Proyecto 1: Servidor Proxy Web - HTTP

Barreto Jiménez, Juan Sebastián
Pontificia Universidad Javeriana
Ingeniería Electrónica e Ingeniería Sistemas
Colombia – Bogotá D.C
juan_barreto@javeriana.edu.co

Chen He, Janet
Pontificia Universidad Javeriana
Ingeniería Electrónica e Ingeniería Sistemas
Colombia – Bogotá D.C
j-chen@javeriana.edu.co

Niño Rodríguez, María José
Pontificia Universidad Javeriana
Ingeniería Electrónica e Ingeniería Sistemas
Colombia – Bogotá D.C
ma.nino@javeriana.edu.co

Quintana Echavarria, David Santiago
Pontificia Universidad Javeriana
Ingeniería Electrónica e Ingeniería Sistemas
Colombia – Bogotá D.C
quintanae-david@javeriana.edu.co

Abstract— This article contains the documentation for the development of the Web Proxy Server with HTTP 1.1 protocol. As delivery of Project 1 of the Communications and Networks subject. In it, the research sources are evidenced as a preliminary approach to address the subject, the design of the program, the implementation, the performance tests and finally the respective conclusions.

Palabras Claves—*HTTP, Protocolo, Proxy, Servidor, Url, GET, POST, Cliente, Virtual.*

I. INTRODUCCIÓN

En esta entrega se tiene como finalidad desarrollar un Servidor Porxy Web que maneje el protocolo HTTP 1.0. Ya más adelante en el documento se especifica un pequeño cambio en trabajar con la versión HTTP 1.1. que cumple con la misma funcionalidad.
Sabiendo esto, entonces cómo funciona el Servidor Proxy Web, pues este actúa como un intermediario, recibiendo solicitudes de clientes que este caso son navegadores como Chrome y Microsoft Edge u otro de tipo comercial (Como notación adicional en este proyecto solo se trabajó con solicitudes de tipo GET y POST). Después estas solicitudes deben ser reenvidas al sitio web de destino y la respuesta del servidor reenviada al navegador.

Adicionalmente el Servidor funciona como "sitios web virtuales", es decir que se trabaja con un nombre de Host virtual que llega con la solicitud y luego será cambiado según corresponda por el Host real. Si no existe esta información en el archivo configurado entonces la solicitud se reenvía al navegador sin modificaciones. [1]

II. MARCO TEÓRICO

Para el desarrollo del Servidor Proxy Web, como punto inicial se realizó lecturas de RFCs correspondientes, para ello se dará una breve contextualización de lo que es y una explicación de algunos términos a tener en cuenta.

La sigla RFCs significa *Request for Comments* (Pedido para Comentarios). Son una serie de documentos en donde contienen notas de trabajo del "*Network Working Group* (Grupo de Trabajo de Red)" conocida como la comunidad de Internet de investigación y desarrollo. [2]

Teniendo en cuenta lo anterior, entonces para poder comprender en la totalidad un RFC que contenga un protocolo de internet es importante resaltar los siguientes términos y categorizaciones.[2]

A. Nivel de madurez

- **Estándar:** Considerado como el protocolo estándar oficial para Internet. Compuesto por dos grupos:

(1) protocolo IP y superiores, protocolos que se aplican a todo Internet; y

(2) protocolos específicos de red, generalmente especificaciones de cómo hacer IP en tipos particulares de redes.

- **Borrador de estándar:** Considerado como un posible protocolo estándar. Como lo indica el nombre, es un borrador antes de oficializarse.
- **Estándar propuesto:** Propuestas de protocolos que pueden ser consideradas para estandarización en un futuro.
- **Experimental:** Son esos que se desarrollan como parte de un proyecto en proceso de investigación.
- **Informativo:** Son protocolos desarrollados por otras entidades.
- **Histórico:** son protocolos que es improbable que lleguen a ser estándares en Internet ya sea porque han sido suplantados por un desarrollo posterior o debido a la carencia de interés.

