

**SC2002 OBJECT ORIENTED DESIGN & PROGRAMMING**

**FINAL YEAR PROJECT MANAGEMENT SYSTEM**

**Report of Project Structure Design & Functionality**

**AY22/23 Sem 2 | A50, Group 6**

|  |  |
| --- | --- |
| **NAME** | **MATRICULATION NUMBER** |
| Pu Fanyi | U2220175K |
| Jiang Jinyi | U2220259H |
| Jin Qingyang | U2220239A |
| Soo Ying Xi | U2220021D |

**Project Demonstration Video:** [**https://youtu.be/8FikWzfHlLA**](https://youtu.be/8FikWzfHlLA)

**Project Main Page:** [**https://pufanyi.github.io/FYPMS/**](https://pufanyi.github.io/FYPMS/)

**GitHub Main Page:** [**https://github.com/pufanyi/FYPMS**](https://github.com/pufanyi/FYPMS)

**Project Document:** [**https://pufanyi.github.io/FYPMS/docs**](https://pufanyi.github.io/FYPMS/docs)

**Declaration of Original Work for SC/CE/CZ2002 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 the 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** |
| Pu Fanyi | SC2002 | A50 |  |
| Jiang Jinyi | SC2002 | A50 |  |
| Jin Qingyang | SC2002 | A50 |  |
| Soo Ying Xi | SC2002 | A50 |  |

# DESIGN CONSIDERATIONS

FYPMS (Final Year Project Management System) is a Java console application designed with a focus on reusability, extensibility, and maintainability. It efficiently manages final year project settings and accommodates different user types and their requirements, allowing for easy upgrades and future development.

## Design Approach

The FYPMS was designed with a focus on high cohesion and loose coupling, with classes separated into three categories: *controllers*, *boundaries*, and *entities*. *Controllers* include *“Project Manager”*, *“Request Manager”*, *boundaries* include *“Student Main Page”*, *“Coordinator Main Page”*, *entities* include *“Students”*, *“Projects”* etc. When the user is using the system, he/she interacts with the boundaries, which then call the controller to perform requested operations such as making changes to the entity or retrieve information from entity to display etc. Each of these categories works together to complete our system while ensuring the dependency on each other is minimized. As such, our system is highly flexible, extendable and easy to maintain. Minimum effort is required when extending our system, for example when a new function in the system is introduced.

## Highlights of Some Designs

Reflection: The interface *Model* uses reflection to convert between classes and strings, enabling dynamic handling of model data without manual mapping.

Generic Repository Class: *Repository<Model>* class with generics allows for flexible data storage and retrieval for any model type, reducing duplication and improving maintainability.

SHA-3 Password Encryption: User passwords are encrypted using the SHA-3 algorithm for enhanced security.

Batch CSV Data Import: The system supports batch importing of initial data from CSV files, making it convenient to process large datasets in chunks.

Factory Design Pattern: The factory pattern is used to quickly generate requests based on different requirements, enhancing system scalability and adaptability.

JUnit 5: We used JUnit 5 to test our classes, it helps to ensure the correctness and robustness of our code.

## Applied Design Principle

### Single Responsibility Principle (SRP)

The Single Responsibility Principle (SRP) recommends that each class should have a clear and singular responsibility, avoiding unrelated tasks. By adhering to the SRP, we can minimize the ripple effect of changes, simplifying the process of modifying, testing, and reusing code, resulting in more maintainable and robust software design.

