

INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Práctica 4: Servidor HTTP

Unidad de aprendizaje: Aplicaciones para comunicaciones en red

Grupo: 3CM7

Alumnos:

- Ontiveros Salazar Alan Enrique
- Sánchez Valencia Sergio Gabriel

Profesor:

Moreno Cervantes Axel Ernesto

3 de diciembre de 2018

Práctica 4: Servidor HTTP

3CM7

3 de diciembre de 2018

1. Marco teórico

1.1. Protocolo HTTP

El *Protocolo de Transferencia de Hipertexto (HTTP)* es un protocolo en la capa de aplicación para transmitir documentos de hipermedia, tales como páginas HTML. Se diseñó para la comunicación entre navegadores Web y servidores, pero se puede usar para otros objetivos. HTTP sigue el clásico modelo cliente-servidor, en donde el cliente abre la conexión para realizar una solicitud, y espera hasta que recibe una respuesta del servidor. HTTP es un protocolo sin estado, lo que significa que el servidor no guarda ninguna información (estado) entre las solicitudes. A pesar de que está basado en la capa de transporte TCP/IP, se puede usar en cualquier otra capa de transporte, como UDP.

1.2. Estructura del mensaje HTTP

Un *mensaje HTTP* es la forma de intercambiar datos entre un servidor y un cliente. Hay dos tipos: solicitudes (*requests*) enviadas por el cliente para realizar una acción en el servidor; y respuestas (*responses*) enviadas desde el servidor al cliente.

Los mensajes están compuestos de información textual codificada en ASCII, que ocupa varias líneas. En la versión HTTP/1.1 y anteriores, estos mensajes se envían totalmente a través de la conexión. En HTTP/2, estos se dividen en tramas HTTP, optimizando más. En esta práctica solo implementaremos HTTP/1.1.

Los webmasters y desarrolladores raramente manejan desde cero estos mensajes HTTP, ya que algún software, navegador, proxy o servidor realiza esta acción.

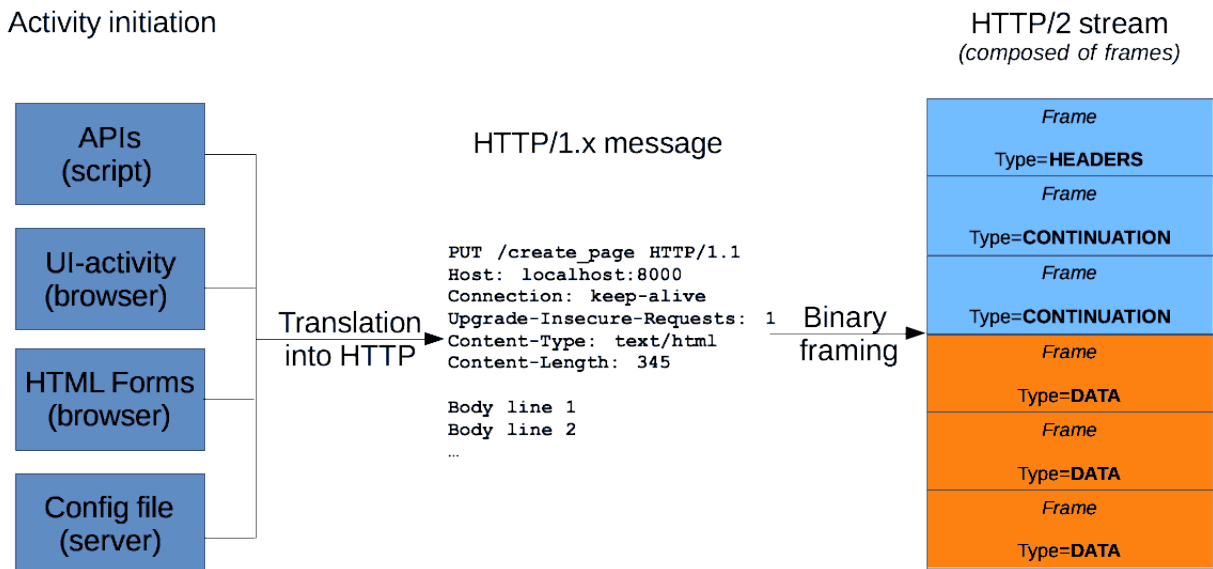


Figura 1: Ejemplo general de la estructura de un mensaje HTTP

Las solicitudes y respuestas HTTP siguen un formato de estructura similar entre sí, compuesto de:

1. Una sola línea de comienzo que describe la solicitud a realizar, o el estado de si una solicitud fue exitosa o falló.
2. Un conjunto opcional de cabeceras (*headers*) HTTP especificando más a detalle la solicitud, o describiendo el cuerpo del mensaje en la respuesta.
3. Una línea en blanco que indica que toda la información previa (*meta-information*) se ha enviado. Estas líneas usan los caracteres `\ r \ n` como separador.
4. Un cuerpo (*body*) opcional, que contiene los datos asociados a la solicitud (como el contenido de un formulario HTML), o el documento asociado a la respuesta. La presencia del cuerpo y su tamaño se indican con la línea de inicio y los encabezados.

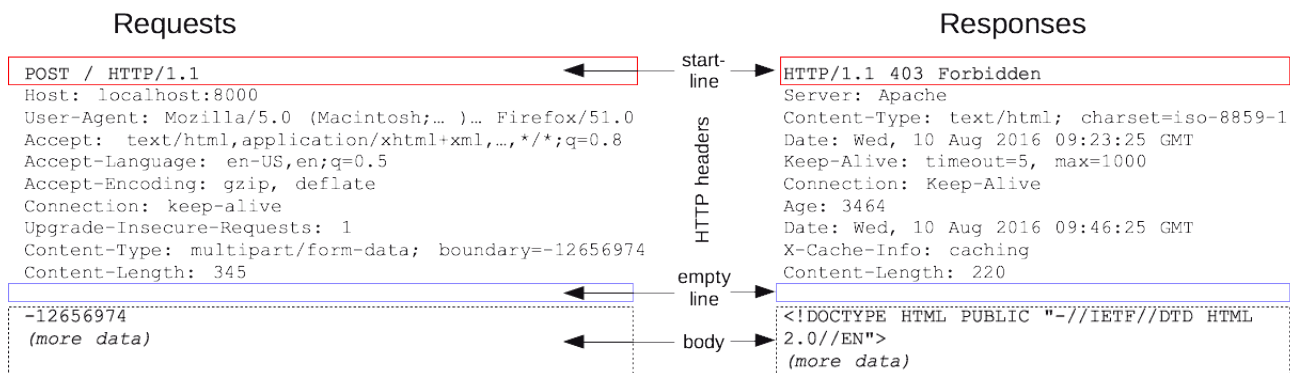


Figura 2: Ejemplo de una solicitud y una respuesta

1.3. Solicitudes

Son mensajes enviados por el cliente para iniciar una acción en el servidor.

