

SC2002: OBJECT ORIENTED DESIGN AND PROGRAMMING

AY22/23 SEMESTER 1 GROUP ASSIGNMENT

**Camp Application and Management System (CAMs)**

DATE OF SUBMISSION: 26 NOVEMBER 2023

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 Module | Lab Group | Signature/Date |
| Leong Wei Hong | SC2002 | SCS1 | 23/11/2023 |
| Lim Hao Zheng | SC2002 | SCS1 | 23/11/2023 |
| Cho Zen Han | SC2002 | SCS1 | 23/11/2023 |
| Keefer Loo Guo Hao | SC2002 | SCS1 | 23/11/2023 |

Table of Contents

[1. Design Considerations 2](#_Toc151898920)

[1.1 Approach 2](#_Toc151898921)

[1.2 Assumptions 2](#_Toc151898922)

[1.3 Design Principles 3](#_Toc151898923)

[1.3.1 SOLID Design Principles 3](#_Toc151898924)

[Single Responsibility Principle (SRP) 3](#_Toc151898925)

[Open Closed Principle (OCP) 3](#_Toc151898926)

[Liskov Substitution Principle (LSP) 4](#_Toc151898927)

[Interface Segregation Principle (ISP) 4](#_Toc151898928)

[Dependency Injection Principle (DIP) 4](#_Toc151898929)

[1.3.2 Other Design Principles 5](#_Toc151898930)

[Law of Demeter (LoD) 5](#_Toc151898931)

[1.4 Object-Oriented Programming Concepts 5](#_Toc151898932)

[Abstraction 5](#_Toc151898933)

[Encapsulation 5](#_Toc151898934)

[Inheritance 5](#_Toc151898935)

[Polymorphism 6](#_Toc151898936)

[2. Reflection 6](#_Toc151898937)

[3. UML Class Diagram 7](#_Toc151898938)

[4. Test Case Demonstration 8](#_Toc151898939)

[5. Appendix 12](#_Toc151898940)

# 

# 1. Design Considerations

## 1.1 Approach

Approaching the development of the Camp Application and Management System (CAMs) as outlined in the assignment requires a systematic, step-by-step strategy. First and foremost, a thorough understanding of the requirements is crucial. Breaking down the functionalities into distinct components like user authentication, camp creation, registration, and reporting, will guide the overall design process. Adopting a modular approach with an object-oriented mindset will be pivotal. This involves identifying key entities such as users (staff, students, camp committee members) and camps, along with their respective attributes and behaviours. Leveraging Object-Oriented Design Programming (OODP) principles like encapsulation, inheritance, and polymorphism will aid in creating a cohesive and extensible class structure. Prioritizing clean code, adherence to Java conventions, and robust error handling will ensure the reliability and maintainability of the system. Iterative development and testing, especially focusing on user interactions and edge cases, will be pivotal to validate and refine the system’s functionalities. Lastly, comprehensive documentation and reflection on design choices, challenges faced, and learning outcomes will form the backbone of the final report, presenting a holistic view of the development process.

## 1.2 Assumptions

The following are the assumptions made while designing the system:

1. IDs for both Staff and Student are unique and fixed.
2. Camp name/ID are unique and fixed.
3. All the faculty information would already be inside the database (closed system), users are not expected to provide any information. For example, registering for a new account.
4. Only one user can use the system at any given time. (no concurrent usage)
5. Participants of a camp cannot register as a camp committee.
6. Registration of camp and camp committee is automatic as long as there is vacancy.
7. The number of camp committee is counted into total slots.
8. A camp is either open to a single faculty or the whole school.
9. Staff are not allowed to transfer their camp to another staff. Staff only create, edit, and delete their own camps.
10. When a suggestion is approved, it is assumed that the respectively Staff will go through the edit process and make the amendment by themselves.
11. Committee members have full knowledge of their own camps and thus, do not need an enquiry option.
12. Every camp has a minimum of 2 attendee slots, 1 camp participant and 1 camp committee member.

