Documentation:

Step 2 Implementations:

Note: Despite being able to make requests via browser, we haven't found a way to modify get requests coming from the browser (ie: changing header fields, altering text), so some testing is done within the terminal. However, any returned html messages have been tested independently before implementation.

400 Bad Request:
Sending any get request where the beginning of the request is not 'GET /' will result in a '400 Bad Request' message being displayed.

Request:

```
request = """ACQUIRE /test.html HTTP/1.1
Host: localhost:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 10 Jan 2023 02:01:00 GMT"""
```

Client Result:

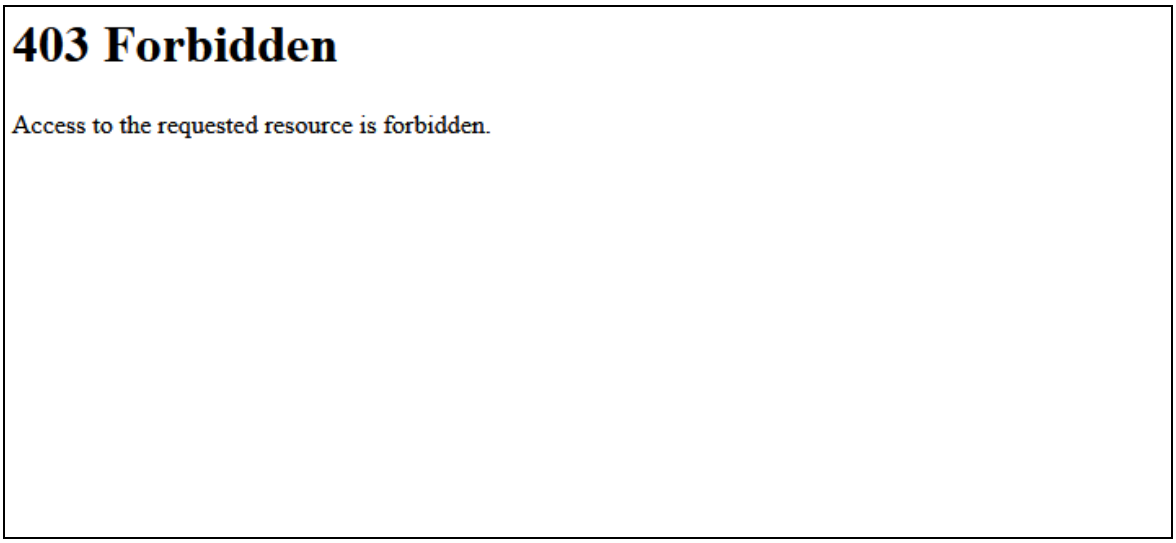
```
From Server: HTTP/1.1 400 Bad Request
Content-Type: text/html

<html><body><h1>400 Bad Request</h1><p>Your request is invalid.</p></body></html>
...
```

403 Forbidden:
We decided to handle the forbidden code with a forbidden directory as opposed to authentication headers because it seemed more streamlined to implement and easier to test. Attempting to access any file in the directory ./forbidden will return a webpage with '403 Forbidden':

Request:

```
request = """GET /forbidden/secret_passwords.html HTTP/1.1
Host: localhost:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 10 Jan 2023 02:01:00 GMT"""
```

Client Result:



