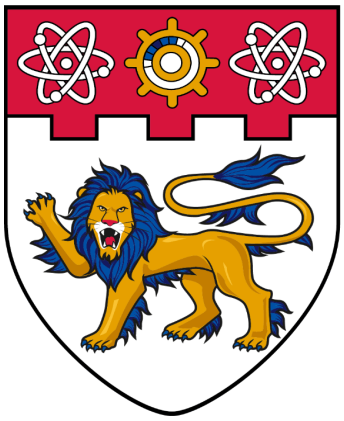

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

PROJECT TITLE: HOSPITAL MANAGEMENT SYSTEM

Github repo: https://github.com/Devininthelab/sc2002-hospital-management-system

Table of contents

1. Introduction

Hospital Management System (HMS) is a solution for automating the management of hospital operations, including patient management, appointment scheduling, staff management, and billing. The system facilitates efficient management of hospital resources, enhances patient care, and streamlines administrative processes. This report details a thorough analysis of the design considerations, principles, and the application of Object-Oriented Programming (OOP) concepts within the HMS project while periodically including references to SOLID design principles. Furthermore, the report includes a link to our UML Class Diagram illustrating the system's architecture, coupled with our test results. Lastly, the report ends with a reflection on the difficulties faced, the lessons learned and future improvements.

2. Design Considerations

2.1 Approach

The Hospital Management System was designed using layered architecture. The overall structure focuses on a database design with 3 layers: Repositories, Entity classes and UI/Boundary/Menu Classes. The 3 layers were designed with the **Single Responsibility Principle (SRP)** in mind, minimising the functionalities of a single class. In the HMS, the Appointment and AppointmentOutcomeRecord repositories are separated instead of having AppointmentOutcomeRecord serve as an attribute of the Appointment class. This allows us to focus solely on scheduling for the Appointment class and the results of the appointment through the AppointmentOutcomeRecord. This ensures extensions to the results functionality will not affect the scheduling of the Appointment class, leading to a robust and scalable system with a clearer code structure.

2.2 Design Patterns

2.2.1 Factory Pattern

Secondly, the Factory Method pattern is similar to having a blueprint for creating objects. An abstract class or interface is implemented by concrete classes. This centralised object creation design makes it easier to maintain and manage multiple complex objects. In the HMS, this can be seen in the implementation of the different User Menu classes, whereby the Menu interface is supported by the singular MenuFactory class containing the implementation for different Menu classes. Additionally, The Factory design pattern within our system utilised the **Open-Closed Principle (OCP)** to define the Menu interface for creating objects, but the actual creation is left to subclasses. The Menu interface is abstracted

to its core functions, and is closed for modification but remains open for extensions such as individual implementations of methods. The MenuFactory class creates different user Menu types using the blueprint of the Menu interface, maintaining core functions like displayMenu, handleChoice and start. As such, the MenuFactory class can be extended to include more menus if more types of staff are introduced, without changing the code for the Menu interface.

2.2.2 Dependency Injection Pattern

Lastly, the Dependency Injection design pattern works by externally providing the required dependencies of a class via parameter passing, instead of letting the class hold them itself. This forms a client-service relationship. For example, the DoctorMenu class depends on the DoctorRepository class, in which the Doctor class acts as client and the DoctorRepository class acts as a service, forming a dependency. In doing so, the Dependency Injection design pattern decouples components from the implementation of their dependencies. This makes it easier to swap in and out different implementations of the DoctorRepository (such as different database access), conforming to the **Dependency Inversion Principle (LSP) and Open-Closed Principle**. Because classes now don't have the additional responsibility to manage their own dependencies, this abides by the **Single Responsibility Principle.** Furthermore, this promotes loose coupling and reduces the impacts of changes within the HMS.

2.3 Extensibility & Maintainability

The system's approach to extensibility and maintainability demonstrates that these concepts can coexist and complement each other when properly implemented. The modular repository structure facilitates easy integration of new data entities and potential migration to different storage methods like CSV, SQL, or NoSQL without affecting other system components. While extensive modularity could potentially introduce additional interfaces and complexity, the system maintains balance through careful adherence to SOLID principles. This approach ensures components remain easy to understand and modify while minimising the impact of changes throughout the HMS. The system's use of Dependency Injection and modular design patterns ensures individual components remain isolated, significantly improving testability and making unit testing and debugging more efficient through independent component testing. However, the design acknowledges that excessive focus on extensibility could lead to maintenance challenges due to increased edge cases and dependencies. To address this, the system implements loose coupling between components, allowing for system extensions while maintaining code clarity and manageability, effectively balancing the trade-off between extensibility and maintainability requirements.

