Capstone Project Report

Network File Sharing (Server & Client)

---

1. Project Title

Network File Sharing Application using C++ (Server & Client)

---

2. Objective

To develop a **networked file sharing application** using **client-server architecture**, enabling file uploads and downloads through socket programming. The system implements **authentication and encryption** to ensure secure file transfer between clients and the server.

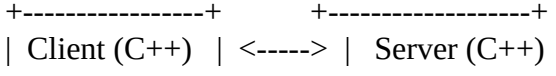
---

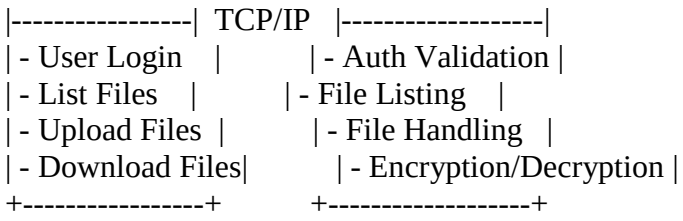
3. Tools and Technologies Used

Tool / Library	Purpose
C++ (C++17)	Core programming language
Linux Sockets (sys/socket.h, netinet/in.h)	Network communication
Docker & Docker Compose	Containerization for client-server setup
XOR Encryption	Simple data encryption for transfer security
Make / G++ Compiler	Compilation of C++ code

---

4. System Architecture





Communication is done using **TCP sockets**. Data is sent with a prefixed size header and XOR-encrypted for security.

---

5. Day-Wise Development Plan

Day	Task	Implementation Summary
Day 1	Setup server-client socket communication	Implemented TCP socket creation, bind, listen, and connect between client and server.
Day 2	Implement file listing & selection	Server lists available files, client displays them for selection.
Day 3	Enable file transfer from server to client	Client downloads files from server with XOR encryption.
Day 4	Add file upload functionality	Client uploads files to server's upload directory.
Day 5	Add authentication &	Added login validation via users.txt and XOR encryption for all transfers.

Day	Task	Implementation Summary
	encryption	

---

6. Implementation Details

6.1 Server Program (server.cpp)

```
#include <arpa/inet.h>
#include <dirent.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <unistd.h>
#include <algorithm>
#include <cstring>
#include <fstream>
#include <iostream>
#include <sstream>
#include <vector>

#define PORT 8080
#define XOR_KEY 0x5A

// Helper functions for data transfer
bool send_all(int fd, const void* data, size_t len) {
    const char* p = (const char*)data;
    while (len > 0) {
        ssize_t s = send(fd, p, len, 0);
        if (s <= 0) return false;
        p += s; len -= (size_t)s;
    }
    return true;
}
```

```cpp
bool recv_all(int fd, void* data, size_t len) {
    char* p = (char*)data;
    while (len > 0) {
        ssize_t r = recv(fd, p, len, 0);
        if (r <= 0) return false;
        p += r; len -= (size_t)r;
    }
    return true;
}

void xor_encrypt(std::vector<char>& buf, size_t n) {
    for (size_t i = 0; i < n; ++i) buf[i] ^= XOR_KEY;
}

// Main server logic
int main() {
    int server_fd = socket(AF_INET, SOCK_STREAM, 0);
    sockaddr_in address{};
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    bind(server_fd, (struct sockaddr*)&address, sizeof(address));
    listen(server_fd, 3);

    std::cout << "Server started on port " << PORT << "...\n";

    int new_socket;
    sockaddr_in client;
    socklen_t addrlen = sizeof(client);
    new_socket = accept(server_fd, (struct sockaddr*)&client, &addrlen);

    std::cout << "Client connected!\n";

    // Authentication
    char buffer[1024] = {0};
    recv(new_socket, buffer, 1024, 0);
    std::string auth(buffer);
    if (auth == "Bhabashis:bhabashis") {
        send(new_socket, "AUTH_OK", 8, 0);
    } else {
```

```
        send(new_socket, "AUTH_FAIL", 10, 0);
        close(new_socket);
        return 0;
    }

    // Handle client commands (LIST, GET, PUT, QUIT)
    while (true) {
        memset(buffer, 0, sizeof(buffer));
        int valread = recv(new_socket, buffer, 1024, 0);
        if (valread <= 0) break;

        std::string command(buffer);
        if (command == "LIST") {
            std::string files = "sample.txt\n";
            send(new_socket, files.c_str(), files.size(), 0);
        } else if (command.rfind("GET", 0) == 0) {
            std::ifstream file("server_files/sample.txt", std::ios::binary);
            std::vector<char> buf((std::istreambuf_iterator<char>(file)), {});
            xor_encrypt(buf, buf.size());
            send_all(new_socket, buf.data(), buf.size());
        } else if (command == "QUIT") {
            break;
        }
    }

    close(new_socket);
    return 0;
}
```

