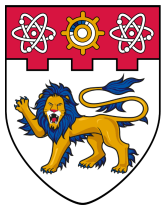


NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

# SC2002 Object-Oriented Design & Programming

# Camp Application Management System

# Report of Project Structure Design & Functionality

**AY 23/24 Sem 1 | Lab SCSX, Team 1**



| NAME | MATRICULATION NUMBER |
|---|---|
| Wang Yangming | U2222553D |
| Liau Zheng Wei | U2222032K |
| Fan Tianyu | U2222105H |
| Clayton Fernalo | U2220422E |
| Christopher Angelo | U2220148L |



**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 | Lab Group | Signature/Date |
|---|---|---|---|
| Wang YangMing | SC2002 | SCSX | 26/11/23 |
| Liau Zheng Wei | SC2002 | SCSX | 26/11/23 |
| Fan Tianyu | SC2002 | SCSX | 26/11/23 |
| Clayton Fernalo | SC2002 | SCSX | 26/11/23 |
| Christopher Angelo | SC2002 | SCSX | 26/11/23 |

# 1. DESIGN CONSIDERATIONS

CAMS (Camp Application Management System) is a Java console application designed with a focus on reusability, extensibility, and maintainability. It accommodates different user types and their requirements, allowing for easy upgrades and future development.

## 1.1 Design Approach

High cohesion and loose coupling were one of the top priorities of this project. Classes were separated into three separate stereotypes: controllers, boundaries, and entities. Controllers include ***CampEnquiryController***, ***UserController***, Boundaries include ***Widget***, ***Window***, and Entities include ***Camp***, ***Enquiry*** The user first interacts with the boundary classes which call the control classes to carry out functions like updating the entity or retrieving information from it to be displayed or used for other functions. These three class stereotypes work hand in hand together while reducing dependency among them, highlighting the extensibility and modularity of the system.

## 1.2 Design Highlights

- Generic Repository Class: An overall ***RepositoryList*** abstract class is used to store data and easily perform various filtering and sorting operations.
- Configurable Central Repository: All CSV data are imported and saved together through the ***RepositoryCollection*** class to ensure persistent data while being easily modifiable by changing the path values in `const/Config.java`.
- Factory Design Pattern: A factory pattern is used to determine if a user is a staff or a student so that proper methods can occur
- Extensible and Reusable User Interface (UI): UI components are created such that they can be reused throughout the entire application.
- Entity-Control-Boundary (ECB) Architecture: Classes are separated into three possible stereotypes (entity, control, and boundary), which provide modularity and separation of concerns.

## 1.3. Design Principles Used

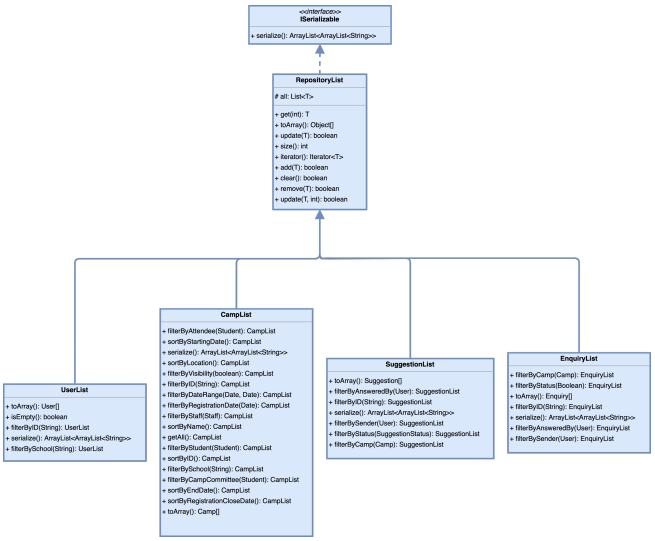
### 1.3.1 The Single Responsibility Principle (SRP)

The single Responsibility Principle states that a module should be responsible for one and only one actor. By adhering to SRP, we reduce the frequency of change and minimize the impacts that each change brings. Just to name a view, classes like ***CampStatusMonitorController*** and ***CampSuggestionController*** exemplify this principle by focusing on specific tasks. For instance, ***CampStatusMonitorController*** solely handles monitoring camp statuses and generating performance reports based on student participation. This design

approach makes classes independent of each other, simplifying code comprehension. As a result, the number of bugs is reduced, ultimately enhancing development speed.

### 1.3.2 The Open Closed Principle (OCP)

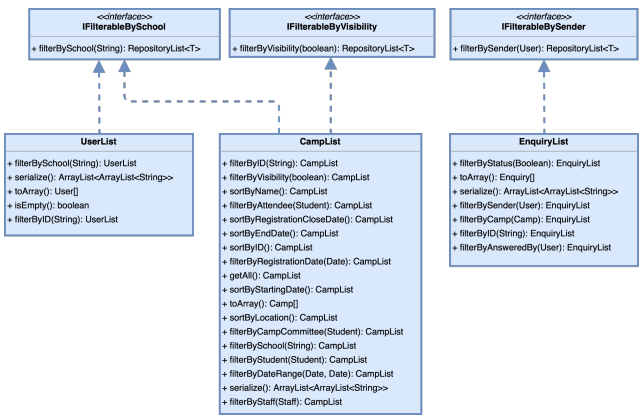
The Closed Principle states that software entities like classes, modules, and functions should be open for extension but closed for modification. OCP uses interfaces instead of superclasses such that different and new implementations extend the software's functionality without changing the code that uses them. With interfaces as an additional layer of abstraction, loose coupling is established. In this project, we apply OCP by creating a general abstract class ***Repository*** that can be extended to create different types of Repositories, such as ***CampRepository*** and ***UserRepository***. Each subclass overrides the *load()* and *save()* methods to allow easy extension of the Repository system. The *load()* and *save()* methods in these Repositories are then called by another class ***RepositoryCollection***, which its *load()* and *save()* methods are called in ***Main.*** This also enables easy readability and debugging of the code. We use an interface ***ITaggedItem*** to get different types of filters like ***IFilterableByID*** and ***IFilterableByCamp***. Each filter extends ***Repository*** and uses its constructor for different filter purposes, allowing new filters to be added without changing the existing code.




