
Tarea programada #1: Servidor web miniatura

Esteban Castillo Gamboa B31548

27 de septiembre de 2017

Índice

1	Introducción	2
1.1	Descripción del problema	2
1.2	Descripción de la metodología	2
2	Análisis del problema	2
3	Diseño de clases (UML)	2
4	Casos de prueba	3
5	Resultados de los casos de prueba	5
6	Análisis de los resultados de las pruebas	7

1. Introducción

Esta sección explica el problema a resolver y el cómo se resolvió.

1.1. Descripción del problema

Se debe implementar un servidor web capaz de resolver *request* con métodos GET, HEAD y POST del protocolo HTTP. Se pueden utilizar lenguajes de alto nivel que cuenten con funcionalidades de *sockets*, pero no se pueden utilizar bibliotecas HTTP.

1.2. Descripción de la metodología

Debido a la restricción de bibliotecas HTTP se escogió el lenguaje de programación Python, específicamente Python 3.6, el cual ofrece muchas facilidades a la hora de recibir y tratar datos enviados por un *socket*. Para las pruebas se utilizó el navegador Firefox y el cliente cURL, que permite enviar *request* HTTP por consola.

2. Análisis del problema

Para solucionar el problema se debe garantizar que el servidor logre crear *socket* para que los clientes que se conecten a él puedan enviar los *request*. Esto se logra haciendo que el cliente abra conexión y espere infinitamente hasta que algún cliente se conecte. Cuando el cliente se conecta, procede a enviar el *request* y el servidor debe ser capaz de revisarlo y determinar qué acción debe realizar. Debe tener en cuenta el tipo de método, si el *request* pide el BODY de algún archivo o si el archivo existe. Para efectos de la tarea, se trabaja con los siguientes códigos de retorno mostrados en el cuadro 1.

Cuadro 1: Códigos de retornos HTTP para la implementación del servidor.

Código	Respuesta
200	OK
404	Not Found
406	Not Acceptable
501	Not Implemented

Se debe tener claro qué respuesta se debe devolver dependiendo del tipo de método. Por ejemplo, un método HEAD nunca retorna el BODY de un archivo. Adicionalmente, se creó un *script* que lleva un registro de todos los *request* que se le han realizado al servidor.

3. Diseño de clases (UML)

La figura 1 muestra el diseño de clases pensado en Python para la implementación del servidor.

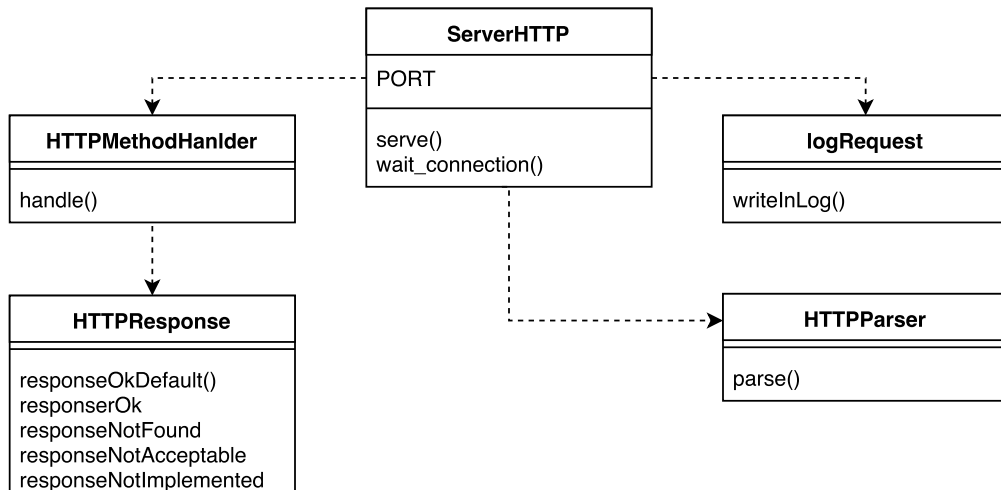


Figura 1: Diagrama UML para el servidor HTTP realizado en Python.

