Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)

Http File Uploading Server using Socket Programming

Course Title: Networking Lab
Course Code: CSE-312
Section: 221-D20

Students Details

Name	ID
Md Nahidul Islam	221902365
Jannatul Fardous Rimi	221902139

Submission Date: 26-12-2024
Course Teacher's Name: Sudip Chandra Ghoshal

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The proposed project aims to develop a File Upload HTTP Server that facilitates the uploading, downloading, modification, and deletion (CRUD operations) of files. The server will communicate with a client through HTTP requests and utilize Java Sockets for handling the connection between the client and the server. The system will include essential features like file validation, file size validation, authentication and authorization, file compression, and robust error handling.

The server will be built with a user-friendly graphical user interface (GUI) to simplify interactions with the system, enabling users to easily upload and manage their files. This system will also include a feature set that ensures file integrity, security, and effective resource management.

1.2 Problem Definition

In modern web-based applications, managing file uploads is a fundamental requirement. However, file uploads pose several challenges such as:

1. **Security Risks:** Malicious files or unauthorized access can lead to vulnerabilities.
2. **File Size Management:** Handling very large files without causing server overload or interruptions.
3. **Data Integrity:** Ensuring that files are not corrupted or lost during the upload process.
4. **User Experience:** Providing a smooth and reliable interface for users to manage file uploads.
5. **Compliance:** Adhering to certain rules and regulations such as file type restrictions and size limits.
6. **Error Handling:** Managing scenarios where the file upload fails or exceeds the server's capabilities.

1.3 Motivation

The motivation behind this project is to build a lightweight, secure, and efficient system for handling file uploads and related operations in a networked environment. Traditional HTTP servers (such as Apache or Nginx) and cloud storage solutions may offer complex or over-engineered solutions for simple file handling needs. This project aims to:

- **Provide a Custom Solution:** A server built specifically for file uploads with custom features like file validation, authentication, and error handling.
- **Optimize for Security:** Ensure secure file uploads through file validation, authentication, and authorization mechanisms.
- **Provide a Custom Solution:** A server built specifically for file uploads with custom features like file validation, authentication, and error handling.
- **Optimize for Security:** Ensure secure file uploads through file validation, authentication, and authorization mechanisms.
- **Manage Resources Effectively:** Handle large files and reduce unnecessary storage by integrating file compression.
- **Improve User Experience:** Provide a user-friendly graphical interface for managing files and operations.

By combining these features, the system can handle file upload operations efficiently, securely, and in a user-friendly manner.

1.4 Design Goals/Objective

The project will be designed with the following goals in mind::

- **Security and Authentication:** Implement authentication and authorization checks to ensure that only authorized users can upload, modify, or delete files. This includes validating the user's credentials and file integrity.
- **File Handling and Validation:** File validation will involve checking the file type, ensuring it matches allowed types (e.g., image, text, PDF) File Size Validation will restrict uploads that exceed a defined limit.
- **Error Handling:** Implement error handling for common issues such as file size exceeding limits, invalid file types, network interruptions, and upload failures.
- **User Interface:** Develop a GUI for the client to interact with the file upload server, providing features like file selection, status tracking, and error reporting.

- **CRUD Operations:** The server will support Create (upload), Read (download), Update (modify), and Delete (remove) operations for managing files.

- **Scalability and Performance:** Ensure the server is optimized for handling a variety of file sizes and can support multiple users concurrently.

1.5 Tools and Technology

The following tools and technologies will be used to build the project:

- **Programming Language: Java**
  Java provides the necessary libraries for socket programming and creating a robust server-client architecture.

- **Socket Programming:**
  Java Sockets (TCP/IP) will be used to create the communication channel between the server and the client.
  The HTTP protocol will be implemented over the socket connection to ensure standard web communication.

1.6 Application

The File Upload HTTP Server with CRUD operations will serve as a backend solution for web or local applications that require reliable file management features. The system can be adapted for various use cases:

- For internal corporate systems that require file storage and management, with specific validation rules for security and compliance.

