

A  
Project Report  
On

## **Remote File Manager**

Submitted in partial fulfillment of the requirement for the degree of  
**Bachelor of Technology**

In  
**Computer Science and Engineering**

By

<b>Aayush Mehra</b>	<b>2261055</b>
<b>Aman Negi</b>	<b>2261086</b>
<b>Himanshu Joshi</b>	<b>2261268</b>
<b>Tanuj Joshi</b>	<b>2261571</b>

Under the Guidance of

**Mr. Prince Kumar**

**ASSISTANT / ASSOCIATE PROFESSOR**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**  
**GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS**  
**SATTAL ROAD, P.O. BHOWALI,**  
**DISTRICT- NAINITAL-263132**

**2024-2025**

## **STUDENT'S DECLARATION**

We, **Aayush Mehra, Aman Negi, Himanshu Joshi** and **Tanuj Joshi** hereby declare the work, which is being presented in the project, entitled **Remote File Manager** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of **Mr. Prince Kumar**. The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Aayush Mehra

Aman Negi

Himanshu Joshi

Tanuj Joshi

## **CERTIFICATE**

**The project report entitled “Remote File Manager” being submitted by Aayush Mehra S/o Pramod Singh Mehra, 2261055, Aman Negi S/o Rajendra Singh Negi, 2261086, Himanshu Joshi S/o Deep Chandra Joshi, 2261268, Tanuj Joshi S/o Pitambar Joshi, 2261571 of B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.**

**Mr. Prince Kumar**  
**(Project Guide)**

**Dr. Ankur Singh Bisht**  
**(Head, CSE)**

## **ACKNOWLEDGEMENT**

We take immense pleasure in thanking the Honorable Director '**Prof. (Col.) Anil Nair (Retd.)**', GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president '**Prof. (Dr.) Kamal Ghanshala**' for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to '**Dr. Ankur Singh Bisht**' (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide '**Mr. Prince Kumar**' (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

**Aayush Mehra, 2261055**  
**Aman Negi, 2261086**  
**Himanshu Joshi, 2261268**  
**Tanuj Joshi, 2261571**

## **Abstract**

The Remote File Manager is a client server-based application that facilitates remote file access and manipulation in a secure and user-friendly environment. Built using Python and PySide6, the client provides a desktop GUI for users to upload, download, delete, rename, search, and edit files hosted on a remote server. Communication between the client and server is achieved using sockets and a structured JSON protocol, while user authentication and access control are managed using a MySQL database and Access Control Lists (ACLs).

This project was conceived to address the limitations of existing file management systems, particularly their lack of open-source alternatives with customizable security features and user access permissions. Through a structured software development lifecycle—comprising requirement analysis, system design, implementation, and testing—the project delivers a scalable and modular architecture. While the current implementation supports core functionality, it also identifies areas for improvement including concurrent processing, encrypted communication, and cloud integration.

The Remote File Manager demonstrates how foundational software engineering practices can produce a reliable system tailored to real-world needs in both academic and organizational settings. It serves as a prototype for more complex enterprise solutions and provides a solid base for future enhancements.

## **TABLE OF CONTENTS**

Declaration.....	i
Certificate.....	ii
Acknowledgement.....	iii
Abstract.....	iv
Table of Contents.....	v
<b>CHAPTER 1 INTRODUCTION.....</b>	<b>7</b>
1.1 Prologue.....	7
2.1 Background and Motivations.....	7
3.1 Problem Statement.....	7
4.1 Objectives and Research Methodology.....	7
5.1 Project Organization.....	9
<b>CHAPTER 2 PHASES OF SOFTWARE DEVELOPMENT CYCLE....</b>	<b>10</b>
1.1 Hardware Requirements.....	11
2.1 Software Requirements.....	11
<b>CHAPTER 3 CODING OF FUNCTIONS.....</b>	<b>12</b>
<b>CHAPTER 4 SNAPSHOT.....</b>	<b>18</b>
<b>CHAPTER 5 LIMITATIONS (WITH PROJECT) .....</b>	<b>21</b>
<b>CHAPTER 6 ENHANCEMENTS.....</b>	<b>22</b>
<b>CHAPTER 7 CONCLUSION.....</b>	<b>23</b>
<b>REFERENCES.....</b>	<b>24</b>

# INTRODUCTION

## 1.1 Prologue

The Remote File Manager project is a robust software application designed to address the challenges of remote file access and manipulation in multi-user environments. The system enables users to interact with files on a remote server securely through a desktop GUI built using PySide6. Key operations supported include uploading, downloading, deleting, renaming, searching, and editing files. The application communicates via sockets using a JSON-based protocol and manages user authentication and access permissions through a MySQL database.

