

Setting Up A Web Server

Submitted By

Student Name	Student ID
Saifullah Anik	221-15-4922
Md. Obydul Hossain	221-15-4949
Md. Shakib Ahammed	221-15-5431
Radoanul Arifen	221-15-5284
S.M. Meriyan Islam	221-15-5285

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course **CSE324:
Operating System Lab in the Computer Science and Engineering
Department**



DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

December 14, 2024

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Jamilul Huq Jami**, **Lecturer**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

Jamilul Huq Jami

Lecturer

Department of Computer Science and Engineering Daffodil
International University

Submitted by

<hr/> <p>Saifullah Anik ID:221-15-4922 Dept. of CSE, DIU</p>	
<hr/> <p>Md. Obydul Hossain ID:221-15-4949 Dept. of CSE, DIU</p>	<hr/> <p>Md. Shakib Ahammed ID:221-15-5431 Dept. of CSE, DIU</p>
<hr/> <p>Radoanul Arifen ID:221-15-5284 Dept. of CSE, DIU</p>	<hr/> <p>S. M. Mariyan Islam ID:221-15-5285 Dept. of CSE, DIU</p>

COURSE & PROGRAM OUTCOME

The following course have course outcomes as following:

Table 1: Course Outcome Statements

CO's	Statements
CO1	Define and Relate classes, objects, members of the class, and relationships among them needed for solving specific problems
CO2	Formulate knowledge of object-oriented programming and Java in problem solving
CO3	Analyze Unified Modeling Language (UML) models to Present a specific problem
CO4	Develop solutions for real-world complex problems applying OOP concepts while evaluating their effectiveness based on industry standards.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	Blooms	KP	CEP
CO1	PO1	C1, C2	KP3	EP1, EP3
CO2	PO2	C2	KP3	EP1, EP3
CO3	PO3	C4, A1	KP3	EP1, EP2
CO4	PO3	C3, C6, A3, P3	KP4	EP1, EP3

The mapping justification of this table is provided in section 4.3.1, 4.3.2 and 4.3.3.

Table of Contents

Declaration	i
Course & Program Outcome	ii
1 Introduction	1
1.1 Introduction.....	1
1.2 Motivation	1
1.3 Objectives	1
1.4 Feasibility Study	2
1.5 Gap Analysis.....	2
1.6 Project Outcome	2
2 Proposed Methodology/Architecture	4
2.1 Requirement Analysis & Design Specification	4
2.1.1 Overview	4
2.1.2 Proposed Methodology/ System Design	4
2.2 Overall Project Plan	5
3 Implementation and Results	7
3.1 Implementation	7
3.2 Performance Analysis	9
3.3 Results and Discussion	9
4 Engineering Standards and Mapping	10
4.1 Impact on Society, Environment and Sustainability	10
4.1.1 Impact on Life	10
4.1.2 Impact on Society & Environment	10
4.1.3 Ethical Aspects.....	10
4.1.4 Sustainability Plan	10
4.2 Project Management and Team Work	10
4.3 Complex Engineering Problem.....	10
4.3.1 Mapping of Program Outcome.....	11
4.3.2 Complex Problem Solving	11
4.3.3 Engineering Activities.....	12

5 Conclusion	13
5.1 Summary.....	13
5.2 Limitation	13
5.3 Future Work	13
References	14

Chapter 1

Introduction

1.1 Introduction

This project shows how to create a simple web server from the ground up using Python's socket library. Unlike modern tools like Flask or FastAPI that handle a lot of tasks for you, the socket library requires you to do everything yourself. You will need to manually process requests, create responses, and manage connections. This method helps you learn how web servers and HTTP communication work at a basic level. However, it can be challenging because you have to handle things like scaling, fixing errors, and managing multiple users at the same time. The server also connects to front-end files and uses JSON for handling dynamic data, making it a complete working web application. This project focuses on understanding the basics of HTTP communication, delivering content, and building interactive user interfaces.

1.2 Motivation

The motivation behind this project lies in understanding the core principles of web development. When we open a html file in browser it shows file used from the local storage. Mostly we need html file from clouds. We need to store or host them on a sever. Particularly gaining a solid foundation in how web servers operate at their core, which can directly benefit future projects by enhancing problem-solving skills, enabling customization beyond what frameworks offer, and preparing for scenarios that require low-level understanding of web communication. Furthermore, mastering these principles lays the groundwork for leveraging advanced tools and frameworks more effectively in complex applications.