- Platforms that require students or users to upload assignments, documents, or media files, with size restrictions and validation features.

- A collaborative file-sharing system where multiple users upload, modify, and delete shared files.

- Used for managing large backups or archives, with the added benefit of file compression to save storage space.

Chapter 2

Design/Development/Implementation of the Project

2.1 Implementation

Server:

```
import java.io.*;
import java.net.*;
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;
import java.security.Key;
import java.util.Base64;

public class Server {

    private static final int SERVER_PORT = 8080;
    private static final String STORAGE_DIR =
       "server_files";
    private static final String SECRET_KEY =
       "1234567812345678"; // 16-byte key for AES

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new
           ServerSocket(SERVER_PORT)) {
            System.out.println("Server is running on
               port " + SERVER_PORT);

            File storageDir = new File(STORAGE_DIR);
            if (!storageDir.exists()) {
                storageDir.mkdir();
            }

            while (true) {
                Socket clientSocket =
```

```
                serverSocket.accept();
            System.out.println("Client connected: "
                + clientSocket.getInetAddress());

            new Thread(() ->
                handleClient(clientSocket)).start();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void handleClient(Socket
    clientSocket) {
    try (DataInputStream in = new
        DataInputStream(clientSocket.getInputStream());
         DataOutputStream out = new
            DataOutputStream(clientSocket.getOutputStream()))
            {

        String operation = in.readUTF();
        System.out.println("Operation received: " +
            operation);

        switch (operation) {
            case "UPLOAD":
                handleUpload(in, out);
                break;
            case "DOWNLOAD":
                handleDownload(in, out);
                break;
            case "DELETE":
                handleDelete(in, out);
                break;
            case "UPDATE":
                handleUpdate(in, out);
                break;
            default:
                out.writeUTF("Invalid operation");
                break;
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
```

```
            }
        }
    }

    private static void handleUpload(DataInputStream
       in, DataOutputStream out) throws IOException {
        String fileName = in.readUTF();
        long fileSize = in.readLong();

        File file = new File(STORAGE_DIR, fileName);
        try (FileOutputStream fileOut = new
           FileOutputStream(file);
             CipherOutputStream cipherOut = new
                CipherOutputStream(fileOut,
                initCipher(Cipher.ENCRYPT_MODE))) {
            byte[] buffer = new byte[4096];
            int bytesRead;
            long remainingBytes = fileSize;

            while (remainingBytes > 0 && (bytesRead =
               in.read(buffer, 0, (int)
               Math.min(buffer.length,
               remainingBytes))) != -1) {
                cipherOut.write(buffer, 0, bytesRead);
                remainingBytes -= bytesRead;
            }
        }

        out.writeUTF("File uploaded and encrypted
           successfully.");
    }

    private static void handleDownload(DataInputStream
       in, DataOutputStream out) throws IOException {
        String fileName = in.readUTF();
        File file = new File(STORAGE_DIR, fileName);

        if (file.exists() && file.isFile()) {
            out.writeUTF("File found");
            out.writeLong(file.length());

            try (FileInputStream fileIn = new
               FileInputStream(file);
                 CipherInputStream cipherIn = new
                    CipherInputStream(fileIn,
                    initCipher(Cipher.DECRYPT_MODE))) {
                byte[] buffer = new byte[4096];
                int bytesRead;
```

```
            while ((bytesRead =
               cipherIn.read(buffer)) != -1) {
                out.write(buffer, 0, bytesRead);
            }
        }
    } else {
        out.writeUTF("File not found");
    }
}

private static void handleDelete(DataInputStream
   in, DataOutputStream out) throws IOException {
    String fileName = in.readUTF();
    File file = new File(STORAGE_DIR, fileName);

    if (file.exists() && file.isFile()) {
        if (file.delete()) {
            out.writeUTF("File deleted
               successfully.");
        } else {
            out.writeUTF("Failed to delete file.");
        }
    } else {
        out.writeUTF("File not found.");
    }
}

private static void handleUpdate(DataInputStream
   in, DataOutputStream out) throws IOException {
    String fileName = in.readUTF();
    File file = new File(STORAGE_DIR, fileName);

    if (file.exists() && file.isFile()) {
        if (file.delete()) {
            out.writeUTF("File ready for update.");
            handleUpload(in, out);
        } else {
            out.writeUTF("Failed to prepare file
               for update.");
        }
    } else {
        out.writeUTF("File not found.");
    }
}

private static Cipher initCipher(int mode) {
    try {
```

```
            Key key = new
               SecretKeySpec(SECRET_KEY.getBytes(),
               "AES");
            Cipher cipher = Cipher.getInstance("AES");
            cipher.init(mode, key);
            return cipher;
        } catch (Exception e) {
            throw new RuntimeException("Error
               initializing cipher", e);
        }
    }
}
```

