UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

PROGRAMACIÓN II

Unidad Temática N°3:
Arquitectura Cliente Servidor

Material Teórico
1° Año – 2° Cuatrimestre

Índice

ARQUITECTURA CLIENTE SERVIDOR

Introducción

Imagine el sistema de información de una empresa como si estuviera constituido por una o más bases de datos con información de la empresa y cierto número de empleados que necesitan acceder a esos datos en forma remota. En este modelo, los datos se almacenan en poderosas computadoras denominadas **servidores**. A menudo estos servidores están alojados en una ubicación central y un administrador de sistemas se encarga de su mantenimiento. Por el contrario, los empleados tienen en sus escritorios máquinas más simples conocidas como **clientes**, con las cuales acceden a los datos remotos, por ejemplo, para incluirlos en las hojas de cálculo que desarrollan. A esta disposición se le conoce como **Modelo cliente-servidor**.

El objetivo de esta unidad es abordar los componentes y características principales de los modelos **cliente-servidor** centrándonos en una de las realizaciones más populares como lo es una aplicación web, en la cual el servidor genera páginas web basadas en su base de datos en respuesta a las solicitudes de los clientes que pueden actualizarla. Este será el sustento para las próximas unidades donde el foco se centra en aplicaciones orientadas a servicios soportados por este tipo de arquitectura.

CARACTERÍSTICAS Y COMPONENTES

La **arquitectura cliente servidor** tiene dos partes claramente diferenciadas, por un lado la parte del servidor y por otro la parte de cliente o grupo de clientes donde lo habitual es que un servidor sea una máquina bastante potente con un hardware y software específico que actúa de depósito de datos y funcione como un sistema gestor de base de datos o aplicaciones.

Esta arquitectura se aplica en diferentes modelos informáticos alrededor del mundo donde su propósito es mantener una comunicación de información entre diferentes entidades de una red mediante el uso de protocolos establecidos y el apropiado almacenaje de la misma. El más claro ejemplo de uso de una arquitectura cliente servidor es la red de **Internet** donde existen ordenadores de diferentes personas conectadas alrededor del mundo, las cuales se conectan a través de los servidores de su proveedor de Internet por ISP donde son redirigidos a los servidores de las páginas que desean visualizar y de esta manera la información de los servicios requeridos viajan a través de Internet dando respuesta a la solicitud demandada.

La principal importancia de este modelo es que permite *conectar a varios clientes a los servicios que provee un servidor* y como se sabe hoy en día, la mayoría de las aplicaciones y servicios tienen como gran necesidad que puedan ser consumidos por varios usuarios de forma simultánea.

Algunos ejemplos de la arquitectura cliente servidor son: navegar una web, usar un protocolo FTP, SSH, los juegos en red o un servidor de Correo, entre otros.

Componentes

Para entender este modelo se nombran a continuación algunos conceptos básicos que forma un modelo cliente-servidor.

Red: Una red es un conjunto de clientes, servidores y base de datos unidos de una manera física o no física en el que existen protocolos de transmisión de información establecidos.

Cliente: El concepto de cliente hace referencia a un *demandante de servicios*, este cliente puede ser un ordenador como también una aplicación de informática, la cual requiere información proveniente de la red para funcionar.

Servidor: Un servidor hace referencia *a un proveedor de servicios*, este servidor a su vez puede ser un ordenador o una aplicación informática la cual envía información a los demás agentes de la red.

Protocolo: Un protocolo es un conjunto de normas o reglas y pasos establecidos de manera clara y concreta sobre el flujo de información en una red estructurada.

Servicios: Un servicio es un conjunto de información que busca responder las necesidades de un cliente, donde esta información pueden ser mail, música, mensajes simples entre software, videos, etc.

Base de datos: Son bancos de información ordenada, categorizada y clasificada que forman parte de la red, que son sitios de almacenaje para la utilización de los servidores y también directamente de los clientes.