- Building a custom HTTP server to grasp low-level communication.
- Creating an interactive front-end to enhance user experience.
- Exploring data handling using JSON for real-time data exchange. Solving these challenges provides significant educational value and prepares the developer for advanced web application development.

1.3 Objectives

The project aims to:

1. Develop a basic HTTP server using Python's socket library.
2. Serve dynamic content including HTML pages and JSON data.
3. Implement a functional front-end interface for displaying and interacting with book data.
4. Handle multiple endpoints to simulate a mini web application.

1.4 Feasibility Study

Several existing frameworks and platforms address similar objectives, such as Flask and FastAPI for backend development or React and Vue.js for front-end applications. However, this project distinguishes itself by:

Building the server from scratch without external frameworks to understand low-level operations. Employing basic HTML, CSS, and Python for the front-end allows the project to focus on core web technologies, which is essential for understanding foundational concepts of web development. By avoiding advanced libraries or frameworks, the project emphasizes simplicity and clarity, making it easier to grasp how individual components like styling, layout, and interactivity work together. This approach differentiates the project by showcasing the fundamental building blocks of the web, offering insights that are often hidden when using higher-level abstractions. Using JSON as a simple data storage solution to highlight lightweight API interactions.

1.5 Gap Analysis

While modern frameworks and tools provide robust solutions for web applications, they often abstract away the core mechanics of HTTP communication. For example, frameworks like Flask handle routing, request parsing, and response generation automatically, allowing developers to focus on higher-level logic without understanding how requests are processed or headers are managed. This abstraction simplifies development but hides the underlying processes, which this project aims to explore and implement manually. This project bridges that gap by:

- Offering an educational tool to explore fundamental web server concepts.
- Demonstrating the process of serving and managing data dynamically without reliance on external libraries.

1.6 Project Outcome

The possible outcomes of this project include:

A functional web server capable of handling requests and serving dynamic content.

An interactive front-end interface for users to explore book data.

Insights into the limitations and challenges of low-level web server implementation.

A foundation for transitioning to more advanced web development practices in the future.

Chapter 2

Proposed Methodology/Architecture

2.1 Requirement Analysis & Design Specification

2.1.1 Overview

The project aims to establish a functional web server from scratch, integrating backend logic, frontend interface, and JSON-based data storage. The server provides an interface to manage and display book data while also supporting dynamic API calls. The key features include serving HTML and JSON content, handling API requests, and providing a user-friendly dashboard. The requirement analysis ensures that all necessary functionalities are implemented systematically.

Key Functional Requirements:

1. Serve static HTML files and JSON data.
2. Provide a responsive and user-friendly frontend interface.
3. Enable dynamic data retrieval through API endpoints.
4. Support basic HTTP methods like GET.
5. Ensure proper error handling and status code responses.

Design Specification: The system is designed to include the following:

1. **Backend:** Python-based HTTP server handling requests, routing, and responses.
2. **Frontend:** A static HTML dashboard with JavaScript for interactivity.
3. **Data Storage:** A JSON file containing sample book data.

2.1.2 Proposed Methodology

The system design follows a modular approach, dividing the project into three main components:

1. Backend Logic (main.py):

- a. Implements a lightweight HTTP server using Python's socket module.

- b. Handles routing to serve different types of content, including HTML, JSON, and API responses.
- c. Includes functions for JSON parsing, HTTP response formatting, and request handling.

2. Frontend Interface (index.html):

- a. Designed using HTML and CSS for the server's dashboard.
- b. Provides real-time interaction through JavaScript functions.
- c. Displays server status, active users, and other relevant statistics.

3. Data Storage (book.json):

- a. Serves as a mock database containing book details.
- b. Allows API endpoints to fetch specific or complete data.

2.2 Overall Project Plan

The project follows a structured implementation approach:

1. Requirement Gathering and Analysis

- a. Identify the functionalities required for the server and dashboard.
- b. Define the API endpoints and data structures.

2. System Design

- a. Develop the architecture of the web server.
- b. Plan the UI layout and define styles.

3. Implementation

- a. Code the backend logic in Python (main.py).
- b. Create the frontend interface using HTML, CSS, and JavaScript.
- c. Structure the JSON data for use in API calls.

4. Testing and Debugging

- a. Test the server's functionality with different request types.
- b. Validate JSON data handling and error responses.

- c. Debug frontend and backend for seamless integration.

