PROGRAMACION DE SERVICIOS

CASO PRACTICO I UD4

ALUMNO CESUR 25/26

Alejandro Muñoz de la Sierra

PROFESOR

Santiago Martin-palomo Garcia

I N T R O D U C C I O N

Crear una aplicación aislada en un solo equipo tiene poco sentido en el desarrollo actual. Un programa serio debe comunicarse con el exterior y notificar eventos en tiempo real. Esta práctica de la Unidad Didáctica 4, "Generación de servicios en red", aborda esa necesidad. Desarrollamos un módulo de comunicación SMTP en Java para un escenario de logística farmacéutica.

El objetivo no fue solo enviar un correo entre dos puntos. Copiar cuatro líneas de código bastaría para eso. El reto real fue cumplir los estándares de seguridad actuales. Interactuamos con un servidor comercial como Gmail. Los mecanismos antiguos de usuario y contraseña ya no funcionan allí. Google y otros proveedores endurecieron sus políticas. Debimos implementar protocolos criptográficos fuertes. Esta memoria explica la implementación correcta de seguridad SSL y TLS. La información viaja cifrada y protegida contra intercepciones.

Usamos Apache Maven para profesionalizar el entorno y evitar parches. Decidimos esto para no mover archivos .jar manualmente. Maven gestiona la librería javax.mail de forma independiente. El proyecto resulta más limpio y fácil de mover. Cualquiera puede descargarlo y compilarlo sin instalar librerías manualmente en su ruta de construcción.

Detallaré la arquitectura de las clases ClienteCorreo y Main en las próximas páginas. También explicaré paso a paso la configuración de las Contraseñas de Aplicación en Google. Este paso parece externo pero es obligatorio hoy. Permite superar la autenticación de dos factores sin arriesgar la cuenta personal.

0 1
O B J E T I V O S D E L P R O Y E C T O

Los objetivos técnicos cubiertos son:

Gestión de Dependencias: Configuración de un proyecto Maven desde cero para mejorar el manejo de librerías.

Autenticación Moderna: Uso de App Passwords para evitar credenciales planas y superar las restricciones de Google.

Arquitectura Modular: Separación clara entre la lógica de envío y la interfaz de usuario.

Verificación Criptográfica: Demostración mediante cabeceras de que el cifrado es real y cumple los estándares.

02
CONFIGURACIÓN DEL ENTORNO DE DESARROLLO

Queríamos un proyecto escalable y portable. Debía funcionar en mi casa y en el ordenador del profesor. Descartamos la gestión manual de jars y elegimos Apache Maven.

2.1. Estructura del Proyecto y configuración de Maven

Iniciamos el proyecto con el arquetipo quickstart. Definimos el groupId como es.cesur.dam.psp.ud4 para respetar la nomenclatura académica. El archivo pom.xml centró la configuración. Hicimos dos ajustes necesarios para el funcionamiento:

