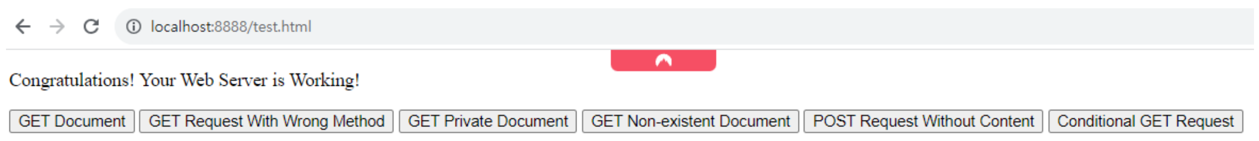
CMPT 371 Mini Project Fall 2023

Jusung Park 301415852
Seunghwan Kim 30136786

Start the web sever with web.py `python3 web.py`
Start the proxy server with proxy.py with the port number of the web server `python3 proxy.py 8080`

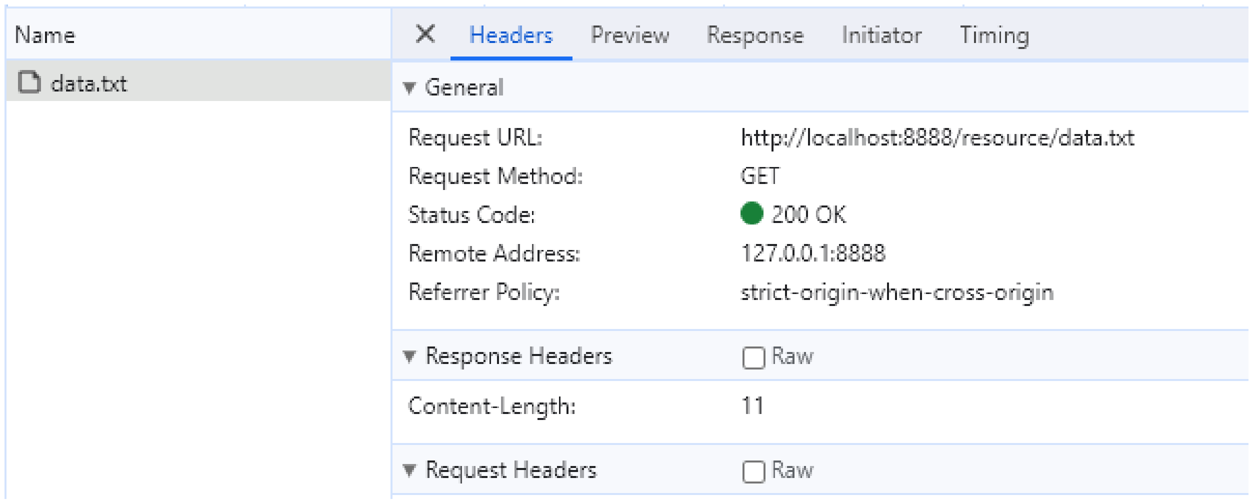


localhost:8888/test.html

Congratulations! Your Web Server is Working!

GET Document
GET Request With Wrong Method
GET Private Document
GET Non-existent Document
POST Request Without Content
Conditional GET Request

1) GET Document (200 OK)



data.txt

General

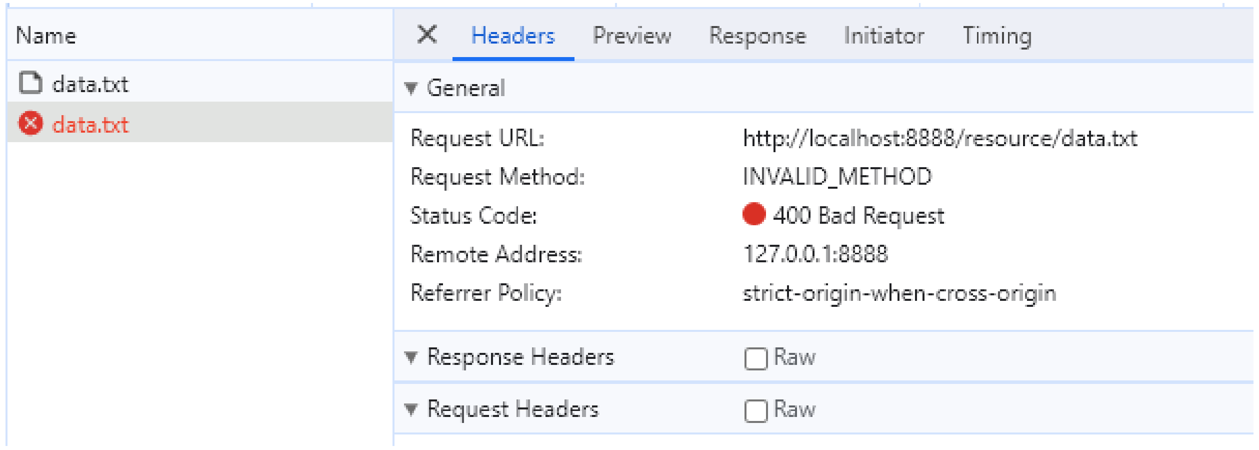
Request URL: http://localhost:8888/resource/data.txt
Request Method: GET
Status Code: ● 200 OK
Remote Address: 127.0.0.1:8888
Referrer Policy: strict-origin-when-cross-origin