### Open/Closed Principle (OCP)

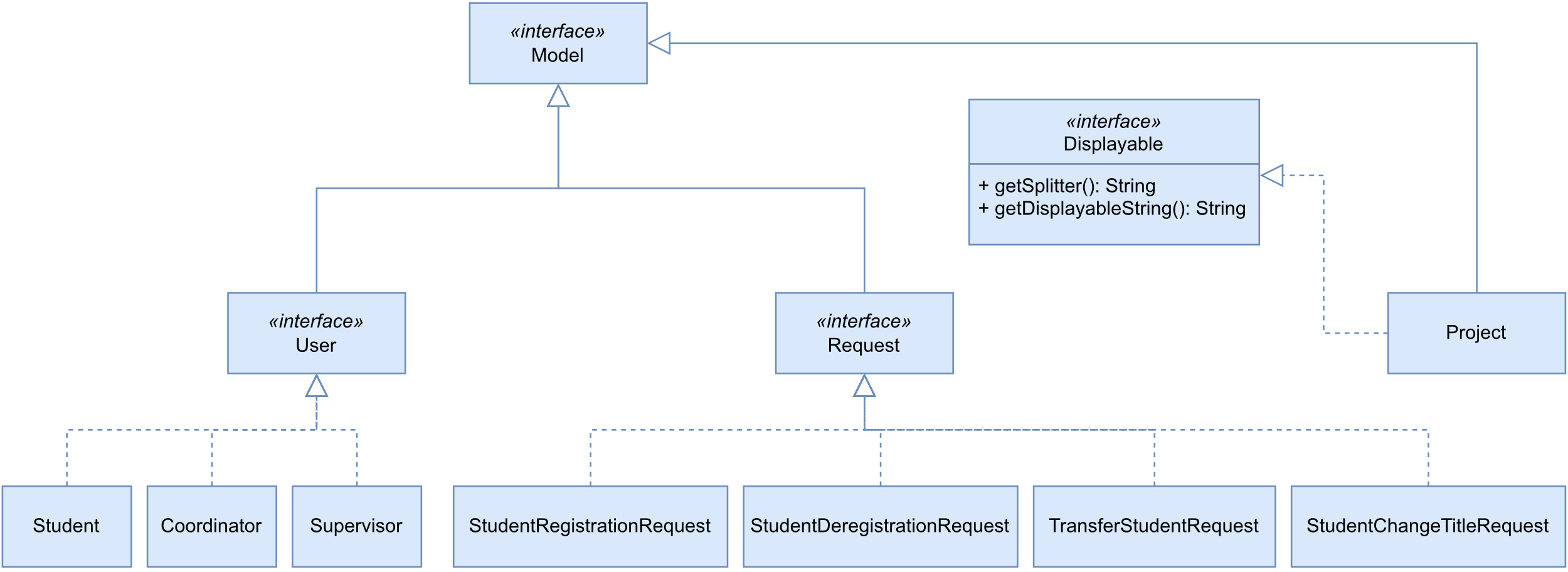
The Open/Closed Principle (OCP) states that classes should be open for extension but closed for modification, allowing for the addition of new functionality without changing existing code. OCP can be implemented through abstraction, inheritance, and polymorphism.

In our project, we apply OCP by creating an abstract class "Repository<Model>" that can be extended to create different types of Repositories such as "ProjectRepository" and "RequestRepository." Each subclass overrides the "getFilePath()" method to allow for easy extension of the Repository system. For requests, we use an interface "Request" to derive different types of requests such as "StudentRegistrationRequest" and "StudentDeregistrationRequest." Each subclass overrides the "get" methods for request details, enabling new request types to be added without changing the existing code.

### Liskov Substitution Principle (LSP)

To put it simply, Liskov Substitution Principle states that the subtypes must be substitutable for the base types. In the case of our system, the usage of this principle is widely applied. One of the examples is the *‘Request’* class. Subclasses of the ‘Request’ include *‘StudentRegistrationRequest’*, *‘TransferStudentRequest’* etc. All those subclasses are substitutable for the ‘Request’ class object while ensuring the methods behave correctly. When determining the request type of each different request, we may use different instances of requests to call the *‘getRequestType’* method, and the corresponding method in the subclass will be called and return the correct request type. Polymorphism is also applied here.

### Interface Segregation Principle (ISP)

The interface segregation principle refers to that many specific interfaces are better than one general interface. In other words, we should always avoid designing a ‘fat interface’. As such, when developing our system, we noticed that this is important to promote maintainability, flexibility and modularity. For instance, we found that the *‘Model’* must be further divided into separate interfaces such as *User, Project* and *Request* so that the different entities can implement the interfaces accordingly. By doing so, we can ensure that the entity classes don’t have to implement methods that are not related and reduce the ripple effect when modifying our system.

