**Declaration of Original Work for SC2002/CE2002/CZ2002 Assignment**

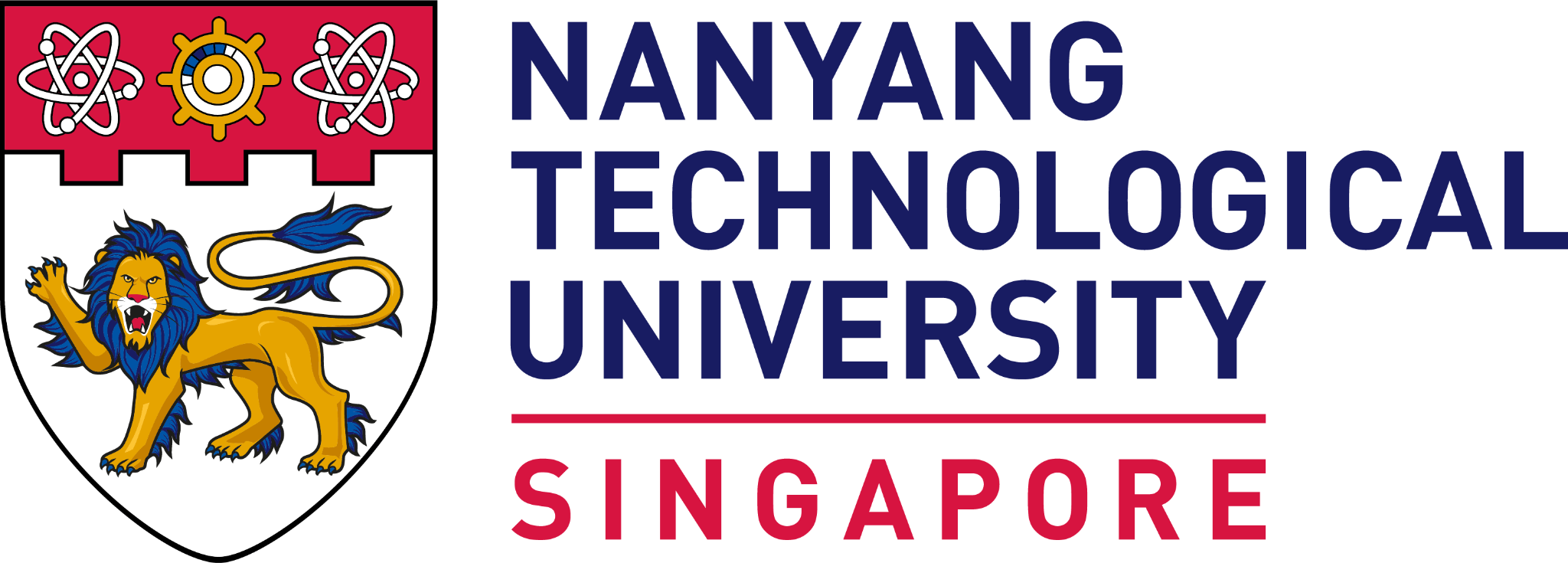
We hereby declare that the attached group assignment has been researched, undertaken, completed, and submitted as a collective effort by the group members listed below.

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 |
| --- | --- | --- | --- |
| Ang Jun Hoa | SC2002 | SCS7 | 16 November 2024 |
| Joanna Wu Haoyue | SC2002 | SCS7 | 16 November 2024 |
| Megan Phua Wei Lin | SC2002 | SCS7 | 16 November 2024 |
| Tan Hsuen Yin Andria | SC2002 | SCS7 | 16 November 2024 |
| Tai Yi Sheng Winston | SC2002 | SCS7 | 16 November 2024 |

**SC2002 – Object Oriented Design & Programming**



**Group Assignment Report**

Ang Jun Hoa (U2323971E)

Joanna Wu Haoyue (U2322092E)

Megan Phua Wei Lin (U2322539E)