1.3.1. Primer línea

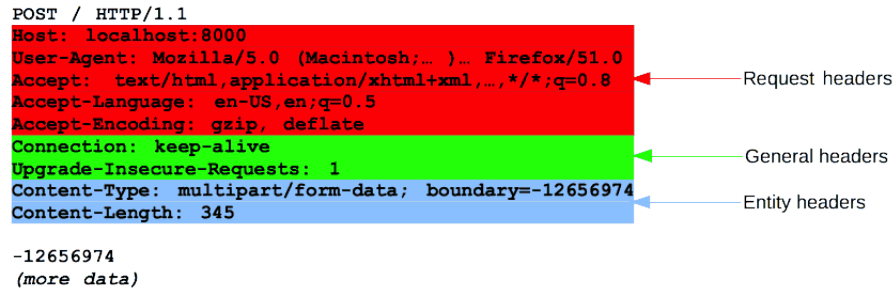
La primer línea contiene tres elementos:

1. El *método HTTP*, que es un verbo que puede ser GET, POST, PUT o DELETE; o un sustantivo, como HEAD o OPTIONS, que describe la acción a realizarse. Por ejemplo, GET indica que se debe devolver un recurso, y POST que se debe enviar datos al servidor.
2. La *URL* de destino o la ruta absoluta del protocolo, puerto y dominio. El formato puede variar:
 - Una ruta absoluta, opcionalmente seguida de un ? y un *query string*. Es la forma más común, y se usa con GET, POST, HEAD y PUT. Ejemplos:
 - POST / HTTP/1.1
 - GET /background.png HTTP/1.0
 - HEAD /test.html?query=alibaba HTTP/1.1
 - OPTIONS /anypage.html HTTP/1.0
 - Una URL completa, se usa comúnmente con un proxy. Ejemplo: GET http://developer.mozilla.org/en-US/docs/ HTTP/1.1
3. La versión de HTTP, usualmente HTTP/1.1.

1.3.2. Encabezados

Son elementos del tipo **clave: valor**. Permiten mandar información adicional con la solicitud. Hay varios tipos:

1. Generales: aplican al mensaje como un todo.
2. Encabezados de solicitud: modifican la solicitud. Algunos ejemplos son **User-Agent**, **Accept-Language**, **Referer**, etc.
3. Encabezados de entidad: aplican al cuerpo del mensaje, como **Content-Length**.



```

POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;... )... Firefox/51.0
Accept: text/html,application/xhtml+xml,.../*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
  
```

El diagrama muestra un mensaje HTTP POST con los siguientes encabezados coloreados y etiquetados:

- Request headers (rojo):** Host: localhost:8000, User-Agent: Mozilla/5.0 (Macintosh;...)... Firefox/51.0, Accept: text/html,application/xhtml+xml,.../*/*;q=0.8, Accept-Language: en-US,en;q=0.5, Accept-Encoding: gzip, deflate.
- General headers (verde):** Connection: keep-alive, Upgrade-Insecure-Requests: 1.
- Entity headers (azul):** Content-Type: multipart/form-data; boundary=-12656974, Content-Length: 345.

El cuerpo del mensaje comienza con `-12656974` y `(more data)`.

Figura 3: Ejemplo de los encabezados más comunes

1.3.3. Cuerpo

Es la parte final de la solicitud. No todas las solicitudes lo tienen, tales como las que solo solicitan recursos, como **GET**, **HEAD**, **DELETE** o **OPTIONS**. Algunas solicitudes envían datos al servidor para actualizarlo, como es el caso de **POST**. Los cuerpos de las solicitudes se clasifican en:

- De recurso único: consisten de un solo archivo, definido con los encabezados **Content-Type** y **Content-Length**.
- De múltiples recursos: consisten de un cuerpo multiparte, cada uno conteniendo una parte diferente de la información. Usualmente se asocia con los formularios **HTML**.

1.4. Respuestas

1.4.1. Línea de estado

La primera línea de una respuesta HTTP contiene la siguiente información:

- La versión del protocolo HTTP, usualmente **HTTP/1.1**.
- El código numérico de estado, indicando si la solicitud fue exitosa o no. Los más comunes son **200 (OK)**, **404 (Not Found)**, **302 (Found)** o **500 (Internal Server Error)**.

1.4.2. Encabezados

Tienen el mismo formato que los encabezados de solicitud. Se dividen en:

1. Generales: aplican a todo el mensaje.
2. De respuesta: proporcionan información adicional sobre el servidor que no caben en la primera línea, como **Vary** o **Accept-Ranges**.
3. De entidad: aplican al cuerpo del mensaje, tales como **Content-Length**.

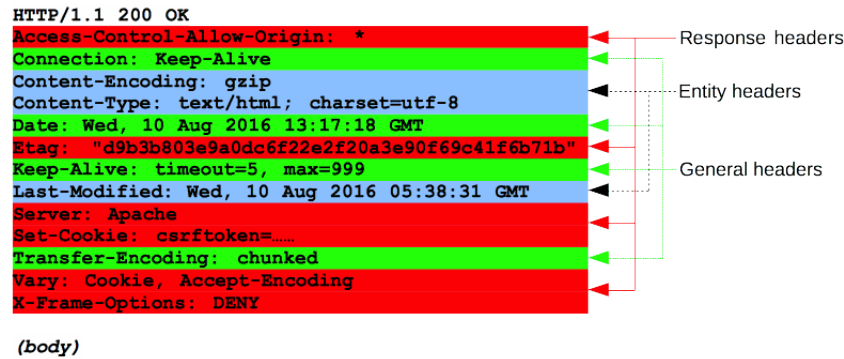


Figura 4: Ejemplo de una respuesta HTTP

1.4.3. Cuerpo

Es la última parte de la respuesta. No todas tienen uno, por ejemplo, las que tienen como código de error 201 o 204. Se clasifican en:

- De un solo recurso: consisten de un solo archivo de longitud conocida, definido en los encabezados `Content-Type` y `Content-Length`.
- De un solo recurso con longitud desconocida: consisten de un solo archivo pero no se sabe su tamaño, se usa el encabezado `Transfer-Encoding: chunked`.
- De múltiples recursos: consisten en varios archivos, cada uno contiene distinta información. Casi no es común este tipo.

