

HTTP-URI - TAREA

Comprendiendo Request y Respond

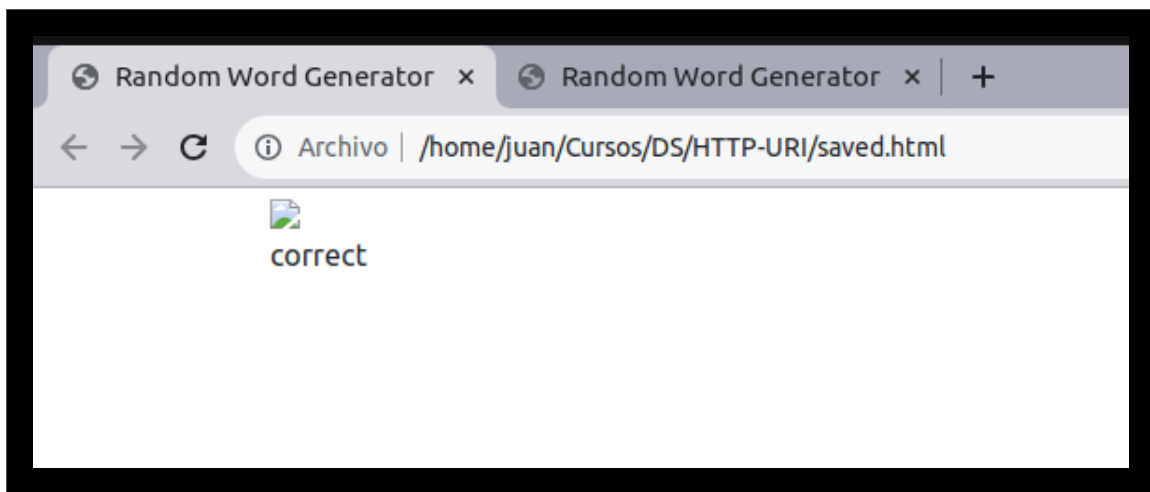
Cuando escribimos el comando: `curl 'http://randomword.saasbook.info'`, nos aparece algo como esto:

```
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ curl 'http://randomword.saasbook.info'
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-giJF6kkoqNQ00vy+HMDP7az0uL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmp1" crossorigin="anonymous">
    <title>Random Word Generator</title>
  <body class="container">
    <div id="image">
      
    </div>
    <div id="word">
      literate
    </div>
  </body>
</html>
```

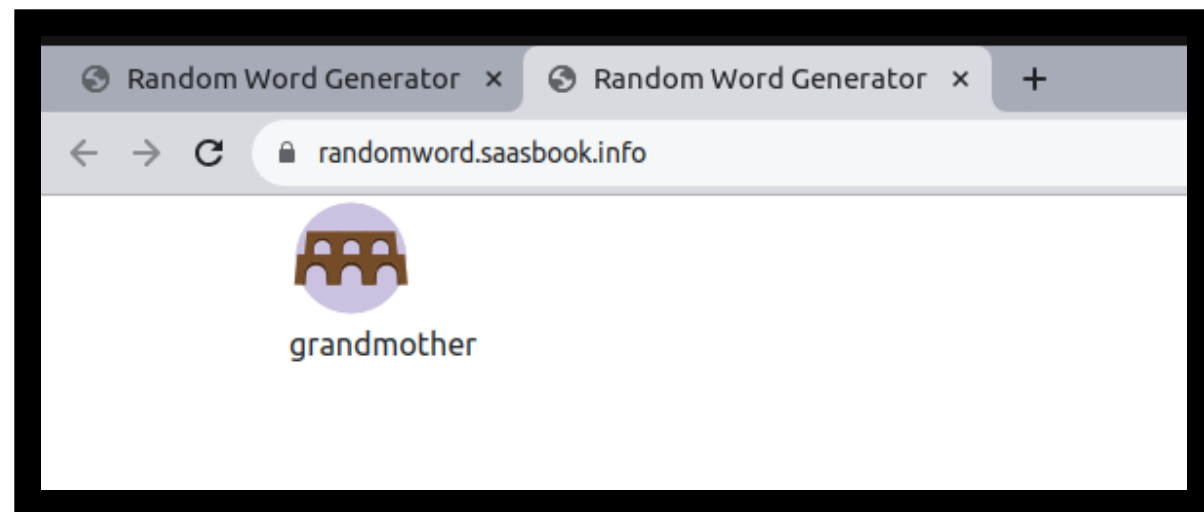
Luego cuando guardamos la respuesta de la solicitud en un archivo (saved.html):

```
</html>
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ curl 'http://randomword.saasbook.info' > saved.html
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100  481    100  481    0     0   918      0  --:--:-- --:--:-- --:--:--   917
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$
```

