PYTHON- WEB SERVER

CMPT 371 - Data Communications and Networking

Group 1

Abhay Jolly (301383885)
Amritpal Singh (301336774)

Submitted to: Ouldooz Baghban Karimi
March 23, 2021
Simon Fraser University

TABLE OF CONTENT

LIST OF FIGURES

1. Introduction

The python web server is implemented which can send responses to client requests with these status codes 200, 304, 400, 404 and 408.

2. Single Thread

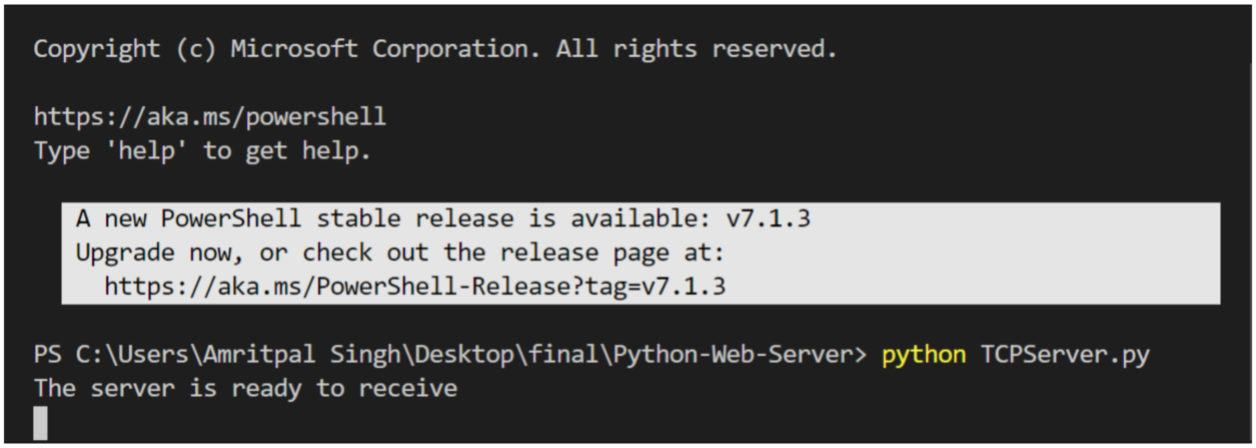
The web server implemented is single threaded which means that the server is only able to accept one request at a time. Two clients if requested at the same time then the server will only accept one request and the other one may get discarded if the server is processing another request.

3. Status Code 200

The status code 200 is used when the request is fulfilled by the server and it is also represented as a standard successful request.

3.1. Testing Status Code 200

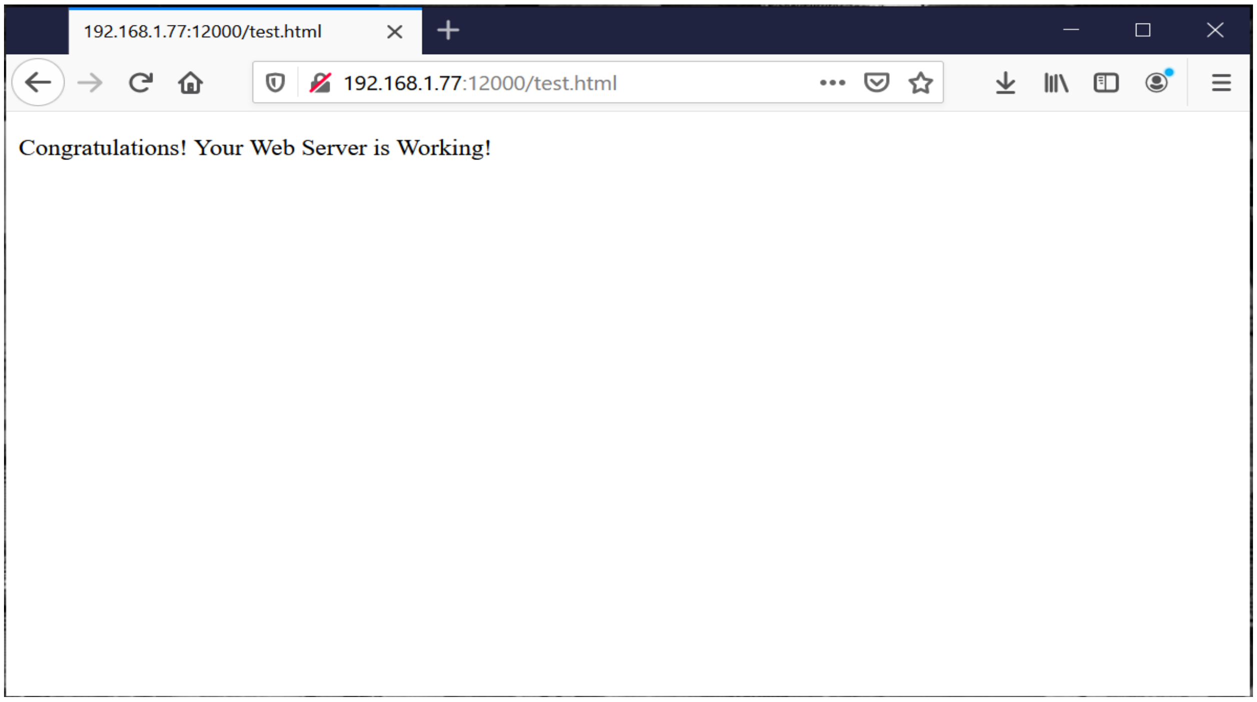
First we run our server-> This is obviously a basic step that we will always have to do.



```
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/powershell
Type 'help' to get help.

  A new PowerShell stable release is available: v7.1.3
  Upgrade now, or check out the release page at:
    https://aka.ms/PowerShell-Release?tag=v7.1.3

PS C:\Users\Amritpal Singh\Desktop\final\Python-Web-Server> python TCPServer.py
The server is ready to receive
```

Fig 3.1.1: Starting Web Server

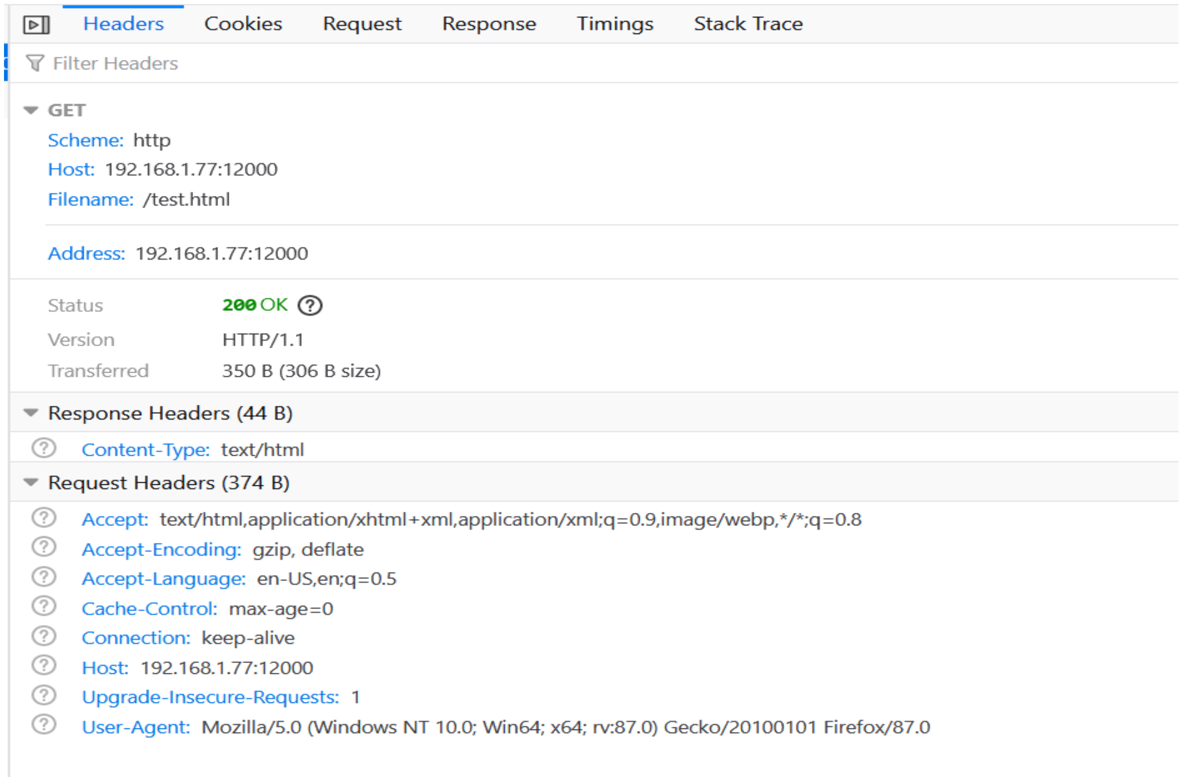
First test is simple for status code 200. We simply do a GET request for our test.html file from the server using the Mozilla browser of the format http://IP_ADDRESS:PORT/test.html
From the screenshot it's visible that 192.168.1.77 is the IP_ADDRESS and the PORT is 12000.
The server returns a file test.html which contains the output "Congratulations! Your Web Server is Working!"