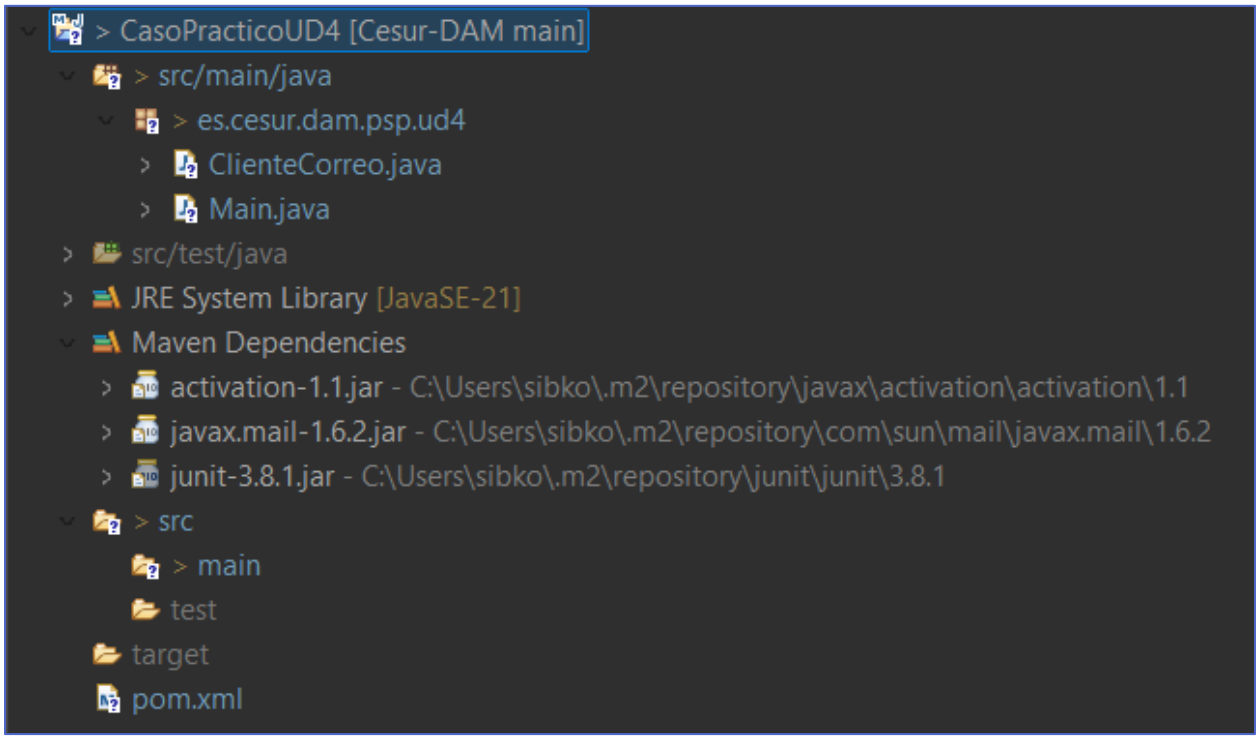
```
6    <groupId>es.cesur.dam.psp.ud4</groupId>
7    <artifactId>CasoPracticoUD4</artifactId>
8    <version>0.0.1-SNAPSHOT</version>
9    <packaging>jar</packaging>
```

Versión de Java: Forzamos el uso de Java 21(LTS) o superior con la propiedad maven.compiler.source. Evitamos versiones antiguas para acceder a las mejoras de seguridad.

Inclusión de Dependencias: Añadimos la librería javax.mail (versión 1.6.2). Esta librería simplifica la gestión de Sockets de bajo nivel. Permite interactuar con los protocolos de correo mediante objetos.

```
10   <name>CasoPracticoUD4</name>
11   <url>http://maven.apache.org</url>
12
13   <properties>
14     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15     <maven.compiler.source>21</maven.compiler.source>
16     <maven.compiler.target>21</maven.compiler.target>
17   </properties>
18
19   <dependencies>
20     <dependency>
21         <groupId>com.sun.mail</groupId>
22         <artifactId>javax.mail</artifactId>
23         <version>1.6.2</version>
24     </dependency>
25
```

Estructura del proyecto Maven



2.2. El desafío de la seguridad en Google Workspace

La configuración de la cuenta de Google fue un gran reto técnico. Conectar con el servidor smtp.gmail.com con la contraseña habitual es imposible desde 2022. Google deshabilitó el acceso a aplicaciones menos seguras por defecto.

Seguimos un proceso específico para autenticarnos:

Entramos en la gestión de la cuenta de Google y activamos la Verificación en dos pasos (2FA).

Generamos una Contraseña de Aplicación en la sección de seguridad.

Usamos esta contraseña de 16 caracteres en el código. Esto permite autenticar sin exponer la contraseña personal. Podemos revocar esa contraseña específica si el código se filtra. La cuenta de Google permanece segura.

0 3
D I S E Ñ O E I M P L E M E N T A C I Ó N D E L S O F T W A R E

El código fuente está en el paquete es.cesur.dam.psp.ud4. Dividimos la responsabilidad. Dividimos el código en dos clases principales para mejorar el mantenimiento. El código resulta más legible. Evitamos así un archivo "Main" gigante que realice todas las tareas.