## 1.3 Design Principles

### 1.3.1 SOLID Design Principles

#### 

|  |  |
| --- | --- |
| **SOLID Principle** | **Example Code (full image of example codes will be in the images folder)** |
| Single Responsibility Principle (SRP) |  |
| **Description**  SRP emphasises a class's singular responsibility, ensuring clarity, maintainability, and minimized interdependence among software components by assigning distinct functionalities to each module or class.  Within our application, Staff members utilize the EditCampOperation interface to execute multiple camp editing functions. Among these, EditRegDeadline is a specific class dedicated to altering the registration deadline of a camp. Each inherited class of the interface adheres to the Single Responsibility Principle (SRP) by focusing on distinct roles and responsibilities within the system. | |
| Open Closed Principle (OCP) |  |
| **Description**  OCP emphasises that created entities (such as classes, modules, functions, etc.) should be open for extension but closed for modification. This means that it allows the addition of new functionality without changing existing code. The key concept in this principle is abstraction.  Our abstract class "Storage" serves as the primary structure for various storage types such as StaffStorage, StudentStorage, and CampStorage, aligning with the Open-Closed Principle (OCP). This design enables the seamless addition of new storage types without necessitating modifications to the main class or any existing subclasses. | |
| Liskov Substitution Principle (LSP) | (Same images as SRP) |
| **Description**  LSP emphasises the idea that objects of a superclass should be replaceable with objects of its subclasses without impacting the capability of the program. Where the subclass must do all the things the superclass does, and it should not generate any errors that the superclass does not.  The EditCampOperations interface features a sole method, perform(), consistently implemented across all its implementing classes, ensuring conformity to the superclass. Additionally, every class conducts null exception checking, leaving no instances where this check is omitted. This adherence to uniformity and error handling across implementations aligns with the Liskov Substitution Principle (LSP). | |
| Interface Segregation Principle (ISP) |  |
| **Description**  ISP emphasises that larger interfaces should be split into smaller ones whereby many specific interfaces are better than a general interface. By doing so, we can ensure that implementing classes only need to be concerned about the methods that are of interest to them. The main idea is to avoid forcing classes realising the interface from implementing methods that they do not use.  In our code examples, we have implemented DataWriter, PerformanceReportWriter, and AttendeeListWriter, each tailored for distinct functionalities. Although all 3 classes are used for I/O write operations, the method signatures used for the 3 scenarios are different. Generalizing the 3 methods into a single interface will force classes realising the interface into implementing all 3 methods even if they only use one. We applied the ISP by breaking down our interfaces into specific ones, aligning with the specific responsibilities of each writer rather than a generalized interface. | |
| Dependency Injection Principle (DIP) |  |
| **Description**  DIP emphasises that high-level modules should not depend on low-level modules but instead on abstraction. The main idea is to depend on interfaces which have a lower probability of being edited.  In our code example, the StaffAccount class aligns with DIP by relying on interfaces like EditCampOperation and PerformanceReportWriter. Staff members utilize these interfaces for editing camps or generating performance reports, adhering to the principle of maximizing class dependency on interfaces. This practice fosters a more modular design, facilitating the addition of new class implementations for these interfaces, such as a new performance report format writer without disrupting main interfaces or subclasses. | |

### 1.3.2 Other Design Principles

#### Law of Demeter (LoD)

Also known as the Principle of Least Knowledge, LoD encourages loose coupling between classes by limiting the interaction between objects to only necessary dependencies.

Our project adheres to the Law of Demeter, emphasizing minimal coupling between classes through specific responsibilities assigned to each. Each class is segregated, maintaining distinct roles and responsibilities, thereby avoiding long chains of method calls and preventing heavy reliance on any single class. Moreover, our application of DIP prioritizes interface implementation, minimizing dependencies on specific classes. This strategic use of interfaces fosters loose coupling, reinforcing the principle of least knowledge across the system's components.

## 1.4 Object-Oriented Programming Concepts

### Abstraction

Abstraction in our code manifests through abstract classes like "Storage," outlining common features shared by various storage types, allowing subclasses to inherit these traits while implementing their specifics. Interfaces, such as EditCampOperation, serve as abstract blueprints defining method structures for classes to adhere to, fostering uniformity. This approach promotes modularity, maintainability, and adaptability across our software design.

### Encapsulation

Encapsulation builds a barrier to protect an object’s private data. We do this by declaring all our attributes as “private,” it protects the attribute from any unwanted changes. If a user wants to access or modify the private data, it can only be done through the public methods, such as get and set methods, of the object’s class.

