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 (CE2002 or CZ2002) Lab Group	Signature	Date
AMULYA MATHUR	Z40	Amulya	15/04/2023
BHUPATHIRAJU MIHIR VARMA	Z40	Mihir	15/04/2023
DHARINISRI SATHIAMOORTHI	Z40	Dharinisri	15/04/2023
IZZAT HAZIM BIN MUSTAFA	Z40	Izzat	15/04/2023
WANG JUN WEI SERENA	Z40	Serena	15/04/2023

Application Overview

A FYP registration system is created. The main system consists of a different interface. The user interface will make use of user subclasses, which implements an abstract method prompt that guides users to perform actions All userAction subclasses implement the act method for user classes to use. To allow effective display of information, various display subclasses are managed by respective display manager subclasses. Lastly, a request record object is created with a subclass of request action as an instance when a user makes a request, which allows the approval or rejection of requests afterward. With the given features of the program, it is made to be dynamic, user friendly, quick, and robust for all users.

Design consideration

Assumption of design

1. All students or supervisors have a respective coordinator which will be allocated to them by the system when their account is created. It will be coordinator Li Fang" by default.
2. For any project that is submitted by a faculty is considered as his/her project. If a faculty requests to transfer a student to a replacement faculty, the project no longer belongs to him/her, but the replacement faculty's. Number of supervising projects of each faculty will be updated accordingly. If the replacement faculty has hit the cap, the request is rejected.
3. Projects are allocated on a first come first serve basis. Students can request a coordinator for project allocation multiple times, the project's status will still be available when the project is allocated. The coordinator will approve the project allocation requests in the order they are received, using a first come first serve principle. In which the project status will not be changed to 'reserved' when a student requests for project allocation. In our system, a wait-list approval system is adopted, assuming the coordinator will always approve a student's project allocation request on a first come first serve principle, thus allowing students a high flexibility when choosing desired projects.
4. Requests will be automatically rejected if certain necessary conditions are not met, even if the coordinator chooses to approve it. For example, to approve a student's request for project allocation while the supervisor of the project has hit the cap already.
5. All users' default password is 'password'

The use of an Object Oriented Approach

1. Abstraction: It is the process of providing the functionality methods to a user, and hiding the implementation details. The abstract methods are described and implemented in the specific user classes only. For example, the abstract act() method in the UserAction class is implemented differently in its subclasses. In the UserChangePassword subclass, the act() method would enable the user to change the password of their account. Likewise, the act() method in the UserRequestAction subclass would enable the user to make a request to the other user.
2. Encapsulation: Encapsulation protects an object's private data by hiding the implementation details of a class. A user only has to understand which methods are available to call and which input parameters are needed, without knowing the implementation details. For example, the requestRecordList attribute in the RequestRecordList class cannot be accessed directly at the interface level, and is only retrieved by calling the getRequestList() method. In this context, the users are allowed to access the full list of requests without being involved in the implementation of it.
3. Inheritance: A subclass inherits all the attributes and methods of a superclass. The derived class can be extended, adding on new instances or methods. This enables code reusability and extensibility of classes. One instance is that the student and the faculty inherit basic properties and methods of a user: userID, name, and requestList, setPassword(password: String), getMyRequestList(), and verifyPassword(password: String). Along with these inherited properties, specific attributes and methods are also 4. included in the subclasses.
4. Polymorphism: It is the ability of the object to take multiple forms. It is commonly used within the inheritance of parent and child classes. When a certain method is used through the base class, methods with the same method name and signature from different subclasses could be called. This could allow different actions to occur, depending on the implementation in the subclass by only using the same method name. Student class could perform different actions such as UserChangePassword, StudentRequest by using the method act() through the userAction base class. The implementation of this principle is method overriding and object typecasting.