2.4 Trade-Offs

2.4.1 Abstraction vs Granularity

One serious consideration in the design thinking process was balancing between granularity and abstraction. Granularity is the level of detail of the components being considered. Abstraction, a key OOP concept, is the process of simplifying complex systems by hiding underlying details, exposing only necessary components. Given the contradicting natures of these concepts, we had to consider performance needs, maintainability and flexibility. A higher granularity and lower abstraction grants greater control but more complexity and vice versa.

2.4.2 Data Integrity vs Complexity

Another significant trade-off involves data integrity versus system complexity. By assuming single-user operations, the system avoids complex concurrency management but sacrifices data integrity capabilities in multi-user scenarios. This decision was deliberately made to align with project constraints and simplify implementation.

2.5 Object-Oriented Programming (OOP) Concepts

2.5.1 Abstraction

Abstraction involves hiding internal details and showing only functionalities, focusing on what an object does rather than how it does it. For example, abstracting the Menu interface to its essential methods enables the creation of different user Menus without causing much adjustments to the current code, leaving the implementation of the functionalities to the individual Menu classes.

2.5.2 Encapsulation

Encapsulation is used to build a barrier around an object's data, preventing the unauthorized access of data. This is achieved using the private access modifiers in the classes. For instance, in the Staff class, attributes like id and password are private. These data can only be accessed using the public, get and set methods. These methods allow the data to be accessible and modified.

2.5.3 Inheritance

Inheritance allows us to derive new classes from existing classes. The child class inherits data attributes and methods from the parent class. Inheritance is helpful as it reduces developer effort and code length, and it enables the reuse of code when implementing new classes. For example, the Administrator class inherits from the Staff class.

2.5.4 Polymorphism

Polymorphism allows an object reference to be treated as different types. For example, our system utilises polymorphism in the mainMenu class which has the roleRedirect method to create the required user Menu class and calls the Start method. The Start method as defined in the Menu interface allows for the acceptance of different users for polymorphism. This causes the different displays of the respective user menus to be run despite being called by the same method. Through polymorphism, our code reduces redundancy and is more reusable.

2.6 Assumptions Made

- No new patients added beyond what is found in the repositories
- Working hours of the doctors are consistently 6 days a week and 8 hours a day
- Weekly calendar scheduling: one 6x8 table to represent the schedule, the slots before today belongs to the next week and after today belong to this week
- Once a patient scheduled a particular time slot, even if it hasn't been accepted, no other patients can schedule that time slot, i.e. only one patient can request per time slot
- There is only one user using the application at any given time, that is, no centralised database,
- Ignore adversarial input (alleviate the work to validate user input)
- The system is intended for lightweight use with a manageable number of records, making CSV a practical choice.
- Each user role can only perform operations explicitly assigned to their menu (e.g., doctors cannot view inventory).

3. UML Class Diagram

Click here to open.

4. Testing

4.1 Patient Test

Test ID	Description	Test steps	Expected outcome	Status
P01&P02	Change password	1. Chooses to update password 2.1 Enters new password with a length of at least 6 2.2 Enters new password with a length of less than 6	2.1 Password updated successfully - Next time patient log in, old password should not log in, new password should log in 2.2 Password not updated - Reprompt for new password	Passed
P03	View medical record	1. Chooses to view medical record	Display patient's medical record, i.e. Patient ID, Name, Date of Birth, Gender, Contact Information, Blood Type, and Past Diagnoses and Treatments.	Passed
P04	Update personal information	1. Chooses to update personal information 2. Choose to either update their name, date of birth, contact or gender	- Personal information updated - When viewing medical record, change is reflected	Passed
P05	View available appointment slot	1. Chooses to view available appointment slot 2. Enter doctor ID	- Display a week calendar with the available/busy/booked state at each entry	Passed
P06& P07	Schedule appointment	1. Choose schedule an appointment 2. Select day, time slot and doctor	2.1 Doctor is available, scheduling successful - Show up on view scheduled appointments and doctor schedule 2.2 Scheduling failed (doctor is BUSY or BOOKED) - No appointment added, doctor schedule remains the same	Passed
P08&P09	Reschedule appointment	1. Choose reschedule an appointment - Enter appointment id - Selects day, time slot and doctor	1. Rescheduling successful (requested/rejected) - Appointment detail updated - Doctor schedule: free up old slot, new slot changed to BOOKED	Passed

			2. Rescheduling failed (accepted/completed) - Appointment and schedule remain the same	
P10&P11	Cancel appointment	- Chooses cancel an appointment - Enters appointment id	1. Appointment cancelled (not completed) - Appointment disappear on appointment list - Doctor schedule become AVAILABLE 2. Cancellation failed (completed)	Passed
P12	View scheduled appointment	- Chooses view scheduled appointment	- Display 3 tables corresponding to REQUESTED, ACCEPTED and REJECTED	Passed
P13	View appointment outcome record	- Chooses view past appointment outcome record	- Display outcome record of every COMPLETED appointment	Passed