### 1.3.3 The Liskov Substitution Principle (LSP)

The Liskov Substitution Principle states that every subclass or derived class should be substitutable for its base or parent class. In scientific terms, when q(x) is a property provable about objects of x of type T, then q(y) should be provable for objects y of type S, where S is a subtype of T. In our code, we used LSP through the use of an abstract class ***User***. Concrete classes ***Student*** and ***Staff*** both implement methods of the abstract ***User*** class while having methods distinct for their class. As such, LSP is satisfied with the subclasses providing more than the superclass and expecting less than what the base class can provide.

### 1.3.4 The Interface Segregation Principle (ISP)




The interface Segregation Principle states that no code should be forced to depend on methods it does not use. ISP splits large interfaces into smaller and more specific ones so that clients only have to know about methods that are of interest to them. To promote maintainability, flexibility, and modularity, we separated a large filter interface into separate interfaces like ***IFilterableByCamp***, ***IFilterableByID,*** and ***IFilterableBySchool***. This ensures

that different entity classes like ***CampList*** and ***UserList*** can utilize these filter interfaces. This makes sure that only filters that are needed by the entity classes are implemented, reducing the butterfly effect from code modification. For the user interface, we applied ISP by splitting the interface into small, separate ones like ***ISelectable***, ***ITextInput,*** and ***IDrawable***. These interfaces allow different widgets to be implemented, with new widgets only needing to add additional interfaces or use existing ones. For example, with ***Widget*** implementing ***IDrawable***, ***WidgetButton*** can extend ***Widget*** and implement ***ITextInput*** and ***ISelectable***. In comparison, ***WidgetLabel*** only extends ***Widget***, since it does not provide selecting and text input functions.

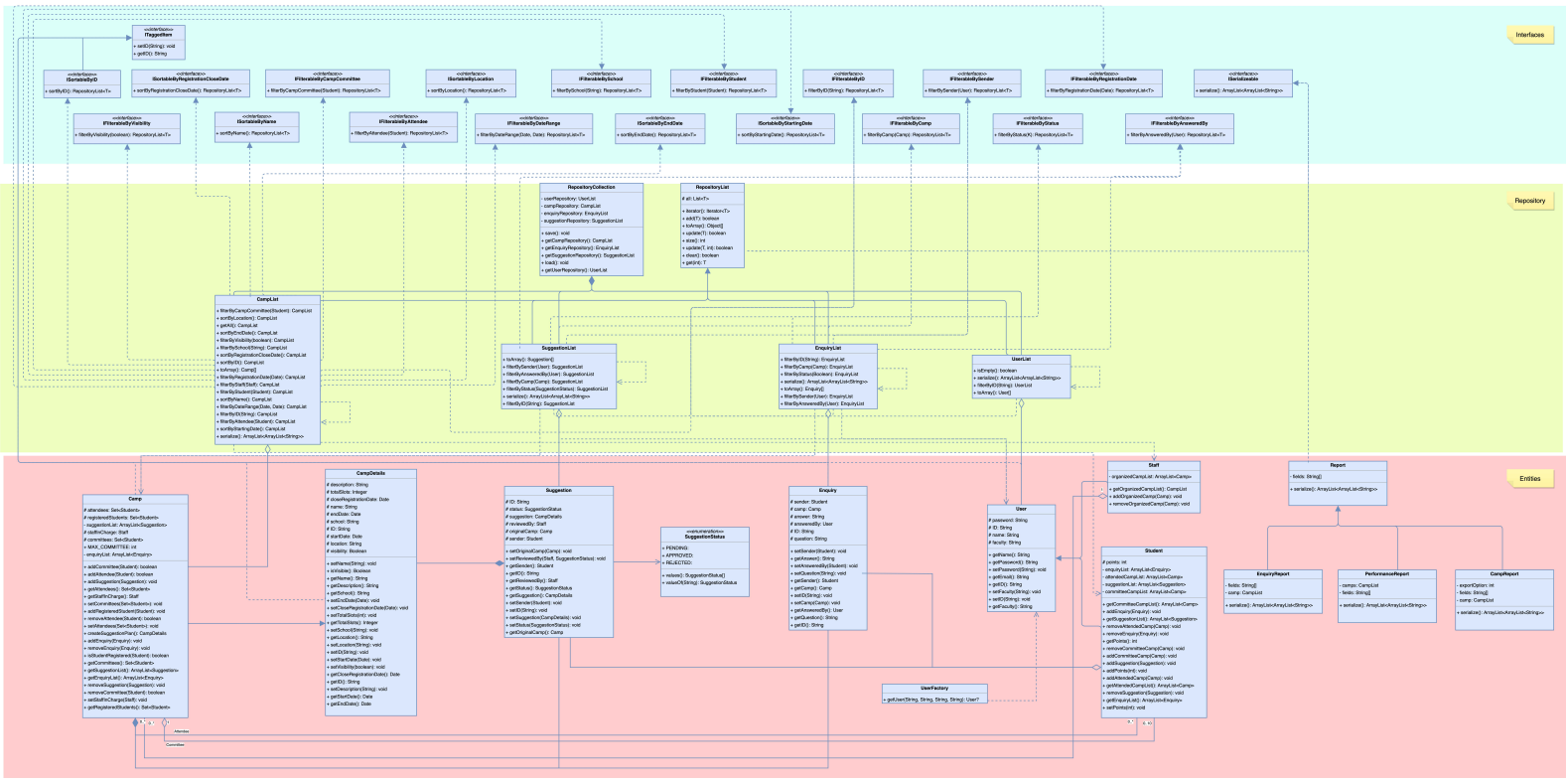
### 1.3.5 The Dependency Injection Principle (DIP)

The Dependency Injection Principle is used to make a class that is independent of the low-level implementation using abstractions, achieving a loosely coupled program and reusable classes. In other words, high-level modules must be independent of low-level implementation, using abstraction as an interface. In our design, we have a util.CSV class that performs read/write operations from/to CSV files that are abstracted for other high-level classes. To write to a CSV file, classes that implement ***ISerializable*** (e.g., ***UserList***, ***CampReport***) have a method that generates a two-dimensional ***ArrayList,*** which will be translated into a CSV file without knowing the low-level implementation. And vice versa, we would just provide a path name for the CSV file we want to read, and the method in util.CSV will return a two-dimensional array.

