PROJECT REPORT

Multi-Threaded Proxy Server and Client

Submitted by

Himanshu Nainwal RA2211003010019
Koshti Vanshika Shaileshbhai RA2211003010021
Gruhit Dilipbhai Kaneriya RA2211003010041

Under the Guidance of

Dr.B.ARTHI
Associate Professor, Department of Computing Technologies

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

OCTOBER 2023

TABLE OF CONTENTS

Problem Statement

The problem at hand is to design and develop a multi-threaded proxy server and client system to facilitate web content retrieval from arbitrary URLs and save the retrieved content as HTML pages. This project aims to address the following key challenges and requirements:

1. **Web Content Access:** In an increasingly digital world, users often need to access and store web content from various online sources. The project seeks to create a solution that simplifies this process.
2. **Concurrent Web Requests:** Users frequently require multiple web requests to be processed simultaneously, and therefore, the project must employ multi-threading to ensure efficient handling of concurrent requests.
3. **Web Content Storage:** After retrieving web content, the project should automatically generate appropriate names for HTML files and store the content in an organized manner.
4. **Versatility:** The system should be capable of accommodating a wide range of URLs and web content types, making it versatile and widely applicable.
5. **Concurrency and Threading:** To provide an efficient solution, multi-threading should be employed for both the server and client components, ensuring that the system can handle multiple requests concurrently.
6. **Synchronization and Error Handling:** The project must include synchronization mechanisms and robust error handling to prevent data corruption and ensure the system's reliability.

Procedure

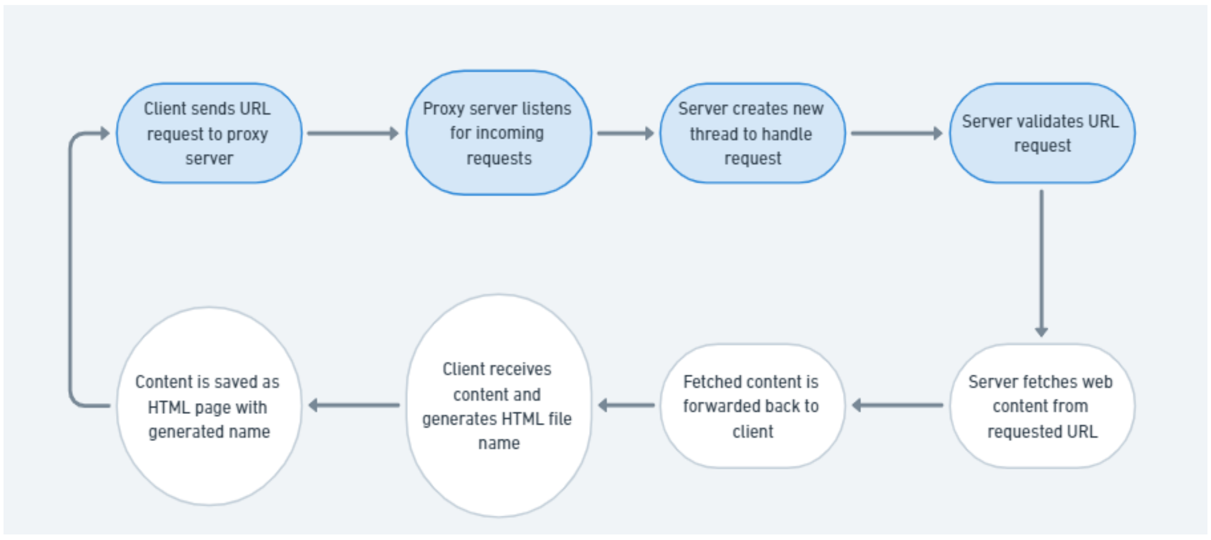
The procedure for implementing the multi-threaded proxy server and client project involves several steps, from setting up the development environment to handling concurrent requests and saving web content as HTML pages. Here's a high-level overview of the project procedure:

1. Environment Setup
2. Server Component
3. Client Component
4. URL Fetching
5. HTML Page Creation
6. Concurrency and Threading
7. Error Handling
8. User Interface
9. Testing

System architecture/Flowchart

1. Client Component
2. Proxy Server Component
3. URL Fetching Component
4. HTML Page Creation Component
5. Concurrency and Threading

