Desarrollador: Agustín Alexis Reyes Castillo

23130568

BITÁCORA DE APLICACIÓN

**VIERNES 7 DE FEBRERO**

Comencé el programa.

Al principio quise hacerlo con JavaFX y FXML, ya que tengo conocimientos en el uso de HTML y CSS, pero no pude compilar el archivo, entonces pasé a java swing.

Comencé con el frame de Login y el de la aplicación "appCorreo" para hacer el interfaz visual primero.

private void botonIngresarActionPerformed(java.awt.event.ActionEvent evt) {

from = remitente.getText();

pw = new String(clave.getPassword());

if(FuncionesLogin.validarCorreo(from)) {

if(FuncionesLogin.validarUsuario(from, pw)){

this.dispose();

appCorreo corr = new appCorreo();

corr.setVisible(true);

}

}

}

primero puse el código del botón de ingresar, aunque aún no tenía las funciones para validar tanto el correo, como que el correo y contraseña pertenecieran a uno real.

Luego me puse a trabajar en las conexiones

public static void conectar(String remitente, String clave, String destinatario, String subject, String content) {

//Correo sugerido por chatGPT

Properties props = new Properties();

props.put("mail.smtp.host", "smtp.gmail.com");

props.put("mail.smtp.port", "465");

props.put("mail.smtp.auth", "true");

props.put("mail.smtp.starttls.enable", "true");

props.put("mail.smtp.starttls.required", "true");

props.put("mail.smtp.ssl.protocols", "TLSv1.2");

props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");

Session sesion = Session.getDefaultInstance(props);

MimeMessage msg = new MimeMessage(sesion);

try {

//for(String destin : destinatario){

msg.setFrom(new InternetAddress(remitente));

msg.setRecipient(Message.RecipientType.TO, new InternetAddress(destinatario));

msg.setSubject(subject);

msg.setContent(tieneArchivo(content));

//}

} catch (AddressException ex) {

ex.printStackTrace();

} catch (MessagingException ex) {

ex.printStackTrace();

} catch (IOException ex) {

ex.printStackTrace();

}

try {

Transport mTransport = sesion.getTransport("smtp");

mTransport.connect(remitente, clave);

mTransport.sendMessage(msg, msg.getRecipients(Message.RecipientType.TO));

mTransport.close();

JOptionPane.showMessageDialog(null, "Correo enviado con éxito!");

} catch (MessagingException ex) {

ex.printStackTrace();

}

}

Hice la clase ConexionYValidación, para el envío del correo electrónico, también puse como comentario un for para luego poder cambiar el destinatario por un array de destinatarios, y poder enviarlo a muchos correos al mismo tiempo.

public static Multipart tieneArchivo(String content) throws MessagingException, IOException {

ArrayList<File> arch = appCorreo.getArchivos();

MimeBodyPart msj = new MimeBodyPart();

msj.setText(content);

Multipart multi = new MimeMultipart();

multi.addBodyPart(msj);

if(!arch.isEmpty()) {

for(File file: arch) {

MimeBodyPart agregarArchivos = new MimeBodyPart();

agregarArchivos.attachFile(file);

multi.addBodyPart(agregarArchivos);

}

}

return multi;

}

ya que los archivos no siempre tienen correo, hice un apartado para crear todo el mensaje, donde en lugar de mandar sólo el texto, hice un array de archivos que serían agregados desde el frame, y si es que contiene algún archivo, los juntaba con el contenido del correo, para luego mandarlo al método principal de envío.

**SÁBADO 8 DE FEBRERO**

Hoy junté el backend con el frontend, le di más estilo y diseñé la appCorreo. También le di las funcionalidades a los botones.

Hice funcionar mi conexión en la interfaz, y también hice la validación del usuario y el correo electrónico.

public class FuncionesLogin {

public static boolean validarUsuario(String remitente, String clave) {

//Generar una conexión rápida

//Función obtenida del método conectar.

Properties props = new Properties();

props.put("mail.smtp.host", "smtp.gmail.com");

props.put("mail.smtp.port", "465");

props.put("mail.smtp.auth", "true");

props.put("mail.smtp.starttls.enable", "true");

props.put("mail.smtp.starttls.required", "true");

props.put("mail.smtp.ssl.protocols", "TLSv1.2");

props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");

Session sesion = Session.getDefaultInstance(props);

try {

Transport mTransport = sesion.getTransport("smtp");

mTransport.connect(remitente, clave);

mTransport.close();

} catch (MessagingException ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Algo salió mal! Compruebe correo o contraseña");

return false;

}

return true;

}

public static boolean validarCorreo(String correo) {

if(!correo.contains("@") || !correo.contains(".")){

JOptionPane.showMessageDialog(null, "Formato de correo incorrecto");

return false;

}

return true;

}

}

me puse a trabajar en los métodos que usará el frame de login. Hice un método para validar el correo, y verificar que cumpla con las condiciones de un correo electrónico. También hice la validación de la cuenta con una pequeña conexión y desconexión; si no me marcaba una excepción, entonces el correo y la contraseña si eran correctas.

private static String from;

private static String pw;

las variables from y pw son el correo y contraseña del usuario, que tienen que ser almacenadas como globales para luego, al enviar el mensaje, se puedan obtener.