SOLID design principle

1. Single responsibility principle (SRP): A class should only be responsible to do one thing. Instead of creating a display method in user classes to display projects lists in various ways, such as displaying both user's project details and all available projects, separated ViewMyProjectDetails and ViewAllAvailableProjects subclasses that extend from the display base class are implemented. Hence, each class only has one responsibility and users could perform the desired action making use of their methods.
2. Open-closed principle: A module should be open for extension but closed for modification. When a new user action is needed, we can easily extend a subclass from the userAction base class, implement the abstract act method. This allows users to use it through the userAction base class and act method, without the need to modify the user classes.
3. Liskov substitution principle: A base class should perform as expected when a subclass substitutes it. ProjectListDisplayAll and ProjectListDisplayAvailable are subclasses of the Display class. Each performs a sanity check to display respective messages when the project list is empty, ensuring that neither is expecting more nor providing less when used. When these subclasses are being used in the AllProjectListDisplayManager through the Display base class, both could perform their action properly. This also shows how these subclasses are substitutable for their superclass.
4. Interface segregation principle: A class should only implement or extend a necessary interface or base class. A general purpose interface which might contain unnecessary implementations is therefore not desirable. Both the UserInterface class and the LoginInterface class implement the AppInterface interface, which includes an abstract method prompt() that is necessary for both the subclasses.
5. Dependency principle: The principle states that both the high and low classes should be independent of each other and depend only on abstraction, with abstractions not depending on details while the details should be dependent on the abstraction. We minimize class dependencies and rely more on abstractions. Each user class performs a different set of actions, yet none of the actions is hardcoded or dependent on a particular user class. Instead, user actions are performed by accessing subclasses that extend from the UserAction abstract class.

Extensibility and maintainability

1. Extensibility: By applying different OOP design principles, new features such as new user action, new request type could be easily added by inheriting from existing relevant base class and implementing the necessary method. Minimal changes are required.
2. Maintainability: Our code is easily understandable and classes could be debugged independently by applying the OOP approach. It also allows high reusability of code.

Reflection

Designing the FYP system has been a challenging task, requiring the shift from the accustomed top-down procedural approach to a bottom-up approach, which involves design consideration on the use of different objects, classes and methods. Despite the time taken to adopt the OOP mindset, we quickly realized how helpful this approach is when designing a system, even of its complexity at first glance. Visualizing the system with a UML diagram is particularly helpful in understanding its structure in this case.

The advantages of an OOP approach became more apparent during the development stage. The system could be broken down in modules and each could be developed and debug separately. Additionally, utilizing the OO approaches allow us to avoid redundant implementation of cate methods for different classes, which both results in higher efficiency.

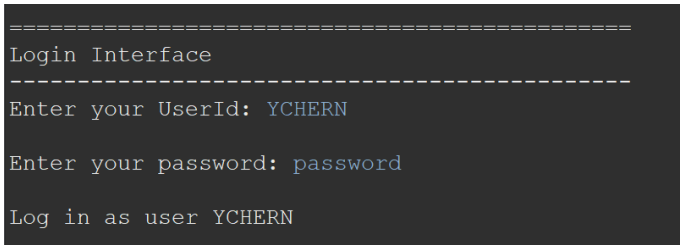
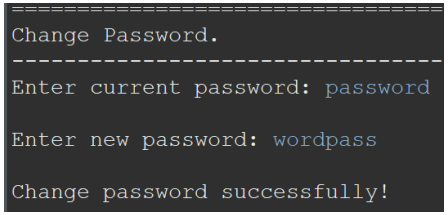
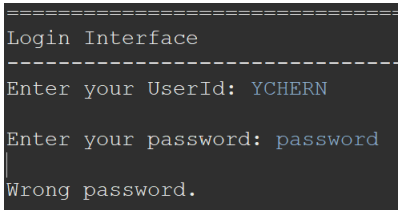
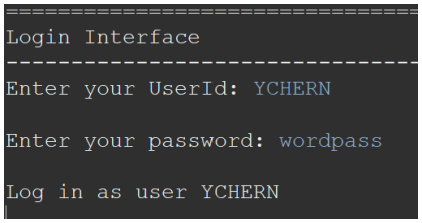
This project allows us to explore a useful alternative in designing and implementing a system. We believe that our knowledge and experience gained is surely helpful for future complex or large scale projects development