B. Nivel de requerimiento

- **Requerido:** Un sistema debe implementar los Protocolos Requeridos.
- **Recomendado:** Un sistema debería implementar los Protocolos Recomendados.
- **Opcional:** Un sistema puede o no implementar un Protocolo Opcional.
- **Uso limitado:** Protocolos son para uso en circunstancias limitadas. Esto puede ser a causa de su estado experimental, naturaleza especializada, funcionalidad limitada, o estado histórico.
- **No recomendado:** Estos protocolos no son recomendados para uso general.

III. FUNCIONAMIENTO PROTOCOLO

Se presenta a continuación los RFCs utilizados este proyecto.

A. Protocolo de transferencia de hipertexto: HTTP / 1.0 [3]

1945 Protocolo de transferencia de hipertexto - HTTP / 1.0. T. Berners-Lee, R. Fielding, H. Frystyk. Mayo de 1996. (Formato: TXT, HTML) (Estado: INFORMACIONAL) (DOI: 10.17487 / RFC1945)

Imagen 1. Documento RFC sobre el protocolo HTTP 1.0 encontrado en el item de los docuemntos RFCs

Cabe resalta con el marco teórico presentado que este protocolo es de tipo informativo, por lo que significa que fue desarrollado por otra entidad.

El protocolo HTTP se basa en un paradigma de solicitud / respuesta. Un cliente establece una conexión con un servidor y envía una solicitud al servidor. El servidor responde con una línea de estado, incluida la versión del protocolo del mensaje y un éxito o código de error, seguido de un mensaje similar a MIME que contiene el servidor información, metainformación de la entidad y posible contenido del cuerpo.

El caso más simple es una sola conexión (V) entre el agente de usuario (UA) y el servidor de origen (O)

```
    cadena de solicitud ------------------------>
UA ------------------- v -------------------- O
    <----------------------- cadena de respuesta
```

Imagen 2. Conexión simple de n protocolo HTTP

B. Protocolo de transferencia de hipertexto: HTTP / 1.1 [4]

Protocolo de transferencia de hipertexto 2068 - HTTP / 1.1. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. Enero de 1997. (Formato: TXT, HTML) (Obsoletado por RFC2616) (Estado: ESTÁNDAR PROPUESTO) (DOI: 10.17487 / RFC2068)

Imagen 3. Documento RFC sobre el protocolo HTTP 1.1 encontrado en el item de los docuemntos RFCs

Luego en la siguiente versión presentada, este protocolo paso de ser informativo a estándar propuesto. Lo que quiere decir que puede llegar estandarizarse.

El funcionamiento del protocolo es igual al protocolo HTTP 1.0, sin embargo, con este nuevo protocolo se mejoró ambigüedades como una conexión que permite se reutilizada. SE permitió que las respuestas a peticiones fueran subdividas en partes y finalmente lo más importante se incluyó la cabecera Host permitiendo alojar varios dominios en la misma dirección IP.

Del mismo modo al igual que el protocolo HTTP 1.0 se publicó a mediados de 1997.

Finalmente, debido a la poca diferencia y con una mejora que se considera bastante importante, como el manejo de la cabecera Host, se tomó este protocolo para el desarrollo del proyecto.

C. Localizadores uniformes de recursos (URL) [5]

1738 Localizadores uniformes de recursos (URL). T. Berners-Lee, L. Masinter, M. McCahill. Diciembre de 1994. (Formato: TXT, HTML) (Obsoleto por RFC4248, RFC4266) (Actualizado por RFC1808, RFC2368, RFC2396, RFC3986, RFC6196, RFC6270, RFC8089) (Estado: ESTÁNDAR PROPUESTO) (DOI: 10.17487 / RFC1738)

Imagen 4. Documento RFC sobre URL encontrado en el item de los docuemntos RFCs

Las URL se utilizan para "localizar" recursos. Una vez encontrado, el sistema puede realizar una variedad de operaciones en el recurso, como acceder, actualizar, reemplazar, buscar atributos, etc.

D. Protocolo de control de transmisión [6]

0793 Protocolo de control de transmisión. J. Postel. Septiembre de 1981. (Formato: TXT, HTML) (obsoleto RFC0761) (actualizado por RFC1122, RFC3168, RFC6093, RFC6528) (También STD0007) (Estado: ESTÁNDAR DE INTERNET) (DOI: 10.17487 / RFC0793)

Imagen 5. Documento RFC sobre el protocolo de control de transmisión encontrado en el item de los docuemntos RFCs

Su nivel de madurez es de estándar, es decir, el protocolo presentado en este RFC es considerado, un documento oficial para Internet.