Clientes y servidores

Como se ha mencionado anteriormente una máquina cliente y servidor se refieren a computadoras que son usadas para diferentes propósitos.

El cliente es un computador pequeño con una estructura al igual a la que se tiene en oficinas u hogares la cual accede a un servidor o a los servicios del mismo a través de Internet o una red interna. Un claro ejemplo a este caso es la forma en que trabaja una empresa modelo con diferentes computadores donde cada uno de ellos se conectan a un servidor para poder obtener archivos de una base de datos o servicios ya sea correos electrónicos o aplicaciones.

El servidor al igual que el cliente, es una computadora pero con diferencia de que tiene una gran capacidad que le permite almacenar gran cantidad de diversos de archivos, o correr varias aplicaciones en simultaneo para así nosotros los clientes poder acceder los servicios. Los mismos pueden contener y ejecutar aplicaciones, sitios web, almacenaje de archivos, diversas bases de datos, entre muchos más.

Es importante mencionar que un cliente también puede tener una función de servidor ya que el mismo puede almacenar datos en su disco duro para luego ser usados en vez de estar conectándose al servidor continuamente por una acción que quizás sea muy sencilla.

Graficamente:

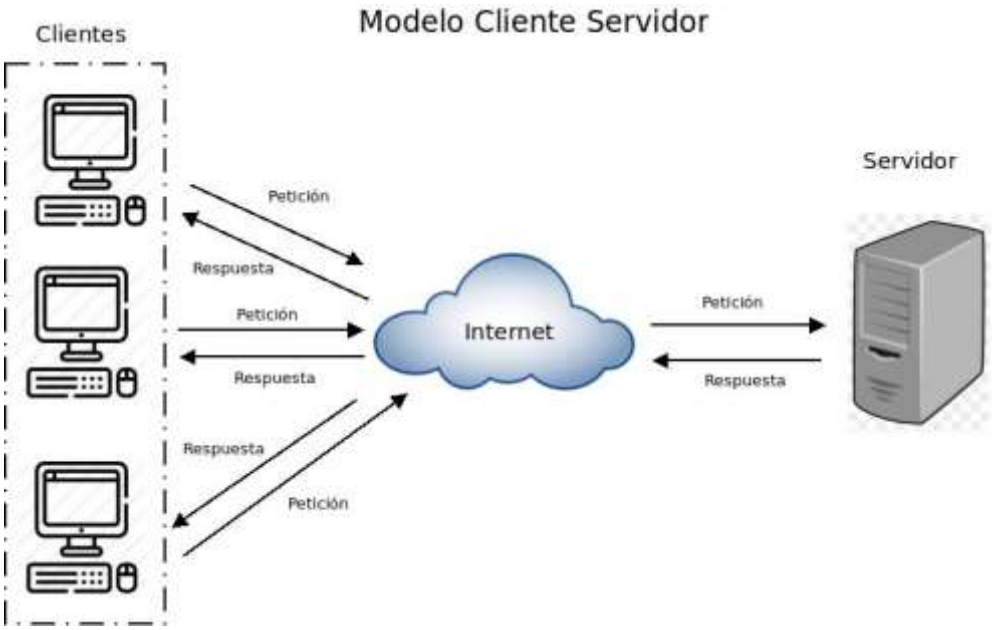

Imagen 1: Elaboración propia

Tipos de arquitecturas cliente servidor

Dentro de la arquitectura cliente servidor se pueden mencionar los siguientes tipos:

- *Arquitectura de dos capas*

Esta se utiliza para describir los sistemas cliente servidor en donde el cliente solicita recursos y el servidor responde directamente a la solicitud con sus propios recursos. Eso significa que el servidor no requiere de una aplicación extra para proporcionar parte del servicio.

- *Arquitectura de tres capas*

En la arquitectura de tres capas existe un nivel intermediario, eso significa que la arquitectura generalmente está compartida por un cliente que como hablamos más arriba es el que solicita los recursos equipado con una interfaz de usuario o mediante un navegador web.