UML Diagram

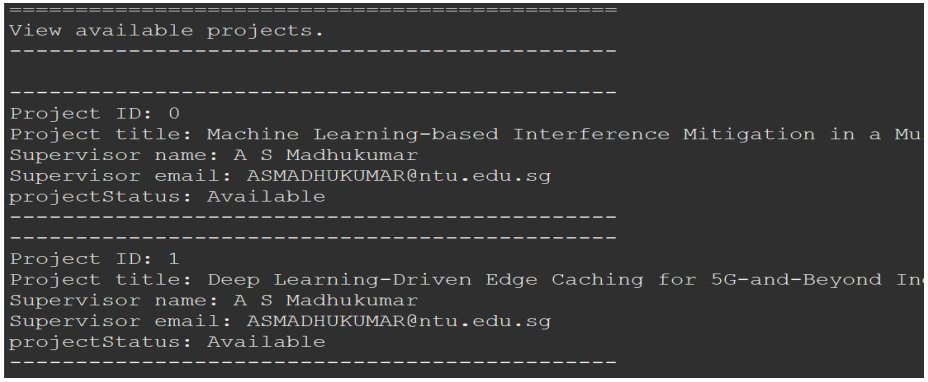
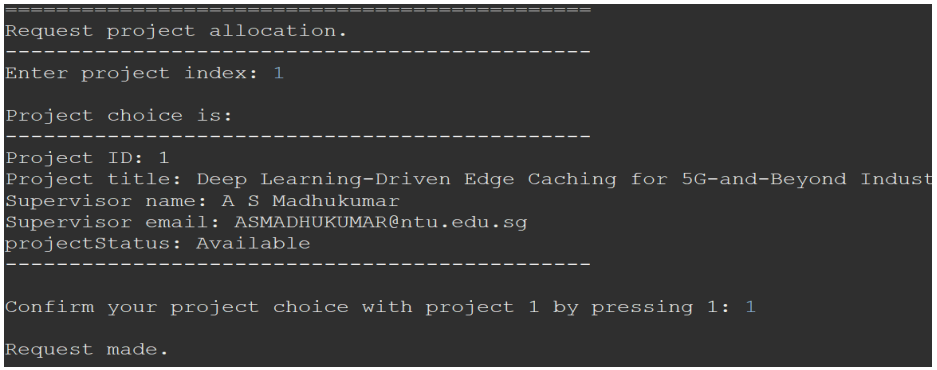
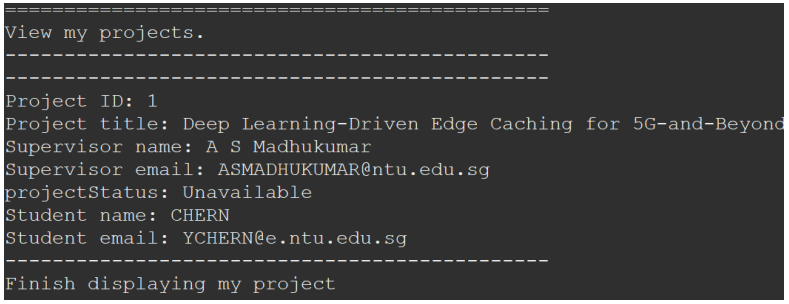
- The pdf of our uml diagram is included in the submission folder to be viewed clearly.

