UNIVERSIDAD DE GRANADA

Tecnologías Web

3º Grado en Ingeniería Informática

Guión de prácticas
El protocolo HTTP

© Prof. Javier Martínez Baena
Dpto. Ciencias de la Computación e I. A.
Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial

1. Objetivo

Esta práctica tiene como objetivo aplicar los conocimientos adquiridos en relación con el protocolo HTTP como medio para el intercambio de información entre clientes (usualmente los navegadores web) y servidores web.

2. El protocolo HTTP

HTTP es un protocolo que se describe a nivel de aplicación en el modelo OSI por lo que es de muy alto nivel. Tiene como característica más destacada que es un protocolo sin estado (*stateless*), lo que significa que cada nuevo mensaje no depende de mensajes previos. Dicho de otra forma: cada mensaje es autocontenido y no depende de peticiones o respuestas previas entre cliente y servidor.

Cuando un cliente se conecta a un servidor web, el procedimiento es el siguiente:

1. El cliente le envía un mensaje al servidor pidiéndole un recurso (página web, imagen, datos, etc.).
2. El servidor web le responde con un mensaje que incluye, si procede, la información solicitada.

El protocolo HTTP es el que describe el formato de dichos mensajes.

2.1. Estructura de los mensajes

Los mensajes tienen un formato de texto plano y están orientados a líneas, es decir, cada línea incluye un elemento de información. La estructura es esta:

1. Primera línea. Indica la petición que se hace o el resultado de una respuesta.
2. Encabezados (*Headers*). Contienen metainformación relacionada con la información que se está enviando.
3. Línea en blanco. Indica la finalización del bloque de encabezados.
4. Cuerpo (*Body*). En caso de que sea necesario, se incluye aquí la información transmitida. Es opcional.

Además, se distinguen dos tipos de mensajes:

1. Mensaje de petición (*request message*). Es el que envía el cliente al servidor solicitándole un recurso.
2. Mensaje de respuesta (*response message*). Es el que envía el servidor al cliente con la respuesta.

Por ejemplo, si desde un navegador nos conectamos a la URL `http://void.ugr.es/tweb/hola.html` veríamos este resultado:

Para conseguirlo, el navegador analiza la URL que hemos escrito y construye un mensaje de petición HTTP como el siguiente:

```
GET /tweb/hola.html HTTP/1.1                    Primera línea
Host: void.ugr.es
User-Agent: Gecko/20100101 Firefox/88.0         Encabezados (headers)
Accept: */*
                                                Línea en blanco
```

A continuación establece una conexión TCP/IP con el servidor `void.ugr.es` y lo envía. Puede observar que en este mensaje:

- La primera línea indica que estamos solicitando (`GET`) una página (`/tweb/hola.html`) y que el mensaje que sigue usa la versión 1.1 del protocolo (`HTTP/1.1`).
- En los encabezados se añade la siguiente información:
  - Host al que estamos haciendo la solicitud (`void.ugr.es`).
  - Agente de usuario (`Gecko/20100101 Firefox/88.0`). Es una cadena de texto que identifica a la aplicación que está haciendo la solicitud. Cada aplicación (usualmente un navegador web) versión y sistema operativo se identifica mediante una cadena diferente.
  - Tipo de contenido que puede aceptar y procesar el cliente expresado como un tipo MIME (`*/*`). En este caso el cliente acepta cualquier tipo MIME.
- Tras los encabezados se envía una línea en blanco.
- No hay cuerpo (*body*).

Cuando el servidor recibe el mensaje lo interpreta y busca en su sistema de ficheros el documento solicitado (`/tweb/hola.html`). Si lo encuentra construye un mensaje de respuesta HTTP similar al siguiente:

```
HTTP/1.1 200 OK                                 Primera línea
Date: Thu, 03 Jun 2021 16:30:47 GMT
Server: Apache/2.4.46 (Ubuntu)
Last-Modified: Thu, 03 Jun 2021 16:07:23 GMT
Accept-Ranges: bytes                            Encabezados (headers)
Content-Length: 151
Vary: Accept-Encoding
Content-Type: text/html
                                                Línea en blanco
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Hola mundo</title>                     Cuerpo
</head>
<body>
  <h1>¡ Hola, mundo !</h1>
</body>
</html>
```

En este mensaje podemos observar que:

- La primera línea indica la versión del protocolo que se está usando (`HTTP/1.1`), un código que indica si la petición ha tenido éxito o no (`200`) y una cadena de texto que describe el código de error previo (`OK`).
- Se añaden encabezados para que el cliente tenga información adicional como, por ejemplo:
  - Fecha en la que se ha enviado la respuesta (`Thu, 03 Jun 2021 16:30:47 GMT`).
  - Cadena identificadora del servidor web (`Apache/2.4.46 (Ubuntu)`).