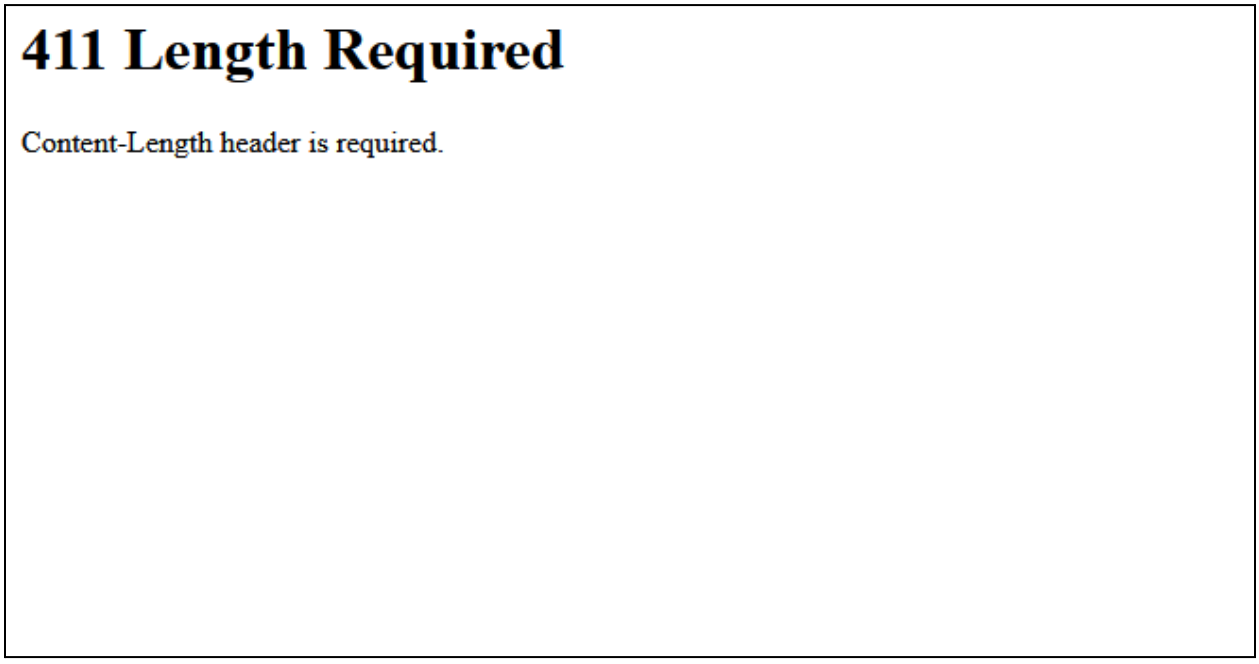
411 Length Required:

Any request that doesn't contain the header 'Content-Length' will return a '411 Length Required' status message:

Request:

```
request = """GET /test.html HTTP/1.1
Host: localhost:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
If-Modified-Since: Sun, 10 Jan 2023 02:01:00 GMT"""
```

Client Result:



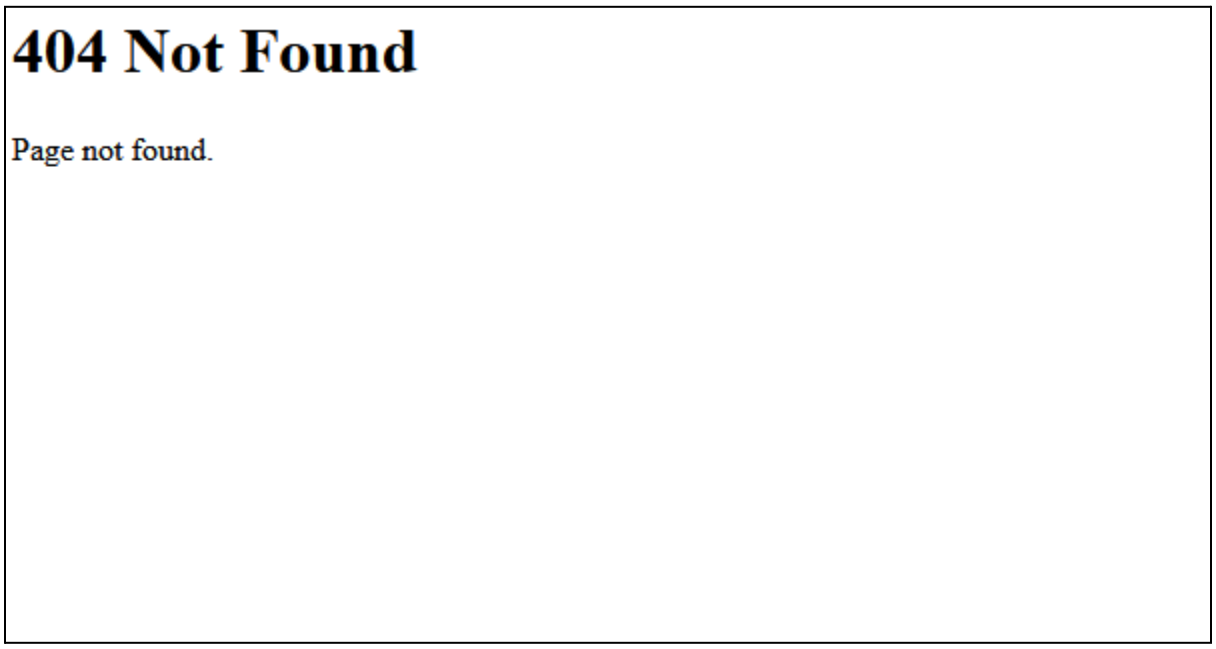
404 Not Found:

