

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 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 | Lab Group | Signature/ Date |
|-------------------|--------|-----------|-----------------|
| Bryan Lim Kai Wen | SC2002 | A25 | Bryan/16-4-2023 |
| Guo Zhiqi | SC2002 | A25 | Zhiqi/16-4-2023 |
| Lam Wei Lin, Zoey | SC2002 | A25 | Zoey/16-4-2023 |
| Ng Zu Wei, Jovin | SC2002 | A25 | Jovin/16-4-2023 |
| Singh Dhruv | SC2002 | A25 | Dhruv/16-4-2023 |

Table of Contents

| | |
|--|-----------|
| 1. Design Consideration | 2 |
| 1.1. Approach Taken | 2 |
| 1.2. Principles Used | 3 |
| 1.2.1. Single Responsibility Principle (SRP) | 3 |
| 1.2.2. Open-Closed Principle (OCP) | 4 |
| 1.2.3. Liskov Substitution Principle (LSP) | 4 |
| 1.2.4. Interface Segregation Principle (ISP) | 5 |
| 1.2.5. Dependency Inversion Principle (DIP) | 5 |
| 1.3. Assumptions Made | 6 |
| 2. Detailed UML Class Diagram | 6 |
| 3. Testing | 6 |
| 4. Reflection | 11 |
| 4.1. Difficulties Encountered | 11 |
| 4.2. Knowledge Learnt | 11 |
| 4.3. Future Enhancements | 12 |

1. Design Consideration

1.1. Approach Taken

Our primary objective for this application is to achieve loose coupling and high cohesion. To achieve this, we used layered architecture and adhered closely to the SOLID design principles during application development.

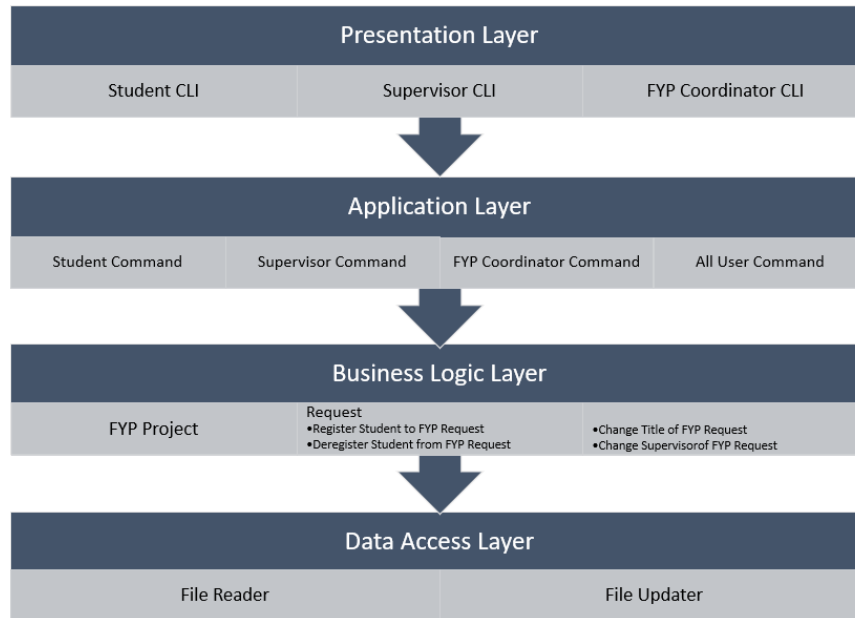


Figure 1. FYPMS Layered Architecture and Components

Our application comprises 4 layers: Presentation layer, Application layer, Business Logic layer and Data Access layer. Each layer satisfies a distinct role and responsibility in the application. Isolating each layer ensures that changes made to any one component of the architecture will not affect components in other layers directly, making it easier to write and develop the application.

The Presentation layer is the user-facing part of the application. It bridges user interactions to the software. There are 3 main components in this layer: Student, Supervisor and FYP Coordinator. After logging in, users will be shown different Command Line Interfaces (CLI) depending on their account type. After that, their respective components will handle the user's interactions with the software.

The Application layer contains the command components: Student Command, Supervisor Command, FYP Coordinator Command and All User Command. This layer lies at the boundary of the application. After receiving user inputs at the Presentation layer, commands from this layer will be used to orchestrate the workflow required to achieve the desired outcome. For example, to allocate an FYP project, the 'ViewAllAvailableFYPCommand' will be instantiated, and its execute() method will be called to facilitate listing all available FYP projects.

The Business Logic layer controls the application functionality. This layer consists of 2 main components: Request and FYP Project. These components work interdependently to fulfil our functionality requirements. Decoupling the components of our application was also a key focus to ensure that the resulting code is easy to maintain and reuse. Subcomponents were also created to further separate responsibilities within each component.

The Data Access Layer comprises components that interact with our database. To store our data, we used Text files as storage. This layer contains the component that loads the database when the application starts and updates the database when the application ends. Upon program runtime, the program reads the input Text files to instantiate the required objects with their relevant attributes. After the application ends, the program updates the Text files with their new attributes.