    - Fecha de la última modificación del fichero solicitado (`Thu, 03 Jun 2021 16:07:23 GMT`).
    - Longitud en bytes del fichero devuelto (`151`).
    - Tipo MIME del fichero devuelto (`text/hml`).
- A continuación hay una línea en blanco.
- En el cuerpo del mensaje se incrusta el documento HTML solicitado.

Finalmente, el servidor envía a través de una conexión TCP/IP el mensaje de respuesta a nuestro navegador (cliente) que es capaz de interpretarlo, extraer el código HTML y renderizarlo en una ventana.

3. Herramientas de análisis del tráfico HTTP

Para poder ver el contenido de estos mensajes tenemos varias posibilidades:

- Navegadores web
- Aplicaciones específicas para analizar el tráfico de red:
    - Con GUI.
    - En línea de órdenes (tcpdump, tshark, ngrep, cURL, ...).

3.1. Navegadores web

Los navegadores habituales disponen de herramientas de ayuda al desarrollo de páginas web. Entre otras, incluyen un módulo para ver y analizar el tráfico HTTP. Para activar este panel, deberá localizar algún ítem en los menús que esté relacionado con herramientas de desarrollo o depuración:

- Firefox: Menú → Desarrollador web → Herramientas para desarrollador.
- Chrome: Menú → Más herramientas → Herramientas para desarrolladores.
- Safari: Desarrollo → Mostrar inspector web.

La interfaz podrá variar de uno a otro y podría ofrecer opciones ligeramente distintas pero la funcionalidad básica para el análisis del tráfico HTTP está disponible en todos. A modo de ejemplo, en la siguiente figura puede ver una captura de pantalla de Firefox en la que puede ver información similar a la que se ha descrito en el ejemplo de la sección 2.1.

3.2. Aplicaciones específicas con GUI

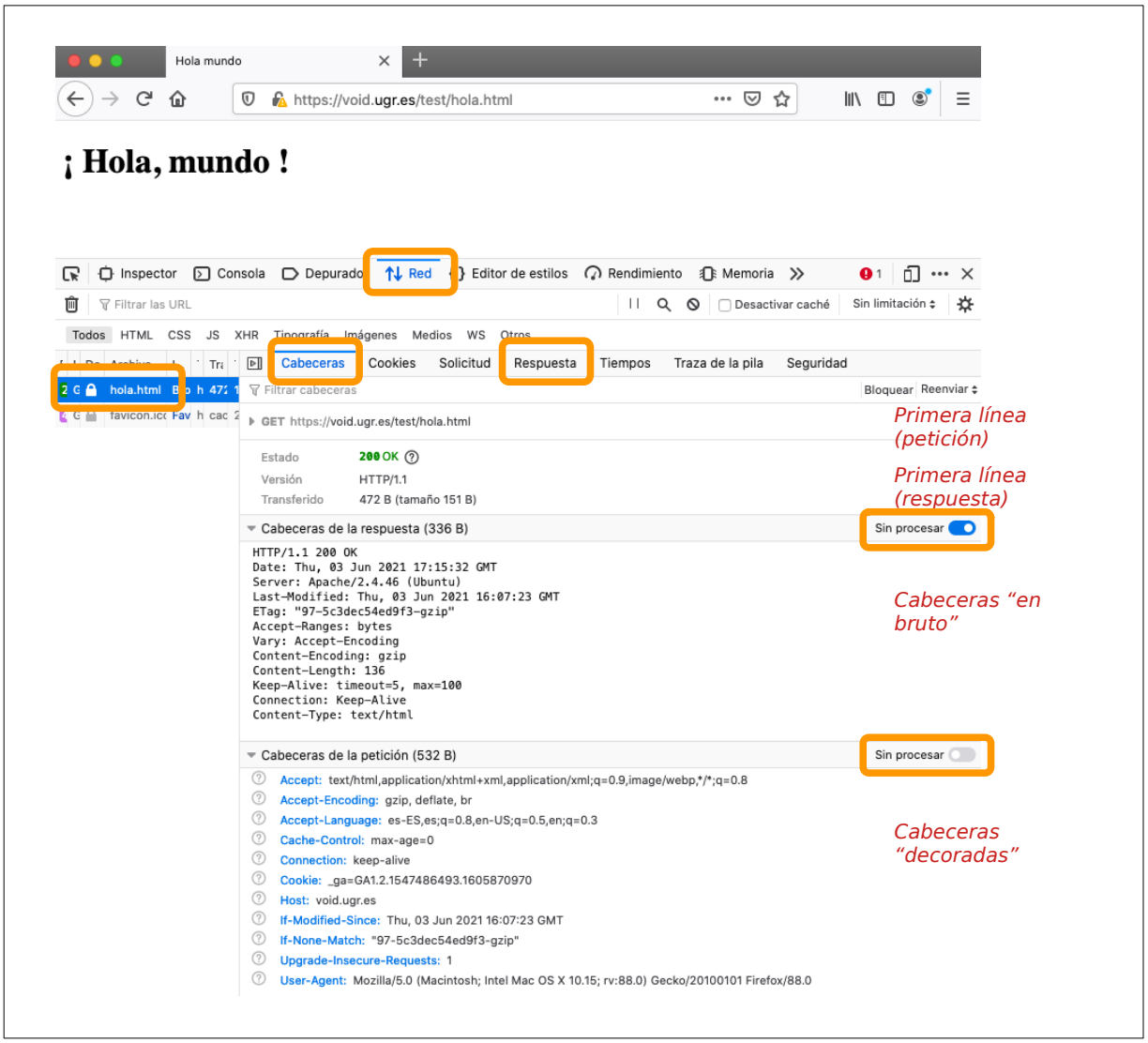
Existen multitud de aplicaciones que permiten realizar peticiones HTTP como si fuesen un navegador, pero con el objetivo de analizar el tráfico. Incluso permiten interceptar el tráfico generado por otras aplicaciones. Se recomiendan dos que son open source:

- HTTP Toolkit[1]. Es multiplataforma, *open source* y gratuito (dispone de una versión comercial con funcionalidad extra). Se comporta como un proxy de manera que, al pasar todo el tráfico a través de él, incluso permite modificarlo antes de que llegue a su destino.
- Wireshark[2]. Esta es una herramienta *open source* de uso genérico para el análisis del tráfico de red y, en particular, para el tráfico web.

Otras alternativas interesantes de pago o con modelos Freemium son estas:

- Fiddler Everywhere[3]. Es otro proxy para analizar el tráfico web multiplataforma y dispone de una versión gratuita y otras de pago. No es *open source*. Necesita registro de usuario.
- Burp Suite[4]. Es multiplataforma y dispone de una versión gratuita y otras de pago. No es *open source*.

1 https://httptoolkit.com/
2 https://www.wireshark.org/
3 https://www.telerik.com/fiddler/fiddler-everywhere
4 https://portswigger.net/burp/communitydownload



4. cURL: una herramienta en línea de órdenes

cURL es una herramienta de código abierto para transferir datos entre un cliente y un servidor que admite diversos protocolos (28 protocolos distintos). Además, permite mostrar los mensajes intercambiados y, por tanto, analizar el tráfico de red. En particular, cURL conoce el protocolo HTTP por lo que nos será de utilidad para realizar los ejercicios de este guión.

Además, cURL está disponible en 89 sistemas operativos y como biblioteca en multitud de lenguajes para usar de forma programática. Existen *bindings* para más de 60 entornos y lenguajes de programación diferentes (C++, Java, PHP, Python, Ruby, JavaScript/Node.js, ...)[5].

La instalación en un sistema Ubuntu es tan simple como instalar el paquete correspondiente:

```
sudo apt install curl
```

Más información en el sitio web de cURL: https://curl.se

4.1. Solicitar una página web (verbo GET)

Para hacer una petición web, usaremos la siguiente sintaxis:

```
curl http://void.ugr.es/tweb/hola.html
```

De esta forma, hacemos la petición y vemos en consola el resultado de la misma, es decir, el contenido del fichero solicitado. Si queremos ver también el contenido de los mensajes HTTP

5 https://daniel.haxx.se/blog/2022/11/25/89-operating-systems/

usaremos el modificador `-v`:

```
curl -v http://void.ugr.es/tweb/hola.html
```

Observe que en el modo *verbose* también se muestra información relativa al establecimiento de la conexión TCP/IP (y que no tiene que ver con los mensajes HTTP).

·▷ Ejercicio 1: Petición web con cURL

Ejecute las acciones indicadas e identifique y describa los mensajes HTTP así como sus diferentes partes (primera línea, encabezados, etc.).

4.1.1. Redirecciones

A veces, los servidores web están configurados para que al recibir determinadas peticiones estas sean redirigidas a otras ubicaciones de forma automática. El encargado de hacer esta redirección es el cliente y el flujo de mensajes es el siguiente:

1. Cliente: solicita URL A.
2. Servidor: al recibir la petición determina que debe redirigirse a la URL B y le envía esa información al cliente como respuesta.
3. Cliente: recibe la información de redirección y hace una nueva petición a la URL B.
4. Servidor: recibe la petición B y devuelve la respuesta.

·▷ Ejercicio 2: Redirecciones con cURL

Abra un navegador y escriba la URL `http://void.ugr.es` (observe que se ha escrito http y no https). Podrá ver una página típica por defecto de un servidor Apache y si observa la URL verá que el esquema es HTTPS (aunque usted solicitó HTTP). Si habilita la herramienta de análisis de tráfico HTTP del navegador verá que se han hecho dos peticiones. Esto ocurre porque en `void.ugr.es`, si se recibe una petición con el esquema HTTP, es redirigida automáticamente para que use el esquema HTTPS. Esto es transparente al usuario del navegador que lo gestiona de forma automática.

