```
import java.io.*;
import java.net.*;
import java.text.*;
import java.util.*;

//Mohamed-Macbook-Air.local

class ClientSender implements Runnable {
	// Directory to store sent emails
	private static final String SENDER_DIRECTORY = "sender_directory";
	// Initial sequence number
	private static int sequenceNumber = 0;
	@Override
	public void run() {
		DatagramSocket clientSocket = null;
		Scanner scanner = new Scanner(System.in);
		InetAddress serverAddress = null;
		int serverPort = 25;
		try {
			// Print the host name where the mail client is running
			System.out.println("Mail Client starting on host: " +
					InetAddress.getLocalHost().getHostName());
			boolean isConnected = false;
			while (!isConnected) {
				try {
					// Prompt user to input the mail server's hostname
					System.out.print("Type name of Mail server: ");
					String serverHostName = scanner.nextLine();
					// Get the server address based on the provided
hostname
					serverAddress =
InetAddress.getByName(serverHostName);
					clientSocket = new DatagramSocket();
					// Perform handshake with the server
					performHandshake(clientSocket, serverAddress,
serverPort);
					isConnected = true; // If handshake is successful, exit
the loop
				} catch (UnknownHostException e) {
					System.err.println("Error: Invalid hostname. Please try
again.");
				}
			}
			while (true) {
```

```
                                System.out.println("\nCreating New Email..");
                                System.out.print("To: ");
                                String to = scanner.nextLine();
                                // Check if recipient's email exists in the mapping
                                if (!isValidRecipient(to)) {
                                        System.out.println("550 Error : Email mapping not
found");
                                        continue; // Skip sending the email
                                }
                                // Prompt user to input sender, subject, and body
                                System.out.print("From: ");
                                String from = scanner.nextLine();
                                System.out.print("Subject: ");
                                String subject = scanner.nextLine();
                                System.out.println("Body: ");
                                String body = scanner.nextLine();
                                // Prompt user for attachment file path
                                System.out.print("Attachment File Path (Leave blank if no
attachment): ");
                                String attachmentPath = scanner.nextLine();
                                String attachmentData = "";
                                // Read and encode attachment file if provided
                                if (!attachmentPath.isEmpty()) {
                                        attachmentData =
encodeAttachment(attachmentPath);
                                }

                                // Construct email message with attachment
                                String smtpMessage = "SEQUENCE:" + sequenceNumber +
"\r\n" +
                                                "TO:" + to + "\r\n" +
                                                "FROM:" + from + "\r\n" +
                                                "SUBJECT:" + subject + "\r\n" +
                                                "BODY:" + body + "\r\n" +
                                                attachmentData;
                                sequenceNumber++;


                                // Send the email message to the server
                                byte[] sendData = smtpMessage.getBytes();
                                DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length,
                                                serverAddress, serverPort);
                                clientSocket.send(sendPacket);
```

```
                                System.out.println("Mail Sent to Server, waiting...");
                                // Receive confirmation from server
                                byte[] receiveData = new byte[1024];
                                DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);
                                clientSocket.receive(receivePacket);
                                // Process confirmation message from server
                                String confirmationMessage = new
String(receivePacket.getData(), 0,
                                                receivePacket.getLength());
                                if (confirmationMessage.contains("250 OK")) {
                                        System.out.println("\nEmail received successfully at " +
getCurrentTimeStamp());
                                        System.out.println("Email Sequence Number: " +
sequenceNumber);
                                        // Save the email to file
                                        saveEmailToFile(smtpMessage);
                                } else if (confirmationMessage.startsWith("501")) {
                                        // Handle error response from server
                                        handleErrorResponse(confirmationMessage);
                                }
                                // Prompt user for next action
                                System.out.flush();