Client:

```
    import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

public class ClientThread extends JFrame {

    private static final String SERVER_ADDRESS =
       "localhost";
    private static final int SERVER_PORT = 8080;

    private JButton uploadButton, downloadButton,
       deleteButton, updateButton;
    private JTextArea consoleArea;
    private JFileChooser fileChooser;

    public ClientThread() {
        setTitle("File Management Client");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        fileChooser = new JFileChooser();

        // Create the layout components
        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());

        // Create a label for the project name
        JLabel projectNameLabel = new JLabel("File
           Management System", SwingConstants.CENTER);
        projectNameLabel.setFont(new Font("Arial",
           Font.BOLD, 20));
```

```
        projectNameLabel.setForeground(Color.BLUE); //
            Set text color
        panel.add(projectNameLabel, BorderLayout.NORTH);

        // Create a console area for output logs
        consoleArea = new JTextArea();
        consoleArea.setEditable(false);
        consoleArea.setFont(new Font("Arial",
            Font.PLAIN, 16)); // Set the font size
        consoleArea.setLineWrap(true);
        consoleArea.setWrapStyleWord(true);
        JScrollPane scrollPane = new
            JScrollPane(consoleArea);
        panel.add(scrollPane, BorderLayout.CENTER);

        // Create buttons for operations
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout());

        uploadButton = new JButton("Upload File");
        uploadButton.addActionListener(e ->
            performOperation("UPLOAD"));
        buttonPanel.add(uploadButton);

        downloadButton = new JButton("Download File");
        downloadButton.addActionListener(e ->
            performOperation("DOWNLOAD"));
        buttonPanel.add(downloadButton);

        deleteButton = new JButton("Delete File");
        deleteButton.addActionListener(e ->
            performOperation("DELETE"));
        buttonPanel.add(deleteButton);

        updateButton = new JButton("Update File");
        updateButton.addActionListener(e ->
            performOperation("UPDATE"));
        buttonPanel.add(updateButton);

        panel.add(buttonPanel, BorderLayout.SOUTH);

        add(panel);
    }

    private void performOperation(String operation) {
        try (Socket socket = new Socket(SERVER_ADDRESS,
            SERVER_PORT);
              DataInputStream in = new
```

```
            DataInputStream(socket.getInputStream());
         DataOutputStream out = new
            DataOutputStream(socket.getOutputStream()))
            {

        out.writeUTF(operation);
        switch (operation) {
            case "UPLOAD":
                uploadFile(out, in);
                break;
            case "DOWNLOAD":
                downloadFile(out, in);
                break;
            case "DELETE":
                deleteFile(out, in);
                break;
            case "UPDATE":
                updateFile(out, in);
                break;
            default:
                displayMessage("Invalid operation",
                   Color.RED);
                break;
        }

    } catch (IOException e) {
        e.printStackTrace();
        displayMessage("Error: " + e.getMessage(),
           Color.RED);
    }
}

private void uploadFile(DataOutputStream out,
   DataInputStream in) throws IOException {
    int result = fileChooser.showOpenDialog(this);
    if (result == JFileChooser.APPROVE_OPTION) {
        File file = fileChooser.getSelectedFile();
        if (file.exists()) {
            out.writeUTF(file.getName());
            out.writeLong(file.length());

            try (FileInputStream fileIn = new
               FileInputStream(file)) {
                byte[] buffer = new byte[4096];
                int bytesRead;

                while ((bytesRead =
                   fileIn.read(buffer)) != -1) {
```

```
                    out.write(buffer, 0, bytesRead);
                }
            }

            displayMessage("File uploaded: " +
               file.getName(), Color.GREEN);
            displayMessage(in.readUTF(),
               Color.BLUE);
        } else {
            displayMessage("File not found.",
               Color.RED);
        }
    }
}

private void downloadFile(DataOutputStream out,
   DataInputStream in) throws IOException {
    String filename =
       JOptionPane.showInputDialog(this, "Enter
       filename to download:");
    if (filename != null && !filename.isEmpty()) {
        out.writeUTF(filename);

        String response = in.readUTF();
        if ("File found".equals(response)) {
            long fileSize = in.readLong();
            File file = new File("downloaded_" +
               filename);

            try (FileOutputStream fileOut = new
               FileOutputStream(file)) {
                byte[] buffer = new byte[4096];
                int bytesRead;
                long remainingBytes = fileSize;

                while (remainingBytes > 0 &&
                   (bytesRead = in.read(buffer, 0,
                   (int) Math.min(buffer.length,
                   remainingBytes))) != -1) {
                    fileOut.write(buffer, 0,
                       bytesRead);
                    remainingBytes -= bytesRead;
                }
            }

            displayMessage("File downloaded
               successfully: " + filename,
               Color.GREEN);
```

```
            } else {
                displayMessage("File not found on
                    server.", Color.RED);
            }
        }
    }

    private void deleteFile(DataOutputStream out,
        DataInputStream in) throws IOException {
        String filename =
            JOptionPane.showInputDialog(this, "Enter
            filename to delete:");
        if (filename != null && !filename.isEmpty()) {
            out.writeUTF(filename);
            displayMessage(in.readUTF(), Color.GREEN);
        }
    }

    private void updateFile(DataOutputStream out,
        DataInputStream in) throws IOException {
        String filename =
            JOptionPane.showInputDialog(this, "Enter
            filename to update:");
        if (filename != null && !filename.isEmpty()) {
            out.writeUTF(filename);
            displayMessage(in.readUTF(), Color.GREEN);
            uploadFile(out, in); // Upload new version
        }
    }
    // Method to display messages with color and larger
        text
    private void displayMessage(String message, Color
        color) {
        consoleArea.setText(message);
        consoleArea.setForeground(color); // Set text
            color
        consoleArea.setFont(new Font("Arial",
            Font.BOLD, 18)); // Increase font size for
            messages
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            ClientThread clientUI = new ClientThread();
            clientUI.setVisible(true);
        });
    }
}
```

