Hypertext Transfer Protocol (http)

Integrantes: Bruno Soria y Santiago Ansa.

Curso: 6°AO

Fecha: 29/5/2023

# Introducción

El Protocolo de Transferencia de Hipertexto (HTTP) es un protocolo de solicitud/respuesta, ubicado en la capa de aplicación, utilizado para la transferencia de datos en sistemas de información hipertextuales**1**, distribuidos**2** y colaborativos**3**, como la World Wide Web.

HTTP proporciona una interfaz uniforme y flexible, ocultando los detalles de implementación y permitiendo la comunicación entre clientes y servidores en la web. El protocolo define la sintaxis de la comunicación, las reglas de enrutamiento, la estructura de los mensajes y otros aspectos necesarios para su funcionamiento.

El intercambio de información se basa en un modelo de solicitud-respuesta, donde los clientes envían solicitudes especificando la acción deseada y los servidores responden con los datos solicitados o información sobre el resultado de la operación. Las solicitudes pueden incluir diversos métodos, mientras que las respuestas contienen códigos de estado para indicar el éxito o el error de la solicitud.

Además de la transferencia de archivos, HTTP también se utiliza para el control de caché, la autenticación de usuarios y la negociación de contenido. Es un protocolo sin estado y juega un papel fundamental en la navegación y la interacción en la web.

**Notas**

1. *Los sistemas hipertextuales están basados en un enfoque en cual el usuario tiene la posibilidad de crear, agregar, enlazar y compartir información de fuentes diversas, proveyendo la posibilidad de acceder a documentos de manera no secuencial a diferencia de sistemas de información más tradicionales en los cuales el acceso es naturalmente secuencial*
2. *Un sistema distribuido es un sistema en el que los componentes hardware o software se encuentran en computadores unidos mediante una red.*
3. *Hablamos de sistemas colaborativos, o groupware, cuando nos referimos a un grupo de programas y métodos que permiten integrar en un solo proyecto el trabajo de dos o más personas conectadas entre sí por medio de Internet o intranet.*

# Tipos y versiones de HTTP:

HTTP/0.9:

Lanzada en 1991, fue la primera versión de HTTP.

Era extremadamente simple y solo permitía la recuperación de documentos HTML.

Solo se admitía el método GET para recuperar los documentos.

No incluía encabezados HTTP ni soporte para otros tipos de contenido o recursos más allá de HTML.

No existía la noción de versiones en las solicitudes o respuestas.

HTTP/1.0:

Lanzada en 1996, fue una mejora significativa sobre HTTP/0.9.

Se agregaron nuevos métodos de solicitud, como POST, para enviar datos al servidor.

Se introdujeron encabezados HTTP más completos, lo que permitió enviar información adicional en las solicitudes y respuestas.

Se agregó la capacidad de transmitir contenido más allá de HTML, como imágenes y archivos multimedia.

Cada solicitud o respuesta incluía una línea de estado que indicaba si la operación fue exitosa o no.

HTTP/1.1:

Introducida en 1997, fue una actualización importante de HTTP/1.0.

Incluyó mejoras en varios aspectos:

Persistencia de la conexión: Permite mantener la conexión abierta para realizar múltiples solicitudes y respuestas.

Compresión de datos: Permite comprimir los datos para reducir el tamaño de las solicitudes y respuestas.

Almacenamiento en caché: Permite almacenar en caché los recursos para una entrega más rápida en solicitudes posteriores.

Soporte para encabezados de host virtual: Permite alojar múltiples sitios web en un mismo servidor utilizando el mismo puerto.

HTTP/1.1 sigue siendo ampliamente utilizado en la actualidad y es la versión por defecto en la mayoría de los servidores y navegadores.

HTTP/2:

Publicada en 2015, esta versión introdujo mejoras significativas en rendimiento y eficiencia en comparación con HTTP/1.1.

Introdujo la multiplexación de solicitudes y respuestas, lo que permite enviar múltiples solicitudes y recibir múltiples respuestas en paralelo a través de una sola conexión TCP.

La compresión de encabezados reduce el tamaño de los encabezados, lo que ahorra ancho de banda.

La transferencia de datos en formato binario en lugar de texto facilita el procesamiento más eficiente.

Permite a los servidores enviar recursos no solicitados al cliente antes de que sean solicitados, mejorando la velocidad de carga de la página.

HTTP/2 sigue utilizando TCP como protocolo de transporte.

HTTP/3:

Es el sucesor de HTTP/2 y fue publicado en 2022.

En lugar de utilizar TCP como protocolo de transporte, HTTP/3 se basa en QUIC (Quick UDP Internet Connections), un protocolo de transporte de próxima generación basado en UDP.