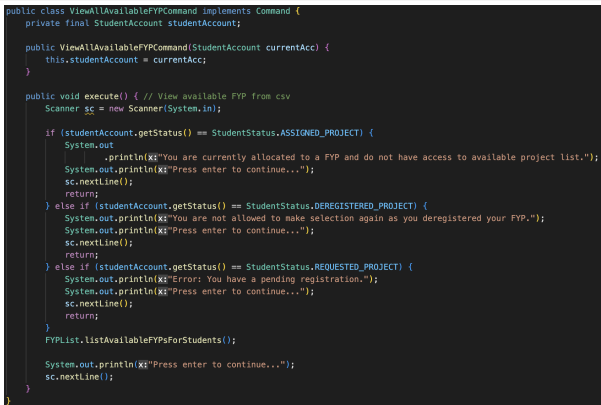

1.2. Principles Used

In our project, we used all 5 of the SOLID principles to ensure that our code was scalable and could deal with all the requirements of the FYP Management System.

1.2.1. Single Responsibility Principle (SRP)

SRP states that every class should have a unique responsibility and that multiple classes should not have the same functionality. To ensure that this was followed throughout our project, we performed a comprehensive distribution of our functionality across our 4 different layers (Figure 1). In each layer, we created unique components for each required function to facilitate clarity and conciseness in what each function would use.

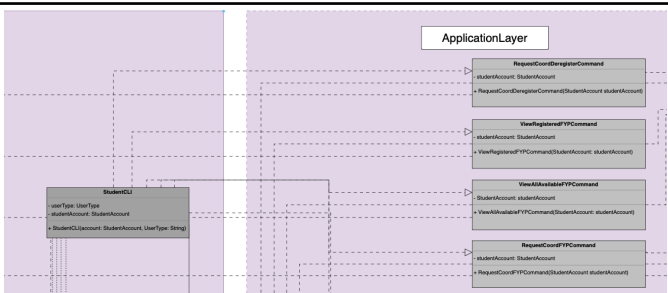
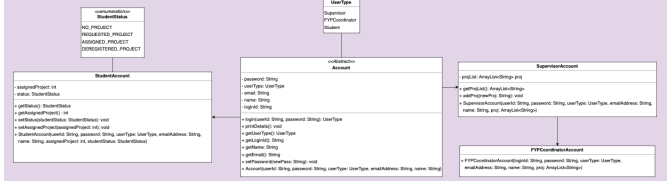
This approach made our code more modular, maintainable, and testable. Each component was also developed, tested, and maintained separately without affecting the other components. It also made errors, exception handling, and bug identification easier since each component had a clearly defined responsibility.

| Example Component adhering to SRP | UML Diagram / Code Snippet |
|--|--|
| <p>The image on the right displays the <i>'ViewAllAvailableFYPCCommand'</i> class which is an implementation of the Command Interface.</p> <p>The class has a single function to print the details of all FYPS that have the AVAILABLE Status.</p> |  <pre> public class ViewAllAvailableFYPCCommand implements Command { private final StudentAccount studentAccount; public ViewAllAvailableFYPCCommand(StudentAccount currentAcc) { this.studentAccount = currentAcc; } public void execute() { // View available FYP from csv Scanner sc = new Scanner(System.in); if (studentAccount.getStatus() == StudentStatus.ASSIGNED_PROJECT) { System.out .println(x:"You are currently allocated to a FYP and do not have access to available project list."); System.out.println(x:"Press enter to continue..."); sc.nextLine(); return; } else if (studentAccount.getStatus() == StudentStatus.DEREGISTERED_PROJECT) { System.out.println(x:"You are not allowed to make selection again as you deregistered your FYP."); System.out.println(x:"Press enter to continue..."); sc.nextLine(); return; } else if (studentAccount.getStatus() == StudentStatus.REQUESTED_PROJECT) { System.out.println(x:"Error: You have a pending registration."); System.out.println(x:"Press enter to continue..."); sc.nextLine(); return; } FYPList.listAvailableFYPSForStudents(); System.out.println(x:"Press enter to continue..."); sc.nextLine(); } } </pre> |
| <p>The image on the right displays the <i>'ViewAllRequestHistoryCommand'</i> class which is an implementation of the Command Interface.</p> <p>The class has a single function to print the details of all requests regardless of the requester, requestee and status.</p> |  <pre> public class ViewAllRequestHistoryCommand implements Command { public ViewAllRequestHistoryCommand() {} public void execute() { ArrayList<Request> requests = RequestHistory.getRequestList(); int empty = 1; for (Request request : requests) { if (request.size() != 0) { empty = 0; for (Request indivRequest : request) { indivRequest.printDetails(); } } } if (empty == 1) { System.out.println(); System.out.println(x:"There is no requests."); } } } </pre> |