### Inheritance

Inheritance plays a significant role in coding by facilitating the creation of hierarchical relationships among classes. For instance, abstract classes like "Storage," serve as a base structure for various specific storages like StaffStorage, StudentStorage, and CampStorage. This hierarchy allows these specific storages to inherit common attributes or methods from the abstract Storage class while also implementing their unique functionalities.

### Polymorphism

Polymorphism in our code enables objects of different classes to be treated as objects of a common superclass, promoting flexibility and extensibility. For instance, consider how various classes implementing the EditCampOperation interface can be utilized interchangeably, allowing diverse editing functionalities while adhering to the same interface. This polymorphic behaviour allows for the generic handling of different objects through a shared interface, facilitating code reuse and scalability.

# 2. Reflection

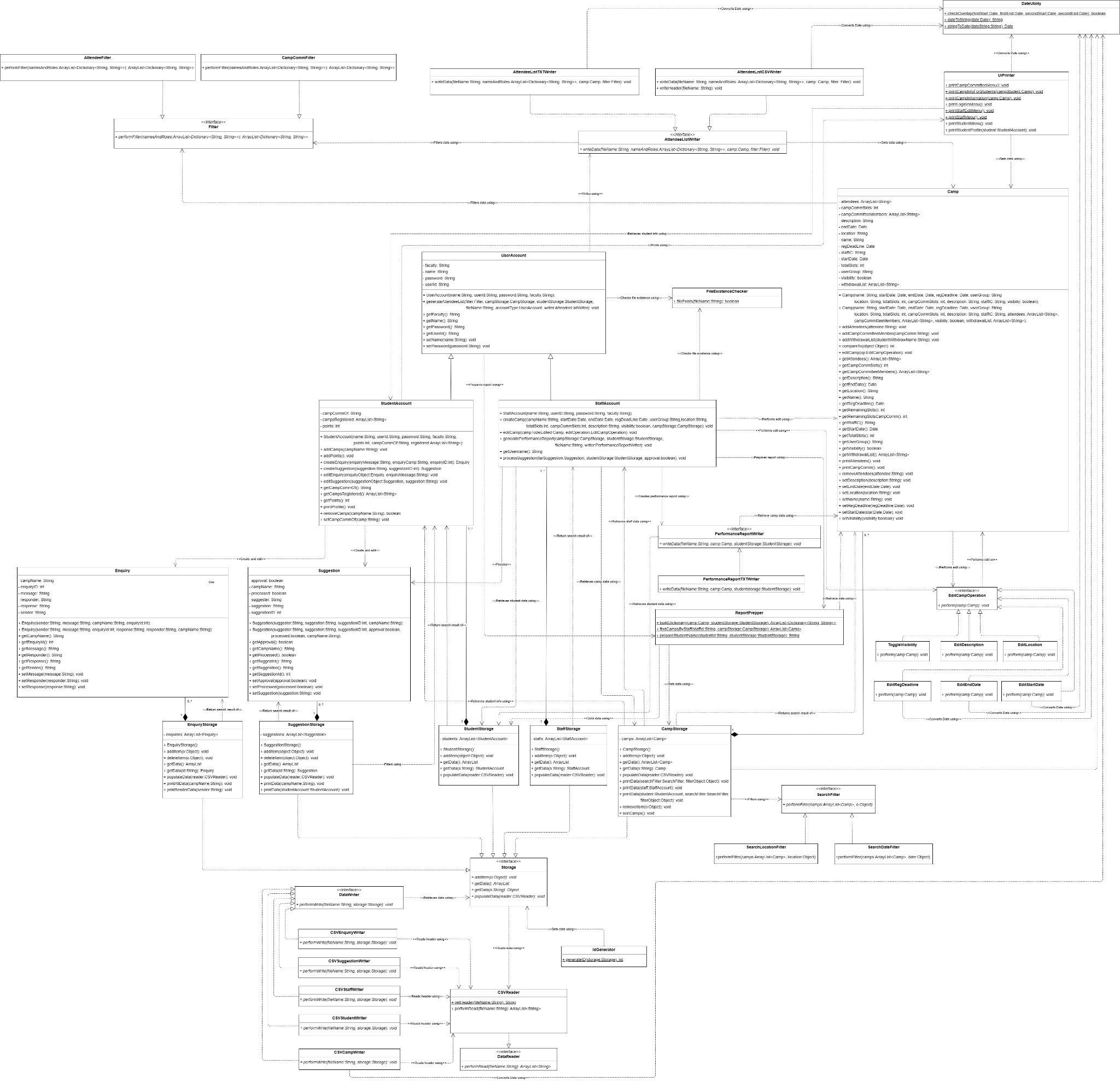
From the very start of the project, we had already encountered a difficulty, “How to accurately represent our code in a class diagram as well as implement the SOLID design”. We then realised that instead of trying to fully complete our class diagram before coding, we should just create the necessary interfaces and classes we will need to implement and only draw the association lines after we have coded. It allows us to have a better idea of how one class/interface is linked to the other.