Congratulations! Your Web Server is Working!

Fig 3.1.2: Response from the server with test.html Page

The following is the header information that the browser contained under Network which is the server response.



```
GET
Scheme: http
Host: 192.168.1.77:12000
Filename: /test.html

Address: 192.168.1.77:12000

Status        200 OK
Version       HTTP/1.1
Transferred   350 B (306 B size)

Response Headers (44 B)
Content-Type: text/html
Request Headers (374 B)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Cache-Control: max-age=0
Connection: keep-alive
Host: 192.168.1.77:12000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
```

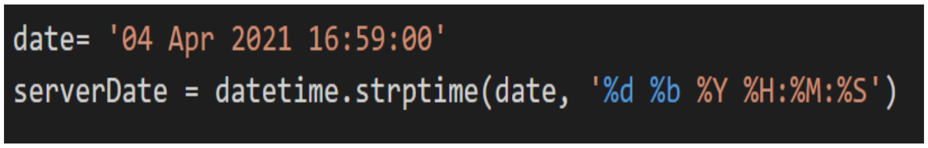
Fig 3.1.3: Server Response Header

4. Status Code 304

The status code 304 is used to indicate that the file or resource at the server side has been modified or not.

4.1. Testing Code 304 Modified

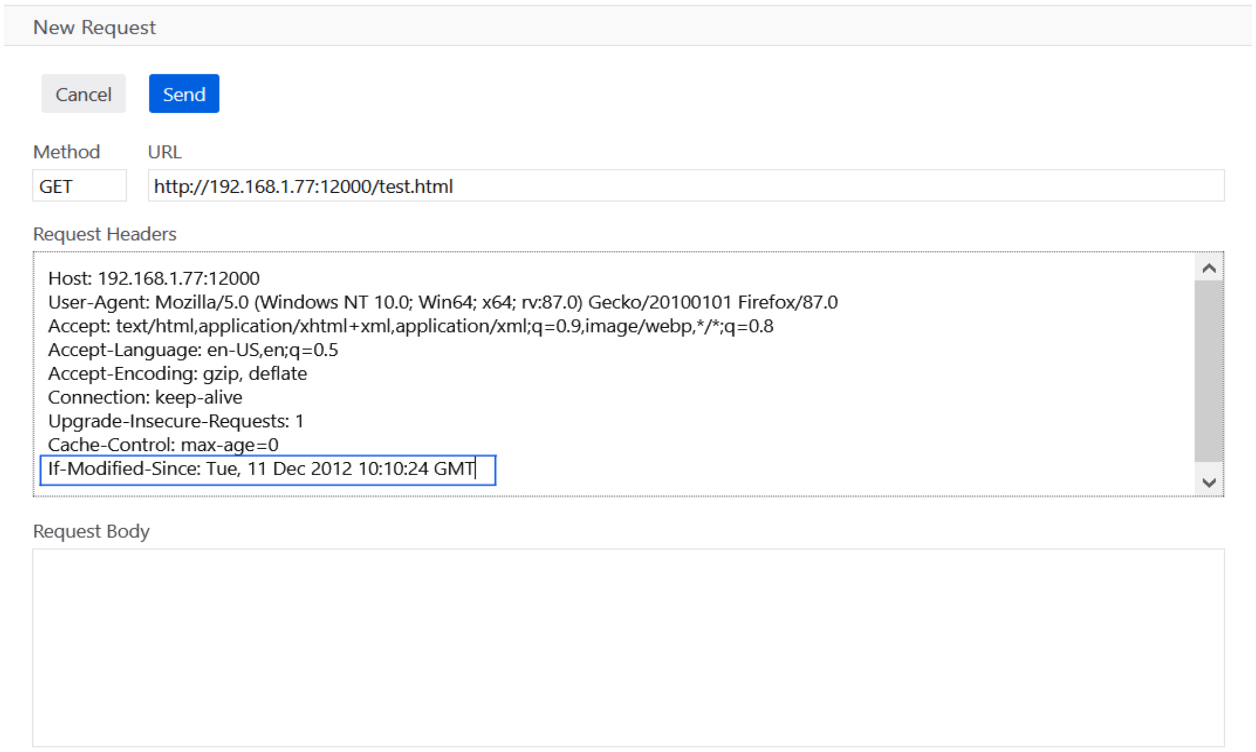
The below shows the date when the file was last modified. This will form our comparison for user's conditional GET requests whose header will contain 'If-Modified-Since'.



```
date= '04 Apr 2021 16:59:00'
serverDate = datetime.strptime(date, '%d %b %Y %H:%M:%S')
```

Fig 4.1.1: Server File Modified Date

Below is how we send our conditional GET request which has the date-> 11 Dec 2012 which is way before the above date-> 04 Apr 2021. This means the server is modified and a new file has to be returned by the server with status code 200.



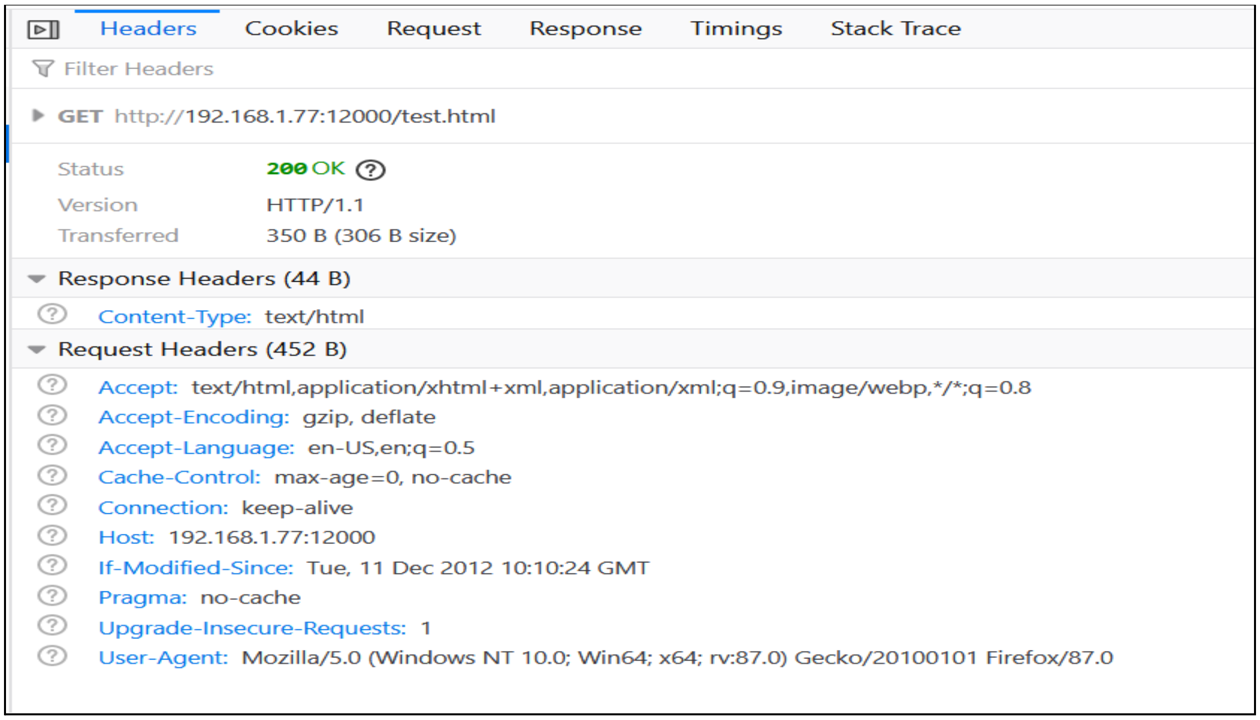
GET

http://192.168.1.77:12000/test.html

```
Host: 192.168.1.77:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
If-Modified-Since: Tue, 11 Dec 2012 10:10:24 GMT
```

Fig 4.1.2 Sending Conditional Request To Server including 'If-Modified-Since:' Date

The below screenshot shows what exactly happens, the new file is sent by the server.



