






# Declaration of Original Work for 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 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
Dominic Lai Meng Xuan (Group Leader)	SC2002	A34 (3)	 10 Apr 2023
Ezra Koh See Hwa	SC2002	A34 (3)	 10 Apr 2023
Pala Tejaswi	SC2002	A34 (3)	 10 Apr 2023
Pey Ruo-Yang	SC2002	A34 (3)	 10 Apr 2023
Ye Yuhan	SC2002	A34 (3)	 10 Apr 2023

# 1. Design considerations

## 1.1. Approach

Our programme is designed with SOLID and OOP principles in mind.

## 1.2. SOLID

### 1.2.1. Single responsibility principle

Each class in our programme is designed to perform one job. For example, reading and writing of each csv file is done by its respective reader and writer classes. Printing of projects and requests are also handled by separate classes. Each request type is split into its own subclass that only handles its own request.

### 1.2.2. Open-closed principle

The base request class is an abstract class that implements all basic attributes of request such as requestID. All request types inherit from the base class in accordance with the open-closed principle. If additional request types are added, it extends from the base class and no changes are required to other classes. Similarly, the user base class is also an abstract class in which concrete users such as students inherit from. As each user has different attributes, they extend the base class and do not modify it.

### 1.2.3. Liskov substitution principle

As FYP coordinator is also a supervisor, FYP coordinator inherits from the supervisor class. Adhering to Liskov Substitution Principle, FYP coordinator can do everything a supervisor can do, by overriding the display options method and adding additional functions such as generating project details and viewing all requests. The user array that stores user

information upcasts all user objects into the user parent class, so any new user introduced can call the same method and array.

#### 1.2.4. Interface segregation principle

Our original consideration for this principle was to make User and Request classes interfaces, but this was not feasible as according to the principle of Abstraction, they had to store certain common variables from their subclasses, such as userID and password for Users, and pending/approval status for Requests. As a future improvement, we would like to add an interface called Database which contains common database methods such as read and write. This would be implemented by databaseProject, databaseRequest, and databaseUser.

#### 1.2.5. Dependency injection principle

The request parent class implements the abstract settleRequest method to handle requests. Each request type has different implementations when approving requests and will override the abstract method. Each implementation is a higher level method that goes through an abstraction layer to access the lower level implementation.

### 1.3. OOD Principles and other considerations

#### 1.3.1. Encapsulation

All attributes and methods are private by default unless otherwise. Getters and setters are used to initialise or modify attributes. This prevents other classes from accessing unnecessary data.

#### 1.3.2. Abstraction

The FYP app and user classes do not need to know the implementations of methods and attributes and only need to call the relevant methods the user requires. Reading and writing of csv files are also abstracted so the app only calls the run() functions.

#### 1.3.3. Inheritance

All request types and user classes extend their respective parent classes to reduce implementing the same attributes and methods multiple times.

#### 1.3.4. Polymorphism

The request array upcasts all request objects to the parent class to ensure subsequent request types are compatible. The requests stored in the csv file are text and they store the type as an integer value. When reading the csv file, the request types are first initialised as their respective type before being upcast into the array. This ensures that only one csv file is needed. By creating the request as its subtype before upcasting, dynamic binding will override the parent settleRequest() method and call the type specific settleRequest() method.

#### 1.3.5. Other considerations

For simplicity, the write method overrides the previous csv file and creates a new one with the same name. Requests and projects are stored in each user class as an integer, which is the request ID and project ID. When a request is updated such as by settling it or when a project name is changed, only the object in the respective array is updated. As the user only keeps reference to it, we do not have to update the requests and projects in their class as well.

