DAW
Despliegue de aplicaciones web

2. Arquitecturas web

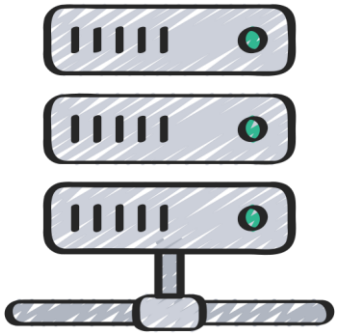
Implantación y administración de servidores

Juan Jesús Tortajada Cordero
jtortajada@florida-uni.es

Florida
Grup Educatiu

Índice

1. Introducción: arquitectura cliente-servidor

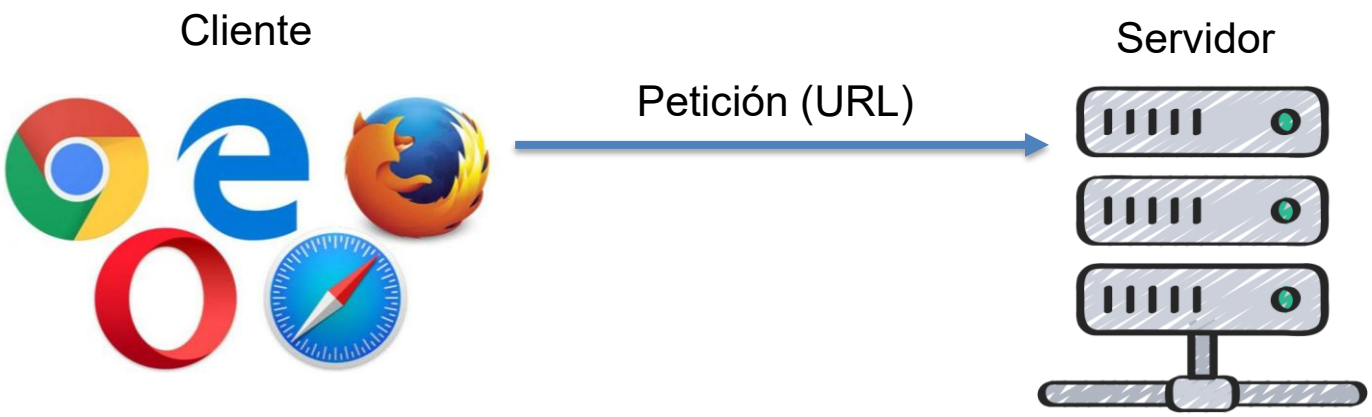


Cliente

Servidor

- ✓ Almacenar contenido
- ✓ Disponibilidad 24/7

1. Introducción: arquitectura cliente-servidor

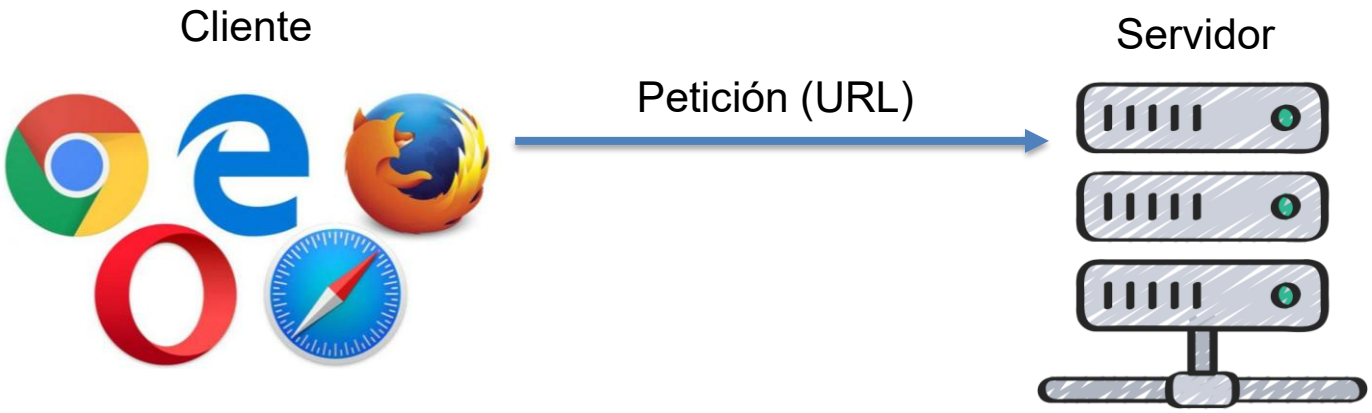


Aplicación web:

- Recibe y procesa la(s) petición(es)
- Ejecuta código (PHP, Java, JS,…)
- Operaciones CRUD (SQL, NoSQL)
- Devuelve respuesta y contenido
- Renderizado dinámico de webs

CRUD: operaciones sobre bases de dados: crear, leer, actualizar y borrar (***C**reate*, ***R**ead*, ***U**pdate*, ***D**elete*)

1. Introducción: arquitectura cliente-servidor



Partes de una aplicación web:

- ***Frontend***: componentes destinados a la interacción con los usuarios
- ***Backend***: componentes relacionados con los datos, la infraestructura, lógica de negocio, microservicios, APIs, seguridad, escalabilidad, eficiencia, etc.

NOTA: el ***frontend*** de la aplicación se lo "envía" el servidor al navegador (cliente), para que este lo muestre (renderizar) y el usuario pueda interactuar (código Javascript) y continuar realizando peticiones. Debe estar optimizado para el cliente (*responsive*).

2. Protocolos de comunicaciones - HTTP

HTTP: ***H**yper**t**ext **T**ransfer **P**rotocol* (protocolo de transferencia de hipertexto)

Protocolo orientado a transacciones con esquema petición-respuesta entre cliente (*user agent*, navegadores web) y servidor. El cliente solicita un recurso identificado unívocamente como URL (***u**niform **r**esource **l**ocator*).

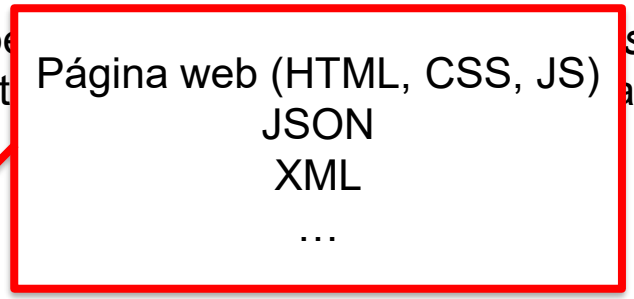
Tipos de peticiones más utilizadas:

- **GET**: solicita una representación del recurso especificado.
- **POST**: envía datos en la URL de la petición para que sean procesados por el recurso identificado del servidor (creación de un nuevo recurso).
- **PUT**: actualización de un recurso.
- **DELETE**: borrado de un recurso.
- **OPTIONS**: devuelve los métodos HTTP que soporta el servidor para un recurso concreto.

2. Protocolos de comunicaciones - HTTP

HTTP: ***H**yper**t**ext **T**ransfer **P**rotocol* (protocolo de transferencia de hipertexto)

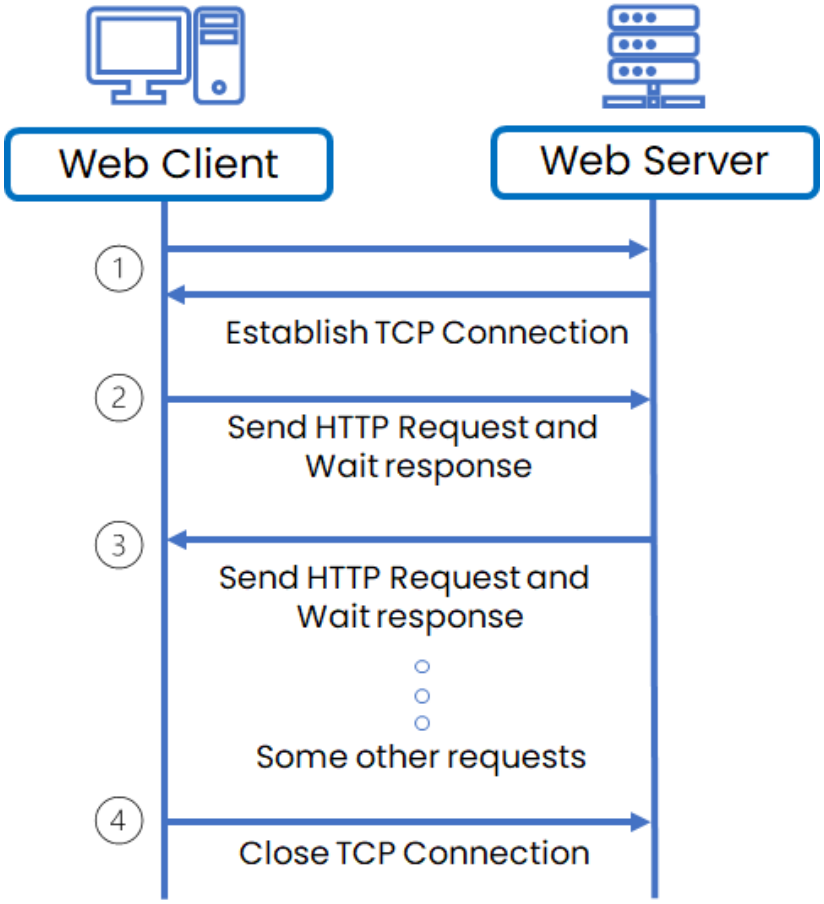
Protocolo orientado a transacciones con esquema pe*ser agent*, navegadores web) y servidor. El cliente solicitamente como URL (***u**niform **r**esource **l**ocator*).



Tipos de peticiones más utilizadas:

- **GET**: solicita una representación del recurso especificado.
- **POST**: envía datos en la URL de la petición para que sean procesados por el recurso identificado del servidor (creación de un nuevo recurso).
- **PUT**: actualización de un recurso.
- **DELETE**: borrado de un recurso.
- **OPTIONS**: devuelve los métodos HTTP que soporta el servidor para un recurso concreto.

2. Protocolos de comunicaciones - HTTP



netburner.com

Cliente: peticiones (*request*)
Servidor: respuestas (*response*)

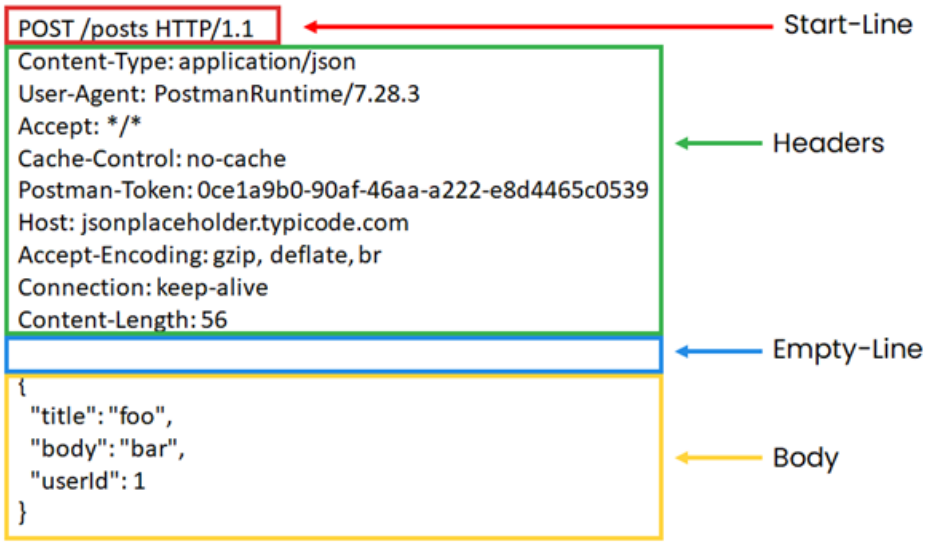
Protocolo TCP/IP: la información (peticiones y respuestas) viaja "troceada" en paquetes y debe confirmarse cada paquete que se recibe (ACK - *acknowledgement*)

Cargar una página implica múltiples peticiones y respuestas (y ACKs)

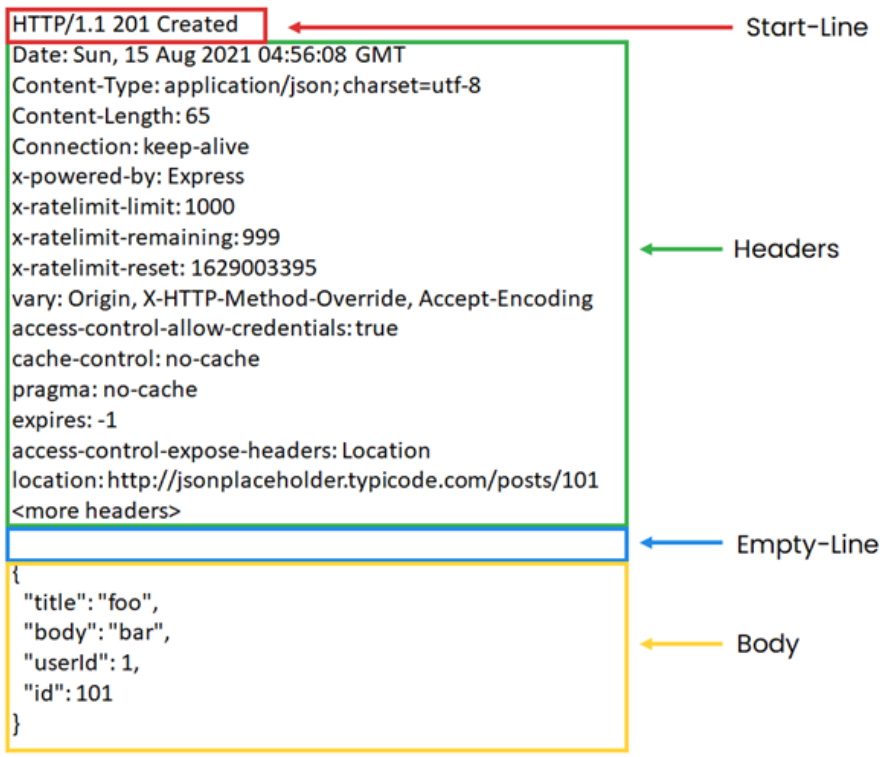
2. Protocolos de comunicaciones - HTTP

Ejemplo de petición POST desde cliente y de su respuesta desde servidor

PETICIÓN



RESPUESTA



https://www.netburner.com/learn/an-introduction-to-http-protocol/

2. Protocolos de comunicaciones - HTTP

Códigos de estado de respuesta HTTP

1xx (Información): petición recibida, continua el proceso.
2xx (Éxito): petición recibida con éxito, comprendida y aceptada.
3xx (Redirección): se necesitan más acciones para completar la petición.
4xx (Error de cliente): la sintaxis de la petición no es correcta o no puede completarse.
5xx (Error de servidor): el servidor no pudo completar la petición (aunque sea válida).

Esto son rangos, no todos los códigos están definidos en la definición del protocolo HTTP:

https://datatracker.ietf.org/doc/html/rfc9110

2. Protocolos de comunicaciones - HTTP

Códigos de estado de respuesta HTTP

100 Continue
El navegador “pregunta” primero al servidor si puede continuar con la petición, especialmente en los casos en que el cuerpo de la petición es grande.

101 Switching Protocols
El servidor acepta un cambio de versión de protocolo HTTP a propuesta del navegador.

102 Processing
El servidor responde que aún está procesando una petición del navegador, evitando así que el navegador considere que se ha perdido.

NOTA: los códigos del rango 1XX no se utilizan directamente en PHP como códigos de respuesta “finales” de una petición, sino que los servidores (Apache o Nginx) los utilizan internamente.

https://datatracker.ietf.org/doc/html/rfc9110

2. Protocolos de comunicaciones - HTTP

Códigos de estado de respuesta HTTP

200 OK
Respuesta estándar para peticiones correctas. Suele ir acompañada de los datos solicitados por el navegador.

201 Created
Petición completada y creación de un nuevo recurso.

202 Accepted
Petición aceptada, aunque no se ha podido completar aún.

203 Non-authoritative Information
Petición se ha completado, pero el contenido procede de otro servidor.

204 No Content
Petición completada, pero la respuesta no tiene contenido.

https://datatracker.ietf.org/doc/html/rfc9110

2. Protocolos de comunicaciones - HTTP

Códigos de estado de respuesta HTTP

205 Reset Content

Petición completada, respuesta sin contenidos y el navegador tiene que reinicializar la página desde la que realizó la petición (p.ej. borrar formularios una vez se han enviado).

206 Partial Content

Petición aceptada, pero devuelve solo parte de los contenidos (p.ej. trocear una descarga o retomar una descarga interrumpida).

https://datatracker.ietf.org/doc/html/rfc9110

2. Protocolos de comunicaciones - HTTP

Códigos de estado de respuesta HTTP

300 Multiple Choices
Indicar opciones múltiples para el contenido solicitado por el cliente (p.ej. videos con distinta resolución).

301 Moved Permanently
Todas las peticiones se redirigen a otra URL.

302 / 303 See Other
La respuesta a la petición está en otra URL accesible mediante otra petición GET.

304 Not Modified
El contenido solicitado en la URL no ha sido modificado desde la última vez que se solicitó, con lo que no hace falta reenviarlo (ahorro de tráfico y ancho de banda).

https://datatracker.ietf.org/doc/html/rfc9110

2. Protocolos de comunicaciones - HTTP

Códigos de estado de respuesta HTTP

400 Bad Request
La solicitud del cliente tiene errores de sintaxis.

401 Authorization Required
La solicitud requiere de autorización previa por parte del cliente.

403 Forbidden
Tanto si el cliente está autorizado como si no, no tiene privilegios suficientes para acceder al recurso solicitado.

404 Not Found
El servidor no encuentra el recurso solicitado.

405 Method Not Allowed
La petición (GET, POST, PUT,…) no es la adecuada para la URL solicitada.

https://datatracker.ietf.org/doc/html/rfc9110

2. Protocolos de comunicaciones - HTTP

Códigos de estado de respuesta HTTP

406 Not Acceptable
El servidor no puede devolver los datos en el formato que solicita el cliente en su petición (campo “Accept” de la cabecera de la petición).

408 Request Timeout
El cliente no ha continuado la petición que tenía pendiente.

414 URI Too Long
La URL es demasiado larga (p.ej. una petición GET que debería ser POST).

415 Unsupported Media Type
El formato solicitado por el navegador no es entendible por el servidor.

429 Too Many Requests
Demasiadas conexiones desde la misma IP cliente.

https://datatracker.ietf.org/doc/html/rfc9110

2. Protocolos de comunicaciones - HTTP

Códigos de estado de respuesta HTTP

451 Unavailable For Legal Reasons
Contenido eliminado por temas legales (p.ej. seguridad, derechos legales, etc.).

418 I'm A Teapot
Código "de broma": la petición del cliente es como pedirle a una tetera que sirva café.

https://datatracker.ietf.org/doc/html/rfc9110

2. Protocolos de comunicaciones - HTTP

Códigos de estado de respuesta HTTP

500 Internal Server Error
Hay un error en el servidor que no depende del servidor (p.ej. hosting).

501 Not Implemented
El servidor no soporta la petición del cliente.

502 Bad Gateway
El servidor funciona como puerta de enlace (gateway) hacia otro servidor y recibe una respuesta errónea por parte de ese otro servidor.

503 Service Temporarily Unavailable
El servidor no puede responder a la petición de manera temporal (p.ej. por sobrecarga, mantenimiento, etc.).

https://datatracker.ietf.org/doc/html/rfc9110

2. Protocolos de comunicaciones - HTTP

Códigos de estado de respuesta HTTP

504 Gateway Timeout
El servidor funciona como puerta de enlace (gateway) hacia otro servidor y no recibe respuesta por parte de ese otro servidor.

