NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2002 ASSIGNMENT : Final Year Project Management System (FYPMS).

Declaration of Original Work for SC2002 Assignment

We hereby declare that the attached group assignment has been researched, undertaken, completed and submitted as a collective effort by the group members listed below.

We have honoured the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

Name	Course	Lab Group	Signature/Date
Suki Ng	SC2002	Z51	Suki Ng 10/4/23
Tan Jie Ning, Jolynn	SC2002	Z51	Jolynn Tan 13/4/23
Koh Heng Woon	SC2002	Z51	Koh Heng Woon 14/4/2023
Rachel Lim (Lin Jiahuan)	SC2002	Z51	Rachel Lim 10/4/23

Table of Contents

1. Introduction

1.1 Overview

The Final Year Project Management System (FYPMS) is an application for staff and students to manage FYP the allocation of FYP projects.

This report comprises a write-up on the Design Principles and Object-Oriented Programming concepts utilised to develop this system. We also have a UML Class diagram highlighting the relationships between different classes, the multiplicity of the classes, coupled with a section on test cases to showcase the functionality of our code. For further elaboration, do refer to our demonstration video **(Z51-group2-demo.avi or the youtube link on pg12)**, UML **(Z51-group2-UML.png)** and javadocs **(Z51-group2-javadocs.html)** files.

1.2 The Data files we used:

Files	Column description
Supervisor.txt	Name, email address, User ID, number of projects assigned
Student.txt	Name, email address, User ID, status(if assigned or unassigned to a project)
Projects.txt	Project ID, Creator supervisor name, project title, project status (available, reserved, allocated, unavailable), supervisor name, supervisor email, student name, student email
Request	requestID, timestamp, requestorID, requesteeID, request status(pending, approved, rejected), new SupervisorID, new project Title, request description
UserList	UserID, Password, name, email address, User type (student, FYP coordinator, supervisor)

1.3 Approach Taken:

Our team has tried to thoroughly apply the Object Oriented concepts that we have learnt throughout this project. Both in the design and the implementation of our FYP system, we have made use of different classes and interfaces. Each control class we created is designed to handle very specific tasks within their specific areas. This sort of organisation makes it easier to maintain and update specific functionality if necessary, allowing for our program to be more scalable and sustainable for future use. We will be elaborating further on our design considerations in the section below.

2. Design Considerations

2.1 Single Responsibility Principle

This principle refers to the idea that each class should have one and only one responsibility. We applied and adapted this principle into our class design process and found that it was really helpful in allowing us to organise our development process. For example, the classes FYPProjectC, StudentProjectC and SupervisorProjectC are solely responsible for calling the functions related to projects for FYP coordinator, Students and Supervisor respectively. Another example would be the App class, which is solely responsible for calling the methods from the other classes. This approach helps us to streamline development and make conceptualization of the program easier since we only have to consider one type of function. Moreover, this allows for easy modification of the function without impacting other classes. It also enables scalability as functions and classes can be added without modifying existing classes.

2.2 Open Closed Principle

This principle refers to ideas whereby classes should be scalable, and we should be able to add new functions or attributes without editing the old ones. Considering the fact that many of us had to work on different classes before combining them, this principle was especially important in allowing the code to function as we expect it to. For example, the Student, Supervisor and FYPCoordinator class are extended from the User class, which shares similar traits such as personal information, since they are based on a human user. The User class has the bare minimum attributes such as name, email, userID, and UserType. Its child classes build upon these attributes and functions that are more specific to their class only. This means that the User class would not need to be modified in future. Therefore, we are able to change what the methods in the subclasses do without changing the source code of the methods in the superclass.