La capa del medio es denominada software intermedio cuya tarea es proporcionar los recursos solicitados pero que requiere de otro servidor para hacerlo. La última capa es el servidor de datos que proporciona al servidor de aplicaciones los datos necesarios para poder procesar y generar el servicio que solicito el cliente en un principio.

- *Arquitectura N capas*

Si se generaliza la arquitectura anterior donde un servidor web puede usar los servicios de otros servidores para poder proporcionar su propio servicio, entonces se puede pensar en N capas necesarias para resolver un servicio específico de un cliente.

Ventajas y Desventajas

Este modelo cliente servidor tiene varias ventajas y desventajas las cuales son importantes mencionar al momento de decidir qué tipo de arquitectura es más conveniente según la solución a desarrollar:

Ventajas

- Facilita la integración entre diferentes sistemas y comparte información permitiendo por ejemplo que las máquinas ya existentes puedan ser utilizadas mediante una interfaz más amigable para el usuario. Es un modelo flexible y adaptable al servicio que se quiere implementar. Esto permite aumentar el rendimiento así como también, envolver variadas plataformas, bases de datos, redes y sistemas operativos que pueden ser de diferentes distribuidores con arquitecturas totalmente diferentes y funcionando todos al mismo tiempo.
- También es escalable y ante una gran demanda de tráfico se pueden utilizar tecnologías complementarias, por lo que cualquier organización que utilice estos sistemas adquiere ventajas competitivas.
- La estructura modular facilita la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional favoreciendo así la estabilidad de las soluciones.
- Permite el acceso simultáneo de varios clientes al mismo servidor. Esto es de gran utilidad ya que por ejemplo una empresa con varios empleados distribuidos físicamente puede trabajar simultáneamente desde su área o lugar remoto de trabajo.

Desventajas

- Se requiere habilidad para que un servidor sea reparado. Por ejemplo si un problema ocurre en la red, se requiere de alguien con un amplio conocimiento de esta para poder repararla en su totalidad para así dejar que la información y el correcto funcionamiento siga su flujo.
- Otro problema es la seguridad, el hecho que se comparte canales de información entre servidores y clientes requieren que estas pasen por procesos de validación, es decir protocolos de seguridad que pueden tener algún tipo de puerta abierta permitiendo que se generen daños físicos, amenazas o ataques de malware.
- Este modelo representa una limitación importante en cuanto a los costos económicos debido a que estos servidores son computadoras de alto nivel con un hardware y software específicos para poder dar un correcto funcionamiento a nuestras aplicaciones.

SISTEMAS BASADOS EN HTTP

HTTP (Hypertext Transfer Protocol) es un protocolo basado en el principio de cliente-servidor: las peticiones son enviadas por una entidad: **el agente** del usuario (o un proxy a petición de uno). La mayoría de las veces el cliente es un navegador Web, pero podría ser cualquier otro programa, como por ejemplo un programa-robot, que explore la Web, para adquirir datos de su estructura y contenido para uso de un buscador de Internet.

Cada petición individual se envía a un **servidor**, el cuál la gestiona y responde. Entre cada *petición* y *respuesta*, hay varios intermediarios, normalmente denominados **proxies**, los cuales realizan distintas funciones, como: *gateways* o *caches*.

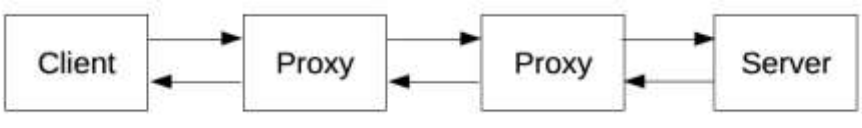

Imagen 2: Elaboración propia

En realidad, hay más elementos intermedios, entre un navegador y el servidor que gestiona su petición: hay otros tipos de dispositivos: como *routers*, *módems*. Es gracias a la arquitectura en capas de la

Web, que estos intermediarios, son transparentes al navegador y al servidor, ya que HTTP se apoya en los protocolos de red y transporte. HTTP es un protocolo de aplicación, y por tanto se apoya sobre los anteriores. Aunque para diagnosticar problemas en redes de comunicación, las capas inferiores son irrelevantes para la definición del protocolo HTTP.

