



CT028-3-2-OODJ
GROUP ASSIGNMENT
OBJECT-ORIENTED DEVELOPMENT WITH JAVA
APU2F2402CS(AI)/APD2F2402CS(AI)

Coursework Title : Project Management System
Lecturer Name : Mr. Muhammad Huzaifah Bin Ismail
Hand in Date : 31/05/2024
Handout Date : 26/03/2024





TP Number	Student Name	Signature
TP073243	Terence Lim Dao Liang	
TP074666	Tay Jun Long	
TP072929	Angelina Leanore	
TP068888	Eraliev Suimonkul	

Table of Content

1.0 Introduction.....	3
2.0 Use Case Diagram.....	4
2.1 Admin.....	4
2.2 Project Manager.....	5
2.3 Lecturer	5
2.4 Student	6
3.0 Class Diagram	7
4.0 Sample Output	8
4.1 Admin	8
4.2 Project Manager.....	19
4.3 Lecturer	22
4.4 Student	26
5.0 Description and Justification of Object-Oriented Concepts	31
5.1 Admin	31
5.2 Project Manager.....	34
5.3 Lecturer	37
5.4 Student	41
6.0 Extra Feature.....	45
7.0 Limitation	49
7.1 Conclusion	50
9.0 Appendix.....	52

1.0 Introduction

A complete Project Management System (PMS) is needed at an academic guidance centre in order to enhance the manual methods of student registration and project management. In addition to accommodating the various features of four kinds of positions-administrator, project manager, lecturer, and student-this system should streamline every facet of student registration, from project assignment to assessment. The Administrator is in charge of lecturer's and student's registration, as well as any required amending or deletion of their personal information.

Additionally, they might give project managers duties and take roles away from users. Project managers are responsible for planning and allocating assignment, appointing secondary markers for every assessment. Lecturers have the power to organize presentations and grade student report in their capacity as supervisors and second markers. It is the responsibility of the students to turn in their project work, look up their grades and ask for a presenting date. (Shaik, 2023)

2.0 Use Case Diagram



Figure 2.0: Use case diagram

2.1 Admin

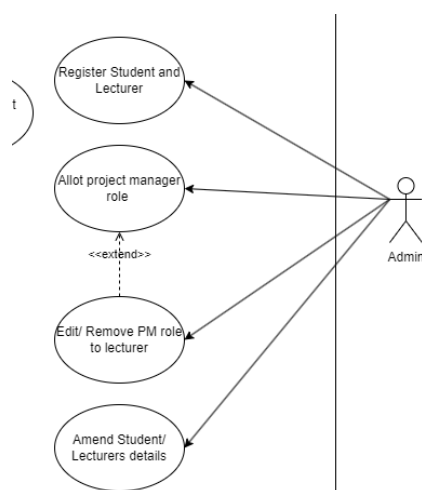


Figure 2.1: Admin use case diagram

2.2 Project Manager

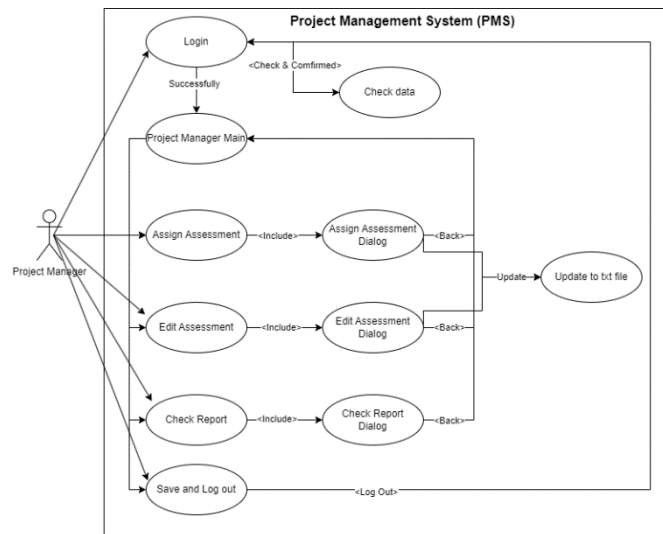


Figure 2.2: Use case diagram for project manager

2.3 Lecturer

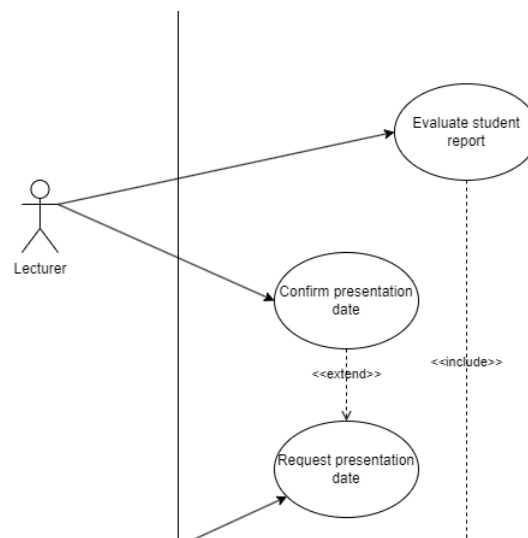


Figure 2.3: Use case diagram for lecturer

2.4 Student

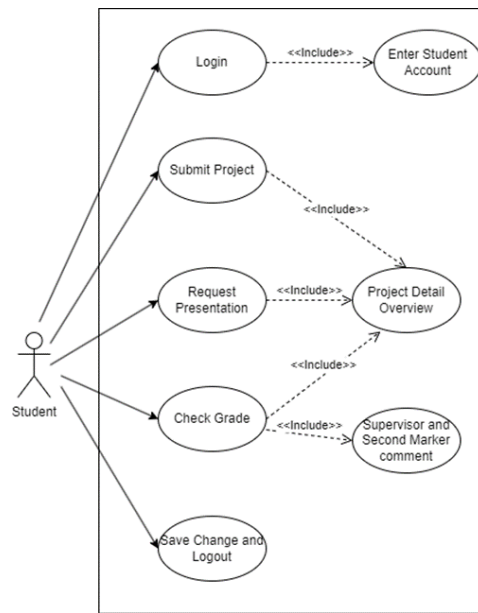


Figure 2.4: Use case diagram for student

3.0 Class Diagram

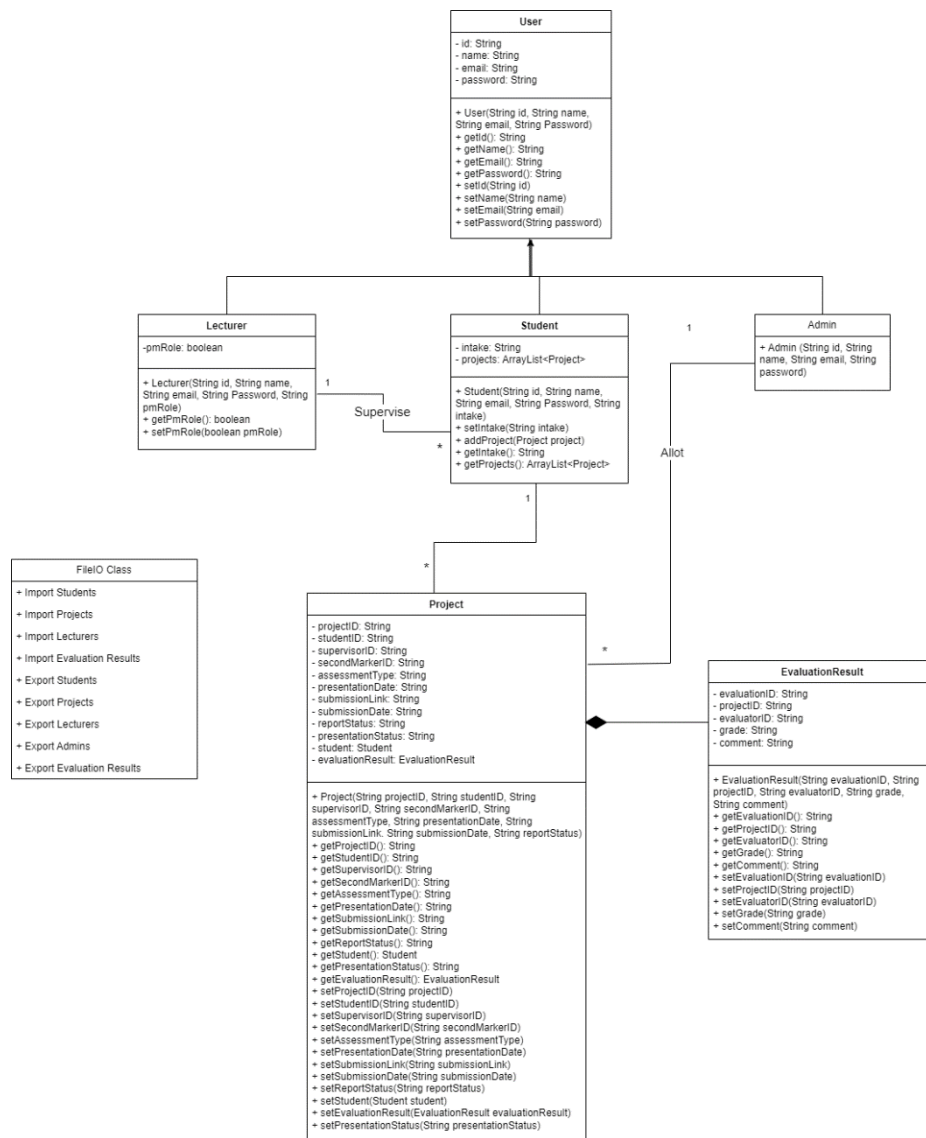
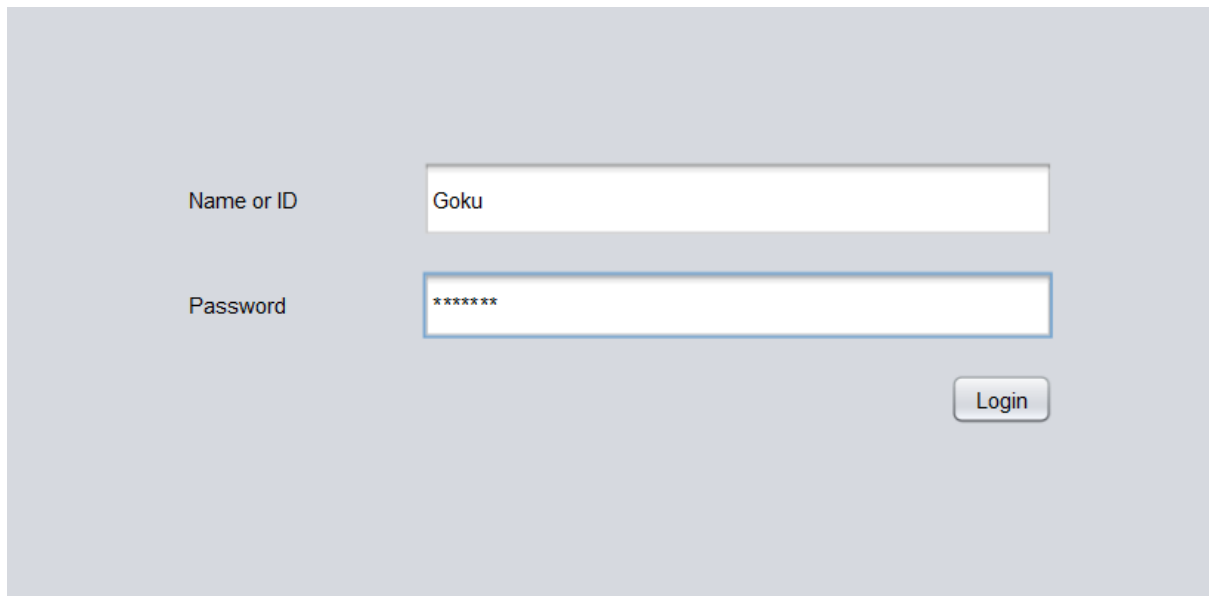


Figure 3.0: Class diagram for all classes

4.0 Sample Output

4.1 Admin



A login form with a light gray background. It contains two input fields: "Name or ID" with the text "Goku" and "Password" with masked characters "*****". A "Login" button is positioned to the right of the password field.