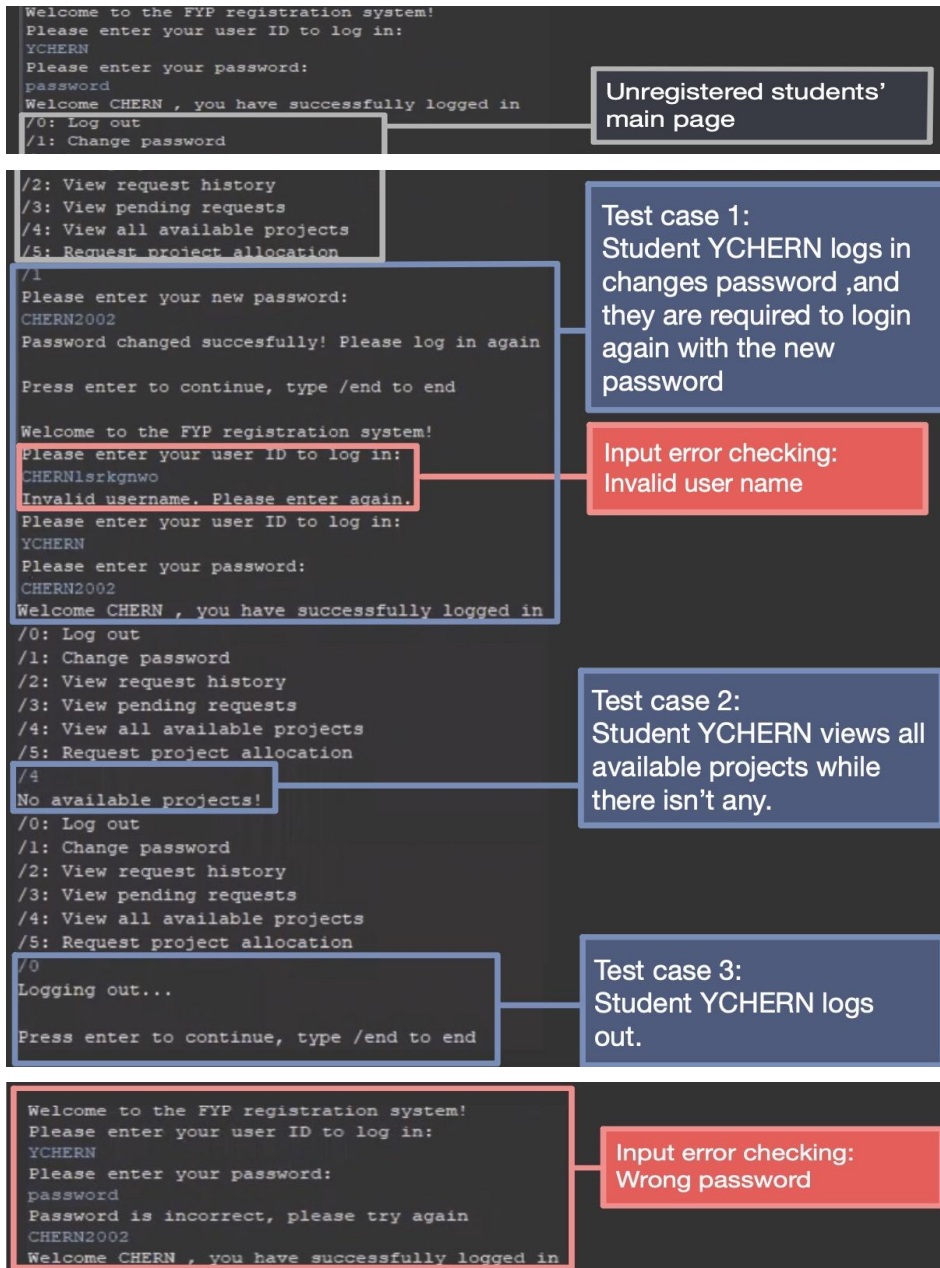
## 2. UML Class Diagram

Refer to the attachment “SC2002\_Group\_Project\_UML.jpg”