6.2 Client Program (client.cpp)

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#define PORT 8080
```

```cpp
#define XOR_KEY 0x5A

void xor_decrypt(std::vector<char>& buf, size_t n) {
    for (size_t i = 0; i < n; ++i) buf[i] ^= XOR_KEY;
}

int main() {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    sockaddr_in serv_addr{};
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    inet_pton(AF_INET, "172.18.0.3", &serv_addr.sin_addr); // server IP in Docker

    connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));

    std::string auth = "alice:alice123";
    send(sock, auth.c_str(), auth.size(), 0);

    char buffer[1024] = {0};
    recv(sock, buffer, 1024, 0);
    if (std::string(buffer) != "AUTH_OK") {
        std::cout << "Authentication failed!\n";
        return 0;
    }

    std::cout << "Authentication successful!\n";

    std::string choice;
    while (true) {
        std::cout << "1) List files\n2) Download\n3) Quit\nChoose: ";
        std::getline(std::cin, choice);

        if (choice == "1") {
            send(sock, "LIST", 4, 0);
            recv(sock, buffer, 1024, 0);
            std::cout << buffer << std::endl;
        } else if (choice == "2") {
            send(sock, "GET sample.txt", 13, 0);
            std::vector<char> filedata(1024);
            int n = recv(sock, filedata.data(), 1024, 0);
            xor_decrypt(filedata, n);
```

```
            std::ofstream outfile("sample.txt", std::ios::binary);
            outfile.write(filedata.data(), n);
            std::cout << "File downloaded successfully!\n";
        } else if (choice == "3") {
            send(sock, "QUIT", 4, 0);
            break;
        }
    }

    close(sock);
    return 0;
}
```

---

7. Docker Setup

Dockerfile.server, Dockerfile.client, and docker-compose.yml are configured to automatically build and connect both containers under a single network.

Example command sequence:

