Criterion C – Development

Database Tables

Database name: EduTrack2.db

Table 1 – login

	ID	email	password
	Filter	Filter	Filter
1	1	preeti@gmail.com	preetiHanda

Table 2 - studentInfo

	sName	age	schoolName	curriculum	grade	email	pName	pEmail	pPhone	address
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Alice Joy	12	St. Jean School	International Baccalaureate	7	alice.j0524@gmail.com	Sylvie Joy	sylvie.j1212@...	0559341...	Golden Sands ...
2	Kendrick Sanders	17	Banglore High School	ISC	12	kendrick.s2024@gmail.com	Luna Sanders	luna.s98@gm...	0563295...	Katha ...
3	Sakina Patel	11	St. Jean-Pierre School	British	6	SAKINA.PATEL@gmail.com	Syed Patel	syed0305@g...	0564982...	C-Block, Trium...
4	John Jacob ...	16	Our Own International High School	International Baccalaureate	DP1	john.jacob42@gmail.com	Claudia Jacob	claudia2324@...	0552337...	Pito Apartment...
5	Jonathan Antony	9	Monty School - Primary	American	4	jonathan.a25@gmail.com	Patrick Antony	patrick2012@...	0521948...	Summit ...

Table 3 – tasks

	taskName	dueDate	completed	dueHour	dueMin	sName
	Filter	Filter	Filter	Filter	Filter	Filter
1	Pronom Toniques	19 Mar 2023	2	12	30	Jonathan Antony
2	Passe Compose	1 Nov 2023	1	12	30	Alice Joy
3	Futur proche	24 Mar 2023	1	23	55	Alice Joy
4	Gerondif	27 Oct 2023	1	23	55	John Jacob Tyreek

Table 4 – meetingRecords

	sName	mName	artHo	startMin	endHour	endMin	date
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Alice Joy	Meeting 1 - Alice	15	20	16	0	10 Mar 2024
2	John Jacob Tyreek	Meeting 1 - John	14	30	15	30	29 Dec 2023

Table 5 – files

	sName	fileName	file ▾¹
	Filter	Filter	Filter
1	Kendri...	French Notes PT	C:\Users\hiral\OneDrive\Desktop\HIRAL\School\IB1 and 2\IB1\3. DS SL\PracticeQuestions.docx
2	Sakina...	Compiled Exam Notes 1	C:\Users\hiral\OneDrive\Desktop\Spanish Vocab compiled\el-barrio.pdf

Table 6 – fees

	sName ▾1	totalAmount	feePaid	startDate	endDate	paid	feeName
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Jonathan Antony	1200	300	6 Oct 2023	6 Nov 2023	0	Fees 1
2	Jonathan Antony	2000	2000	6 Nov 2023	6 Dec 2023	1	Fees 2
3	Jonathan Antony	2000	0	6 Sep 2023	5 Oct 2023	2	Fees 3

Database Connection (using JDBC API – Java Database Connectivity API)

```
package com.mycompany.edutrack2;

//This is a java class which contains the main() function.
import java.sql.*;
import javax.swing.*;
import java.io.*;

public class clsDB {
    public static login log;
    public static Connection conn = null;

    /*Function to create a connection with the database and store the connection in a
    variable of type Connection.*/
    public static void java_db() {
        try {
            Class.forName( className: "org.sqlite.JDBC");
            //storing the connection in Connection conn
            conn = DriverManager.getConnection( url: "jdbc:sqlite:database/EduTrack2.db");
        } catch (ClassNotFoundException | SQLException e) {
            JOptionPane.showMessageDialog( parentComponent: null, message: e.getLocalizedMessage());
        }
    }
    public static void main(String[] args) throws IOException{
        //calling the function java_db() to form the connection
        java_db();
        //checking if connection has been created and stored
        if (conn != null) {
            JOptionPane.showMessageDialog( parentComponent: null, message: "Connected to the database.");
            //once connection has been created, the login page is opened up
            log = new login(conn);
            log.setVisible( b: true);
        } else {
            //else an error message is shown
            JOptionPane.showMessageDialog( parentComponent: null, "Failed to connect to the database."
                    + " Please ensure that the correct database has been downloaded.");
        }
    }
}
```

Conn is a variable of type Connection which stores the connection to the database, EduTrack2. The function java_db() contains code which is responsible for establishing the connection between the Java project and the database.

Database connection passed to the '*login*' JFrame.

In the main function, the java_db() function is called to form a connection with the database.

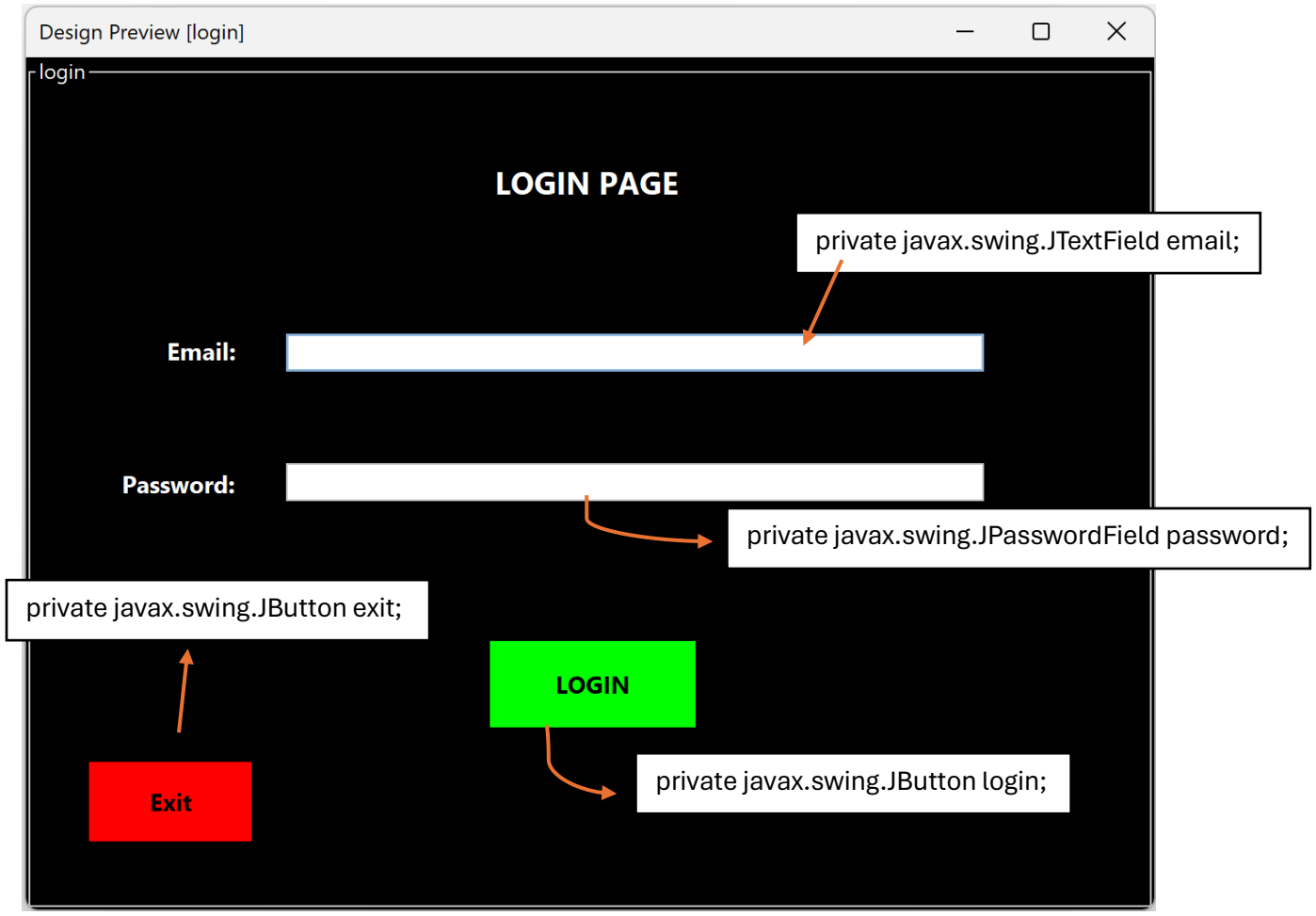
Calling and displaying the '*login*' JFrame.

If variable conn is not null (means it contains the connection to the database) then it opens up the Login page. Else, it displays an error message in the Output terminal.

The snippet above is a Java class called '*clsDB*'. This Java class is responsible for connecting the Java project with the database (which in this case is called EduTrack2). For this application, the SQL package, swing package as well as the io package has been imported in

