PRACTICA 2 DE REDES

1)

La capa de aplicación es la mas alta dentro de los modelos de referencia (osi o tcp/ip) y su función principal es proveer servicios directamente a los usuarios y a las aplicaciones para que puedan comunicarse a través de la red

Funciones principales:

* + Interfaz de usuario: permite que programas como navegadores web, clientes de correo electrónico o apps de mensajería usen la red
  + Servicios de red comunes: Proporciona servicios como transferencia de archivos (FTP), correo electrónico (SMTP, IMAP), web (HTTPS/HTTP), DNS, etc.
  + Gestión de recursos compartidos: acceso a archivos, impresoras o bases de datos en red
  + Presentación de datos al usuario: convierte la información en un formato que las aplicaciones puedan entender y mostrar

En resumen, la capa de aplicación hace posible que el usuario y los programas se comuniquen a través de la red usando protocolos específicos.

2)

Si dos procesos deben comunicarse y se encuentran en diferentes maquinas la forma de lograrlo es a través de la red utilizando protocolos de comunicación que permitan el intercambio de mensajes entre ellas. En este caso es necesario establecer una conexión mediante la capa de transporte que proporcione servicios como TPC o UDP además utilizar direcciones IP y números de puerto que identifiquen de manera única a cada proceso en la red. De esta forma los mensajes que genera un proceso en una maquina pueden viajar a través de la infraestructura de la red y ser entregados al proceso destino en la otra máquina.

En cambio, si los procesos están en la misma maquina no es necesario recurrir a la red y existen alternativas mas directas para la comunicación. Entre ellas se puede usar comunicación entre procesos mediante memoria compartida que permite a ambos acceder a un mismo espacio de direcciones o bien utilizar mecanismos de paso de mensajes como pipes, colas de mensajes o sockets locales que no salen de la máquina. Estas opciones suelen ser más rápidas y eficientes porque evitan el tráfico de red y aprovechan los mecanismos que ofrece el sistema operativo para la sincronización y el intercambio de datos entre procesos residentes en un mismo sistema.

3)

El modelo cliente servidor es una forma de organización de la comunicación en la que una de las partes denominada cliente realiza solicitudes de servicios o recursos mientras que la otra parte denominada servidor se encarga de atender esas solicitudes y proveer la respuesta adecuada. El servidor normalmente centraliza los recursos o la lógica principal del sistema mientras que el cliente actúa como la interfaz que permite al usuario interactuar con dichos recursos.

Un ejemplo claro y sencillo podría ser un restaurante donde un cliente (cliente) pide un plato de comida (mensaje con información) al mozo quien se lo trasmite a la cocina (servidor) la cual preparara el plato y posteriormente se lo entregara al cliente para que lo reciba.

Un ejemplo claro en la informática es cualquier pagina web que consultemos constantemente donde solicitemos información que el servidor deberá traer y mostrar en la pagina para que podamos accederla.

4)

Un **agente de usuario** o **user agent** es la entidad de software que actúa como un intermediario entre el usuario final y los servicios de red permitiéndole acceder, enviar o recibir información de manera comprensible. Su función principal es traducir las acciones del usuario en solicitudes de comunicación hacia la red y a su vez transformar las respuestas recibidas en una forma que el usuario pueda interpretar. De esta manera el agente de usuario facilita la interacción sin que el usuario deba preocuparse por los detalles técnicos del protocolo o la transmisión de datos.

Un ejemplo típico es un navegador web que interpreta paginas HTML y recursos asociados o un cliente de correo electrónico que envía y recibe mensajes mediante protocolos como SMTP, POP o IMAP. En todos los casos el agente de usuario cumple la tarea de representar al usuario frente al sistema de comunicación y gestionar la interacción de manera transparente.

5)

HTML y HTTP son conceptos distintos, aunque complementarios dentro del funcionamiento de la web. HTML es un lenguaje de marcado que se utiliza para estructurar y presentar la información de un documento de manera que un navegador pueda interpretarla y mostrarla al usuario con texto, imágenes, enlaces o formularios. Es decir, HTML define la forma en que se organiza y representa el contenido de una pagina web. En cambio, HTTP es un protocolo de comunicación que especifica como se envían y reciben los mensajes entre un cliente y un servidor en la red. Su objetivo es establecer las reglas de transferencia de documentos y como el servidor responde entregándola.