2.3 LisKov Substitution Principle

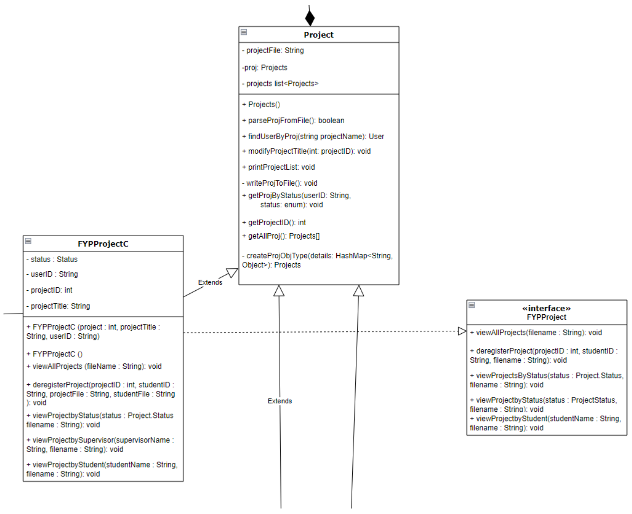
This principle refers to the idea that Superclasses must be able to be replaced by subclasses without breaking, the subclasses should enhance the functionality of the superclass but not reduce it. For example, The Request Superclass is extended by its subclasses, FYPRequestC, supervisorRequestC and studentRequestC subclasses. These subclasses contain additional functions specific to them only, and seek to enhance the functionality of the Request class. The studentRequestC class for instance has additional methods such as reqToAllocateProject, which is only unique to students, and at the same time does not reduce the functionality of the Requests class

2.4 Interface Segregation Principle

This principle refers to the idea that we use separate interfaces to separate different functions, and do not implement unnecessary methods for classes, which leads to unnecessary dependencies and increased complexity. For example, the View class has only one responsibility of formatting the display for the user interface. This also creates a more flexible and adaptable software system that is easier to modify and maintain and encourages a modular design approach that promotes the separation of concerns and improves the overall cohesion of the system.

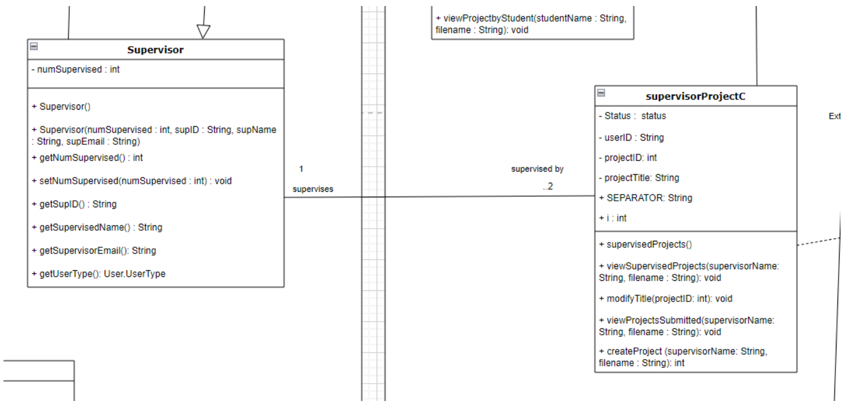
2.5 Dependency Injection Principle

This principle states that high-level modules should not depend on low-level modules, but instead should depend on abstractions. This implies that instead of writing code that relies on concrete implementation classes, we should write code that relies on interfaces or abstract classes. Throughout our UML class design, we implemented interfaces so classes can extend them. Implementing an interface allows a class to become more formal about the behaviour it promises to provide. This can be seen in various parts of our UML such as the Projects and Requests interfaces. These interfaces allow for a level of abstraction and would ensure that the classes which implement them are required to use the functions. As seen in the example below, The FYPProjectC class implements from the FYPProject interface, this ensures that the class is required to use that function. This also allows for greater flexibility, as implementation details can be changed without impacting the overall system, thus we can easily switch out implementations without affecting the rest of the codebase.


