**School of Computer Science and Engineering**

**SC/CE/CZ2002: Object-Oriented Design & Programming**

**Assignment:** *Building an OO Application*

**2022/2023 Semester 2**

| Tutorial/Lab Group | A49 |
| --- | --- |
| Team Number | 2 |

**Members:**

| GERALD ONG JON KAI | U2122864K |
| --- | --- |
| NAHATA ANOUSHKA | U2123824K |

**TABLE OF CONTENTS**

[**1. Application Overview\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3**](#_f110blxps0dh) [**2. Design Consideration \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3**](#_dxei40oz7e9e)

[2.1 Approach Taken\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **3**](#_d2iv09lh526g)

[2.2 Object-Oriented Programming\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **3**](#_qx6i9d610dvr)

[2.3 S.O.L.I.D Principles\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **5**](#_r5gj4tmfqu06)

[2.4 Proposed Future Features\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **7**](#_gzsjcfoyfltv)

[2.5 Assumptions Made\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](#_mpxy8teppljn)  **7**

[**3. Detailed UML Class Diagram\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 8**](#_rm6ku15440wo) [**4. Testing\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 8**](#_hcg829vh3ucx)

[4.1 Change the password\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](#_vxg13isb7lck) 8

[4.2 Viewing available projects and requests\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **10**](#_ti6vbd21q1ao)

[4.3 Viewing Projects by Filter\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **10**](#_bi93svmmf8q0)

[4.4. Detailed Testing Link\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **10**](#_5vm47z35kox3)

[**5. Video Link of Demonstration\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 10**](#_8iwzqu5tlxqw)

[**6. Declaration of Original Work for SC/CE/CZ2002 Assignment\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 11**](#_nzl9s1ze6de8)

# **1. Application Overview**

Final Year Project Management System (FYPMS) is a console-based application, which will computerize the processes to improve efficiency and simplify the Final Year Project (FYP) request process. The application can be used to streamline the registration, allocation, and creation of FYPs. The FYPMS would be used by students, faculty and FYP coordinator.

# **2. Design Consideration**

## **2.1 Approach Taken**

We have implemented the Entity-Controller-Boundary (ECB) design pattern for the FYPMS. The ECB approach has structured our code to separate the different components of the application. This has also helped us achieve the ease of maintenance and modularity.

* **Entity** classes - It represents the real-world objects of the FYPMS like Student, Faculty and Request.
* **Controller** classes - It controls the flow of information between the entities and controllers and handles the logic behind functions.
* **Boundaries** classes - It represents the interface in the FYPMS that enables users like students and faculty to interact with the system.

## **2.2 Object-Oriented Programming**

**2.2.1. Abstraction**

Abstraction refers to a concept of object-oriented programming which simplifies real-world complex problems in the program. It hides unnecessary information and presents the useful attributes to the user, reducing the complexity of the program.

We applied abstraction in the project via our various entity classes. Our abstract classes - *User* and *Request* served as a basic framework for the behaviors and attributes of our real entities. It helps to reduce the complexity of our system by ensuring that only essential methods are being created and called. We also performed abstraction for our project and request managers, making it so that different entities would only function with their own project and request managers. This helped massively in streamlining our code as well as debugging.

**2.2.2 Encapsulation**

Encapsulation is a fundamental concept in object-oriented programming (OOP) that allows the hiding of an object's implementation details while exposing selected information for users to interact with through public interfaces. Encapsulation helps achieve data abstraction and code hiding by binding fields and methods together as a single unit called a class, which is accessed through setter and getter functions.

In FYPMS, encapsulation is applied by keeping all variables in the entities and controllers private and accessing them through setter and getter functions such as *getProject* and *getStudentID*. The entities contain specific setter and getter functions for variables related to the overall class. Another example of encapsulation we performed was the deliberate exclusion of certain functions like *getPassword* to protect sensitive data. Encapsulation was also performed on our project and request managers through interfaces, which allowed a majority of our project’s function to adhere to a specific behavior depending on the current user. Overall, the use of encapsulation in FYPMS helps improve code organization, maintainability, and security.

**2.2.3. Inheritance**

Inheritance refers to the derivation of a class from an already existing class within the hierarchy of the classes by inheriting properties and methods from the parent class. It enables code reusability, the subclasses can leverage on the already existing functionalities of the parent class without having to implement it, thus, making the code shorter.

In FYPMS, student, faculty and coordinator all inherit from the User superclass, which contains many useful getter and setter features that are relevant to all entity types. This is the case also with the Request superclass, with *TransferRequest, AllocRequest, TitleRequest* and *DeregRequest* all inheriting methods from the superclass.

**2.2.4. Polymorphism**

The word ‘poly’ means ‘many’ and ‘morphism’ means forms, thus, polymorphism is the ability of an object or a method to take many forms. Polymorphism helps us access objects of different types throught the same interface. This makes the code more flexible and usable and it can be achieved through overriding or method overloading in the code.