505 HTTP Version Not Supported
El servidor no soporta la versión de HTTP de la petición del cliente.

507 Insufficient Storage
El servidor no dispone de suficiente espacio libre para procesar la petición.

https://datatracker.ietf.org/doc/html/rfc9110

2. Protocolos de comunicaciones - HTTP

Ejemplo PHP

```php
<?php

function enviarRespuestaHTTP($codigo) {

      http_response_code($codigo);

       switch ($codigo) {
              case 200:
                     echo "Código 200: OK - La solicitud se procesó correctamente.";
                     break;
              case 201:
                     echo "Código 201: Created - El recurso fue creado correctamente.";
                     break;
              case 300:
                     echo "Código 300: Multiple Choices - Existen múltiples opciones para este recurso.";
                     break;
              case 400:
                     echo "Código 400: Bad Request - La solicitud es incorrecta o no se pudo procesar.";
                     break;
              case 401:
                     echo "Código 401: Unauthorized - Se requiere autenticación para acceder a este recurso.";
                     break;
              case 404:
                     echo "Código 404: Not Found - El recurso solicitado no se encontró.";
                     break;
              case 500:
                     echo "Código 500: Internal Server Error - El servidor encontró un error.";
                     break;
              default:
                     echo "Código $codigo: Código de respuesta no implementado en este ejemplo.";
                     break;
       }
}
```

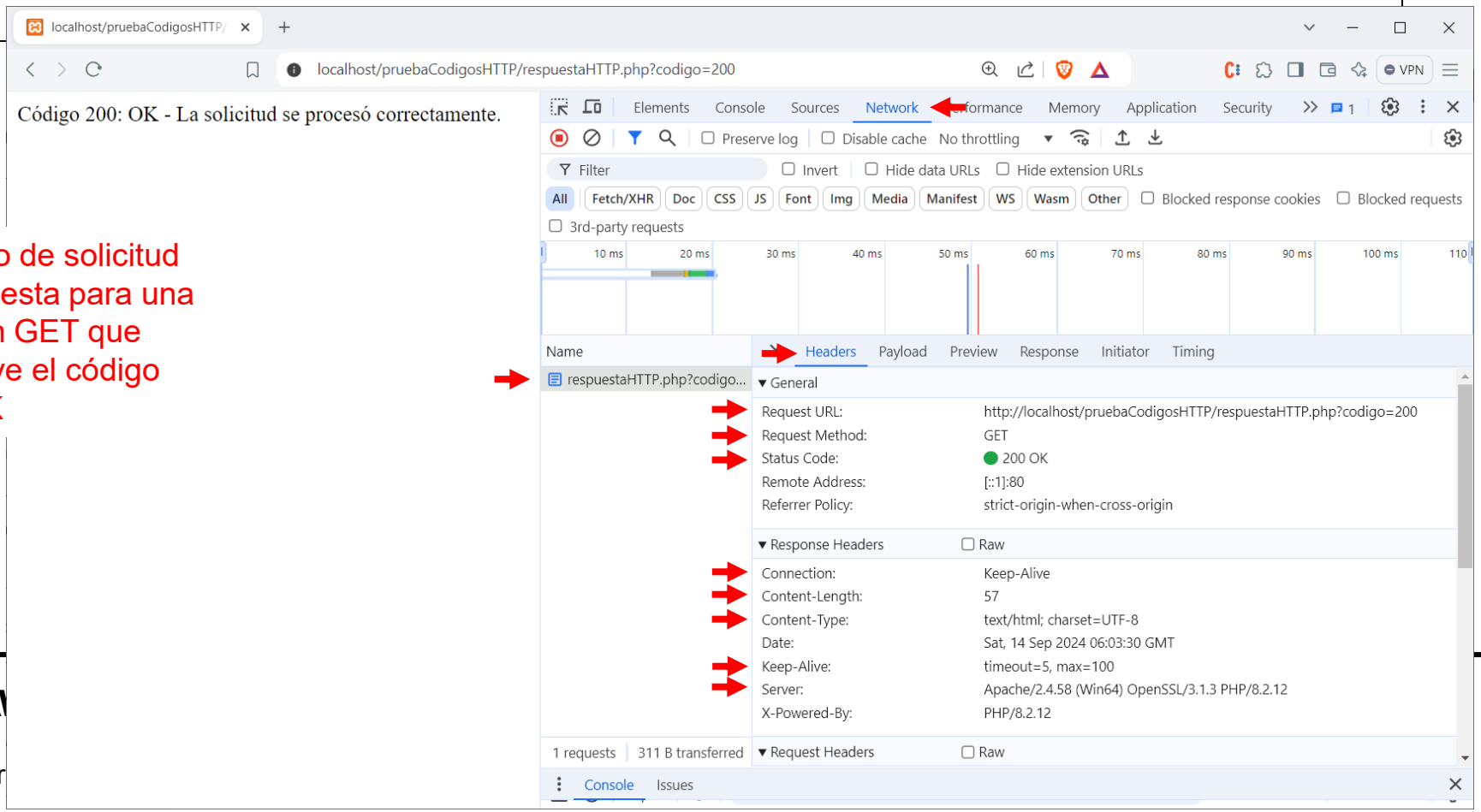
continua…

2. Protocolos de comunicaciones - HTTP

Ejemplo PHP

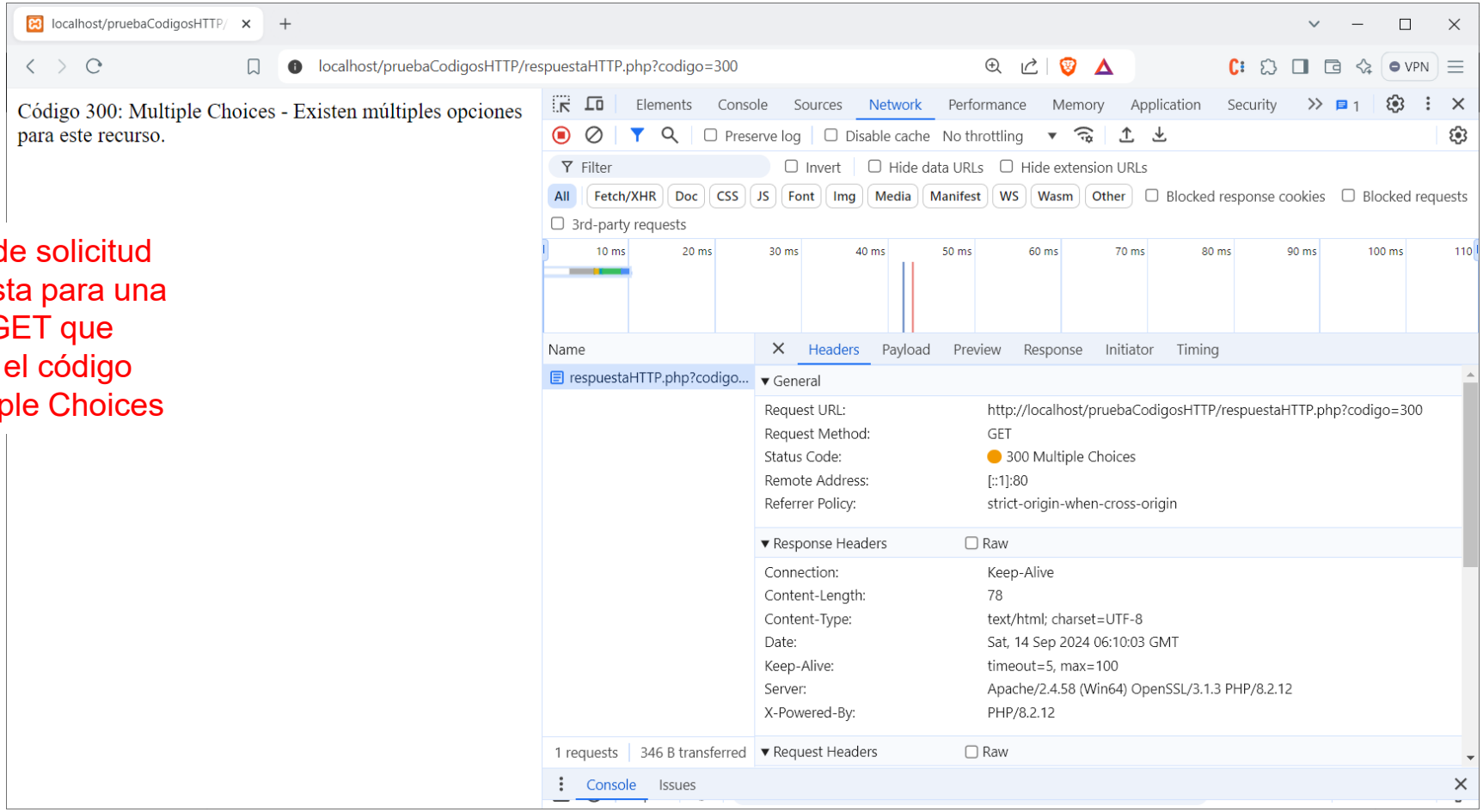
```
if ($_SERVER['REQUEST_METHOD'] === 'GET') {
        $codigoRespuesta = $_GET['codigo'];
        enviarRespuestaHTTP($codigoRespuesta);
}

?>
```

Ejemplo de solicitud y respuesta para una petición GET que devuelve el código 200 OK



2. Protocolos de comunicaciones - HTTP

Ejemplo PHP

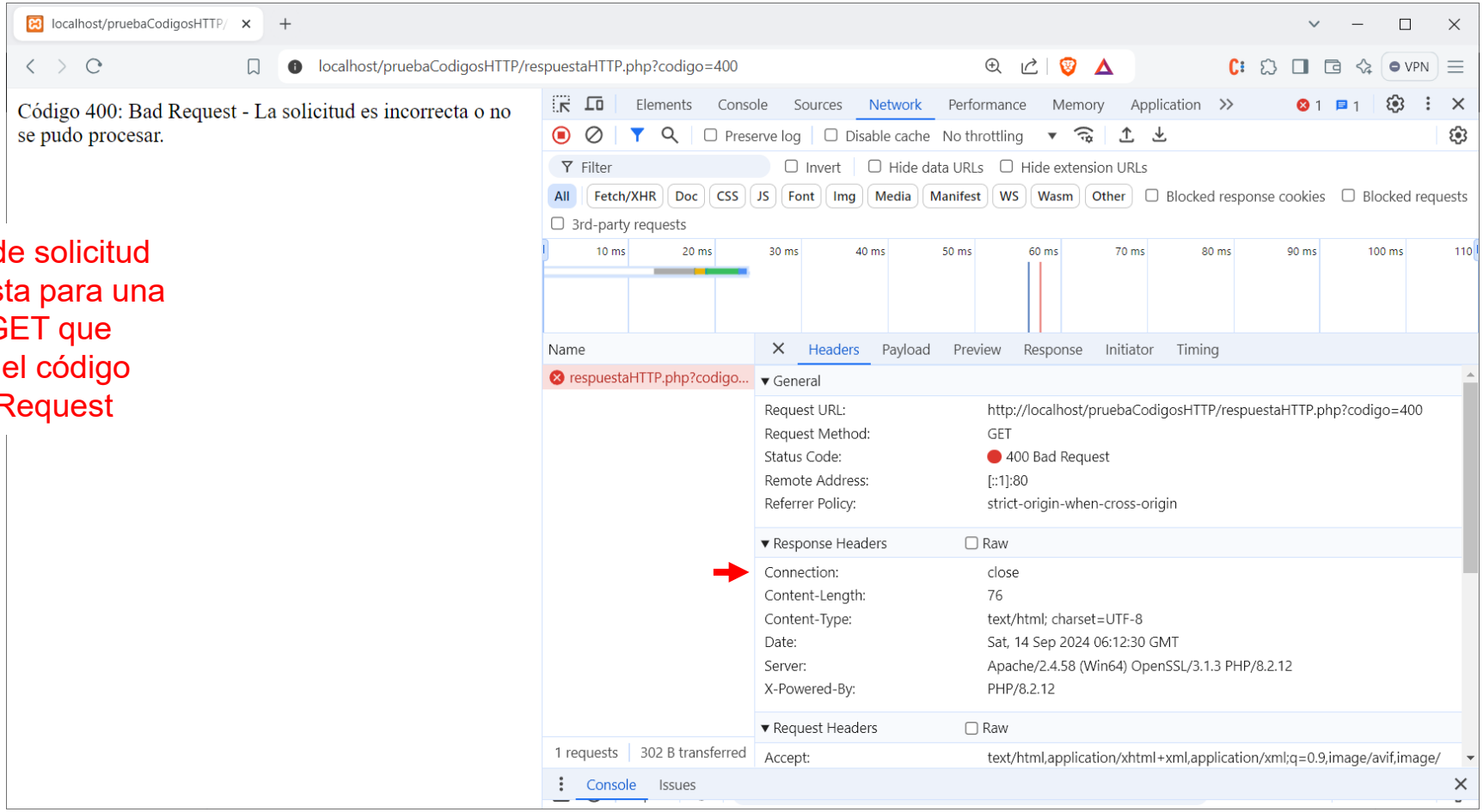


Ejemplo de solicitud y respuesta para una petición GET que devuelve el código 300 Multiple Choices

2. Protocolos de comunicaciones - HTTP

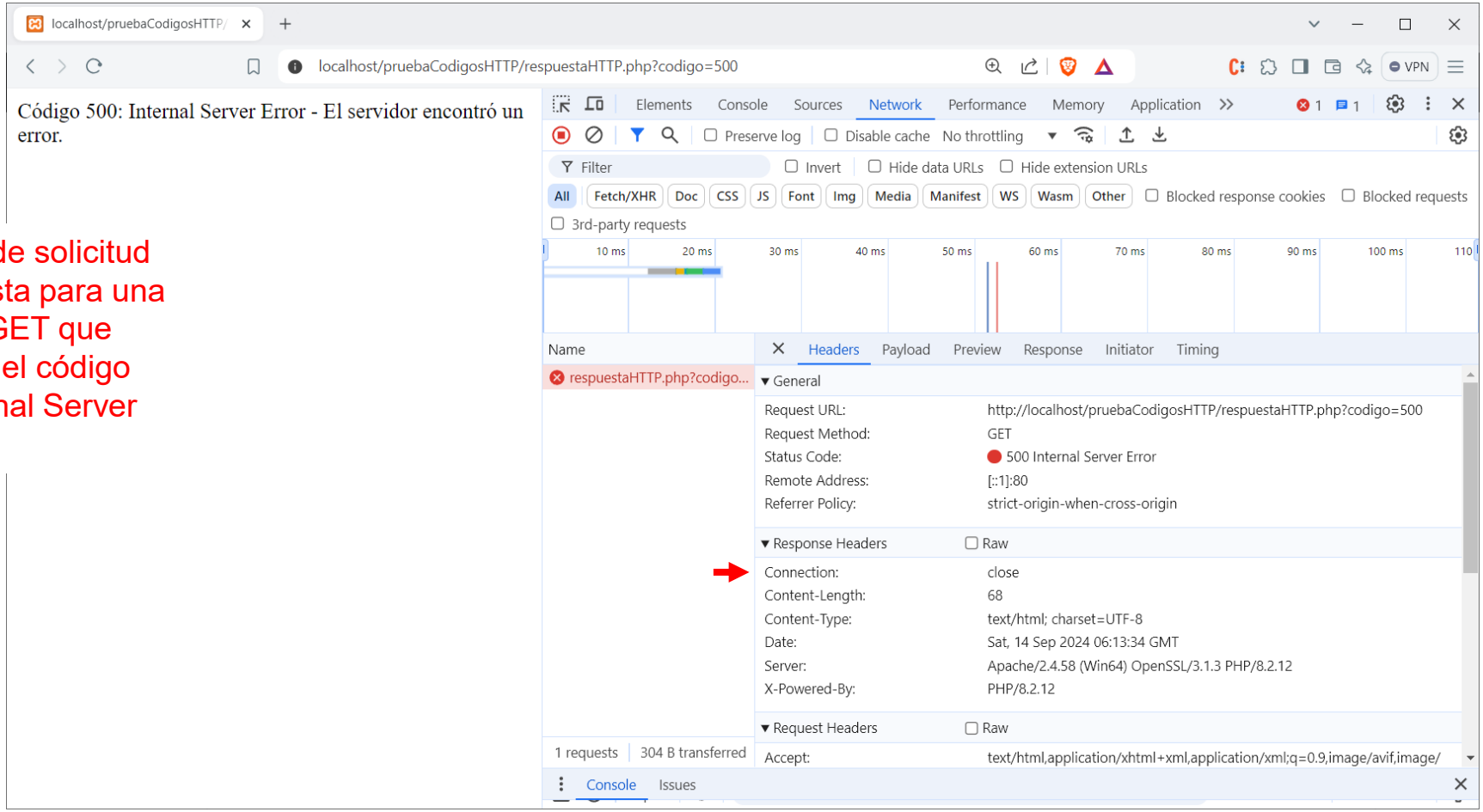
Ejemplo PHP

Ejemplo de solicitud y respuesta para una petición GET que devuelve el código 400 Bad Request



2. Protocolos de comunicaciones - HTTP

Ejemplo PHP



Ejemplo de solicitud y respuesta para una petición GET que devuelve el código 500 Internal Server Error