Response Headers ☐ Raw

Content-Length: 11

Request Headers ☐ Raw

2) GET Request with Wrong Method(400 Bad Request)



data.txt
data.txt

General

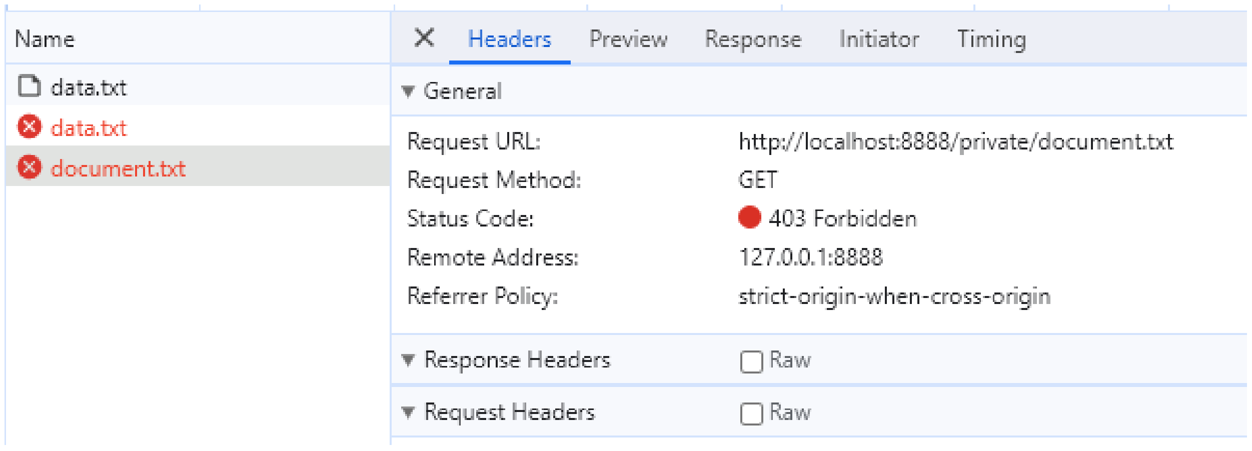
Request URL: http://localhost:8888/resource/data.txt
Request Method: INVALID_METHOD
Status Code: ● 400 Bad Request
Remote Address: 127.0.0.1:8888
Referrer Policy: strict-origin-when-cross-origin

Response Headers ☐ Raw

Request Headers ☐ Raw

Check whether the method of HTTP request is included in [GET, HEAD, POST, DELETE]. Response 400 Bad Request when it's not.

