

Republic of the Philippines

BATANGAS STATE UNIVERSITY

The National Engineering University Lipa Campus

COLLEGE OF INFORMATICS AND COMPUTING SCIENCES

Department

FINAL PROJECT REPORT

CS 211: Object-Oriented Programming

1ST Semester, A.Y 2023-2024

**SCHOOLTECH –ICT SUPPORT**

Submitted by

DIMACULANGAN, EMMANUEL

SALUDAGA, GILBERT

VIRTUCIO, GERARD MICHAEL R

Instructor

# Ms. Aileen V. Suarez

DECEMBER 2023

# OVERVIEW

The project, implemented in Java, is designed to address diverse concerns within the technology and communication field. Its primary objective is to engage users, understand their issues, and provide recommendations and solutions to enhance their experience in the ICT domain. The system boasts a user-friendly interface tailored for students, ensuring easy navigation. The "Viewing Solutions" feature displays student concerns alongside potential resolutions, streamlining problem-solving. User authentication is prioritized, allowing secure logins via email and password. Each student benefits from a personalized dashboard, presenting relevant information and features. Communication tools are integrated, enabling student interaction with peers and instructors. The system allows students to submit concerns seamlessly. In summary, the project aims to create an efficient platform, fostering communication and issue resolution for students in the ICT domain.

# OBJECTIVE

The primary objective of the project is to create an inclusive and efficient platform for students in the ICT domain. By addressing their needs through personalized dashboards, streamlined issue resolution, and robust communication tools, the system aims to enhance the overall user experience and foster a proactive and responsive ICT community. The project seeks to optimize engagement, facilitate effective communication, and provide practical solutions to ensure a seamless experience for students navigating the challenges in the field of Information and Communication Technology.

# METHODOLOGY

**Users:**

The program initiates with a welcoming message and seamlessly transitions into the Viewing Solution features.

**Viewing Solution**

Initiating the feature involves establishing a connection to a dedicated database for concerns. This connection retrieves and displays all tables within the database, featuring an inactive button and an 'Other Concerns' button redirecting users to the login interface. Upon selecting a category (table), the system directs users to an interface listing all questions within that specific table. Choosing a concern triggers the retrieval of its solution from the dedicated database, leading users to an interface displaying the solution. Below, a question asks if the solution fixes the concern, allowing users to choose 'Yes' or 'No.' Depending on the response, the system may suggest an alternative solution, express gratitude, or provide contact information for Schooltech support. Following the 'Yes' or 'No' decision, a 'Like' button is presented. If no contact message is displayed, users return to the main interface; otherwise, they can choose to email Schooltech, with 'Yes' leading to the login interface.

**Login Features**

The "Login" feature facilitates user authentication by requiring input of email and password to access the emailing features of our system. It utilizes the LoginClass, an extension of the User class, inheriting attributes and introducing login-specific functionalities. When the login button is clicked, the performLogin method validates credentials, and verifyCredentials checks them against the database. This design promotes code reusability through inheritance and maintains modular, organized code with shared database connection methods. Upon successful login, users are seamlessly directed to our Emailing System.

**Register User**

The "Register User" feature is responsible for managing user registration functionalities, incorporating fields like SrCode, Name, Email, DateofBirth, Password, and ConfirmPass. Upon clicking "Register," the system invokes the getAllInputValues method within the RegisterUser class, an extension of the User class, to retrieve and validate user inputs, checking for empty fields and ensuring password consistency. The emailAlreadyExists method verifies email uniqueness by querying the database. Upon successful validation, it displays the user input details. A confirmation message prompts the user to verify the provided information, offering options to proceed or go back for further editing. Choosing "Yes" triggers the saveToDatabase method from the RegisterUser class, an extension of the User class, establishing a database connection and executing SQL statements to insert registration data, ensuring data integrity. Upon completion, the user is redirected to the Login Interface.

**Emailing System**

This interface features an accounts menu, email sorting, and functional buttons for tasks like composing, archiving/unarchiving, and deleting emails.

Email Sorting.

The sorting buttons utilize a dedicated panel with checkboxes, and clicking a button reveals emails corresponding to specific statuses. The display is tailored to show emails based on their status, with the displayAllEmailintoButtonCheckbox method from the UserEmailing class orchestrating this retrieval and presentation. Button statuses include "Inbox" for all emails sent and replied but not archived, "Archived" for all replied emails, and "Replied" for emails with replies, even if archived.

Well as you chose on the email u can see email information especially the reply and the message

Composing Email

When composing an email, the program automatically sets the sender as the logged-in user and the receiver as Schooltech ICT support. Users input a unique subject, ensuring it differs from previous emails. Clicking "Send" triggers the sendEmailToICTSupport method from the UserEmailing class, which validates the subject and message, establishes a database connection, checks for duplicates, timestamps the submission, and inserts essential email details. Users receive feedback on the outcome, and after a successful submission, clicking "OK" returns them to the Emailing main interface, displaying the sent email in the inbox.

Email Status Button

In the Emailing System Interface, functional buttons for tasks like archiving/unarchiving and deleting selected emails are present.

Clicking "Delete" utilizes the deleteSelectedEmails method in the UserEmailing class to delete selected emails.

Clicking "Archive" triggers archiveSelectedEmails, updating email statuses to "Archived. Users can view archived emails through the "Archived Emails" sorting option.

For unarchiving, specifically for archived emails, the call unarchiveSelectedEmails method manages the process, adjusting email statuses based on replies.

Clicking any of these actions automatically updates the display to reflect the changes.

Account Viewing

On the Email System menu, your name is displayed in the upper right, and clicking it reveals your account information, including SR Code, Name, Email, and Date of Birth. This information is retrieved from the database based on the logged-in email. Additionally, a "Delete Account" button is available. Clicking this button, associated with the AccountSettings class and the deleteLoggedInEmail() method, deletes the logged-in email and associated sent emails after user confirmation. A "Logout" button is also present on the upper right side.

**Admin:**

**Admin Login**

This interface requires the admin to input a specified username and email. Clicking the login button invokes the performAdminLogin method from the Admin class, verifying the login by checking if the entered username and password match records in the "admin" table. The method handles empty fields, limits login attempts to 3, and disables the login button after reaching the limit. Upon successful login, it redirects to the admin email interface.

**AdminEmail Interface**

The admin interface mirrors user features with options like "Emails Sent," displaying emails sent by all users without replies, and "Replied Emails," showcasing emails the admin has responded to (calling displayUserEmailButtons from AdminEmail Class). Additionally, there's a "Delete" button enabling the admin to delete selected emails.

Clicking on an email enables you to view all its information.

Admin Reply

Clicking on an email in "EmailUserSent" allows viewing sender details, date, time, subject, and message. If the email lacks a reply, a "Reply" button appears, leading to the reply interface where you can address the email's concerns. After composing your reply, clicking "Reply" utilizes the updateEmailStatus method from the AdminEmail class to send the response.

**Admin Control**

In the Admin Control section, clicking on "Account" reveals additional control functions, such as "Delete All Email" and "Delete User."

Clicking "Delete All Email" triggers the deleteAllEmails function from the Admin class (ParentClass), completely removing all emails stored in the database.

For "Delete User," users choose a name from the menu box, and upon clicking "Delete," it prompts confirmation for deleting the selected user. If confirmed, the system removes the chosen user from the database, updates the dropdown with the latest user list, and deletes the emails associated with that user.

