SISTEMAS WEB
CURSO 2021/2022

HTTP - HyperText Transfer Protocol

Solicitud y respuesta. Ejemplo utilizando Python: librería *requests*
Códigos de Error: 400 y 404.
Tráfico derivado de la solicitud de una página web. Herramientas de desarrollo en navegador.

M 04-02-2022

Oskar Casquero y María Luz Álvarez
Dpto. Ingeniería de Sistemas y Automática

FUNCIONAMIENTO DE HTTP: SOLICITUD DEL CLIENTE

Sintaxis de una solicitud HTTP

```
Método  URI HTTP/1.1
Cabeceras
CRLF
Cuerpo del mensaje
```

solicitud HTTP del ejemplo

```
GET /recurso HTTP/1.1
Host: sw2022.com:8080
Accept: text/html
Accept-Encoding: gzip,identity;q=0.5
Accept-Language: en-US,es-ES;q=0.8
User-Agent: Mozilla Windows Desktop
```

Método: GET

El método describe el tipo de acción CRUD (Create, Read, Update and Delete) que se desea llevar a cabo sobre el recurso. En este caso, GET → Lectura.

URI: /recurso

La identificación del recurso puede realizarse con el URI completo o con el URI relativo.

GET http://sw2022.com:8080/recurso HTTP/1.1

GET /recurso HTTP/1.1
Host: sw2022.com:8080

Cabeceras: indican las características del cliente y sus preferencias en la respuesta.

Accept: el navegador indica que acepta contenido en HTML.

Accept-Encoding: el navegador indica que prefiere contenido comprimido (en formato *gzip*), aunque también acepta contenido no comprimido (*identity*)

Accept-Language: el navegador indica que su preferencia de idioma es el inglés y su segunda opción es el castellano.

User-Agent: el navegador se identifica como Mozilla sobre una plataforma Windows de escritorio.

Cuerpo del mensaje: en este caso está vacío.

FUNCIONAMIENTO DE HTTP: RESPUESTA DEL SERVIDOR

Sintaxis de una respuesta HTTP

```
HTTP/1.1 Status Descripción
Cabeceras
CRLF
Cuerpo del mensaje (en octetos)
```

Respuesta HTTP del ejemplo

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 20:25:52 GMT
Last-Modified: Tue, 17 Sep 2013 13:00:02 GMT
ETag: "1a968-3ec-4e693e61bb8b6"
Content-Length: 76
Content-Type: text/html; charset=ISO-8859-1

<html><head><title>index.html</title></head>
<body>Hello World!</body></html>
```

Status: 200

Código que describe el resultado de la solicitud

Concretamente, el código 200 indica que la petición está bien formada y que ha sido procesada correctamente.

Para programas.

Descripción: OK

Descripción textual asociada al Status.

Para usuarios.

Cabeceras: caracterizan determinados aspectos de la respuesta.

Content-Length: el servidor indica la longitud (número de octetos) del contenido de la respuesta.

Content-Type: el servidor indica que el contenido es de tipo HTML y que sus octetos están codificados en latin-1.

FUNCIONAMIENTO DE HTTP

RESPUESTA DEL SERVIDOR

Sintaxis de una respuesta HTTP

HTTP/1.1 Status Descripción
Cabeceras
CRLF
Cuerpo del mensaje (en octetos)

Respuesta HTTP del ejemplo

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 20:25:52 GMT
Last-Modified: Tue, 17 Sep 2013 13:00:02 GMT
ETag: "1a968-3ec-4e693e61bb8b6"
Content-Length: 76
Content-Type: text/html; charset=ISO-8859-1

<html><head><title>index.html</title></head>
<body>Hello World!</body></html>
```

Cabeceras: (continuación)

Date: fecha en la que el servidor creo la respuesta (formato definido en RFC 822, apartado 5 , 1s de resolución).

Last-Modified: fecha de la última modificación del recurso.

ETag: identificador de entidad*; se usa para distinguir dos versiones del mismo recurso, por ejemplo:

URI: http://ws2022.com:8080/recurso

```
Last-Modified: Tue, 17 Sep 2013 13:00:02 GMT
Content-Length: 12
Content-Type: text/plain; charset=ISO-8859-1

Hello World!
```

```
Last-Modified: Tue, 17 Sep 2013 13:00:02 GMT
Content-Length: 76
Content-Type: text/html; charset=ISO-8859-1

<html><head><title>index.html</title></head>
<body>Hello World!</body></html>
```

* entidad: conjunto de determinadas cabeceras y cuerpos del mensaje(RFC 2616, apartado 7).

Cuerpo del mensaje: contenido; en este caso, documento HTML (página web).