El "protocolo de control de transmisión" (*'Transmission Control Protocol'*, TCP) está pensado para ser utilizado como un protocolo 'host' a 'host' muy fiable entre miembros de redes de comunicación de computadoras por intercambio de paquetes y en un sistema interconectado de tales redes.

IV. DISEÑO DEL PROGRAMA

Para el diseño del programa, se tiene en cuenta lo mencionado en el apartado anterior. Adicionalmente se utilizó como referencia varios archivos de internet [7], como videos [8] [9] para construir el servidor.

El lenguaje que se escogió para el desarrollo es Java y se trabajó desde la herramienta del Neatbeans con JDK 14.0.

El programa contará con tres paquetes, el principal que será el Servidor Proxy Web que se encarga de invocar el servidor. Luego está el paquete del Servidor en ella se desarrolla la funcionalidad tanto de las peticiones GET y POST como el desarrollo para el manejo de "sitios web virtuales". Finalmente, el último paquete es para el archivo de configuración en donde se almacena el host virtual, real y la raíz del directorio.

Entrando en más detalles con el desarrollo del servidor se comienza con la creación de la comunicación mediante un socket, proceso que existe en la maquina cliente y maquina servidora permitiendo de este modo que entre ellos se pueda dar la función de lectura y escritura de información. Para ello toca establecer el puerto de conexión que son valores entre 0 y 65535.

Posteriormente, con la conexión establecida se procede a administrar las respectivas peticiones.

A. Descripción solicitudes tipo GET

El método GET consta de enviar la información codificada del usuario directamente en la URL. Tiene un límite de 2000 caracteres, es información visible y no se puede mandar información binaria como archivos o imágenes. [10]

Para su desarrollo, se obtiene principalmente la información de la solicitud al Servidor Proxy para comprobar que está se ha realizado. Luego se procedería a analizar los encabezados o toda la información obtenida desde el objeto de solicitud. Se analiza el Host para determinar si corresponde a un "sitio web virtual", finalmente se reenvía la solicitud nuevamente al web de destino y confirma la respuesta del servidor.

A continuación, se presenta las definiciones de cada encabezado identificado como relevantes para analizar.

Encabezados [11] [12]

- **Tipo de solicitud, URL y Protocolo:** Identificación del tipo
- **Referer**: URL origen de la petición.
- **Proxy-Connection:** El mecanismo de conexión que tiene el proxy
- **User-Agent**: Cliente que está realizando la petición (normalmente muestra datos del navegador, como nombre, etc).
- **Pragma**: Compatibilidad con versiones anteriores de las memorias caché. Encabezado presente solo en la versión HTTP 1.0
- **Host**: Única cabecera requerida por HTTP 1.1. Indica el host y el puerto tal y como se especifica en la URL original.
- **Accept**: Tipos MIME que puede manejar el cliente
- **Accept-Encoding**: Define si el navegador puede aceptar datos codificados
- **Accept-Language**: Idiomas aceptados
- **Accept-Charset**: Conjunto de caracteres que el cliente puede manejar

B. Descripción solicitudes tipo POST

En el caso del método POST funciona similar que el GET, en este caso, la información es enviada a través del *body*, por lo que no se encontrara en la URL.
No tienen límite de cantidad de información, no es visible por lo que es ideal para información sensible o que necesita ser ocultada. Se puede enviar tanto datos en texto como binarios.

Puesto que la información es enviada en el *body*, para esta solicitud se extrae la información para realizar la respectiva validación y reenviar correctamente la información al web de destino, una vez realizado esto, se puede obtener la respuesta del servidor para comprobar que la operación es exitosa.

Se presenta al igual que el método GET, los encabezados con sus respectivas definiciones del método POST.