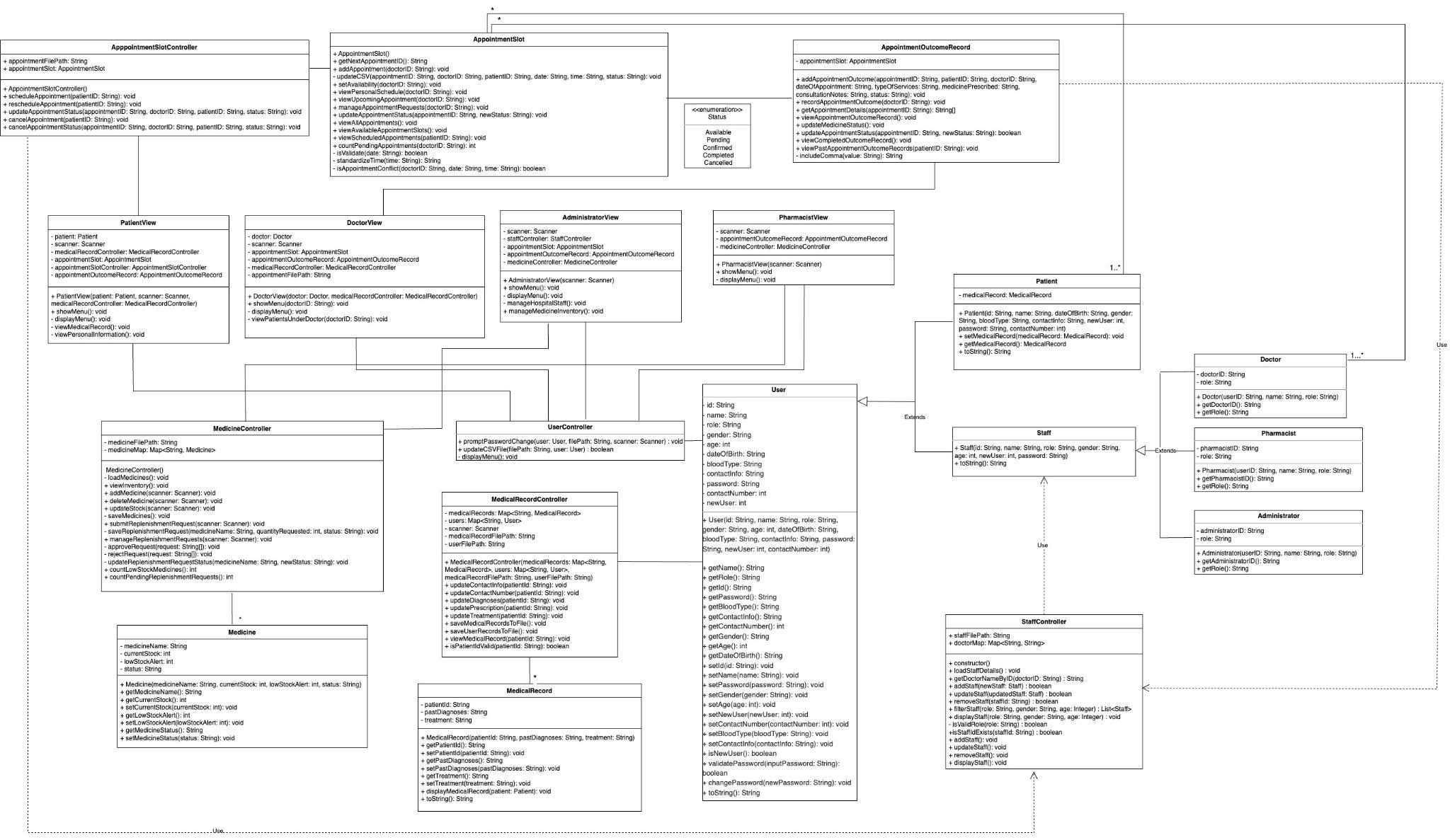
Tan Hsuen Yin Andria (U2320699F)

Tai Yi Sheng Winston (U2321548J)

Date: 17 November 2024

Live demonstration was presented on 18 November 2024.

