CSEN 317: Distributed Systems
Programming Assignment 1

Naveena Avula
SCU ID : 07700009231

Introduction:

The objective of this programming assignment is to develop a functional web server, focusing on essential distributed programming principles, client-server communication, and the challenges of building web server functionality.

Once a connection is established, the server uses a dedicated thread to handle image file requests, ensuring efficient processing for both HTML and image data. This multithreaded design allows the server to manage the resource-heavy task of image handling without impacting overall server responsiveness.

A key feature of the program is its connection timeout mechanism. Each connection is maintained for a specified period of 10 seconds, ensuring connections are not held open unnecessarily. This timeout approach aids in resource management, preventing bottlenecks and potential resource exhaustion.

The server adheres to HTTP/1.1 specifications to handle incoming requests, while also supporting HTTP/1.0. Adding support for HTTP/1.0 allows the server to work with an earlier version of the Hypertext Transfer Protocol, which uses stateless connections and limited headers, with individual connections per request-response cycle. This standards-compliant approach ensures that the server remains compatible with a wide range of browsers and web clients.

Program Specifications:

- **Functionality**:
    - The program operates as a functional web server in Python.
    - It serves files from a designated directory upon receiving HTTP requests from clients.
- **HTTP Methods**:
    - Primarily handles GET requests as part of basic HTTP method support.
- **HTTP Status Codes**:
    - Implements various HTTP status codes, including:
        - 200 (OK)
        - 400 (Bad Request)

        - 403 (Forbidden)
        - 404 (File Not Found)
        - 501 (Not Supported)
- **Response Headers**:
    - Incorporates key response headers such as:
        - Content-Type
        - Content-Length
        - Date
- **File Types**:
    - Supports a range of file formats, including .pdf, .jpg, .txt, .png, .gif, .mp4 and .html.
- **Concurrency**:
    - Utilizes multi-threading to manage multiple client requests concurrently.
    - Features a default idle timeout of 500 seconds for active connections.

Uploaded Files:

- error (contains 400.html, 403.html, 404.html, 501.html)
- all_files.html
- collegelogo.jpeg
- error_page.gif
- index.html
- index1.html
- makefile
- sample_video.mp4
- server_def.txt
- server_image.png
- server.py
- socket_photo.jpg
- web_server.pdf

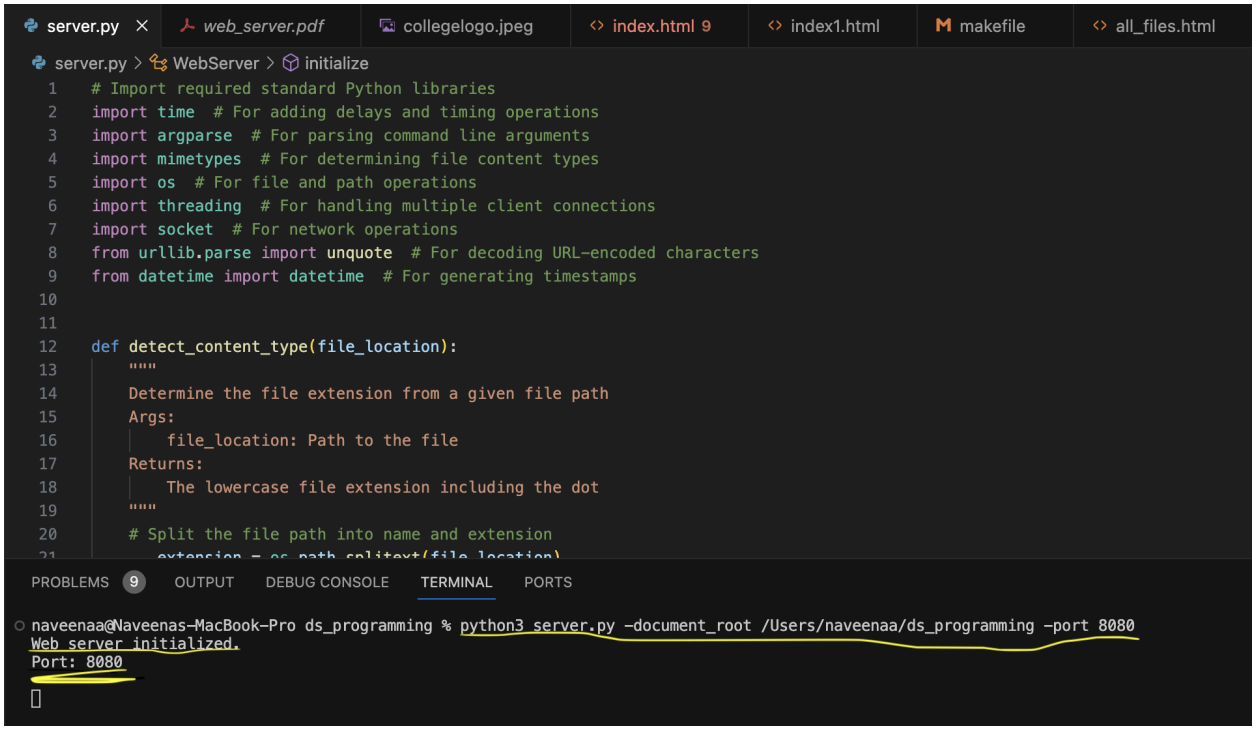
Code Execution:

Python3 <document name>.py -document_root <path to folder> -port <number btw 8000 to 9999>

- Example:python3 server.py -document_root /Users/naveenaa/ds_programming -port 8080