Name or ID	<input type="text" value="Goku"/>
Password	<input type="password" value="*****"/>
	<input type="button" value="Login"/>

Figure 4.1.1: Login

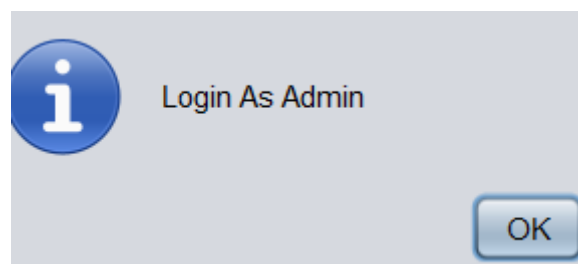
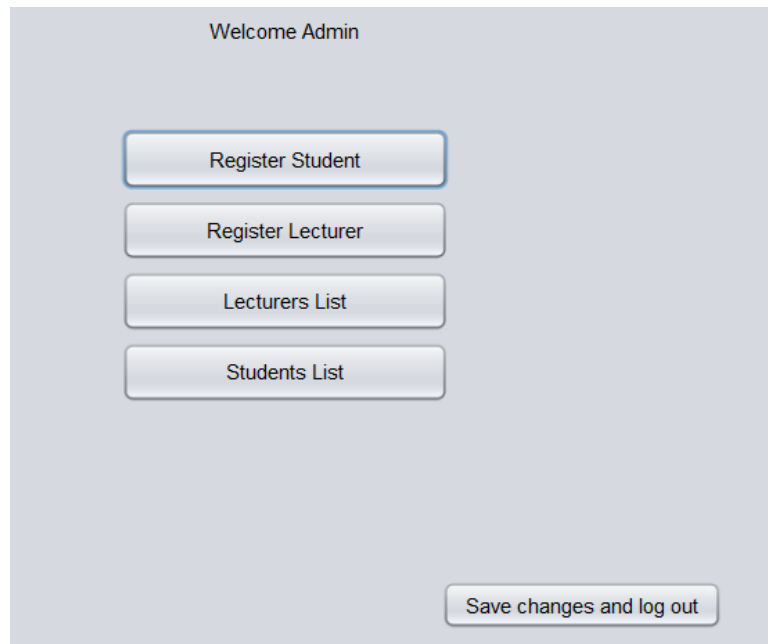


Figure 4.1.2: Successful login as admin



Welcome Admin

Register Student

Register Lecturer

Lecturers List

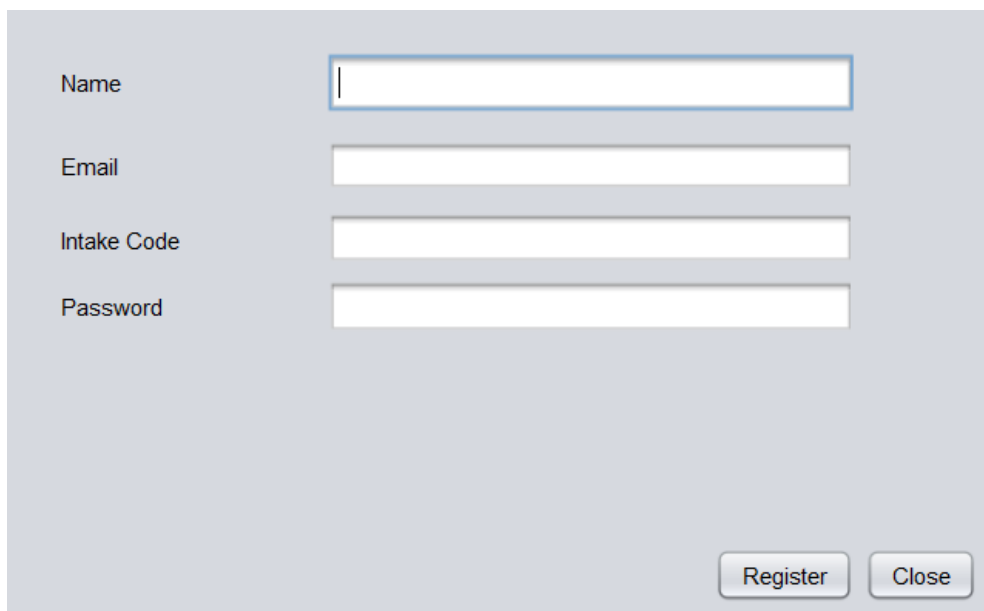
Students List

Save changes and log out

This screenshot shows the Admin Main Menu. At the top, it says "Welcome Admin". Below this, there are four buttons stacked vertically: "Register Student", "Register Lecturer", "Lecturers List", and "Students List". At the bottom right, there is a button labeled "Save changes and log out".

Figure 4.1.3: Admin Main Menu

Admin main menu is the first menu which pops up after logging in as admin. There are functions such as: Register student, Register Lecturer, Lecturers List, Students List and Save changes and log out. When admin finishes he clicks on “Save changes and log out” button to go back to login page.



Name

Email

Intake Code

Password

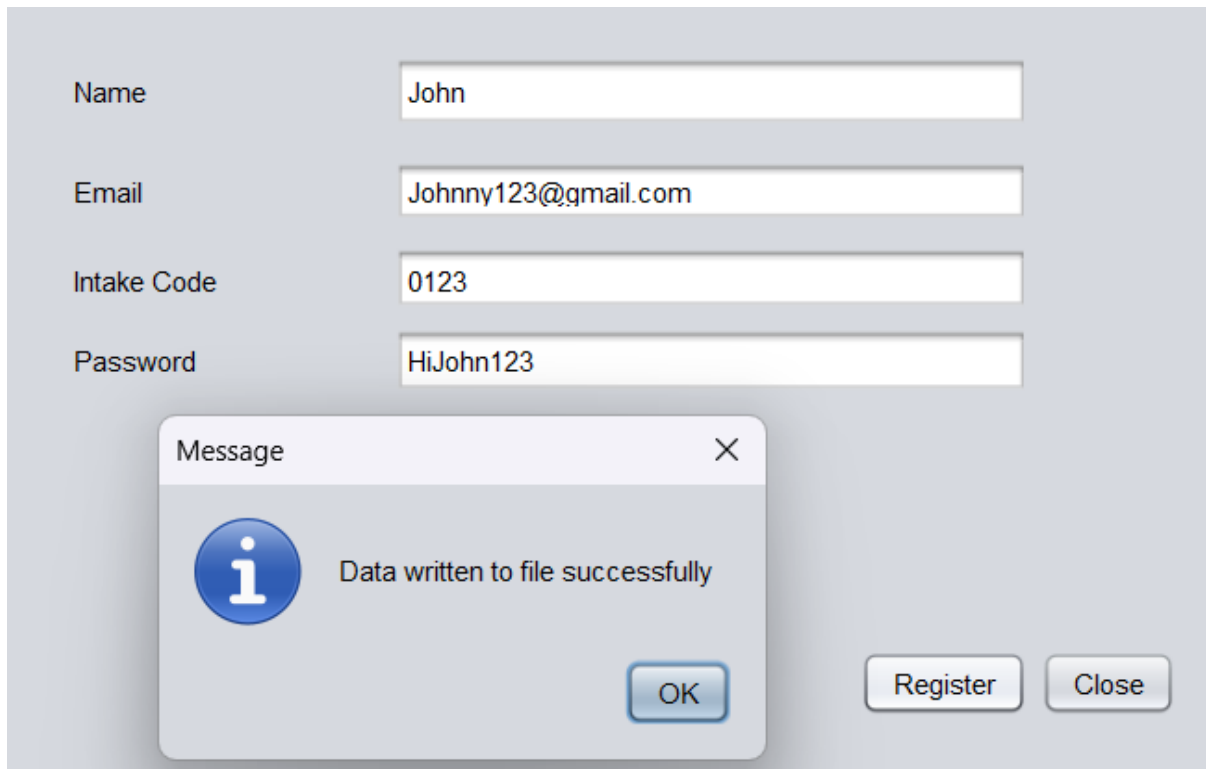
Register

Close

This screenshot shows the Register Student form. It has four input fields labeled "Name", "Email", "Intake Code", and "Password". At the bottom right, there are two buttons: "Register" and "Close".

Figure 4.1.4: Register Student

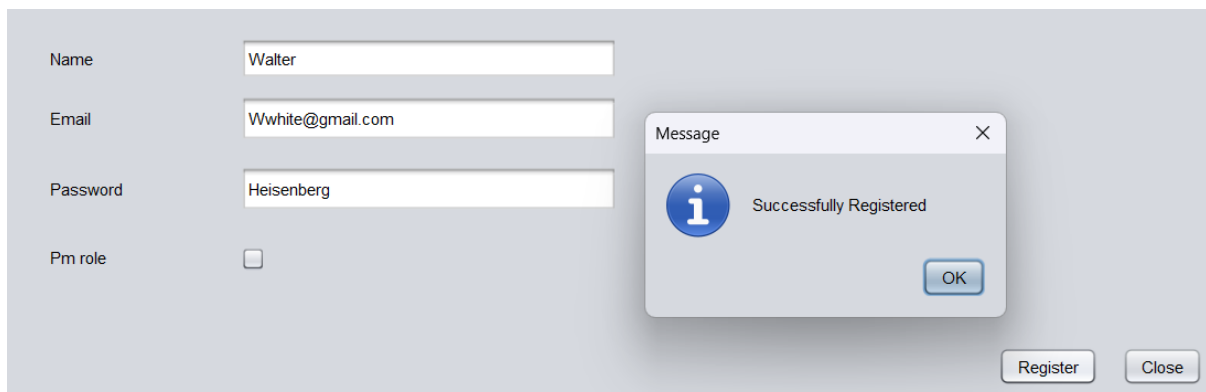
There is a menu with fields Name, Email, Intake Code and Password that need to be filled. Admin will register all the new students using this menu. And there is no ID Field because the program will generate it itself



The screenshot shows a registration form with four input fields: Name (John), Email (Johnny123@gmail.com), Intake Code (0123), and Password (HiJohn123). Below the form, a modal message box titled "Message" displays an information icon and the text "Data written to file successfully". At the bottom right of the form, there are three buttons: "OK" (part of the message box), "Register", and "Close".

Figure 4.1.5: Successfully registered student

After registration a new student the information message will appear.



The screenshot shows a registration form for a lecturer with four input fields: Name (Walter), Email (Wwhite@gmail.com), Password (Heisenberg), and a "Pm role" checkbox which is unchecked. A modal message box titled "Message" displays an information icon and the text "Successfully Registered". At the bottom right of the form, there are three buttons: "OK" (part of the message box), "Register", and "Close".

Figure 4.1.6: Register Lecturer with successfully created account

Similar registration menu for the lecturers. It has different fields and also admin will see the message of successfully registered account.

The screenshot shows a web interface titled "Lecturers List". At the top left, there is a text input field for searching, followed by a "Search" button. Below this is a table with the following data:

ID	Name	Email	PM
LR01	Spongebob	spongebob@mail.com	true
LR02	Patrick	Partrick@mail.com	false
LR03	Potato	Potato@mail.com	false
LR04	Patrick	Patrick@mail.com	false
LR05	Pikachu	Pikachu@mail.com	false
LR06	Walter	Wwhite@gmail.com	false

At the bottom right of the interface, there are three buttons: "Delete", "Edit", and "Close".

Figure 4.1.7: Lecturers list menu

The lecturers list menu where program gets the data from the arrays filled with data from Lecturers text file. Above the table there is a searching field and search button to make it more convenient to find exact account. There are all the columns in the table including ID but there is no password column for safety. Here admin can choose an account that needs to be changed or deleted.

Lecturers List

po

ID	Name	Email	PM
LR01	Spongebob	spongebob@mail.com	true
LR03	Potato	Potato@mail.com	false

Figure 4.1.8: Searching function. Any matches between Searching Field and Lecturers

Using the search button admin can find any matched accounts by any of its attribute. In this figure there are 2 matched accounts containing “po” in any of its attributes (in Name). After pressing Search button the table will be updated by only matching results. If there are no matches the table will be empty and if admin wants to get all the data he should leave the searching field empty.

Lecturers List

lr04

ID	Name	Email	PM
LR04	Patrick	Patrick@mail.com	false

Figure 4.1.9: Search by ID

In figure 4.1.9, the searching is by its exact ID and this account is ready to be updated.

