**Ex No: 8**

**Date:**

**DEVELOPMENT OF SUITABLE PROTOCOL TO SEND, RECEIVE AND SYNCHRONIZE MAIL.**

**Aim:**

To write a Java program to implement the client server model to send, receive and synchronize mail.

**Theory:**

SMTP and POP3 are both protocols used to transmit email data between devices, but they have different purposes and functions:

* SMTP (Simple Mail Transfer Protocol)

Used to send emails from the sender's device to the recipient's email server. SMTP is also used by servers to forward messages to their final destination. SMTP is a push protocol.

* POP3 (Post Office Protocol 3)

Used to retrieve emails from the recipient's email server to the recipient's device. POP3 is also known as a pop protocol. POP3 is an inbound protocol.

Here are some other differences between SMTP and POP3:

* Encryption

POP3 has standard and encrypted ports, with 110 being the standard and 995 being the SSL-encrypted alternative. SMTP has ports 25 and 587, with 25 being the standard for outgoing emails and 587 being registered as a secure SMTP port.

* Benefits of POP3

POP3 allows for offline access to emails, is easy to set up, and is bandwidth-friendly.

* Benefits of SMTP

SMTP ensures simplicity, speed, and reliability with notifications for delivery failures.

**Algorithm:**

# TO SEND MAIL: SMTP PROTOCOL

# Initialize Constants:

# Set the SMTP server (SMTP\_SERVER) to "smtp.gmail.com".

# Set the SMTP port (SMTP\_PORT) to 465.

# Set the email USER and PASSWORD for authentication.

# Create SSL Socket:

# Get an instance of SSLSocketFactory.

# Use the SSLSocketFactory to create an SSLSocket connected to the SMTP server at the specified port.

# Create Input and Output Streams:

# Create a BufferedReader for reading responses from the server.

# Create a BufferedWriter for sending commands to the server.

# Read Server Response:

# Read and print the server's initial response after connecting.

# Send SMTP Commands:

# Send the HELO command to the SMTP server and read the response.

# Send the AUTH LOGIN command to initiate authentication and read the response.

# Authenticate User:

# Encode the USER (email address) in Base64 and send it to the server.

# Read and print the server's response.

# Encode the PASSWORD in Base64 and send it to the server.

# Read and print the server's response.

# Specify Sender and Recipient:

# Send the MAIL FROM command with the user's email address and read the response.

# Send the RCPT TO command with the recipient's email address and read the response.

# Send Email Data:

# Send the DATA command to indicate the start of the email content and read the response.

# Send the email headers (Subject, From, To) followed by the email body content.

# End the email content with a single period . on a new line, then read the server's response.

# Terminate the Session:

# Send the QUIT command to terminate the session and read the server's response.

# Close Resources:

# Close the BufferedWriter, BufferedReader, and SSLSocket to release resources.

# Handle Exceptions:

# Use a try-catch block to handle exceptions that may occur during the execution of the code, printing the stack trace if an exception is caught.