# SOURCE CODE

**USERS:**

***User Class (Parent Class)***

public abstract class User {

protected Connection con;

private String srCode;

private String name;

private String email;

private String dateOfBirth;

private String password;

private String confirmPassword;

// Constructors

public User() {

// Default constructor

}

public User(String srCode, String name, String email, String dateOfBirth, String password, String confirmPassword) {

this.srCode = srCode;

this.name = name;

this.email = email;

this.dateOfBirth = dateOfBirth;

this.password = password;

this.confirmPassword = confirmPassword;

}

// Getters and Setters

public String getSrCode() {

return srCode;

}

public void setSrCode(String srCode) {

this.srCode = srCode;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getDateOfBirth() {

return dateOfBirth;

}

public void setDateOfBirth(String dateOfBirth) {

this.dateOfBirth = dateOfBirth;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getConfirmPassword() {

return confirmPassword;

}

public void setConfirmPassword(String confirmPassword) {

this.confirmPassword = confirmPassword;

}

protected void connectToDatabase() throws SQLException {

try {

Class.forName("com.mysql.cj.jdbc.Driver");

con = DriverManager.getConnection("jdbc:mysql://localhost:3306/schooltech?zeroDateTimeBehavior=CONVERT\_TO\_NULL", "root", "");

// Set autocommit to false for manual transaction management

con.setAutoCommit(false);

// Additional database setup, if needed

// If everything is successful, commit the changes

con.commit();

} catch (ClassNotFoundException | SQLException ex) {

// Rollback if an exception occurs

if (con != null) {

con.rollback();

}

ex.printStackTrace(); // Handle the exception appropriately

} finally {

// Always set autocommit back to true to avoid issues in subsequent transactions

if (con != null) {

con.setAutoCommit(true);

}

}

}

}

**Register User Class**

public class RegisterUser extends User{

private final JTextField SrCode;

private final JTextField Name;

private final JTextField Email;

private final JTextField DateofBirth;

private final JTextField Password;

private final JTextField ConfirmPass;

// Constructor

public RegisterUser(JTextField SrCode, JTextField Name, JTextField Email,

JTextField DateofBirth, JTextField Password, JTextField ConfirmPass) {

this.SrCode = SrCode;

this.Name = Name;

this.Email = Email;

this.DateofBirth = DateofBirth;

this.Password = Password;

this.ConfirmPass = ConfirmPass;

}

// Method to get all input values and perform validation

public boolean getAllInputValues(JLabel UserInput) {

// Get all input values

setSrCode(SrCode.getText().trim());

setName(Name.getText().trim());

setEmail(Email.getText().trim());

setDateOfBirth(DateofBirth.getText().trim());

setPassword(Password.getText());

setConfirmPassword(ConfirmPass.getText());

// For example, check if any of the required fields is empty

if (getSrCode().isEmpty() || getName().isEmpty() || getEmail().isEmpty() ||

getDateOfBirth().isEmpty() || getPassword().isEmpty() || getConfirmPassword().isEmpty()) {

// Display an error message or handle the case where required fields are empty

JOptionPane.showMessageDialog(null, "Please fill in all required fields.");

// TODO: Add code to handle validation error

return false; // Exit the method early if there's an error

}

// Check if password and confirm password match

if (!getPassword().equals(getConfirmPassword())) {

// Passwords don't match, display an error message

JOptionPane.showMessageDialog(null, "Passwords do not match. Please enter matching passwords.");

return false; // Exit the method early if there's an error

}

try {

if (emailAlreadyExists(getEmail())) {

// Email already exists, display an error message

JOptionPane.showMessageDialog(null, "Email is already in use. Please choose a different email.");

// TODO: Add code to handle duplicate email

return false; // Exit the method early if there's an error

}

} catch (SQLException ex) {

// Handle SQL exception, e.g., log the error

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Error checking email existence.");

return false; // Exit the method early if there's an error

}

UserInput.setText("<html>SR Code: " + getSrCode() + "<br>Name: " + getName() +

"<br>Email: " + getEmail() + "<br>Date of Birth: " + getDateOfBirth() +

"<br>Password: " + getPassword() + "</html>");

// All validations passed

return true;

}

private boolean emailAlreadyExists(String email) throws SQLException {

connectToDatabase();

String sql = "SELECT \* FROM users WHERE Emails = ?";

try (PreparedStatement preparedStatement = con.prepareStatement(sql)) {

preparedStatement.setString(1, email);

try (ResultSet resultSet = preparedStatement.executeQuery()) {

// If there is any result, the email already exists

return resultSet.next();

}

} catch (SQLException ex) {

// Handle the exception, e.g., log the error

ex.printStackTrace();

throw ex; // Re-throw the exception to propagate it up

}

}

public void saveToDatabase() throws SQLException {

connectToDatabase();

// Execute SQL statements to save user data to the database

// Example: Insert a new user record into a 'users' table

String sql = "INSERT INTO users (name, SrCode, Emails, DateOfBirth, Password) VALUES (?, ?, ?, ?, ?)";

try (PreparedStatement preparedStatement = con.prepareStatement(sql)) {

preparedStatement.setString(1, getName());

preparedStatement.setString(2, getSrCode());

preparedStatement.setString(3, getEmail());

preparedStatement.setString(4, getDateOfBirth());

preparedStatement.setString(5, getPassword());

// Execute the update

int rowsAffected = preparedStatement.executeUpdate();

if (rowsAffected <= 0) {

throw new SQLException("Failed to insert user data into the database.");

}

}

**Application of RegisterUser Class in RegisterAccount Jframe form:**

public class RegisterAccount extends javax.swing.JFrame {

public RegisterAccount() throws SQLException {

initComponents();

DoyouWanttoRegister.setText("<html> Ready to register? Press 'Yes' if everything is correct, or 'No' to edit. </html>");

FilloutPanel.setBackground(new Color (255,255,255,45));

RegisterConfirmation. setLocationRelativeTo(null);

setLocationRelativeTo(null);

}

private void NoActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_NoActionPerformed

RegisterConfirmation.dispose();

}//GEN-LAST:event\_NoActionPerformed

private void YESActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_YESActionPerformed

RegisterUser registerUser = new RegisterUser(SrCode, Name, Email, DateofBirth, Password, ConfirmPass);

try {

registerUser.getAllInputValues(AllUserInput);

// Call the method to save user data to the database

registerUser.saveToDatabase();

this.dispose();

RegisterConfirmation.dispose();

new Login\_Interface().setVisible(true);

// Optionally, display a success message or perform additional actions

JOptionPane.showMessageDialog(null, "User data saved to the database successfully!");

} catch (SQLException ex) {

// Handle the SQLException (e.g., display an error message)

JOptionPane.showMessageDialog(null, "Error saving user data to the database: " + ex.getMessage());

}

}//GEN-LAST:event\_YESActionPerformed

private void BackActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_BackActionPerformed

new Login\_Interface().setVisible(true);

this.dispose();

}//GEN-LAST:event\_BackActionPerformed

private void RegisterActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_RegisterActionPerformed

RegisterUser registerUser = new RegisterUser(SrCode, Name, Email, DateofBirth, Password, ConfirmPass);