ID	LR04
Name	<input type="text" value="Patrick"/>
Email	<input type="text" value="Patrick@mail.com"/>
Password	<input type="text" value="Patrick123"/>
PM	<input type="checkbox"/>
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

Figure 4.1.10: Edit lecturer menu

The edit lecturer menu has the same fields already filled by its data and in this menu admin can change all the fields. Also there is an option to allot Project Manager role to this lecturer or remove this role if lecturer already have been allotted.

ID	LR04
Name	<input type="text" value="PatrickStar"/>
Email	<input type="text" value="Patrickstar@mail.com"/>
Password	<input type="text" value="Patrick12345"/>
PM	<input checked="" type="checkbox"/>
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

Figure 4.1.11: Edited data ready to update

Edited data and allotted Project Manager role is ready to be updated.

<input type="text"/> <input type="button" value="Search"/>			
ID	Name	Email	PM
LR01	Spongebob	spongebob@mail.com	true
LR02	Patrick	Partrick@mail.com	false
LR03	Potato	Potato@mail.com	false
LR04	PatrickStar	Patrickstar@mail.com	true
LR05	Pikachu	Pikachu@mail.com	false
LR06	Walter	Wwhite@gmail.com	false

Figure 4.1.12: Updated Lecturers list

Figure 4.1.12 shows the lecturers list with updated Name, Email and PM role.

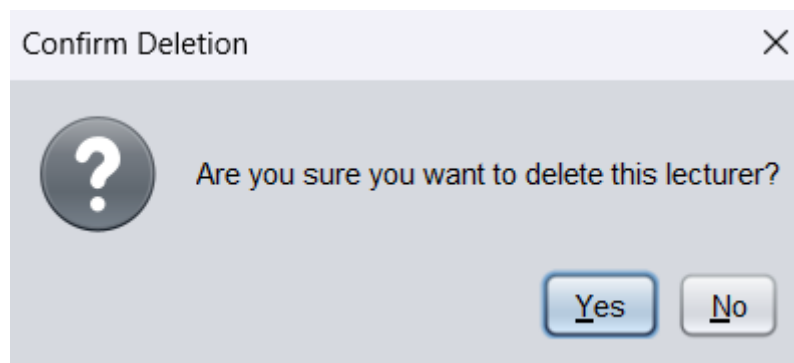


Figure 4.1.13: Delete Confirmation message

After pressing delete button there will be a confirmation message to not accidentally delete account by miss clicking on it.

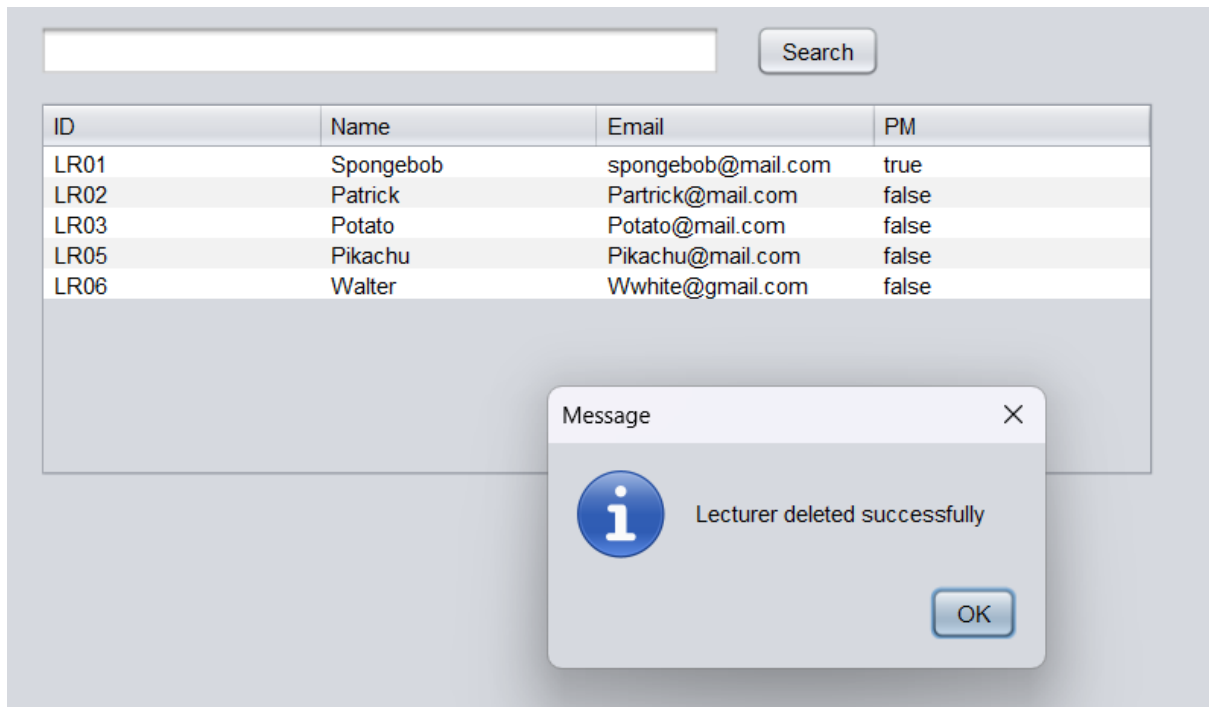


Figure 4.1.14: Updated Lecturers List without deleted account

After deleting an account the table content will be updated and the message will pop telling the account was successfully deleted.

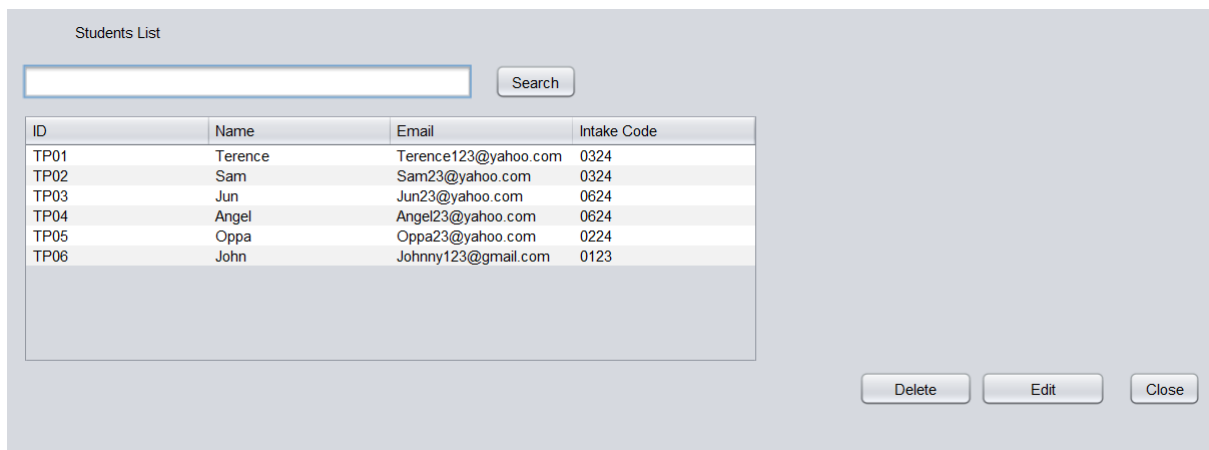


Figure 4.1.15: Students List menu

j

ID	Name	Email	Intake Code
TP03	Jun	Jun23@yahoo.com	0624
TP06	John	Johnny123@gmail.com	0123

Figure 4.1.16: Search by a single character

Edit Student Information

ID TP06

Name

Email

Intake Code

Password

Figure 4.1.17: Student Edit Menu

ID	TP06
Name	<input type="text" value="Johny"/>
Email	<input type="text" value="Johnnyzxc@gmail.com"/>
Intake Code	<input type="text" value="0222"/>
Password	<input type="text" value="zxcasd23"/>
<input type="button" value="Update"/>	

Figure 4.1.18: Editted data ready to update

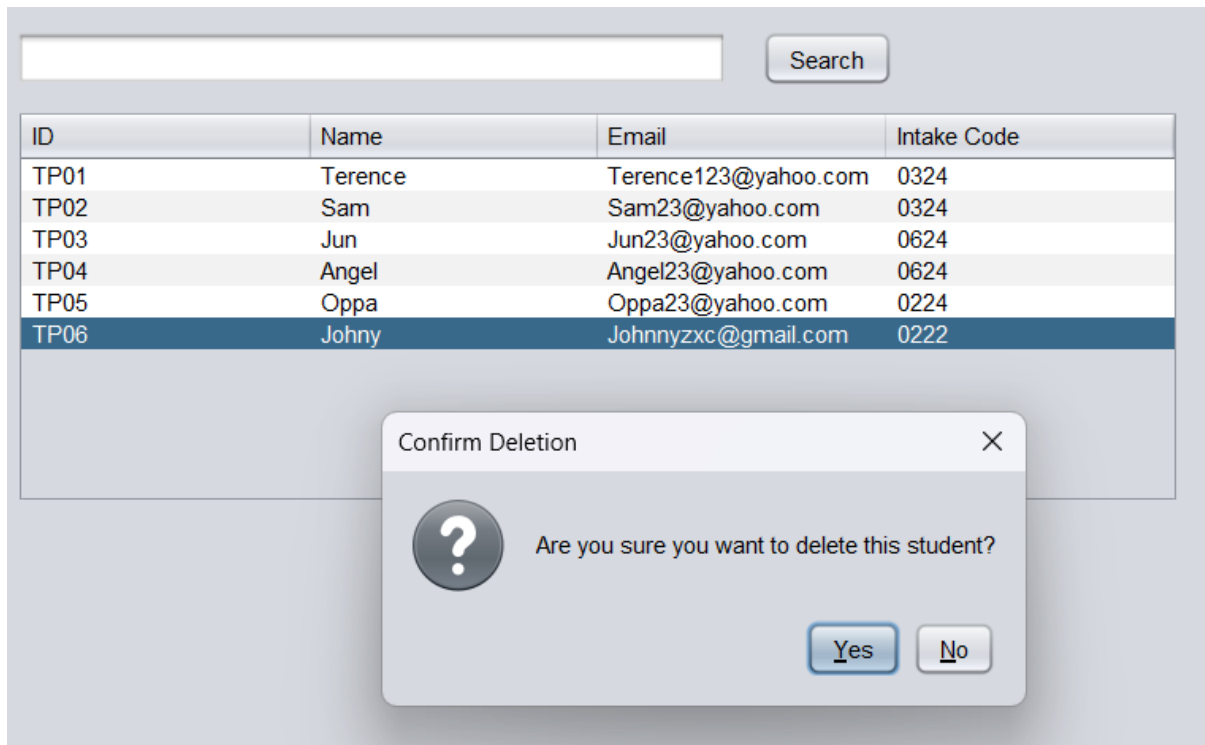


Figure 4.1.19: Updated table and Delete Confirmation Message

ID	Name	Email	Intake Code
TP01	Terence	Terence123@yahoo.com	0324
TP02	Sam	Sam23@yahoo.com	0324
TP03	Jun	Jun23@yahoo.com	0624
TP04	Angel	Angel23@yahoo.com	0624
TP05	Oppa	Oppa23@yahoo.com	0224

Figure 4.1.20: Updated table without deleted account

In Student List menu all the functions are similar to Lecturers such as: Search, Edit, Delete.

4.2 Project Manager

The screenshot displays the 'Project Manager Main' interface. At the top right is a button labeled 'Save changes and Logout'. Below it is a table titled 'Alloted Students list' with the following data:

Student ID	Name	AssessmentType	Supervisor ID	Second Marker ID
TP01	Terence	CP2	LR04	LR02
TP02	Sam	CP2	LR03	LR04
TP03	Jun	CP1	LR04	LR03
TP04	Angel	CP1	LR04	LR02
TP05	Oppa	Internship	LR04	LR05

Below the table are three buttons: 'Assign Assessment', 'Edit Assessment', and 'Check Report'.

Figure 4.2.1: Project Manager Main

Project Manager main interface obtain several of function such as “Assign Assessment”, “Edit Assessment”, “Check Report”, “Save Change and Logout” and “Student List”. In the student list displayed all of the student information such as ‘Student ID’, “Name”, “Assess Type”, “Supervisor ID” and ‘Second Marker ID” to Project Manager.