2. Protocolos de comunicaciones - HTTP

Ejemplo PHP: ampliar el script PHP para gestionar peticiones POST

```
<?php

function enviarRespuestaHTTP($codigo) {

	http_response_code($codigo);

	switch ($codigo) {
		case 200:
			echo "Código 200: OK - La solicitud se procesó correctamente.";
			break;
		case 201:
			echo "Código 201: Created - El recurso fue creado correctamente.";
			break;
		case 300:
			echo "Código 300: Multiple Choices - Existen múltiples opciones para este recurso.";
			break;
		case 400:
			echo "Código 400: Bad Request - La solicitud es incorrecta o no se pudo procesar.";
			break;
		case 401:
			echo "Código 401: Unauthorized - Se requiere autenticación para acceder a este recurso.";
			break;
		case 404:
			echo "Código 404: Not Found - El recurso solicitado no se encontró.";
			break;
		case 500:
			echo "Código 500: Internal Server Error - El servidor encontró un error.";
			break;
		default:
			echo "Código $codigo: Código de respuesta no implementado en este ejemplo.";
			break;
	}
}
```

continua…

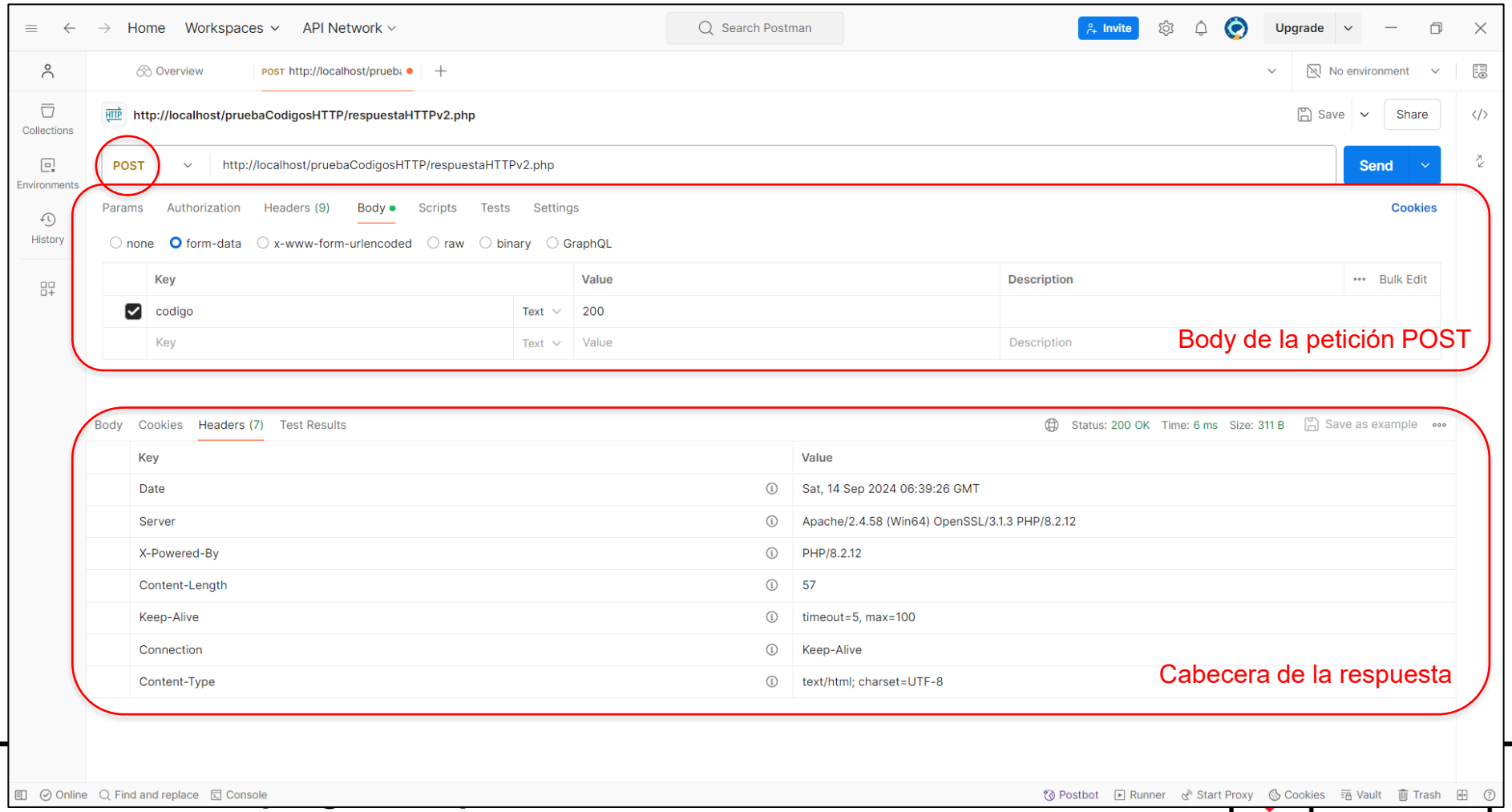
2. Protocolos de comunicaciones - HTTP

Ejemplo PHP: ampliar el script PHP para gestionar peticiones POST

```
if ($_SERVER['REQUEST_METHOD'] === 'GET') {

	$codigoRespuesta = $_GET['codigo'];
	enviarRespuestaHTTP($codigoRespuesta);

} elseif ($_SERVER['REQUEST_METHOD'] === 'POST') {

	$codigoRespuesta = $_POST['codigo'];
	enviarRespuestaHTTP($codigoRespuesta);

} else {

	http_response_code(405);
	echo "Código 405: Method Not Allowed - Solo se permiten solicitudes GET y POST.";

}

?>
```

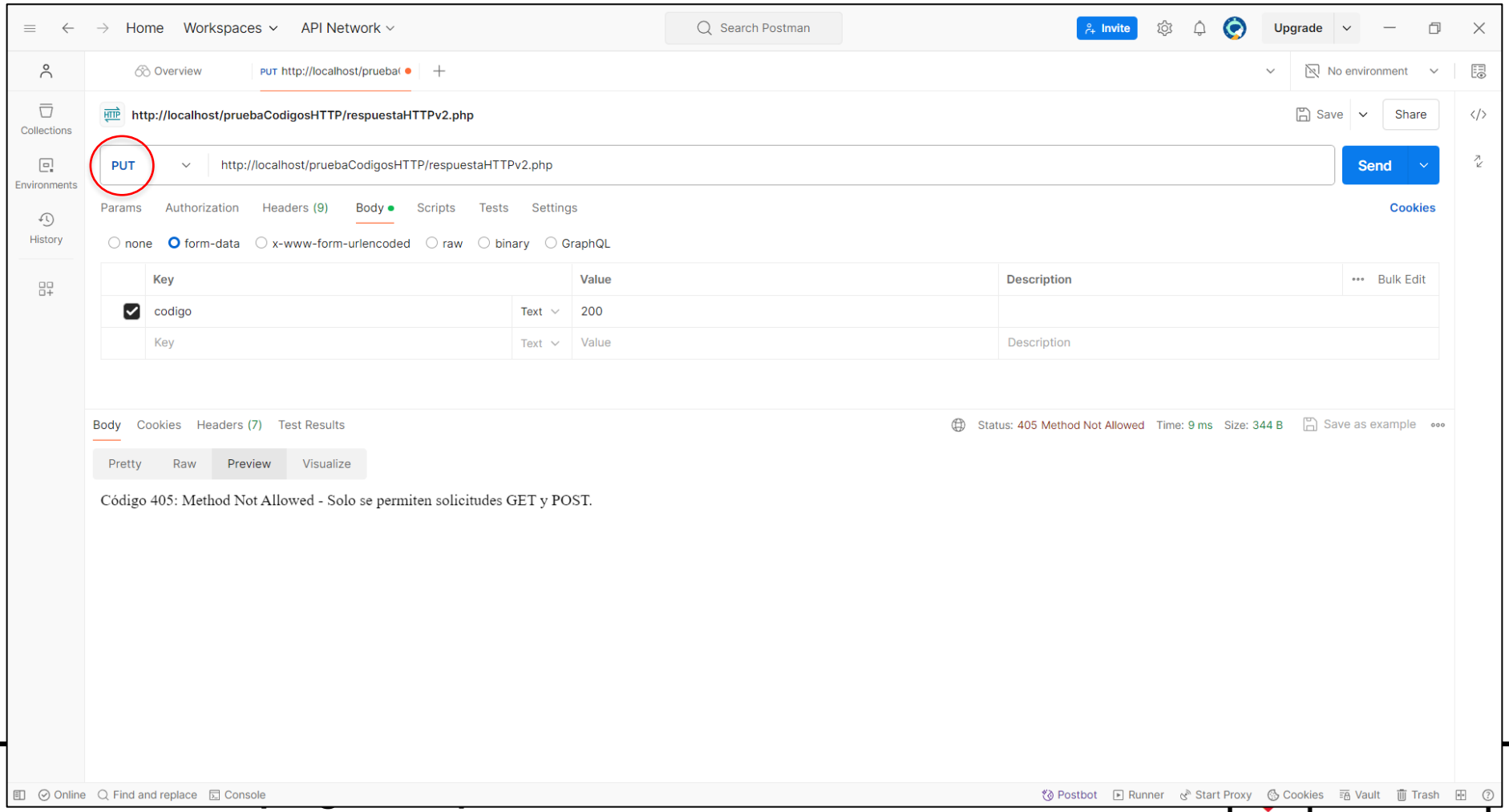
2. Protocolos de comunicaciones - HTTP

Ejemplo PHP: ampliar el script PHP para gestionar peticiones POST



2. Protocolos de comunicaciones - HTTP

Ejemplo PHP: ampliar el script PHP para gestionar peticiones POST



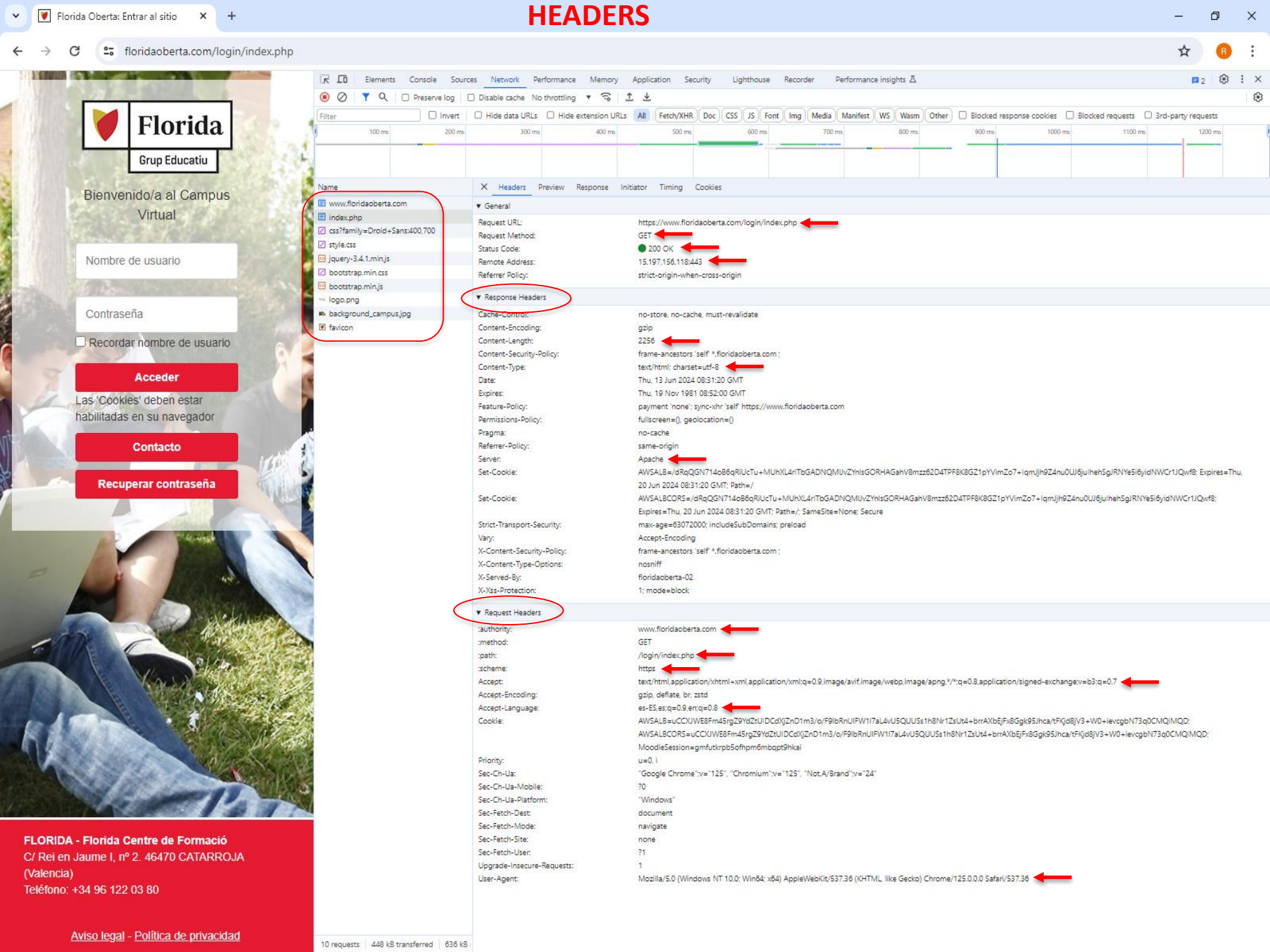
2. Protocolos de comunicaciones - HTTP

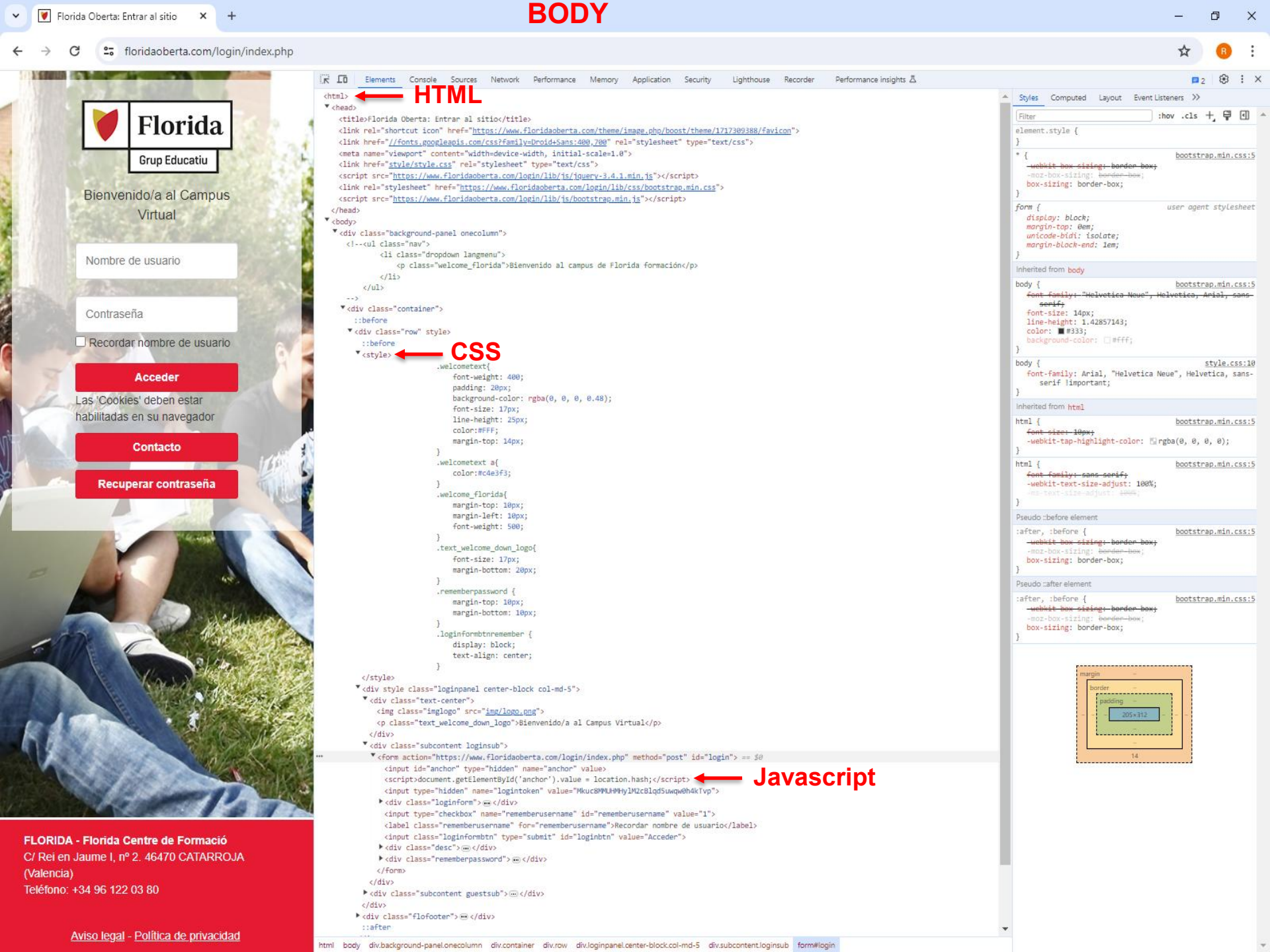
Sugerencia

Implementa un sencillo formulario web para poder hacer las peticiones POST desde el navegador web, en vez de utilizar Postman.

Al hacer clic en “Enviar”, debe llamar al script PHP anterior.

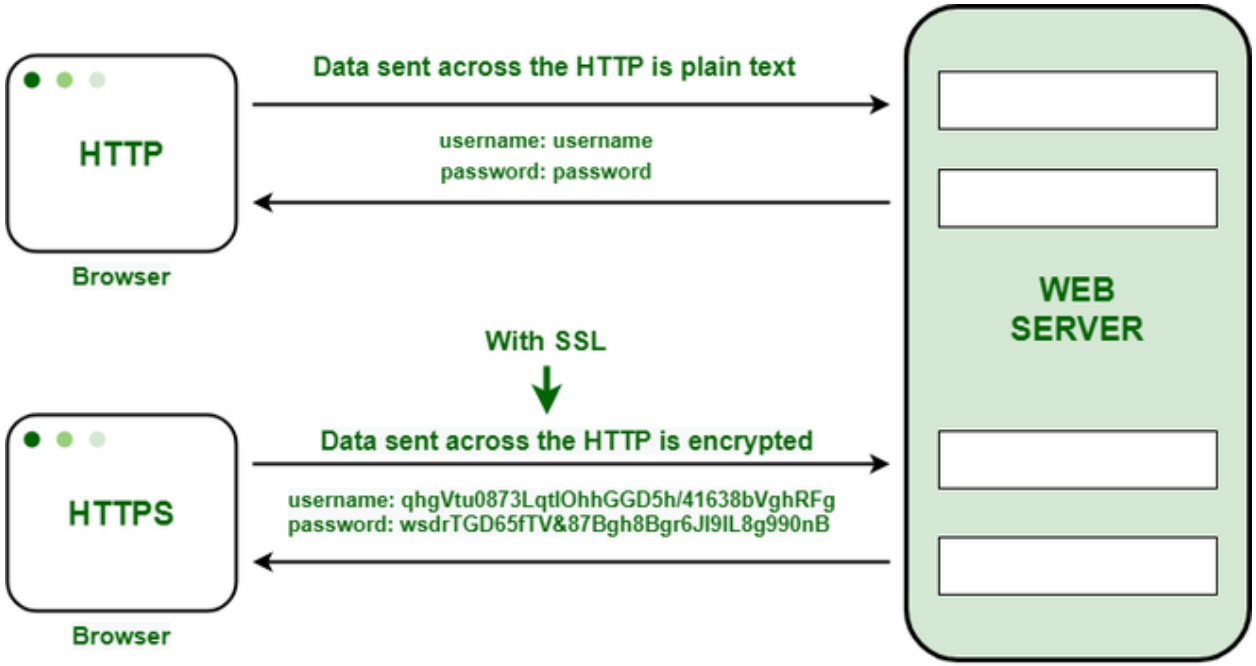
HEADERS


Florida Oberta: Entrar al sitio
floridaoberta.com/login/index.php
Florida
Grup Educatiu
Bienvenido/a al Campus Virtual
Nombre de usuario
Contraseña
Recordar nombre de usuario
Acceder
Las 'Cookies' deben estar habilitadas en su navegador
Contacto
Recuperar contraseña
FLORIDA - Florida Centre de Formació
C/ Rei en Jaume I, nº 2. 46470 CATARROJA (Valencia)
Teléfono: +34 96 122 03 80
Aviso legal - Política de privacidad
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights
Name
www.floridaoberta.com
index.php
css?family=Droid+Sans:400,700
style.css
jquery-3.4.1.min.js
bootstrap.min.css
bootstrap.min.js
logo.png
background_campus.jpg
favicon
Headers Preview Response Initiator Timing Cookies
General
Request URL: https://www.floridaoberta.com/login/index.php
Request Method: GET
Status Code: 200 OK
Remote Address: 15.197.156.118:443
Referrer Policy: strict-origin-when-cross-origin
Response Headers
Cache-Control: no-store, no-cache, must-revalidate
Content-Encoding: gzip
Content-Length: 2256
Content-Security-Policy: frame-ancestors 'self' *.floridaoberta.com ;
Content-Type: text/html; charset=utf-8
Date: Thu, 13 Jun 2024 08:31:20 GMT
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Feature-Policy: payment 'none'; sync-xhr 'self' https://www.floridaoberta.com
Permissions-Policy: fullscreen=(), geolocation=()
Pragma: no-cache
Referrer-Policy: same-origin
Server: Apache
Set-Cookie: AWSALB=/dRqQGN714oB6qRiUcTu+MUhXL4rITbGADNQMIUvZYnlsGORHAGahV8mzz62D4TPF8K8GZ1pYVimZo7+lqmJjh9Z4nu0UJ6juIhehSgJRNYe5i6yidNWCr1JQwf8; Expires=Thu, 20 Jun 2024 08:31:20 GMT; Path=/
Set-Cookie: AWSALBCORS=/dRqQGN714oB6qRiUcTu+MUhXL4rITbGADNQMIUvZYnlsGORHAGahV8mzz62D4TPF8K8GZ1pYVimZo7+lqmJjh9Z4nu0UJ6juIhehSgJRNYe5i6yidNWCr1JQwf8; Expires=Thu, 20 Jun 2024 08:31:20 GMT; Path=/; SameSite=None; Secure
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
Vary: Accept-Encoding
X-Content-Security-Policy: frame-ancestors 'self' *.floridaoberta.com ;
X-Content-Type-Options: nosniff
X-Served-By: floridaoberta-02
X-Xss-Protection: 1; mode=block
Request Headers
:authority: www.floridaoberta.com
:method: GET
:path: /login/index.php
:scheme: https
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=0.7
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: es-ES,es;q=0.9,en;q=0.8
Cookie: AWSALB=uCCXJWE8Fm45rgZ9YdZtUIDCdXjZnD1m3/o/F9IbRnUIFW1I7aL4vU5QUUSs1h8Nr1ZsUt4+brrAXbEjFx8Ggk95Jhca/tFKjd8jV3+W0+ievcgbN73q0CMQIMQD; AWSALBCORS=uCCXJWE8Fm45rgZ9YdZtUIDCdXjZnD1m3/o/F9IbRnUIFW1I7aL4vU5QUUSs1h8Nr1ZsUt4+brrAXbEjFx8Ggk95Jhca/tFKjd8jV3+W0+ievcgbN73q0CMQIMQD; MoodleSession=gmfutkrpb5ofhpm6mbqpt9hkai
Priority: u=0, i
Sec-Ch-Ua: "Google Chrome";v="125", "Chromium";v="125", "Not.A/Brand";v="24"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36
10 requests 448 kB transferred 636 kB