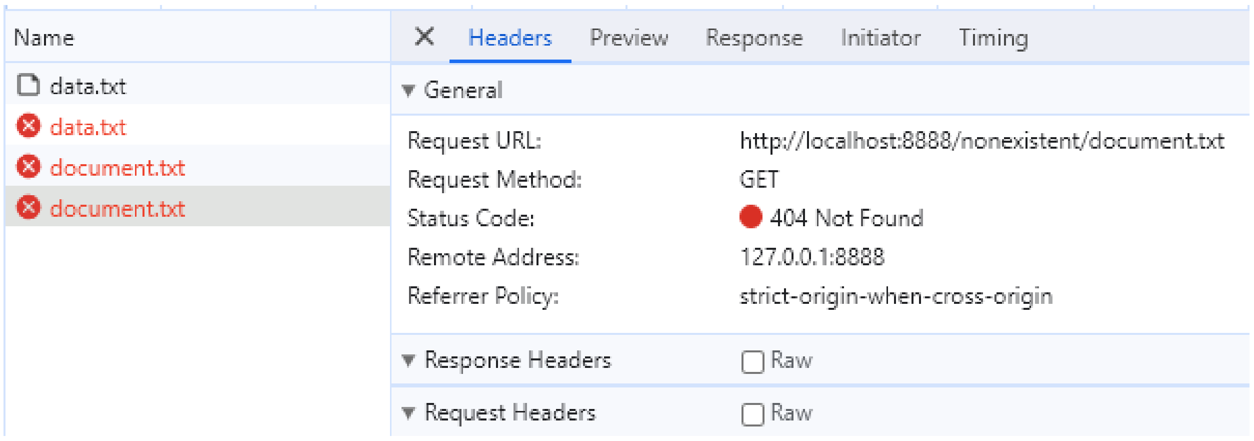
3) GET Private Document (403 Forbidden)



Request URL: http://localhost:8888/private/document.txt
Request Method: GET
Status Code: ● 403 Forbidden
Remote Address: 127.0.0.1:8888
Referrer Policy: strict-origin-when-cross-origin

We add '/private' directory in our project folder. When the client tries to access the resource inside of this directory, the server will response 403 Forbidden error since the client don't have the authority. (There is no specific authentication checking function)

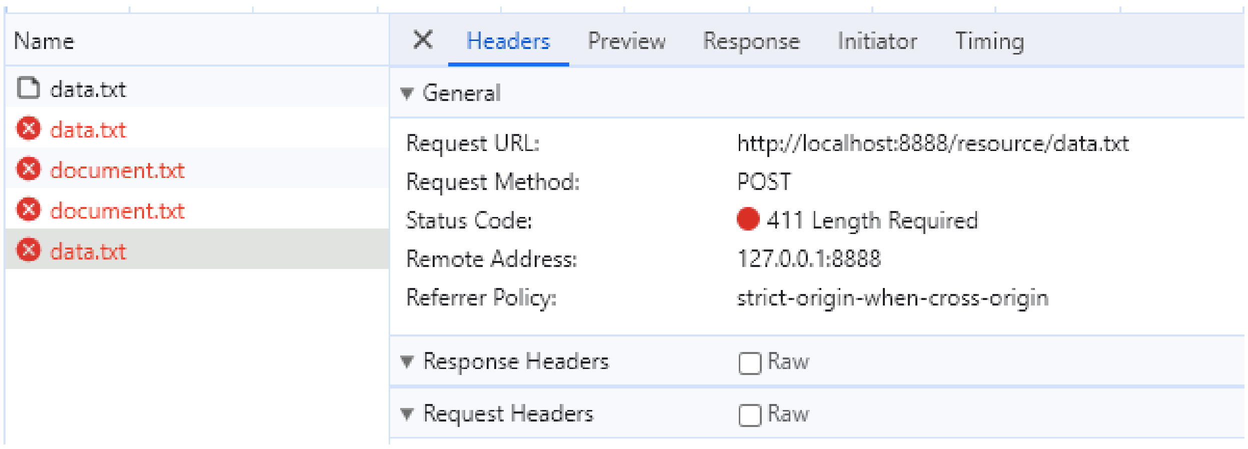
4) GET Non-existent Document (404 Not Found)



Request URL: http://localhost:8888/nonexistent/document.txt
Request Method: GET
Status Code: ● 404 Not Found
Remote Address: 127.0.0.1:8888
Referrer Policy: strict-origin-when-cross-origin

When the user requests the non-existent directory, the server responds 404 Not Found error.

