Práctica 2 - Redes y Comunicaciones

Ejercicios de la Práctica

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

- **Funciones de la Capa de Aplicación del modelo TCP/IP:**
  - Provee servicios de comunicación a los usuarios y a las aplicaciones, incluye las aplicaciones mismas.
  - Las aplicaciones que usan la red y los protocolos que implementan las aplicaciones pertenecen a esta capa.
  - Interfaz con el usuario (UI) u otras aplicaciones/servicios.
- **Funciones de la Capa de Aplicación del modelo OSI:**
  - Define el formato (sintaxis y semántica) de los mensajes. Existen protocolos que trabajan con formatos específicos de mensajes. A su vez, define cómo debe ser el intercambio de mensajes.

2. **Si dos procesos deben comunicarse:**
   a. **¿Cómo podrían hacerlo si están en diferentes máquinas?**

   - Los procesos de dos sistemas terminales diferentes se comunican entre ellos intercambiando mensajes a través de la red de computadoras. Un proceso emisor crea y envía mensajes a la red; un proceso receptor recibe estos mensajes y posiblemente responde devolviendo mensajes.

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

   - Cuando los procesos se ejecutan en el mismo sistemas terminal, pueden comunicarse entre sí mediante la comunicación entre procesos aplicando las reglas gobernadas por el sistema operativo del sistema terminal.

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 cliente se conecta al servidor y se comunica a través de este haciendo pedidos y el servidor corre el servicio esperando de forma pasiva la conexión y el pedido de datos/requerimientos. La carga de trabajo se ve más repartida entre el cliente y el servidor. Un ejemplo en la vida cotidiana sería un negocio de venta de pizzas donde el Servidor es la pizzería que tiene todos los ingredientes y prepara las pizzas y los Clientes son las personas que llaman a la pizzería y hacen un pedido. Un ejemplo en un sistema informático podría ser un navegador web y un servidor web donde están

los Servidores que almacenan páginas web, aplicaciones, recursos, etc. como pueden ser Apache o Nginx, y luego están los Clientes que serían los navegadores web como Google o Firefox que hacen solicitudes de esos recursos al Servidor. Existen otros modelos de comunicación como lo son el Modelo Mainframe (Dumb Client) o el Modelo Peer-to-Peer.

4. **Describa la funcionalidad de la entidad genérica "Agente de usuario" o "User agent".**

- El Agente de Usuario o User Agent es un software que actúa como intermediario entre un usuario y un servicio o recurso en la web. Este término se refiere a cualquier aplicación o dispositivo que envía solicitudes a servidores web en nombre del usuario. Los navegadores web son el tipo más común de Agente de Usuario, pero también existen otros, como aplicaciones móviles, asistentes de voz, motores de búsqueda, etc. El User Agent comunica al servidor detalles del navegador, su versión, y a veces información sobre el sistema operativo y dispositivo. Esto permite que los sitios adapten su contenido a las capacidades del navegador. En una solicitud HTTP del navegador, el User Agent es un encabezado que informa sobre el navegador y sistema operativo.

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

- **HTML (HyperText Markup Language):**
    - HTML es un lenguaje de marcado que se utiliza para estructurar y presentar contenido en la web. Los documentos HTML consisten en una serie de elementos y etiquetas que definen la estructura y el formato del contenido, como encabezados, párrafos, enlaces, imágenes, tablas, y más.
    - HTML describe cómo debe presentarse la información en un navegador web. Es el código subyacente que los navegadores interpretan y renderizan como páginas web. Aunque HTML por sí solo no incluye estilos (que suelen manejarse con CSS) ni comportamientos dinámicos (que suelen manejarse con JavaScript), es el esqueleto de todas las páginas web.
- **HTTP (HyperText Transfer Protocol):**
    - Protocolo de comunicación utilizado en la web para la transferencia de datos, define cómo se deben enviar y recibir las solicitudes y respuestas en la web, sigue el Modelo Cliente/Servidor, es sin estado, es decir, por cada requerimiento que hace el cliente, el servidor no almacena información sobre el estado de la relación con el cliente.
    - Es un protocolo que corre sobre TCP y que usa el puerto 80 por default.

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

- La información que nos indica si un mensaje es de requerimiento/solicitud o de respuesta se da en la primera línea de los mensajes.
    - En un **Mensaje de Solicitud**, la primera línea se denomina línea de solicitud y tiene el formato → **<Método> <URI> <Versión de HTTP>.** Un ejemplo sería **GET /unaDirección/página.html HTTP/1.1**.
        - **Método:** Acción a realizar, por ejemplo GET, POST, HEAD, PUT, DELETE, etc.
        - **URI (Uniform Resource Identifier):** Especifica el recurso que se solicita.
        - **Versión de HTTP:** Indica la versión del protocolo utilizada.
    - En un Mensaje de Respuesta, la primera línea se denomina línea de estado y tiene el formato → **<Versión de HTTP> <Código de Estado> <Mensaje de Estado>.** Un ejemplo sería **HTTP/1.1 200 OK**.
        - **Versión de HTTP:** Indica la versión del protocolo usada.
        - **Código de Estado:** Código numérico que indica el resultado de la solicitud.
        - **Mensaje de Estado:** Mensaje explicativo del estado.
- Las cabeceras en HTTP proporcionan información adicional sobre la solicitud o la respuesta, como detalles sobre el cliente o servidor, tipo de contenido, codificaciones aceptadas, políticas de caché, autenticación, y más.