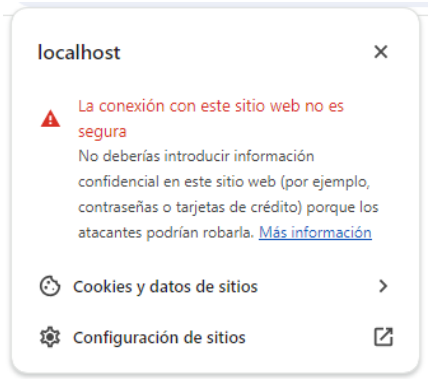

BODY
Florida Oberta: Entrar al sitio
floridaoberta.com/login/index.php
Florida
Grup Educatiu
Bienvenido/a al Campus Virtual
Nombre de usuario
Contraseña
Recordar nombre de usuario
Acceder
Las 'Cookies' deben estar habilitadas en su navegador
Contacto
Recuperar contraseña
FLORIDA - Florida Centre de Formació
C/ Rei en Jaume I, nº 2. 46470 CATARROJA (Valencia)
Teléfono: +34 96 122 03 80
Aviso legal - Política de privacidad
HTML
CSS
Javascript
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Performance insights
Styles
Computed
Layout
Event Listeners

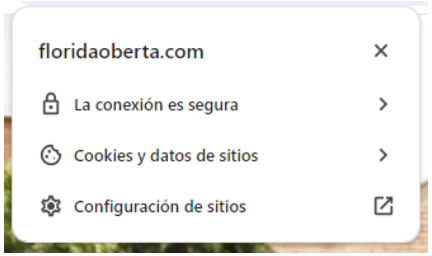
2. Protocolos de comunicaciones - HTTPS

En el protocolo HTTP los mensajes viajan en texto plano

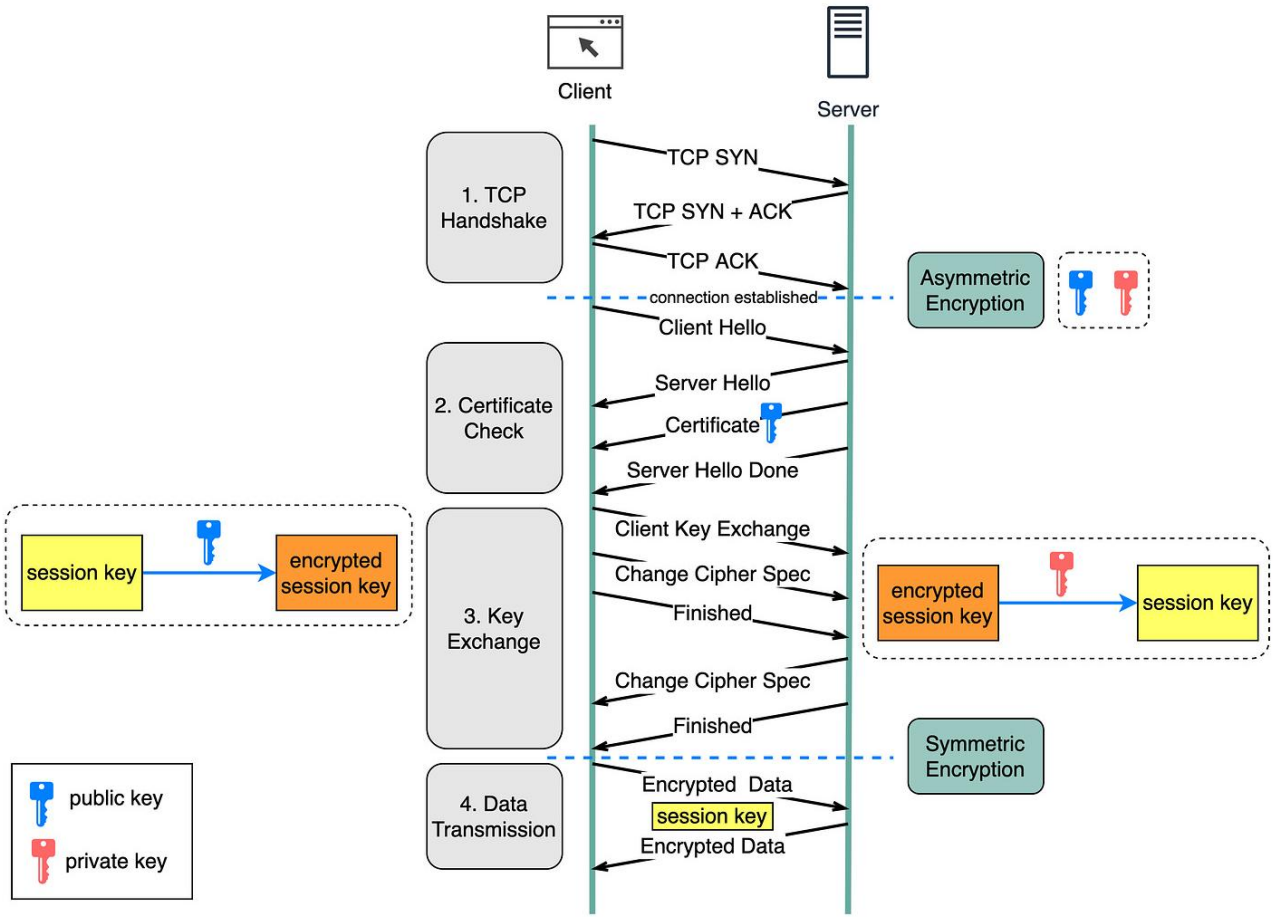


geeksforgeeks.org





2. Protocolos de comunicaciones - HTTPS



Conexión HTTP cliente-servidor

Se acuerda qué sistema de encriptación (TLS) se va a usar. El servidor envía su certificado digital (SSL) al cliente (incluye su clave pública).

El cliente valida el certificado y genera una clave de sesión con él y se la envía encriptada al servidor utilizando su clave pública. El servidor recibe la clave y la desencripta con su clave privada.

El resto de los mensajes se encriptan con la clave de sesión.

https://blog.bytebytego.com/p/how-does-https-work-episode-6

2. Protocolos de comunicaciones - HTTPS

Algunas consideraciones (el precio de la seguridad):

Necesidad de autoridades certificadoras: entidades confiables que emiten certificados digitales a personas, organismos, empresas, etc. comprobando previamente ciertos requisitos legales.
Aunque puedes crear tu propio certificado digital (p.ej. en Java), no será admitido como fiable por ningún navegador actual.

Pérdida de eficiencia:

- Encriptar y desencriptar mensajes implica más consumo de CPU (latencia).
- Los servicios de caché de servidores intermedios no trabajan de forma tan eficaz, por lo que las actualizaciones de contenido a través de Internet son más lentas.

¿Hay que sustituir completamente HTTP por HTTPS?
Depende de nuestras necesidades de seguridad o de la fase en que se encuentra una aplicación (p.ej. desarrollo inicial).

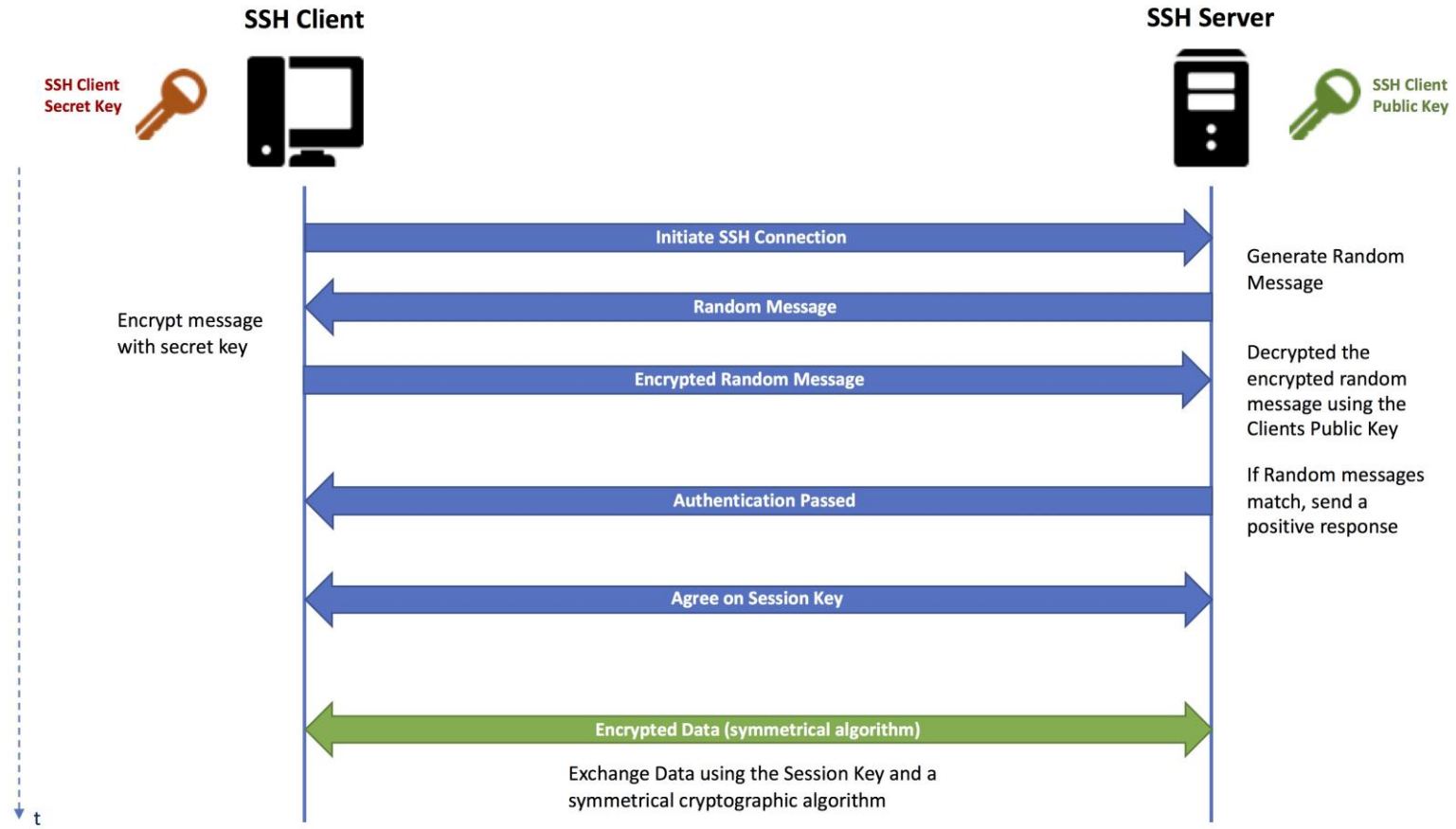
2. Protocolos de comunicaciones - SSH

SSH: ***S**ecure **SH**ell* (intérprete de órdenes seguro)

- Acceso seguro (encriptado) a máquinas remotas para manejarlas por completo a través de línea de comandos o interfaz gráfica (servidor X).
- Copia de datos de forma segura.
- Evolución del protocolo Telnet (texto plano ➔ no seguro).
- Se basa en el uso de claves
  - Criptografía asimétrica para autenticación y establecimiento de la conexión (claves pública y privada).
  - Criptografía simétrica (clave acordada) para encriptar/desencriptar los datos que se transfieren a través de la conexión (túnel SSH).

2. Protocolos de comunicaciones - SSH

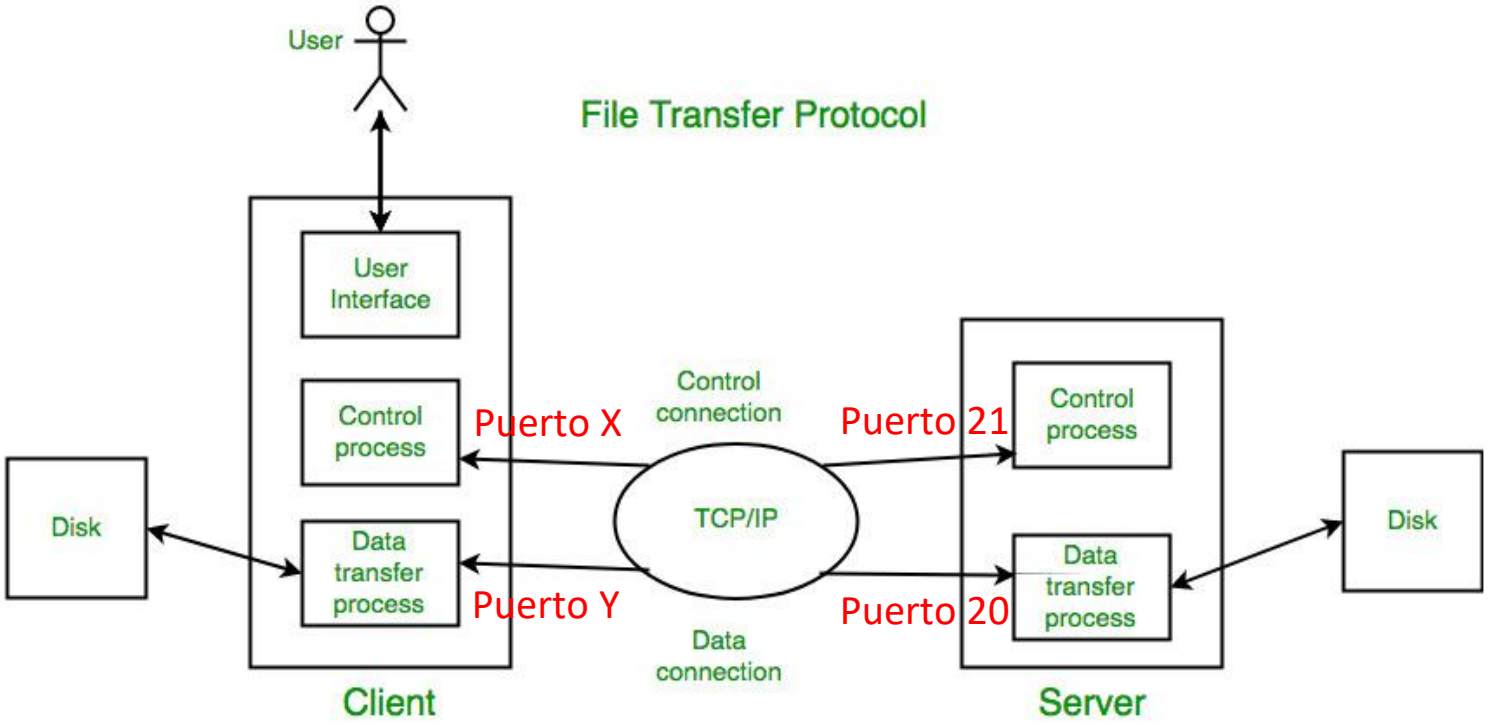
SSH: ***S**ecure **SH**ell* (intérprete de órdenes seguro)



https://dev.vividbreeze.com/ssh-protocol-and-key-generation/

2. Protocolos de comunicaciones - FTP

FTP: ***File Transfer Protocol*** (protocolo de transferencia de ficheros)



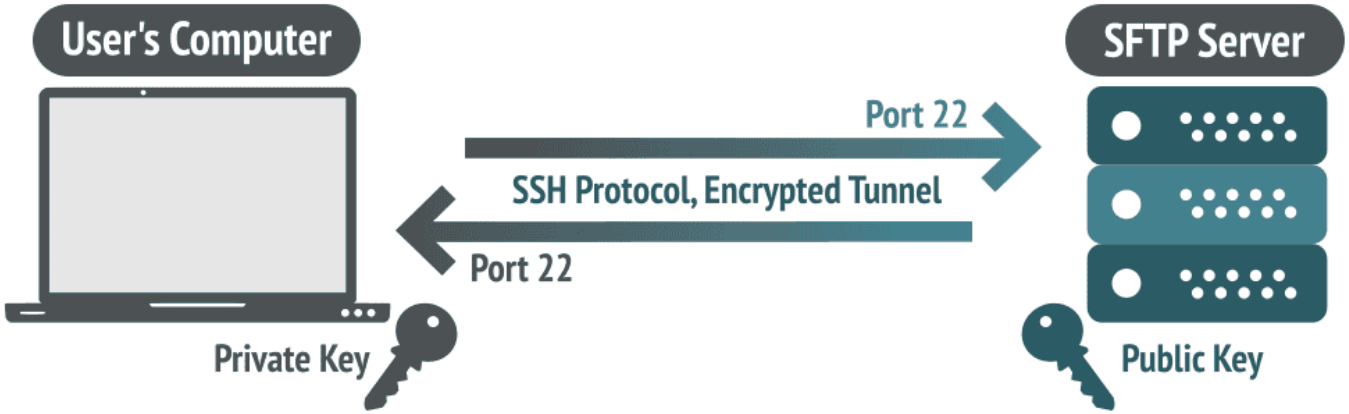
geeksforgeeks.org

2. Protocolos de comunicaciones - SFTP

SFTP: ***S**ecure Shell **F**ile **T**ransfer **P**rotocol*

Utiliza claves SSH para encriptación

Permite autenticación con usuario/contraseña o con claves SSH



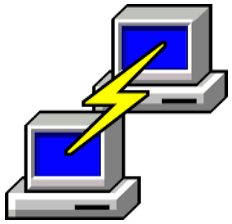
linuxiac.com

2. Protocolos de comunicaciones - SSH y SFTP

Ejemplos

Servidores
OpenSSH: SSH/SFTP Linux
freeSSHd: SSH/SFTP Windows
FileZilla: FTP (SFTP de pago)

