Práctica 2 - Capa de Aplicación
HTTP

1. ¿Cuál es la función de la capa de aplicación?

- La capa de aplicación en el modelo OSI (Open Systems Interconnection) es la más alta de las siete capas y en TCP/IP la mas alta de las 4. Proporciona interfaces de comunicación para las aplicaciones de usuario. Su función principal es facilitar servicios de red específicos a las aplicaciones, permitiendo que estas se comuniquen entre sí a través de la red.

Algunas de las funciones específicas de la capa de aplicación incluyen:

1. **Interfaz con el usuario**: Proporciona servicios de red a las aplicaciones y permite a los usuarios acceder a recursos de red como correos electrónicos, páginas web, transferencia de archivos, etc. (LA INTERFAZ QUE UTILIZA EL USUARIO)

2. **Transferencia de datos**: Facilita la transferencia de datos entre las aplicaciones de origen y destino, asegurando que los datos se transmitan de manera adecuada y según el protocolo correspondiente.

3. **Protocolos de aplicación**: Define los protocolos específicos que las aplicaciones utilizan para comunicarse, como HTTP para la web, SMTP para el correo electrónico, FTP para la transferencia de archivos, entre otros.

4. **Traducción de formatos**: La capa de aplicación puede realizar la traducción de formatos de datos entre diferentes sistemas operativos y aplicaciones para garantizar la interoperabilidad.

En resumen, la capa de aplicación es esencial para proporcionar servicios de red a las aplicaciones de usuario y asegurar una comunicación eficiente y fiable en toda la red.

2. Si dos procesos deben comunicarse:

a. ¿Cómo podrían hacerlo si están en diferentes máquinas?

Si dos procesos están en diferentes máquinas y necesitan comunicarse, pueden hacerlo a través de la red utilizando diferentes técnicas de comunicación. Tales como:

- **Comunicación a través de sockets:** Un socket es un extremo de comunicación bidireccional entre dos programas que se ejecutan en diferentes computadoras a través de una red. Los procesos pueden enviar y recibir datos a través de estos sockets utilizando protocolos como TCP/IP o UDP.

- **Protocolos de aplicación**: Los procesos pueden utilizar protocolos de aplicación existentes como HTTP, FTP, SMTP, etc., para comunicarse a través de la red. Por ejemplo, si necesitan transferir archivos, pueden utilizar el protocolo FTP; si necesitan enviar correos electrónicos, pueden utilizar SMTP.

- **Servicios web**: Los procesos pueden exponer sus funcionalidades a través de servicios web utilizando protocolos como HTTP y comunicarse mediante intercambio de mensajes SOAP (Simple Object Access Protocol) o REST (Representational State Transfer).

b. Y si están en la misma máquina, ¿qué alternativas existen?

Si los dos procesos están en la misma máquina, existen varias alternativas para que puedan comunicarse:

- **Comunicación a través de memoria compartida**: Los procesos pueden compartir memoria en la misma máquina para intercambiar datos de manera eficiente. Esto implica que ambos procesos tengan acceso a una región de memoria común donde puedan leer y escribir datos compartidos.

- **Pipes (tuberías)**: Los pipes son un mecanismo de comunicación entre procesos en sistemas operativos tipo UNIX. Un proceso puede escribir datos en un extremo del pipe y otro proceso puede leerlos desde el otro extremo. Los pipes pueden ser anónimos (unidireccionales) o con nombre (bidireccionales).

- **Colas de mensajes**: Las colas de mensajes son estructuras de datos que permiten a los procesos enviar y recibir mensajes de forma asíncrona. Un proceso puede colocar mensajes en una cola y otro proceso puede leerlos de la misma cola. Esto proporciona una forma flexible de comunicación entre procesos en la misma máquina.

- **Sockets locales**: Similar a la comunicación a través de sockets en redes, los procesos en la misma máquina pueden utilizar sockets locales para comunicarse. Esto implica el uso de direcciones de socket especiales que representan conexiones dentro de la misma máquina.

- **Llamadas a funciones o métodos**: Los procesos pueden comunicarse directamente llamando a funciones o métodos en el espacio de memoria del otro proceso. Esto es común en entornos de programación multi-hilo o multi-proceso donde los procesos comparten el mismo espacio de memoria.

3. Explique brevemente cómo es el modelo Cliente/Servidor. Dé un ejemplo de un sistema Cliente/Servidor en la "vida cotidiana" y un ejemplo de un sistema informático que siga el modelo Cliente/Servidor. ¿Conoce algún otro modelo de comunicación?

El modelo Cliente/Servidor es un paradigma de comunicación en el que un cliente envía solicitudes a un servidor, que a su vez procesa estas solicitudes y envía respuestas de vuelta al cliente. Es un modelo comúnmente utilizado en sistemas informáticos distribuidos, donde los clientes y servidores pueden estar en diferentes máquinas y se comunican a través de una red.

- **Ejemplo en la vida cotidiana**: Una analogía de un sistema Cliente/Servidor en la vida cotidiana sería un restaurante. En este caso, los clientes son como los clientes en un sistema informático: llegan al restaurante y hacen pedidos al camarero (el servidor). El camarero recibe los pedidos, los pasa a la cocina (donde se procesan) y luego entrega los platos terminados de vuelta a los clientes.

- **Ejemplo en un sistema informático**: Un ejemplo común de un sistema Cliente/Servidor en el ámbito informático es un servidor web. En este caso, los navegadores web actúan como clientes que envían solicitudes HTTP a un servidor web (como Apache o Nginx). El servidor web procesa estas solicitudes y envía las páginas web correspondientes de vuelta al navegador del cliente.

Otros modelos de comunicación:

- Peer-to-Peer (P2P): En este modelo, los nodos de la red pueden actuar tanto como clientes como servidores, compartiendo recursos directamente entre sí sin la necesidad de un servidor centralizado. Un ejemplo sería un sistema de intercambio de archivos P2P como BitTorrent.
- Broadcast : En este modelo, un mensaje enviado por un nodo se transmite a todos los demás nodos de la red. Es común en redes de difusión de noticias, radio o televisión.

- Arquitectura de bus: En este modelo, todos los nodos de la red están conectados a un bus de comunicación compartido, y los mensajes enviados por cualquier nodo son recibidos por todos los demás nodos en el bus. Es común en sistemas de automatización industrial y sistemas embebidos.

4. Describa la funcionalidad de la entidad genérica “Agente de usuario” o “User agent”.

Un "Agente de usuario" o "User Agent" es una entidad genérica en el contexto de las comunicaciones de red, que se refiere a cualquier programa o dispositivo que actúa en nombre de un usuario en una red. La funcionalidad de un agente de usuario puede variar dependiendo del contexto en el que se utilice, pero en general, su función principal es interactuar con otros sistemas en la red en nombre del usuario.

Aquí hay algunas funciones y características comunes de un agente de usuario:

- **Interfaz de usuario**: Proporciona una interfaz a través de la cual los usuarios pueden interactuar con otros sistemas en la red. Esto puede incluir interfaces gráficas de usuario (GUI), interfaces de línea de comandos (CLI) o interfaces de voz, dependiendo del tipo de dispositivo o programa que actúe como agente de usuario.