Me puse a trabajar en la appCorreo, para permitir su funcionamiento y agregarle diferentes cosas. Esta parte fue la más trabajada, debido a que, ya que la conexión estaba implementada, había que implementar toda la parte de elementos visuales y funcionamiento, además de los métodos de validación.

después de la interfaz visual, hice el método de enviar

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

String[] destinatarios = destinatario.getText().split(", ");

boolean flag = true;

for(String des: destinatarios) {

if(!FuncionesLogin.validarCorreo(des)){

flag = false;

}

}

if(flag){

for(String des: destinatarios) {

ConexionYValidacion.conectar(LOGIN.getRemitente(), LOGIN.getClave(), des, asunto.getText(), mensaje.getText());

borrarTodo();

}

}

}

implementada para poder enviar a distintos usuarios el mismo correo, y validando cada correo con la función que utilicé para validar el correo del login.

Una vez enviado el correo, se usa la función borrar todo, aunque aún no estaba definida.

Los archivos que se envían se mandan con un método.

private static ArrayList<File> archivos = new ArrayList<>();

JLabel[] bas;

int archivosPermitidos;

int megasPermitidos;

creé el arrayList para almacenar los archivos, y las variables de archivos y megas permitidos por archivo, para poder cambiarlos más rápido, también, para hacer una implementación de los botones de borrar, hice un array de labels con dibujo de botes de basura (solo de los archivos), para que se muestre cuando haya archivos, y con el fin de borrar al clickear en estos.

**DOMINGO 9 DE FEBRERO**

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

Frame frame = new Frame();

FileDialog fileDialog = new FileDialog(frame, "Abrir Archivos", FileDialog.LOAD);

fileDialog.setMultipleMode(true); // Permitir selección múltiple

fileDialog.setVisible(true);

File[] files = fileDialog.getFiles(); // Obtener archivos seleccionados

boolean permitido = false;

if(validarExtensiones(files)){

if(validarPeso(files)){

if(validarNumeroDeArchivos(files)){

permitido = true;

}

}

}

if(permitido){

archivos.addAll(Arrays.asList(files));

}

actualizarListView();

}

private boolean validarExtensiones(File[] files) {

String[] extensionesPermitidas = {

"png", "jpg", "jpeg", "gif", "bmp", "tiff", "svg", "pptx",

"zip", "rar", "tar", "7z", "gz",

"mp4", "mkv", "avi", "mov", "wmv", "flv",

"pdf", "doc", "docx", "txt", "rtf", "odt", "xls", "xlsx"

};

for(File file : files) {

String nombre = file.getName();

String extension = nombre.substring(nombre.lastIndexOf('.') + 1);

boolean esValida = false;

for (String ext : extensionesPermitidas) {

if (ext.equalsIgnoreCase(extension)) {

esValida = true;

break;

}

}

if (!esValida) {

JOptionPane.showMessageDialog(null, "La siguiente extensión de archivo no es válida: " + extension);

return false; // Si algún archivo no tiene una extensión válida, retorna false.

}

}

return true; // Todos los archivos tienen extensiones válidas.

}

private boolean validarPeso(File[] files) {

for(File file : files) {

if(file.length() > megasPermitidos \* 1024 \* 1024){ //20mb

JOptionPane.showMessageDialog(null, "Lo siento, no se permiten archivos mayores a 20Mb");

return false;

}

}

return true;

}

private boolean validarNumeroDeArchivos(File[] files) {

if (files.length + listaArchivos.getModel().getSize() > archivosPermitidos) {

JOptionPane.showMessageDialog(null, "Lo siento, no es posible enviar más de 3 archivos por correo");

return false;

}

return true;

}

private void actualizarListView() {

String[] nombresArchivos = archivos.stream().map(File::getName).toArray(String[]::new);

listaArchivos.setListData(nombresArchivos);

for(int i = 0; i < archivosPermitidos; i++) {

if(i < archivos.size()) {

bas[i].setVisible(true);

} else {

bas[i].setVisible(false);

}

}

}

Todos los métodos anteriores son los que cree para enviar archivos. El primero es el que se ejecuta cuando clickeas en el botón de enviar, donde abre una ventana para seleccionar los archivos, pero también puse restricciones en cuanto al tamaño del archivo y la cantidad de archivos que se pueden enviar.