In the context of FYPMS, polymorphism was performed through our interfaces. Functions such as *viewInbox, viewHistory* can have vastly different results when executed on different entities. However, through our interfaces, the system allows both students and faculty to make requests and outputs the relevant data based on the entity. This shows Polymorphism, where request objects can take different forms depending upon the type of user.

## **2.3 S.O.L.I.D Principles**

**2.3.1. Single-Responsibility Principle**

Single Responsibility Principle (SRP) is a design principle, which states that a class should have only a single responsibility and no other, meaning it should have only one reason to change. We have used guiding principles to delegate specific responsibilities to separate classes so that each class has only one reason to change. All the controllers act as individual state controllers for the entity classes and therefore, leads to loose coupling and cohesion.

For example, in the project, classes such as *StudentController* and *FacultyController* exist only to provide the logic between boundaries and managers. Furthermore, we created 3 different request and project managers to ensure that each class was only responsible for the functions of one entity.

**2.3.2. Open-Closed Principle**

The open-closed principle states that software entities should be open for extension but closed for modification. This statement implies that a class should be extendable without modifying the class itself.

For example, our User class can be extended to be FYP coordinator or Faculty. If in future we want to make different user types, we can simply create a new class that extends from user. This design can also be seen in our Request class, which can be extended to each of the 4 different request types. If a new request type was required, for example, to delete a project, it could be implemented easily by extending from Request without any changes to the existing entities.

**2.3.4 Liskov Substitution Principle**

The Liskov substitution principle states that any superclass object should be able to be replaced with objects of a subclass without affecting the functionality of the program. This means that a subclass should provide the same functionality if not more than its superclass, adhere to the interface of the superclass, and must not have stronger preconditions and weaker postconditions than the superclass.

This is demonstrated in our superclasses mentioned earlier, Request and User. For example, in User, this principle was adhered to by design as we let the subclasses directly inherit all of the superclass methods, even instantiation. The same can be said for Request superclass. Since our subclasses have the same parameters and the same functions as the superclass, we adhere strongly to the Liskov Substitution Principle.

**2.3.5 Interface Segregation Principle**

The Interface Segregation Principle states that interfaces should be designed to the needs of the user and not be too bulky – not overloading the user with implementations they may not need. Interfaces should be segregated into smaller, focused interfaces that are streamlined to the user’s needs. This also reduces coupling and hence improves the maintainability of our code.

FYPMS has 3 very distinct interfaces for its functions, splitting project and request managers for each user type. Since the bulk of the program relies on these managers, it is incredibly beneficial to separate them and achieve higher modularity.

**2.3.3. Dependency Inversion Principle**

The dependency inversion principle states that high level modules should not depend upon low level modules. Instead, they should both depend upon abstraction. This implies that when changes are made to low level modules, the high-level modules should not be affected. This allows loose coupling and makes the system more flexible and maintainable.

In the context of FYPMS, we have performed this with the use of a database manager *AccountsController* that is responsible only for the reading and writing of data. This decouples our database operations from controller classes.

## **2.4 Proposed Future Features**

**2.4.1. Multi-threading**

To enhance the FYPMS, we can apply multithreading, where multiple tasks can be performed simultaneously, instead of executing one task at a time. This will make the application faster and more efficient. For implementing multithreading, we can create threads handling smaller and different processes like registering and allocating the FYP that can run concurrently. We can also use synchronized methods and blocks to ensure that threads access shared resources such as FYP request data in a thread-safe manner. Lastly, we need to carefully identify specific tasks that can be parallelized and resources that can be accessed in a synchronized manner to implement multithreading.