- **Interacción con servidores**: Un agente de usuario puede enviar solicitudes a servidores en la red en nombre del usuario y recibir respuestas para mostrarlas al usuario. Esto puede incluir solicitudes HTTP a servidores web, solicitudes de correo electrónico a servidores de correo electrónico, solicitudes de archivos a servidores de archivos, entre otros.

- **Interpretación de datos**: Un agente de usuario puede interpretar y procesar datos recibidos de servidores en la red para que sean comprensibles para el usuario. Por ejemplo, un navegador web puede interpretar datos HTML y mostrar una página web formateada al usuario.

- **Gestión de sesiones**: Un agente de usuario puede gestionar sesiones de usuario, manteniendo el estado de la sesión y gestionando la autenticación y la autorización en nombre del usuario.

- **Adaptación de contenido:** Algunos agentes de usuario pueden adaptar el contenido para que sea adecuado para el dispositivo o el entorno en el que se está utilizando. Por ejemplo, un navegador web puede ajustar el diseño de una página web para que se vea bien en dispositivos móviles.

En resumen, un agente de usuario es una entidad fundamental en las comunicaciones de red que actúa en nombre de un usuario para interactuar con otros sistemas en la red, proporcionando interfaces, gestionando solicitudes y respuestas, interpretando datos y gestionando sesiones de usuario.

5. ¿Qué son y en qué se diferencian HTML y HTTP?

HTML (Hypertext Markup Language) y HTTP (Hypertext Transfer Protocol) son dos tecnologías fundamentales en el funcionamiento de la World Wide Web, pero cumplen funciones muy diferentes:

- **HTML (Hypertext Markup Language)**:
  - HTML es un lenguaje de marcado utilizado para crear páginas web.
  - Define la estructura y el contenido de una página web mediante etiquetas que describen diferentes elementos como encabezados, párrafos, enlaces, imágenes, formularios, entre otros.
  - HTML proporciona la estructura básica de una página web, pero no se encarga de la transferencia de datos entre el servidor y el cliente.
  - Los navegadores web interpretan el código HTML recibido del servidor y lo muestran al usuario en forma de página web.

- **HTTP (Hypertext Transfer Protocol)**:
  - HTTP es un protocolo de comunicación utilizado para transferir datos a través de la web.
  - Define las reglas y los procedimientos que permiten que los navegadores y los servidores web se comuniquen entre sí.
  - HTTP funciona como un sistema de solicitud y respuesta, donde un cliente (como un navegador web) envía solicitudes a un servidor web para obtener recursos (como páginas HTML, imágenes, archivos, etc.), y el servidor responde con los datos solicitados.
  - HTTP es un protocolo sin estado, lo que significa que cada solicitud y respuesta se procesa de forma independiente, sin tener en cuenta las solicitudes o respuestas anteriores.
  - Las URLs (Uniform Resource Locators) son utilizadas por HTTP para identificar recursos específicos en la web.

En resumen, mientras que HTML se utiliza para crear la estructura y el contenido de una página web, HTTP se encarga de la transferencia de datos entre el cliente y el servidor, permitiendo que el navegador solicite y reciba los recursos necesarios para mostrar la página web al usuario.

6. HTTP tiene definido un formato de mensaje para los requerimientos y las respuestas. (Ayuda: apartado "Formato de mensaje HTTP", Kurose).

El formato de mensaje HTTP define la estructura de los mensajes utilizados para las solicitudes y las respuestas en el protocolo HTTP. Aquí está un resumen de los elementos clave en el formato de mensaje HTTP según lo especificado en el libro de Kurose:

Formato de mensaje de solicitud HTTP:

1. **Línea de solicitud**: Esta línea contiene el método de solicitud (GET, POST, etc.), la URI del recurso solicitado y la versión de HTTP.

NOTA: La URI (Uniform Resource Identifier) es una cadena de caracteres que identifica un recurso particular en la web de manera única. Es un identificador estándar utilizado para localizar y acceder a recursos en internet. La URI consta de dos partes principales: el identificador del esquema y el identificador específico del recurso.

2. **Encabezados de solicitud**: Los encabezados proporcionan información adicional sobre la solicitud, como el tipo de contenido aceptado, las cookies, el agente de usuario, etc.

3. **Cuerpo de la solicitud**: Opcionalmente, una solicitud puede incluir datos adicionales enviados al servidor, como datos de formulario en una solicitud POST.

Formato de mensaje de respuesta HTTP:

1. **Línea de estado**: Esta línea contiene el código de estado de la respuesta, junto con un mensaje de estado breve.
   Los códigos de estado indican el resultado de la solicitud realizada por el cliente al servidor.
2. **Encabezados de respuesta**: Proporcionan información adicional sobre la respuesta, como el tipo de contenido devuelto, la fecha de la respuesta, cookies, etc.

   NOTA: Una cookie es un pequeño archivo de texto que un sitio web puede almacenar en el navegador web del usuario. Estas cookies contienen información específica sobre la actividad del usuario en ese sitio web y se utilizan para diversos propósitos, como

recordar la información de inicio de sesión, personalizar la experiencia del usuario, rastrear la actividad del usuario y mantener el estado de la sesión.
Las cookies se utilizan comúnmente en la web para:

- **Autenticación y sesión**: Las cookies pueden almacenar información de inicio de sesión del usuario, lo que permite a los usuarios mantenerse autenticados en un sitio web incluso después de cerrar y volver a abrir el navegador.
- **Personalización**: Las cookies pueden almacenar preferencias del usuario, como idioma, tema, tamaño de fuente, etc., para personalizar la experiencia del usuario en el sitio web.
- **Seguimiento del usuario**: Las cookies pueden rastrear la actividad del usuario en un sitio web, como las páginas visitadas, los enlaces clicados y el tiempo pasado en el sitio. Esta información puede ser utilizada para análisis de usuarios y marketing dirigido.
- **Carritos de compras:** En sitios web de comercio electrónico, las cookies pueden almacenar los productos agregados al carrito de compras por un usuario, permitiendo que el usuario navegue por el sitio y realice compras sin perder la información del carrito.
- **Publicidad dirigida**: Las cookies pueden utilizarse para mostrar anuncios específicos para un usuario, basados en su historial de navegación y comportamiento en línea.

Es importante tener en cuenta que las cookies pueden ser utilizadas tanto por el sitio web que las establece (cookies de primera parte) como por terceros, como redes de publicidad o servicios de análisis (cookies de terceros). Además, los usuarios tienen control sobre las cookies en su navegador y pueden configurar preferencias de privacidad para aceptar, rechazar o eliminar cookies según sea necesario.

3. **Cuerpo de la respuesta**: Contiene los datos devueltos por el servidor, como el contenido de una página web, imágenes, archivos, etc.