Clientes



SSH

https://www.putty.org/

SFTP

https://filezilla-project.org/download.php?show_all=1

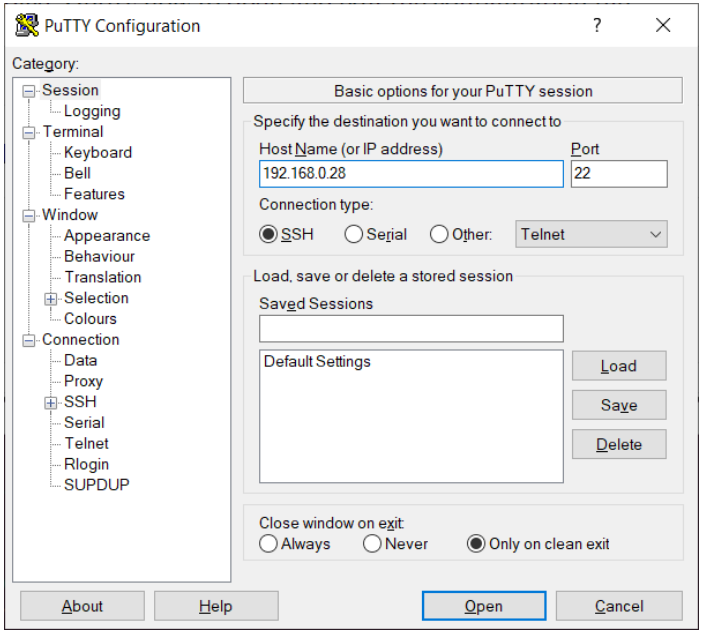
2. Protocolos de comunicaciones - SSH y SFTP

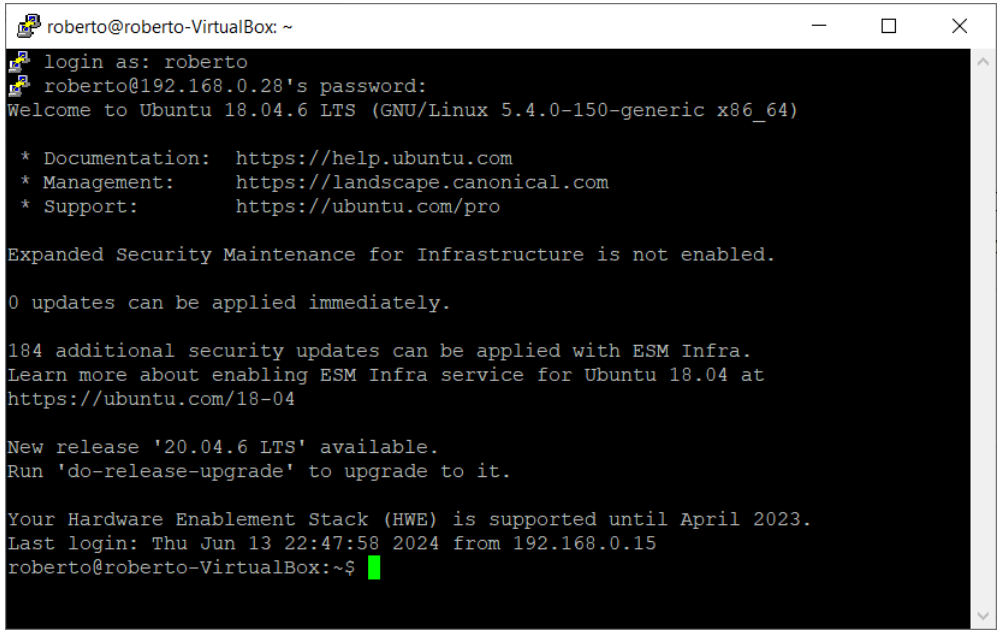
OpenSSH (utilizar la máquina virtual Ubuntu del tema 1 u otra distribución Linux)

1. Abrir un terminal e instalar OpenSSH: `sudo apt install openssh-server`
2. Comprobar que aparece "*active (running)*": `sudo systemctl status ssh`
3. En caso de que no esté activo, iniciarlo manualmente:
   ```
   sudo systemctl enable ssh
   sudo systemctl start ssh
   ```
4. Abrir el puerto SSH: `sudo ufw allow ssh`
5. Para modificar el puerto (22 por defecto), editar el fichero de configuración:
   ```
   sudo nano /etc/ssh/sshd_config
   ```

2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH (utilizar la máquina virtual Ubuntu del tema 1 u otro Linux)
Ahora el servidor SSH ya está operativo, así que podemos conectarnos vía PuTTy desde la máquina Windows:





NOTA RECORDATORIA TEMA 1: para saber la IP que tiene la MV Linux, introduce en un terminal el comando **ip addr**

Ahora este terminal tiene la misma funcionalidad y permisos (para nuestro usuario) que el terminal de la máquina Linux, es decir, estamos “dentro” de la máquina Linux como “roberto"

2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH (utilizar la máquina virtual Ubuntu del tema 1 u otro Linux)
El siguiente paso es configurar un servidor SFTP partiendo del servidor SSH:

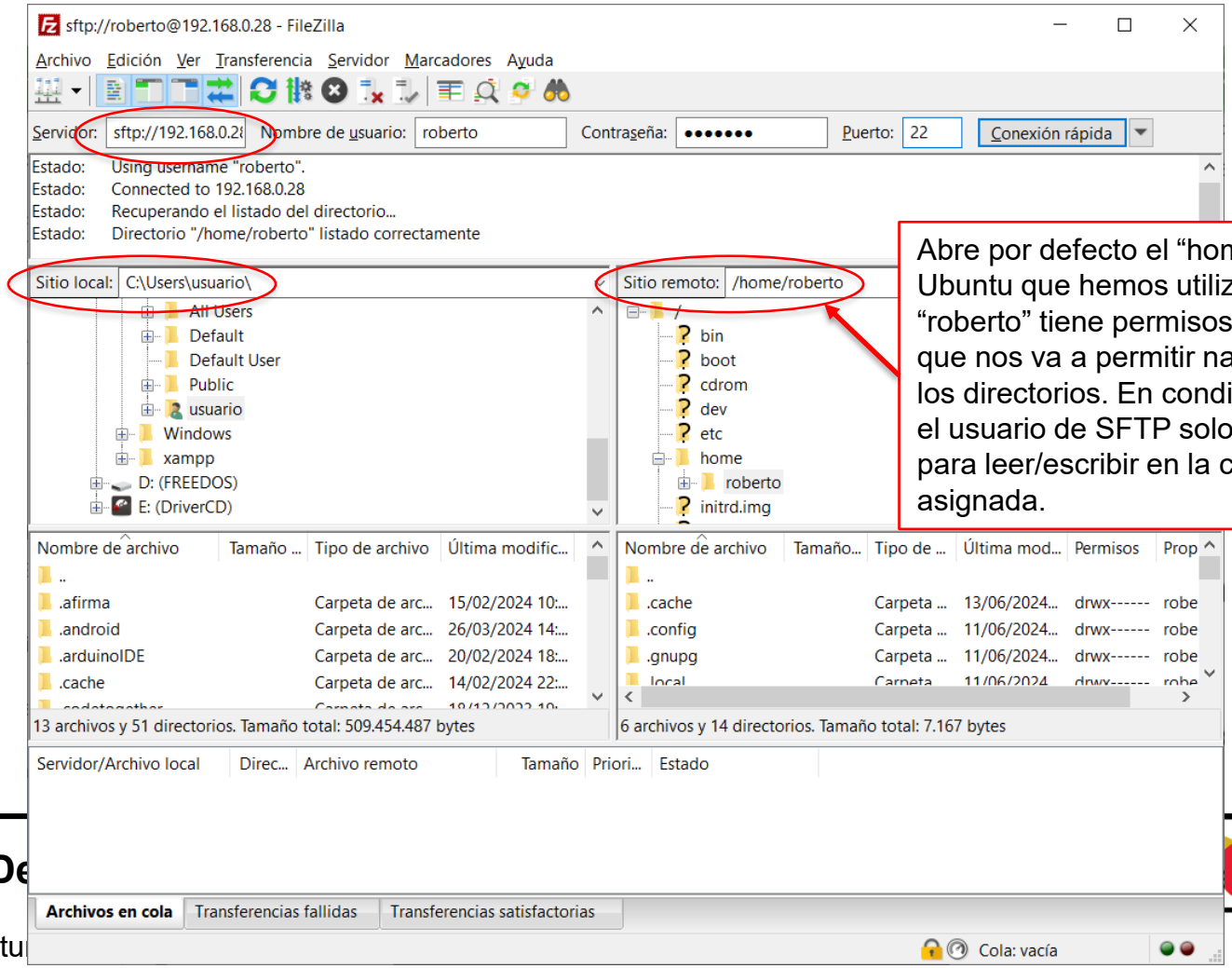
1. Abrir un Terminal
2. Abrir la configuración del servidor SSH: `sudo nano /etc/ssh/sshd_config`
3. Añadir al final las siguientes líneas:

```
Match group sftp
ChrootDirectory /home
X11Forwarding no
AllowTcpForwarding no
ForceCommand internal-sftp
```

4. Guardar el fichero: `CTRL+X, Y, ENTER`
5. Reiniciar el servidor SSH: `sudo systemctl restart ssh`
6. Crear un grupo SFTP: `sudo addgroup sftp`
7. Crear un nuevo usuario SFTP: `sudo useradd -m usuario -g sftp`
8. Establecer su contraseña: `sudo passwd contrasenya`

2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH (utilizar la máquina virtual Ubuntu del tema 1 u otro Linux)
Ahora el servidor SFTP ya está operativo y nos podemos conectar con FileZilla:



Abre por defecto el "home" del usuario de Ubuntu que hemos utilizado. En este caso, "roberto" tiene permisos de "root", por lo que nos va a permitir navegar por todos los directorios. En condiciones normales, el usuario de SFTP solo estará habilitado para leer/escribir en la carpeta que tenga asignada.

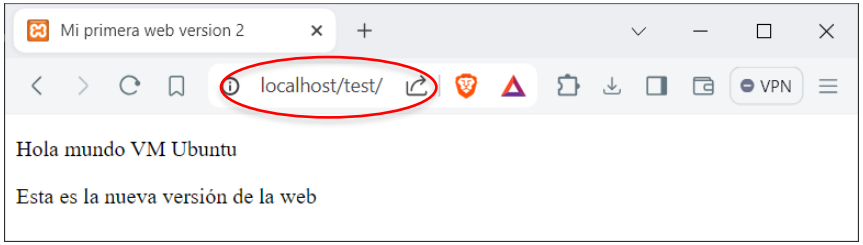
2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH (utilizar la máquina virtual Ubuntu del tema 1 u otro Linux)

Imaginemos ahora que hemos actualizado nuestra web de “Hola mundo”, realizando todo el desarrollo y testeo correspondientes en local. Ahora queremos desplegar la nueva web en nuestro servidor Apache de producción que, recordemos, está alojado en la máquina virtual de Ubuntu. Basta con acceder por SFTP y actualizar el directorio.

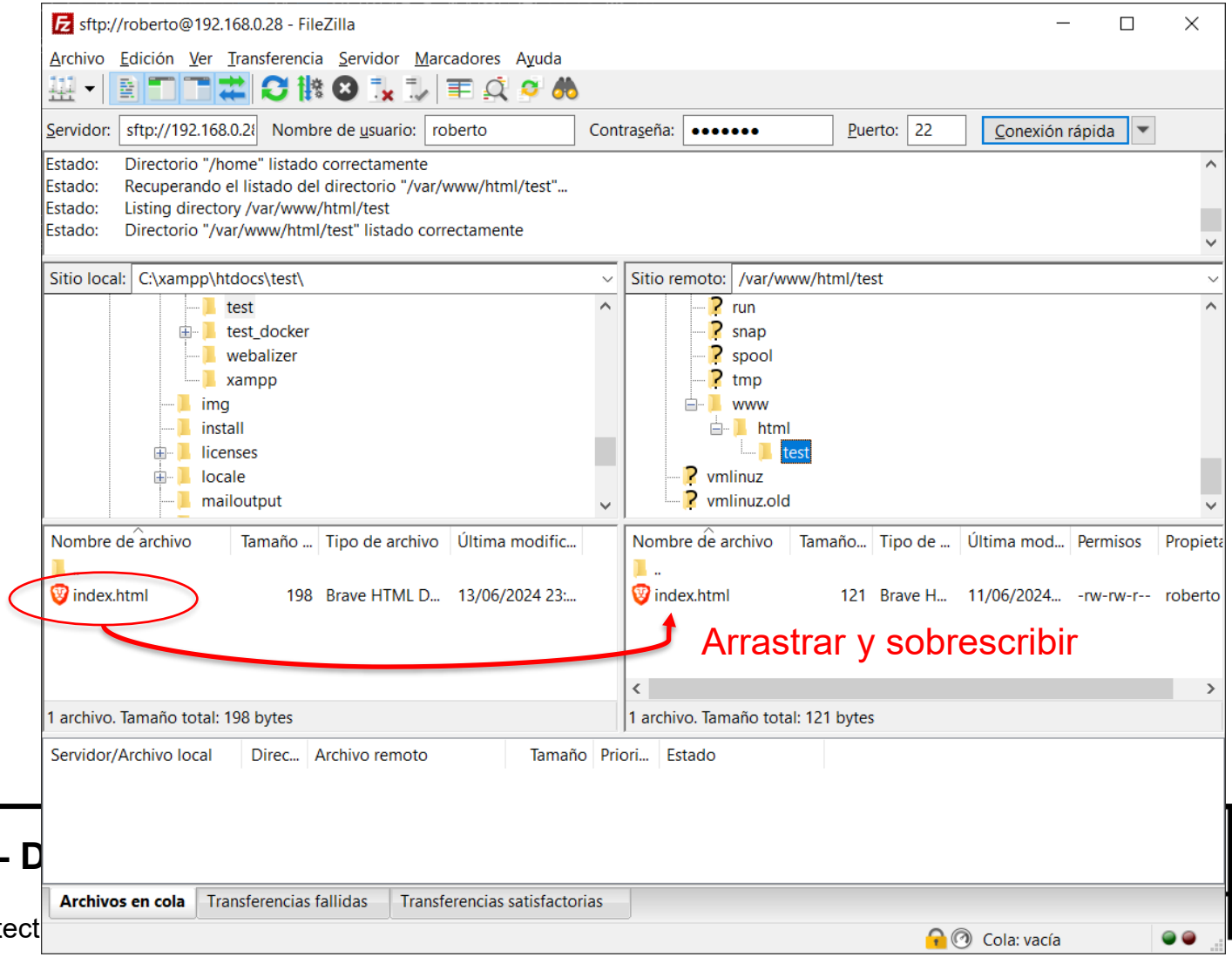
Recordemos… Esta es la versión que actualmente está desplegada en la MV Ubuntu:

Y esta es la nueva web que hemos desarrollado y testeado en local:



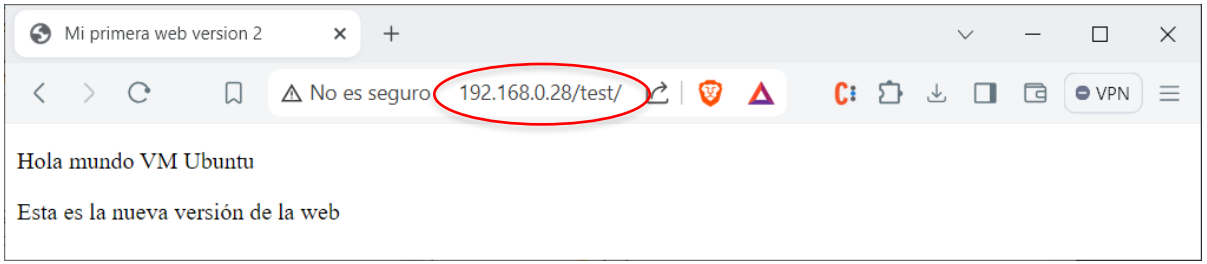
2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH (utilizar la máquina virtual Ubuntu del tema 1 u otro Linux)



2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH (utilizar la máquina virtual Ubuntu del tema 1 u otro Linux)



☺ Despliegue realizado con éxito, la versión nueva de la web ya está en producción y es accesible para nuestros clientes.

NOTA: es posible que el navegador tenga guardada en caché la versión antigua de la web. Para forzar que se traiga la nueva versión, es aconsejable recargar la web con CTRL + F5, ya que con F5 es posible que se cargue otra vez la versión en caché.

2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH: configuración de acceso para EC2 Ubuntu de AWS

La **máquina EC2 de AWS con Ubuntu** que utilizamos en el módulo no permite el acceso SSH (o SFTP) con usuario/contraseña, sino que se necesita un par de claves.

Lo primero que hay que hacer es descargar la clave del AWS Academy Learner Lab. Como utilizaremos PuTTy como cliente SSH, descarga el fichero PPK (PuTTy Private Key) del AWS Academy Learner Lab (desde AWS details).

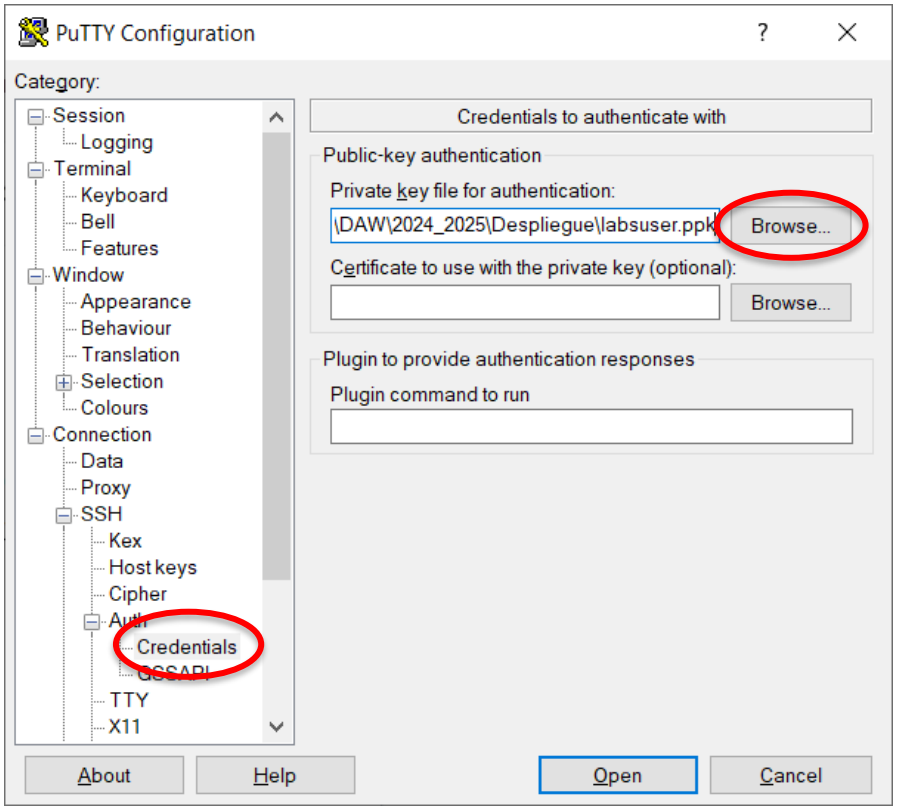
Este fichero es la clave privada que almacenarás de manera segura como usuario. Por otro lado, la clave pública está almacenada en la instancia EC2.