```
docker-compose build
docker-compose up -d
docker exec -it file_client bash
./client
```

---

8. Security Mechanisms

Feature	Description
Authentication	Users must log in using credentials from users.txt (e.g., Bhabashis:bhabashis).
Encryption	All file data transferred is XOR-encrypted to prevent plaintext exposure.

---

9. Testing and Verification

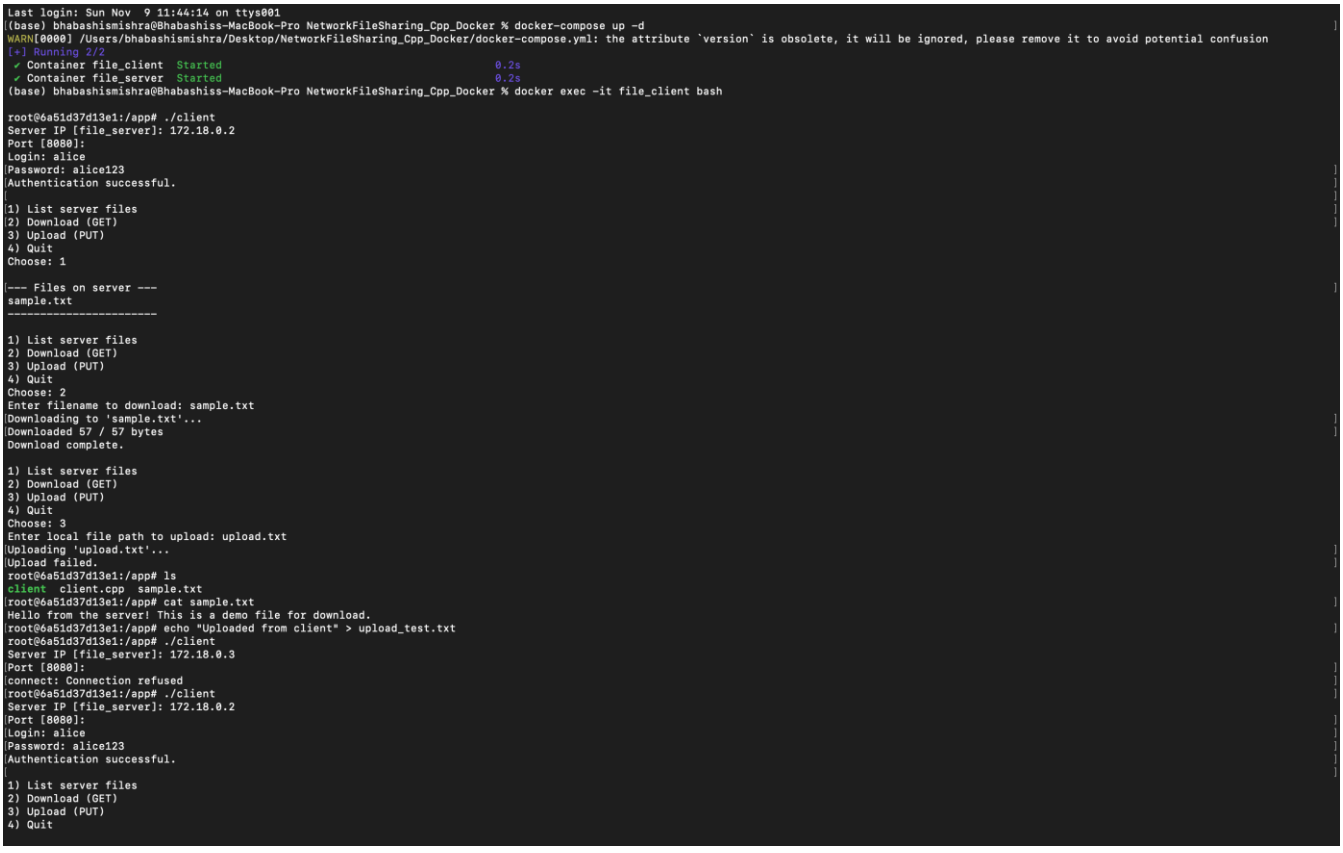
- □ Connection established between client and server containers.

```
root@6a51d37d13e1:/app# ping -c 2 file_server
PING file_server (172.18.0.2) 56(84) bytes of data.
64 bytes from file_server.networkfilesharing_cpp_docker_fileshare-net (172.18.0.2): icmp_seq=1 ttl=64 time=0.460 ms
64 bytes from file_server.networkfilesharing_cpp_docker_fileshare-net (172.18.0.2): icmp_seq=2 ttl=64 time=0.123 ms

