

CMPT 371 - Mini Project

William Ting - 301177870

Barisukara - 301353211

Dec 5, 2023

Step 2

- a) Files in Webserver.py, proxy_server.py, and test.html
- b) Message Codes found when you run python3 -m Webserver.py
- c) Use <http://localhost:8000/test.html> to test / upon getting to this link, you must enter 'password' in the console when prompted for a password to check the below message codes.

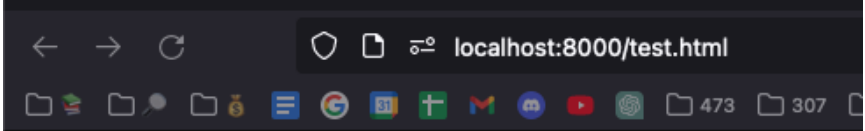
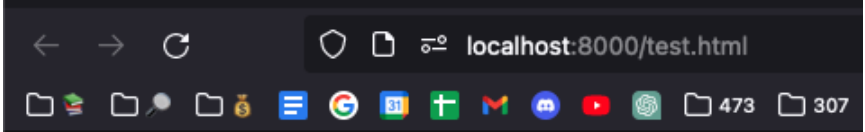
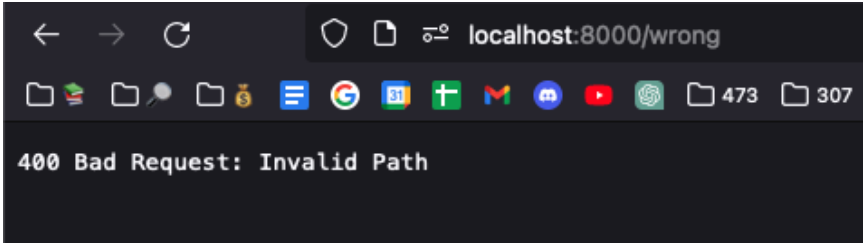
Important: Please note, for our webserver.py code our server uses filename == '/test.html' as the mock endpoint to process data. As such the server works when /test.html is being processed by the server. This applies to the webpage that is open with /test.html and also to the way we tested our GET/POST requests which had the sender endpoint both set as /test.html for convenience.

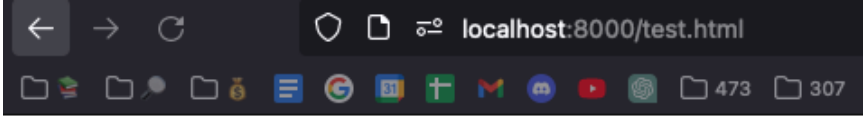
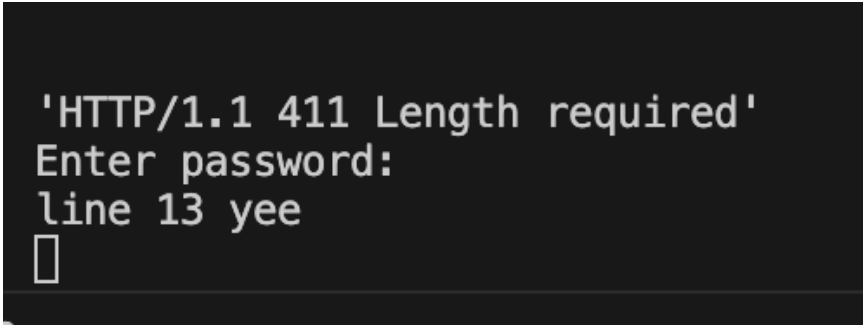
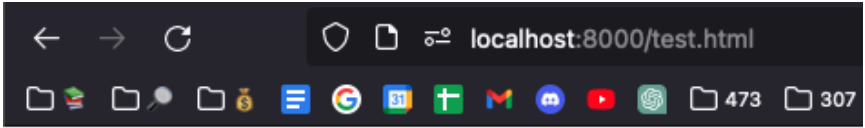
As such, you will be able to see the message codes under situations where /test.html is passed or not passed and message codes 200/304/411 are dependent on the endpoint being /test.html for testing. This was intentionally hardcoded this way as instructions gave us test.html as the source file for testing.