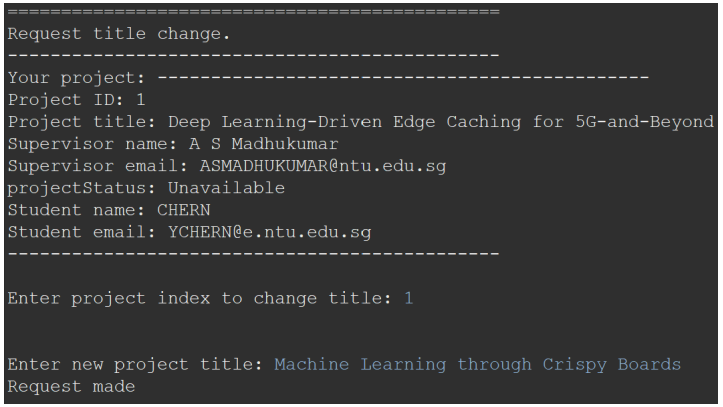
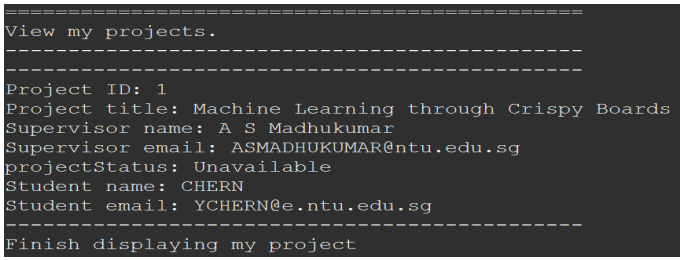
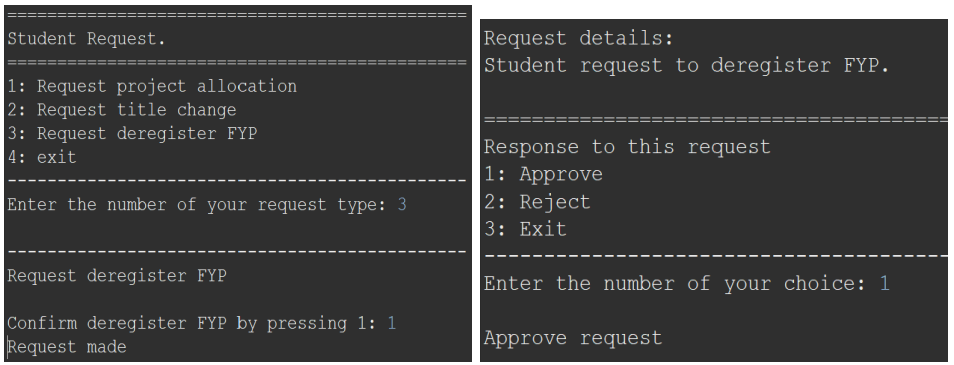
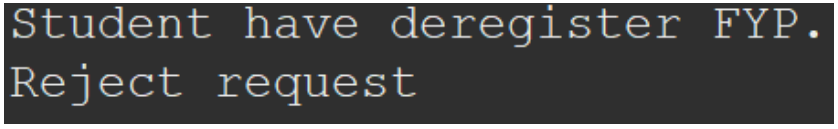
Testing

User

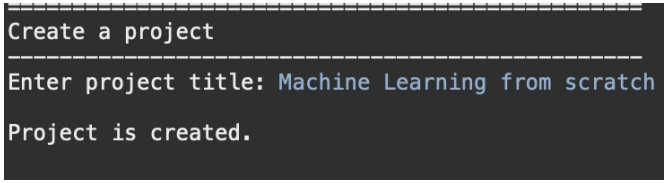
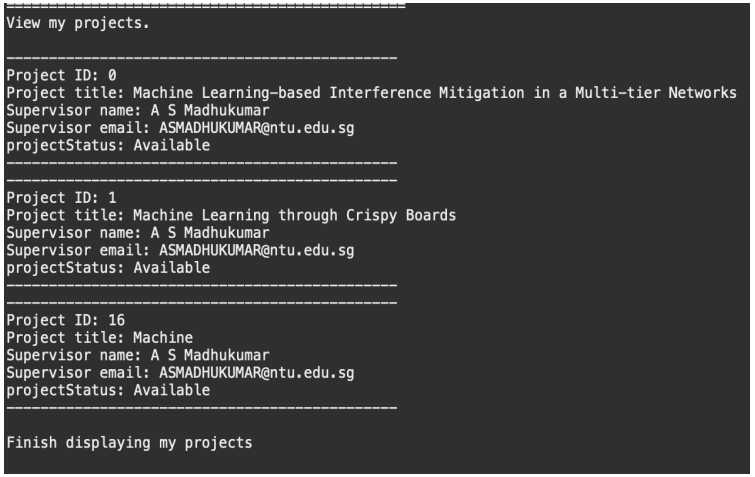
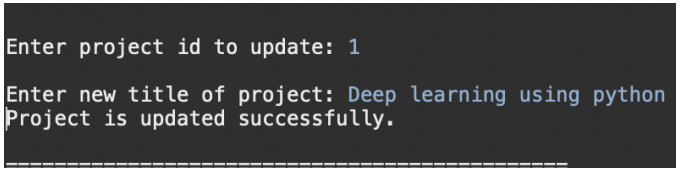
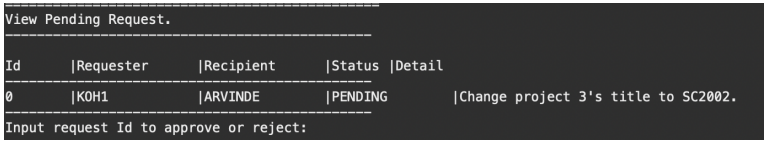
Function	Expected functionality	Program output
Login	Login with userId(YCHERN) and password(password).	
Change password	Change password in the system- (from password to wordpass). Failed login with old password- (The login failed since the new password is wordpass). Successful login with the new password.	  