Encabezados [11] [12]

- **Tipo de solicitud, URL y Protocolo:** Identificación del tipo
- **Referer**: URL origen de la petición.
- **Proxy-Connection:**
- **User-Agent**: Cliente que está realizando la petición (normalmente muestra datos del navegador, como nombre, etc).
- **Host**: Unica cabecera requerida por HTTP 1.1. Indica el host y el puerto tal y como se especifica en la URL original.
- **Accept**: Tipos MIME que puede manejar el cliente
- **Accept-Encoding**: Define si el navegador puede aceptar datos codificados
- **Accept-Language**: Idiomas aceptados
- **Accept-Charset**: Conjunto de caracteres que el cliente puede manejar
- **Content-Type**: Tipo MIME de los datos enviados. Aplicable a peticiones POST
- **Content-Length**: Longitud de los datos enviados. Aplicable a peticiones POST

C. Descripción de la respuesta del servidor

Como ya mencionado anteriormente, luego de cada solicitud existe una respuesta del servidor para determinar cómo finalizo la operación ya que una vez enviado los datos se cierra el socket y la solicitud termina. Para ello, se presenta a continuación los encabezados respectivos de esta respuesta.

Encabezados [11] [12]

- **Protocolo, Código de éxito o fracaso:** Protocolo utilizado y estado de la respuesta del servidor

A continuación, se presenta una pequeña tabla con los posibles códigos que define el estado de la solicitud:

CÓDIGO	DESCRIPCIÓN
Aceptación	
200	Todo está bien
204	No hay documento nuevo
Redirección	
301	Documento en otro lugar
302	Documento en otro lugar con URL temporal
304	Documento perdido
Error del cliente	
400	Mala sintaxis de petición
401	Cliente no tiene permiso para acceder
403	El recurso no está disponible
404	No se encontró el recurso
408	Cliente tarda demasiado en enviar la petición
Error del servidor	
500	Error en el servidor
501	Servidor no soporta petición realizada
504	No obtuvo respuesta a tiempo de un servidor remoto

Tabla 1. Códigos de estado de la solicitud

- **Date**: Hora y fecha, en formato GMT, en que la respuesta ha sido generada
- **Server**: Contiene información sobre el software utilizado por el servidor de origen para gestionar la petición.
- **Last-Modified**: Fecha en que el documento servido se modificó por última vez.
- **ETag**: Se trata de un validador, un tipo de hilo único identificando la versión del recurso.
- **Accept-Ranges:** Indica si el servidor acepta peticiones de rango y, de ser así, en qué unidades puede expresarse ese rango.
- **Content-Length**: Número de bytes de la respuesta
- **Connection**: Controla si la conexión a la red se mantiene activa después de que la transacción en curso haya finalizado.
- **Content-Type**: Tipo MIME de la respuesta

D. Manejo de "sitio web virtuales"

Para esta parte, como se indica en la guía se necesita un archivo de configuración, para ello se hizo un archivo con formato txt, en donde contienen los datos necesarios, separados por comas.

- **Nombre Host virtual**
- **Host real**
- **Directorio raíz**

Una vez obtenida esta información y llegue una solicitud sin importar si es GET o POST. Se analizará el nombre host virtual para cambiarla al Host real.

V. PRUEBAS

Para la realización de las pruebas del Servidor Proxy Web.

Como primera instancia se habilito desde la configuración del sistema del PC en donde se realiza la acción el "Proxy" de manera manual, como se muestra en la imagen a continuación.

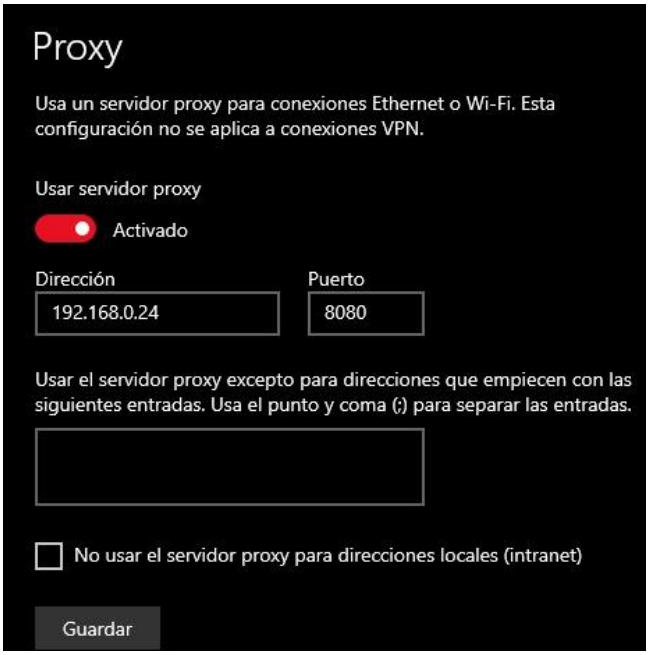

Imagen 6. Configuración manual del Proxy

Teniendo lo anterior, se presenta los escenarios de las pruebas, con los parámetros utilizados y respectivos resultados.