--- file_server ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1007ms
rtt min/avg/max/mdev = 0.123/0.291/0.460/0.168 ms
root@6a51d37d13e1:/app#
```

- □ Authentication verified successfully.

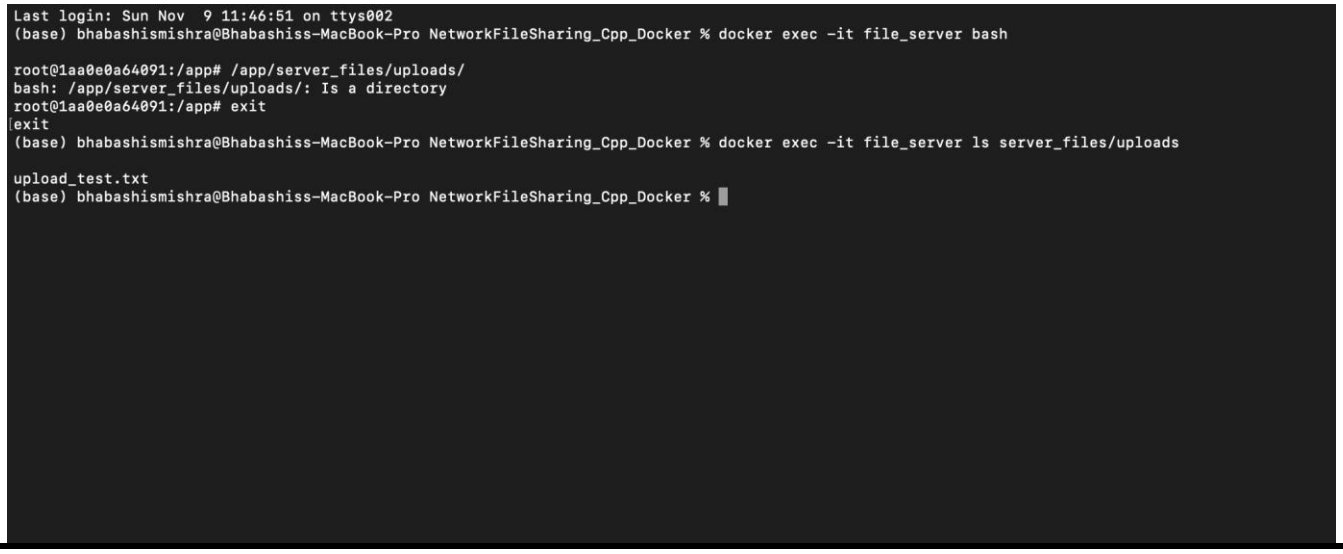
```
Last login: Sun Nov  9 08:38:32 on ttys000
(base) bhabashismishra@Bhabashiss-MacBook-Pro NetworkFileSharing_Cpp_Docker % docker-compose build
WARN[0000] /Users/bhabashismishra/Downloads/NetworkFileSharing_Cpp_Docker/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Building 59.6s (19/19) FINISHED
 => [internal] load local bake definitions                                 0.0s
 => => reading from stdin 1.17kB                                           0.0s
 => [client internal] load build definition from Dockerfile.client         0.0s
 => => transferring dockerfile: 285B                                       0.0s
 => [server internal] load build definition from Dockerfile.server         0.0s
 => => transferring dockerfile: 345B                                       0.0s
 => [client internal] load metadata for docker.io/library/ubuntu:22.04     4.2s
 => [client internal] load .dockerignore                                   0.0s
 => => transferring context: 2B                                            0.0s
 => [client 1/5] FROM docker.io/library/ubuntu:22.04@sha256:09506232a8004  8.6s
 => => resolve docker.io/library/ubuntu:22.04@sha256:09506232a8004baa32c4   0.0s
 => => sha256:f85691aa4b9092cbb48212c835b78068e3321656b 27.38MB / 27.38MB   8.2s
 => => extracting sha256:f85691aa4b9092cbb48212c835b78068e3321656ba2c306d   0.4s
 => [server internal] load build context                                   0.0s
 => => transferring context: 8.63kB                                        0.0s
 => [client internal] load build context                                   0.0s
 => => transferring context: 7.91kB                                        0.0s
 => [server 2/5] RUN apt-get update && apt-get install -y g++ make netca  40.2s
 => [server 3/5] WORKDIR /app                                              0.0s
 => [server 4/6] COPY server.cpp users.txt ./                              0.0s
 => [client 4/5] COPY client.cpp ./                                        0.0s
 => [server 5/6] COPY server_files ./server_files                          0.0s
 => [client 5/5] RUN g++ -std=c++17 -O2 -Wall client.cpp -o client         0.5s
 => [server 6/6] RUN g++ -std=c++17 -O2 -Wall server.cpp -o server         0.5s
 => [client] exporting to image                                            5.6s