// Check if all input values are valid before proceeding

if (!registerUser.getAllInputValues(AllUserInput)) {

// If there are validation errors, do not proceed

return;

}

RegisterConfirmation.setPreferredSize(new Dimension(231, 300));

RegisterConfirmation.pack();

RegisterConfirmation.setVisible(true);

}

}

**LoginUser Class extends User:**

public class LoginClass extends User {

private JTextField EmailInput;

private JPasswordField PasswordInput;

private String loggedInEmail;

public LoginClass(JTextField EmailInput, JPasswordField PasswordInput){

this.EmailInput = EmailInput;

this.PasswordInput = PasswordInput;

}

public boolean performLogin(){

String userEmail = EmailInput.getText();

String userPassword = new String(PasswordInput.getPassword());

if (userEmail.isEmpty() || userPassword.isEmpty()) {

JOptionPane.showMessageDialog(null, " Please fill all Required Field");

return false;

}

try {

// Call verifyCredentials to check login credentials

boolean credentialsValid = verifyCredentials(userEmail, userPassword);

if (credentialsValid) {

// Login successful, you can proceed with further actions

JOptionPane.showMessageDialog(null, "Login successful");

loggedInEmail = userEmail;

return true;

} else {

// Login failed, show an error message

JOptionPane.showMessageDialog(null, "Invalid email or password");

return false;

}

} catch (SQLException ex) {

// Handle SQL exception, e.g., log the error

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Error during login");

}

return true;

}

public String getloginemail(){

return loggedInEmail;

}

public boolean verifyCredentials(String email, String password) throws SQLException{

// Connect to the database using the 'con' connection obtained from the User class

connectToDatabase();

// Execute SQL statements to check login credentials

// Example: Select user record from a 'users' table based on the provided email and password

String sql = "SELECT \* FROM users WHERE Emails = ? AND password = ?";

try (PreparedStatement preparedStatement = con.prepareStatement(sql)) {

preparedStatement.setString(1, email);

preparedStatement.setString(2, password);

// Execute the query

ResultSet resultSet = preparedStatement.executeQuery();

// Check if a user with the provided credentials exists

return resultSet.next();

} catch (SQLException ex) {

// Handle the exception, e.g., log the error

ex.printStackTrace();

return false;

}

}

}

**Application of LoginClass to Login\_Interface Jframe form:**

public class Login\_Interface extends javax.swing.JFrame {

public Login\_Interface() {

initComponents();

setLocationRelativeTo(null);

LoginInput.setBackground(new Color (255,255,255,150));

ProjectName.setText("<html><div style='text-align: center;'> SchoolTech ICT Support </div></html>");

}

private void BackActionPerformed(java.awt.event.ActionEvent evt) try {

new MainInterface().setVisible(true);

} catch (SQLException ex) {

Logger.getLogger(Login\_Interface.class.getName()).log(Level.SEVERE, null, ex);

}

this.dispose();

}

private void PasswordInputActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_PasswordInputActionPerformed

char[] passwordChars = PasswordInput.getPassword();

LoginButtonActionPerformed(evt);

}

private void RegisterActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_RegisterActionPerformed

try {

new RegisterAccount().setVisible(true);

} catch (SQLException ex) {

Logger.getLogger(Login\_Interface.class.getName()).log(Level.SEVERE, null, ex);

}

this.dispose();

}

private void EmailInputActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_EmailInputActionPerformed

String userEmail = EmailInput.getText();

//if you press enter it wll jump to password input box

PasswordInput.requestFocusInWindow();

}

private void LoginButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_LoginButtonActionPerformed

LoginClass Loginverification = new LoginClass(EmailInput, PasswordInput);

if(!Loginverification.performLogin()){

return;

};

String LoginUser = Loginverification.getloginemail();

new UserEmailMainInterface(LoginUser).setVisible(true);

this.dispose();

}

**UserEmailing Class extends User:**

public class UserEmailing extends User{

private String loggedInEmail;

private final String emailFor = "SchoolTech@ict-support.com";

private String emailSubject;

private String emailMessage;

private String Replies = "";

private String Status = "Sent";

// Constructors, getters, and setters for the new properties

public UserEmailing() {

// Default constructor

}

public UserEmailing(String loggedInEmail, String emailSubject, String emailMessage, java.util.Date date) {

this.loggedInEmail = loggedInEmail;

this.emailSubject = emailSubject;

this.emailMessage = emailMessage;

}

// Method to get email data from UI components

public void getEmailDataFromUI(String LoginEmail, JTextField subjectField, JTextArea messageArea) {

this.loggedInEmail = LoginEmail;

this.emailSubject = subjectField.getText();

this.emailMessage = messageArea.getText();

}

// Method to send email information to the database

public boolean sendEmailToICTSupport() {

try {

// Check if subject and message are not empty

if (emailSubject.isEmpty() || emailMessage.isEmpty()) {

// Subject or message is empty, show a message dialog to the user

JOptionPane.showMessageDialog(null, "Subject and message cannot be empty", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

// Connect to the database using the 'con' connection obtained from the User class

connectToDatabase();

// Check if an email with the same subject from the same sender already exists

String checkDuplicateSql = "SELECT COUNT(\*) FROM emails WHERE EmailSender = ? AND Subject = ?";

try (PreparedStatement checkDuplicateStatement = con.prepareStatement(checkDuplicateSql)) {

checkDuplicateStatement.setString(1, loggedInEmail);

checkDuplicateStatement.setString(2, emailSubject);

ResultSet resultSet = checkDuplicateStatement.executeQuery();

if (resultSet.next() && resultSet.getInt(1) > 0) {

// A duplicate email with the same subject and sender exists

JOptionPane.showMessageDialog(null, "An email with the same subject already exists.", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

}

LocalDateTime currentDateTime = LocalDateTime.now();

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

String formattedDateTime = currentDateTime.format(formatter);

// Execute SQL statement to insert email information into the 'emails' table

String insertSql = "INSERT INTO emails (EmailSender, Receiver, DateTime, Subject, Message, Status, Replies) " +

"VALUES (?, ?, ?, ?, ?, ?, ?)";

try (PreparedStatement insertStatement = con.prepareStatement(insertSql)) {

insertStatement.setString(1, loggedInEmail);

insertStatement.setString(2, emailFor);

insertStatement.setString(3, formattedDateTime);

insertStatement.setString(4, emailSubject);

insertStatement.setString(5, emailMessage);

insertStatement.setString(6, Status);

insertStatement.setString(7, Replies);

// Execute the update

int rowsAffected = insertStatement.executeUpdate();

// Check if the insertion was successful

if (rowsAffected > 0) {

JOptionPane.showMessageDialog(null, "Email sent successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

return true;

} else {

JOptionPane.showMessageDialog(null, "Failed to send Email.", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

}

} catch (SQLException ex) {

// If there's an error during insertion, show a message dialog to the user

JOptionPane.showMessageDialog(null, "Failed to send email. Error: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

}

// Method to create buttons with checkboxes from SQL and add them to the target panel

public void displayAllEmailintoButtonCheckbox(String loggedInEmail, JPanel targetPanel, String EmailStatus) {

try {

// Connect to the database using the 'con' connection obtained from the User class

connectToDatabase();

// Execute SQL statement to select emails based on the predefined status

// Adjust the SQL query to handle the case where the status is not equal to a specific value