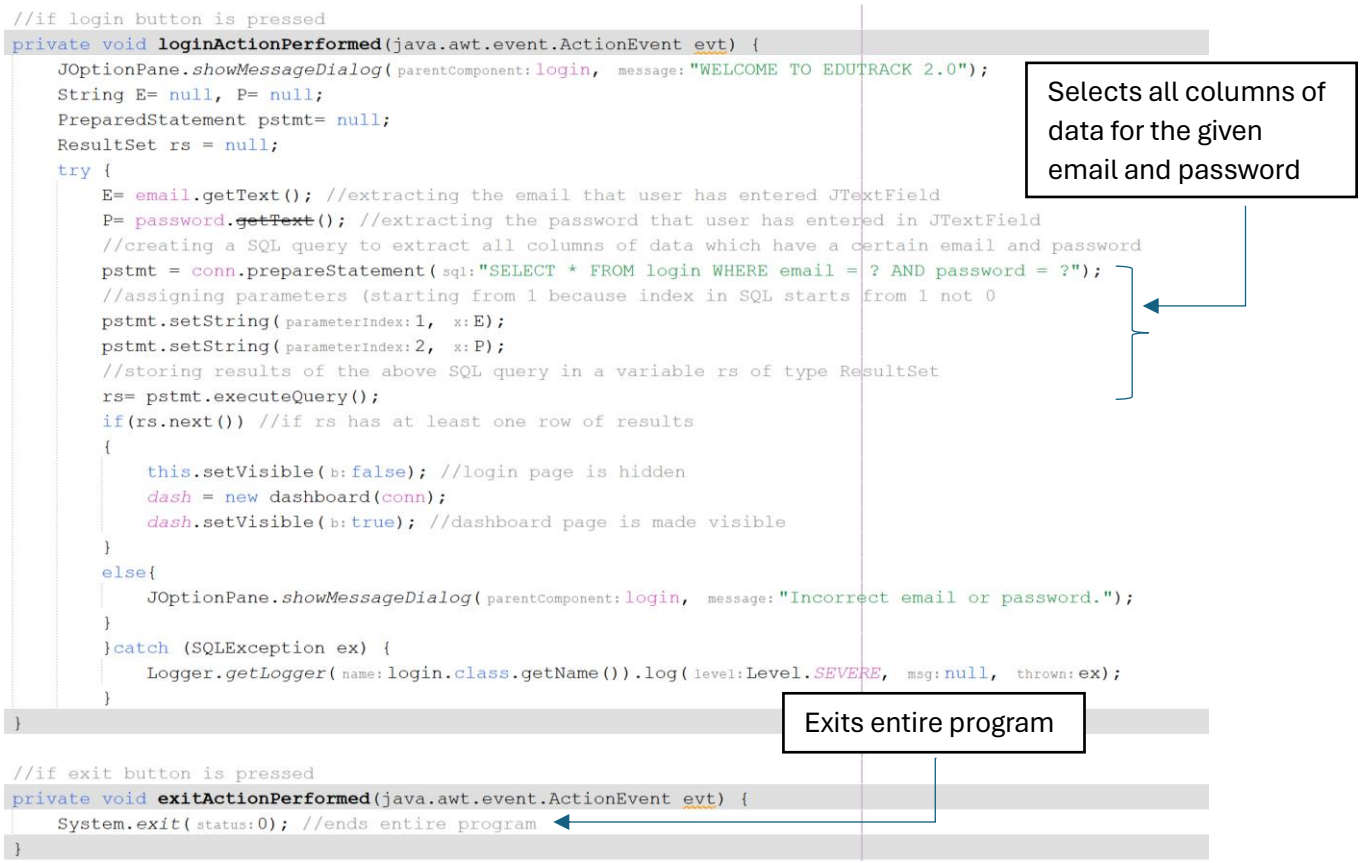
order to use pre-defined methods to communicate with the database, implement GUI and facilitate input and output in the Output terminal.

Login Page



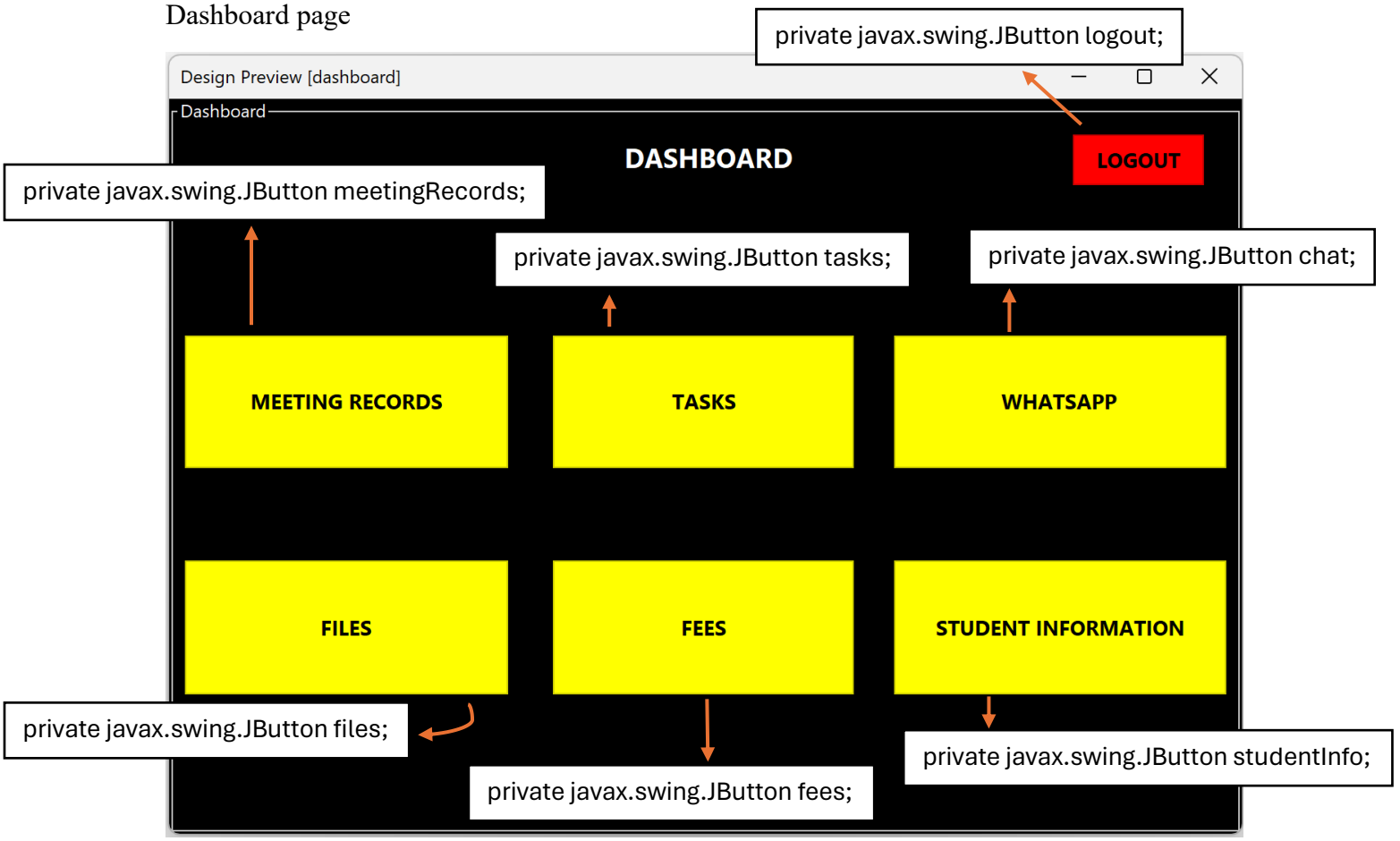

```
public class login extends javax.swing.JFrame {

    Connection conn;
    public static dashboard dash;

    public login(Connection conn) {
        //receiving database connection
        this.conn= conn;
        initComponents();
    }
```

Database connection received from clsDB in login



The snippet above is a JFrame called '*login*'. The snippet above contains methods such as '*loginActionPerformed(java.awt.event.ActionEvent evt)*' and '*exitActionPerformed(java.awt.event.ActionEvent evt)*' as well as '*initComponents()*'. The function '*initComponents()*' is responsible for the GUI design whereas '*exitActionPerformed(java.awt.event.ActionEvent evt)*' is a method of the '*exit*' button which is responsible for exiting and ending the entire program. The function '*loginActionPerformed(java.awt.event.ActionEvent evt)*' is a method for the '*login*' button which is responsible for checking if the username and password which has been entered is correct. If it is correct and present in the database, then it will close the Login page and display the Dashboard page. In the '*loginActionPerformed(java.awt.event.ActionEvent evt)*' method, a PreparedStatement SQL query has been made where it retrieves all columns of data which have the user-entered email '*E*' and user-entered password '*P'*. The results are stored in a ResultSet object '*rs*' and the Dashboard page is opened up.

Dashboard page



The purpose of the Dashboard page is to help the client navigate to the different pages available and access all the features. Any button that is clicked takes the client to another page.

Button which closes dashboard and opens Student Information page.

```
//When Student Information button is pressed,
private void studentInfoActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        this.setVisible( b: false); //dashboard page hidden
        stu = new studentInfo(conn);
        stu.setVisible( b: true); //student information page shown
    } catch (SQLException ex) {
        Logger.getLogger( name: dashboard.class.getName()).log( level: Level.SEVERE, msg: null, thrown: ex);
    }
}
```

Button which closes dashboard and opens Files page.

```
//When Files button is pressed,
private void filesActionPerformed(java.awt.event.ActionEvent evt) {
    this.setVisible( b: false); //dashboard page hidden
    file = new files(conn);
    file.setVisible( b: true); //files page shown
}
```

Button which closes dashboard and opens Meeting Records page.

```
//When Meeting Records button is pressed,
private void meetingRecordsActionPerformed(java.awt.event.ActionEvent evt) {
    this.setVisible( b: false); //dashboard page hidden
    meet = new meetingRecords(conn);
    meet.setVisible( b: true); //meeting records page shown
}
```

```
//When Tasks button is pressed,
private void tasksActionPerformed(java.awt.event.ActionEvent evt) {
    this.setVisible( b: false); //dashboard page hidden
    task = new tasks(conn);
    task.setVisible( b: true);
}
```

hich closes
oard and
asks page.