The next main issue was error handling. While testing our code, we realised that we were only expecting “correct” input from the user. That is to say, if we want the user to input an integer, we expect them to only input integers and not characters. However, when the user does input characters, our code will break. Therefore, in line with good design and implementation practices, we implemented error handling for every section of our code that accepts user input. This way, if we want an integer and they input a character, it will prompt the user to input again. Likewise, in our code, we have also implemented exception handling which was taught to us in chapter 4 of this course. For example, in our writer and reader classes, we have “try” and “catch” statements to catch any exception being thrown at us. Taking on the role as a CAMs user allowed us to find unexpected inputs and make the experience more user friendly.

The vast knowledge of OODP we have learnt from this course was extensively and rigorously applied to our implementation. As seen from the previous section where we provided snapshots of our code, we implemented SOLID design that was taught to us as well as various OODP concepts such as polymorphism.

Lastly, some additional features that we propose will be to implement password masking when the user is typing in their password. As well as using a hash table to store the passwords of our CAMs users.

# 3. UML Class Diagram

(For full image, check image folder)

# 4. Test Case Demonstration

Actual output images will be under ‘Appendix’.

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Description | Expected output | Actual Output |
| 1. | Cannot Login: Invalid user or Invalid password. | An invalid user or Invalid password should prompt the user to enter their user and password again. | Wrong password: cannot login.  Wrong user: cannot login.  Correct user and password: login successful. |
| 2. | After logging in, they should display the menu for the different users. | Staff account should print the staff menu.  Student account should print the student menu.  Camp committee menu should only be printed if the student is a camp committee member. | Staff account can see the staff menu.  Student account can see the student menu.  Students cannot see and access the camp committee menu if they are not a camp committee.  Camp Committee can see and access the camp Committee menu. |
| 3. | User is able to change the password. | Users are able to log in using their new password after changing. | Users can log in using their new password. |
| 4. | Staff: Staff is able to create, edit and delete camps | When staff create a camp, it is added to the database with the relevant details.  Staff can edit the camp details and will update the database.  Staff can delete camps; camps will be removed from the database. | Staff can create a camp.  Staff can edit a camp.  Staff can delete camps. |
| 5. | Staff: Staff can view all camps | Staff can view all camps created.  Staff can filter by date.  Staff can filter by location.  Staff can apply no filter. | Staff can view all camps created.  Staff can filter by date.  Staff can filter by location.  Staff can apply no filter. |
| 6. | Staff: Staff can see the list of camps that he/her created | Staff can view only camps created by them. | Staff are only able to view all camps created by them. |
| 7. | Staff: Staff can view and reply to enquiries from student to the camp he/her created | Staff can view enquiries from the camps he/she created.  Staff is able to reply to the enquiry.  Staff cannot view enquiries to camps not created by them. | Staff can view enquiries from the camps he/she created.  Staff is able to reply to the enquiry.  Staff cannot view enquiries to camp not created by him. |
| 8. | Staff: Staff can view and approve suggestions | Staff can view suggestions.  Staff can accept/reject suggestions. | Message “There is no suggestion made for this camp” when there is no suggestion.  Staff cannot view suggestions to camps that they did not create.  Staff can view suggestions to camp that they created.  Staff can approve suggestions.  Staff can reject suggestions.  Staff cannot approve/reject suggestions that are already processed. |
| 9. | Staff: Staff can generate a report of students attending each camp.  Staff can also filter what they want to generate | An attendee list report will be generated.  Staff can filter based on output type and report type. | An attendee list report is generated based on the different format and filter chosen. |
| 10. | Staff: Staff can generate a performance report of the camp committee members | A report on the performance of the camp committee member will be generated. | Staff can generate a performance report for the camp committee. |
| 11. | Student: Students can only view a list of camps that are open to their group and that the visibility is toggled on. | Students can only see camps that are open to their faculty/NTU with visibility on.  Students can filter based on location and date. | Students can only see camps that are open to their faculty/NTU with visibility on.  Students can filter based on location and date. |