Furthermore, we have applied the ISP in our project by incorporating the *Displayable* interface. This aligns with the principle of preferring specific interfaces over general ones, avoiding a bloated interface. The *Displayable* interface defines two methods, *getSplitter()* and *getdisplayableString()*, allowing objects to be formatted and displayed.

### Dependency Injection Principle (DIP)

The dependency injection principle suggests that higher modules must not depend on lower modules, but both should depend on abstraction. In other words, instead of directly depending on the concrete class to perform some operation, we can depend on the interfaces, which are less likely to be changed. In our design, we highly focus on this principle. For example, when getting the ID of a student user, instead of depending on the *‘Student’* concrete class, we depend on the *<User>* interface. This will allow us to add more users with the least effort needed in the future and make our system more extendable.

## Further Enhancement

For the further development of our system, we considered the situation that there may more than two students requesting to register for a project created by the same supervisor. When any two of the requests approved, the other requests will remain pending, and the students will be waiting forever for the request to be approved. As such, we designed a feature such that whenever a supervisor is supervising 2 projects, the other requests which are to register for the projects from the same supervisor will be rejected. By doing this, we can ensure that there is no miscommunication occurred, and students will be able to know their status and make their following decisions effectively.

On the other hand, we have made our system more friendly to the users. Every time the user is prompted to enter the ID of a project or request, we will first display the viewable projects or requests (depending on user type), then follow by asking the user to input the ID. With this, the users can interact better with our system’s interface and thus increase its effectiveness.

## Reflection

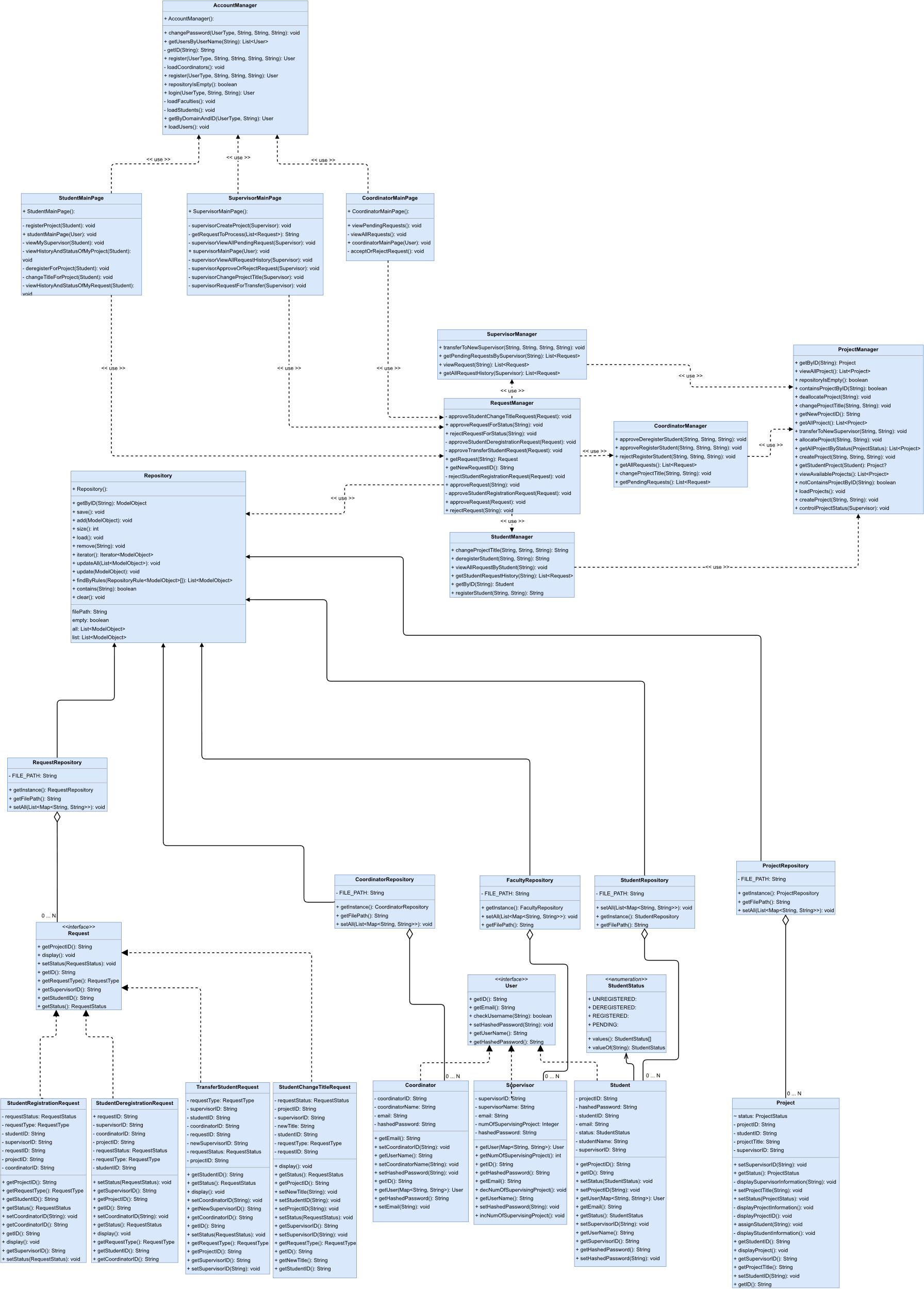
From this assignment, we have seen the importance of design principles through real application. At the beginning phase of this assignment, we found that a slight change in our code will trigger a ripple effect, causing that almost all other parts of our code must be changed. We then refer to the design principles and apply them widely in our assignment. Therefore, software with high cohesion and loose coupling is ultimately important, as to make it to have high flexibility, easily maintainable and extendable.