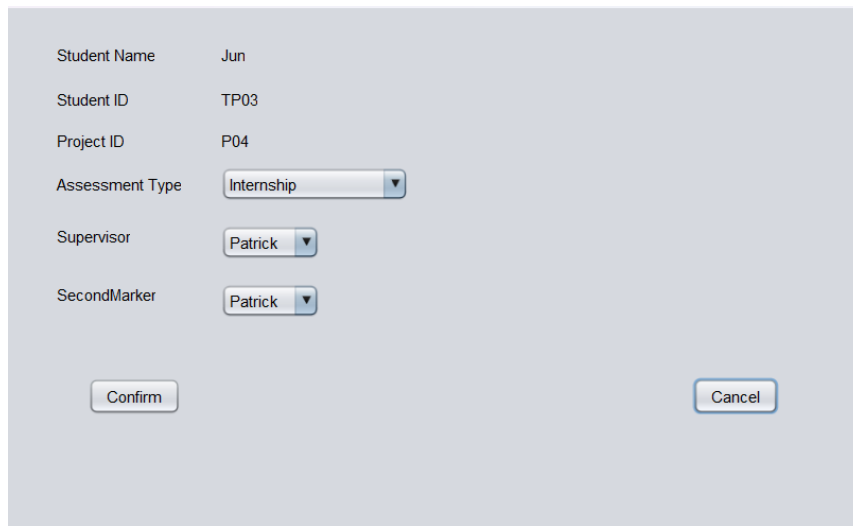
The screenshot displays the 'Assign Assessment' interface. At the top, there are filters: 'Allot by' with buttons for 'Individual' and 'Intake'; 'Supervisor' with a dropdown menu showing 'Patrick'; 'Second Marker' with a dropdown menu showing 'Potato'; and 'Assessment Type' with a dropdown menu showing 'Internship'. Below these filters is a table with the following data:

Student ID	Student Name	Intake
TP06	Fib	2406
TP07	Quary	2410
TP08	Pias	2406
TP09	Mono	2402

Below the table is an 'Assign' button. A dropdown menu for 'Assessment Type' is open, showing options: 'Internship', 'Investigation Reports', 'CP1', 'CP2', 'RMCP', and 'FYP'.

Figure 4.2.2: Assign Assessment

In the “Assign Assessment” dialog, Project Manager can assign assessment to the student by intake or individual. Then, must select the student by click their row. Lastly, choose “Supervisor”, “Second Marker” and “Assessment Type” from the combo box.



The screenshot shows a dialog box titled "Edit Assessment". It contains the following fields and controls:

- Student Name: Jun
- Student ID: TP03
- Project ID: P04
- Assessment Type: A dropdown menu with "Internship" selected.
- Supervisor: A dropdown menu with "Patrick" selected.
- SecondMarker: A dropdown menu with "Patrick" selected.
- At the bottom, there are two buttons: "Confirm" and "Cancel".

Figure 4.2.3: Edit Assessment

Project Manager allowed to edit the student “Assessment Type”, “Supervisor” and “Second Marker” by the “Edit Assessment” button. Before clicking the edit button, Project Manager have to select which student from the list display on “Project Manager Main” interface. Otherwise, the error handler will pop up a message box like Figure.

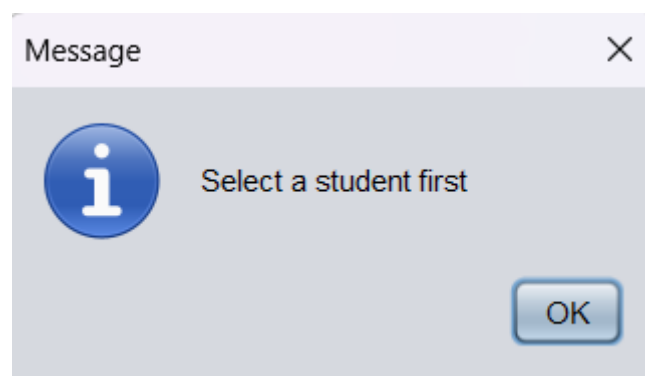


Figure 4.2.4: Error Handler

Status of Report

Filter by intake 2402 ▼ Filter Clear

Student ID	Student Name	Intake	Assessment Type	Report Status
TP01	Terence	2410	CP2	Submitted
TP02	Sam	2406	CP2	Submitted
TP03	Jun	2406	CP1	Pending
TP04	Angel	2406	CP1	Evaluated
TP05	Oppa	2410	Internship	Evaluated

Back

Figure 4.2.5: Check Report

The last function of Project Manager is "Check Report". Project Manager can filter the student data by the intake, for example intake "2402", the list will just show those students who is from "2402" intake. Check Report also allowed Project Manager to track "Report Status".

4.3 Lecturer

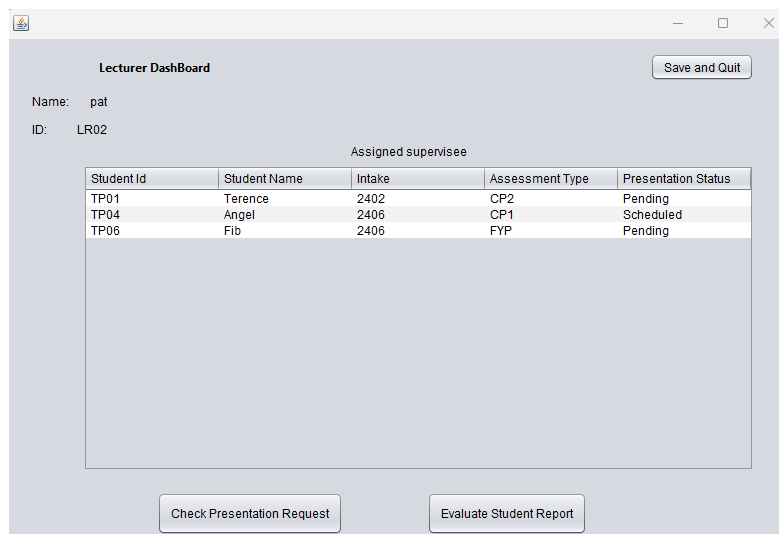


Figure 4.3.1: Dashboard for lecturer

When user successfully login, he will see a dashboard with a supervisee table. Figure 4.3.1 shows the table where only the assigned supervisee will show up whether the lecturer is assigned as supervisor or second marker. There two labels on the top left to indicate the account information, where user can see the username and id of the logged in lecturer. There are two buttons on the dashboard, first is the check presentation request button where user can select a supervisee from the table and check his presentation request.

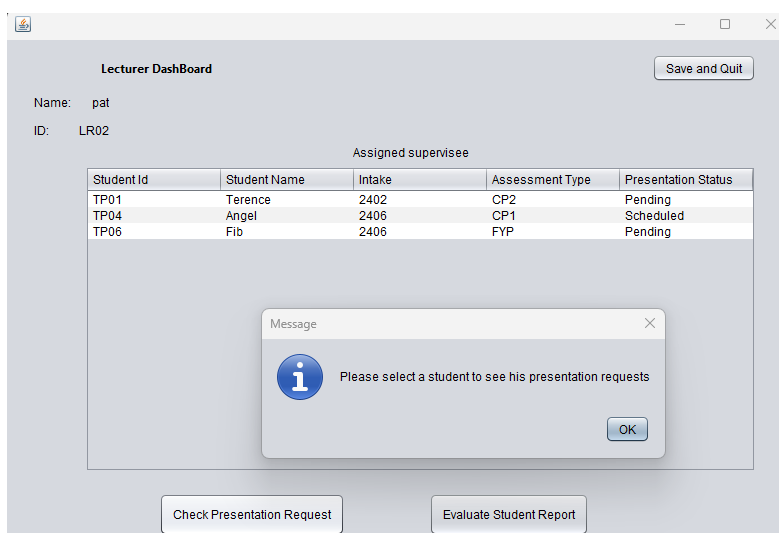


Figure 4.3.2: User does not select a row for checking presentation request

Figure 4.3.3 above showcase the situation when user does not select a student in the table before he presses the check presentation request. The program will show a message box that instruct the user to choose a supervisee from the table first.

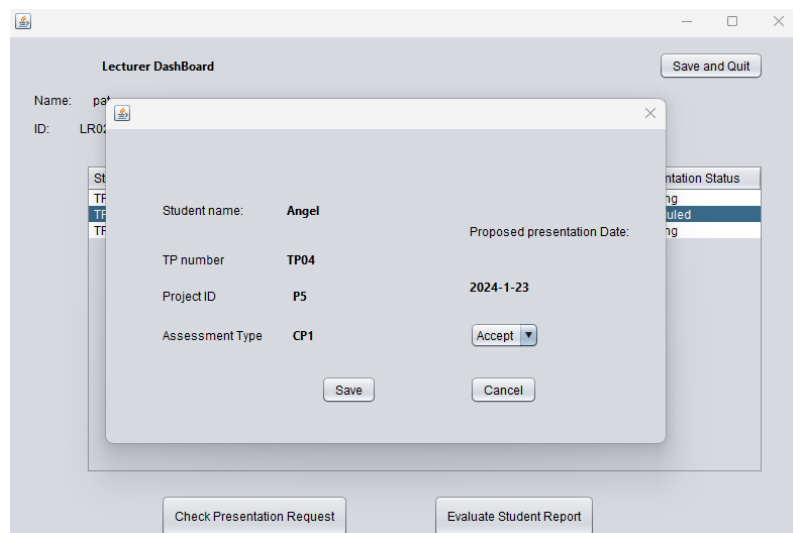


Figure 4.3.3: Check Presentation Interface

Figure 4.3.3 showcase the UI when user pressed the check presentation request. User can check their details of the presentation like project Id and assessment type for the ongoing project with the selected student. If the lecturer has verified the presentation date, he can choose to accept or reject the proposed presentation date. To finalize the decision the lecturer can press the confirm button below or cancel his changes to go back to the dashboard

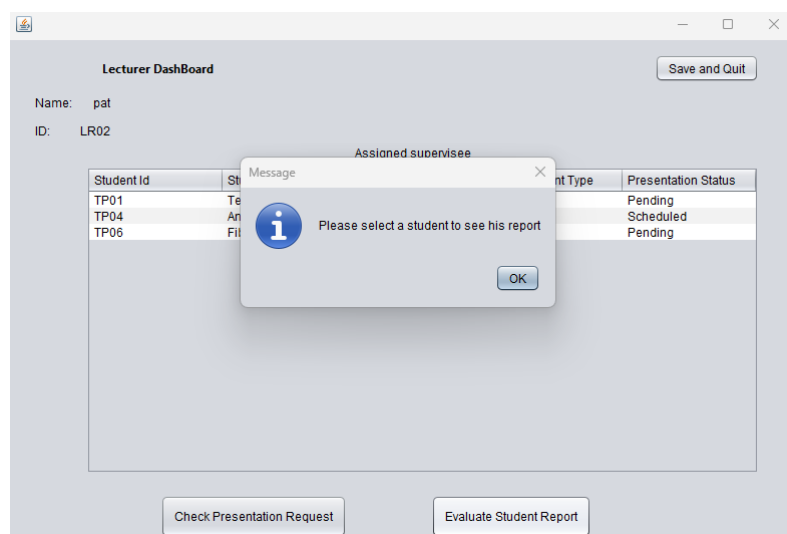


Figure 4.3.4: Error when user does not select a student to evaluate his report

Figure 4.3.4 showcase the situation when user wanted to evaluate a report but does not select a student in the dashboard table.

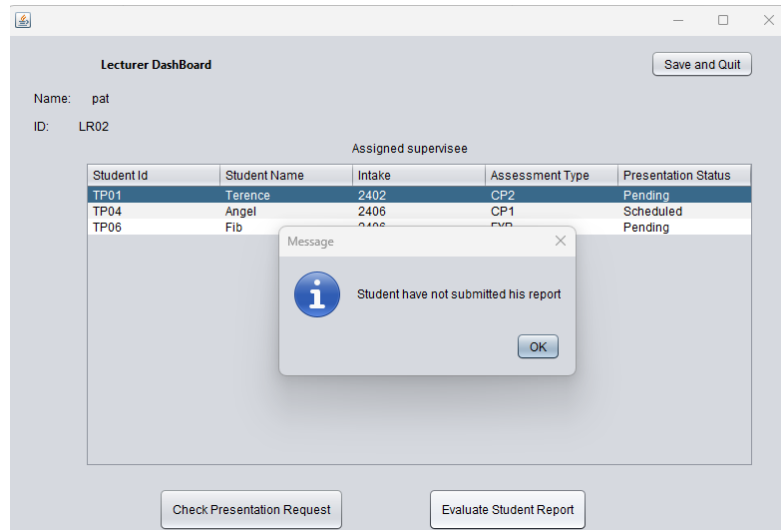


Figure 4.3.5: Error when user wanted to evaluate a student that have not submitted his report

Figure 4.3.5 showcase the situation when user wanted to evaluate a student report, but the selected student has not submitted a report.