4.2 Doctor Test

Test ID	Description	Test steps	Expected outcome	Status
D01	View patient medical records	- Doctor select view patients medical record - Doctor enter patient's id	- Display patient's medical record, including Patient ID, Name, Date of Birth, Gender, Contact Information, Blood Type, and Past Diagnoses and Treatments.	Passed
D02	Update patient's medical record	- Doctor select update patients medical record - Doctor add either diagnoses, treatment plan, or prescription	- Medical record is updated successfully, reflecting the new information.	Passed
D03	View personal schedule	- Doctor select view personal schedule	- Display a week calendar with the available/busy/booked state at each entry	Passed
D04&5	Doctor set schedule	- Doctor select set availability for appointments - Doctor choose date and time slot - Doctor choose AVAILABLE or BUSY as new status	1. Availability is updated accordingly (not booked) - Changes reflected in "View personal schedule" 2. Fail to update schedule (booked) - Doctor's schedule remain the same	Passed

D06&7	Doctor accept appointment request	1. Doctor choose manage appointment request 2.1 For a certain appointment, doctor choose y 2.2 For a certain appointment, doctor choose n	2.1 Appointment accepted - Appointment status is reflected on patient's menu and view upcoming appointments 2.2 Appointment rejected - Appointment status is reflected in patient's menu and personal schedule is freed up	Passed
D08	Doctor view upcoming appointment	- Doctor choose view upcoming appointments	- Display a table of upcoming appointment, that is all ACCEPTED appointments	Passed
D09&10	Doctor complete appointment	1. Doctor choose complete an appointment 2. Doctor enter appointment id 2.1 Doctor does belong to appointment - Type of service, prescription and quantity, and consultation notes 2.1 Doctor does not belong to appointment	2.1 Appointment completed - Doctor schedule's free up (AVAILABLE) - Patient can see their Appointment Outcome Record 2.2 Appointment not completed - Doctor don't have permission to complete the appointment	Passed
D11	Doctor change password	1. Doctor choose change password 2.1 Doctor enter new password with at least 6 characters 2.2 Doctor enter new password with less than 6 characters	2.1. Password changed successfully - Next login require new password 2.2. Password not changed - Prompt for new password again	Passed

4.3 Pharmacist Test

Test ID	Description	Preconditions	Test steps	Expected outcome	Status
P10	View Appointment Outcome Record	Pharmacist is logged in	1.Choose View Appointment Outcome Record	Display all Appointment Outcome Records	Passed
P20	Dispense Prescription	Pharmacist is logged in	1.Choose Dispense Prescription 2.Type id that does not in database of AOR	"Appointment Outcome Record not found or Appointment does not exist."	Passed

P21	Dispense Prescription	Pharmacist is logged in	1.Choose Dispense Prescription 2.Type id that does not have any prescription or all prescriptions are dispensed	"No pending prescriptions for this appointment."	Passed
P22	Dispense Prescription	Pharmacist is logged in	1.Choose Dispense Prescription 2.Type id that has pending prescriptions. 3.Type any string that does not in Pending Prescriptions shown. (1) 4.Type prescription that has stock level in Inventory larger than quantity to dispense.(2) 5.Type again the same prescription more times (3) 6.Type prescription that has stock level in Inventory smaller than quantity to dispense.(4) 7.Type enter to finish(5)	(1)"Prescription not found in pending list. Please try again." (2)"Prescription ibuprofen dispensed successfully." StockLevel in Inventory reduces (3)"This prescription has already been dispensed. Please select another." (4)"Medicine stock level not adequate. Please submit replenishment request." (5)"Finish dispensing"	Passed
P30	View Medication Inventory	Pharmacist is logged in	1.Choose view Medication Inventory 2.Type y to show low stock medicines	All low stock medicines display	Passed
P31	View Medication Inventory	Pharmacist is logged in	1.Choose view medication inventory 2.Type n to show all medicines in inventory	All medicines display	Passed
P40	Submit replenishment request	Pharmacist is logged in	1.Choose submit replenishment request 2.Type enter without typing any prescriptions (1) 3.Type any string that isn't shown in list of low stock medicines(2) 4.Type medicines that are shown in list of low stock medicines(3) 5.1.Type y to submit(4) 5.2.Type n to cancel submit(5)	(1)"No medicines were selected for replenishment." (2)"This medicine is not low on stock or does not exist. Try again." (3)"Medicine has been added to the request successfully." (4)"Replenishment request submitted." New request in database. (5)"Request cancelled."	Passed
P50	Update password	Pharmacist is logged in	1.Choose Change Password 2.1.Type new password that is not the same as the old	"Password Updated Successfully" New password in database	OK

			password and has more than 6 characters		
P51	Update password	Pharmacist is logged in	1.Choose Change Password 2. Type new password that is not the same as the old password and has less than 6 characters	“Password must be at least 6 characters”	OK
P52	Update password	Pharmacist is logged in	1.Choose Change password 2. Type new password that is the same as the old password	“Same password. Please change to the new password.”	OK