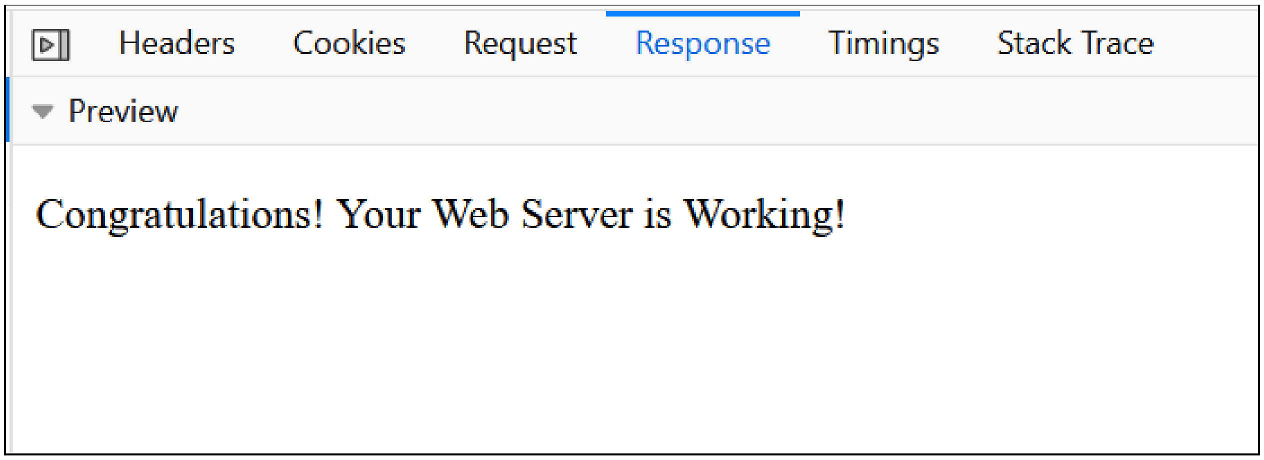
Headers | Cookies | Request | Response | Timings | Stack Trace

Filter Headers

```
GET http://192.168.1.77:12000/test.html

Status        200 OK
Version       HTTP/1.1
Transferred   350 B (306 B size)

Response Headers (44 B)
Content-Type: text/html
Request Headers (452 B)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Cache-Control: max-age=0, no-cache
Connection: keep-alive
Host: 192.168.1.77:12000
If-Modified-Since: Tue, 11 Dec 2012 10:10:24 GMT
Pragma: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
```

Fig 4.1.3 Server Response For the Conditional Get Request

The below is simply the response HTML page content from the server.



Headers | Cookies | Request | Response | Timings | Stack Trace

Preview

Congratulations! Your Web Server is Working!

Fig 4.1.4 HTML page output of server response

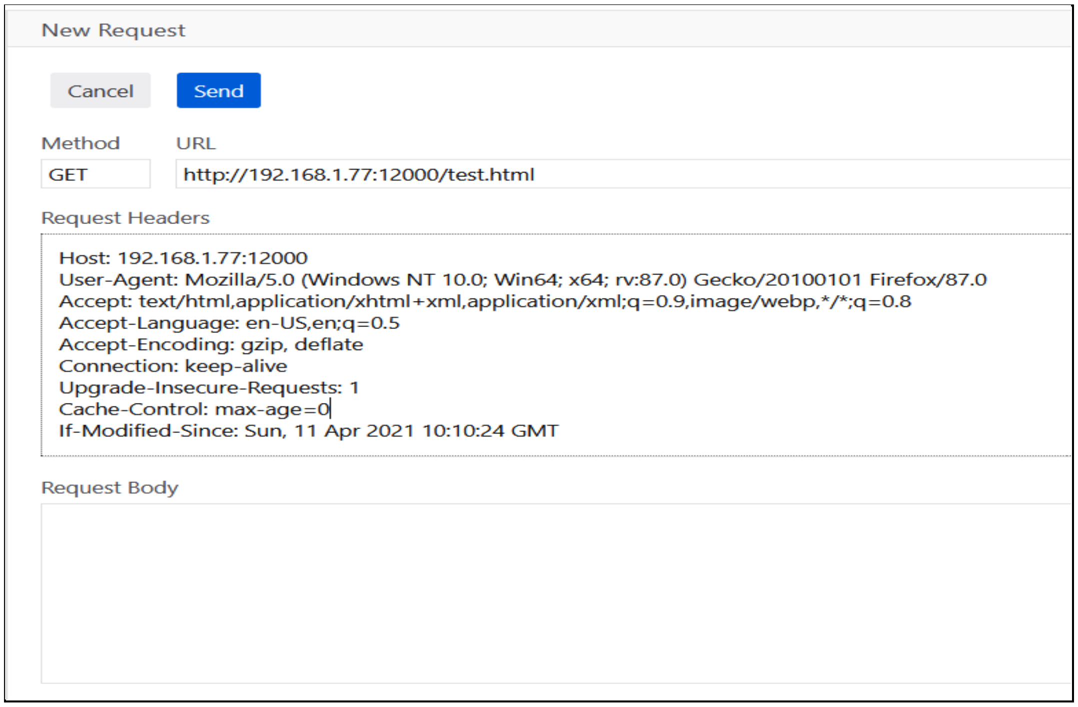
4.2. Testing Code 304 Not Modified

Similarly for the same date when the server was updated.

```
date= '04 Apr 2021 16:59:00'
serverDate = datetime.strptime(date, '%d %b %Y %H:%M:%S')
```

Fig 4.2.1: Server File Modified Date

Now we send a conditional GET request after the modified date. The desired output will be that no response is sent by the server but a header with status code 30



Method: GET
URL: http://192.168.1.77:12000/test.html

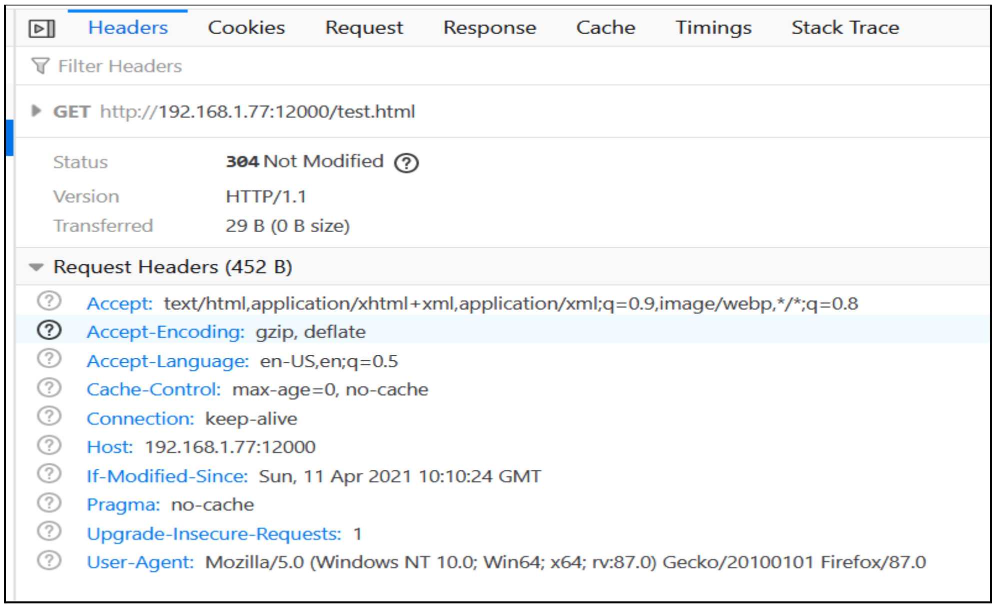
Request Headers

```
Host: 192.168.1.77:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
If-Modified-Since: Sun, 11 Apr 2021 10:10:24 GMT
```

Request Body

Fig 4.2.2 Sending Conditional Request To Server including ‘If-Modified-Since:’ Date

The below screenshot shows that the browser receives the response with status code 304 Not Modified and hence no response.