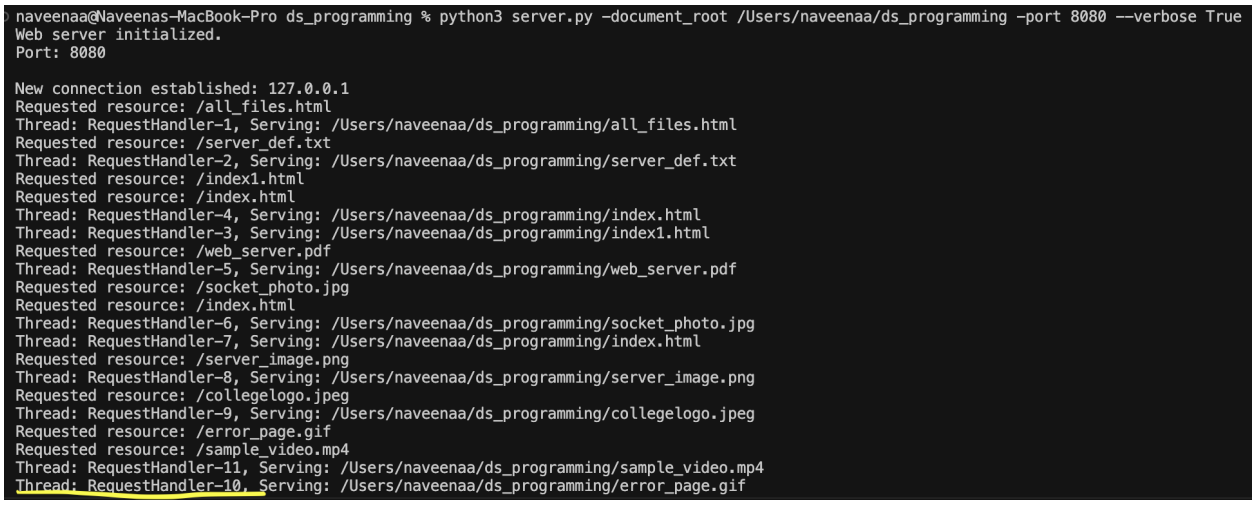
Launch a web browser or client that can connect to the server's designated port and send HTTP requests.

- Example: http://localhost:8080.



Multi Threading:

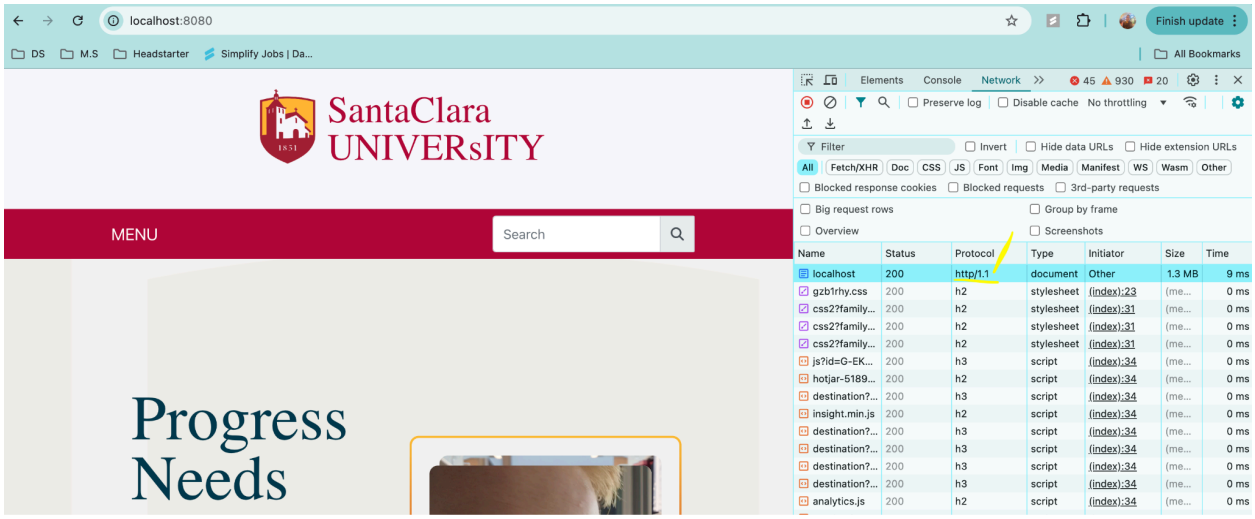
Incorporating multithreading into a web server allows for concurrent handling of multiple client requests through individual threads. However, the Global Interpreter Lock (GIL) in Python restricts true parallelism, making it more suitable for I/O-bound tasks where threads can effectively manage multiple connections at once. While this enhances responsiveness when dealing with numerous concurrent requests, the GIL's effect is more significant in CPU-bound tasks, which may limit performance improvements in parallel processing.