1.2.2. Open-Closed Principle (OCP)

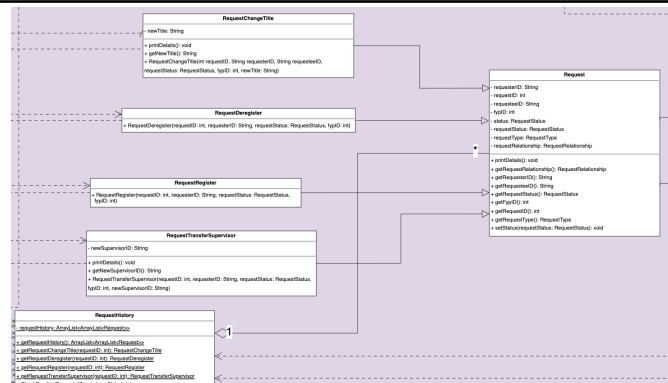
OCP states that a class should be open to extensions of the functionality by adding new specific extensions or implementations which allow conformity to the original class. However, simultaneously, it is closed for modification (i.e. no change to its source code is required). This is visible in our Presentation Layer's various CLIs, which implement the Logout, Menu, and getCommand Interfaces to access the Application Layer without having to modify the classes within the Application Layer (Figure 2).

Another implementation of the OCP is the design of our Account class as an abstract class which was extended to implement different Accounts types for different user types (Student, Supervisor, FYP Coordinator) (Figure 3). This means that for future user types, we could add new subclasses of the abstract Account Class with no modifications to the class itself.

| Example Component adhering to OCP | UML Diagram / Code Snippet |
|--|--|
| Figure 2. Example of the relationship between individual Account Type CLIs and Application Layer commands (StudentCLI shown). |  |
| Figure 3. Abstract Account class and its subclasses, which were extended from it to work with related members without modifications. |  |

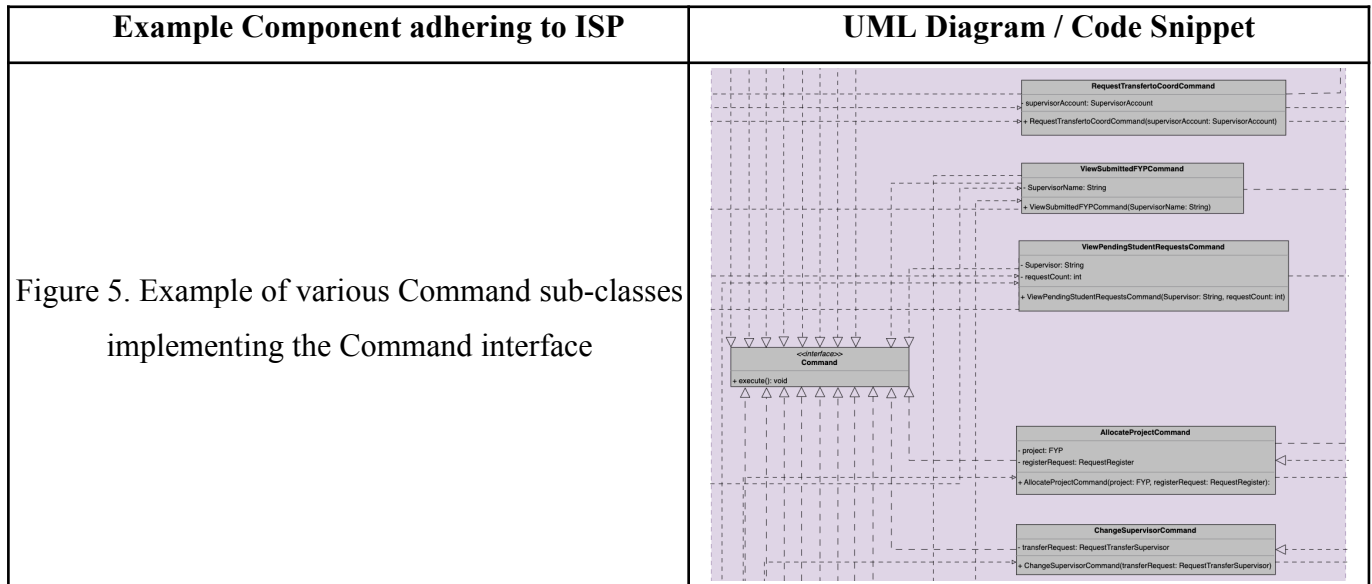
1.2.3. Liskov Substitution Principle (LSP)

For our request class, we used LSP to ensure that all classes of type Request could be substituted with its sub-classes (RequestDeregister, RequestTransfer, RequestRegister, RequestChangeTitle) (Figure 4). This meant that within our storage of requests, we were able to read the data as a nested array of Request objects of different types. This made it more universally usable, prevented redundant implementations of runtime storage structures, and prevented the need for code duplication, making our project easier to scale.

| Example Component adhering to LSP | UML Diagram / Code Snippet |
|---|--|
| Figure 4. Relationship between Request and its subclasses, each of which are replaced by the Request type object in the storage structure of ArrayList<ArrayList<Request>> containing all different types of requests which makes it easier to extend the functionality of the program. |  |