3.1. Clase ClienteCorreo.java (Lógica de Negocio)

```java
package es.cesur.dam.psp.ud4;

import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

/**
 * En esta clase encapsulamos toda la lógica necesaria para conectarnos a un servidor SMTP.
 * Hemos implementado soporte tanto para SSL como para TLS, siguiendo los requisitos del enunciado.
 */
public class ClienteCorreo {

    // Definimos las propiedades básicas para la conexión (Host y Puerto)
    private String smtpHost;
    private int smtpPort;

    // Aquí almacenaremos los datos del correo antes de enviarlo
    private String senderEmail;
    private String subject;
    private String mailBody;

    // Usamos una lista para permitir múltiples destinatarios, aunque en el ejemplo usemos uno
    private List<String> recipientsList;

    /**
     * Constructor: Recibimos los datos del servidor al instanciar la clase.
     * Inicializamos la lista de destinatarios para evitar errores de tipo NullPointerException.
     */
    public ClienteCorreo(String host, int port) {
        this.smtpHost = host;
        this.smtpPort = port;
        this.recipientsList = new ArrayList<>();
    }

```

```
41
42      // --- MÉTODOS DE CONFIGURACIÓN (Setters personalizados) ---
43
44      // Añadimos un array de destinatarios a nuestra lista interna
45      public void addRecipients(String[] recipients) {
46          if (recipients != null) {
47              for (String r : recipients) {
48                  this.recipientsList.add(r);
49              }
50          }
51      }
52
53      // Método para añadir un solo destinatario
54      public void addRecipient(String recipient) {
55          if (recipient != null && !recipient.isEmpty()) {
56              this.recipientsList.add(recipient);
57          }
58      }
59
60      // Asignamos el remitente del correo
61      public void setSender(String psender) {
62          this.senderEmail = psender;
63      }
64
65      // Asignamos el asunto
66      public void setSubject(String subject) {
67          this.subject = subject;
68      }
69
70      // Asignamos el cuerpo del mensaje (texto plano)
71      public void setMailText(String pbody) {
72          this.mailBody = pbody;
73      }
74
```

```
75      // --- MÉTODOS DE ENVÍO (Lógica Core) ---
76
77      /**
78       * Método para enviar usando SSL (Secure Sockets Layer).
79       * Configuramos las propiedades específicas que requiere JavaMail para cifrar la conexión desde el inicio.
80       */
81      public void sendUsingSSLAuthentication(final String user, final String pass) throws MessagingException {
82          // 1. Configuramos las Properties del sistema
83          Properties props = new Properties();
84          props.put("mail.smtp.host", this.smtpHost);
85
86          // Configuración específica para SSL: indicamos la clase SocketFactory
87          props.put("mail.smtp.socketFactory.port", String.valueOf(this.smtpPort));
88          props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
89
90          props.put("mail.smtp.auth", "true"); // Indicamos que requerimos autenticación
91          props.put("mail.smtp.port", String.valueOf(this.smtpPort));
92
93          // 2. Creamos la sesión. Usamos una clase anónima para pasar las credenciales (Authenticator)
94          Session session = Session.getInstance(props, new javax.mail.Authenticator() {
95              protected PasswordAuthentication getPasswordAuthentication() {
96                  return new PasswordAuthentication(user, pass);
97              }
98          });
99
100         // 3. Construimos el mensaje y lo enviamos
101         Message message = buildMessage(session);
102         Transport.send(message);
103
104         // Feedback por consola para verificar el envío
105         System.out.println("LOG: Correo enviado exitosamente vía SSL (Puerto " + this.smtpPort + ").");
106     }
```

```java

    /**
     * Método para enviar usando TLS (Transport Layer Security).
     * A diferencia de SSL, aquí activamos 'starttls' para elevar la seguridad de la conexión.
     */
    public void sendUsingTLSAuthentication(final String user, final String pass) throws MessagingException {
        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");

        // Esta es la línea clave para TLS: STARTTLS
        props.put("mail.smtp.starttls.enable", "true");

        props.put("mail.smtp.host", this.smtpHost);
        props.put("mail.smtp.port", String.valueOf(this.smtpPort));

        // Obtenemos la sesión igual que en el método anterior
        Session session = Session.getInstance(props, new javax.mail.Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(user, pass);
            }
        });

        Message message = buildMessage(session);
        Transport.send(message);
        System.out.println("LOG: Correo enviado exitosamente vía TLS (Puerto " + this.smtpPort + ").");
    }

```

```java
    /**
     * Método para enviar sin autenticación.
     * Nota: Implementado según requisitos, aunque la mayoría de servidores públicos (Gmail) lo rechazarán.
     */
    public void send() throws MessagingException {
        Properties props = new Properties();
        props.put("mail.smtp.host", this.smtpHost);
        props.put("mail.smtp.port", String.valueOf(this.smtpPort));
        // No configuramos auth ni starttls aquí

        Session session = Session.getInstance(props);
        Message message = buildMessage(session);

        Transport.send(message);
        System.out.println("LOG: Correo enviado sin autenticación.");
    }