                                System.out.println("\nSend 'Quit' to exit \nPress Enter to send
another email");
                                String decision = scanner.nextLine();
                                if ("quit".equalsIgnoreCase(decision)) {
                                        System.out.println("\n Connection Terminated! ");
                                        // Send "Quit" message to server
                                        String quitMessage = "Quit";
                                        byte[] emailData = quitMessage.getBytes();
                                        DatagramPacket quitPacket = new
DatagramPacket(emailData, emailData.length,
                                                        serverAddress, serverPort);
                                        clientSocket.send(quitPacket);
                                        break;
                                }
                        }
                } catch (UnknownHostException e) {
                        System.err.println("Host could not be determined" + e.getMessage());
                } catch (IOException e) {
                        e.printStackTrace();
                } finally {
                        // Close socket and scanner
```

```
                    if (clientSocket != null && !clientSocket.isClosed()) {
                        clientSocket.close();
                    }
                    scanner.close();
                }
        }
        // Method to handle error response from the server
        private static void handleErrorResponse(String errorResponse) {
                System.out.println("Server Error: " + errorResponse);
        }
        // Method to perform handshake with the server
        private static void performHandshake(DatagramSocket clientSocket, InetAddress
serverAddress,
                    int serverPort)
                            throws IOException {
                // Send SYN packet to server
                byte[] handshakeData = "SYN".getBytes();
                DatagramPacket handshakePacket = new DatagramPacket(handshakeData,
                            handshakeData.length, serverAddress,
                            serverPort);
                clientSocket.send(handshakePacket);
                // Receive SYN-ACK response from server
                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket = new DatagramPacket(receiveData,
                            receiveData.length);
                clientSocket.receive(receivePacket);
                // Process server's response
                String response = new String(receivePacket.getData(), 0,
                            receivePacket.getLength());
                if (response.equals("SYN-ACK")) {
                    // Send ACK to complete handshake
                    byte[] ackData = "ACK".getBytes();
                    DatagramPacket ackPacket = new DatagramPacket(ackData,
ackData.length,
                                serverAddress, serverPort);
                    clientSocket.send(ackPacket);
                    System.out.println("\nHandshake completed. Connection
established.");
                } else {
                    System.out.println("Connection failed.");
                }
        }
        // Method to get current timestamp
        private static String getCurrentTimeStamp() {
```

```
                SimpleDateFormat sdf = new SimpleDateFormat("EEE, MMM d, yyyy
HH:mm:ss");
                return sdf.format(new Date());
        }
        // Method to encode attachment file as Base64
        private static String encodeAttachment(String filePath) throws IOException {
                byte[] attachmentBytes = readFileToBytes(filePath);
                String encodedAttachment =
java.util.Base64.getEncoder().encodeToString(attachmentBytes);
                return "ATTACHMENT:" + encodedAttachment + "\r\n";
        }
        // Method to read file into bytes
        private static byte[] readFileToBytes(String filePath) throws IOException {
                File file = new File(filePath);
                byte[]fileBytes = new byte[(int) file.length()];
                FileInputStream fis = new FileInputStream(file);
                fis.read(fileBytes);
                fis.close();
                return fileBytes;
        }
        // Method to save email content to a file
        private static void saveEmailToFile(String emailContent) throws IOException {
                // Create the directory if it doesn't exist
                File directory = new File(SENDER_DIRECTORY);
                if (!directory.exists()) {
                        directory.mkdir();
                }
                // Extract the subject from the email content
                String subjectLine = extractSubject(emailContent);
                // Replace characters in subject that are invalid in file names
                String safeSubject = subjectLine.replaceAll("[^a-zA-Z0-9\\-]", "");
                // Shorten the subject if it is too long to be a file name
                if (safeSubject.length() > 50) {
                        safeSubject = safeSubject.substring(0, 50) + "...";
                }
                // Format the timestamp
                String timestamp = new
SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
                // Create the file name with subject and timestamp
                String fileName = SENDER_DIRECTORY + File.separator + safeSubject + "_" +