2. Desarrollo

En esta práctica implementaremos un servidor HTTP 1.1 sencillo corriendo sobre el puerto 8888, que soporte los métodos GET, POST, HEAD, PUT y DELETE. Soportará los encabezados más sencillos, tales como `Content-Length`, `Content-Type`, `Date` y `WWW-Authenticate`.

Tendrá como códigos de error los más comunes: 404 (Not Found), 403 (Forbidden), 200 (OK), 401 (Unauthorized) y 201 (Created).

Soportará el examen de directorios con previa autenticación, así como el borrado y subida de archivos (no se permite borrado de directorios).

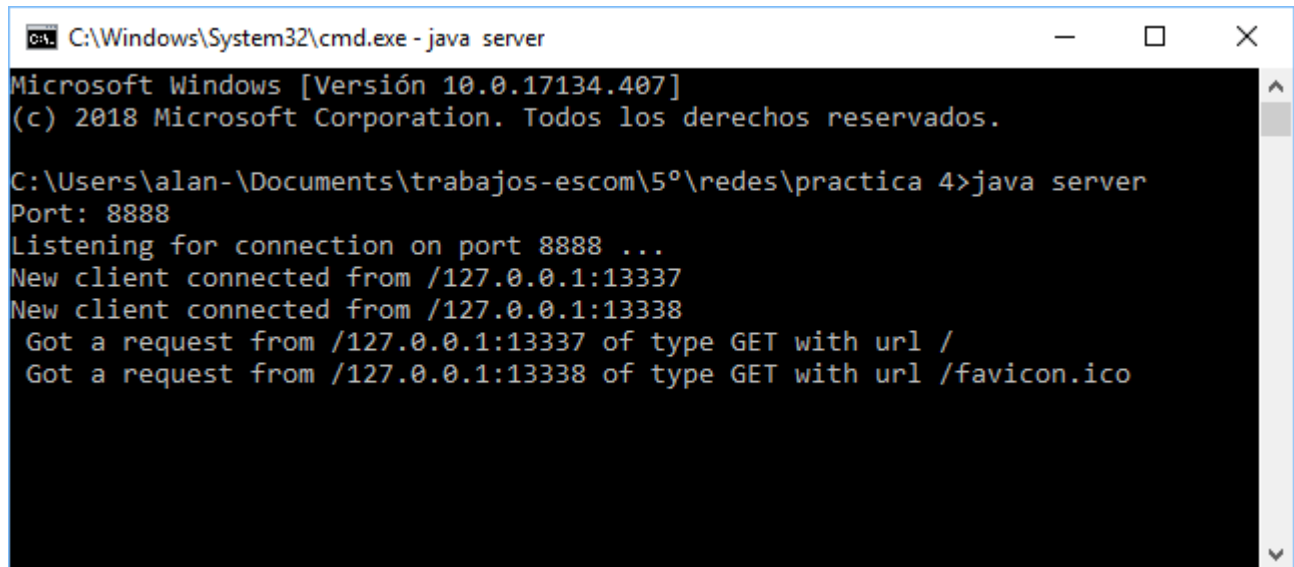
Además, para solicitudes GET y POST, las páginas `.html` podrán contener al estilo PHP las expresiones `<?_GET["key"]?>` y `<?_POST["key"]?>`, las cuales el servidor se encargará de sustituir con los valores en la solicitud del mensaje.

El documento predeterminado para carpeta será `index.html`.

Por último, tendremos una alberca de hilos con capacidad de 3 cada vez que se conecte un cliente, esto con el fin de no saturar el servidor.

3. Pruebas

Corremos el servidor:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\System32\cmd.exe - java server'. The window has standard Windows window controls (minimize, maximize, close). The command prompt shows the following text:

```
Microsoft Windows [Versión 10.0.17134.407]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\alan-\Documents\trabajos-escom\5º\redes\practica 4>java server
Port: 8888
Listening for connection on port 8888 ...
New client connected from /127.0.0.1:13337
New client connected from /127.0.0.1:13338
  Got a request from /127.0.0.1:13337 of type GET with url /
  Got a request from /127.0.0.1:13338 of type GET with url /favicon.ico
```

Abrimos un navegador y solicitamos la página principal:



The screenshot shows a web browser window with the title 'Página principal'. The address bar displays '127.0.0.1:8888'. The browser's toolbar includes various icons for navigation and extensions. The main content area of the browser displays the following:

BIENVENIDO

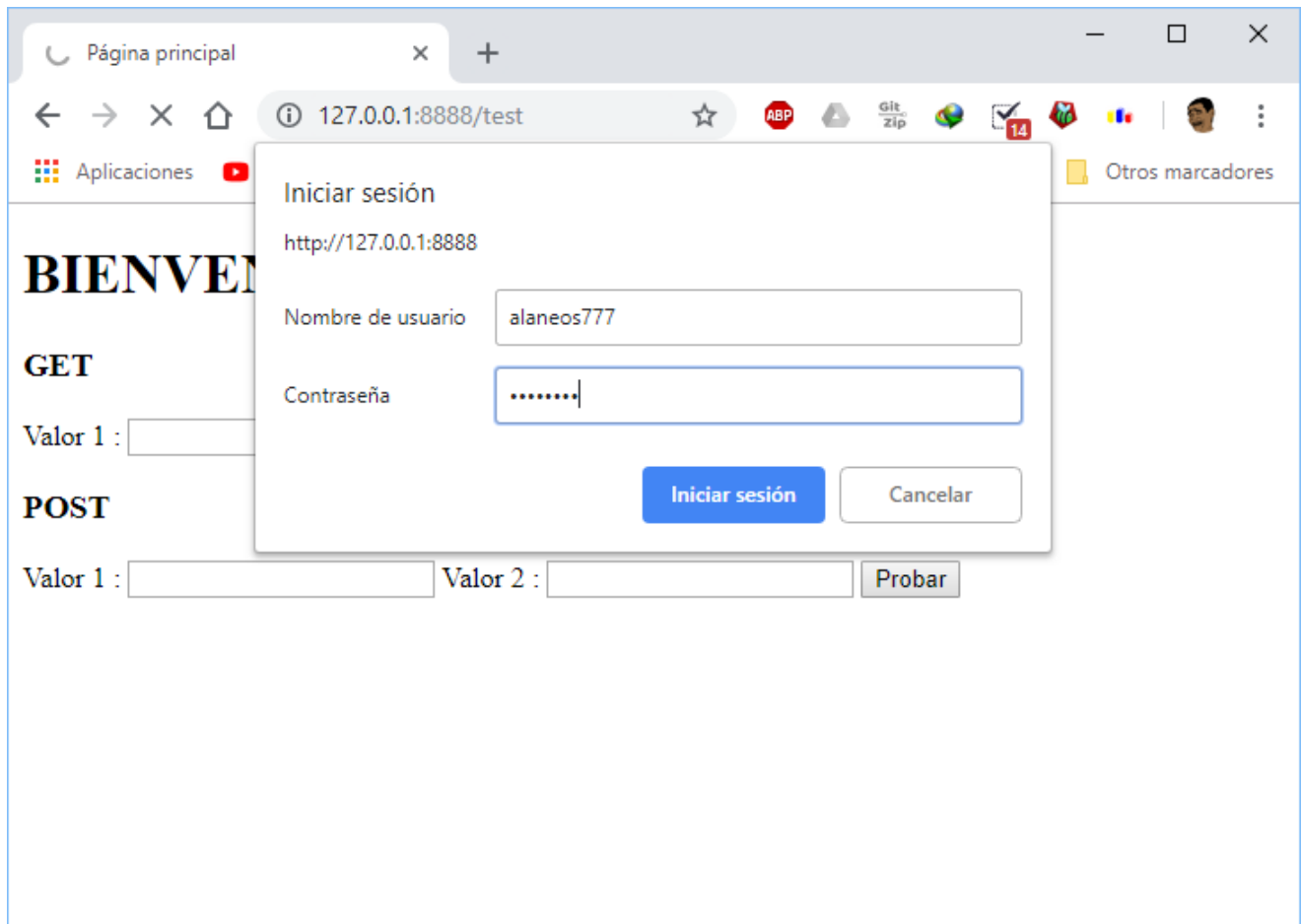
GET

Valor 1 : Valor 2 :

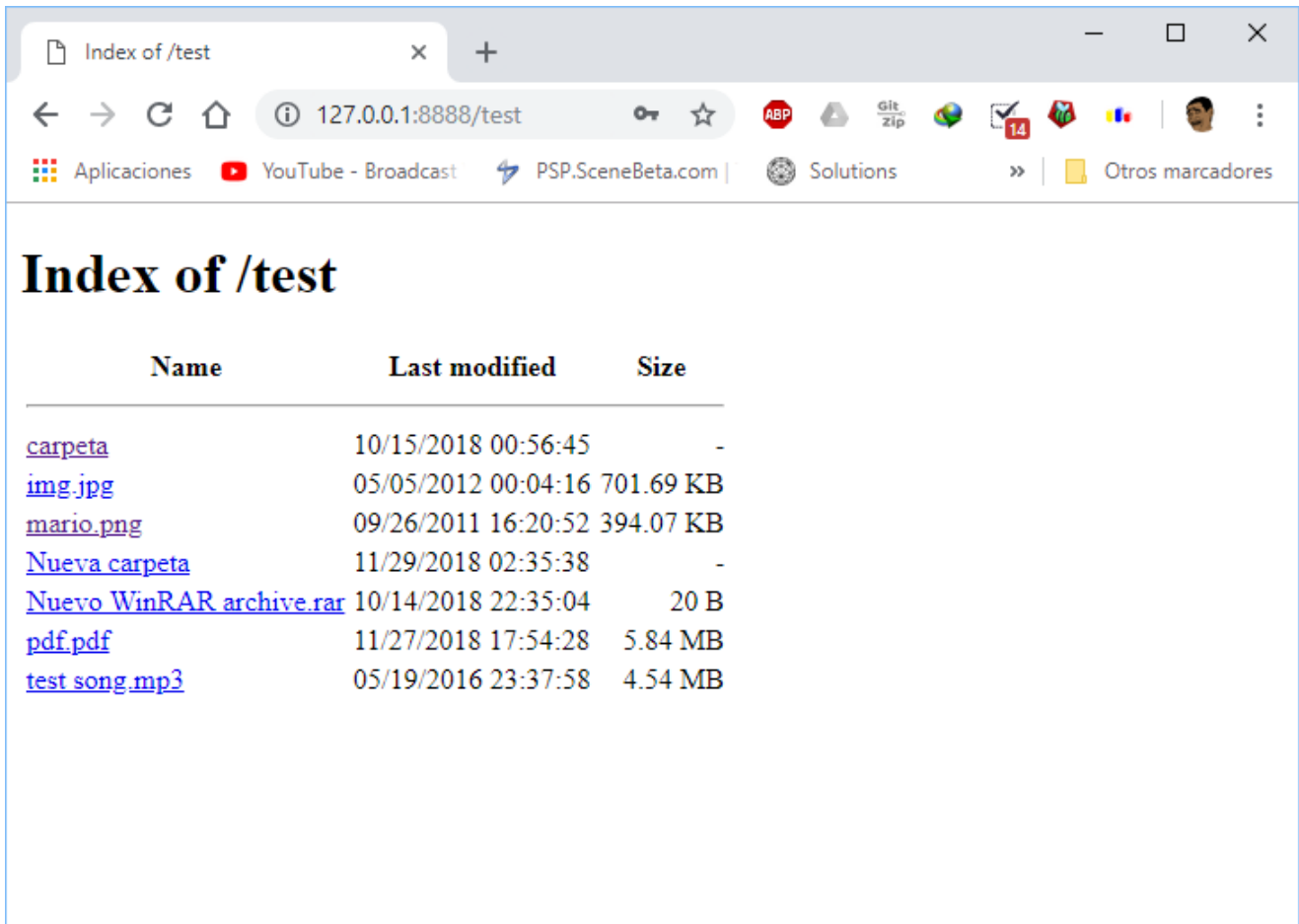
POST

Valor 1 : Valor 2 :

Nos logueamos:



