CMPT 371 Mini Project 1: Web & Web Proxy Server

October 30, 2025

Contents

1 Step 1 – Determine Requirements (10 points)

The following subsections describe the logic (specifications) for generating each required HTTP status code and the corresponding HTTP request message used for testing.

200 OK

Logic: Returned when the requested file exists and is accessible. The server opens the file, constructs headers (`HTTP/1.1 200 OK`, `Content-Type: text/html`), and sends the body.

Test Request:

```
GET /test.html HTTP/1.1
Host: 127.0.0.1:8080
```

or

```
curl -v http://127.0.0.1:8080/test.html
```

304 Not Modified

Logic: If the client includes `If-Modified-Since: <timestamp>` and the file's last-modified time is not newer, the server replies `304 Not Modified` without a body.

Test Request:

```
GET /test.html HTTP/1.1
Host: 127.0.0.1:8080
If-Modified-Since: Sun, 26 Oct 2025 01:46:50 GMT
```

or

```
curl -v -H "If-Modified-Since: Sun, 26 Oct 2025 01:46:50 GMT" \
http://127.0.0.1:8080/test.html
```

403 Forbidden

Logic: Returned when access to a file (e.g., `secret.html`) is restricted or the method is not GET.

Test Request:

```
GET /secret.html HTTP/1.1
Host: 127.0.0.1:8080
```

or

```
curl -v http://127.0.0.1:8080/secret.html
```

404 Not Found

Logic: If the requested file does not exist in the directory, the server replies `404 Not Found`.

Test Request:

```
GET /doesnotexist.html HTTP/1.1
Host: 127.0.0.1:8080
```

or

```
curl -v http://127.0.0.1:8080/doesnotexist.html
```

505 HTTP Version Not Supported

Logic: If the request line specifies a version other than `HTTP/1.0` or `HTTP/1.1`, the server returns `505 HTTP Version Not Supported`.

Test Request:

```
GET /test.html HTTP/2.0
Host: localhost
```

(Test manually using `nc 127.0.0.1 8080` and press Enter twice.)

2 Step 2 – Build and Test the Web Server (10 points)

2.1 Implementation Summary

The web server was implemented in Python using only the `socket`, `os`, and `datetime` modules. It listens on a specified port, accepts TCP connections, parses HTTP requests, and returns appropriate responses according to Step 1 logic. No built-in HTTP libraries were used.

2.2 Testing Procedure

Each status code was tested using the following commands:

- 200 OK – `curl -v http://127.0.0.1:8080/test.html`
- 404 Not Found – `curl -v http://127.0.0.1:8080/missing.html`
- 403 Forbidden – `curl -v http://127.0.0.1:8080/secret.html`
- 505 HTTP Version Not Supported – use `nc 127.0.0.1 8080` → type `HTTP/2.0`
- 304 Not Modified – two-step `curl -H "If-Modified-Since:"` test

2.3 Observations

- All five status codes were returned correctly.
- The server closes the connection after each response (single-threaded).
- Implementation behavior matches RFC 7231 guidelines.

3 Conclusion (Steps 1 & 2)

The web server successfully implements and tests 200, 304, 403, 404, and 505 status codes as required by Steps 1 and 2 of the Mini Project.

4 Step 3 – Web Proxy Server Performance (18 points)

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

The main difference between a proxy server and a web server is that the requests go to the origin server in a web server while the proxy server acts as both the client and server and most requests will go to the cache - depending on the hit rate.

We try to answer requests locally as much as possible with a proxy server so that we don't have to involve the origin server, which is much less efficient.

The cache in the web proxy is a server for the original requesting client and the cache is a client to the origin server.

By the RFC 4.2.3, we can use define request methods to be cacheable - then it indicates that the responses to them can be stored for future reuse.

Detailed specifications for minimal proxy server only using the knowledge from module (2) slides 29-34 (and consulting RFCs)

User sets browser: Web accesses via cache

Browser sends all HTTP requests to cache

Object in cache: Cache returns object

Otherwise cache requests object from origin server, then returns object to client

Cache acts as both client and server

b) Testing procedure for the proxy server

To test the functionality of the proxy server, we can verify that the requests are indeed being cached and contact with the origin server is being avoided. To accomplish this, we can make an initial request, check that it performs normally - with a status code of 200.