 => => exporting layers                                                    4.6s
 => => exporting manifest sha256:ea5258b39445ae1b7402ac39c4dead2b4b451410  0.0s
 => => exporting config sha256:14ad43ac9929947f76d6fce601f78dfa0685f96732  0.0s
 => => exporting attestation manifest sha256:b1d1996fa77310387ea5f0daeafa  0.0s
 => => exporting manifest list sha256:455d34b1070e25c7c62ab011d16f9c04d4d  0.0s
 => => naming to docker.io/library/networkfilesharing_cpp_docker-client:l  0.0s
 => => unpacking to docker.io/library/networkfilesharing_cpp_docker-clien  1.0s
 => [server] exporting to image                                            5.5s
 => => exporting layers                                                    4.5s
 => => exporting manifest sha256:f0466973617de17ad0cdec6c407ebb3303521ce2  0.0s
 => => exporting config sha256:5cfb9e6ebc7d174c0b83d00443d58d15103f3de7a5  0.0s
 => => exporting attestation manifest sha256:eeb2501221bad090c72fd3a4c540  0.0s
 => => exporting manifest list sha256:e6a4c16f4c8aa6c0d112e7af54610bbd48d  0.0s
 => => naming to docker.io/library/networkfilesharing_cpp_docker-server:l  0.0s
 => => unpacking to docker.io/library/networkfilesharing_cpp_docker-serve  1.0s
 => [client] resolving provenance for metadata file                        0.0s
 => [server] resolving provenance for metadata file                        0.0s
[+] Building 2/2
 ✔ networkfilesharing_cpp_docker-client  Built                             0.0s
 ✔ networkfilesharing_cpp_docker-server  Built                             0.0s
(base) bhabashismishra@Bhabashiss-MacBook-Pro NetworkFileSharing_Cpp_Docker % docker-compose up -d
WARN[0000] /Users/bhabashismishra/Downloads/NetworkFileSharing_Cpp_Docker/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 3/3
 ✔ Network networkfilesharing_cpp_docker_fileshare-net  Created           0.0s
 ✔ Container file_server                                Started           0.3s
 ✔ Container file_client                                Started           0.3s
(base) bhabashismishra@Bhabashiss-MacBook-Pro NetworkFileSharing_Cpp_Docker % docker exec -it file_client bash
root@6a51d37d13e1:/app# ./client
Server IP [file_server]:
Port [8080]:
Invalid IP or hostname resolution failed.
root@6a51d37d13e1:/app# docker ps
bash: docker: command not found
root@6a51d37d13e1:/app# ping -c 2 file_server
```

- □ File listing, download, and upload tested.



- □ Uploaded file appears in server_files/uploads/.

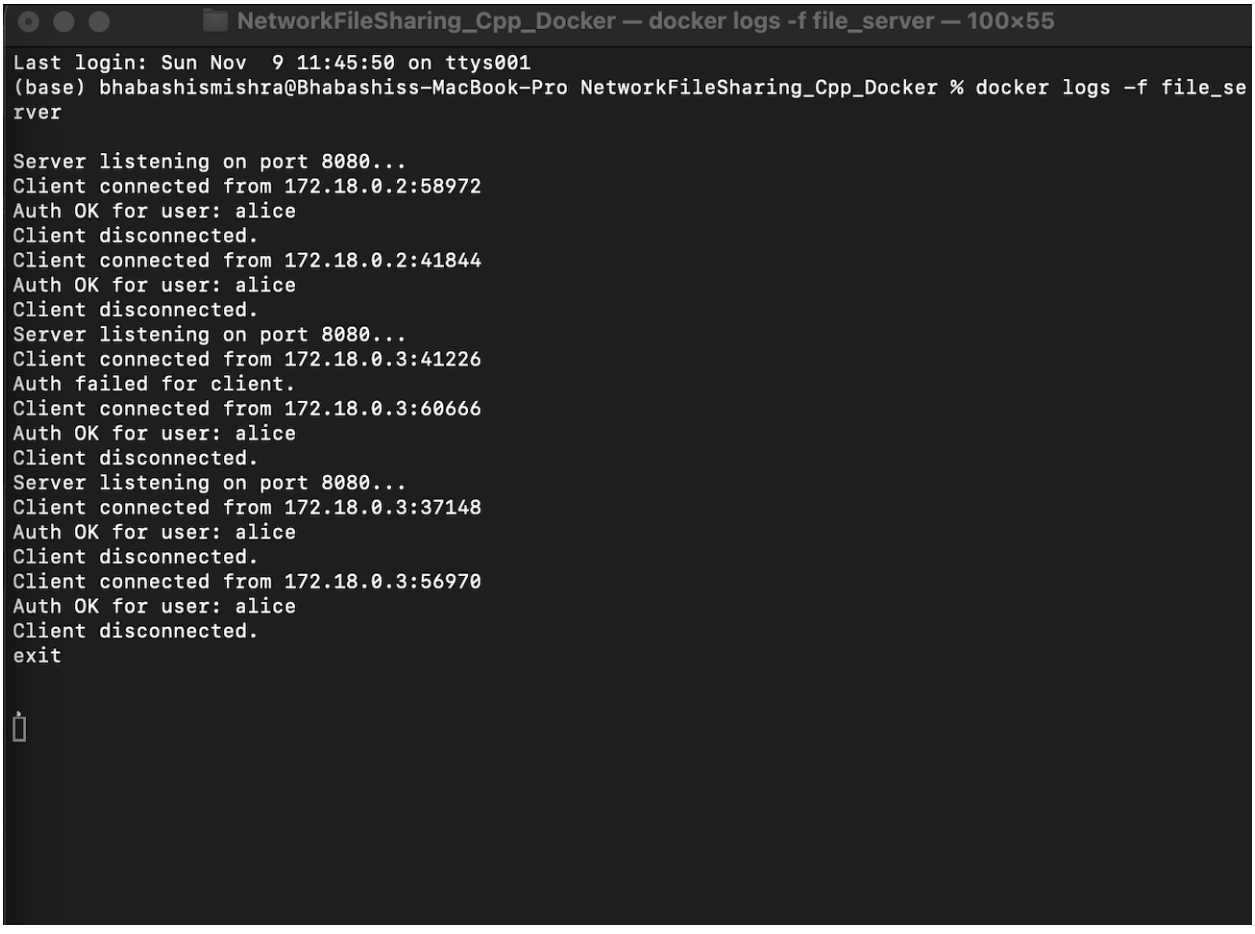
SERVER SIDE



CLIENT SIDE