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

- El formato de las cabeceras/headers es → **<Nombre de la Cabecera>: <Valor de la Cabecera>.** Cada cabecera ocupa una línea y el bloque de cabeceras termina con una línea en blanco que 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.

- **El requerimiento quedaría:**
    - GET /index.html HTTP/1.1
      Host: www.misitio.com
      User-Agent: curl/7.74.0
      Accept: */*

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

- curl es una herramienta de línea de comandos utilizada para transferir datos desde o hacia un servidor utilizando distintos protocolos, incluyendo DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, MQTT, POP3, POP3S, RTMP,

RTMPS, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET y TFTP. Este comando está diseñado para trabajar sin interacción del usuario y es muy útil para hacer solicitudes HTTP. Algunos de sus parámetros posibles:

- **-I, –head**
  - El parámetro -I (equivalente a --head) se utiliza para realizar una solicitud HEAD al servidor en lugar de una solicitud GET. Esto significa que curl solo solicitará las cabeceras del recurso en lugar de todo el contenido. Si se usa en un archivo FTP o FILE, curl muestra el tamaño del archivo y la hora de la última modificación.
- **-H, –header <header/@file>**
  - El parámetro -H (equivalente a --header) se utiliza para añadir o modificar una cabecera HTTP en la solicitud. Se puede especificar cualquier número de encabezados adicionales. Se debe tener en cuenta que si se agrega un encabezado personalizado que tiene el mismo nombre que uno de los internos que usaría curl, se usará el encabezado establecido externamente en lugar del interno.
- **-X, –request <command>**
  - El parámetro -X (equivalente a --request) se utiliza para especificar el método HTTP que deseas usar en la solicitud, como GET, POST, PUT, DELETE, etc. Por defecto, curl utiliza el método GET, pero este parámetro te permite cambiarlo.
- **-s, –silent**
  - El parámetro -s (equivalente a --silent) se utiliza para suprimir la salida del progreso de la transferencia y otros mensajes que normalmente mostraría. Solo se muestra el resultado de la solicitud.

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.**

- Realicé un solo requerimiento usando el método GET y recibí un HTML.

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

- **Atributo href en tag link:**
  - La etiqueta <link> se utiliza principalmente para enlazar recursos externos que son necesarios para la presentación de la página web, como hojas de estilo CSS, iconos, fuentes, etc. El atributo href en esta etiqueta especifica la URL del recurso que se desea enlazar. Cuando el navegador encuentra una etiqueta <link> en el documento HTML, utiliza el valor del atributo href para cargar el recurso especificado.
- **Atributo href en tag img:**
  - En realidad, la etiqueta <img> no utiliza el atributo href. En su lugar, la etiqueta <img> utiliza el atributo src para especificar la URL de la

imagen que debe mostrarse en la página. Cuando el navegador encuentra una etiqueta <img>, carga la imagen desde la URL especificada en src y la inserta en la página en el lugar donde está ubicada la etiqueta <img>.

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

- No, no alcanzaría con un solo requerimiento, por cada recurso como pueden ser un archivo CSS, un archivo JavaScript, imágenes, etc. Se solicita por separado.

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.

- Se podrían realizar 8 o 9 requerimientos, **si la página posee favicon serían:**
  - 1 para el HTML.
  - 1 para el favicon.
  - 2 para CSS.
  - 2 para JavaScript.
  - 3 para las 3 imágenes.
- **Si la página no posee favicon:**
  - 1 para el HTML.
  - 2 para CSS.
  - 2 para JavaScript.
  - 3 para las 3 imágenes.
- Un navegador primero realiza un requerimiento HTTP para obtener el archivo HTML. Luego, analiza el contenido del HTML y realiza automáticamente requerimientos adicionales para cada recurso referenciado. El comando curl solo realiza el requerimiento del archivo base HTML, los demás requerimientos se tienen que hacer de forma manual.

9. Ejecute a continuación los siguientes comandos:

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

a. ¿Qué diferencias nota entre cada uno?

- **Diferencias:**
  - El primer comando hace una solicitud GET, mientras que el segundo hace una solicitud tipo HEAD.
  - El primer comando descarta el contenido de la respuesta al enviarlo a /dev/null, pero las cabeceras y detalles de la conexión se muestran. El segundo comando al tratarse de un HEAD , no se descarga el contenido de la página, solo las cabeceras se muestran.