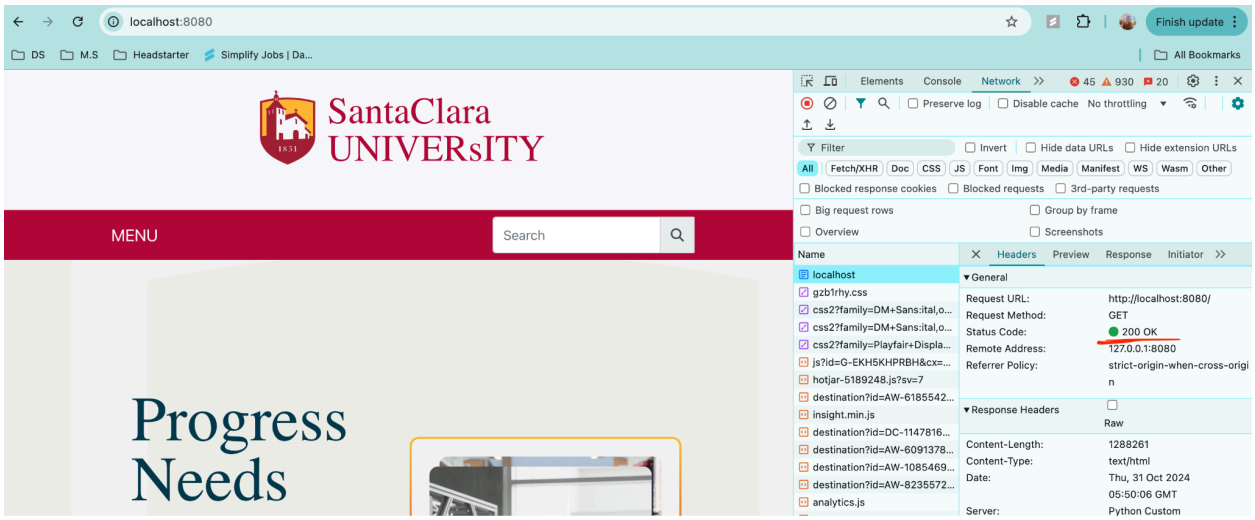


File formats:

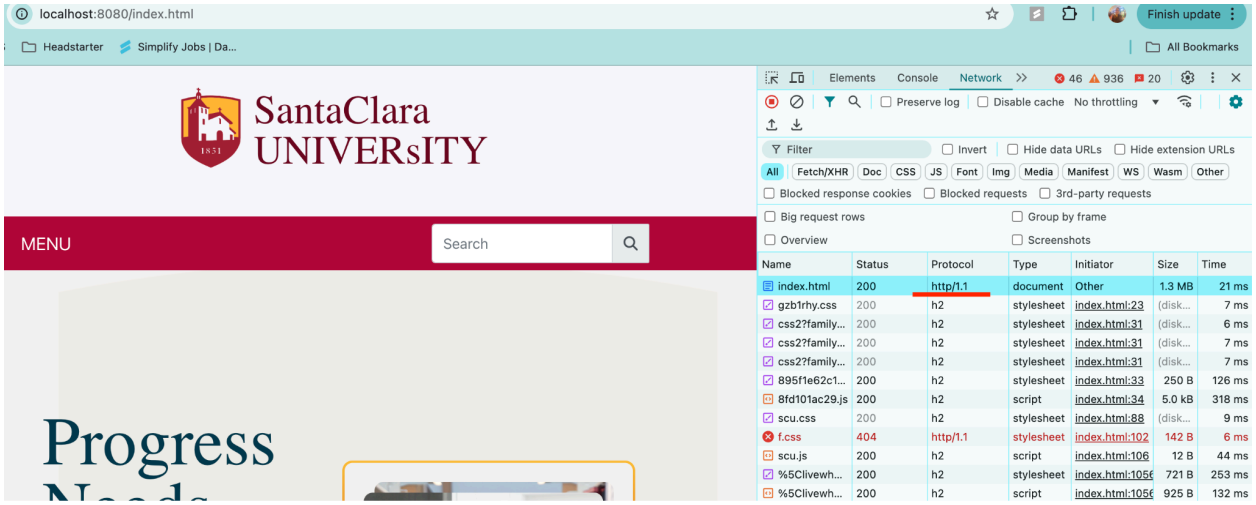
.jpg, .png, .jpeg, .gif, .txt, .html, .pdf, .mp4

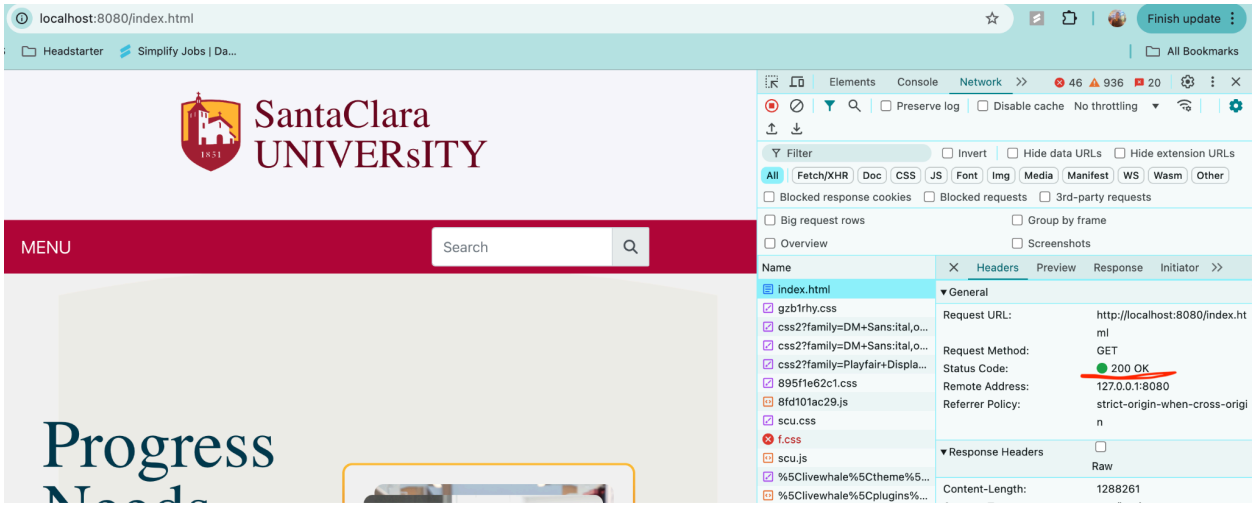
1. Launch a browser and input localhost:[port number] (Example: localhost:8080). The server will respond by delivering the content linked to the default endpoint with a status code of 200.





2.Open a browser and type localhost:[port number]/index.html. The server will respond by providing the content associated with the default endpoint (/index.html) from our document root directory, returning a status code of 200.