We used fetch API and sent HTTP requests from the test.html website using this API with /test.html as the sender endpoint. We did this because requirements for how to generate HTTP requests were not explicitly given. Doing the HTTP request from the client side web page seemed workable.

Also you may see this favicon.co pop up in the console. Please ignore that - I believe it is a firefox or browser thing that is not supposed to be there. Ignore it for testing as it may trigger message codes unintentionally.

Code	Message	How to Test / Results
200	OK	Cd into cmpt_miniproject / run python3 -m Webserver.py / go to http://localhost:8000/test.html / 200 message will show Output:

		 <p>HTTP/1.0 200 OK</p> <p>Congratulations! Your Web Server is Working!</p>
304	Not Modified	<p>Cd into cmpt_miniproject / run python3 -m Webserver.py / go to http://localhost:8000/test.html / 304 message will show only after the user has visited /test.html once before - i.e. act as a web cache. This can be generated if you refresh /test.html again. The server will flag that you've been here before and load the 304 message. Please note it uses a bool flag to do this so the content is technically re-displayed for our purposes but the idea behind the implementation is to mock a web server cache.</p> <p>Output:</p>  <p>HTTP/1.0 304 Not Modified</p> <p>Congratulations! Your Web Server is Working!</p>
400	Bad request	<p>Cd into cmpt_miniproject / run python3 -m Webserver.py / go to http://localhost:8000/wrong / 400 message will show if the entered url is not /test.html</p> <p>Output:</p>  <p>400 Bad Request: Invalid Path</p>
403	Forbidden	<p>Cd into cmpt_miniproject / run python3 -m Webserver.py / go to http://localhost:8000/test.html , the console will prompt the user to enter a password. If 'password' is not entered, the user will get a screen that shows the 403 forbidden message. The console will</p>

		<p>show the wrong password entered. If you wish to try again. Please reload /test.html to refresh the server.</p> <p>If 'password' is entered, it will load up /test.html and you can see the webpage.</p> <p>Output:</p>  <p>403 Forbidden - Invalid Password</p> 
404	Not Found	<p>Cd into cmpt_miniproject / run python3 -m Webserver.py / go to http://localhost:8000/test.html / 404 message will show if the test.html file is removed from the cmpt_miniproject directory</p> <p>Output:</p>  <p>404 File Not Found</p>
411	Length required	<p>Cd into cmpt_miniproject / run python3 -m Webserver.py / go to http://localhost:8000/test.html / 411 message will show if the content header length is not provided by a specific HTTP request on the server side</p> <p>Output if content length is provided:</p> <p>Can be tested by going to the /test.html webpage and clicking the 'send POST request' button. This will send a post request that will</p>

		<p>generate an output confirming the presence of content-length in the console.</p> <pre> line 63 POST /test.html HTTP/1.1 Host: localhost:8000 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:121.0) Gecko/20100101 Firefox/121.0 Accept: */* Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate, br Referer: http://localhost:8000/test.html Content-Type: application/json Content-Length: 33 Origin: http://localhost:8000 Connection: keep-alive Sec-Fetch-Dest: empty Sec-Fetch-Mode: cors Sec-Fetch-Site: same-origin {"message":"Hello, server! POST"} line 78, Content-Length header present: 33 bytes line 32 /test.html line 80 HTTP/1.1 304 Not Modified Cache-Control: no-cache Content-Length: 0 Connection: close </pre> <p>Output if content length is NOT provided:</p> <p>If you refresh /test.html or send a get request by clicking the 'send GET request' button. This will send a get request that has no content length in the header and as such the console will output a message that says "HTTP/1.1 411 Length required"</p> <pre> line 63 GET /test.html HTTP/1.1 Host: localhost:8000 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:121.0) Gecko/20100101 Firefox/121.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate, br Connection: keep-alive Upgrade-Insecure-Requests: 1 Sec-Fetch-Dest: document Sec-Fetch-Mode: navigate Sec-Fetch-Site: none Sec-Fetch-User: ?1 line 87 'HTTP/1.1 411 Length required' line 32 /test.html line 80 HTTP/1.1 304 Not Modified Cache-Control: no-cache Content-Length: 0 Connection: close </pre>
--	--	---

