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

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

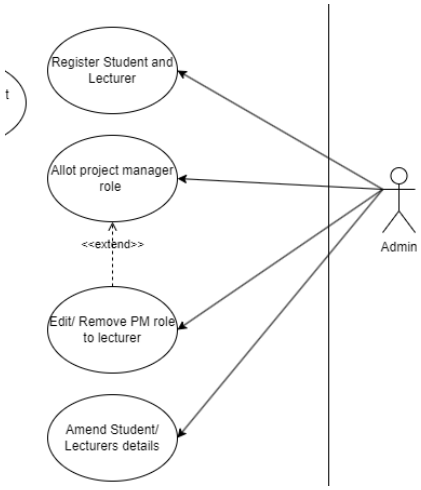


Figure 2.1: Admin use case diagram

2.2 Project Manager

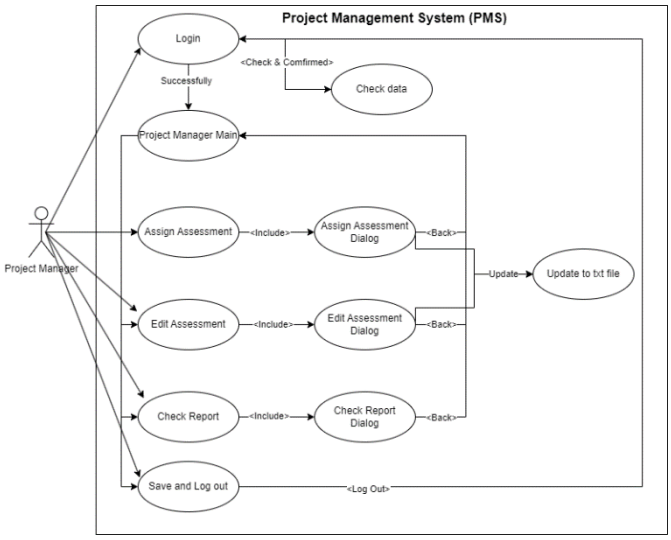


Figure 2.2: Use case diagram for project manager

2.3 Lecturer

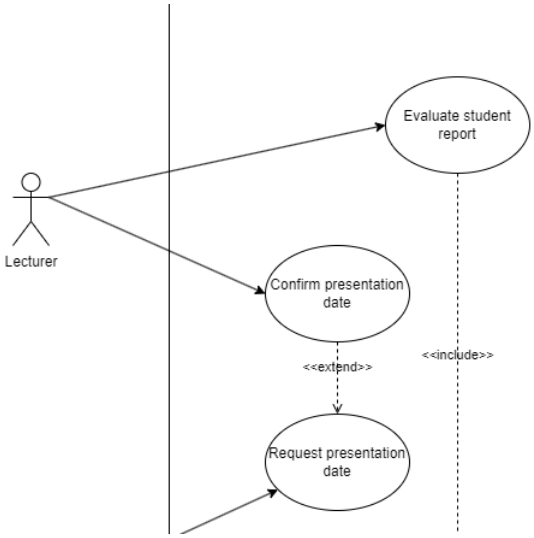


Figure 2.3: Use case diagram for lecturer

2.4 Student

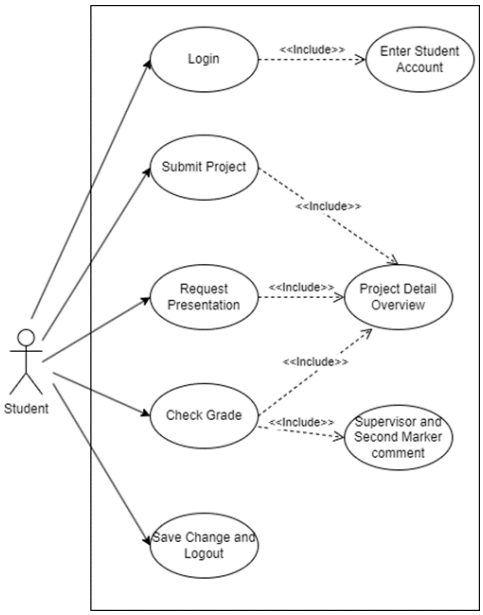


Figure 2.4: Use case diagram for student

3.0 Class Diagram

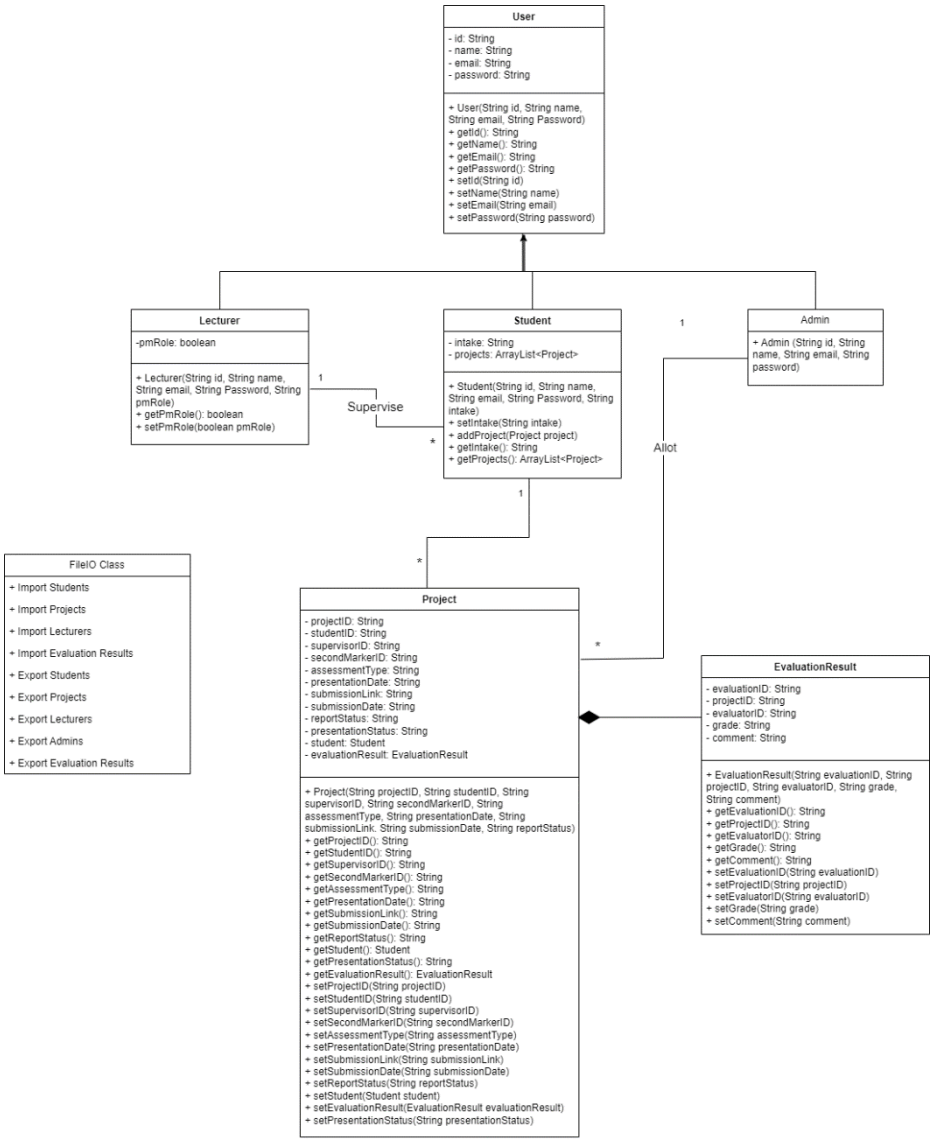


Figure 3.0: Class diagram for all classes

4.0 Sample Output

4.1 Admin

Figure 4.1.1: Login

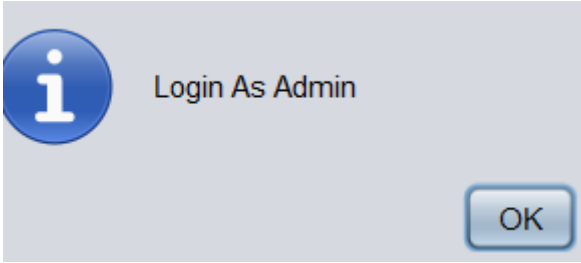


Figure 4.1.2: Successful login as admin

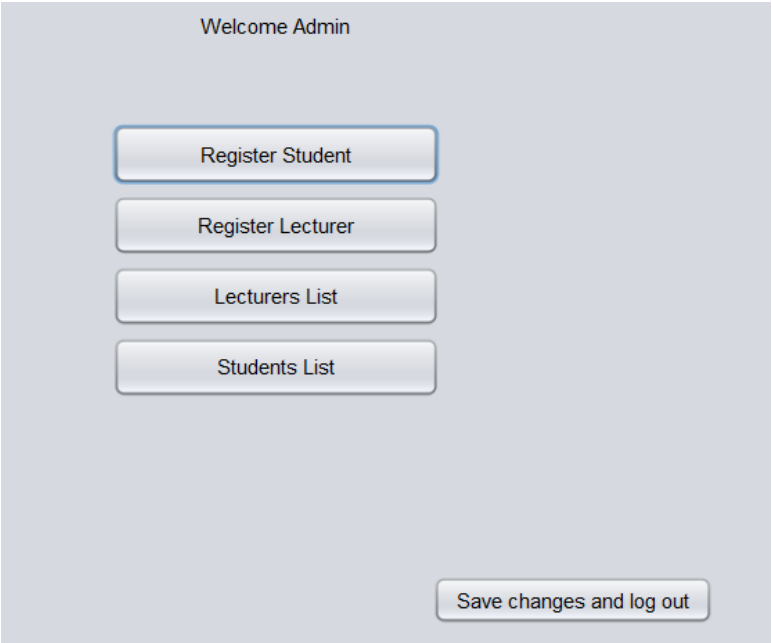


Figure 4.1.3: Admin Main Menu

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.

