A logo of a university

Description automatically generated PATUAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

**Project Report On: ‘Web Server’**

**Course Code: CCE-314**

**Course Title: Computer Networks Sessional**

Submitted to:

**Md. Samsuzzaman**

Professor

Department of Computer and Communication Engineering

Faculty of Computer Science and Engineering

&

**Swarna Majumdar**

Associate Professor

Department of Computer and Communication Engineering

Faculty of Computer Science and Engineering

Submitted by:

**Tasnim Ahammed Sohan**

ID No: 2002061

Registration No: 09578

Level-3, Semester-1, Session: 2020-21

Faculty of Computer Science and Engineering

**Date of Submission: 26.11.2024**

**Table of Index**

**Abstract…………………………………………………………1**

**Introduction…………………………………………………… 2**

**Objective of the project…………………………………. 3**

**Technology used………………………………………… 4**

**How its Work……………………………………………. 5**

**Result……………………………………………………..6**

**References………………………………………………..8**

**Conclusion………………………………………………..9**

**Abstract**

This project aims to develop a basic yet functional web server using Python that handles HTTP requests and serves static files to clients. By leveraging Python's built-in socket programming, the server will demonstrate how fundamental networking concepts are applied in web technologies. The server will listen for incoming HTTP requests, parse them, and respond with the appropriate resources, such as HTML, CSS, or images.

The project involves key components such as request handling, response generation, static file serving, and error management. It will support common HTTP methods like GET and generate appropriate response codes (e.g., 200 OK, 404 Not Found). The server will map incoming URLs to files in a designated directory, ensuring smooth access to web content.

Optional enhancements include concurrency for handling multiple client connections, basic security features like input validation, and logging for tracking server activities. The server design is modular and can be extended to serve dynamic content or implement advanced security measures.

This project serves as a practical introduction to HTTP, socket programming, and file handling, equipping the developer with foundational knowledge of web server operations.

**Introduction**

**Web servers form the backbone of the internet, responsible for handling client requests and delivering the appropriate web content. This project, "Simple Web Server in Python," aims to demonstrate the essential concepts behind web server functionality by building a minimal yet fully functional server using Python. The project focuses on using socket**

**programming to establish communication between the server and clients over a network, simulating the operation of larger, more complex web servers.**

**The web server will listen on a specified port and handle HTTP requests from clients, such as web browsers, returning static resources like HTML, CSS, and image files. It will support the GET method, parse requests to extract relevant information (such as requested resources and headers), and generate appropriate HTTP responses, including status codes like 200 OK and 404 Not Found.**

**Beyond basic operations, the project provides opportunities to explore error handling and concurrency, ensuring the server can manage multiple client requests either through**

**multithreading or asynchronous programming. Additionally, the web server can be extended to include features such as logging, input validation, and dynamic content generation.**

**This project serves as both a practical introduction to network programming and an educational exercise in understanding how web servers operate. By the end of the implementation, the developer will gain valuable experience with HTTP protocols, socket communication, file handling, and server design, laying the groundwork for more advanced web development projects in the future**

**Objective of the project**

**The goal of this project is to create a basic web server using Python that can handle HTTP**

**requests. The server will be capable of serving static files, such as HTML, CSS, and images, to clients. It will listen for incoming HTTP requests, parse them, and send back the appropriate response.**

**Technology Used**

1. **Socket Programming**

* **Technology: Python's built-in socket module.**
* **Purpose: This module is used to create a communication channel between the server and clients over TCP/IP.**
* **Details:**
* **The server creates a socket with socket. socket (socket. AF\_INET, socket.sock \_ STREAM), indicating that it will use IPv4 and TCP.**
* **The server binds to a specific IP address (0.0.0.0, which means it listens on all available network interfaces) and port (8080)**
* **It listens for incoming connections with server\_socket.listen (5).**