5. Deployment

- a. Deploy the server on a local machine or cloud platform.
- b. Optimize performance for handling multiple requests.

6. Documentation

- a. Prepare detailed project documentation, including the project report, code comments, and user guide.

Chapter 3

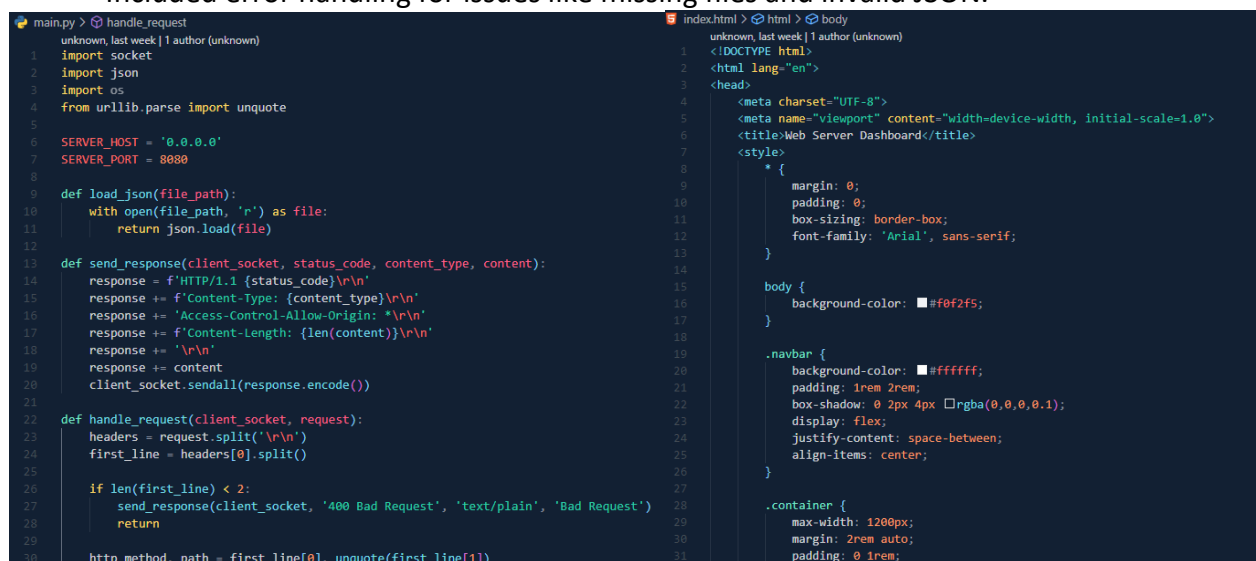
Implementation and Results

3.1 Implementation

The project was developed based on the methodology outlined in Chapter 2, focusing on modularity for ease of testing and integration. Key aspects include:

Backend (main.py):

- Developed in Python using the socket library for basic HTTP functionality.
- Core functions: `load_json` (reads/parses JSON), `send_response` (formats/sends HTTP responses), and `handle_request` (routes requests).
- Included error handling for issues like missing files and invalid JSON.