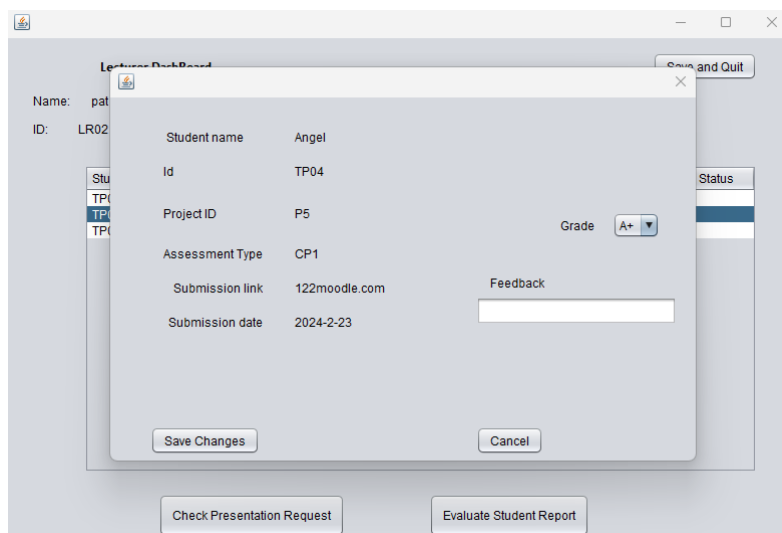


Figure 4.3.6: Evaluate student report Interface

When user click on the evaluate student report, they will see an interface where they can see details of report submitted by the selected student. Figure 4.3.6 showcase the interface where user can see details of the submitted report like the submission link and submission date. User also can grade the report and leave short feedback to the report. To

finalize the decision user, need to press the save changes button or cancel button to cancel the changes.

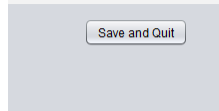
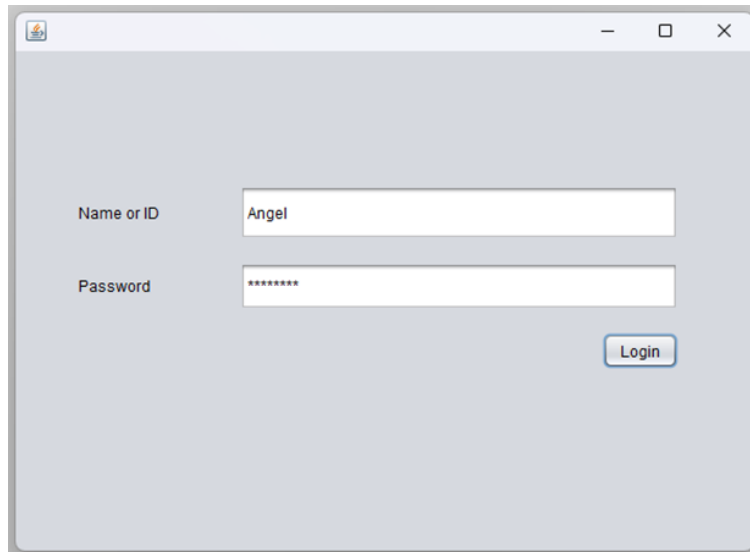


Figure 4.3.7: Save and quit button

Figure 4.3.7 show the save and quit button located in the dashboard. For safety purposes, this button is used to prevent accidental changes made by the user would not directly affect the database (text file). After user confirm and finalized his changes, he can press the button to safely update the changes to the text file.

4.4 Student

Login



A screenshot of a web browser window displaying a login interface. The window has a standard title bar with minimize, maximize, and close buttons. The login form is centered and consists of two text input fields. The first field is labeled 'Name or ID' and contains the text 'Angel'. The second field is labeled 'Password' and contains masked characters represented by seven asterisks '*****'. To the right of the password field is a button labeled 'Login'.

Figure 4.4.1: Login

Explanation:

The system will verify that each user has entered their name, ID, and password. Users will be directed to various roles and functionalities by distinct forms of ID. In this instance, the user may access the student site by checking in using the name “Angel” and the password “Angel123”.

Student Page

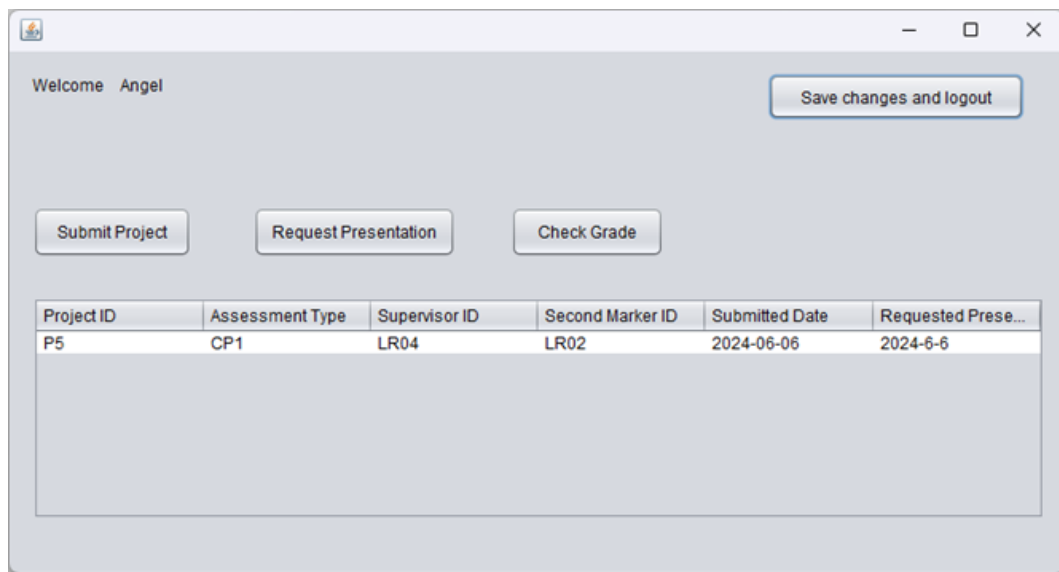
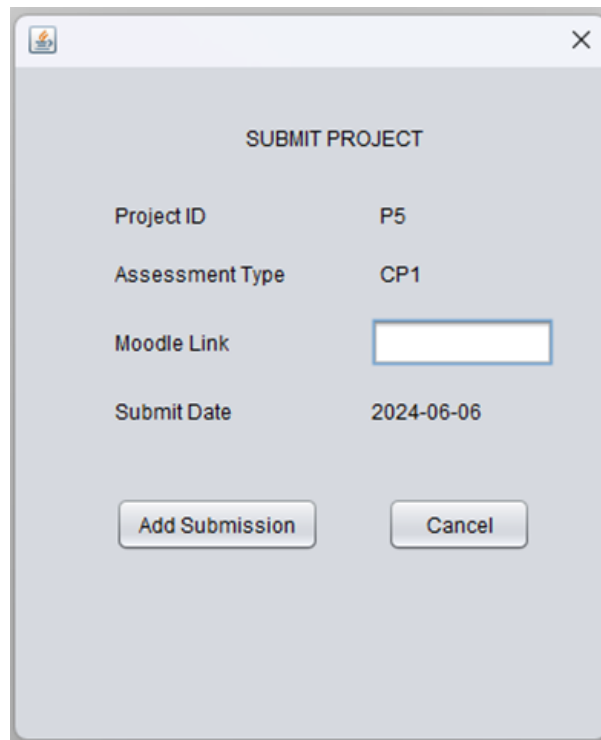


Figure 4.4.2: Student interface

Explanation:

Firstly, a welcome message named "Welcome <student name>" will appear on the student website. This page has many buttons that can lead the user to another user interface, including the Submit Project, Request Presentation, and Check Grade buttons. The project information that must be turned in by the student is displayed in a table. Before clicking the button, students must pick the row; otherwise, a prompt to select the row will appear. Lastly, the user can return to the login page by clicking the "Save changes and logout" button.

Submit Project Page

A screenshot of a 'SUBMIT PROJECT' dialog box. The dialog has a title bar with a small icon on the left and a close button (X) on the right. The main area is light gray and contains the following fields: 'Project ID' with the value 'P5', 'Assessment Type' with the value 'CP1', 'Moodle Link' with an empty text input field, and 'Submit Date' with the value '2024-06-06'. At the bottom, there are two buttons: 'Add Submission' and 'Cancel'.

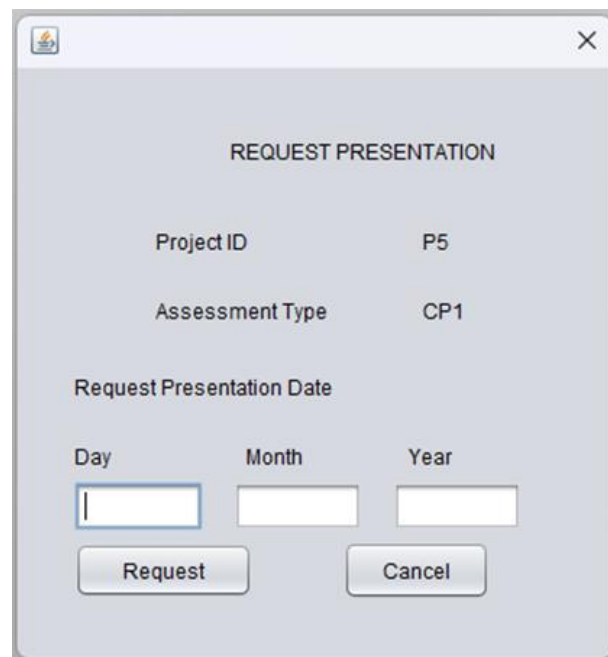
SUBMIT PROJECT	
Project ID	P5
Assessment Type	CP1
Moodle Link	<input type="text"/>
Submit Date	2024-06-06
<div><button>Add Submission</button><button>Cancel</button></div>	

Figure 4.4.3: Submit project interface

Explanation:

The specifics of the Project ID, Assessment Type, Moodle Link that the student must complete, and the automatically generated Submit Date are displayed on the “Submit Project” page. Through this, students may submit work and have the “Project.txt” file updated.

Request Presentation Page



The screenshot shows a window titled "REQUEST PRESENTATION" with a close button (X) in the top right corner. Inside the window, the following information is displayed:

Project ID	P5
Assessment Type	CP1

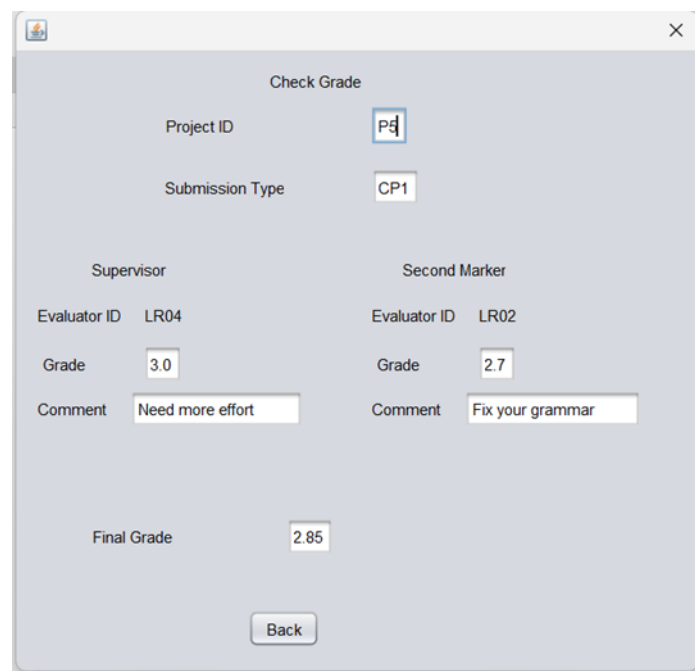
Below this, the text "Request Presentation Date" is followed by three input fields labeled "Day", "Month", and "Year". The "Day" field is currently active, indicated by a blue border and a vertical cursor. At the bottom of the window, there are two buttons: "Request" and "Cancel".

