File Transfer System

Project submitted to the

SRM University – AP, Andhra Pradesh

Submitted in partial fulfillment of the requirement for the award of the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted By

M. Venkata Narayana || AP23110011573

M. Preveen || AP23110011587

D. Lekith kishore || AP23110011591

S. Vivek || AP23110011595

Under the guidance of

Ms. Kavitha Rani

Department of Computer Science and Engineering

SRM University,AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240 [April,2025]

Department of Computer Science and Engineering

SRM University, AP

CERTIFICATE

This is to certify that the Project report entitled **“File transfer System”** is being submitted by **Lekith kishore Dasari (AP23110011591),** a student of Department of Computer Science and Engineering, SRM University,AP, in partial fulfillment of the requirement for the degree of **“B.Tech(CSE)”** carried out by her/his during the academic year 2024-2025.

Signature of the Supervisor   Signature of Head of the Dept.

Department of Computer Science and Engineering

SRM University,AP

CERTIFICATE

This is to certify that the Project report entitled **“File transfer System”** is being submitted by **Venkata Narayana Mogili (AP23110011573),** a student of Department of Computer Science and Engineering, SRM University,AP, in partial fulfillment of the requirement for the degree of **“B.Tech(CSE)”** carried out by her/his during the academic year 2024-2025.

Signature of the Supervisor Signature of Head of the Dept.

Department of Computer Science and Engineering

SRM University,AP

CERTIFICATE

This is to certify that the Project report entitled **“File transfer System”** is being submitted by **Vivek Seseti (AP23110011595),** a student of Department of Computer Science and Engineering, SRM University,AP, in partial fulfillment of the requirement for the degree of **“B.Tech(CSE)”** carried out by her/his during the academic year 2024-2025.

Signature of the Supervisor Signature of Head of the Dept.

Department of Computer Science and Engineering

SRM University,AP

CERTIFICATE

This is to certify that the Project report entitled **“File transfer System”** is being submitted by **Praveen Maradana (AP23110011587),** a student of Department of Computer Science and Engineering, SRM University,AP, in partial fulfillment of the requirement for the degree of **“B.Tech(CSE)”** carried out by her/his during the academic year 2024-2025.

Signature of the Supervisor Signature of Head of the Dept.

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, **Ms.Kavitha Rani**, Department of Computer Science & Engineering, SRM

University,Andhra pradesh, for her kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

D.Lekith Kishore

(AP23110011591)

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, **Ms.Kavitha Rani**, Department of Computer Science & Engineering, SRM

University,Andhra pradesh, for her kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

M. Venkata Narayana

(AP23110011573)

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, **Ms.Kavitha Rani**, Department of Computer Science & Engineering, SRM

University,Andhra pradesh, for her kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

S. Vivek

(AP23110011595)

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, **Ms.Kavitha Rani**, Department of Computer Science & Engineering, SRM

University,Andhra pradesh, for her kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

M. Preveen

(AP23110011587)

ABSTRACT

This project uses Python's socket programming to create a file transfer system. To enable safe file uploads and downloads, it has a client-server architecture with simple authentication. Socket, threading, operating system, and time are the main libraries utilized. TCP sockets are utilized to ensure reliable data transmission. While the client communicates with the server to exchange files, the server waits for incoming connections, verifies users, and executes commands. The project serves as a basis for more complex systems and uses minimal resources to demonstrate useful networking concepts.

TABLE OF CONTENTS

Introduction

In today's interconnected digital world, the ability to transfer files securely and efficiently between systems is essential. This project, titled **"File Transfer with Authentication Using Sockets in Python"**, was built with the goal of providing a simple, secure, and reliable method for transferring files over a network using socket programming.

At its core, this project establishes a **Client-Server architecture**, where the server is responsible for handling incoming connections, authenticating users, and managing file uploads and downloads. The client, on the other hand, initiates requests and interacts with the server based on user inputs.