# UML Class Diagram



For clearer view: [Link to UML](https://app.diagrams.net/#G152GC9uau995_Aj3j2c_gKSEmM4Nvw7pe#%7B%22pageId%22%3A%22O2wpBT_MUn3Jg5LllMUN%22%7D)

# 

# Design Considerations

The design of the Hospital Management system (HMS) is centred around object-oriented design and programming (OODP) principles, which enhances the system’s extendability, maintainability and scalability. The following key OODP concepts were utilised:

## Design Principles

### Single Responsibility Principle (SRP)

Each class is designed with a single responsibility. For example, the User class is solely responsible for creating a user, while common actions performed by users are implemented in the UserController class like promptPasswordChange function that handles logic for user to change default password upon initial login. Another example is PatientView class which focuses on presenting options specific to a patient’s needs. Adhering to SRP ensures high cohesion as each class focuses on its specialised task, and low coupling such that changes in one class does not ramify throughout the system.

### Open-Close Principle (OCP)

A class can be extended independently without modifying the others. For example, introducing a new staff type to the system does not affect the User or Staff class. This separation creates a flexible, modular design that remains stable and closed to modification, even as new features are added. Additionally, this makes the code easier to maintain for future developers of the HMS.

With OCP, inheritance is also demonstrated. The Patient and Staff class extends the User class, allowing it to inherit common attributes and behaviours such as ID, name and password. This promotes code reusability and reduces redundancy.

### Encapsulation

The appropriate visibility modifiers were used to protect sensitive data like medical personal details. In MedicalRecord class, attributes such as pastDiagnoses and treatment are encapsulated by being set to private visibility. This ensures that they cannot be accessed and modified directly, but only through get and set methods, thus promoting data integrity and reducing the risk of unintended security side effects.

### Polymorphism

The system uses polymorphism through method overriding. For instance, different user roles (Doctor, Patient, Administrator, Pharmacist) can implement their own version of methods like displayMenu, and showMenu in Main.java. This induces dynamic binding at runtime and allows for role-specific functionality.

## Design Patterns

### Model-View-Controller (MVC) Architecture

Our HMS is separated into Model, View and Controller components that make up a MVC structure. View classes implement the user interface by managing how information is displayed to the user. Control classes also implement the user interface by managing user interactions and capturing user inputs to make modifications to the system, such as adding, deleting or updating data. Finally, model classes handle the data. This separation of Model from View and Controller classes allows multiple Views of the same Model by separating presentation and interaction from the data. Hence our HMS is able to support multiple different user interfaces, and also allows for easy addition of new user interfaces for any new user type added.

### Singleton Pattern

The MedicalRecordController and AppointmentSlotController can be implemented as singleton if their instances need to be shared across the application. This ensures that only one instance exists, reducing the memory usage and ensuring consistent state.

### Factory Pattern

The DataInitializer class to create instances from CSV data helps to simplify object creation and enhance code readability. This was done as there are multiple CSV data files to initialise.

## Coding Practices

### Code documentation

To enhance readability and collaboration among our team, we have consistently added comments throughout our code. This documentation approach has facilitated smoother handoffs and reduced the time spent deciphering code during review sessions.

## 

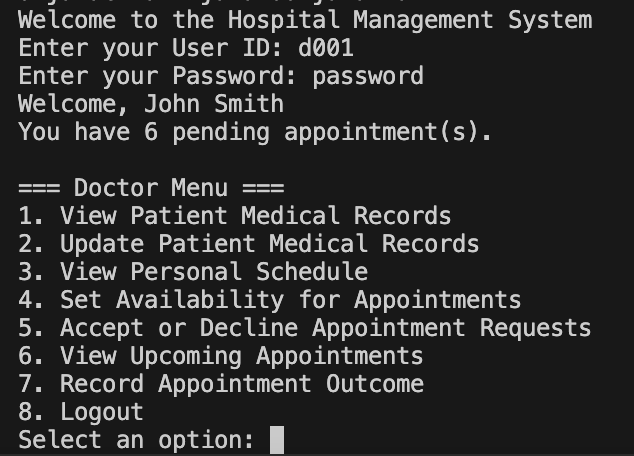
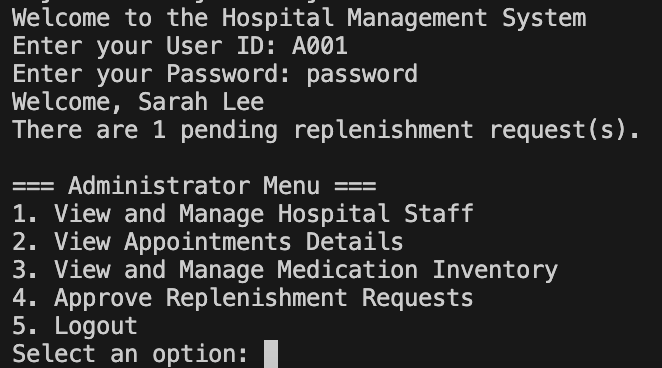
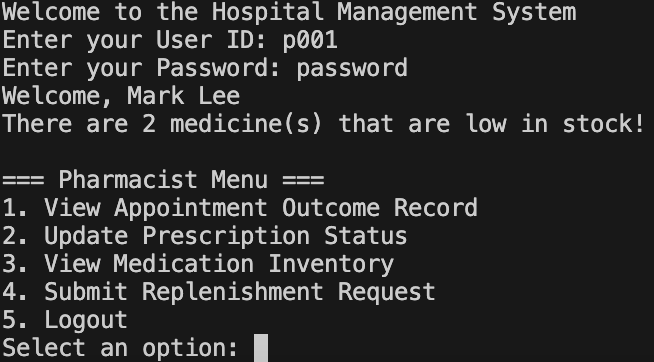
### Exception Handling