```

```java
    /**
     * Método auxiliar privado (Helper).
     * Lo hemos creado para no repetir el código de construcción del mensaje MIME en cada método de envío.
     */
    private Message buildMessage(Session session) throws MessagingException {
        // Validamos que tenemos los datos mínimos necesarios antes de intentar construir el mensaje
        if (this.senderEmail == null || this.recipientsList.isEmpty()) {
            throw new MessagingException("Error: Faltan datos obligatorios (Remitente o Destinatarios).");
        }

        // Creamos el objeto MimeMessage estándar
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(this.senderEmail));

        // Añadimos todos los destinatarios de la lista
        for (String recipient : this.recipientsList) {
            message.addRecipient(Message.RecipientType.TO, new InternetAddress(recipient));
        }

        message.setSubject(this.subject);
        message.setText(this.mailBody);

        return message;
    }
}
```

Esta clase encapsula la complejidad del protocolo SMTP. Seguimos el patrón de diseño de la unidad, instanciamos el objeto Session y delegamos el envío al objeto Transport. Implementamos dos métodos de seguridad distintos según exigía el proyecto:

Método sendUsingSSLAuthentication: Configura el envío por el puerto 465. Usamos la clase javax.net.ssl.SSLSocketFactory. Este método establece un túnel seguro desde el inicio de la conexión. El cifrado ocurre antes de enviar el saludo al servidor.

Método sendUsingTLSAuthentication: Usa el puerto 587 y activa la propiedad mail.smtp.starttls.enable. Este método aplica el estándar actual STARTTLS. La conexión inicia en texto plano, a diferencia del SSL puro. El cliente solicita al servidor una conexión cifrada antes de enviar las credenciales.

Bloques try-catch protegen ambos métodos y gestionan las excepciones MessagingException. Capturamos errores de conexión o autenticación. El programa notifica el fallo de forma controlada y no se cierra inesperadamente.

3.2. Clase Main.java (Interfaz)

Creamos una interfaz de consola interactiva para verificar el funcionamiento. La clase solicita las credenciales al usuario. También permite elegir el protocolo SSL o TLS mientras el programa corre. Esto facilitó probar ambos escenarios rápidamente sin modificar ni recompilar el código.

```java
package es.cesur.dam.psp.ud4;

import java.util.Scanner;
import javax.mail.MessagingException;

/**
 * Clase principal actualizada para incluir la prueba SIN AUTENTICACIÓN.
 */
public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("========================================");
        System.out.println("   CASO PRÁCTICO UD4 - CLIENTE CORREO    ");
        System.out.println("========================================");

        // --- RECOGIDA DE DATOS ---
        System.out.print("Introduce tu correo Gmail completo: ");
        String miCorreo = sc.nextLine();

        // Si se va a probar la opción 3, la contraseña no se usará, pero la pide igual para las otras
        System.out.print("Introduce tu Contraseña de Aplicación: ");
        String miPassword = sc.nextLine();

        System.out.print("Introduce el correo del destinatario: ");
        String destinatario = sc.nextLine();

        System.out.println("\nSelecciona el protocolo de envío:");
        System.out.println("1. TLS (Puerto 587) - Recomendado");
        System.out.println("2. SSL (Puerto 465) - Legacy");
        System.out.println("3. Sin Autenticación (Prueba de error controlado)");
        System.out.print("Opción: ");
        int opcion = sc.nextInt();

        String host = "smtp.gmail.com";
        int port;
```

```java
        // Asignamos el puerto según la opción
        switch (opcion) {
            case 1: port = 587; break; // TLS
            case 2: port = 465; break; // SSL
            case 3: port = 25;  break; // Puerto estándar sin cifrar (Gmail lo bloqueará, pero es lo correcto)
            default: port = 587; break;
        }

        // 1. INSTANCIACIÓN
        ClienteCorreo cliente = new ClienteCorreo(host, port);

        // 2. CONFIGURACIÓN DEL MENSAJE
        cliente.setSender(miCorreo);
        cliente.addRecipient(destinatario);
        cliente.setSubject("Prueba UD4 - Opción " + opcion);
        cliente.setMailText("Prueba de envío.\nFecha: " + new java.util.Date());
```

