

González Pardo Adrián

## HTTP

(Hyper Text Transfer Protocol)

Es un protocolo de comunicación que permite las transferencias de información en la World Wide Web.

Este protocolo 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.

Para esto se usan las cookies, que es información que un servidor puede almacenar en el cliente. Esto permite a las aplicaciones web instituir la nación de sesión, y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por un tiempo indeterminado.

### Pila



HTTP (Puerto 80)

TCP

IP

### Descripción

Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

- El cliente se suele llamar User Agent, este realiza una petición enviando un mensaje, con cierto formato al servidor.
- El servidor le envía un mensaje de respuesta.

### Mensajes

Los mensajes HTTP son en texto plano lo que lo hace más legible y fácil de depurar. Esto tiene inconveniente de hacer los mensajes más largos.

Estos mensajes cuentan con la estructura siguiente:

- Línea Inicial (termina con retorno de carro y un salto de línea.) con:
  - \* Para las peticiones: la acción requerida por el servidor (método de petición) seguido de la URL del recurso y la versión HTTP que soporta el cliente.

\* Para las respuestas: La versión del HTTP usado, seguido del código de respuesta (que indica que ha pasado con la petición, seguido de la URL del recurso), y la frase asociada a dicho retorno.

- Las cabeceras del mensaje que terminan con una linea en blanco. Son metadatos. Estas cabeceras le dan flexibilidad al protocolo.
- Cuerpo del mensaje. Es opcional. Su presencia depende de la linea anterior del mensaje y del tipo de recurso al que hace referencia la URL.

## Métodos de petición

HTTP define una serie predefinida de métodos de petición que pueden utilizarse, estos son los siguientes.

**GET** → Solicita una representación del recurso especificado. Los solicitudes que usan GET solo deben recuperar datos, y no deben tener otro efecto.

**HEAD** → Pide una respuesta idéntica a la que correspondería 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. (RFC 2616).

**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 blogs o subir ej. ficheros; multipart/form-data

**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 identificaba a los propios datos. Otra diferencia con POST es semántica. (REST): mientras que POST está orientado a la creación de nuevos contenidos, PUT está orientado a la actualización de los mismos.

**DELETE** → Borra recurso especificado.

**TRACE** → Este método solicita al servidor que introduzca en la respuesta todos los datos que reciba en el mensaje de petición. Se utiliza con fines de depuración y diagnóstico, ya que el cliente puede ver lo que llega al servidor y de esta forma ver todo lo que pasan al mensaje los servidores intermedios.

**OPTIONS** → Devuelve los métodos HTTP que el servidor soporta para un URL específico. Esto puede ser utilizada para comprobar la funcionalidad de un servidor web mediante petición en lugar de un recurso específico.

**CONNECT** → Se utiliza para saber si se tiene acceso a un host, no necesariamente la petición llega al servidor, este método se utiliza principalmente para saber un proxy nos da acceso a un host bajo condiciones especiales, como por ejemplo "corrientes" de datos bidireccionales encriptadas (SSL).

**PATCH** → Su función es la misma que PUT, el cual sobrescribe completamente un recurso. Se utiliza para actualizar de manera parcial una o varias partes. Esta orientada también para el uso con proxy. [RFC 5789]

**MOVE** → [RFC 2518] ← **PROPFIND**

**MOVE** → [RFC 2518] ← **PROPPATH**

**MERGE** → [RFC 3253] ← **UPDATE**

↑  
**LABEL**

### Códigos de Respuesta

Sirven para indicar que ha pasado con la petición  
Código con formato

- **1XX** Respuesta informativa  
Indica qué la petición fue recibida y se está procesando.

• **2XX** Respuesta correcta  
Indica que la petición ha sido procesada correctamente.

• **3XX** Resuestas de redirección  
Indica que el cliente necesita realizar más acciones para realizar la petición.

• **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.

• **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.

### Instalación de servidor HTTP

Debian

```
# apt install apache2 -y
```

Fedora

```
# dnf install httpd -y
```

```
# systemctl enable httpd
```

```
# firewall-cmd --add-service=http  
--permanent
```

```
# firewall-cmd --reload
```

En python es posible correr/crear un servidor HTTP

Python 2

```
python -m SimpleHTTPServer (port)
```

Python 3

```
python3 -m http.server (port)
```

Cabe destacar que estos servicios de python3 corren en el path en el que estan siendo ejecutados, pero si se desea modificar el path se puede hacer uso de la bandera -d, en python 2 no es posible modificar el path de acceso