String selectSql;

if (EmailStatus.startsWith("!")) {

// If EmailStatus starts with "!", it means we want emails with a status not equal to the provided value

selectSql = "SELECT Subject, DateTime FROM emails WHERE EmailSender = ? AND Status != ? ORDER BY DateTime DESC";

// Remove "!" from the beginning of EmailStatus for comparison

EmailStatus = EmailStatus.substring(1);

} else if ("Replied".equalsIgnoreCase(EmailStatus)) {

// If EmailStatus is "togot", it means we want emails that have been replied

selectSql = "SELECT Subject, DateTime FROM emails WHERE EmailSender = ? AND (Status = ? OR (Status = 'Archived' AND Replies <> '')) ORDER BY DateTime DESC";

}else {

// Otherwise, select emails with the specified status

selectSql = "SELECT Subject, DateTime FROM emails WHERE EmailSender = ? AND Status = ? ORDER BY DateTime DESC";

}

try (PreparedStatement selectStatement = con.prepareStatement(selectSql)) {

selectStatement.setString(1, loggedInEmail);

selectStatement.setString(2, EmailStatus);

// Execute the query

ResultSet resultSet = selectStatement.executeQuery();

targetPanel.setLayout(new BoxLayout(targetPanel, BoxLayout.Y\_AXIS));

while (resultSet.next()) {

String emailSubject = resultSet.getString("Subject");

String dateTime = resultSet.getString("DateTime");

targetPanel.add(Box.createVerticalStrut(6));

boolean isReplied = isEmailReplied(loggedInEmail, emailSubject);

createButtonWithCheckbox(emailSubject, dateTime, isReplied, targetPanel);

}

}

} catch (SQLException e) {

e.printStackTrace(); // Handle the exception appropriately

}

SwingUtilities.invokeLater(() -> {

targetPanel.revalidate();

targetPanel.repaint();

});

}

private void createButtonWithCheckbox(String subject, String dateTime, boolean isReplied, JPanel targetPanel ) {

// Create a button

JButton emailButton = new JButton("<html><style>body { white-space: nowrap; }</style><font size='1'>" + dateTime + "</font> - <b><font size='3'>" + subject + "</font></b></html>");

buttonProperties(emailButton, isReplied);

// Create a checkbox

JCheckBox checkBox = new JCheckBox();

// Create a panel for the button and checkbox with BoxLayout

JPanel buttonPanel = new JPanel();

buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X\_AXIS));

// Add rigid area before the button for fixed horizontal space

buttonPanel.add(Box.createRigidArea(new Dimension(10, 0)));

// Add the checkbox and button to the panel

buttonPanel.add(checkBox);

buttonPanel.add(emailButton);

// Add the panel to the target panel

targetPanel.add(buttonPanel);

// Add rigid area after the button for fixed horizontal space

targetPanel.add(Box.createRigidArea(new Dimension(10, 0)));

// Revalidate and repaint the target panel

targetPanel.revalidate();

targetPanel.repaint();

// Add an ActionListener to the emailButton

emailButton.addActionListener((ActionEvent e) -> {

try {

handleEmailButtonClick(subject);

} catch (SQLException ex) {

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

}

});

checkBox.addActionListener((ActionEvent e) -> {

boolean isChecked = checkBox.isSelected();

checkedStatusMap.put(subject, isChecked);

});

}

// Add a Map to track the checked status for each subject

private Map<String, Boolean> checkedStatusMap = new HashMap<>();

// Method to set properties for the button

private void buttonProperties(JButton button, boolean isReplied) {

button.setHorizontalAlignment(SwingConstants.LEFT);

button.setAlignmentX(Component.LEFT\_ALIGNMENT);

button.setFont(new Font("Courier New", Font.BOLD, 13));

button.setForeground(new Color(33, 33, 33));

button.setBackground(new Color(153, 153, 153));

if (isReplied) {

button.setBorder(BorderFactory.createEtchedBorder(Color.PINK, Color.PINK));

} else {

button.setBackground(new Color(153, 153, 153));

button.setBorder(BorderFactory.createEtchedBorder());

}

button.setFocusPainted(false);

int newWidth = 370; // Set your preferred width

int newHeight = 40; // Set your preferred height

button.setPreferredSize(new Dimension(newWidth, newHeight));

button.setMinimumSize(new Dimension(newWidth, newHeight));

button.setMaximumSize(new Dimension(newWidth, newHeight));

}

public void archiveSelectedEmails(JPanel targetPanel, String loggedInEmail) {

// Display a confirmation dialog before proceeding with the archive action

int confirmResult = JOptionPane.showConfirmDialog(null,

"Are you sure you want to archive the selected emails?",

"Archive Confirmation", JOptionPane.YES\_NO\_OPTION);

if (confirmResult == JOptionPane.YES\_OPTION) {

for (Map.Entry<String, Boolean> entry : checkedStatusMap.entrySet()) {

String subject = entry.getKey();

boolean isChecked = entry.getValue();

if (isChecked) {

try {

// Perform archive action for the selected email

archiveEmail(subject, loggedInEmail);

// Update the display after archiving

displayAllEmailintoButtonCheckbox(loggedInEmail, targetPanel, Status);

} catch (SQLException ex) {

ex.printStackTrace(); // Handle the exception appropriately

}

}

}

// Clear the checked subjects map

checkedStatusMap.clear();

} else {

// User clicked No, do nothing or handle as needed

}

}

// Method to archive an email

private void archiveEmail(String subject, String emailSender) throws SQLException {

String archiveSql = "UPDATE emails SET Status = 'Archived' WHERE Subject = ? AND EmailSender = ?";

try (PreparedStatement archiveStatement = con.prepareStatement(archiveSql)) {

archiveStatement.setString(1, subject);

archiveStatement.setString(2, emailSender);

int rowsUpdated = archiveStatement.executeUpdate();

if (rowsUpdated > 0) {

// Display a JOptionPane to inform the user about the archiving action

JOptionPane.showMessageDialog(null, "Email with subject '" + subject + "' has been archived.");

} else {

// Handle the case where the update didn't succeed

JOptionPane.showMessageDialog(null, "Failed to archive email with subject '" + subject + "'.");

}

} catch (SQLException ex) {

ex.printStackTrace(); // Handle the exception appropriately

}

}

private void handleEmailButtonClick(String subject) throws SQLException {

new ViewSelectedEmail(subject).setVisible(true);

}

public void unarchiveSelectedEmails(String emailSender) {

// Display a confirmation dialog before proceeding with the unarchive action

int confirmResult = JOptionPane.showConfirmDialog(null,

"Are you sure you want to unarchive the selected emails?",

"Unarchive Confirmation", JOptionPane.YES\_NO\_OPTION);

if (confirmResult == JOptionPane.YES\_OPTION) {

// Proceed with unarchiving for selected emails

for (Map.Entry<String, Boolean> entry : checkedStatusMap.entrySet()) {

String subject = entry.getKey();

boolean isChecked = entry.getValue();

if (isChecked) {

try {

// Perform unarchive action for the selected email

unarchiveEmail(subject, emailSender);

} catch (SQLException ex) {

ex.printStackTrace(); // Handle the exception appropriately

}

}

}

// Clear the checked subjects map

checkedStatusMap.clear();