2.2 Result

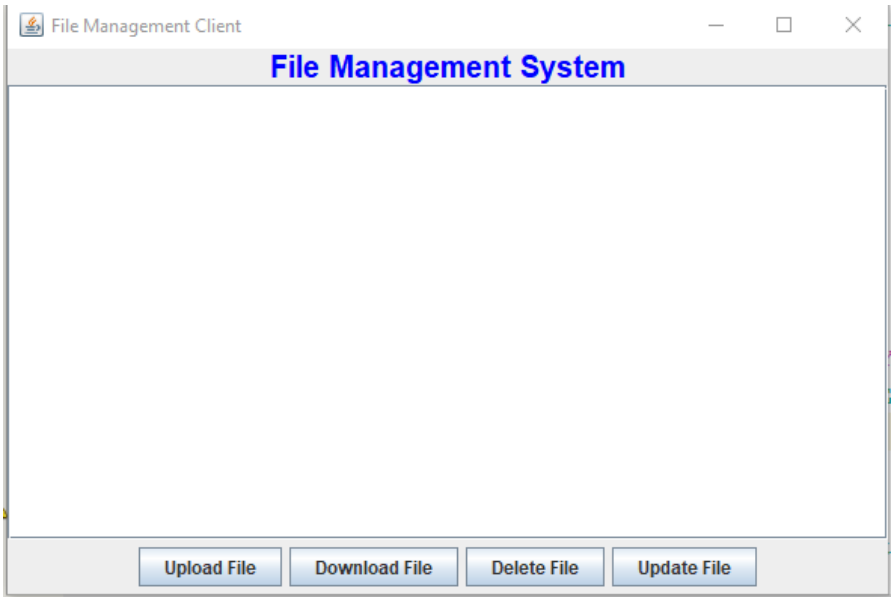

Figure 2.1: User Interface

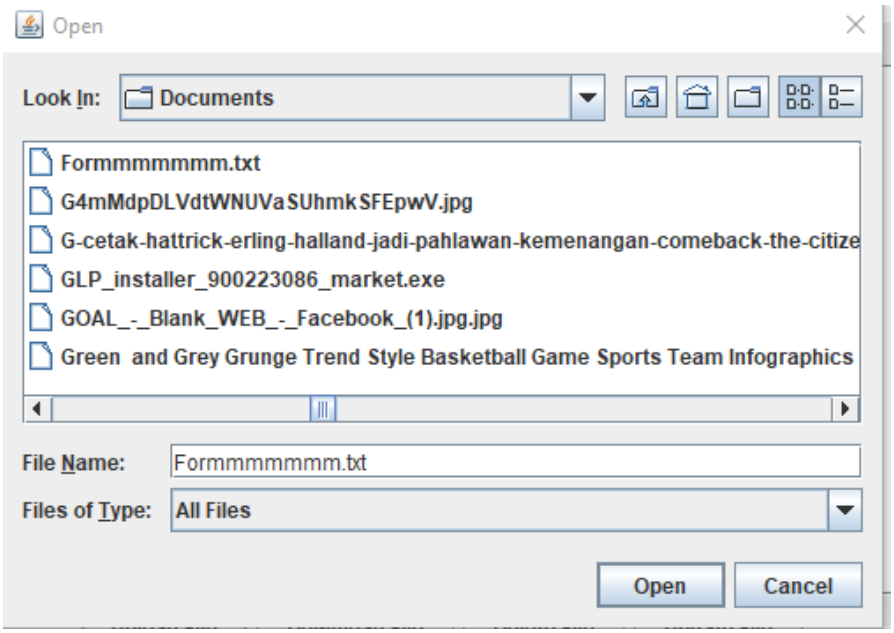

Figure 2.2: File Upload

Figure 2.3: After file upload successful message

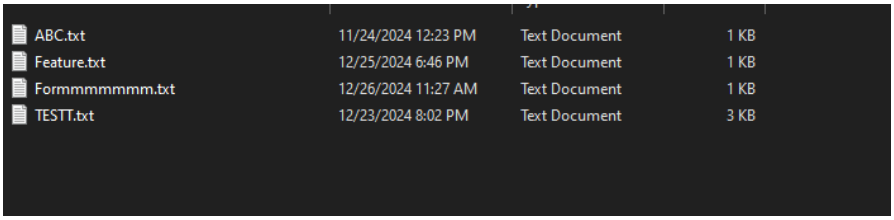

Figure 2.4: File save in directories

