SC2002 OBJECT ORIENTED DESIGN & PROGRAMMING

HOSPITAL MANAGEMENT SYSTEM (HMS)

AY24/25 SCSF Group 6

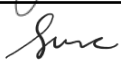
Name	Matriculation Number
Glynis Looi Xin Lin	U2321198L
Lee Yun Jia	U2322087F
Elayalwar Surya Prakash	U2320720K
Rio Tiffany Estralita	U2322354H
Tan Ying Xuan	U2322283C

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.

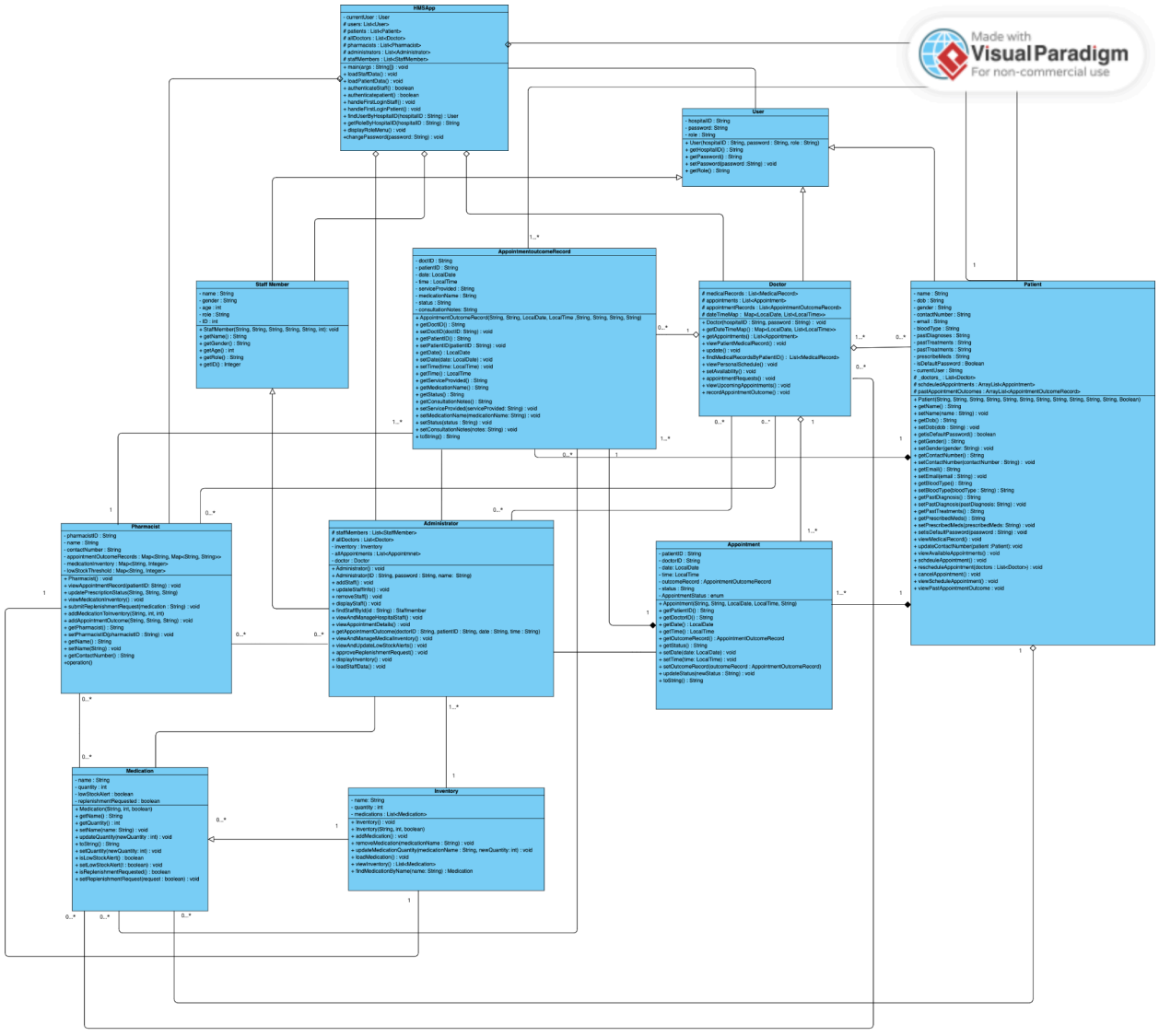
We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