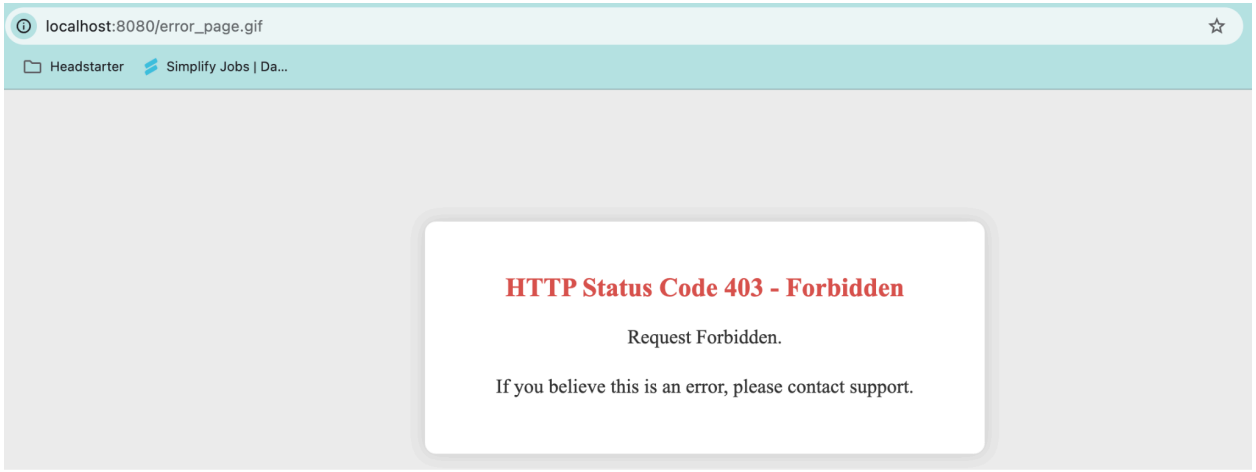


3.After removing read access to the file, the web server will respond with a 403 Forbidden status code, indicating that permissions to access the file are insufficient.

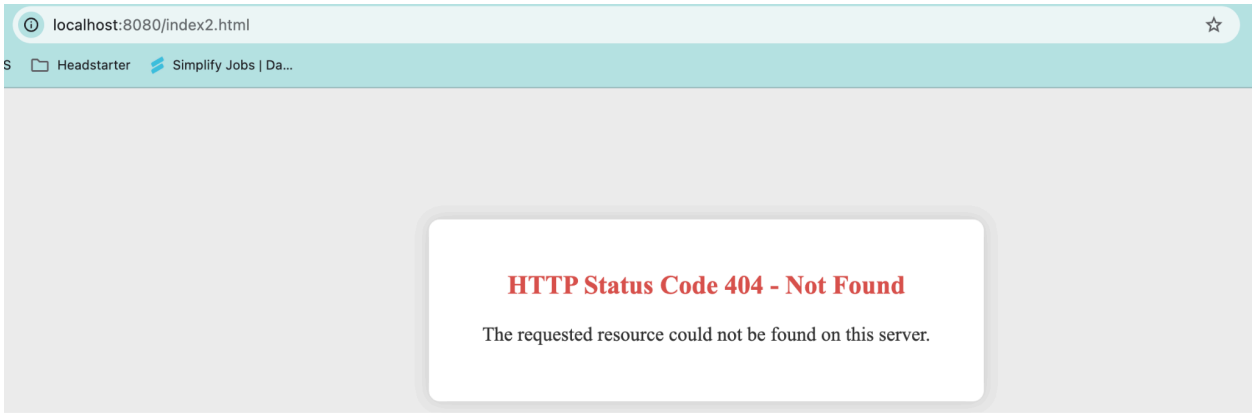
Before removing permissions:

```
naveenaa@Naveenas-MacBook-Pro ds_programming % ls -lart error_page.gif
-rw-r--r--@ 1 naveenaa  staff  759650 Oct 30 15:08 error_page.gif
```

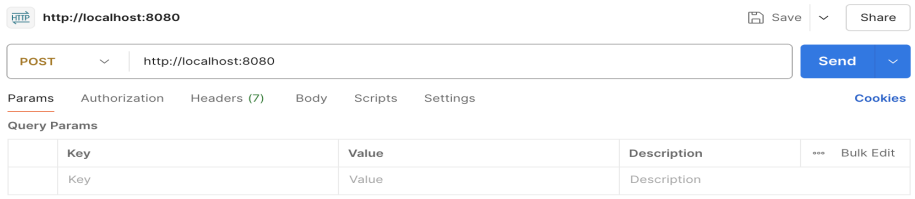
After removing permissions:



4. The web server generates a 404 Not Found response because the requested file index2.html (or any file that is not present in the root directory) cannot be found in the specified document root directory.