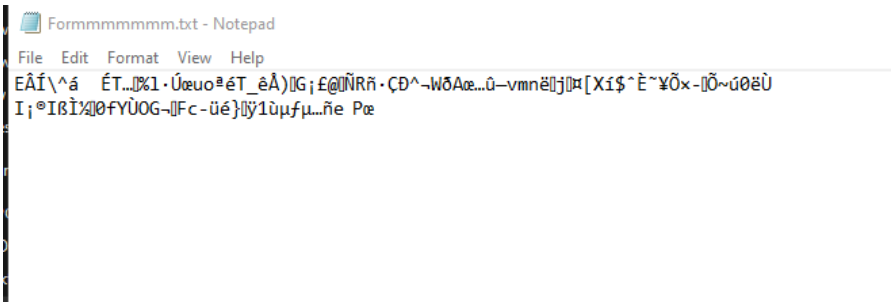

Figure 2.5: Encrypted File

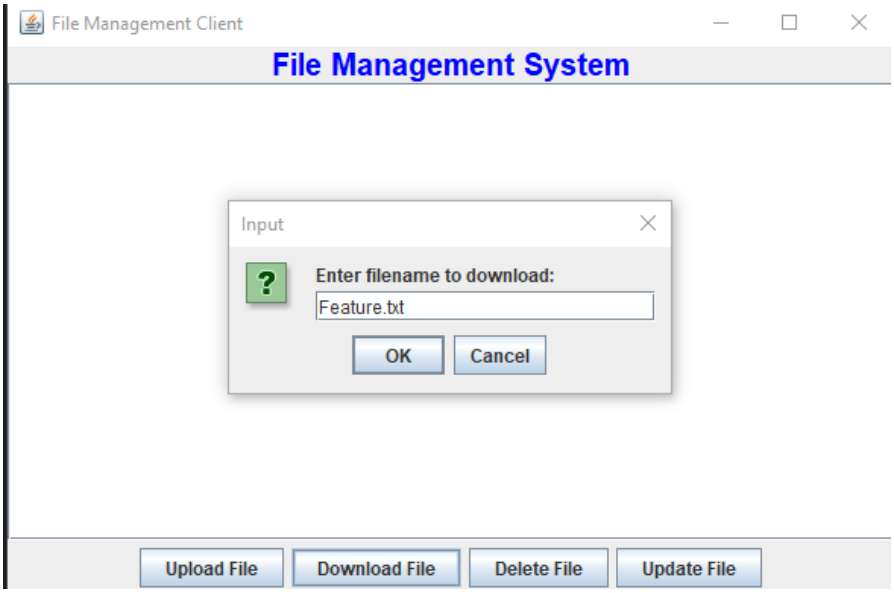

Figure 2.6: File Download

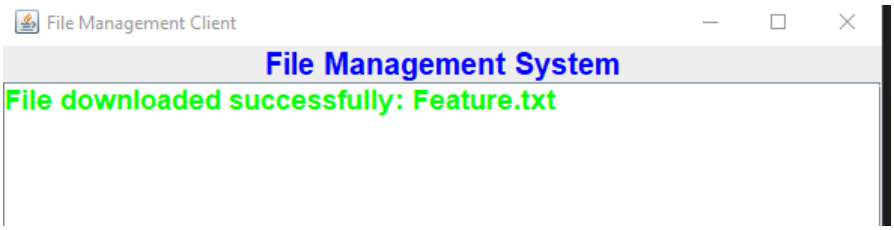

Figure 2.7: After file download successful message

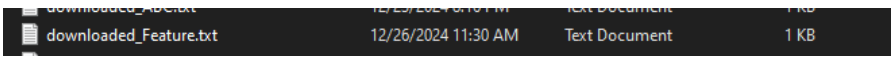

