RAMAPURAM
SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

21CSC302J-Computer Networks

Batch No - 16

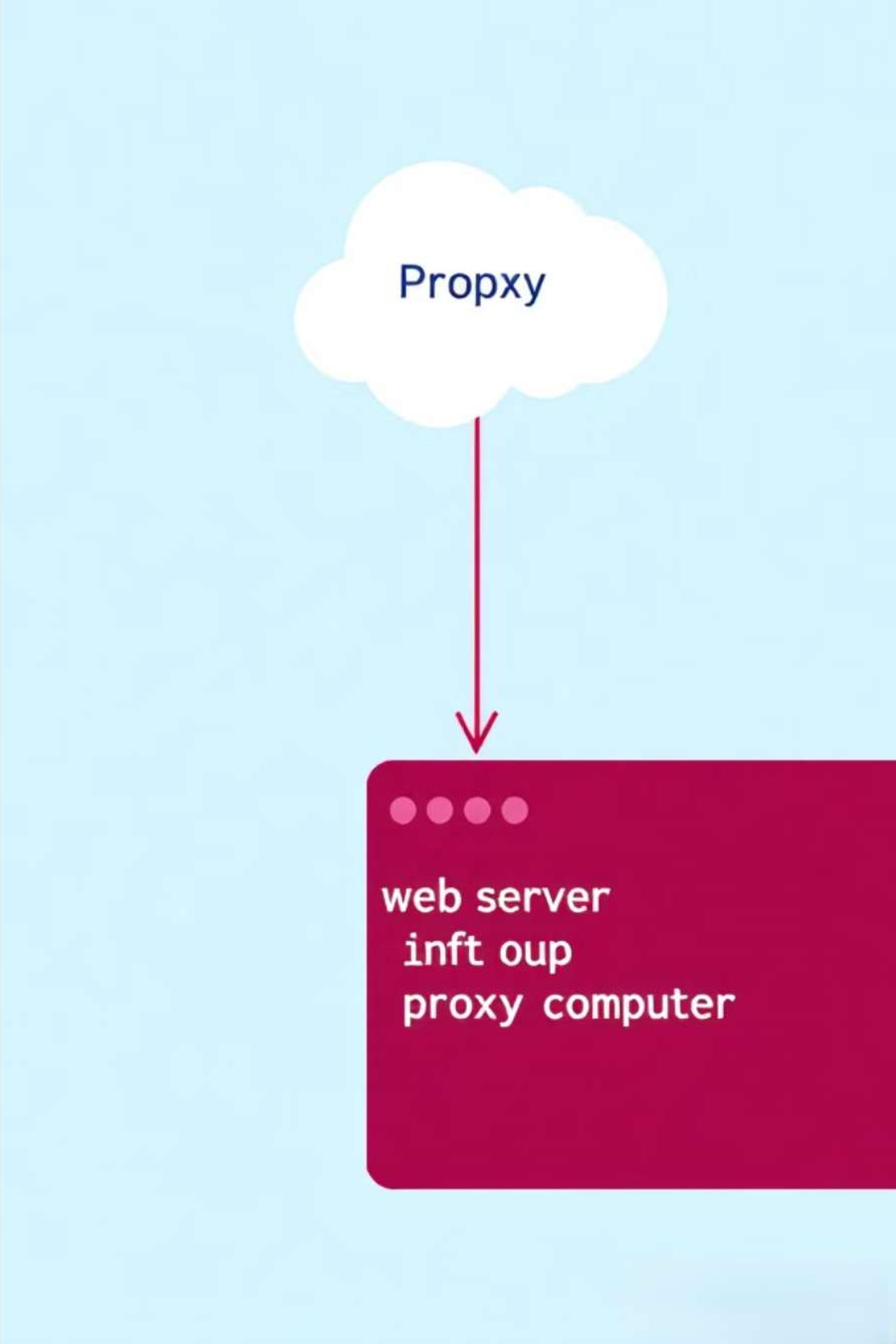
Team Members	Supervisor
Sarathi V (RA2211042020026)	Dr. G. Gangadevi, M. Tech, Ph.D.,
Aribalan I (RA2211042020028)	Assistant Professor
Shakthivel K (RA2211042020060)	Department of Computer Science of Engineering
Manyam V Avinash (RA2211042020069)	