HTTP es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor como se mencionó anteriormente, esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (el cliente), normalmente un navegador Web. Así, una página web completa resulta de la unión de distintos sub-documentos recibidos, como, por ejemplo: un documento que especifique el estilo de maquetación de la página web (CSS), el texto, las imágenes, vídeos, scripts, etc...

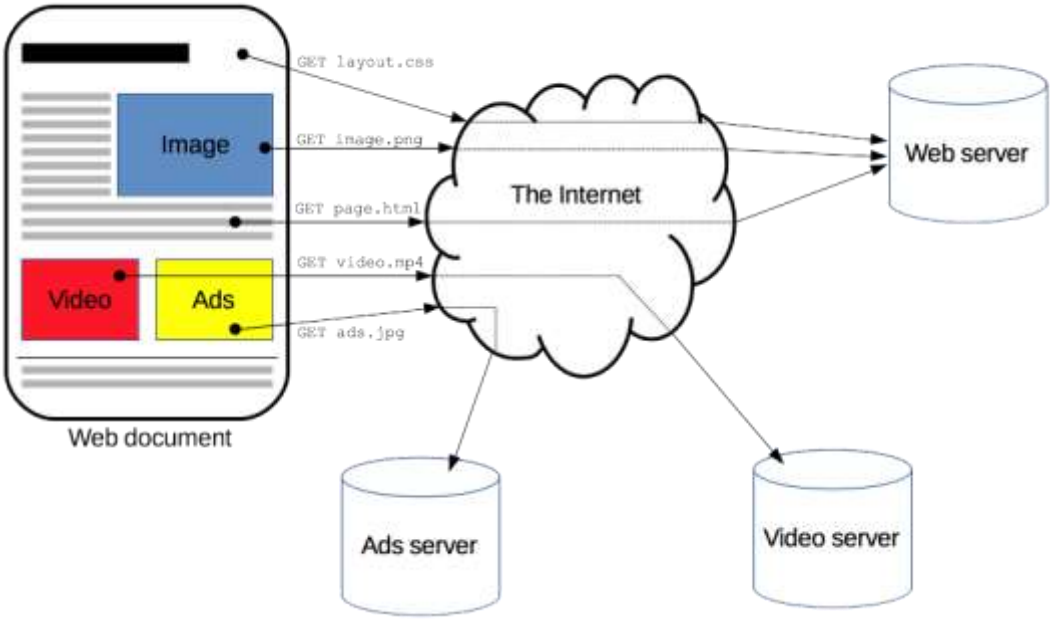

Imagen 3: Elaboración propia

Clientes y servidores se comunican intercambiando mensajes individuales (en contraposición a las comunicaciones que utilizan flujos continuos de datos). Los mensajes que envía el cliente, normalmente un navegador Web, se llaman *peticiones*, y los mensajes enviados por el servidor se llaman *respuestas*.

Cliente: el agente del usuario

El 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.

El navegador es **siempre** el que inicia una comunicación (petición), y el servidor nunca la comienza (hay algunos mecanismos que permiten esto, pero no son muy habituales).

Para poder mostrar una página Web, el navegador envía una petición de documento HTML al servidor. Entonces procesa este documento, y envía más peticiones para solicitar scripts, hojas de estilo (CSS), y otros datos que necesite (normalmente vídeos y/o imágenes). El navegador, une todos estos documentos y datos, y compone el resultado final: la página Web. Los scripts, los ejecuta también el navegador, y también pueden generar más peticiones de datos en el tiempo, y el navegador, gestionará y actualizará la página Web en consecuencia.