At the same time, we have learnt to design a software that fits its functions and real-world applications. Considering all users for the software, we kept modifying our design so that it takes into considerations different possible scenarios and ensures that there are no conflicts between the users.

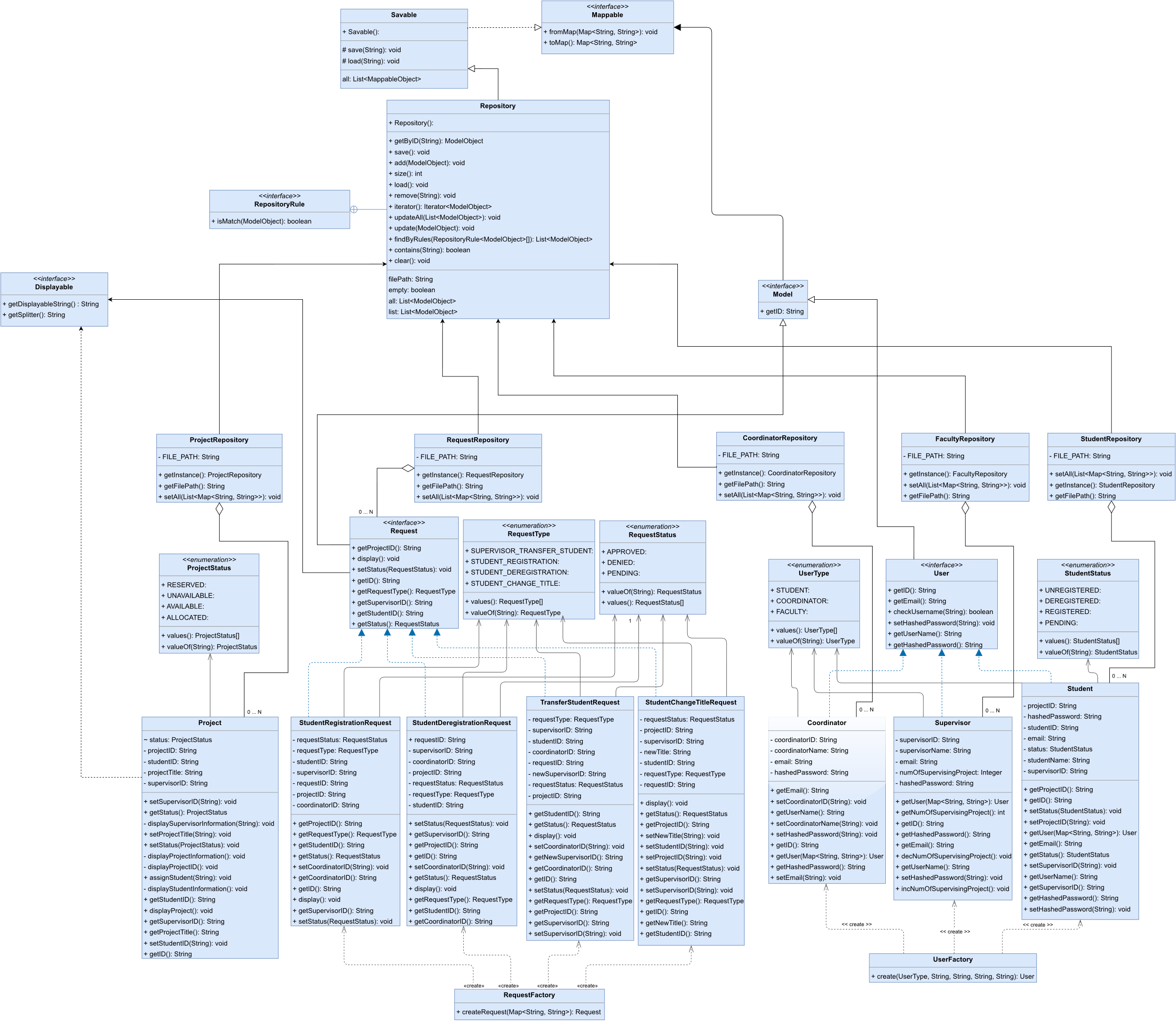
# DETAILED /UML CLASS DIAGRAM

Please Refer to the *<UMLClassDiagram>* for further details.