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
Glynis Looi Xin Lin	SC2002	SCSF	
Lee Yun Jia	SC2002	SCSF	
Elayalwar Surya Prakash	SC2002	SCSF	
Rio Tiffany Estralita	SC2002	SCSF	
Tan Ying Xuan	SC2002	SCSF	

Link to Github Repository:
https://github.com/glynislxl/Hospital-Management-System-HMS.git

1 UML CLASS DIAGRAM



2 ADDITIONAL FEATURES / FUNCTIONALITIES

In order to improve the functionality and security of our HMS system, we have added additional features to it. These include a Drug Allergy Check which ensures a Doctor does not prescribe a drug that a patient is allergic to, and a Password Validation Check which validates changes in passwords through re-entry and enforces stricter password requirements. These features reduce the risk of unintentional input error as well as strengthens data protection.

2.1 Drug Allergy Check

This feature enables patients to input their drug allergies during profile set up. If they do not have any drug allergy, they can input "None". If a patient inputs a drug allergy that does not match any medication that is provided in the Hospital, the system will inform the patient that "Medication not provided in Hospital", and the system will record "None" as the patient's drug allergy in the CSV file, for the data to remain relevant to the hospital's available medications.

```
Enter any drug allergy, enter None if nil:
Ibuprofen
Patient details validated successfully!
New patient account created and authenticated successfully.
```

Figure 2.1.1 Example of inputting "Ibuprofen" as Patient's Drug Allergy, recorded successfully in the system

After a completed appointment between the doctor and the patient, when the doctor records the appointment outcome and attempts to prescribe medication, the system would check the prescribed medication against the patient's previously inputted drug allergy. If a match is found, a warning message "Warning: This medication ('Medication Name') is on the patient's allergy list. It cannot be prescribed." is displayed. This alerts the doctor and thus prevents the prescription of allergic medicines to the patient.

```
Enter medication name: ibuprofen
Warning: This medication (ibuprofen) is on the patient's allergy list. It cannot be prescribed.
Enter another medication name: Ibuprofen
Warning: This medication (Ibuprofen) is on the patient's allergy list. It cannot be prescribed.
Enter another medication name: panadol
Enter consultation notes: drink more water
Appointment outcome recorded successfully.
```

Figure 2.1.2 Example of prescribing "Ibuprofen" (Patient's Drug Allergy) to patient, system detecting allergies in a case-insensitive manner (eg. "ibuprofen" vs "Ibuprofen). Warning message is displayed.

This feature greatly contributes to patient safety as it prevents the prescription of allergic drugs, which may cause adverse reactions. When this risk is eliminated, safe prescription of medications is ensured. Ultimately, it enhances the quality of care and treatment provided to patients, which is essential in healthcare services.

2.2 Password Validation Check

This feature prompts all users in the system, including patients and staff, to re-enter their new password for validation. Such a re-entry check ensures that there are no typos, and that the entered password corresponds with the user's intent. Additionally, the system enforces strict password requirements: the password must be at least 8 characters long, include at least one

uppercase and one lowercase letter, one numerical digit, and at least one special character (eg. _, *, @, $).
Besides, the new password should not match the default password “password”, or be the same as the previous password. These measures ensure effective protection of user account credentials and boost security within our HMS system.