Figure 4.4.4: Request presentation interface

Explanation:

The students must enter the day, month, and year they wish to do their presentation on the “Request Presentation” page, which displays the Project ID and Assessment Type data. It will be updated to the “Projects.txt” file upon request.

Check Grade Page



The screenshot shows a 'Check Grade' window with a light gray background. At the top, the title 'Check Grade' is centered. Below it, the 'Project ID' is 'P4' and the 'Submission Type' is 'CP1'. The window is divided into two columns for evaluators. The left column is for the 'Supervisor' (Evaluator ID: LR04) with a grade of 3.0 and a comment 'Need more effort'. The right column is for the 'Second Marker' (Evaluator ID: LR02) with a grade of 2.7 and a comment 'Fix your grammar'. At the bottom, the 'Final Grade' is calculated as 2.85. A 'Back' button is located at the very bottom center.

Check Grade	
Project ID	P4
Submission Type	CP1
Supervisor	
Evaluator ID	LR04
Grade	3.0
Comment	Need more effort
Second Marker	
Evaluator ID	LR02
Grade	2.7
Comment	Fix your grammar
Final Grade	2.85
Back	

Figure 4.4.5: Check grade interface

Explanation:

Students can view their marks in detail and comment on their submissions made by the supervisor and second marker by using the “Check Grade” feature. The student may receive different grades on remarks from the supervisor and second marker. The average of the grades obtained by the supervisor and second maker determines the final grade.

5.0 Description and Justification of Object-Oriented Concepts

5.1 Admin

Abstraction

```
abstract class User {  
    public String id,name, email, password;  
  
    //Constructor  
    public User(String id, String name, String email, String password){  
        this.id = id;  
        this.name = name;  
        this.email = email;  
        this.password = password;  
    }  
  
    // Creating abstract methods  
    // Getter  
    public abstract String getID();  
  
    public abstract String getName();  
  
    public abstract String getEmail();  
}
```

Figure 5.1.1: Abstract class User

The abstract class User defines common attributes and behaviors using abstract methods, promoting code structure and polymorphic behavior through subclasses that extend it.

Inheritance

```

public class Student extends User{
    private String intake;
    ArrayList<Project> projects = new ArrayList<>();

    public Student(String id, String name, String email, String password, String intake) {
        super(id, name, email, password);
        this.intake = intake;
    }

    // Example of Polymorphism
    //getter
    @Override public String getID(){
        return id;
    }

    @Override public String getName(){
        return name;
    }

    @Override public String getEmail(){

```

Figure 5.1.2: Class Student inherited from abstract class User

```

public class Lecturer extends User{
    private boolean pmRole;

    public Lecturer(String id, String name, String email, String password, boolean pmRole){
        super(id, name, email, password);
        this.pmRole = pmRole;
    }

    // Example of Polymorphism
    //getter
    @Override public String getID(){
        return id;
    }

    @Override public String getName(){
        return name;
    }

    @Override public String getEmail(){

```

Figure 5.1.3: Class Lecturer inherited from abstract class User

Classes Student and Lecturer are extend class User inheriting its attributes and methods. The classes implement specific behaviors while maintaining a common interface with the superclass, enforced through private attributes and public accessors.

Polymorphism


```

private void writeDataToFile(Object data, String fileName) {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName, true))) {
        if (data instanceof Lecturer lecturer) {
            String line = lecturer.getID() + "/" + lecturer.getName() + "/" + lecturer.getEmail() + "/" + lecturer.getPassword();
            writer.write(line);
            writer.newLine();
        } else if (data instanceof Student student) {
            String line = student.getID() + "/" + student.getName() + "/" + student.getEmail() + "/" + student.getPassword();
            writer.write(line);
            writer.newLine();
        }
        JOptionPane.showMessageDialog(null, "Data written to file successfully");
    } catch (IOException e) { // Print the exception stack trace for debugging
        JOptionPane.showMessageDialog(null, "Error writing data to file");
    }
}

```

Figure 5.1.4: Method writeDataToFile

This code uses the instanceof operator to determine the runtime type of a data object, dynamically resolving and calling methods for Lecturer and Student objects, demonstrating polymorphic behavior in a single method.

Encapsulation

```

public class Lecturer extends User{
    private boolean pmRole;

    public Lecturer(String id, String name, String email, String password) {
        super(id, name, email, password);
        this.pmRole = pmRole;
    }
}

public class Student extends User{
    private String intake;
    ArrayList<Project> projects = new ArrayList<>();

    public Student(String id, String name, String email, String password, String intake) {
        super(id, name, email, password);
        this.intake = intake;
    }
}

```

Figures 5.1.5: Capsulated attributes in classes Lecturer and Student

This code demonstrates encapsulation by using access modifiers to restrict direct access to class fields, ensuring data integrity, hiding implementation details, and allowing controlled modification and access of class attributes.

5.2 Project Manager

Abstraction

```
abstract class User {  
    public String id, name, email, password;  
  
    //Constructor  
    public User(String id, String name, String email, String password)  
    {  
        this.id = id;  
        this.name = name;  
        this.email = email;  
        this.password = password;  
    }  
  
    // Creating abstract methods  
    // Getter  
    public abstract String getID();  
  
    public abstract String getName();  
  
    public abstract String getEmail();  
  
    public abstract String getPassword();  
  
    //Setter  
    public abstract void setID(String id);  
}
```

Figure 5.2.1: Abstract Class User

Abstraction involves hiding the complex implementation details and showing only the essential features of an object. This can be achieved using abstract classes and interfaces. For example, figure shown is an abstract class “User”, it is a super class that allow other sub class such as “Lecturer” and “Student” to use the method directly Figure.

```
public class Student extends User{  
    private String intake;  
    ArrayList<Project> projects = new ArrayList<>();  
  
    public Student(String id, String name, String email, String password, String intake)  
    {  
        super(id, name, email, password);  
        this.intake = intake;  
    }  
  
    // Example of Polymorphism  
    //getter  
    @Override public String getID(){  
        return id;  
    }  
  
    @Override public String getName(){  
        return name;  
    }  
  
    @Override public String getEmail(){  
        return email;  
    }  
}
```

Figure 5.2.2: Sub Class Student

Encapsulation

```
public class Project {  
    private String projectID, studentID, supervisorID, secondMarkerID, assessmentType, presentationDate, submissionLink;  
    private Student student;  
    private EvaluationResult evaluationResult;  
  
    public Project(String projectID, String studentID, String supervisorID, String secondMarkerID, String assessmentType, String presentationDate, String submissionLink) {  
        this.projectID = projectID;  
        this.studentID = studentID;  
        this.supervisorID = supervisorID;  
        this.secondMarkerID = secondMarkerID;  
        this.assessmentType = assessmentType;  
        this.presentationDate = presentationDate;  
        this.submissionLink = submissionLink;  
        this.submissionDate = submissionDate;  
        this.reportStatus = reportStatus;  
        this.presentationStatus = presentationStatus;  
    }  
  
    //Getter  
    public String getProjectID() {  
        return projectID;  
    }  
}
```

Figure 5.2.3: Capsulated Attribute in Class Project

In class “Project”, there have several oh capsulated attribute such as “Project ID”, “Student ID”, “Supervisor ID”, “Second Marker ID” and so on. All these attributes have been capsulated in the class “Project”, so that we need to have settler and getter methods to access and modify them and call to other class like Figure.

```
//update project in array  
for(Project project:projects){  
    if(project.getProjectID().equals(EditAssProjIDLbl.getText())){  
        project.setAssessmentType(selectedType);  
        project.setSupervisorID(supervisorID);  
        project.setSecondMarkerID(secondMarkerID);  
    }  
}
```

Figure 5.2.4: Get Object Attribute from Capsulated Attribute

Inheritance

```
public class Student extends User{  
    private String intake;  
    ArrayList<Project> projects = new ArrayList<>();  
  
    public Student(String id, String name, String email, String password, String intake) {  
        super(id, name, email, password);  
        this.intake = intake;  
    }  
}
```

Figure 5.2.5: Student Inherit User

Inheritance allows one class to inherit the properties and methods of another class. As figure we can see subclass “Student” is type of user, so it enrolls in the hierarchy of superclass ‘User’.

```

public ProjectManagerMain(ArrayList<Student> students, ArrayList<Project> projects, ArrayList<Lecturer> lecturers)
{
    this.students = students;
    this.projects = projects;
    this.lecturers = lecturers;
    initComponents();
    DefaultTableModel model = (DefaultTableModel)MainStdListTable.getModel();

    for (Student student: students){
        // get student latest project
        if (!student.getProjects().isEmpty()){
            Project latestProject = student.getProjects().get(student.getProjects().size()-1);
            String [] tableDataRow = {student.getID(), student.getName(), latestProject.getAssessmentType(), latestProject.getAssessmentScore()};
            model.addRow(tableDataRow);
        }
    }
}

```

Figure 5.2.6: Class ProjectManagerMain Call Subclass Student's Method

In Figure 5.2.6 we can see “GetProjects” method were called from another subclass.

Polymorphism

```

for (Student student: students){
    if (student.getProjects().isEmpty()){
        String [] tableDataRow = {student.getID(), student.getName(), student.getIntake()};
        Indmodel.addRow(tableDataRow);
    }
}

for (Student student: students){
    if (student.getProjects().isEmpty()){
        String [] tableDataRow = {student.getID(), student.getName(), student.getIntake()};
        Intmodel.addRow(tableDataRow);
    }
}

```

Figure 5.2.7: Method “getID” in Object “student”

```

for(Lecturer lecturer:lecturers){
    if (lecturer.getName().equals(supervisor)){
        supervisorID = lecturer.getID();
    }
    if (lecturer.getName().equals(secondMarker)){
        secondMarkerID = lecturer.getID();
    }
}

```

Figure 5.2.8: Method “getID” in Object “lecturer”

From Figure 5.2.7 and Figure 5.2.8 we can see that they are using a same method even they are different object.

5.3 Lecturer

Abstraction

```
// Create a abstract user class
abstract class User {
    public String id,name, email, password;

    //Constructor
    public User(String id, String name, String
        this.id = id;
        this.name = name;
        this.email = email;
        this.password = password;
    }

    // Creating abstract methods
    // Getter
    public abstract String getID();

    public abstract String getName();

    public abstract String getEmail();

    public abstract String getPassword();

    //Setter
    public abstract void setID(String id);
    public abstract void setName(String name);
}
```

Figure 5.3.1 Abstract user class for Lecturer

Figure 5.3.1 shows the codes for the abstract user class, this abstract class can be used for super class for the lecturer class because they have similar attributes such as id, name, email and password.

```
public class Lecturer extends User{
    private boolean pmRole;

    public Lecturer(String id, String name,
        super(id, name, email, password);
}
```

Figure 5.3.2: Extending abstract class for Lecturer class

Figure 5.3.2 shows the codes for Lecturer class to extends the user abstract class, `super()` is used to inherit all the common attributes.

Encapsulation

```
public class Lecturer extends User{
    private boolean pmRole;

    public Lecturer(String id, String name, String email, String password, boolean pmRole){
        super(id, name, email, password);
        this.pmRole = pmRole;
    }

    // Example of Polymorphism
    //getter
    @Override public String getID(){
        return id;
    }

    @Override public String getName(){
        return name;
    }

    @Override public String getEmail(){
        return email;
    }

    @Override public String getPassword(){
        return password;
    }
}
```

Figure 5.3.3: Access modifier in the Lecturer class

Figure 5.3.3 shows the access modifiers in the Lecturer class. There is setter and getter for all of the attributes in the Lecturer class.

```
public LecturerMain(Lecturer lecturer, ArrayList<
    this.loggedinLecturer = lecturer;
    this.students = students;
    this.projects = projects;
    this.evaluationResults = evaluationResults;
    initComponents();
    MainNameLbl.setText(lecturer.getName());
    MainIDLbl.setText(lecturer.getID());
}
```

Figure 5.3.4: Using Access modifier in Lecturer class to get object attribute

Figure 5.3.4 showcase the usage of access modifier in the lecturer dashboard code. A Lecturer object is instantiated using the Lecturer class, with getName() and getID(). The program was able to get name and id attributes from the object.