Flowchart


Client sends URL request to proxy server
Proxy server listens for incoming requests
Server creates new thread to handle request
Server validates URL request
Server fetches web content from requested URL
Fetched content is forwarded back to client
Client receives content and generates HTML file name
Content is saved as HTML page with generated name

Server’s Code

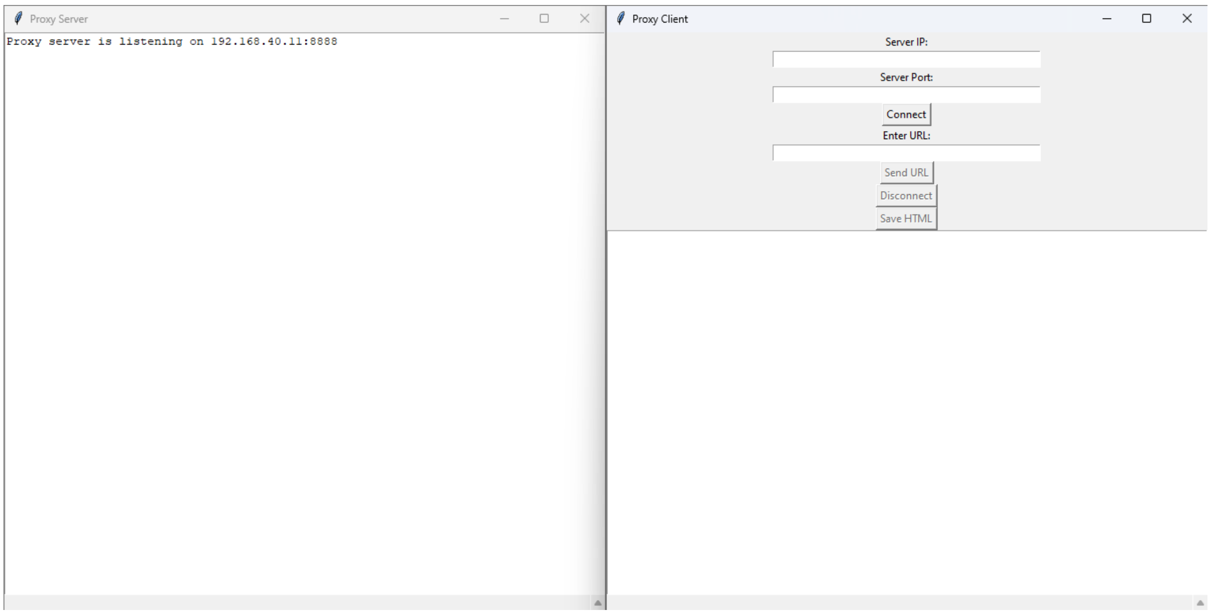
```python
import socket
import threading
import http.client
from urllib.parse import urlparse
import tkinter as tk
from tkinter import Text, Scrollbar

# Proxy server configuration
SERVER_HOST = '192.168.40.11'
SERVER_PORT = 8888

class ServerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Proxy Server")

        self.text_area = Text(root, wrap=tk.WORD)
        self.text_area.pack(expand="true", fill="both")

        self.scroll = Scrollbar(root, command=self.text_area.yview)
        self.scroll.pack(side="right", fill="y")
        self.text_area.config(yscrollcommand=self.scroll.set)

        self.connected_clients = {}

        self.server_thread = threading.Thread(target=self.start_server)
        self.server_thread.start()

    def proxy_server(self, client_socket, client_id):
        try:
            request = client_socket.recv(4096).decode('utf-8')

            if not request:
                print(f"Empty request received from
{self.connected_clients[client_id][1][0]}:{self.connected_clients[client_id][1
][1]}")
                return

            parts = request.split(' ')
            if len(parts) < 2:
                print(f"Invalid request format from
{self.connected_clients[client_id][1][0]}:{self.connected_clients[client_id][1
][1]}")
                return

            url = parts[1]
            target_url = urlparse(url)

            target_conn = http.client.HTTPSConnection(target_url.netloc)

            try:
                target_conn.request("GET", target_url.path)
                response = target_conn.getresponse()
                page_content = response.read()
            except Exception as e:
                print(f"Failed to fetch the URL: {e}")
                return

            client_socket.send(b"HTTP/1.1 200 OK\r\n\r\n")
            client_socket.send(page_content)
        except Exception as e:
            print(f"An error occurred while processing the request: {e}")
        finally:
            client_socket.close()
            del self.connected_clients[client_id]
            self.display_connected_clients()

    def start_server(self):
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.bind((SERVER_HOST, SERVER_PORT))
        server.listen(5)
        self.text_area.insert("1.0", f"Proxy server is listening on
{SERVER_HOST}:{SERVER_PORT}\n")

        while True:
            client_socket, addr = server.accept()
            self.text_area.insert("1.0", f"Accepted connection from
{addr[0]}:{addr[1]}\n")

            client_id = len(self.connected_clients) + 1
            self.connected_clients[client_id] = (client_socket, addr)
            self.display_connected_clients()

            client_handler = threading.Thread(target=self.proxy_server,
args=(client_socket, client_id))
            client_handler.start()

    def display_connected_clients(self):
        clients_text = "\nConnected Clients:\n"
        for client_id, (client_socket, client_addr) in
self.connected_clients.items():
            clients_text += f"{client_addr[0]}:{client_addr[1]}\n"
        self.text_area.delete("2.0", "end")
        self.text_area.insert("2.0", clients_text)
```

```python
def main():
    root = tk.Tk()
    app = ServerApp(root)
    root.mainloop()

if __name__ == '__main__':
    main()
```