Figure 2.8: File in Local Computer

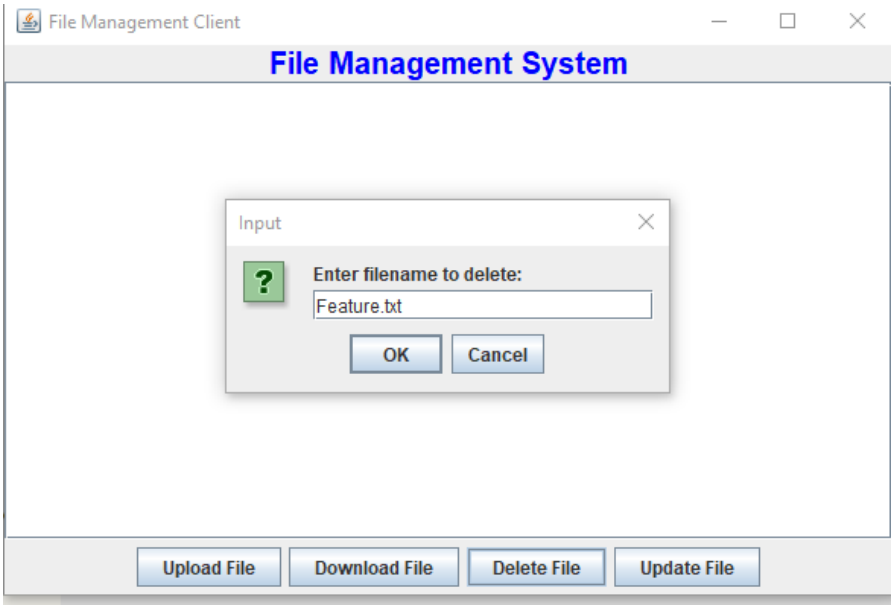

Figure 2.9: File Delete

Figure 2.10: After file delete successful message

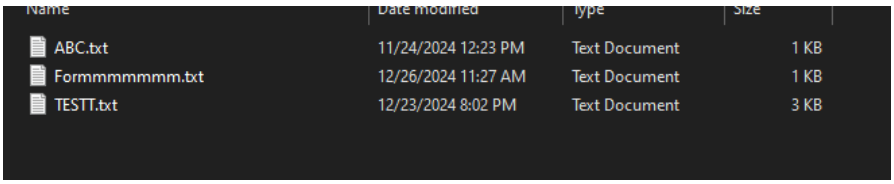

Figure 2.11: File in server files (Feature.txt is removed)