Ejemplo de formato de mensaje HTTP:

****Solicitud HTTP:****
```
GET /index.html HTTP/1.1
Host: www.ejemplo.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
```

****Respuesta HTTP:****
```
HTTP/1.1 200 OK
Date: Sat, 03 Apr 2024 12:00:00 GMT
Server: Apache/2.4.41 (Unix)
Content-Length: 255
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo</title>
</head>
<body>
  <h1>Bienvenido a mi sitio web</h1>
  <p>Esta es una página de ejemplo.</p>
</body>
```

</html>
```

En este ejemplo, se muestra una solicitud GET para obtener el recurso "/index.html" del servidor "www.ejemplo.com", seguido de una respuesta con un código de estado 200 OK, junto con el contenido HTML de la página web solicitada.

**a. ¿Qué información de la capa de aplicación nos indica si un mensaje es de requerimiento o de respuesta para HTTP? ¿Cómo está compuesta dicha información?¿Para qué sirven las cabeceras?**

En el protocolo HTTP, la información de la capa de aplicación que indica si un mensaje es de solicitud (requerimiento) o de respuesta se encuentra en la primera línea del mensaje, conocida como la "línea de solicitud" para las solicitudes y la "línea de estado" para las respuestas.

**### Para las solicitudes (requerimientos):**
La línea de solicitud contiene tres partes principales:

1. **Método HTTP**: Indica el tipo de acción que se realizará en el recurso identificado por la URI. Algunos ejemplos comunes de métodos son GET, POST, PUT, DELETE, etc.
2. **URI (Uniform Resource Identifier)**: Identifica el recurso al que se está accediendo en el servidor. Especifica la ubicación y el nombre del recurso dentro del servidor.
3. **Versión de HTTP**: Indica la versión del protocolo HTTP que se está utilizando

Ejemplo de línea de solicitud HTTP: **GET /index.html HTTP/1.1**

**### Para las respuestas:**
La línea de estado contiene tres partes principales:

1. **Versión de HTTP**: Indica la versión del protocolo HTTP que se está utilizando.
2. **Código de estado HTTP**: Indica el resultado de la solicitud realizada por el cliente. Algunos ejemplos comunes de códigos de estado son 200 (OK), 404 (No encontrado), 500 (Error interno del servidor), etc.
3. **Mensaje de estado**: Proporciona una breve descripción del resultado de la solicitud.

Ejemplo de línea de estado HTTP: **HTTP/1.1 200 OK**

**Cabeceras en HTTP:**
Las cabeceras son componentes adicionales en los mensajes HTTP que proporcionan información adicional sobre la solicitud o la respuesta. Estas cabeceras están formadas por pares de nombre-valor y se utilizan para una variedad de propósitos, como controlar el caché, especificar el tipo de contenido, establecer cookies, manejar la autenticación, entre otros.

Las cabeceras permiten a los clientes y servidores comunicarse de manera más detallada y controlar el comportamiento de la transferencia de datos. Proporcionan información contextual importante que ayuda a interpretar y procesar los mensajes HTTP de manera adecuada.

**b. ¿Cuál es su formato? (Ayuda: https://developer.mozilla.org/es/docs/Web/HTTP/Headers)**

El formato de las cabeceras en HTTP sigue una estructura sencilla de pares de nombre-valor, separados por dos puntos y seguidos de un espacio en blanco. Cada cabecera se presenta en una línea separada dentro del mensaje HTTP. Aquí está el formato básico:
Por ejemplo, una cabecera de solicitud HTTP para indicar el tipo de contenido puede ser:

**Content-Type: text/html**

En este ejemplo, "Content-Type" es el nombre de la cabecera y "text/html" es el valor de la cabecera. Esto indica que el contenido del mensaje es de tipo HTML.
```

Es importante tener en cuenta que las cabeceras pueden repetirse en un mensaje HTTP si es necesario. Por ejemplo, un mensaje de solicitud puede contener varias cabeceras "Cookie" para enviar múltiples cookies al servidor.

Además, hay algunas convenciones especiales para el formato de las cabeceras:

- Mayúsculas y minúsculas: Los nombres de las cabeceras no son sensibles a mayúsculas y minúsculas, lo que significa que "Content-Type" y "content-type" se consideran iguales.
- Espacios en blanco: Las cabeceras no pueden contener espacios en blanco al principio o al final del nombre o valor de la cabecera. Sin embargo, los valores de las cabeceras pueden contener espacios en blanco dentro del valor.
- Líneas vacías: Una línea vacía indica el final de las cabeceras y el comienzo del cuerpo del mensaje (si lo hay).

c. Suponga que desea enviar un requerimiento con la versión de HTTP 1.1 desde curl/7.74.0 a un sitio de ejemplo como www.misitio.com para obtener el recurso /index.html. En base a lo indicado, ¿qué información debería enviarse mediante encabezados? Indique cómo quedaría el requerimiento.

Para enviar un requerimiento necesitaríamos incluir al menos las siguientes cabeceras:
1. **Host**: Indica el nombre de dominio del servidor al que se está enviando la solicitud.
2. **User-Agent**: Proporciona información sobre el cliente que realiza la solicitud, en este caso, la versión de curl utilizada.
3. **Método HTTP**: Especifica el tipo de acción que se realizará en el recurso solicitado. Para obtener el recurso, usaremos el método GET.
El requerimiento resultante se vería así:

GET /index.html HTTP/1.1
Host: www.misitio.com
User-Agent: curl/7.74.0

7. Utilizando la VM, abra una terminal e investigue sobre el comando curl. Analice para qué sirven los siguientes parámetros (-I, -H, -X, -s).

El comando `curl` es una herramienta de línea de comandos utilizada para transferir datos a través de varios protocolos, como HTTP, HTTPS, FTP, entre otros. Se utiliza comúnmente en sistemas Unix-like y Windows para realizar solicitudes de red desde la línea de comandos.

Parámetros comunes de `curl`:

- **-I (--head)**: Este parámetro hace que `curl` envíe una solicitud HEAD al servidor en lugar de una solicitud GET.
  La solicitud HEAD es similar a una solicitud GET, pero el servidor solo devuelve los encabezados de respuesta y no el cuerpo del mensaje.
  Esto es útil para obtener información sobre el recurso sin descargar su contenido completo.

- **-H (--header)**: Permite agregar encabezados personalizados a la solicitud HTTP.
  Se utiliza para especificar cabeceras adicionales en la solicitud HTTP, como User-Agent, Content-Type, etc. Por ejemplo,
  «curl -H "Content-Type: application/json"» agrega un encabezado Content-Type con el valor "application/json" a la solicitud.

- **-X (--request)**: Especifica el método HTTP que se utilizará en la solicitud, como GET, POST, PUT, DELETE, etc. Por defecto, `**curl**` usa el método GET. Con este parámetro, se puede especificar explícitamente el método deseado.Por ejemplo, `-X POST` hará que `curl` realice una solicitud POST en lugar de GET.

- **-s (--silent)**: Este parámetro hace que `curl` opere en modo silencioso, es decir, que no muestre el progreso ni los mensajes de estado. Se utiliza para suprimir la salida estándar y simplificar la salida cuando se realizan solicitudes de `curl` en scripts u otras herramientas automatizadas. Es útil cuando solo se necesita el contenido de la respuesta y no se desea que se muestren los detalles de la transferencia.