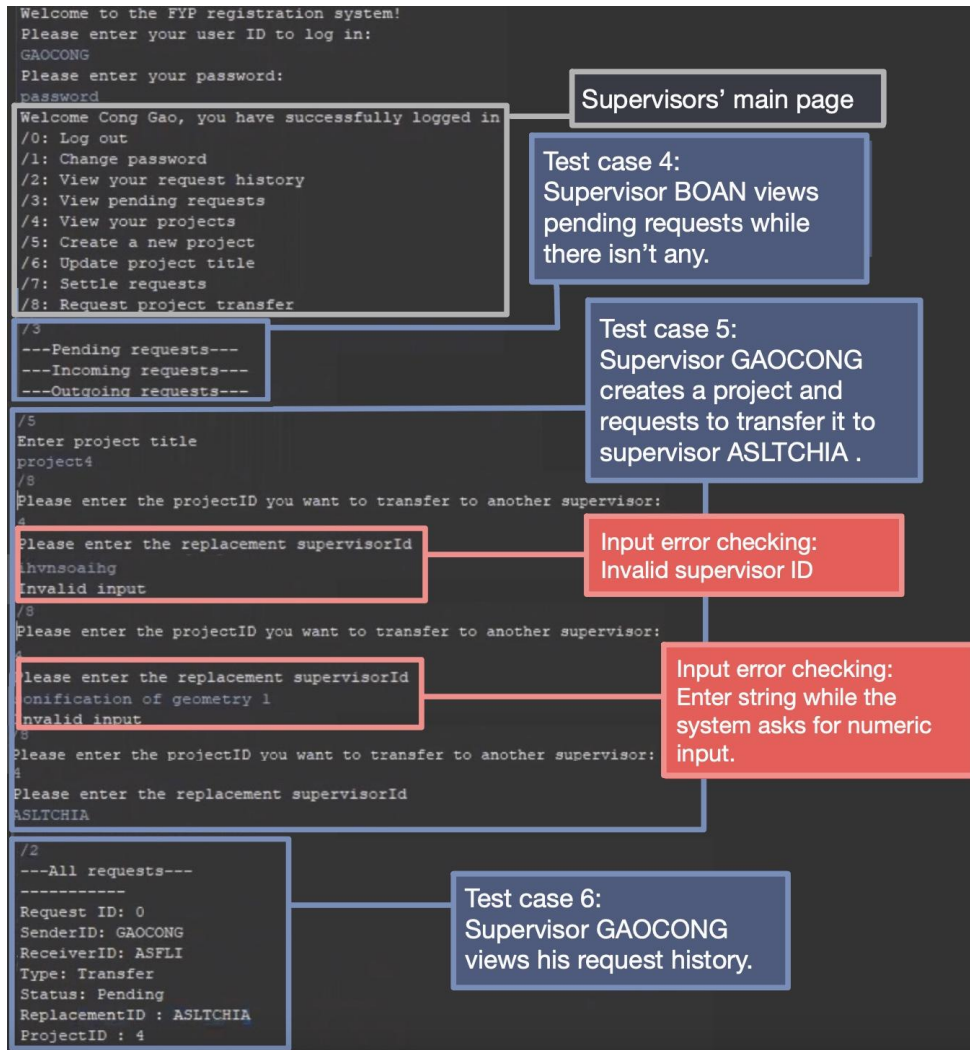
## 3. Test Cases and Results

Below are some of the important test cases. Please refer to our video (SC2002\_Group\_Project\_Video) for the whole testing.