Button which closes dashboard and opens Fees page.

```
//When Fees button is pressed,
private void feesActionPerformed(java.awt.event.ActionEvent evt) {
    this.setVisible( b: false); //dashboard page hidden
    fee = new fees(conn);
    fee.setVisible( b: true); //fees page shown
}
```

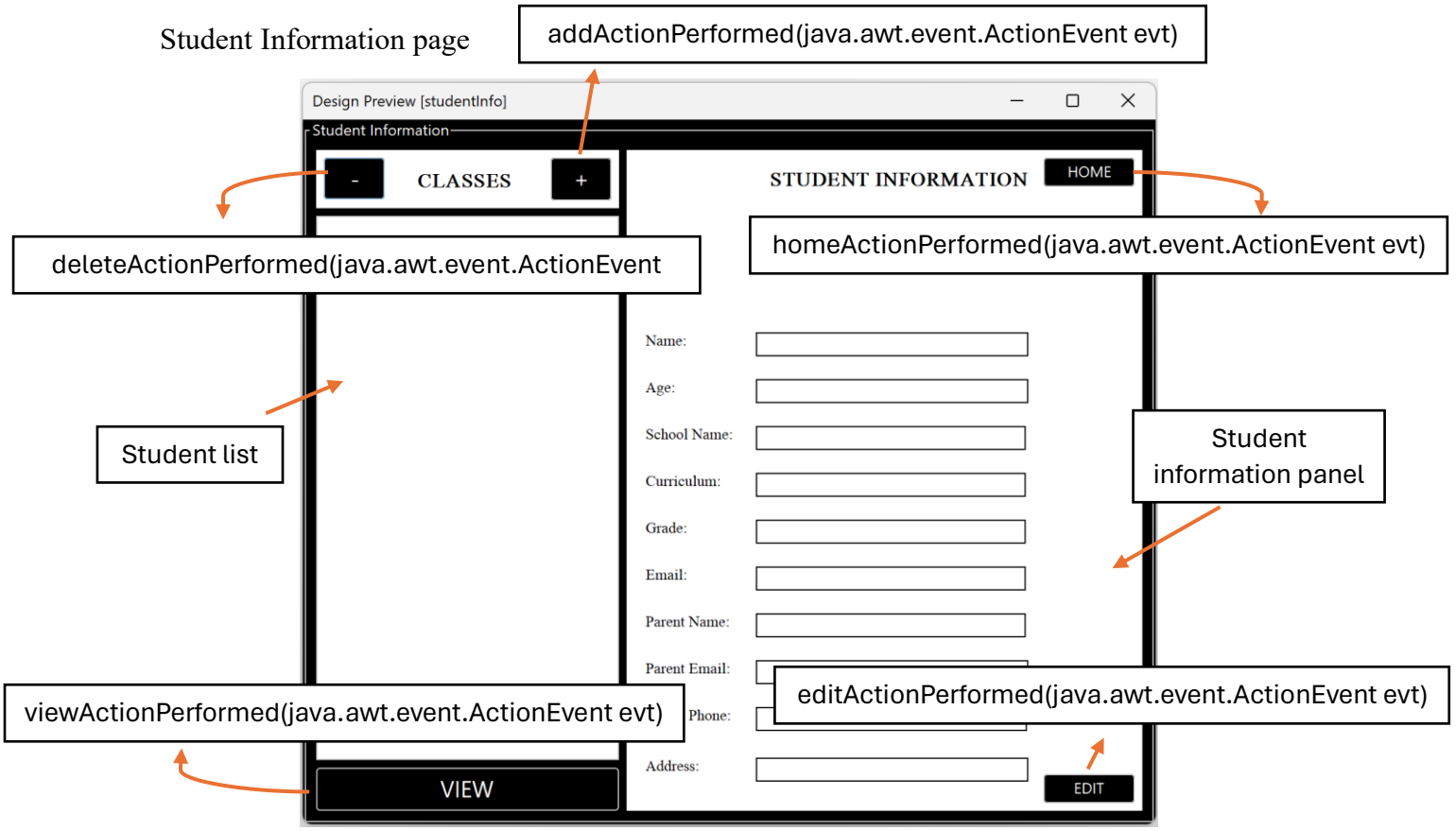
Button which closes dashboard and opens Login page.

```
//When Logout button is pressed,
private void logoutActionPerformed(java.awt.event.ActionEvent evt) {
    this.setVisible( b: false); //dashboard page hidden
    log = new login(conn);
    log.setVisible( b: true); //login page shown
}
```

```
//When WhatsApp button is pressed,
private void chatActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        //WhatsApp Web chat link opened
        Desktop.getDesktop().browse(new URI( str: "https://web.whatsapp.com/"));
    } catch (IOException | URISyntaxException ex) {
    }
}
```

Button which opens WhatsApp Web.

Student Information page



The Student Information page allows the user to view, add and remove student names using the '*view*', '+' and '-' buttons respectively as well as view their details like name, age, grade, school etc. by clicking on their names on the list. The client can press the '*edit*' button to edit the details of the selected student. To navigate back to the Dashboard page, the client can press the '*home*' button.

```
//When the + button is pressed,
private void addActionPerformed(java.awt.event.ActionEvent evt) {
    //enter name of student to be added
    String newStudentName = JOptionPane.showInputDialog( message: "Enter the new student's name:");
    if (newStudentName != null && !newStudentName.isEmpty()) {
        try {
            //add student name to database
            String query = "INSERT INTO studentInfo (sName) VALUES (?)";
            PreparedStatement pstmt = conn.prepareStatement( sql: query);
            pstmt.setString( parameterIndex: 1,  x: newStudentName);
            pstmt.executeUpdate();

            //add student to list
            mod.addElement( element: newStudentName);
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Refresh the list.");

        } catch (SQLException ex) {
            Logger.getLogger( name: studentInfo.class.getName()).log( level: Level.SEVERE,  msg: null,  thrown: ex);
        }
    }
}
```

Code to add student name to studentInfo database and to student list

```
//When the home button is pressed,
private void homeActionPerformed(java.awt.event.ActionEvent evt) {
    this.setVisible( b: false); //student information page closed
    dash= new dashboard(conn);
    dash.setVisible( b: true); //dashboard page shown
}
```

Code to close current page and open dashboard

```
selectedStudentName = studentList.getSelectedValue();
if (selectedStudentName != null && !selectedStudentName.isEmpty()) {
    try {
    //create a SQL query to retrieve the current details of the selected student from the database
    String query = "SELECT sName, age, schoolName, curriculum, grade, "
            + "email, pName, pEmail, pPhone, address FROM studentInfo WHERE sName = ?";
    try (PreparedStatement pstmt = conn.prepareStatement( sql: query)) {
        pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
        try (ResultSet rs = pstmt.executeQuery()) {
            //display the retrieved details on the JPanel called 'informationPanel'
            if (rs.next()) {
                name.setText( text: rs.getString( columnLabel: "sName"));
                age.setText( text: Integer.toString( i: rs.getInt( columnLabel: "age")));
                schoolName.setText( text: rs.getString( columnLabel: "schoolName"));
                curriculum.setText( text: rs.getString( columnLabel: "curriculum"));
                grade.setText( text: rs.getString( columnLabel: "grade"));
                email.setText( text: rs.getString( columnLabel: "email"));
                parentName.setText( text: rs.getString( columnLabel: "pName"));
                parentEmail.setText( text: rs.getString( columnLabel: "pEmail"));
                parentPhone.setText( text: rs.getString( columnLabel: "pPhone"));
                address.setText( text: rs.getString( columnLabel: "address"));
            }
        }
```

Code in studentListValueChanged(javax.swing.event.ListSelectionEvent evt) to display selected student's details

```
//When view button is pressed,
private void viewListActionPerformed(java.awt.event.ActionEvent evt) {
    //studentList is assigned to a list model
    studentList.setModel( model: mod);
    //list model is cleared in case any names are being displayed
    mod.clear();
    try {
        //all student names currently registered are retrieved
        try (Statement stmt = conn.createStatement()) {
            ResultSet rs = stmt.executeQuery( sql: "SELECT sName FROM studentInfo");

            //all student names retrieved in ResultSet rs are added to list model
            while (rs.next()) {
                String studentName = rs.getString( columnLabel: "sName");
                mod.addElement( element: studentName);
            }
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Failed to load student names.",
                title: "Error",  messageType: JOptionPane.ERROR_MESSAGE);
    }
}
```

Code to view student name list

```
        try {
            //create a SQL query to check if the student exists in the database
            String query = "SELECT * FROM studentInfo WHERE sName = ?";
            PreparedStatement pstmt = conn.prepareStatement( sql: query);
            pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                //if student exists, delete the record from the tasks table in database
                query = "DELETE FROM tasks WHERE sName = ?";
                pstmt = conn.prepareStatement( sql: query);
                pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
                pstmt.executeUpdate();

                //delete from files table
                query = "DELETE FROM files WHERE sName = ?";
                pstmt = conn.prepareStatement( sql: query);
                pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
                pstmt.executeUpdate();

                //delete from fees table
                query = "DELETE FROM fees WHERE sName = ?";
                pstmt = conn.prepareStatement( sql: query);
                pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
                pstmt.executeUpdate();

                //delete from meetingRecords table
                query = "DELETE FROM meetingRecords WHERE sName = ?";
                pstmt = conn.prepareStatement( sql: query);
                pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
                pstmt.executeUpdate();

                //delete from studentInfo table
                query = "DELETE FROM studentInfo WHERE sName = ?";
                pstmt = conn.prepareStatement( sql: query);
                pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
                pstmt.executeUpdate();
```