- **-v (o --verbose)**: Este parámetro hace que curl funcione en modo verbose (detallado), lo que significa que mostrará información detallada sobre la solicitud y la respuesta HTTP en la salida estándar. Esto incluye los encabezados de solicitud y respuesta, así como información adicional sobre la transferencia de datos, como la dirección IP del servidor, los códigos de estado HTTP, etc.

8. Ejecute el comando curl sin ningún parámetro adicional y acceda a www.redes.unlp.edu.ar. Luego responda:
a. ¿Cuántos requerimientos realizó y qué recibió? Pruebe redirigiendo la salida (>) del comando curl a un archivo con extensión html y abrirlo con un navegador.

Curl realiza un unico requerimiento inicial solicitando la pagina HTML principal, sin embargo por detras el navegador realiza multiples envios de requerimiento adicionales para cargar datos incrustados en el HTML para poder ser visibles, tales como imagenes, diseño CSS, etc.

b. ¿Cómo funcionan los atributos href de los tags link e src -> img en html?

Los atributos `href` de los elementos `<link>` y src en `<img>` son fundamentales en HTML y se utilizan para especificar la ubicación de recursos externos que deben ser vinculados o incrustados en una página web. Aquí está cómo funcionan cada uno de ellos:

Atributo `href` en el elemento `<link>`:
- El atributo `href` se utiliza en el elemento `<link>` para especificar la ubicación de un recurso externo, como una hoja de estilo CSS. Este atributo indica la URL del recurso que se va a vincular con el documento HTML actual.
- Cuando se utiliza en una etiqueta `<link>`, el navegador interpreta el recurso enlazado como una hoja de estilo externa y lo carga para aplicar estilos al documento HTML.
- Ejemplo: `<link rel="stylesheet" href="estilos.css">`

Atributo `src` en el elemento `<img>`:
- El atributo `src` se utiliza en el elemento `<img>` para especificar la ubicación de la imagen que se va a incrustar en la página.
- Indica la URL de la imagen que se va a mostrar en el elemento `<img>`.
- Cuando se utiliza en una etiqueta `<img>`, el navegador descarga la imagen de la URL especificada y la muestra en el documento HTML.
- Ejemplo: `<img src="imagen.jpg" alt="Descripción de la imagen">`

En resumen, el atributo `href` se utiliza para vincular recursos externos como hojas de estilo CSS, mientras que el atributo `src` se utiliza para incrustar imágenes en una página web. Ambos atributos son fundamentales para la construcción y el diseño de páginas web modernas.

c. Para visualizar la página completa con imágenes como en un navegador, ¿alcanza con realizar un único requerimiento?

No, para visualizar una página completa con imágenes como en un navegador, generalmente no basta con realizar un único requerimiento. Esto se debe a que una página web típica está compuesta por múltiples recursos, como HTML, hojas de estilo CSS, scripts JavaScript, imágenes, fuentes, etc., que se enlazan desde la página principal.

Cuando se accede a una página web, el navegador realiza una solicitud inicial para obtener el documento HTML principal. Luego, el navegador analiza el HTML y encuentra enlaces a otros

recursos, como imágenes, archivos CSS, JavaScript, etc. Para cargar completamente la página, el navegador debe realizar solicitudes adicionales para cada uno de estos recursos enlazados.

En resumen, para visualizar una página completa con imágenes como en un navegador, es necesario realizar múltiples requerimientos HTTP para obtener todos los recursos asociados con la página.

d. ¿Cuántos requerimientos serían necesarios para obtener una página que tiene dos CSS, dos Javascript y tres imágenes? Diferencie cómo funcionaría un navegador respecto al comando curl ejecutado previamente.

Para obtener una página que tiene dos archivos CSS, dos archivos JavaScript y tres imágenes, se necesitarían varios requerimientos HTTP. Y el calculo seria:

- Requerimiento para el HTML principal: 1
- Requerimientos para los archivos CSS: dos requerimientos para los dos archivos CSS.
- Requerimientos para los archivos JavaScript: Dos requerimientos para los dos archivos JavaScript.
- Requerimientos para las imágenes: Tres requerimientos para las tres imágenes.

En total, serían necesarios 1(HTML)+ 2 (CSS) + 2 (JavaScript) + 3 (imágenes) = 8 requerimientos HTTP para obtener todos los recursos asociados con la página.

En cuanto a cómo funcionaría un navegador respecto al comando `curl` ejecutado previamente:

- Un navegador web, al cargar una página, realiza automáticamente todos los requerimientos necesarios para obtener todos los recursos asociados con la página, como archivos CSS, JavaScript, imágenes, etc. Esto se realiza de forma paralela para acelerar la carga de la página.

- Por otro lado, el comando `curl` ejecutado previamente solo realizó un requerimiento HTTP para obtener el HTML de la página principal. Si deseamos obtener todos los recursos asociados con la página, deberíamos realizar múltiples requerimientos adicionales utilizando `curl` con los URLs de cada recurso específico. Sin embargo, esto sería más complicado y menos eficiente que la carga automática realizada por un navegador.

9. Ejecute a continuación los siguientes comandos:

curl -v -s www.redes.unlp.edu.ar > /dev/null

```
*   Trying 172.28.0.50:80...
* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)
> GET / HTTP/1.1                    //se está realizando una solicitud GET al recurso raíz ("/")
> Host: www.redes.unlp.edu.ar
> User-Agent: curl/7.74.0           //especifica el agente de usuario que se está utilizando para
                                      realizar la solicitud
> Accept: */*                       //Esta línea especifica el tipo de contenido que el cliente está
                                      dispuesto a aceptar en la respuesta del servidor. En este
                                      caso, */* indica que el cliente acepta cualquier tipo de
                                      contenido.
>                                   //respuesta del servidor a la solicitud HTTP
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK                   // Estado de la respuesta, en este caso solicitud exitosa
< Date: Thu, 04 Apr 2024 13:11:21 GMT      //fecha y hs de la respuesta del servidor
< Server: Apache/2.4.56 (Unix)      //indica el tipo y versión del servidor web que está
                                    procesando la solicitud.
< Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT  //ultima fecha de modificacion el recurso
< ETag: "1322-5f7457bd64f80"        // es una etiqueta de entidad única para el recurso. Es
                                      utilizado para la validación de caché.
< Accept-Ranges: bytes              //Esta línea indica que el servidor acepta solicitudes de rango
                                      para el recurso, lo que significa que el cliente puede solicitar
```

solo una parte específica del recurso en lugar de todo el contenido.
< Content-Length: 4898 //Esta línea especifica la longitud del contenido de la respuesta en bytes. En este caso, el contenido de la respuesta tiene una longitud de 4898 bytes.
< Content-Type: text/html //Esta línea indica el tipo de contenido del cuerpo de la respuesta.En este caso, el tipo de contenido es texto/html, lo que significa que el cuerpo de la respuesta es un documento HTML
<
{ [4898 bytes data]
* Connection #0 to host www.redes.unlp.edu.ar left intact

curl -I -v -s www.redes.unlp.edu.ar

* Trying 172.28.0.50:80...
* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)
> HEAD / HTTP/1.1
> Host: www.redes.unlp.edu.ar
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< Date: Thu, 04 Apr 2024 13:16:09 GMT
Date: Thu, 04 Apr 2024 13:16:09 GMT
< Server: Apache/2.4.56 (Unix)
Server: Apache/2.4.56 (Unix)
< Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT
Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT
< ETag: "1322-5f7457bd64f80"
ETag: "1322-5f7457bd64f80"
< Accept-Ranges: bytes
Accept-Ranges: bytes
< Content-Length: 4898
Content-Length: 4898
< Content-Type: text/html
Content-Type: text/html

<
* Connection #0 to host www.redes.unlp.edu.ar left intact

a. ¿Qué diferencias nota entre cada uno?