Simply enough, if the specified path in the GET request is not found in the server directory (ie: filepath does not exist), then a '404 Not Found' status message is shown to the client:

Request:

```
request = """GET /doesnotexist.html HTTP/1.1
Host: localhost:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
If-Modified-Since: Sun, 10 Jan 2023 02:01:00 GMT"""
```

Client Result:



304 Not Modified:

For simplicity in implementing this status code, we hardcoded a datetime object with a specific date as the web server's 'last modified date', equal to November 30th, 2023, and used it to compare against the requests 'If-Modified-Since' conditional header value. If the header's date was before the web server's 'last modified date', then the client received the web page along with a '200 OK' status, but if the header's date was after, the client would receive a '304 Not Modified' status code in lieu of the web browser loading the webpage from the cache and displaying it.

Request (If-Modified-Since before the server's last modified date):

```
request = """GET /test.html HTTP/1.1
Host: localhost:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 10 Oct 2023 02:01:00 GMT"""
```

Client Result:

```
From Server: HTTP/1.1 200 OK
Content-Type: text/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>
```

Request (If-Modified-Since after the server's last modified date):

```
request = """GET /test.html HTTP/1.1
Host: localhost:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 10 Dec 2023 02:01:00 GMT"""

clientSocket.send(request.encode('utf-8'))
```

Client Result:

```
From Server: HTTP/1.1 304 Not Modified
Content-Type: text/html

<h1>304 Not Modified</h1>
...
```

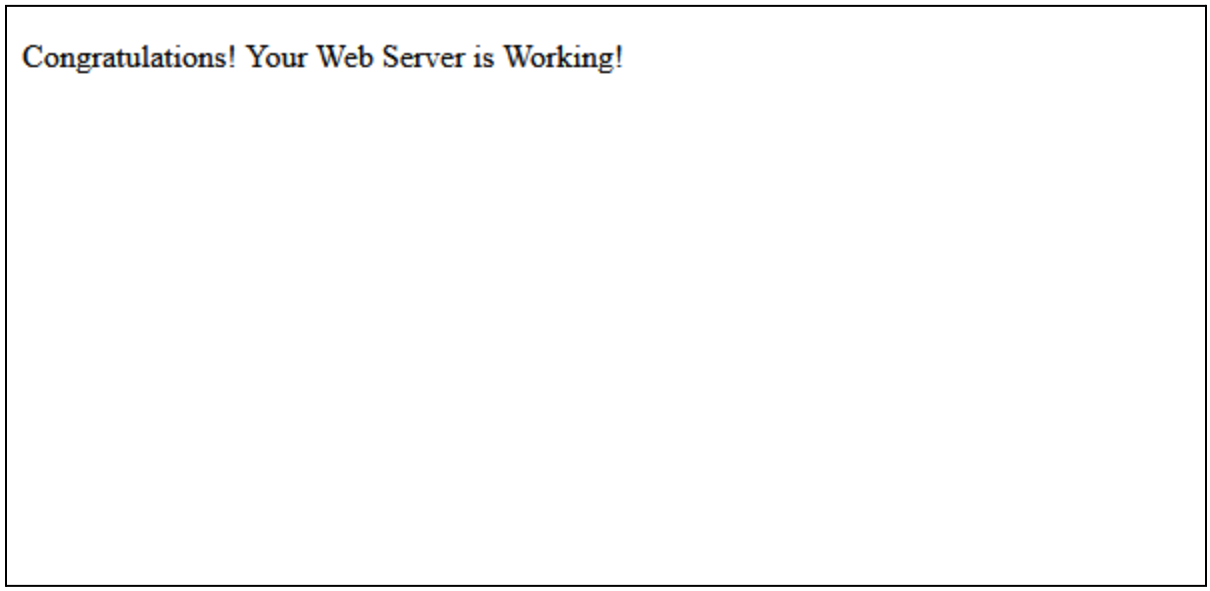
200 OK:

Finally, if there are no errors resulting from the GET request (as mentioned above), the server will return the requested webpage.