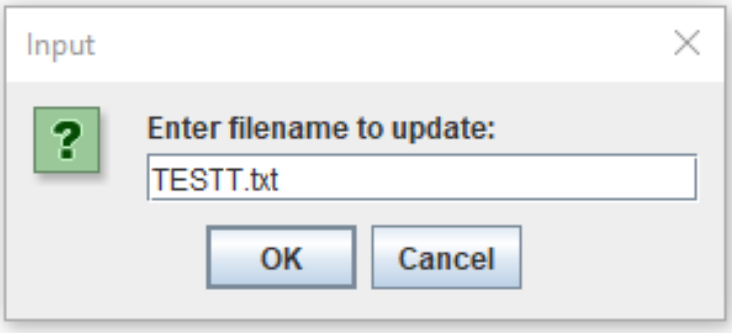

Figure 2.12: Update a File

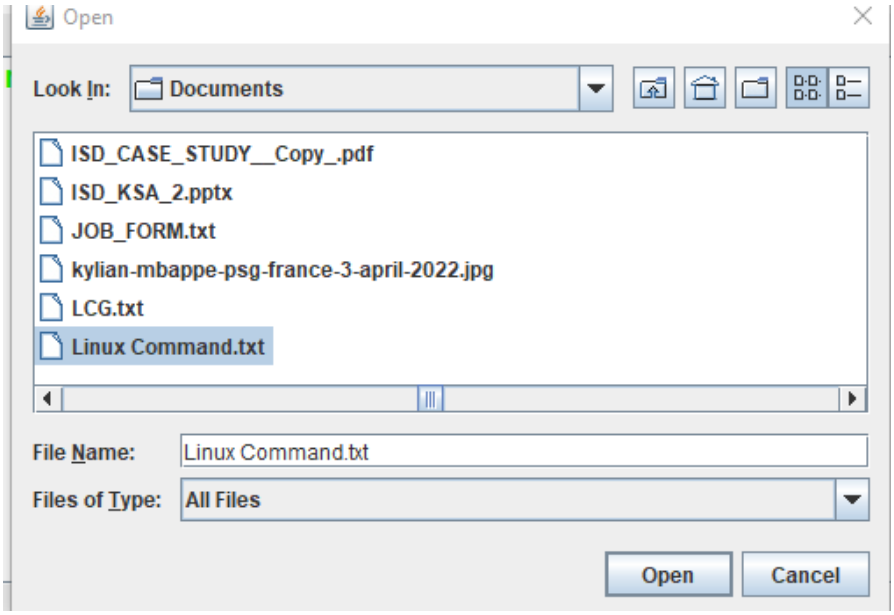

Figure 2.13: Select the file we want to update

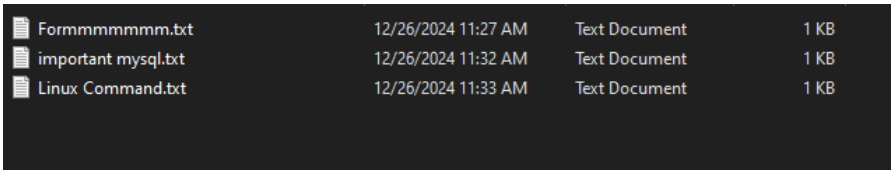

Figure 2.14: File in server files (UPDTAED)

Chapter 3

Conclusion

3.1 Discussion

The project "HTTP File Uploading Server using Socket Programming" aims to establish a secure and efficient file management system with the ability to upload, download, update, and delete files. By utilizing Java Sockets, the server can handle communication with clients, making it scalable and effective for handling multiple users in real-time.
Key Features of the System:

- **File Upload and Download:** The client-server model facilitates secure and reliable transfer of files. The server accepts HTTP requests from the client, processes these requests, and sends appropriate responses (e.g., file upload acknowledgment or download data).
- **AES Encryption/Decryption:** To ensure data security and privacy, the system uses the AES (Advanced Encryption Standard) algorithm. This encryption protects sensitive data during file transfers, preventing unauthorized access to files, even when they are intercepted during transit.
- **User Interface (UI):** The graphical user interface provides users with an intuitive way to interact with the server. The UI includes options for uploading, downloading, modifying, and deleting files. It enhances the user experience by presenting clear feedback on operations (success, failure, etc.) and ensuring that users can easily navigate through the system.
- **Socket Programming:** The use of socket programming is central to the communication between the client and server. Java Sockets enable real-time, two-way communication, allowing the client to send requests, and the server to respond with the necessary actions. This method is efficient, even for real-time file management.
- **CRUD Operations:** The system allows the client to perform the basic CRUD (Create, Read, Update, Delete) operations on files stored on the server. This makes the system versatile, as users can add new files, retrieve existing ones, update file content, or remove files as required.

3.2 Limitations