timestamp
                                + ".txt";
                // Write the email content to the file
                try (FileWriter fw = new FileWriter(fileName);
```

```
                        BufferedWriter bw = new BufferedWriter(fw);
                        PrintWriter out = new PrintWriter(bw)) {
                    out.println(emailContent);
                } catch (IOException e) {
                    System.err.println("An error occurred while trying to save the email to
the file: " +
                            e.getMessage());
                }
                System.out.println("Email saved to file: " + fileName);
        }
        // Method to extract subject from email content
        private static String extractSubject(String emailContent) {
                String[] lines = emailContent.split("\r\n");
                for (String line : lines) {
                    if (line.toUpperCase().startsWith("SUBJECT:")) {
                        return line.substring(8).trim(); // Extract the subject text after
"SUBJECT:"
                    }
                }
                return "NoSubject"; // Default if no subject is found
        }
        // Set of valid email recipients
        private static final Set<String> validRecipients = new HashSet<>();
        static {
                validRecipients.add("mawdhi@gmail.com");
                validRecipients.add("hind@gmail.com");
                validRecipients.add("shaikha@gmail.com");
                validRecipients.add("shaikha2@gmail.com");
        }
        // Method to check if recipient is valid
        private static boolean isValidRecipient(String recipient) {
                return validRecipients.contains(recipient.toLowerCase());
        }
}
```

```
import java.io.*;
import java.net.*;
import java.text.*;
import java.util.*;

class ClientReceiver implements Runnable {
	// Specify the port on which the receiver will listen for incoming messages
	private static final int LISTENING_PORT = 2626;
	// Define a valid directory path for saving received emails
	private static final String RECEIVER_DIRECTORY = "receiver_directory"; // Make sure
this directory exists or can be created
	@Override
	public void run() {
		try (DatagramSocket clientSocket = new DatagramSocket(LISTENING_PORT))
{
			InetAddress serverAddress = InetAddress.getByName("Mohamed-
Macbook-Air.local"); // Server IP
			int serverPort = 2626; // Server listening port, used for initial
handshake
			System.out.println("Mail Receiver Client Starting...");
			// Send SYN to initiate the handshake
			byte[] synBytes = "SYN".getBytes();
			DatagramPacket synPacket = new DatagramPacket(synBytes,
synBytes.length,
					serverAddress, serverPort);
			clientSocket.send(synPacket);
			// Wait for SYN-ACK from the server
			byte[] buffer = new byte[1024];
			DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
			clientSocket.receive(packet);
			String response = new String(packet.getData(), 0, packet.getLength());
			if ("SYN-ACK".equals(response)) {
				// Send ACK to complete the handshake
				byte[] ackBytes = "ACK".getBytes();
				DatagramPacket ackPacket = new DatagramPacket(ackBytes,
ackBytes.length,
						serverAddress, serverPort);
				clientSocket.send(ackPacket);
				System.out.println("Handshake completed. Connection
established. Waiting for emails...");
				// After handshake, continuously listen for incoming emails
				while (true) {
```

```
                                        DatagramPacket receivePacket = new
DatagramPacket(buffer, buffer.length);
                                        clientSocket.receive(receivePacket);
                                        String emailData = new String(receivePacket.getData(),
0, receivePacket.getLength());
                                        System.out.println("Email received:\n" + emailData);
                                        // Call saveEmailToFile to save the received email
                                        saveEmailToFile(emailData);
                                }
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
        // Method to save received email content to a file
        private static void saveEmailToFile(String emailContent) throws IOException {
                // Create the directory if it doesn't exist
                File directory = new File(RECEIVER_DIRECTORY);
                if (!directory.exists()) {
                        directory.mkdir();
                }
                // Extract the subject from the email content
                String subjectLine = extractSubject(emailContent);
                // Replace characters in subject that are invalid in file names
                String safeSubject = subjectLine.replaceAll("[^a-zA-Z0-9\\-]", "");
                // Shorten the subject if it is too long to be a file name
                if (safeSubject.length() > 50) {
                        safeSubject = safeSubject.substring(0, 50) + "...";
                }
                // Format the timestamp
                String timestamp = new
SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
                // Create the file name with subject and timestamp
                String fileName = RECEIVER_DIRECTORY + File.separator + safeSubject + "_"
+
                                        timestamp + ".txt";
                // Write the email content to the file