## 1.2 Background and Motivations

As more organizations migrate to remote working environments and distributed infrastructures, managing files over a network has become a core requirement. Most existing solutions are either proprietary or lack fine-grained access control. This project was initiated to create a custom remote file management system that integrates security, usability, and extensibility. It aims to serve educational purposes and be a prototype for more advanced implementations in enterprise settings.

With increasing reliance on cloud-based and remote data storage systems, there is a growing need for efficient, secure, and easily accessible file management tools. This project was initiated to provide a lightweight remote file manager system that integrates access control, user authentication, and full file manipulation features without relying on heavyweight third-party tools or platforms.

## 1.3 Problem Statement

Modern users need to access and manage files from different locations without compromising security. However, existing tools often lack features like permission-based access or seamless integration with GUI clients. The Remote File Manager addresses this gap by offering a secure, efficient, and easy-to-use interface for managing remote files while controlling user actions based on access rights.

## 1.4 Objectives and Research Methodology

primary objectives of the project are:

- To design and implement a secure client-server application for remote file

operations.

- To develop a graphical interface that simplifies user interactions.
- To support full file manipulation capabilities (upload, download, delete, etc.).
- To implement user authentication and Access Control List (ACL) mechanisms.
- To ensure data communication is structured, using a JSON-based protocol.

**Research Methodology:** The research methodology adopted for the Remote File Manager project is a combination of exploratory and applied research. The following steps outline the approach:

**1. Exploratory Research:**

- Studied various existing file management systems and remote access tools.
- Identified key limitations in publicly available tools, especially around access control and user-friendly interfaces.
- Researched protocols like FTP, SCP, and modern cloud storage APIs to understand industry standards.

**2. Requirement Gathering:**

- Defined user personas and use cases by analyzing common operations performed on remote files.

**3. Design and Modeling:**

- Designed the system architecture using a modular client-server model.
- Prepared data models and access control schemes using Entity-Relationship Diagrams (ERDs).

**4. Prototype Development:**

- Developed initial versions of the client GUI and server handlers.
- Iteratively tested modules with sample users to validate the usability and correctness.

**5. Testing and Evaluation:**

- Used test cases for each operation (upload, download, etc.) under normal and edge scenarios.
- Evaluated the system based on usability, response time, and correctness of



access control enforcement.

#### 6. Refinement and Validation:

- Incorporated feedback from user trials to refine interface elements and server-side logic.
- Validated results by comparing expected versus actual behavior.

This methodology ensured that the system was not only functional but also grounded in practical needs and informed by existing best practices in software development.

### 1.5 Project Organization

The project is divided into several modules:

- **Authentication Module:** Handles user registration and login, ensuring secure credential management.
- **Client Interface:** A PySide6-based GUI allowing users to perform file operations intuitively.
- **File Management Module:** Processes commands related to file handling such as upload, download, delete, rename, and edit.
- **ACL Module:** Manages user permissions on files, providing fine-grained control over file access.
- **Database Integration:** Uses MySQL to store user information and access control data.

## PHASES OF SOFTWARE DEVELOPMENT CYCLE

The software development lifecycle (SDLC) of the Remote File Manager project followed a structured and iterative approach to ensure a reliable and efficient system. The key phases are described below:

### 1. Requirement Analysis:

- Identified the needs of users for remote file access, manipulation, and security.
- Conducted research into similar systems and gathered input for functional and non-functional requirements.
- Defined use cases including upload, download, edit, delete, rename, and access control.

### 2. System Design:

- Designed the architecture using a client-server model based on socket programming.
- Created database schema to manage users and access control lists (ACLs).
- Designed the GUI layout using PySide6 for clarity and usability.