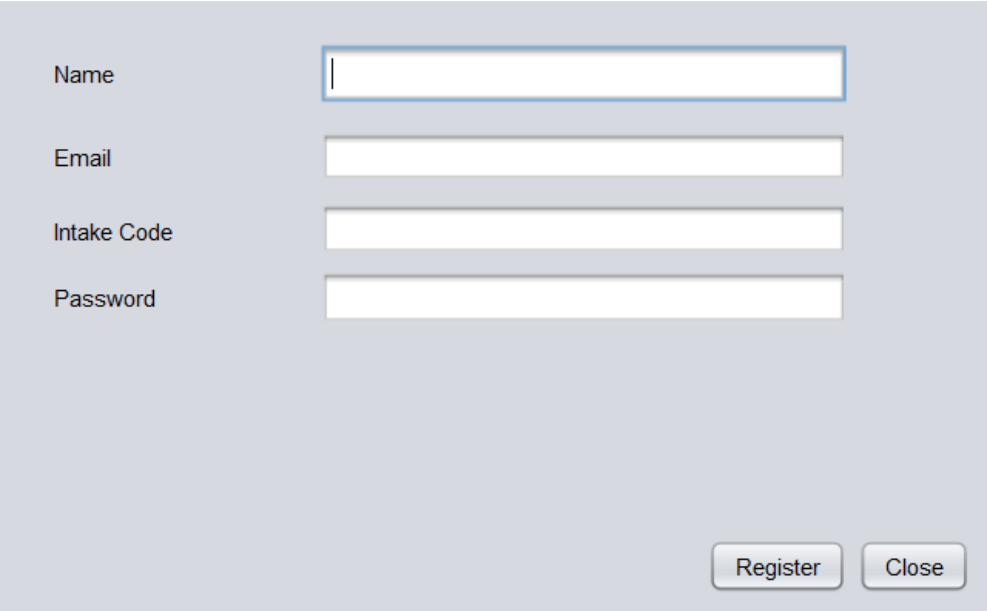


Figure 4.1.4: Register Student

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

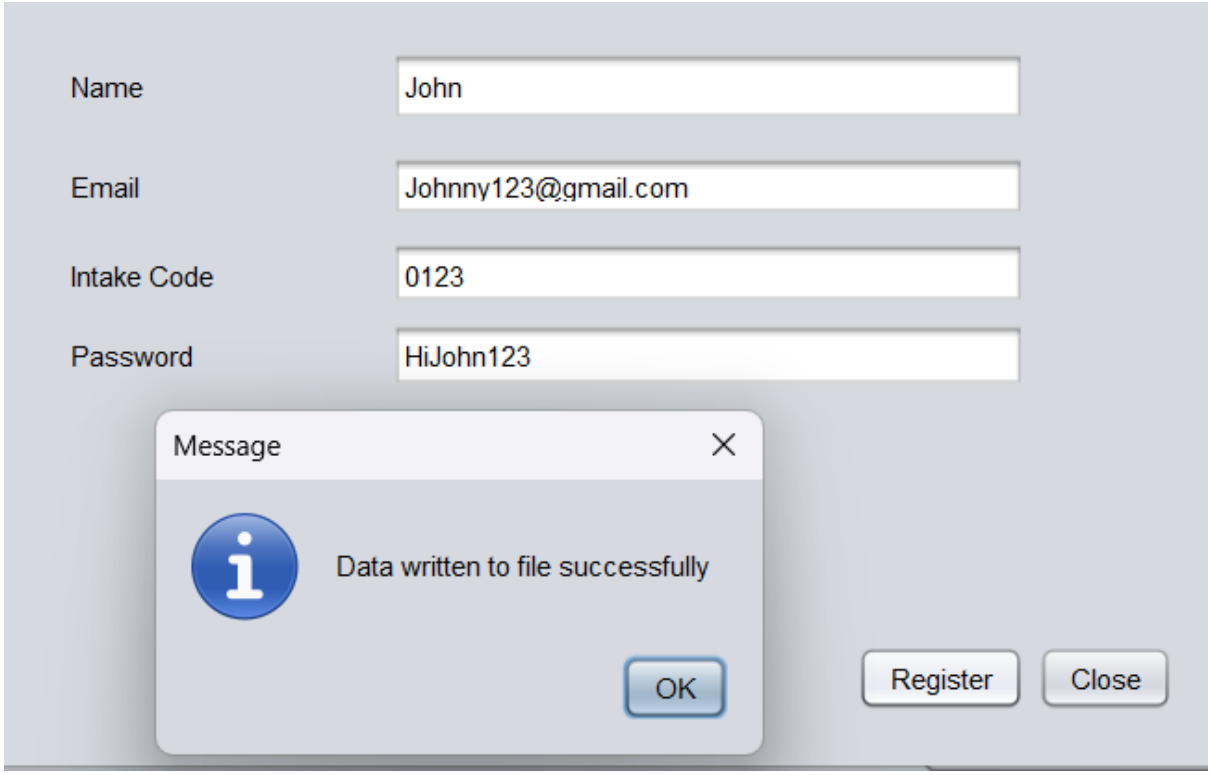


Figure 4.1.5: Successfully registered student

After registration a new student the information message will appear.

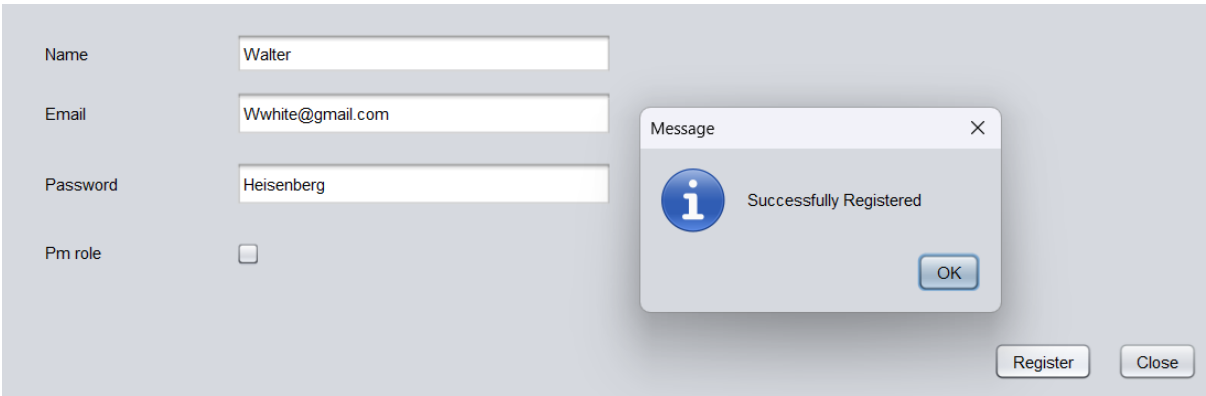


Figure 4.1.6: Register Lecturer with successfully created account

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

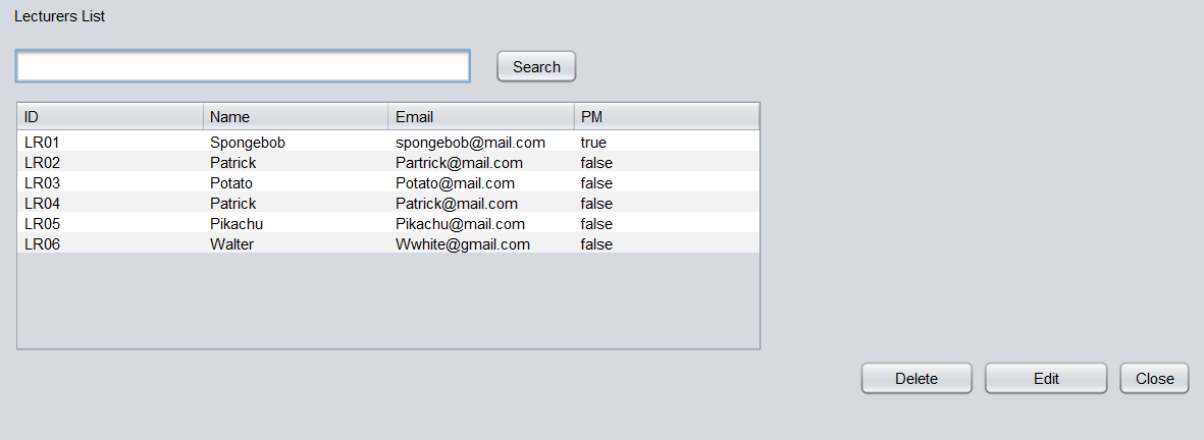


Figure 4.1.7: Lecturers list menu

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.

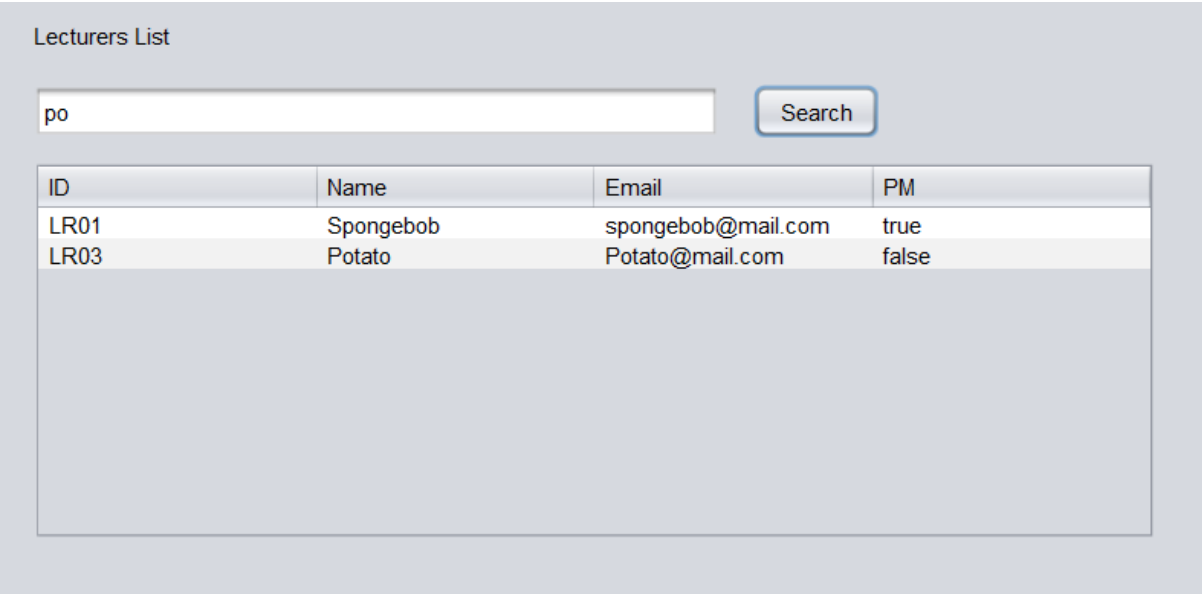


Figure 4.1.8: Searching function. Any matches between Searching Field and Lecturers

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.

