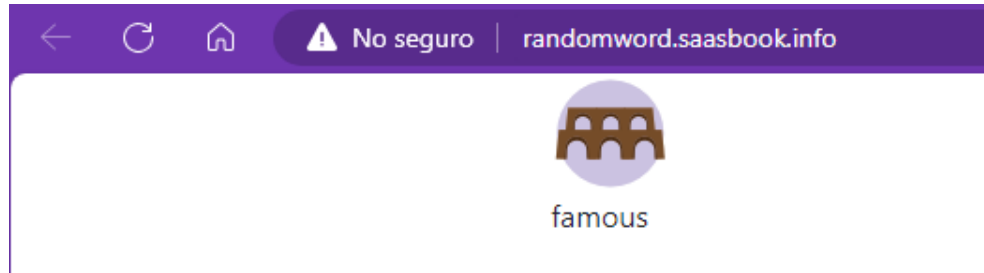


# Introducción a HTTP y URI

Comenzamos visitando el generador de palabras aleatorias en tu navegador favorito para obtener una vista de usuario de lo que hay en la página principal.



## Comprendiendo Request y Response

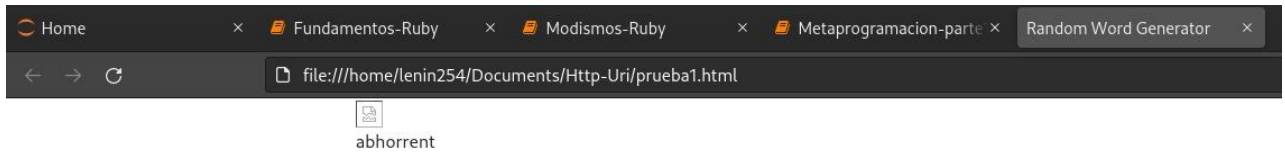
### 1. Realizamos un HTTP GET a un sitio, usando CURL

Ingresamos el comando `curl 'http://randomword.sasbook.info'`

```
lenin254@debian ~$ curl 'http://randomword.sasbook.info'
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta
anonymous">
    <title>Random Word Generator</title>
    <body class="container">
      <div id="image">
        
      </div>
      <div id="word">
        lunchroom
      </div>
    </body>
  </html>
lenin254@debian ~$
```

### 2. Guardamos la salida de curl en un archivo y lo visualizamos

```
lenin254@debian ~$ mkdir Http-Uri
mkdir: created directory 'Http-Uri'
lenin254@debian ~$ cd Http-Uri/
/home/lenin254/Documents/Http-Uri
lenin254@debian ~$ ls
lenin254@debian ~$ curl 'http://randomword.sasbook.info' > prueba1.html
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left  Speed
100  483  100  483    0     0  1359    0 --:--:-- --:--:-- --:--:-- 1360
lenin254@debian ~$ ls
prueba1.html
lenin254@debian ~$
```



**Pregunta: ¿Cuáles son las dos diferencias principales que has visto anteriormente y lo que ves en un navegador web 'normal'? ¿Qué explica estas diferencias?**

Las dos diferencias principales que se observan son: la dirección de la página ahora muestra una dirección local de archivo y la imagen no se carga correctamente.

La dirección de la página se debe al contexto en el que se interpreta la URL, ya sea como una dirección web en línea o como una dirección local de archivo. La imagen no cargada correctamente se debe a que está vinculada a una dirección web externa que no está accesible desde tu sistema de archivos local.

### 3. Simulamos un servidor Web usando Netcat

```
server@server:~$ nc -l 8081
GET / HTTP/1.1
Host: 192.168.18.18:8081
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.0.0 Safari/537.36 Edg/117.0.2045.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en,es-419;q=0.9,es;q=0.8,es-ES;q=0.7,en-GB;q=0.6,en-US;q=0.5,fr;q=0.4
Cookie: wz-api=default
server@server:~$
```

**Pregunta: Suponiendo que estás ejecutando curl desde otro shell ¿qué URL tendrás que pasarle a curl para intentar acceder a tu servidor falso y por qué?**

La URL que tendrías que pasarle a curl para intentar acceder al servidor falso es `http://192.168.18.18:8081`. Esto se debe a que en la solicitud GET enviada al servidor falso, la línea `Host: 192.168.18.18:8081` indica la dirección IP (192.168.18.18) y el puerto (8081) al cual debes apuntar con curl.