4. Casos de prueba

A continuación, se muestran los casos de prueba con sus salidas esperadas.

Caso de prueba 1: *Request* directo al localhost.

```
>curl -i http://localhost:8000/

HTTP/1.1 200 OK
Date: Fri, 29 May 2015 13:24:18 GMT
Server: MiServidor/1.0
Content-Length: 44
Content-Type: text/html

<html><body><h1>Bienvenido</h1></body></html>
```

Caso de prueba 2: *Request* hacia al index.html del servidor.

```
>curl http://localhost:8000/

<html><body><h1>Bienvenido</h1></body></html>
```

Caso de prueba 3: *Request* para probar la respuesta 406.

```
>curl -i -H "Accept: image/gif" http://localhost:8000/index.html
```

Caso de prueba 4: *Request* para probar el método HEAD.

```
>curl -i -X HEAD http://localhost:8000/index.html

HTTP/1.1 200 OK
Date: Fri, 29 May 2015 13:27:37 GMT
Server: MiServidor/1.0
Content-Length: 44
Content-Type: text/html

curl: (18) transfer closed with 44 bytes remaining to read
```

Caso de prueba 5: *Request* para probar el método POST.

```
>curl -i -X POST -d "mensaje=Hola+Mundo" http://localhost:8000/index.html

HTTP/1.1 200 OK
Date: Fri, 29 May 2015 13:28:46 GMT
Server: MiServidor/1.0
Content-Length: 44
Content-Type: text/html

<html><body><h1>Bienvenido</h1></body></html>
```

Caso de prueba 6: *Request* para probar GET con archivos .jpg.

```
> http://localhost:8000/pintura.jpg

HTTP/1.1 200 OK
Date: Fri, 29 May 2015 13:28:46 GMT
Server: MiServidor/1.0
Content-Length: 152150
Content-Type: text/html
```

Caso de prueba 7: *Request* para probar GET con archivos .txt.

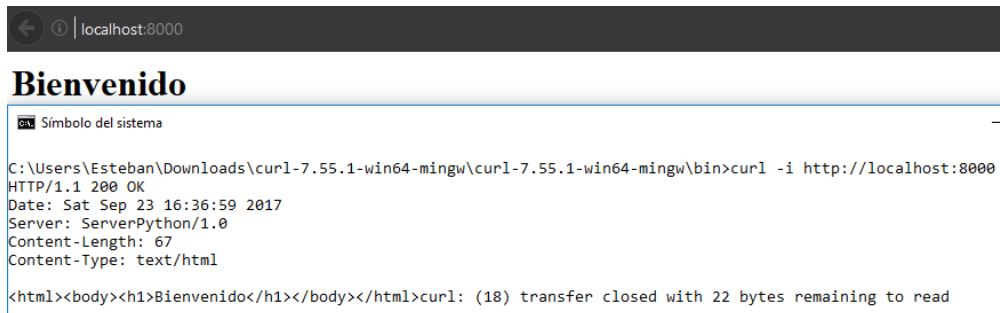
```
> http://localhost:8000/alice.txt

HTTP/1.1 200 OK
Date: Fri, 29 May 2015 13:28:46 GMT
Server: MiServidor/1.0
Content-Length: 152141
Content-Type: text/html
```

5. Resultados de los casos de prueba

A continuación se muestran los resultados de los casos de prueba.