HTTP/3 ofrece una menor latencia y mejor rendimiento en comparación con HTTP/2, lo que resulta en tiempos de carga más rápidos para las páginas web.

La multiplexación de solicitudes y respuestas es aún más eficiente, ya que se basa en QUIC, que está diseñado específicamente para ser rápido y confiable en entornos de red desafiantes.

HTTP/3 también ofrece una mejor tolerancia a la pérdida de paquetes y una recuperación más rápida.

Cabe destacar que la versión de HTTP utilizada en una aplicación web dependerá de la compatibilidad del servidor y el cliente, así como de las necesidades específicas del proyecto.

# Identificación de Recursos y Encabezados HTTP en el Protocolo HTTP

En el protocolo HTTP (Hypertext Transfer Protocol), la identificación de recursos se lleva a cabo mediante la utilización de la URL (Uniform Resource Locator) y los encabezados HTTP.

Una URL es una cadena de caracteres que sigue una estructura específica para identificar de manera única un recurso en la web. Está compuesta por varios componentes, incluyendo el esquema (como "http://" o "https://"), el nombre de dominio o dirección IP del servidor, el puerto (opcional), la ruta del recurso y posibles parámetros adicionales. Por ejemplo, en la URL "https://www.ejemplo.com/pagina.html", "www.ejemplo.com" es el nombre de dominio y "/pagina.html" es la ruta del recurso.

Los encabezados HTTP son metadatos que se envían junto con las solicitudes y respuestas HTTP. Estos encabezados proporcionan información adicional sobre la solicitud o respuesta, como el tipo de contenido, la codificación, las cookies, la autenticación y otros detalles relevantes. Los encabezados permiten que el cliente y el servidor se comuniquen e intercambien información adicional necesaria para procesar adecuadamente la solicitud o respuesta.

Además de la identificación de recursos, los encabezados HTTP también se utilizan para identificar y negociar las versiones y capacidades del protocolo entre el cliente y el servidor. Por ejemplo, el encabezado "Host" se utiliza para especificar el nombre de dominio o dirección IP del servidor al que se está realizando la solicitud.

En resumen, la identificación de recursos en el protocolo HTTP se logra mediante el uso de la URL, que proporciona una dirección única para acceder a un recurso en la web. Los encabezados HTTP complementan esta identificación al proporcionar información adicional sobre la solicitud o respuesta, incluyendo la versión del protocolo y las capacidades del servidor. Estos componentes son fundamentales para el correcto funcionamiento y la comunicación efectiva entre el cliente y el servidor en una transacción HTTP.

# Métodos

Los métodos de solicitud (request) son tokens que indican el propósito de la solicitud realizada por el cliente y lo que se espera como resultado exitoso. Estos métodos proporcionan la semántica de la solicitud y determinan cómo se debe tratar la solicitud por parte del servidor.

Un token es una secuencia de caracteres que se utiliza para representar un valor específico en un mensaje HTTP.

Los métodos de solicitud en HTTP pueden ser especializados aún más por los valores de ciertos campos de encabezado presentes en la solicitud. Por ejemplo, se pueden enviar campos de encabezado condicionales para que la acción solicitada dependa del estado actual del recurso objetivo.

## Propiedades comunes de los métodos

**Métodos seguros:**

Los métodos seguros en HTTP son aquellos métodos de solicitud que se consideran de solo lectura y no causan cambios en el estado del servidor. Estos métodos no esperan modificaciones en el recurso objetivo y no se espera que causen daño o una carga inusual en el servidor. Es importante tener en cuenta que si bien los métodos seguros no deben tener efectos perjudiciales, las implementaciones pueden incluir comportamientos adicionales que no son completamente de solo lectura o que pueden tener efectos secundarios. Sin embargo, el cliente no puede ser responsabilizado por estos comportamientos adicionales, ya que no los solicitó.

Los métodos seguros definidos en la especificación de HTTP son GET, HEAD, OPTIONS y TRACE. Distinguir entre métodos seguros y no seguros permite que los procesos de recuperación automatizados y la optimización de la memoria caché funcionen sin causar daño. Los agentes de usuario deben presentar acciones potenciales al usuario, de manera que este sea consciente de una acción insegura antes de solicitarla.