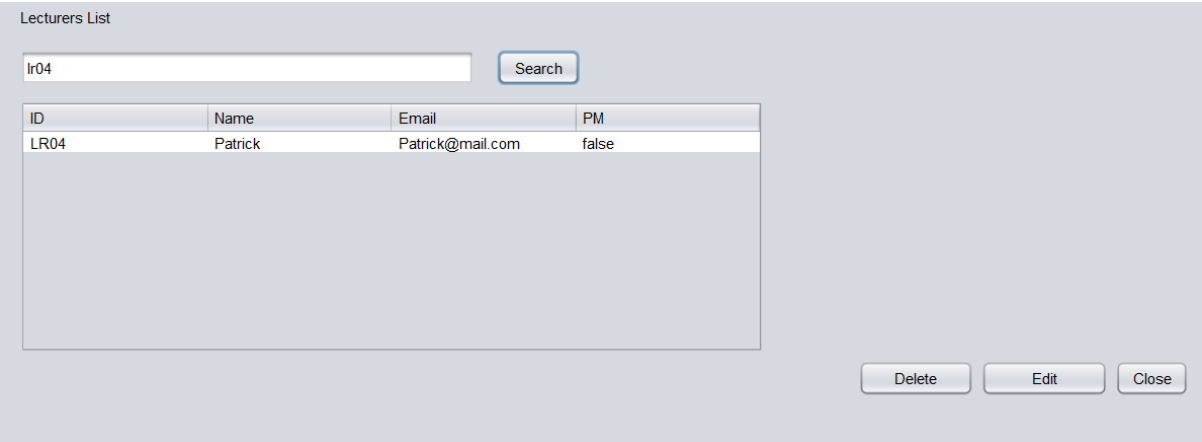


Figure 4.1.9: Search by ID

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

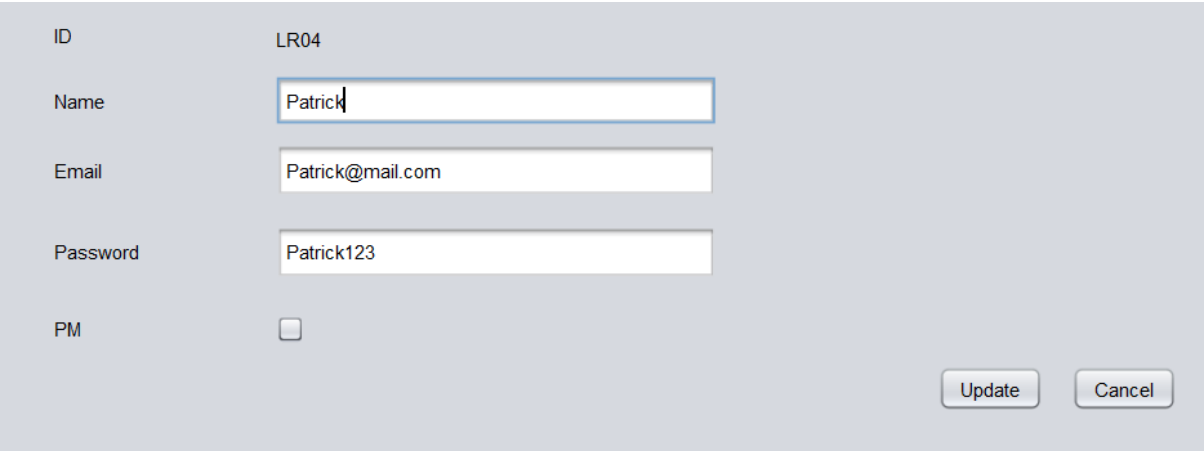


Figure 4.1.10: Edit lecturer menu

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.