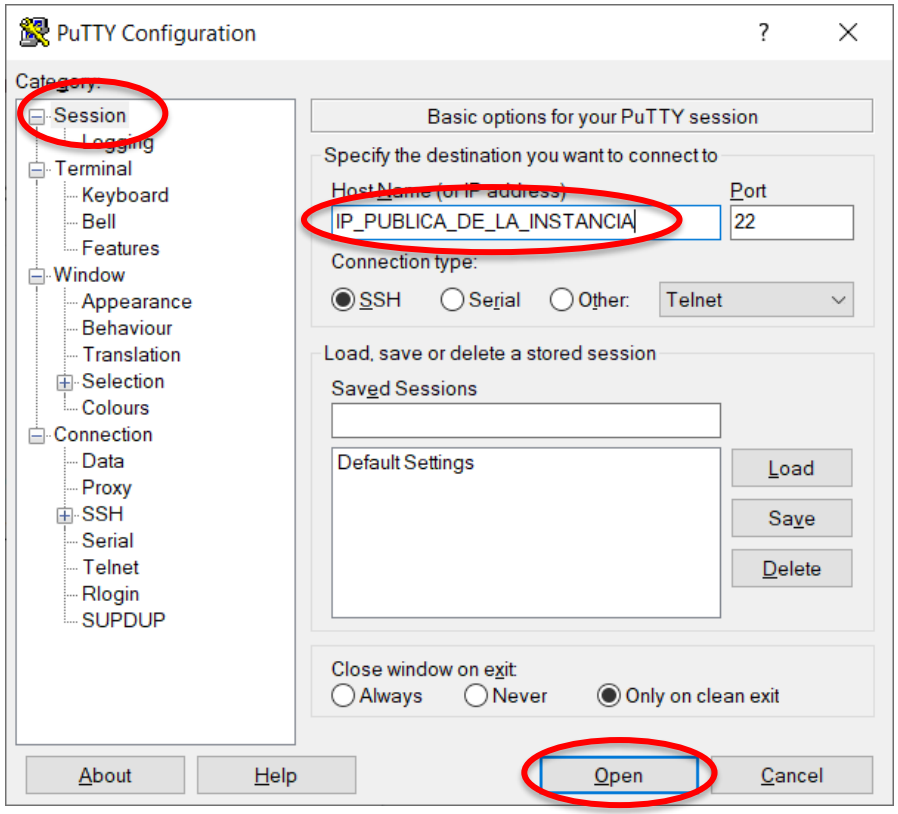
A continuación, vamos a ver cómo configurar el acceso vía SSH con PuTTy y el acceso vía SFTP con FileZilla.

2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH: configuración de acceso para EC2 Ubuntu de AWS

En Windows abre un cliente SSH (p.ej. PuTTy). Configurar "*Credentials*" y "*Session*".

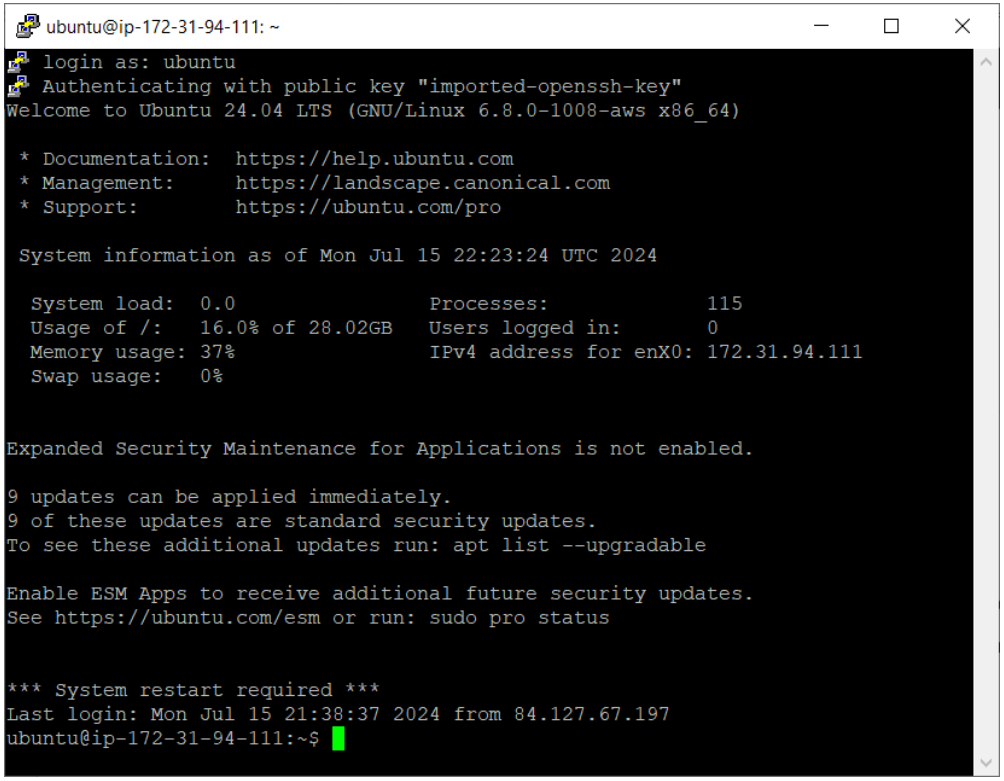




2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH: configuración de acceso para EC2 Ubuntu de AWS

Le indicamos que sí aceptamos la conexión e indicamos que se conecte como usuario “ubuntu” (no solicitará contraseña porque ya accedemos con clave privada).

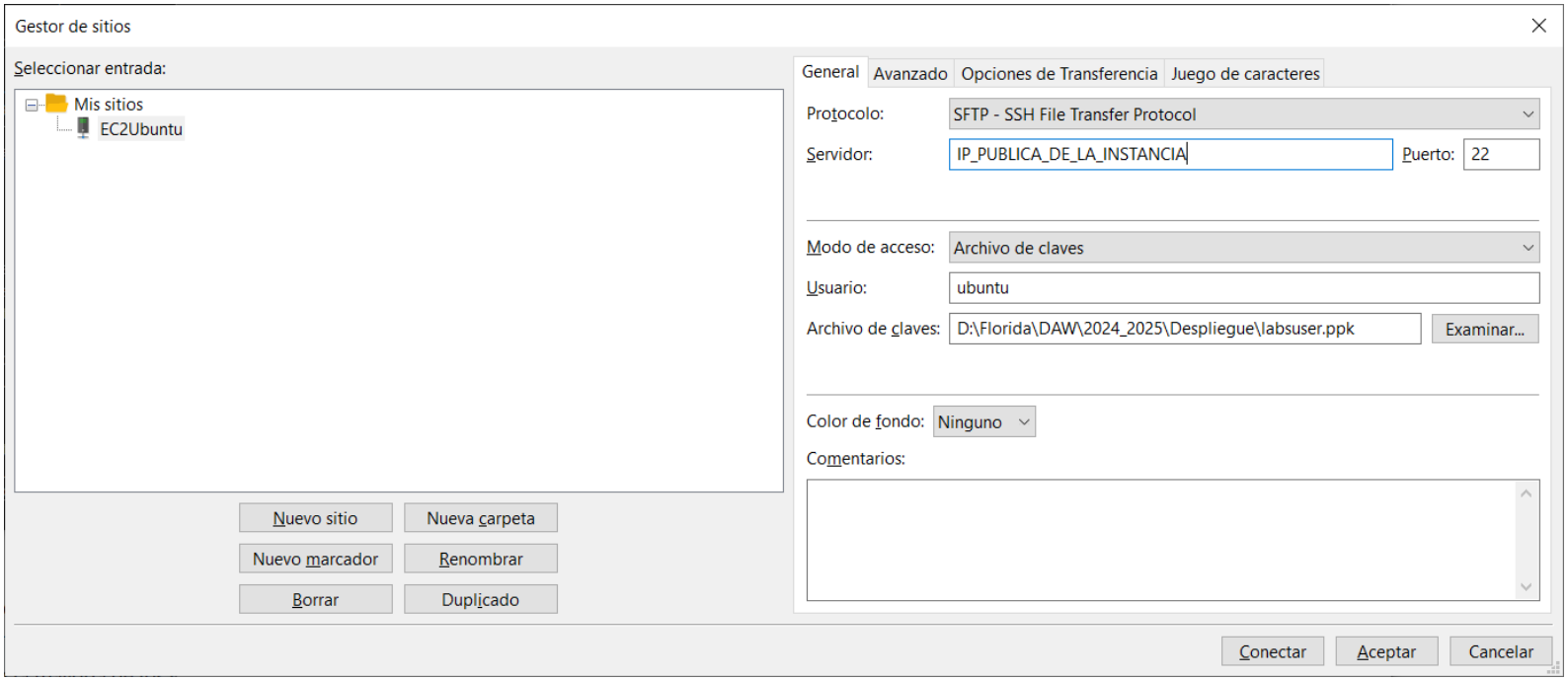


Ya estamos “dentro” de la instancia EC2 de AWS.

2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH: configuración de acceso para EC2 Ubuntu de AWS

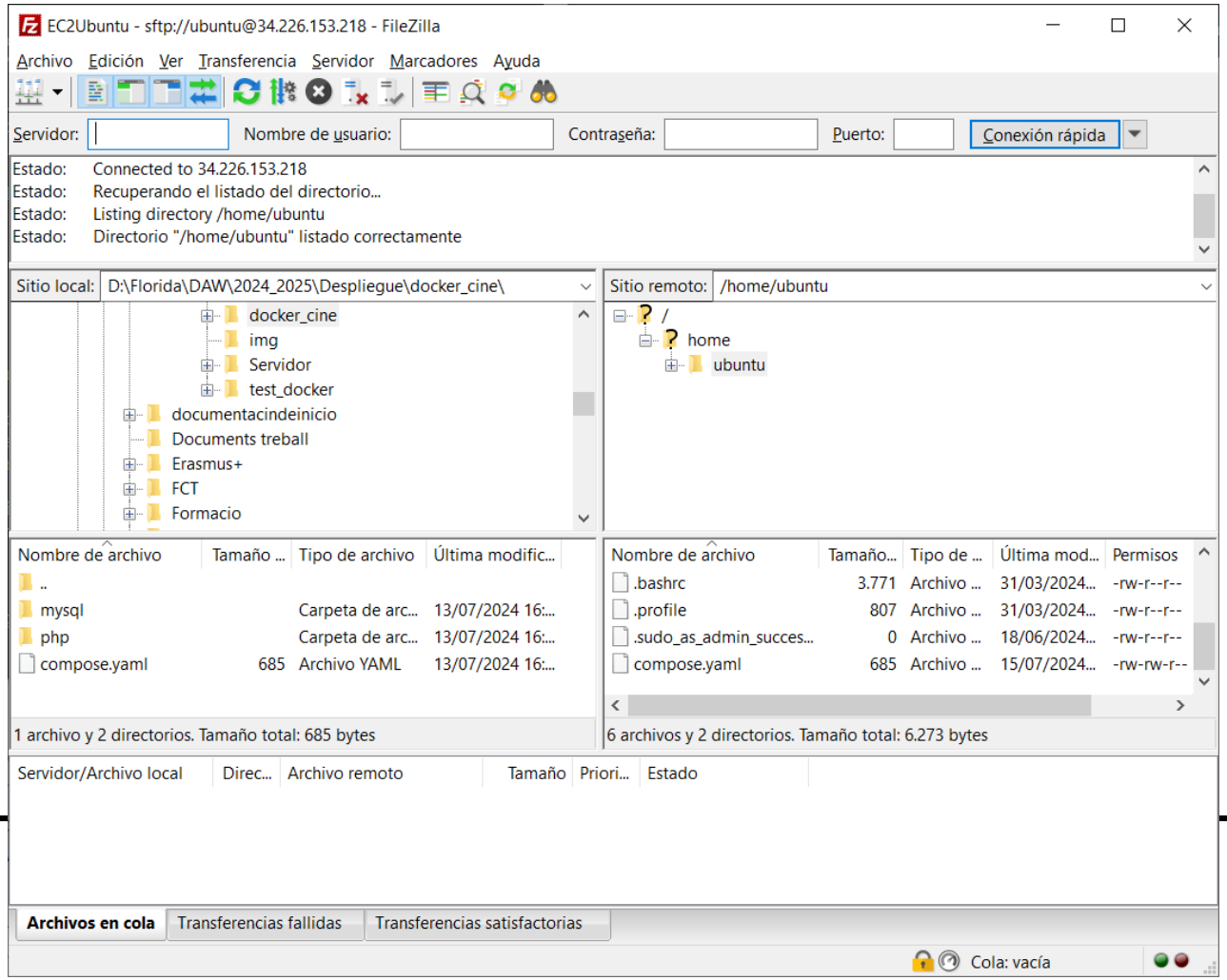
En Windows abre un cliente FTP (p.ej. FileZilla). Accede a Archivo → Gestor de sitios. Creamos un nuevo sitio indicando el protocolo (SFTP), la IP pública y puerto de nuestra máquina EC2 Ubuntu de AWS, usuario (habitualmente “ubuntu”) y el fichero PPK. Hacer clic en “Conectar”.



2. Protocolos de comunicaciones - SSH y SFTP

OpenSSH: configuración de acceso para EC2 Ubuntu de AWS

Aceptar la conexión y ya podemos acceder directamente al FTP de la máquina EC2.

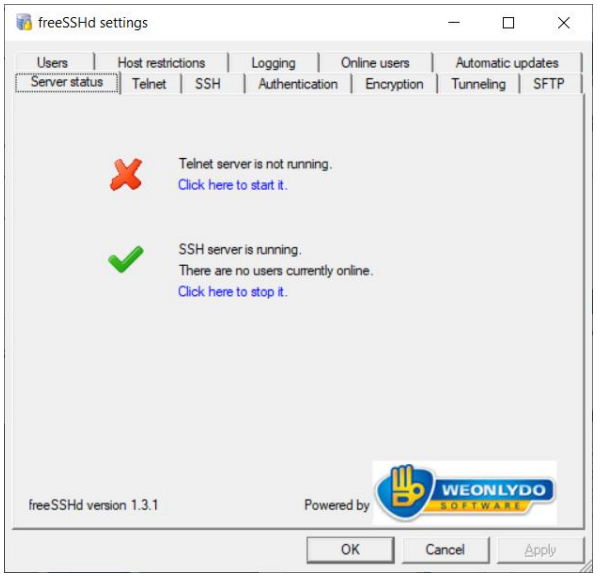


Y ya podemos transferir ficheros de local a remoto

Florida
Grup Educatiu

2. Protocolos de comunicaciones - SSH y SFTP

Como opción gratuita de servidor SSH/SFTP para Windows está la aplicación freeSSHd. Esta opción sería adecuada si nuestro servidor web funciona en Windows (aunque es más habitual que los servidores sean máquinas Linux).



https://freesshd.informer.com/

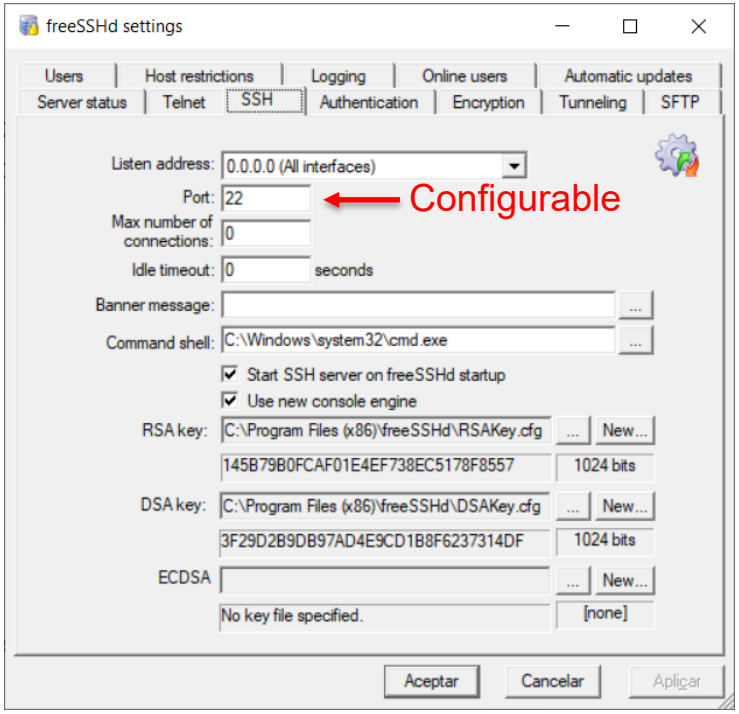
Para probar cómo funciona, hay que instalar el servidor en un equipo y conectarse desde otro utilizando PuTTy (SSH) o FileZilla (SFTP). El funcionamiento sería equivalente al ejemplo que hemos visto con Ubuntu accediendo con usuario y contraseña.

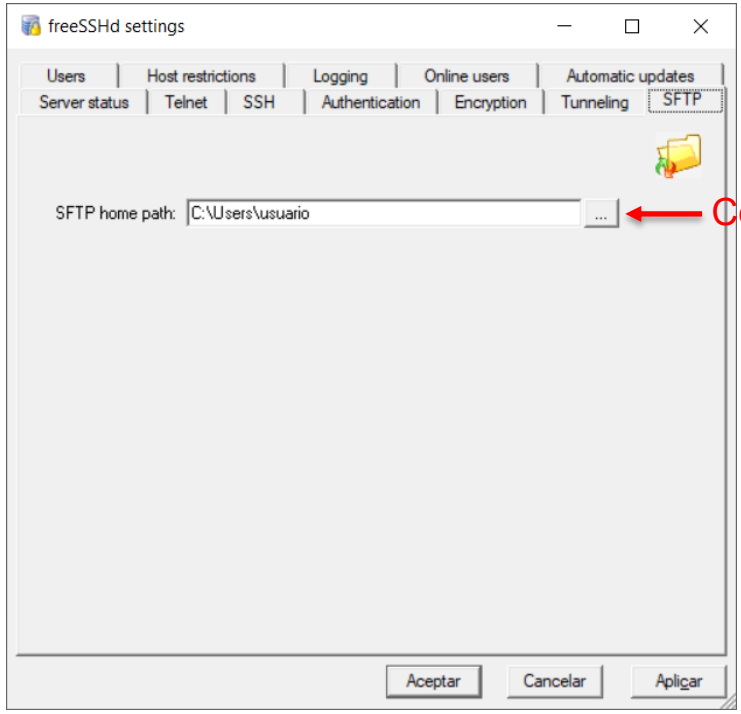
Recuerda ejecutar la aplicación como administrador (suele aparecer como minimizado en el área de notificaciones).