Project
- projectFile: String
-proj: Projects
- projects list<Projects>
+ Projects()
+ parseProjFromFile(): boolean
+ findUserByProj(string projectName): User
+ modifyProjectTitle(int: projectID): void
+ printProjectList: void
- writeProjToFile(): void
+ getProjByStatus(userID: String, status: enum): void
+ getProjectID(): int
+ getAllProj(): Projects[]
- createProjObjType(details: HashMap<String, Object>): Projects
FYPProjectC
- status : Status
- userID : String
- projectID: int
- projectTitle: String
+ FYPProjectC (project : int, projectTitle : String, userID : String)
+ FYPProjectC ()
+ viewAllProjects (fileName : String): void
+ deregisterProject(projectID : int, studentID : String, projectFile : String, studentFile : String): void
+ viewProjectbyStatus(status : Project.Status filename : String): void
+ viewProjectbySupervisor(supervisorName : String, filename : String): void
+ viewProjectbyStudent(studentName : String, filename : String): void
Extends
Extends
«interface»
FYPProject
+ viewAllProjects(filename : String): void
+ deregisterProject(projectID : int, studentID : String, filename : String): void
+ viewProjectsByStatus(status : Project.Status, filename : String): void
+ viewProjectbyStatus(status : ProjectStatus, filename : String): void
+ viewProjectbyStudent(studentName : String, filename : String): void

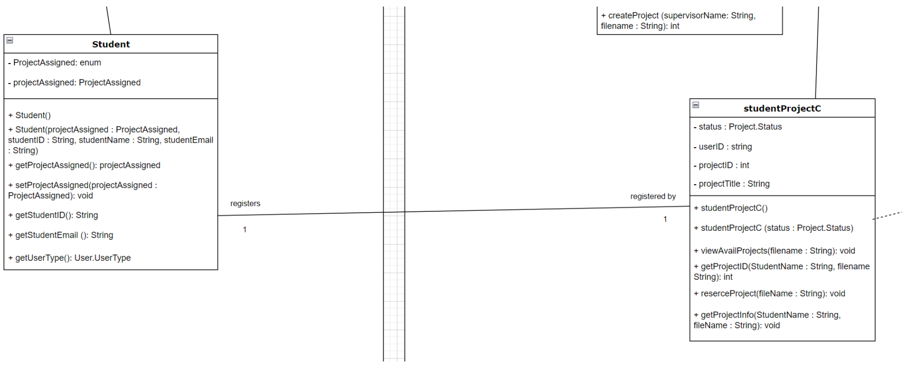
2.6 Multiplicity

We included multiplicity in our UML diagram as well. For the example shown below, 1 supervisor supervises at most 2 Projects. The relation between the Supervisor class and SupervisorProjectC class is bi-directional association, the supervisor supervises the Project while the Project is supervised by the supervisor.



+ viewProjectbyStudent(studentName : String, filename : String): void
Supervisor
- numSupervised : int
+ Supervisor()
+ Supervisor(numSupervised : int, supID : String, supName : String, supEmail : String)
+ getNumSupervised() : int
+ setNumSupervised(numSupervised : int) : void
+ getSupID() : String
+ getSupervisedName() : String
+ getSupervisorEmail(): String
+ getUserType(): User.UserType
1
supervises
supervised by
..2
supervisorProjectC
- Status : status
- userID : String
- projectID: int
- projectTitle: String
+ SEPARATOR: String
+ i : int
+ supervisedProjects()
+ viewSupervisedProjects(supervisorName: String, filename : String): void
+ modifyTitle(projectID: int): void
+ viewProjectsSubmitted(supervisorName: String, filename : String): void
+ createProject (supervisorName: String, filename : String): int
Ext