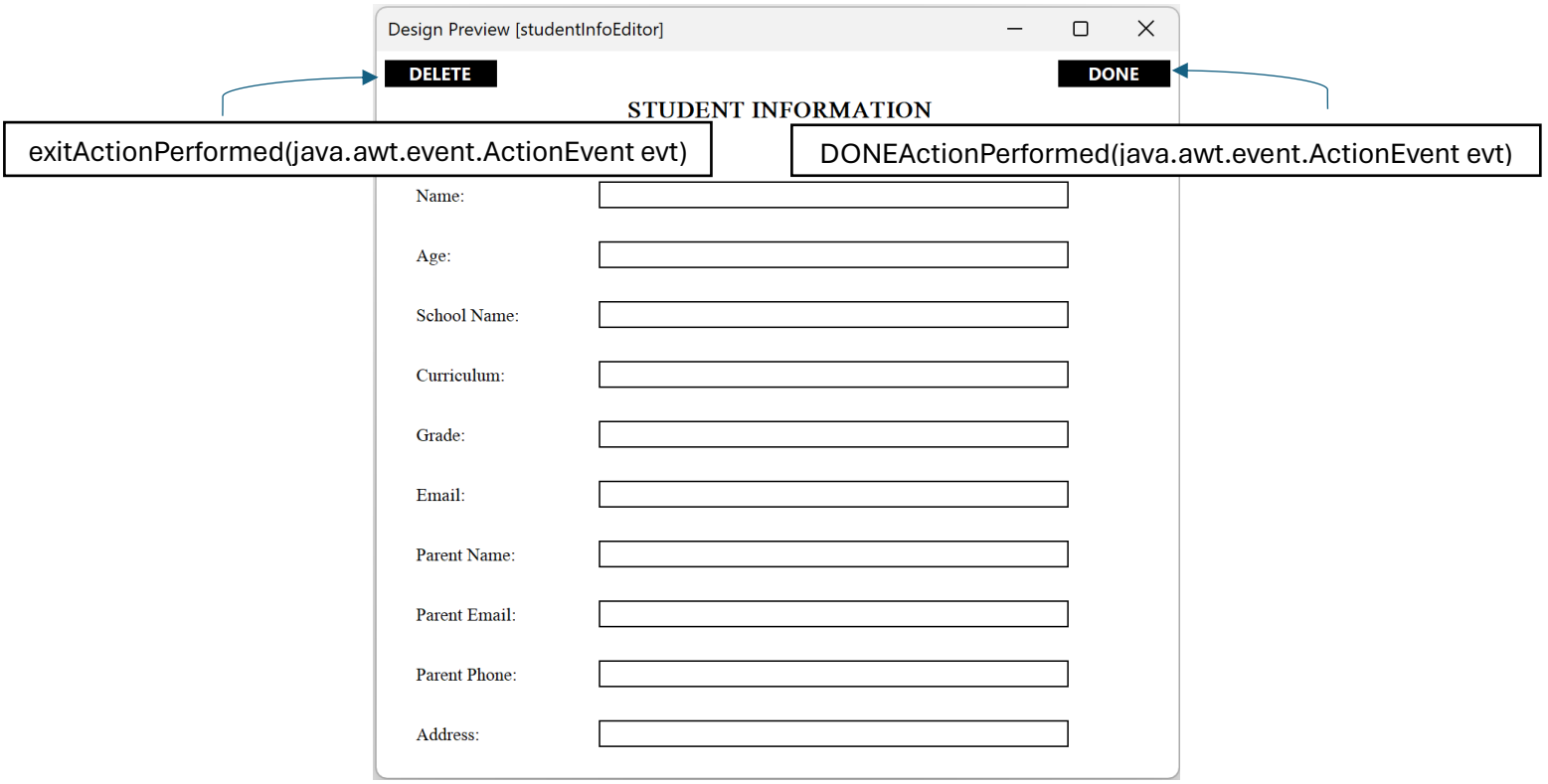
Code in deleteActionPerformed(java.awt.event.ActionEvent evt) to delete all student records in the database for the chosen student

```java
//When the edit button is pressed,
private void editActionPerformed(java.awt.event.ActionEvent evt) {
    //retrieve name of selected student from the student list
    selectedStudentName = studentList.getSelectedValue();
    //open editor for that student
    openStudentInfoEditor( sName: selectedStudentName);
}
```

Code to open editor to edit student information for selected student

Check page 11

When the client presses the '*edit*' button, another window will open up. Hence, opening up another JFrame called studentInfoEditor where the client can edit the selected student's information.




```java
//When the delete button is pressed,
private void exitActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Delete edits?");
        this.setVisible( b: false); //student information editor page closed
        stu= new studentInfo(conn);
        stu.setVisible( b: true); //student information page opened
    } catch (SQLException ex) {
        Logger.getLogger( name: studentInfoEditor.class.getName()).log( level: Level.SEVERE,  msg: null,  thrown: ex);
    }
}
```

Takes user back to Student Information page without saving details

```java
//When the done button is pressed,
private void DONEActionPerformed(java.awt.event.ActionEvent evt) {
    //retrieve all info that has been entered into the JTextFields
    String studentName = name.getText();
    int studentAge = Integer.parseInt( s:age.getText());
    String studentSchoolName = schoolName.getText();
    String studentCurriculum = curriculum.getText();
    String studentGrade = grade.getText();
    String studentEmail = email.getText();
    String parentNameValue = parentName.getText();
    String parentEmailValue = parentEmail.getText();
    String parentPhoneValue = parentPhone.getText();
    String studentAddress = address.getText();
    try {
        //create a SQL query updating the database with the currently entered information
        String query = "UPDATE studentInfo SET sName=?, age=?, schoolName=?, curriculum=?, "
                + "grade=?, email=?, pName=?, pEmail=?, pPhone=?, address=? WHERE sName=?";

        try (
            PreparedStatement pstmt = conn.prepareStatement( sql:query)) {
            pstmt.setString( parameterIndex:1,  x:studentName);
            pstmt.setInt( parameterIndex:2,  x:studentAge);
            pstmt.setString( parameterIndex:3,  x:studentSchoolName);
            pstmt.setString( parameterIndex:4,  x:studentCurriculum);
            pstmt.setString( parameterIndex:5,  x:studentGrade);
            pstmt.setString( parameterIndex:6,  x:studentEmail);
            pstmt.setString( parameterIndex:7,  x:parentNameValue);
            pstmt.setString( parameterIndex:8,  x:parentEmailValue);
            pstmt.setString( parameterIndex:9,  x:parentPhoneValue);
            pstmt.setString( parameterIndex:10,  x:studentAddress);
            pstmt.setString( parameterIndex:11,  x:selectedStudentName);
            pstmt.executeUpdate();
        }
        //updating the sName in the meetingRecords database
        query = "UPDATE meetingRecords SET sName=? WHERE sName=?";
        try (PreparedStatement pstmt = conn.prepareStatement( sql:query)) {
            pstmt.setString( parameterIndex:1,  x:studentName);
            pstmt.setString( parameterIndex:2,  x:selectedStudentName);
            pstmt.executeUpdate(); // Execute the update operation
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog( parentComponent:this,
                    "Error changing student name in meetingRecords database table: " + ex.getMessage());
        }
        //updating the sName in the tasks table
        query = "UPDATE tasks SET sName=? WHERE sName=?";
        try (PreparedStatement pstmt = conn.prepareStatement( sql:query)) {
            pstmt.setString( parameterIndex:1,  x:studentName);
            pstmt.setString( parameterIndex:2,  x:selectedStudentName);
            pstmt.executeUpdate(); // Execute the update operation
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog( parentComponent:this,
                    "Error changing student name in tasks database table: " + ex.getMessage());
        }
        //updating the sName in the files database
        query = "UPDATE files SET sName=? WHERE sName=?";
        try (PreparedStatement pstmt = conn.prepareStatement( sql:query)) {
            pstmt.setString( parameterIndex:1,  x:studentName);
            pstmt.setString( parameterIndex:2,  x:selectedStudentName);
            pstmt.executeUpdate(); // Execute the update operation
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog( parentComponent:this,
                    "Error changing student name in files database table: " + ex.getMessage());
        }

        //updating the sName in the fees database
        query = "UPDATE fees SET sName=? WHERE sName=?";
        try (PreparedStatement pstmt = conn.prepareStatement( sql:query)) {
            pstmt.setString( parameterIndex:1,  x:studentName);
            pstmt.setString( parameterIndex:2,  x:selectedStudentName);
            pstmt.executeUpdate(); // Execute the update operation
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog( parentComponent:this,
                    "Error changing student name in fees database table: " + ex.getMessage());
        }
```

Extracts text entered by user in provided textboxes

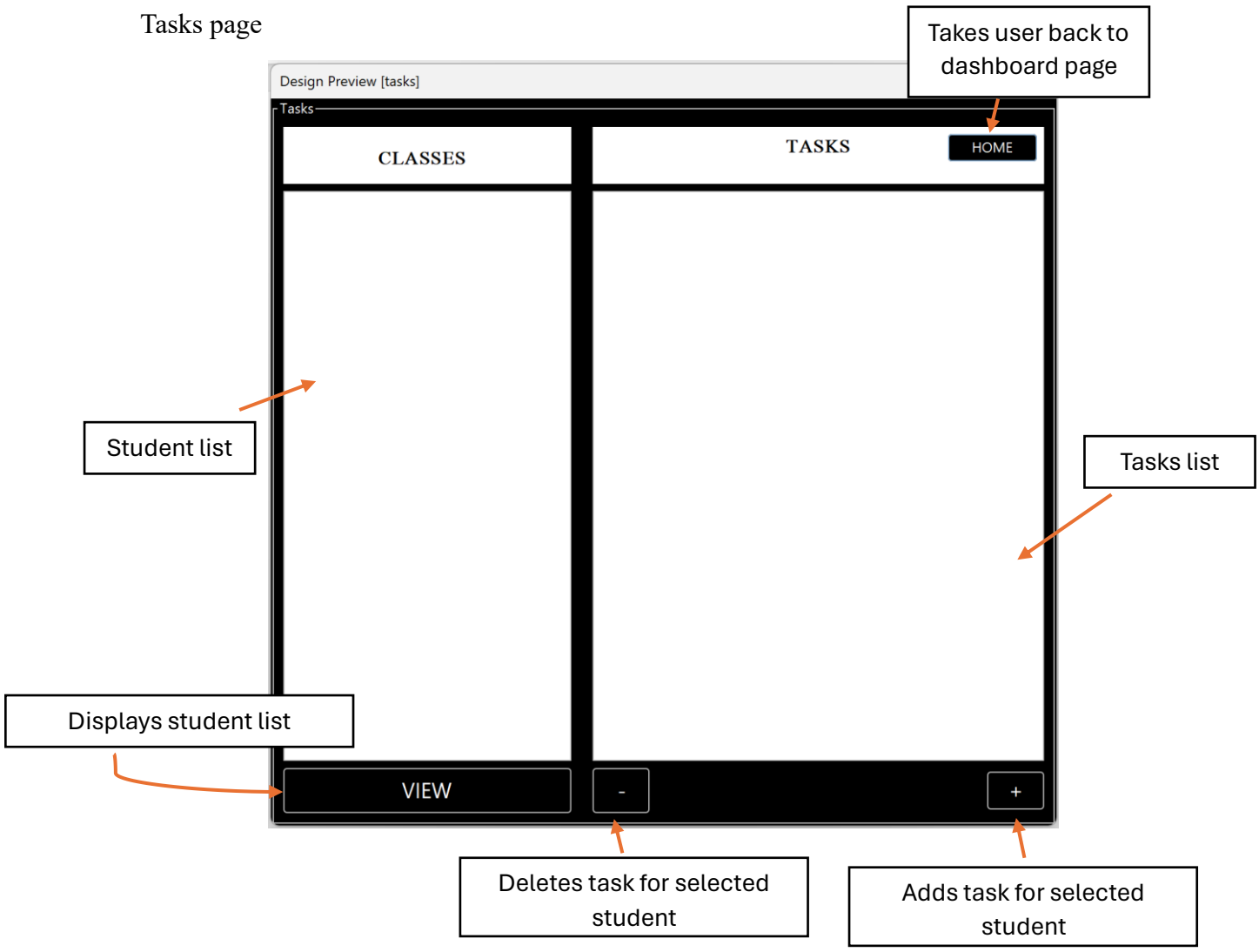
Updates studentInfo database with extracted text