Como igualdad, cada solicitud es enviada de la misma manera y los datos resultantes mostrados en la terminal son iguales. La diferencia se nota en como se muestran los datos resultantes, En la primer solicitud se reenvian los datos a un archivo en /dev/null, pero en terminal muestra el encabezado con los datos correctamente. En la segunda solicitud el encabezado muestra una duplicacion de cada item, los cuales las KEY se encuentran en negritam no se por que.

b. ¿Qué ocurre si en el primer comando se quita la redirección a /dev/null? ¿Por qué no es necesaria en el segundo comando?

La diferencia esta dada en que el comando **curl -v -s www.redes.unlp.edu.ar** sin el redireccionamiento de la salida estandar muestra el archivo HTML tambien ademas de mostrar la solicitud y respuesta del servidor. Mientras que el redireccionamiento lo descarta.

c. ¿Cuántas cabeceras viajaron en el requerimiento? ¿Y en la respuesta?

Cabeceras visibles 1 sola, la inicial de la solicitud del HTML. En cuanto a solicitudes extras: **href**: 3 + 1 + 4 + 1+ 2+3 = 14 solicitudes adicionales.

10. ¿Qué indica la cabecera Date?

La cabecera date indica la fecha y hora de la respuesta del servidor.

11. En HTTP/1.0, ¿cómo sabe el cliente que ya recibió todo el objeto solicitado de manera completa? ¿Y en HTTP/1.1?

En HTTP/1.0 y HTTP/1.1, los clientes pueden determinar si han recibido todo el objeto solicitado de manera completa mediante diferentes métodos:

En HTTP/1.0:

- En HTTP/1.0, el cliente determina que ha recibido todo el objeto solicitado de manera completa basándose en la longitud del contenido (especificada en el encabezado Content-Length). Cuando el servidor envía un objeto en respuesta a una solicitud HTTP/1.0, incluye el encabezado **Content-Length** para indicar la longitud del contenido del objeto. El cliente utiliza este valor para determinar cuántos bytes debe recibir. Cuando el cliente ha recibido la cantidad especificada de bytes, considera que ha recibido todo el objeto.

En HTTP/1.1:

- En HTTP/1.1, tendremos dos maneras de registrar si se ha recibido todo el objeto solicitado de manera completa:
  - Al igual que en HTTP/1.0 tendremos el encabezado **Content-Length**, donde el cliente puede constatar que la cantidad de datos recibidos son = la cantidad de datos de la respuesta del servidor en el **Content-Length.**
  - Si el servidor no proporciona un encabezado Content-Length, el cliente puede utilizar el mecanismo de Transfer-Encoding "chunked" para recibir el contenido en trozos y determinar que ha recibido todo el objeto cuando recibe el marcador de finalización de chunk.

En resumen, en HTTP/1.0, el cliente determina que ha recibido todo el objeto basándose en la longitud del contenido especificada en el encabezado Content-Length, mientras que en HTTP/1.1, el cliente puede utilizar el encabezado Content-Length o el mecanismo de Transfer-Encoding "chunked" para determinar que ha recibido todo el objeto solicitado de manera completa.

12. Investigue los distintos tipos de códigos de retorno de un servidor web y su significado. Considere que los mismos se clasifican en categorías (2XX, 3XX, 4XX, 5XX).

Los códigos de retorno de un servidor web se utilizan para indicar el resultado de una solicitud HTTP realizada por un cliente. Estos códigos se dividen en cinco categorías principales, cada una con un significado específico:

1. **Códigos de éxito (2XX)**: Los códigos de éxito indican que la solicitud del cliente fue recibida, entendida y aceptada correctamente por el servidor. Ejemplos comunes incluyen:
   - 200 OK: Indica que la solicitud se realizó con éxito y el servidor ha devuelto los datos solicitados.
   - 201 Created: Indica que la solicitud ha sido cumplida y se ha creado un nuevo recurso.
   - 204 No Content: Indica que la solicitud se ha procesado correctamente, pero no hay contenido para devolver.
2. **Redirecciones (3XX)**: Los códigos de redirección indican que el cliente necesita realizar más acciones para completar la solicitud. Ejemplos comunes incluyen:
   - 301 Moved Permanently: Indica que la URL solicitada ha sido permanentemente movida a una nueva ubicación.
   - 302 Found: Indica que la URL solicitada ha sido temporalmente movida a una nueva ubicación.
   - 304 Not Modified: Indica que el recurso no ha sido modificado desde la última vez que fue solicitado, por lo que se puede usar la versión en caché.

3. **Errores del cliente (4XX)**: Los códigos de error del cliente indican que la solicitud del cliente no pudo ser completada debido a un error en el lado del cliente. Ejemplos comunes incluyen:
   - 400 Bad Request: Indica que la solicitud del cliente es incorrecta o está mal formada.
   - 403 Forbidden: Indica que el cliente no tiene permiso para acceder al recurso solicitado.
   - 404 Not Found: Indica que el recurso solicitado no fue encontrado en el servidor.
4. **Errores del servidor (5XX)**: Los códigos de error del servidor indican que la solicitud del cliente no pudo ser completada debido a un error en el lado del servidor. Ejemplos comunes:
   - 500 Internal Server Error: Indica que se produjo un error interno en el servidor al procesar la solicitud.
   - 502 Bad Gateway: Indica que el servidor actuando como un proxy o puerta de enlace recibió una respuesta no válida del servidor ascendente.
   - 503 Service Unavailable: Indica que el servidor no está disponible temporalmente para manejar la solicitud debido a una sobrecarga o mantenimiento.

Estas categorías y códigos de retorno son estándares definidos por el Protocolo de Transferencia de Hipertexto (HTTP) y son utilizados por los servidores web para comunicar el resultado de una solicitud HTTP al cliente.

13. Utilizando curl, realice un requerimiento con el método HEAD al sitio www.redes.unlp.edu.ar e indique:

```
curl -I www.redes.unlp.edu.ar
HTTP/1.1 200 OK
Date: Thu, 04 Apr 2024 14:17:46 GMT
Server: Apache/2.4.56 (Unix)
Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT
ETag: "1322-5f7457bd64f80"
Accept-Ranges: bytes
Content-Length: 4898
Content-Type: text/html
```

a. ¿Qué información brinda la primera línea de la respuesta?

HTTP/1.1 200 OK: Este es el encabezado de estado que indica que la solicitud fue exitosa. También incluye el código de estado HTTP 200 y el mensaje de estado "OK".

b. ¿Cuántos encabezados muestra la respuesta?

Muestra 8 encabezados.

c. ¿Qué servidor web está sirviendo la página?

Server: Apache/2.4.56 (Unix)

d. ¿El acceso a la página solicitada fue exitoso o no?