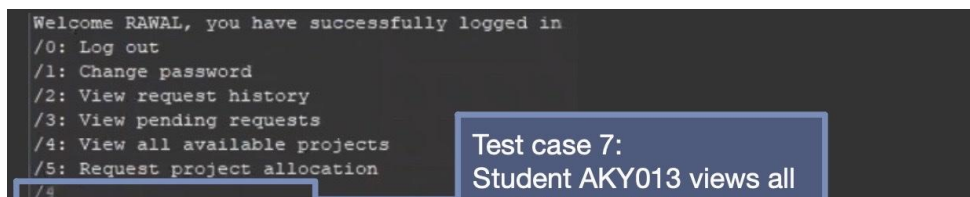
### 3.1. A student's first-time login

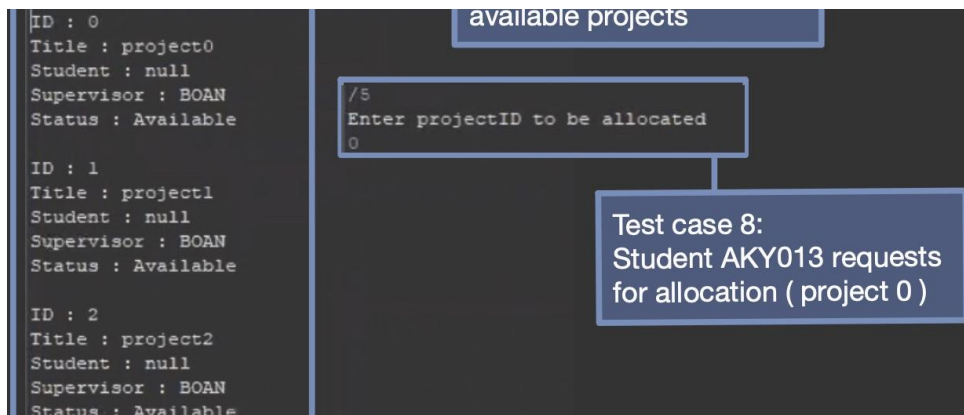


### 3.2. A supervisor's first-time login

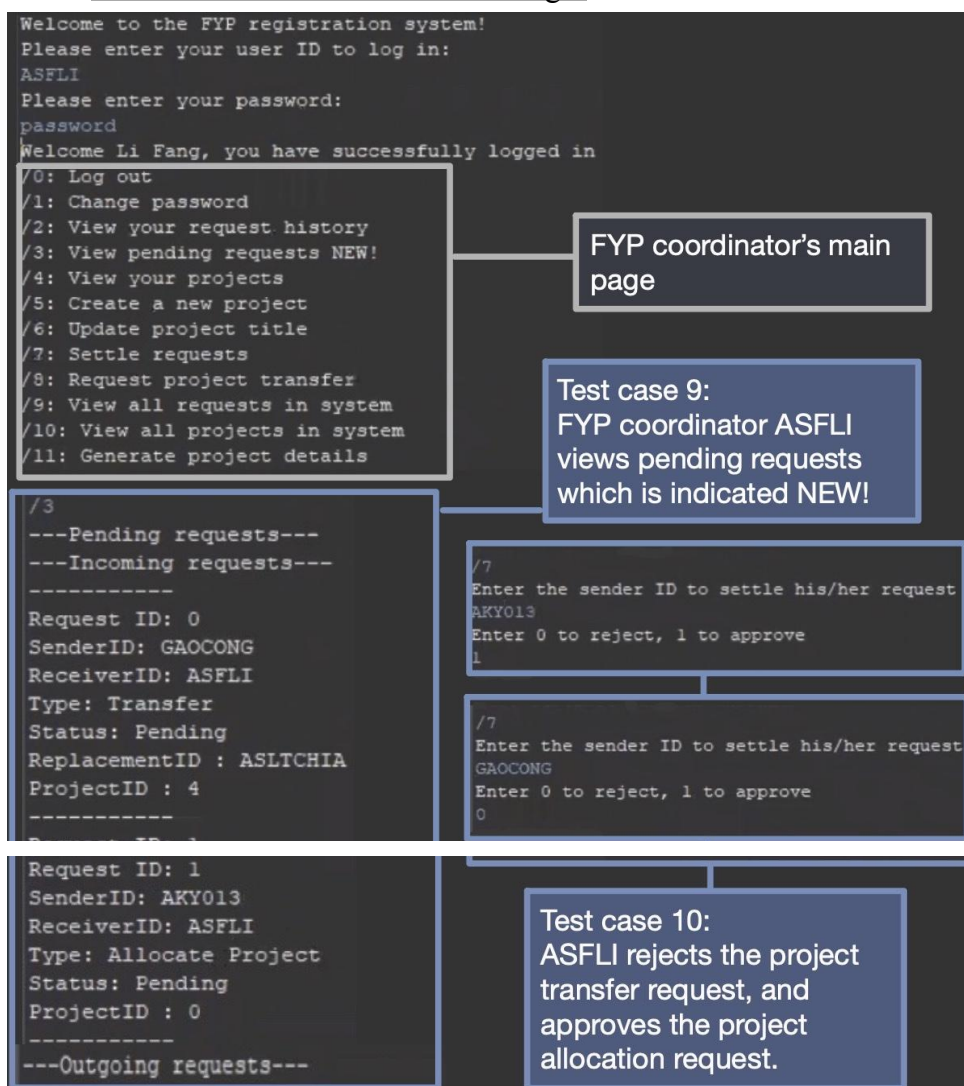


### 3.3. A student's first-time login





### 3.4. The FYP coordinator's first-time login



### 3.5. A registered student login again

```
Welcome to the FYP registration system!
Please enter your user ID to log in:
AKY013
Please enter your password:
password
Welcome RAWAL, you have successfully logged in
/0: Log out
/1: Change password
/2: View request history
/3: View pending requests
/4: View your project
/5: Request to change your project title
/6: Request project deregistration
/5
Enter new title
new title here
/6
Deregistering project request sent
```

Registered students' main page

Test case 11: Student AKY013 requests for title changing.

Test case 12: Student AKY013 requests for deregistration.

### 3.6. A supervisor, who already has projects, login

```
Welcome to the FYP registration system!
Please enter your user ID to log in:
ASMADHUKUMAR
Please enter your password:
password
Welcome A S Madhukumar, you have successfully logged in
/0: Log out
/1: Change password
/2: View your request history
/3: View pending requests: NEW!
/4: View your projects
/5: Create a new project
/6: Update project title
/7: Settle requests
/8: Request project transfer
/3
---Pending requests---
---Incoming requests---
```