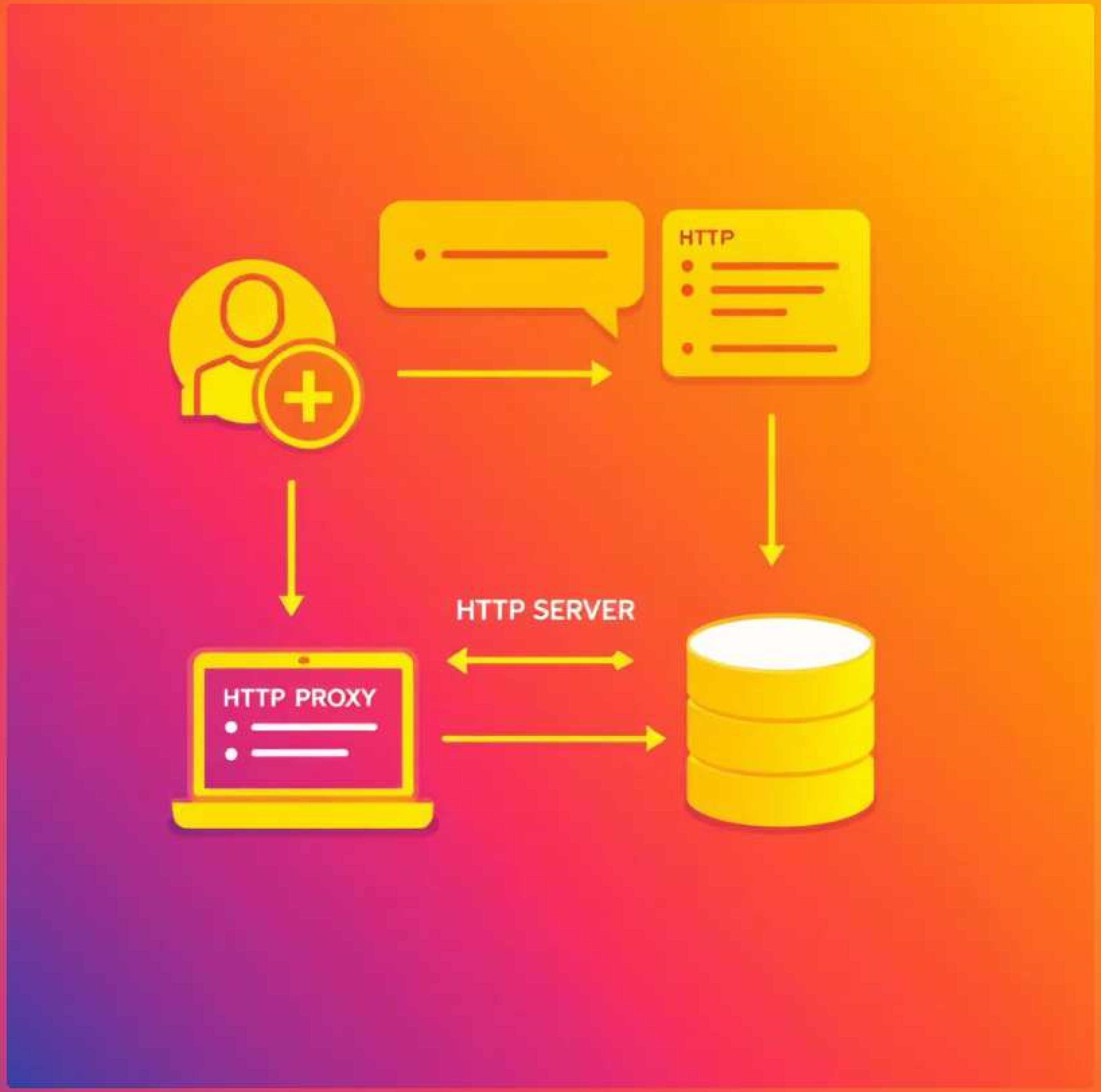
Agenda

1. Introduction
2. Literature Survey
3. Objective
4. Problem Statement
5. Proposed Work
6. Software & Hardware Requirements
7. Implementation
8. Conclusion
9. Future Work
10. References