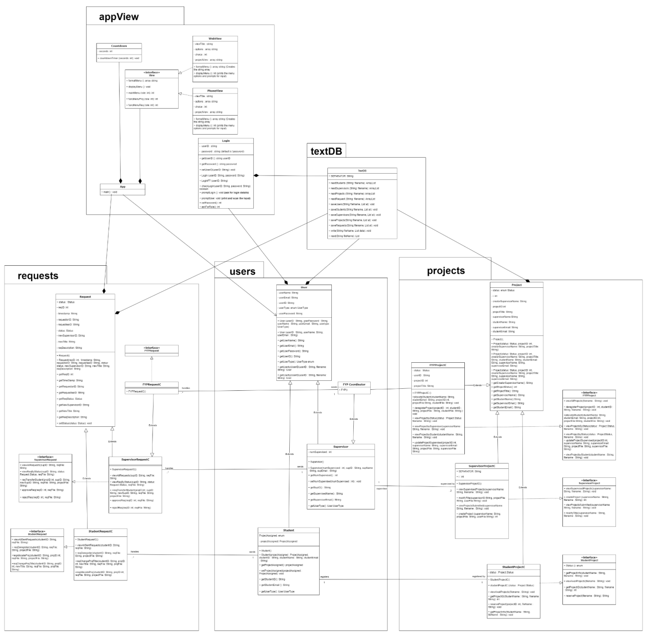
Another example of multiplicity used is between the Student class and the studentProjectC, where the relation is a bi-directional association, Student registers for studentProjectC, and studentProjectC is registered by the student. One studentProjectC can only be registered by only 1 student. Thus the multiplicity is 1.



+ createProject (supervisorName: String, filename : String): int
Student
- ProjectAssigned: enum
- projectAssigned: ProjectAssigned
+ Student()
+ Student(projectAssigned : ProjectAssigned, studentID : String, studentName : String, studentEmail : String)
+ getProjectAssigned(): projectAssigned
+ setProjectAssigned(projectAssigned : ProjectAssigned): void
+ getStudentID(): String
+ getStudentEmail (): String
+ getUserType(): User.UserType
registers
1
registered by
1
studentProjectC
- status : Project.Status
- userID : string
- projectID : int
- projectTitle : String
+ studentProjectC()
+ studentProjectC (status : Project.Status)
+ viewAvailProjects(filename : String): void
+ getProjectID(StudentName : String, filename String): int
+ resercеProject(fileName : String): void
+ getProjectInfo(StudentName : String, fileName : String): void

3. UML Class Diagram

Below is our UML Class Diagram. We have tried to cover the class diagram for the whole application. A clear diagram is also included in the **Z51-group2-UML.png** file.



4. Test cases

CASE 1.1: LOGIN: WRONG LOGIN

Description	Wrong username or password that does not exist in the database.
Expected Result	Prompt User to login again. After every wrong attempt, the number of remaining attempts left will be displayed. The user can only have 3 tries at once. After it is exceeded, the user will be blocked from logging in again for 5 seconds.
Pass or Fail	Pass

CASE 1.2: LOGIN: CHANGING PASSWORD

Description	In order to set a new password that satisfies requirements of (At least one capital letter, at least 8 characters, at least one special character). The passwords should be entered twice to confirm.. Change in password is prompted for first time users and as a function after logging in.
Expected Result	If the password is not strong enough, prompt User to create a stronger password. If the 2 passwords do not match, prompt User to enter a new password again (error checking). If successful change of password, prompt User to login again with the new password.
Pass or Fail	Pass

CASE 1.3: LOGIN: FIRST-TIME LOGIN

Description	Prompt User to enter if it is their first-time logging in.
Expected Result	User logs in with the default set password “Password”. If yes, prompt change password. If not, continue to the respective menu based on their user categories.
Pass or Fail	Pass

CASE 2.1: PROJECTS (STUDENTS): VIEW AVAILABLE PROJECTS

Description	Prompt Student User with a menu of functions. Student User inputs option to view all available projects.