6)

En HTTP la información que permite distinguir si un mensaje pertenece a un requerimiento o a una respuesta esta en la primera línea del mensaje. En el caso de un requerimiento se utiliza una línea de petición compuesta por el método HTTP que puede ser por ejemplo GET o POST, seguido de la ruta del recurso solicitado y de la versión del protocolo. En cambio, en una respuesta se emplea una línea de estado formada por la versión HTTP, un código numérico que indica el resultado de la petición y una frase de estado explicativa. Además de esta línea inicial los mensajes incluyen cabeceras que sirven para aportar información adicional como el tipo de contenido que se esta enviando o aceptando, la fecha, los parámetros de conexión, la identificación del cliente o servidor y otras directivas que facilitan la interpretación y gestión del mensaje.

El formato de las cabeceras es de tipo par clave valor, cada cabecera ocupa una línea y sigue la estructura Nombre Cabecera: valor. Existen cabeceras de solicitud, de respuesta y también generales que pueden aparecer en ambos tipos de mensaje. Tras las cabeceras se inserta una línea en blanco que marca el final de esta sección y eventualmente puede continuar el cuerpo del mensaje con contenido propiamente dicho.  
  
 Para enviar un requerimiento HTTP 1.1 con curl/7.74.0 al recurso /index.html de [www.misitio.com](http://www.misitio.com) la información que debe incluirse en los encabezados comprende la línea de petición con el método GET, la versión del protocolo y la ruta del recurso, además del encabezado Host que es obligatorio en HTTP 1.1 para identificar el servidor al que se dirige la solicitud y el encabezado USER AGENT que informa sobre el cliente que realiza la petición. El requerimiento quedaría de la siguiente forma:

GET /index.html HTTP/1.1

Host: www.misitio.com

User-Agent: curl/7.74.0

Accept: \*/\*

Con ello el servidor recibirá un mensaje correctamente formado y sabría que el cliente solicita el recurso /index.html indicando además que acepta cualquier tipo de contenido como respuesta.

7)

El comando curl es una herramienta de línea de comandos que permite realizar transferencia de datos con un servidor a través de distintos protocolos siendo el mas habitual HTTP. Su función principal es simular el comportamiento de un cliente enviando solicitudes y recibiendo respuestas lo que lo hace muy útil para aprobar servicios web, verificar encabezados, descargar recursos o automatizar interacciones con servidores.

**-I**: muestra solo las cabeceras de la respuesta.

**-H**: agrega cabeceras personalizadas al pedido.

**-X**: define el método HTTP (GET, POST, etc.).

**-S**: modo silencioso, oculta mensajes de progreso o error.

8)

a)

Si ejecuto el comando curl con la url del sitio de redes el comando ejecuta un requerimiento HTTP de tipo GET hacia el servidor que aloja el sitio. El servidor responde con un mensaje HTTP que incluye las cabeceras y el cuerpo, que en este caso es el archivo HTML del sitio. Si redirigimos la salida (curl [www.redes.unlp.edu.ar](http://www.redes.unlp.edu.ar) > pagina.html) del comando esto creará un archivo pagina.html con el contenido HTML que el navegador podrá interpretar y mostrar como una pagina web normal.

b)

El HTML el atributo href indica la ubicación de un recurso que le navegador debe cargar. En el caso del tag <link>, que normalmente se usa para vincular hojas de estilo CSS, href especifica la ruta al archivo CSS que el navegador debe descargar y aplicar al documento. Por ejemplo, <link rel="stylesheet" href="estilos.css"> le dice al navegador que la hoja de estilos se encuentra en el archivo estilos.css y eso se usara para dar formato a la página, en el caso de <img> aunque el atributo equivalente más usado es src, si se utilizara un href en un contexto valido induraría la ubicación de un recurso vinculado. Sin embargo, para <img> propiamente dicho, la ruta de la imagen se define con src, que funciona de manera similar: le indica al navegador donde encontrar el archivo de imagen que debe mostrar.

c)