The journey of building this system wasn't just about writing code—it involved understanding how networks work, how data travels over sockets, and how to ensure that unauthorized users are kept out through a basic authentication mechanism. While the authentication here is intentionally kept simple (for demonstration purposes), it reflects the importance of security even in small-scale applications.

What started as a simple local-host test eventually grew into a two-laptop setup using a wifi network, turning into a real-world test of functionality and patience. This project is not only a technical implementation but also a reflection of adaptability, collaboration, and learning under constraints. It showcases the practicality of Python's socket library and serves as a solid foundation for more advanced networked systems in the future.

Methodology

The development of this project was carried out following a modular and stepwise approach to ensure clarity, efficiency, and ease of debugging. The entire system was designed in two parts: **Server** and **Client**, which communicate through a socket-based connection using the **TCP protocol**.

Technology Stack

- **Programming Language:** Python
- **Socket Type:** TCP (SOCK_STREAM)
- **Libraries Used:**
    - socket – for network communication
    - os – for file system interaction on the client side

Server-Side Methodology:-

```
# Segment 1: Libraries
import socket
import threading
import os
import time
```

This segment imports essential libraries:

- socket: For handling network connections.
- threading: For managing multiple clients concurrently.
- os: For file operations (e.g., checking if a file exists).

- time: To add delays when needed (e.g., before sending data).

Segment 2: Dummy Authentication Data

```
# Segment 2: Dummy Authentication Data
VALID_USERS = {
    "Dhaya": "Police",
    "user": "pass"
}
```

This dictionary stores valid usernames and passwords for basic authentication. You can add more users by updating this dictionary.

Segment 3: Client Handler

```
# Segment 3: Client Handler
def handle_client(client_socket, addr):
    print(f"[+] Connection from {addr}")
```

This function handles each client who connects to the server.

Segment 3.1: Authentication

```
# Segment 3.4: File Download

    elif command == "download":

        filename = client_socket.recv(1024).decode().strip()

        if not os.path.exists(filename):

            client_socket.send(b"File not found.")

            continue
        client_socket.send(b"OK")

        time.sleep(0.5)

        with open(filename, "rb") as f:

            while chunk := f.read(4096):

                client_socket.send(chunk)

        client_socket.send(b"<EOF>")
```

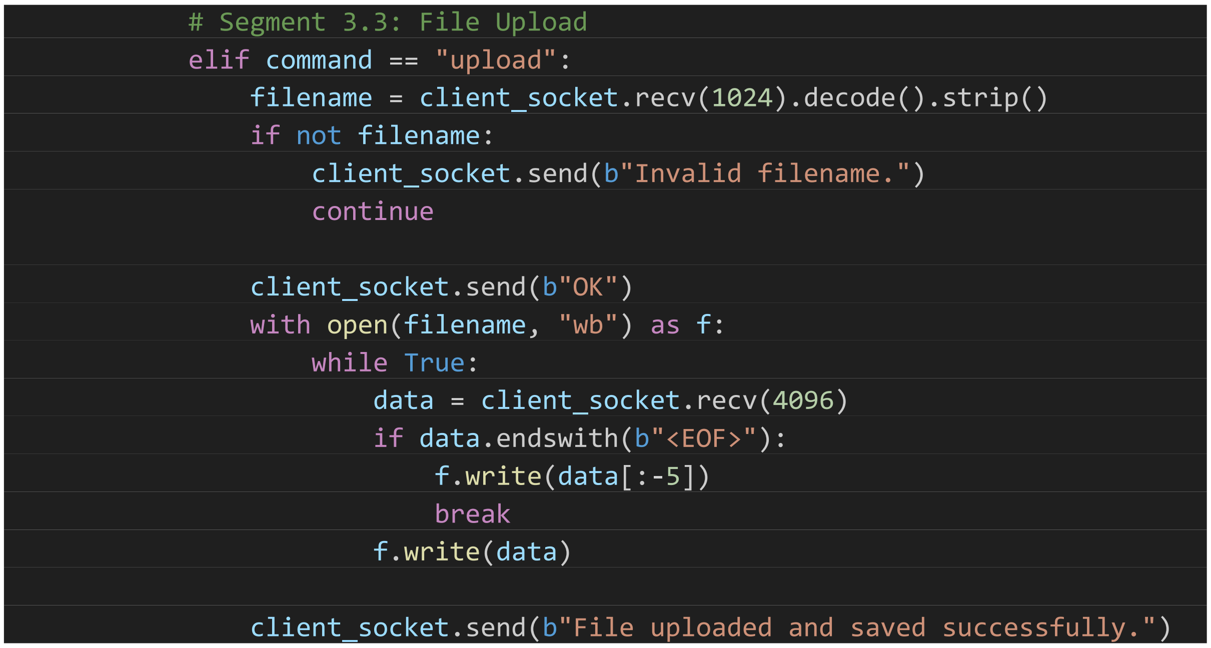
- Asks the client for username and password.
- Verifies credentials against VALID_USERS.
- Disconnects if authentication fails.

