Step One: Determine Requirements

200 OK -

Requirement in Request: Supported HTTP Version, Valid HTTP request for an existing file, Resource pointed in the URI exists.
Request Method: GET
Logic for Generation: The server successfully finds and reads the resource specified in the URI.
Example HTTP Request:
GET /test.html HTTP/1.1
Host: localhost:8000
Connection: close

304 Not Modified -

Requirement in Request: Must include the conditional header “If-Modified-Since”, Resource pointed in the URI exists.
Request Method: GET
Logic for Generation: Request contains the conditional header, if file’s last modified time is <= date given in the header, server responds with 304 Not Modified.
Example HTTP Request:
GET /test.html HTTP/1.1
Host: localhost:8000
If-Modified-Since: Sun, 26 Oct 2025 12:00:00 GMT
Connection: close

403 Forbidden -

Requirement in Request: Resource pointed in the URI exists, Supported HTTP Version, Valid HTTP request for an existing file
Request Method: GET
Logic for Generation: The requested file in the URI exists, but the server has no permission to access it.
Example HTTP Request:
GET /private/test.txt HTTP/1.1
Host: localhost:8000
Connection: close

404 Not Found -

Requirement in Request: Supported HTTP Version, Valid HTTP request, resource pointed to in the URI does not exist.
Request Method: GET
Logic for Generation: Server receives a valid HTTP request but is unable to find the resource in the server.
Example HTTP Request:
GET /nonexistent.html HTTP/1.1
Host: localhost:8000

Connection: close

505 HTTP Version Not Supported -

Requirement in Request: The request uses an unsupported or invalid HTTP version.
Request Method: Any
Logic for Generation: The client sends a request using an HTTP version that is not supported by the server or is invalid.
Example HTTP Request:
GET /test.html HTTP/2.0
Host: localhost:8000
Connection: close

Step 2: Build Your Minimal Web Server & Test

c. Now you need to test your web server for generating the status codes provided in step one. You can use curl, telnet, or your browser for each of these steps. You can also edit your test file, or send the request properly to test your implementation for different message scenarios. Print screen, or cut and paste the output and document your test procedures. (8 points)

200 OK -

Simply curl the file location with the local port to get a response.

```
C:\Users\alfre>curl http://localhost:8000/test.html
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>Congratulations! Your Web Server is Working!</p>

</body>

</html>
```

304 Not Modified -

Add the If Modified Since Header through the command line

```
C:\Users\alfre>curl -v -H "If-Modified-Since: Tue, 29 Oct 2025 00:00:00 GMT" http://localhost:8000/test.html
* Host localhost:8000 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:8000...
*   Trying 127.0.0.1:8000...
* Connected to localhost (127.0.0.1) port 8000
* using HTTP/1.x
> GET /test.html HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/8.14.1
> Accept: */*
> If-Modified-Since: Tue, 29 Oct 2025 00:00:00 GMT
>
< HTTP/1.1 304 Not Modified
< Date: Thu, 30 Oct 2025 00:37:27 GMT
< Server: Webserver
< Content-Type: text/html
< Content-Length: 25
< Connection: close
<
* Excess found writing body: excess = 25, size = 0, maxdownload = 0, bytecount = 0
* we are done reading and this is set to close, stop send
* abort upload
* shutting down connection #0
```

403 Forbidden -

Point the file location to a private directory

```
C:\Users\alfre>curl -v http://localhost:8000/private/test.html
* Host localhost:8000 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:8000...
*   Trying 127.0.0.1:8000...
* Connected to localhost (127.0.0.1) port 8000
* using HTTP/1.x
> GET /private/test.html HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/8.14.1
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Date: Thu, 30 Oct 2025 00:39:56 GMT
< Server: Webserver
< Content-Type: text/html
< Content-Length: 22
< Connection: close
<
<h1>403 Forbidden</h1>* we are done reading and this is set to close, stop send
* abort upload
* shutting down connection #0
```

404 Not Found -

Point to a non-existing file

```
C:\Users\alfre>curl -v http://localhost:8000/hi.html
* Host localhost:8000 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:8000...
*   Trying 127.0.0.1:8000...
* Connected to localhost (127.0.0.1) port 8000
* using HTTP/1.x
> GET /hi.html HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/8.14.1
> Accept: */*
>
< HTTP/1.1 404 Not Found
< Date: Thu, 30 Oct 2025 00:42:26 GMT
< Server: Webserver
< Content-Type: text/html
< Content-Length: 22
< Connection: close
<
<h1>404 Not Found</h1>* we are done reading and this is set to close, stop send
* abort upload
* shutting down connection #0
```

505 HTTP Version Not Supported -

Send a request with an unsupported HTTP version such as HTTP 2 in this case.

```
C:\Users\alfre>ncat localhost 8000
GET /test.html HTTP/2
HTTP/1.1 505 HTTP Version Not Supported
Date: Thu, 30 Oct 2025 01:23:23 GMT
Server: Webserver
Content-Type: text/html
Content-Length: 39
Connection: close

<h1>505 HTTP Version Not Supported</h1>
```

Step 3: Performance

a. A web server hosts resources locally and serves them to clients. A proxy server (web cache) acts as an intermediary between the client and the origin server. It acts as a server to the client and a client to the origin server. It stores (caches) recently requested objects to reduce response times and network traffic to the origin server for frequently accessed content.