Expected Result	List of available projects and its project details will be printed. Project details include projectID, Supervisor’s name and email, and Project title.
Pass or Fail	Pass

CASE 2.2: PROJECTS (STUDENTS): SELECT PROJECT TO RESERVE

Description	The student users can pick the “Reserve” option and key in the project ID they want to reserve.
Expected Result	After the student reserves the project they want, the status of that project is updated to reserved and other users can no longer see it under available projects.
Pass or Fail	Pass

CASE 2.3: PROJECTS (STUDENTS): VIEW HIS/HER OWN PROJECT

Description	The student picks the “ get project ID” or “get project information” option. They can then view their allocated project after the FYP Coordinator accepts their request to select the project.
Expected Result	The project details of the project that is allocated to the student is displayed.
Pass or Fail	Pass

CASE 2.4: REQUESTS (STUDENTS): REQUEST FOR RESERVE OF PROJECT and VIEW REQUESTS STATUS AND HISTORY

Description	The student can request for the project. The request can be approved by the FYP coordinator. While the request is pending, the project is reserved. The student can view their sent requests. Request details include the time and date it was sent, description and status (pending, approved or rejected).
Expected Result	Request is made and “view all sent requests” is updated for the student. Project is reserved while the request is pending.
Pass or Fail	Pass

CASE 2.5: REQUESTS (STUDENTS): REQUEST TO CHANGE TITLE / REQUEST COORDINATOR TO DEREGISTER FYP

Description	Students can: 1. Request for the project title to be changed. Requests can be approved by the supervisor. 2. Request for the project to be deregistered. Requests can be approved by the FYP coordinator.
Expected Result	Request is sent successfully and appears in the student's sent request history. Request appears in supervisor’s / FYP coordinator’s pending received requests.
Pass or Fail	Pass

CASE 3.1: PROJECT (SUPERVISORS): CREATE PROJECTS AND VIEW SUBMITTED PROJECTS

Description	Supervisors can create a project and view projects submitted by him/her.
Expected Result	New project is updated to the Projects database. The relevant details of the projects will be shown when viewing submitted projects.
Pass or Fail	Pass

CASE 3.2: PROJECTS (SUPERVISORS): UPDATE PROJECT TITLE

Description	The supervisor can update and modify the project title by inputting the projectID.
Expected Result	The project title of the corresponding project ID chosen will be updated.
Pass or Fail	Pass

CASE 3.3: PROJECTS (SUPERVISORS): VIEW PROJECTS SUPERVISED

Description	The supervisor cannot supervise more than 2 projects.
Expected Result	Once the cap of 2 projects is reached, all the remaining projects created by the supervisor becomes unavailable.

	When selecting view all projects, the detail of the status of projects created by that supervisor becomes unavailable.
Pass or Fail	Pass

CASE 3.4: REQUESTS (SUPERVISORS) APPROVE/REJECT REQUEST

Description	Supervisors can select the request ID they would like to approve/reject.
Expected Result	If the request is approved, the project title is changed and the request status updates to approved. If the request is rejected, the project title remains unchanged and the request status updates to reject. Request status changes from pending to approved/rejected respectively.
Pass or Fail	Pass

CASE 3.5: REQUEST (SUPERVISORS): REQUEST TRANSFER OF STUDENT TO FYP COORDINATOR AND VIEW REQUEST HISTORY

Description	The supervisor can request transfer of students by inputting project ID and the replacement supervisor ID. The request will be sent to the coordinator.
Expected Result	The supervisor's request is sent to the FYP Coordinator. The supervisor's request history and the FYP Coordinator's received pending requests are updated.
Pass or Fail	Pass

CASE 4.1: REQUEST (FYP COORDINATOR): VIEW ALL REQUESTS AND APPROVE TRANSFER STUDENT REQUEST

Description	The FYP Coordinator can approve the request sent by the supervisor to change the supervisor supervising the project.
Expected Result	The project detail of the supervisor name and supervisor ID is updated to the replacement supervisor's name and ID.
Pass or Fail	Pass