                try (FileWriter fw = new FileWriter(fileName);
                                        BufferedWriter bw = new BufferedWriter(fw);
                                        PrintWriter out = new PrintWriter(bw)) {
                        out.println(emailContent);
                } catch (IOException e) {
                        System.err.println("An error occurred while trying to save the email to
the file: " +
```

```
                                        e.getMessage());
                }
                System.out.println("Email saved to file: " + fileName);
                // Now, let's handle the attachment part
                String[] parts = emailContent.split("ATTACHMENT:");
                if (parts.length > 1) {
                        // Assuming there's only one attachment for simplicity
                        String attachmentEncoded = parts[1].trim(); // Get the Base64
encoded attachment data
                        saveAttachmentToFile(attachmentEncoded);
                }
        }
        // Method to extract subject from email content
        private static String extractSubject(String emailContent) {
                String[] lines = emailContent.split("\r\n");
                for (String line : lines) {
                        if (line.toUpperCase().startsWith("SUBJECT:")) {
                                return line.substring(8).trim(); // Extract the subject text after
"SUBJECT:"
                        }
                }
                return "NoSubject"; // Default if no subject is found
        }
        // Method to save attachment data to file
        private static void saveAttachmentToFile(String attachmentEncoded) {
                try {
                        // Decode attachment data from Base64
                        byte[] attachmentData =
Base64.getDecoder().decode(attachmentEncoded);
                        // Create directory to store attachments if it doesn't exist
                        String attachmentsDirPath = RECEIVER_DIRECTORY + File.separator +
                                        "received_attachments";
                        File attachmentsDir = new File(attachmentsDirPath);
                        if (!attachmentsDir.exists() && !attachmentsDir.mkdirs()) {
                                System.err.println("Failed to create attachment directory.");
                                return;
                        }
                        // Generate a unique filename for the attachment
                        String attachmentFileName = "Attachment_" + new

        SimpleDateFormat("yyyyMMdd_HHmmssSSS").format(new Date()) + ".bin";
                        File attachmentFile = new File(attachmentsDir,
attachmentFileName);
                        // Write attachment data to file
```

```
                        try (FileOutputStream fos = new FileOutputStream(attachmentFile)) {
                                fos.write(attachmentData);
                                System.out.println("Attachment saved to file: " +
attachmentFile.getAbsolutePath());
                        } catch (IOException e) {
                                System.err.println("Error writing the attachment file: " +
e.getMessage());
                        }
                } catch (IllegalArgumentException e) {
                        System.err.println("Error decoding attachment: " + e.getMessage());
                } catch (Exception e) {
                        System.err.println("Unexpected error in saveAttachmentToFile: " +
e.getMessage());
                }
        }
}
```

```
import java.io.*;
import java.net.*;
import java.text.*;
import java.util.*;
public class Server {
	// ------------- Directory to save attachments
	private static final String ATTACHMENT_DIRECTORY = "attachment";
	private static final String MAPPING_FILE_PATH = "mapping.txt"; // Path to mapping
file
	private static final String MappingError = "550 Error";
	private static final String HeaderError = "501 Invalid Headers";
	private static final HashMap<String, String> emailToHostnameMapping = new
HashMap<>();
	public static void main(String[] args) throws UnknownHostException {
		System.out.println("Mail Server Starting..." +
InetAddress.getLocalHost().getHostName());
		DatagramSocket serverSocket = null;
		loadMappingFile();
		// Create separate threads for sender and receiver
		new Thread(new SenderHandler()).start();
		new Thread(new ReceiverHandler()).start();
	}
	// Sender handler thread
	static class SenderHandler implements Runnable {
		private static final int SENDER_PORT = 25; // Separate port for sender
		@Override
		public void run() {
			try {
				DatagramSocket serverSocket = new
DatagramSocket(SENDER_PORT);
				while (true) {
					// Perform 3-way handshake with sender
					performHandshake(serverSocket);
					// Now handle sender communication
					DatagramPacket packet = receivePacket(serverSocket);
					byte[] data = packet.getData();
					String smtpMessage = new String(data, 0,
packet.getLength());
					// Assuming processSMTPMessage method now
correctly processes the message
					processSMTPMessage(smtpMessage, serverSocket,
packet.getAddress(), packet.getPort());
					// Check if the smtpMessage contains "Quit" and break
if so
```

```
                                        if (smtpMessage.contains("Quit")) {
                                                System.out.println("\nClient disconnected.