Exception handling is implemented primarily through the use of try-catch blocks during file-related operations and data processing such as reading and writing the CSV files. These blocks ensure that errors are caught and the appropriate error messages are displayed to the user, ensuring robustness and program continuity. Moreover, when a user input an incorrect password when logging in, the system will generate an error message and prompt the user to re-enter password.

## 

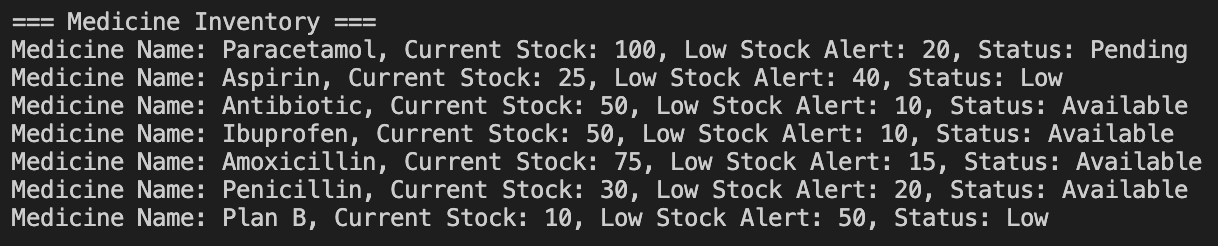
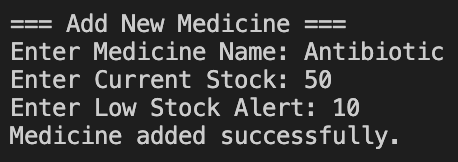
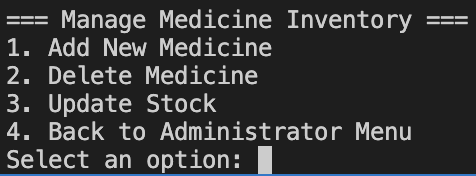
## Additional Features

We chose to enhance the user interface by prioritising the display of urgent messages before presenting the respective menus (Figure 1). When doctors log in, they are immediately notified of pending appointments. When pharmacists log in, they are alerted if there is medicine low in stock. When administrators log in, they receive a notification for any pending replenishment requests from the pharmacists. By implementing this feature, we address common healthcare challenges such as delayed responses to critical needs, overlooked tasks, and miscommunication among staff. For example, low-stock alerts for pharmacists prevent medication shortages, while pending appointment reminders for doctors help avoid scheduling conflicts or missed consultations. These proactive notifications reflect real-life priorities of a hospital setting, ensuring that critical matters are addressed promptly, that contribute to a more efficient and patient-centred healthcare environment.

*Figure 1. An example of a doctor, pharmacist and administrator interface respectively*

To reflect how hospitals are always importing and removing drugs due to ever-changing FDA regulations, the administrator has an additional function of adding or removing medicine from the medicine inventory (Figure 2). This feature addresses common healthcare challenges such as the risk of using outdated or non-compliant medications, as well as the inefficiencies caused by holding excess inventory or facing critical shortages.