**Pregunta: La primera línea de la solicitud identifica qué URL desea recuperar el cliente. ¿Por qué no ves `http://localhost:8081` en ninguna parte de esa línea?**

La primera línea de la solicitud no contiene "`http://localhost:8081`" porque el cliente está especificando la ruta relativa ("/") en lugar de una URL completa. El cliente y el servidor ya establecieron la conexión a través de la dirección IP y puerto.

#### 4. Vemos los encabezados de respuesta del servidor

```
server@server:~$ curl -i 'http://randomword.saasbook.info'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/html; charset=utf-8
Content-Length: 480
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Server: WEBrick/1.4.2 (Ruby/2.6.6/2020-03-31)
Date: Mon, 25 Sep 2023 01:06:35 GMT
Via: 1.1 vegur

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1
anonymous">
    <title>Random Word Generator</title>
  <body class="container">
    <div id="image">
      
    </div>
    <div id="word">
      settle
    </div>
  </body>
</html>
server@server:~$ |
```

**Pregunta:** Según los encabezados del servidor, ¿cuál es el código de respuesta HTTP del servidor que indica el estado de la solicitud del cliente y qué versión del protocolo HTTP utilizó el servidor para responder al cliente?

Según los encabezados del servidor:

- El código de respuesta HTTP del servidor es 200 OK, indicando que la solicitud del cliente fue exitosa.
- El servidor utilizó la versión HTTP/1.1 del protocolo HTTP para responder al cliente.

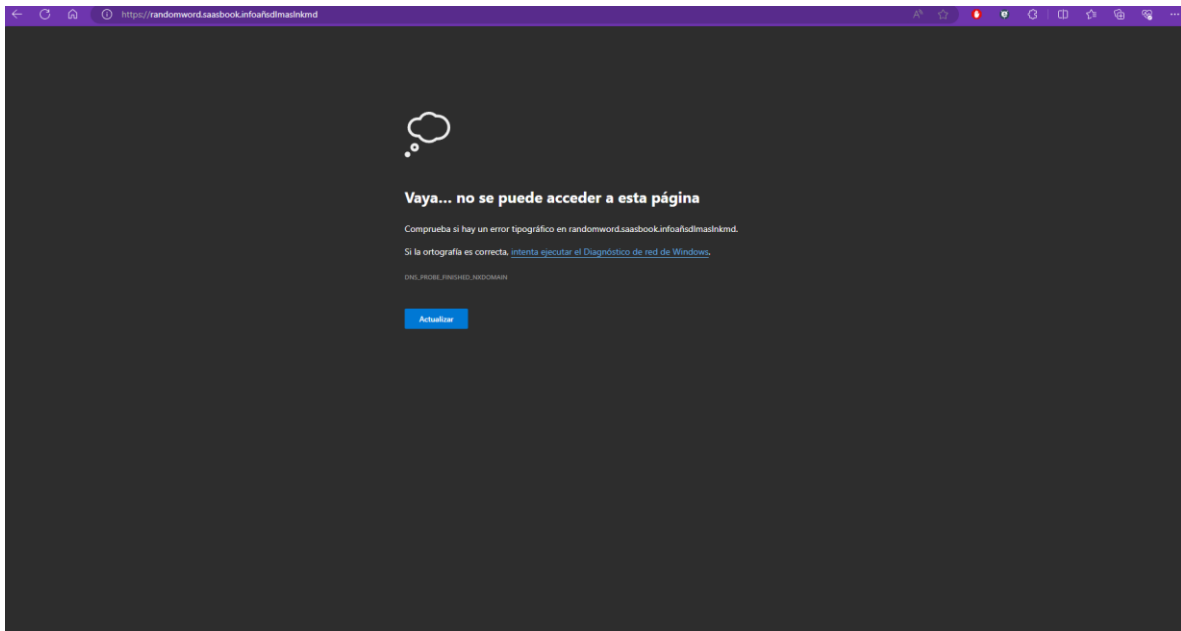
**Pregunta:** Cualquier solicitud web determinada puede devolver una página HTML, una imagen u otros tipos de entidades. ¿Hay algo en los encabezados que crea que le dice al cliente cómo interpretar el resultado?

Sí, el encabezado "Content-Type" indica al cliente cómo interpretar el tipo de contenido que está recibiendo. En este caso, el encabezado "Content-Type" es "text/html; charset=utf-8", indicando que el contenido es HTML y debe ser interpretado como tal por el cliente.

