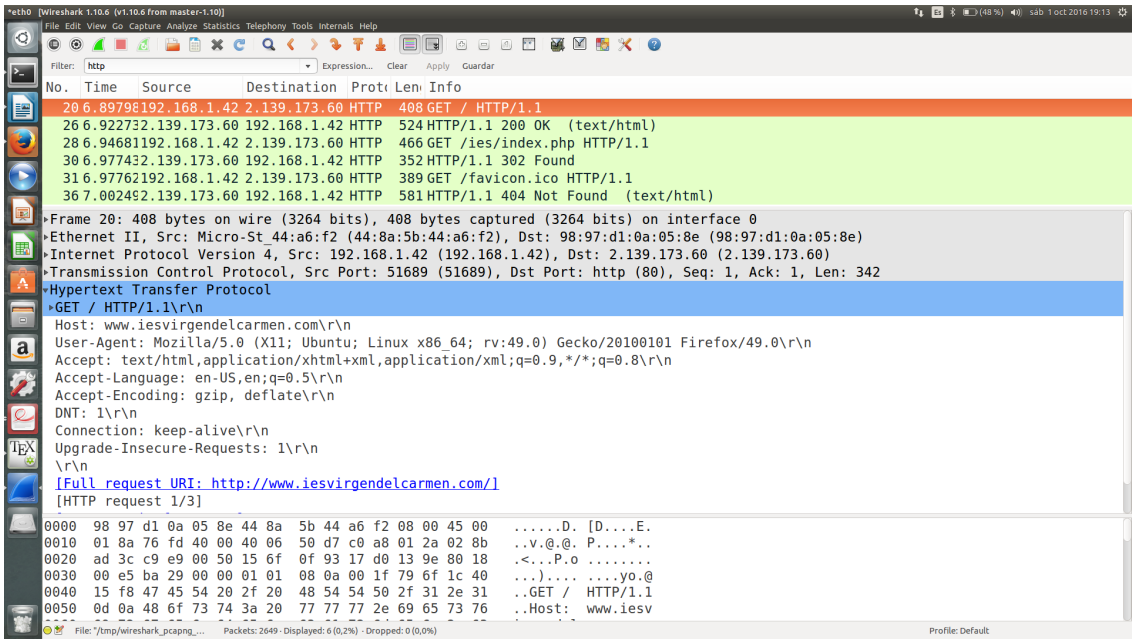
Ejemplo de petición HTTP y su correspondiente respuesta:

```
usuario@portatil:~$ telnet www.iesvirgendelcarmen.com 80
Trying 2.139.173.60...
Connected to iesvirgendelcarmen.com.
Escape character is '^]'.
GET / HTTP/1.1
Host:www.iesvirgendelcarmen.com

HTTP/1.1 200 OK
Date: Sat, 01 Oct 2016 17:09:46 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Fri, 21 Dec 2012 12:13:07 GMT
ETag: "88-4d15bc52d8a2c"
Accept-Ranges: bytes
Content-Length: 136
Vary: Accept-Encoding
Content-Type: text/html

<html>
<head>
<script>
document.location.href="http://www.iesvirgendelcarmen.com/ies/index.php"
</script>
</head>
<body>
</body>
<html>
```

Ahora la petición la hacemos desde un navegador:



1.4.1. Métodos de petición HTTP

- 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 han ido aumentando según se avanzaban en las versiones
- Cada método indica la acción que desea que se efectúe sobre el recurso identificado.
- Métodos mas comunes:

  HEAD Se solicita solo la cabecera de respuesta, no el cuerpo.

  GET Solicitamos tanto cabecera como respuesta.

  POST Envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Ejemplo un formulario.

  PUT Sube, carga o realiza un upload de un recurso especificado.

  DELETE Borra el recurso especificado.

OPTIONS Indica los métodos soportados por el servidor.

Ejemplo de uso del método *OPTIONS*:

```
usuario@portatil:~$ telnet www.iesvirgendelcarmen.com 80
Trying 2.139.173.60...
Connected to iesvirgendelcarmen.com.
Escape character is '^]'.
OPTIONS / HTTP/1.1
Host:www.iesvirgendelcarmen.com

HTTP/1.1 200 OK
Date: Sat, 01 Oct 2016 17:48:23 GMT
Server: Apache/2.4.7 (Ubuntu)
Allow: GET,HEAD,POST,OPTIONS
Content-Length: 0
Content-Type: text/html
```

1.5. Respuesta HTTP

Las respuestas constan de una **cabecera** y un **cuerpo**. En la cabecera nos encontramos datos como:

- Content-Length (longitud del mensaje)
- Accept-Encoding (indica el método de compresión aceptado)
- Accept-Charset (indica el código de caracteres aceptado)
- Accept (indica el MIME aceptado)
- Server (indica el tipo de servidor)
- Location (indica donde está el contenido)
- Date (fecha de creación)
- Set-Cookie, Cookie para las cookies.
- ...