En caso de usar cURL, podrá comprobar que la redirección no se hace de forma automática. Ejecute:

```
curl -v http://void.ugr.es
```

Para que cURL haga también las redirecciones si fuesen necesarias debe usar el modificador `-L`:

```
curl -vL http://void.ugr.es
```

Ejecute las acciones indicadas con cURL y muestre los resultados. A continuación, hágalo también con el navegador e incluya capturas de pantalla de la herramienta de análisis de tráfico de red.

4.1.2. Encabezados

cURL permite construir las peticiones HTTP a medida, es decir, podemos añadir los encabezados que estimemos convenientes. Para ello se utiliza el modificador `-H`. En esta sección se ilustran algunos encabezados que pueden ser útiles en diferentes escenarios y que, además, ejemplifican algunos aspectos del protocolo HTTP.

Compresión de datos

Por defecto cURL no envía encabezados más allá de los imprescindibles. Es frecuente que tanto los clientes como los servidores web permitan comprimir la información que se intercambian para optimizar el uso del ancho de banda. El siguiente comando añade un encabezado a la petición que indica que este cliente admite información comprimida en formato *gzip*:

```
curl -v -H "Accept-Encoding: gzip" http://void.ugr.es/tweb/hola.html
```

El servidor, que también conoce como comprimir con *gzip*, comprime el cuerpo del mensaje. cURL, al detectar que es información binaria evita escribirla en consola salvo que lo forcemos a ello con el modificador `--output` con el que podemos mostrar los datos binarios en consola o escribirlos en un fichero:

```
curl -v -H "Accept-Encoding: gzip" http://void.ugr.es/tweb/hola.html --output -
curl -v -H "Accept-Encoding: gzip" http://void.ugr.es/tweb/hola.html --output fichero.html.gz
```

El fichero obtenido puede descomprimirse con:

```
gunzip fichero.html.gz
```

cURL, para el caso concreto de la compresión, dispone de un modificador `--compressed` que permite incluir automáticamente el encabezado anterior y, además, descomprime la información para que podamos verla en consola y que el proceso de descompresión sea transparente:

```
curl -v --compressed http://void.ugr.es/tweb/hola.html
```

·▷ Ejercicio 3: Añadiendo encabezados sobre compresión gzip con cURL

Ejecute las acciones indicadas y muestre los resultados.

Conexiones persistentes

Por defecto, en `HTTP/1.1` las conexiones cliente-servidor son persistentes. Esto quiere decir que si un mismo cliente hace varias peticiones consecutivas, se mantiene una misma conexión TCP/IP para todas ellas ahorrándose así los tiempos para establecer nuevas conexiones. Puede comprobarlo ejecutando:

```
curl -v http://void.ugr.es/tweb/hola.html http://void.ugr.es/tweb/hola.html
```

En la información que ofrece cURL podrá comprobar si se mantuvo o no la misma conexión TCP/IP para las dos solicitudes.

Para desactivar esa posibilidad, podemos añadir un encabezado en la petición del cliente:

```
curl -v -H "Connection: Close" http://void.ugr.es/tweb/hola.html
http://void.ugr.es/tweb/hola.html
```

Por contra, para indicarle de forma expresa al servidor que deseamos hacer uso de conexiones persistentes usaremos este otro valor para ese encabezado:

```
curl -v -H "Connection: Keep-Alive" http://void.ugr.es/tweb/hola.html
http://void.ugr.es/tweb/hola.html
```

·▷ Ejercicio 4: Añadiendo encabezados sobre conexiones persistentes con cURL

Ejecute las acciones indicadas y muestre los resultados. Marque la información que aporta cURL en relación con el uso o no de una conexión persistente.

Peticiones condicionales

HTTP también permite cambiar una respuesta condicionada a que ocurran o no determinadas circunstancias. Por ejemplo, suponga que desde su navegador hace una petición de una página en un momento dado y que, en un futuro, vuelve a solicitarla. Si la página no hubiese cambiado no haría falta transmitirla de nuevo ya que el navegador podría mantenerla en su caché y recuperarla desde allí siendo mucho más eficiente la operación.

Puede consultar algunos encabezados relativos a esto en
`https://developer.mozilla.org/en-US/docs/Web/HTTP/Conditional_requests`

4.2. Solicitud de encabezados (verbo HEAD)

En ocasiones no nos interesa solicitar un documento concreto sino únicamente averiguar algo sobre el servidor web (si está funcionando, si existe un documento, versión del software, etc.). Para esos casos HTTP provee un verbo denominado `HEAD`, que le dice al servidor que en el mensaje de respuesta omita el cuerpo. En la práctica funciona igual que una petición `GET` salvo por que la respuesta no tiene cuerpo. Para indicarle a cURL que use HEAD en lugar de GET se usa el modificador `-I`:

```
curl -I http://void.ugr.es/tweb/hola.html
```

Podrá ver que la respuesta es idéntica a la del ejercicio 2.1 salvo por la omisión del cuerpo.