HTTP/1.1 200 OK, si fue exitosa.

e. ¿Cuándo fue la última vez que se modificó la página?

Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT

f. Solicite la página nuevamente con curl usando GET, pero esta vez indique que quiere obtenerla sólo si la misma fue modificada en una fecha posterior a la que efectivamente fue modificada. ¿Cómo lo hace? ¿Qué resultado obtuvo? ¿Puede explicar para qué sirve?

Para solicitar la página nuevamente con `curl` usando el método GET y especificando que se quiere obtenerla solo si ha sido modificada después de una fecha determinada, se puede utilizar el encabezado `If-Modified-Since` en la solicitud. Este encabezado se usa para indicar al servidor que solo envíe el contenido solicitado si ha sido modificado desde la fecha especificada. La solicitud se vería así:

curl -v -s -H "If-Modified-Since: Sun, 19 Mar 2023 19:04:46 GMT" www.redes.unlp.edu.ar

En esta solicitud, se especifica la fecha (Sun, 19 Mar 2023 19:04:46 GMT) en la que la página fue modificada por última vez. El servidor verificará si la página ha sido modificada desde esta fecha.

Si la página no ha sido modificada, el servidor devolverá un código de estado `**304 Not Modified**` y no enviará el cuerpo de la respuesta, lo que indica al cliente que puede usar su versión en caché local. Si la página ha sido modificada, el servidor enviará el contenido actualizado junto con un código de estado `**200 OK**`.

El propósito de este mecanismo es minimizar el tráfico de red y la carga en el servidor, permitiendo que el cliente use versiones en caché de recursos web cuando sea posible. Si el recurso no ha cambiado desde la última vez que se solicitó, no es necesario volver a descargarlo, y el servidor puede indicar esto al cliente mediante el código de estado `304 Not Modified`. Esto ayuda a reducir el consumo de ancho de banda y mejorar el rendimiento general de la aplicación web.

14. Utilizando curl, acceda al sitio www.redes.unlp.edu.ar/restringido/index.php y siga las instrucciones y las pistas que vaya recibiendo hasta obtener la respuesta final. Será de utilidad para resolver este ejercicio poder analizar tanto el contenido de cada página como los encabezados.

1. curl www.redes.unlp.edu.ar/restringido/index.php
   Respuesta:
   <h1>Acceso restringido</h1>

   Para acceder al contenido es necesario autenticarse. Para obtener los datos de acceso seguir las instrucciones detalladas en www.redes.unlp.edu.ar/obtener-usuario.php

2. curl www.redes.unlp.edu.ar/obtener-usuario.php
   Respuesta:
   Para obtener el usuario y la contraseña haga un requerimiento a esta página seteando el encabezado 'Usuario-Redes' con el valor 'obtener'

3. curl -H 'Usuario-Redes: obtener' www.redes.unlp.edu.ar/obtener-usuario.php
   Respuesta:
   Bien hecho! Los datos para ingresar son:

   Usuario: redes

   Contraseña: RYC

   Ahora vuelva a acceder a la página inicial con los datos anteriores.

   PISTA: Investigue el uso del encabezado Authorization para el método Basic. El comando base64 puede ser de ayuda!

4. man base64
   Respuesta:
   Base64 encode or decode FILE, or standard input, to standard output.
   With no FILE, or when FILE is -, read standard input.
   Mandatory arguments to long options are mandatory for short options too.
   -d, --decode: decode data
   -i, --ignore-garbage when decoding, ignore non-alphabet characters
   -w, --wrap=COLS wrap encoded lines after COLS character (default 76). Use 0 to disable line wrapping
   --help display this help and exit
   --version output version information and exit

5. Para realizar este paso debemos transformar la contraseña dada en formato base64:
   La codificación de las credenciales "redes:RYC" en Base64. **"cmVkZXNfOlJZQw=="**

   curl -H 'Authorization: Basic cmVkZXNfOlJZQw==' www.redes.unlp.edu.ar

Respuesta:
¡Felicitaciones, llegaste al final del ejercicio! Fecha: 2024-04-05 01:50:50 Verificación: 567fa63d4e3981efe65920a5c8341abd5132b9761dce311ffb89af033d336d06

15. Utilizando la VM, realice las siguientes pruebas:
a. Ejecute el comando 'curl www.redes.unlp.edu.ar/extras/prueba-http-1-0.txt' y copie la salida completa (incluyendo los dos saltos de línea del final).

curl www.redes.unlp.edu.ar/extras/prueba-http-1-0.txt

RESPUESTA:
GET /http/HTTP-1.1/ HTTP/1.0
User-Agent: curl/7.38.0
Host: www.redes.unlp.edu.ar
Accept: */*

b. Desde la consola ejecute el comando telnet www.redes.unlp.edu.ar 80 y luego pegue el contenido que tiene almacenado en el portapapeles. ¿Qué ocurre luego de hacerlo?
SOLICITUD:
telnet www.redes.unlp.edu.ar 80
Trying 172.28.0.50… **//RESPUESTA DE LA SOLICITUD, piden los datos del host.**
Connected to www.redes.unlp.edu.ar.
Escape character is '^]'.
GET /http/HTTP-1.1/ HTTP/1.0 //Solicitud nueva
User-Agent: curl/7.38.0
Host: www.redes.unlp.edu.ar
Accept: */*

RESPUESTA:
HTTP/1.1 200 OK
Date: Sat, 06 Apr 2024 00:58:21 GMT
Server: Apache/2.4.56 (Unix)
Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT
ETag: "760-5f7457bd64f80"
Accept-Ranges: bytes
Content-Length: 1888
Connection: close
Content-Type: text/html

Y una pagina HTML ...

c. Repita el proceso anterior, pero copiando la salida del recurso /extras/prueba-http-1-1.txt. Verifique que debería poder pegar varias veces el mismo contenido sin tener que ejecutar el comando telnet nuevamente.
El resultado es el mismo, la diferencia que hay es que la conexión no se cierra una vez que el servidor me envía la respuesta.

16. En base a lo obtenido en el ejercicio anterior, responda:
a. ¿Qué está haciendo al ejecutar el comando telnet?

Al ejecutar `telnet www.redes.unlp.edu.ar 80`, estamos estableciendo una conexión a través del protocolo Telnet al puerto 80 del servidor www.redes.unlp.edu.ar, que es el puerto estándar para HTTP. Telnet no utiliza un método HTTP en sí mismo, ya que es solo una herramienta de comunicación de red. Sin embargo, podemos enviar manualmente una solicitud HTTP utilizando el Telnet. Para hacerlo, normalmente se envía una solicitud GET para solicitar un recurso específico.

b. ¿Qué método HTTP utilizó? ¿Qué recurso solicitó?

La solicitud GET que se envía manualmente a través de Telnet generalmente solicitaría el recurso raíz ("/") o algún recurso específico del servidor web. Solicitando los datos del recurso obtenido en el punto 15.

c. ¿Qué diferencias notó entre los dos casos? ¿Puede explicar por qué?

