Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (spring, Year: 2025), B.Sc. in CSE (Day)

HTTP File Uploading Server using Socket Programming

Course Title: Computer Networking Lab
Course Code: CSE-312
Section: 223-D1

Students Details

Name	ID
Sheikh Mahatab Uddin	221902314
Hridoy Mia	223902010

Submission Date: 13/05/2025
Course Teacher's Name: Md. Sabbir Hosen Mamun

[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 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 complexor over-engineered solutions for simple file handling needs. This project aims to:

• **PProvide 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.

1.3 Problem Definition

1.3.1 Problem Statement

This project is a secure File Management System built using Java Socket Programming.It enables users to upload, download, update, and delete files through a simple Java Swing GUI. The server stores all files in encrypted form using AES encryption, ensuring data confidentiality. The system supports multiple clients via multithreading and provides real-time feedback for all operations, making it both user-friendly and secure.

1.3.2 Complex Engineering Problem

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributess	**Explain how to address**
P1: Depth of knowledge required	This project requires knowledge of Java programming, socket communication, multithreading, Swing GUI, file I/O, and basic AES encryption.
P2: Range of conflicting requirements	The project must balance ease of use via a simple GUI with secure file transmission, efficient network communication, and robust error handling across multiple operations like upload,download, delete, and update.
P3: Depth of analysis required	Requires moderate analysis to integrate secure file handling, encryption, and client server communication effectively.
P4: Familiarity of issues	Requires basic familiarity with network programming, file I/O, and encryption techniques.
P5: Extent of applicable codes	
P6: Extent of stakeholder involvement and conflicting requirements	—-
P7: Interdependence	The project components, like client-server communication, file operations, and encryption, are tightly interdependent for smooth functionality.

1.4 Design Goals/Objectives

The design goals/objectives of this project are: **Efficient File Management:** To enable seamless file upload, download, deletion,and updates between a client and server.

Data Security: Implement AES encryption to ensure secure transmission of files over the network.

User-Friendly Interface: Provide an intuitive GUI for users to interact with the system easily.

Scalability: Design the system to handle multiple client requests simultaneously with reliable performance.

Error Handling: Ensure proper error handling and informative feedback to users in case of failures.

Server Reliability: Maintain consistent server uptime and data integrity during file operations.

1.5 Application

The application of this project is to provide a secure and efficient file management system for organizations or individuals that need to upload, download, update, and delete files over a network. It can be used in:

Cloud Storage Systems: Facilitating secure file management in cloud services.

File Sharing Platforms: Enabling users to easily share and manage files in a secure environment.

Backup Solutions: Ensuring safe storage and retrieval of critical data in case of system failures.

Enterprise File Management: Allowing businesses to securely manage documents and data across multiple locations or departments. Personal File Management: Enabling individuals to securely store and organize personal files remotely.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

In today's digital era, secure and efficient file management is essential for both individuals and organizations. This project presents a network-based File Management System that enables users to perform core operations such as uploading, downloading, updating, and deleting files over a client-server architecture. The system is designed with a user-friendly graphical interface on the client side and incorporates data encryption on the server side to ensure secure file transfers. Built using Java and socket programming, the project demonstrates key concepts in networking, encryption, and GUI development, making it both practical and educational. [1] [2] [3].

2.2 Project Details

This project is a Java-based File Management System using Client-Server architecture and Socket Programming. It allows users to upload, download, delete, and update files over a network through a user-friendly Java Swing GUI.

The Client provides buttons for each operation, a console for logs, and a file chooser for selecting files. The Server, running on port 8080, handles multiple client requests using threads and stores files in a secure directory. It uses AES encryption to ensure secure file storage and transmission.

Key Features: Upload: Sends and encrypts the file on the server.

Download: Retrieves and decrypts the file from the server.

Delete: Removes a file from the server. Update: Replaces an existing file with a new version.

Technologies Used: Java, Swing, Socket Programming, AES encryption, NetBeans IDE

This project demonstrates secure, real-time file management and serves as a foundation for secure file-sharing systems.