Una página Web, es un documento de hipertexto (HTTP), luego habrá partes del texto en la página que puedan ser enlaces (links) que pueden ser activados (normalmente al hacer click sobre ellos) para hacer una petición de una nueva página Web, permitiendo así dirigir su agente de usuario y navegar por la Web. El navegador, traduce esas direcciones en peticiones de HTTP, e interpretara y procesará las respuestas HTTP, para presentar al usuario la página Web que desea.

El servidor Web

Al otro lado del canal de comunicación, está el servidor, el cual *"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.

Un servidor no tiene que ser necesariamente un único equipo físico, aunque si que varios servidores pueden estar funcionando en un único computador. En el estándar HTTP/1.1 y Host, pueden incluso compartir la misma dirección de IP.

Proxies

Entre el cliente y el servidor, además existen distintos dispositivos que gestionan los mensajes HTTP. Dada la arquitectura en capas de la Web, la mayoría de estos dispositivos solamente gestionan estos mensajes en los niveles de protocolo inferiores: capa de transporte, capa de red o capa física, siendo así transparentes para la capa de comunicaciones de aplicación del HTTP, además esto aumenta el rendimiento de la comunicación. Aquellos dispositivos, que sí operan procesando la capa de aplicación son conocidos como proxies. Estos pueden ser transparentes, o no (modificando las peticiones que pasan por ellos), y realizan varias funciones:

- caching (la caché puede ser pública o privada, como la caché de un navegador)
- filtrado (como un anti-virus, control parental, ...)
- balanceo de carga de peticiones (para permitir a varios servidores responder a la carga total de peticiones que reciben)
- autentificación (para el control al acceso de recursos y datos)
- registro de eventos (para tener un histórico de los eventos que se producen)

Características clave del protocolo HTTP

HTTP es 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 está 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.

HTTP es 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.

HTTP es un protocolo con sesiones, pero 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. Pero, mientras HTTP ciertamente es un protocolo sin estado, el uso de HTTP cookies, si permite guardar datos con respecto a la sesión de comunicación. Usando la capacidad de ampliación del protocolo HTTP, las cookies permiten crear un contexto común para cada sesión de comunicación.

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). De los dos protocolos más comunes en Internet, TCP es fiable, mientras que UDP, no lo es. Por lo tanto HTTP, se apoya en el uso del protocolo TCP, que está orientado a conexión, aunque una conexión continua no es necesaria siempre.