5) POST Request Without Content (411 Length Required)

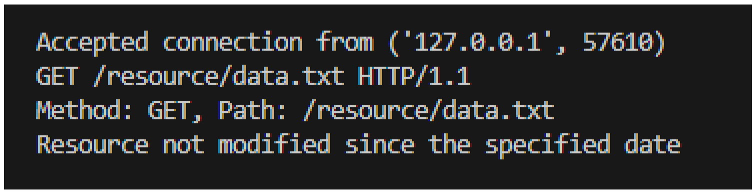


data.txt
data.txt
document.txt
document.txt
data.txt

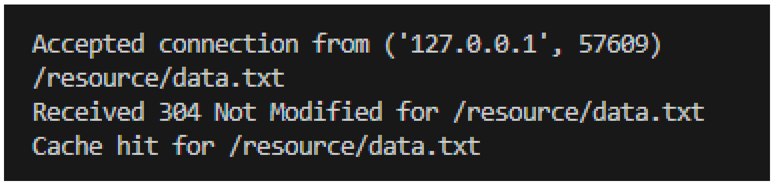
Request URL: http://localhost:8888/resource/data.txt
Request Method: POST
Status Code: ● 411 Length Required
Remote Address: 127.0.0.1:8888
Referrer Policy: strict-origin-when-cross-origin

POST is the most well-known method which requires ‘Content-Length’ header. In our code, we send a POST request with 0 as a content-length value and interpret it as the non-existence of a content-length header.

6) Conditional GET Request (304 Not Modified)

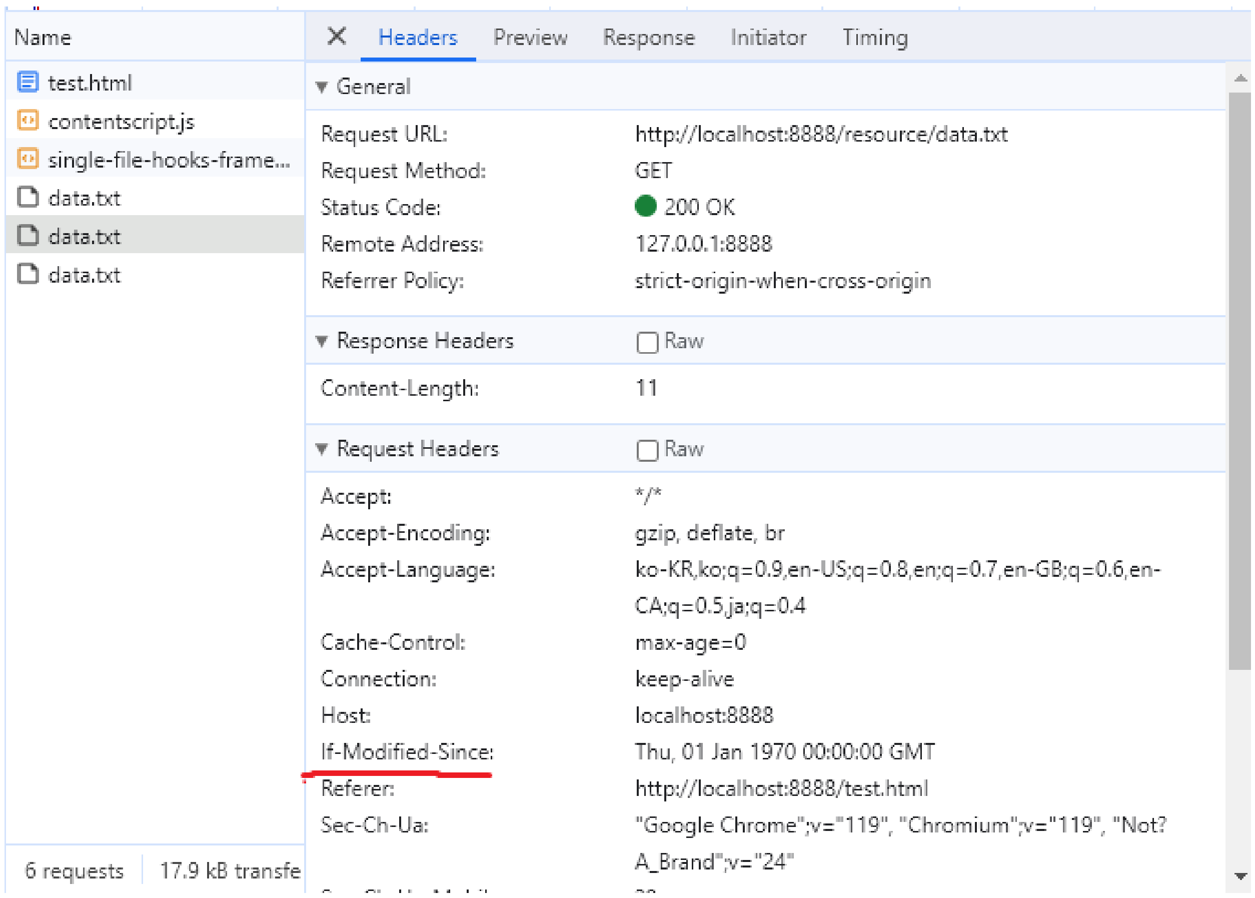


```
Accepted connection from ('127.0.0.1', 57610)
GET /resource/data.txt HTTP/1.1
Method: GET, Path: /resource/data.txt
Resource not modified since the specified date
```



```
Accepted connection from ('127.0.0.1', 57609)
/resource/data.txt
Received 304 Not Modified for /resource/data.txt
Cache hit for /resource/data.txt
```