1. **HTTP Protocol**

* **Technology: Implementation of the HTTP/1.1 protocol.**
* **Purpose: The server processes HTTP requests from clients and generates appropriate HTTP responses.**
* **Details:**
* **The server handles GET requests for serving static files (like index.html and book. json).**
* **It responds with appropriate HTTP status codes, such as 200 OK and 404 Not Found, based on whether the requested resource exists.**
* **The server also handles POST requests for saving data.**

1. **File Handling**

* **Technology: Python’s built-in file handling operations.**
* **Purpose: To read static content (HTML and JSON files) and write data to a text file**
* **Details:**
* **The server uses open() to read files like index.html and book. json.**
* **It appends data to data.txt when a POST request is received**

1. **JSON (JavaScript Object Notation)**

* **Technology: JSON format for data interchange.**
* **Purpose: To serve structured data in a standardized format.**
* **Details:**
* **The server serves JSON data from book.json with the content type set to application/json.**

1. **Basic Error Handling**

* **Technology: HTTP error handling**
* **Purpose: To handle cases where a requested resource is not found or when unsupported methods are used.**
* **Details:**
* **The server returns 404 Not Found for nonexistent URLs and 405 Method Not Allowed for unsupported HTTP methods**

1. **Encoding and Decoding of Data**

* **Technology: UTF-8 encoding.**
* **Purpose: To handle the transmission of HTTP messages**
* **Details:**
* **The server decodes incoming request data from bytes to a string using.decode()**
* **Responses are encoded into bytes before sending them back to the client with .encode().**

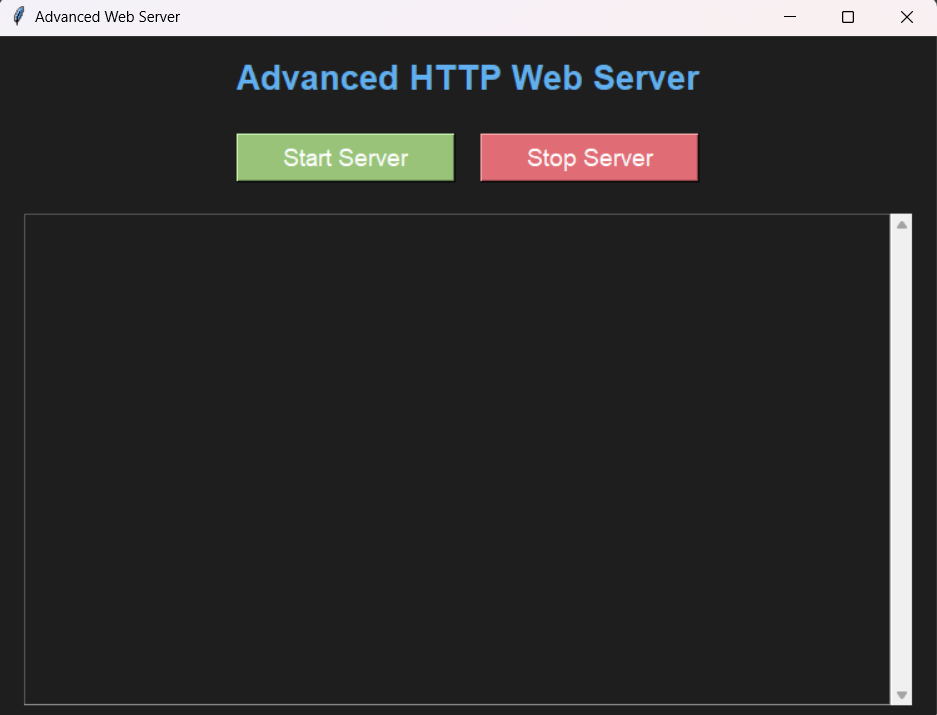
1. **Concurrency (Optional Feature)**

* **While the current implementation does not support concurrency, it can be enhanced to handle multiple client connections simultaneously using:**
* **Threading: By creating a new thread for each client connection**
* **Asynchronous Programming: Using frameworks like asyncio for non-blocking I/O operations.**