} else {

}

}

private void unarchiveEmail(String subject, String emailSender) throws SQLException {

// Display a confirmation dialog before proceeding with the unarchive action

String selectSql = "SELECT \* FROM emails WHERE Status = 'Replied' AND EmailSender = ? AND Subject = ?";

try (PreparedStatement selectStatement = con.prepareStatement(selectSql)) {

selectStatement.setString(1, emailSender);

selectStatement.setString(2, subject);

// Execute the query

try (ResultSet resultSet = selectStatement.executeQuery()) {

boolean hasReply = resultSet.next();

// Update the email status based on whether there is a reply

String updateSql;

if (hasReply) {

updateSql = "UPDATE emails SET Status = 'Replied' WHERE Subject = ? AND EmailSender = ?";

} else {

updateSql = "UPDATE emails SET Status = 'Sent' WHERE Subject = ? AND EmailSender = ?";

}

try (PreparedStatement updateStatement = con.prepareStatement(updateSql)) {

updateStatement.setString(1, subject);

updateStatement.setString(2, emailSender);

int rowsUpdated = updateStatement.executeUpdate();

if (rowsUpdated > 0) {

// Display a JOptionPane to inform the user about the unarchiving action

JOptionPane.showMessageDialog(null, "Email with subject '" + subject + "' has been unarchived.");

} else {

// Handle the case where the update didn't succeed

JOptionPane.showMessageDialog(null, "Failed to unarchive email with subject '" + subject + "'.");

}

}

}

} catch (SQLException ex) {

ex.printStackTrace(); // Handle the exception appropriately

}

}

private boolean isEmailReplied(String loggedInEmail, String emailSubject) {

try {

// Connect to the database

connectToDatabase();

// Execute SQL statement to check if the email has a reply

String selectSql = "SELECT \* FROM emails WHERE EmailSender = ? AND Subject = ? AND Replies <> ''";

try (PreparedStatement selectStatement = con.prepareStatement(selectSql)) {

selectStatement.setString(1, loggedInEmail);

selectStatement.setString(2, emailSubject);

// Execute the query

try (ResultSet resultSet = selectStatement.executeQuery()) {

// Check if there is at least one row in the result set

return resultSet.next();

}

}

} catch (SQLException e) {

e.printStackTrace(); // Handle the exception appropriately

return false; // Return false in case of an exception

}

}

public void deleteSelectedEmails(JPanel targetPanel) {

// Display a confirmation dialog before proceeding with the delete action

int confirmResult = JOptionPane.showConfirmDialog(null,

"Are you sure you want to delete the selected emails?",

"Delete Confirmation", JOptionPane.YES\_NO\_OPTION);

if (confirmResult == JOptionPane.YES\_OPTION) {

// Proceed with deletion for selected emails

targetPanel.removeAll();

for (Map.Entry<String, Boolean> entry : checkedStatusMap.entrySet()) {

String subject = entry.getKey();

boolean isChecked = entry.getValue();

if (isChecked) {

try {

// Perform delete action for the selected email

deleteEmail(subject, targetPanel);

} catch (SQLException ex) {

ex.printStackTrace(); // Handle the exception appropriately

}

}

}

// Clear the checked subjects map

checkedStatusMap.clear();

} else {

// User clicked No, do nothing or handle as needed

}

}

// Update the deleteEmail method to include the targetPanel parameter

private void deleteEmail(String subject, JPanel targetPanel) throws SQLException {

// Perform the email deletion logic without asking for confirmation again

String deleteSql = "DELETE FROM emails WHERE Subject = ?";

try (PreparedStatement deleteStatement = con.prepareStatement(deleteSql)) {

deleteStatement.setString(1, subject);

int rowsDeleted = deleteStatement.executeUpdate();

if (rowsDeleted > 0) {

// Display a JOptionPane to inform the user about the deletion action

JOptionPane.showMessageDialog(null, "Email with subject '" + subject + "' has been deleted.");

} else {

// Handle the case where the deletion didn't succeed

JOptionPane.showMessageDialog(null, "Failed to delete email with subject '" + subject + "'.");

}

} catch (SQLException ex) {

ex.printStackTrace(); // Handle the exception appropriately

}

}

}

**Account Setting Class Extends User Class:**

public class AccounTSettings extends User {

private String loggedInEmail;

public AccounTSettings (String Loginemail){

this.loggedInEmail = Loginemail;

}

public boolean deleteLoggedInEmail() throws SQLException {

try {

// Connect to the database using the 'con' connection obtained from the User class

connectToDatabase();

// Ask the user for confirmation

int confirm = JOptionPane.showConfirmDialog(null,

"Are you sure you want to delete the login email and all sent emails?", "Confirmation", JOptionPane.YES\_NO\_OPTION);

if (confirm == JOptionPane.YES\_OPTION) {

// Execute SQL statement to delete the login email

String deleteLoginSql = "DELETE FROM users WHERE Emails = ?";

String deleteSentEmailsSql = "DELETE FROM emails WHERE EmailSender = ?";

try {

// Set autocommit to false to ensure atomicity of the operations

con.setAutoCommit(false);

// Delete sent emails

try (PreparedStatement deleteSentEmailsStatement = con.prepareStatement(deleteSentEmailsSql)) {

deleteSentEmailsStatement.setString(1, loggedInEmail);

deleteSentEmailsStatement.executeUpdate();

}

// Delete the login email

try (PreparedStatement deleteLoginStatement = con.prepareStatement(deleteLoginSql)) {

deleteLoginStatement.setString(1, loggedInEmail);

// Execute the deletion

int rowsAffected = deleteLoginStatement.executeUpdate();

// Check if the deletion was successful

if (rowsAffected > 0) {

// Display a JOptionPane message for successful deletion

JOptionPane.showMessageDialog(null, "Login email and sent emails deleted successfully");

// Commit the transaction

con.commit();

// Return true indicating successful deletion

return true;

} else {

// Display a JOptionPane message for unsuccessful deletion

JOptionPane.showMessageDialog(null, "Error deleting login email");

// Rollback the transaction

con.rollback();

// Return false indicating unsuccessful deletion

return false;

}

}

} finally {

// Set autocommit back to true after completion

con.setAutoCommit(true);

}

} else {

// Return false if the user chooses not to delete

return false;

}

} catch (SQLException ex) {

// Handle the exception, e.g., log the error

// Display a JOptionPane message for the exception

JOptionPane.showMessageDialog(null, "Error deleting login email: " + ex.getMessage());

// Return false indicating an error occurred

return false;

} catch (Exception ex) {

// Handle other exceptions, e.g., user closed the confirmation dialog

// Display a JOptionPane message for the exception

JOptionPane.showMessageDialog(null, "Error: " + ex.getMessage());

// Return false indicating an error occurred

return false;

}

}

public void displayAllUserInfo(JLabel userInfoLabel) {

try {

// Connect to the database using the 'con' connection obtained from the User class

connectToDatabase();

// Execute SQL statement to select user information based on the login email

String selectSql = "SELECT \* FROM users WHERE Emails = ?";

try (PreparedStatement selectStatement = con.prepareStatement(selectSql)) {

selectStatement.setString(1, loggedInEmail);

// Execute the query

ResultSet resultSet = selectStatement.executeQuery();

// Check if a user with the provided email exists

if (resultSet.next()) {

// Get user information from the result set

String srCode = resultSet.getString("SrCode");