Abstract

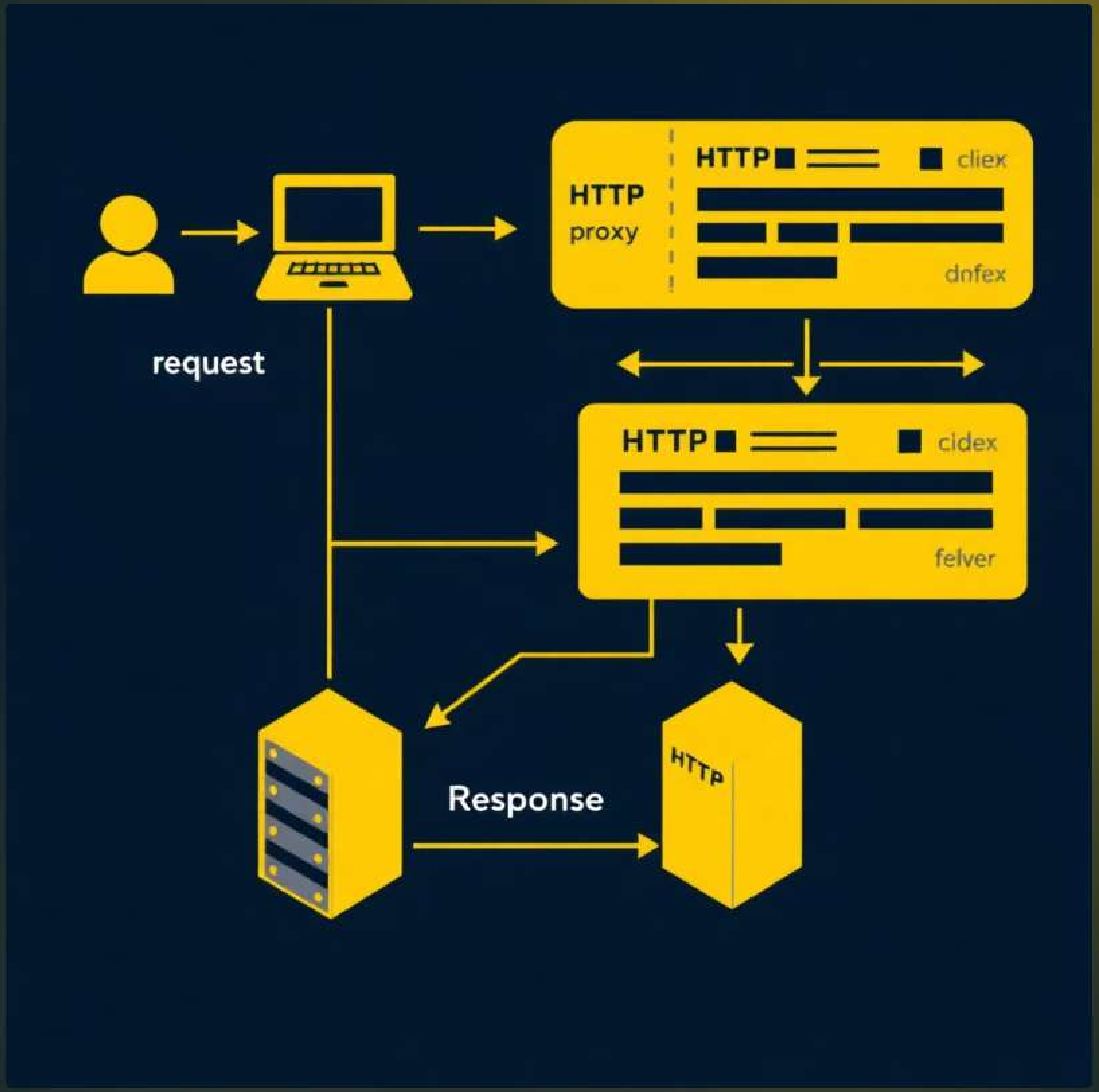
This project presents the development of a simple HTTP proxy server using Python. The proxy acts as an intermediary between clients and a destination server, facilitating the forwarding of client requests and server responses. The implementation uses socket programming for network communication and multithreading to handle multiple client connections concurrently. This project highlights core concepts in networking and provides a basic but functional example of how proxy servers operate within the client-server architecture.





Scope and Motivation

This project implements a basic HTTP proxy server that forwards client requests to a destination server and returns the response. The motivation behind this project is to understand the role of proxy servers in managing network traffic and enhancing security in real-world applications.