También se encuentran los códigos de respuesta que indica que ha pasado con la petición:

Códigos con formato 1xx: 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.

Reply Code	Reason Phrase	Definition
200	OK	The request was successful.
201	Created	The request was successful and a new resource was created.
202	Accepted	The request was accepted for processing, but the processing is not yet complete.
204	No Content	The server has processed the request but there is no new information to be returned.
300	Multiple	The requested resource is available at one or more locations.
301	Moved Permanently	The requested resource has been assigned a new URL and any further references should use this new URL.
302	Moved Temporarily	The requested resource resides at a different location, but will return to this location in the future.
304	Not Modified	The requested resource has not been modified since the date specified in the If-Modified_Since header.
400	Bad Request	The server could not properly interpret the request.
401	Unauthorized	The request requires user authorization.
403	Forbidden	The server has understood the request and has refused to satisfy it.
404	Not Found	The server cannot find the information specified in the request.
500	Internal Server Error	The server could not satisfy the request due to an internal error condition.
501	Not Implemented	The server does not support the requested feature.
502	Bad Gateway	The server received an invalid response from the server from which it was trying to retrieve information.
503	Service Unavailable	The server cannot process this request at the current time.

1.6. Protocolos TCP/IP

Hypertext Transfer Protocol (HTTP)

Familia	Familia de protocolos de Internet
Función	Transferencia de hipertexto
Última versión	2.0
Puertos	80/TCP

Ubicación en la pila de protocolos

Aplicación	**HTTP**
Transporte	TCP
Red	IP

Estándares

RFC 1945 (HTTP/1.0, 1996)
RFC 2616 (HTTP/1.1, 1999)
RFC 2774 (HTTP/1.2, 2000)
RFC 7540 (HTTP/2.0, 2015)

editar datos en Wikidata

1.7. URI

- URI son las siglas en inglés de Uniform Resource Identifier (en español identificador uniforme de recursos),
- Sirve para identificar recursos en Internet
- Tene un formato estándar definido

```
esquema://máquina/directorio/archivo#fragmento
```

- Ejemplos:

  http http:www.pierobon.org/iis/review1.htm

  mailto mailto:someone@example.com

  file file:///home/someuser/somefile.txt

En relación a las URL:

- URL son las siglas en inglés de uniform resource locator (en español, localizador uniforme de recursos)
- Sirve para nombrar recursos en Internet.
- Su propósito es asignar una dirección única a cada uno de los recursos disponibles en Internet, como por ejemplo páginas, imágenes, vídeos, ...
- Una URL tiene un formato estándar, que es:

```
esquema://máquina/directorio/archivo
```

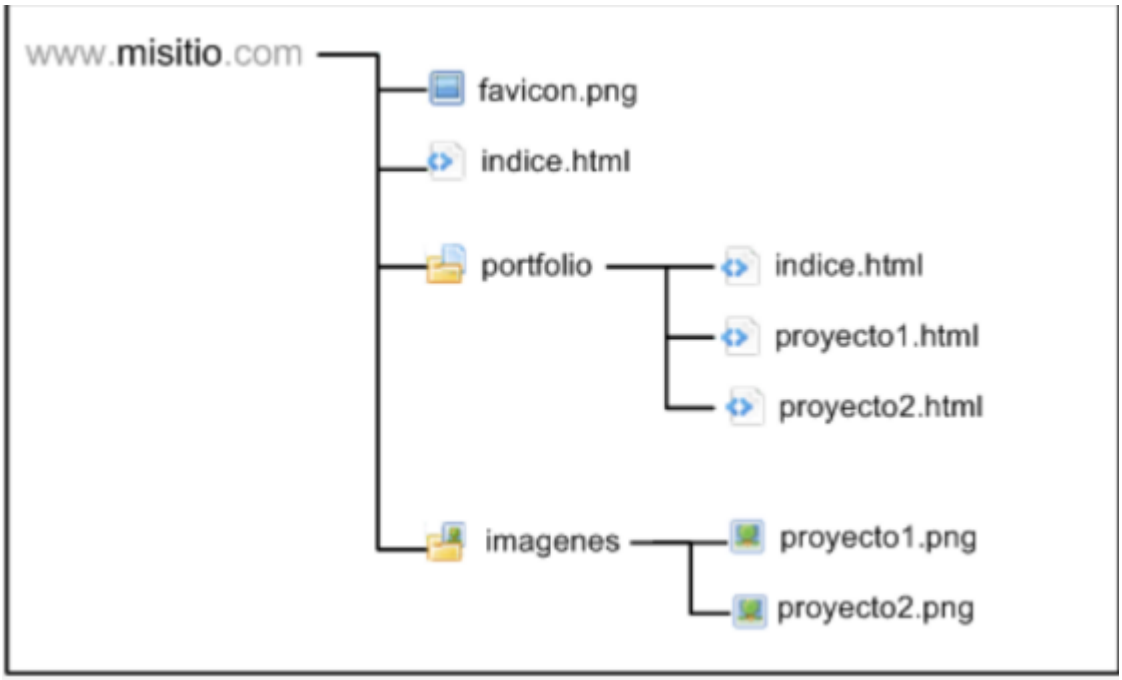
- El formato específico para HTTP es:

```
http://<máquina>:<puerto>/<path>?<cadena de búsqueda>
```