FLUJO DE MENSAJES 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 requested web
page)
```

4. Cierre o reuso de la conexión para futuras peticiones.

En las versiones del protocolo HTTP/1.1 y anteriores los mensajes eran de formato texto y eran totalmente comprensibles directamente por una persona. En HTTP/2, los mensajes están estructurados en un nuevo formato binario y las tramas permiten la compresión de las cabeceras y su multiplexación. Así pues, incluso si solamente parte del mensaje original en HTTP se envía en este formato, la semántica de cada mensaje es la misma y el cliente puede formar el mensaje original en HTTP/1.1. Luego, es posible interpretar los mensajes HTTP/2 en el formato de HTTP/1.1.

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

Peticiones

Un ejemplo de petición HTTP:

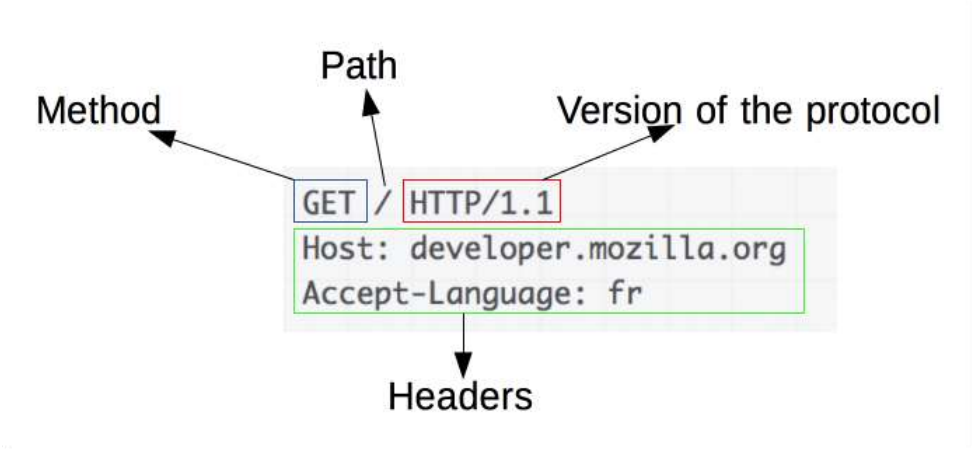

Imagen 4: Elaboración propia

Una petición de HTTP, está formado por los siguientes campos:

- Un **método HTTP**, normalmente pueden ser un verbo,ç 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

Un ejemplo de repuesta:

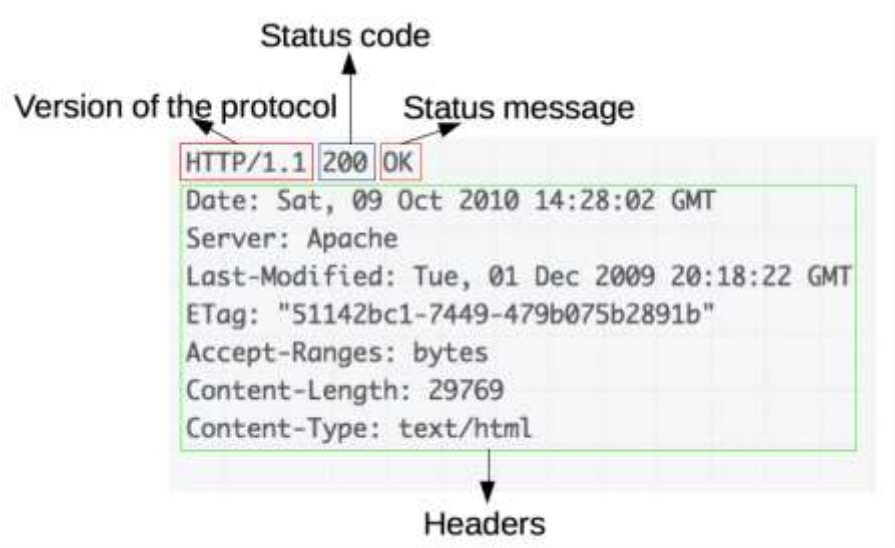

Imagen 5: Elaboración propia

Las respuestas están formadas por los siguientes 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.

Métodos de solicitud

HTTP define un conjunto de métodos de solicitud para indicar la acción que se desea realizar para un recurso determinado. Aunque también pueden ser sustantivos, estos métodos de solicitud a veces se denominan *verbos HTTP*. Aunque comparten algunas características comunes, cada uno de ellos implementa una semántica diferente. A continuación la tabla 1 enumera cada método HTTP con su correspondiente significado.

Método HTTP	Significado
GET	Solicita una representación del recurso especificado. Las solicitudes que utilizan GET solo deben recuperar datos, esto implica que una de las principales características de una petición GET es que no debe causar efectos secundarios en el servidor, no deben producir nuevos registros, ni modificar los ya existentes.
HEAD	Solicita una respuesta idéntica a la de una solicitud *GET*, pero sin el cuerpo de la respuesta.
POST	Se utiliza para enviar una entidad al recurso especificado, lo que a menudo provoca un cambio de estado o efectos secundarios en el servidor.
PUT	Reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la solicitud. Al igual que PATCH ambos se usan para modificar un recurso existente.
DELETE	Elimina un recurso especificado.
CONNECT	Convierte la solicitud en un túnel TCP/IP. Normalmente se usa para crear comunicaciones HTTPS a través de proxys HTTP sin encriptación.
OPTIONS	Se utiliza para describir las opciones de comunicación para el recurso de destino.
TRACE	Realiza una prueba de bucle de mensajes a lo largo de la ruta al recurso de destino.
PATCH	Se utiliza para aplicar modificaciones parciales a un recurso.

Tabla 1: Elaboración propia.

Códigos de Estado

Los códigos de estado de respuesta HTTP indican si se ha completado satisfactoriamente una solicitud HTTP específica. Las respuestas se agrupan en cinco clases:

- Respuestas informativas (`100`–`199`),
- Respuestas satisfactorias (`200`–`299`),
- Redirecciones (`300`–`399`),