1.2.4. Interface Segregation Principle (ISP)

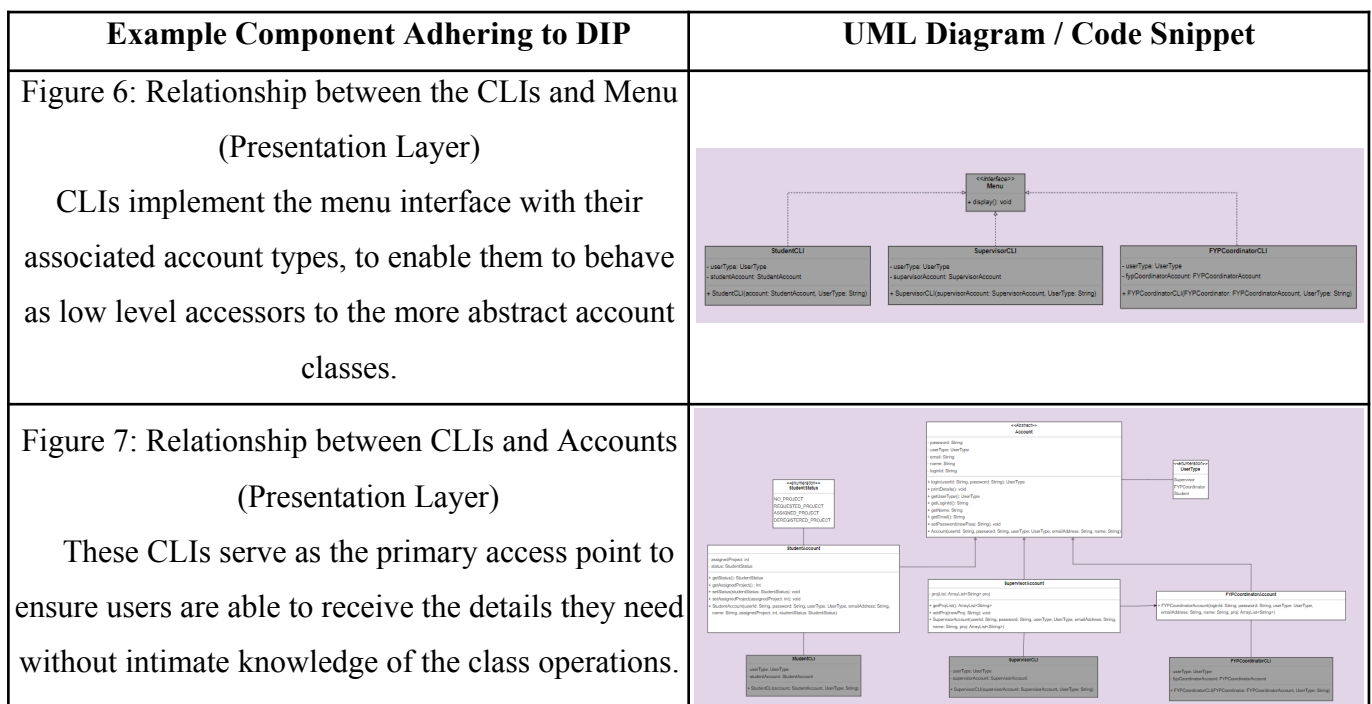
ISP, which states that clients of a class should not be forced to depend on those of its methods that they do not use, was applied to the command, menu and login interfaces within our presentation layer. We used interfaces to ensure that we could scale our necessary components to connect with implementations of these interfaces with only the required features. An example can be found in Figure 5 below.



1.2.5. Dependency Inversion Principle (DIP)

The DIP states that both high-level and low-level classes should depend on abstractions. These abstractions should not depend upon details, but details should depend on abstractions.