Step 3

a)

Implementation is in proxy_server.py

Web Server: When a client requests a web page or a resource, the web server processes the request and delivers the requested files directly to the client, whilst,

Proxy Server: When a client sends a request for a web page or resource, the proxy server intercepts the request and forwards it to the destination web server on behalf of the client. The proxy server's response is then sent back to the web server which then forwards it back to the client.

Specifications for proxy server

- Listens on port 8080 for client request
- Processes the client request and returns the response back to destination server

b)

Test procedure:

to test the proxy server

1. open 2 terminal windows
2. run `python3 -m proxy_server.py` on terminal 1
3. run `python3 -m Webserver.py` on terminal 2
4. then go to `http://localhost:8080/test.html` , terminal 2 will ask for password , which shows a successful redirection from proxy server to web server

What happens behind the scenes is the client request (loading the contents of test.html) goes to the web server then it forward the request to the proxy server which processes the request using its socket server and returns a response to the web server, the web server takes this response and sends it back to the client

For our purposes, the client side does not need changes as the client does not know about what server nor does the client care about what server is handling the request. Given that it is a static html the content delivery is done on the server side the web server forwards the content for the proxy server to process without client side interaction.

In other circumstances, the client side would need adjustments if the proxy server changes the client side request or needs to send information back to the client side for something such as 'authentication' or if the port / address has changed.

Output:

```
cmpt_miniproject — python3 proxy_server.py — 80x24
Last login: Mon Dec  4 00:22:01 on ttys021
chruby: unknown Ruby: ruby-3.1.1
chruby: unknown Ruby: ruby-3.1.1
[(base) barisukara@d207-023-186-155 ~ % cd CMPT371
[(base) barisukara@d207-023-186-155 CMPT371 % cd cmpt_miniproject
(base) barisukara@d207-023-186-155 cmpt_miniproject % python3 proxy_server.py

Proxy server listening on port 8080 ...

cmpt_miniproject — python3 -m Webserver.py — 80x24
Last login: Mon Dec  4 16:44:18 on ttys016
chruby: unknown Ruby: ruby-3.1.1
chruby: unknown Ruby: ruby-3.1.1
[(base) barisukara@d207-023-186-155 ~ % cd CMPT371
[(base) barisukara@d207-023-186-155 CMPT371 % cd cmpt_miniproject
[(base) barisukara@d207-023-186-155 cmpt_miniproject % python3 -m Webserver.py
Listening on port 8000 ...
GET /test.html HTTP/1.1
```

Step 4

Does your server have an HOL problem?:

No, technically our Webserver.py does not have a HOL problem, the requests are ordered from LIFO so in a HOL problem they are stuck in a queue to be processed while the always ON loop processes each request.

However, for our design and testing, the requests are sent via a 'button' or 'refresh' so the queue processes a single request at a time leading to technically no backup as the while loop processes each request immediately.

We implemented condition checks that ensure that when a single request is sent over, if the password isn't right or the path isn't right (as two examples) - the server does not keep trying to process but sends a prompt with a message code right away and breaks out of the program. This leads to no backup from our server.

If we had to handle multiple requests from many different clients, then they would be put in a queue that could face HOL problems. However, the processing of the server would be quick and none of the requests that we tested on were dependent on other data or events or dependencies so timing and asynchronous wait times were not factors for us.

So hard to say if HOL would occur for us still because it would likely be the case that HOL would occur if the requests to be processed exceed a boundary input size that we don't know what that would be and if the requests had behaviours that are more complex (like sending to a database and waiting for a response back from a database).

Our answer is thus subjective and dependent on how many and what types of requests and events occurred to our server. In one case, no HOL, in a case with complex, multiple requests, maybe HOL.