A. Escenario 1

PARÁMETROS	VALOR
Dirección	192.168.0.24
Puerto	8080
Navegador	Brave
Url	http://fsmsolutions.125mb.com/login.php
Tipo de solicitud	GET – Sin sitio virtual
Solicitud Servidor Proxy	
GET http://fsmsolutions.125mb.com/css/master.css HTTP/1.1 Headers: Accept-encoding=[gzip, deflate] Accept=[text/css,*/*;q=0.1] Referer=[http://fsmsolutions.125mb.com/login.php] Host=[fsmsolutions.125mb.com] User-agent=[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36] Accept-language=[es-CO,es-419;q=0.9,es;q=0.8] Proxy-connection=[keep-alive] Sec-gpc=[1]	
Reenvío Solicitud Proxy	
GET http://fsmsolutions.125mb.com/css/master.css HTTP/1.1 Headers: Accept-encoding=[gzip, deflate] Accept=[text/css,*/*;q=0.1] Referer=[http://fsmsolutions.125mb.com/login.php] Host=[fsmsolutions.125mb.com] User-agent=[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36] Accept-language=[es-CO,es-419;q=0.9,es;q=0.8] Proxy-connection=[keep-alive] Sec-gpc=[1]	
Respuesta servidor proxy	
HTTP/1.1 200 Date: 25-10-2021 Last modification: 29-09-2021 Content-Length: 1510 Content Type: text/css	

Tabla 2. Escenario 1

B. Escenario 2

PARÁMETROS	VALOR
Dirección	192.168.0.24
Puerto	8080
Navegador	Brave
Url	http://juan_barreto/index2.html
Tipo de solicitud	GET – Con sitio virtual
Solicitud Servidor Proxy	
GET http://juan_barreto/index2.html HTTP/1.1 Headers: Accept-encoding=[gzip, deflate] Accept=[text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9] Host=[juan_barreto] User-agent=[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36] Accept-language=[es-CO,es-419;q=0.9,es;q=0.8] Proxy-connection=[keep-alive] Upgrade-insecure-requests=[1] Sec-gpc=[1]	
Reenvío Solicitud Proxy	
GET http://192.168.0.24/redes-test/juan_barreto/index2.html HTTP/1.1 Headers: Accept-encoding=[gzip, deflate] Accept=[text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9] Host=[juan_barreto] User-agent=[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36] Accept-language=[es-CO,es-419;q=0.9,es;q=0.8] Proxy-connection=[keep-alive] Upgrade-insecure-requests=[1] Sec-gpc=[1]	
Respuesta servidor proxy	
HTTP/1.1 200 Date: 25-10-2021 Last modification: 25-10-2021 Content-Length: 65 Content Type: text/html	