2. Protocolos de comunicaciones - SSH y SFTP

freeSSHd

Configurar servidor SSH y servidor SFTP

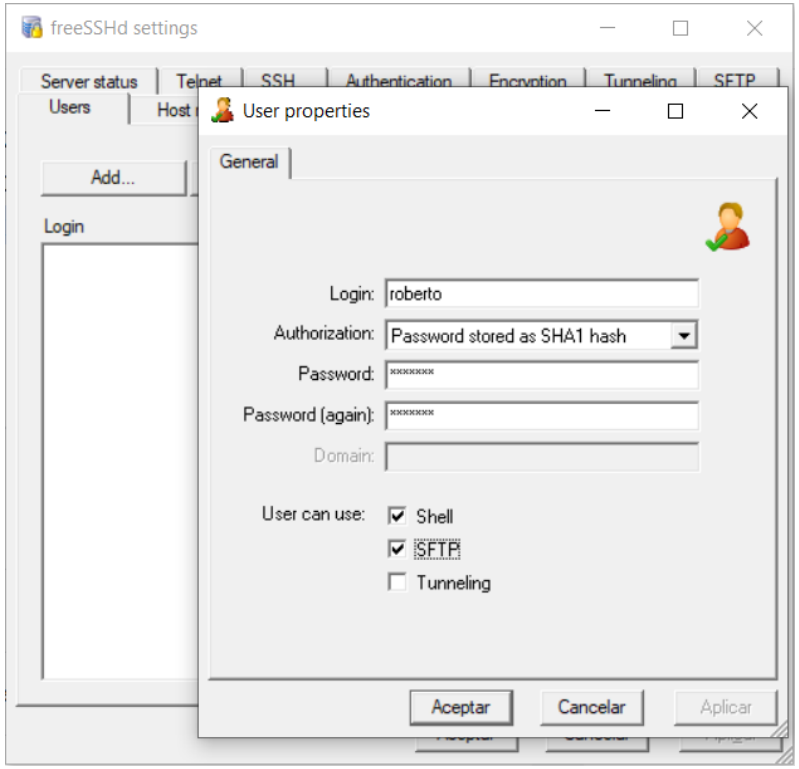


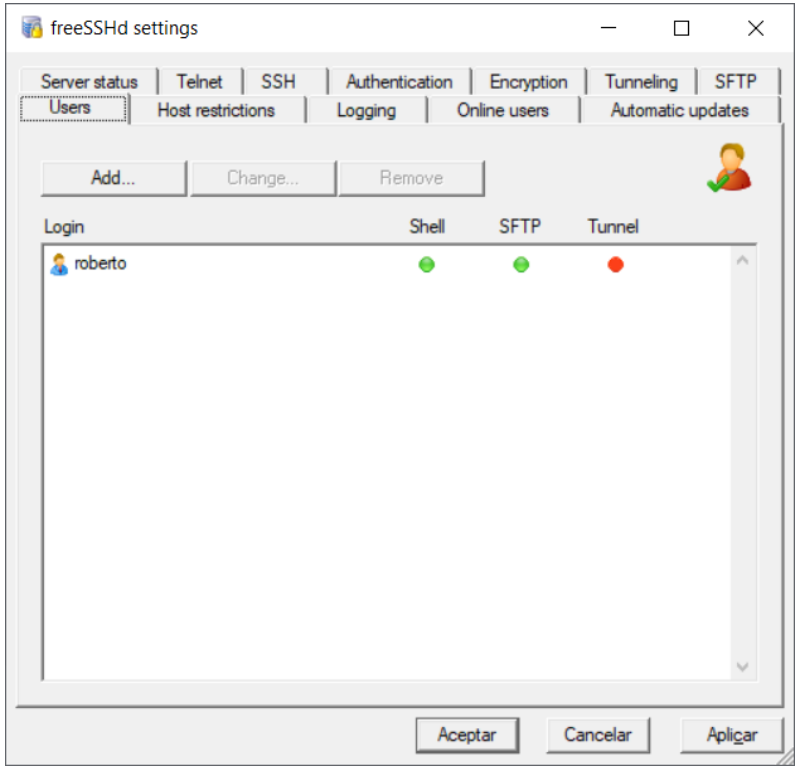


2. Protocolos de comunicaciones - SSH y SFTP

freeSSHd

Añadir nuevo usuario (cliente autorizado a conectarse al servidor SSH y FTP)

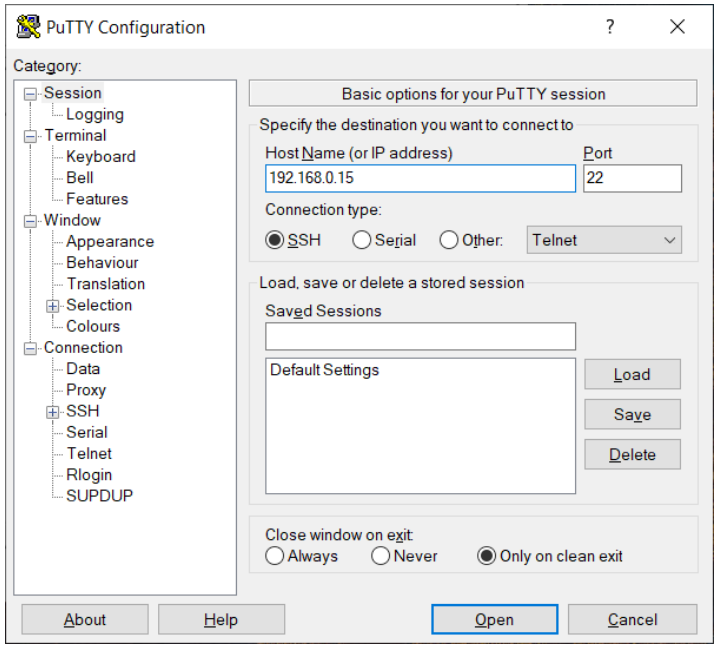


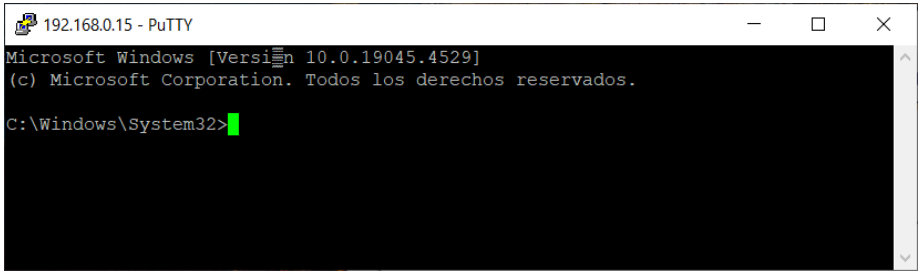


2. Protocolos de comunicaciones - SSH y SFTP

freeSSHd

Igual que antes, accedemos vía PuTTy para probar SSH:



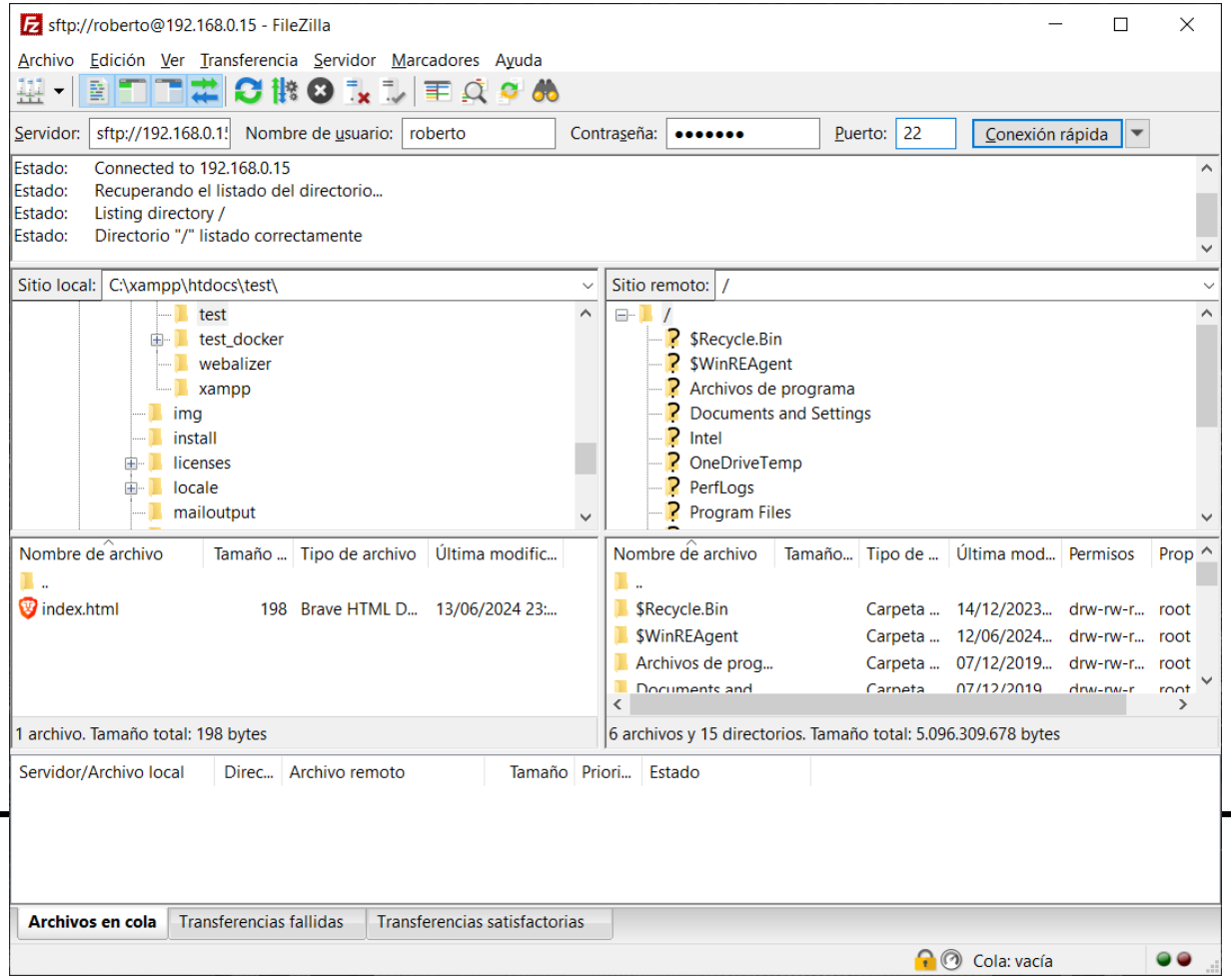


En este caso, tras introducir usuario y contraseña vemos que nos aparece la ventana de comandos de Windows (equivalente al Terminal de Ubuntu). Estamos “dentro” de la máquina Windows remota.

2. Protocolos de comunicaciones - SSH y SFTP

freeSSHd

Igual que antes, accedemos vía FileZilla para SFTP:



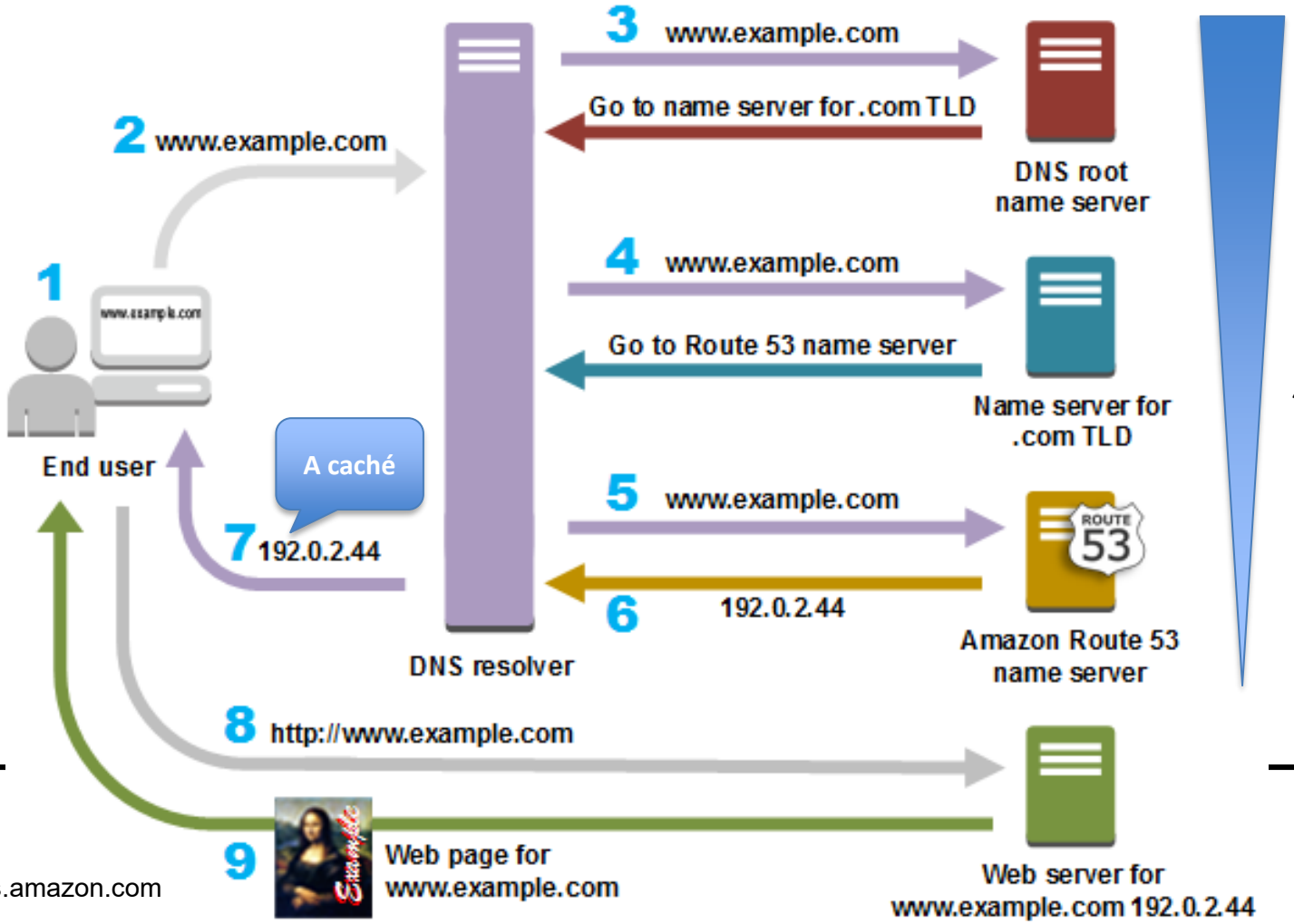
Y ya tenemos acceso al sistema de archivos del equipo remoto para poder transferir ficheros.

Igual que antes, podemos acceder al directorio donde esté desplegada la web y sobrescribir su contenido.

2. Protocolos de comunicaciones - DNS

DNS (*Domain Name System*): Sistema de Nombres de Dominio

Protocolo de Internet que traduce nombres “amigables” (p.ej. wikipedia.org) a direcciones IP (p.ej. 185.15.58.224) y viceversa.



2. Protocolos de comunicaciones - DNS

Principales registros DNS

Registro A (*Address Record*): registro básico de DNS que se utiliza para mapear un nombre de dominio a una dirección IPv4.

```
wikipedia.org IN A 185.15.58.224
```

Registro CNAME (*Canonical Name Record*): registro que se utiliza como alias y que redirige a otro nombre de dominio (que se denomina nombre canónico). Es útil para tener varios dominios que apuntan al mismo registro A, aunque requiere una consulta adicional para conocer la IP.

```
www.wikipedia.org IN CNAME wikipedia.org
```

En este caso, `wikipedia.org` sería el nombre canónico.

2. Protocolos de comunicaciones - DNS

Algunas herramientas útiles

nslookup: permite obtener la dirección IP asociada a un nombre DNS y viceversa. Hay disponibles versiones desde terminal/CMD para UNIX/Linux y Windows. También hay disponible una versión online (https://www.nslookup.io/).

dig (*domain information groper*): más moderna y potente que Nslookup. Disponible por defecto en UNIX/Linux y como ejecutable en Windows. También tiene su versión online (https://digwebinterface.com/).

ping: no es específica de DNS, pero se puede utilizar para resolver nombres de dominio.

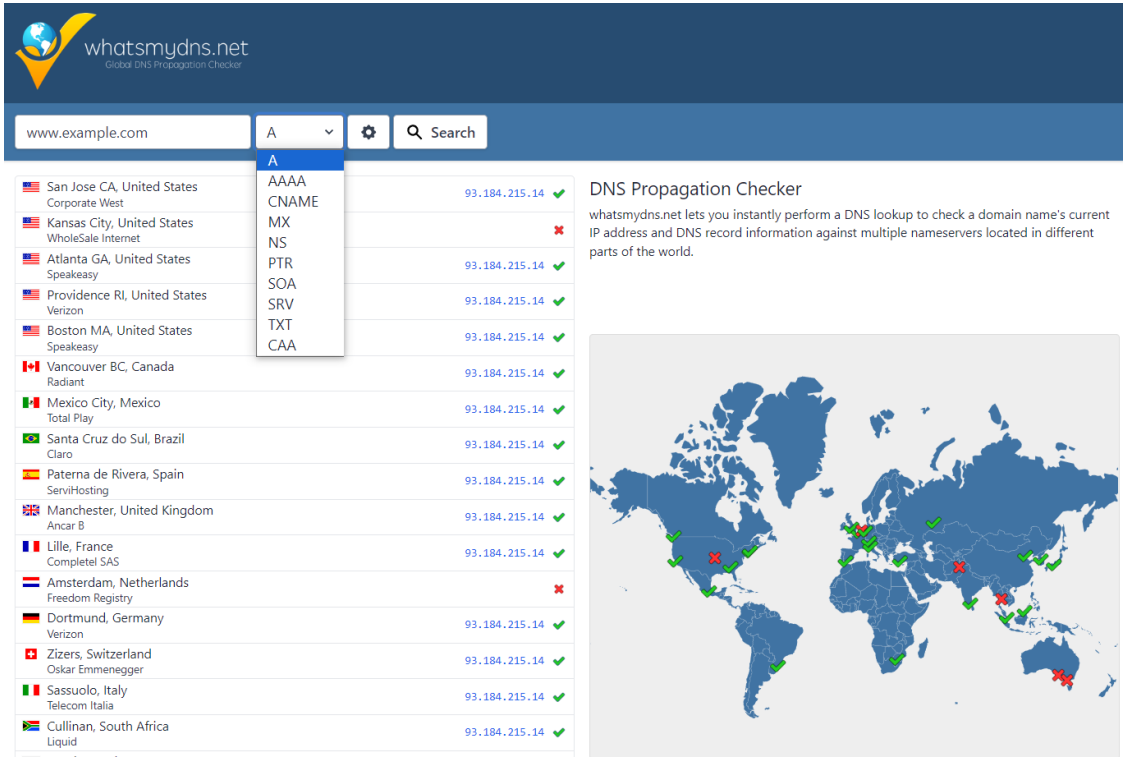
traceroute (UNIX/Linux) y **tracert** (Win): traza la ruta que siguen los paquetes desde el origen al destino, mostrando también la resolución DNS de cada salto.

whois: consulta la base de datos de registro de dominios para obtener información sobre la propiedad y detalles de registro de un dominio.

2. Protocolos de comunicaciones - DNS

Algunas herramientas útiles

DNS *Propagation Checker* (https://www.whatsmydns.net/)
Esta herramienta online permite verificar la propagación DNS en varios servidores alrededor del mundo, mostrando el estado de la resolución de nombres en diferentes ubicaciones geográficas. Es especialmente útil cuando se despliega una web y queremos ver cómo se va propagando el DNS globalmente.



3. Servidores web: Apache y Nginx

Apache y Nginx son los servidores web más populares en la actualidad.
Ambos son de código abierto, gratuitos y compatibles.

Apache:

- Primer servidor web consolidado (soporte)
- Arquitectura basada en procesos e hilos para cada conexión (⇑ recursos)
- Pérdida de eficiencia en cargas elevadas
- Flexible y personalizable con módulos
- Maneja páginas estáticas y dinámicas desde el propio servidor (PHP/Perl)

Nginx:

- Desarrollado con posterioridad a Apache
- Arquitectura basada en eventos y asincronía (más eficiente)
- Ideal para cargas altas y concurrencia (rendimiento y escalabilidad)
- También es modular, pero menos flexible y personalizable
- Optimizado para servir páginas estáticas muy rápidamente
- Para contenidos dinámicos utiliza otras aplicaciones (Apache, FastCGI,…)

NOTA: recuerda que para desarrollo también puedes utilizar el servidor incorporado (*built-in*) de **PHP**. No es un servidor para producción, pero es útil para entorno de desarrollo y pruebas.

```
cd /ruta/al/proyecto
php -S localhost:8000 -t <carpeta_src>
```

Acceso a la aplicación desde el navegador con http://localhost:8000

3. Servidores web: Apache y Nginx

Apache

Ya hemos visto cómo se instala y se ejecuta en el tema 1 (Linux y Windows).