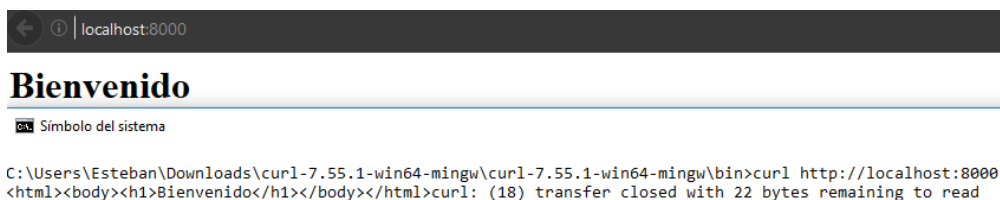
Resultado de caso de prueba 1



```
localhost:8000
Bienvenido
C:\Users\Esteban\Downloads\curl-7.55.1-win64-mingw\curl-7.55.1-win64-mingw\bin>curl -i http://localhost:8000
HTTP/1.1 200 OK
Date: Sat Sep 23 16:36:59 2017
Server: ServerPython/1.0
Content-Length: 67
Content-Type: text/html
<html><body><h1>Bienvenido</h1></body></html>curl: (18) transfer closed with 22 bytes remaining to read
```

Figura 2: Resultado del caso de prueba 1.

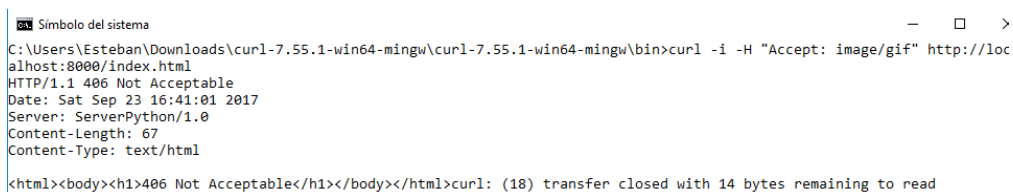
Resultado de caso de prueba 2



```
localhost:8000
Bienvenido
C:\Users\Esteban\Downloads\curl-7.55.1-win64-mingw\curl-7.55.1-win64-mingw\bin>curl http://localhost:8000
<html><body><h1>Bienvenido</h1></body></html>curl: (18) transfer closed with 22 bytes remaining to read
```

Figura 3: Resultado del caso de prueba 2.

Resultado de caso de prueba 3



```
localhost:8000
406 Not Acceptable
C:\Users\Esteban\Downloads\curl-7.55.1-win64-mingw\curl-7.55.1-win64-mingw\bin>curl -i -H "Accept: image/gif" http://localhost:8000/index.html
HTTP/1.1 406 Not Acceptable
Date: Sat Sep 23 16:41:01 2017
Server: ServerPython/1.0
Content-Length: 67
Content-Type: text/html
<html><body><h1>406 Not Acceptable</h1></body></html>curl: (18) transfer closed with 14 bytes remaining to read
```

Figura 4: Resultado del caso de prueba 3.

Resultado de caso de prueba 4

```
C:\Users\Esteban\Downloads\curl-7.55.1-win64-mingw\curl-7.55.1-win64-mingw\bin>curl -i -X HEAD http://localhost:8000/index.html
Warning: Setting custom HTTP method to HEAD with -X/--request may not work the
Warning: way you want. Consider using -I/--head instead.
HTTP/1.1 200 OK
Date: Sat Sep 23 16:47:00 2017
Server: ServerPython/1.0
Content-Length: 67
Content-Type: text/html

curl: (18) transfer closed with 67 bytes remaining to read
```

Figura 5: Resultado del caso de prueba 4.

Resultado de caso de prueba 5

```
C:\Users\Esteban\Downloads\curl-7.55.1-win64-mingw\curl-7.55.1-win64-mingw\bin>curl -i -X POST -d "mensaje=Hola+Mundo" http://localhost:8000/index.html
HTTP/1.1 200 OK
Date: Sat Sep 23 16:48:47 2017
Server: ServerPython/1.0
Content-Length: 67
Content-Type: text/html

<html><body><h1>Bienvenido</h1></body></html>curl: (18) transfer closed with 22 bytes remaining to read
```

Figura 6: Resultado del caso de prueba 5.