EJEMPLO UTILIZANDO PYTHON : LIBRERÍA *REQUESTS*

– Utilizando la librería *requests* de Python solicita el siguiente recurso: http://www.httpwatch.com/httpgallery/chunked/chunkedimage.aspx

```
import requests

# La petición tiene 4 partes: metodo, uri, cabecera y cuerpo
metodo = 'GET'
uri = "http://www.httpwatch.com/httpgallery/chunked/chunkedimage.aspx"
cabeceras= {'Host': 'www.httpwatch.com'}
cuerpo= ''
respuesta= requests.request(metodo, uri, headers=cabeceras, data=cuerpo)

# La respuesta tiene también 4 apartados: codigo, descripción, cabeceras eta
cuerpo
codigo= respuesta.status_code
descripcion= respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])
cuerpo= respuesta.content
print(cuerpo)

fichero= open("imagen.jpg", 'wb')
fichero.write(cuerpo)
fichero.close()
```

FUNCIONAMIENTO DE HTTP: ERRORES

- **¿Qué ocurre si una petición no puede ser satisfecha?**

  Códigos de respuesta 4xx y 5xx.

  – Cuando el cliente o el servidor no puede completar una solicitud, se devuelven los siguientes códigos de estado (STATUS) en la respuesta.

Errores provocados por el **cliente**:
4xx (RFC 2616, Sección 10.4)
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- …

Errores provocados por el **servidor**:
5xx (RFC 2616, Sección 10.5):
- 500 Internal Server Error
- 503 Service Unavailable
- …

FUNCIONAMIENTO DE HTTP
ERROR 400

- **400 Bad Request**
  - El servidor web no entiende la petición, su sintaxis/semántica es incorrecta.
  - Supongamos que se solicita el recurso */html/main page.html*

Petición de ejemplo

```
GET /html/main page.html HTTP/1.1
Host: sw2022.com
Accept: text/html
Accept-Encoding: gzip,identity;q=0.5
Accept-Language: en-US,es-ES;q=0.8
User-Agent: Mozilla Windows Escritorio
```

Respuesta de ejemplo

```
HTTP/1.1 400 Bad Request
Date: Wed, 25 Nov 2015 08:07:43 GMT
Content-Length: 138
Content-Type: text/html; charset=UTF-8

<html><head><title>Error 400 (Bad
Request)</title></head><body><p>Your
client has issued a malformed or illegal
request.</p></body></html>
```

Lo que interpreta el servidor al procesar la petición:
- Método: GET
- RequestURI: /html/main
- Versión del protocolo: page.html -> SEMÁNTICA ERRONEA (La cadena debe ser HTTP/1.1)
- Cadena adicional: HTTP/1.1 -> SINTAXIS ERROEA (la primera línea tiene que tener 3 elementos)

Solución: Codificar el espacio en la RequestURI: /html/main%20page.html

EJEMPLO: ERROR 400 (SEMÁNTICA ERRÓNEA)

– En el código del ejemplo anterior, añadir al final del nombre de la cabecera Host un espacio.

```
import requests

# La petición tiene 4 partes: metodo, uri, cabecera y cuerpo
metodo = 'GET'
uri = "http://www.httpwatch.com/httpgallery/chunked/chunkedimage.aspx"
cabeceras= {'Host ': 'www.httpwatch.com'}
cuerpo= ''
respuesta= requests.request(metodo, uri, headers=cabeceras, data=cuerpo)

# La respuesta tiene también 4 apartados: codigo, descripción, cabeceras eta
cuerpo
codigo= respuesta.status_code
descripcion= respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])
cuerpo= respuesta.content
print(cuerpo)

fichero= open("imagen.jpg", 'wb')
fichero.write(cuerpo)
fichero.close()
```

FUNCIONAMIENTO DE HTTP
ERROR 404

- **404 Not Found**
  - La petición esta bien formada, pero el servidor web **no encuentra el recurso.**
  - Supongamos que se solicita el recurso */login/index.php*, que no existe en el servidor

Petición de ejemplo

```
GET /login/index.php  HTTP/1.1
Host: sw2022.com
Accept: text/html
Accept-Encoding: gzip,identity;q=0.5
Accept-Language: en-US,es-ES;q=0.8
User-Agent: Mozilla Windows Escritorio
```

Respuesta de ejemplo

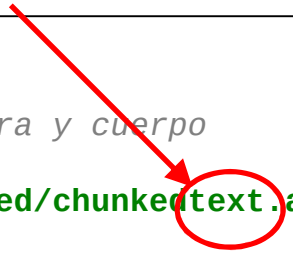
```
HTTP/1.1 404 Not Found
Date: Wed, 25 Nov 2015 08:07:43 GMT
Content-Length: 134
Content-Type: text/html; charset=UTF-8

<html><head><title>Error 404 (Not
Found)</title></head><body><p>The
requested resource was not found in this
server.</p></body></html>
```

EJEMPLO: ERROR 404

- En el código del ejemplo anterior, cambiar el nombre del recurso:

/httpgallery/chunked/chunked<u>text</u>.aspx



```
import requests

# La petición tiene 4 partes: metodo, uri, cabecera y cuerpo
metodo = 'GET'
uri = "http://www.httpwatch.com/httpgallery/chunked/chunkedtext.aspx"
cabeceras= {'Host': 'www.httpwatch.com'}
cuerpo= ''
respuesta= requests.request(metodo, uri, headers=cabeceras, data=cuerpo)

# La respuesta tiene también 4 apartados: codigo, descripción, cabeceras eta
cuerpo
codigo= respuesta.status_code
descripcion= respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])
cuerpo= respuesta.content
print(cuerpo)

fichero= open("imagen.jpg", 'wb')
fichero.write(cuerpo)
fichero.close()
```