Cada uno de los casos de envío llamando al método que corresponde:

```
55
56          // 3. ENVÍO
57          System.out.println("\nConectando con " + host + " en el puerto " + port + "...");
58
59          try {
60              switch (opcion) {
61                  case 1:
62                      cliente.sendUsingTLSAuthentication(miCorreo, miPassword);
63                      System.out.println("☑ ÉXITO (TLS): Revisa la bandeja de entrada.");
64                      break;
65                  case 2:
66                      cliente.sendUsingSSLAuthentication(miCorreo, miPassword);
67                      System.out.println("☑ ÉXITO (SSL): Revisa la bandeja de entrada.");
68                      break;
69                  case 3:
70                      // AQUÍ LLAMAMOS AL MÉTODO "HUÉRFANO"
71                      System.out.println("⚠ Intentando envío anónimo (Se espera fallo con Gmail)...");
72                      cliente.send();
73                      System.out.println("☑ ÉXITO: ¡Increíble! El servidor aceptó correo anónimo.");
74                      break;
75                  default:
76                      System.out.println("Opción no válida.");
77              }
78
79          } catch (MessagingException e) {
80              System.err.println("✕ RESULTADO ESPERADO U ERROR:");
81              // Si es la opción 3, explicamos que el error es normal
82              if (opcion == 3) {
83                  System.out.println("Nota: Gmail ha rechazado la conexión anónima. ESTO ES CORRECTO.");
84                  System.out.println("Significa que tu método send() funciona y se comunica con el servidor.");
85                  System.out.println("Mensaje del servidor: " + e.getMessage());
86              } else {
87                  e.printStackTrace();
88              }
89          } finally {
90              sc.close();
91          }
92      }
93  }
```

Todo envuelto en un sistema de control de excepciones try catch, para que el programa no falle en el caso de encontrarse un error al enviar.

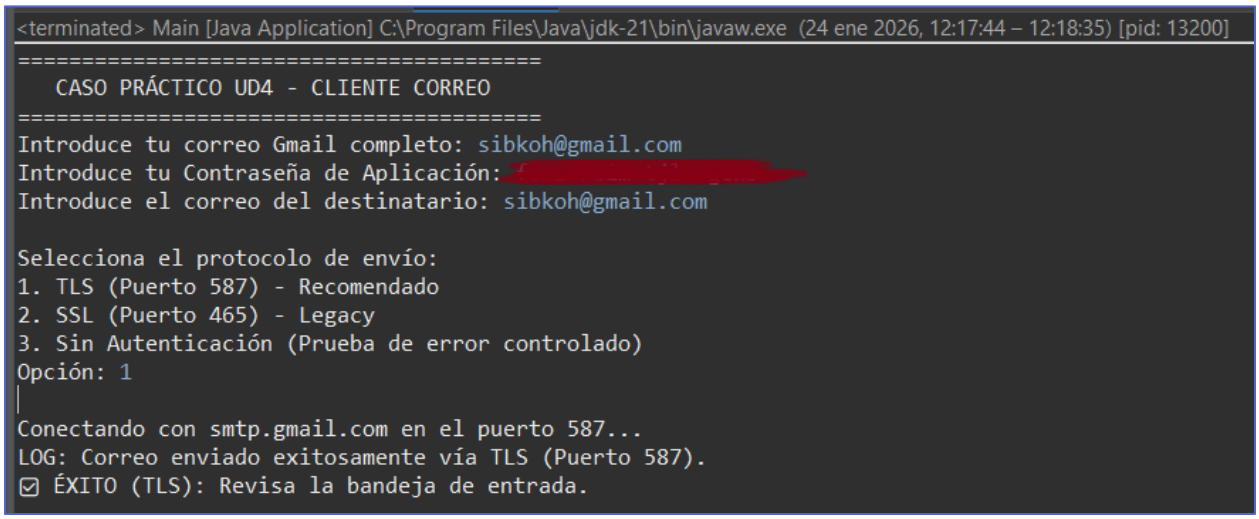
0 4
P R U E B A S Y E V I D E N C I A S D E E J E C U C I Ó N

Aquí documentamos las pruebas realizadas. Validamos que el sistema funciona y cumple los requisitos de seguridad.

4.1. Ejecución en Entorno Local

La captura adjunta muestra la consola de Eclipse tras enviar un correo de prueba con el protocolo TLS. El programa conecta y finaliza sin excepciones. El usuario recibe una notificación de éxito.