Cuando se construye un recurso de manera que los parámetros dentro de la URI de solicitud efectiva tienen el efecto de seleccionar una acción, es responsabilidad del propietario del recurso asegurarse de que la acción sea coherente con la semántica del método de solicitud. Por ejemplo, es común que el software de edición de contenido basado en la web utilice acciones dentro de parámetros de consulta, como "page?do=delete" (página?hacer=borrar). Si el propósito de dicho recurso es realizar una acción insegura, entonces el propietario del recurso DEBE deshabilitar o no permitir esa acción cuando se acceda mediante un método de solicitud seguro. No hacerlo resultará en efectos secundarios desafortunados cuando los procesos automatizados realicen un GET en cada referencia de URI con el fin de mantener los enlaces, realizar pre-capturas, construir un índice de búsqueda, etc.

**Métodos idempotentes:**

Los métodos idempotentes en HTTP son aquellos que se pueden repetir varias veces sin cambiar el resultado más allá del primer intento. Esto significa que si se envía el mismo método con los mismos parámetros varias veces, el resultado final será el mismo que si se hubiera enviado solo una vez. Los métodos PUT, DELETE y los métodos seguros son considerados idempotentes según la especificación de HTTP.

La propiedad idempotente se aplica únicamente a lo que ha sido solicitado por el usuario. El servidor tiene libertad para registrar cada solicitud por separado, mantener un historial de control de revisiones u aplicar otros efectos secundarios no idempotentes para cada solicitud idempotente.

Los métodos idempotentes son distinguibles porque la solicitud puede repetirse automáticamente en caso de que ocurra una falla de comunicación antes de que el cliente pueda leer la respuesta del servidor. Por ejemplo, si un cliente envía una solicitud PUT y la conexión subyacente se cierra antes de recibir alguna respuesta, el cliente puede establecer una nueva conexión y reintentar la solicitud idempotente. El cliente sabe que repetir la solicitud tendrá el mismo efecto previsto, incluso si la solicitud original tuvo éxito, aunque la respuesta pueda ser diferente.

**Métodos almacenables en caché:**

Una respuesta almacenable en caché es una respuesta HTTP que se puede almacenar en caché para más tarde recuperarla y usarla más. Los métodos seguros como GET, HEAD y POST se consideran almacenables en caché según esta especificación, aunque, la mayoría de las implementaciones de caché solo admiten GET y HEAD.

**Definición de los métodos:**

1. GET: El método GET se utiliza para recuperar información actual de un recurso específico en el servidor. La solicitud GET solicita que el servidor envíe el recurso identificado por la URL al cliente. Es el método más comúnmente utilizado y se utiliza para acceder a páginas web, imágenes, archivos y otros recursos.
2. HEAD: El método HEAD es similar al método GET, pero solicita solo los encabezados de respuesta del recurso y el estado actual, sin recuperar el contenido real. Se utiliza para obtener información sobre el recurso, como su tamaño, fecha de modificación o encabezados de caché, sin transferir los datos completos.
3. POST: El método POST se utiliza para enviar datos desde el cliente al servidor para su procesamiento. La solicitud POST envía los datos en el cuerpo de la solicitud, que puede contener información como formularios HTML, datos de usuario, carga de archivos, etc. El servidor procesa los datos y puede devolver una respuesta.
4. PUT: El método PUT se utiliza para actualizar o reemplazar completamente un recurso existente en el servidor con los datos proporcionados en la solicitud. Si el recurso no existe, se puede crear uno nuevo. El contenido enviado en la solicitud PUT reemplaza completamente el contenido existente del recurso.
5. DELETE: El método DELETE se utiliza para eliminar un recurso específico en el servidor. La solicitud DELETE solicita que el servidor elimine el recurso identificado por la URL. Después de la eliminación exitosa, el servidor puede responder con un código de estado 204 (No Content) o 200 (OK) si se proporciona información adicional.
6. CONNECT: El método CONNECT es utilizado en solicitudes a un proxy para establecer un túnel hacia el servidor de origen identificado por el objetivo de la solicitud (request-target). Una vez establecido el túnel con éxito, el comportamiento del proxy se limita a reenviar los paquetes en ambas direcciones hasta que el túnel se cierre. Los túneles se utilizan comúnmente para crear una conexión virtual de extremo a extremo a través de uno o más proxies, que luego puede ser asegurada mediante TLS (Transport Layer Security).
7. OPTIONS: El método OPTIONS se utiliza para solicitar información sobre las opciones y capacidades de comunicación permitidas por el servidor para un recurso específico. El servidor responde proporcionando una lista de métodos de solicitud y otros detalles de configuración permitidos.
8. TRACE: El método TRACE en el protocolo HTTP se utiliza para solicitar una confirmación en bucle de nivel de aplicación del mensaje de solicitud. El destinatario final de la solicitud debe reflejar el mensaje recibido, excluyendo algunos campos, devolviéndole al cliente como cuerpo de respuesta 200 (OK) con un tipo de contenido "message/http". El destinatario final puede ser el servidor de origen o el primer servidor que recibe un valor de Max-Forwards de cero en la solicitud.