GET http://192.168.1.77:12000/test.html

Status	304 Not Modified
Version	HTTP/1.1
Transferred	29 B (0 B size)

Request Headers (452 B)

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Cache-Control: max-age=0, no-cache
Connection: keep-alive
Host: 192.168.1.77:12000
If-Modified-Since: Sun, 11 Apr 2021 10:10:24 GMT
Pragma: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
```

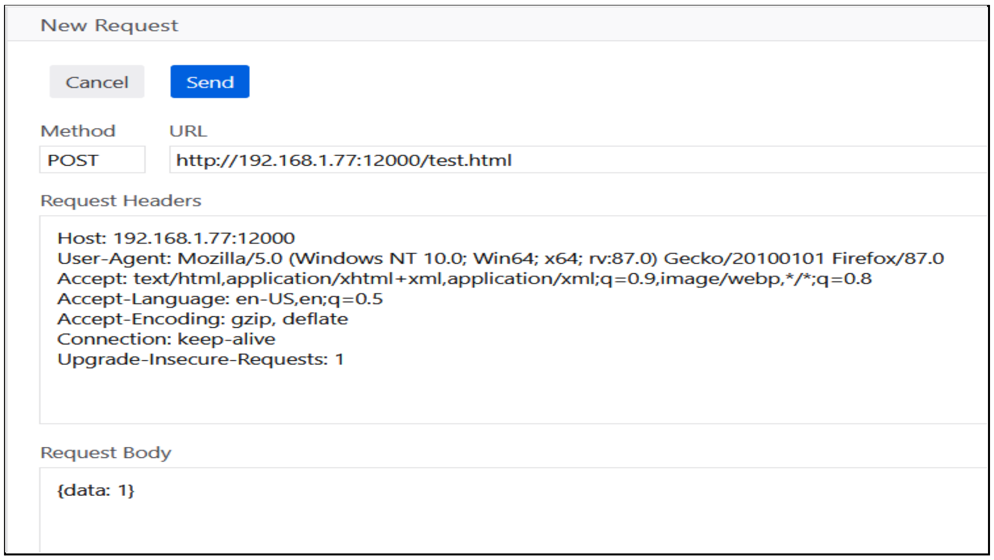
Fig 4.2.3: Response from Server for 304 Not Modified request

5. Status Code 400

The status code 400 is used when the server cannot or will not process the request due to client error.

5.1. Testing Status Code 400 with change method type

Using Mozilla Network Tab a POST request is sent to the server as described in the screen shot.



Method: POST
URL: http://192.168.1.77:12000/test.html

Request Headers

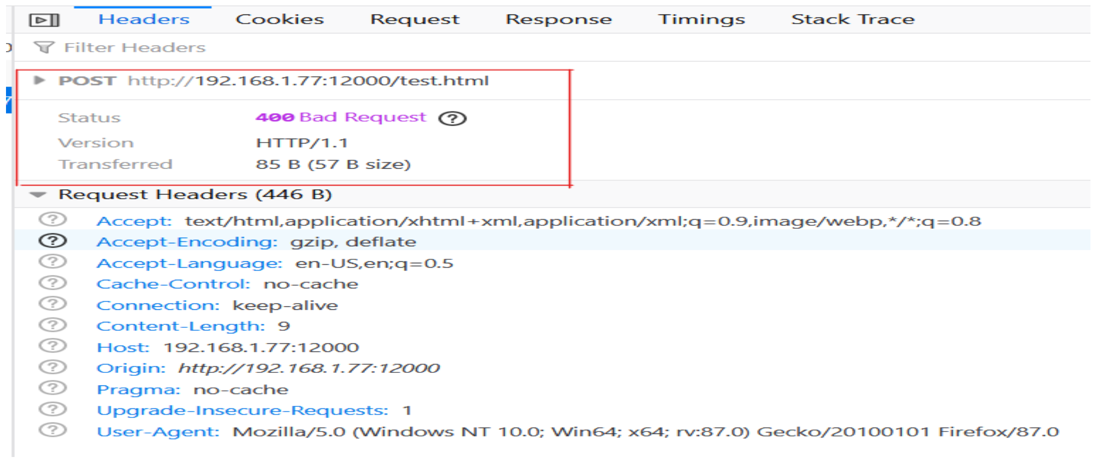
```
Host: 192.168.1.77:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Request Body

```
{data: 1}
```

Fig 5.1.1: Sending Post Request To Server

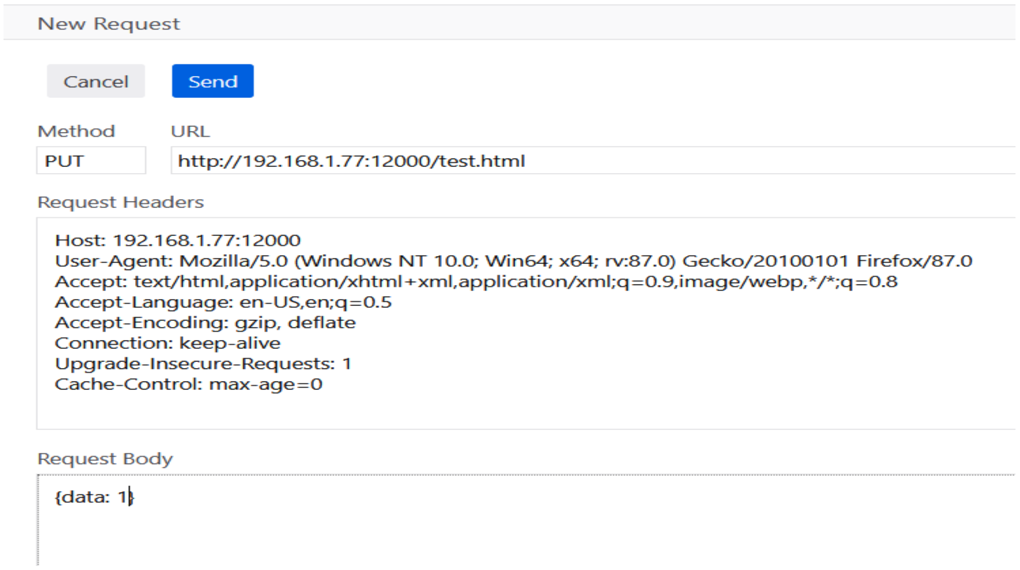
The server only contains the one html file and only able to accept GET requests. So the POST request will be considered as a bad request for the server and the screenshot below shows the response of the server.



```
POST http://192.168.1.77:12000/test.html

Status        400 Bad Request
Version       HTTP/1.1
Transferred   85 B (57 B size)

Request Headers (446 B)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Cache-Control: no-cache
Connection: keep-alive
Content-Length: 9
Host: 192.168.1.77:12000
Origin: http://192.168.1.77:12000
Pragma: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
```

Fig 5.1.2: Server Response to POST request

Similarly, a PUT request is sent to the server.



New Request

Cancel | Send

Method: PUT

URL: http://192.168.1.77:12000/test.html

Request Headers

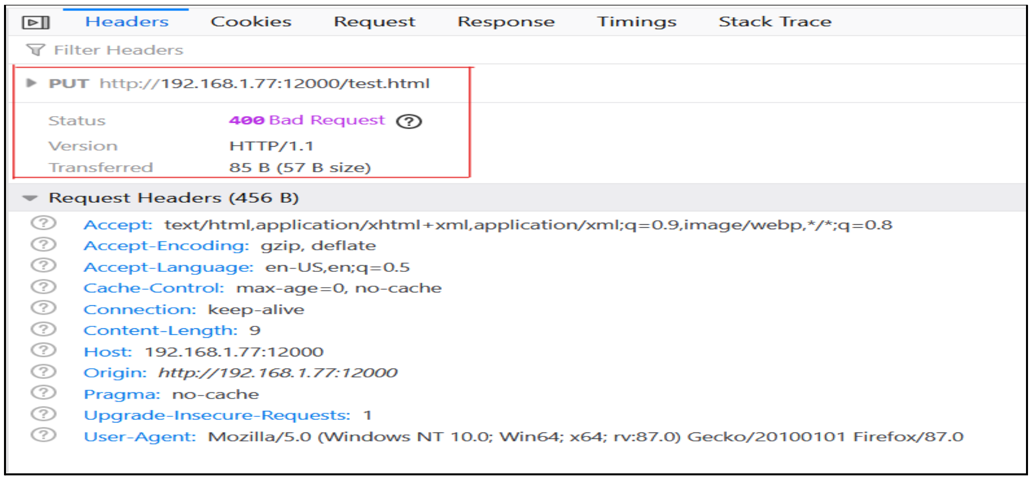
```
Host: 192.168.1.77:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Request Body