Segment 3.2: Exit Command

```
# Segment 3.2: Exit Command
            if command == "exit":
                print(f"[-] {addr} disconnected.")
                break
```

If the client sends "exit", the connection is closed gracefully.

Segment 3.3: File Upload



```
            # Segment 3.3: File Upload
            elif command == "upload":
                filename = client_socket.recv(1024).decode().strip()
                if not filename:
                    client_socket.send(b"Invalid filename.")
                    continue

                client_socket.send(b"OK")
                with open(filename, "wb") as f:
                    while True:
                        data = client_socket.recv(4096)
                        if data.endswith(b"<EOF>"):
                            f.write(data[:-5])
                            break
                        f.write(data)

                client_socket.send(b"File uploaded and saved successfully.")
```

- Waits for the client to send a filename.
- Receives file data in chunks until <EOF> is found.
- Saves the file on the server.

Segment 3.4: File Download

```
# Segment 3.4: File Download
            elif command == "download":
                filename =
client_socket.recv(1024).decode().strip()
                if not os.path.exists(filename):
                    client_socket.send(b"File not found.")
                    continue

                client_socket.send(b"OK")
                time.sleep(0.5)
                with open(filename, "rb") as f:
                    while chunk := f.read(4096):
                        client_socket.send(chunk)
                client_socket.send(b"<EOF>")
```

- Checks if the requested file exists.
- If yes, reads and sends the file to the client in chunks.
- Marks end of file with <EOF>.

Segment 3.5: Error Handling

```
# Segment 3.5: Error Handling
        except Exception as e:
            print(f"Error with client {addr}: {e}")
            break
```

- Catches any exceptions during communication.
- Helps identify issues without crashing the server

Segment 3.6: Close Connection

```
# Segment 3.6: Close Connection
    client_socket.close()
```

- Always closes the client socket after handling.

Segment 4: Server Starter

```
# Segment 4: Server Starter
def start_server():
    server = socket.socket(socket.AF_INET, 
socket.SOCK_STREAM)
    server.bind(("0.0.0.0", 8080))
    server.listen(5)
    print("[*] Server listening on port 8080...\n")

    while True:
        client_socket, addr = server.accept()
        threading.Thread(target=handle_client, 
args=(client_socket, addr)).start()
```

- Creates a socket.
- Binds to IP 0.0.0.0 and port 8080 (accepts all interfaces).
- Listens for incoming connections.
- For each connection, starts a new thread using handle_client.

Segment 5: Entry Point

```
# Segment 4: Main Function

if __name__ == "__main__":

   start_client()
```

- Entry point of the script.
- Runs the server when you execute the Python file.

Client-Side Methodology

Segment 1: Libraries

```
# Segment 1: Import Libraries
import socket
import os
```

- socket: Used to connect to the server and send/receive data.
- os: To check if a file exists on your system before uploading.

Segment 2: Start Client Function

```
# Segment 2: Client Function
def start_client():
    try:
```

This is the main function that:

- Connects to the server.

- Handles login.
- Lets the user upload/download files or exit.