Specifications:

- Listens for requests from client
- Forward requests to origin server
- Stores responses to its local cache
- Streams responses back to client
- Handle headers
- Handle error codes

b. To test the implementation, we will use three different terminals:

- Origin server (Listens to port 8000)
- Proxy server (Listens to port 8888)
- Client

Part 1: Initial Request (Cache Miss)

The client sends a request to the proxy server [curl.exe -v -x http://localhost:8888 http://localhost:8000/test.html]. The proxy will first check its local cache, but since no entry is found, it will result in a cache miss. The request will then be forwarded to the origin server. The proxy stores the response in its local cache, and forwards it to the client.

Client

```
PS C:\Users\eulun> curl.exe -v -x http://localhost:8888 http://localhost:8000/test.html
* Host localhost:8888 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:8888...
*   Trying 127.0.0.1:8888...
* Connected to localhost (127.0.0.1) port 8888
* using HTTP/1.x
> GET http://localhost:8000/test.html HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/8.14.1
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Request completely sent off
< HTTP/1.1 200 OK
< Date: Thu, 30 Oct 2025 00:14:20 GMT
< Server: Webserver
< Content-Type: text/html
< Content-Length: 308
< Connection: close
<
```

Proxy

```
PS C:\Users\eulun\Desktop\CMPT371_MP\CMPT371-Mini-Project-1> python proxy.py
Proxy listening on localhost:8888
[PROXY] ('127.0.0.1', 51218): GET http://localhost:8000/test.html HTTP/1.1
[PROXY] Connecting to localhost:8000
[CACHE STORED] localhost:8000/test.html
```

Origin Server

```
PS C:\Users\eulun\Desktop\CMPT371_MP\CMPT371-Mini-Project-1> python origin.py
Server listening on port 8000...
[CONNECTED] ('127.0.0.1', 51220)
[REQUEST] from ('127.0.0.1', 51220):
GET /test.html HTTP/1.1
host: localhost:8000
user-agent: curl/8.14.1
accept: */*
proxy-connection: Keep-Alive
via: 1.1 localhost:8888
connection: close
```

Part 2: Repeated Request (Cache Hit)

The client sends the same request to the proxy server. Since the response from before had been stored in its local cache, it will result in a cache hit. The proxy server will send the cached response to the client without having to request from the origin server. The screenshots below show that the origin server never receives the second request, and the client received the same cached response.

Client

```
PS C:\Users\eulun> curl.exe -v -x http://localhost:8888 http://localhost:8000/test.html
* Host localhost:8888 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:8888...
*   Trying 127.0.0.1:8888...
* Connected to localhost (127.0.0.1) port 8888
* using HTTP/1.x
> GET http://localhost:8000/test.html HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/8.14.1
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Request completely sent off
< HTTP/1.1 200 OK
< Date: Thu, 30 Oct 2025 00:14:20 GMT
< Server: Webserver
< Content-Type: text/html
< Content-Length: 308
< Connection: close
<
```

Proxy

```
PS C:\Users\eulun\Desktop\CMPT371_MP\CMPT371-Mini-Project-1> python proxy.py
Proxy listening on localhost:8888
[PROXY] ('127.0.0.1', 51218): GET http://localhost:8000/test.html HTTP/1.1
[PROXY] Connecting to localhost:8000
[CACHE STORED] localhost:8000/test.html
[PROXY] ('127.0.0.1', 51859): GET http://localhost:8000/test.html HTTP/1.1
[CACHE HIT] localhost:8000/test.html
```

Origin Server

```
PS C:\Users\eulun\Desktop\CMPT371_MP\CMPT371-Mini-Project-1> python origin.py
Server listening on port 8000...
[CONNECTED] ('127.0.0.1', 51220)
[REQUEST] from ('127.0.0.1', 51220):
GET /test.html HTTP/1.1
host: localhost:8000
user-agent: curl/8.14.1
accept: */*
proxy-connection: Keep-Alive
via: 1.1 localhost:8888
connection: close
```

c. The implemented web server is multi-threaded. Each client connection results in a new thread, allowing the server to have multiple clients connected at the same time. This enables the server to respond to each of them independently in parallel, resulting in better throughput and response times. This also leads to better scalability since the web server can serve many clients at once.