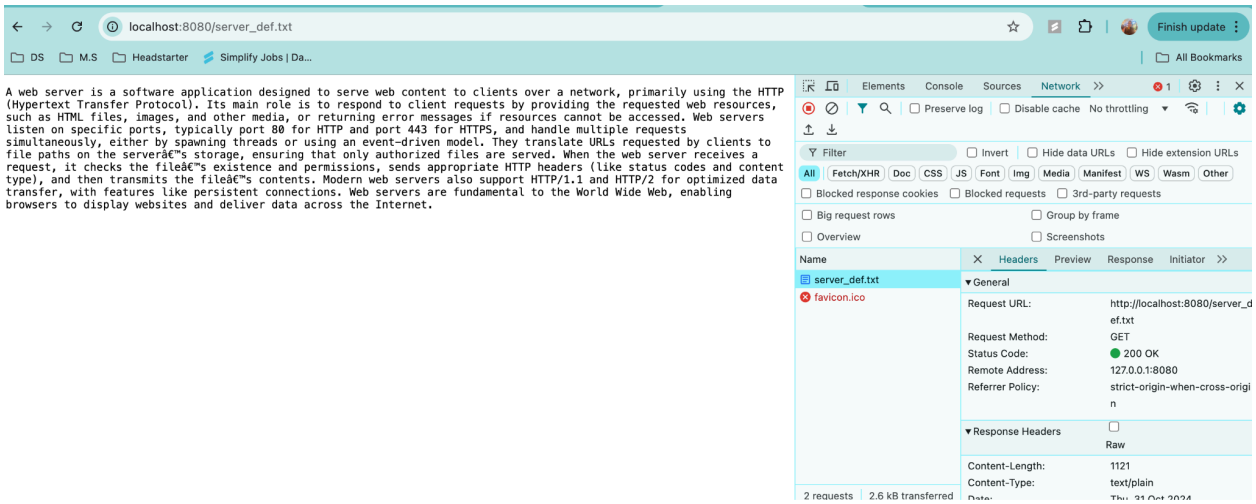


5. Submitting an invalid POST request to index.html results in a 400 Bad Request response from the web server.

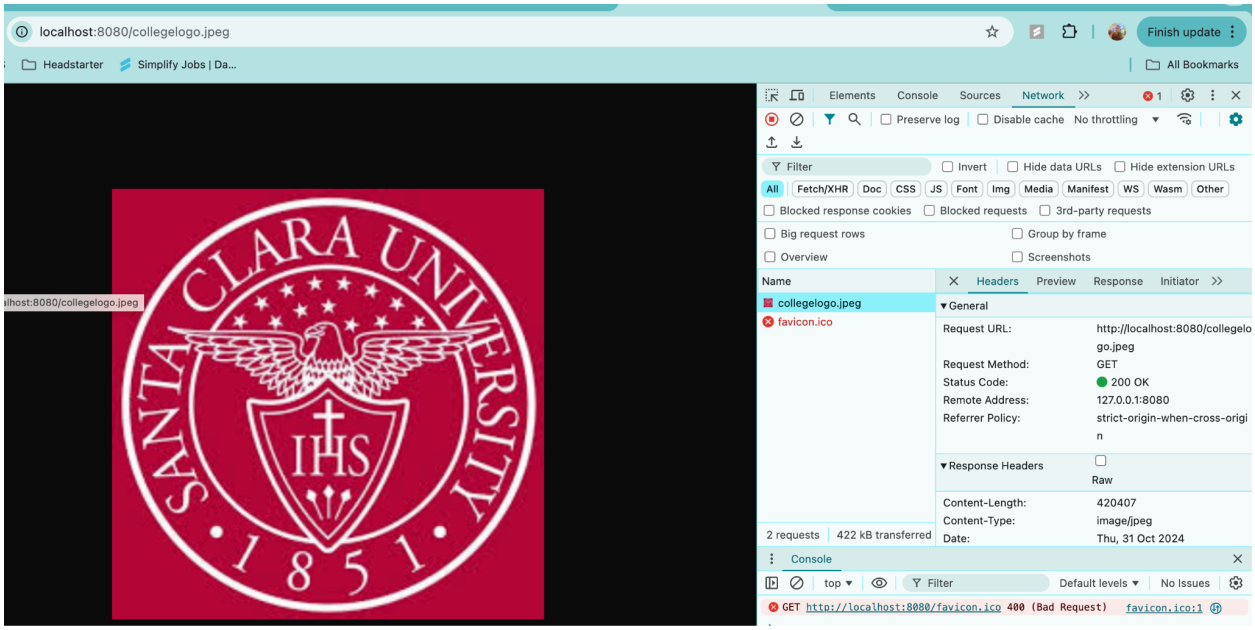


Client accessing different file contents:

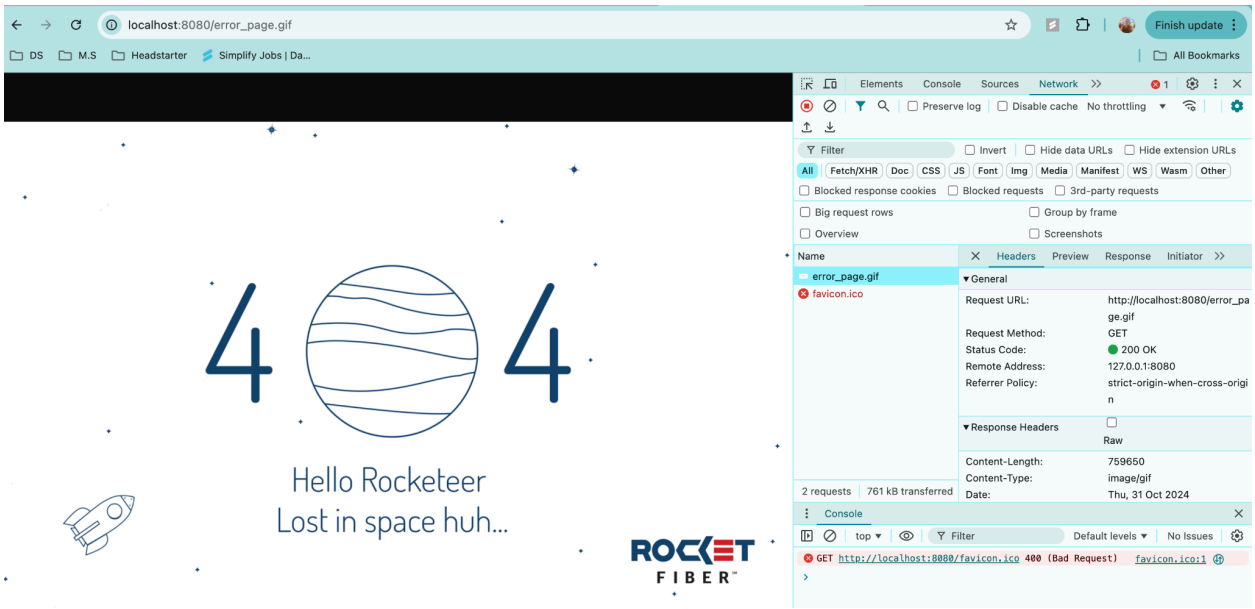
1.The server responds by delivering the content of the existing text file server_def.txt from the specified document root directory.