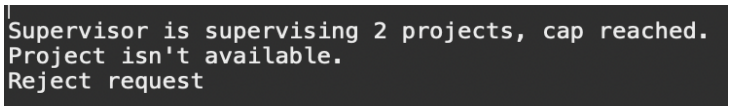
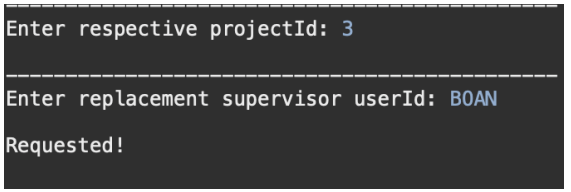
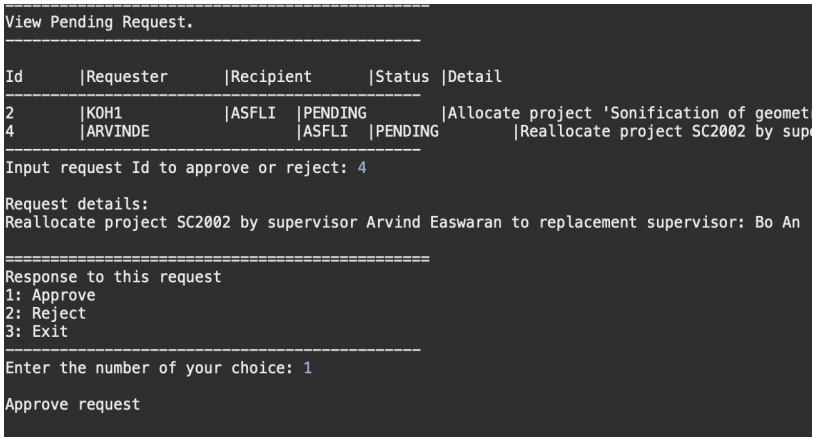
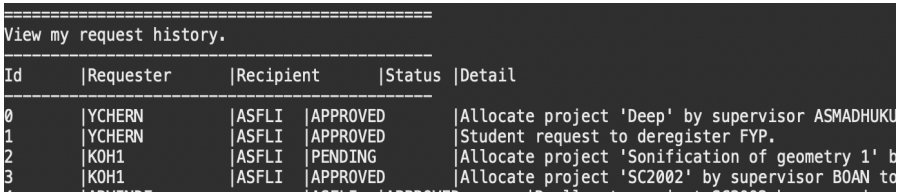
Student

Function	Expected functionality	Program output
View available projects	View all available projects if they still need to register or deregister FYP.	
Request coordinator to allocation project	Select a project with projectId, and request the FYP coordinator to allocate the project to yourself.	
View user registered project	View registered project once the registration is done.	