4.4 Administrator test

4.4.1 Staff Management

Test ID	Description	Preconditions	Test steps	Expected outcome	Status
AD1	Add Staff Member	Administrator is logged in.	1. Select option 1 from the menu. 2. Enter Staff ID, Name, Password, Role, Gender, Age	"Staff S001 added successfully" is displayed, and the staff is added to the database.	Passed
AD2	Remove Self (Admin)	Administrator is logged in.	1. Select option 3 from the menu. 2. Enter the logged-in admin's own ID.	"You cannot remove yourself." message is displayed.	Passed
AD3	View Staff List Sorted by Role	At least one staff member exists in the system.	1. Select option 4 from the menu. 2. When prompted, enter "Role".	Staff list is displayed, sorted by role.	Passed

4.4.2 Appointment Management

Test ID	Description	Preconditions	Test steps	Expected outcome	Status
AD4	View Scheduled Appointments	At least one appointment exists in the system.	1. Select option 5 from the menu. 2. Observe the output.	Appointment details are displayed in table format.	Passed
AD5	Update Appointment Status	At least one appointment exists in the system.	1. Select option 6 from the menu. 2. Enter Appointment ID, Status	"Appointment status updated successfully." is displayed.	Passed

4.4.3 Inventory Management

Test ID	Description	Preconditions	Test steps	Expected outcome	Status
AD6	View Medicine Inventory (Empty)	No medicines exist in the system.	1. Select option 7 from the menu. 2. Observe the output.	"No medicines in inventory." is displayed.	Passed
AD7	Add Medication to Inventory	Administrator is logged in.	1. Select option 8 from the menu. 2. Enter Name, Stock, Low Threshold, High Threshold	"Paracetamol added to inventory." is displayed.	Passed

4.4.4 Replenishment Request Management

Test ID	Description	Preconditions	Test steps	Expected outcome	Status
AD8	Show All Replenishment Requests (Empty)	No replenishment requests exist in the system.	1. Select option 13 from the menu. 2. Observe the output.	"No replenishment requests." is displayed.	Passed
AD9	Approve Replenishment Request	A valid replenishment request exists with ID: 1.	1. Select option 14 from the menu. 2. Enter Request ID: 1.	"Request approved. Inventory updated." is displayed. Update in database of both medicines and requests.	Passed

4.5. Account access

Test ID	Description	Preconditions	Test steps	Expected outcome	Status
AC1	First time log in	First time log in	1. User choose log in 2. User type in their role 3. User type in their ID and default password	1. Login successfully 2. Role-specific menu is displayed	Passed
AC2	Login failed	User have an account	1. User choose log in 2. User type in their role 3. User type in their ID and wrong password	1. Password incorrect 2. Prompt user to try again until correct password	Passed

5. Reflection

Problems Encountered:

While working on the project, we encountered certain bugs regarding data format inconsistencies, schema mismatches and small implementation bugs. Data format inconsistencies like Date/Time, Number formats and serialisation issues arise while working on the Appointment and AppointmentOutcomeRecord repositories and the scheduling of such appointments. Schema mismatches occurred as well due to the different parts of the HMS having different expectations of how data should be structured. Small implementation bugs also subtly surfaced, leading to some unexpected behaviour that required us to meticulously pay attention to the details in our code. The difficult process of identifying edge cases and subconscious assumptions made within our design necessitated us to observe patiently the intricate logic of the HMS, consuming a significant amount of time.

Furthermore, designing a complex and highly connected system like the HMS needed efficient management of resources and removal of redundancies, for example, the MenuFactory Class emulating the Singleton design pattern. Given the potentially large scaling up of users in the HMS, it is crucial to reduce overheads as much as possible. Hence, the design considerations posed another challenge.

Lastly, user experience had to be seamless, so ensuring the interface was intuitive, robust and provided error feedback when handling user inputs was vital.

Knowledge Gained:

Through the implementation of the HMS, we learned the applications of different design patterns to facilitate our solution and improve the efficiency of our code. Moreover, we honed our problem-solving and debugging skills, especially when integrating various components.

Future Improvements:

An alternative approach to our design may have utilised a service layer and object serialisation.