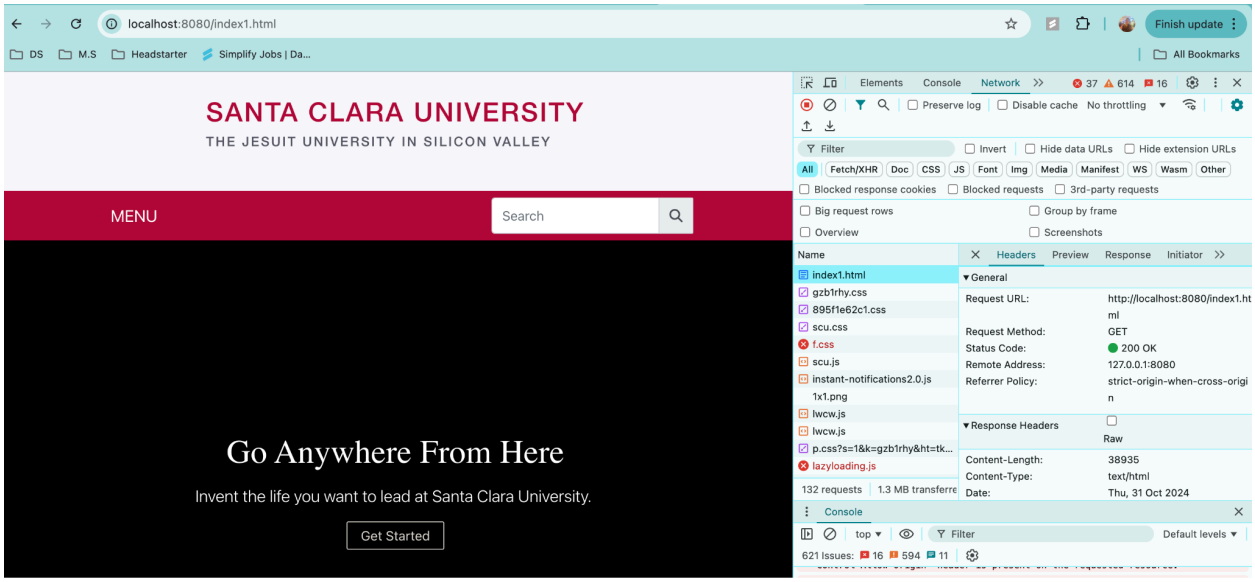
2.The server responds by delivering the content of the existing jpeg file collegelogo.jpeg from the specified document root directory.



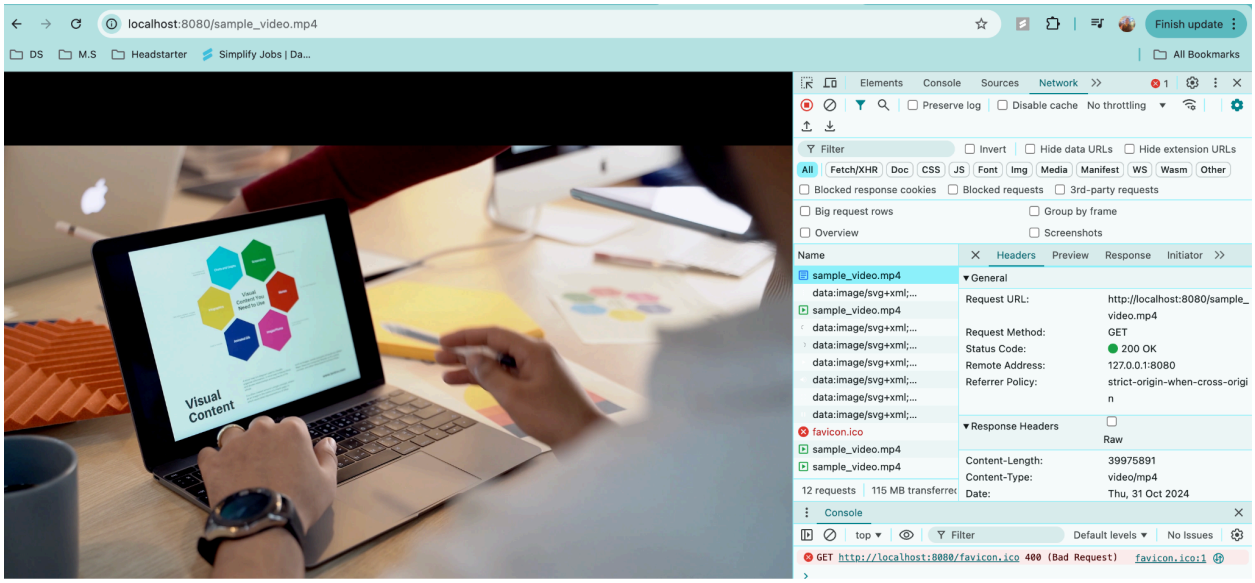
3.The server responds by delivering the content of the existing gif file error_page.gif from the specified document root directory.