## ¿Qué sucede cuando falla un HTTP request?

**Pregunta:** ¿Cuál sería el código de respuesta del servidor si intentaras buscar una URL inexistente en el sitio generador de palabras aleatorias? Prueba esto utilizando el procedimiento anterior.

probablemente recibiríamos un código de respuesta HTTP 404 Not Found, que indica que la página no ha sido encontrada en el servidor.



## **¿Qué otros códigos de error HTTP existen?**

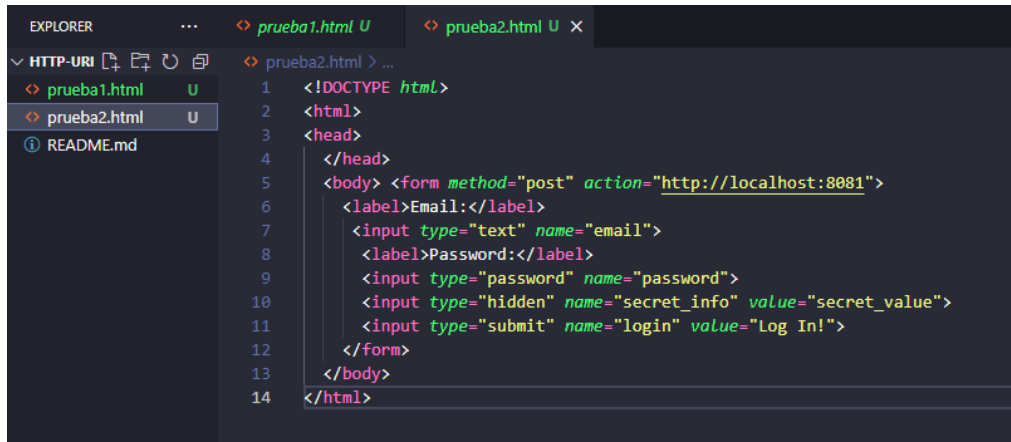
Códigos de Respuesta HTTP:

- 2xx (Éxito)
  - 200 OK: La solicitud fue exitosa.
  - 201 Created: Se ha creado un nuevo recurso como resultado de la solicitud.
- 3xx (Redirecciones)
  - 301 Moved Permanently: La página solicitada se ha movido permanentemente a una nueva URL.
  - 302 Found (o 307 Temporary Redirect): La página solicitada se ha movido temporalmente a una nueva URL.
- 4xx (Errores del Cliente)
  - 400 Bad Request: La solicitud realizada al servidor es incorrecta o no se puede entender.
  - 404 Not Found: El servidor no puede encontrar la página solicitada.
- 5xx (Errores del Servidor)
  - 500 Internal Server Error: El servidor ha encontrado una situación no prevista que impide cumplir la solicitud.

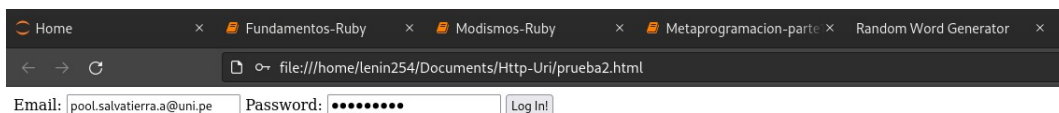
## ¿Cuál es la principal diferencia entre 4xx y 5xx?

La principal diferencia es que los códigos de respuesta que comienzan con "4xx" indican errores del cliente, mientras que los códigos que comienzan con "5xx" indican errores del servidor.

## ¿Qué es un cuerpo de Request?



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 </head>
5 <body> <form method="post" action="http://localhost:8081">
6   <label>Email:</label>
7   <input type="text" name="email">
8   <label>Password:</label>
9   <input type="password" name="password">
10  <input type="hidden" name="secret_info" value="secret_value">
11  <input type="submit" name="login" value="Log In!">
12 </form>
13 </body>
14 </html>
```



Home x Fundamentos-Ruby x Modismos-Ruby x Metaprogramacion-parte x Random Word Generator x

file:///home/lenin254/Documents/Http-Uri/prueba2.html

Email: pool.salvatierra.a@uni.pe Password: [masked] [Log In!]