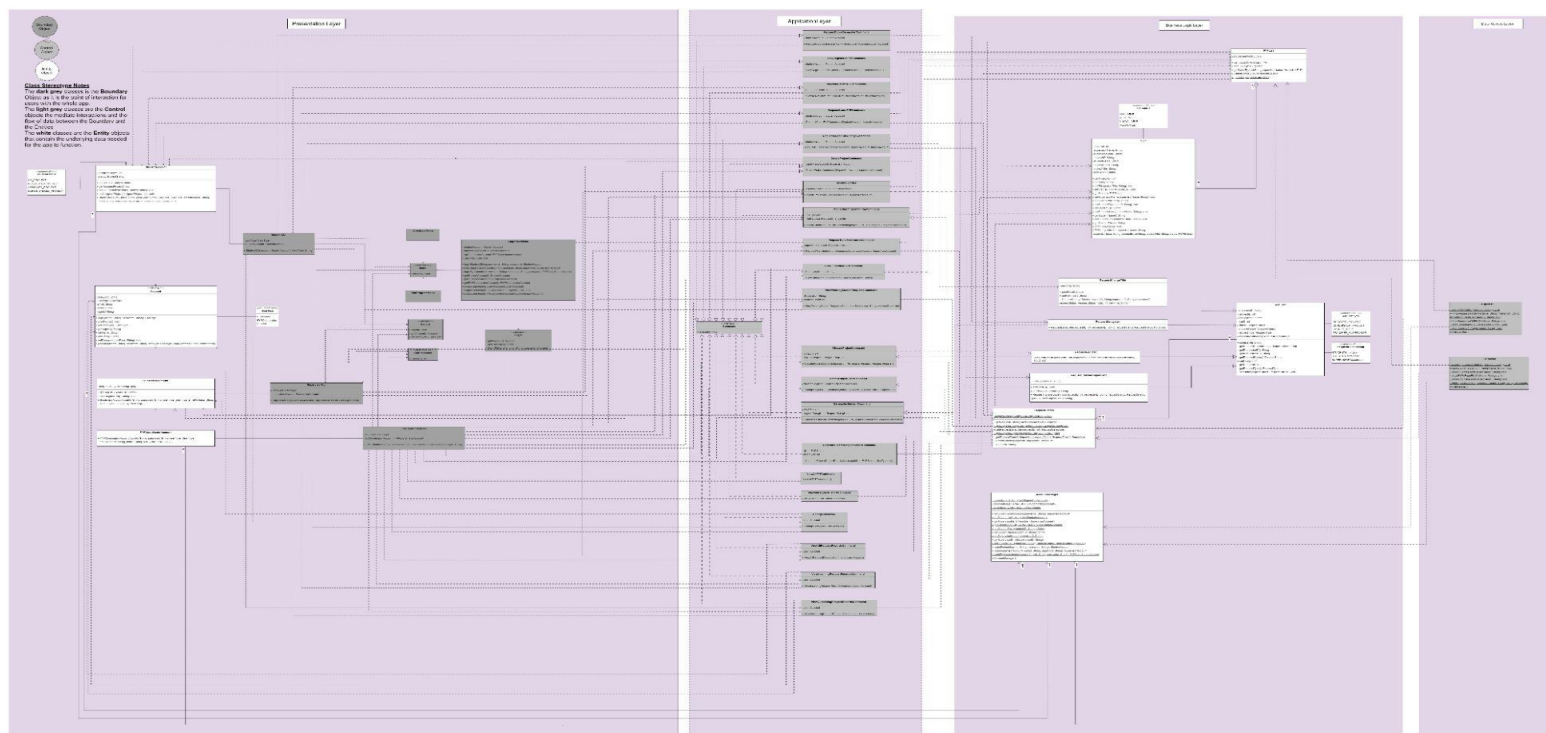
We were able to demonstrate the use of DIP through our various command classes, which allowed users to access the lower-level CLI which implemented the menu interface (Figure 6) and accessed the account classes (Figure 7) to communicate and display the outputs of the functions from the higher-level command class, which were, in turn, accessing the business logic components for the users' respective account types.



1.3. Assumptions Made

1. All users will require logging in to access the functions.
2. This is a single-user application, and there is no need to consider concurrent access.
3. Each StudentID, SupervisorID and Coordinator ID is unique.
4. The number of requests for each type will be at most 1000 as we used the first digit instead of iteration to find the type. This is to reduce complexity and dependencies.
5. The coordinator can overwrite the maximum project limit of 2 when approving project transfer requests.
6. The coordinator will need to make and approve their own requests.

2. Detailed UML Class Diagram



For a clearer view, please open up the attached Scalable Vector Graphics (SVG) file.

3. Testing

| Test Case | Test Case Results |
|--|---|
| Scenario 1: Logging in and changing password Expected Output: User is able to log in successfully before and after changing password | |
| 1. Student inputs incorrect userID or password for login. | Please enter your Login ID: 123 Please enter your Password: 123 Invalid Details. Please try again. |
| 2. Student inputs correct userID and password for login. | Please enter your Login ID: YCHERN Please enter your Password: password ===== StudentAccount Menu ===== Logged in as User: CHERN |

| | |
|---|--|
| 3. Student changes password. | Please enter your Login ID: YCHERN Please enter your Password: password Invalid Details. Please try again. |
| 4. Student inputs userID and previous password for login. | |
| 5. Student inputs userID and new password for login. | Please enter your Login ID: YCHERN Please enter your Password: 123 ===== |
| Scenario 2: Student CHERN deregisters his FYP project | |
| Expected Output: User cannot view and register another FYP after deregistration | |
| 1. CHERN views registered project without having registered for a project. | You have not registered for any FYP. Press enter to continue... |
| 2. CHERN views all available projects. | ===== FYP ID 16 ===== Project ID: 16 Supervisor Name: Ke Yiping, Kelly Supervisor Email: YPKE@NTU.EDU.SG Project Title: Graph-based Deep Models for Image Semantic Segmentation Status: AVAILABLE Student Name: null Student Email: null ===== |
| 3. CHERN selects a project 5 by specifying the projectID, requesting for registration. | Input project ID: 5 Successfully Applied for project 5 |
| 4. CHERN is no longer able to view all available projects after FYP coordinator approves the request. | You are currently allocated to a FYP and do not have access to available project list. |
| 5. CHERN views registered project. | Project ID: 5 Supervisor Name: Bo An Supervisor Email: BOAN@NTU.EDU.SG Project Title: Deep Reinforcement Learning for Complex Environment Status: ALLOCATED Student Name: CHERN Student Email: YCHERN@e.ntu.edu.sg; |
| 6. CHERN requests to change the title by providing a new title. | Input new project title: New Title Request for title change submitted. |
| 7. After the Supervisor approves the request, CHERN views the registered project to verify the title change. | Project ID: 5 Supervisor Name: Bo An Supervisor Email: BOAN@NTU.EDU.SG Project Title: New Title Status: ALLOCATED Student Name: CHERN Student Email: YCHERN@e.ntu.edu.sg; |
| 8. CHERN requests to change project title to New title 2. | Input new project title: New title 2 Request for title change submitted. |
| 9. CHERN requests to deregister the project. | ===== |
| 10. After the FYP coordinator approves the deregister request, CHERN cannot view their previously registered project. | Successfully Applied to deregister for project 5 |
| 11. CHERN views request history. Request to change title has been automatically rejected. | You have not registered for any FYP. |
| | ===== Request ID 1 ===== Requester: YCHERN Requestee: BOAN Request Type: CHANGE_TITLE Request Status: REJECTED Request Relationship: STUDENTSupervisor FYP ID: 5 New Project Title: New title 2 |