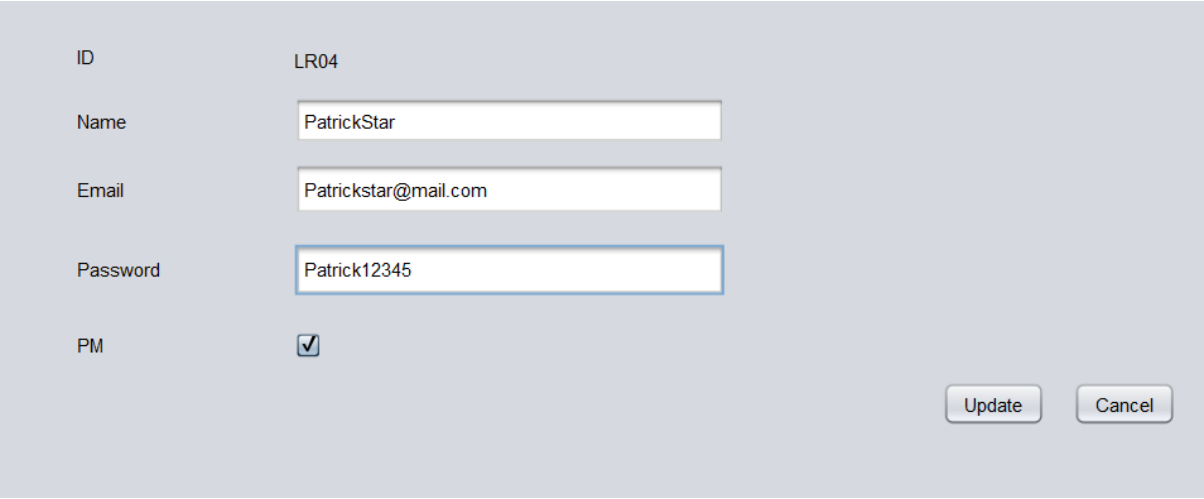


Figure 4.1.11: Edited data ready to update

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

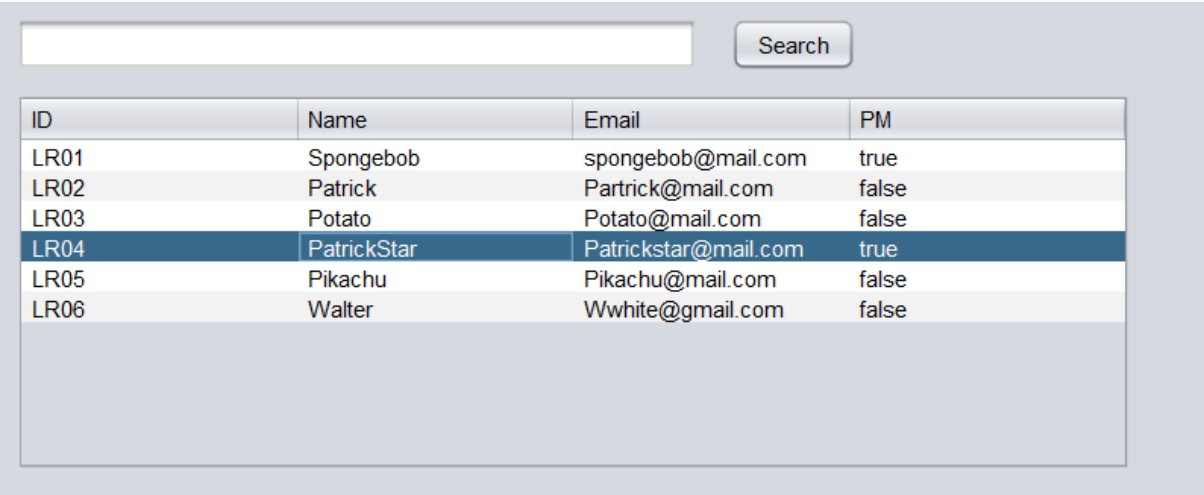


Figure 4.1.12: Updated Lecturers list

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

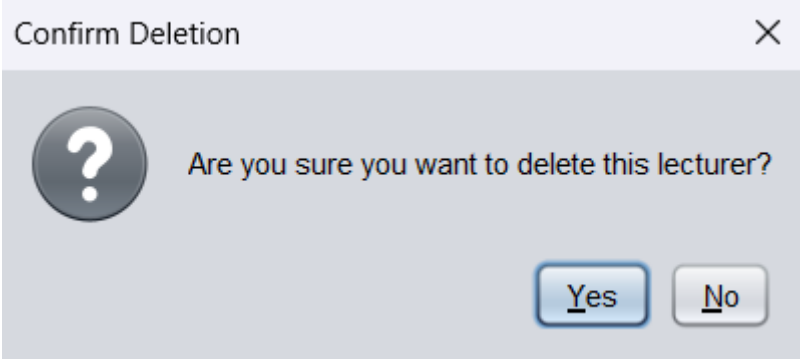


Figure 4.1.13: Delete Confirmation message

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

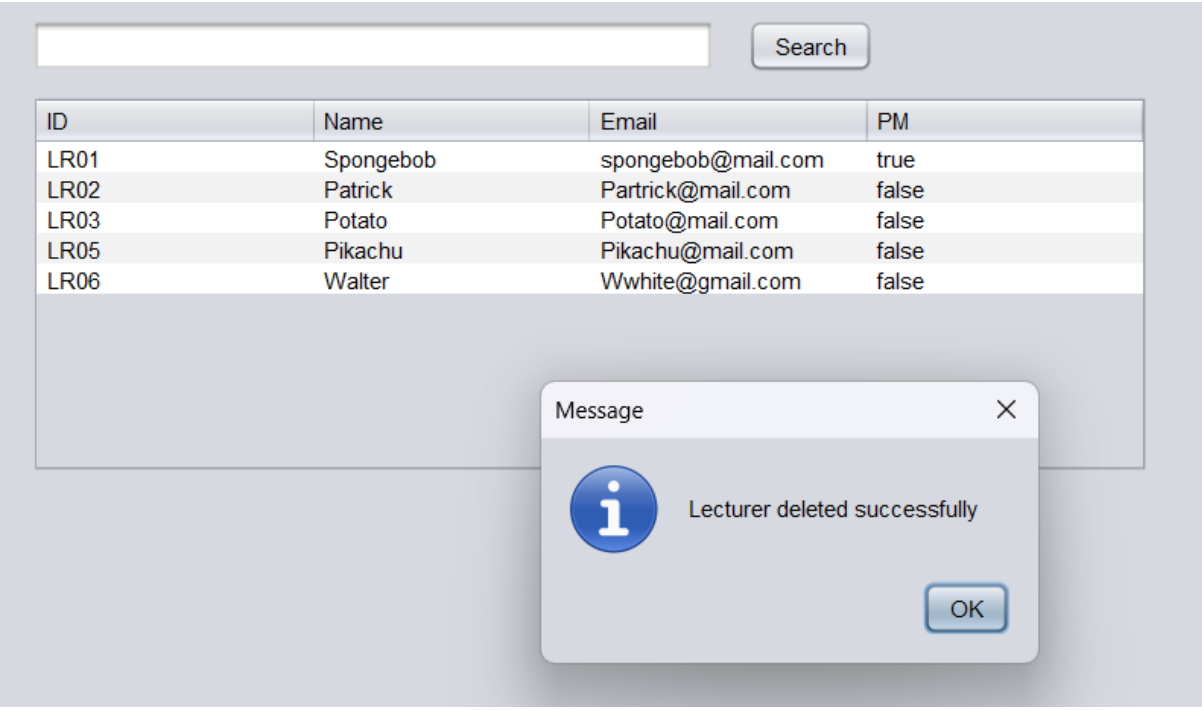


Figure 4.1.14: Updated Lecturers List without deleted account

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

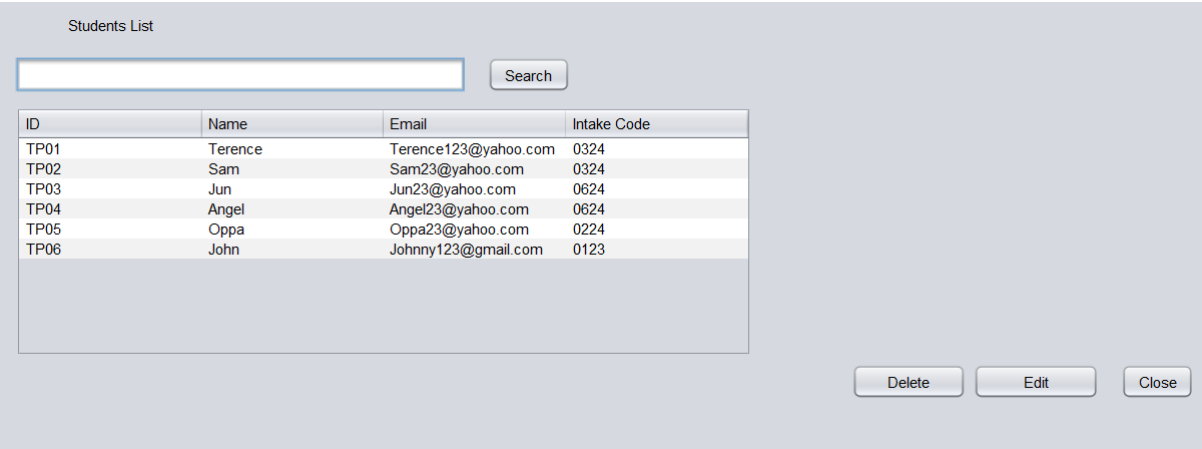


Figure 4.1.15: Students List menu

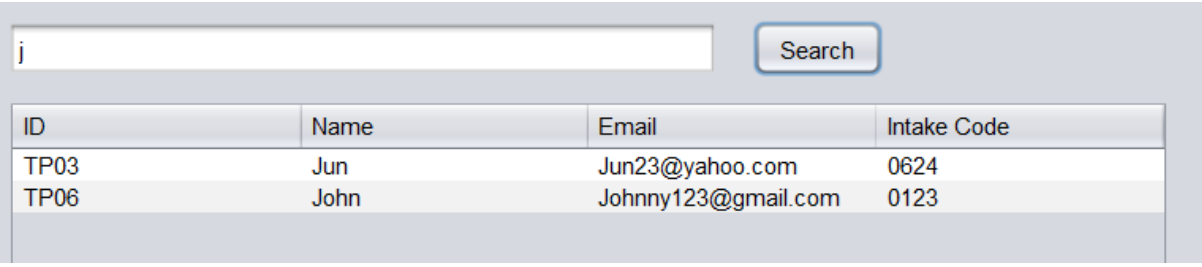


Figure 4.1.16: Search by a single character

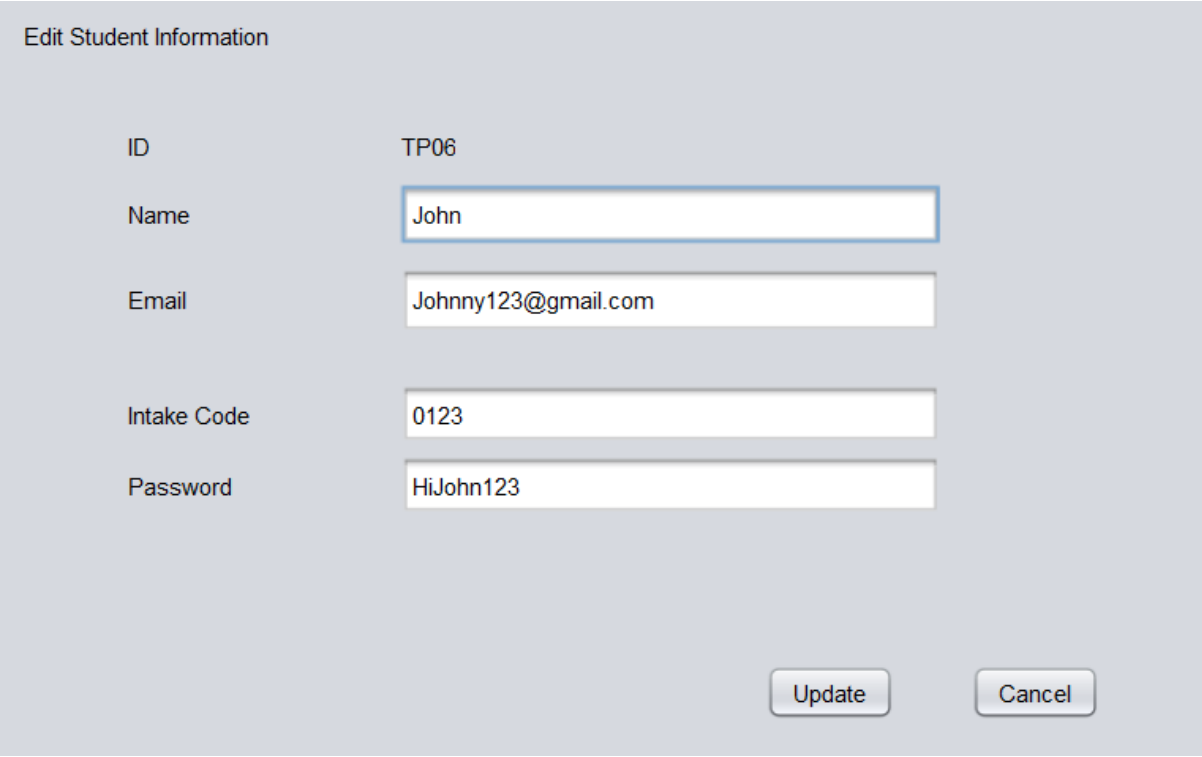