Updates all other database tables with updated name

```
public void setStudentDetails(String studentName, int studentAge, String schoolName,
        String curriculum, String grade, String email, String parentName, String parentEmail,
        String parentPhone, String address)
{
    //set student details into the JTextFields on the student information editor page
    if (studentName != null) {
        this.name.setText( t:studentName);
    } else {
        this.name.setText( t:"");
    }

    if (studentAge!=0) {
        this.age.setText( t:String.valueOf( i:studentAge));
    } else {
        this.age.setText( t:"0");
    }

    if (schoolName != null) {
        this.schoolName.setText( t:schoolName);
    } else {
        this.schoolName.setText( t:"");
    }
   if (parentPhone != null) {
       this.parentPhone.setText( t:parentPhone);
   } else {
       this.parentPhone.setText( t:"");
   }

   if (address != null) {
       this.address.setText( t:address);
   } else {
       this.address.setText( t:"");
   }
}
if (curriculum != null) {
    this.curriculum.setText( t:curriculum);
} else {
    this.curriculum.setText( t:"");
}

if (grade != null) {
    this.grade.setText( t:grade);
} else {
    this.grade.setText( t:"");
}

if (email != null) {
    this.email.setText( t:email);
} else {
    this.email.setText( t:"");
}

if (parentName != null) {
    this.parentName.setText( t:parentName);
} else {
    this.parentName.setText( t:"");
}

if (parentEmail != null) {
    this.parentEmail.setText( t:parentEmail);
} else {
    this.parentEmail.setText( t:"");
}
```

Displaying current student information in the textboxes

Tasks page




```java
//When - button pressed,
private void deleteActionPerformed(java.awt.event.ActionEvent evt) {
    //enter name of the task to be deleted
    selectedTaskName = JOptionPane.showInputDialog( message: "Enter the name of the task:");
    if (selectedTaskName != null && !selectedTaskName.isEmpty()) {
        try {
            //create a SQL query to select all columns of data for the selected student name, specifically for the entered task name
            String query = "SELECT * FROM tasks WHERE sName = ? AND taskName= ?";
            PreparedStatement pstmt = conn.prepareStatement( sql: query);
            pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
            pstmt.setString( parameterIndex: 2,  x: selectedTaskName);
            ResultSet rs = pstmt.executeQuery();
            if(rs.next()){
                //deleting the entire record of the entered task name for the selected student
                query = "DELETE FROM tasks WHERE taskName= ? AND sName= ?";
                pstmt = conn.prepareStatement( sql: query);
                pstmt.setString( parameterIndex: 1,  x: selectedTaskName);
                pstmt.setString( parameterIndex: 2,  x: selectedStudentName);
                pstmt.executeUpdate();

                //remove task name from the list model called 'tasMod' for the selected student
                JOptionPane.showMessageDialog( parentComponent: this, "Task '" + selectedTaskName + "' deleted successfully.");
                tasMod.removeElement( obj: selectedTaskName);
            }
            else {
            JOptionPane.showMessageDialog( parentComponent: this, "No task found with the name '" + selectedTaskName + "'.");
        }
        } catch (SQLException ex) {
            Logger.getLogger( name: studentInfo.class.getName()).log( level: Level.SEVERE,  msg: null,  thrown: ex);
        }
    }
}
```

Code for deleting task for selected student

```java
//When the + button is pressed,
private void addActionPerformed(java.awt.event.ActionEvent evt) {
    //enter name of task to be added
    String newTaskName = JOptionPane.showInputDialog( message: "Enter the name of new task:");
    if (newTaskName != null && !newTaskName.isEmpty()) {
        try {
            //create a SQL query to create a task record for the selected student with the entered task name
            String query = "INSERT INTO tasks (sName, taskName) VALUES (?, ?)";
            PreparedStatement pstmt = conn.prepareStatement( sql: query);
                    pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
                    pstmt.setString( parameterIndex: 2,  x: newTaskName);
                    pstmt.executeUpdate();

                    //add task name in the list model called 'tasMod' for the selected student
                    tasMod.addElement( element: newTaskName);
        } catch (SQLException ex) {
            Logger.getLogger( name: studentInfo.class.getName()).log( level: Level.SEVERE,  msg: null,  thrown: ex);
        }
    }
}
```

Code for adding task for selected student

```java
//When a task name is selected from the task list of the selected student
private void taskListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {
        //retrieve name of selected task from taskList
        selectedTaskName = taskList.getSelectedValue();
        //open task information editor page for the selected task
        openTaskEditor( taskName: selectedTaskName);
    }
}
```

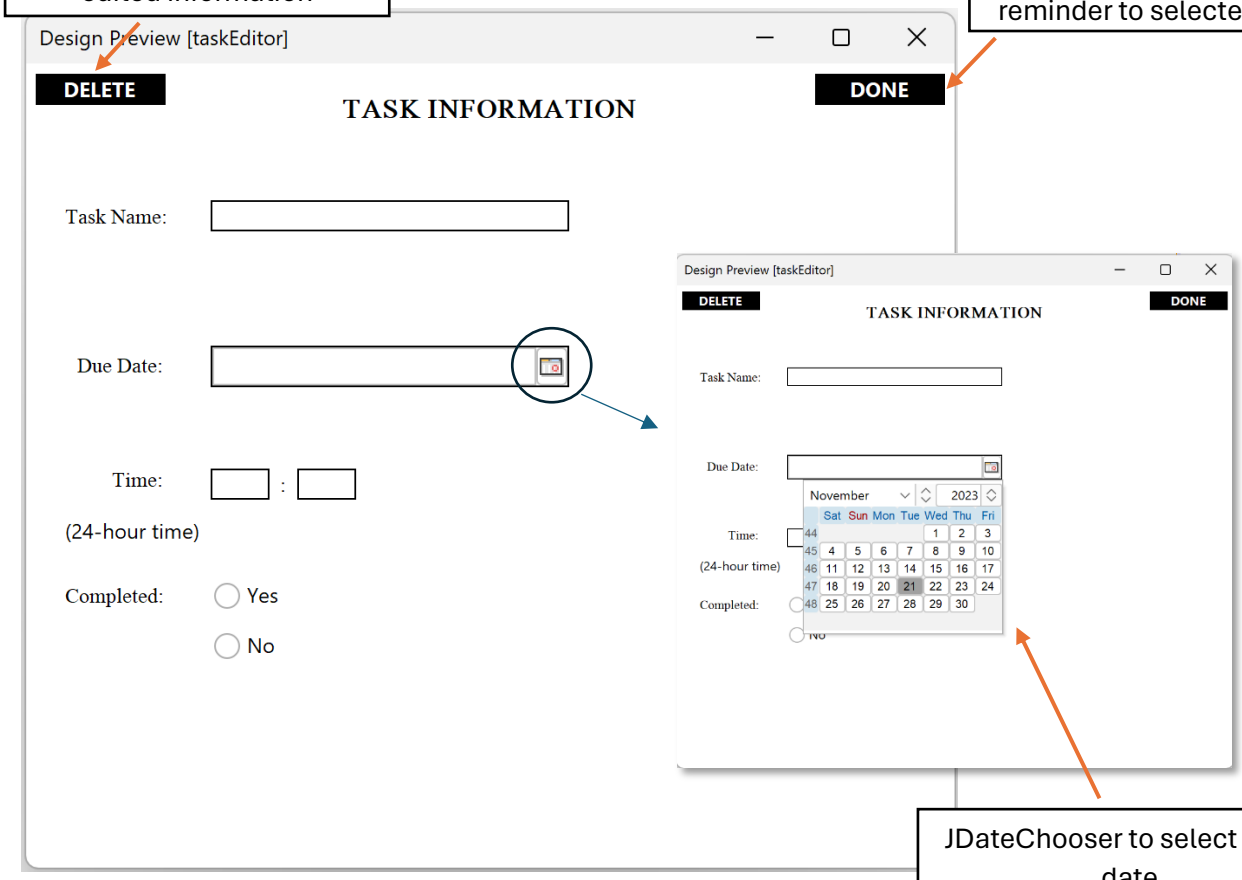
Code for viewing tasks for selected student

The Tasks page is responsible for displaying and manipulating the list of tasks for every student from the '*studentInfo*' database. The '*studentList*' contains the student names from the '*studentInfo*' database and the '*taskList*' contains the list of tasks for the selected student from the '*tasks*' database. The client will be able to view the list of all the students by pressing the '*view*' button from which the client can click on a student's name, showcasing the list of all the tasks for the selected student. Client can add and delete tasks as well as manipulate task details by click on a task name from the task list. Once a task name is clicked, the Task Editor page will open up for the selected task.

The Task Editor page allows the user to enter and manipulate details of the selected task and sending an email to the selected student to inform and remind them about the task and due date of the task. The '*delete*' button can be used to exit the editor page and go back to the '*Tasks*' page without saving the details in the database and the '*done*' button is used to save the details in the database and then exit the editor page to go back to the '*Tasks*' page.