2.3 Implementation

Server Implementation:

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$_{P}ORT = 8080$;

private static final String STORAGE$_{D}IR = ”server_{f}iles”$;

private static final String SECRET$_{K}EY = ”1234567812345678”; //16-bytekeyforAES$

public static void main(String[] args)

try (ServerSocket serverSocket = new ServerSocket(SERVER$_{P}ORT))$

System.out.println("Server is running on port " + SERVER$_{P}ORT)$;

File storageDir = new File(STORAGE$_{D}IR)$;

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_MOL
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$_{M}ODE)))$

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(SECRETKEY.getBytes(),”AES”);
Cipher cipher = Cipher.getInstance("AES");
cipher.init(mode, key);
return cipher;
catch (Exception e)
throw new RuntimeException("Error initializing cipher", e);
```

Client Handler Implementation:

```
package project;
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;
import java.io.*;
import java.net.Socket;
import java.security.*;
import java.io.File;
public class ClientHandler implements Runnable
private final Socket clientSocket;
private static final String STORAGEDIR = ”serverfiles”;
private static final String SECRETKEY = ”1234567812345678”;
public ClientHandler(Socket socket)
this.clientSocket = socket;
@Override
public void run()
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);
```

case "DOWNLOAD" -> handleDownload(in, out);

case "DELETE" -> handleDelete(in, out);

case "UPDATE" -> handleUpdate(in, out);

default -> out.writeUTF("Invalid operation");

catch (IOException e)

System.out.println("Client error: " + e.getMessage());

finally

try

clientSocket.close();

catch (IOException ignored)

private 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$_{MOD}$

byte[] buffer = new byte[4096];

int bytesRead;

long remaining = fileSize;

while (remaining > 0 (bytesRead = in.read(buffer, 0, (int) Math.min(buffer.length, remaining))) != -1)

cipherOut.write(buffer, 0, bytesRead);

remaining -= bytesRead;

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

private void handleDownload(DataInputStream in, DataOutputStream out) throws IOException

String fileName = in.readUTF();

File file = new File(STORAGE$_{DIR}, fileName)$;

if (file.exists())

out.writeUTF("File found");

out.writeLong(file.length());

try (FileInputStream fileIn = new FileInputStream(file);

CipherInputStream cipherIn = new CipherInputStream(fileIn, initCipher(Cipher.DECRYPT$_{M}ODE)))$

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 void handleDelete(DataInputStream in, DataOutputStream out) throws IOException

String fileName = in.readUTF();

File file = new File(STORAGE$_{DIR}, fileName)$;

if (file.exists() file.delete())

out.writeUTF("File deleted successfully.");

else

out.writeUTF("File not found or deletion failed.");

private void handleUpdate(DataInputStream in, DataOutputStream out) throws IOException

String fileName = in.readUTF();

File file = new File(STORAGE$_{DIR}, fileName)$;

if (file.exists() file.delete())

out.writeUTF("File ready for update.");

handleUpload(in, out);

else

out.writeUTF("File not found.");

private Cipher initCipher(int mode)

try

Key key = new SecretKeySpec(SECRET$_{K}EY.getBytes(), ”AES$”);

Cipher cipher = Cipher.getInstance("AES");

cipher.init(mode, key);

return cipher;

catch (Exception e)

throw new RuntimeException("Cipher init failed", e);

:Thread Handling

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$_{A}DDRESS, SERVER_{P}ORT)$;
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$_{O}PTION$)

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 IOEx-
ception

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 IOEx-
ception

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 displayMes-
sage(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);

);

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

The project was developed and simulated using NetBeans IDE 17 with Java JDK 21 on a Windows 10 environment. The simulation setup includes a client and server communicating over a local network using TCP sockets. File transfer operations (upload, download, delete, update) were tested by running the server program and executing multiple client requests through the GUI. The encryption and decryption functionalities were also validated using various file sizes to evaluate security and efficiency.

3.2 Results Analysis/Testing

3.2.1 Result_portion_1

File Upload

The upload functionality was tested by selecting a file from the client GUI. The file was successfully transferred to the server and encrypted using AES encryption.