El resultado al abrir el archivo guardado es el siguiente:



El resultado al abrir la url desde el navegador:



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

La primera diferencia es que no se puede ver la imagen en el archivo guardado, esto es porque la solicitud curl solo devuelve el contenido html como respuesta mas no devuelve otros elementos como los css o imagenes, por eos no se reconce la imagen

La segunda diferencia es la palabra generada debajo de la imagen, idenpentiendmente que se haga la solicitud desde el navegador o desde la terminal, ambas son solicitudes indepentiendes y por lo tanto la respuesta será diferente (en la generacion del texto aleatorio)

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?

Rpta: curl <http://localhost:8081>

Porque dado que queremos hacer una solicitud http debemos poner http, ponemos localhost pues el servidor esta en nuestra computadora local, y 8081 hace referencia al puerto en donde se encuentra el servidor falso

Visita tu servidor 'falso' con curl y la URL correcta. Tu servidor 'falso' recibirá la solicitud del cliente HTTP.

RESULTADO: visita al servidor falso

```
100  481  100  481    0    0  918    0 --:--:-- --:--:-- --:--:--  
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ curl http://localhost:8081
```

RESULTADO: Solicitud recibida desde el servidor falso

```
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ nc -l 8081  
GET / HTTP/1.1  
Host: localhost:8081  
User-Agent: curl/7.81.0  
Accept: */*
```

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?

Porque el servidor falso está aceptando la solicitud desde un puerto LOCAL, por lo tanto no es necesario incluir localhost

Toma nota de los encabezados que ves: así es como un servidor web real percibe una conexión desde curl.

```
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ nc -l 8081
GET / HTTP/1.1
Host: localhost:8081
User-Agent: curl/7.81.0
Accept: */*
```

Encabezado:
GET / HTTP/1.1
Host: localhost:8081
User-Agent: curl/7.81.0
Accept: */*

Prueba curl --help para ver la ayuda y verificar que la línea de comando curl -i 'http://randomword.saasbook.info' mostrará ambos (BOTH) encabezados de respuesta del servidor y(AND) luego el cuerpo de la respuesta.

```
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ curl -i 'http://randomword.saasbook.info'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/html; charset=utf-8
Content-Length: 482
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: Sun, 24 Sep 2023 22:39:17 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/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-giJF6kkoqN00vy+HMDP7azOuL0xtbfIcaT9wjKhr8RbDVddVHyTfAASrekwKmp1" crossorigin="anonymous">
    <title>Random Word Generator</title>
  <body class="container">
    <div id="image">
      
    </div>
    <div id="word">
      beginner
    </div>
  </body>
</html>
```

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?

```
For all options use the name of the header.  
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ curl -i 'http://randomword.saasbook.info'  
HTTP/1.1 200 OK  
Connection: keep-alive  
Content-Type: text/html; charset=utf-8  
Content-Length: 482
```

Basicamente la línea de la flecha roja responde a la pregunta, el http/1.1 indica la que la versión de protocolo que se usó es 1.1 y el número 200 indica que la solicitud se realizó con éxito.

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?

Si, en la imagen de abajo se aprecia la información que proporciona el encabezado para que el cliente interprete el resultado, por ejemplo Content-Type indica el formato del cuerpo de la respuesta, en este caso será de formato html

```
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ curl -i
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/html; charset=utf-8
Content-Length: 482
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: Sun, 24 Sep 2023 22:39:17 GMT
Via: 1.1 vegur
```

¿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? Pruéba esto utilizando el procedimiento anterior.

Como se observa en la imagen de abajo, pusimos un link inexistente y no retorna un cuerpo html en la respuesta y en la primera linea del encabezado, devuelve el numero 404 este es un error que significa que lo que se solicito no se encontró

```
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ curl -i 'http://randomword.saasbook.info/extrainexsitente'  
HTTP/1.1 404 Not Found  
Connection: keep-alive  
X-Cascade: pass  
Content-Type: text/html; charset=utf-8  
Content-Length: 21  
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: Sun, 24 Sep 2023 23:12:38 GMT  
Via: 1.1 vegur
```

¿Qué otros códigos de error HTTP existen? Utiliza Wikipedia u otro recurso para conocer los significados de algunos de los más comunes: 200, 301, 302, 400, 404, 500. Ten en cuenta que estas son familias de estados: todos los estados 2xx significan funcionó, todos los 3xx son redireccionar etc.