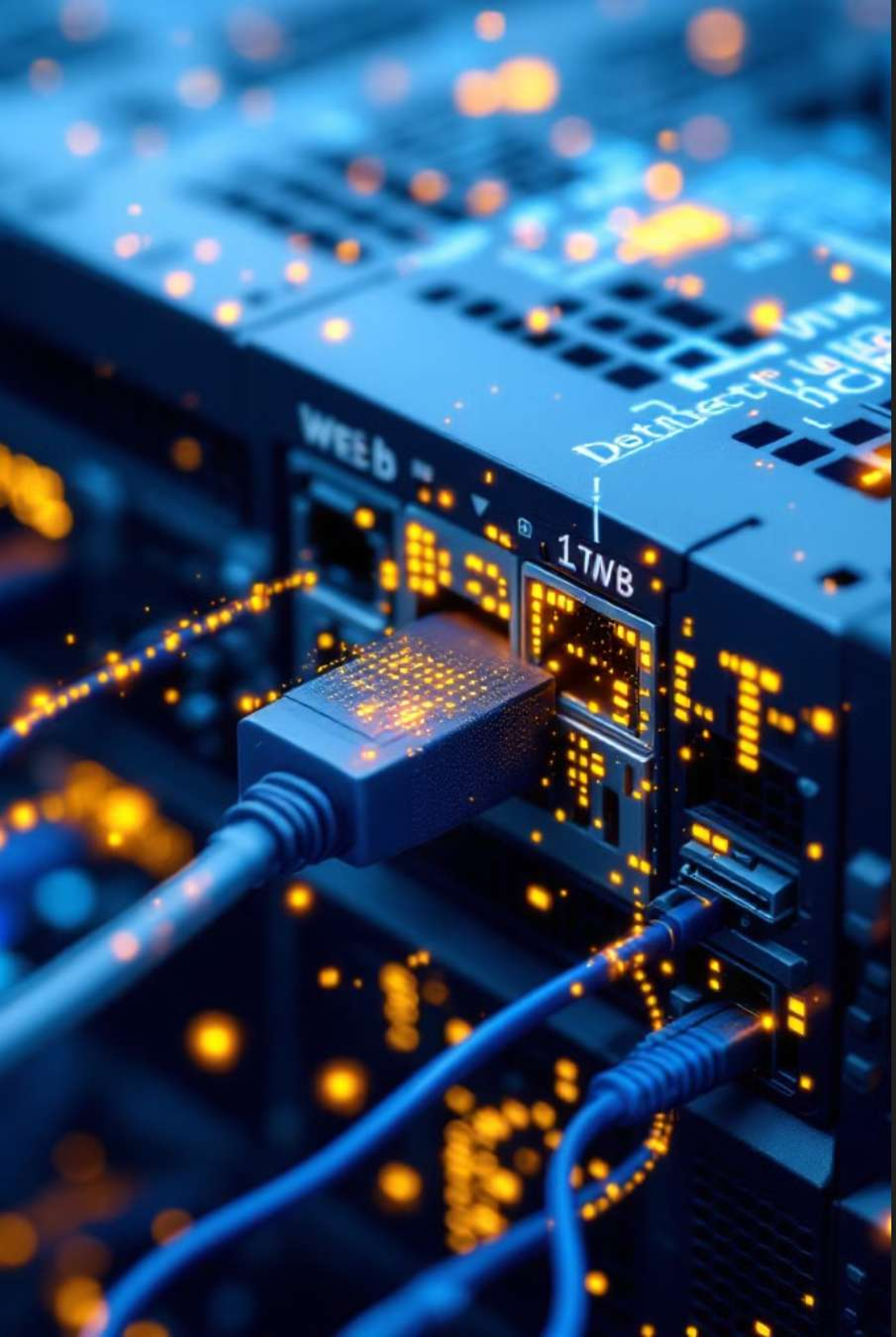


Introduction

This project involves creating a simple HTTP proxy server using Python. The proxy server acts as an intermediary between the client and the destination server, forwarding requests and receiving responses. This concept is widely used for monitoring, filtering, and caching web traffic in various network setups.