CASE 4.2: REQUEST (FYP COORDINATOR): MAXIMUM CAP OF REPLACEMENT SUPERVISOR REACHED

Description	The replacement supervisor’s capacity is reached.
Expected Result	The hint message is displayed to the FYP Coordinator that the replacement supervisor’s capacity of supervising 2 projects is reached. The FYP Coordinator can decide whether to approve or reject after the hint message.
Pass or Fail	Pass

<u>CASE 4.3: REQUEST (FYP COORDINATOR): APPROVE ALLOCATION OF STUDENT TO PROJECT</u>

Description	The FYP Coordinator approves the allocation of a student after the student selects a project.
Expected Result	After the FYP Coordinator’s approval, the project status of the selected project is updated to allocated.
Pass or Fail	Pass

<u>CASE 4.4: REQUEST (FYP COORDINATOR): APPROVE DEALLOCATION OF STUDENT TO PROJECT</u>

Description	The FYP Coordinator approves the deallocation of a student from a project after the student requests to do so.
Expected Result	After the FYP Coordinator’s approval, the project status that the student requested to be deallocated from is no longer assigned.
Pass or Fail	Pass

<u>CASE 4.5: REQUEST (FYP COORDINATOR): VIEW PROJECT ACCORDING TO FILTERS</u>

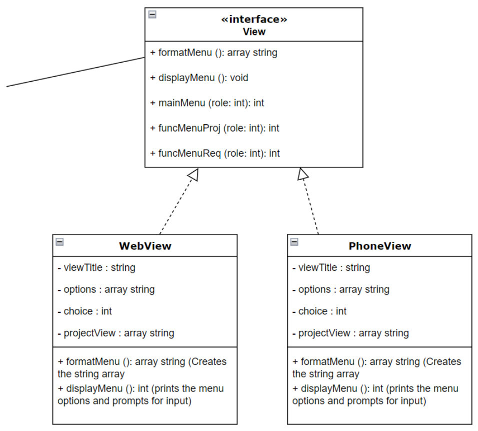
Description	The FYP Coordinator can view all projects according to different filters: 1. By project status 2. By supervisor ID 3. By student ID
Expected Result	The project details are displayed according to what filters were selected.

Pass or Fail	Pass

5. Future Scope

5.1 Extendability of our FYP system to be used on various kinds of web devices

By creating a view interface, we are able to implement the interface of View to different classes, such that the FYP system can be accessed through different web views, such as through phones, or laptops.


«interface»
View
+ formatMenu (): array string
+ displayMenu (): void
+ mainMenu (role: int): int
+ funcMenuProj (role: int): int
+ funcMenuReq (role: int): int
WebView
- viewTitle : string
- options : array string
- choice : int
- projectView : array string
+ formatMenu (): array string (Creates the string array
+ displayMenu (): int (prints the menu options and prompts for input)
PhoneView
- viewTitle : string
- options : array string
- choice : int
- projectView : array string
+ formatMenu (): array string (Creates the string array
+ displayMenu (): int (prints the menu options and prompts for input)

5.2 Additional Considerations

To make the app more user-friendly, we could sort the pending request for the supervisors to view from either the newest request to the oldest request and vice-versa.

In addition, we could increase the security of the app and users' personal information by only allowing users to change their password after they have successfully verified their identity through a programmed-sent email or text message on their mobile devices. This can be done with the addition of the library javax.mail.* and javax.mail.internet.*.

6. Reflection

One of the difficulties we encountered along the way was trying to read in input from text files, as we have never done it before in our previous labs or tutorials. The way we managed to conquer it was through exploring Java's string tokenizer functions to read our text files and then storing our various objects into an array list. A future improvement suggestion is that we should try to hash the passwords instead of just storing them in a text file. This is to ensure the passwords are encrypted and cannot be retrieved easily.

Another difficulty we faced was trying to apply interfaces to our UML diagram. After understanding the purpose and significance of an interface, we were better able to decrease the complexity of each class and reduce complications in the future.

7. Link to Demo Video

Link: https://www.youtube.com/watch?v=7jy9z7P41ik