```
lenin254@debian ~$ nc -l -p 8081
POST / HTTP/1.1
Host: localhost:8081
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 93
Connection: keep-alive
Cookie: username=localhost-8888="2|1:0|10:1694009715|23:username=localhost-8888|192:eyJ1c2VybmFtZSI6ICJvbnltb3VzIEFvZWRLIiwiaWwImluaXRpYWxzIjogIkFBIiwgImNvbG9yIjogbnVsbH0=|e32e4726775d7377a62fb25deee92e682ec7a
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Sec-Fetch-User: ?1

email=pool.salvatierra.a%40uni.pe&password=123456798&secret_info=secret_value&login=Log+In%21
```

**Pregunta:** Cuando se envía un formulario HTML, se genera una solicitud HTTP POST desde el navegador. Para llegar a tu servidor falso, ¿con qué URL deberías reemplazar Url-servidor-falso en el archivo anterior?

`<form method="post" action="http://localhost:8081">`

**Pregunta: ¿Cómo se presenta al servidor la información que ingresó en el formulario? ¿Qué tareas necesitaría realizar un framework SaaS como Sinatra o Rails para presentar esta información en un formato conveniente a una aplicación SaaS escrita, por ejemplo, en Ruby?**

Se representa:

```
email=pool.salvatierra.a%40uni.pe&password=123456798&secret_info=secret_value&login=Log+In%21
```

@ -> %40

el framework se encargaría de manejar la solicitud, extraer los datos, decodificarlos, procesarlos y luego pasarlos a la aplicación para que realice las acciones específicas de la aplicación basándose en esos datos.

**¿Cuál es el efecto de agregar parámetros URI adicionales como parte de la ruta POST?**

Agregar parámetros en la dirección web (URI) cuando haces un POST no afecta en los datos.

**¿Cuál es el efecto de cambiar las propiedades de nombre de los campos del formulario?**

Cambiar los nombres de los campos en un formulario es como cambiar las etiquetas que identifican qué información va en cada espacio del formulario.

**¿Puedes tener más de un botón Submit? Si es así, ¿cómo sabe el servidor en cuál se hizo clic? (Sugerencia: experimenta con los atributos de la etiqueta <submit>).**

Sí, es posible tener más de un botón "Submit" en un formulario HTML. Para lograrlo, puedes usar el atributo name en cada botón "Submit". De esta forma, el servidor puede identificar cuál de los botones fue presionado y actuar en consecuencia.

```
<form method="post" action="http://localhost:8081">
  <button type="submit" name="submit_action"
value="guardar">Guardar</button>
  <button type="submit" name="submit_action"
value="eliminar">Eliminar</button>
</form>
```

**¿Se puede enviar el formulario mediante GET en lugar de POST? En caso afirmativo, ¿cuál es la diferencia en cómo el servidor ve esas solicitudes?**

Sí, se puede enviar un formulario tanto mediante GET como mediante POST. La diferencia principal radica en cómo se envían los datos:

-GET: Los datos del formulario se envían en la URL.

-POST: Los datos del formulario se envían en el cuerpo de la solicitud

¿Qué otros verbos HTTP son posibles en la ruta de envío del formulario? ¿Puedes hacer que el navegador web genere una ruta que utilice PUT, PATCH o DELETE?

- GET: Para obtener información del servidor.
- POST: Para enviar información al servidor (el más comúnmente utilizado en formularios HTML).
- PUT: Para actualizar una entidad en el servidor.
- PATCH: Para modificar parcialmente una entidad en el servidor.
- DELETE: Para eliminar una entidad en el servidor.

```
GNU nano 7.2 prueba2.html
<!DOCTYPE html>
<html>
<head>
</head>
<body> <form method="post" action="http://localhost:8081">
  <label>Email:</label>
  <input type="text" name="email">
  <label>Password:</label>
  <input type="password" name="password">
  <input type="hidden" name="secret_info" value="secret_value">
  <input type="submit" name="login" value="Log In!">
</form>
<form method="put" action="http://localhost:8081">
  <button type="submit">Enviar con PUT</button>
</form>

<!-- Formulario para DELETE -->
<form method="delete" action="http://localhost:8081">
  <button type="submit">Enviar con DELETE</button>
</form>
</body>
</html>
```

## HTTP sin estados y cookies

Pregunta: Prueba las dos primeras operaciones GET anteriores. El cuerpo de la respuesta para la primera debe ser "Logged in: false" y para la segunda "Login cookie set". ¿Cuáles son las diferencias en los encabezados de respuesta que indican que la segunda operación está configurando una cookie? (Sugerencia: usa curl -v, que mostrará tanto los encabezados de solicitud como los encabezados y el cuerpo de la respuesta, junto con otra información de depuración. curl --help imprimirá una ayuda voluminosa para usar cURL y man curl mostrará la página del manual de Unix para cURL en la mayoría de los sistemas.)