# 2. UML Class Diagram

Please refer to the **"./diagrams"** folder for all class diagrams.

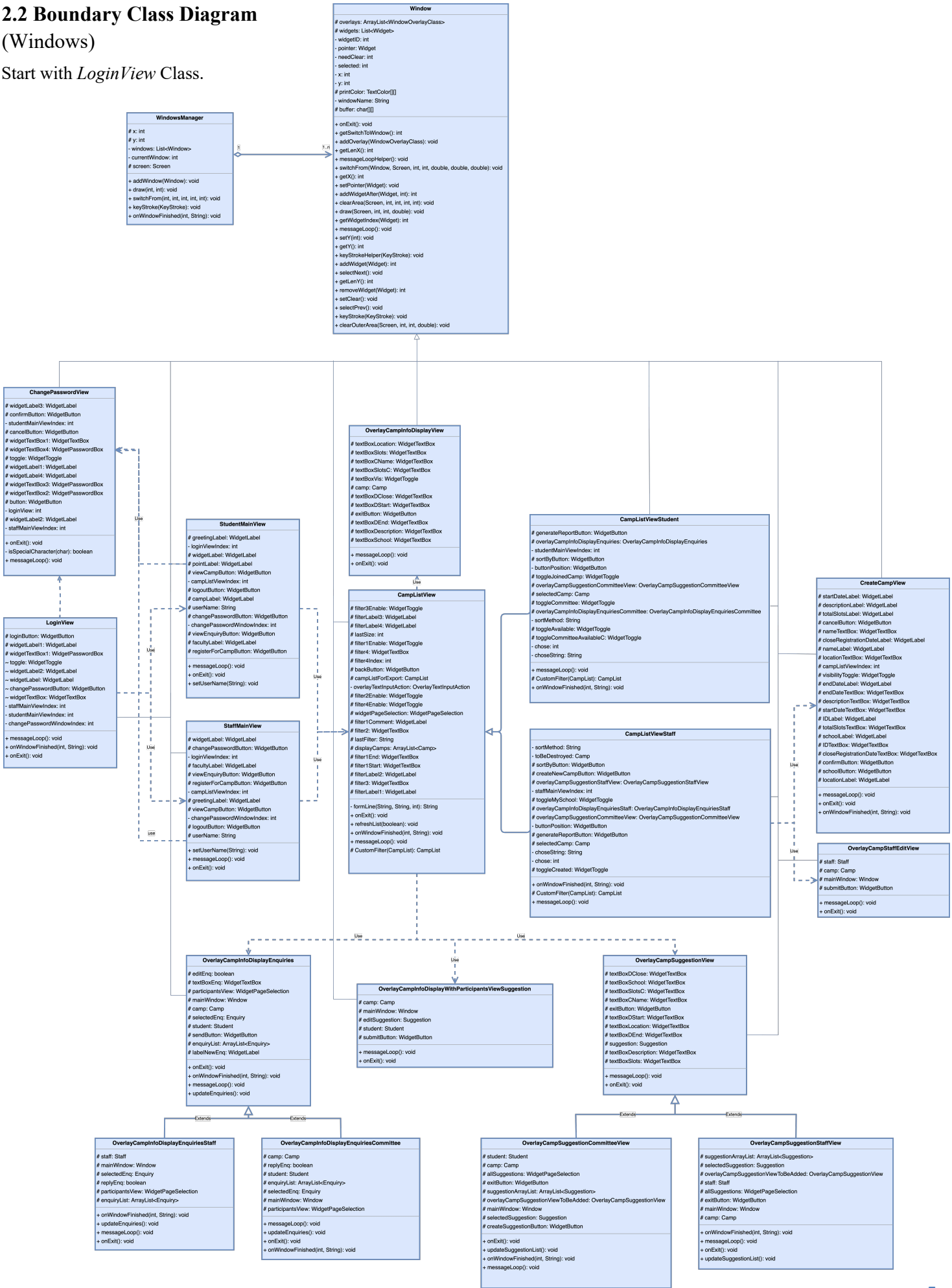
## 2.1 Entity Class Diagram