Literature Survey

S.No	Title of the Paper	Year	Journal/Conference Name	Inferences
1	"Squid Internet Object Cache"	1996	ACM Conference on USENIX	Explored caching mechanisms to reduce bandwidth consumption and improve response times.
2	"Transparent Proxy Servers: Design and Performance"	1998	IEEE International Conference	Discussed transparent proxy server architecture, improving network transparency.
3	"Performance Enhancement in Proxy Servers for Web Caching"	2001	IEEE Transactions on Networks	Analyzed performance bottlenecks in proxy servers and proposed caching optimizations.
4	"Security Implications of Proxy Servers"	2005	IEEE Security & Privacy	Investigated security vulnerabilities in proxy servers and proposed security frameworks.
5	"Improving Web Performance through HTTP/2 Proxies"	2016	WWW Conference	Highlighted the benefits of HTTP/2 in reducing latency and improving encryption in proxy communication.

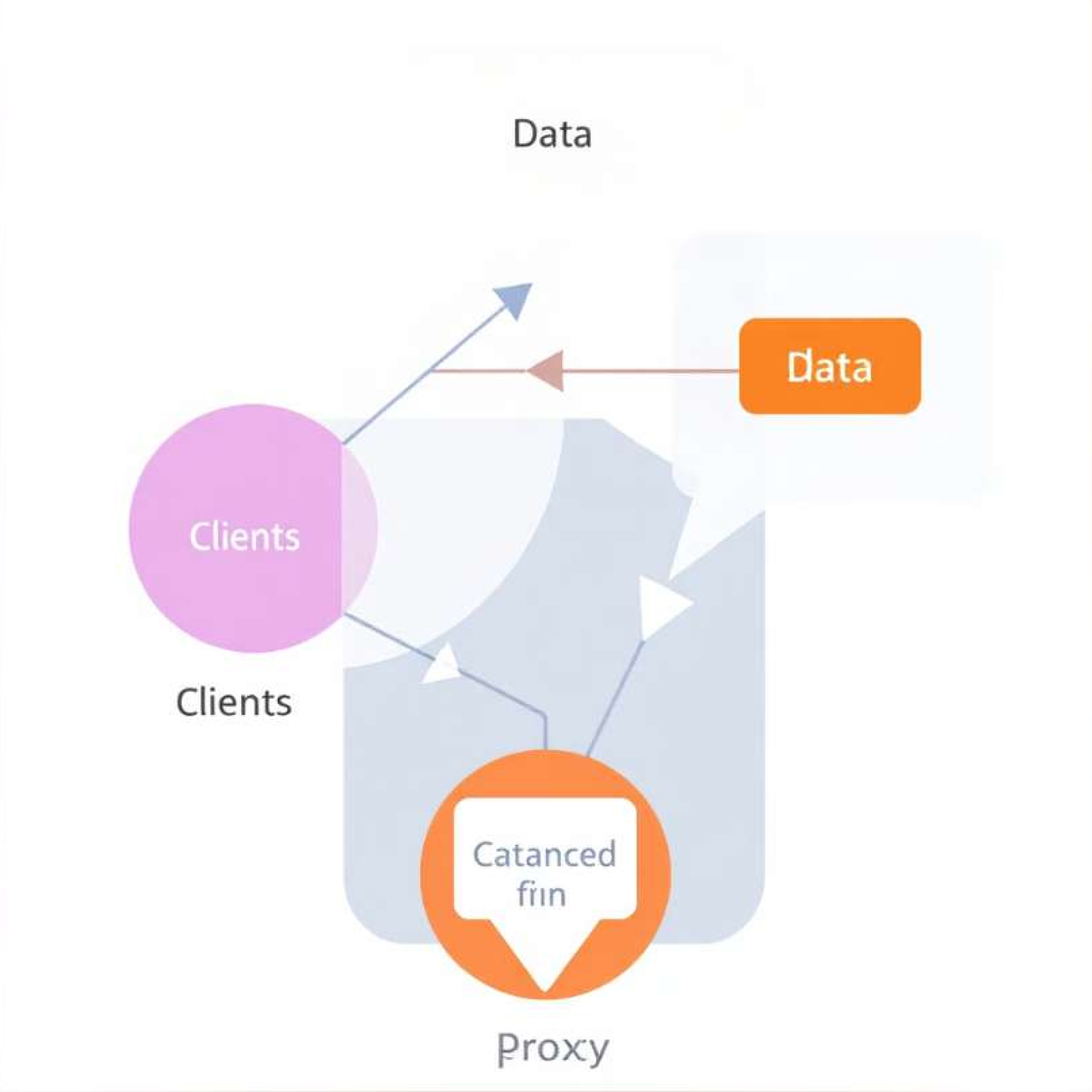


Objective

The objective of this project is to develop a proxy server that efficiently handles client requests, forwards them to the destination server, and relays the responses back to the clients. This will enable monitoring of network traffic, testing of web applications, and potential enhancements in data caching and security. The server will demonstrate key concepts of socket programming, multithreading, and network protocols like HTTP.