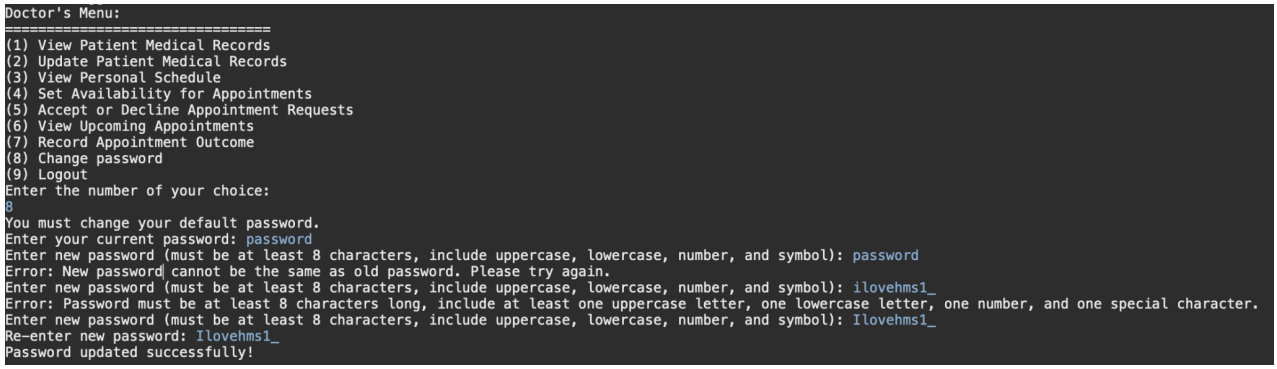


Figure 2.2.1 Example of user Doctor attempting to change password to the default password “password”. System prompts the user that the new password cannot be the same as the old password. In addition, system prompts users that “Password must be at least 8 characters long, include at least one uppercase letter, one lowercase letter, one number, and one special character” if password requirements are not met

3 DESIGN CONSIDERATIONS

The Hospital Management System (HMS) is a Command Line Interface (CLI) Java Application designed to optimise hospital operations with essential functions like patient management, appointment scheduling and inventory tracking. HMS is structured to support distinct users – patients, doctors, pharmacists, and administrators, while focusing on modularity and maintainability. It ensures that the system can be easily extended and adapted for future usage and enhancements. It adheres to principles of SOLID Design and follows Object-Oriented Programming concepts, which will be discussed below.

3.1 SOLID Design Principles

3.1.1 Single Responsibility Principle (SRP)

Our HMS is divided into several classes, where each class performs a single highly specific task. This can be seen in our implementation of the various User classes - Administrator, Doctor, Pharmacist and Patient. The different user classes have the singular responsibility of performing functions pertinent to that class, thereby adhering to SRP.

3.1.2 Open/Closed Principle (OCP)

Owing to the abstraction of our base classes, our Application is open for extension and closed for modification; new classes such as ‘Nurse’ can be added without altering existing code. This enhances maintainability by reducing the need for direct changes in the current code. It also promotes reusability and scaling in the future by encouraging the development of flexible components that can be seamlessly extended without breaking existing code.

3.1.3 Liskov Substitution Principle (LSP)

Our User Subclasses - Administrator, Doctor, Patient and Pharmacist have been designed such that none of them override the User Superclass in a way that changes their expected behaviour. Our Subclasses do not throw unexpected exceptions or add stricter constraints on input parameters, thereby adhering to LSP.

3.1.4 Interface Segregation Principle (ISP)

We have used abstract and compact classes such as User and StaffMember that adhere to ISP. Our subclasses thus only implement methods they actively use, without leaving space for redundant methods. This reduces unnecessary code, alleviates difficulty in maintenance, minimises risk of bugs and promotes flexibility

3.2 OOP Concepts

3.2.1 Abstraction

Abstraction is a key principle used in our hospital management system to simplify complex operations by exposing only the essential functionalities of each component. The design employs base classes, such as User and StaffMember, which encapsulate shared attributes and behaviours across different entities like administrators, doctors, and pharmacists. These classes provide a foundation for extending functionality without delving into the details of each specific role. For instance, methods like `` `viewAppointmentRecord()` `` in the Pharmacist class abstract the process of retrieving and displaying appointment details, presenting only the necessary information for the task at hand.

3.2.2 Polymorphism

Polymorphism is implemented in our hospital management system to enable entities to exhibit different behaviours based on their roles and contexts. This is achieved through method overriding and dynamic method dispatch. For instance, derived classes like StaffMember and its specific implementations redefine or extend behaviours inherited from the base class User. This allows methods such as `` `displayInfo()` `` to adapt to the needs of each role, providing tailored output without altering the base class logic.

3.2.3 Inheritance

Inheritance was used to establish a hierarchy between superclasses and their subclasses - in our Application, a specific example would be our parent User class and its child classes Administrator, Doctor, Pharmacist and Patient. This is justified because all our child classes share common attributes with the Parent class such as hospitalID, which they can inherit. These examples are illustrated in our UML Diagram. This inheritance relationship allows us to reuse code and implement the aforementioned polymorphic relationship.