```
{data: 1}
```

Fig 5.1.3: Sending PUT Request To Server

The server responded with the bad request message.

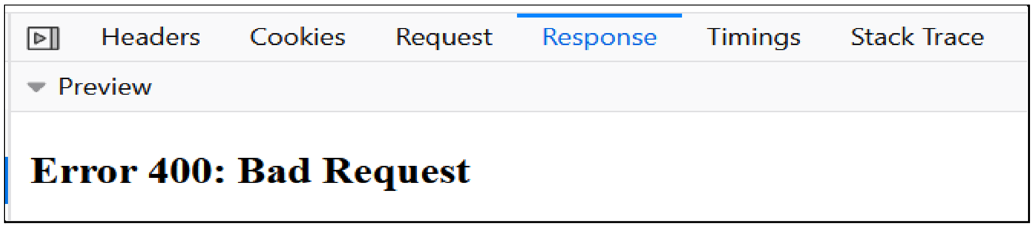


Headers | Cookies | Request | Response | Timings | Stack Trace

Filter Headers

```
PUT http://192.168.1.77:12000/test.html

Status        400 Bad Request
Version       HTTP/1.1
Transferred   85 B (57 B size)

Request Headers (456 B)
Accept:  text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Cache-Control:  max-age=0, no-cache
Connection:  keep-alive
Content-Length:  9
Host:  192.168.1.77:12000
Origin:  http://192.168.1.77:12000
Pragma:  no-cache
Upgrade-Insecure-Requests:  1
User-Agent:  Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
```

Fig 5.1.4: Server Response to PUT request



Headers | Cookies | Request | Response | Timings | Stack Trace

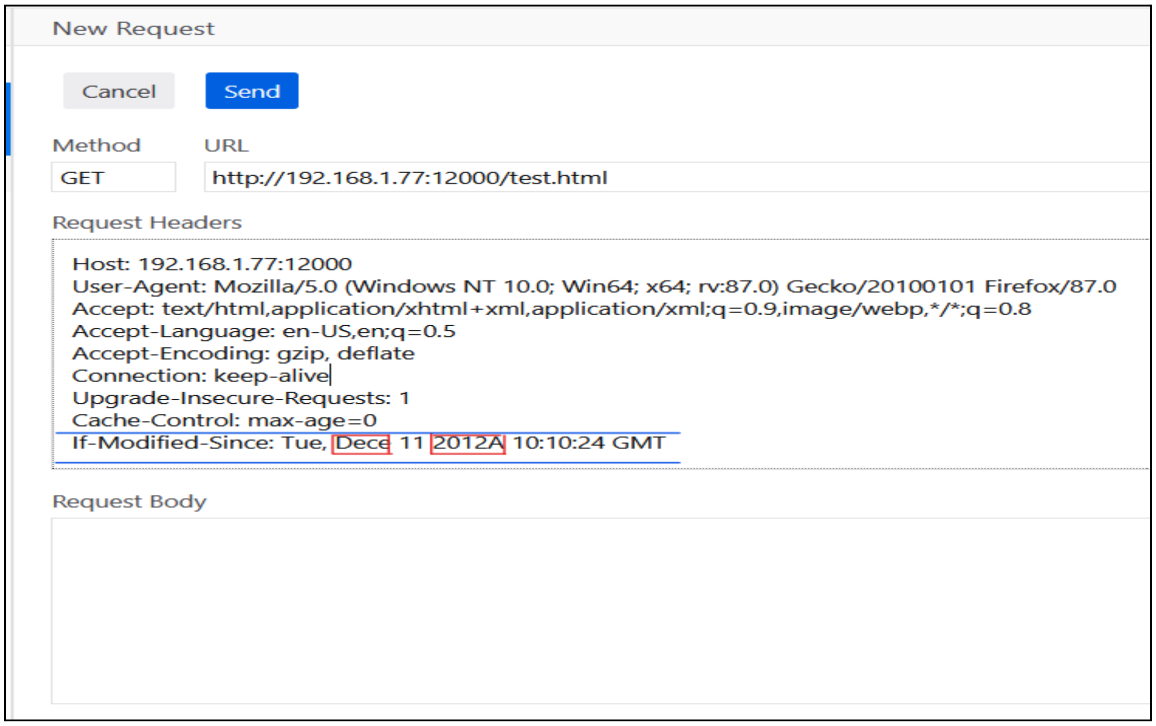
Preview

Error 400: Bad Request

Fig 5.1.5: Response HTML page from server

5.2. Testing Status Code 400 with Error in request

Testing with the scenario when there is error in the date value as described in the screenshot below.


New Request

Cancel Send

Method: GET

URL: http://192.168.1.77:12000/test.html

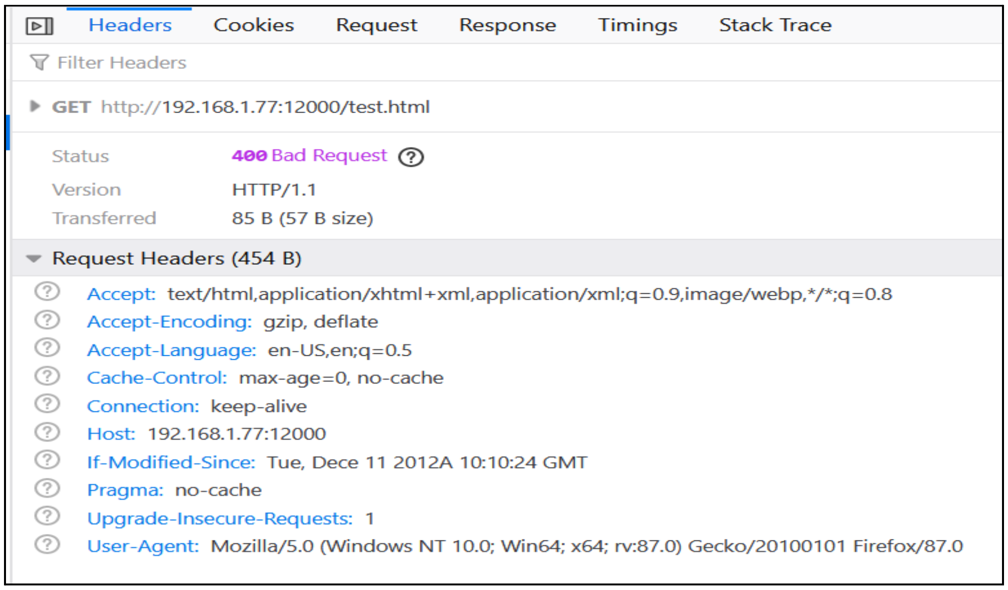
Request Headers

```
Host: 192.168.1.77:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
If-Modified-Since: Tue, Dece 11 2012A 10:10:24 GMT
```

Request Body

Fig 5.2.1: Sending Conditional GET with error in date

As there is an error in the 'If-Modified-Since' date value then the request is considered as a bad request and the below screenshot shows the server response to this request.


Headers Cookies Request Response Timings Stack Trace

Filter Headers

GET http://192.168.1.77:12000/test.html

```
Status        400 Bad Request
Version       HTTP/1.1
Transferred   85 B (57 B size)
```

Request Headers (454 B)

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Cache-Control: max-age=0, no-cache
Connection: keep-alive
Host: 192.168.1.77:12000
If-Modified-Since: Tue, Dece 11 2012A 10:10:24 GMT
Pragma: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0
```

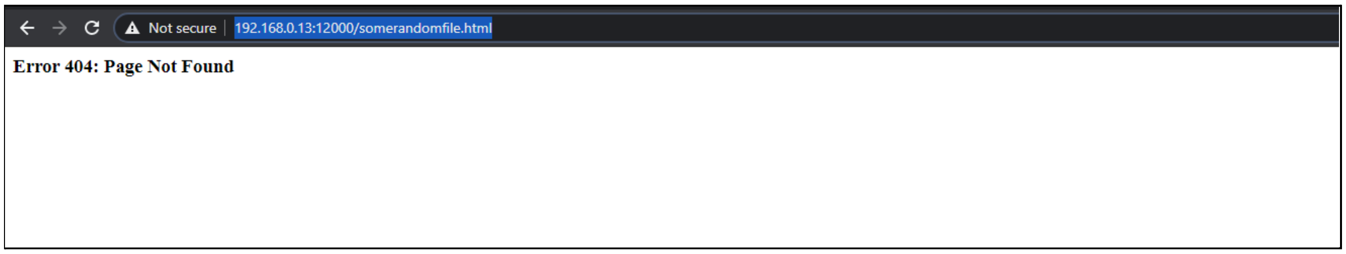
Fig 5.2.2: Server Response Header to Conditional GET request with errors

6. Status Code 404

The status code 404 is used when the requested file by the client does not exist on the server.