String name = resultSet.getString("Name");

String email = resultSet.getString("Emails");

String dateOfBirth = resultSet.getString("DateOfBirth");

// Set user information to the JLabel

String userInfoText = "<html> <div style='margin: 5px;'> Name: " + name + "</div>"

+ "<div style='margin: 5px;'>SR Code: " + srCode + "</div>"

+ "<div style='margin: 5px;'>Email: " + email + "</div>"

+ "<div style='margin: 5px;'>Date of Birth: " + dateOfBirth + "</div></html>";

userInfoLabel.setText(userInfoText);

} else {

// If no user with the provided email is found, set an error message

userInfoLabel.setText("User not found");

}

}

} catch (SQLException ex) {

// Handle the exception, e.g., log the error

// If there's an error, set an error message to the JLabel

userInfoLabel.setText("Error fetching user information: " + ex.getMessage());

}

}

public void updateButtonWithUserName(JButton loginButton) {

try {

// Connect to the database using the 'con' connection obtained from the User class

connectToDatabase();

// Execute SQL statement to select the user's name based on the login email

String selectSql = "SELECT Name FROM users WHERE Emails = ?";

try (PreparedStatement selectStatement = con.prepareStatement(selectSql)) {

selectStatement.setString(1, loggedInEmail);

// Execute the query

ResultSet resultSet = selectStatement.executeQuery();

// Check if a user with the provided email exists

if (resultSet.next()) {

// Get the user's name from the result set

String userName = resultSet.getString("Name");

// Update the button's text with the user's name

loginButton.setText( userName);

} else {

// If no user with the provided email is found, set a default text for the button

loginButton.setText("Login");

}

}

} catch (SQLException ex) {

// Handle the exception, e.g., log the error

// If there's an error, set a default text for the button

loginButton.setText("Login");

}

}

**}**

**Apllication of Useremailing Class and AccountSetting class to EmailingInterface Form:**

public class UserEmailMainInterface extends javax.swing.JFrame {

UserEmailing userEmail = new UserEmailing();

private final String LoginUser;

private String currentStatus;

public UserEmailMainInterface(String LoginUser) {

initComponents();

this.LoginUser = LoginUser;

//set UserAccount ButtonNametoLoginusername

UserAccountButton.setText(LoginUser);

Inbox.doClick();

AccounTSettings accinfo = new AccounTSettings(LoginUser);

accinfo.updateButtonWithUserName(UserAccountButton);

setLocationRelativeTo(null);

}

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

AccounTSettings accountSettings = new AccounTSettings(LoginUser);

boolean deletionSuccessful;

try {

deletionSuccessful = accountSettings.deleteLoggedInEmail();

if (deletionSuccessful) {

Account.dispose();

this.dispose();

new Login\_Interface().setVisible(true);

} else {

}

} catch (SQLException ex) {

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

}

}

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

// Get email data from UI components

userEmail.getEmailDataFromUI(LoginUser, ComposeSubject, MessageTextArea);

// Send email information to the database

boolean emailSentSuccessfully = userEmail.sendEmailToICTSupport();

if (!emailSentSuccessfully) {

return;

}

ComposeEmail.dispose();

MessageTextArea.setText("");

ComposeSubject.setText("");

EmailPanel.removeAll();

revalidate();

repaint();

InboxActionPerformed(evt);

Inbox.doClick();

}

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

AccounTSettings accounTSettings = new AccounTSettings(LoginUser);

accounTSettings.displayAllUserInfo(UserInfo);

Account.setPreferredSize(new Dimension(280, 400));

Account.pack();

Account.setLocationRelativeTo(null);

Account.setVisible(true);

}

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

UnarchiveButton.setVisible(false);

currentStatus = "Replied";

ButtonCLick(evt);

ArchivingButton.setVisible(true);

EmailPanel.removeAll();

userEmail.displayAllEmailintoButtonCheckbox(LoginUser, EmailPanel, currentStatus);

revalidate();

repaint();

}

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

UnarchiveButton.setVisible(true);

ArchivingButton.setVisible(false);

EmailPanel.removeAll();

jScrollPane3.revalidate();

currentStatus = "Archived";

userEmail.displayAllEmailintoButtonCheckbox(LoginUser, EmailPanel, currentStatus);

revalidate();

repaint();

}

private void InboxActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_InboxActionPerformed

ButtonCLick(evt);

UnarchiveButton.setVisible(false);

ArchivingButton.setVisible(true);

EmailPanel.removeAll();

currentStatus = "!Archived";

userEmail.displayAllEmailintoButtonCheckbox(LoginUser, EmailPanel, currentStatus);

revalidate();

repaint();

}

private void BackActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_BackActionPerformed

int confirmLogout = JOptionPane.showConfirmDialog(null,

"Are you sure you want to log out?", "Logout Confirmation", JOptionPane.YES\_NO\_OPTION);

if (confirmLogout == JOptionPane.YES\_OPTION) {

new Login\_Interface().setVisible(true);

this.dispose();

} else {

// User chose not to log out, you can add any additional actions here

}

}

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

EmailerName.setText("From: " +LoginUser);

ComposeEmail.setPreferredSize(new Dimension(369, 475));

ComposeEmail.pack();

ComposeEmail.setLocationRelativeTo(null);

ComposeEmail.setVisible(true);

}

private void DeleteActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_DeleteActionPerformed

userEmail.deleteSelectedEmails(EmailPanel);

EmailPanel.removeAll();

userEmail.displayAllEmailintoButtonCheckbox(LoginUser, EmailPanel, currentStatus);

EmailPanel.revalidate();

EmailPanel.repaint();

}

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

Account.dispose();

}

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

new Login\_Interface().setVisible(true);

Account.dispose();

this.dispose();

}

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

userEmail.archiveSelectedEmails(EmailPanel, LoginUser);

EmailPanel.removeAll();

userEmail.displayAllEmailintoButtonCheckbox(LoginUser, EmailPanel, currentStatus);

EmailPanel.revalidate();

EmailPanel.repaint();

}

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

userEmail.unarchiveSelectedEmails(LoginUser);

EmailPanel.removeAll();

userEmail.displayAllEmailintoButtonCheckbox(LoginUser, EmailPanel, currentStatus);

EmailPanel.revalidate();

EmailPanel.repaint();

}

}

**ADMIN:**

**AdminClass(Parent Class):**

public class Admin {

private String loggedInUsername;

protected Connection con;

private static int MAX\_LOGIN\_ATTEMPTS = 3;

public boolean performAdminLogin(JTextField adminUsernameField, JPasswordField adminPasswordField, JButton loginbutton ){

String username = adminUsernameField.getText();

String password = new String(adminPasswordField.getPassword());

if (username.isEmpty() || password.isEmpty()) {

JOptionPane.showMessageDialog(null, "Please fill all required fields");

return false;

}

try {

connectToDatabase();

// Execute SQL statements to check login credentials

String sql = "SELECT \* FROM admin WHERE AdminEmail = ? AND AdminPass = ?";

try (PreparedStatement preparedStatement = con.prepareStatement(sql)) {

preparedStatement.setString(1, username);

preparedStatement.setString(2, password);

ResultSet resultSet = preparedStatement.executeQuery();