Button to go back to the Tasks page without saving edited information

Button to go back to the Tasks page after saving edited details and sending a reminder to selected student



JDateChooser to select a due date

```
//When the done button is pressed,
private void DONEActionPerformed(java.awt.event.ActionEvent evt) {
    //retrieve entered task name, due hour and due min
    tasName = taskName.getText();
    dHour = dueHour.getText();
    dMin = dueMin.getText();

    //format date according to database
    SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "d MMM y");

    try {
        //retrieve entered date as a Date object
        dDate = dueDate.getDate();

        //convert dDate to a String
        String dDateString= dateFormat.format( date: dDate);
        //convert hour and min to integer type
        int hour = dHour.isEmpty() ? 0 : Integer.parseInt( s: dHour);
        int min = dMin.isEmpty() ? 0 : Integer.parseInt( s: dMin);

        //store yes or no in terms of numbers (easily stored in and retrieved from database)
        if(yes.isSelected()){
            no.setSelected( b: false);
            completedValue= 1;
        } else if (no.isSelected()){
            yes.setSelected( b: false);
            completedValue= 2;
        }
        else {
            completedValue= 0;
        }
```

Extracting and formatting data for storing in database

```
        //create query to update database with user-entered information
        String query = "UPDATE tasks SET taskName=?, dueDate=?, completed=?, dueHour=?, "
                + "dueMin=? WHERE sName=? AND taskName= ?";

            try {
                PreparedStatement pstmt = conn.prepareStatement( sql:query)) {
                pstmt.setString( parameterIndex: 1,  x: tasName);
                pstmt.setString( parameterIndex: 2,  x: dDateString);
                pstmt.setInt( parameterIndex: 3,  x: completedValue);
                pstmt.setInt( parameterIndex: 4,  x: hour);
                pstmt.setInt( parameterIndex: 5,  x: min);
                pstmt.setString( parameterIndex: 6,  x: selectedStudentName);
                pstmt.setString( parameterIndex: 7,  x: selectedTaskName);
                pstmt.executeUpdate();

                JOptionPane.showMessageDialog( parentComponent: this,  message: "Details have been updated");
            }

            //retrieve selected student's email from database
            try (PreparedStatement pstmt = conn.prepareStatement( sql: "SELECT email FROM studentInfo WHERE sName= ?")) {
                    pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
                    ResultSet rs = pstmt.executeQuery();
                    if (rs.next()) {
                        String studentEmail = rs.getString( columnLabel: "email");
                        if(studentEmail!=null) {
                            //sending an email reminder about the task to the student to whom the task has been assigned
                            sendReminderEmail( student: selectedStudentName,  sEmail: studentEmail,  taskName: tasName,
                                    date: dDateString, hour, min);
                        }
                        else {
                            JOptionPane.showMessageDialog( parentComponent: this,"Email not given."
                                    + " Enter email in Student Information page.");
                        }
                    }
                } catch (SQLException ex) {
                    JOptionPane.showMessageDialog( parentComponent: this, "Error fetching student email: " +ex.getMessage());
                }
```

Updating database and sending reminder email

```
public static void sendReminderEmail(String student, String sEmail, String taskName, String date, int hour, int min)
    //set sender email and password
    String senderEmail = "hiral.h0305@gmail.com";
    //using app password for more security and to avoid 2-factor authentication
    String senderPassword = "cqhv inpj gvmf glxb";

    //assigning mail server properties to a Properties object
    Properties props = new Properties();

    //command to request authentication to connect to SMTP (Simple Mail Transfer Protocol) server
    props.put( key: "mail.smtp.auth",  value: "true");

    //enabling STARTTLS command to use a TLS-protected connection to increase security and encrypt data
    props.put( key: "mail.smtp.starttls.enable",  value: "true");

    //specifying that SMTP server for sending emails through Gmail
    props.put( key: "mail.smtp.host",  value: "smtp.gmail.com");

    //setting port for sending mails with STARTTLS
    props.put( key: "mail.smtp.port",  value: "587");

    //creating a mail Session object to build and send mails
    Session session = Session.getInstance(props, new javax.mail.Authenticator() {
    @Override
    protected PasswordAuthentication getPasswordAuthentication() {
        //for Session object, to authenticate sender with SMTP server
        return new PasswordAuthentication( userName: senderEmail,  password: senderPassword);
    }
});
```

Setting email properties and creating email session

Authenticating sender's account

```
try {
    //create a mime message object with Session session
    Message message = new MimeMessage(session);

    //set sender's email address to the From field
    message.setFrom(new InternetAddress( address: senderEmail));

    //set student's email address to the To field
    message.setRecipients( rt: Message.RecipientType.TO,  adrss: InternetAddress.parse( addresslist: sEmail));

    //set subject of email
    message.setSubject("Reminder: Task Assigned - " + taskName);

    //set body of email
    message.setText("This is a reminder that you have been assigned the task " + taskName.toUpperCase()
            + " which is due on " +date +" at " +hour +":" +min +".");

    //sends email
    Transport.send( msg: message);

} catch (MessagingException e) {
    e.printStackTrace();
}
}
```

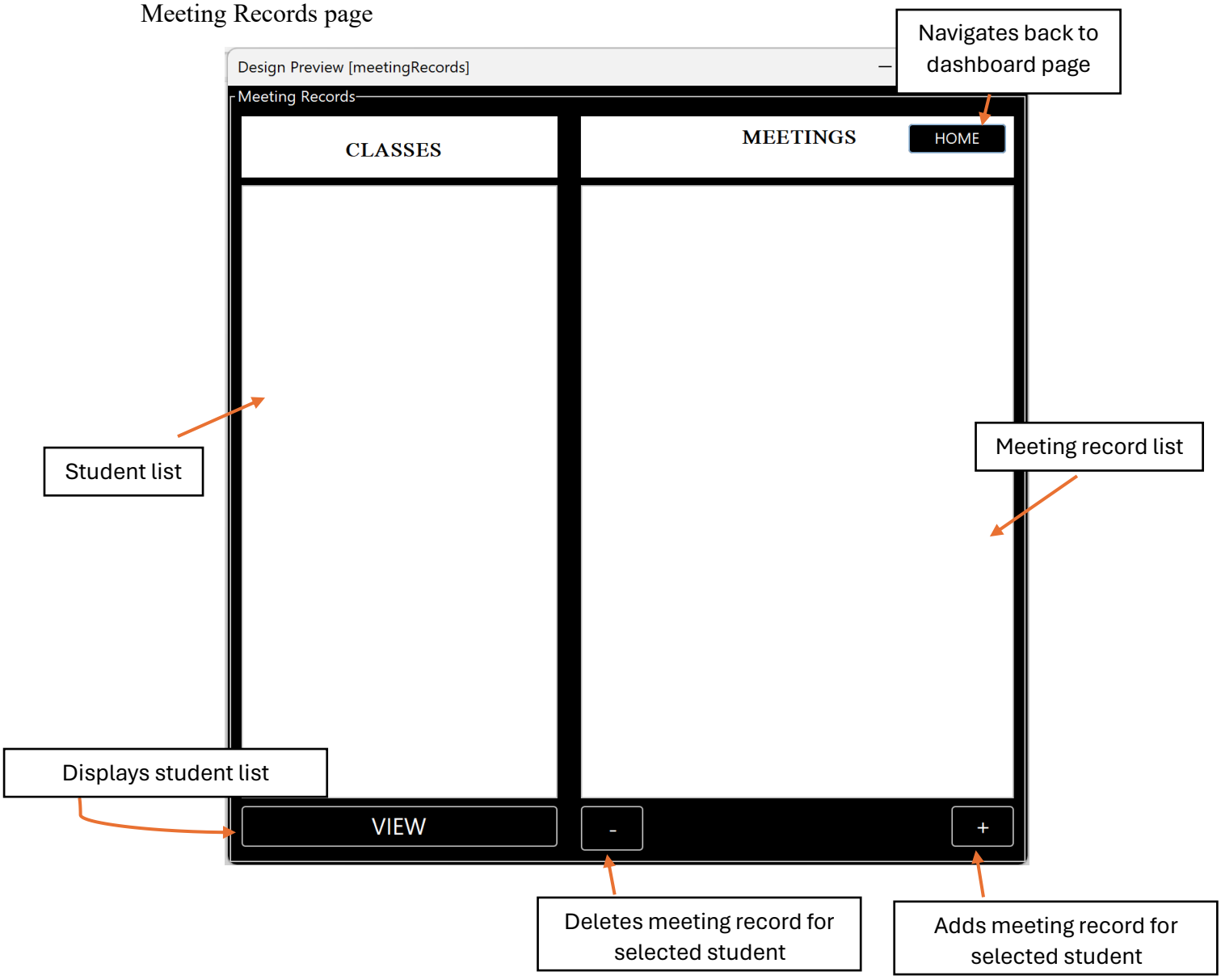
Setting the From and To field of the email