| | |
|---|--|
| 12. CHERN cannot view all available projects. | You are not allowed to make selection again as you deregistered your FYP. |
| Scenario 3: Handling Supervisor Bo An and Dusit Niyato's Project cap Expected Output: User is unable to view supervisor's projects once supervisor has reached their cap | |
| 1. Both Student KOH and Student CHERN are registered for Bo An's project. Student BRANDON views all available projects. Bo An's remaining project (project ID 7) is NOT included in the list. | <pre> ===== FYP ID 4 ===== Project ID: 4 Supervisor Name: Arvind Easwaran Supervisor Email: ARVINDE@NTU.EDU.SG Project Title: Edge/Cloud Resource Management for Time-Sensitive Applications (2) Status: AVAILABLE Student Name: null Student Email: null ----- ===== FYP ID 8 ===== Project ID: 8 Supervisor Name: Cai Wentong Supervisor Email: ASWTCAT@ntu.edu.sg Project Title: Creation of Meta-model for Agent-based Simulation Using Machine Learning Approach Status: AVAILABLE Student Name: null Student Email: null ----- </pre> |
| 2. Bo An transfers a project to Dusit Niyato. 3. Student BRANDON view all available projects, Bo An's remaining project (project ID 7) is included in the list. | <pre> ===== FYP ID 4 ===== Project ID: 4 Supervisor Name: Arvind Easwaran Supervisor Email: ARVINDE@NTU.EDU.SG Project Title: Edge/Cloud Resource Management for Time-Sensitive Applications (2) Status: AVAILABLE Student Name: null Student Email: null ----- ===== FYP ID 7 ===== Project ID: 7 Supervisor Name: Bo An Supervisor Email: BOAN@NTU.EDU.SG Project Title: Designing Negotiation Agents to Participate in In ternational Competition Status: AVAILABLE Student Name: null Student Email: null ----- </pre> |
| 4. Student BRANDON is registered to Dusit Niyato's project (project ID 15). 5. Dusit Niyato submits a new project. | <pre> Input Title of FYP: Scenario 2 DNIYATO Project Warning! You are already in charge of at least 2 projects Proceeding to make new project unavailable... </pre> |
| 6. Student LIU views all available projects, Dusit Niyato's remaining project (project ID 17) is NOT included in the list. | <pre> ===== FYP ID 16 ===== Project ID: 16 Supervisor Name: Ke Yiping, Kelly Supervisor Email: YPKE@NTU.EDU.SG Project Title: Graph-based Deep Models for Image Semantic Segmentation Status: AVAILABLE Student Name: null Student Email: null ----- ===== There are 13 Final Year Projects available! ===== </pre> |