Request:

```
request = """GET /test.html HTTP/1.1
Host: localhost:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 10 Nov 2023 02:01:00 GMT"""
```

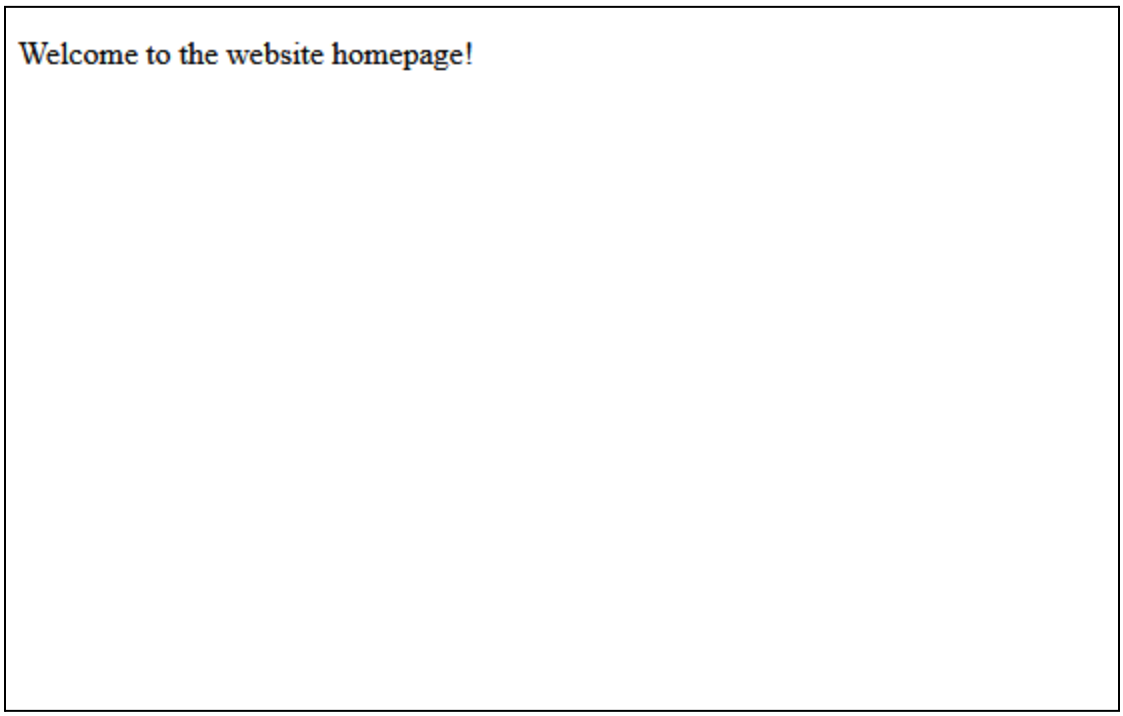
Client Result:



Request:

```
request = """GET /index.html HTTP/1.1
Host: localhost:12000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 10 Nov 2023 02:01:00 GMT"""
```

Client Result:



Step 3 Implementation:

Specifications:

Based on slide 30, we implemented a proxy server that acts as the middleman between the client and the web server, forwarding requests and responses from one end to the other through itself. This required opening both a long-term connection with the web server as well as many short-term connections with the client and their requests.

Adding concepts from slides 29 and 30, we implemented a cache structure within the proxy server that caches response data and links it to a request so that if a similar request is made we can access the cache instead of the web server itself, reducing latency between the client and server as well as lightening the load on the web server.