Server is running to make other connections.");
                                                break; // Exit the loop
                                        }
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
        }
        private static void handleSenderCommunication(DatagramPacket packet,
DatagramSocket serverSocket) {
                byte[] data = packet.getData();
                String smtpMessage = new String(data, 0, packet.getLength());
                // Process sender communication logic
                try {
                        processSMTPMessage(smtpMessage, serverSocket,
packet.getAddress(), packet.getPort());
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
        // Receiver handler thread
        static class ReceiverHandler implements Runnable {
                private static final int RECEIVER_PORT = 2626;
                @Override
                public void run() {
                        try (DatagramSocket serverSocket = new
DatagramSocket(RECEIVER_PORT)) {
                                while (true) {
                                        // Perform 3-way handshake with receiver
                                        performHandshake(serverSocket);
                                        // Now handle receiver communication
                                        DatagramPacket packet = receivePacket(serverSocket);
                                        handleReceiverCommunication(packet, serverSocket);
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
                private static void handleReceiverCommunication(DatagramPacket packet,
DatagramSocket
                                serverSocket) {
```

```
                        byte[] data = packet.getData();
                        String smtpMessage = new String(data, 0, packet.getLength());
                        // Process receiver communication logic
                        try {
                                processSMTPMessage(smtpMessage, serverSocket,
packet.getAddress(), packet.getPort());
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
        }
        private static void processSMTPMessage(String smtpMessage, DatagramSocket
serverSocket,
                        InetAddress clientAddress, int clientPort) throws IOException {
                // Ignore control messages
                if (smtpMessage.equals("ACK") || smtpMessage.equals("SYN") ||
smtpMessage.equals("SYN-ACK")) {
                        return; // Skip processing for control messages
                }
                // Validate the email format
                if (!isValidEmailFormat(smtpMessage)) {
                        sendResponse("501 Invalid Headers", clientAddress, clientPort,
serverSocket);
                        return; // Stop processing this email
                }
                String[] lines = smtpMessage.split("\r\n");
                String from = "", to = "", subject = "", body = "";
                StringBuilder bodyBuilder = new StringBuilder();
                String attachmentData = null;
                boolean attachmentPart = false; // Flag to indicate attachment data part
                for (String line : lines) {
                        if (line.toUpperCase().startsWith("FROM:")) {
                                from = line.substring(5).trim();
                        } else if (line.toUpperCase().startsWith("TO:")) {
                                to = line.substring(3).trim();
                        } else if (line.toUpperCase().startsWith("SUBJECT:")) {
                                subject = line.substring(8).trim();
                        } else if (line.toUpperCase().startsWith("BODY:")) {
                                bodyBuilder.append(line.substring(5)).append("\r\n");
                        } else if (line.toUpperCase().startsWith("ATTACHMENT:")) {
                                attachmentPart = true; // Start of attachment data
                                attachmentData =
line.substring("ATTACHMENT:".length()).trim(); // Assuming single line attachment for
simplicity
```

```
                } else if (attachmentPart) {
                        // Assuming the attachment data could span multiple lines
                        attachmentData += line.trim(); // Append additional
attachment data
                }
            }
            body = bodyBuilder.toString();
            if (!validateHeader(from, to, subject)) {
                sendResponse("501 Invalid Headers", clientAddress, clientPort,
serverSocket);
                return;
            }
            saveEmailToFile(from, to, subject, body);
            // Save attachment if present
            if (attachmentData != null) {
                saveAttachmentToFile(from, subject, attachmentData);
            }
            // Log received email
            logReceivedEmail(from, to, subject, body, clientAddress);
            forwardEmailToReceiver(from, to, subject, body, attachmentData,
serverSocket);
            // Further processing like forwarding the email
            sendResponse("250 OK - Email processed successfully", clientAddress,
clientPort, serverSocket);
        }
        private static void logReceivedEmail(String from, String to, String subject, String
body,
                InetAddress clientAddress) {
            System.out.println();
            System.out.println("Mail Received from [" + clientAddress.getHostName() +
"]");
            System.out.println("FROM: [" + from + "]");