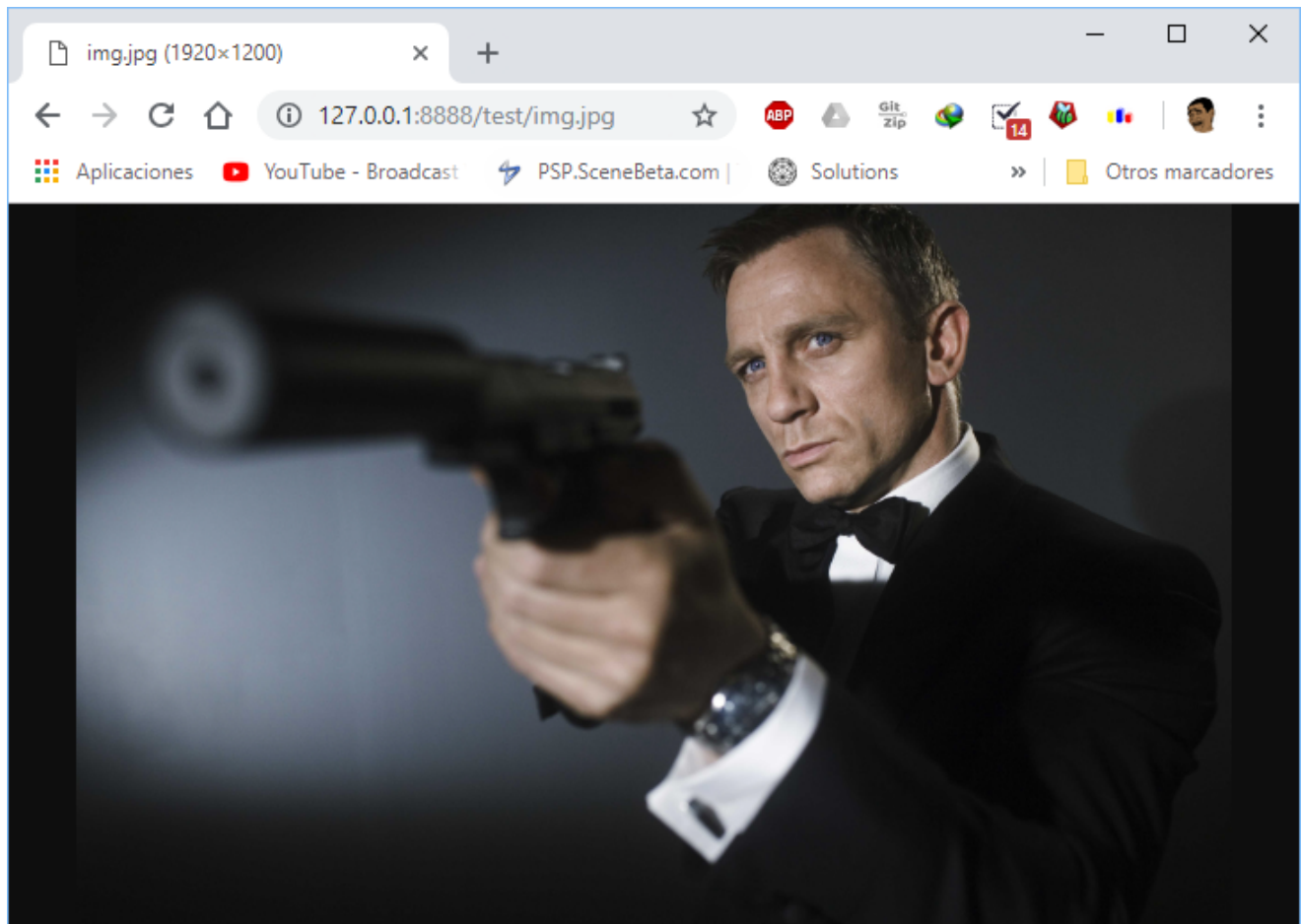
Probamos el examen de directorios:



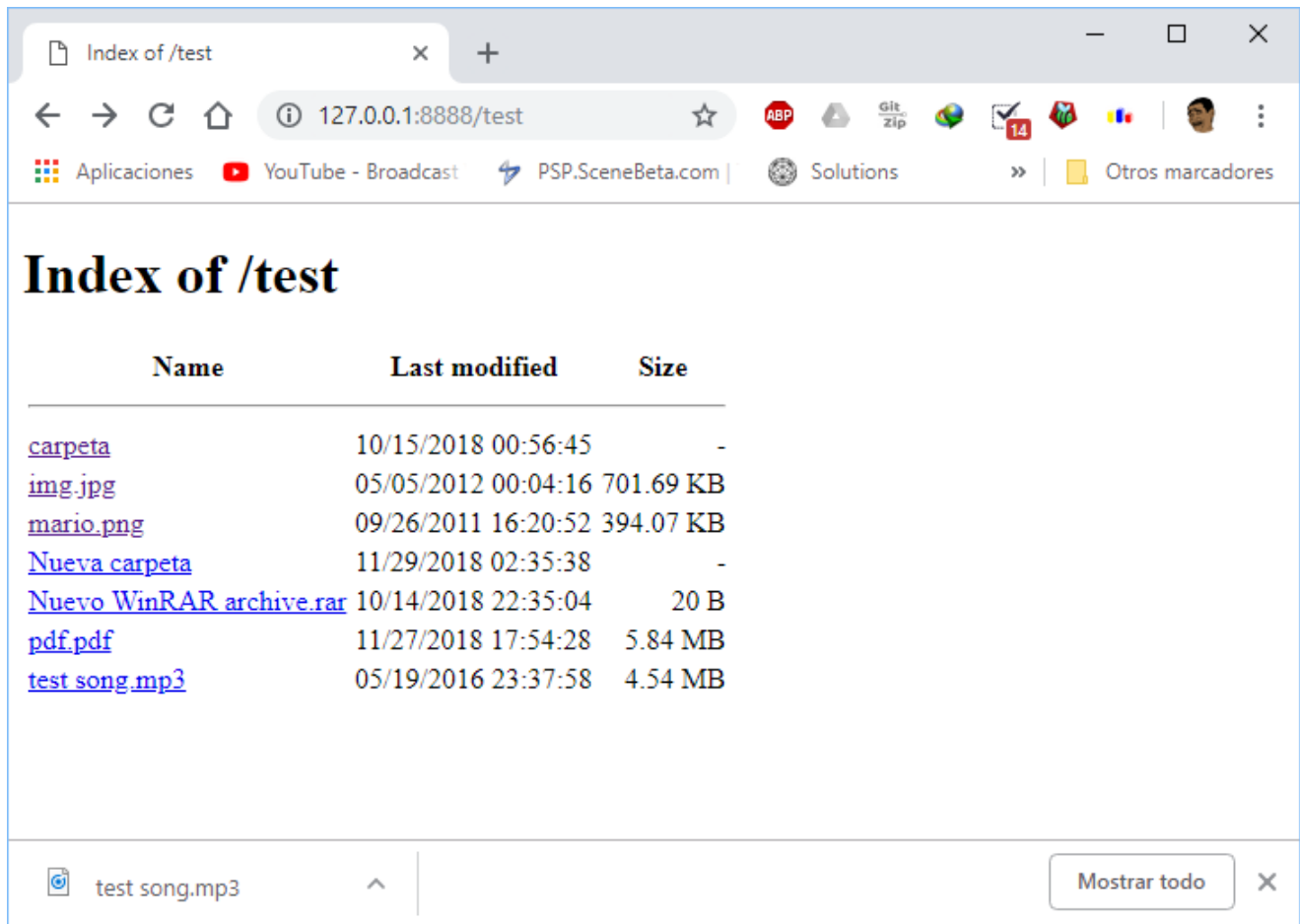
The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8888/test'. The page title is 'Index of /test'. Below the title, there is a table listing the contents of the directory. The table has three columns: 'Name', 'Last modified', and 'Size'. The listed items are: 'carpeta' (a directory), 'img.jpg' (701.69 KB), 'mario.png' (394.07 KB), 'Nueva carpeta' (a directory), 'Nuevo WinRAR archive.rar' (20 B), 'pdf.pdf' (5.84 MB), and 'test song.mp3' (4.54 MB). Each item is preceded by a blue underlined link icon.