Sub-segment 2.1: Socket Creation and Connection

```
# Sub-segment 2.1: Socket Creation and Connection
        client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        host = input("Enter the server address: ").strip()
        client_socket.connect((host, 8080))
        print("Connected to the server.")
```

- Creates a TCP socket using IPv4 (AF_INET, SOCK_STREAM).
- Asks user for the server's IP address (can be localhost or LAN IP).
- Connects to the server on port 8080.

he server.")

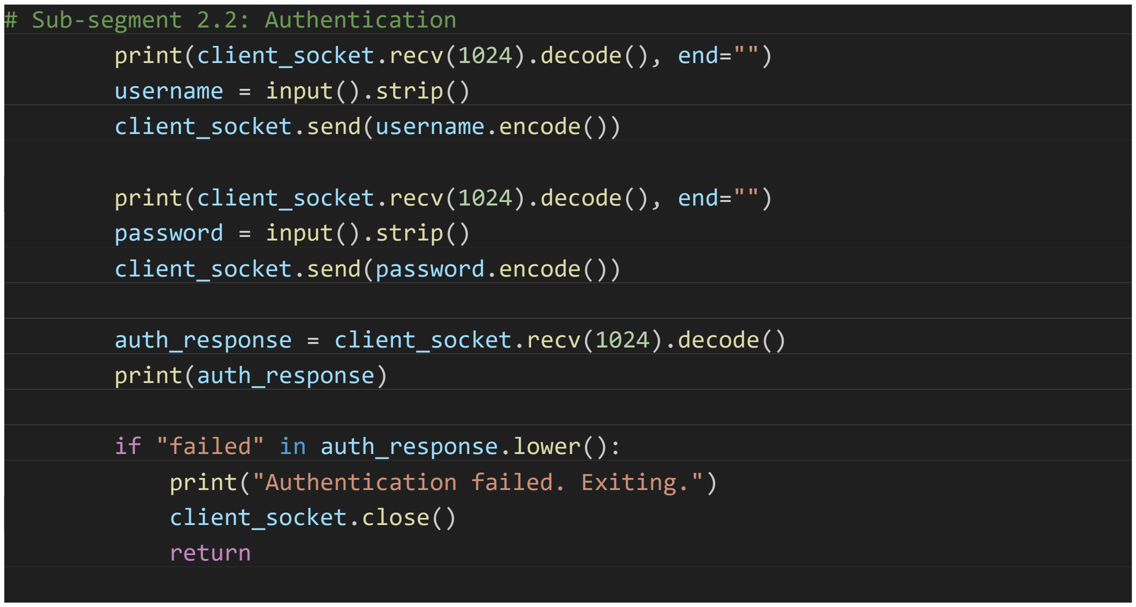
```
# Sub-segment 2.2: Authentication

print(client_socket.recv(1024).decode(), end="")

username = input().strip()
```

-

Sub-segment 2.2: Authentication



```python
# Sub-segment 2.2: Authentication
        print(client_socket.recv(1024).decode(), end="")
        username = input().strip()
        client_socket.send(username.encode())

        print(client_socket.recv(1024).decode(), end="")
        password = input().strip()
        client_socket.send(password.encode())

        auth_response = client_socket.recv(1024).decode()
        print(auth_response)

        if "failed" in auth_response.lower():
            print("Authentication failed. Exiting.")
            client_socket.close()
            return
```

- Waits for prompts from the server and sends username/password.
- Encodes the strings before sending (as network data is in bytes).
- Receives authentication status and prints the result.