6.1. Testing Status Code 404

The request was sended to the server using the Morilla browser. Wrong file name entered 'somerandomfile.html' into the request to check the response of the server. The below results show the response from the server.

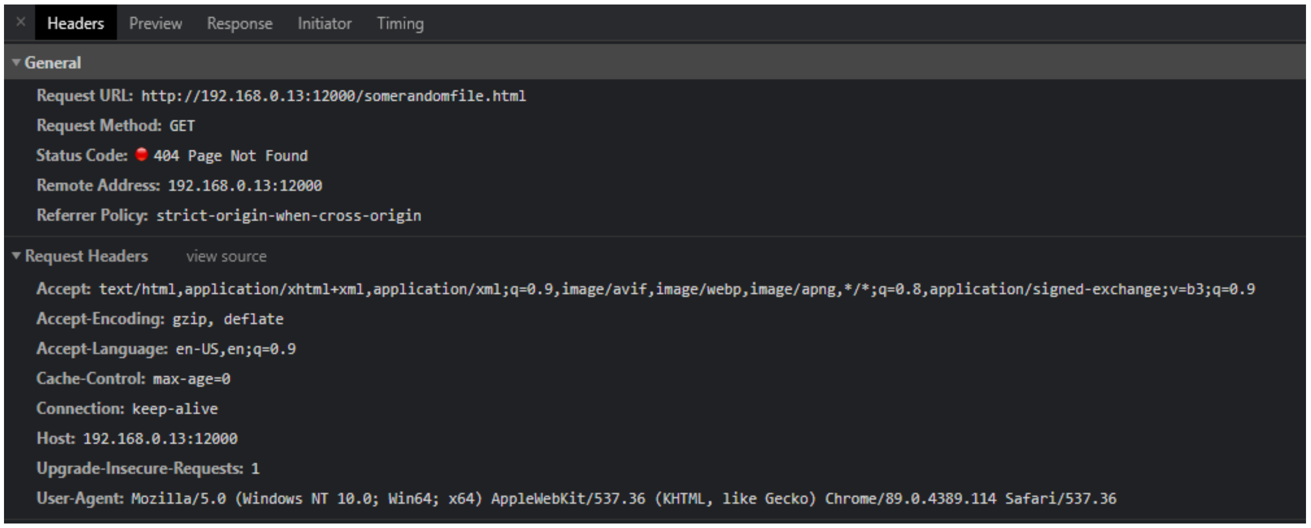


```
Not secure | 192.168.0.13:12000/somerandomfile.html
```

Error 404: Page Not Found

Fig 6.1.1: Sending Server Request with wrong file

The server responded with 404 Page Not Found status code in the header and the HTML page stating that the '404: Page Not Found'.



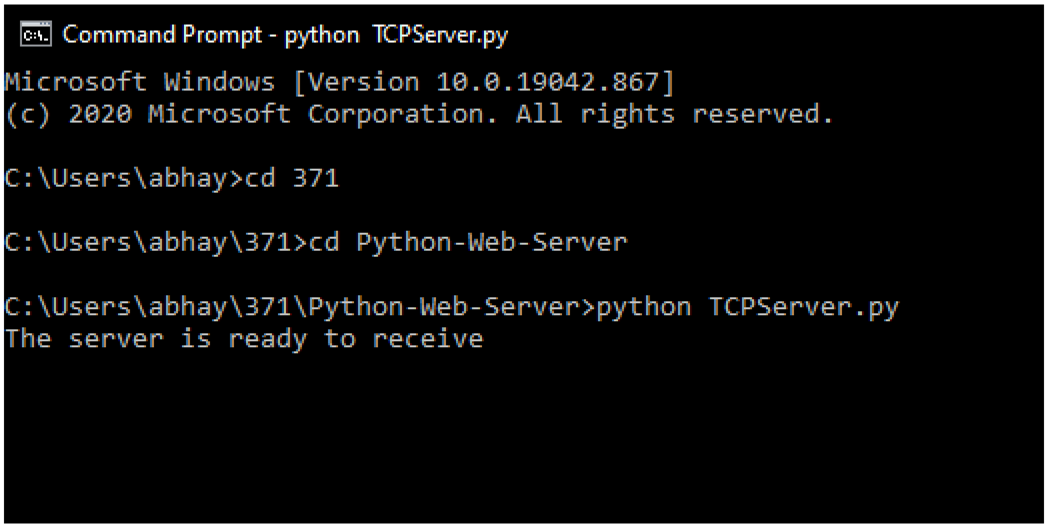
```
General
  Request URL: http://192.168.0.13:12000/somerandomfile.html
  Request Method: GET
  Status Code: 404 Page Not Found
  Remote Address: 192.168.0.13:12000
  Referrer Policy: strict-origin-when-cross-origin
Request Headers
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
  Accept-Encoding: gzip, deflate
  Accept-Language: en-US,en;q=0.9
  Cache-Control: max-age=0
  Connection: keep-alive
  Host: 192.168.0.13:12000
  Upgrade-Insecure-Requests: 1
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.114 Safari/537.36
```

Fig 6.1.2: Server Response to the request with wrong file

7. Status Code 408

7.1. Testing Status Code 408

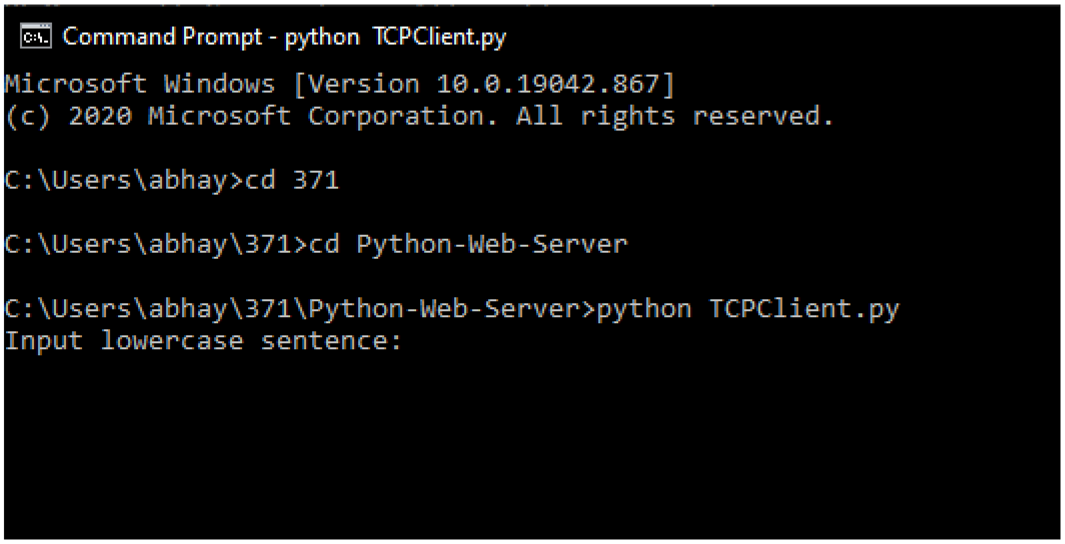
First we run our server and when it is ready to receive we will run our client



```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\abhay>cd 371

C:\Users\abhay\371>cd Python-Web-Server

C:\Users\abhay\371\Python-Web-Server>python TCPServer.py
The server is ready to receive
```