*Figure 2. An example of how an administrator adds new medicine to medicine inventory*

## 

## Trade-offs in Design

### Complexity vs Simplicity

While the use of multiple classes improves the modularity, it can introduce complexity. For example, managing dependencies between controllers and models requires careful design to avoid circular dependencies (*relying on one another*). However, this complexity is justified by the benefits of modularity, which allows for independent testing and development of components.

### Performance vs Flexibility

The design prioritises flexibility over raw performance. For instance, using a file-based system for data storage such as *CSV files* simplifies the implementation. This may not perform as good for a database solution. This trade-off is acceptable in the context of an educational project such as this, where ease of implementation and understanding is prioritised over performance.

# Testing

## Patient Actions

| Test Cases | Results |
| --- | --- |
| Test Case 1: View Medical Record |  |
| Test Case 2: Update Personal Information |  |
| Test Case 3: View Available Appointment Slots |  |
| Test Case 4: Schedule Appointment Slots |  |
| Test Case 5: Reschedule an Appointment |  |
| Test Case 6: Cancel an Appointment |  |
| Test Case 7: View Scheduled Appointments |  |
| Test Case 8: View Past Appointment Outcome Records |  |

## Doctor Actions

| Test Cases | Results |
| --- | --- |
| Test Case 9: View Patient Medical Records |  |
| Test Case 10: Update Patient Medical Records |  |
| Test Case 11: View Personal Schedule |  |
| Test Case 12: Set Availability for Appointments |  |
| Test Case 13: Accept or Decline Appointment Requests |  |
| Test Case 14: View Upcoming Appointments |  |
| Test Case 15: Record Appointment Outcome |  |

## Pharmacist Actions

| Test Cases | Results |
| --- | --- |
| Test Case 16: View Appointment Outcome Record |  |
| Test Case 17: Update Prescription Status |  |
| Test Case 18: View Medication Inventory |  |
| Test Case 19: Submit Replenishment Request |  |

## Administrator Actions

| Test Cases | Results |
| --- | --- |
| Test Case 20: View and Manage Hospital Staff |  |
| Test Case 21: View Appointments Details |  |
| Test Case 22: View and Manage Medication Inventory |  |
| Test Case 23: Approve Replenishment Requests |  |

## Login System and Password Management

| Test Cases | Results |
| --- | --- |
| Test Case 25: First-Time Login and Password Change |  |
| Test Case 26: Login with Incorrect Credentials |  |

# 

# Reflections

### Difficulties encountered & Knowledge Learnt

This HMS involves numerous entities, which made the relationships between classes seem overwhelming. To tackle this, we recognised the need for a clear and well structured visual representation that could outline the relationship between the classes, and hence guide the coding process. As such, we initially approached this project using a waterfall model, where we began by drafting a comprehensive UML diagram before proceeding with the coding phase. This plan-driven approach helped facilitate delegation of tasks amongst our group members. However, as we began coding, we realised that some aspects of our diagram were incorrect or insufficient. As a result, we pivoted to an agile project management approach in order to iteratively refine our UML class diagram and more effectively manage dependencies between classes and functions while coding.

Additionally, our original UML class diagram was designed with the assumption that each user class would encompass all the relevant functions. However, this led to a tightly coupled code whereby small changes in implementation required extensive modifications, leading to frequent recompilation and maintenance challenges. To address this, we refactored our design considerations to better adhere to OODP principles, as mentioned above.

Accessing XLSX files requires the use of third-party libraries such as Apache POI or JExcelAPI, which adds complexity to the project. Therefore we decided to convert to CSV and use hashmap to manage the data. Its efficiency is better with time complexity of O(1) for lookups, insertion, and deletion. It is easier to handle and prevent any duplicate keys for user IDs.

### 

This project allows us to better appreciate core OODP concepts like SRP and polymorphism as we experience how it can improve code maintainability and efficiency. We also recognised the critical role of planning through UML diagrams, using them as a foundational guide for our coding process. Most importantly, this experience taught us the value of teamwork and effective communication, enabling us to collaboratively develop a functional system which can be applied for development of future real-world systems.

### Further Improvement & Suggestions

Further improvements include implementing a method/library for catching scanner errors to use globally as well as interface classes to achieve even looser coupling and abstraction.

**Github Repository link**: <https://github.com/ajunhoa/OOP-Project>