Name	Last modified	Size
carpeta	10/15/2018 00:56:45	-
img.jpg	05/05/2012 00:04:16	701.69 KB
mario.png	09/26/2011 16:20:52	394.07 KB
Nueva carpeta	11/29/2018 02:35:38	-
Nuevo WinRAR archive.rar	10/14/2018 22:35:04	20 B
pdf.pdf	11/27/2018 17:54:28	5.84 MB
test song.mp3	05/19/2016 23:37:58	4.54 MB

Vemos una imagen:

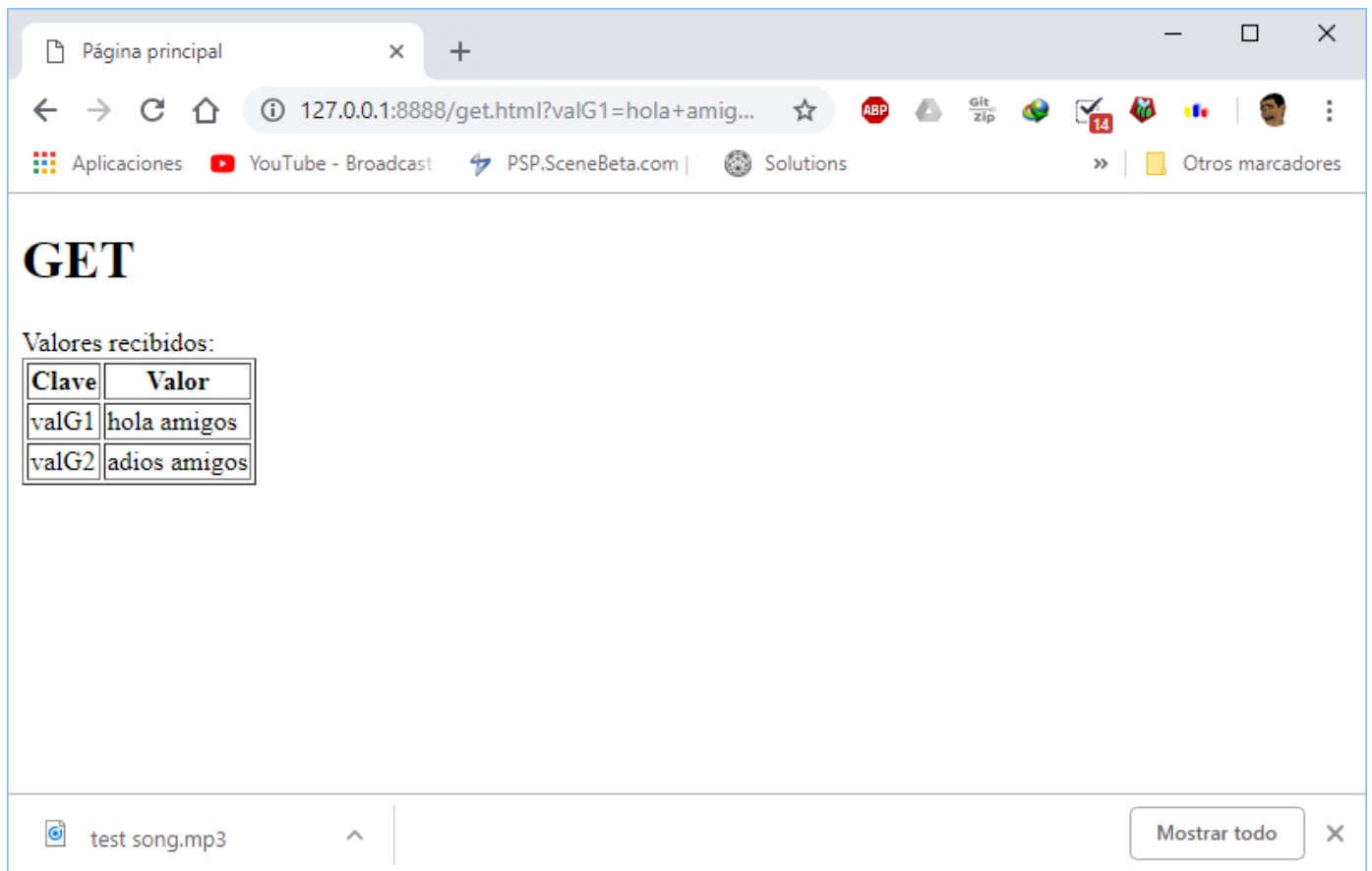


Descargamos un archivo más pesado:



Name	Last modified	Size
carpeta	10/15/2018 00:56:45	-
img.jpg	05/05/2012 00:04:16	701.69 KB
mario.png	09/26/2011 16:20:52	394.07 KB
Nueva carpeta	11/29/2018 02:35:38	-
Nuevo WinRAR archive.rar	10/14/2018 22:35:04	20 B
pdf.pdf	11/27/2018 17:54:28	5.84 MB
test song.mp3	05/19/2016 23:37:58	4.54 MB