·▷ Ejercicio 5: Averiguar el software que usa un servidor web

Averigüe qué servidor web utiliza cada uno de los sitios del siguiente listado. Para ello use el navegador y despliegue la ventana de herramientas de desarrollo. Para hacer este ejercicio debe tener en cuenta que:

- Algunas peticiones HTTP podrían redirigirle a HTTPS.
- Es frecuente que los servidores web estén detrás de algún proxy.
- Al solicitar una URL podrá comprobar que se hacen muchas conexiones para obtener todos los recursos asociados (imágenes, fichero .js, hojas de estilo, etc.) y podrá ver que en algunos sitios web la infraestructura puede ser bastante compleja.
- Algunas cabeceras que pueden ser de interés:
  - `Server`
  - `X-Served-By` (relacionada con servidores de caché).
  - `Via` (relacionada con proxy).

Puede complementar la información que le muestra su navegador con la que muestra este sitio web: https://www.wmtips.com/tools/info/

Lista de sitios a comprobar:

- `void.ugr.es`
- `www.ugr.es`
- `www.lamoncloa.gob.es`
- `es.wikipedia.org`
- `elpais.com`
- `www.elmundo.es`
- `www.medium.com`
- `www.slack.com`
- `www.whitehouse.gov`

4.3. Enviando datos al servidor (verbo POST)

HTTP se utiliza para pedir recursos a un servidor web pero también permite enviarle datos. Un caso típico se da cuando pedimos una URL que contiene un formulario: tras rellenarlo y pulsar un botón los datos se transfieren al servidor y este los procesa o almacena para usos futuros.

·▷ Ejercicio 6: Envío de datos en un formulario GET

Cree un fichero llamado `formulario.html` en su carpeta `public_html` del servidor `void.ugr.es` con el siguiente contenido:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Formulario</title>
</head>
<body>
  <form action="https://void.ugr.es/procesar.php" method="GET">
    <label>Nombre: <input type="text" name="nombre"></label>
    <label>Clave: <input type="password" name="password"></label>
    <input type="submit" name="enviar" value="Enviar">
  </form>
</body>
</html>
```

A continuación ábralo desde un navegador, rellene el formulario y pulse el botón de envío. Compruebe con las herramientas de desarrollo cómo se ha enviado la información del formulario al servidor. Al tratarse de un envío con el método `GET` los datos se envían codificados en la propia URL.

Para hacer esta misma operación con cURL basta con hacer la siguiente petición:

```
curl -v https://void.ugr.es/procesar.php?nombre=Pepito&password=laclave&enviar=Enviar
```

·▷ Ejercicio 7: Envío de datos en un formulario POST

Modifique el formulario y cambie el método de envío a "`POST`". A continuación repita el procedimiento del ejercicio anterior desde el navegador. Podrá comprobar que se ha modificado la forma en la que se envían los datos al servidor.

Para hacer una petición `POST` con cURL, debe usar el modificador `-d` de la siguiente forma:

```
curl -d nombre=Pepito -d password=laclave -d enviar=Enviar -v
https://void.ugr.es/procesar.php
```

o bien:

```
curl -d "nombre=Pepito&password=laclave&enviar=Enviar" -v https://void.ugr.es/procesar.php
```

De esa forma, los datos se adjuntan en el cuerpo de la petición.

Observe que cURL no muestra en la terminal el cuerpo del mensaje enviado, es decir, en el mensaje de petición solo aparecen las cabeceras seguidas de una línea en blanco pero no se ven en ninguna parte los datos enviados. Para mostrar esa información hay que añadir el modificador `--trace-ascii`, que muestra con más detalle la información intercambiada:

```
curl -d "nombre=Pepito&password=laclave&enviar=Enviar" --trace-ascii -
https://void.ugr.es/procesar.php
```

Observe el guión tras el modificador, sirve para que se muestre la salida en consola. Se podría poner un nombre de fichero en su lugar.

·▷ Ejercicio 8: Autenticación básica con cURL

En los dos ejercicios previos el script procesar.php está en una carpeta sin protección de `void.ugr.es`. Sin embargo, el documento `formulario.html` está en su *home* por lo que para poder acceder a él debe autenticarse. Copie el documento `procesar.php` en su carpeta `public_html` y modifique el formulario para que el `action` apunte hacia él.

Una vez autenticado en el navegador el funcionamiento será el mismo que antes. En cambio, si ahora intenta ejecutar el comando cURL para hacer el envío de datos obtendrá un error 401. Para incluir las credenciales en la ejecución de cURL debe usar el modificador `--user` de la siguiente forma:

```
curl --user usuario:clave -d "nombre=pepito&password=laclave&enviar=Enviar" -v
https://void.ugr.es/procesar.php
```

·▷ Ejercicio 9: Envío de ficheros con cURL

Cree un nuevo documento llamado `formulario2.html` con el siguiente contenido:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Formulario</title>
</head>
<body>
  <form action="https://void.ugr.es/procesar2.php" method="POST" enctype="multipart/form-data">
    <label>Nombre: <input type="text" name="nombre"></label>
    <label>Clave: <input type="password" name="password"></label>
    <label>Imagen: <input type="file" name="imagen"></label>
    <input type="submit" name="enviar" value="Enviar">
  </form>
```

```
</body>
</html>
```

Puede ver que se ha añadido un nuevo campo para enviar un fichero y se ha modificado la línea en la que se inicia el formulario. Pruébelo desde un navegador incluyendo un fichero de tipo imagen (jpg, png, gif, etc) en el nuevo campo y verifique que funciona correctamente.

Ahora los cambios sustanciales son dos:

- Por una parte se ha añadido un control para enviar un fichero. El fichero se incluirá en el cuerpo de la petición.
- Se ha incluido un atributo llamado `enctype` con un valor que permite incrustar información binaria en la petición (el fichero adjunto).

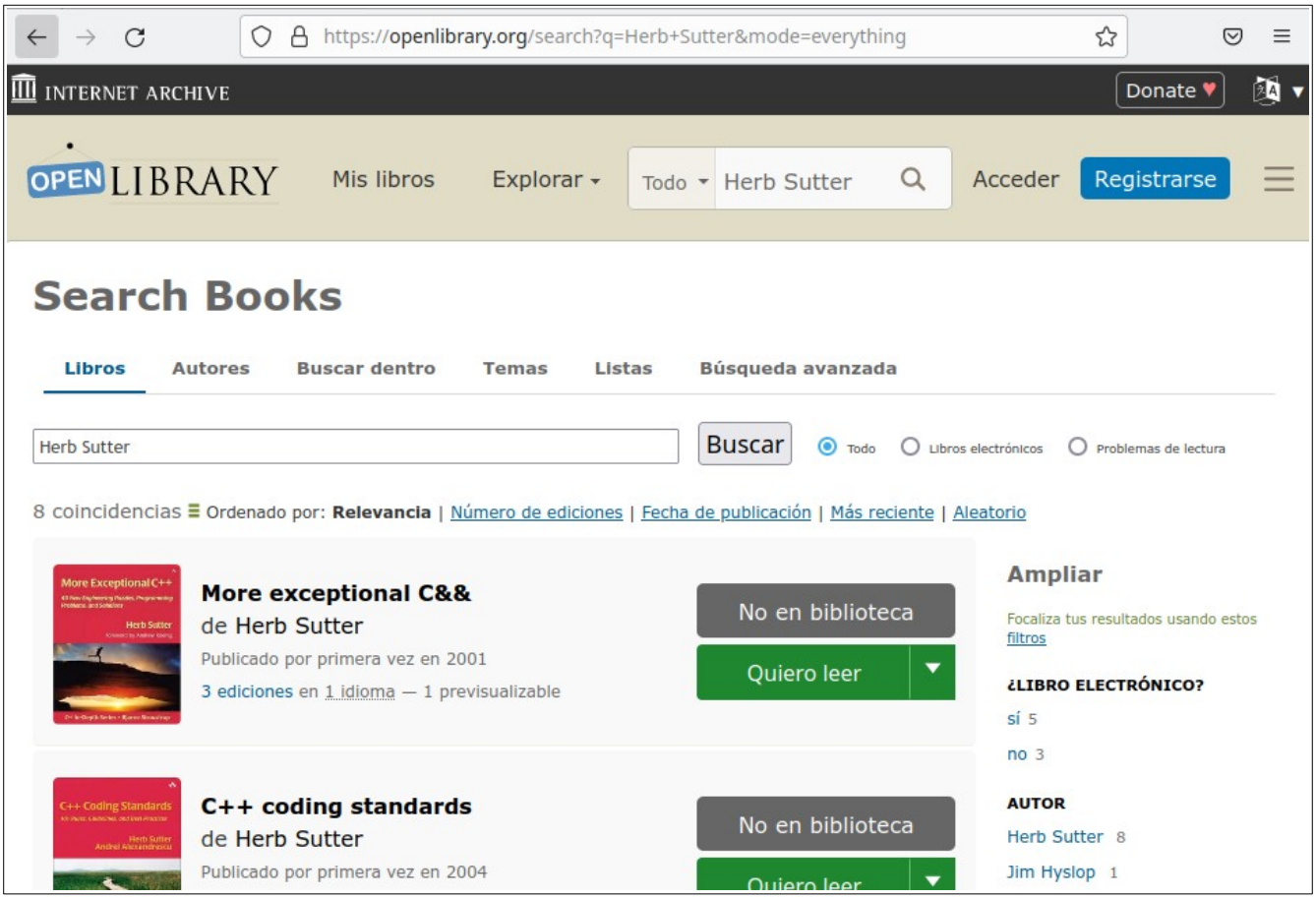
Si queremos hacer esta petición con cURL debemos usar un nuevo modificador `-F`, que es el que añade el comportamiento de `enctype="multipart/form-data"` a la petición permitiendo añadir información binaria al cuerpo. Suponiendo que el fichero que deseamos enviar al servidor se llama `tux.jpg` la petición sería esta:

```
curl -F "nombre=Pepito&password=laclave&enviar=Enviar" -F "imagen=@tux.jpg"
https://void.ugr.es/procesar2.php > resultado.html
```