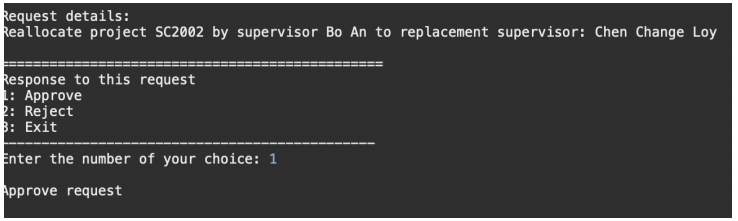
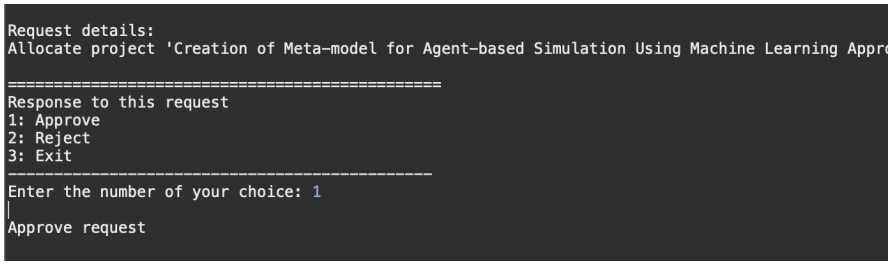
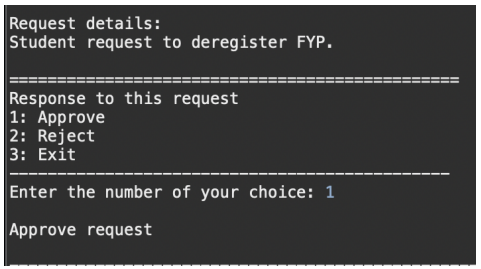
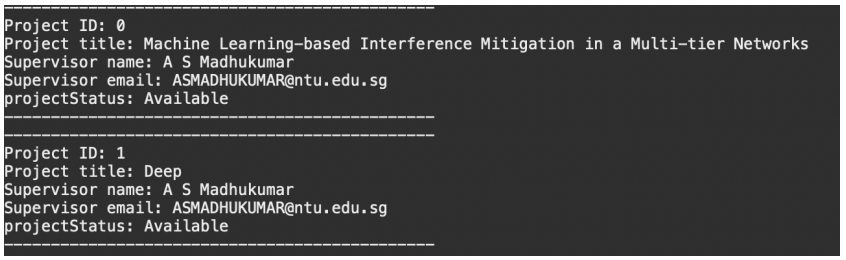
Request supervisor to change project title	Request the supervisor to change the title for a student proposed project after registration.	
	Once the supervisor approves the request, the title will be changed.	
Request coordinator to deregister FYP	Request the coordinator to deregister FYP after allocation. FYP is deregistered once the coordinator approves it.	
	Student isn't allowed to select project again after deregistration is approved.	

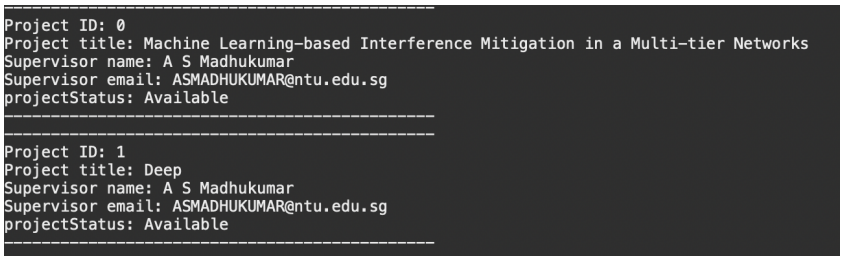
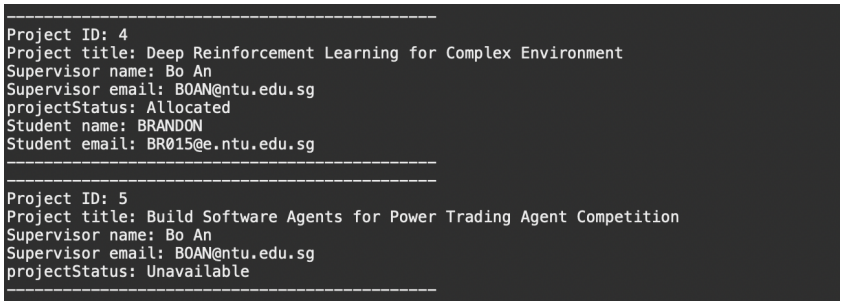
Supervisor

Function	Expected functionality	Program output
Create projects	Create project one at a time.	
View my projects	Information of the project submitted by the supervisor.	
Modify title	Modification of the title.	
Modify title by approving student's request	Project title is changed when request is approved	

Cannot supervise more than 2 projects	Coordinator cannot allocate projects to the supervisor if the cap is reached.	
Request coordinator to transfer a student	The supervisor is requested to be replaced for the respective Project.	
	The student is transferred once the request is approved and if the conditions are met. (the replacement supervisor has not hit his/her supervision cap yet)	
View all pending requests	View all pending requests from students and approve/reject them. [new] will indicate if there's pending requests.	
View request history	View all incoming and outgoing requests' history and details	

Coordinator

Function	Expected functionality	Program output
Approve to transfer a student	Change supervisor of a project upon request.	
Approve to allocate a project to student	Allocate a project to a student upon request.	
Approve a student to deregister FYP	Deregister FYP of a student upon request.	
View all projects by filter	View all projects	 ...

	View available projects	
	View unavailable projects (allocated to students, or unavailable as the supervisor has hit the cap)	
	View projects by supervisor