Setting the subject of the email

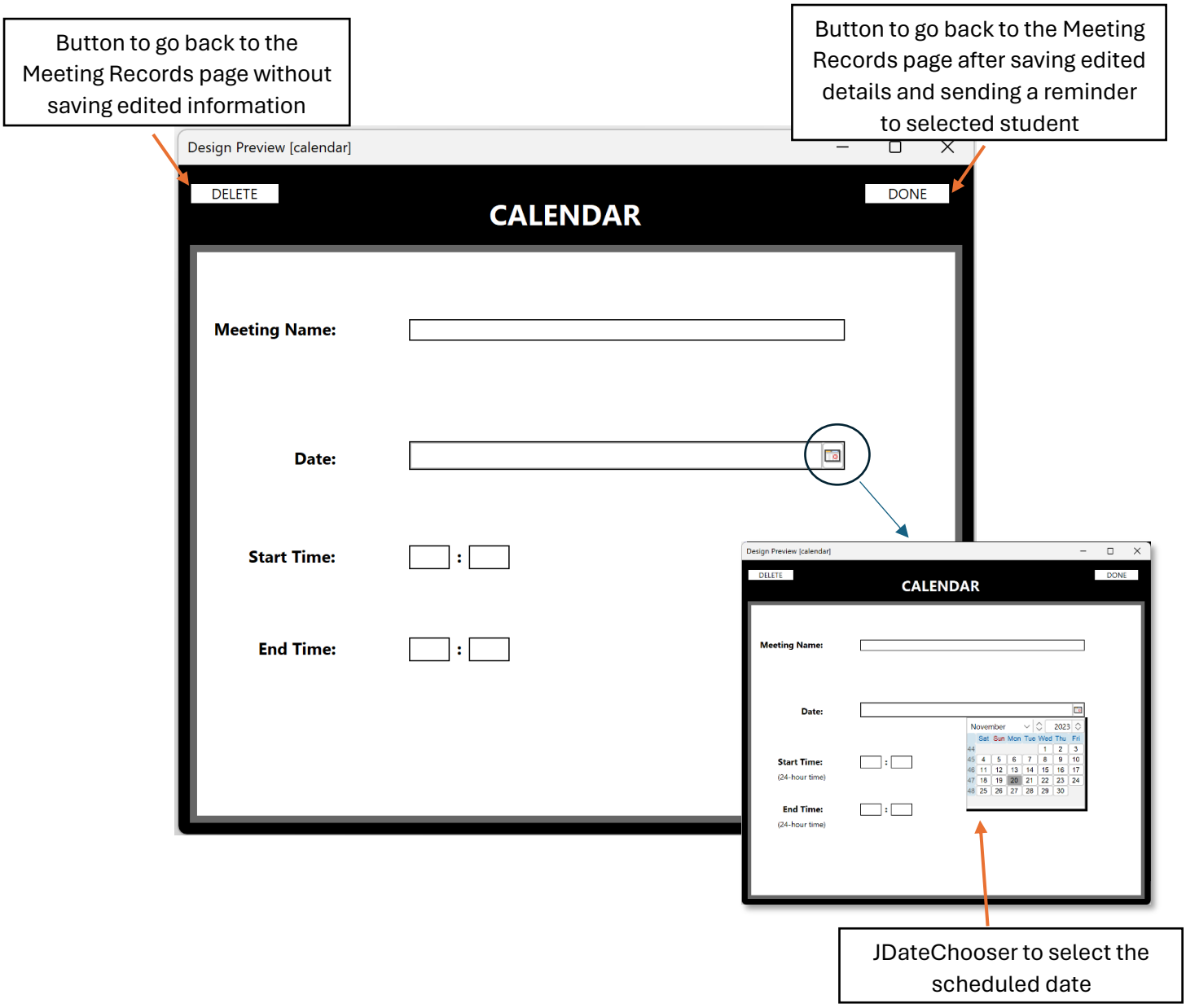
Setting the body of the email

Sending email

Meeting Records page

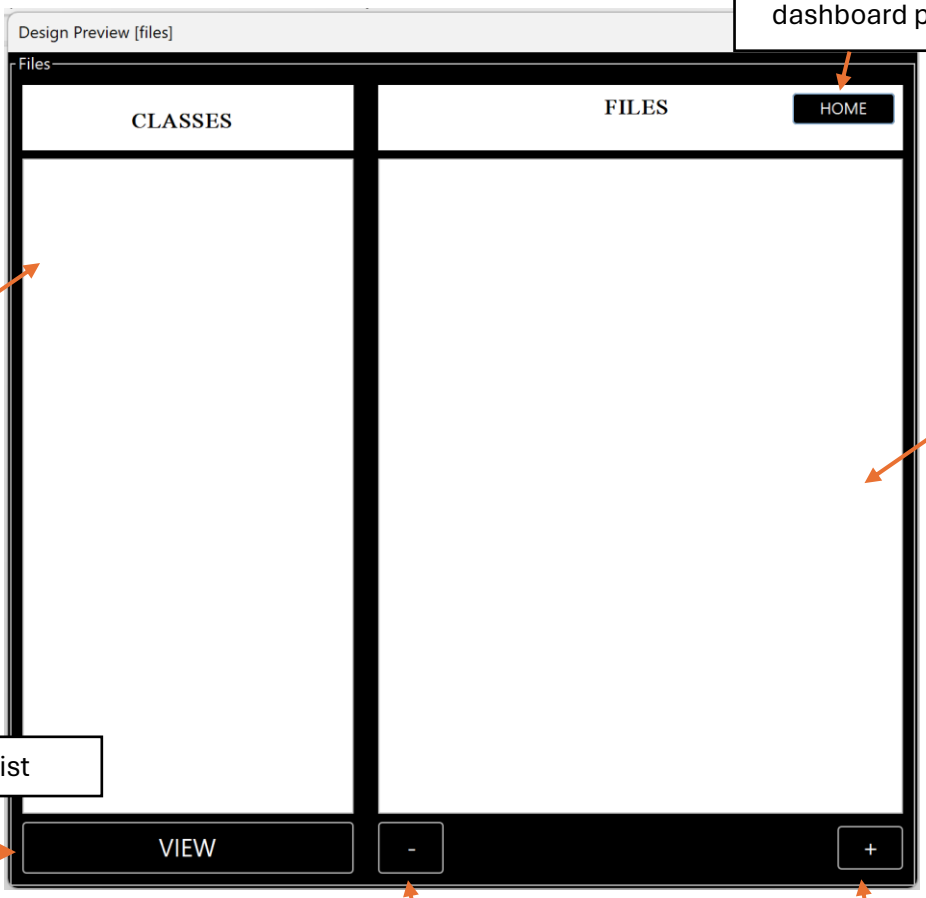


The image above is the Meeting Records page, which is responsible for viewing, adding, and deleting meeting records for each and every student registered by the client. Similar to the Tasks page, when the '*home*' button is pressed, the meeting records page is closed, and the dashboard is shown, when the '*view*' button is pressed, all names of currently registered students are displayed on the student list, when a student's name is clicked, all the meeting records for the selected student are shown and when a meeting name from the meeting list is clicked, the meeting editor page called '*calendar*' is opened for the selected meeting record.



The above image is the Calendar page. It acts as a scheduler and allows the user to enter details for the meeting record that was selected on the meeting list from the Meeting Records page. Client can enter meeting name, date, start time and end time and once the '*done*' button is pressed, these details will be saved in the database in its respective record and an email will be sent to the student to remind them of the scheduled meeting. If the '*delete*' button is pressed, then any changes made will not be saved in the database, no email will be sent, and the Meeting Records page will open up again.

Files page



Navigates back to dashboard page

Student list

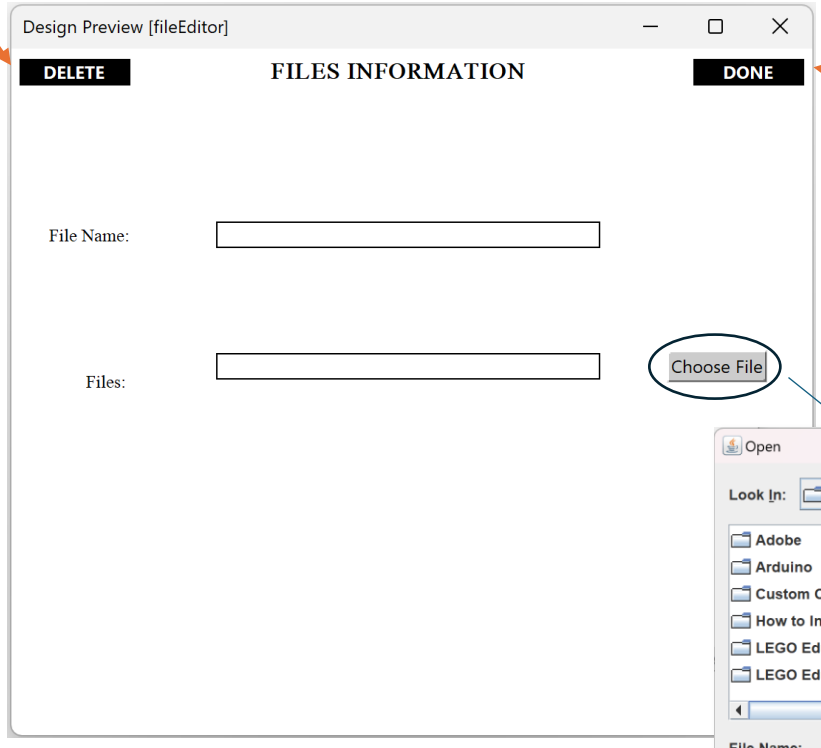
Files list

Displays student list

Button to go back to the Files page without saving edited information

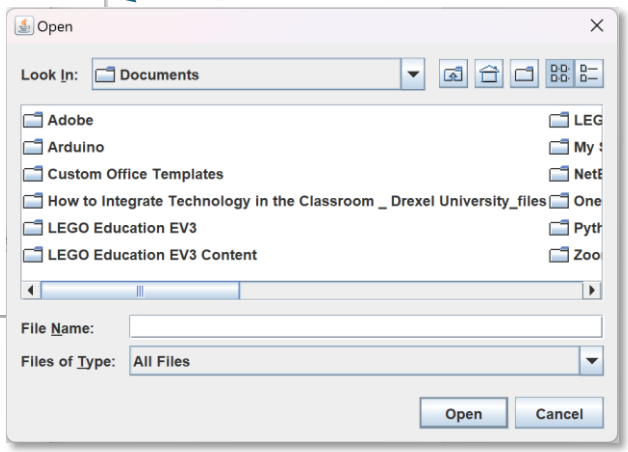
Deletes file for selected student

Adds file for selected student



Button to go back to the Files page after saving edited details for selected student