Figure 4.1.17: Student Edit Menu

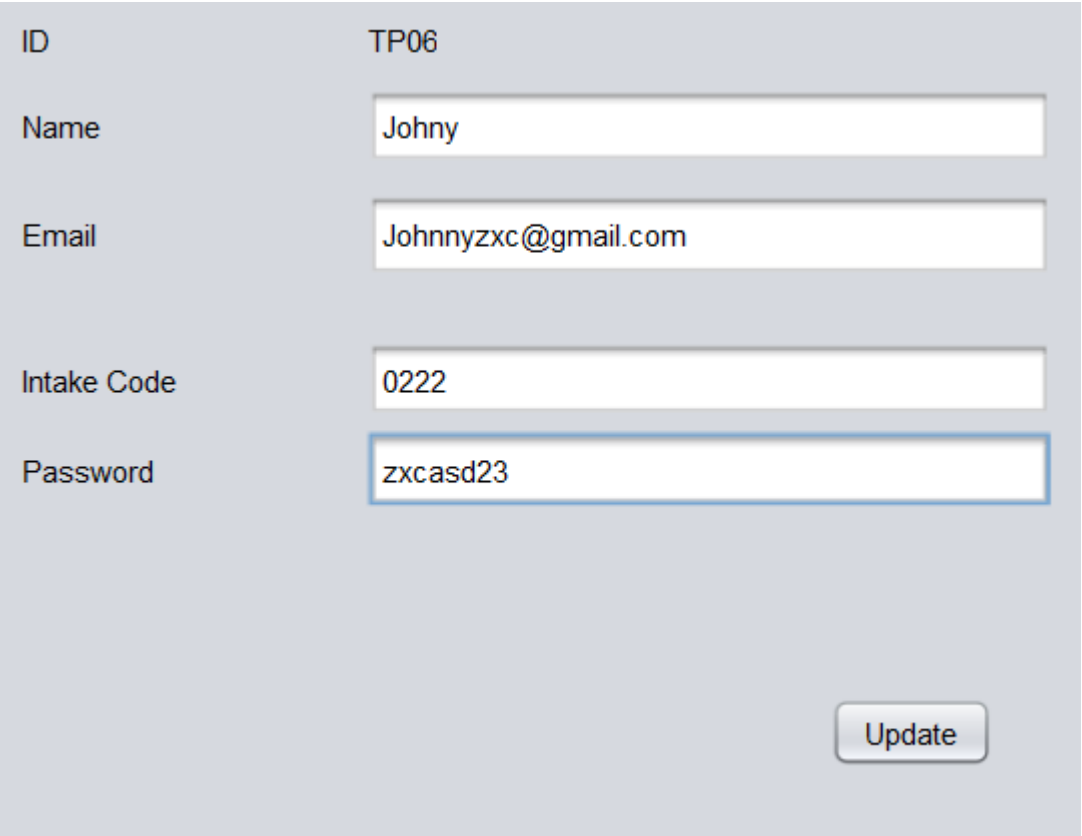


Figure 4.1.18: Editted data ready to update

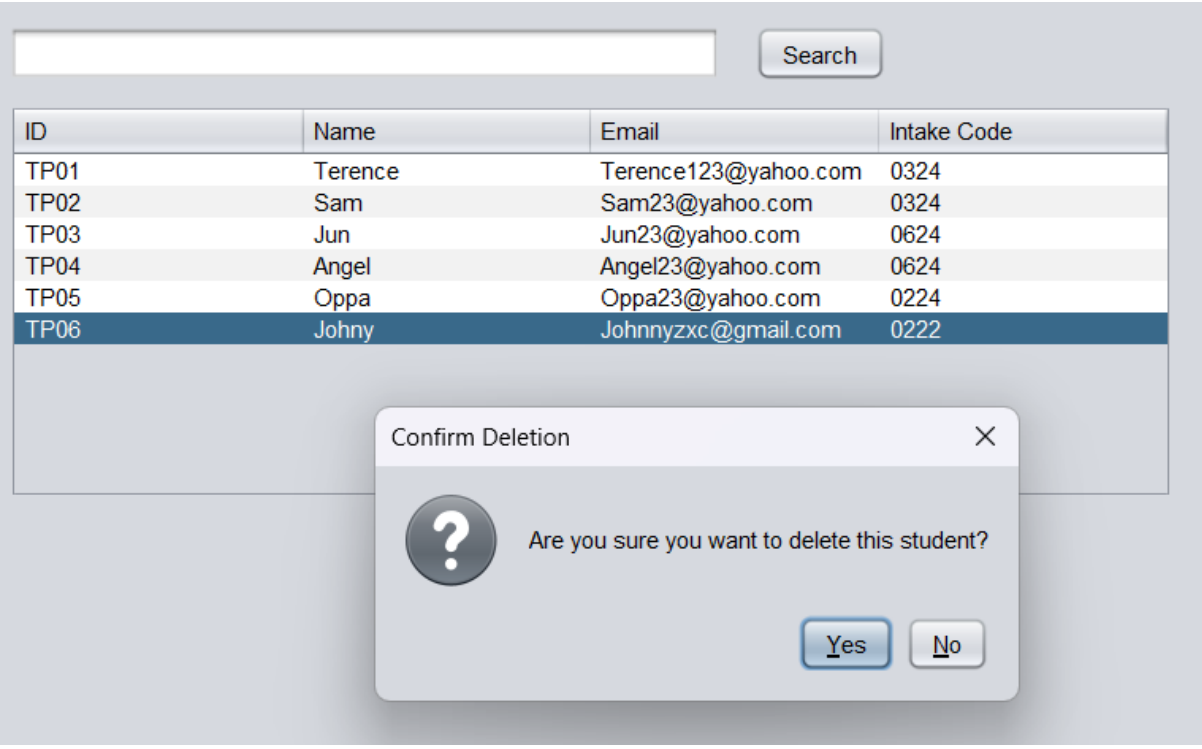


Figure 4.1.19: Updated table and Delete Confirmation Message

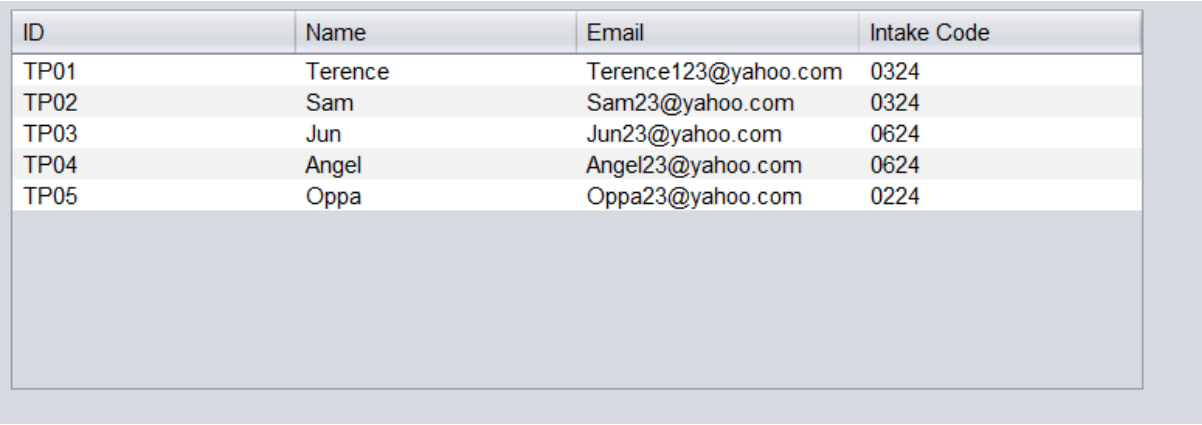


Figure 4.1.20: Updated table without deleted account

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

4.2 Project Manager

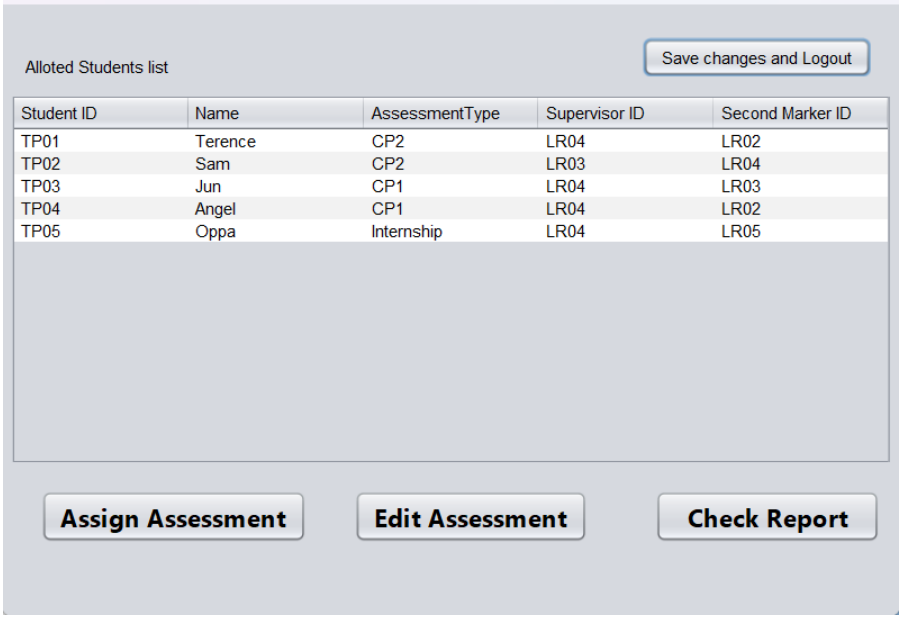


Figure 4.2.1: Project Manager Main

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.

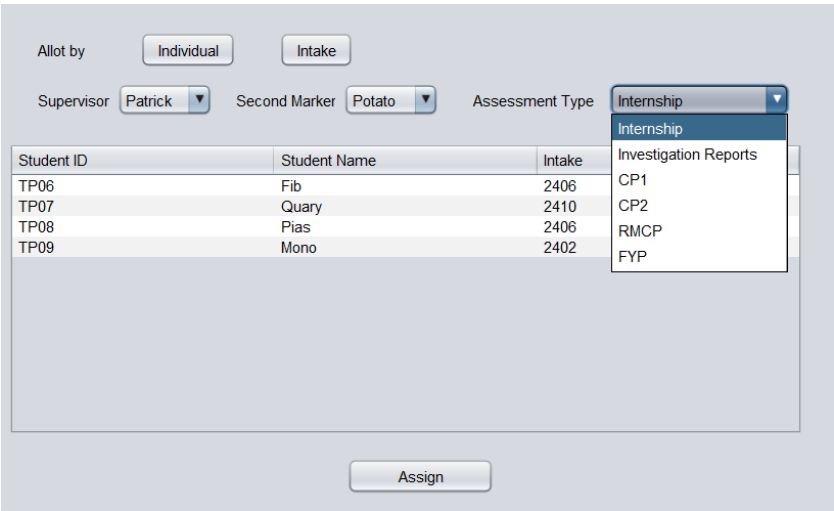


Figure 4.2.2: Assign Assessment

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.

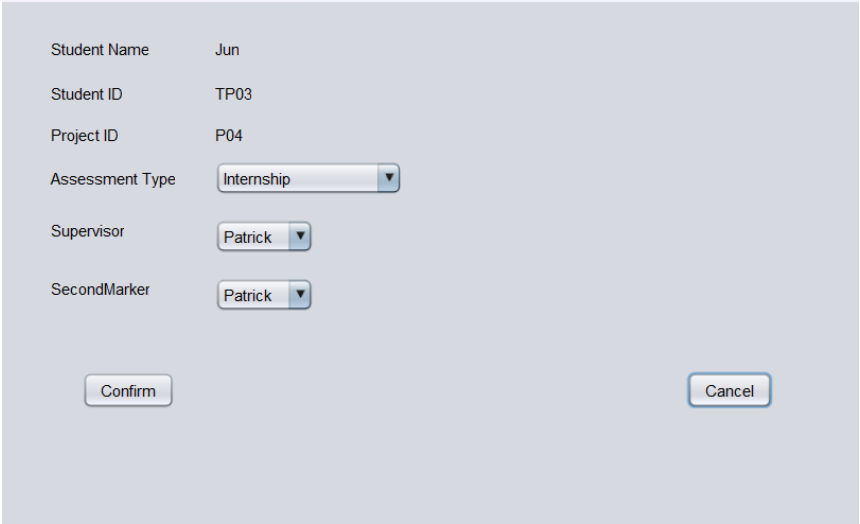


Figure 4.2.3: Edit Assessment

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.