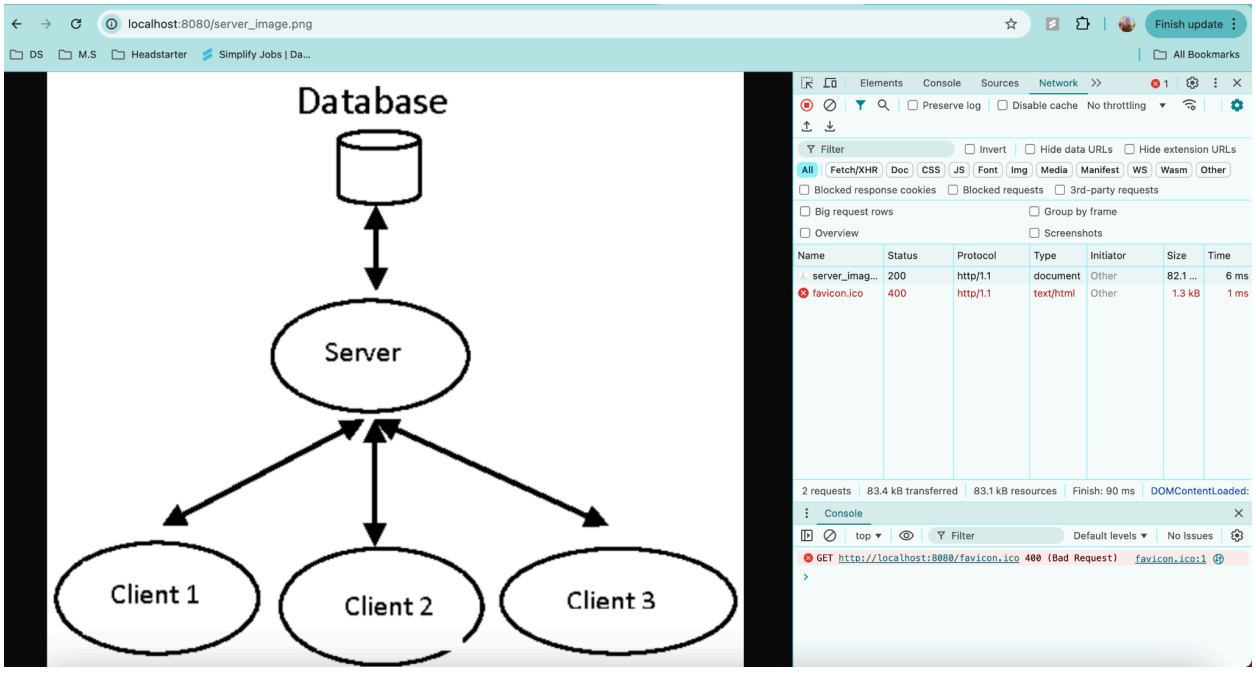
4.The server responds by delivering the content of the existing html file index1.html from the specified document root directory.



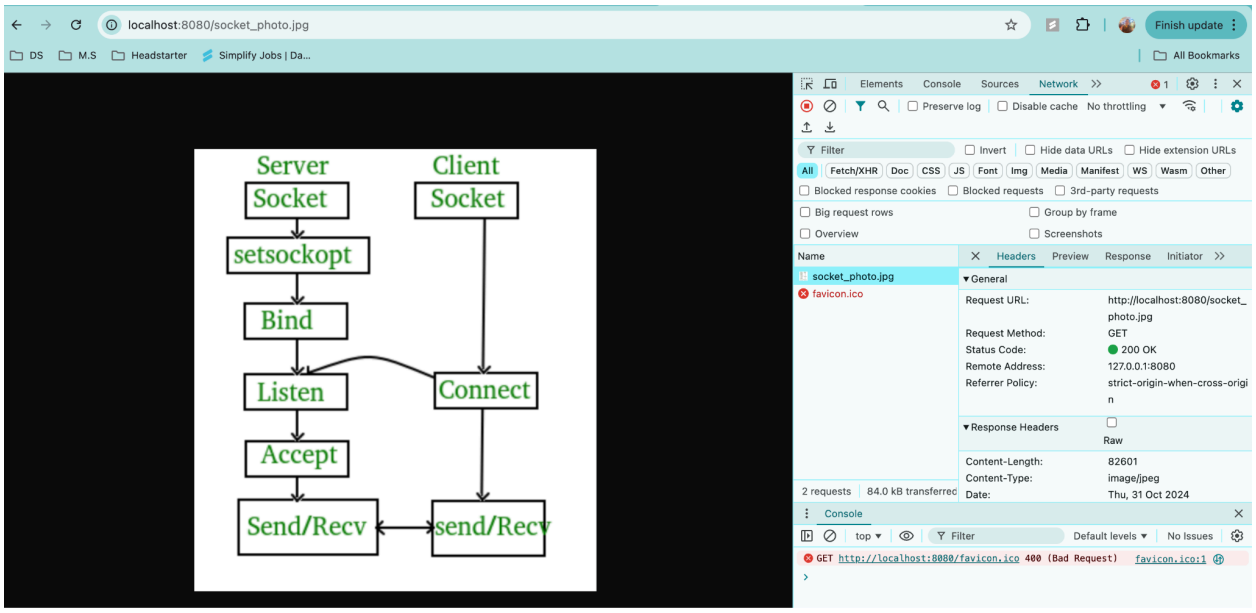
5.The server responds by delivering the content of the existing video file sample_video.mp4 from the specified document root directory.