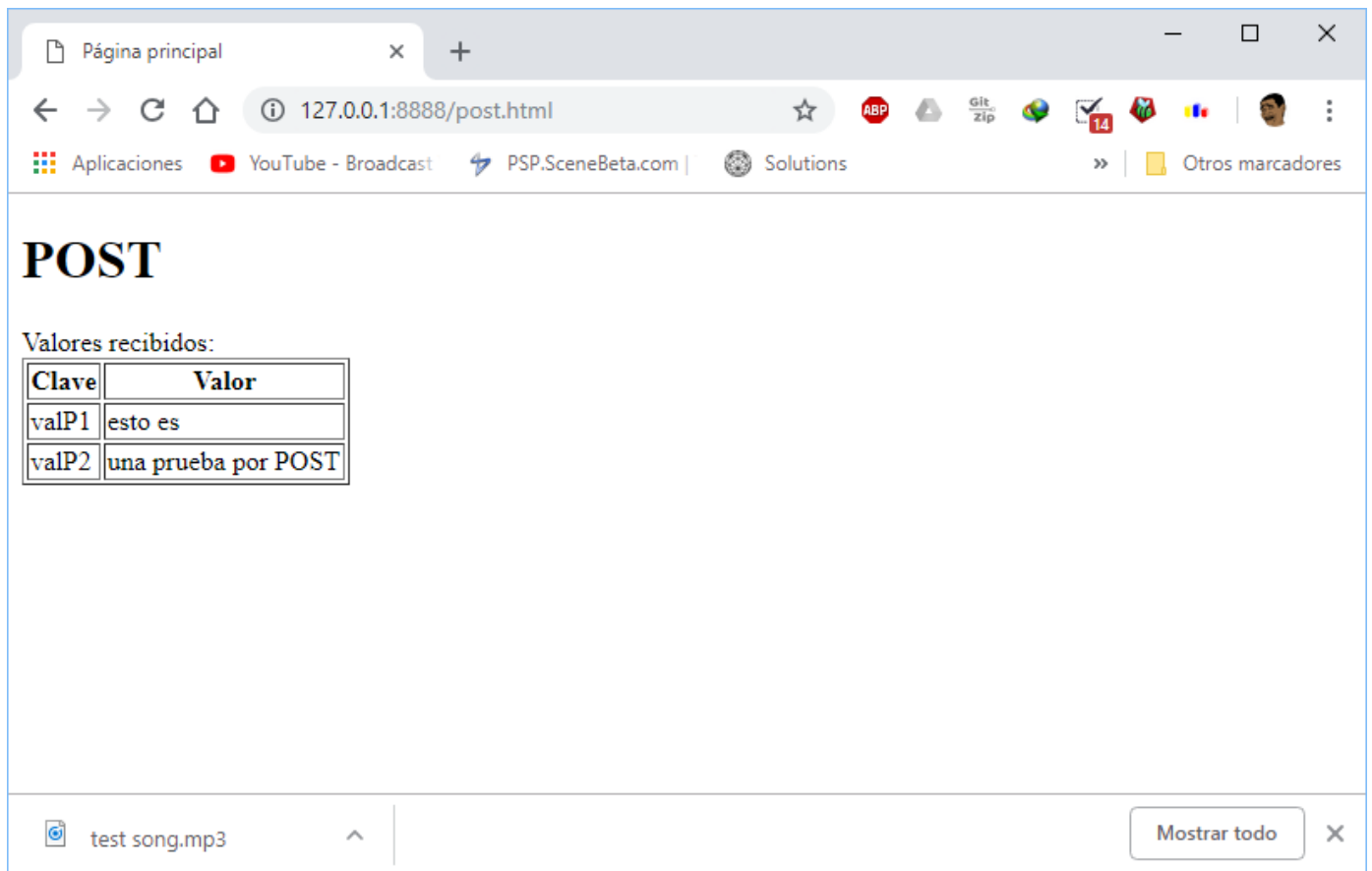
Prueba del método GET:



The screenshot shows a web browser window with a single tab titled 'Página principal'. The address bar displays the URL '127.0.0.1:8888/get.html?valG1=hola+amig...'. The browser's toolbar includes navigation buttons (back, forward, refresh, home), a search bar, and various extension icons. Below the toolbar, a row of bookmarks is visible, including 'Aplicaciones', 'YouTube - Broadcast', 'PSP.SceneBeta.com', and 'Solutions'. The main content area of the browser displays the word 'GET' in a large, bold, serif font. Below this, the text 'Valores recibidos:' is followed by a table with two columns: 'Clave' and 'Valor'. The table contains two rows of data: 'valG1' with the value 'hola amigos' and 'valG2' with the value 'adios amigos'. At the bottom of the browser window, a download bar shows a file named 'test song.mp3' with a download icon and an upward arrow. To the right of the download bar is a button labeled 'Mostrar todo' and a close button (X).

Clave	Valor
valG1	hola amigos
valG2	adios amigos

Prueba del método POST:



The screenshot shows a web browser window with a single tab titled 'Página principal'. The address bar displays '127.0.0.1:8888/post.html'. The browser's toolbar includes navigation buttons, a star icon, and several extension icons (ABP, Git Zip, etc.). Below the toolbar, there are bookmarks for 'Aplicaciones', 'YouTube - Broadcast', 'PSP.SceneBeta.com', and 'Solutions'. The main content area features a large heading 'POST' and a section titled 'Valores recibidos:' containing a table with two columns: 'Clave' and 'Valor'. The table lists two received values: 'valP1' with the value 'esto es' and 'valP2' with the value 'una prueba por POST'. At the bottom of the browser window, a download bar shows a file named 'test song.mp3' with an upward arrow icon. A button labeled 'Mostrar todo' with a close icon is located in the bottom right corner of the download bar.