Test case 13: Supervisor ASMADHUKUMAR views pending requests which is indicated NEW!

```
Request ID: 12
SenderID: BR015
ReceiverID: ASMADHUKUMAR
Type: Change Title
Status: Pending
Title : another title
---Outgoing requests---
/3
Enter the sender ID to settle his/her request
CT134124124
This student didn't make request to you
/7
Enter the sender ID to settle his/her request
BR015
Enter 0 to reject, 1 to approve
1
/4
ID : 5
Title : another title
Student : BR015
Supervisor : ASMADHUKUMAR
Status : Allocated
/6
Enter the project id:
5
Enter the new title:
Machine Learning
```

Input error checking: Enter the ID of student who doesn't make requests to current supervisor.

Test case 14: ASMADHUKUMAR approves the student's request.

Test case 15: ASMADHUKUMAR views his project. The title has been changed.

Test case 16: ASMADHUKUMAR changes the title of his project.

### 3.7. A deregistered student login



```
Welcome to the FYP registration system!
Please enter your user ID to log in:
SL22
Please enter your password:
password
Welcome LIU , you have successfully logged in

/0: Log out
/1: Change password
/2: View request history
/3: View pending requests
/4: View all available projects
/5: Request project allocation
/5
Since you have previously deregistered a project, you are not allowed to register for another one
```

Deregistered students' main page

Test case 17:  
SL22 requests for project allocation, but he is rejected.