La diferencia principal entre usar Telnet y usar `curl` o un navegador web es la conveniencia y la capacidad de formatear y enviar solicitudes HTTP de manera más fácil y estructurada. Con Telnet, debemos escribir manualmente cada línea de la solicitud HTTP, lo que puede llevar más tiempo y es propenso a errores. `Curl` o un navegador web, por otro lado, manejan toda la comunicación HTTP por nosotros de manera más eficiente y sin necesidad de una intervención manual.

d. ¿Cuál de los dos casos le parece más eficiente? Piense en el ejercicio donde analizó la cantidad de requerimientos necesarios para obtener una página con estilos, javascripts e imágenes. El caso elegido, ¿puede traer asociado algún problema?

En términos de eficiencia, `curl` o un navegador web son claramente más eficientes, ya que manejan la comunicación HTTP de manera automática y más estructurada. Además, pueden realizar solicitudes HTTP concurrentes y manejar mejor la complejidad de las páginas web modernas que contienen múltiples recursos. Sin embargo, Telnet puede ser útil para propósitos de depuración y comprensión de la comunicación HTTP a un nivel más bajo.

En cuanto al ejercicio de analizar la cantidad de requerimientos necesarios para obtener una página con estilos, JavaScript e imágenes, usar Telnet manualmente sería extremadamente ineficiente y propenso a errores, por ejemplo sintacticos. Además, no podríamos ver el contenido real de los recursos como lo haríamos con `curl` o un navegador web, lo que dificultaría el análisis. Por lo tanto, `curl` o un navegador web serían la opción preferida en términos de eficiencia y facilidad de uso para este tipo de tarea.

17. En el siguiente ejercicio veremos la diferencia entre los métodos POST y GET. Para ello, será necesario utilizar la VM y la herramienta Wireshark. Antes de iniciar considere:
■ Capture los paquetes utilizando la interfaz con IP 172.28.0.1. (Menú “Capture-Options” (es el icono de settings al lado del stop capture). Luego seleccione la interfaz correspondiente (en mi caso red br-XXXX addresses: 172.28.0.1) y presione Start).
■ Para que el analizador de red sólo nos muestre los mensajes del protocolo http introduciremos la cadena ‘http’ (sin las comillas) en la ventana de especificación de filtros de visualización (display-filter). Si no hiciéramos esto veríamos todo el tráfico que es capaz de capturar nuestra placa de red. De los paquetes que son capturados, aquel que esté seleccionado será mostrado en forma detallada en la sección que está justo debajo. Como sólo estamos interesados en http ocultaremos toda la información que no es relevante para esta práctica (Información de trama, Ethernet, IP y TCP). Desplegar la información correspondiente al protocolo HTTP bajo la leyenda “Hypertext Transfer Protocol”.
■ Para borrar la cache del navegador, deberá ir al menú “Herramientas->Borrar historial reciente”. Alternativamente puede utilizar Ctrl+F5 en el navegador para forzar la petición HTTP evitando el uso de caché del navegador.
■ En caso de querer ver de forma simplificada el contenido de una comunicación http,

utilice el botón derecho sobre un paquete HTTP perteneciente al flujo capturado y seleccione la opción Follow TCP Stream.
a. Abra un navegador e ingrese a la URL: www.redes.unlp.edu.ar e ingrese al link en la sección “Capa de Aplicación” llamado “Métodos HTTP”. En la página mostrada se visualizan dos nuevos links llamados: Método GET y Método POST. Ambos muestran un formulario como el siguiente:

b. Analice el código HTML:

Para analizarlo tuve que abrir en el mozilla con f12 la ventana de analisis:
POST:

```
<!DOCTYPE html>
<html lang="en"> scroll overflow
 <head> ··· </head>
 <body>
  <div id="wrap">
   <div class="navbar navbar-inverse navbar-fixed-top"> ··· </div>
   <div class="container">
     ::before
     <h1>Métodos HTTP: POST</h1>
    <p> ··· </p>
    <form method="POST" action="metodos-lectura-valores.php"> ··· </form>
     <p></p>
     ::after
    </div>
   </div>
  <div id="footer"> ··· </div>
  </body>
</html>
```

GET:

```
<!DOCTYPE html>
<html lang="en"> scroll overflow
 <head> ··· </head>
 <body>
  <div id="wrap">
   <div class="navbar navbar-inverse navbar-fixed-top"> ··· </div>
   <div class="container">
     ::before
     <h1>Métodos HTTP: GET</h1>
    <p> ··· </p>
    <form method="GET" action="metodos-lectura-valores.php"> ··· </form>
     <p></p>
     ::after
    </div>
   </div>
  <div id="footer"> ··· </div>
  </body>
</html>
```

c. Utilizando el analizador de paquetes Wireshark capture los paquetes enviados y recibidos al presionar el botón Enviar.

d. ¿Qué diferencias detectó en los mensajes enviados por el cliente?

La diferencia que se detecta es que si se trata de un GET, se envia la información como parámetro de la URL, mientras que si se trata de un POST, esta se envia en el cuerpo de la solicitud.

e. ¿Observó alguna diferencia en el browser si se utiliza un mensaje u otro?

Si, en el caso del GET, se pueden ver en la URL los parámetros enviados, en el caso del POST no, se mantiene la misma URL que se tenia antes de apretar el botón ENVIAR.

18. Investigue cuál es el principal uso que se le da a las cabeceras Set-Cookie y Cookie en HTTP y qué relación tienen con el funcionamiento del protocolo HTTP.

Las cabeceras `Set-Cookie` y `Cookie` en HTTP se utilizan principalmente para la gestión de sesiones y la implementación de la funcionalidad de seguimiento de estado en las aplicaciones web. Aquí está cómo se utilizan y su relación con el funcionamiento del protocolo HTTP:

1. **Set-Cookie**:
   - Cuando un servidor envía una respuesta HTTP que contiene la cabecera `Set-Cookie`, está instruyendo al navegador del cliente a almacenar una cookie específica en su lado. La cookie generalmente contiene información como un identificador de sesión único, preferencias del usuario, datos de autenticación, etc.
   - La cabecera `Set-Cookie` tiene la siguiente sintaxis: `Set-Cookie: <nombre>=<valor>[; <atributo>=<valor>]`. Puede incluir varios atributos opcionales, como el tiempo de expiración de la cookie, el dominio y la ruta para los cuales es válida, etc.
   - Cuando el navegador recibe una respuesta con la cabecera `Set-Cookie`, almacena la cookie asociada en su almacenamiento local y la incluirá automáticamente en todas las solicitudes posteriores al mismo dominio y ruta.

2. **Cookie**:
   - Cuando el navegador envía una solicitud HTTP al servidor, incluirá automáticamente todas las cookies almacenadas asociadas con ese dominio y ruta en la cabecera `Cookie`.
   - La cabecera `Cookie` tiene la siguiente sintaxis: `Cookie: <nombre>=<valor>`.
   - Al recibir una solicitud con la cabecera `Cookie`, el servidor puede acceder a la información almacenada en las cookies y usarla para personalizar la respuesta, como autenticar al usuario, recuperar sus preferencias, etc.