### 3. Implementation:

- Developed the client application using PySide6 for the interface and Python for logic.
- Implemented the server using socket programming, handling multiple file operations and user commands via a JSON-based protocol.
- Integrated MySQL for backend storage of user and file metadata.

### 4. Testing and Validation:

- Performed unit testing on individual modules such as file operations, authentication, and permission checks.
- Conducted integration testing to ensure smooth communication between client and server.
- Validated functionalities through use-case scenarios and simulated multiple users.

### 5. Deployment:

- Set up the server environment and ensured the client could reliably connect.
- Deployed the MySQL database and ensured secure connectivity.

## HARDWARE AND SOFTWARE REQUIREMENTS

### 2.1 Hardware Requirement

The development and testing of the Remote File Manager project required a system with the following minimum hardware specifications to ensure smooth compilation, execution:

Component	Specification (Minimum)
Processor	Intel Core i5 (8th Gen) or better
RAM	8 GB
Storage	250 GB SSD or HDD
Display	1024 x 768 resolution or higher
Input Devices	Keyboard and Mouse
Network	Stable Wi-Fi or Ethernet connection

### 2.2 Software Requirement

The software tools and platforms used for the development of the **Remote File Manager** project are as follows:

- **Operating System:** Windows 10 or later (64-bit)
- **Programming Language:** Python
- **Compiler:** Python: Version 3.12
- **Build System:** Visual Studio IDE
- **Editor/IDE:** Visual Studio Code / Visual Studio 2022
- **GUI Library:** PySide6
- **Database:** Mysql
- **Required Python Libraries:** socket, json, os, shutil, threading, mysql-connector-python

These tools enabled efficient development, debugging, version tracking, and GUI rendering support for the Remote File Manager project.

## CODE:

### Server:

```
import socket
import threading
import json
import mysql.connector
import os

SERVER_HOST = '0.0.0.0'
SERVER_PORT = 5050
BUFFER_SIZE = 65536
FILE_STORAGE_PATH = "server_files"
os.makedirs(FILE_STORAGE_PATH, exist_ok=True)

db_config = {
    "host": "localhost",
    "user": "root",
    "password": "Caillin#26",
    "database": "registration_db"
}

def insert_user(user_data):
    try:
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO users (first_name, last_name, email, contact, password, security_question, security_answer)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
            """, (
                user_data["first_name"],
                user_data["last_name"],
                user_data["email"],
                user_data["contact"],
                user_data["password"],
                user_data["security_question"],
                user_data["security_answer"]
            ))
        conn.commit()
        return {"status": "ok", "message": "Registration successful"}
```

```

        conn.commit()
        return {"status": "ok", "message": "Registration successful"}
except mysql.connector.IntegrityError:
    return {"status": "error", "message": "Email already registered"}
except Exception as e:
    return {"status": "error", "message": f"DB error: {str(e)}"}
finally:
    cursor.close()
    conn.close()

def validate_login(data):
    try:
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()
        cursor.execute("SELECT password FROM users WHERE email=%s", (data["email"],))
        result = cursor.fetchone()
        if result and result[0] == data["password"]:
            return {"status": "ok", "message": "Login successful"}
        else:
            return {"status": "error", "message": "Invalid email or password"}
    except Exception as e:
        return {"status": "error", "message": f"DB error: {str(e)}"}
    finally:
        cursor.close()
        conn.close()

def check_acl(email, filename, permission):
    try:
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()
        cursor.execute(f"""
            SELECT {permission} FROM file_acl
            WHERE email = %s AND filename = %s
            """, (email, filename))
        result = cursor.fetchone()
        if result:

```

```

def check_acl(email, filename, permission):
    try:
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()
        cursor.execute(f"""
            SELECT {permission} FROM file_acl
            WHERE email = %s AND filename = %s
            """, (email, filename))
        result = cursor.fetchone()
        if result:
            return bool(result[0])
        return False
    except Exception as e:
        print(f"[ACL ERROR] {e}")
        return False
    finally:
        cursor.close()
        conn.close()

def handle_file_command(client_socket, command, email):
    action = command.get("action")
    try:
        if action == "upload":
            filename = command["filename"]
            filesize = command["filesize"]
            if not check_acl(email, filename, "can_upload"):
                client_socket.send(json.dumps({"status": "error", "message": "Permission denied (upload)"}).encode())
                return

            filepath = os.path.join(FILE_STORAGE_PATH, filename)
            client_socket.send(json.dumps({"status": "ready"}).encode())

            with open(filepath, "wb") as f:
                total = 0

```

```

def handle_client(client_socket, address):
    print(f"[NEW CONNECTION] {address} connected.")
    session_email = None # Track logged-in user

    try:
        while True:
            data = client_socket.recv(BUFFER_SIZE).decode()
            if not data:
                break
            try:
                command = json.loads(data)
            except json.JSONDecodeError:
                client_socket.send(json.dumps({"status": "error", "message": "Invalid JSON"}).encode())
                continue

            if command.get("type") == "register":
                response = insert_user(command)
                client_socket.send(json.dumps(response).encode())
            elif command.get("type") == "login":
                response = validate_login(command)
                if response["status"] == "ok":
                    session_email = command["email"]
                    client_socket.send(json.dumps(response).encode())
            elif command.get("action"):
                if not session_email:
                    client_socket.send(json.dumps({"status": "error", "message": "Not authenticated"}).encode())
                    continue
                handle_file_command(client_socket, command, session_email)
            else:
                client_socket.send(json.dumps({"status": "error", "message": "Invalid request"}).encode())

```

```

def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((SERVER_HOST, SERVER_PORT))
    server.listen(5)
    print(f"[LISTENING] Server is listening on {SERVER_HOST}:{SERVER_PORT}")
    while True:
        client_socket, address = server.accept()
        thread = threading.Thread(target=handle_client, args=(client_socket, address))
        thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.active_count() - 1}")

if __name__ == "__main__":
    start_server()

```

## Client:

```

class RemoteFileManager(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Remote File Manager")
        self.setMinimumSize(1000, 600)
        self.sock = None

        self.init_ui()
        self.connect_to_server()
        self.refresh_file_list()

    def init_ui(self):
        main_layout = QHBoxLayout(self)

        left_panel = QFrame()
        left_panel.setFrameShape(QFrame.StyledPanel)
        left_layout = QVBoxLayout()
        left_layout.setAlignment(Qt.AlignmentFlag.AlignTop)

        self.buttons = {
            "Refresh": self.refresh_file_list,
            "Upload": self.upload_file,
            "Download": self.download_file,
            "Delete": self.delete_file,
            "Rename": self.rename_file,
            "Open File": self.read_file,
        }

        for label, func in self.buttons.items():
            btn = QPushButton(label)
            btn.setFixedHeight(40)
            btn.clicked.connect(func)
            left_layout.addWidget(btn)

        self.progress = QProgressBar()
        self.progress.setFixedHeight(20)
        self.progress.setValue(0)

```

```

class LoginWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Login")
        self.setFixedSize(400, 450)

        self.login_frame = QFrame(self)
        self.login_frame.setGeometry(0, 0, 400, 450)
        self.login_frame.setStyleSheet("background-color: #222; border-radius: 10px;")

        self.setup_form()

    def setup_form(self):
        layout = QVBoxLayout()
        layout.setContentsMargins(40, 40, 40, 40)
        layout.setSpacing(20)

        title = QLabel("Get Started")
        title.setFont(QFont("Times New Roman", 20, QFont.Bold))
        title.setStyleSheet("color: white;")
        title.setAlignment(Qt.AlignCenter)
        layout.addWidget(title)

        user_label = QLabel("Username/Email")
        user_label.setFont(QFont("Times New Roman", 14))
        user_label.setStyleSheet("color: white;")
        layout.addWidget(user_label)

        self.username_input = QLineEdit()
        self.username_input.setPlaceholderText("Enter your username")
        self.username_input.setStyleSheet("padding: 8px; font-size: 14px;")
        layout.addWidget(self.username_input)

        pass_label = QLabel("Password")