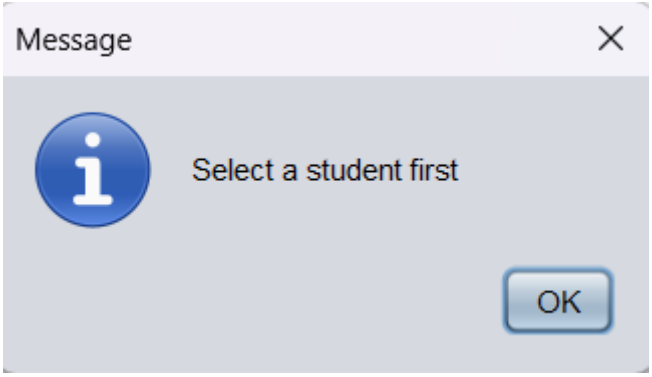


Figure 4.2.4: Error Handler

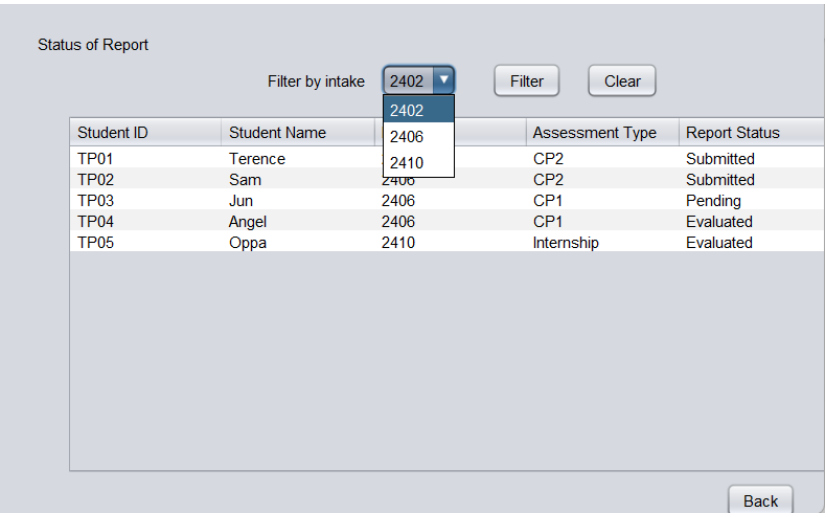


Figure 4.2.5: Check Report

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

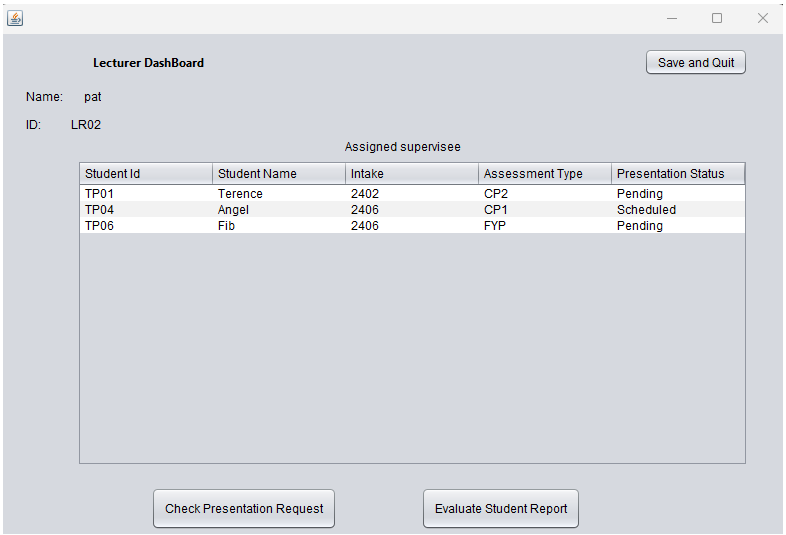


Figure 4.3.1: Dashboard for lecturer

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.

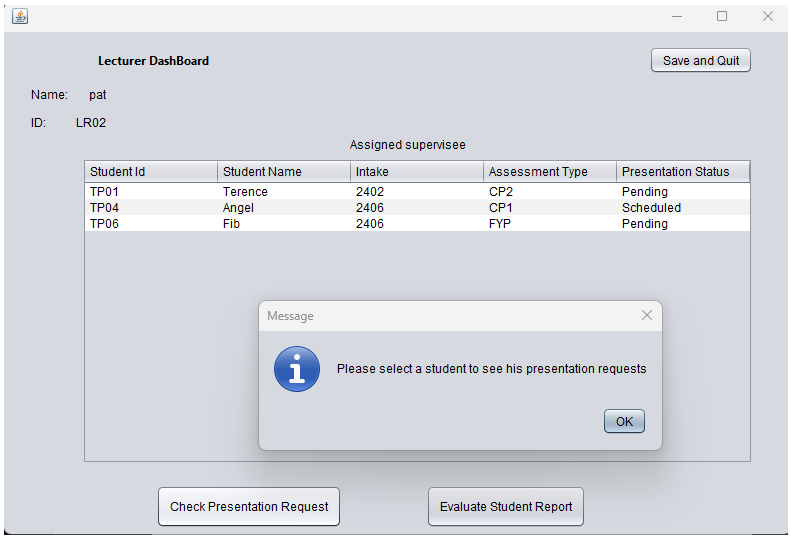


Figure 4.3.2: User does not select a row for checking presentation request

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.

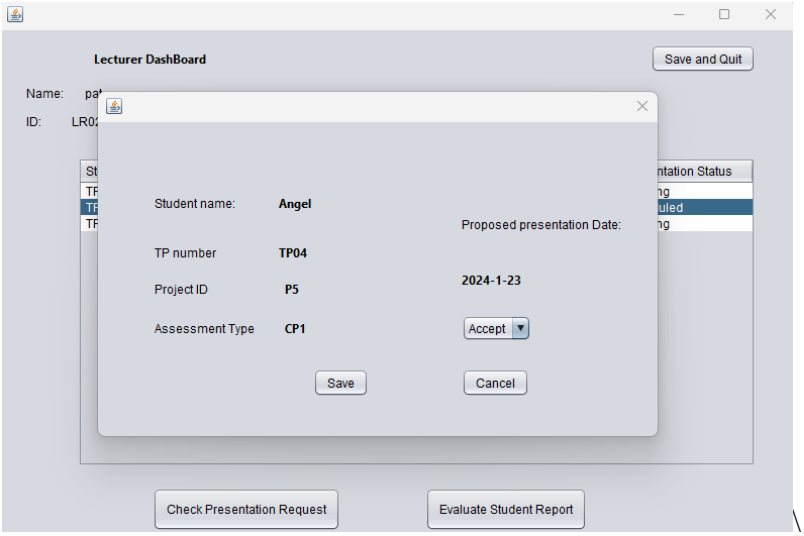


Figure 4.3.3: Check Presentation Interface

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

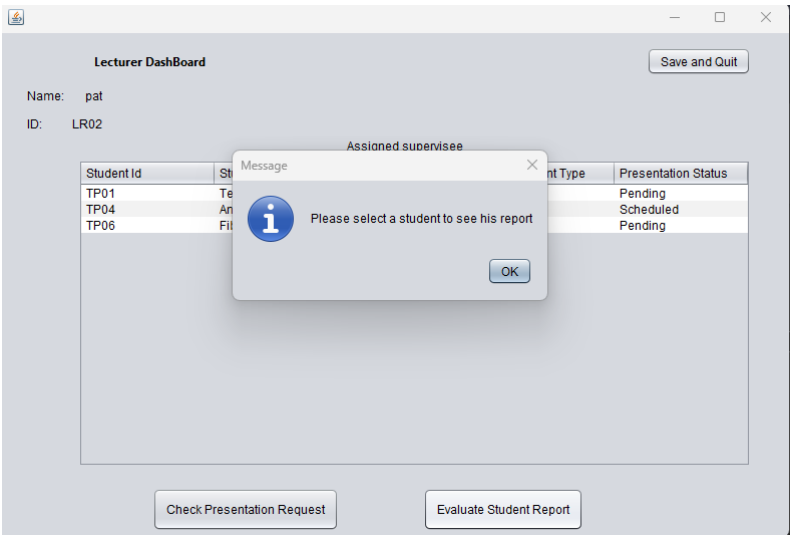


Figure 4.3.4: Error when user does not select a student to evaluate his report

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

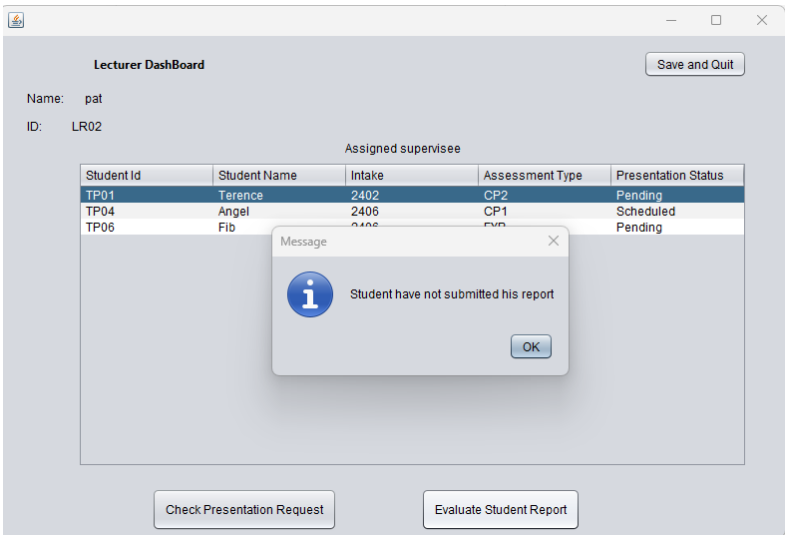


Figure 4.3.5: Error when user wanted to evaluate a student that have not submitted his report

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

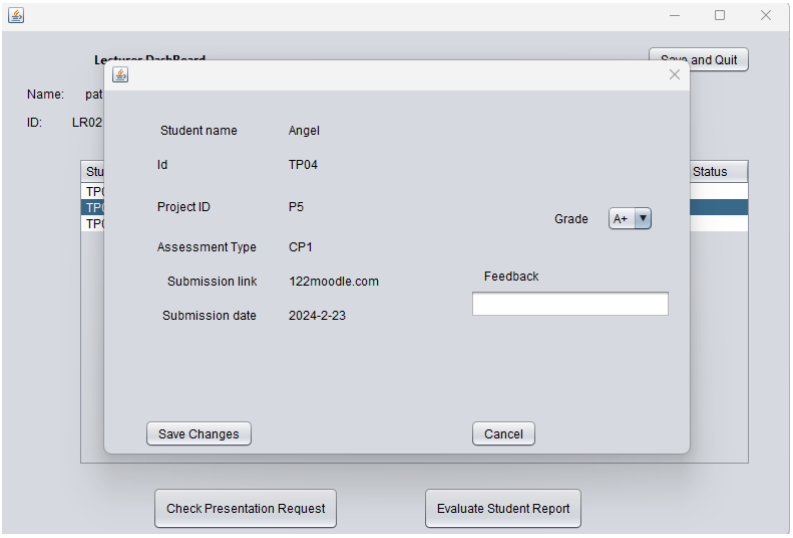


Figure 4.3.6: Evaluate student report Interface

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.