GUI showing upload confirmation and server-side file received.

Discussion: This confirms that the encryption and transmission processes are functional, with the server storing the file securely.

3.2.2 Result_portion_2

File Download

Upon entering the filename in the client interface, the server decrypted the requested file and sent it back.

GUI showing file download success, downloaded file visible in directory.

Discussion: The decryption and download functionality ensures that clients receive readable files, verifying AES decryption works as expected.

3.2.3 Result_portion_3

Delete and Update Operations

Delete and update operations were validated by entering filenames via GUI prompts. The server successfully removed or replaced files upon request.

Client console confirming deletion/update, server directory showing updated content.

Discussion: This demonstrates that file manipulation requests are being handled correctly by the server, ensuring real-time content management over the network.

3.3 Results Overall Discussion

The project successfully handles secure file operations—upload, download, delete, and update—via a client GUI and encrypted server communication. Results confirm functionality, though minor issues like limited log display and lack of overwrite confirmation were observed. Overall, the system meets its objectives effectively.

Chapter 4

Conclusion

4.1 Discussion

This chapter summarizes the development and testing of a secure file management system using Java socket programming. The project implemented encrypted file operations—uploading, downloading, deleting, and updating—through a user-friendly client GUI and a multithreaded server. The results demonstrated that the system performs reliably under various scenarios, with encryption successfully securing data transfer. Minor limitations were observed, but the overall objectives were achieved, validating the design and implementation approach.

4.2 Limitations

Despite the system's successful functionality, there are several limitations. First, the encryption uses a fixed symmetric key, which poses security risks if the key is exposed. Second, the client-server communication does not use SSL/TLS, making it vulnerable to network-level attacks. Third, the system lacks user authentication, meaning any client can connect and access file operations without verification. Finally, the error handling is basic and may not gracefully manage complex or unexpected network failures, limiting its robustness in real-world deployment. These limitations highlight areas for future enhancement.

4.3 Scope of Future Work

The project can be extended in several ways to enhance its functionality and security. Future work may include implementing user authentication and role-based access control to restrict unauthorized usage. Additionally, integrating secure communication using SSL/TLS would protect data transmission over the network. A graphical dashboard for server-side monitoring and file management could also be added. Moreover, enabling file version control, supporting large file transfers with resume capability, and migrating to cloud-based storage are potential future improvements to increase scalability and usability.

References

[1] Uthayasankar Sivarajah, Muhammad Mustafa Kamal, Zahir Irani, and Vishanth Weerakkody. Critical analysis of big data challenges and analytical methods. *Journal of Business Research*, 70:263–286, 2017.

[2] Douglas Laney. 3d data management: controlling data volume, velocity and variety. gartner, 2001.

[3] MS Windows NT kernel description. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm. Accessed Date: 2010-09-30.

Java Socket Programming Networking Oracle Java Tutorials – Networking: https://docs.oracle.com/jav

Baeldung – A Guide to Java Sockets: https://www.baeldung.com/a-guide-to-java-sockets

Java File I/O Oracle Java Tutorials – File I/O (NIO.2): https://docs.oracle.com/javase/tutorial/essential/i

Java Cryptography (AES) Oracle Java Tutorials – Cryptography: https://docs.oracle.com/javase/8/docs/

Baeldung – AES Encryption and Decryption in Java: https://www.baeldung.com/java-aes-encryption-decryption

Java Swing (GUI Programming) Oracle Java Swing Tutorial: https://docs.oracle.com/javase/tutorial/uis

GeeksforGeeks – Java Swing Introduction: https://www.geeksforgeeks.org/java-swing/

Client-Server Examples JournalDev – Java Socket Server Example: https://www.journaldev.com/741/ja socket-server-client-read-write-example

CodeJava – Java Socket Programming Examples: https://www.codejava.net/java-se/networking/java-socket-server-examples-tcp-ip