Problem Statement

The project aims to address the challenge of intercepting and relaying client-server communication over the network. With growing demands for monitoring, debugging, and securing web traffic, there is a need for a proxy server that can handle multiple client requests, forward them to the target server, and return responses efficiently. The solution will demonstrate effective request routing, response handling, and network communication, essential for testing and monitoring web applications.



Proposed Work

The project aims to develop a Python-based proxy server that intercepts and forwards HTTP requests from clients to target servers. Utilizing multithreading, it will efficiently manage multiple connections, enabling effective monitoring and analysis of network traffic for improved security and performance.

Software & Hardware Requirements

Software

- Python 3.x: Programming language used for developing the proxy server.
- Required libraries: socket and threading for network communication and multithreading capabilities.
- Operating System: Windows/Linux/MacOS (depends on user preference).

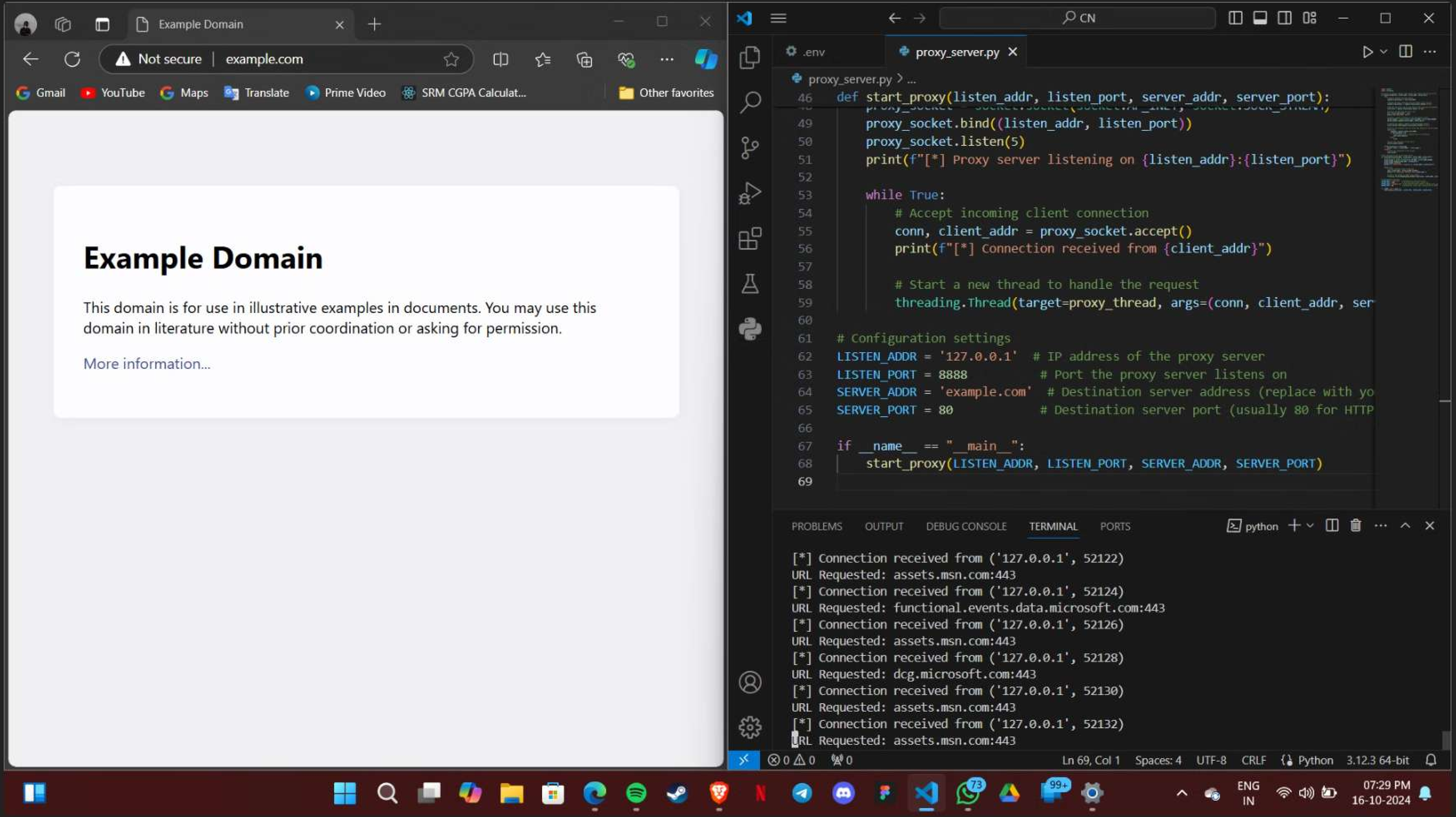
Hardware

- Processor: Minimum dual-core processor for handling multiple threads efficiently.
- RAM: At least 4 GB of RAM to support concurrent connections and smooth operation.
- Network Interface: Active internet connection for testing and running the proxy server.

Implementation

The proxy server is implemented using Python's socket library to create a TCP socket that listens for client connections. Each client request is handled in a separate thread, allowing for concurrent processing. The server forwards HTTP requests to the target server and relays the responses back to the client, ensuring seamless communication and efficient request handling.