Inheritance

```
public class Lecturer extends User{
    private boolean pmRole;

    public Lecturer(String id, String name, String email, String password, boolean pmRole){
        super(id, name, email, password);
        this.pmRole = pmRole;
    }
}
```

Figure 5.3.5: Lecturer class extending from user superclass as a subclass

Figure 5.3.5 shows the Lecture subclass extending from user superclass, which means it inherited all the methods and attributes under the user superclass.

```
Lecturer lecturerur;
ArrayList<Student> students = new ArrayList<>(); //
ArrayList<Project> projects = new ArrayList<>(); //
ArrayList<EvaluationResult> evaluationResults = new

/**
 * Creates new form MainMenu
 * @param lecturer
 * @param students
 * @param projects
 * @param evaluationResults
 */
public LecturerMain(Lecturer lecturer, ArrayList<Stu
    this.loggedinLecterur = lecturer;
    this.students = students;
    this.projects = projects;
    this.evaluationResults = evaluationResults;
    initComponents();
    MainNameLbl.setText(lecturer.getName());
    MainIDLbl.setText(lecturer.getID());
```

Figure 5.3.6: Instantiate object using lecturer subclass

Figure 5.3.6 showcase the usage of the Lecturer subclass to instantiate Lecturer Object. In the codes for lecturer dashboard, a Lecture object was pass through using to constructor to determine which lecturer was logged in.

Polymorphism

```
public class Lecturer extends User{
    private boolean pmRole;

    public Lecturer(String id, String name, String email, String password){
        super(id, name, email, password);
        this.pmRole = pmRole;
    }

    // Example of Polymorphism
    //getter
    @Override public String getID(){
        return id;
    }

    @Override public String getName(){
        return name;
    }

    @Override public String getEmail(){
        return email;
    }

    @Override public String getPassword(){
        return password;
    }

    //Setter
    @Override public void setID(String id){
        this.id = id;
    }

    @Override public void setName(String name){
        this.name = name;
    }

    @Override public void setEmail(String email){
        this.email = email;
    }

    @Override public void setPassword(String password){
        this.password = password;
    }
}
```

Figure 5.3.7: Method overriding in the Lecturer class

Figure 5.3.8 showcase the usage of the usage of polymorphism for the Lecturer class. Since all methods inherited from the user abstract class are abstract methods, method overriding is used to rewrite the functionalities of the abstract methods.

5.4 Student

Abstraction

```
// Create a abstract class for student
abstract class User {
    public String id,name, email, password;

    //Constructor
    public User(String id, String name, String email, String password){
        this.id = id;
        this.name = name;
        this.email = email;
        this.password = password;
    }

    // Creating abstract methods
    // Getter
    public abstract String getID();

    public abstract String getName();

    public abstract String getEmail();

    public abstract String getPassword();

    //Setter
    public abstract void setID(String id);

    public abstract void setName(String name);

    public abstract void setEmail(String email);
}
```

Figure 5.4.1

All of the abstract methods described in User have concrete implementations available in the Student class. Since Student extends User, it is required that you implement all of the abstract methods of any class that extends an abstract class.

```
public class Student extends User{
    private String intake;
    ArrayList<Project> projects = new ArrayList<>();

    public Student(String id, String name, String email, String password, String intake) {
        super(id, name, email, password);
        this.intake = intake;
    }
}
```

Figure 5.4.2

The extends keyword is used to specify the Student class as a subclass of User. This implies that all fields and methods, even abstract ones, belong to User and are inherited by Student. Student has an extra field input that is unique to the Student class in addition to the

fields inherited from User. An ArrayList is used in the projects field to hold project items connected to the student.

Inheritance

```
public class StudentMain extends javax.swing.JFrame {
    Student LoggedInStudent;
    ArrayList<Project> AssignedProjects;
    ArrayList<Project> projects;
    ArrayList<Student> students;
    ArrayList<EvaluationResult> evaluationResults;

    /**
     * Creates new form MainFrame
     * @param student
     * @param students
     * @param projects
     * @param evaluationResults
     */
}
```

Figure 5.4.3

StudentMain is inherited from javax.swing.JFrame. In other words, StudentMain is a JFrame type that may be used whenever a JFrame is anticipated. All of JFrame's fields and methods are inherited by StudentMain. This covers how to add components, handle events, and set the title, size, and visibility of the window.

Polymorphism

```
//getter
@Override public String getID(){
    return id;
}

@Override public String getName(){
    return name;
}

@Override public String getEmail(){
    return email;
}

@Override public String getPassword(){
    return password;
}

//Setter
@Override public void setID(String id){
    this.id = id;
}

@Override public void setName(String name){
    this.name = name;
}

@Override public void setEmail(String email){
    this.email = email;
}
```

Figure 5.4.4

By enabling the Student class to improve or modify the functionality of methods inherited from the User class while preserving a consistent interface, this illustrates polymorphism.

Composition

```
Student LoggedInStudent;
ArrayList<Project> AssignedProjects;
ArrayList<Project> projects;
ArrayList<Student> students;
ArrayList<EvaluationResult> evaluationResults;
```

Figure 5.4.5

```
// Initialize table
DefaultTableModel model = (DefaultTableModel)MainProjectsTable.getModel();
for(Project AssignedProject:AssignedProjects){
    String [] tableDataRow = {AssignedProject.getProjectID(), AssignedProject.getAssessmentType(), AssignedProject.getSupervisorID(), AssignedProject.getAssessmentDate()};
    model.addRow(tableDataRow);
}
```

Figure 5.4.6

By utilizing these classes' capability StudentMain can demonstrate a "has-a" connection. The fields in the StudentMain class that show composition are LoffedInStudent, AssignedProjects, projects, students, and evaluationResults. StudentMain is made up of these objects since these fields are instances or collections of other classes.

6.0 Extra Feature

```
ArrayList<Student> students = new ArrayList<>(); // An arraylist to store all student object
ArrayList<Project> projects = new ArrayList<>(); // An arraylist to store all project object
ArrayList<Admin> admins = new ArrayList<>(); // An arraylist to store all admin object
ArrayList<Lecturer> lecturers = new ArrayList<>(); // An arraylist to store all lecturer objects
ArrayList<EvaluationResult> evaluationResults = new ArrayList<>(); // An arraylist to store all evaluation Results objects
```

Figure 6.1: Object oriented system using array list

Figure 6.1 shows the usage of array list to store data imported from the text file. The usage of array list to store objects allows the system to be run-on object-oriented system and does not need to rely on reading data from the file. Since all data manipulation is reflected on the arraylist and not the text file itself, user is able to write into text file whenever they like.

```
public class FileIO {

    // static global method
    public static ArrayList<Student> ImportStudents() {...23 lines }

    public static ArrayList<Project> ImportProjects() {...23 lines }

    public static ArrayList<Lecturer> ImportLecturer() {...23 lines }

    public static ArrayList<Admin> ImportAdmin() {...23 lines }

    public static ArrayList<EvaluationResult> ImportEvaluationResult() {...23 lines }

    public static void ExportStudents(ArrayList<Student> students) {...25 lines }

    public static void ExportProjects(ArrayList<Project> projects) {...25 lines }

    public static void ExportLecturer(ArrayList<Lecturer> lecturers) {...25 lines }

    public static void ExportAdmin(ArrayList<Admin> admins) {...25 lines }

    public static void ExportEvaluationResult(ArrayList<EvaluationResult> evaluationResults) {...25 lines }

}
```

Figure 6.2: Self defined FileIO class

Figure 6.2 shows the codes for a self-defined class named FileIO. This class includes multiple static global methods that can be call anywhere in the project. The methods in the class include Import and Export of all the text file so user is able to read and write data into the text files with ease.

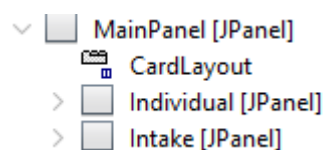


Figure 6.3: Card Layout J Panel arrangement

Figure 6.3 shows the usage of card layout for multiple J panel under a main panel. This allows the program to dynamically shows the preferred panel using any event listener.

Student ID	Student Name	Intake
TP07	Quary	2410
TP08	Pias	2406
TP09	Mono	2402

Figure 6.4: First Card for card layout

Figure 6.4 shows the first card set to the card layout. The interface shows the allot student system, when the individual button is pressed, the program will switch to this interface and when intake button is pressed instead the program will switch to the second card.

Student ID	Student Name	Intake
TP07	Quary	2410
TP08	Pias	2406
TP09	Mono	2402

Figure 6.5 Second Card for card layout

Figure 6.5 shows the second card set to the card layout. The interface shows the interface when the intake button is pressed, and the second card J panel is shown under the same window.

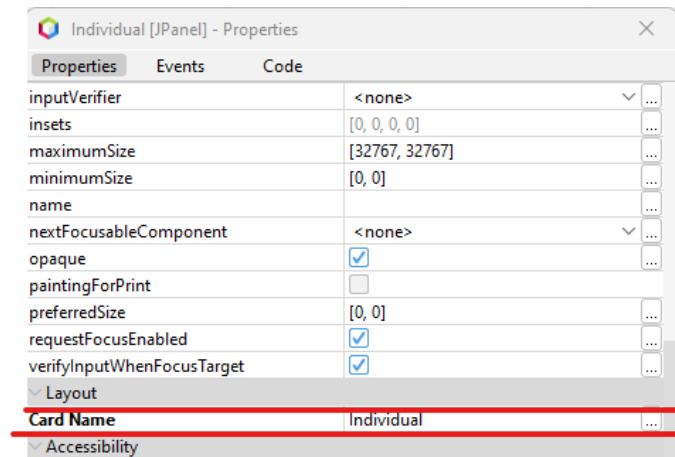


Figure 6.6: Properties setting for card name

Figure 6.6 shows the properties interface to set the card name. Using the card name settings, each of the J panel can be instantiate using different event

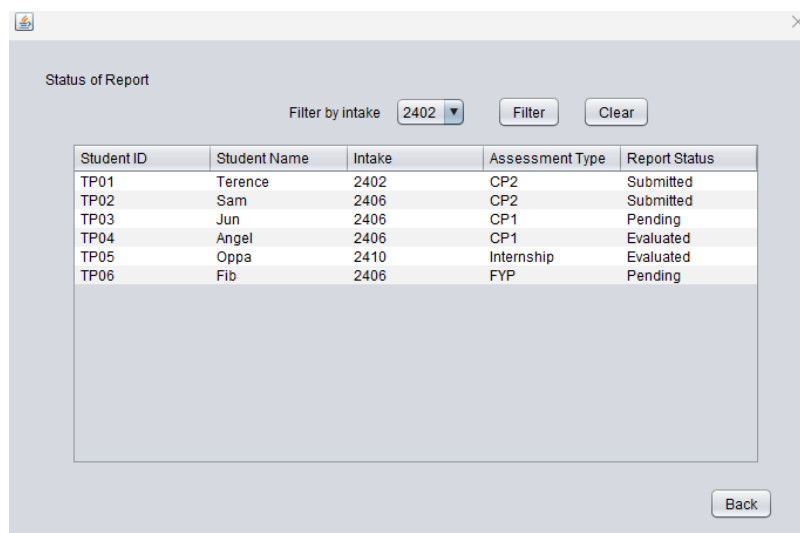


Figure 6.7: Filtering table functionality

Figure 6.7 shows a table before filtering in the project manager interface. The filter button can be used to filter the table and show only the student with the selected intake.

The screenshot shows a web application window titled "Status of Report". At the top, there is a "Filter by intake" section with a dropdown menu set to "2406", a "Filter" button, and a "Clear" button. Below this is a table with five columns: "Student ID", "Student Name", "Intake", "Assessment Type", and "Report Status". The table contains four rows of data, all with an intake of 2406. Below the table is a large empty rectangular box. At the bottom right of the window is a "Back" button.

Student ID	Student Name	Intake	Assessment Type	Report Status
TP02	Sam	2406	CP2	Submitted
TP03	Jun	2406	CP1	Pending
TP04	Angel	2406	CP1	Evaluated
TP06	Fib	2406	FYP	Pending

Figure 6.8: Student filtered by 2406 intake

Figure 6.8 shows the table with filtered intake. Only student that are 2406 intake will be shown in the table.