Relación con el funcionamiento del protocolo HTTP:
- Las cookies son esenciales para el funcionamiento de muchas aplicaciones web modernas que requieren seguimiento de estado y autenticación de usuarios.
- Permiten que los servidores web mantengan información sobre las sesiones de los usuarios a lo largo del tiempo, lo que es fundamental para implementar características como la autenticación de usuarios, la personalización del contenido y el seguimiento del estado de la sesión.
- La capacidad de enviar y recibir cookies a través de las cabeceras `Set-Cookie` y `Cookie` está integrada en el protocolo HTTP y es fundamental para la comunicación cliente-servidor en la web.

19. ¿Cuál es la diferencia entre un protocolo binario y uno basado en texto? ¿De qué tipo de protocolo se trata HTTP/1.0, HTTP/1.1 y HTTP/2?
La diferencia principal entre un protocolo binario y uno basado en texto radica en cómo se representan los datos que se envían a través del protocolo:

1. **Protocolo binario:**
- En un protocolo binario, los datos se representan en forma de secuencias de bits, lo que significa que los datos se transmiten en su forma binaria (códigos binarios).
- Los datos son estructurados y compactos, lo que permite una transmisión más eficiente, especialmente en redes con ancho de banda limitado.
- Los datos en un protocolo binario son menos legibles para los humanos y generalmente requieren herramientas especiales para interpretar y visualizar su contenido.

2. **Protocolo basado en texto:**
- En un protocolo basado en texto, los datos se representan en forma de texto legible para los humanos, como ASCII o UTF-8.
- Los datos son fácilmente legibles y comprensibles para los humanos, lo que facilita la depuración y el diagnóstico de problemas de comunicación.
- Los datos en un protocolo basado en texto suelen ocupar más espacio en comparación con un protocolo binario, ya que cada carácter se representa como un byte.

En cuanto a los protocolos HTTP:

- **HTTP/1.0**: Es principalmente un protocolo basado en texto. Las solicitudes y respuestas HTTP se componen de líneas de texto ASCII legibles para humanos, seguidas de encabezados y opcionalmente un cuerpo. Aunque algunos componentes de la solicitud o respuesta pueden estar en formato binario (como el cuerpo del mensaje si se envían archivos binarios), el protocolo en sí se basa en texto.

- **HTTP/1.1**: Al igual que HTTP/1.0, HTTP/1.1 es principalmente un protocolo basado en texto. Sin embargo, introduce mejoras y optimizaciones con respecto a HTTP/1.0 para mejorar el rendimiento y la eficiencia de la comunicación entre el cliente y el servidor.

- **HTTP/2**: HTTP/2 es un protocolo binario. Introduce un nuevo formato de marco binario para mejorar la eficiencia de la transmisión de datos, reducir la latencia y mejorar el rendimiento general de las aplicaciones web. Aunque el protocolo HTTP/2 se define en binario, a menudo se puede visualizar en una forma legible para humanos utilizando herramientas especializadas.

20. Responder las siguientes preguntas:
a. ¿Qué función cumple la cabecera Host en HTTP 1.1? ¿Existía en HTTP 1.0? ¿Qué sucede en HTTP/2? (Ayuda: https://undertow.io/blog/2015/04/27/An-in-depth- overview-of-HTTP2.html para HTTP/2)

La cabecera `Host` en HTTP/1.1 y su comportamiento en HTTP/1.0 y HTTP/2 son los siguientes:
En **HTTP/1.0** no existia la cabecera HOST. Esta fue introducida en HTTP/1.1 para permitir el alojamiento virtual de múltiples sitios web en un único servidor basado en nombres de dominio.
En **HTTP/1.1**, la cabecera `Host` es obligatoria en todas las solicitudes. Esta cabecera indica al servidor web el nombre de dominio al que se está dirigiendo la solicitud. Cuando un cliente envía una solicitud HTTP/1.1 a un servidor, debe incluir la cabecera `Host` para indicar al servidor qué dominio está solicitando.
Por ejemplo: - Una solicitud HTTP/1.1 con la cabecera `Host: www.ejemplo.com` indica al servidor que la solicitud está destinada al dominio www.ejemplo.com.

En **HTTP/2**, la cabecera `Host` conserva su función de indicar al servidor web el nombre de dominio al que se está dirigiendo la solicitud, similar a HTTP/1.1.

La diferencia principal en **HTTP/2** es que la cabecera `Host` es parte del encabezado de estado inicial y no se repite en cada trama de datos, como en HTTP/1.1. Esto se debe a que HTTP/2 utiliza la multiplexación de flujo y el marco de encabezado binario, lo que permite que múltiples solicitudes se envíen y reciban simultáneamente en una única conexión TCP.

- La cabecera `Host` sigue siendo obligatoria en todas las solicitudes en HTTP/2, pero su tratamiento es diferente debido a la arquitectura de multiplexación de flujo del protocolo.

b. ¿Cómo quedaría en HTTP/2 el siguiente pedido realizado en HTTP/1.1 si se está usando https?

```
GET /index.php HTTP/1.1
Host: www.info.unlp.edu.ar
```

RESPUESTA:

```
GET /index.php HTTP/2
Host: www.info.unlp.edu.ar
```

//EJERCICIO DE EXAMEN//

```
curl -X ?? www.redes.unlp.edu.ar/??
> HEAD /metodos/ HTTP/??
> Host: www.redes.unlp.edu.ar
> User-Agent: curl/7.54.0
< HTTP/?? 200 OK
< Server: nginx/1.4.6 (Ubuntu)
< Date: Wed, 31 Jan 2018 22:22:22 GMT
< Last-Modified: Sat, 20 Jan 2018 13:02:41 GMT
< Content-Type: text/html; charset=UTF-8
< Connection: close
```

a. ¿Qué versión de HTTP podría estar utilizando el servidor?

La versión de HTTP que podría estar utilizando el servidor es HTTP/1.1. Esto se puede inferir por la presencia del encabezado "HTTP/?? 200 OK" en la respuesta, donde "200 OK" indica que la solicitud fue exitosa, y el servidor está utilizando el protocolo HTTP.

b. ¿Qué método está utilizando? Dicho método, ¿retorna el recurso completo solicitado?

El método utilizado es HEAD. Este método solicita únicamente los encabezados de la respuesta HTTP sin el cuerpo de la entidad. No retorna el recurso completo solicitado, solo la información de los encabezados.

c. ¿Cuál es el recurso solicitado?

El recurso solicitado es "/metodos/".

d. ¿El método funcionó correctamente?

Si debido al mensaje de éxito del codigo “200 OK”

e. Si la solicitud hubiera llevado un encabezado que diga:
If-Modified-Since: Sat, 20 Jan 2018 13:02:41 GMT
¿Cuál habría sido la respuesta del servidor web? ¿Qué habría hecho el navegador en este caso?

Si la solicitud hubiera llevado un encabezado "If-Modified-Since" indicando la fecha de la última modificación, el servidor habría comprobado si el recurso ha sido modificado desde la fecha

especificada. Si el recurso no ha sido modificado, el servidor habría respondido con un código de estado "304 Not Modified" y no habría enviado el cuerpo de la respuesta. En este caso, el navegador habría utilizado la versión en caché del recurso si está disponible y si aún es válido según la política de almacenamiento en caché.