Output



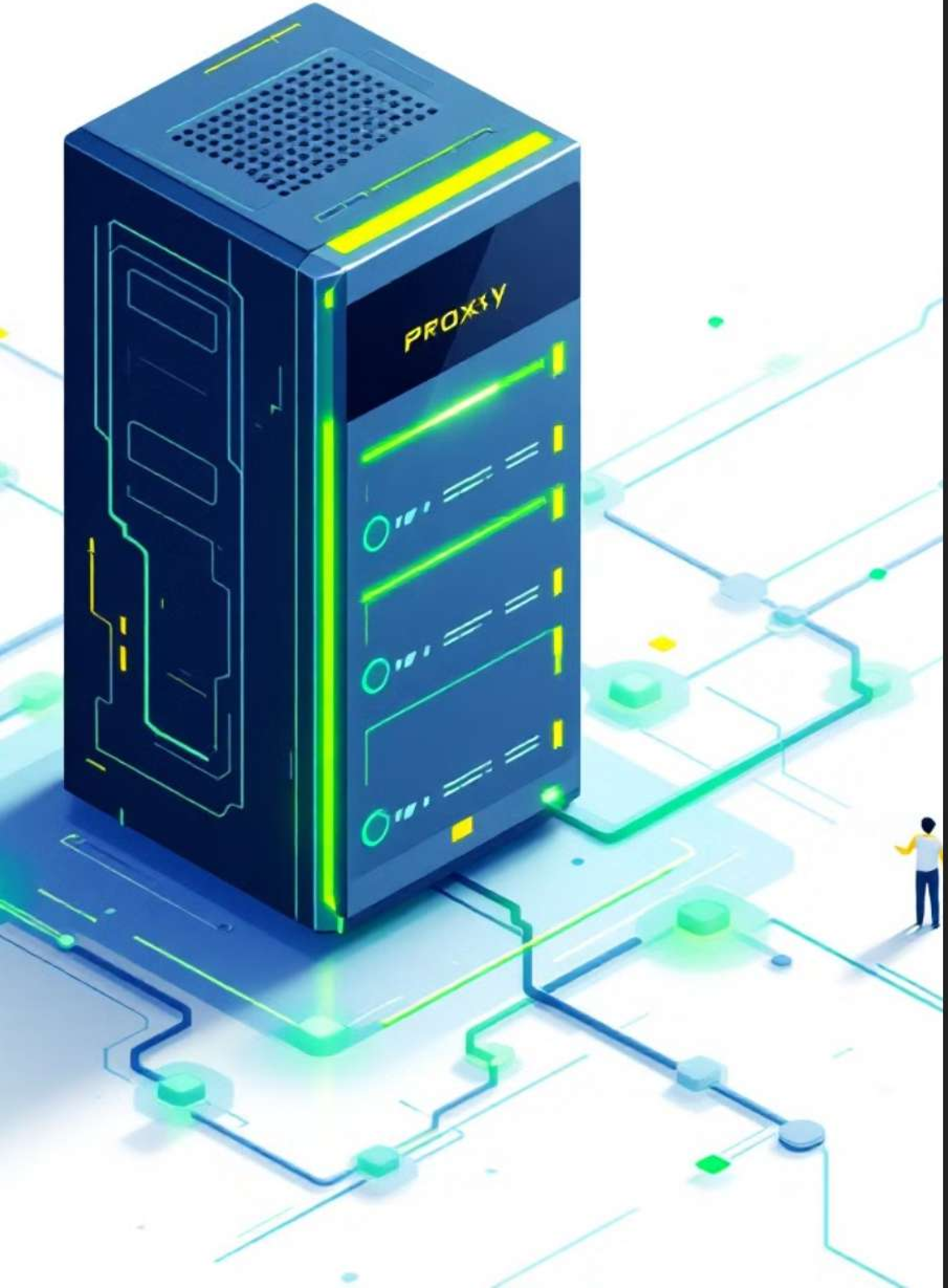
Example Domain

This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

More information...

```python
def start_proxy(listen_addr, listen_port, server_addr, server_port):
    proxy_socket.bind((listen_addr, listen_port))
    proxy_socket.listen(5)
    print(f"[*] Proxy server listening on {listen_addr}:{listen_port}")

    while True:
        # Accept incoming client connection
        conn, client_addr = proxy_socket.accept()
        print(f"[*] Connection received from {client_addr}")

        # Start a new thread to handle the request
        threading.Thread(target=proxy_thread, args=(conn, client_addr, ser

# Configuration settings
LISTEN_ADDR = '127.0.0.1'  # IP address of the proxy server
LISTEN_PORT = 8888          # Port the proxy server listens on
SERVER_ADDR = 'example.com'  # Destination server address (replace with yo
SERVER_PORT = 80            # Destination server port (usually 80 for HTTP

if __name__ == "__main__":
    start_proxy(LISTEN_ADDR, LISTEN_PORT, SERVER_ADDR, SERVER_PORT)
```

```
[*] Connection received from ('127.0.0.1', 52122)
URL Requested: assets.msn.com:443
[*] Connection received from ('127.0.0.1', 52124)
URL Requested: functional.events.data.microsoft.com:443
[*] Connection received from ('127.0.0.1', 52126)
URL Requested: assets.msn.com:443
[*] Connection received from ('127.0.0.1', 52128)
URL Requested: dcg.microsoft.com:443
[*] Connection received from ('127.0.0.1', 52130)
URL Requested: assets.msn.com:443
[*] Connection received from ('127.0.0.1', 52132)
URL Requested: assets.msn.com:443
```



Conclusion

1

Efficient Request Handling