- Se le llama URL semántica a una URL que tiene un formato más fácilmente entendible o interpretable por alguien que la lee.
- Ejemplo:

```
URL no semántica: http://ejemplo.com/principal.php?page=noticias
URL semántica: http://ejemplo.com/noticias
```

El path hace referencia donde se encuentra ubicado el recurso en el servidor web:



La URL para acceder pueden ser:

```
www.sitio.com
www.sitio.com/portafolio
```

Las direcciones a utilizar pueden ser relativas o absolutas a la hora de especificar los recursos en el sitio web. Ejemplo:

- / *especifica el sitio*
- /portafolio *especifica la ubicación del recurso de forma absoluta*
- portafolio *especifica la ubicación del recurso de forma relativa*
- /imagenes/proyecto1.png *absoluta*
- imagenes/proyecto1.png *relativa*

1.8. Ejercicio

- Instala un servidor web en tu equipo y crea una estructura de sitio web del proyecto anterior. Comprueba su funcionamiento. Los documentos del proyectos están en la plataforma.
- Crea un nuevo proyecto en tu sitio web que incluya un fichero *index.html* que incluya un formulario con tres campos: nombre, apellidos y dni/nif. Valida el dni o nif usando un script de javascript que se encuetre en una carpeta denominada *script*

- Crea un fichero denominado telefono.html que solicite un número de teléfono y realiza una validación con *HTML5*. Un número de teléfono es válido si y solo si, empieza por 6, 7, 8 o 9 y tiene 9 dígitos.

1.9. Creando servidor web con node

```
var http = require("http");
var server = http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/html"});
  response.write("<!DOCTYPE 'html'>");
  response.write("<html>");
  response.write("<head>");
  response.write("<title>Hello World Page</title>");
  response.write("</head>");
  response.write("<body>");
  response.write("Hello World!");
  response.write("</body>");
  response.write("</html>");
  response.end();
});

server.listen(4000);
console.log("Server is listening");
```

- La primera línea carga el módulo *HTTP*
- *http* es un objeto que tiene el método *createServer* que usa como argumentos una *callback* que incluye la petición (*request*) y la respuesta response. El uso de la *callback* es crear un servicio **no bloqueante y asíncrono acorde con la filosofía de nodejs.**
- Tanto *request* como *response* son objetos, por lo que disponemos de métodos para tratar la petición y la respuesta.
- En el caso de la respuesta se ha usado el método *writeHead* para especificar la respuesta, en este caso el código de estado y el tipo de respuesta. Aunque por defecto hay otras características incluidas en la respuesta.
- Y otro método *write* para especificar el cuerpo de la respuesta. Se usa una codificación UTF-8 por defecto.
- Y por último el método *end* para indicar la finalicación de la respuesta.
- Luego tenemos el método *listen* que especifica el puerto.

1.9.1. Sirviendo páginas estática

Si incluimos como primera línea tras el callback que gestiona las peticiones web, la siguiente sentencia:

```
console.log(request.url);
```

Nos mostrará el *path* de la petición y partir de aquí podemos decidir que contenido servir.
Ya podemos servir contenido estático del sitio. Supongamos la estructura de un sitio web como pueda ser:

Un servidor que atienda a dichas peticiones podría ser:

```
var fs = require('fs');
var http = require("http");
var path = require('path');
__dirname+='/proyecto2/'; //establecemos el directorio de trabajo
var server = http.createServer(function(request, response) {
    //petición de la página índice
    if (request.url === '/' || request.url === '/index.html'){
      fs.readFile(__dirname+'/index.html',function(err,data){
        if (err) throw err;
        response.write(data);
        response.end();
      });
    }
    //petición del favicon.ico
    else if (request.url === '/favicon.ico' || request.url === '/index.html'){
      fs.readFile(__dirname+'/favicon.ico',function(err,data){
        if (err) throw err;
        response.write(data);
        response.end();
      });
    }
    //petición para la página índice de la carpeta directorio
    else if (request.url === '/directorio/' || request.url === '/directorio/index.html'){
      fs.readFile(__dirname+'/directorio/index.html',function(err,data){
       if (err) throw err;
       response.write(data);
       response.end();
      });
    }
    //dejamos para el final las imágenes
    else {
      var patron = /imagenes\/\w+.png$/;
      if (patron.test(request.url)){
      var imagenes = request.url.split('/');
      fs.readFile(__dirname+'/imagenes/'+imagenes[2],function(err,data){
       if (err) throw err;
       response.write(data);
       response.end();
      });
    }
    }
});
server.listen(4000);
console.log("Server is listening");
```

1.10. Ejercicio

Crea un servidor web que atienda a las peticiones del proyecto que hiciste en el ejecicio anterior.