- Errores de los clientes (`400`–`499`),

- Errores de los servidores (`500`–`599`).

La tabla 2 enumera los códigos de uso más frecuentes:

Código	Nombre	Tipo	Descripción
100	Continue	Informativa	Esta respuesta provisional indica que todo hasta ahora está bien y que el cliente debe continuar con la solicitud o ignorarla si ya está terminada.
101	Switching Protocol	Informativa	Este código se envía en respuesta a un encabezado de solicitud *Upgrade* por el cliente e indica que el servidor acepta el cambio de protocolo propuesto por el agente de usuario.
200	Ok	Satisfactoria	La solicitud ha tenido éxito. El significado de un éxito varía dependiendo del método HTTP
201	Created	Satisfactoria	La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado de ello. Ésta es típicamente la respuesta enviada después de una petición PUT.
202	Acepted	Satisfactoria	La solicitud se ha recibido, pero aún no se ha actuado. Es una petición "sin compromiso", lo que significa que no hay manera en HTTP que permite enviar una respuesta asíncrona que indique el resultado del procesamiento de la solicitud. Está pensado para los casos en que otro proceso o servidor maneja la solicitud, o para el procesamiento por lotes.
301	Moved Permanently	Redirección	Este código de respuesta significa que la URI del recurso solicitado ha sido cambiado. Probablemente una nueva URI sea devuelta en la respuesta.
308	Permanent Redirect	Redirección	Significa que el recurso ahora se encuentra permanentemente en otra URI, especificada por la respuesta de encabezado HTTP **Location**. Tiene la misma semántica que el código de respuesta HTTP 301

			Moved Permanently, con la excepción de que el agente usuario no debe cambiar el método HTTP usado: si un POST fue usado en la primera petición, otro POST debe ser usado en la segunda petición.
400	Bad request	Error de cliente	Esta respuesta significa que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.
401	Unauthorized	Error de cliente	Es necesario autenticar para obtener la respuesta solicitada. Esta es similar a 403, pero en este caso, la autenticación es posible.
403	Forbidden	Error de cliente	El cliente no posee los permisos necesarios para cierto contenido, por lo que el servidor está rechazando otorgar una respuesta apropiada.
404	Not found	Error de cliente	El servidor no pudo encontrar el contenido solicitado. Este código de respuesta es uno de los más famosos dada su alta ocurrencia en la web.
500	Server Internal Error	Error de servidor	El servidor ha encontrado una situación que no sabe cómo manejarla.
503	Service Unavailable	Error de servidor	El servidor no está listo para manejar la petición. Causas comunes puede ser que el servidor está caído por mantenimiento o está sobrecargado

Tabla 2: Elaboración propia.

BIBLIOGRAFÍA

Andrew. Tanenbaum, David Wetherall. (2012). Andrew. Tanenbaum, David Wetherall. (2011). Redes de computadoras. Quinta Edición. Pearson Educación, México.

Documentación de .NET. Recuperado de: https://docs.microsoft.com/es-es/dotnet/?view=net-5.0

Generalidades del protocolo HTTP. Recuperado de: https://developer.mozilla.org/es/docs/Web/HTTP

Modelo Cliente-Servidor. Recuperado de: https://blog.infranetworking.com/modelo-cliente-servidor/