| | |
|--|--|
| 7. Student BRANDON deregisters FYP. Student LIU views all available projects. Dusit Niyato's remaining projects including the deregistered project will be displayed in the available project list. | <pre> ===== FYP ID 15 ===== Project ID: 15 Supervisor Name: Dusit Niyato Supervisor Email: DNIYATO@NTU.EDU.SG Project Title: Metaverse for virtual education 1 Status: AVAILABLE Student Name: null Student Email: null ----- ===== FYP ID 16 ===== Project ID: 16 Supervisor Name: Ke Yiping, Kelly Supervisor Email: YPKE@NTU.EDU.SG Project Title: Graph-based Deep Models for Image Semantic Segmentation Status: AVAILABLE Student Name: null Student Email: null ----- ===== FYP ID 17 ===== Project ID: 17 Supervisor Name: Dusit Niyato Supervisor Email: DNIYATO@NTU.EDU.SG Project Title: Scenario 2 DNIYATO Project Status: AVAILABLE Student Name: Student Email: </pre> |
| 8. Student LIU selects the recycled project. | <pre> Input project ID: 15 Successfully Applied for project 15 </pre> |
| <u>Scenario 4: Creation of new project and its subsequent approval and title change</u> Expected Output: Newly created project can be selected by student and its title can be changed subsequently | |
| 1. FYP Coordinator creates project | <pre> * Input Title of FYP: NLP For Beginners Project has successfully been created! ===== FYP ID 17 ===== Project ID: 17 Supervisor Name: Li Fang Supervisor Email: ASFLI@NTU.EDU.SG Project Title: NLP For Beginners Status: AVAILABLE Student Name: Student Email: </pre> |
| 2. A student selects the project | <pre> Input project ID: 17 Successfully Applied for project 17 </pre> |
| 3. FYP Coordinator approves request | <pre> ===== Allocated project NLP For Beginners to CHERN </pre> |
| 4. Student requests to change title | <pre> Input new project title: NLP For Dummies Request for title change submitted. </pre> |
| 5. FYP Coordinator approves change | <pre> ===== Project title has been change to NLP For Dummies </pre> |
| 6. Student's project title has changed | <pre> Project ID: 17 Supervisor Name: Li Fang Supervisor Email: ASFLI@NTU.EDU.SG Project Title: NLP For Dummies Status: ALLOCATED Student Name: CHERN Student Email: YCHERN@e.ntu.edu.sg; </pre> |
| <u>Scenario 5: View filtered projects, view self request history and view all request history</u> Expected Output: Project filtered by supervisor and project status. Only Coordinator can view all request history | |
| 1. CHERN registers for project ID 5, and gets approved 2. BRANDON registers for project ID 6, and is pending approval 3. Coordinator filters project by supervisor. | <pre> ===== Enter Supervisor's name: (1) A S Madhukumar (2) Alexei Sourin (3) Arvind Easwaran (4) Bo An (5) Cai Wentong (6) Chen Change Loy (7) Chia Liang Tien (8) Cong Gao (9) Douglas Leslie Maskell (10) Dusit Niyato (11) Ke Yiping, Kelly Enter the number of your choice: ===== </pre> |

| | |
|--|--|
| <p>4. Select Supervisor Bo An to view his projects.</p> | <pre> List of all Final Year Projects ===== FYP No. 1 ===== Project ID: 5 Supervisor Name: Bo An Supervisor Email: BOAN@NTU.EDU.SG Project Title: Deep Reinforcement Learning for Complex Environment Status: ALLOCATED Student Name: CHERN Student Email: YCHERN@ntu.edu.sg; ===== FYP No. 2 ===== Project ID: 6 Supervisor Name: Bo An Supervisor Email: BOAN@NTU.EDU.SG Project Title: Build Software Agents for Power Trading Agent Competition Status: RESERVED Student Name: null Student Email: null ===== FYP No. 3 ===== Project ID: 7 Supervisor Name: Bo An Supervisor Email: BOAN@NTU.EDU.SG Project Title: Designing Negotiation Agents to Participate in International Competition Status: AVAILABLE Student Name: null Student Email: null ===== There are 3 Final Year Projects! ===== </pre> |
| <p>5. CHERN logs in and create request to change title. 6. Bo An approves and transfer project to Dusit Niyato. 7. Bo An view incoming request history.</p> | <pre> Request History ===== Request ID 0 ===== Requester: YCHERN Requestee: BOAN Request Type: CHANGE_TITLE Request Status: APPROVED Request Relationship: STUDENTSupervisor FYP ID: 5 New Project Title: Machine Learning You have 1 incoming requests. </pre> |
| <p>8. Coordinator logs in and view all request history</p> | <pre> Requester: YCHERN Requestee: BOAN Request Type: CHANGE_TITLE Request Status: APPROVED Request Relationship: STUDENTSupervisor FYP ID: 5 New Project Title: Machine Learning Requester: YCHERN Requestee: ASFLI Request Type: REGISTER_PROJECT Request Status: APPROVED Request Relationship: STUDENTCoordinator FYP ID: 5 Requester: BR015 Requestee: ASFLI Request Type: REGISTER_PROJECT Request Status: PENDING Request Relationship: STUDENTCoordinator FYP ID: 6 Requester: BOAN Requestee: ASFLI Request Type: TRANSFER_SUPERVISOR Request Status: PENDING Request Relationship: SUPERVISORCoordinator FYP ID: 5 New Supervisor ID: DNIYATO </pre> |
| <p><u>Scenario 6: Transfer of project to supervisor with project capacity</u> <u>Expected Output: Message prompt to Coordinator to confirm approval. Project exceeds capacity if approved.</u></p> | |
| <p>1. CHERN registers for project ID 5 and gets approved. 2. KOH registers for project ID 6 and gets approved 3. BRANDON registers for project ID 15 and gets approved. 4. Dusit Niyato requests to transfer project ID 15 to Bo An.</p> | <pre> Input project ID to transfer: 15 Input new supervisor ID: BOAN </pre> |
| <p>5. Coordinator tries to approve the request 6. Coordinator approves the request</p> | <pre> Warning: BOAN is already in charge of at least 2 projects Do you wish to proceed? Y/N y Bo An has a new project Deep Reinforcement Learning for Complex Environment Current Supervisor is: Bo An Change in progress </pre> |