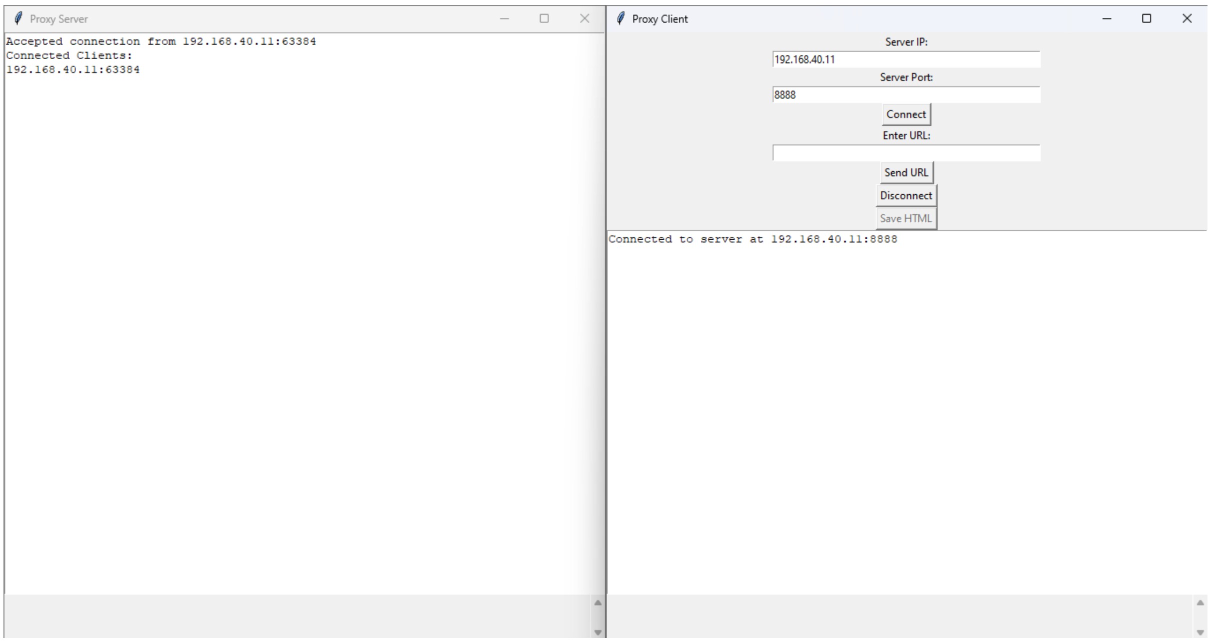
Client’s Code

```
import socket
import tkinter as tk
from tkinter import Entry, Button, Text, Scrollbar, filedialog

class ClientApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Proxy Client")

        self.ip_label = tk.Label(root, text="Server IP:")
        self.ip_label.pack()

        self.ip_entry = Entry(root, width=50)
        self.ip_entry.pack()

        self.port_label = tk.Label(root, text="Server Port:")
        self.port_label.pack()

        self.port_entry = Entry(root, width=50)
        self.port_entry.pack()

        self.connect_button = Button(root, text="Connect",
command=self.connect_to_server)
        self.connect_button.pack()

        self.url_label = tk.Label(root, text="Enter URL:")
        self.url_label.pack()

        self.url_entry = Entry(root, width=50)
        self.url_entry.pack()

        self.send_url_button = Button(root, text="Send URL",
command=self.send_url, state=tk.DISABLED)
        self.send_url_button.pack()

        self.disconnect_button = Button(root, text="Disconnect",
command=self.disconnect, state=tk.DISABLED)
        self.disconnect_button.pack()

        self.save_button = Button(root, text="Save HTML",
command=self.save_html, state=tk.DISABLED)
        self.save_button.pack()

        self.text_area = Text(root, wrap=tk.WORD)
        self.text_area.pack(expand="true", fill="both")
        self.scroll = Scrollbar(root, command=self.text_area.yview)
        self.scroll.pack(side="right", fill="y")
        self.text_area.config(yscrollcommand=self.scroll.set)

        self.client_socket = None
        self.connected = False

    def connect_to_server(self):
        server_ip = self.ip_entry.get()
        server_port = int(self.port_entry.get())

        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            client.connect((server_ip, server_port))
            self.client_socket = client
            self.send_url_button["state"] = tk.NORMAL