// Check if a user with the provided credentials exists

boolean credentialsValid = resultSet.next();

if (credentialsValid) {

// Reset login attempts on successful login

// Login successful, you can proceed with further actions

// For example, show a new frame or perform other operations

JOptionPane.showMessageDialog(null, "Admin login successful");

loggedInUsername = username;

return true;

} else {

MAX\_LOGIN\_ATTEMPTS--;

if (MAX\_LOGIN\_ATTEMPTS == 0) {

// Disable the frame or take appropriate action

JOptionPane.showMessageDialog(null, "Too many invalid login attempts. Frame will be disabled.");

loginbutton.setEnabled(false);

} else {

JOptionPane.showMessageDialog(null, "Invalid admin username or password. Attempts left: " + MAX\_LOGIN\_ATTEMPTS);

}

}

}

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

protected void connectToDatabase() throws SQLException {

try {

Class.forName("com.mysql.cj.jdbc.Driver");

con = DriverManager.getConnection("jdbc:mysql://localhost:3306/schooltech?zeroDateTimeBehavior=CONVERT\_TO\_NULL", "root", "");

} catch (ClassNotFoundException ex) {

}

}

public String getLoggedInUsername() {

return loggedInUsername;

}

public boolean verifyAdminCredentials(String username, String password) throws SQLException {

// Connect to the database using the 'con' connection obtained from the Admin class

connectToDatabase();

// Execute SQL statements to check admin login credentials

String sql = "SELECT \* FROM admin WHERE AdminEmail = ? AND AdminPass = ?";

try (PreparedStatement preparedStatement = con.prepareStatement(sql)) {

preparedStatement.setString(1, username);

preparedStatement.setString(2, password);

// Execute the query

ResultSet resultSet = preparedStatement.executeQuery();

// Check if an admin with the provided credentials exists

return resultSet.next();

} catch (SQLException ex) {

// Handle the exception, e.g., log the error

ex.printStackTrace();

return false;

}

}

public void deleteAllEmails(JPanel targetPanel) {

try {

connectToDatabase();

int dialogResult = JOptionPane.showConfirmDialog(null,

"Are you sure you want to delete all emails?", "Confirmation",

JOptionPane.YES\_NO\_OPTION);

if (dialogResult == JOptionPane.YES\_OPTION) {

// Remove all email panels from the target panel

targetPanel.removeAll();

// Delete all emails from the database

String deleteSql = "DELETE FROM emails";

try (Statement statement = con.createStatement()) {

statement.executeUpdate(deleteSql);

}

// Refresh the target panel

targetPanel.revalidate();

targetPanel.repaint();

}

} catch (SQLException e) {

// Handle the exception appropriately (e.g., logging or displaying an error message)

}

}

public void populateUserComboBox(JComboBox<String> userComboBox, JButton deleteButton) {

try {

connectToDatabase();

// Execute SQL statement to select names and emails of users

String selectSql = "SELECT Name, Emails FROM users";

try (Statement statement = con.createStatement();

ResultSet resultSet = statement.executeQuery(selectSql)) {

// Clear existing items in the combo box

userComboBox.removeAllItems();

// Add "All" as the first item

userComboBox.addItem("Choose User");

// Add names and emails to the combo box

while (resultSet.next()) {

String name = resultSet.getString("Name");

String email = resultSet.getString("Emails");

String displayInfo = name + " - " + email;

userComboBox.addItem(displayInfo);

}

}

} catch (SQLException e) {

// Handle the exception appropriately

}

// Add an ActionListener to the deleteButton

deleteButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// Get the selected item from the combo box

String selectedItem = (String) userComboBox.getSelectedItem();

// Ask the user for confirmation

int confirm = JOptionPane.showConfirmDialog(null,

"Are you sure you want to delete the selected user?", "Confirmation", JOptionPane.YES\_NO\_OPTION);

if (confirm == JOptionPane.YES\_OPTION && selectedItem != null && !"All".equals(selectedItem)) {

// Extract the email from the selected item (assuming the email is separated by " - ")

String[] parts = selectedItem.split(" - ");

if (parts.length > 1) {

String selectedEmail = parts[1];

// Execute SQL statement to delete the user

String deleteSql = "DELETE FROM users WHERE Emails = ?";

try (PreparedStatement deleteStatement = con.prepareStatement(deleteSql)) {

deleteStatement.setString(1, selectedEmail);

deleteStatement.executeUpdate();

} catch (SQLException ex) {

// Handle the exception appropriately

}

// Repopulate the combo box after deletion

populateUserComboBox(userComboBox, deleteButton);

}

}

}

});

}

**Application of Admin Class to Admin Login:**

public class AdminLogin extends javax.swing.JFrame {

public AdminLogin() {

initComponents();

ADMINLOGININPUT.setBackground(new Color(102,102,102, 100));

setLocationRelativeTo(null);

}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton1ActionPerformed

Admin admin = new Admin();

boolean loginSuccess = admin.performAdminLogin(AdminUsername, AdminPassword, jButton1 );

if (loginSuccess) {

this.dispose();

new AdminEmailMainInterface().setVisible(true);

}

}

}

**Admin Email Class extends Admin class:**

public class AdminEmail extends Admin{

public AdminEmail() {

// Default constructor

}

public AdminEmail(String loggedInEmail, String emailSubject, String emailMessage, java.util.Date date) {

}

public boolean updateEmailStatus(String subject, JTextPane textArea) {

try {

// Connect to the database using the 'con' connection obtained from the User class

connectToDatabase();

// Check if the text area is empty

if (textArea.getText().trim().isEmpty()) {

JOptionPane.showMessageDialog(null, "Reply cannot be empty.", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

// Ask the user for confirmation

int confirmation = JOptionPane.showConfirmDialog(null, "Do you want to send this reply?", "Confirmation", JOptionPane.YES\_NO\_OPTION);

if (confirmation == JOptionPane.YES\_OPTION) {

// Update the status to "Replied" in the 'emails' table

String updateSql = "UPDATE emails SET Status = 'Replied', Replies = ? WHERE Subject = ?";

try (PreparedStatement updateStatement = con.prepareStatement(updateSql)) {

// Set the content of the JTextArea as the reply in the database

updateStatement.setString(1, textArea.getText());

updateStatement.setString(2, subject);

// Execute the update query

updateStatement.executeUpdate();

return true; // Return true if the update was successful

}

} else {

// User clicked NO, you can handle this case if needed

return false;

}

} catch (SQLException e) {

e.printStackTrace(); // Handle the exception appropriately

return false;

}

}

// Method to create buttons with checkboxes from SQL and add them to the target panel

public void displayUserEmailButtons(JPanel targetPanel, String status) {

try {

// Connect to the database using the 'con' connection obtained from the User class

connectToDatabase();

String selectSql;

if ("Replied".equalsIgnoreCase(status)) {

// If EmailStatus is "Replied", it means we want emails that have been replied or archived

selectSql = "SELECT Subject, DateTime FROM emails WHERE Status = ? OR (Status = 'Archived' AND Replies <> '') ORDER BY DateTime DESC";

} else if ("AllArchivedSent".equalsIgnoreCase(status)) {

// If EmailStatus is "AllArchivedSent", it means we want all archived and sent emails without a reply

selectSql = "SELECT Subject, DateTime FROM emails WHERE (Status IN ('Archived', 'Sent') AND Replies = '') ORDER BY DateTime DESC";