JFileChooser used to allow user to upload files

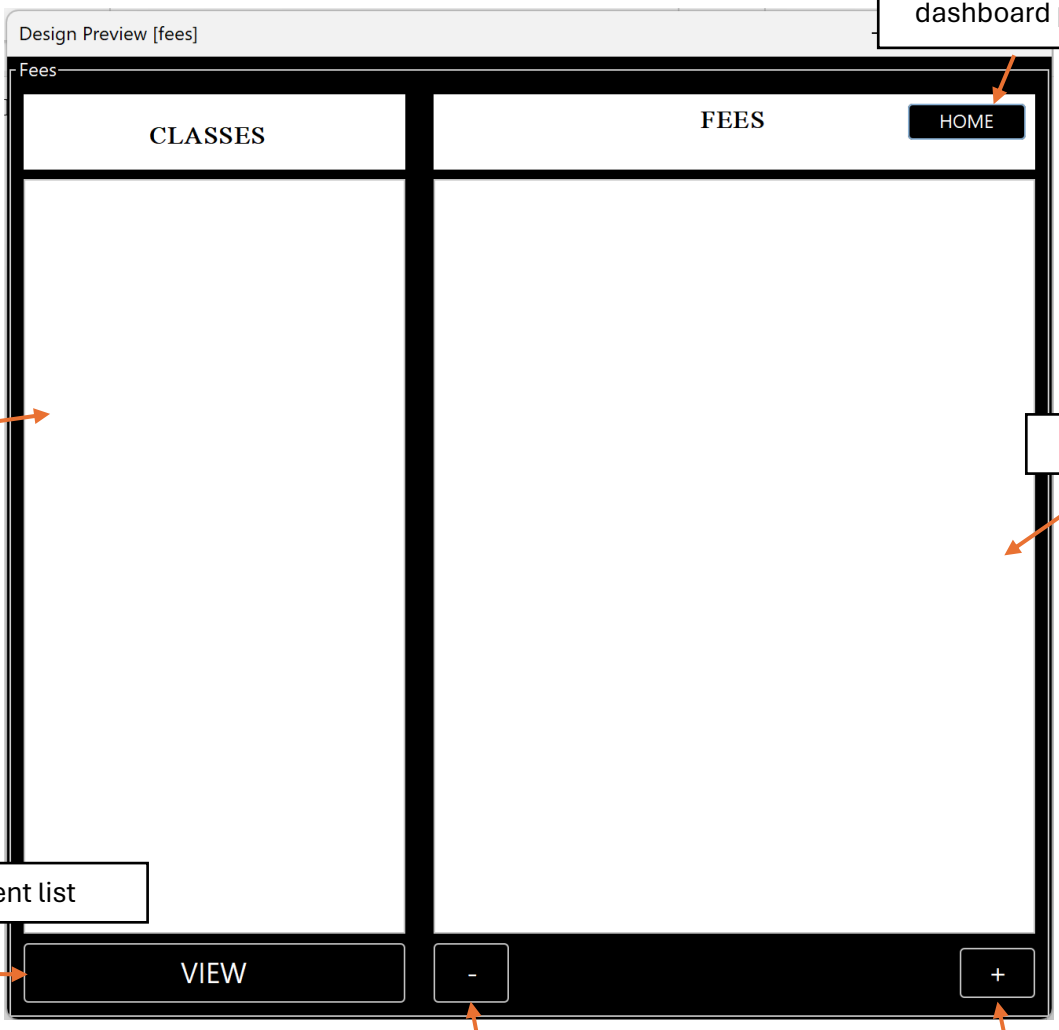


```java
//When choose file button is pressed,
private void chooseButtonActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser chooseFile= new JFileChooser();

    //display the file chooser dialog to the user in the centre of the screen
    chooseFile.showOpenDialog( parent: null);

    //returning a File object 'f' representing the selected file.
    File f= chooseFile.getSelectedFile();

    //retrieve the absolute path of the selected file.
    filePath= f.getAbsolutePath();
    fileSpace.setText( t: filePath);
}
```

Code for implementing the JFileChooser

filePath is a public static String variable

Fees Page



Navigates back to dashboard page

Student list

Fee list

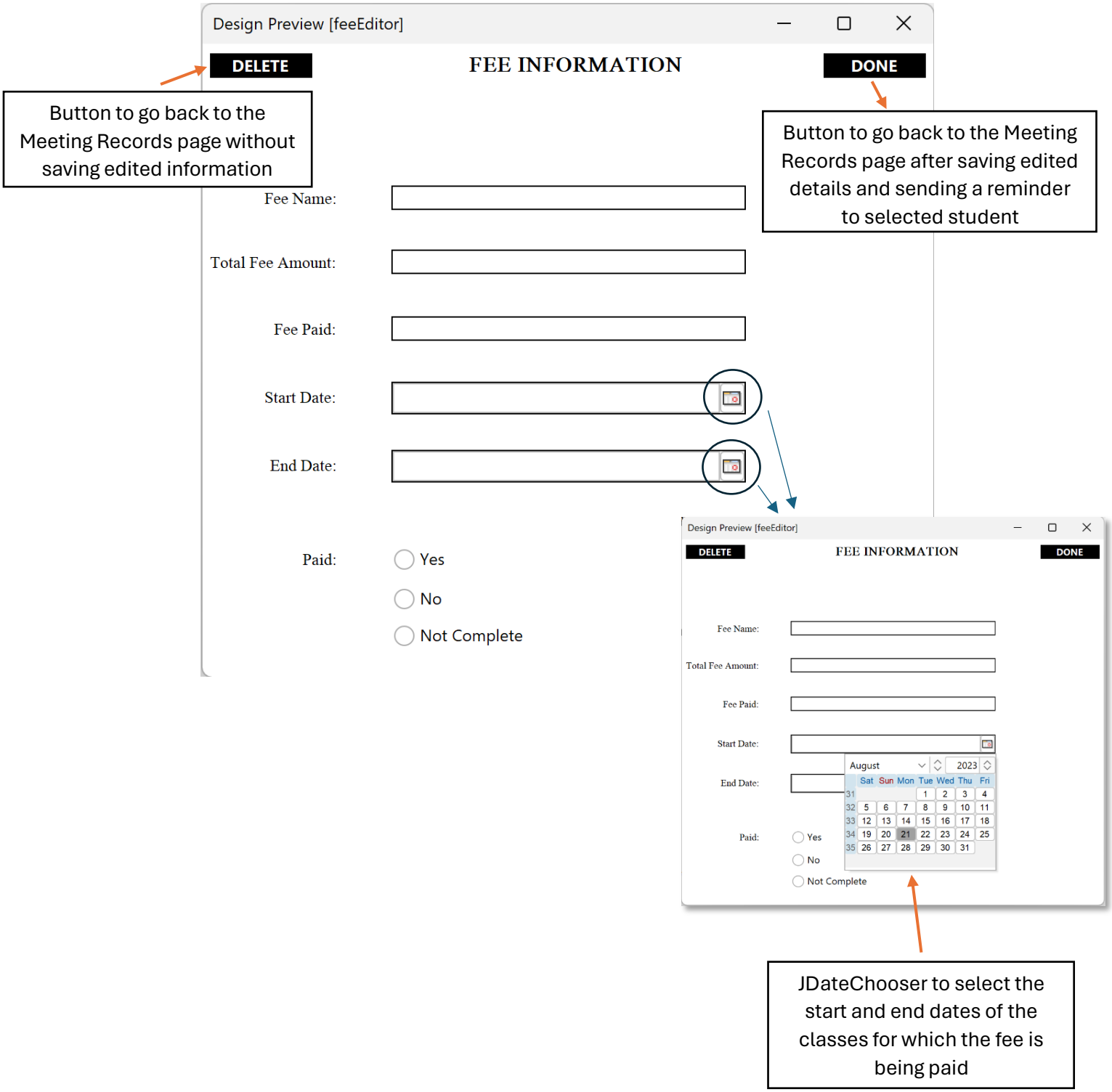
Displays student list

Deletes fee record for selected student

Adds fee record for selected student

The Fees page above allows the client to manage (view, add, delete) fee records for each student using the '*view*', '-', and '+' buttons. When the view button is pressed, all student names are displayed on the student list. When a student name is pressed, all the fee records for that student will be displayed on the fee list and when a fee record is clicked on, the fee editor for that fee record is opened up.

The Fee Editor page allows the client to edit details such as fee name, total fee amount, actual amount paid, start date of the classes, end date of the classes and whether the full fee amount has been paid.



```
//retrieve selected student's email
try (PreparedStatement pstmt = conn.prepareStatement( sql: "SELECT email FROM studentInfo WHERE sName= ?")) {
        pstmt.setString( parameterIndex: 1,  x: selectedStudentName);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            String studentEmail = rs.getString( columnLabel: "email");
            if(studentEmail!=null) {
            //send email to student regarding pending or completed fee
            sendReminderEmail( student: selectedStudentName,  sEmail: studentEmail,  feeName: fName,
                    sDate: sDateString,  eDate: eDateString);
            }
```

Line of code in the DONEActionPerformed(java.awt.event.ActionEvent evt) method which stores/updates the entered details and calls another method to send an email regarding the Fee details

```
        try {
            //create a mime message object with Session session
            Message message = new MimeMessage(session);

            //set sender's email address to the From field
            message.setFrom(new InternetAddress( address: senderEmail));

            //set student's email address to the To field
            message.setRecipients( rt: Message.RecipientType.TO,  adrss: InternetAddress.parse( addresslist: sEmail));

            if(fPaid<tAmount) {
                //set subject of email
                message.setSubject("Reminder: Pending Fee Details - " + feeName);
                //set body of email
                message.setText("This is a reminder that you have a fee '" + feeName.toUpperCase() + "' of amount "
                        +(tAmount-fPaid) +" pending for classes from " +sDate +" to " +eDate +".");
            }
        else {
             //set subject of email
             message.setSubject("Reminder: Fee Receipt - " + feeName);
             //set body of email
             message.setText("You have paid the fee '" + feeName.toUpperCase() + "' of amount " +tAmount
                     +" for classes from " +sDate +" from " +eDate +".");
        }

        //sends email
        Transport.send( msg: message);

    } catch (MessagingException e) {
        e.printStackTrace();
    }
}
```

Code in the '*sendReminderEmail*' method which sets the appropriate subject and body depending on the difference between the total fee amount and the amount actually paid

WhatsApp feature

```
//When WhatsApp button is pressed,
private void chatActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        //WhatsApp Web chat link opened
        Desktop.getDesktop().browse(new URI( str: "https://web.whatsapp.com/"));
    } catch (IOException | URISyntaxException ex) {
    }
}
```

Code from the dashboard JFrame which links the 'WHATSAPP' button to the WhatsApp Web.

Works Cited

1. "1. Datatypes in SQLite." *Datatypes In SQLite*, www.sqlite.org/datatype3.html#:~:text=2.1.-,Boolean%20Datatype,)%20and%201%20(true). Accessed 2023.

2. "How to Install Add Jdatechooser, Jcalendar Date Picker in Netbeans Ide Swing." *YouTube*, YouTube, 23 Nov. 2019, www.youtube.com/watch?v=4c3WYK5FpqI.

3. "Java Joptionpane - Javatpoint." *Www.Javatpoint.Com*, www.javatpoint.com/java-joptionpane. Accessed 2023.

4. "Part 9 | Getting and Setting Text in Text Fields and Labels | JTEXTFIELD Jlabel | Java Gui Tutorial." *YouTube*, YouTube, 24 Nov. 2019, www.youtube.com/watch?v=dcwRqrk0beM.

5. "Send Email Using Java Program." *GeeksforGeeks*, GeeksforGeeks, 14 Sept. 2023, www.geeksforgeeks.org/send-email-using-java-program/.