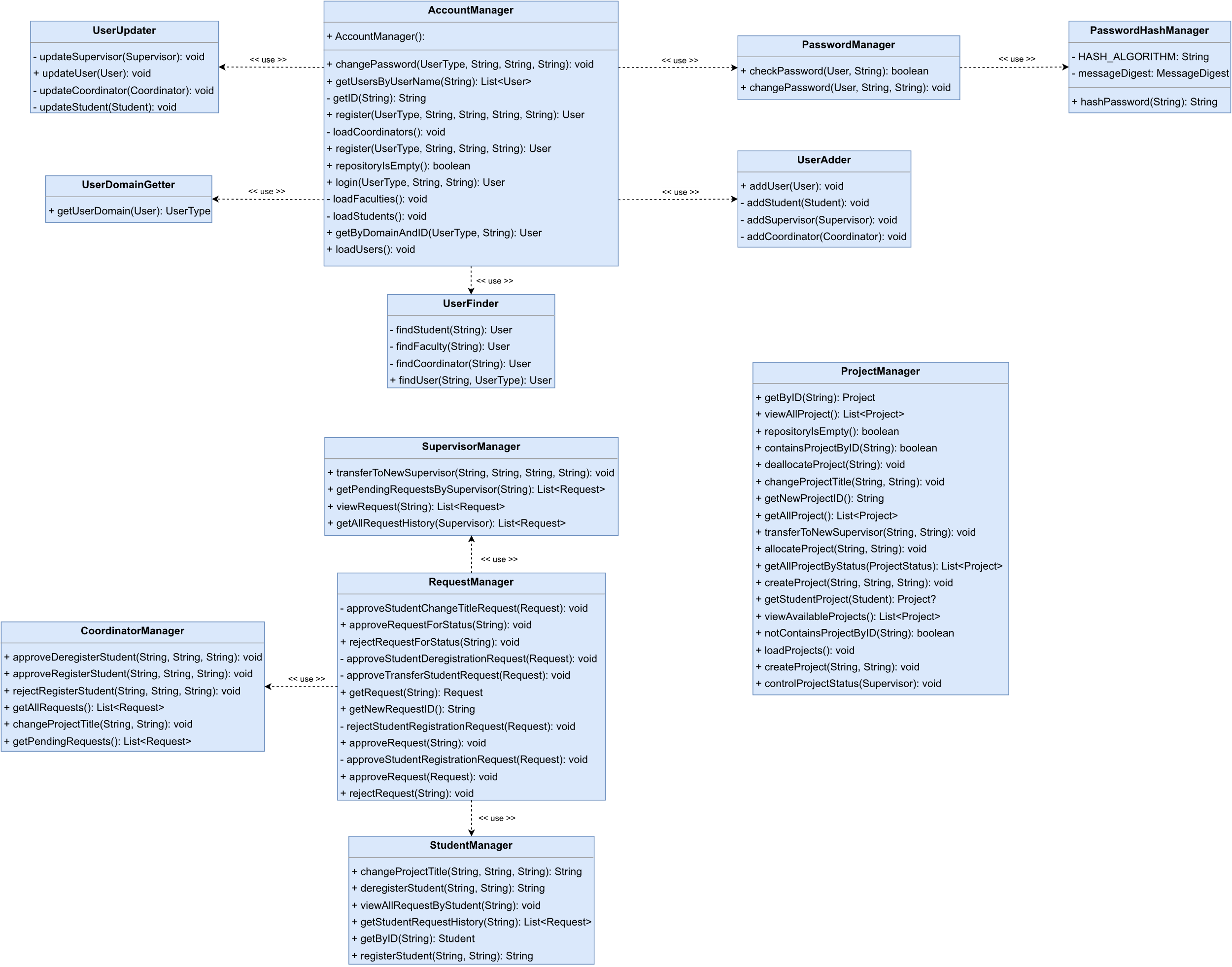
## Main Diagram



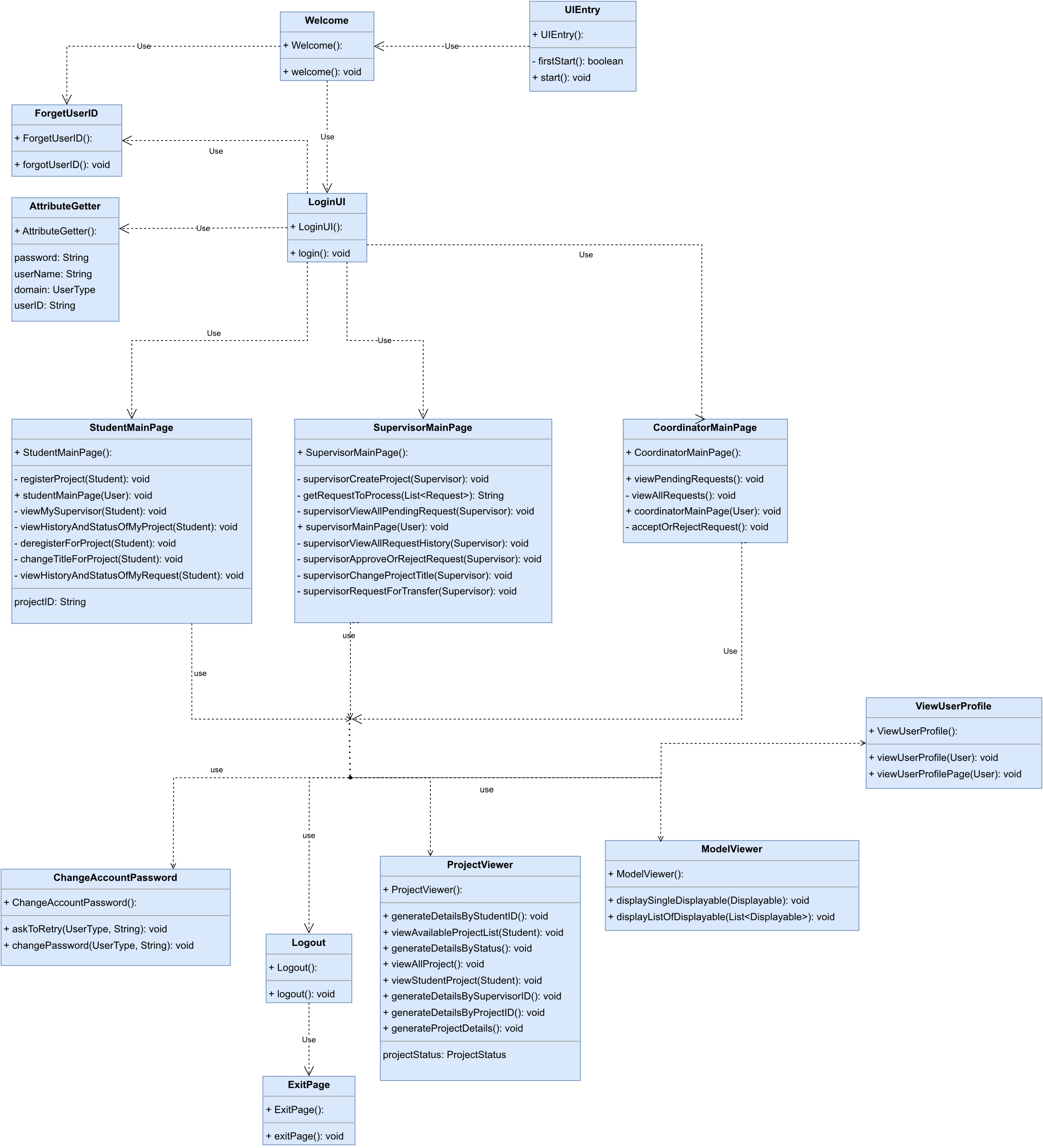
## Entity Sub Diagram



## Controller Sub Diagram



## Boundary Sub Diagram



# Testing

## Welcome Page

### Login Page

|  |
| --- |
| Upon entering the system, the user can choose from 3 different domains. |
| Here we use students as an example. |

### Forgot User ID

|  |
| --- |
| If the user chooses the option of *forget UserID*, the system will return the UserID of the user. |

### Change Password

|  |
| --- |
| Successfully change password |
| Entering a new password with fewer than 8 characters or incorrectly entering the password twice is not permitted. |

### Logout

|  |
| --- |
|  |

### 3.2 Student Register and Deregister

**3.2.1 Student sends a request to register a project**

|  |
| --- |
| The system will first show a list of available projects to choose from. Student then chooses a project, the system will ask student to confirm then submit the request |