```



```

class RegistrationPage(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("User Registration")
        self.setFixedSize(600, 600)
        self.setup_ui()

    def setup_ui(self):
        font_label = QFont("Arial", 11)
        font_title = QFont("Arial", 22, QFont.Bold)

        main_layout = QVBoxLayout(self)
        main_layout.setContentsMargins(40, 30, 40, 30)
        main_layout.setSpacing(25)

        title = QLabel("Create Your Account")
        title.setFont(font_title)
        title.setAlignment(Qt.AlignCenter)
        main_layout.addWidget(title)

        form_layout = QGridLayout()
        form_layout.setHorizontalSpacing(20)
        form_layout.setVerticalSpacing(15)

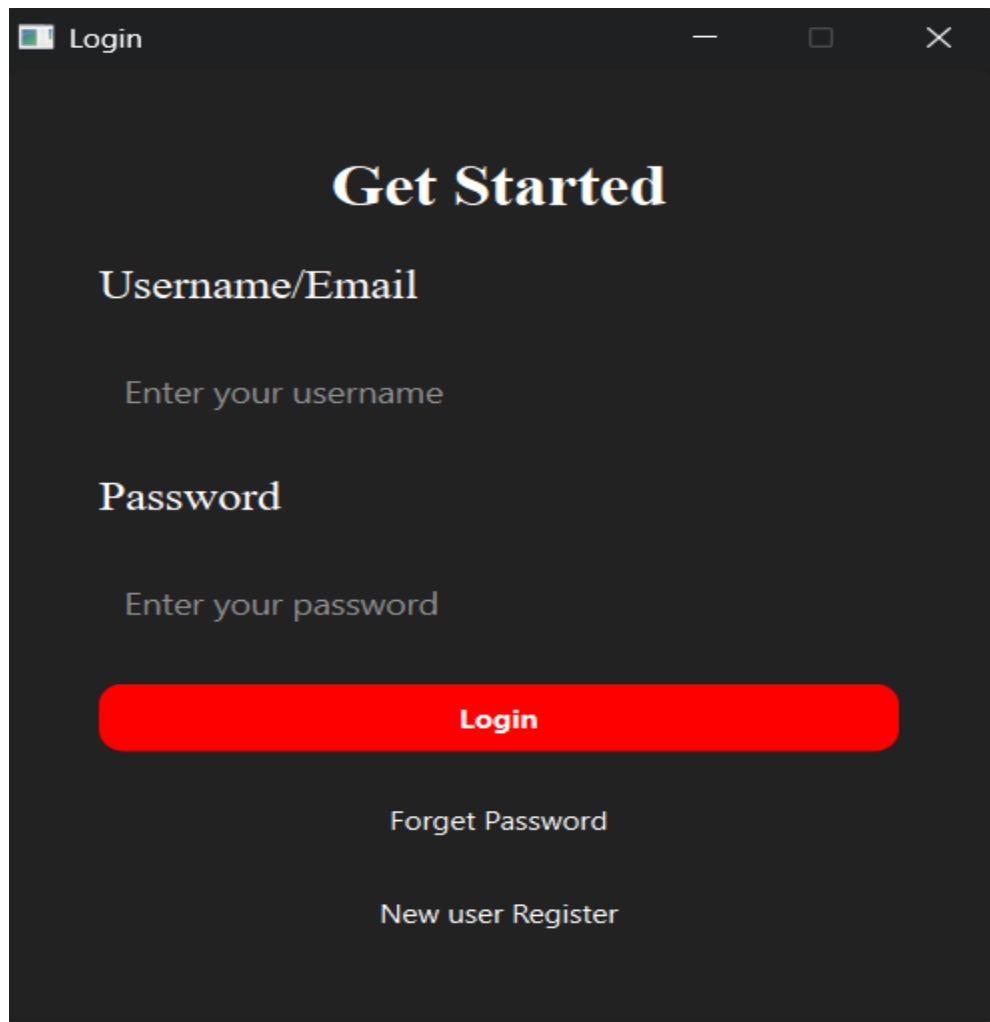
        form_layout.addWidget(self.make_label("First Name:", font_label), 0, 0)
        self.fname_entry = QLineEdit()
        self.fname_entry.setPlaceholderText("Enter your first name")
        form_layout.addWidget(self.fname_entry, 0, 1)

        form_layout.addWidget(self.make_label("Last Name:", font_label), 0, 2)
        self.lname_entry = QLineEdit()
        self.lname_entry.setPlaceholderText("Enter your last name")
        form_layout.addWidget(self.lname_entry, 0, 3)

        form_layout.addWidget(self.make_label("Email:", font_label), 1, 0)
        self.email_entry = QLineEdit()