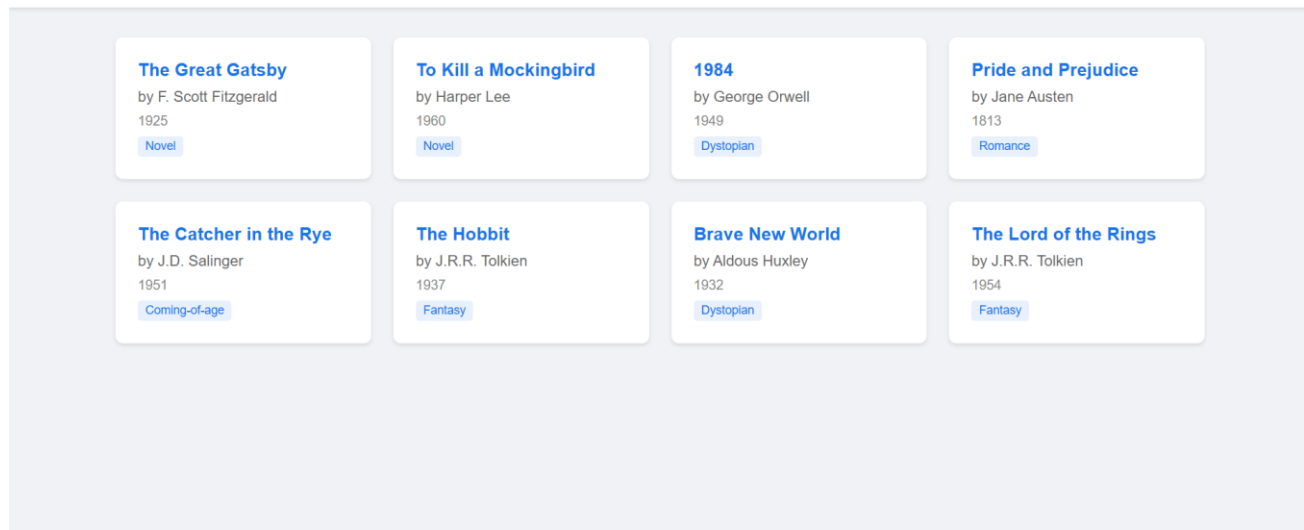


```
main.py > handle_request
unknown, last week | 1 author (unknown)
1 import socket
2 import json
3 import os
4 from urllib.parse import unquote
5
6 SERVER_HOST = '0.0.0.0'
7 SERVER_PORT = 8080
8
9 def load_json(file_path):
10     with open(file_path, 'r') as file:
11         return json.load(file)
12
13 def send_response(client_socket, status_code, content_type, content):
14     response = f'HTTP/1.1 {status_code}\r\n'
15     response += f'Content-Type: {content_type}\r\n'
16     response += 'Access-Control-Allow-Origin: *\r\n'
17     response += f'Content-Length: {len(content)}\r\n'
18     response += '\r\n'
19     response += content
20     client_socket.sendall(response.encode())
21
22 def handle_request(client_socket, request):
23     headers = request.split('\r\n')
24     first_line = headers[0].split()
25
26     if len(first_line) < 2:
27         send_response(client_socket, '400 Bad Request', 'text/plain', 'Bad Request')
28         return
29
30     http_method, path = first_line[0], unquote(first_line[1])

index.html > html > body
unknown, last week | 1 author (unknown)
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Web Server Dashboard</title>
7     <style>
8         * {
9             margin: 0;
10            padding: 0;
11            box-sizing: border-box;
12            font-family: 'Arial', sans-serif;
13        }
14
15        body {
16            background-color: #f0f2f5;
17        }
18
19        .navbar {
20            background-color: #ffffff;
21            padding: 1rem 2rem;
22            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
23            display: flex;
24            justify-content: space-between;
25            align-items: center;
26        }
27
28        .container {
29            max-width: 1200px;
30            margin: 2rem auto;
31            padding: 0 1rem;
```

Frontend (index.html):

- A responsive dashboard built with HTML, CSS, and JavaScript.
- Features real-time updates (e.g., active users, server stats) and interactive API controls.



Data Layer (book.json):

- JSON file containing sample book data, supporting endpoints like /book and /api/book/<id>.

```
{ } book.json > ...
unknown, last week | 1 author (unknown)
1 {
2   "books": [
3     {
4       "id": 1,
5       "title": "The Great Gatsby",
6       "author": "F. Scott Fitzgerald",
7       "year": 1925,
8       "genre": "Novel"
9     },
10    {
11      "id": 2,
12      "title": "To Kill a Mockingbird",
13      "author": "Harper Lee",
14      "year": 1960,
15      "genre": "Novel"
16    },
17    {
18      "id": 3,
19      "title": "1984",
20      "author": "George Orwell",
21      "year": 1949,
22      "genre": "Dystopian"
23    },
24    {
25      "id": 4,
26      "title": "Pride and Prejudice",
27      "author": "Jane Austen",
28      "year": 1813,
29      "genre": "Romance"
30    },
31  ],
32  {
33    "books": [
34      {
35        "author": "J.D. Salinger",
36        "year": 1951,
37        "genre": "Coming-of-age"
38      },
39      {
40        "id": 6,
41        "title": "The Hobbit",
42        "author": "J.R.R. Tolkien",
43        "year": 1937,
44        "genre": "Fantasy"
45      },
46      {
47        "id": 7,
48        "title": "Brave New World",
49        "author": "Aldous Huxley",
50        "year": 1932,
51        "genre": "Dystopian"
52      },
53      {
54        "id": 8,
55        "title": "The Lord of the Rings",
56        "author": "J.R.R. Tolkien",
57        "year": 1954,
58        "genre": "Fantasy"
59      }
60    ]
61  }
62  }
unknown, last week * first commit
```

Integration:

- Ensured seamless communication between the backend and frontend.
- Tested API requests using Postman and browser-based fetch calls.