Al final, tuve que actualizar la lista donde se veían los archivos, y también muestro u oculto los botes de basura para borrar cada archivo, según sea el caso. Además agrego los archivos al array que será enviado a la clase de ConexionYValidación.

public static ArrayList<File> getArchivos() {

return archivos;

}

public void borrarTodo() {

asunto.setText("");

mensaje.setText("");

archivos.clear();

actualizarListView();

}

Hice los métodos para obtener los archivos, que usa la clase ConexiónYValidación para enviarlos, además de la clase de borrar todo, que borra cada parte del frame menos el correo, ya que ese se borra de manera individual

También le di funcionalidad a las labels de los botes de basura haciendo que eliminaran el campo correspondiente, y cree el botón de borrar, que sólo llama al método borrarTodo().

**LÚNES 10 DE FEBRERO**

Hoy me puse a trabajar en un nuevo frame para gestionar los múltiples correos a los que se va a enviar el mensaje. Tiene una jTable donde se escribirán los correos, y además se podrán eliminar directamente de ahí.

private void labelClickMouseClicked(java.awt.event.MouseEvent evt) {

CorreosAEnviar corr = new CorreosAEnviar();

corr.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

corr.setVisible(true);

}

Inserté un botón para mostrar el frame.

public CorreosAEnviar() {

initComponents();

tablaCorreos.getColumnModel().getColumn(0).setMaxWidth(100);

destinatarios = new ArrayList<>();

destinatarios.addAll(Arrays.asList(appCorreo.getDestinatarios()));

spinerCorreos.setValue(destinatarios.length);

DefaultTableModel modelo = (DefaultTableModel) tablaCorreos.getModel();

modelo.setRowCount(destinatarios.length);

for(int i = 0; i < destinatarios.length; i++) {

modelo.setValueAt(i+1, i, 0);

modelo.setValueAt(destinatarios[i], i, 1);

}

}  
  
En los componentes iniciales, puse los correos que el usuario ya podría haber puesto en el formulario, además de un spinner para poder determinar la cantidad de correos a los que desea enviar el mensaje.

private void spinerCorreosStateChanged(javax.swing.event.ChangeEvent evt) {

DefaultTableModel modelo = (DefaultTableModel) tablaCorreos.getModel();

int numCorreos = Integer.parseInt(spinerCorreos.getValue().toString());

modelo.setRowCount(numCorreos);

for(int i = 0; i < numCorreos; i++) {

modelo.setValueAt(i+1, i, 0);

}

}

También hice el código para cuando cambia el número de correos que el usuario va a manejar, modificando la tabla para tener el número de renglones correspondientes.

private void tablaCorreosKeyPressed(java.awt.event.KeyEvent evt) {

DefaultTableModel modelo = (DefaultTableModel) tablaCorreos.getModel();

if(evt.getKeyCode() == KeyEvent.VK\_ENTER){

System.out.println("entró");

destinatarios.clear();

for(int i = 0; i < modelo.getRowCount(); i++) {

if(modelo.getValueAt(i, 1) != null && !modelo.getValueAt(i, 1).toString().equals("")) {

destinatarios.add(modelo.getValueAt(i, 1).toString());

}

}

}

}

private void botonEliminarCorreoActionPerformed(java.awt.event.ActionEvent evt) {

try {

int numAEliminar = Integer.parseInt(CorreoAEliminar.getText())-1;

destinatarios.remove(numAEliminar);

} catch (Exception ex) {

JOptionPane.showMessageDialog(null, "Por favor, introduce un número de correo");

}

actualizarTabla();

}

public void actualizarTabla() {

DefaultTableModel modelo = (DefaultTableModel) tablaCorreos.getModel();

int numCorreos = Integer.parseInt(spinerCorreos.getValue().toString());

for(int i = 0; i < numCorreos; i++) {

modelo.setValueAt(i+1, i, 0);

if(i < destinatarios.size()) {

modelo.setValueAt(destinatarios.get(i), i, 1);

} else {

modelo.setValueAt("", i, 1);

}

}

}

Cree el método para borrar un correo en caso de que ya no quieras mandar el mensaje a ese usuario. También para actualizar la tabla cuando se borran los correos, y para modificar el arrayList.

Tuve que modificar el array de destinatarios, y ponerlo como arraylist, ya que no sabemos la cantidad de destinatarios.

private void botonListoCorreosActionPerformed(java.awt.event.ActionEvent evt) {

appCorreo.setDestinatarios(destinatarios);

this.dispose();

}

private void actualizarCorreosActionPerformed(java.awt.event.ActionEvent evt) {

String destin = "";

for(int i = 0; i < destinatarios.size(); i++) {

if(i < destinatarios.size()-1) {

destin += destinatarios.get(i) + ", ";

} else {

destin+= destinatarios.get(i);

}

}

destinatario.setText(destin);

}

Codifiqué el botón de listo, que cierra la pestaña para seleccionar correos, y agregué un botón para actualizar los destinatarios, ya que no me dejaba referenciar la label debido a que no era estática.