            self.disconnect_button["state"] = tk.NORMAL
            self.text_area.insert("1.0", f"Connected to server at
{server_ip}:{server_port}\n")
            self.connected = True
        except ConnectionRefusedError:
            self.text_area.insert("1.0", "Connection to the server failed.
Make sure the server is running.\n")

    def send_url(self):
        if self.client_socket is None:
            return

        url = self.url_entry.get()
        self.client_socket.send(f"GET {url} HTTP/1.1\r\nHost:
{url}\r\n\r\n".encode('utf-8'))

        response = b""
        while True:
            part = self.client_socket.recv(4096)
            if not part:
                break
            response += part

        content_start = response.find(b'\r\n\r\n') + 4
        page_content = response[content_start:]

        self.save_button["state"] = tk.NORMAL
        self.text_area.insert(tk.END, "Received HTML content. You can now save
it.\n")
        self.html_content = page_content

    def disconnect(self):
        if self.client_socket:
            self.client_socket.close()
            self.client_socket = None
            self.text_area.insert(tk.END, "Disconnected from the server.\n")
            self.send_url_button["state"] = tk.DISABLED
            self.disconnect_button["state"] = tk.DISABLED
            self.save_button["state"] = tk.DISABLED
            self.connected = False

    def save_html(self):
        if self.html_content:
            filename = filedialog.asksaveasfilename(defaultextension=".html",
filetypes=[("HTML files", "*.html")])
            if filename:
                with open(filename, 'wb') as file:
                    file.write(self.html_content)
                self.text_area.insert(tk.END, f"Saved the HTML content as
{filename}\n")
                self.disconnect()

def main():
    root = tk.Tk()
    app = ClientApp(root)
    root.mainloop()

if __name__ == '__main__':
    main()
```