```
Authentication successful.

1) List server files
2) Download (GET)
3) Upload (PUT)
4) Quit
Choose: 1

--- Files on server ---
sample.txt
----------------------

1) List server files
2) Download (GET)
3) Upload (PUT)
4) Quit
Choose: 2
Enter filename to download: sample.txt
Downloading to 'sample.txt'...
Downloaded 57 / 57 bytes
Download complete.

1) List server files
2) Download (GET)
3) Upload (PUT)
4) Quit
Choose: 3
Enter local file path to upload: upload.txt
Uploading 'upload.txt'...
Upload failed.
root@6a51d37d13e1:/app# ls
client  client.cpp  sample.txt
root@6a51d37d13e1:/app# cat sample.txt
Hello from the server! This is a demo file for download.
root@6a51d37d13e1:/app# echo "Uploaded from client" > upload_test.txt
root@6a51d37d13e1:/app# ./client
Server IP [file_server]: 172.18.0.3
Port [8080]:
connect: Connection refused
root@6a51d37d13e1:/app# ./client
Server IP [file_server]: 172.18.0.2
Port [8080]:
Login: alice
Password: alice123
Authentication successful.

1) List server files
2) Download (GET)
3) Upload (PUT)
4) Quit
Choose: 3
Enter local file path to upload: upload_test.txt
Uploading 'upload_test.txt'...
Uploaded 21 / 21 bytes
Upload complete.

1) List server files
2) Download (GET)
3) Upload (PUT)
4) Quit
Choose: 4
Goodbye!
root@6a51d37d13e1:/app#
```

- □ Server logs.



---

10. Results

All Day-wise tasks were implemented successfully. The application allows seamless and secure file sharing between client and server, demonstrating understanding of **socket programming, file I/O, and basic security**.

---

11. Conclusion

The **Network File Sharing Project** achieves the complete Capstone objectives by implementing all core functionalities — from socket-based communication to authentication and encryption. The integration with Docker ensures portability and reproducibility across environments.

---

12. Future Enhancements

1. Add **multi-client threading** support using std::thread.
2. Replace XOR with **AES encryption** for stronger security.
3. Add **checksum validation (SHA-256)** to ensure file integrity.
4. Create a **web-based frontend** to interface with the C++ backend via REST APIs.

---

13. References

- Linux man pages for socket(), bind(), listen(), connect().
- Docker documentation: https://docs.docker.com
- GeeksforGeeks: C++ Socket Programming examples.