While the project demonstrates fundamental file management with AES encryption, there are a few limitations:

1. **File Size Limitation:** Due to the way sockets handle data, larger files may encounter delays or memory limitations when being transmitted. The server's ability to handle large files can be impacted by network bandwidth and server resources, limiting scalability for very large files.

2. **Security Concerns:** Although AES encryption is used for securing the data during transmission, the system may still be vulnerable to certain attacks (e.g., man-in-the-middle attacks, if the keys are not managed securely). Furthermore, this project only covers encryption during data transfer, and there is no in-depth protection for the server's file storage system

3. **Dependence on HTTP:** The use of HTTP, while useful for file transfer, is not the most efficient protocol for large-scale or high-performance systems. HTTP's overhead can be a limitation when dealing with large amounts of data or frequent requests.

3.3 Scope of Future Work

There are numerous possibilities for extending and enhancing the system, including:

1. **Scalability Improvements:** To handle a large number of simultaneous clients and large file uploads/downloads, the system could be adapted to use a multi-threaded or multi-process approach. Implementing a thread pool or using a more scalable framework like Netty or Java NIO can enhance performance under heavy load.

2. **Advanced Security Features:** Instead of using symmetric encryption (AES), incorporating **RSA or ECC (Elliptic Curve Cryptography)** for asymmetric encryption could improve key management and security.

3. **Support for Large Files:** Implementing file chunking and resumable uploads can help overcome issues related to large files. This would involve splitting files into smaller chunks, sending them separately, and then reassembling them on the server.

4. **File Storage Optimization:** The current system stores files in a basic manner. In a real-world application, implementing **database-backed file storage** (e.g., using SQL or NoSQL) would allow for more complex queries, better organization, and more efficient management of file metadata.