Finally, implementing the Conditional Get idea from slide 34, we implemented more functionality in the cache to allow the proxy server to receive a request, search for any similar requests within the cache, and if those cached requests are up to date based on the requests If-Modified-Since

header, simply return the response data stored in the cache instead of accessing the server. Of course, if there is no such data in the cache, the proxy server will access the web server and fetch the response for the client, as well as adding that new response data and its date into the cache.

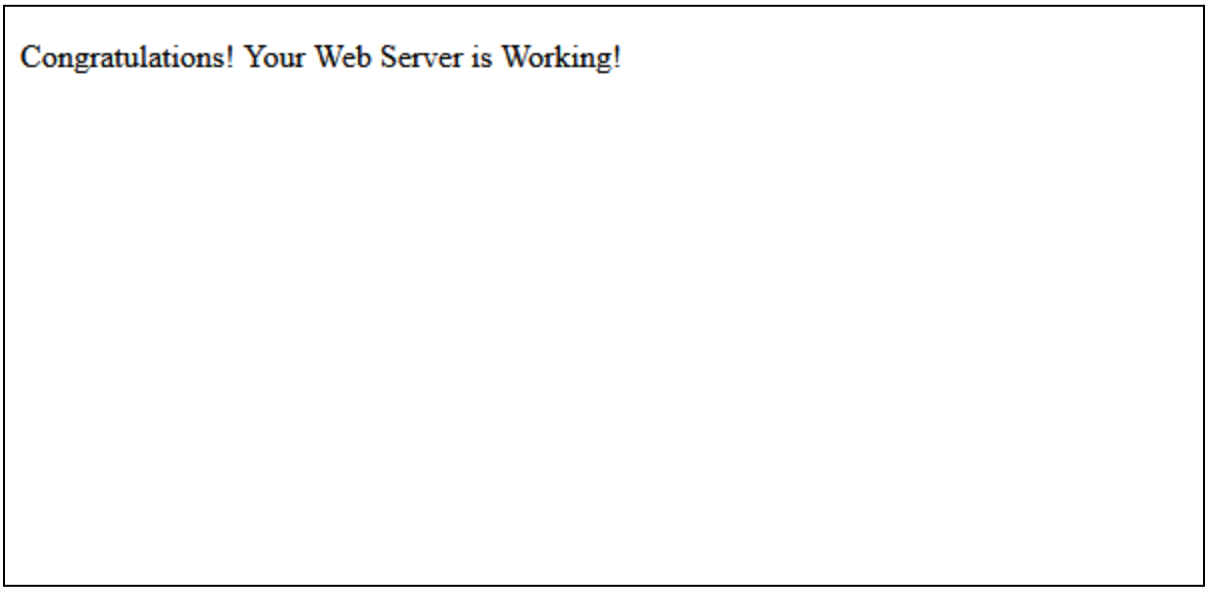
Testing Methodology:

In the initial case (first client request), the proxy server functionally does nothing more than forwarding the request to the web server, then forwarding the web server response back to the client.

Request:

```
request = """GET /test.html HTTP/1.1
Host: localhost:13000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 10 Jan 2023 02:01:00 GMT"""
```

Client Result:



A subsequent request for a new (unique) webpage will function similarly.

Request:

```
request = """GET /index.html HTTP/1.1
Host: localhost:13000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 10 Jan 2023 02:01:00 GMT"""
```

Client Result:

(Since the hard-coded last modified date on the web server is November 30, 2023, and the If-Modified-Since header is dated January 10, 2023 for both requests, the current version of the webpage is both returned to the client AND cached within the proxy server.)

However, should there be a duplicate request (ie: a GET request for the same webpage by the same user) that has already been stored in the cache, if the cached version has been updated on a date after the client is requesting, the client will receive the cached response data.