data up-to-date in cache



```
Name                       ✕  Headers   Preview   Response   Initiator   Timing
test.html                  ▼ General
contentscript.js           Request URL:          http://localhost:8888/resource/data.txt
single-file-hooks-frame... Request Method:       GET
data.txt                   Status Code:          ● 200 OK
data.txt                   Remote Address:       127.0.0.1:8888
data.txt                   Referrer Policy:      strict-origin-when-cross-origin

                           ▼ Response Headers    ☐ Raw
                           Content-Length:       11

                           ▼ Request Headers     ☐ Raw
                           Accept:               */*
                           Accept-Encoding:      gzip, deflate, br
                           Accept-Language:      ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7,en-GB;q=0.6,en-
                                                 CA;q=0.5,ja;q=0.4
                           Cache-Control:        max-age=0
                           Connection:           keep-alive
                           Host:                 localhost:8888
                           If-Modified-Since:    Thu, 01 Jan 1970 00:00:00 GMT
                           Referer:              http://localhost:8888/test.html
                           Sec-Ch-Ua:            "Google Chrome";v="119", "Chromium";v="119", "Not?
                                                 A_Brand";v="24"
6 requests | 17.9 kB transfe
```

Successfully returns the data to the client
Web server -> 304 -> Proxy -> 200 -> Client

In our code, we sent the conditional GET request with a static ‘if-modified-since’ value (1970-01-01). The proxy server always shows 200: OK to the client while the communication between the proxy and the original web server varies due to the ‘Last-modified’ header value.

What is different in request handling in a proxy server and a web server that hosts your files?

In the proxy server, the request-handling process involves interpreting the incoming request to determine whether the response is available in the cache. If the response is cached, it is sent directly to the client. If not, the request is forwarded to the web server. Subsequently, the proxy receives the response from the web server and forwards it to the client.

On the web server side, the request-handling process includes interpreting the request received from the proxy. The web server generates an appropriate response based on the request, which may include a 304 status after checking the "If-Modified-Since" header, a 200 status for successful requests, a 404 status for nonexistent resources, and so on. The generated response is then sent back to the proxy for further processing.

Write down the detailed specifications you come up with for a minimal proxy server only using the knowledge you have from module (2) slides 29-34r files.

Rather than establishing a direct connection to the web server from the browser, we route connections through the proxy server. The browser sends requests to the proxy, which, in turn, checks its cache for the requested file. If the file is found and is up-to-date, the proxy server retrieves it directly from the cache. In the case of an outdated file, the proxy forwards the request to the web server and updates its cache with the latest data.

When a client requests access to a file, the proxy server first checks its cache. If the cached file is current, the web server responds with a 304 status, indicating that the cached version is valid. In this scenario, the proxy efficiently delivers the cached file to the client with a 200 OK response. All other requests and responses, apart from the 304 response, are seamlessly processed through the proxy server.

Utilizing stored data in the proxy cache can lead to time savings, especially for larger files or when the web server is located at a considerable distance.

Decide the test procedure to show the working of your proxy server. Does this need possible changes at the client side?

At the client side, the browser needs to be configured to use the proxy server by setting the browser’s network settings. The client needs to use the URL navigating to the proxy server.
Use the network tools to inspect the HTTP requests and response including the “If-Modified-Since” header.
Check the cache dictionary after making a request through the proxy to see if the response is being cached.

Does your server have an HOL problem?

No, our proxy server effectively addresses the HOL (Head-of-Line) problem by implementing threading using the threading module. This allows the server to handle socket connections concurrently, ensuring that multiple connections can be processed simultaneously without encountering HOL issues.