Sub-segment 2.3: Command Loop

```
# Sub-segment 2.3: Command Loop
        while True:
            user_input = input("Enter command
(upload/download/exit): ").strip().lower()
            client_socket.send(user_input.encode())

            if user_input == "exit":
                print("Closing connection...")
                break
```

- After authentication, the client enters a loop to send commands to the server.
- Valid commands: upload, download, or exit.
- **Sub-segment 2.4: Upload Functionali**

```
# Sub-segment 2.4: Upload Functionality
        elif user_input == "upload":
            filename = input("Enter the filename to upload: ").strip()
            if not os.path.exists(filename):
                print("File not found. Please check the filename and try again.")
                continue

            client_socket.send(filename.encode())
            server_response = client_socket.recv(1024).decode()
            if server_response != "OK":
                print("Server rejected the file upload.")
                continue

            with open(filename, 'rb') as f:
                while chunk := f.read(4096):
                    client_socket.send(chunk)

            client_socket.send(b"<EOF>")
            print(client_socket.recv(1024).decode())
```

-

Asks for the file name.

- Verifies that the file exists on the client’s system.
- Sends the file in chunks of 4096 bytes to the server.
- Uses <EOF> to tell the server when the file is completely sent.
- Waits for a confirmation message from the server.

Sub-segment 2.5: Download Functionality

```
Sub-segment 2.5: Download Functionality
            elif user_input == "download":
                filename = input("Enter the filename to 
download: ").strip()
                client_socket.send(filename.encode())

                response = 
client_socket.recv(1024).decode(errors="ignore")
                if response == "OK":
                    print(f"Downloading 
'{filename}'...\n")
                    with open(filename, 'wb') as f:
                        while True:
                            file_data = 
client_socket.recv(4096)
                            if 
file_data.endswith(b"<EOF>"):
                                f.write(file_data[:-5])
                                break
                            f.write(file_data)

                    print(f"✅ File '{filename}' saved 
successfully.")
                else:
                    print(f"❌ Error: {response}")
```

- Sends a filename request to the server.
- If the server confirms with "OK", it starts downloading.
- File data is written to a local file.
- It stops once it detects <EOF> in the data.

Sub-segment 2.6: Close Connection

```
# Sub-segment 2.6: Close Connection
        client_socket.close()

```

- Closes the socket cleanly after exit or when done with transfer.

Segment 3: Exception Handling

```
 # Segment 3: Exception Handling
    except ConnectionRefusedError:
        print("❌ Error: Unable to connect to the server. 
Please check the server address and try again.")
    except Exception as e:
        print(f"❌ An error occurred: {e}")
    finally:
        client_socket.close()
```

- Handles issues like:
    - Server not running
    - Invalid IP
    - Network failure
- Prevents crashes by showing user-friendly messages.

Segment 4: Main Function

```python
# Segment 4: Main Function
if __name__ == "__main__":
    start_client()
```

- Ensures that the start_client() function only runs when this file is executed directly.
- Standard Python practice for entry-point logic.

This structured methodology helped in building a simple yet robust file transfer system with basic user authentication and proper error handling mechanisms

CONCLUSION

This project successfully demonstrates the practical implementation of a client-server based file transfer system using Python sockets. By leveraging the socket, threading, os, and time modules, we were able to design a system that supports secure communication, authenticated access, and reliable file transfer between devices connected on the same network.

The server is designed to handle multiple client connections simultaneously through threading, allowing parallel file transfers without blocking or interference. Basic authentication ensures that only authorized users can interact with the system. The client, on the other hand, provides a simple command-line interface where users can upload files to the server, download files from it, or exit the connection gracefully.

The system uses TCP sockets for data transmission, ensuring reliability and order of data packets. A custom <EOF> marker was implemented to signify the end of file transfers, ensuring accurate file reconstruction on both ends.

Overall, this project serves as a foundational model for more complex systems such as secure file sharing platforms, FTP servers, or cloud-sync applications. It also reinforces core networking concepts and Python programming practices, making it an excellent learning experience and a stepping stone for advanced development in network-based applications.

REFERENCES

- GeeksforGeeks – Socket Programming in Python https://www.geeksforgeeks.org/socket-programming-python/
- Real Python – Python Socket Programming https://realpython.com/python-sockets/
- Python Official Documentation – socket module https://docs.python.org/3/library/socket.html
- Python Official Documentation – threading module https://docs.python.org/3/library/threading.html
- Python Official Documentation – os module https://docs.python.org/3/library/os.html