| 12. | Student: Students can view the remaining slots of each camp that is open to them. | Students can see the remaining slots of the camps that are open to them only. | Students can see the remaining slots of the camps that are open to them only. |
| 13. | Student: Students can register for camp either as camp attendee or camp committee | Students can register as camp attendees.  Students can register as a camp committee. | Students can register for a camp that is open to them.  Students cannot register for camps that are not open to them.  Students can register for the camp committee. |
| 14. | Student: Students can submit enquiries regarding a camp. | Students can create enquiries. | Students can send enquiries to camps that are open to them.  Students cannot send enquiries to camps that are not open to them or do not exit. |
| 15. | Student: Students can view, edit, and delete their enquiries before it is processed. | Students can view, edit, and delete their enquiries.  Once processed, students will not be able to edit or delete their enquiries. | Students can view, edit, and delete their enquiries before it is processed.  Students cannot edit or delete their enquiries after it is processed. |
| 16. | Student: Status of the student’s camp committee will be reflected in their profile | Students can see their camp committee status. | Students can see their camp committee status. |
| 17. | Student: Student can only be camp committee for one camp but can attend multiple camps | Students can only register as a camp committee for one camp.  Students can register for multiple camps. | Students can only register for one camp committee.  Students can register for multiple camps. |
| 18. | Student: Cannot register for camps if there are conflicts | Student cannot register for camp if one of the following holds.   1. There is a clash in dates. 2. Camp if full 3. Pass the camp’s registration deadline | Students cannot register for a camp that has a clash in dates with other camps.  Students cannot register for a camp if it is full.  Students cannot register for a camp past its registration deadline. |
| 19. | Student: Student can see the camps that they have already register and his/her role | Students can see the camps they registered for and their roles. | Students can see all camps that they registered for as well as their role in those camps. |
| 20. | Student: Student is allowed to withdraw from camps but not allowed to register for the same camp again | Students can withdraw from camp.  Students cannot join back once they withdraw. | Students can withdraw from camp.  Students cannot join back camps that they withdrew from. |
| 21. | Camp Committee: Not allowed to directly edit camp details | No option for the camp committee to edit camp details. | There is no option in the camp committee menu to edit camp details. |
| 22. | Camp Committee: Can submit suggestions for change to camp details to staff | Camp committee can create suggestions. | Students are able to create a suggestion.  Students are not able to submit blank suggestions. |
| 23. | Camp Committee: Can view, edit, and delete the details of their suggestion before it is processed | Camp committee can view, edit, and delete suggestions before it is processed.  Camp committee cannot view, edit, and delete suggestions after it is processed. | Camp committee can view, edit, and delete their suggestions.  Camp committee cannot edit or delete their suggestion after it is processed. |
| 24. | Camp Committee: Can view enquiry and reply to enquiries | Camp committee can view enquiries.  Camp committee can reply to enquiries. | Message “There are no enquiries for this camp.” when there are no enquiries.  Camp committee can see enquiries made.  Camp committee can reply to enquiries made.  Camp committee cannot reply to enquiries that have already been responded to. |
| 25. | Camp Committee: Can generate a report of the list of students attending. Can filter how they want to generate the list | A report of the list of students attending will be generated.  Students can filter based on output type and report type. | A report will be generated based on the chosen filter. |
| 26. | Camp Committee: Gets one point for each enquiry replied and each suggestion given. Additional point for each accepted suggestion | Students get one point for each enquiry replied.  Students get one point for each suggestion given.  Students get one point for accepted suggestions. | Points indicated in the profile menu will be updated accordingly. |
| 27. | Camp Committee: Cannot quit from camp | Camp committee can withdraw from camps that they are attendee of but not from the camp that they are camp committee of. | Camp committee can withdraw from camps that they are attendee of but not from the camp that they are camp committee of. |

# 5. Appendix