Tabla 3. Escenario 2

C. Escenario 3

PARÁMETROS	VALOR
Dirección	192.168.0.24
Puerto	8080
Navegador	Brave
Url	http://fsmsolutions.125mb.com/login.php
Tipo de solicitud	POST – Sin sitio virtual
Solicitud Servidor Proxy	
POST http://fsmsolutions.125mb.com/login.php HTTP/1.1 Headers: Origin=[http://fsmsolutions.125mb.com] Accept-encoding=[gzip, deflate] Accept=[text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,/;q=0.8,application/signed-exchange;v=b3;q=0.9] Referer=[http://fsmsolutions.125mb.com/login.php] Host=[fsmsolutions.125mb.com] User-agent=[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36] Content-type=[application/x-www-form-urlencoded] Accept-language=[es-CO,es-419;q=0.9,es;q=0.8] Proxy-connection=[keep-alive] Upgrade-insecure-requests=[1] Sec-gpc=[1] Content-length=[49] Cache-control=[max-age=0] Body: email=juanda%40javeriana.edu.co&password=Hola1234	
Reenvío Solicitud Proxy	
POST http://fsmsolutions.125mb.com/login.php HTTP/1.1 Headers: Origin=[http://fsmsolutions.125mb.com] Accept-encoding=[gzip, deflate] Accept=[text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,/;q=0.8,application/signed-exchange;v=b3;q=0.9] Referer=[http://fsmsolutions.125mb.com/login.php] Host=[fsmsolutions.125mb.com] User-agent=[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36] Content-type=[application/x-www-form-urlencoded] Accept-language=[es-CO,es-419;q=0.9,es;q=0.8] Proxy-connection=[keep-alive] Upgrade-insecure-requests=[1] Sec-gpc=[1] Content-length=[49] Cache-control=[max-age=0] Body: email=juanda%40javeriana.edu.co&password=Hola1234	
Respuesta servidor proxy	
HTTP/1.1 302 Date: 25-10-2021 Last modification: 31-12-1969 Content-Length : 5712 Content Type: text/html; charset=UTF-8	

Tabla 4. Escenario 3

VI. CONCLUSIONES

Se logro desarrollar exitosamente el proxy Web. Analizando los paquetes HTTP, diferenciando y redirigiendo solicitudes POST y GET en distintos sitios web. Procesando los paquetes recibidos logramos desarrollar el modo de sitios virtuales, generando el redireccionamiento de tráfico adecuadamente. En el mismo sentido se logra ver los datos relevantes de cada paquete generando un registro de estos valores.

Por medio de herramientas como WireShark se desarrollaron y ejecutaron las adecuadas pruebas para la verificación y análisis del tráfico de la red, recibiendo y reenviado los paquetes como era esperado.

Se logro afianzar los conceptos vistos en clase por medio del proyecto, poniendo en práctica estos mismos. Desarrollando en general un proyecto acorde con lo esperado. HTTP es un protocolo sencillo que es fácil de analizar para la implementación y análisis del proyecto.

VII. LINK DE VIDEO

Link: https://youtu.be/JYWGH6Nmv5Y

VIII. REFERENCIAS

[1] Á. C. M. G. Macizo Vega Roberto David, “Proyecto Comunicaciones y Redes,” *Univ. Priv. del Norte*, p. 116, 2018, [Online]. Available: http://repositorio.ucv.edu.pe/handle/UCV/27098.

[2] E. de A. de Internet, “PROTOCOLOS OFICIALES ESTÁNDARES DE INTERNET.” https://www.rfc-es.org/rfc/rfc1780-es.txt.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk, “Protocolo de transferencia de hipertexto - HTTP / 1.0,” May 1996. doi: 10.17487/rfc1945.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, “Protocolo de transferencia de hipertexto - HTTP / 1.1,” Jan. 1997. doi:

10.17487/rfc2068.

[5] T. Berners-Lee, L. Masinter, and M. McCahill, “Localizadores uniformes de recursos (URL),” Dec. 1994. doi: 10.17487/rfc1738.

[6] J. Postel, “PROTOCOLO DE CONTROL DE TRANSMISIÓN,” Sep. 1981. doi: 10.17487/rfc0793.

[7] Baeldung, “Hacer una solicitud HTTP simple en Java.” https://www.baeldung.com/java-http-request.

[8] J. O. Ordoñez, “Java - Ejercicio 253: Crear un Servidor HTTP para Responder a una Solicitud GET.” .

[9] J. O. Ordoñez, “Java - Ejercicio 254: Mostrar la Información de una Solicitud GET con la Clase HttpExchange.” https://www.youtube.com/watch?v=Fyjzz2gJjUE&list=PL2PZw96yQChyMNJA6JYHzmZ6QX-i1nDPI&index=255.

[10] D. Lázaro, “GET y POST en PHP.” https://diego.com.es/get-y-post-en-php.

[11] V. Cgi, “Procesamiento de peticiones,” pp. 1–23, 2013.

[12] M. W. Docs, “HTTP headers.” https://developer.mozilla.org/es/docs/Web/HTTP/Headers.