The implemented proxy server demonstrates effective handling of HTTP requests, showcasing the ability to relay client communications to target servers efficiently.

2

Networking Fundamentals

This project highlights the fundamental principles of networking, multithreading, and socket programming, providing valuable insights into web traffic management.

3

Potential Enhancements

The proxy server can be further enhanced to support additional features such as HTTPS handling, logging, and security measures, making it a robust tool for managing web traffic.

Future Work

1

HTTPS Support

Future enhancements for the proxy server include implementing support for HTTPS connections to ensure secure data transmission between clients and servers.

2

Enhanced Security and Monitoring

Additionally, features such as request logging, traffic analysis, and user authentication can be integrated to improve security and monitoring capabilities.

3

Performance Optimization

Furthermore, optimizing performance through caching strategies and load balancing will enhance the server's efficiency and responsiveness, making it a more robust solution for managing web traffic in real-world applications.

References

1. "Understanding Proxy Servers." TechTarget. Available at: **https://www.techtarget.com/**
2. "How Proxy Servers Work." Lifewire. Available at: **https://www.lifewire.com/**
3. "Building a Simple Proxy Server in Python." Real Python. Available at: **https://realpython.com/**
4. "Introduction to Socket Programming in Python." GeeksforGeeks. Available at: **https://www.geeksforgeeks.org/**
5. "An Overview of Proxy Servers." Cloudflare. Available at: https://www.cloudflare.com/