Output Screenshot



```
Proxy server is listening on 192.168.40.11:8888
```

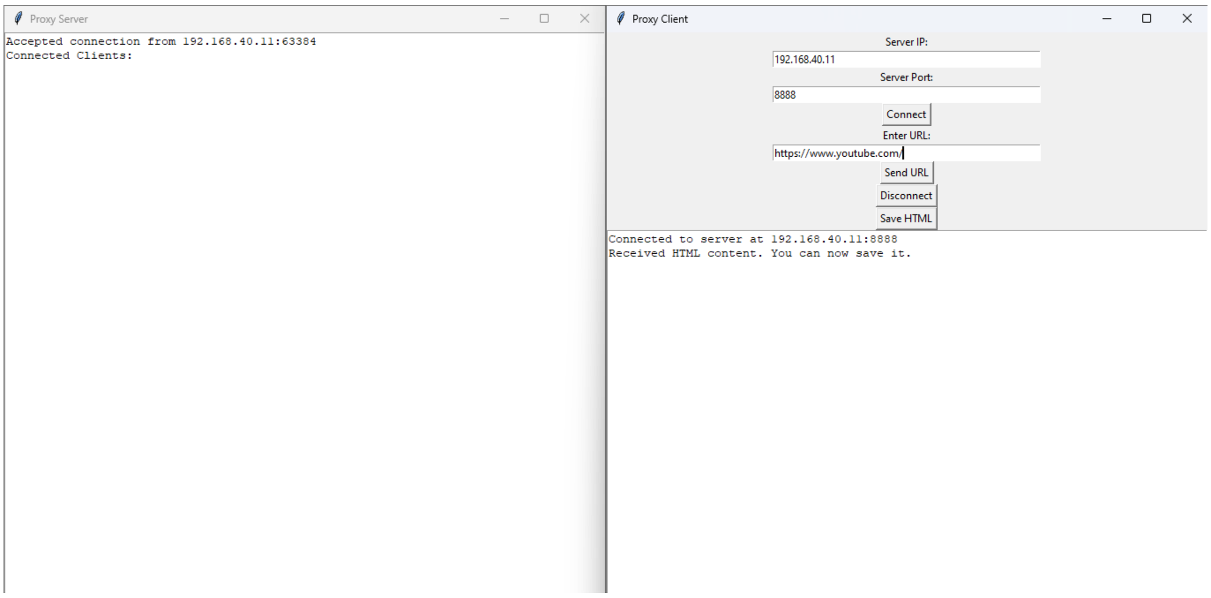


```
Accepted connection from 192.168.40.11:63384
Connected Clients:
192.168.40.11:63384
```

Server IP: 192.168.40.11

Server Port: 8888

```
Connected to server at 192.168.40.11:8888
```



Proxy Server

```
Accepted connection from 192.168.40.11:63384
Connected Clients:
```

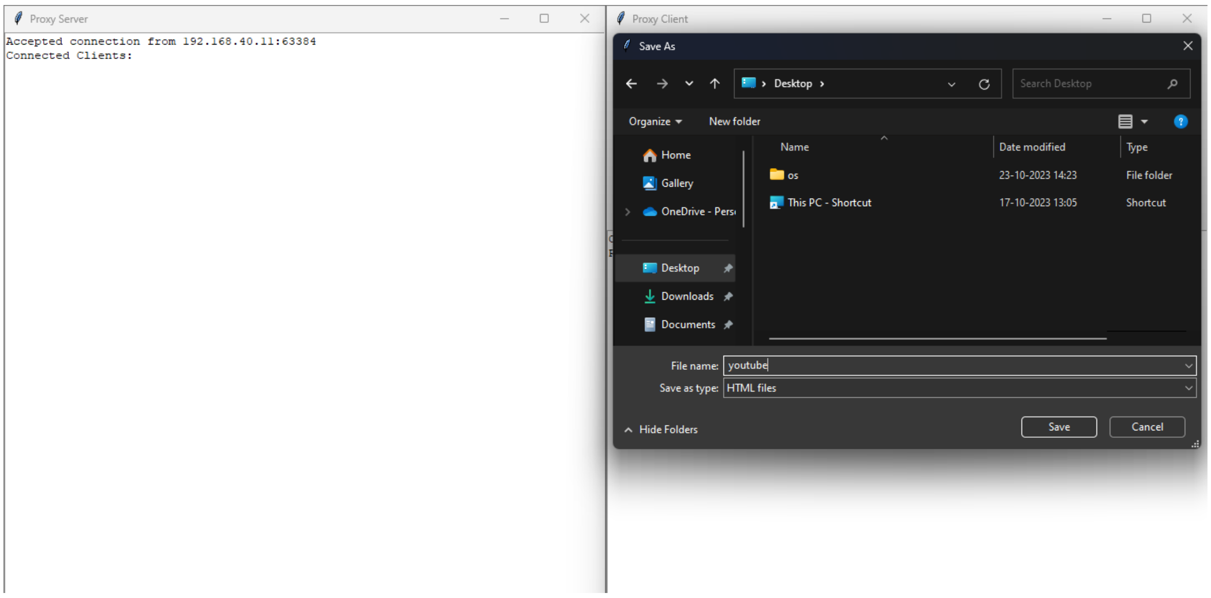
Proxy Client

Server IP: 192.168.40.11

Server Port: 8888

Enter URL: https://www.youtube.com/

```
Connected to server at 192.168.40.11:8888
Received HTML content. You can now save it.
```