7.0 Limitation

Overall, The Project Management System has successfully met all the initial requirements. These include enabling project managers to assign projects and lecturer, lecturer to assign tasks, students to submit reports, check grades, and request presentation date, and administrators to manage academic projects effectively. The system has facilitated all the required processes. However, there is still room for improvement to further enhance the user experience for both student and staff.

One of the limitation is the system must store and retrieve data using text files. This may restrict the system's ability to scale and operate efficiently, particularly when handling massive volumes of data. Complex queries, transactions, and multi-user access cannot be supported by text files as effectively as they can be by database management systems.

Next, Using text files to manage data integrity and security can be difficult. Text files have weak procedures for guaranteeing data security and consistency and are more easily corrupted or altered. Compared to databases, reading from and writing to text files can be slower, especially as data sizes increase. This may influence the system's performance and reduce its responsiveness.

Other than that, Keeping text file backups on a regular basis can be a laborious and error-prone manual task. Compared to database system solutions, automated backup solutions are less simple. In the event of corruption or unintentional erasure, retrieving data from text files might be more difficult and less dependable.

7.1 Conclusion

The primary objective of the Project Management System (PMS) is to optimise the process of student registration, project allocation, and assessment within the Academic Guidance Hub (AGH). The system aims to improve the productivity of project management duties by implementing Object-Oriented Programming (OOP) principles in Java, resulting in a robust solution.

Although the suggested system has various benefits, such as enhancing organisation and minimising human labour, it also encounters specific constraints principally because it depends on text file storage. These limits encompass possible constraints on performance, issues about the accuracy and protection of data, and obstacles in expanding and facilitating simultaneous access.

While there are some drawbacks, the system offers a strong basis for properly organising academic projects. By using Object-Oriented Programming (OOP) concepts, the code is structured in a way that promotes modularity, reusability, and maintainability. This approach facilitates the ability to modify and adapt the code in the future.

To address the constraints that have been highlighted, potential future enhancements could include implementing a more resilient data storage system, improving the user interface, and integrating mechanisms to enhance data handling and security. In general, the PMS is a notable advancement in automating and enhancing the project management procedures at AGH.

8.0 References

Cloudmersive. (n.d.). *Why text files are unsafe for file upload*. Cloudmersive.

<https://cloudmersive.com/article/Why-Text-Files-Are-Unsafe-for-File-Upload>

Shaik, S. P. (2023, August 30). *Academic student course registration system (skill development project-9)*. LinkedIn. <https://www.linkedin.com/pulse/academic-student-course-registration-system-skill-project-9-shaik/>

The University of Auckland 5.2 Plain text formats. (n.d.).

<https://www.stat.auckland.ac.nz/~paul/ItDT/HTML/node38.html>

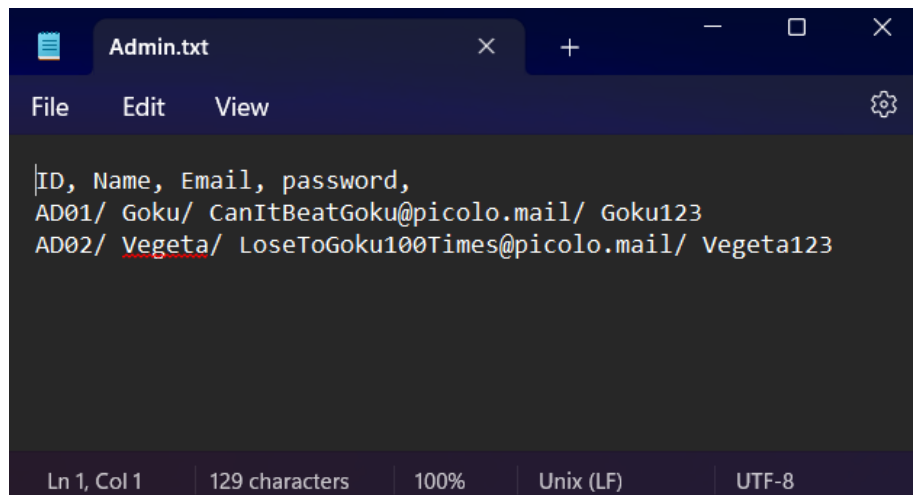
What are the advantages of Text Files? | Lenovo US. (n.d.).

<https://www.lenovo.com/us/en/glossary/text->

[file/?orgRef=https%253A%252F%252Fwww.google.com%252F](https://www.google.com/?orgRef=https%253A%252F%252Fwww.google.com%252F)

9.0 Appendix

Admin.txt File

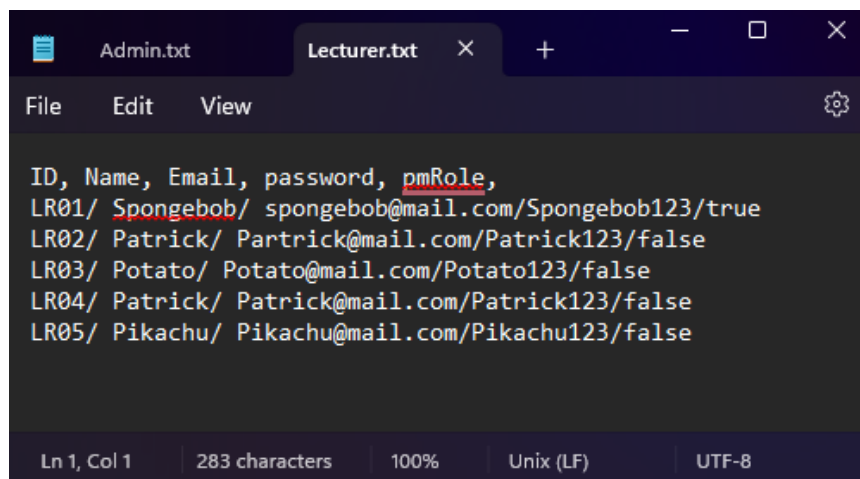


A screenshot of a code editor window titled 'Admin.txt'. The editor has a dark theme and a menu bar with 'File', 'Edit', and 'View'. The text content is as follows:

```
ID, Name, Email, password,  
AD01/ Goku/ CanItBeatGoku@picolo.mail/ Goku123  
AD02/ Vegeta/ LoseToGoku100Times@picolo.mail/ Vegeta123
```

The status bar at the bottom shows 'Ln 1, Col 1', '129 characters', '100%', 'Unix (LF)', and 'UTF-8'.

Lecutrer.txt File

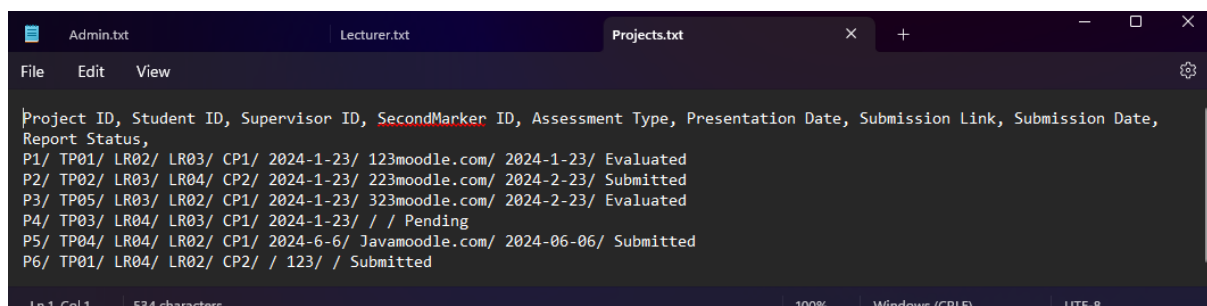


A screenshot of a code editor window titled 'Lecturer.txt'. The editor has a dark theme and a menu bar with 'File', 'Edit', and 'View'. The text content is as follows:

```
ID, Name, Email, password, pmRole,  
LR01/ Spongebob/ spongebob@mail.com/Spongebob123/true  
LR02/ Patrick/ Partrick@mail.com/Patrick123/false  
LR03/ Potato/ Potato@mail.com/Potato123/false  
LR04/ Patrick/ Patrick@mail.com/Patrick123/false  
LR05/ Pikachu/ Pikachu@mail.com/Pikachu123/false
```

The status bar at the bottom shows 'Ln 1, Col 1', '283 characters', '100%', 'Unix (LF)', and 'UTF-8'.

Project.txt File

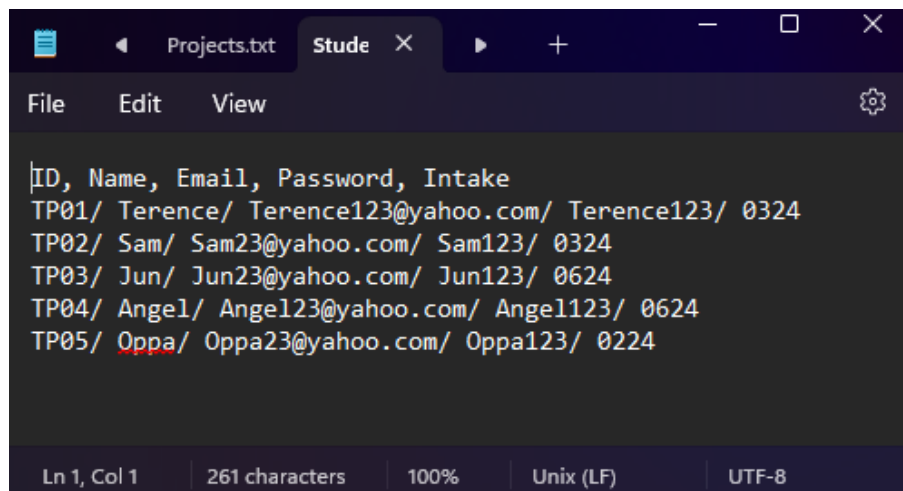


A screenshot of a code editor window titled 'Project.txt'. The editor has a dark theme and a menu bar with 'File', 'Edit', and 'View'. The text content is as follows:

```
Project ID, Student ID, Supervisor ID, SecondMarker ID, Assessment Type, Presentation Date, Submission Link, Submission Date,  
Report Status,  
P1/ TP01/ LR02/ LR03/ CP1/ 2024-1-23/ 123moodle.com/ 2024-1-23/ Evaluated  
P2/ TP02/ LR03/ LR04/ CP2/ 2024-1-23/ 223moodle.com/ 2024-2-23/ Submitted  
P3/ TP05/ LR03/ LR02/ CP1/ 2024-1-23/ 323moodle.com/ 2024-2-23/ Evaluated  
P4/ TP03/ LR04/ LR03/ CP1/ 2024-1-23/ / / Pending  
P5/ TP04/ LR04/ LR02/ CP1/ 2024-6-6/ Javamoodle.com/ 2024-06-06/ Submitted  
P6/ TP01/ LR04/ LR02/ CP2/ / 123/ / Submitted
```

The status bar at the bottom shows 'Ln 1, Col 1', '534 characters', '100%', 'Windows (CRLF)', and 'UTF-8'.

StudentData.txt File






The image shows a code editor window with a dark theme. The title bar at the top contains a file icon, the name 'Projects.txt', and a tab labeled 'Stude' with a close button. Below the title bar is a menu bar with 'File', 'Edit', and 'View' options, and a settings gear icon on the right. The main text area contains a CSV file with the following content:

```
ID, Name, Email, Password, Intake
TP01/ Terence/ Terence123@yahoo.com/ Terence123/ 0324
TP02/ Sam/ Sam23@yahoo.com/ Sam123/ 0324
TP03/ Jun/ Jun23@yahoo.com/ Jun123/ 0624
TP04/ Angel/ Angel123@yahoo.com/ Angel123/ 0624
TP05/ Oppa/ Oppa23@yahoo.com/ Oppa123/ 0224
```

The text 'Oppa' in the last row is highlighted in red. At the bottom of the window, a status bar displays the following information: 'Ln 1, Col 1', '261 characters', '100%', 'Unix (LF)', and 'UTF-8'.

WORKLOAD MATRIX

Student Name	Task	Signature
Terence Lim Dao Liang	Lecturer part, Lecture Use Case diagram and Compiler	
Tay Jun Long	Project manager part, Project manager Use Case diagram and Compiler	
Angelina Leanore	Student Part, Student Use Case Diagram and Introduction	
Eraliev Suimonkul	Admin part, Admin Use Case diagram and Compiler	