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

- En el primer comando si se quita la redirección a /dev/null, curl mostrará tanto la información detallada de la conexión (debido a -v) como el contenido HTML de la página (debido a que es una solicitud GET sin redirección). En el segundo comando no es necesaria porque la solicitud HEAD ya no descarga el contenido HTML, solo las cabeceras, por lo que no hay contenido adicional que necesite ser redirigido o descartado.

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

- **Para el primer comando:**
  - En el requerimiento viajaron 3 cabeceras (Host, User-Agent y Accept).
  - En la respuesta viajaron 7 cabeceras (Date, Server, Last-Modified, ETag, Accept-Ranges, Content-Length y Content-Type).
- **Para el segundo comando:**
  - En el requerimiento viajaron 3 cabeceras (Host, User-Agent y Accept).
  - En la respuesta viajaron 7 cabeceras (Date, Server, Last-Modified, ETag, Accept-Ranges, Content-Length y Content-Type).

10. ¿Qué indica la cabecera Date?

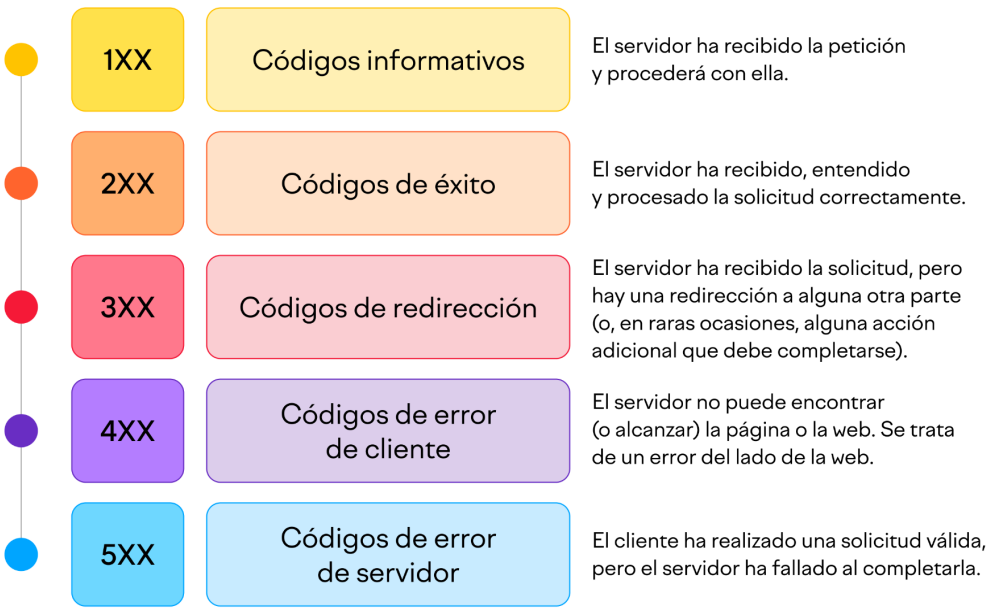
- La cabecera Date en una respuesta HTTP indica la fecha y la hora exactas en que el servidor generó y envió la respuesta al cliente. Esta fecha y hora se expresan en el formato GMT (Greenwich Mean Time).

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:**
  - El cliente puede saber que ha recibido todo el objeto solicitado de manera completa de las siguientes formas:
    - **Cierre de Conexión** → La forma más común de determinar el final de una respuesta es el cierre de la conexión TCP por parte del servidor. Una vez que el servidor ha enviado todo el contenido del objeto, cierra la conexión, lo que indica al cliente que la transferencia ha terminado.
    - **Cabecera Content-Length** → Alternativamente, el servidor puede incluir la cabecera Content-Length en la respuesta, que especifica el tamaño en bytes del contenido que se va a enviar. El cliente puede leer esta cabecera y, a medida que recibe datos, saber cuándo ha recibido el número exacto de bytes especificados, lo que le indica que ha recibido todo el contenido. Sin embargo, si no se incluye Content-Length, el cliente solo puede confiar en el cierre de la conexión.
- **En HTTP/1.1:**

- El cliente puede saber que ha recibido todo el objeto solicitado de manera completa de las siguientes formas:
  - **Transfer-Encoding: chunked** → En HTTP/1.1, se introdujo la codificación por bloques o chunked transfer encoding. Cuando un servidor usa Transfer-Encoding: chunked, divide el contenido en una serie de fragmentos (chunks), cada uno precedido por un encabezado que indica el tamaño de ese fragmento en hexadecimal. La transferencia termina cuando se envía un fragmento de tamaño cero, lo que indica al cliente que ha recibido todo el contenido.
  - **Cabecera Content-Length** → Al igual que en HTTP/1.0, Content-Length sigue siendo una opción válida. Si el servidor conoce el tamaño del contenido de antemano, puede incluir esta cabecera, y el cliente sabrá que la transferencia ha terminado cuando haya recibido el número exacto de bytes especificados.
  - **Conexiones Persistentes** → A diferencia de HTTP/1.0, donde la conexión generalmente se cierra después de cada solicitud/respuesta, HTTP/1.1 introduce conexiones persistentes por defecto, lo que permite reutilizar la misma conexión TCP para múltiples solicitudes/respuestas. En este caso, si no se usa Transfer-Encoding: chunked o Content-Length, se necesita algún otro mecanismo, como cerrar la conexión, para indicar el final del contenido, aunque esto es menos común en HTTP/1.1.