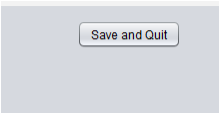


Figure 4.3.7: Save and quit button

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

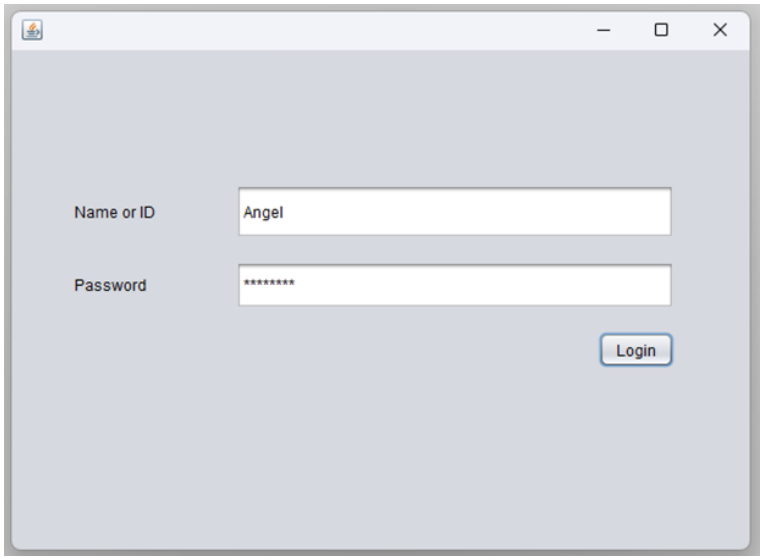


Figure 4.4.1: Login

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

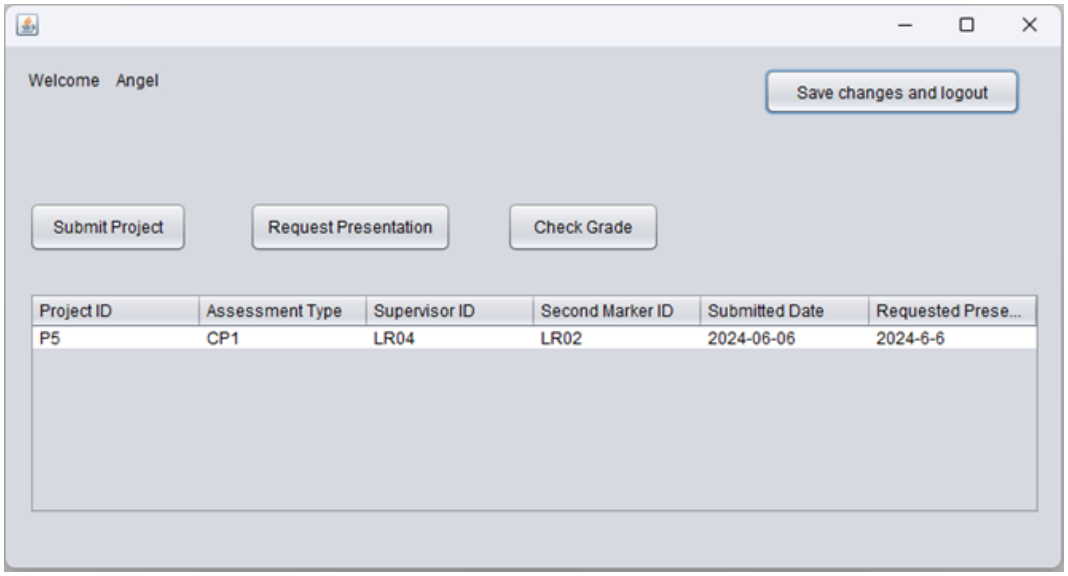


Figure 4.4.2: Student interface

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

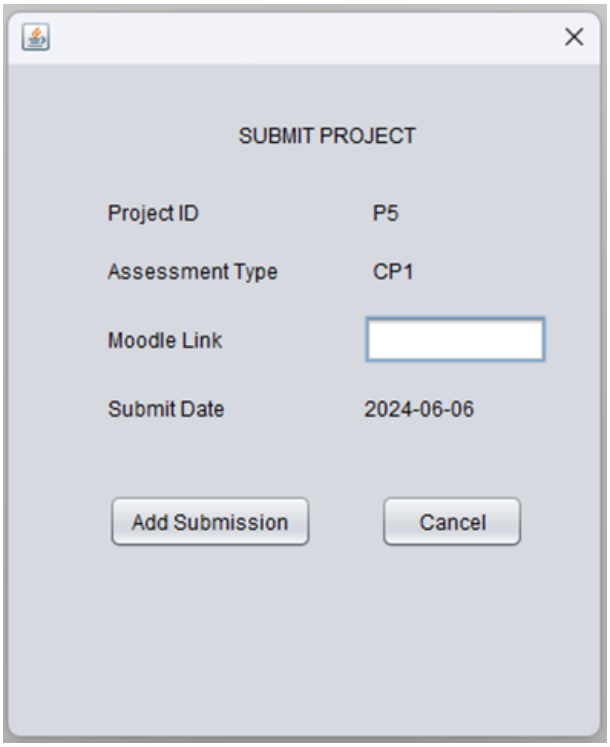


Figure 4.4.3: Submit project interface

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

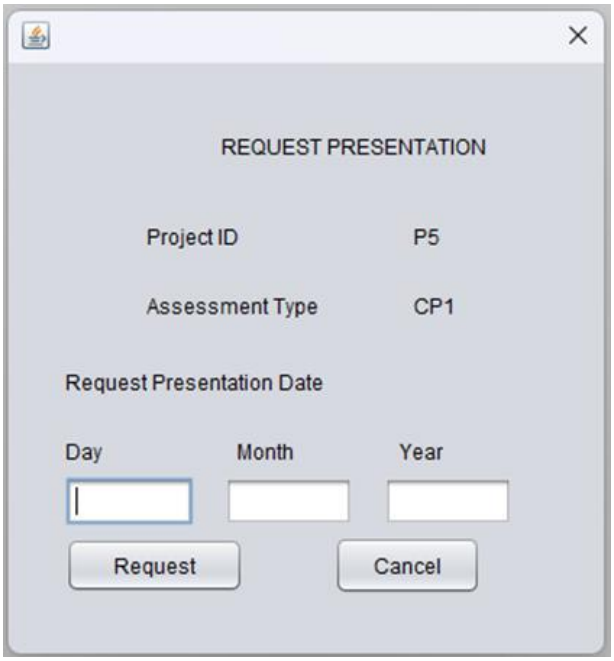


Figure 4.4.4: Request presentation interface

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

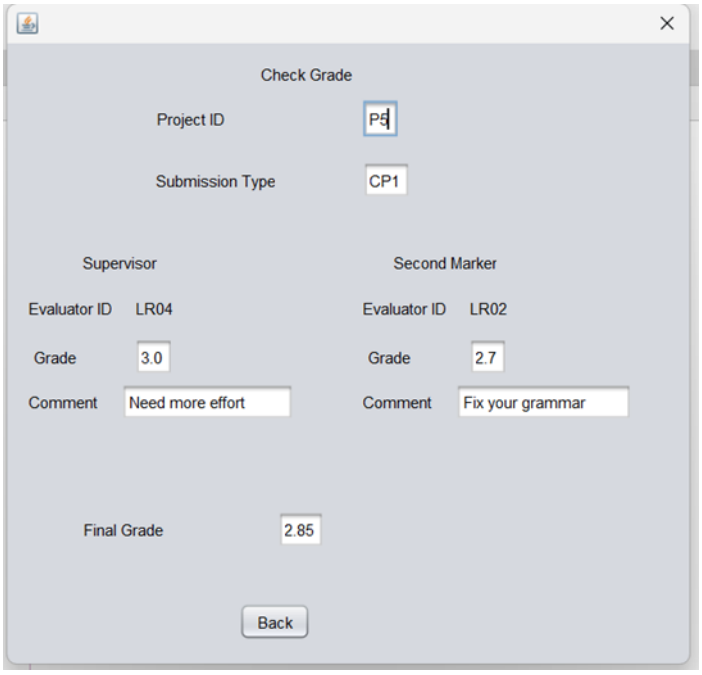


Figure 4.4.5: Check grade interface

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

```java
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

```java
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

```java
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() + "/ " + lecture
            writer.write(line);
            writer.newLine();
        } else if (data instanceof Student student) {
            String line = student.getID() + "/ " + student.getName() + "/ " + student.getEmail() + "/ " + student.ge
            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

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

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

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

    public Student(String id, String name, St
        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

```java
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.

```java
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

```java
public class Project {
    private String projectID, studentID, supervisorID, secondMarkerID, assessmentType, presentationDate, submissionL
    private Student student;
    private EvaluationResult evalutaionResult;

    public Project(String projectID,String studentID, String supervisorID,String secondMarkerID, String assessmentTy
        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.

```java
//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

```java
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 enrols in the hierarchy of superclass ‘User”.

```java
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 lastest project
        if (!student.getProjects().isEmpty()){
            Project latestProject = student.getProjects().get(student.getProjects().size()-1);
            String [] tableDataRow = {student.getID(), student.getName(), latestProject.getAssessmentType(), lates
            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

```java
    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"

```java
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

```java
// 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);
```

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.

```java
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

```java
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.

```java
public LecturerMain(Lecturer lecturer, ArrayList<
    this.loggedinLecterur = 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 5.3.4showcase 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

```java
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.

```java
Lecturer loggedinLecterur;
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

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

    public Lecturer(String id, String name, String email, Strin
        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

```java
// 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.

```java
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(), AssignedProjec
    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.

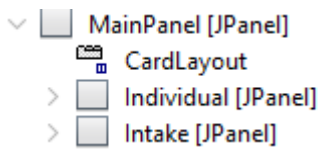


Figure 6.3: Card Layout J Panel arrangement

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 prefered panel using any event listener.

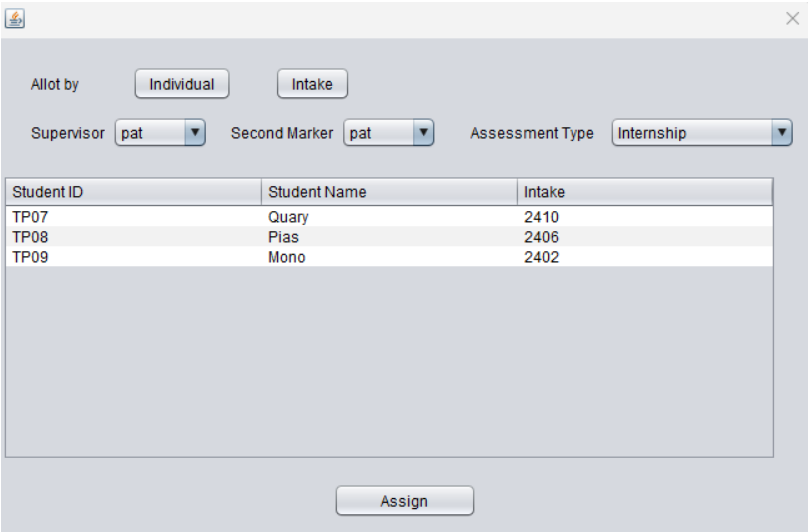


Figure 6.4: First Card for card layout

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.