Clave	Valor
valP1	esto es
valP2	una prueba por POST

Prueba del método PUT

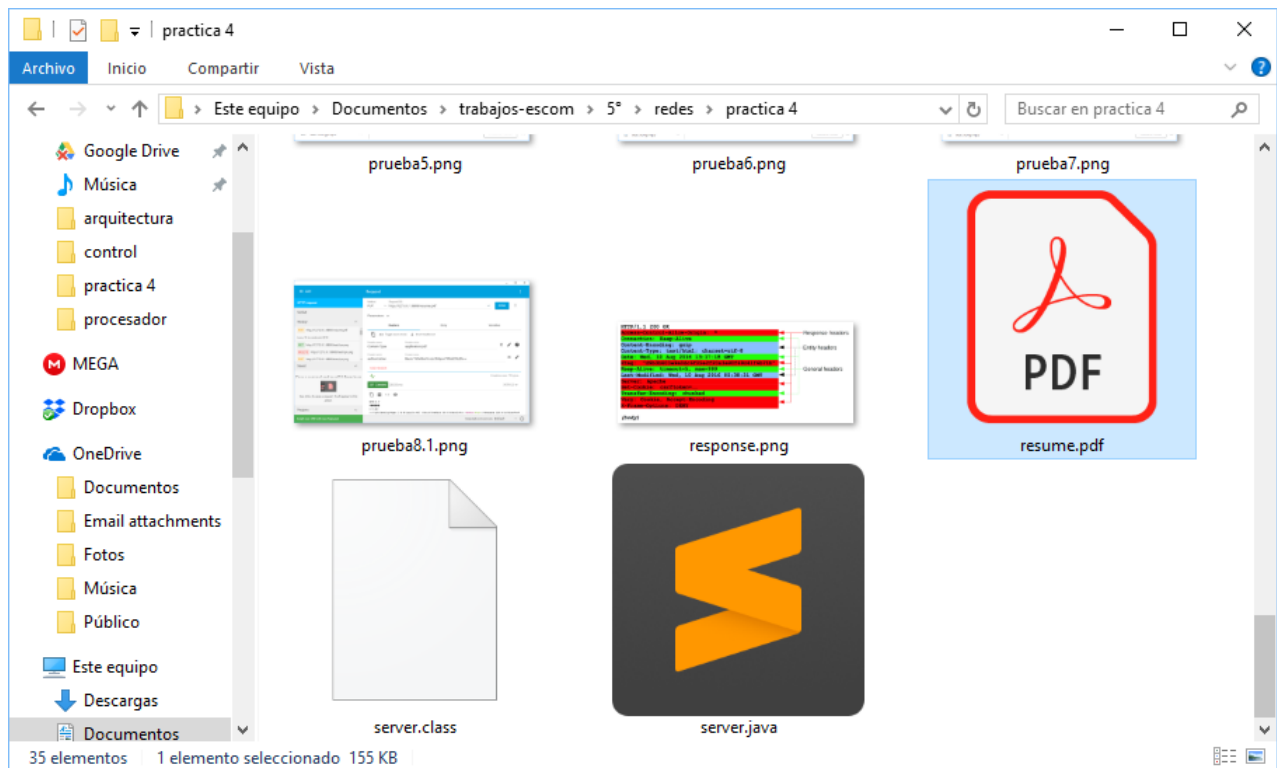
The screenshot shows the ARC (Advanced REST Client) interface. On the left, the 'HTTP request' tab is active, displaying a list of requests. The selected request is a PUT request to `http://127.0.0.1:8888/resume.pdf`. The main panel shows the request configuration:

- Method:** PUT
- Request URL:** `http://127.0.0.1:8888/resume.pdf`
- Parameters:** None
- Headers:**
 - Content-Type:** `application/pdf`
 - authorization:** `Basic YWxhbmVvczc3NzpwYXNzd29yZA==`
- Body:** None
- Variables:** None

The response is displayed below the headers, showing a **201 Created** status with a response time of 38.00 ms. The response body is a PDF document structure:

```
%PDF-1.7
%❖❖❖❖
1 0 obj
<</Type/Catalog/Pages 2 0 R/Lang(es-ES) /StructTreeRoot 38 0 R/MarkInfo<</Marked true>>/Metadata 226 0 R/ViewerPref
```

The bottom status bar indicates the selected environment is 'Default'.



Prueba del método DELETE

The screenshot displays the ARC HTTP client interface. On the left sidebar, the 'History' section shows a list of recent requests, including a DELETE request to `http://127.0.0.1:8888/resume.pdf`. The main panel is titled 'Request' and shows the selected request details. The 'Method' is set to 'DELETE' and the 'Request URL' is `http://127.0.0.1:8888/resume.pdf`. A 'SEND' button is visible. Below the URL, there are tabs for 'Parameters', 'Headers', 'Body', and 'Variables'. The 'Headers' tab is active, showing a single header: 'authorization' with the value 'Basic YWxhbmVvczc3NzpwYXNzd29yZA=='. A red warning message states 'Content-Type header is not defined'. The response panel shows a '200 OK' status with a response time of '8.70 ms'. The response body is an HTML document with the title 'Eliminado' and the message 'Se eliminó el archivo' and 'No se podrá acceder de nuevo al archivo.'.

ARC

Request

HTTP request

Socket

History

Today

DELETE http://127.0.0.1:8888/resume.pdf

PUT http://127.0.0.1:8888/resume.pdf

lunes, 15 de octubre de 2018

GET http://127.0.0.1:8888/test/ipn.png

DELETE http://127.0.0.1:8888/test/ipn.png

PUT http://127.0.0.1:8888/test/ipn.png

Saved

Save a request and recall it from here

Use `ctrl+s` to save a request. It will appear in this place.

Projects

Install new ARC with new features!

Method: DELETE

Request URL: `http://127.0.0.1:8888/resume.pdf`

Parameters

Headers

Body

Variables

Header name: authorization

Header value: Basic YWxhbmVvczc3NzpwYXNzd29yZA==

ADD HEADER

Content-Type header is not defined

Headers size: 49 bytes

200 OK 8.70 ms

DETAILS

`<!DOCTYPE html>`

`<meta charset="utf-8">`

`<html>`

`<head>`

`<title>Eliminado</title>`

`</head>`

`<body>`

`<h1>Se eliminó el archivo</h1>`

`No se podrá acceder de nuevo al archivo.`

`</body>`

`</html>`

Selected environment: Default

4. Conclusiones

En esta práctica implementamos una versión muy sencilla del protocolo HTTP 1.1. Tuvimos que tener un buen manejo de los `streams` en java y de escribir correctamente los headers y el cuerpo a la salida, tal y como lo dice el protocolo HTTP; y por cada método que íbamos recibiendo, ejecutar las acciones correspondientes, validando todos los posibles códigos de estado de acuerdo a lo que el cliente diga. Usamos expresiones regulares para extraer de forma mucho más sencilla los datos que recibimos de los clientes.

A pesar de ser una versión muy reducida, cumple muy bien con el objetivo de demostrar la implementación del protocolo HTTP desde cero y comprobar que funciona en cualquier navegador, e incluso soporta sustituciones muy básicas de los campos almacenados en el formulario, a través de `GET` y `POST`; así como el soporte de un muy básico inicio de sesión mediante el navegador.

Por último, haberlo implementado sobre un pool de conexiones nos da mayor eficiencia, pues no se crea un hilo por cada cliente que va llegando, sino que si se supera el máximo número de clientes simultáneos, tendrán que esperar hasta que uno se desocupe. Pero como HTTP es *stateless*, esto pasa muy rápido y, a menos que un cliente esté descargando algo muy pesado, la asignación de los recursos entre clientes se dará de una forma transparente.