**How it’s work**

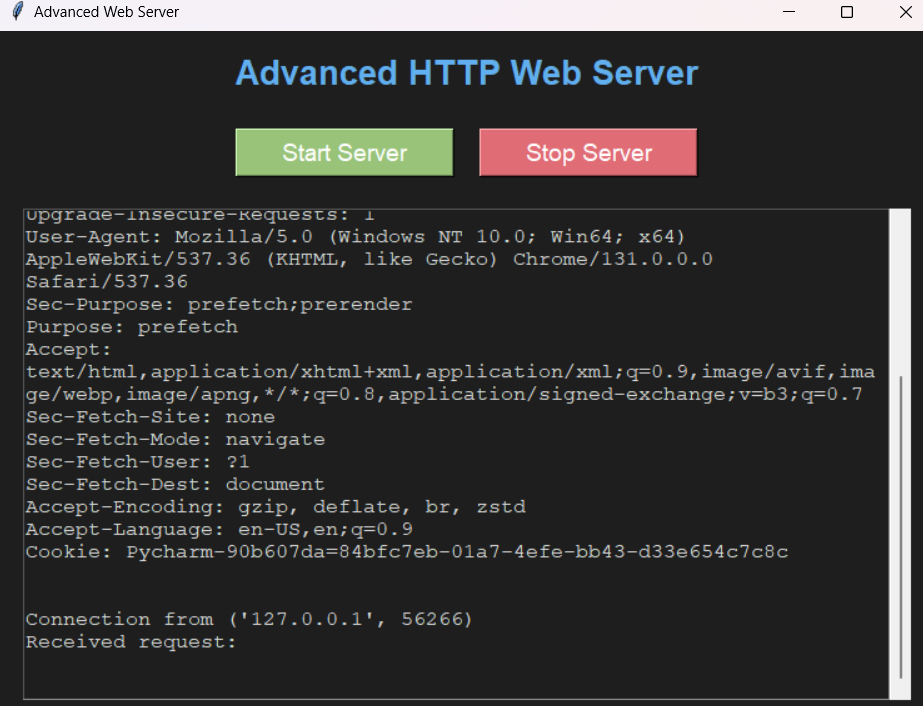
1. Server Initialization:

* The server is initialized when the user starts it through the GUI.
* The server creates a socket using the socket library and binds it to a specified host (0.0.0.0 for all interfaces) and port (9090).
* The server begins listening for incoming client connections on the specified port. The server uses server\_socket.listen(5) to listen for up to 5 simultaneous connections



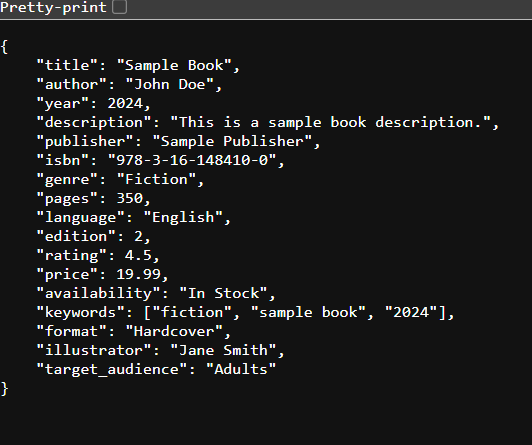
**2. Accepting Client Connection:**

* **The server continuously accepts incoming client connections in a separate thread using the accept\_clients function.**
* **When a client connects, the server creates a new socket dedicated to that client and starts a new thread (threading.Thread) to handle the client request via the handle\_client function.**
* **The accept\_clients function allows the server to handle multiple clients concurrently, without blocking, by delegating the actual request handling to separate threads**