Para Windows, si hemos realizado la instalación con XAMPP:

Por defecto utiliza el puerto 80 para conexiones HTTP, pero se puede modificar si lo necesitamos en el fichero `httpd.conf`, definiendo las líneas:

```
Listen 80
ServerName localhost:80
```

Hay otros parámetros que se pueden modificar en este fichero según nuestras necesidades (directorios, permisos, etc.).

Para conexiones HTTPS utiliza por defecto el puerto 443. Este puerto y el resto de la configuración relacionada con los certificados SSL/TLS se pueden modificar en el fichero `httpd-ssl.conf` (por ejemplo, si disponemos de un certificado emitido por una AC -autoridad certificadora-, indicaríamos en este fichero la ruta de disco donde estuviera).

Por último, también se puede configurar aspectos de PHP en el fichero `php.ini`.

3. Servidores web: Apache y Nginx

Apache

Ya hemos visto cómo se instala y se ejecuta en el tema 1 (Linux y Windows).

Para Ubuntu la nomenclatura y localización de los ficheros es algo diferente, pero se pueden configurar aspectos similares:

Fichero `/etc/apache2/apache2.conf`: aspectos generales
Fichero `/etc/apache2/ports.conf`: puertos HTTP y HTTPS

3. Servidores web: Apache y Nginx

Apache - Control de acceso

Vamos a configurar Apache para que exista una restricción de acceso a un determinado recurso o directorio. Por ejemplo, tenemos una carpeta llamada “confidencial” a la que solo pueden acceder usuarios autorizados mediante usuario/contraseña. En esta carpeta alojamos unos ficheros txt de acceso restringido.

La carpeta “confidencial” es un subdirectorio del directorio donde has alojado la web. Por ejemplo, siguiendo lo hecho en el tema anterior:

Windows
Ruta `C:\xampp\htdocs\test` ➔ Subdirectorio: `C:\xampp\htdocs\test\confidencial`

Linux
Ruta `/var/www/html/test` ➔ Subdirectorio: `/var/www/html/test/confidencial`

El control de acceso se realiza creando un par de ficheros en el directorio restringido: `.htaccess` (política de acceso) y `.htpasswd` (usuarios y contraseñas autorizados). Notar que el “.” forma parte del nombre del archivo y que no tienen extensión.

3. Servidores web: Apache y Nginx

Apache - Control de acceso (Windows)

Contenido del fichero .htaccess (ubicado en la carpeta “confidencial”):

```
AuthType Basic
AuthName “Directorio restringido”
AuthUserFile ../htdocs/test/confidencial/.htpasswd
Require valid-user
Options Indexes
```

- `AuthType Basic` — Tipo de autenticación
- `AuthName “Directorio restringido”` — Nombre de la política
- `AuthUserFile ../htdocs/test/confidencial/.htpasswd` — Ruta relativa fichero .htpasswd
- `Require valid-user` — Política de acceso
- `Options Indexes` — Permite listar archivos existentes

Para crear el fichero `.htpasswd` hay que utilizar la utilidad `htpasswd` desde CMD (utilizar la opción “-c” solo para el primer usuario, excluirla para añadir siguientes):

```
C:\xampp\apache\bin>htpasswd -c "C:\xampp\htdocs\test\confidencial\.htpasswd" usuario1
New password: ********
Re-type new password: ********
Adding password for user usuario1
```

Finalmente, reiniciar Apache para que los cambios surtan efecto.

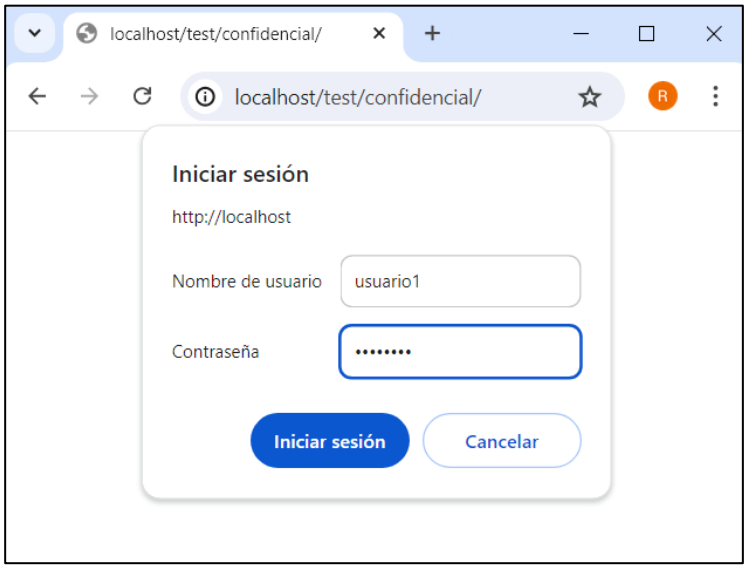
Versión web: https://www.web2generators.com/apache-tools/htpasswd-generator

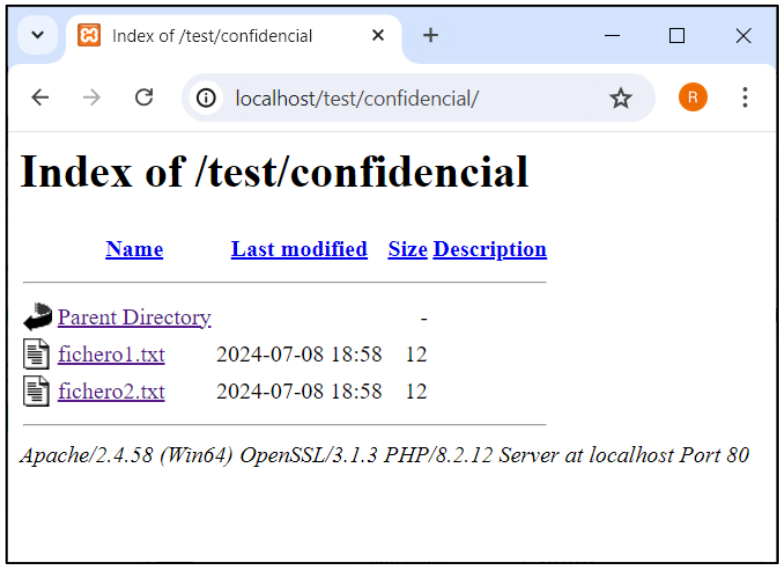
NOTA: para que se apliquen las políticas de control de acceso, el valor de `AllowOverride` de la configuración del directorio de trabajo debe ser “`All`” en el fichero de configuración de Apache (`httpd.conf`).

```
<Directory "C:/xampp/htdocs">
    ...
    AllowOverride All
    ...
</Directory>
```

3. Servidores web: Apache y Nginx

Apache - Control de acceso (Windows)





En este caso se muestra un listado de ficheros, pero también podría ser una página web de nuestro sitio que estuviera restringida.
Podríamos hacer exactamente lo mismo controlando los permisos de acceso desde nuestra aplicación web, en vez de hacerlo directamente desde el servidor.

3. Servidores web: Apache y Nginx

Apache - Control de acceso (Linux, utilizando la MV Ubuntu)

Configurar Apache para admitir `.htaccess`: `sudo nano /etc/apache2/apache2.conf`
Asegurarse que la línea `AccessFileName .htaccess` no está comentada.
Cambiar "None" por "All" en la parte del fichero donde aparece:

```
<Directory /var/www>
    Options Indexes FollowSymLinks
    AllowOverride None (➔ All)
    Require all granted
</Directory>
```

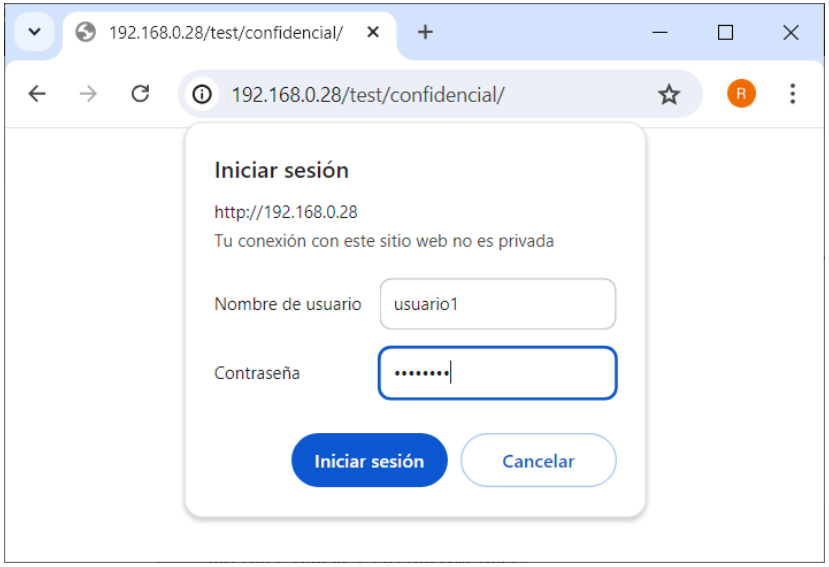
Contenido del fichero `.htaccess` (ubicado en la carpeta "confidencial"):

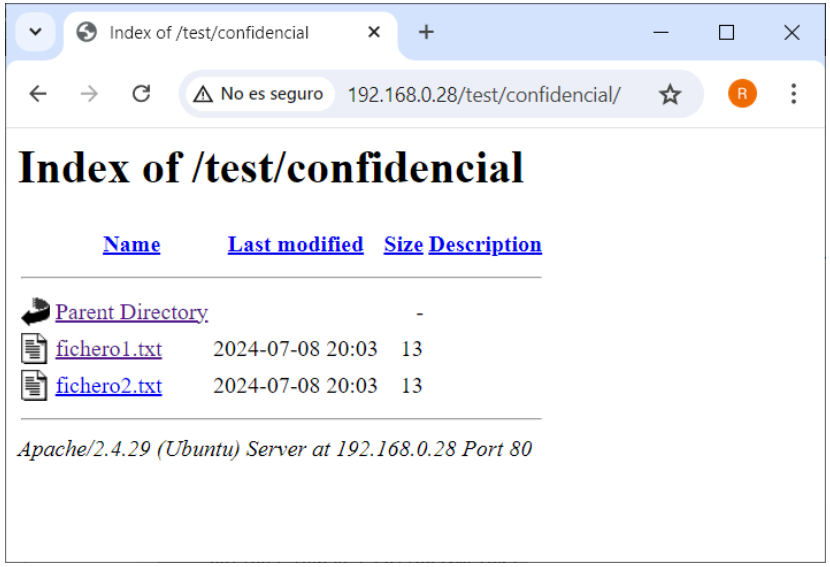
```
AuthType Basic
AuthName "Directorio restringido"
AuthUserFile /var/www/html/test/confidencial/.htpasswd
Require valid-user
Options Indexes
```

Para crear el fichero `.htpasswd` hay que utilizar la utilidad `htpasswd` desde el terminal (igual que en CMD). Finalmente, reiniciar Apache para que los cambios surtan efecto.

3. Servidores web: Apache y Nginx

Apache - Control de acceso (Linux, utilizando la MV Ubuntu)

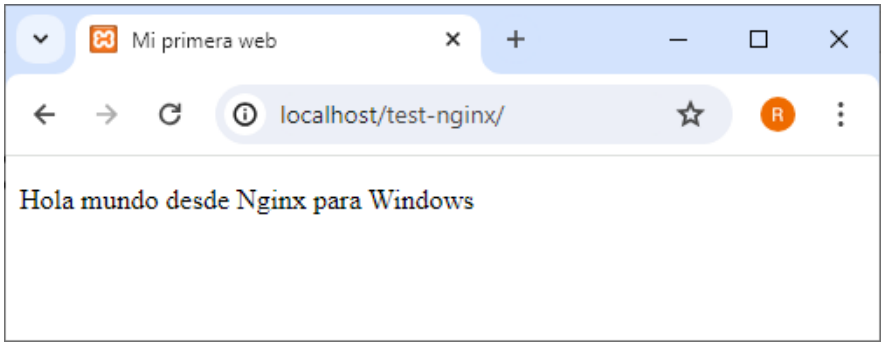




3. Servidores web: Apache y Nginx

Nginx para Windows

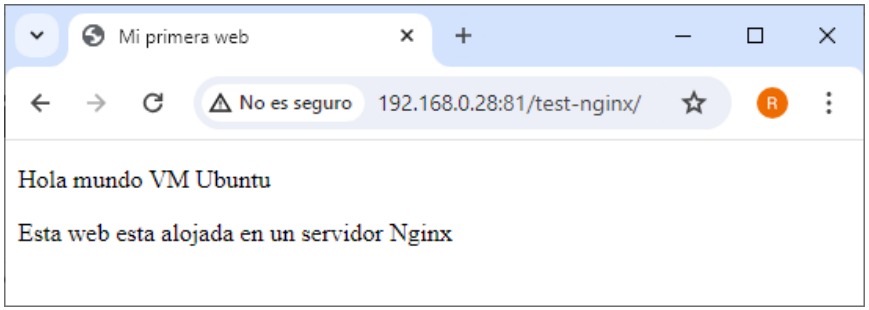
- Descargar desde https://nginx.org/ el fichero ZIP con la última versión estable.
- Descomprimir el fichero y ejecutar el programa `nging.exe` (no tiene interfaz gráfica).
- Las opciones de configuración/administración se gestionan en los ficheros de la carpeta `conf` (p.ej. `nginx.conf` para puertos, directorios, etc.).
- Por defecto, la carpeta de trabajo (equivalente a `htdocs` de Apache) es `html`, dentro del directorio `nginx-X.X.X` (donde X.X.X es la versión de Nginx).
- Podemos crear una subcarpeta `test-nginx` y alojar ahí un `index.html` “Hola mundo”.



3. Servidores web: Apache y Nginx

Nginx para Linux (Ubuntu)

- Basta introducir en el Terminal la instrucción: `sudo apt install nginx`
- En este caso la carpeta de donde se alojan por defecto las webs es `/var/www/html`
- NOTA: es posible que el puerto por defecto de Nginx (80) esté ocupado por el servidor Apache, que hemos instalado con anterioridad. Para evitar conflictos, puedes editar el fichero de configuración de Nginx (`sudo nano /etc/nginx/sites-enabled/default`) y poner el puerto 81 (u otro puerto libre)
- Si no habías iniciado el servidor lo puedes iniciar ahora: `sudo systemctl start nginx`
- O reiniciar para activar los cambios: `sudo systemctl restart nginx`



En este ejemplo hemos creado una subcarpeta `test-nginx` en el directorio `/var/www/html` con un `index.html` como el que se muestra aquí.
NOTA: la IP corresponde a la máquina virtual Ubuntu que se está ejecutando.

3. Servidores web: Apache y Nginx

Despliegue en servidores Apache y Nginx

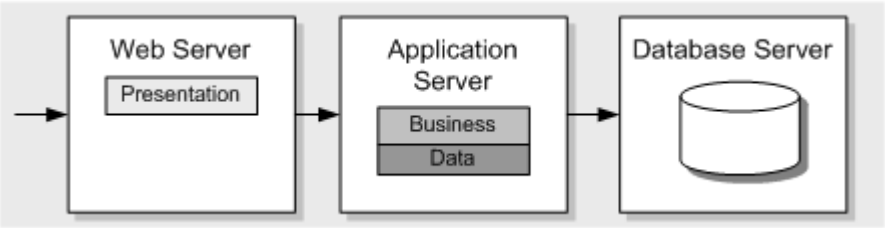
- Anteriormente hemos visto cómo conectarnos vía SFTP al servidor donde queremos desplegar nuestra web:
  - Servidor SSH: OpenSSH en Linux, freeSSHd en Windows,…
  - Cliente FTP: FileZilla

- Procederemos igual, tanto si el servidor web es Apache como Nginx, ya que simplemente tenemos que crear o actualizar los ficheros que necesite nuestra aplicación web (HTML, CSS, JS, PHP, imágenes, etc.), que están alojados en los directorios correspondientes.

4. Servidores web vs Servidores de aplicaciones

Servidor web: sirve contenido HTTP (HTML con texto, imágenes, videos, etc.) generado de manera estática (sin interacción del cliente) o dinámica (interacción y ejecución de scripts como PHP, Perl, Python, etc.)
➔ Apache, Nginx,…

Servidor de aplicaciones: sirve contenido HTTP dinámico (y también de otros protocolos), que depende de interacciones del usuario y que implica aplicar la lógica de negocio avanzada de la aplicación (ejecución de código, acceso a bases de datos, mensajería, seguridad, etc.). El resultado de esta ejecución de código puede ser o no una página HTML que el servidor web envía al cliente.
➔ Tomcat (Java), Gunicorn (Python), Node.js (Javascript), Apache (PHP), Nginx (PHP), Microsoft IIS (.NET),…



stackoverflow.com

4. Servidores web vs Servidores de aplicaciones

Frameworks

Conjunto de herramientas, bibliotecas (librerías) y estructuras que proporcionan la base para el desarrollo de aplicaciones.
Los *frameworks* relacionados con desarrollo web proporcionan normalmente un servidor de aplicaciones embebido que facilita el desarrollo y testeo de la aplicación.

BACKEND

- Express (Node.js / Javascript)
- Spring (Java)
- Laravel (PHP)
- Symfony (PHP)
- Django (Python)
- Flask (Python)
- Ruby on Rails (Ruby)
- ASP.NET Core (.NET)

FRONTEND

- React (Javascript)
- Angular (Typescript)
- Vue.js (Javascript)

5. Conclusiones

1. En aplicaciones web, la arquitectura más común es la denominada **arquitectura cliente-servidor**.
2. Una aplicación web se puede dividir en dos partes: ***frontend*** y ***backend***.
3. El **protocolo HTTP** se utiliza para enviar y recibir información entre cliente-servidor.
4. Las peticiones y respuestas HTTP incluyen cabecera (***header***) y cuerpo (***body***).
5. HTTP es un protocolo no seguro: **HTTPS** incorpora cifrado (TLS/SSL) por pares de claves.
6. El protocolo **SSH** permite administrar equipos (servidores) remotos de forma segura. Permite el uso del terminal y la transferencia segura de ficheros (protocolo **SFTP**). SSH y SFTP también tienen una arquitectura cliente-servidor y facilitan el despliegue de aplicaciones y la administración de servidores.
7. El protocolo **DNS** permite traducir direcciones web (amigables) por direcciones IP (entendibles por Internet).

Sugerencia para practicar

1. Crea una aplicación web de temática libre (o toma alguna de otro módulo) que contenga al menos dos páginas (`index.html` y `areaprivada.html`).
2. La *landing page* debe contener algunas imágenes y también estilos (en un fichero .css aparte), así como un botón para navegar a la página `areaprivada.html`.
3. El acceso a `areaprivada.html` debe estar protegido con contraseña (el servidor web debe controlar este acceso).
4. El contenido de `areaprivada.html` debe ser una lista (aplica también estilos para que quede "profesional") con algunos documentos de tipo PDF, Word, etc. Añade también un botón para regresar a `index.html`.
5. Prueba que la aplicación funciona correctamente en local.
6. Despliega y prueba la aplicación en una máquina virtual Ubuntu (ubicada en Virtual Box).