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).



13. **Utilizando curl, realice un requerimiento con el método HEAD al sitio www.redes.unlp.edu.ar e indique:**
    a. **¿Qué información brinda la primera línea de la respuesta?**

    - La primera línea de la respuesta (línea de estado) fue: **HTTP/1.1 200 OK.** Esta línea nos permite saber que el protocolo utilizado fue HTTP/1.1 y que el servidor ha encontrado y está enviando el objeto solicitado debido a que el código de estado fue 200 y el mensaje de estado fue OK.

    b. **¿Cuántos encabezados muestra la respuesta?**

    - En la respuesta viajaron 7 cabeceras (Date, Server, Last-Modified, ETag, Accept-Ranges, Content-Length y Content-Type).

    c. **¿Qué servidor web está sirviendo la página?**

    - El servidor que está sirviendo la página es Apache/2.4.56 (Unix).

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

    - El acceso a la página fue exitoso debido a que el código de estado fue 200 y el mensaje de estado fue OK.

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

    - La página se modificó por última vez el día 19 de marzo de 2023 a las 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 acceder a la página usando GET indicando que se la quiere obtener sólo si la misma fue modificada en una fecha posterior a la que efectivamente fue modificada se hace de esta forma → **curl -H "If-Modified-Since: Sun, 19 Mar 2023 19:04:46 GMT" www.redes.unlp.edu.ar.** No se obtuvo ningún resultado. El encabezado **If-Modified-Since: <fecha>** nos sirve para evitar la transferencia innecesaria de datos. Si el recurso no ha cambiado, el servidor responde con 304 Not Modified (si añadimos el parámetro -v al comando podemos ver esa respuesta) y no envía el contenido del recurso, ahorrando ancho de banda. Además, permite que los clientes (navegadores, aplicaciones) aprovechen cachés locales para evitar descargas repetitivas de contenido que no ha cambiado.

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.**

```
redes@debian:~/Desktop/ejercicios$ curl www.redes.unlp.edu.ar/restringido/index.php
<h1>Acceso restringido</h1>

<p>Para acceder al contenido es necesario autenticarse. Para obtener los datos de acceso seguir las instrucciones detalladas en www.red
es.unlp.edu.ar/obtener-usuario.php</p>
redes@debian:~/Desktop/ejercicios$ curl www.redes.unlp.edu.ar/obtener-usuario.php
<p>Para obtener el usuario y la contraseña haga un requerimiento a esta página seteando el encabezado 'Usuario-Redes' con el valor 'obt
ener'</p>
redes@debian:~/Desktop/ejercicios$ curl -H "Usuario-Redes: obtener" www.redes.unlp.edu.ar/obtener-usuario.php
<p>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!</p>
redes@debian:~/Desktop/ejercicios$ echo -n "redes:RYC" | base64
redes@debian:~/Desktop/ejercicios$ curl -v -H "Authorization: Basic cmVkZXM6UllD" www.redes.unlp.edu.ar/restringido/index.php
*   Trying 172.28.0.50:80...
* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)
> GET /restringido/index.php HTTP/1.1
> Host: www.redes.unlp.edu.ar
> User-Agent: curl/7.74.0
> Accept: */*
> Authorization: Basic cmVkZXM6UllD
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 Found
< Date: Wed, 04 Sep 2024 16:06:52 GMT
< Server: Apache/2.4.56 (Unix)
< X-Powered-By: PHP/7.4.33
< Location: http://www.redes.unlp.edu.ar/restringido/the-end.php
< Content-Length: 230
< Content-Type: text/html; charset=UTF-8
<
<h1>Excelente!</h1>

<p>Para terminar el ejercicio deberás agregar en la entrega los datos que se muestran en la siguiente página.</p>
<p>ACLARACIÓN: la URL de la siguiente página está contenida en esta misma respuesta.</p>

* Connection #0 to host www.redes.unlp.edu.ar left intact
redes@debian:~/Desktop/ejercicios$ curl -v -H "Authorization: Basic cmVkZXM6UllD" www.redes.unlp.edu.ar/restringido/the-end.php
*   Trying 172.28.0.50:80...
* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)
> GET /restringido/the-end.php HTTP/1.1
> Host: www.redes.unlp.edu.ar
> User-Agent: curl/7.74.0
> Accept: */*
> Authorization: Basic cmVkZXM6UllD
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Wed, 04 Sep 2024 16:09:39 GMT
< Server: Apache/2.4.56 (Unix)
< X-Powered-By: PHP/7.4.33
< Content-Length: 159
< Content-Type: text/html; charset=UTF-8
<
¡Felicitaciones, llegaste al final del ejercicio!

Fecha: 2024-09-04 16:09:39
* Connection #0 to host www.redes.unlp.edu.ar left intact
```

- El encabezado **Authorization** se usa en HTTP para enviar credenciales de autenticación al servidor. En el contexto de la autenticación básica (Basic Authentication), el encabezado Authorization incluye un token codificado en Base64 que contiene el nombre de usuario y la contraseña. El motivo por el cual las credenciales se codifican en base64 es principalmente para evitar problemas con caracteres especiales que podrían afectar la comunicación HTTP. Base64 es una

forma de representar datos binarios en una cadena de caracteres ASCII, lo que lo hace seguro para transmitir a través de HTTP.