No, no alcanza con un único requerimiento. Cuando un navegador carga una pagina web, primero solicita el archivo HTML principal mediante un GET, pero ese documento suele contener referencias a otros recursos como imágenes, hojas de estilos, scripts o fuentes. Cada uno de estos recursos requiere un requerimiento HTTP adicional para ser descargado. Por eso, aunque el HTML inicial llegue con un solo GET, el navegador genera múltiples solicitudes paralelas o secuenciales para obtener todos los elementos necesarios y mostrar la página completa como está diseñada.

d)

Para una página que incluye dos archivos CSS, dos scripts JavaScript y tres imágenes, se necesitarían ocho requerimientos HTTP en total: uno por cada recurso referenciado (HTML + 2 JS + 2 CSS + 3 img = 8 )

En un navegador al cargar la página todos estos requerimientos se generan automáticamente, muchas veces en paralelo, para mostrar la pagina completa con todos sus elementos y estilos en cambio si se utiliza el comando curl como antes sin opciones adicionales solo se realiza un único GET HTML los recursos referenciados no se descargan automáticamente por lo que la página guarda con curl no se muestra completa.

9)

Al ejecutar -v -s www.redes.unlp.edu.ar > /dev/null se realiza un GET hacia el servidor y la opción -v muestra información detallada sobre la conexión y los mensajes HTTP, mientras que -s silencia la barra de progreso. La redirección a /dev/null descarga el cuerpo de la respuesta, por lo que solo se ve la información de depuración y las cabeceras enviadas y recibidas

Ahora curl -I -v -s [www.redes.unlp.edu.ar](http://www.redes.unlp.edu.ar) realiza una petición HEAD que solicita únicamente las cabeceras de la respuesta sin descargar el cuerpo del recurso por eso no es necesario redirigir la salida, el contenido de la página no se recibe, solo se muestran las cabeceras

Si se quita la redireccion a /dev/null el primer comando, el HTML completo del sitio se imprimiría junto con la información de depuración, lo que puede resultar muy extenso en la terminal

En cuanto a la cantidad de cabeceras, el requerimiento HTTP suelen viajar varias cabeceras como host, user-Agent, accept y otras opcionales según la configuración de curl. En la respuesta, el servidor devuelve las cabeceras estándar como DATE, content-type, content-length, server y posibles cabeceras adicionales de control de seguridad. Exactamente cuantas cabeceras dependen de la implementación del servidor, pero en general hay entren 5 y 15 en cada mensaje

10)

La cabecera date indica la fecha y hora en que el servidor genero la respuesta sirve para que el cliente conozca el momento exacto en que se creo el mensaje, lo que es útil para la cache, sincronización de contenido y control de ver5siones ya que permite comparar la antigüedad de los recursos recibidos o decidir si se deben solicitar actualizaciones al servidor

11)

En HTTP 1.0 el protocolo no soportaba conexiones persistentes por defecto, el cliente sabe que recibió todo el objeto porque el servidor cierra la conexión TCP una vez enviado el recurso completo es decir el fin del flujo de datos lo marca el fin de la conexión, esto obliga a abrir una nueva conexión para cada recurso lo que resulta ineficiente

En HTTP 1.1 por defecto se introducen las conexiones persistentes como ya no se puede depender del cierre de la conexión, el cliente necesita otra forma de saber cuándo termino un objeto  
existen 2 mecanismos, el primero es la cabecera el servidor indica explícitamente el tamaño del objeto en bytes. El cliente sabe que cuando recibió esa cantidad exacta, termino la respuesta. El segundo método es trasnfer-encoding: el objeto se envía en fragmentos (chunks) cada uno con su tamaño indicado y un chunk de tamaño 0 que indica el final.

12)

Los códigos de retorno de un servidor web también llamados códigos de estado HTTP se utilizan para indicar el resultado de una petición realizada por un cliente. Estos códigos se agrupan en distintas categorías que permiten interpretar rápidamente el estado de la respuesta. Los q comienzan con 2XX indican éxito, lo que significa que la petición fue recibida comprendida y procesada correctamente, el más común es el 200 OK que indica que todo está correcto.

Luego la categoría de 3XX se utiliza para redirecciones es decir cuando el recurso solicitado no se encuentra en la ubicación original, sino que debe ser buscado en otra dirección indicada por el servidor. Un ejemplo es el 301 que corresponde a una redirección permanente, mientras que el 302 indica una redirección temporal.