```
lenin254@debian ~$ curl -v http://esaas-cookie-demo.herokuapp.com/
* Trying 3.210.192.5:80...
* Connected to esaas-cookie-demo.herokuapp.com (3.210.192.5) port 80 (#0)
> GET / HTTP/1.1
> Host: esaas-cookie-demo.herokuapp.com
> User-Agent: curl/7.88.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Connection: keep-alive
< Content-Type: text/plain; charset=utf-8
< Content-Length: 16
< X-Content-Type-Options: nosniff
< Server: WEBrick/1.6.1 (Ruby/2.7.8/2023-03-30)
< Date: Mon, 25 Sep 2023 05:18:18 GMT
< Via: 1.1 vegur
<
* Connection #0 to host esaas-cookie-demo.herokuapp.com left intact
Logged in: false lenin254@debian ~$
```

```
lenin254@debian ~$ curl -v http://esaas-cookie-demo.herokuapp.com/login
* Trying 54.83.6.65:80...
* Connected to esaas-cookie-demo.herokuapp.com (54.83.6.65) port 80 (#0)
> GET /login HTTP/1.1
> Host: esaas-cookie-demo.herokuapp.com
> User-Agent: curl/7.88.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Connection: keep-alive
< Content-Type: text/plain; charset=utf-8
< Content-Length: 16
< X-Content-Type-Options: nosniff
< Server: WEBrick/1.6.1 (Ruby/2.7.8/2023-03-30)
< Date: Mon, 25 Sep 2023 05:21:11 GMT
< Set-Cookie: logged_in=true; domain=esaas-cookie-demo.herokuapp.com; path=/; HttpOnly
< Via: 1.1 vegur
<
* Connection #0 to host esaas-cookie-demo.herokuapp.com left intact
Login cookie set lenin254@debian ~$
```

La diferencia clave es el encabezado Set-Cookie en la respuesta de la segunda operación (GET /login). Este encabezado indica la configuración de una cookie, proporcionando detalles como nombre, valor y configuraciones adicionales.

**Pregunta:** Bien, ahora supuestamente "logged in" porque el servidor configuró una cookie que indica esto. Sin embargo, si intentas GET / nuevamente, seguirá diciendo "Logged: false". ¿Qué está sucediendo? (Sugerencia: usa curl -v y observa los encabezados de solicitud del cliente).

El cliente no está enviando la cookie al servidor en la solicitud GET a '/', por lo tanto, el servidor no reconoce la sesión iniciada y muestra "Logged in: false".



**Pregunta:** Al observar el encabezado Set-Cookie o el contenido del archivo cookies.txt, parece que podría haber creado fácilmente esta cookie y simplemente obligar al servidor a creer que ha iniciado sesión. En la práctica, ¿cómo evitan los servidores esta inseguridad?

Los servidores utilizan técnicas de seguridad como cifrado y verificación para evitar manipulaciones no autorizadas de cookies y asegurar la autenticidad de la sesión del usuario.

```
lenin254@debian ~$ curl -i --cookie-jar cookies.txt http://esaas-cookie-demo.herokuapp.com/login
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/plain; charset=utf-8
Content-Length: 16
X-Content-Type-Options: nosniff
Server: WEBrick/1.6.1 (Ruby/2.7.8/2023-03-30)
Date: Mon, 25 Sep 2023 05:30:53 GMT
Set-Cookie: logged_in=true; domain=esaas-cookie-demo.herokuapp.com; path=/; HttpOnly
Via: 1.1 vegur

Login cookie set lenin254@debian ~$ ls
cookies.txt Desktop Downloads kitty Pictures RubyMine-2023.2.1 RubymineProjects Videos
cv_debug.log Documents fonts Music Public RubyMine-2023.2.1.tar.gz Templates
lenin254@debian ~$ cat cookies.txt
# Netscape HTTP Cookie File
# https://curl.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.

#HttpOnly .esaas-cookie-demo.herokuapp.com TRUE / FALSE 0 logged_in true
```

## Conclusiones:

- Las cookies son cruciales para que los servidores realicen un seguimiento del estado de sesión y la interacción de un cliente.
- HTTP es un protocolo sin estado, pero las cookies permiten crear una noción de "estado de sesión".
- Los servidores confían en que el cliente incluya y actualice correctamente las cookies para mantener la sesión.
- Deshabilitar cookies puede impedir el funcionamiento adecuado de sitios que requieren inicio de sesión o interacciones secuenciales.