3.2.4 Encapsulation

Encapsulation was implemented to safeguard the internal state of objects and restrict unauthorised access to data by other classes. As a result, some of our class attributes were declared private and could only be accessed or modified through public getter and setter methods.

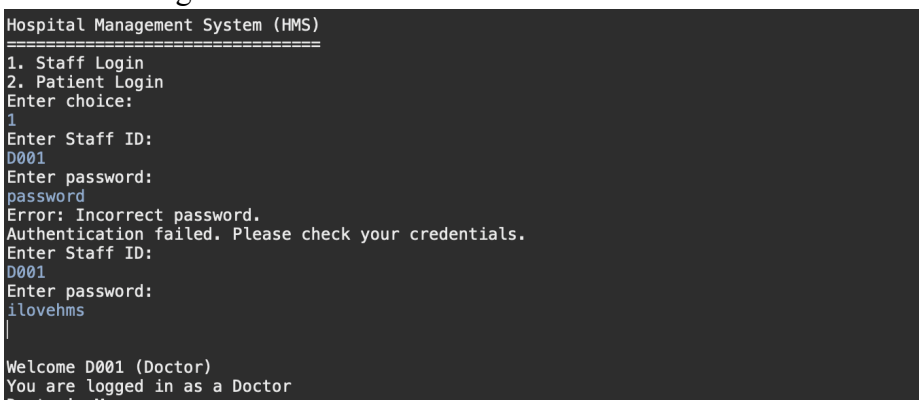
3.3 Trade Offs

Some methods, such as `updatePrescriptionStatus()` in Pharmacist class, perform multiple tasks (reading files, updating data, managing inventory, etc.), which violates the Single Responsibility Principle. This can make the code harder to test and maintain. The tradeoff here was to minimise the number of methods or classes to simplify the system, at the expense of modularity and clarity.
File paths and configurations, such as the locations of CSV files, are hardcoded in the system. This approach limits portability and makes it difficult to deploy the system in different environments without modifying the source code. However, we found that it simplifies the design and eliminates the need for additional configuration management or environment-specific setup.

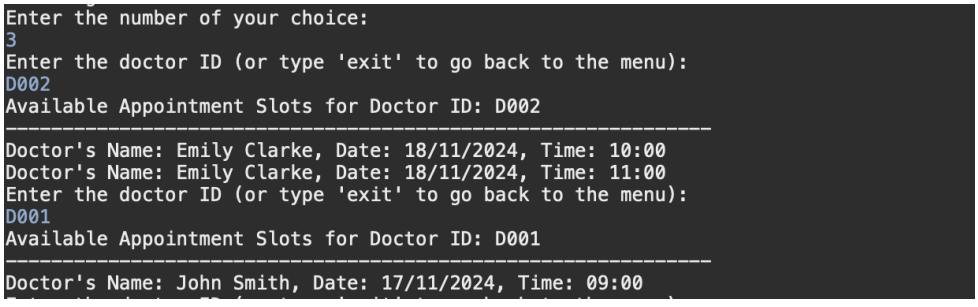
4 TEST CASES & RESULTS

Here is a suggestive but not exhaustive list of test cases to validate the functionalities of our HMS.

4.1 Authentication

No.	Test Description
1	**Login with Incorrect Credentials** a) User attempts to log in with an incorrect password. b) Verify that the system displays an error message indicating invalid credentials, and login is denied.  *(Password has been changed to "ilovehms" beforehand)*

4.2 Patient Functionality

No.	Test Description
1	**View Available Appointment Slots** a) Patient views available appointment slots with doctors. b) Verify that the system displays a list of available appointment slots, showing doctors' names, dates, and times 

2	**Schedule an Appointment**
	a) Patient schedules a new appointment with a doctor.
	b) Verify that the appointment is scheduled successfully with status "Scheduled". The selected time slot becomes unavailable to other patients. The system should prevent the patient from booking a time slot that is unavailable/already booked.
	