Resultado de caso de prueba 6

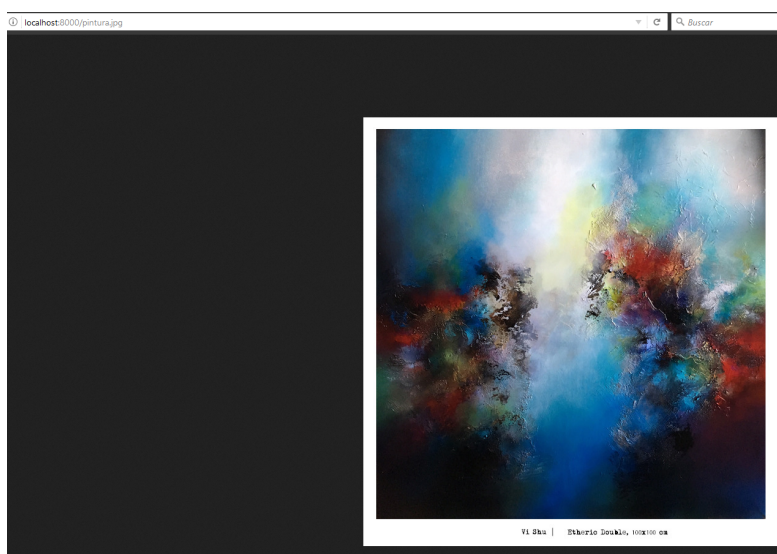


Figura 7: Resultado del caso de prueba 6.

Resultado de caso de prueba 7

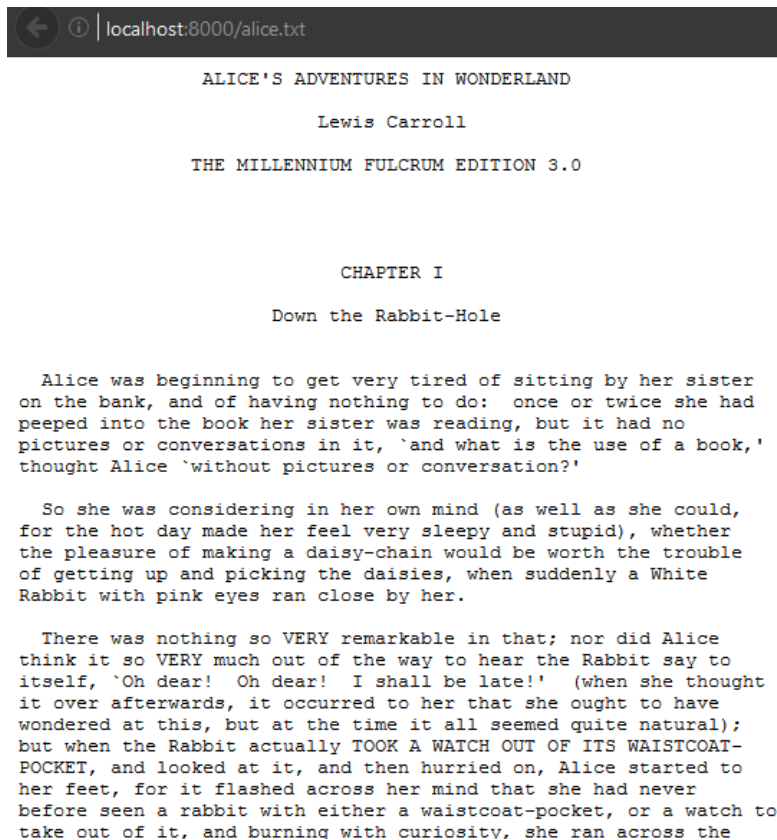


Figura 8: Resultado del caso de prueba 7.

6. Análisis de los resultados de las pruebas

Al realizar las pruebas que se sugerían en el enunciado se puede comprobar el funcionamiento correcto del manejador de respuestas. Se puede observar en los *request* que se hacen desde el navegador que, efectivamente, se están renderizando los archivos HTML y se muestran las imágenes y los archivos de texto. Adicionalmente, para comprobar que los otros métodos como HEAD y POST funcionan correctamente se realizan *requests* a través de la consola de cURL, donde se puede observar que la respuesta que envía el servidor es consistente con lo que se esperaba que respondiera.