En esta ocasión, además, hemos omitido el modificador `-v` para que solo nos muestre el resultado de la petición y hemos almacenado ese resultado en un fichero llamado `resultado.html` de forma que sea más fácil comprobar que todo ha funcionado bien. Si abrimos `resultado.html` en un navegador deberíamos ver la respuesta del servidor en la que se incluyen los valores y datos recibidos.

▷ Ejercicio 10: Servicios web

Conéctese a la página https://openlibrary.org y familiarízese con ella. Se trata de una biblioteca online. Busque libros del autor Herb Sutter introduciendo el nombre en la caja de búsqueda. Debería obtener un resultado similar al de la figura:



Observe que en la página de resultados, además del listado de libros y de mantener en la caja de búsqueda el nombre del autor, la caja de la URL tiene este texto:

> https://openlibrary.org/search?q=Herb+Sutter&mode=everything

Cuando se pulsó el botón para hacer la búsqueda se envió el texto a buscar en la propia URL y por eso ahora lo vemos así.

Si necesitásemos hacer una aplicación que tuviese que recuperar información bibliográfica sería interesante que esta pudiese hacer consultas programáticas a Open Library para obtener listados de libros. Sin embargo, el servidor de Open Library está pensado para acceder a él mediante un navegador. Podríamos hacer la petición web con herramientas como `cURL`:

```
curl -v https://openlibrary.org/search?q=Herb+Sutter
```

Hemos omitido la parte "`&mode=everything`" porque es irrelevante para nuestro propósito. Podrá ver en consola el código HTML de la página web de manera que con el software adecuado se podría obtener, por ejemplo, un listado de títulos en texto plano como este:

- More exceptional&&
- C++ coding standards
- Exceptional C++ style
- Exceptional C++
- ...

Esta forma de proceder no es particularmente eficiente ya que para obtener el listado de títulos hemos recuperado una página web con muchísima información y después hemos tenido que analizarla para extraer la parte que nos interesa. Sin embargo, hemos aprovechado la infraestructura web (protocolos, servidores, etc.) para hacer la petición de información automatizada, sin intervención humana: mi aplicación le ha solicitado al servidor la información.

Este es un caso típico en el que es interesante el uso de HTTP para solicitar información a un servidor que no es un documento HTML. Y cada vez es más frecuente que unas máquinas (clientes) soliciten información a otras máquinas (servidores) para hacer alguna tarea automática. En estos casos diremos que los clientes están accediendo a lo que se conoce como servicios web. Estos siguen siendo aplicaciones alojadas en servidores web pero que están pensadas para el intercambio de datos entre máquinas en lugar de para enviar páginas web a navegadores. Los datos suelen tener formatos estandarizados como, por ejemplo, JSON.

En el caso de Open Library además de tener la aplicación web que hemos usado, y que está pensada para que una persona haga búsquedas obteniendo páginas HTML como resultado, también disponemos de un servicio web diseñado para hacer solicitudes de información en línea con lo que se ha explicado. Para ello Open Library define una API (RESTful API) que establece la forma en la que se deben realizar las peticiones y la forma en la que responderá. En la web https://openlibrary.org/developers/api tiene información sobre ella. Verá que dispone de varias API, cada una orientada a una tarea distinta. La API que permite hacer búsquedas en la biblioteca es https://openlibrary.org/dev/docs/api/search y vemos que dispone de un parámetro `q` para hacer la consulta. Por ejemplo:

> http://openlibrary.org/search.json?q=Herb+Sutter

Tras hacer esa petición el servidor devuelve información en formato JSON en lugar de HTML con los registros que ha recuperado. Podemos hacer la petición con `cURL`:

```
curl -v http://openlibrary.org/search.json?q=Herb+Sutter
```

En internet dispone de multitud de servicios web, muchos de ellos de pago. Uno de libre acceso es Music Brainz que dispone de un amplio catálogo musical: https://musicbrainz.org.