            System.out.println("TO: [" + to + "]");
            System.out.println("SUBJECT: " + subject);
            System.out.println("TIME: " + getCurrentTimeStamp());
            System.out.println(body);
        }
        private static boolean isValidEmailFormat(String smtpMessage) {
            // Check for presence of mandatory headers
            if (!smtpMessage.contains("TO:") || !smtpMessage.contains("FROM:") ||
                    !smtpMessage.contains("SUBJECT:") ||
!smtpMessage.contains("BODY:")) {
                return false;
            }
```

```
                // Split message and verify parts
                String[] parts = smtpMessage.split("\r\n");
                // Check format of sequence number
                if (!parts[0].toUpperCase().startsWith("SEQUENCE:")) {
                        return false;
                }
                try {
                        Integer.parseInt(parts[0].substring(9).trim());
                } catch (NumberFormatException e) {
                        return false;
                }
                // Validate headers and body
                boolean hasEmptyHeader = false;
                for (int i = 1; i < parts.length - 1; i++) { // Skip first (sequence number) and last
(attachment) parts
                        if (parts[i].toUpperCase().startsWith("TO:") &&
parts[i].substring(3).trim().isEmpty() ||
                                        parts[i].toUpperCase().startsWith("FROM:") &&
parts[i].substring(5).trim().isEmpty() ||
                                        parts[i].toUpperCase().startsWith("SUBJECT:") &&
parts[i].substring(8).trim().isEmpty() ||
                                        parts[i].toUpperCase().startsWith("BODY:") &&
parts[i].substring(5).trim().isEmpty()) {
                                hasEmptyHeader = true;
                                break;
                        }
                }
                return !hasEmptyHeader;
        }
        private static final Set<String> validEmails = new HashSet<>();
        static {
                validEmails.add("mawdhi@gmail.com");
                validEmails.add("hind@gmail.com");
                validEmails.add("shaikha@gmail.com");
                validEmails.add("shaikha2@gmail.com");
        }
        private static boolean validateHeader(String from, String to, String subject) {
                return validEmails.contains(from.toLowerCase()) &&
validEmails.contains(to.toLowerCase())
                                && !subject.trim().isEmpty();
        }
        private static void sendResponse(String response, InetAddress clientAddress, int
clientPort,
                        DatagramSocket serverSocket) throws IOException {
```

```
                String replyMessage = response + "\n" + getCurrentTimeStamp();
                byte[] sendData = replyMessage.getBytes();
                DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, clientAddress,
                                clientPort);
                serverSocket.send(sendPacket);
        }
        private static String getCurrentTimeStamp() {
                SimpleDateFormat sdf = new SimpleDateFormat("EEE, MMM d, yyyy
HH:mm:ss");
                return sdf.format(new Date());
        }
        private static void saveAttachmentToFile(String from, String subject, String
attachmentData)
                        throws IOException {
                File directory = new File(ATTACHMENT_DIRECTORY);
                if (!directory.exists()) {
                        directory.mkdir();
                }
                // Create a file name for the attachment
                String sanitizedSubject = subject.replaceAll("[^a-zA-Z0-9.]", "_");
                String timestamp = new
SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
                String fileName = sanitizedSubject + "_" + timestamp;
                // Assuming attachment data is Base64 encoded
                byte[] decodedData = Base64.getDecoder().decode(attachmentData);
                File attachmentFile = new File(directory, fileName);
                try (FileOutputStream fos = new FileOutputStream(attachmentFile)) {
                        fos.write(decodedData);
                } catch (IOException e) {
                        System.err.println("An error occurred while trying to save the
attachment to the file: " +
                                        e.getMessage());
                        throw e; // Re-throw or handle as needed
                }
                System.out.println("Attachment saved to file: " + attachmentFile.getPath());
        }
        private static DatagramPacket receivePacket(DatagramSocket socket) throws
IOException {
                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
                socket.receive(receivePacket);
                return receivePacket;
```

```
	}
	private static void sendPacket(DatagramSocket socket, InetAddress address, int
port, String
			message)
					throws IOException {
		byte[] sendData = message.getBytes();
		DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, address, port);
		socket.send(sendPacket);
	}
	private static void loadMappingFile() {
		try (BufferedReader br = new BufferedReader(new
FileReader(MAPPING_FILE_PATH))) {
			String line;
			while ((line = br.readLine()) != null) {
				String[] parts = line.split(",");
				if (parts.length == 2) {
					emailToHostnameMapping.put(parts[0].toLowerCase(),
parts[1].toLowerCase());
				}
			}
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