```

## SNAPSHOTS



Login

### Get Started

Username/Email

Enter your username

Password

Enter your password

Login

[Forget Password](#)

[New user Register](#)

### Create Your Account

First Name:

Enter your first ...

Last Name:

Enter your last n...

Email:

Enter your email address

Contact No.:

Enter your phone number

Password:

Enter password

Confirm Password:

Re-enter passw...

Security Question:

Select



Security Answer:

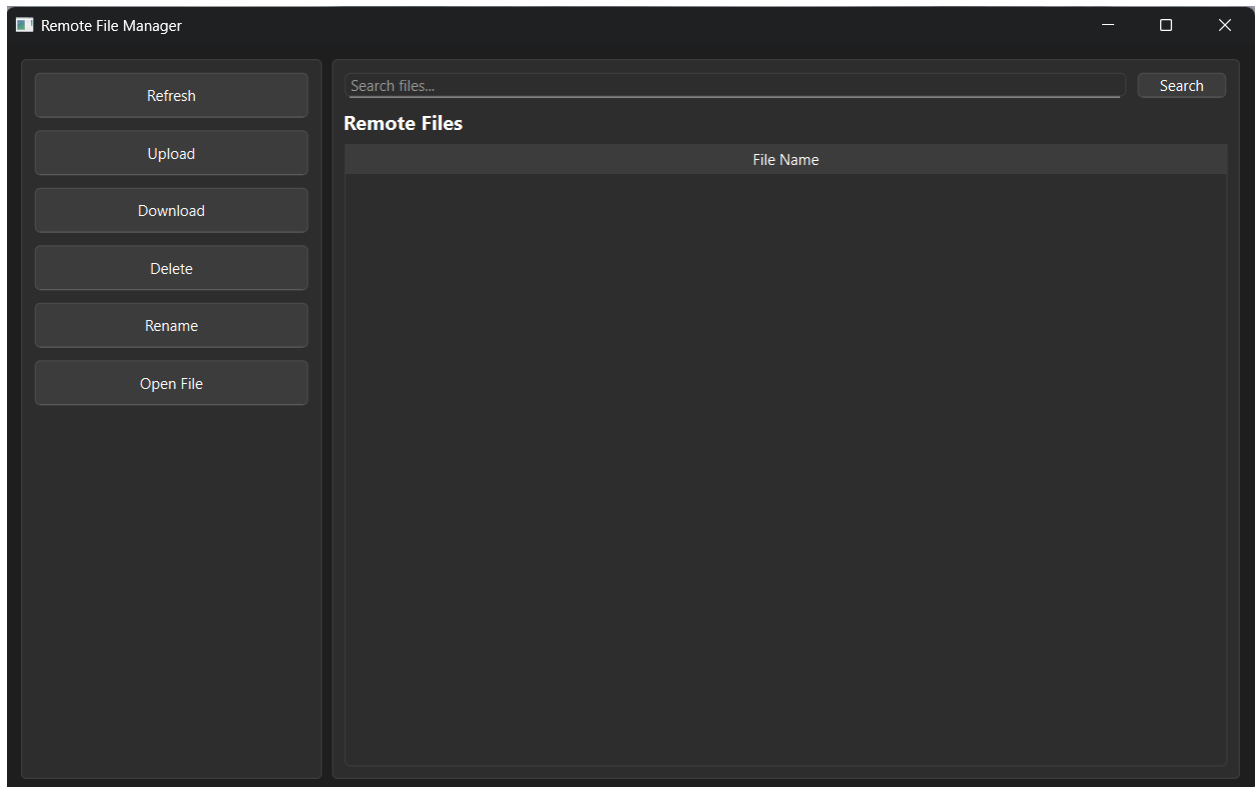
Answer to your security question

☐

I agree to the Terms Conditions

Register

Cancel



## LIMITATIONS

- **Single-threaded server:** Only one client connection is handled at a time, which limits the scalability and responsiveness of the system during concurrent access.
- **Lack of encryption:** Communication between the client and server is in plaintext, making it vulnerable to packet sniffing or man-in-the-middle attacks.
- **No cloud integration:** Files are stored solely on the local server machine without options for redundancy, scalability, or remote backup using cloud platforms.
- **Basic error handling:** The system has limited resilience to unexpected client disconnections, invalid requests, and server-side exceptions.
- **Manual configuration:** Requires manual setup of MySQL database and initial ACL records without GUI assistance.
- **Limited user interface feedback:** The client GUI lacks comprehensive notifications for success, failure, or detailed errors.
- **No file versioning:** Edits or overwrites do not maintain historical versions, increasing the risk of unintentional data loss.

## ENHANCEMENTS

- **Multi-threading:** Implement concurrent client handling using Python's threading or asyncio modules to support scalable multi-user operations.
- **TLS encryption:** Use ssl sockets to encrypt all communication and ensure secure authentication and file transfer.
- **Cloud sync:** Add synchronization with third-party storage providers like AWS S3, Google Drive, or Dropbox to provide redundancy and cross-platform availability.
- **Logging and version control:** Introduce a logging system to record actions (upload, download, edits) with timestamps and implement versioning for files to allow rollback.
- **Admin dashboard:** Develop a web-based interface (using Flask or Django) to manage users, roles, and file access rights more efficiently.
- **Improved GUI notifications:** Add status bars and success messages for better user experience.
- **Automatic database setup:** Include scripts or wizards to automatically create required MySQL tables and indexes during first-time setup.
- **Role-based access control (RBAC):** Extend ACL to support roles (e.g., admin, editor, viewer) for more streamlined permission assignment.
- **Concurrent file access control:** Implement locking mechanisms to prevent simultaneous conflicting edits on the same file.