Por otro lado, los 4XX corresponden a errores del cliente, lo que implica que la solicitud contiene un error, por ejemplo, el 400 señala una solicitud incorrecta, el 403 deniega el acceso al recurso, el 404 indica falta de autorización.

Por último la categoría de los 5XX que indica errores del servidor por ejemplo el 502 corresponde al Gateway defectuoso, el 503 señala que el servicio no está disponible

Cada código no solo se comunica si la petición tuvo éxito o no, sino que además detalla la causa y el tipo de situación que produjo la interacción entre cliente y servidor.

13)

Al realizar un requerimiento con el método HEAD utilizando curl -I www.redes.unlp.edu.ar, la primera línea de la respuesta corresponde a la línea de estado del protocolo HTTP, en la que se indica la versión utilizada, el código de estado y su mensaje textual asociado, por ejemplo, HTTP/1.1 200 OK. Esa información permite saber que protocolo esta en uso y si el acceso fue exitoso o no.

Luego aparecen las cabeceras de respuesta, que incluyen distintos campos con información sobre el recurso. El numero exacto de encabezados puede variar según la configuración del servidor, pero se observa un conjunto de cabeceras como date, server, last-modified, content-length, content-type y eventualmente connection.

En este caso se pueden contar alrededor de cinco o seis cabeceras. Entre ellas la que identifica que software está sirviendo la página es server, que típicamente devuelve algo como Apache/2.4.29 (Ubuntu) o una versión similar, lo que indica el servidor web en uso. Dado que la respuesta en la primera línea es un código 200 OK, se confirma que el acceso de la pagina fue exitoso. La cabecera last-modified indica cuando fue la ultima vez que se modifico el recurso en el servidor, proporcionando la fecha y hora exacta.

HTTP/1.1 200 OK

Date: Mon, 01 Sep 2025 16:45:22 GMT

Server: Apache/2.4.29 (Ubuntu)

Last-Modified: Tue, 20 Aug 2024 10:15:12 GMT

Content-Length: 5321

Content-Type: text/html

Connection: keep-alive

Si ahora se solicita la página con un GET condicionado a que se haya modificado después de una fecha específica, se puede utilizar el parámetro -z de curl>

curl -z "2025-09-01" [www.redes.unlp.edu.ar](http://www.redes.unlp.edu.ar)

Si se indica una fecha posterior a la efectivamente registrada en la cabecera last-modified, en el servidor devuelve un código 304 not modified, lo que significa que el recurso no fue modificado desde esa fecha y por lo tanto no se reenvía el contenido completo. Esto sirve para implementar mecanismos de cache en los clientes, ya que evita descargar de nuevo un recurso que no cambio, ahorrando ancho de banda y mejorando la eficiencia.

14)

Hecho. Primero te redirecciona a otra página que solicita autenticación. La misma se logra enviando el header “Authentication: -user:password-“, siendo el user:password en Base64. Después necesitamos ver los encabezados “-I”. Y por último redireccionarnos a otra última página con autenticación nuevamente.

16)

El comando telnet abre una conexión TCP a un host (dirección IP o dominio) y un puerto (80) específico. Se conecta sin encriptación y lo que se escribe se envía al servidor. Por eso al enviar:

GET /http/HTTP-1.1/ HTTP/1.0  
…  
se recibe la respuesta HTTP como si se lo enviara normalmente.

Se usó un GET para solicitar el recurso

17)

Los HTML son iguales, a diferencia del <form method=”POST”> que se distingue del <form method=”GET”> pese a que la acción es la misma en ambos.

El browser al mostrar los datos ingresados, en el caso del GET tiene los parámetros dentro de la URL. Mientras que si lo hacemos desde el POST, la URL tiene solo el recurso solicitado y los datos están en el body.

Ej de parcial)

Vemos la cabecera HOST/ lo cual nos indica que la versión de HTML es 1.1 o superior.  
 El método utilizado es HEAD, que es el que solicita únicamente los encabezados y no el body.  
 El recurso solicitado, por lo tanto, son las cabeceras de la ruta métodos/  
 Sí, todo funcionó correctamente. Esto lo podemos saber que la respuesta tiene el código 200.  
 Si se hubiese enviado el encabezado mencionado, al ser la misma fecha de modificación, devolvería una respuesta 304 NOT MODIFIED.