Fig 7.1.1: Server is ready to receive

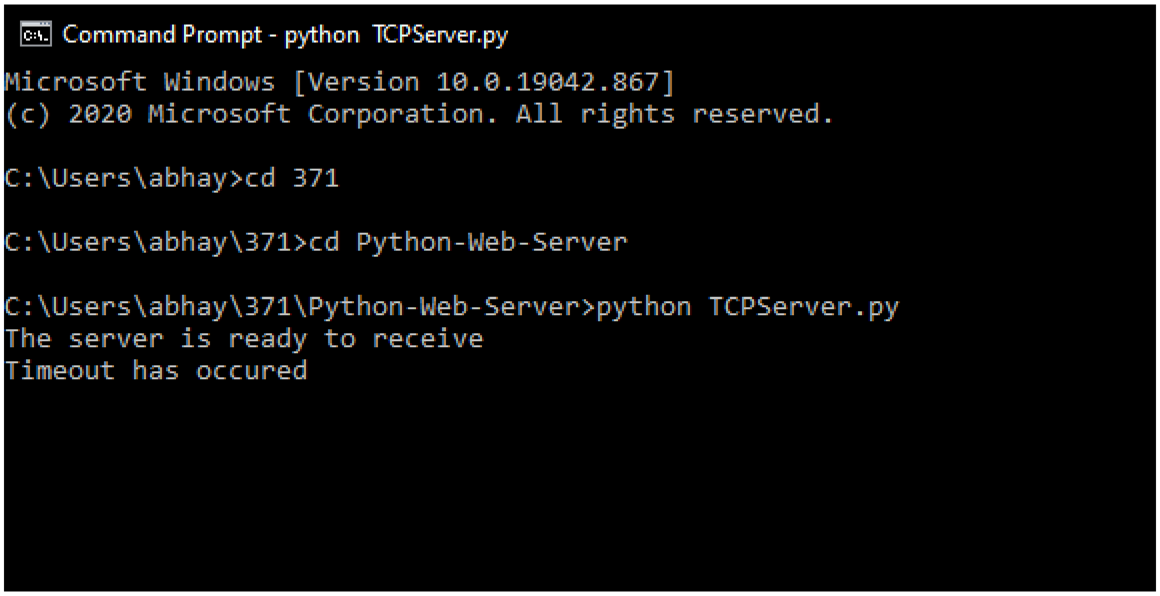
Below is our client file where we use the one provided in class, but with a small change. The client will not send any request to the server. This will cause the timeout and to test we made a print statement in the server which runs in case of a timeout.



```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\abhay>cd 371

C:\Users\abhay\371>cd Python-Web-Server

C:\Users\abhay\371\Python-Web-Server>python TCPClient.py
Input lowercase sentence:
```

Fig 7.1.2: Client establishes a connection but requests nothing

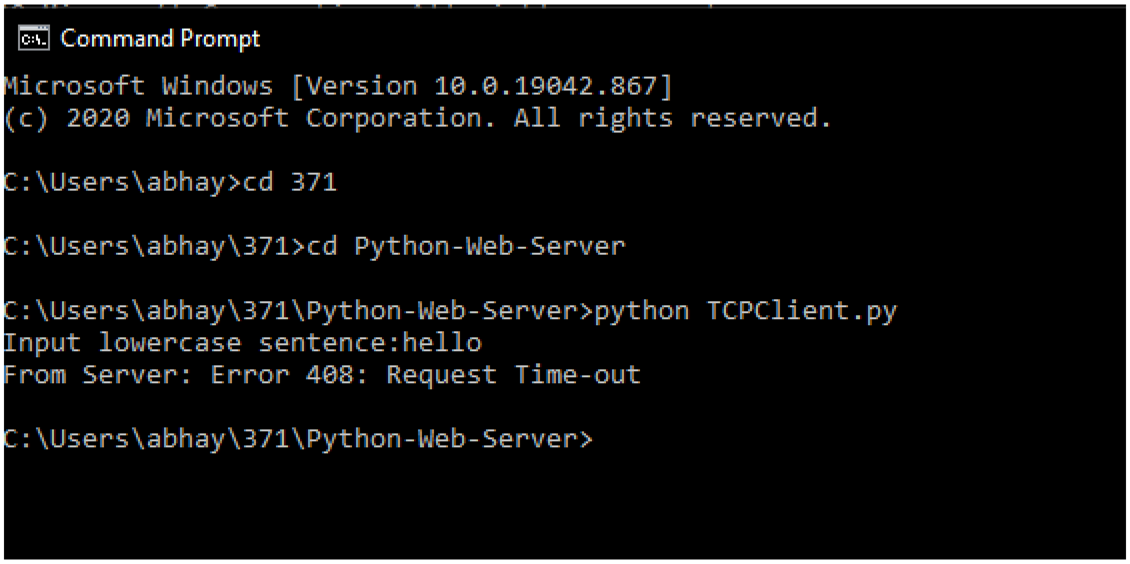
Below the server prints that the timeout has occured. The server will send an Error 408 message to the client and close the connection.



```
Command Prompt - python  TCPServer.py
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\abhay>cd 371

C:\Users\abhay\371>cd Python-Web-Server

C:\Users\abhay\371\Python-Web-Server>python TCPServer.py
The server is ready to receive
Timeout has occured
```

Fig 7.1.3: Timeout occurs and the server sends Error 408

Here as you can see the client has received the message. Our final implementation works for the web browser and the below used client file is only used for testing and is commented out if needed.



```
Command Prompt
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\abhay>cd 371

C:\Users\abhay\371>cd Python-Web-Server

C:\Users\abhay\371\Python-Web-Server>python TCPClient.py
Input lowercase sentence:hello
From Server: Error 408: Request Time-out

C:\Users\abhay\371\Python-Web-Server>
```

Fig 7.1.4: The client gets the error message

8. Multithreaded Web Server

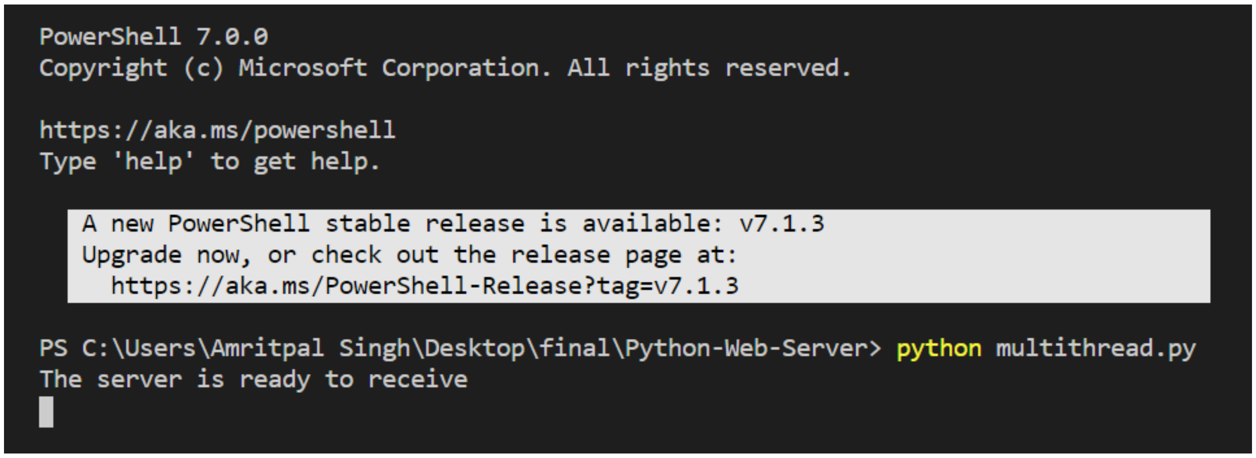
The new modified web server implemented is multi-threaded which means that the server is able to accept multiple requests at the same time. This means that if two clients send a request at the same time to the server then the server will create a thread for each connection and will be able to handle them parallel.

9. Testing Multithreading

To test the multi threading our team came up with the scenario below.

To check that the server is creating a new thread to handle connection, the first thing we have to do is to comment out the 108 line connectionSocket.close() and 21 line connectionSocket.settimeout(5) in the multithread python file. This will help us in making the connection forever. This will never close the thread and connection for the clients. Now we use our client file (in the same directory) and run that file in three different terminals.