# TO RECEIVE MAIL: POP3 PROTOCOL

# Initialize Constants:

# Set the POP3 server (POP3\_SERVER) to "pop.gmail.com".

# Set the POP3 port (POP3\_PORT) to 995.

# Set the email USER (email address) and PASSWORD for authentication.

# Create SSL Socket:

# Get an instance of SSLSocketFactory.

# Use the SSLSocketFactory to create an SSLSocket connected to the POP3 server at the specified port.

# Create Input and Output Streams:

# Create a BufferedReader for reading responses from the server.

# Create a BufferedWriter for sending commands to the server.

# Read Server Greeting:

# Read and print the server's greeting message after connecting.

# Authenticate User:

# Send the USER command with the email address and read the server's response.

# Send the PASS command with the password and read the server's response.

# Check if the authentication was successful by verifying if the response starts with -ERR.

# If authentication fails, print an error message and exit the program.

# Check Number of Messages:

# Send the STAT command to get the total number of messages in the mailbox and read the response.

# Extract the number of total messages from the response.

# Determine Messages to Fetch:

# Calculate the number of messages to fetch (last 5 messages) using Math.min(5, totalMessages).

# Fetch Latest Emails:

# Loop from the total number of messages down to the number of messages to fetch:

# Send the RETR command for each message to retrieve its content.

# Read and print the server's response (which contains the email content).

# Read the email content until a line with just . is encountered, indicating the end of the email.

# Store the email content in a StringBuilder.

# Parse Email Headers:

# Split the email content into lines and extract relevant headers:

# Initialize variables for subject, from, to, and date.

# Loop through the headers, extracting the required information based on header prefixes.

# Display Extracted Fields:

# Print the email index along with the extracted subject, from, to, and date fields.

# Terminate the Session:

# Send the QUIT command to end the POP3 session and read the server's response.

# Close Resources:

# Close the BufferedWriter, BufferedReader, and SSLSocket to release resources.

# Handle Exceptions:

# Use a try-catch block to handle exceptions that may occur during the execution of the code, printing the stack trace if an exception is caught.

**NOTE: Use only gmail email id. Generate 16 key App Password in your google account, which must be entered in the program, as password.**

**Program:**

# TO SEND MAIL: SMTP PROTOCOL

# import javax.net.ssl.SSLSocket; import javax.net.ssl.SSLSocketFactory; import java.io.\*; import java.util.Base64; public class SENDMAIL { private static final String *SMTP\_SERVER* = "smtp.gmail.com"; private static final int *SMTP\_PORT* = 465; private static final String *USER* = "*yourmail@gmail.com*"; private static final String *PASSWORD* = "*your 16key app password*"; // Use App Password if using 2FA public static void main(String[] args) { try { // Create an SSL socket to connect to the SMTP server SSLSocketFactory ssf = (SSLSocketFactory) SSLSocketFactory.*getDefault*(); SSLSocket socket = (SSLSocket) ssf.createSocket(*SMTP\_SERVER*, *SMTP\_PORT*); BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream())); BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())); System.*out*.println("Response: " + reader.readLine()); writer.write("HELO " + *SMTP\_SERVER* + "\r\n"); writer.flush(); System.*out*.println("Response: " + reader.readLine()); writer.write("AUTH LOGIN\r\n"); writer.flush(); System.*out*.println("Response: " + reader.readLine()); // Send username (Base64 encoded) writer.write(Base64.*getEncoder*().encodeToString(*USER*.getBytes()) + "\r\n"); writer.flush(); System.*out*.println("Response: " + reader.readLine()); // Send password (Base64 encoded) writer.write(Base64.*getEncoder*().encodeToString(*PASSWORD*.getBytes()) + "\r\n"); writer.flush(); System.*out*.println("Response: " + reader.readLine()); writer.write("MAIL FROM:<" + *USER* + ">\r\n"); writer.flush(); System.*out*.println("Response: " + reader.readLine()); writer.write("RCPT TO:<ngianna28@gmail.com>\r\n"); writer.flush(); System.*out*.println("Response: " + reader.readLine()); writer.write("DATA\r\n"); writer.flush(); System.*out*.println("Response: " + reader.readLine()); writer.write("Subject: Test Email\r\n"); writer.write("From: " + *USER* + "\r\n"); writer.write("To: *recipientmail@gmail.com*\r\n"); writer.write("\r\n"); writer.write("HI THIS IS A TEST MAIL\r\n"); writer.write(".\r\n"); writer.flush(); System.*out*.println("Response: " + reader.readLine()); writer.write("QUIT\r\n"); writer.flush(); System.*out*.println("Response: " + reader.readLine()); writer.close(); reader.close(); socket.close(); } catch (Exception e) { e.printStackTrace(); } } }