Request:

```
request = """GET /test.html HTTP/1.1
Host: localhost:13000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 6 Jan 2023 02:01:00 GMT"""
```

Proxy Server Output:

```
Successfully connected to client address ('127.0.0.1', 64587)
Request received from client:
GET /test.html HTTP/1.1
Host: localhost:13000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 6 Jan 2023 02:01:00 GMT
//////////////////////////////////////////
Cache hit! Sending cached response to client.
//////////////////////////////////////////
```

Client Result:

```
From Server: HTTP/1.1 304 Not Modified
Content-Type: text/html

<h1>304 Not Modified</h1>
...
```

Because of the way we have configured our cache functionality, a successful cache hit in the proxy server will trigger a 304 Not Modified message to signify that the content the client is requesting is already contained in the cache (ie: the ‘new’ content they are requesting is the same as the cached content, meaning the ‘new’ content has not been modified.) In a fully functional web/proxy server setup, the client would simply receive the cached webpage, however, for testing purposes, we wanted to differentiate a cache hit from the proxy server from a response from the actual web server. Speaking of…

In the case where a request and its response is contained in the cache BUT the cached response date is older than the If-Modified-Since date in the request, the proxy server is required to update its cache with a version of the webpage that fulfills the new requests’ date requirements:

Request:

```
request = """GET /test.html HTTP/1.1
Host: localhost:13000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 12 Jan 2023 02:01:00 GMT"""
```

(The newest version of test.html in the cache is dated January 10th, 2023)
Proxy Server Output:

```
Successfully connected to client address ('127.0.0.1', 64633)
Request received from client:
GET /test.html HTTP/1.1
Host: localhost:13000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 12 Jan 2023 02:01:00 GMT
Response received from web server:
HTTP/1.1 200 OK
Content-Type: text/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>
```

Client Result:

```
From Server: HTTP/1.1 200 OK
Content-Type: text/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>

...
```

Despite the client receiving the same response as in the initial case, the proxy server has operated a bit differently. In this case, there was no cache hit within the proxy server's cache, which meant that the proxy server was required to forward the client's GET request directly to the web server in order to obtain an updated version of the webpage. The returned webpage is then stored into the cache.

If we now send a request with the If-Modified-Since date before the most recent cache addition (but after the initial cache entry), the proxy server will again have a cache hit and not require any interaction with the web server.

Request:

```
request = """GET /test.html HTTP/1.1
Host: localhost:13000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 11 Jan 2023 02:01:00 GMT"""
```

Proxy Server Output:

```
Successfully connected to client address ('127.0.0.1', 62681)
Request received from client:
GET /test.html HTTP/1.1
Host: localhost:13000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 11 Jan 2023 02:01:00 GMT
////////////////////////////////////////////
Cache hit! Sending cached response to client.
////////////////////////////////////////////
```

Client Result:

```
From Server: HTTP/1.1 304 Not Modified
Content-Type: text/html

<h1>304 Not Modified</h1>
...
```

Finally, since the proxy server simply forwards the request if the cache doesn't have an up to date version, if the requested If-Modified-Since header date is newer than the actual web server's last modified date, the client result will again be a 304 Not Modified status code, but this time from the web server itself.

Request:

```
request = """GET /test.html HTTP/1.1
Host: localhost:13000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 11 Dec 2023 02:01:00 GMT"""
```

Proxy Server Output:

```
Successfully connected to client address ('127.0.0.1', 62685)
Request received from client:
GET /test.html HTTP/1.1
Host: localhost:13000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html
Connection: keep-alive
Content-Length: 0
If-Modified-Since: Sun, 11 Dec 2023 02:01:00 GMT
Response received from web server:
HTTP/1.1 304 Not Modified
Content-Type: text/html

<h1>304 Not Modified</h1>
```

(Notice there is no cache hit message in the proxy server, but a 304 code is still received.)

Client Result:

```
From Server: HTTP/1.1 304 Not Modified
Content-Type: text/html

<h1>304 Not Modified</h1>
...
```

As well, in this case the cache is not updated with the response because there is no returned data to cache.

Step 4 - HOL Mitigation:

While reading through the MP, we were made aware of the potential of an HOL blocking issue by the mention of the step 4 bonus point. With that in mind, we decided to proactively use the ‘threading’ python library to handle this issue. The way we have implemented it, every incoming request for both our proxy server and web server is directed to its own thread, where the request is either handled and returned (in the case of the web server) or checked against the proxy server cache and either handled or forwarded to the web server. Designating each request to a thread allows both servers to handle requests in parallel, ensuring that a slow request or lost packets will not hold up subsequent requests from being processed and handled.