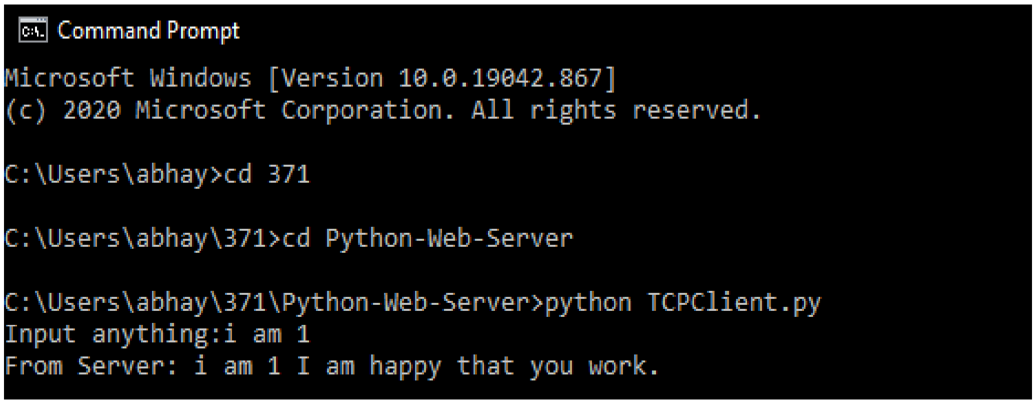
Now first we start our server so that clients can send a request to the server.



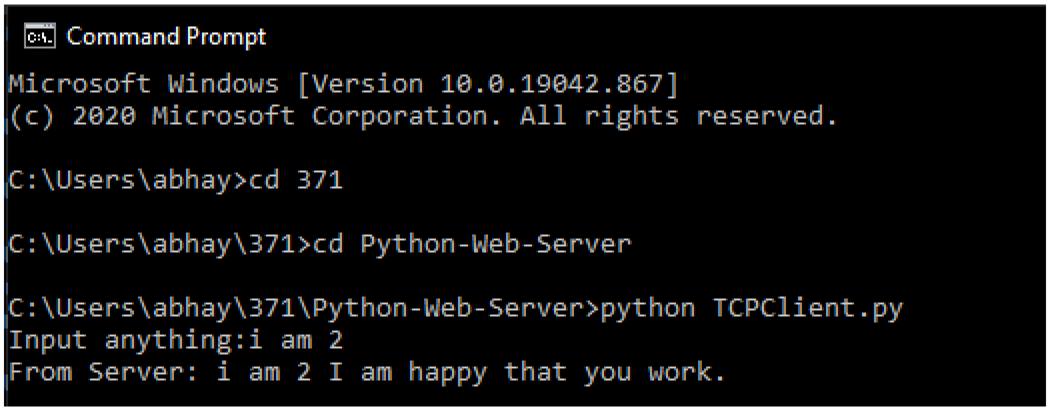
```
PowerShell 7.0.0
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/powershell
Type 'help' to get help.

   A new PowerShell stable release is available: v7.1.3
   Upgrade now, or check out the release page at:
     https://aka.ms/PowerShell-Release?tag=v7.1.3

PS C:\Users\Amritpal Singh\Desktop\final\Python-Web-Server> python multithread.py
The server is ready to receive
```

Fig 9.1: Starting Multithreaded Server

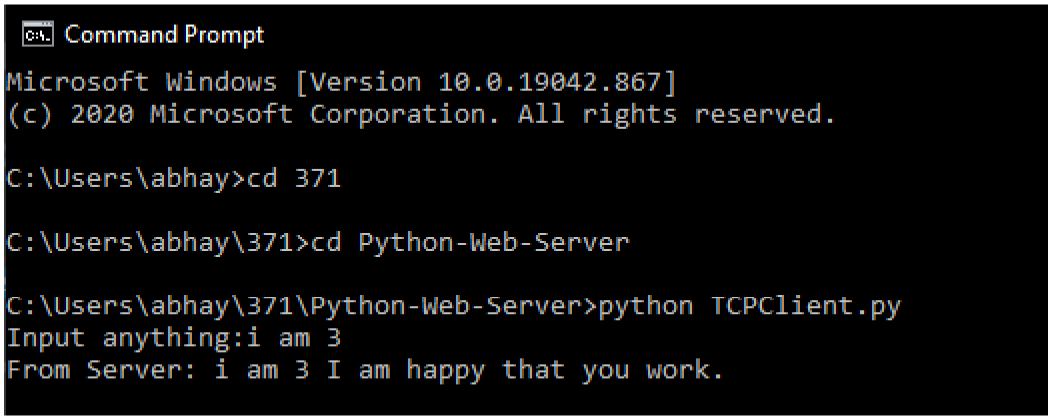
Now we run the clients file on different terminals.



```
Command Prompt
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\abhay>cd 371

C:\Users\abhay\371>cd Python-Web-Server

C:\Users\abhay\371\Python-Web-Server>python TCPClient.py
Input anything:i am 1
From Server: i am 1 I am happy that you work.
```



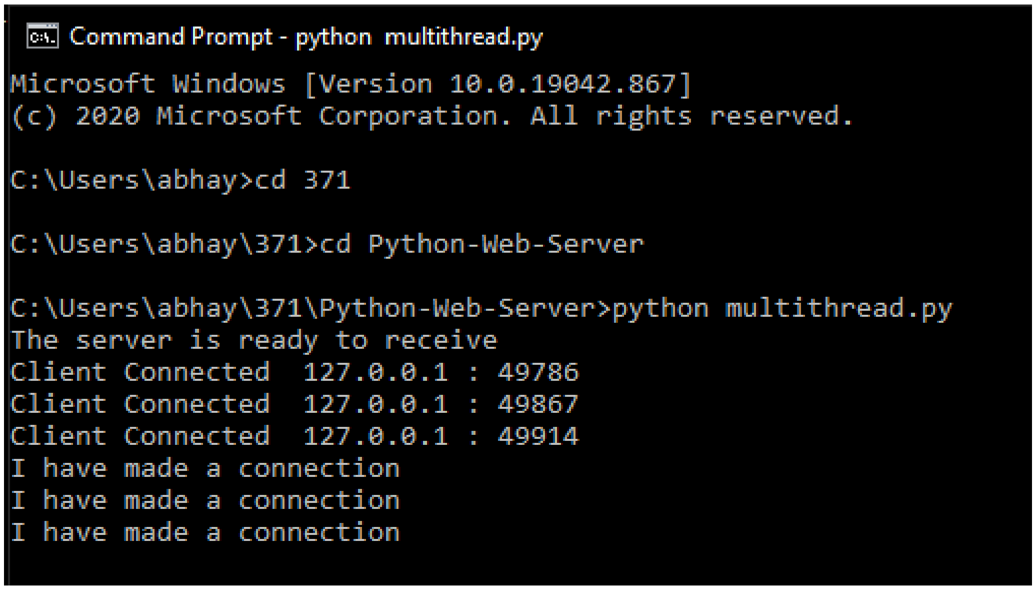
```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\abhay>cd 371

C:\Users\abhay\371>cd Python-Web-Server

C:\Users\abhay\371\Python-Web-Server>python TCPClient.py
Input anything:i am 2
From Server: i am 2 I am happy that you work.
```



```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\abhay>cd 371

C:\Users\abhay\371>cd Python-Web-Server

C:\Users\abhay\371\Python-Web-Server>python TCPClient.py
Input anything:i am 3
From Server: i am 3 I am happy that you work.
```

Fig: 9.2: Setting Up Clients

After the request was sent by each client to the server and we know that when the connection is set up it can not be closed as we have commented on the connection close code. The screenshot below prints the name of the host the server is connected to.



```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\abhay>cd 371

C:\Users\abhay\371>cd Python-Web-Server

C:\Users\abhay\371\Python-Web-Server>python multithread.py
The server is ready to receive
Client Connected  127.0.0.1 : 49786
Client Connected  127.0.0.1 : 49867
Client Connected  127.0.0.1 : 49914
I have made a connection
I have made a connection
I have made a connection
```

Fig: 9.3: Connected Host Address

As we can see in the screenshots that the “i am 1/2/3” messages were sent from client to server and then the server sends the response to all the three clients according to their messages. All the three clients received the message. This shows the server is creating multiple threads for handling connections and each of these threads are running in parallel while sending response to the clients.

10. References

- Request For Comments, https://tools.ietf.org/html/rfc7231
- Python library for sockets - https://docs.python.org/3/library/socket.html
- Python library for threading - https://docs.python.org/3/library/_thread.html
- If-Modified-Since-
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/If-Modified-Since