- El encabezado **Location** en HTTP se usa en las respuestas del servidor para indicar la URL a la que el cliente debe redirigirse. Permite a los servidores notificar a los clientes sobre la nueva ubicación de recursos que se han movido. En aplicaciones web, se utiliza para controlar el flujo de navegación del usuario, redirigiendo a páginas relevantes después de acciones específicas.

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).

- **La salida después de ejecutar el comando fue:**
  - GET /http/HTTP-1.1/ HTTP/1.0
    User-Agent: curl/7.38.0
    Host: www.redes.unlp.edu.ar
    Accept: */*
    // SALTO DE LÍNEA //
    // SALTO DE LÍNEA //

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?

```
redes@debian:~/Desktop/ejercicios$ telnet www.redes.unlp.edu.ar 80
Trying 172.28.0.50...
Connected to www.redes.unlp.edu.ar.
Escape character is '^]'.
GET /http/HTTP-1.1/ HTTP/1.0
User-Agent: curl/7.38.0
Host: www.redes.unlp.edu.ar
Accept: */*


HTTP/1.1 200 OK
Date: Wed, 04 Sep 2024 17:45:49 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

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Protocolo HTTP: versiones</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="">
    <meta name="author" content="">
```

```
    <!-- Le styles -->
    <link href="../../bootstrap/css/bootstrap.css" rel="stylesheet">
    <link href="../../css/style.css" rel="stylesheet">
    <link href="../../bootstrap/css/bootstrap-responsive.css" rel="stylesheet">

    <!-- HTML5 shim, for IE6-8 support of HTML5 elements -->
    <!--[if lt IE 9]>
      <script src="./bootstrap/js/html5shiv.js"></script>
    <![endif]-->
  </head>

  <body>


    <div id="wrap">

    <div class="navbar navbar-inverse navbar-fixed-top">
      <div class="navbar-inner">
        <div class="container">
          <a class="brand" href="../../index.html"><i class="icon-home icon-white"></i></a>
          <a class="brand" href="https://catedras.info.unlp.edu.ar" target="_blank">Redes y Comunicaciones</a>
          <a class="brand" href="http://www.info.unlp.edu.ar" target="_blank">Facultad de Inform&aacute;tica</a>
          <a class="brand" href="http://www.unlp.edu.ar" target="_blank">UNLP</a>
        </div>
      </div>
    </div>

    <div class="container">
    <h1>Ejemplo del protocolo HTTP 1.1</h1>
    <p>
        Esta p&aacute;gina se visualiza utilizando HTTP 1.1. Utilizando el capturador de paquetes analice cuantos flujos utiliza el nav
egador para visualizar la p&aacute;gina con sus im&aacute;genes en contraposici&oacute;n con el protocolo HTTP/1.0.
    </p>
    </p>
    <h2>Imagen de ejemplo</h2>
    <img src="13532-tuxkiller03green.png" width="800px"/>
    </div>


    </div>
    <div id="footer">
      <div class="container">
        <p class="muted credit">Redes y Comunicaciones</p>
      </div>
    </div>
  </body>
</html>
Connection closed by foreign host.
```

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.

- La salida fue la misma, lo que cambió fue que la conexión se mantuvo persistente luego de que el servidor me haya dado la respuesta.

16. En base a lo obtenido en el ejercicio anterior, responda:

a. ¿Qué está haciendo al ejecutar el comando telnet?

- Al ejecutar el comando estamos intentando establecer una conexión con un servidor remoto en un puerto específico. El formato básico del comando es → **telnet <hostName> <port>.** En nuestro caso intentamos conectarnos al servidor en www.redes.unlp.edu.ar en el puerto 80.
  El protocolo **Telnet** permite que un usuario acceda a un servidor de manera interactiva a través de una sesión de texto, como si estuviera trabajando directamente en el servidor. Es una herramienta útil para conectarse a servicios en red y realizar pruebas de conectividad y diagnóstico, aunque su uso para aplicaciones en producción ha sido reemplazado en gran parte por herramientas más seguras como SSH que lo hacen con cifrado.

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

- Se utilizó el método HTTP GET y se solicitó el recurso /http/HTTP-1.1/.

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

- La diferencia estaba en la persistencia de la conexión, en el inciso b) al acceder con HTTP 1.0 cada solicitud y respuesta requería que la conexión se abra y se cierre, mientras que en el inciso c) al acceder con HTTP 1.1 la conexión se mantiene persistente haciendo que esa conexión pueda ser reutilizada para el envío y recepción de varios recursos.