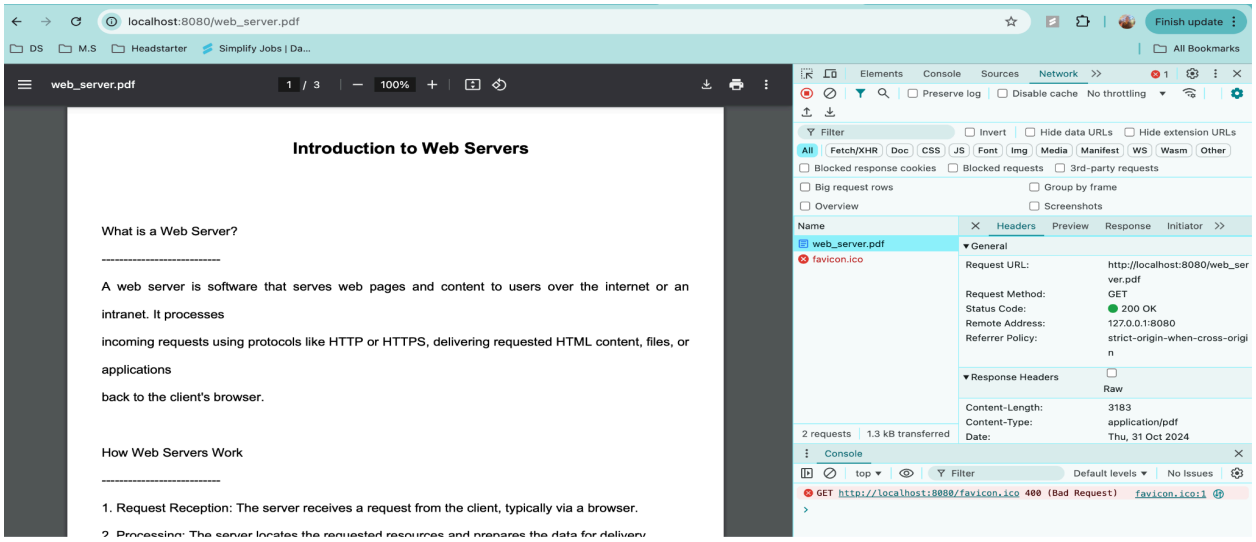
6.The server responds by delivering the content of the existing png file server_image.png from the specified document root directory.



7.The server responds by delivering the content of the existing jpg file socket_photo.jpg from the specified document root directory.



8.The server responds by delivering the content of the existing pdf file web_server.pdf from the specified document root directory.



Extra Functionality:

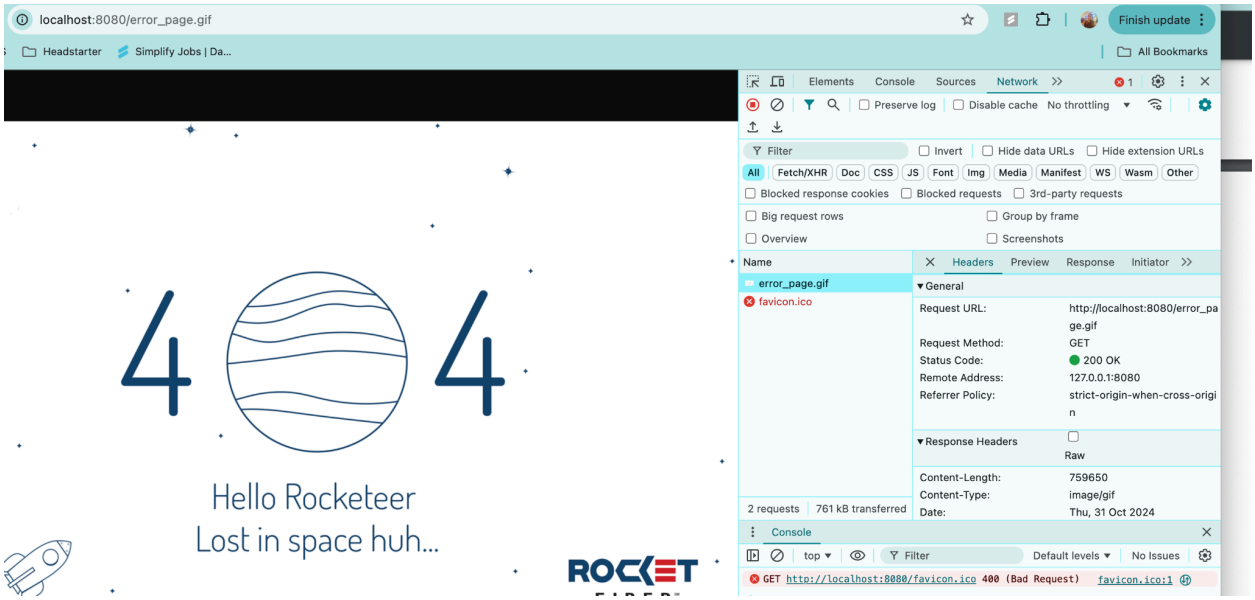
By default, the server.py program utilizes HTTP protocol 1.1 (persistent connection). To enable support for protocol 1.0, use the following optional command-line argument:

- python3 server.py -document_root /Users/naveenaa/ds_programming -port 8080 --http_version 1.0 --verbose True

You can notice that the connection is being closed in the output below:

```
naveenaa@Naveenas-MacBook-Pro ds_programming % python3 server.py -document_root /Users/naveenaa/ds_programming -port 8080 --http_version 1.0
e True
Web server initialized.
Port: 8080

New connection established: 127.0.0.1
Requested resource: /error_page.gif
Thread: RequestHandler-1, Serving: /Users/naveenaa/ds_programming/error_page.gif
Requested resource: /favicon.ico
400 Bad Request: GET
HTTP/1.0 connection terminated
```



References:

- Reference from class notes/slides.
- https://docs.python.org/3/library/argparse.html
- https://docs.python.org/3/tutorial/errors.html
- https://docs.python.org/3/library/mimetypes.html
- https://docs.python.org/3/library/socket.html