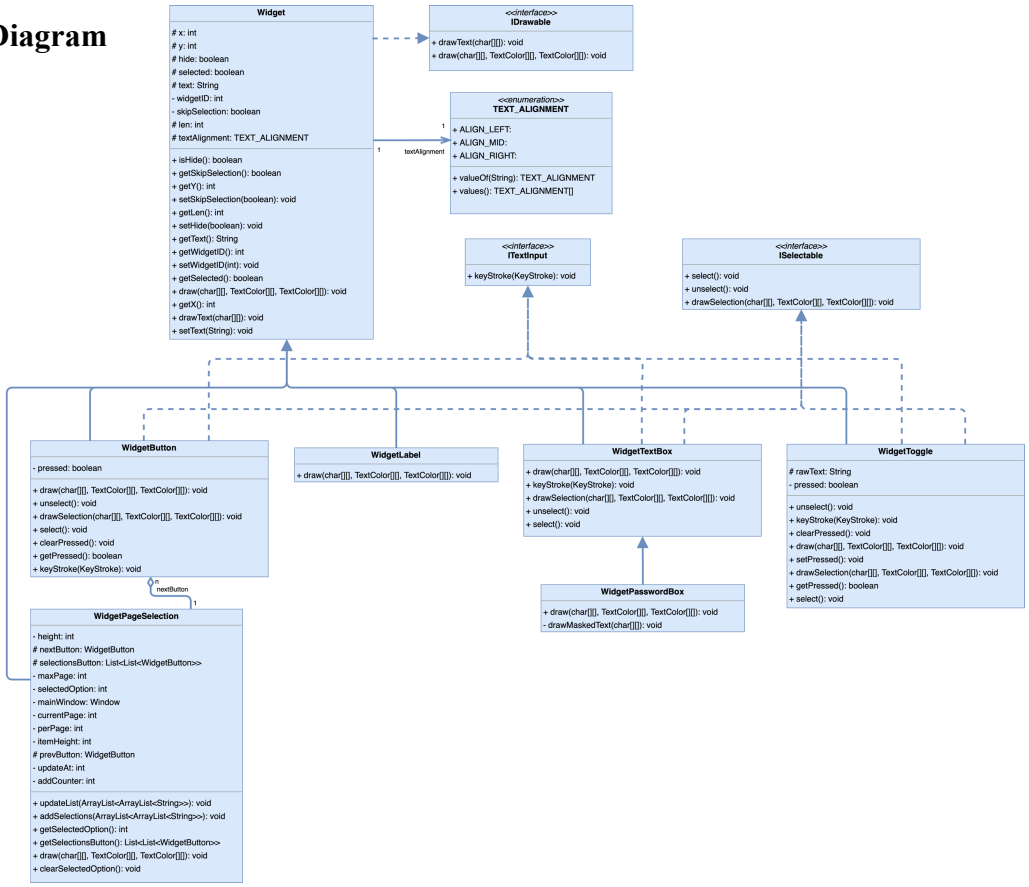
## 2.2 Boundary Class Diagram

(Windows)

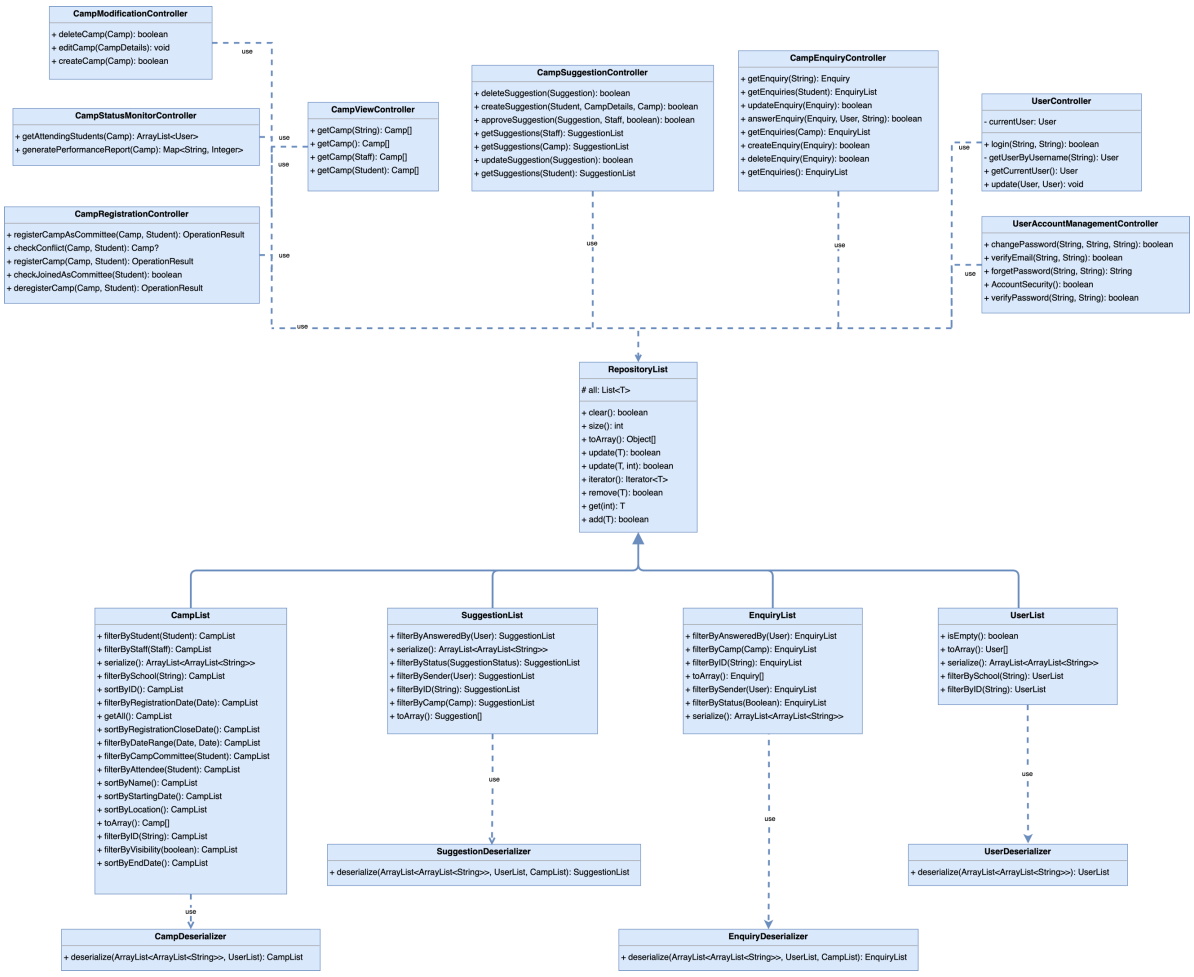
Start with *LoginView* Class.