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?

- Me parece más eficiente el caso de HTTP 1.1 ya que al persistir la conexión soluciona el problema de HTTP 1.0 que es el Overhead de aperturas y cierres de conexiones TCP para poder transportar los requerimientos. Aún así, el caso de HTTP 1.1 que persiste conexiones y hace uso de Pipelining puede traer un problema que es el Head Of Line Blocking, ya que si se solicita un objeto muy grande con un tiempo de obtención muy lento y luego se solicitan objetos más chicos y rápidos de obtención, al tener que mantener el orden de petición para la devolución de los objetos, el primero bloquea a los demás disminuyendo la eficiencia.

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”. Luego seleccione la interfaz correspondiente 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 caché 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.**

    - **Método GET:**

```
<!DOCTYPE html>
<html lang="en">
▶ <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>
```

    - **Método 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>
```

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

- **Método GET:**

GET /http/metodos-lectura-valores.php?form_nombre=Joaqu%C3%ACn+Manuel&form_apellido=Gonzalez&form_mail=mail%40gmail.com&form_sexo=sexo_masc&form_pass=123456&form_confirma_mail=on HTTP/1.1
Host: www.redes.unlp.edu.ar
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.redes.unlp.edu.ar/http/metodo-get.html
Upgrade-Insecure-Requests: 1

HTTP/1.1 200 OK
Date: Wed, 04 Sep 2024 18:36:54 GMT
Server: Apache/2.4.56 (Unix)
X-Powered-By: PHP/7.4.33
Content-Length: 2642
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Métodos HTTP: Lectura de valores desde REQUEST</title>

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="">
<meta name="author" content="">
<!-- Le styles -->
<link href="../bootstrap/css/bootstrap.css" rel="stylesheet">
<link href="../css/style.css" rel="stylesheet">
<link href="../bootstrap/css/bootstrap-responsive.css" rel="stylesheet">
<!-- HTML5 shim, for IE6-8 support of HTML5 elements -->
<!--[if lt IE 9]>
<script src="./bootstrap/js/html5shiv.js"></script>
<![endif]-->
</head>
<body>
<div id="wrap">
<div class="navbar navbar-inverse navbar-fixed-top">
<div class="navbar-inner">
<div class="container">
<a class="brand" href="../index.html"><i class="icon-home icon-white"></i></a>
<a class="brand" href="https://catedras.info.unlp.edu.ar" target="_blank">Redes y
Comunicaciones</a>
<a class="brand" href="http://www.info.unlp.edu.ar" target="_blank">Facultad de
Inform&aacute;tica</a>
<a class="brand" href="http://www.unlp.edu.ar" target="_blank">UNLP</a>
</div>
</div>
</div>
<div class="container">
<h1>M&eacute;todos HTTP: Lectura de valores desde REQUEST</h1>
<p>
&Eacute;sta p&aacute;gina es el resultado de leer los valores recibidos en un mensaje HTTP.
Desde el punto de vista del programador, puede obligarse la lectura seg&uacute;n el
m&eacute;todo sea <em>GET, POST u ambos</em>. En este caso, utilizamos el
&uacute;ltimo caso, es decir que este mismo c&oacute;digo sirve para leer valores enviados
utilizando GET o POST indistintamente.
<h2> Los valores recibidos son </h2>
<table border="0" width="80">
<tr>
<td nowrap>Nombre:</td><td>Joaqu..n Manuel </td>
</tr>
<tr>
<td nowrap>Apellido:</td><td>Gonzalez </td>
</tr>
<tr>
<td nowrap>Email:</td><td>mail@gmail.com </td>
</tr>
<tr>
```

<td nowrap>Sexo:</td><td>Masculino </td>
</tr>
<tr>
<td nowrap>Contraseña:</td><td>123456 </td>
</tr>
<tr>
<td nowrap>Recibir confirmaciones por email:</td><td>Si </td>
</tr>
</table>

Redes y Comunicaciones

</body>
</html>

- **Método POST:**

POST /http/metodos-lectura-valores.php HTTP/1.1
Host: www.redes.unlp.edu.ar
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 140
Origin: http://www.redes.unlp.edu.ar
Connection: keep-alive
Referer: http://www.redes.unlp.edu.ar/http/metodo-post.html
Upgrade-Insecure-Requests: 1
form_nombre=Joaqu%C3%ACn+Manuel&form_apellido=Gonzalez&form_mail=mail%40gmail.com&form_sexo=sexo_masc&form_pass=123456&form_confirma_mail=on