}else {

// Otherwise, select emails with the specified status

selectSql = "SELECT Subject, DateTime FROM emails WHERE Status = ? ORDER BY DateTime DESC";

}

try (PreparedStatement selectStatement = con.prepareStatement(selectSql)) {

if (!"AllArchivedSent".equalsIgnoreCase(status)) {

// Set the status parameter only if it's not "AllArchivedSent"

selectStatement.setString(1, status);

}

// Execute the query

ResultSet resultSet = selectStatement.executeQuery();

// Use BoxLayout for vertical arrangement

targetPanel.setLayout(new BoxLayout(targetPanel, BoxLayout.Y\_AXIS));

while (resultSet.next()) {

String emailSubject = resultSet.getString("Subject");

String dateTime = resultSet.getString("DateTime");

createButtonWithCheckbox(emailSubject, dateTime, targetPanel);

targetPanel.add(Box.createVerticalStrut(6));

}

}

} catch (SQLException e) {

// Handle the exception appropriately

}

targetPanel.revalidate();

targetPanel.repaint();

}

private void createButtonWithCheckbox(String subject, String dateTime, JPanel targetPanel) {

// Create a button

JButton emailButton = new JButton("<html><style>body { white-space: nowrap; }</style><font size='1'>" + dateTime + "</font> - <b><font size='3'>" + subject + "</font></b></html>");

buttonProperties(emailButton);

// Create a checkbox

JCheckBox checkBox = new JCheckBox();

// Create a panel for the button and checkbox with BoxLayout

JPanel buttonPanel = new JPanel();

buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X\_AXIS));

// Add rigid area before the button for fixed horizontal space

buttonPanel.add(Box.createRigidArea(new Dimension(10, 0)));

// Add the checkbox and button to the panel

buttonPanel.add(checkBox);

buttonPanel.add(emailButton);

// Add the panel to the target panel

targetPanel.add(buttonPanel);

// Add rigid area after the button for fixed horizontal space

targetPanel.add(Box.createRigidArea(new Dimension(10, 0)));

// Revalidate and repaint the target panel

targetPanel.revalidate();

targetPanel.repaint();

// Add an ActionListener to the emailButton

emailButton.addActionListener((ActionEvent e) -> {

try {

String emailSubject = e.getActionCommand();

handleEmailButtonClick(subject);

targetPanel.revalidate();

} catch (SQLException ex) {

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

}

});

checkBox.addActionListener((ActionEvent e) -> {

boolean isChecked = checkBox.isSelected();

checkedStatusMap.put(subject, isChecked);

System.out.println("Selected Subject: " + subject + ", Checked: " + isChecked);

});

}

// Method to set properties for the button

private void buttonProperties(JButton button) {

button.setHorizontalAlignment(SwingConstants.LEFT);

button.setAlignmentX(Component.LEFT\_ALIGNMENT);

button.setFont(new Font("Courier New", Font.BOLD, 13));

button.setForeground(new Color(33, 33, 33));

button.setBackground(new Color(153, 153, 153));

button.setBorder(BorderFactory.createEtchedBorder());

button.setFocusPainted(false);

int newWidth = 370; // Set your preferred width

int newHeight = 40; // Set your preferred height

button.setPreferredSize(new Dimension(newWidth, newHeight));

button.setMinimumSize(new Dimension(newWidth, newHeight));

button.setMaximumSize(new Dimension(newWidth, newHeight));

}

private void handleEmailButtonClick(String subject) throws SQLException {

new AdminViewSelectedEmail(subject).setVisible(true);

}

private Map<String, Boolean> checkedStatusMap = new HashMap<>();

public void deleteSelectedEmails(JPanel targetPanel) {

// Display a confirmation dialog before proceeding with the delete action

int confirmResult = JOptionPane.showConfirmDialog(null,

"Are you sure you want to delete the selected emails?",

"Delete Confirmation", JOptionPane.YES\_NO\_OPTION);

if (confirmResult == JOptionPane.YES\_OPTION) {

// Proceed with deletion for selected emails

targetPanel.removeAll();

for (Map.Entry<String, Boolean> entry : checkedStatusMap.entrySet()) {

String subject = entry.getKey();

boolean isChecked = entry.getValue();

if (isChecked) {

try {

// Perform delete action for the selected email

deleteEmail(subject, targetPanel);

} catch (SQLException ex) {

ex.printStackTrace(); // Handle the exception appropriately

}

}

}

// Clear the checked subjects map

checkedStatusMap.clear();

} else {

// User clicked No, do nothing or handle as needed

}

}

// Update the deleteEmail method to include the targetPanel parameter

private void deleteEmail(String subject, JPanel targetPanel) throws SQLException {

connectToDatabase();

// Perform the email deletion logic without asking for confirmation again

String deleteSql = "DELETE FROM emails WHERE Subject = ?";

try (PreparedStatement deleteStatement = con.prepareStatement(deleteSql)) {

deleteStatement.setString(1, subject);

int rowsDeleted = deleteStatement.executeUpdate();

if (rowsDeleted > 0) {

// Display a JOptionPane to inform the user about the deletion action

JOptionPane.showMessageDialog(null, "Email with subject '" + subject + "' has been deleted.");

// Revalidate and repaint the target panel to refresh the UI

SwingUtilities.invokeLater(() -> {

targetPanel.revalidate();

targetPanel.repaint();

System.out.println("Target panel refreshed after deletion.");

});

displayUserEmailButtons(targetPanel, subject);

} else {

// Handle the case where the deletion didn't succeed

JOptionPane.showMessageDialog(null, "Failed to delete email with subject '" + subject + "'.");

}

} catch (SQLException ex) {

ex.printStackTrace(); // Handle the exception appropriately

}

SwingUtilities.invokeLater(() -> {

targetPanel.revalidate();

targetPanel.repaint(); });

}

}

**Aplication of admin class and UsereEmailing class to AdminEmailInterface Jframe form:**

public class AdminEmailMainInterface extends javax.swing.JFrame {

AdminEmail adminemailreceive = new AdminEmail();

private String currentStatus;

public AdminEmailMainInterface() {

initComponents();

//set UserAccount ButtonNametoLoginusername

adminemailreceive.populateUserComboBox(UserList, DeleteUserButton);

EmailSent.doClick();

setLocationRelativeTo(null);

}

private void RepliedActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_RepliedActionPerformed

// Store the current status before updating to "Replied"

currentStatus = "Replied";

ButtonCLick(evt);

EmailPanel.removeAll();

adminemailreceive.displayUserEmailButtons(EmailPanel, currentStatus);

revalidate();

repaint();

}

private void EmailSentActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_EmailSentActionPerformed

ButtonCLick(evt);

currentStatus = "AllArchivedSent";

EmailPanel.removeAll();

adminemailreceive.displayUserEmailButtons(EmailPanel, currentStatus);

revalidate();

repaint();

}

private void DeleteActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_DeleteActionPerformed

EmailPanel.removeAll();

adminemailreceive.deleteSelectedEmails( EmailPanel);

adminemailreceive.displayUserEmailButtons(EmailPanel, currentStatus);

revalidate();

repaint();

}

private void DeleteAllEMailActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_DeleteAllEMailActionPerformed

adminemailreceive.deleteAllEmails(EmailPanel);

}

}