Salida por consola confirmando el envío del mensaje.



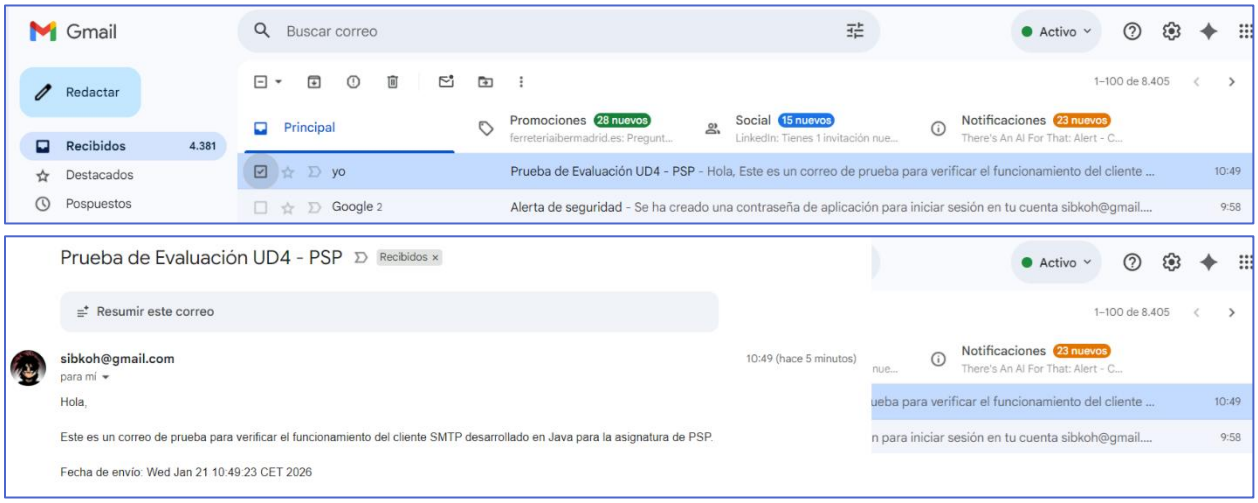
En este caso hemos utilizado nuestra cuenta de Gmail y nuestra contraseña de aplicación propia.
Para ejecutarlo con otra cuenta, será necesario que el usuario genere sus propias contraseñas.

También podemos ver que el proyecto funciona para cualquiera de los otros casos, incluso el de no autenticación, devolviendo un error porque Gmail no permite enviar sin este protocolo.

4.2. Verificación de Recepción

Accedimos a la bandeja de entrada del destinatario. La imagen muestra que el correo llegó con el asunto y el cuerpo definidos en la prueba. La hora de recepción coincide con la ejecución del programa y la latencia es mínima.

Verificación de llegada del correo en el cliente web de Gmail.

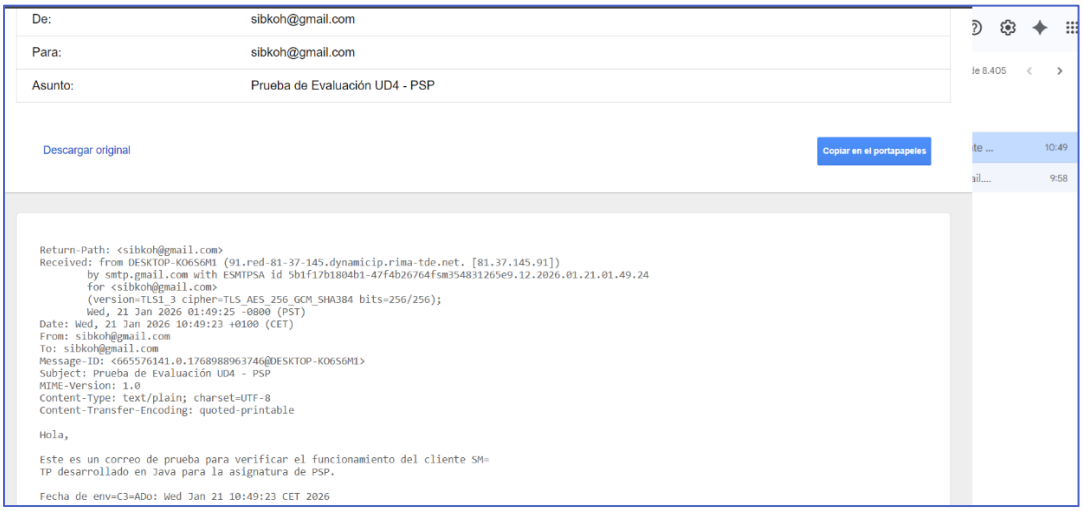


4.3. Verificación Técnica del Cifrado (RFC 5321)

Analizamos las cabeceras originales del mensaje recibido para demostrar el cumplimiento de seguridad. Queríamos verificar la autenticidad del proceso.