HTTP/1.1 200 OK
Date: Wed, 04 Sep 2024 18:46:24 GMT
Server: Apache/2.4.56 (Unix)
X-Powered-By: PHP/7.4.33
Content-Length: 2642
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
<!DOCTYPE html>
<html lang="en">

```
<head>
<meta charset="utf-8">
<title>M&eacute;todos HTTP: Lectura de valores desde REQUEST</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="">
<meta name="author" content="">
<!-- Le styles -->
<link href="../bootstrap/css/bootstrap.css" rel="stylesheet">
<link href="../css/style.css" rel="stylesheet">
<link href="../bootstrap/css/bootstrap-responsive.css" rel="stylesheet">
<!-- HTML5 shim, for IE6-8 support of HTML5 elements -->
<!--[if lt IE 9]>
<script src="./bootstrap/js/html5shiv.js"></script>
<![endif]-->
</head>
<body>
<div id="wrap">
<div class="navbar navbar-inverse navbar-fixed-top">
<div class="navbar-inner">
<div class="container">
<a class="brand" href="../index.html"><i class="icon-home icon-white"></i></a>
<a class="brand" href="https://catedras.info.unlp.edu.ar" target="_blank">Redes y
Comunicaciones</a>
<a class="brand" href="http://www.info.unlp.edu.ar" target="_blank">Facultad de
Inform&aacute;tica</a>
<a class="brand" href="http://www.unlp.edu.ar" target="_blank">UNLP</a>
</div>
</div>
</div>
<div class="container">
<h1>M&eacute;todos HTTP: Lectura de valores desde REQUEST</h1>
<p>
&Eacute;sta p&aacute;gina es el resultado de leer los valores recibidos en un mensaje HTTP.
Desde el punto de vista del programador, puede obligarse la lectura seg&uacute;n el
m&eacute;todo sea <em>GET, POST u ambos</em>. En este caso, utilizamos el
&uacute;ltimo caso, es decir que este mismo c&oacute;digo sirve para leer valores enviados
utilizando GET o POST indistintamente.
<h2> Los valores recibidos son </h2>
<table border="0" width="80">
<tr>
<td nowrap>Nombre:</td><td>Joaqu..n Manuel </td>
</tr>
<tr>
<td nowrap>Apellido:</td><td>Gonzalez </td>
</tr>
<tr>
```

```
<td nowrap>Email:</td><td>mail@gmail.com </td>
</tr>
<tr>
<td nowrap>Sexo:</td><td>Masculino </td>
</tr>
<tr>
<td nowrap>Contrase&ntilde;a:</td><td>123456 </td>
</tr>
<tr>
<td nowrap>Recibir confirmaciones por email:</td><td>Si </td>
</tr>
</table>
</p>
</div>
</div>
<div id="footer">
<div class="container">
<p class="muted credit">Redes y Comunicaciones</p>
</div>
</div>
</body>
</html>
```

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

- La diferencia que detecto es que si se utiliza el método GET la información del formulario viaja en los parámetros de la URL, mientras que con el método POST la información viaja en el cuerpo de la solicitud.

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

- Si, la diferencia es que con el método GET podemos ver la información ingresada en el formulario en los parámetros de la URL, mientras que con el método POST, no la vemos, solo cambia de URL a la página para visualizar los valores ingresados.

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.

- Dado que HTTP es un protocolo sin estado, las Cookies permiten que las aplicaciones web mantengan un estado persistente entre las solicitudes y las respuestas. Son esenciales para implementar la gestión de sesiones en aplicaciones web, a su vez, también se utilizan para almacenar preferencias del usuario, como la configuración de idioma o el tema de la página. Esto permite que las preferencias se mantengan entre sesiones y dispositivos.

La tecnología de las cookies utiliza cuatro componentes: (1) una línea de cabecera de la cookie en el mensaje de respuesta HTTP; (2) una línea de cabecera de la cookie en el mensaje de solicitud HTTP; (3) el archivo de cookies almacenado en el sistema terminal del usuario y gestionado por el navegador del usuario; y (4) una base de datos back-end en el sitio web.

- **Set-Cookie**
  - La cabecera Set-Cookie es enviada por el servidor en la respuesta HTTP para establecer una cookie en el navegador del cliente. Esta cookie se almacena en el cliente y será enviada de vuelta al servidor en solicitudes subsecuentes al mismo dominio.
  - **Formato** → Set-Cookie: <nombre>=<valor>; Expires=<fecha>; Path=<ruta>; Domain=<dominio>; Secure; HttpOnly.
    - **<nombre>=<valor>:** Define el nombre y valor de la cookie.
    - **Expires:** Indica la fecha de expiración de la cookie. Después de esta fecha, la cookie será eliminada.
    - **Path:** Especifica la ruta en la que la cookie es válida. Solo se enviará la cookie si la solicitud coincide con la ruta.
    - **Domain:** Define el dominio para el cual la cookie es válida.
    - **Secure:** Si se establece, la cookie solo se enviará a través de conexiones HTTPS.
    - **HttpOnly:** Indica que la cookie no está disponible para acceso mediante JavaScript, aumentando la seguridad contra ataques XSS.
- **Cookie**
  - La cabecera Cookie es enviada por el cliente (generalmente un navegador) en solicitudes HTTP subsecuentes para transmitir al servidor las cookies almacenadas que corresponden al dominio y la ruta de la solicitud.
  - **Formato** → Cookie: <nombre1>=<valor1>

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?