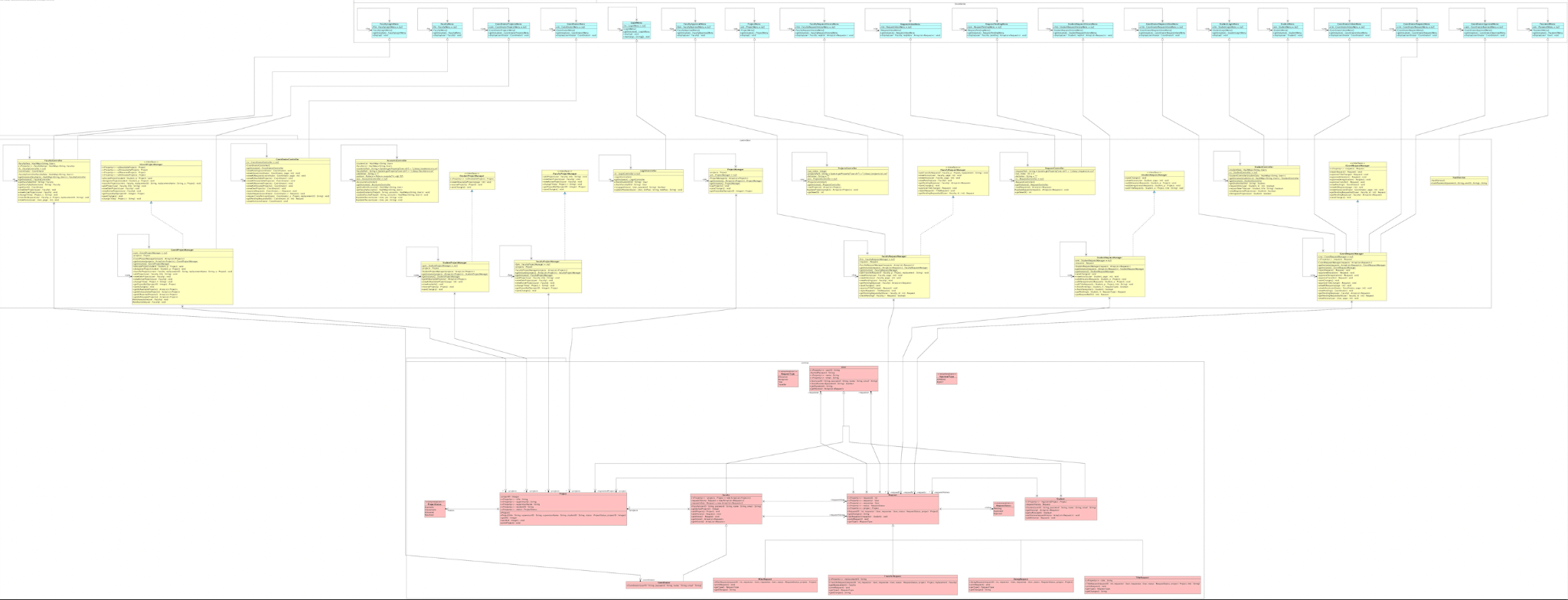
**2.4.2. Integration of Project Management Tools**

One more feature that can improve the FYPMS is the implementation of project management tools. For the implementation of the tools, we can use frameworks or libraries like Agile and Scrum or can implement famous project management tools like Trello, Asana, and Basecamp. These tools help students and supervisors track timelines, progress reporting, and task management. These tools can collaborate with students and faculty to ensure effectiveness.

## **2.5 Assumptions Made**

1. The replacement supervisor would always agree with the transfer of a new student under his/her supervision if he/she has less than 2 active projects.
2. The FYP coordinator can directly approve her own request for transfer of student under replacement supervisor.
3. In current design, all the requests are considered final and cannot be retracted.
4. Ms. Li Fang would serve as the FYP coordinator throughout the time FYPMS’s usage.
5. Projects can have the same name.

# **3. Detailed UML Class Diagram**



[SC2002-FYPManager/UML.jpg at main · geraldd-d/SC2002-FYPManager (github.com)](https://github.com/geraldd-d/SC2002-FYPManager/blob/main/UML.jpg)

The classes in the class diagram are color coded:

- **Yellow -** Controllers in the FYPMS

- **Pink** - Entities in the FYPMS

- **Blue** - Boundaries in the FYPMS

**4. Testing**

| Scenario | Code Output |
| --- | --- |
| A user attempts to login and hits the 5 attempt limit. He logs in and attempts to change his password multiple times, trying to enter his old password, or using a password that is too short. After changing his password, he is logged out and must now use his new password. |  |
| A student with no projects views all available requests, then requests allocation to a project. Now, he tries to view projects or request allocation but can no longer do that. In the coordinator menu, the coordinator can see that the project is now reserved. |  |
| A student with an allocated project makes a title change request, and attempts with various erroneous titles including using the same title again and very short titles. After making the request, he tries to submit another but he already has one pending and is hence prevented from doing so.  Project supervisor menu now has a NEW alert, alerting him of pending requests from students. He goes in and attempts to approve the request. Now, when he views his own projects, the title has changed. This is the same for the student. |  |
| A faculty member attempts to transfer a project to another supervisor who already has 2 projects allocated. The request fails because that supervisor is at maximum load.  He requests a transfer to another supervisor, and the coordinator approves. Now, the coordinator can see that his project is now under Ke Yi Ping.  In this context, A S Madhukumar is attempting to transfer to Bo An, and then to Ke Yi Ping |  |

# 

# **4.4. Detailed Testing Link**

[Custom Test Cases](https://docs.google.com/document/d/14ncxh5pqIaEA6AzjWJDg4Zfqpw1kDFfexiotyOMRWIw/edit?usp=sharing)

# **5. Video Link of Demonstration**

# <https://youtu.be/qaRrolbLsi4>

# **6. 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 honored 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 |
| --- | --- | --- | --- |
| GERALD ONG JON KAI | SC2002 | A49 | 16/04/2023 |
| NAHATA ANOUSHKA | SC2002 | A49 | 16/04/2023 |

Important notes:

1. Name must **EXACTLY MATCH** the one printed on your Matriculation Card.