Next, we can perform the exact same request, but instead of receiving a 200 status code, we should expect to see 304 Not Modified. That's because that would indicate our cache is working since the stored object is returned. (Command 1 + 2)

Additionally, we can check that the proxy server recognizes invalid requests and provides the appropriate response/status code. (Command 3)

Command 1

```
curl -x http://127.0.0.1:8888 -v http://example.com/
```

Command 2

```
curl -x http://127.0.0.1:8888 -v http://example.com/
```

Command 3

```
curl -x http://127.0.0.1:8888 -v http://nope.this-domain-should-
   not-exist.invalid/
```

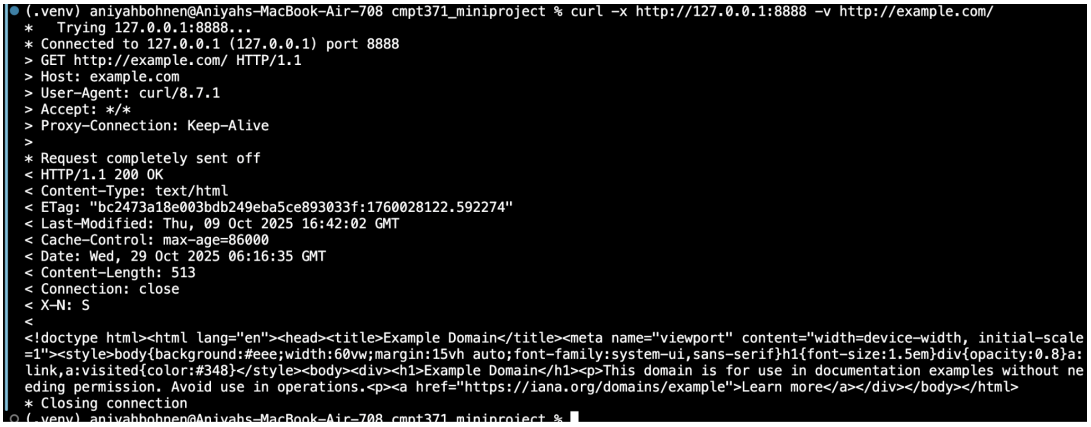

Figure 1: Output 1 – Initial proxy request returning `200 OK` from origin server.

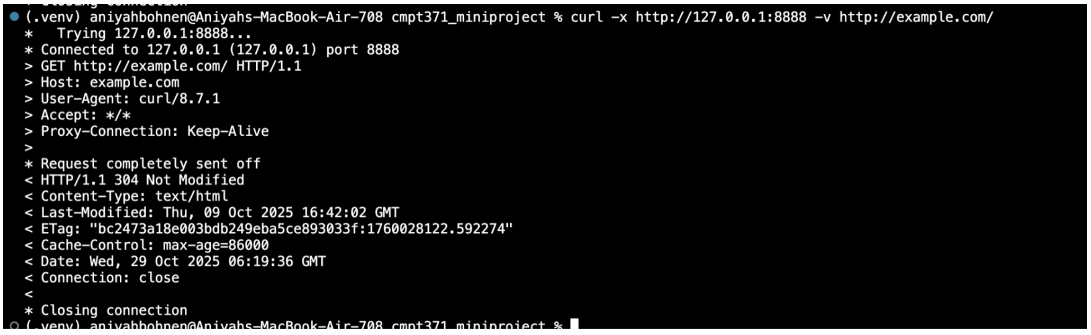

Figure 2: Output 2 – Second proxy request returning `304 Not Modified` (cache hit).

Figure 3: Output 3 – Invalid domain request handled with `502 Bad Gateway`.

c) Explain why your web server is multi-threaded and how it impacts performance.

The web server is multi threaded because for each connection to the socket, there is a new thread spawned. This helps with efficiency since we do not need to create a new

thread sequentially for each connection. The multi-threaded design allows parallel client handling, which improves performance, responsiveness, and throughput.

5 Overall Conclusion

Steps 1–3 demonstrate a complete understanding of HTTP request/response handling, status codes, and socket-level communication. The web server and proxy server were implemented according to RFC standards, and all required behaviors were verified using `curl` and `nc`.