- **Protocolo Binario**
  - Utiliza un formato binario que no es legible por humanos para la comunicación. Los mensajes están compuestos de datos binarios que son más compactos y rápidos de procesar por las máquinas.
  - **Ventajas**
    - Mayor eficiencia en términos de tamaño de mensaje y velocidad de procesamiento.
    - Menor uso de ancho de banda debido a la compresión implícita en el formato binario.
    - Menor ambigüedad en la interpretación de los datos, ya que el formato es más rígido.
  - **Desventajas**
    - Más difícil de depurar y analizar sin herramientas especializadas.

        - Mayor complejidad en la implementación.
- **Protocolo basado en Texto**
    - Utiliza texto legible por humanos para la comunicación entre clientes y servidores. Los mensajes son cadenas de caracteres normalmente en ASCII o UTF-8 que se pueden leer y escribir directamente, lo que facilita la depuración y el análisis.
    - **Ventajas**
        - Facilidad de depuración y análisis de tráfico, ya que el contenido es legible.
        - Más fácil de implementar y comprender.
    - **Desventajas**
        - Generalmente, menos eficiente en términos de tamaño y velocidad, ya que los mensajes pueden ser más largos y requieren más procesamiento para analizar el texto.
- HTTP/1.0 y HTTP/1.1 son protocolos basados en texto mientras que HTTP/2 es un protocolo binario.

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).

- **Host en HTTP 1.0**
    - En HTTP 1.0, la cabecera Host no es obligatoria pero puede ser necesaria dependiendo de cómo el servidor web esté configurado. Esto significa que si la cabecera "Host" no estaba presente en una solicitud HTTP 1.0, el servidor asumiría que la solicitud estaba destinada al dominio asociado con la dirección IP del servidor.
- **Host en HTTP 1.1**
    - La cabecera Host en HTTP/1.1 especifica el nombre de dominio del servidor (y opcionalmente el número de puerto) al que el cliente desea enviar la solicitud. Es crucial para identificar cuál de los muchos sitios web alojados en un servidor debe responder a la solicitud. El uso de esta cabecera es obligatorio. Si la cabecera está ausente, el servidor responderá con un error 400 (Bad Request). Es sumamente importante su uso ya que muchos servidores web pueden alojar múltiples sitios web (hosting virtual) en una sola dirección IP. La cabecera Host permite al servidor diferenciar entre estos sitios y responder de manera apropiada según el dominio solicitado.
- **Host en HTTP 2**
    - HTTP/2 es un protocolo binario, pero sigue utilizando la misma estructura básica de solicitudes y respuestas HTTP. La cabecera Host sigue siendo relevante y se utiliza de manera similar a HTTP/1.1 para identificar el dominio solicitado. Sin embargo, en HTTP/2, la cabecera Host se envía como parte de los pseudo-headers. El pseudo-header

:authority es el que se utiliza para enviar la misma información que la cabecera Host en HTTP/1.1. Incluye el nombre del servidor y, si es necesario, el puerto. El uso de pseudo-headers permite a HTTP/2 ser más eficiente al tratar las cabeceras de manera binaria y optimizada.

b. En HTTP/1.1, ¿es correcto el siguiente requerimiento?

GET /index.php HTTP/1.1
User-Agent: curl/7.54.0

- No, es incorrecto ya que hace falta el uso de la cabecera obligatoria Host.

c. ¿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

- **El pedido quedaría:**
  - :method: GET
    :path: /index.php
    :scheme: https
    :authority: www.info.unlp.edu.ar
- **:method: GET:** Indica que se está realizando una solicitud GET.
- **:path: /index.php:** Indica el recurso que se está solicitando.
- **:scheme: https:** Indica que se está utilizando HTTPS como protocolo de comunicación.
- **:authority: www.info.unlp.edu.ar:** Es el equivalente a la cabecera Host en HTTP/1.1, especificando el servidor objetivo.

Ejercicio de Parcial

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?

- El servidor podría estar usando HTTP/1.1 porque veo que la respuesta incluye el encabezado Connection: close, es decir, no quiere que la conexión sea persistente, si fuera HTTP/1.0 esa especificación no sería necesaria.

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

- Está utilizando el método HEAD, no retorna el recurso completo ya que HEAD solo retorna los encabezados del recurso.

C. ¿Cuál es el recurso solicitado?

- El recurso solicitado es /metodos/.

D. ¿El método funcionó correctamente?

- Si, funcionó correctamente ya que el servidor retorno su respuesta con código de estado 200 y mensaje de estado 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?

- El servidor al comparar las fechas se daría cuenta que el recurso solicitado no se modificó por lo que su respuesta sería con un código de estado 304 Not Modified. El navegador lo que habría hecho sería utilizar la versión almacenada en caché del recurso, reduciendo el tiempo de carga y el uso de ancho de banda.