# Explorando los Componentes y Aplicaciones del Protocolo HTTP

**Códigos de estado HTTP:** Una mirada detallada a los resultados de las solicitudes

HTTP utiliza códigos de estado para indicar el resultado de una solicitud. Estos códigos se envían en las respuestas del servidor y proporcionan información sobre el éxito, redirección, errores del cliente o del servidor, entre otros. Algunos ejemplos comunes son el código 200 (OK) para una solicitud exitosa, 404 (Not Found) para un recurso no encontrado y 500 (Internal Server Error) para un error interno del servidor. Estos códigos de estado son cruciales para comprender el estado y la respuesta de una solicitud en el entorno HTTP.

**Gestión de sesiones y cookies:** Añadiendo persistencia a la comunicación HTTP

HTTP es un protocolo sin estado, lo que significa que no mantiene información sobre las solicitudes previas. Sin embargo, para permitir el seguimiento de la interacción entre el cliente y el servidor, se utilizan las sesiones y las cookies. Las cookies son pequeños archivos de texto que se almacenan en el lado del cliente y se envían con cada solicitud HTTP. Permiten que el servidor almacene y recupere información sobre el estado de la sesión del usuario, brindando una forma de mantener cierta persistencia en la comunicación HTTP.

**Seguridad en HTTP:** Protegiendo la confidencialidad e integridad de los datos

HTTP es un protocolo que transmite datos en texto plano, lo que implica que la información intercambiada puede ser interceptada y leída por terceros. Para mejorar la seguridad en la comunicación, se utiliza HTTPS (HTTP Secure). HTTPS utiliza una capa adicional de cifrado SSL/TLS para proteger los datos transmitidos, garantizando así la confidencialidad e integridad de la información. Con HTTPS, la comunicación se vuelve más segura y se evita que los datos sean comprometidos o alterados durante la transmisión.

**REST y API:** Potenciando la comunicación entre sistemas distribuidos

HTTP se utiliza ampliamente en la implementación de arquitecturas RESTful y en la construcción de API (Application Programming Interface). REST (Representational State Transfer) es un estilo arquitectónico que utiliza los métodos de HTTP (GET, POST, PUT, DELETE) para permitir la comunicación entre sistemas distribuidos. Las API basadas en HTTP proporcionan un conjunto de endpoints que permiten el acceso y la manipulación de recursos a través de HTTP. Esto facilita la integración y la interacción entre diferentes sistemas y aplicaciones, permitiendo el intercambio de datos y funcionalidades de manera eficiente y segura.

Explorando los componentes y aplicaciones del protocolo HTTP, desde los códigos de estado que revelan el resultado de una solicitud, hasta la gestión de sesiones y cookies que brindan persistencia en la comunicación. Además, la seguridad en HTTP se fortalece mediante el uso de HTTPS, y la implementación de arquitecturas RESTful y la construcción de API basadas en HTTP permiten la comunicación efectiva entre sistemas distribuidos. Estos elementos combinados hacen de HTTP un protocolo esencial para la comunicación en la web y en entornos de desarrollo de aplicaciones.

**Fuentes:**

<https://www.youtube.com/watch?v=t3jrsuD9Djs>

<https://www.rfc-editor.org/rfc/rfc2616.html#section-1>

<https://datatracker.ietf.org/doc/html/rfc7231>

[https://en.m.wikipedia.org/wiki/HTTP#](https://en.m.wikipedia.org/wiki/HTTP)

<https://developer.mozilla.org/es/docs/Web/HTTP>

[https://j2logo.com/python/python-requests-peticiones-http/#:~:text=C%C3%B3mo%20hacer%20una%20petici%C3%B3n%20GET%20en%20Python%20con%20requests,-Una%20de%20las&text=Para%20ello%2C%20simplemente%20tienes%20que,la%20que%20hacer%20la%20petici%C3%B3n.&text=La%20funci%C3%B3n%20devuelve%20un%20objeto,la%20informaci%C3%B3n%20de%20la%20respuesta](https://j2logo.com/python/python-requests-peticiones-http/#:~:text=C%C3%B3mo%20hacer%20una%20petici%C3%B3n%20GET%20en%20Python%20con%20requests,-Una%20de%20las&text=Para%20ello%2C%20simplemente%20tienes%20que,la%20que%20hacer%20la%20petici%C3%B3n.&text=La%20funci%C3%B3n%20devuelve%20un%20objeto,la%20informaci%C3%).