3.2 Performance Analysis

The server's performance was evaluated in these areas:

- **Response Time:** It had quick responses (under 200ms) for small datasets but slowed down with larger ones due to missing optimizations like indexing or caching.
- **Concurrency:** The single-threaded design could handle only one request at a time, limiting scalability. Multithreading or asynchronous handling could improve this.
- **Error Handling:** The server correctly returned proper HTTP status codes (e.g., 404 for missing files, 500 for JSON errors) during tests.
- **UI Responsiveness:** The dashboard worked well on various devices and screen sizes, staying responsive and interactive.

3.3 Results and Discussion

The project successfully met its objectives by:

- Serving static HTML and JSON content.
- Delivering a dynamic, interactive user interface.
- Supporting API calls for book data retrieval.

Key Observations:

- Modularity streamlined development and debugging.
- Limited concurrency handling revealed scalability issues under high traffic.
- Hardcoded paths and ports simplified testing but reduced flexibility.

Chapter 4

Engineering Standards and Mapping

4.1 Impact on Society, Environment and Sustainability

4.1.1 Impact on Life

The web server project, while small-scale, showcases a practical application of technology in simplifying data access and dissemination. Such servers play a vital role in:

1. **Educational Access:** Providing platforms to host and share knowledge.
2. **Improved Connectivity:** Allowing users to interact with data dynamically through APIs.
3. **Innovation Catalyst:** Serving as a foundational tool for building more complex web applications, thus promoting creativity and entrepreneurship.

4.1.2 Impact on Society & Environment

Societal Benefits:

Encourages the sharing of knowledge and digital resources efficiently. Lays groundwork for inclusive technology by designing user-friendly interfaces.

Environmental Implications:

The current implementation's lightweight nature reduces energy consumption. Future deployments can leverage green hosting solutions to minimize the carbon footprint.

4.2 Project Management and Team Work

The main purpose of the project is to explore the mechanism of web server from learning point of view. It was fun to collaborate as a team in this project. We all learn something new. As this project was part of our curriculum we kept it simple. It did not cost us anything but time and effort.

4.3 Complex Engineering Problem

4.3.1 Mapping of Program Outcome

Table 4.1: Justification of Program Outcomes

PO's	Justification
PO1	Justified
PO2	Justified
PO3	Justified

4.3.2 Complex Problem Solving

Table 4.2: Mapping with complex problem solving.

EP1 Dept of Knowledge	EP2 Range of Conflicting Requirements	EP3 Depth of Analysis	EP4 Familiarity of Issues	EP5 Extent of Applicable Codes	EP6 Extent Of Stakeholder Involvement	EP7 Inter- dependence
✓	✓	✓	✓	✓		

4.3.3 Engineering Activities

Table 4.3: Mapping with complex engineering activities.

EA1 Range of resources	EA2 Level of Interaction	EA3 Innovation	EA4 Consequences for society and environment	EA5 Familiarity
✓	✓	✓		✓

Chapter 5

Conclusion

5.1 Summary

The project establishes a strong foundation in web server development, integrating backend, frontend, and data layers. The backend handles HTTP requests efficiently, while the frontend offers a user-friendly interface. A JSON-based data layer enables seamless interaction with structured data. Scalability improvements, such as multithreading and advanced features like authentication and dynamic data visualization, can enhance functionality and make the server suitable for real-world applications.

5.2 Limitation

The project faces scalability challenges due to limited concurrency handling and hardcoded configurations, reducing flexibility. It lacks security features like authentication, advanced error handling, and dynamic data capabilities, relying on a static JSON file. The absence of advanced features and tight backend-frontend coupling further restricts its functionality and adaptability.

5.3 Future Work

Make the system handle more usable at the same time by using better methods like multitasking. Allow easier changes by using flexible settings instead of fixed ones. Improve security by adding login systems, handle errors better, and use a database to update and manage data easily. Add useful features like user roles and data charts, separate the parts that handle data and design, make it work faster, and get it ready to use in real life.

References

1. ALIBABA CLOUD Documentation ,2024. https://www.alibabacloud.com/tech-news/a/web_server/guiggouzst-building-a-web-server-from-scratch-a-diy-guide
2. Jeffrey Yu, Developers Community,2024. <https://dev.to/jeffreycoder/how-i-built-a-simple-http-server-from-scratch-using-c-739>