### 3.8. The FYP coordinator login

```
Welcome to the FYP registration system!
Please enter your user ID to log in:
ASFLI
Please enter your password:
password
Welcome Li Fang, you have successfully logged in

/0: Log out
/1: Change password
/2: View your request history
/3: View pending requests
/4: View your projects
/5: Create a new project

/6: Update project title
/7: Settle requests
/8: Request project transfer
/9: View all requests in system
/10: View all projects in system
/11: Generate project details
/11
/1: Generate based on status
/2: Generate based on student id
/3: Generate based on supervisor id
/4: Generate based on project id
/3
Enter supervisor ID
SL22
This user is not a supervisor

/11
/1: generate based on status
/2: generate based on student id
/3: generate based on supervisor id
/4: generate based on project id
/3
Enter supervisor id
ASSOURIN
ID : 6
Title : project6
Student : null
Supervisor : ASSOURIN
Status : Unavailable
ID : 7
Title : Sonification of geometry 1
Student : LES1
Supervisor : ASSOURIN
Status : Allocated
```

Input error checking:  
Enter the ID of a student where requires the supervisor's.

Test case 18:  
FYP coordinator ASFLI generates projects details based on supervisor.

## 4. Reflections

### 4.1. Database classes

We also faced issues with designing our database. We originally wanted to create the required object from the csv only when needed, for example, if the FYPcoordinator approves an allocated project request, only then do we create the project object and the student object and update their fields. However, this leads to many repeated querying of the csv files and we deemed that to be inefficient. Hence, we decided to create all the objects and store them in local arrays upon log in.

### 4.2. Request class

Each type of request requires different data and has different senders and receivers. We originally tried to record each request as a string containing all the information but that was prone to type errors or formatting errors and also violates the Open-Closed Principle as adding a new type of request would be very difficult.

### 4.3. Reading and writing of CSV files

Reading and writing of CSV files was a challenge to implement as we had not worked with files before using java. At first, we tried to download the jar file for opencsv's writer and reader classes but we encountered several different issues and were unable to. In the end, we stuck to the inbuilt reader and writer classes from the java std library. We also faced issues with the design of reading and writing of CSV files. Our original design reads from the csv files on start up and writes on programme termination. Interrupting the programme for any reason such as accidentally closing the console or a mishandled exception would not write to the csv as the write functions were only called when the "end" option was chosen. Thus, this design was infeasible and we decided to call the write functions whenever a user logs out.

## 5. Future Improvements

### 5.1. Improvements to current design

#### 5.1.1. CSV files

Currently, our reading and writing of text files involves overwriting the previous text file and creating the file again with the new data. A possible improvement would be to update the same text file immediately when there are any changes. Currently, the request base class implements all attributes used by the request subtypes. This simplifies the writing of csv files as the method simply creates a reference of the base request class and gets all attributes from it. Future designs could have each subtype implement its necessary attributes and not carry over unused ones. The write method checks the request type and writes the relevant attributes while the unused ones are set to default value.

#### 5.1.2. Error Checking

We used try catch without creating specific error classes. Custom errors such as NoProjectCreated could be created when a supervisor attempts to transfer a project when no project has been created.

#### 5.1.3. Enumeration classes

Our request type and project status are stored and encoded as integers. When printing, we used switch/case statements to print the associated type based on the integer. A better way to do this would be using enumeration classes and an enumeration printer class. This would allow for easier inheritance when creating a new request type and follow open-closed principle.

#### 5.1.4. Database classes

Add an interface called Database which contains common database methods such as read and write. This would be implemented by databaseProject, databaseRequest, and databaseUser.

#### 5.1.5. Display options

The display options for the various types of users could be changed to a separate class. This would make the code easier to read and manage. Furthermore, it would also make it easier to debug when there are problems with the display.

### 5.2. Additional features

- 5.2.1. If there is space available, a sign up feature could be added where users who are not yet registered can apply for an account. The request would be sent to the FYP coordinator who then approves/rejects the application. A forget password feature could also be added. Any users apart from FYP coordinator can make a forget password request to the FYP coordinator to reset their password.