### 3.2.2 Coordinator approve registration request

|  |
| --- |
| The coordinator can choose to approve/reject request inside the coordinator main page |
| The system will display all the pending request, and the coordinator can choose a request by the RequestID to approve/reject. |

### 3.2.3 Student requests to deregister

|  |
| --- |
| Student select the option to deregister. Student confirm to deregister. |
| After the coordinator approves the deregistration request, the student is not allowed to choose other projects. |

### 3.2.4 Three students request one supervisor

|  |
| --- |
| When two students have registered for the projects of a supervisor, the supervisor’s remaining project will become unavailable. |

### 3.3 Change Title

**3.3.1 Supervisor change title**

|  |
| --- |
| The supervisor can choose a project from list of his projects to change the title. The system will then prompt the user to enter new title. The system will change the title after confirming with the supervisor. |

### 3.3.2 Student change title

|  |
| --- |
| 图形用户界面, 文本, 应用程序  描述已自动生成 表格  中度可信度描述已自动生成  Student can choose to modify his registered project’s title and send the request to the supervisor  表格  描述已自动生成 表格  中度可信度描述已自动生成  Only the supervisor of the project have the permission to process the request  表格  描述已自动生成  The new project title is successfully updated |

### 3.4 Transfer Student

|  |
| --- |
| 表格  描述已自动生成 图形用户界面, 应用程序  描述已自动生成  Professor An has two allocated project, he decided to transfer one of the student to other supervisor.  图形用户界面  中度可信度描述已自动生成 图形用户界面, 表格  中度可信度描述已自动生成  After the Coordinator approve the request, the supervisor of the student will be successfully changed to the new supervisor. |

**3.5 Supervisor create project**

|  |
| --- |
| When two students have registered for the projects of a supervisor, the supervisor’s remaining project will become unavailable. |