## 2.3 Boundary Class Diagram
(Widgets)



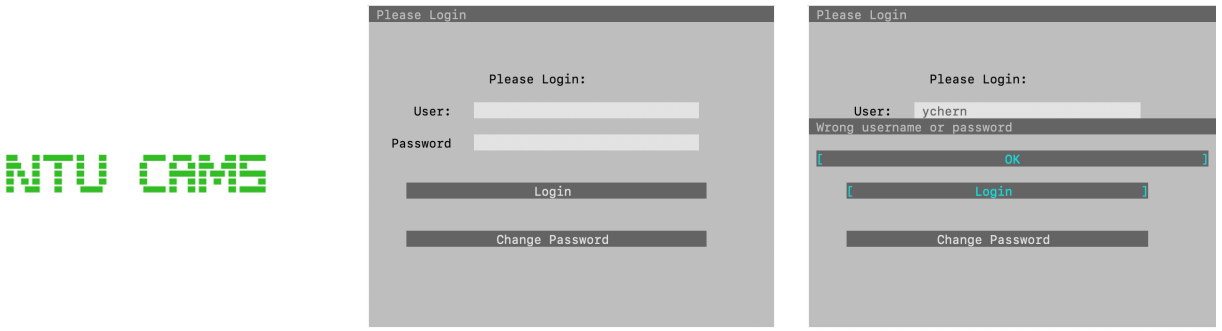

## 2.4 Controller Class Diagram



# 3. Testing

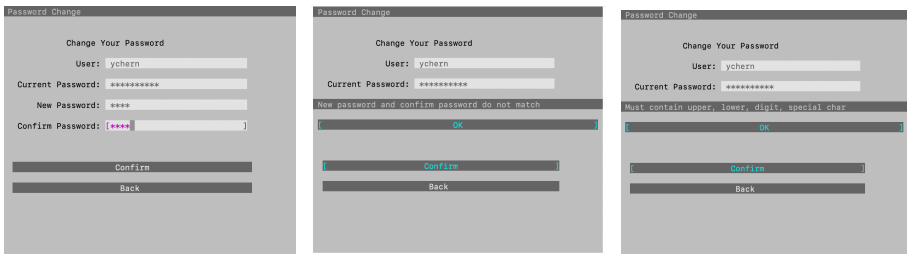
## 3.1 Login and Change Password

### 3.1.1 Login Page



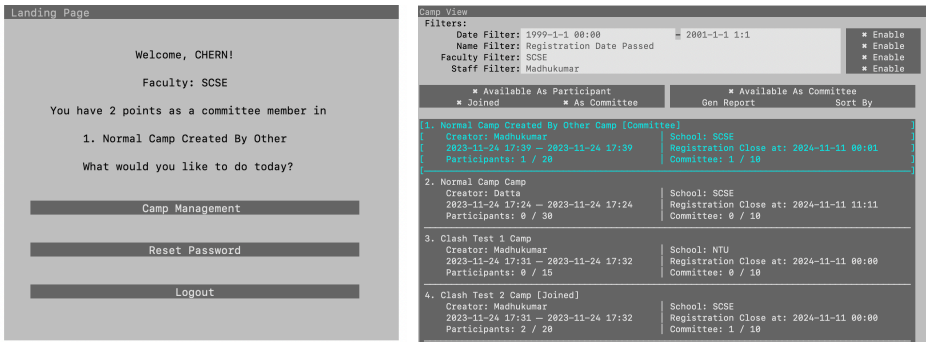

### 3.1.2 Change Password (prompt on first login)

If the user's new Password is not a 'strong password', the user is prompted to re-enter the Password.



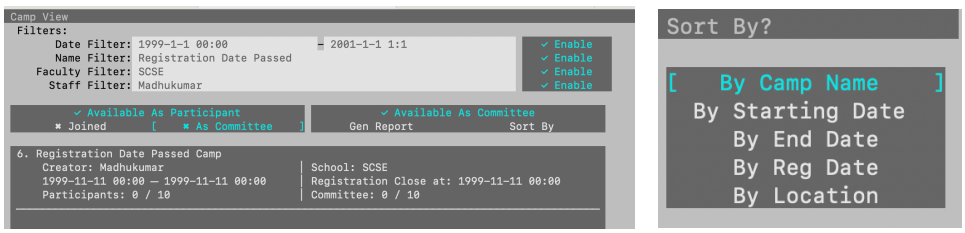

## 3.2 Student Main Page

The system will greet the user and direct them to a Camp View Page where the user can view all available camps and check their role (if any) in the camp:

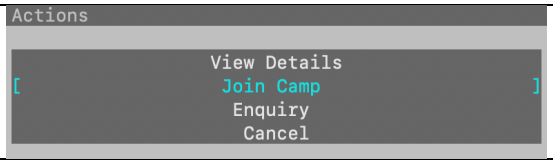


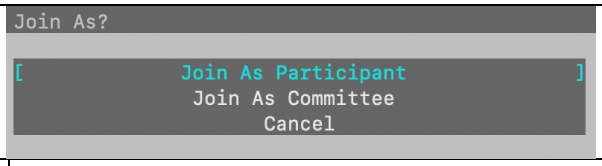

### 3.2.1 Filter Camps View

On top of the main page, there's a section where user can filter their Viewed camps using filters like (date, name, faculty, etc). Users can also filter their registered/registered as committee camps:

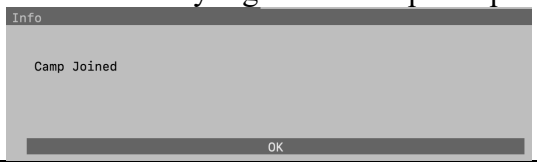


### 3.2.2 Register and Deregister as Participants or Committee

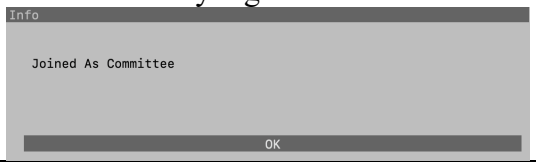


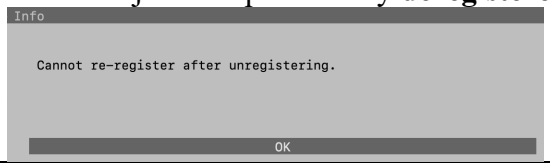

User successfully registered as a participant:



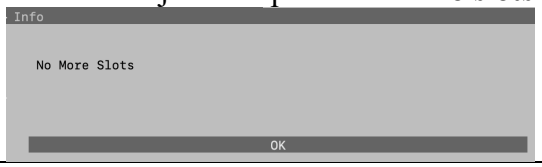

User Successfully registered as a committee:



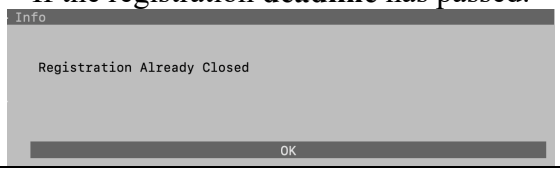

User cannot join camps that they **deregistered**:



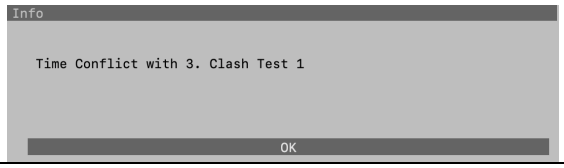

User cannot join camps that have **no slots**:



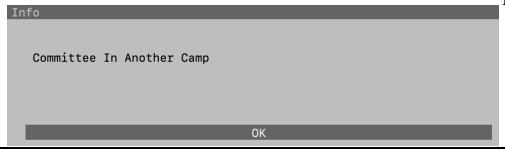

If the registration **deadline** has passed:




If a **Time conflict** exists:

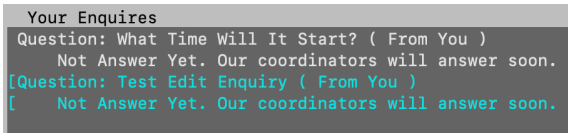



If user is the **committee in another camp**:

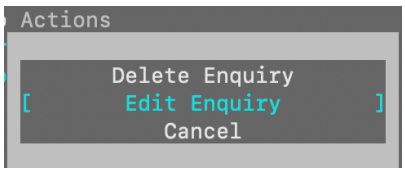



### 3.2.3 Create/View/Edit/Delete Enquiries

Student Create Enquiry:

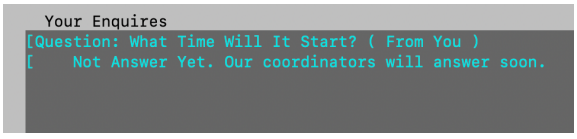




Edit Enquiry:



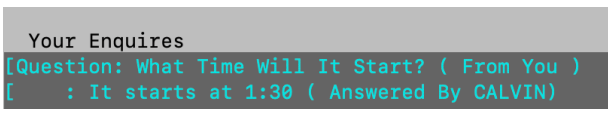


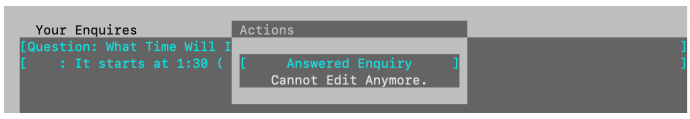
Delete Enquiry:





Students will **not** be allowed to edit or delete any replied enquiry anymore.

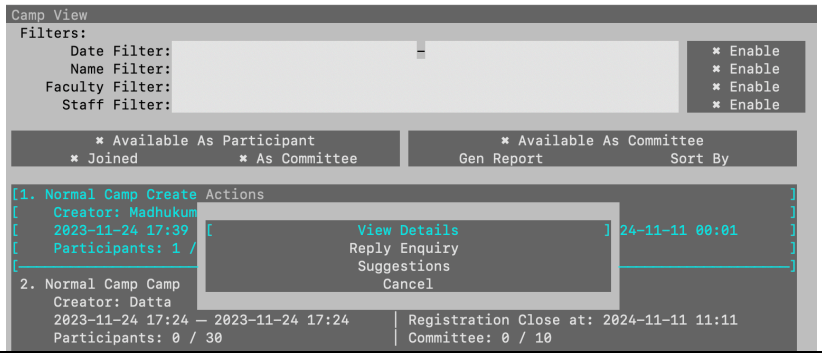


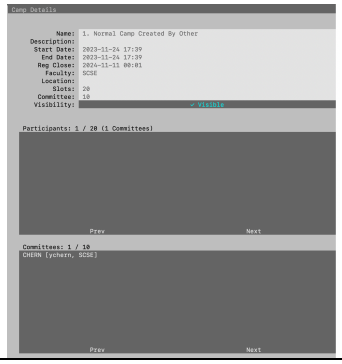



## 3.3 Committee Main Page

Committee members not only can do what students can do, but can also view camp details (including participant list, and committee list), view/reply to student enquiry, create/delete/edit suggestions for camps.

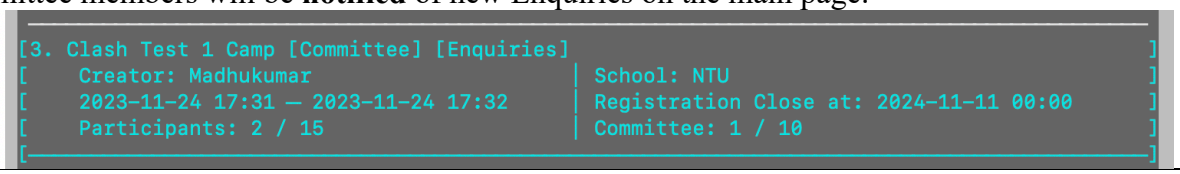
Committee member's actions to a camp



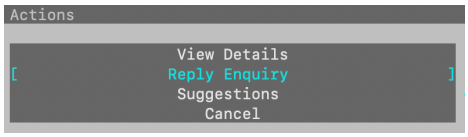



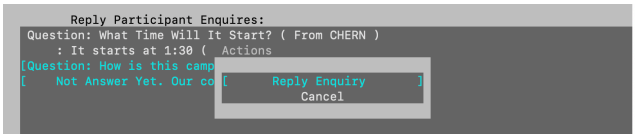
### 3.3.1 Answer Enquires

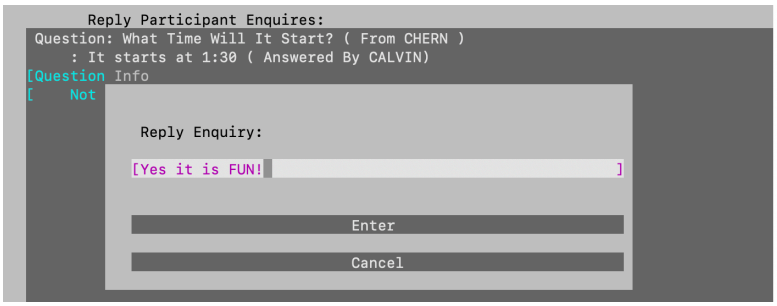
Committee members will be **notified** of new Enquiries on the main page:

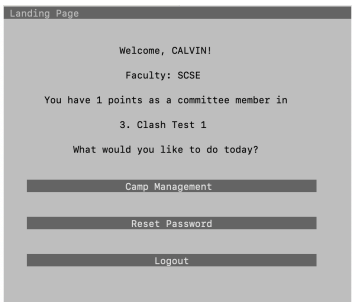



View and Reply to Enquiries from students from the camp he/she oversees, and earn **one** bonus point:



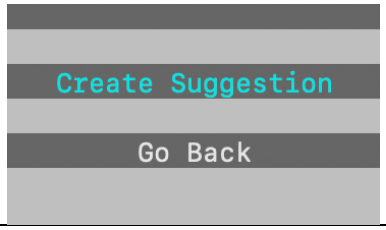



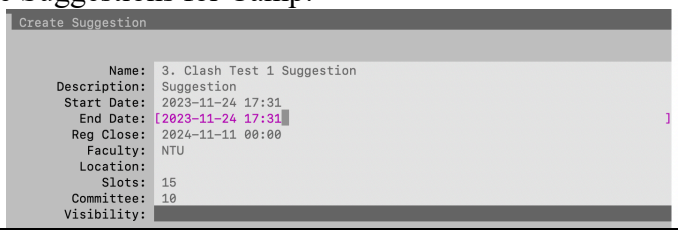



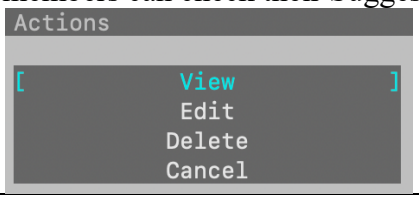

### 3.3.2 Create/Delete/Edit Suggestion

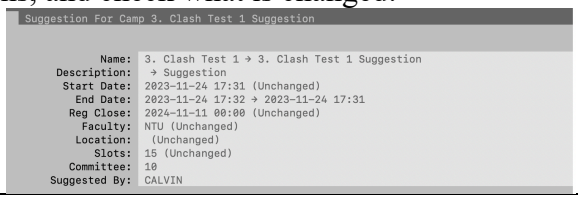
Committee members will be able to create Suggestions for Camp:



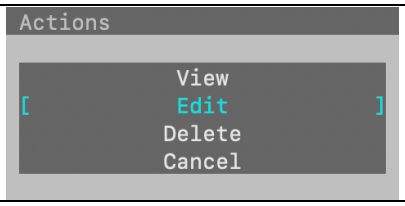



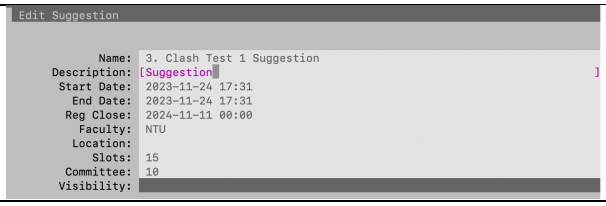

Committee members can check their Suggestions, and check what is changed:




Committee members will be able to view their Suggestions, and edit/delete them before it is processed:

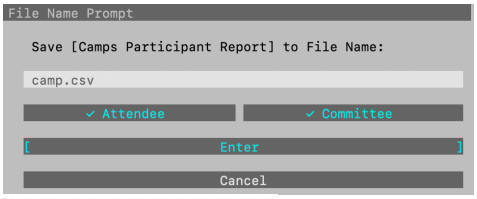



### 3.3.3 Generate Report (Camp Report and Enquiries Report)

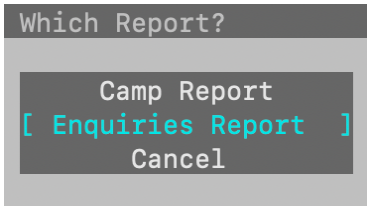
Before exporting the report, student may first by name/date/faculty first.

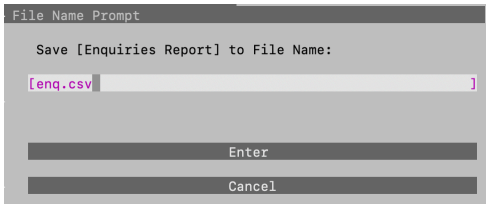
1. Student can generate the **Camp Participant Report** with the list of students attending each camp. User can also apply filters. Student can choose to generate for attendee or committee members only.


```
fantianyu@Erics-MacBook-Pro SC2002 % cat camp.csv
Camp ID,Camp,Student ID,Name,Faculty,Role
1700818282264,3. Clash Test 1,ct113,CALVIN,SCSE,Committee
1700818282264,3. Clash Test 1,ychern,CHERN,SCSE,Attendee
fantianyu@Erics-MacBook-Pro SC2002 %
```