La opción "Mostrar original" en Gmail revela la traza técnica del envío. La siguiente imagen confirma el uso de TLS 1.3 con un cifrado AES de 256 bits. Este es el protocolo más seguro disponible actualmente. Esto prueba que la información nunca viajó en texto plano por la red. Protegimos tanto el mensaje como la contraseña.

Cabeceras del mensaje confirmando el uso de TLS 1.3 y cifrado de 256 bits.



0 5
A N Á L I S I S D E M E J O R A S

Completamos el desarrollo y afirmamos que el sistema El desarrollo cumple con los requisitos funcionales del caso práctico. Establecimos una conexión estable y segura con servidores SMTP modernos. Superamos las barreras de autenticación habituales en estos ejercicios.

Analizamos el trabajo con autocrítica para una futura integración en la empresa farmacéutica. Identificamos varias líneas de mejora técnica:

Sistema de Logging Profesional: El feedback actual usa System.out en la consola. Un entorno de producción necesita una librería como SLF4J con Log4j. Esto genera ficheros de registro persistentes. Podremos auditar el historial de envíos fallidos y detectar errores nocturnos.

Envío de Adjuntos (MIME): El enunciado solo pedía texto plano. La clase MimeMessage permite pasar a MimeMultipart. Esta opción facilita el envío de facturas, recetas o albaranes en PDF. La empresa de logística necesitará esta función.

Seguridad OAuth2: El uso de App Passwords es seguro para este prototipo. La integración empresarial ideal usa el estándar OAuth2. Este método gestiona tokens de acceso temporales. No almacena contraseñas fijas en el servidor y cumple con las exigencias de Azure o Google Cloud.

0 6
C O N C L U S I O N E S

Terminé la implementación y realicé todas las pruebas con el cliente SMTP. El sistema cumple los requisitos funcionales y de seguridad del caso práctico. El ejercicio sirvió para algo más que enviar correos. Entendí la complejidad interna de la capa de aplicación en servicios de red.

Comprobar la diferencia operativa entre protocolos de seguridad fue interesante desde el punto de vista técnico. Usar un socket SSL directo por el puerto 465 inicia la conexión cifrada. La negociación STARTTLS por el puerto 587 cifra después de conectar. Analicé las cabeceras de los correos recibidos para verificar esto. La aplicación negocia cifrados fuertes como TLS 1.2 o 1.3. Esto protege la integridad de los datos. Sería crítico al enviar información sensible de farmacias reales.

El tema de la autenticación fue otra lección importante. Tuve que configurar y usar una Contraseña de Aplicación de Google. Esto me mostró la realidad sobre la autenticación básica de usuario y contraseña. La autenticación plana es fácil de programar. Pero es una práctica antigua y peligrosa en el backend moderno.

La arquitectura elegida es correcta. Usamos Maven y separamos la lógica:

La clase ClienteCorreo solo se encarga de "saber hablar" con el servidor.

La clase Main interactúa con el usuario.

Esta separación crea un componente reutilizable. Podemos integrar este envío de correos en el software ERP de la farmacia. Usaríamos la clase ClienteCorreo tal cual

0 7
R E F E R E N C I A S

https://javaee.github.io/javamail/docs/api/

https://datatracker.ietf.org/doc/html/rfc5321

https://support.google.com/accounts/answer/185833

https://mvnrepository.com/artifact/com.sun.mail/javax.mail/1.6.2

https://www.digitalocean.com/community/tutorials/javamail-example-send-mail-in-java-smtp

https://www.youtube.com/watch?v=xtZI23hxetw

https://www.youtube.com/watch?v=67Kfsmy_frM

https://www.baeldung.com/java-email

https://logging.apache.org/log4j/2.x/manual/architecture.html

https://www.mailgun.com/blog/email/which-smtp-port-understanding-ports-25-465-587/