	As Patient PA1 schedules an appointment with Doctor D001, appointment scheduled successfully:
	`Schedule an Appointment`
	`----------`
	`Enter the doctor ID (or type 'exit' to go back to the menu): D001`
	`Available slots for doctor D001 (John Smith) :`
	`1. Date: 17/11/2024, Time: 09:00`
	`Enter the number corresponding to the appointment slot: 1`
	`Appointment scheduled successfully for PA1 on 17/11/2024 at 09:00`
	
	Appointment status changes to “Scheduled”,
	In Patient’s System:
	`Your Scheduled Appointments:`
	`----------`
	`Doctor ID: D001, Doctor's Name: Emily Clarke, Date: 17/11/2024, Time: 09:00, Status: Scheduled`
	
	In Doctor’s System:
	`Doctor D001's Schedule:`
	`Doctor ID: D001, Doctor's Name: John Smith, Date: 17/11/2024, Time: 09:00, Status: Scheduled`
	
	Time slot becomes unavailable to other patients:
	`Enter the number of your choice:`
	`4`
	`Schedule an Appointment`
	`----------`
	`Enter the doctor ID (or type 'exit' to go back to the menu): D001`
	`This doctor does not have any available slots. Please choose another doctor.`

4.3 Doctor Functionality

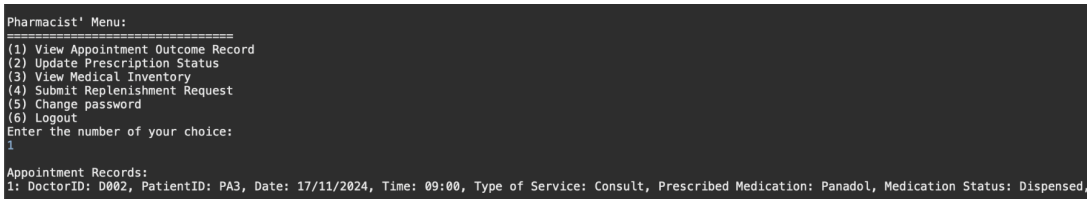
No.	Test Description

1	**Accept or Decline Appointment Requests**
	a) Doctor accepts or declines an appointment request from a patient.
	b) Verify that the appointment status changes to "Confirmed" when accepted or “Cancelled” when declined , and the patient is able to see the updated status of the appointment.
	
	Doctor D002 accepts 1 appointment request and declines 1 appointment request from Patient PA3:
	`Scheduled appointments requests:`
	`--------------------------------------------------`
	`1. PatientID: PA3, Date: 17/11/2024, Time: 09:00`
	`2. PatientID: PA3, Date: 17/11/2024, Time: 10:00`
	`Enter the appointment index to update (or type 'exit' to finish): 1`
	`Confirm or Cancel this appointment? (confirm/cancel): confirm`
	`Appointment status updated to: Confirmed`
	`1. PatientID: PA3, Date: 17/11/2024, Time: 10:00`
	`Enter the appointment index to update (or type 'exit' to finish): 1`
	`Confirm or Cancel this appointment? (confirm/cancel): cancel`
	`Appointment status updated to: Cancelled`
	`Enter the appointment index to update (or type 'exit' to finish): exit`
	
	Patient PA3 is able to view the appointment status as “Confirmed” and “Cancelled”:
	`Your Scheduled Appointments:`
	`------------------------------------------------------------`
	`Doctor ID: D002, Doctor's Name: Emily Clarke, Date: 17/11/2024, Time: 09:00, Status: Confirmed`
	`Doctor ID: D002, Doctor's Name: Emily Clarke, Date: 17/11/2024, Time: 10:00, Status: Cancelled`