En este ejercicio deberá averiguar cómo usar la API de Music Brainz para obtener la lista de grabaciones (solo álbums) de un grupo o artista determinado. El dato del que partimos es el nombre del grupo o artista y el resultado final será la lista de álbums. Por ejemplo, si el dato de entrada es "Bruce Springsteen" el resultado contendrá la lista siguiente:

- Greetings From Asbury Park, N.J.

- The Wild, the Innocent & The E Street Shuffle
- Born to Run
- Darkness on the Edge of Town
- The River
- Nebraska
- Born in the U.S.A.
- Tunnel of Love
- Human Touch
- Lucky Town
- The Ghost of Tom Joad
- The Rising
- Devils & Dust
- We Shall Overcome: The Seeger Sessions
- Magic
- Working on a Dream
- The Promise
- Wrecking Ball
- High Hopes
- Western Stars
- Letter to You
- Only the Strong Survive

El resultado será un documento JSON que podría incluir algún campo más además del título del álbum (año de grabación, etc.). No hace falta automatizar el proceso completo, solo ejemplificar qué llamadas hay que hacer a la API con `cURL`.

5. Herramientas de captura de tráfico de red

Existen muchas herramientas para el análisis del tráfico de red. Lo que hemos hecho hasta ahora con cURL es sustituir al navegador para hacer las peticiones web y así poder tener el máximo control sobre los encabezados de nuestros mensajes. Sin embargo, esta forma de proceder puede ser difícil de manejar si estamos probando una aplicación web ya que no siempre es fácil hacer desde línea de órdenes todo lo que hacemos desde un navegador.

Existen herramientas que permiten interceptar el tráfico de red que pasa por una determinada interfaz de red, es decir, que permiten monitorizar lo que otras aplicaciones hacen a través de la red. De esta forma, podemos usar el navegador para trabajar con una aplicación web y usar una de estas herramientas para que supervise todo el tráfico de forma independiente.

Hay herramientas con interfaz gráfica como HTTP Toolkit o Wireshark y otras que se pueden usar en línea de órdenes como tcpdump o ngrep.

·▷ Ejercicio 11: Monitorizar el tráfico de red

Una de estas herramientas básicas es tcpdump. Mire la documentación de este software y averigüe cómo usarlo para capturar los mensajes HTTP entre su máquina y `void.ugr.es`. Para ello, desde un navegador Firefox, solicite las URL que se indican a continuación y use tcpdump para ver el tráfico HTTP:

- `http://void.ugr.es/tweb/hola.html`
- `https://void.ugr.es/tweb/hola.html`

La diferencia entre ambas URL es el esquema utilizado ¿Puede ver el contenido del documento html en alguno de los casos anteriores? Tenga en cuenta que es muy probable que el navegador esté recibiendo el cuerpo del mensaje comprimido con gzip u otro algoritmo, en cuyo caso deberá indicarle a su navegador que no acepte datos comprimidos. Deberá configurarlo de esta forma:

1. Escriba `about:config` en la caja de la URL. Podrá ver multitud de parámetros de configuración del navegador. Algunos de ellos son de bajo nivel así que tenga cuidado con lo que modifica.
2. Busque uno llamado `network.http.accept-encoding`. En él se indica el contenido que Firefox enviará en la cabecera `accept-encoding`.

3. Modifíquelo para que no acepte tráfico de red comprimido.
4. Cuando acabe el ejercicio recuerde restablecer este parámetro a su valor original.

Ejemplos de uso de tcpdump:

- `sudo tcpdump -A 'host void.ugr.es'`
- `sudo tcpdump -A -s 10240 'tcp port 443 and (((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)' -vv`
- `sudo tcpdump -A -s 10240 'host void.ugr.es and (((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)' -vv`

A continuación, instale Wireshark y úselo para hacer la misma tarea con este filtro:

```
ip.addr==150.214.190.100 and (tcp.port==80 or tcp.port==443)
```

6. Entrega de la práctica

El profesor dará instrucciones concretas sobre lo que hay que incluir en la entrega de esta práctica así como los plazos para ello.

7. Enlaces y bibliografía

- David Gourley and Brian Totty. “HTTP. The Definitive Guide”. O’Reilly. 2002.
- https://developer.mozilla.org/en-US/docs/Web/HTTP
- https://everything.curl.dev
- https://curl.se/docs/manpage.html

Software:

- https://curl.se
- https://httptoolkit.tech
- https://www.telerik.com/fiddler/fiddler-everywhere
- https://portswigger.net/burp/communitydownload
- https://www.tcpdump.org
- https://github.com/jpr5/ngrep/
- https://www.wireshark.org/

Otros:

- https://reqbin.com/curl
- https://www.middlewareinventory.com/blog/tcpdump-capture-http-get-post-requests-apache-weblogic-websphere/