| | |
|---|---|
| 7. Coordinator selects to filter projects by status. | <div style="background-color: #002060; color: white; padding: 10px;"> <p>Filter by:</p> <p>(1) Available</p> <p>(2) Reserved</p> <p>(3) Unavailable</p> <p>(4) Allocated</p> <p>Enter the number of your choice:</p> <p>=====</p> </div> |
| 8. Coordinator selects Allocated. Bo An is the supervisor for all Allocated projects (Project ID 5, 6, 7) | <div style="background-color: #002060; color: white; padding: 10px;"> <p>List of allocated Final Year Projects</p> <p>===== FYP No. 1 =====</p> <p>Project ID: 5 Supervisor Name: Bo An Supervisor Email: BOAN@NTU.EDU.SG Project Title: Deep Reinforcement Learning for Complex Environment Status: ALLOCATED Student Name: KOH Student Email: KOH1@e.ntu.edu.sg;</p> <p>===== FYP No. 2 =====</p> <p>Project ID: 6 Supervisor Name: Bo An Supervisor Email: BOAN@NTU.EDU.SG Project Title: Build Software Agents for Power Trading Agent Competition Status: ALLOCATED Student Name: CHERN Student Email: YCHERN@e.ntu.edu.sg;</p> <p>===== FYP No. 3 =====</p> <p>Project ID: 15 Supervisor Name: Bo An Supervisor Email: BOAN@NTU.EDU.SG Project Title: Metaverse for virtual education 1 Status: ALLOCATED Student Name: BRANDON Student Email: BR015@e.ntu.edu.sg;</p> <p>===== There are 3 Final Year Projects allocated! =====</p> </div> |

4. Reflection

4.1. Difficulties Encountered

During the project, we encountered difficulties adhering to SOLID principles while creating the UML diagram and implementing its corresponding code.

Notably, we needed help accommodating the different types of requests. Since similar methods were needed for all request types, implementing each type as its own class would be inefficient. Thus, we created the individual classes through inheritance from a general Request class. The ability to reuse code inherited from the superclass increased efficiency as similar methods did not need to be rewritten for each request type.

Additionally, to further avoid tight coupling and achieve loose cohesion, we decided to utilise interfaces. This allowed for greater adherence to SOLID principles, as mentioned in **Section 1.2**.

4.2. Knowledge Learnt

An important concept we learnt to use is inheritance. Inheritance is a fundamental concept in object-oriented design and programming (OODP) that allows new classes to be based on existing classes, inheriting their attributes and methods. We first created abstract classes and interfaces that define a set of common behaviours for a group of related classes, such as Menu for the CLIs and Account for the different account types. This creates code that is more flexible and adaptable to changing requirements, without having to change the underlying implementation. The flexibility makes the code reusable for the creation of related classes, saving time and effort. The modular structure for these classes also makes the code easier to understand, maintain, and modify, as each class can be tested and debugged independently.

Another important principle we learnt in OODP is runtime and compile-time polymorphism. Polymorphism allows objects of different classes to be treated as if they were of the same class. By designing polymorphic objects, we were able to create more modular and loosely coupled (flexible) code that can be easily reused, modified or extended without breaking the existing code. For example, our Request class enabled runtime polymorphism through the creation of a nested array list which could conform to its superclasses.

Additionally, we also learnt that object class is the root of the class hierarchy as every class has an Object as a superclass. We could store objects of different Requests types into an ArrayList by utilising the Object class.

Lastly, we learnt that ConcurrentModificationException would occur when the object being iterated on is being modified in Java. As a result, to circumvent this error, we had to keep track of changes within the iteration before modifying the object after the iteration has completed.

4.3. Future Enhancements

Since the Account class supports the **OCP**, it can easily accommodate new types of accounts by extending them from the Account class. Additionally, as an interface, the Menu class supports the **OCP**. This allows the implementation of different CLIs for any additional types of users from the Menu interface.

Additionally, the CLI could be replaced by a Graphical User Interface (GUI) for user-friendliness. A visual representation of the system would make it easier for users to navigate and interact with the computer. Additionally, using buttons and icons to represent functions allows users to easily see what they are doing, reducing the risk of errors. The visual representation of the system's functions also allows for accessibility for individuals with visual disabilities, as they can navigate and interact with the computer with greater ease.

Furthermore, instead of reading in text files to instantiate the database, we could read in Excel files. Excel is one of the most common and favourite file formats for storing data. Thus, the ability to read in Excel files would make it easy for administrators as they would not need to convert their Excel files into text files for use in the application. Furthermore, excel files allow users to easily manipulate data outside the program using various tools, such as sorting, filtering, and searching. Thus, it is easy for them to find or edit any data if there need to be any changes.

To further enhance our database, a dedicated database such as a SQL database could allow easier real-time updates. Multithreading could also be implemented to facilitate concurrent access and modification of the system, allowing multiple users to log in concurrently whilst processing requests and updates in real time.

Lastly, behavioural and structural patterns could also be considered together with the SOLID principles to increase the adaptability of our code. Some examples of pattern concepts we could include are: chains of responsibility, and states among various structural and behavioural properties of OODP.