## CONCLUSION

The Remote File Manager is a comprehensive solution that addresses the growing demand for secure and efficient remote file access and management. Through its modular architecture, the system integrates multiple technologies including socket programming, graphical user interfaces, and relational databases to provide a functional and user-friendly application.

This project successfully demonstrates the implementation of a client-server model with essential file manipulation capabilities such as upload, download, delete, rename, and edit. It also incorporates user authentication and a robust Access Control List (ACL) mechanism, ensuring that file operations are restricted based on user permissions.

The software development lifecycle followed—ranging from requirement analysis to testing and refinement—ensured systematic progress and quality control throughout the project. Each module was designed and tested independently, followed by integration testing to validate overall system performance. Feedback from early user trials was invaluable in identifying usability issues and improving the interface.

Despite its strengths, the system also revealed limitations including single-threaded server operations, lack of encryption, and absence of cloud integration. These areas highlight the potential for future enhancements. Suggested improvements such as multi-threading, TLS encryption, cloud syncing, and role-based access control would significantly increase the system's usability, scalability, and security.

In conclusion, the Remote File Manager serves as a strong foundation for academic exploration and future enterprise applications. It demonstrates how software engineering principles can be effectively applied to build practical tools that meet real-world needs. With continued development, the system holds promise for broader deployment in both educational and organizational contexts. of a distributed file management system. It successfully combines socket-based communication, GUI-based client interaction, and secure access control. While limited in scale and features, it lays the foundation for future enterprise-grade systems. With enhancements, it can evolve into a full-fledged file management suite for academic, corporate, or personal use.

## REFERENCES

1. Kurose, J. F., & Ross, K. W. (2017). Computer Networking: A Top-Down Approach. Pearson.
2. Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems. Pearson.
3. Stevens, W. R. (1998). Unix Network Programming, Volume 1: The Sockets Networking API. Prentice Hall.
4. Stallings, W. (2013). Data and Computer Communications. Pearson.
5. Geeks for Geeks. (n.d.). How to Build a File Management System in Python.
6. A review: Remote file manager for Android platform using FTP4Android. (2013, July 1).