	private static void saveEmailToFile(String from, String to, String subject, String body)
throws
	IOException {
		// Sanitize the from address to use as a directory name
		String senderDirectory = from.replaceAll("[^a-zA-Z0-9.]", "_");
		// Ensure there's a base directory for emails
		File baseDir = new File("emails");
		if (!baseDir.exists()) baseDir.mkdir();
		// Create a directory for the sender if it doesn't exist
		File clientDir = new File(baseDir, senderDirectory);
		if (!clientDir.exists()) clientDir.mkdir();
		// Format the timestamp for the file name
		SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd_HHmmss");
		String timestamp = sdf.format(new Date());
		// Sanitize and truncate subject to be safe for file name
		String safeSubject = subject.length() > 20 ? subject.substring(0,
20).replaceAll("[^a-zA-Z0-9 ]",
				"_") : subject.replaceAll("[^a-zA-Z0-9 ]", "_");
		// Construct the file name and create the file
```

```
                String fileName = safeSubject + "_" + timestamp + ".txt";
                File emailFile = new File(clientDir, fileName);
                // Save the email content to the file
                try (PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(emailFile)))) {
                        out.println("FROM: " + from);
                        out.println("TO: " + to);
                        out.println("SUBJECT: " + subject);
                        out.println("TIME: " + getCurrentTimeStamp());
                        out.println(body);
                } catch (IOException e) {
                        System.err.println("An error occurred while trying to save the email: " +
e.getMessage());
                }
                System.out.println("Email saved to file: " + emailFile.getPath());
        }
        private static boolean isHandshakeComplete = false;
        private static void performHandshake(DatagramSocket socket) throws IOException
{
                byte[] receiveData = new byte[1024];
                while (!isHandshakeComplete) {
                        DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
                        socket.receive(receivePacket);
                        String received = new String(receivePacket.getData(), 0,
receivePacket.getLength()).trim();
                        if (received.equals("SYN")) {
                                InetAddress clientAddress = receivePacket.getAddress();
                                int clientPort = receivePacket.getPort();
                                sendPacket(socket, clientAddress, clientPort, "SYN-ACK");
                                // Don't set isHandshakeComplete here; wait for "ACK"
                        } else if (received.equals("ACK")) {
                                isHandshakeComplete = true; // Handshake complete, ready
to process emails
                                System.out.println("Handshake complete. Ready to receive
emails.");
                        } else if (received.startsWith("SEQUENCE:")) {
                                // If a SEQUENCE message is received, assume handshake
was missed and proceed
                                isHandshakeComplete = true;
                                processSMTPMessage(received, socket,
receivePacket.getAddress(), receivePacket.getPort());
                        } else {
```

```
                                System.out.println("Received unexpected message during
handshake: " + received);
                        }
                }
        }
        private static void forwardEmailToReceiver(String from, String to, String subject,
String body,
                        String attachmentData, DatagramSocket serverSocket) {
                try {
                        String recipientInfo =
emailToHostnameMapping.get(to.toLowerCase());
                        if (recipientInfo == null) {
                                System.err.println("Recipient address not found in mapping.");
                                return;
                        }
                        String[] recipientParts = recipientInfo.split(":");
                        if (recipientParts.length != 2) {
                                System.err.println("Invalid recipient address format.");
                                return;
                        }
                        InetAddress recipientAddress =
InetAddress.getByName(recipientParts[0]);
                        int recipientPort = Integer.parseInt(recipientParts[1]);

                        // Include the attachment data in the forwarded email if it's present
                        String fullEmail = "FROM:" + from + "\r\nTO:" + to + "\r\nSUBJECT:" +
subject + "\r\nBODY:" +
                                        body;
                        if (attachmentData != null && !attachmentData.isEmpty()) {
                                fullEmail += "\r\nATTACHMENT:" + attachmentData; // Append
the attachment data
                        }
                        byte[] emailData = fullEmail.getBytes();
                        DatagramPacket emailPacket = new DatagramPacket(emailData,
emailData.length,
                                        recipientAddress, recipientPort);
                        serverSocket.send(emailPacket);
                        System.out.println("Email sent to receiver: " + to);
                } catch (Exception e) {
                        System.err.println("Failed to forward email: " + e.getMessage());
                }
        }
}
```