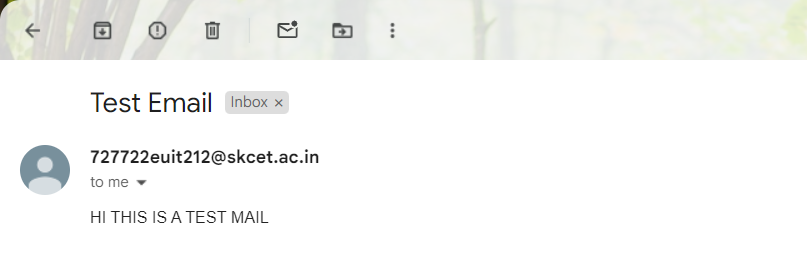
# TO RECEIVE MAIL: POP3 PROTOCOL

import javax.net.ssl.SSLSocket;  
import javax.net.ssl.SSLSocketFactory;  
import java.io.\*;  
  
public class RECEIVEMAIL {  
 private static final String *POP3\_SERVER* = "pop.gmail.com"; // POP3 server for Gmail  
 private static final int *POP3\_PORT* = 995; // POP3 port with SSL  
 private static final String *USER* = "*yourmail@gmail.com*";  
 private static final String *PASSWORD* = "*your 16key app password*"; // Use App Password if using 2FA  
  
  
 public static void main(String[] args) {  
 try {  
 // Create an SSL socket to connect to the POP3 server  
 SSLSocketFactory ssf = (SSLSocketFactory) SSLSocketFactory.*getDefault*();  
 SSLSocket socket = (SSLSocket) ssf.createSocket(*POP3\_SERVER*, *POP3\_PORT*);  
 BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
  
 // Read server greeting  
 System.*out*.println("Response: " + reader.readLine());  
  
 // Send USER command  
 writer.write("USER " + *USER* + "\r\n");  
 writer.flush();  
 System.*out*.println("Response: " + reader.readLine());  
  
 // Send PASS command  
 writer.write("PASS " + *PASSWORD* + "\r\n");  
 writer.flush();  
 String response = reader.readLine();  
 System.*out*.println("Response: " + response);  
  
 // Check if authentication was successful  
 if (response.startsWith("-ERR")) {  
 System.*out*.println("Authentication failed. Please check your username and password.");  
 return; // Exit the program  
 }  
  
 // Send STAT command to get the number of messages  
 writer.write("STAT\r\n");  
 writer.flush();  
 response = reader.readLine();  
 System.*out*.println("Response: " + response);  
  
 // Extract the number of messages  
 int totalMessages = Integer.*parseInt*(response.split(" ")[1]);  
 System.*out*.println("Total messages: " + totalMessages);  
  
 // Determine how many messages to fetch (last 5)  
 int messagesToFetch = Math.*min*(5, totalMessages);  
  
 // Fetch the latest emails  
 for (int i = totalMessages; i > totalMessages - messagesToFetch; i--) {  
 // Send RETR command to fetch the email  
 writer.write("RETR " + i + "\r\n");  
 writer.flush();  
 response = reader.readLine();  
 System.*out*.println("Response: " + response); // This shows the email content  
  
 // Read the email content  
 StringBuilder emailContent = new StringBuilder();  
 String line;  
 while (!(line = reader.readLine()).equals(".")) {  
 emailContent.append(line).append("\n");  
 }  
  
 // Parse the email content for headers  
 String[] headers = emailContent.toString().split("\n");  
 String subject = "", from = "", to = "", date = "";  
  
 // Extract headers  
 for (String header : headers) {  
 if (header.startsWith("Subject:")) {  
 subject = header.substring(9).trim();  
 } else if (header.startsWith("From:")) {  
 from = header.substring(6).trim();  
 } else if (header.startsWith("To:")) {  
 to = header.substring(4).trim();  
 } else if (header.startsWith("Date:")) {  
 date = header.substring(6).trim();  
 }  
 }  
  
 // Display extracted fields  
 System.*out*.println("Email " + i + ":");  
 System.*out*.println("Subject: " + subject);  
 System.*out*.println("From: " + from);  
 System.*out*.println("To: " + to);  
 System.*out*.println("Date: " + date);  
 System.*out*.println("\n--- End of Email " + i + " ---\n");  
 }  
  
 // Send QUIT command  
 writer.write("QUIT\r\n");  
 writer.flush();  
 System.*out*.println("Response: " + reader.readLine());  
  
 writer.close();  
 reader.close();  
 socket.close();  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

# SCREENSHOTS OF OUTPUT:

# SENDING MAIL

# 



# RECEIVING MAIL

# 

**Result:**Thus the client server model for mail transfer application using TCP/IP protocol, was successfully implemented in Java coding and output obtained.