Proxy Server

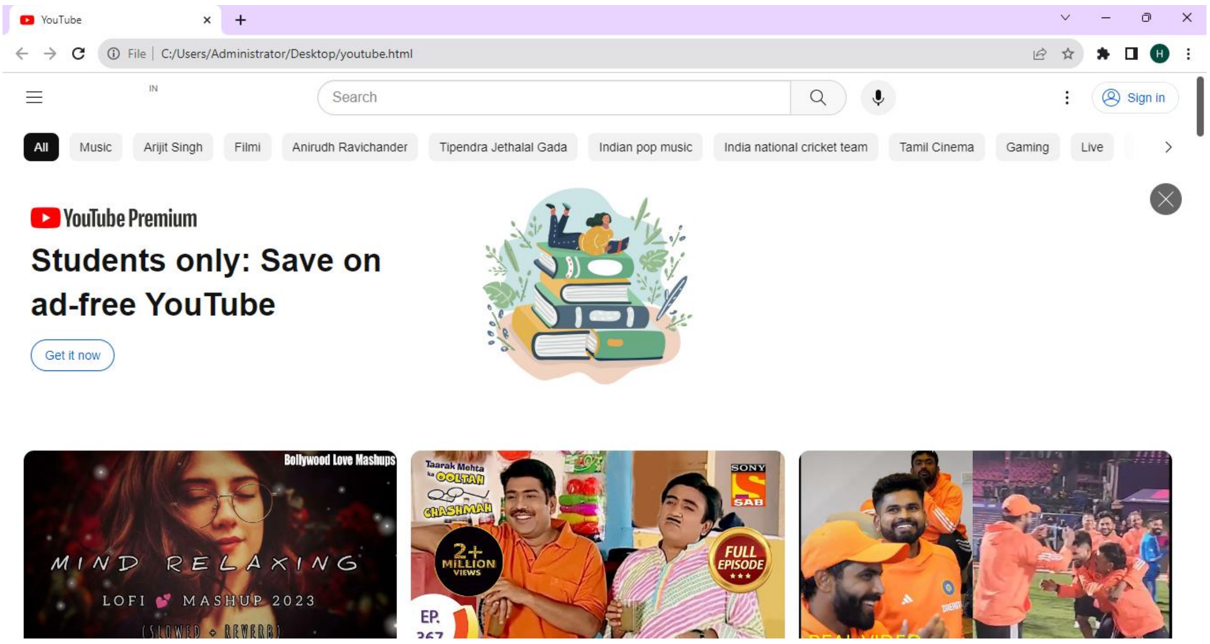
```
Accepted connection from 192.168.40.11:63384
Connected Clients:
```

Proxy Client — Save As

Name	Date modified	Type
os	23-10-2023 14:23	File folder
This PC - Shortcut	17-10-2023 13:05	Shortcut

File name: youtube

Save as type: HTML files


YouTube
File | C:/Users/Administrator/Desktop/youtube.html
IN
Search
Sign in
All
Music
Arijit Singh
Filmi
Anirudh Ravichander
Tipendra Jethalal Gada
Indian pop music
India national cricket team
Tamil Cinema
Gaming
Live
YouTube Premium
Students only: Save on ad-free YouTube
Get it now
Bollywood Love Mashups
MIND RELAXING
LOFI MASHUP 2023
Taarak Mehta ka Ooltah Chashmah
2+ MILLION VIEWS
FULL EPISODE
SONY SAB
EP.

Conclusion

In summary, the multi-threaded proxy server and client project serves as a practical solution for web content retrieval and storage, benefiting users who need to access and organize web content from various sources. This project showcases the power of multi-threading, networking, and error handling in building a reliable and efficient system. It is important to keep in mind that the success of the project depends on thorough testing, clear documentation, and robust implementation that meets the specific needs of the intended users.

References

1- Python Documentation
2- Multi-Threading References
3- Web Development and HTML
4- Online Tutorials and Forums
5- Networking and Security Resources
6- GitHub
7- ChatGPT