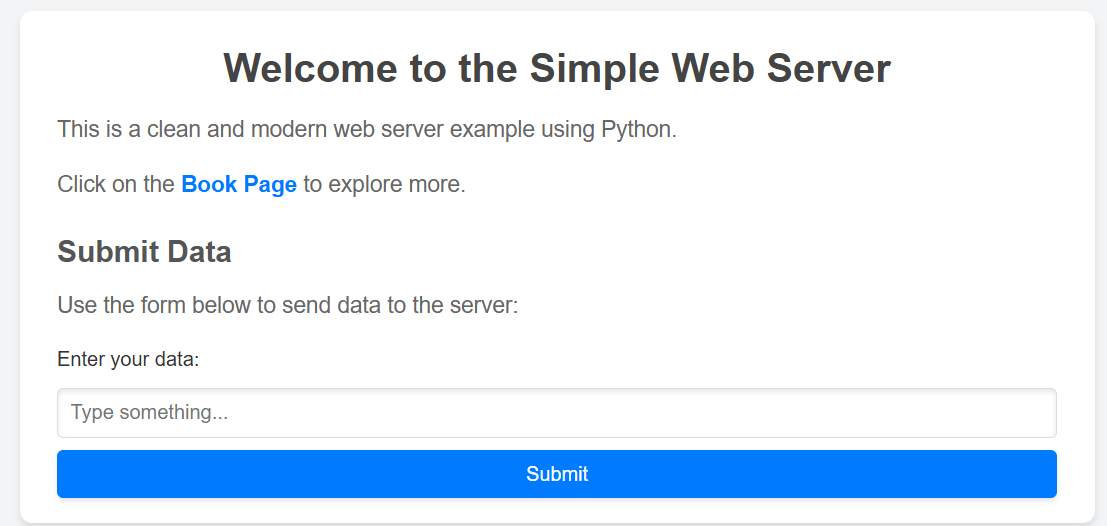
**3.Handle Client Request:**

* **Each time a client sends a request, the handle\_client function is executed. The request is received by the server through the client\_socket.recv(1500) call, where 1500 is the maximum number of bytes the server expects in a request**
* **The request is then decoded and split into HTTP headers. The first line contains the HTTP method (e.g., GET, POST) and the requested path (e.g., /, /book).**



**4.Processing Get Request:**

* **If the request is a GET method, the server checks the requested path**
* **If the path is /, the server serves the index.html file.**
* **If the path is /book, the server serves the book.json file.**
* **If the file exists, the server reads its contents and responds with the file's content and the appropriate HTTP headers (e.g., 200 OK, Content-Type: text/html or application/json).**
* **If the file does not exist, the server responds with a 404 Not Found error.**



**5.Processing Post Requests:**

* **If the request is a POST method, the server checks if the path is /save**
* **If it is, the server extracts the body of the request and saves the data to a file (data.txt).**
* **After saving the data, the server sends a 200 OK response with a message confirming that the data was saved.**
* **If the path is not /save, the server responds with a 404 Not Found error**

****

**6. Error Handling:**

* **If the server receives a request with an unsupported HTTP method (other than GET or POST), it responds with a 405 Method Not Allowed error, listing the allowed methods (e.g., GET, POST).**
* **If any other errors occur (e.g., invalid request formatting, file read/write errors), the server catches them and logs the error in the GUI**

Result

**The result of this project is a functional Python-based web server that listens on a specified port, handles HTTP requests, and serves static files like HTML and JSON. It supports GET and POST methods, processes requests for resources such as /book for JSON content, and provides downloadable files through URLs like /download/book. json. The server also handles errors gracefully, returning appropriate status codes such as 404 for missing resources and 405 for unsupported methods. Additionally, the GUI interface enables easy control to start and stop the server while displaying real-time logs of client requests**

**Refrencences**

* Python Documentation: <https://docs.python.org/3/>
* Tkinter Documentation: <https://docs.python.org/3/library/tkinter.html>
* Web browser Module Documentation: <https://docs.python.org/3/library/webbrowser.html>

Conclusion

In conclusion, this project successfully demonstrates how to build a simple web server in Python using socket programming. It handles HTTP requests, serves static and downloadable files, and supports basic GET and POST operations. With its error handling, logging, and a GUI for easy control, the server provides a foundational understanding of networking concepts and HTTP communication. This project can be further extended with features like security, threading, or dynamic content to enhance its functionality.