2. Student can also export **Students' Enquiry Report**:

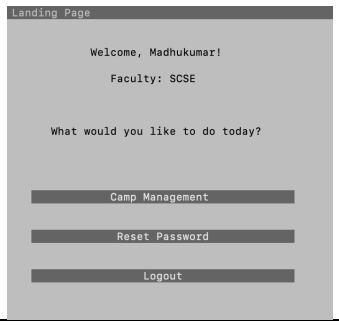


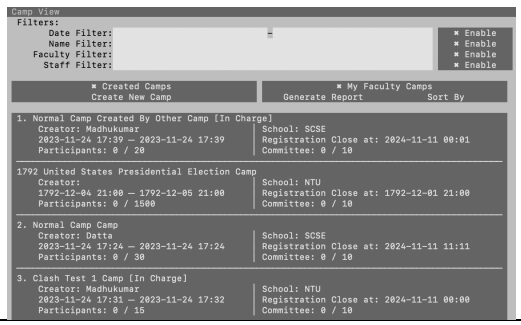




```
fantianyu@Erics-MacBook-Pro SC2002 % cat enq.csv
Camp ID,Camp Name,Sender ID,Sender Name,Question,Answered by Name,Answered by ID,Answer
1700818282264,3. Clash Test 1,ychern,CHERN,What Time Will It Start?,ct113,CALVIN,It starts at 1:30
1700818282264,3. Clash Test 1,ychern,CHERN,How is this camp?,ct113,CALVIN,Yes it is FUN!
```

## 3.4 Staff Main Page

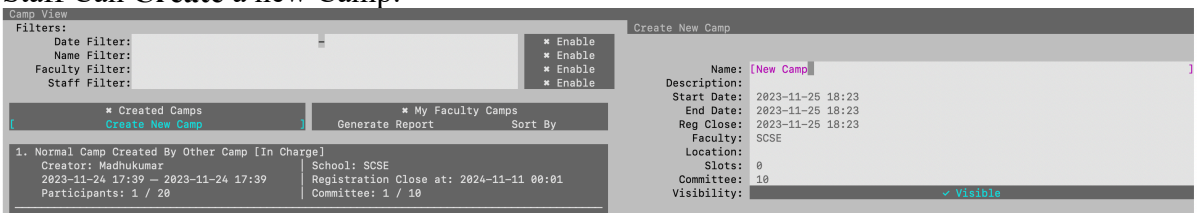
After logging in, the system will greet the staff and direct them to a page where the user can change Password and see all the available camps. The main page shows which camps are created by this staff, and if there are still any pending enquiries or suggestion. It also shows the staff's role in the camp and whether there are any **pending** suggestions/enquiries.




### 3.4.1 View/Create/Edit/Delete Camp

Staff Can **Create** a new Camp:




Staff can **View** list, user filter to filter camps of specific time/faculty/staff, and check the details of each camp, which contains of students that have registered for the camp as attendees or committee can be viewed. Staff can also sort camps by name/date/location. Staff Can **Edit/Delete** camps he/she has created and not able to edit/delete camps from other staff.

Delete camps will **fail** unless there are no more participants in the camp.

### 3.4.2 Accept or Reject Suggestions from Camp Committee

Staff can **View/Accept/Reject** Suggestions from camp committee.



Student will also be able to see the result of their Suggestions, and they will earn 2 points for every suggestion being approved (1 for submission and 1 for approval).

### 3.4.3 Generate Report (Camp Report, Performance Report, Enquiries Report)

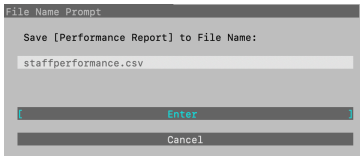
Staff can filter which camp to generate **Camp Participants Report** and filter by date, name, faculty, and whether to generate report for both attendee and committee or either of them.


```
fantianyu@Erics-MacBook-Pro SC2002 % cat staffcamp.csv
Camp ID,Camp,Student ID,Name,Faculty,Role
1700818772760,1. Normal Camp Created By Other,ychern,CHERN,SCSE,Committee
1700818282264,3. Clash Test 1 Suggestion,ct113,CALVIN,SCSE,Committee
1700818282264,3. Clash Test 1 Suggestion,ychern,CHERN,SCSE,Attendee
```

Staff can also generate **Enquiries Report** like a committee.

However, Staff can generate an additional report, **Committee Performance Report**, which will give the names of the committee members alongside their respective points.

Staff can generate camp **committee performance report**:




```
fantianyu@Erics-MacBook-Pro SC2002 % cat staffperformance.csv
Camp ID,Camp,Student ID,Student,Points
1700818772760,1. Normal Camp Created By Other,ychern,CHERN,0
1700818282264,3. Clash Test 1 Suggestion,ct113,CALVIN,3
fantianyu@Erics-MacBook-Pro SC2002 %
```

# 4. Reflection

Reflecting on our journey in this Object-Oriented Programming project, the transition from our initial approach to one firmly rooted in SOLID design principles was a profound learning experience. Initially, we grappled with a codebase where classes like ***AccountController*** and ***CampController*** were overloaded with responsibilities. This design led to a fragile system where minor changes often cascaded into significant issues, hindering our ability to meet project requirements effectively.

The adoption of SOLID principles marked a pivotal shift in our approach. We began by re-evaluating our class structure, creating more focused classes such as ***CampEnquiryController***. This restructuring was guided by the Single Responsibility Principle, ensuring that each class had a clearly defined purpose. For instance, ***CampEnquiryController*** was now solely responsible for handling camp inquiries, a stark contrast to our previous approach, where a single class would juggle multiple tasks.

Our adherence to the Open/Closed Principle was evident in how we redefined our data handling with classes like ***RepositoryList***. This generic abstract class allowed us to extend its functionalities without altering existing code, a significant departure from our earlier design, where each entity had its own list and save-load functions. This not only made our codebase more flexible but also significantly reduced the complexity of adding new features or making modifications.

This shift to a design-centric development approach was transformative. It brought about a newfound efficiency in our workflow, and more importantly, it changed our perspective on software development. We began to see design not just as a part of coding but as an essential foundation for building robust, scalable, and maintainable software. The lessons learned from this project extend beyond the technical realm; they have imbued us with a deeper appreciation for the art of software design and its critical role in the successful evolution of a software system. Looking forward, we are confident that this design-first methodology will significantly aid in seamlessly integrating advanced features like password hashing and supporting concurrent user access, ensuring our system remains adaptable and forward-compatible.