También corregí un error de duplicado de correo al reabrir el frame de correos.

private void listaArchivosValueChanged(javax.swing.event.ListSelectionEvent evt) {

int numeroArchivo = listaArchivos.getSelectedIndex();

if (numeroArchivo == -1) return;

DecimalFormat formato = new DecimalFormat("#.###");

tamañoDelArchivo.setText(formato.format(archivos.get(numeroArchivo).length() / 1024.0) + " Kb");

String nombreArchivo = archivos.get(numeroArchivo).getName();

if(!"pdf".equals(nombreArchivo.substring(nombreArchivo.lastIndexOf('.') + 1))){

limpiarPanelVisualizacion();

panelVisualizacion.setLayout(new BorderLayout());

panelVisualizacion.add(new JLabel("Sin Visualizacion Previa", SwingConstants.CENTER), BorderLayout.CENTER);

return;}

String selectedFile = archivos.get(numeroArchivo).getAbsolutePath();

if (selectedFile != null) {

renderPDF(selectedFile);

}

}

public void limpiarPanelVisualizacion() {

panelVisualizacion.removeAll();

panelVisualizacion.setPreferredSize(new Dimension(panelVisualizacion.getWidth(), panelVisualizacion.getHeight()));

panelVisualizacion.revalidate();

panelVisualizacion.repaint();

}

public void renderPDF(String filePath) {

try {

File file = new File(filePath);

if (!file.exists() || file.length() == 0) {

JOptionPane.showMessageDialog(null, "El archivo PDF no existe o está vacío.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

//Código sugerido por chatGPT

PDDocument document = PDDocument.load(file);

PDFRenderer pdfRenderer = new PDFRenderer(document);

BufferedImage image = pdfRenderer.renderImageWithDPI(0, 300);

document.close();

int panelWidth = panelVisualizacion.getWidth();

int panelHeight = panelVisualizacion.getHeight();

BufferedImage scaledImage = smoothScale(image, panelWidth, panelHeight);

imageLabel.setIcon(new ImageIcon(scaledImage));

// Asegurar que el panel se muestra en el contenedor principal

panelVisualizacion.removeAll();

panelVisualizacion.setLayout(new BorderLayout());

panelVisualizacion.add(imageLabel, BorderLayout.CENTER);

panelVisualizacion.revalidate();

panelVisualizacion.repaint();

} catch (FileNotFoundException ex) {

Logger.getLogger(appCorreo.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

Logger.getLogger(appCorreo.class.getName()).log(Level.SEVERE, null, ex);

}

}

//Método sugerido por ChatGPT para reescalar la imagen, adaptado para mi aplicativo

private BufferedImage smoothScale(BufferedImage img, int targetWidth, int targetHeight) {

int currentWidth = img.getWidth();

int currentHeight = img.getHeight();

BufferedImage scaledImage = img;

// Reducir la imagen en pasos del 50% hasta llegar al tamaño deseado

while (currentWidth > targetWidth \* 2 || currentHeight > targetHeight \* 2) {

currentWidth /= 2;

currentHeight /= 2;

scaledImage = resizeImage(scaledImage, currentWidth, currentHeight);

}

// Última reducción precisa al tamaño final

scaledImage = resizeImage(scaledImage, targetWidth, targetHeight);

return scaledImage;

}

// Método sugerido por chatGPT para interpolación bicúbica que mantiene la imagen con calidad después de resizearla

private BufferedImage resizeImage(BufferedImage original, int width, int height) {

BufferedImage resized = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_ARGB);

Graphics2D g2d = resized.createGraphics();

// Activar interpolación de alta calidad

g2d.setRenderingHint(RenderingHints.KEY\_INTERPOLATION, RenderingHints.VALUE\_INTERPOLATION\_BICUBIC);

g2d.setRenderingHint(RenderingHints.KEY\_RENDERING, RenderingHints.VALUE\_RENDER\_QUALITY);

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

// Dibujar la imagen escalada

g2d.drawImage(original, 0, 0, width, height, null);

g2d.dispose();

return resized;

}

Esta fue la última parte que desarrollé del proyecto. Es el funcionamiento para el visualizador de archivos, aunque sólo pude implementarlo en pdf. Permite mostrar archivos PDF en un panel, así como el tamaño de estos, y en caso de no tener previsualización, simplemente se da a conocer al usuario.