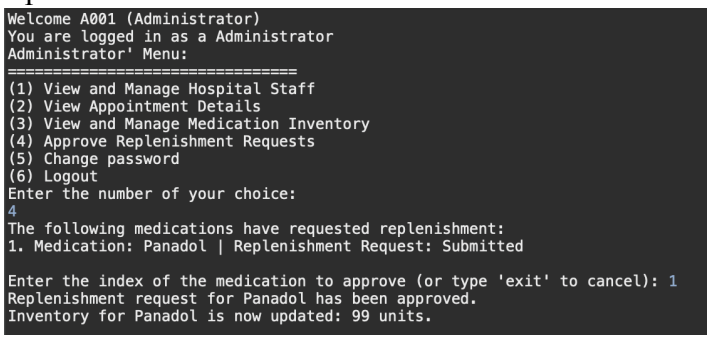
4.4 Pharmacist Functionality

No.	Test Description

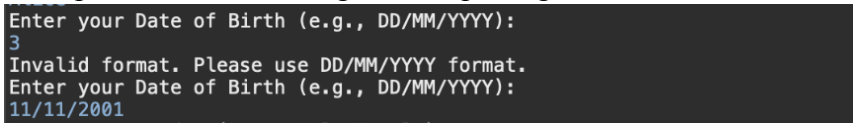
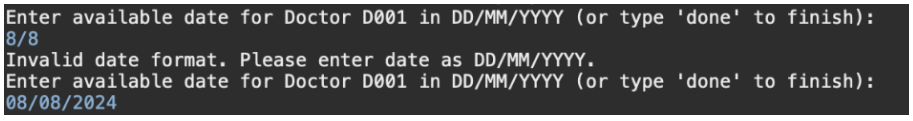
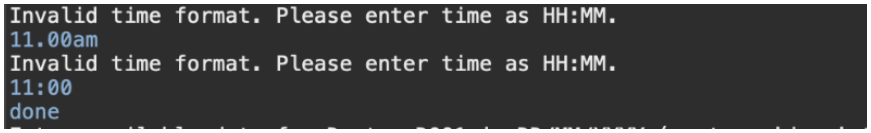
1	**Update Prescription Status**
	a) Pharmacist updates the status of a prescription to "Dispensed."
	b) Verify that the prescription status is updated, and the change is reflected in the patient's records
	
	Pharmacist P001 updates prescription of panadol to PA3, prescription status changed from “Pending” to “Dispensed”:
	`Appointment Records:`
	`1: DoctorID: D002, PatientID: PA3, Date: 17/11/2024, Time: 09:00, Type of Service: Consult, Prescribed Medication: Panadol, Medication Status: Pending, C`
	
	`Enter the index of the appointment to fulfill the prescribed medication or type 'exit' to quit:`
	`1`
	`The appointment's prescribed medication has been fulfilled.`
	
	`Appointment Records:`
	
	`Enter the index of the appointment to fulfill the prescribed medication or type 'exit' to quit:`
	`exit`
	`Exiting the program.`

4.5 Administrator Functionality

No.	Test Description
1	**Approve Replenishment Requests** a) Administrator approves a replenishment request from a pharmacist. b) Verify that the request status changes to "Approved," and the medication inventory is updated accordingly. Administrator A001 approves replenishment request from pharmacist P001 when the medicine stock is low (below 50 units), and 50 additional units will be automatically replenished:  *(Inventory for Panadol updated from 49 units to 99 units)*

4.6 Error Handling

No.	Test Description
1	**Invalid Inputs** Entering an invalid input when for Date, Time, Personal Information. A message will prompt the user to provide a valid input. Example of Error Handling when inputting Patient's Personal Information:   When inputting Date/Time for Appointments:  

5 REFLECTION

5.1 Difficulties encountered and how we conquered them

From this assignment we have learned how important the design principles are in enhancing the efficiency of the code and how to effectively work together as a team. We encountered numerous challenges throughout the assignment. The first issue was the challenge of putting together the code that each of us prepared to form our integrated system. Since much of the classes were highly dependent on each other, one person's code significantly affected another person. Hence, through proper communication we were able to solve the issue. Secondly, because the code system was rather large, using design principles to reduce the redundancy of the code, therefore maximising efficiency was an important lesson that we learned.

5.2 Knowledge Learnt

This project has offered a learning experience in designing and implementing a hospital management system, integrating theoretical knowledge with practical application. Working with file-based storage highlighted the challenges of data management, teaching valuable lessons about handling data, data integrity, and performance bottlenecks.

5.3 Further Improvement Suggestions

Moving away from a console-based interface to a web-based one would drastically enhance user experience. HTML, CSS, and JavaScript could bring the system to life with a clean, interactive frontend. For instance, a web dashboard could allow users to visually manage appointments, inventory, and medical records, rather than relying on text-based commands.

Additionally, the security of staffs' and patients' personal information can be improved. A feature where passwords are securely hashed before being stored in data CSV files would protect sensitive information and reduce the risk of unauthorised access.