200 (OK): Indica que la solicitud fue realizada con éxito y si se encontró la información solicitada

301 (Moved Permanently): indica que el host **si ha sido capaz** de comunicarse con el servidor pero que el recurso solicitado ha sido movido a otra dirección **permanentemente**. [1]

302 (Found): se produce cuando el recurso solicitado ha sido trasladado **temporalmente** a una nueva ubicación [2]

400 (Bad Request): El error 400 solicitud incorrecta ocurre cuando el servidor no puede entender la petición. Por lo tanto, no la procesa y te envía el código de error en su lugar. [3]

404 (Not Found): Este error indica que la información solicitada al servidor no existe y por lo tanto no ha sido encontrada

500 (Internal Server Error): El error HTTP 500, en particular, indica que el servidor ha encontrado una condición inesperada que le impidió cumplir con la solicitud.

En otras palabras, el servidor de alojamiento no puede determinar el problema exacto y mostrar un mensaje más específico [4]

Tanto el encabezado 4xx como el 5xx indican condiciones de error. ¿Cuál es la principal diferencia entre 4xx y 5xx?.

Los errores del tipo 4xx son errores causados por el cliente, como por ejemplo poner una url inexistente en la solicitud (votará error 404) en cambio los errores del tipo 5xx son errores del servidor, no del cliente, al momento de procesar la solicitud del cliente

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?

Resultado del primer comando GET/

```
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ 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/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< 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 00:34:58 GMT
< Via: 1.1 vegur
<
* Connection #0 to host esaas-cookie-demo.herokuapp.com left intact
Logged in: falsejuan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$
```


Resultado del segundo comando GET/login

```
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ 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/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< 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 00:47:09 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
```

En la flecha roja se indica una parte de la cabecera en donde se establece un cookie y se configura algunos parametros como logged_in=true etc.. Esto no se encuentra en la anterior captura de la diapositiva anterior.


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

Lo que sucede es que la cookie solo se envía al servidor cuando el cliente lo especifica en la solicitud, es por eso que cuando hacemos el Get/ (curl -v '<http://esaas-cookie-demo.herokuapp.com>') no especificamos alguna cookie (login_in = true) por lo tanto el servidor me responde como si no estuviera conectado

Ahora debemos decirle a curl que incluya las cookies apropiadas de este archivo cuando visite el sitio, lo cual hacemos con la opción -b:

```
curl -v -b cookies.txt http://esaas-cookie-demo.herokuapp.com/
```

El resultado que se muestra en la flecha roja indica que la cookie ha sido enviada, y el servidor a respondido con un logged in: true

```
juan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$ curl -v -b cookies.txt http://esaas-cookie-demo.herokuapp.com/
* Trying 54.146.248.82:80...
* Connected to esaas-cookie-demo.herokuapp.com (54.146.248.82) port 80 (#0)
> GET / HTTP/1.1
> Host: esaas-cookie-demo.herokuapp.com
> User-Agent: curl/7.81.0
> Accept: */*
> Cookie: logged_in=true 
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Connection: keep-alive
< Content-Type: text/plain; charset=utf-8
< Content-Length: 15
< X-Content-Type-Options: nosniff
< Server: WEBrick/1.6.1 (Ruby/2.7.8/2023-03-30)
< Date: Mon, 25 Sep 2023 02:04:23 GMT
< Via: 1.1 vegur
<
* Connection #0 to host esaas-cookie-demo.herokuapp.com left intact
Logged in: truejuan@juan-Lenovo-C365:~/Cursos/DS/HTTP-URI$
```

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?

- Una opción puede ser la etiqueta HttpOnly, esta es una etiqueta agregada a una cookie del navegador que evita que los scripts del lado del cliente accedan a los datos. [5]
- Autenticación basada en token, cuando un usuario inicia sesión luego un servidor de autorización valida esa autenticación inicial y luego emite un token de acceso, que es un pequeño dato que le permite a una aplicación de cliente realizar una llamada o señal segura a un servidor API, de esta manera el token funciona como un boleto de acceso para que el usuario acceda a todos los recursos, el ciclo de vida del token finaliza cuando el usuario cierra sesión/abandona sesión [6]

Bibliografias:

- [1] Obtenido de: https://es.wikipedia.org/wiki/HTTP_301
- [2] Obtenido de: <https://www.hostinger.es/tutoriales/http-302>
- [3] Obtenido de: <https://www.hostinger.es/tutoriales/error-400>
- [4] Obtenido de: <https://es.siteground.com/kb/500-internal-error/>
- [5] Obtenido de: <https://www.cookiepro.com/knowledge/httponly-cookie/>
- [6] Obtenido de <https://www.entrust.com/es/resources/faq/what-is-token-based-authentication>