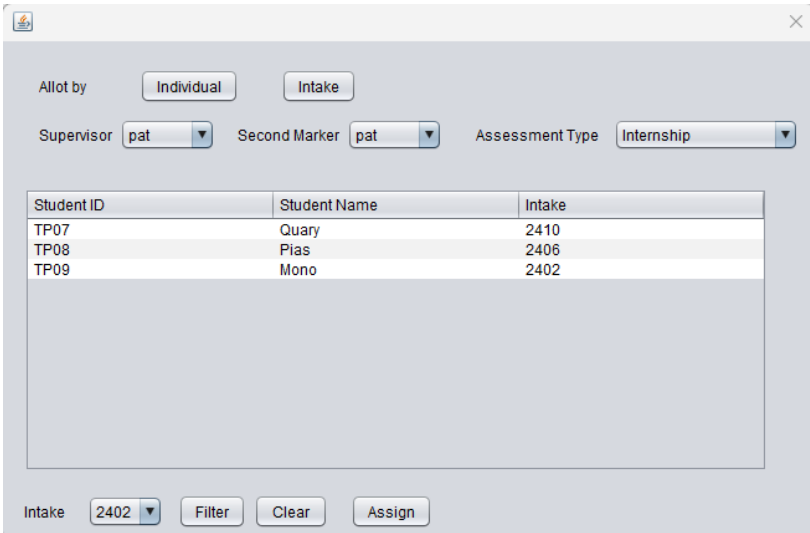


Figure 6.5 Second Card for card layout

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.

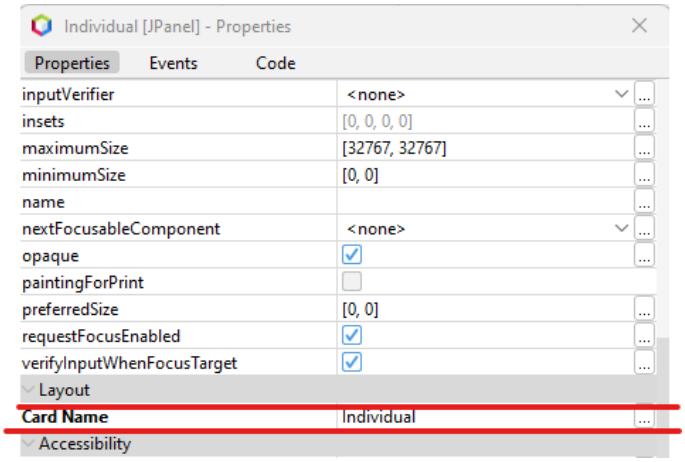


Figure 6.6: Properties setting for card name

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

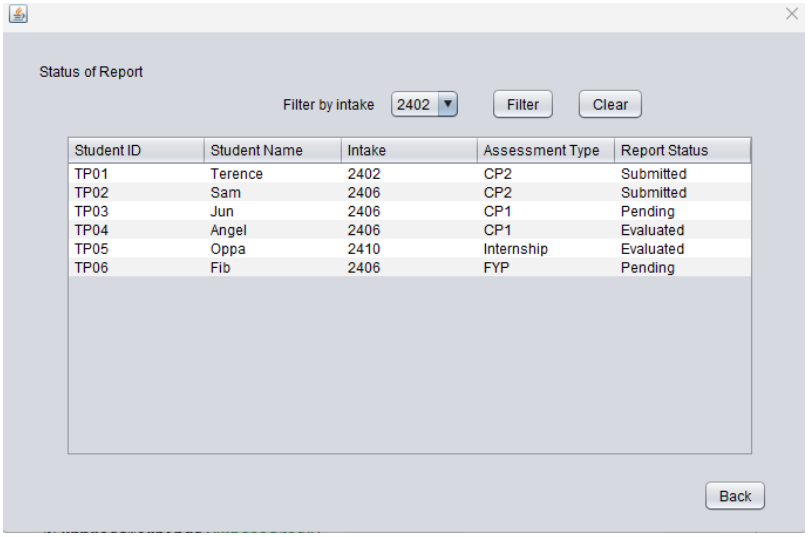


Figure 6.7: Filtering table functionality

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.

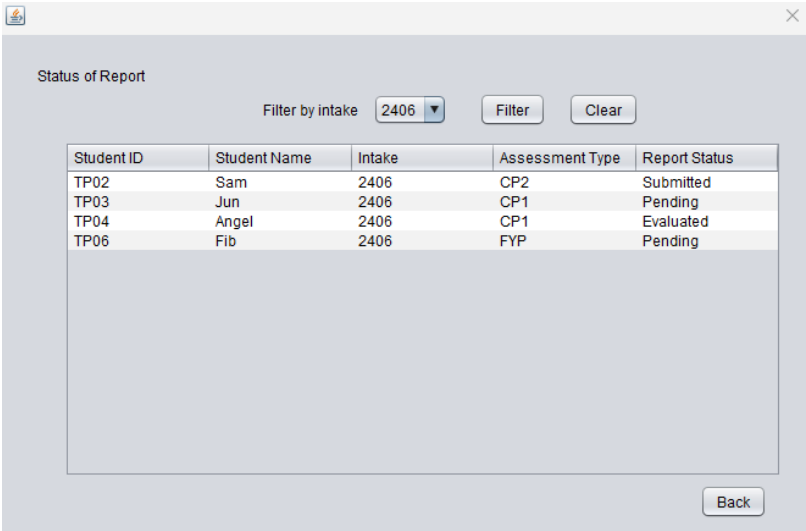


Figure 6.8: Student filtered by 2406 intake

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

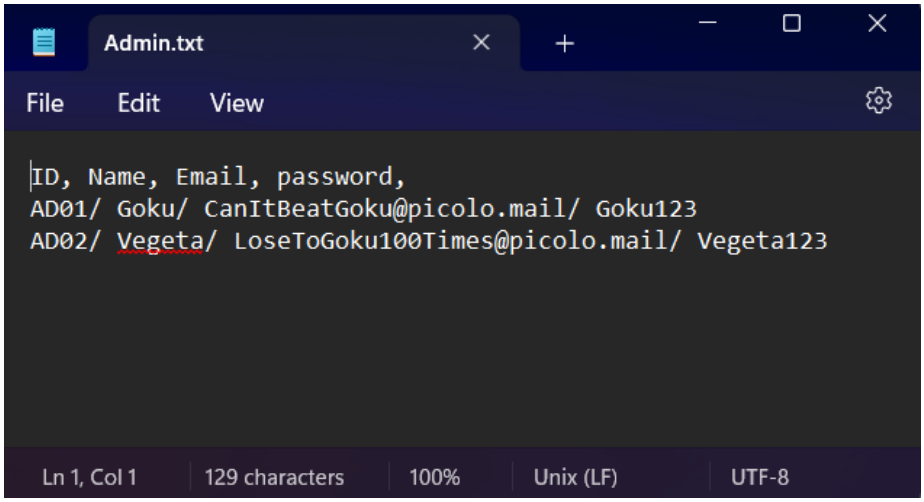
What are the advantages of Text Files? | Lenovo US. (n.d.).

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

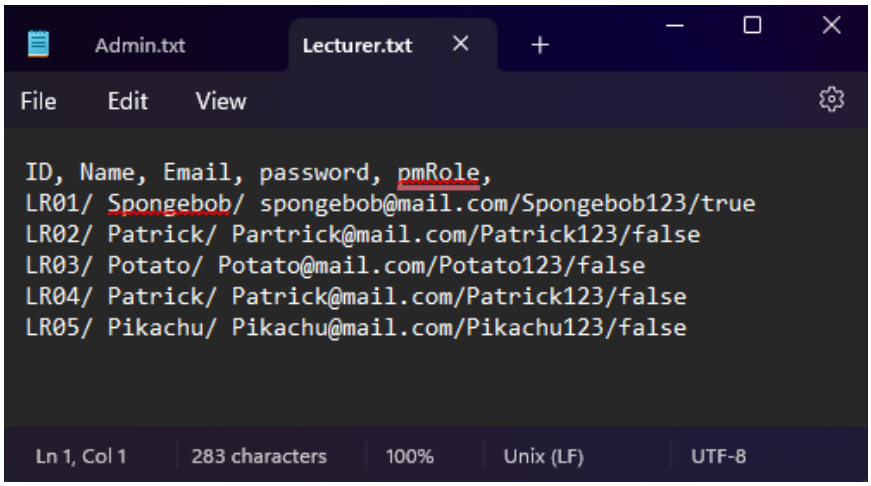
file/?orgRef=https%253A%252F%252Fwww.google.com%252F

9.0 Appendix

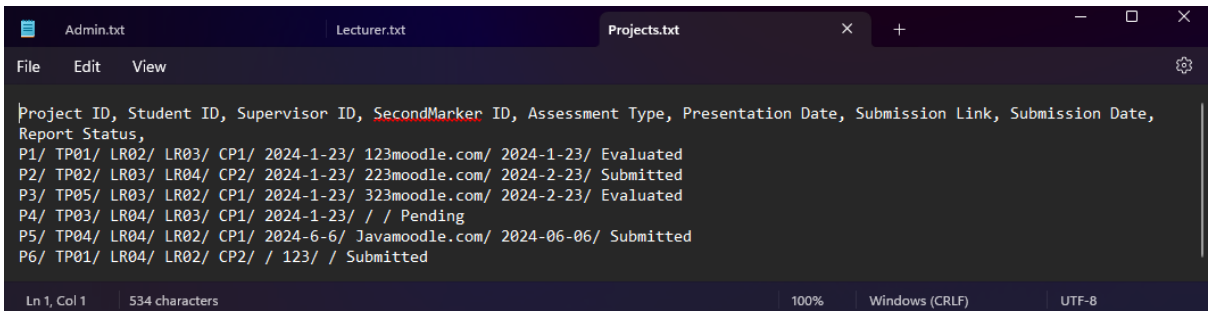
Admin.txt File



Lecutrer.txt File



Project.txt File



StudentData.txt File

Projects.txt Stude

File Edit View

```
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/ Angel23@yahoo.com/ Angel123/ 0624
TP05/ Oppa/ Oppa23@yahoo.com/ Oppa123/ 0224
```

Ln 1, Col 1 | 261 characters | 100% | Unix (LF) | 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	