

**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE






SC2002 Hospital Management System Report

Declaration of Original Work for SC2002 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
ARIEL KWOK DAI HUI (U2321403K)	SC2002	SCS4 - Group 5	 20 Nov 2024
CHARMAINE LIEW YINGSUI (U2322168J)	SC2002	SCS4 - Group 5	 20 Nov 2024
LEE JIA QIAN VALERIE (U2322878H)	SC2002	SCS4 - Group 5	 17 Nov 2024
LI LIYI (U2220985F)	SC2002	SCS4 - Group 5	 20 Nov 2024
NICHOLAS CHANG CHIA KUAN (U2322962F)	SC2002	SCS4 - Group 5	 20 Nov 2024

College of Computing & Data Science
Nanyang Technological University (NTU)

1. Introduction

1.1. About HMS

Hospital Management System (HMS) is an application aimed at automating the management of hospital operations, including patient management, appointment scheduling, staff management, and billing. The system is expected to facilitate efficient management of hospital resources, enhance patient care, and streamline administrative processes.

1.2. Approach Taken

The system implementation is split into high and low-level packages.

High-level packages are the view classes, which provide UI for hospital staff and patients to interact with the system.

Low-level packages are object entities that are supposed to mirror real-world objects with which the user can interact. Low-level objects act like models, and Managers store an array of these models and manage the interaction of the high-level classes with them.

Managers also enforce error checking and exception catching to ensure that models are manipulated as designed. Our system was designed with classes within each package to be encapsulated from each other. Interaction between classes is handled by interfaces to reduce the overall coupling in the system.

1.3. GitHub Link

https://github.com/liliyigz/SC2002-OOP-Grp_Proj

2. Design Considerations

2.1. Principles Involved

2.1.1. Abstraction

As the staff (administrator, doctors, pharmacists) and patients have similar log-ins and information as attributes, we create a User Class with these attributes declared as private and instance methods to get and set these private attributes.

2.1.2. Inheritance

The Patient and Staff Classes inherit from the User Class, while the Administrator, Doctor and Pharmacist Classes indirectly inherit from the intermediary class - Staff Class.

2.1.3. Polymorphism (Overloading)

Create the related role-application object with the hospitalId of the logged-in user as the initial input value.

2.1.4. Encapsulation & Information Hiding

Encapsulation is for data protection. All member attributes are declared private to restrict direct access and their states for each object attribute are only accessible through the get functions, and can only be overwritten through the set functions defined in their respective classes.

2.1.5. SOLID Principles

2.1.5.1. Single Responsibility Principle (SRP)

SRP states that each class should have a clear singular purpose. This principle is well-adopted in our HMS and is explained in section 2.2.

2.1.5.2. Open-Closed Principle (OCP)

OCP states that each class should not be modified but can still be open for extension. In our HMS, this is directly supported by our application architecture, where adding new roles such as Receptionist or Pharmacist can be extended with the User class in the entity package. While adding these new roles, the User class is not affected, while new functionalities like role-specific permissions can be added seamlessly.

2.1.5.3. Liskov Substitution Principle (LSP)

LSP states that all subclasses must be substitutable for their parent classes. This is well-supported in our HMS by ensuring that subclasses like Doctor and Pharmacist, which inherit from the Staff class, maintain the behavior defined by their parent. For instance, a Doctor object can be passed wherever a Staff type is expected,

such as in staff scheduling or reporting modules, without causing any disruptions or unexpected behavior.

2.1.5.4. Interface Segregation Principle (ISP)

ISP states that a subclass should not depend on interfaces that they do not use. This is well-supported in our HMS through the use of the role-specific interfaces. For example, the AppointmentHandler interface is only implemented by classes that deal with appointment management, such as Receptionist, while the other roles like Pharmacist are not involved by these unnecessary methods. This ensures that each class only depends on the interfaces relevant to its function.

2.1.5.5. Dependency Injection Principle (DIP)

DIP states that higher-level classes should not depend on concrete implementations of an object but rather on abstractions of it. This is well-supported in our HMS where high-level classes such as AdministratorControl depend on abstractions like UserRepository rather than concrete implementations. This is widely used in our application where we would refer to objects by their base class or interface, rather than the object itself.

2.1.6. Model - View - Controller (MVC) Structure

2.1.6.1. Model

It represents the data and rules of the application. Which manages the state of the application and notifies the View of any data changes.

2.1.6.2. View

It gets the updated Model and displays data to the user based on the user input.

2.1.6.3. Controller

It is the intermediary between the Model and the view which processes user inputs and updates the Model or View as needed. In our application, an example of the Controllers is the repository packages.

2.2. Packages & Classes

2.2.1. constants package

FilePath Class is declared as final to prevent inheritance and since it just provides the class/static variables, which are the paths to the Excel files we are using as databases, to retrieve and store information, instantiation is unnecessary. Therefore, the constructor is made private.

2.2.2. enums package

It contains classes to define the states or options for gender, role and appointment status. It is used to make code more readable, maintainable, and less error-prone.

2.2.3. main package

It contains all classes that are the application for the different users, essentially for users to interact with the application

2.2.4. models package

It contains classes of all user types, Medication, MedicalRecord and Appointment. Each class consists of their respective attributes, get and set methods to extract or change their respective private attributes.

2.2.5. repository package

It contains classes for handling of data which includes updating, retrieving and storing of information.

2.2.6. security package

It contains the Encryptor class, which is used to encrypt the password.

2.2.7. services package

It contains classes with methods to fulfill the actions required by the respective roles.

2.2.8. utility package

The package contains 2 classes, DateUtils and Validator, and both classes contain static methods that are constantly reused in various parts of the Hospital Management System application.

2.2.8.1. DateUtils

It contains methods to extract and format the date and time of the appointments.

2.2.8.2. Validator

It contains static methods to prevent errors resulting from invalid user input, by validating:

1. If user inputs contain integers for scenarios that only allow integer inputs
2. If user inputs are empty for scenarios that require string inputs
3. If patient input for contact number only contains 8 digits
4. If patient input for email address is in the basic email format

2.3. Assumptions made

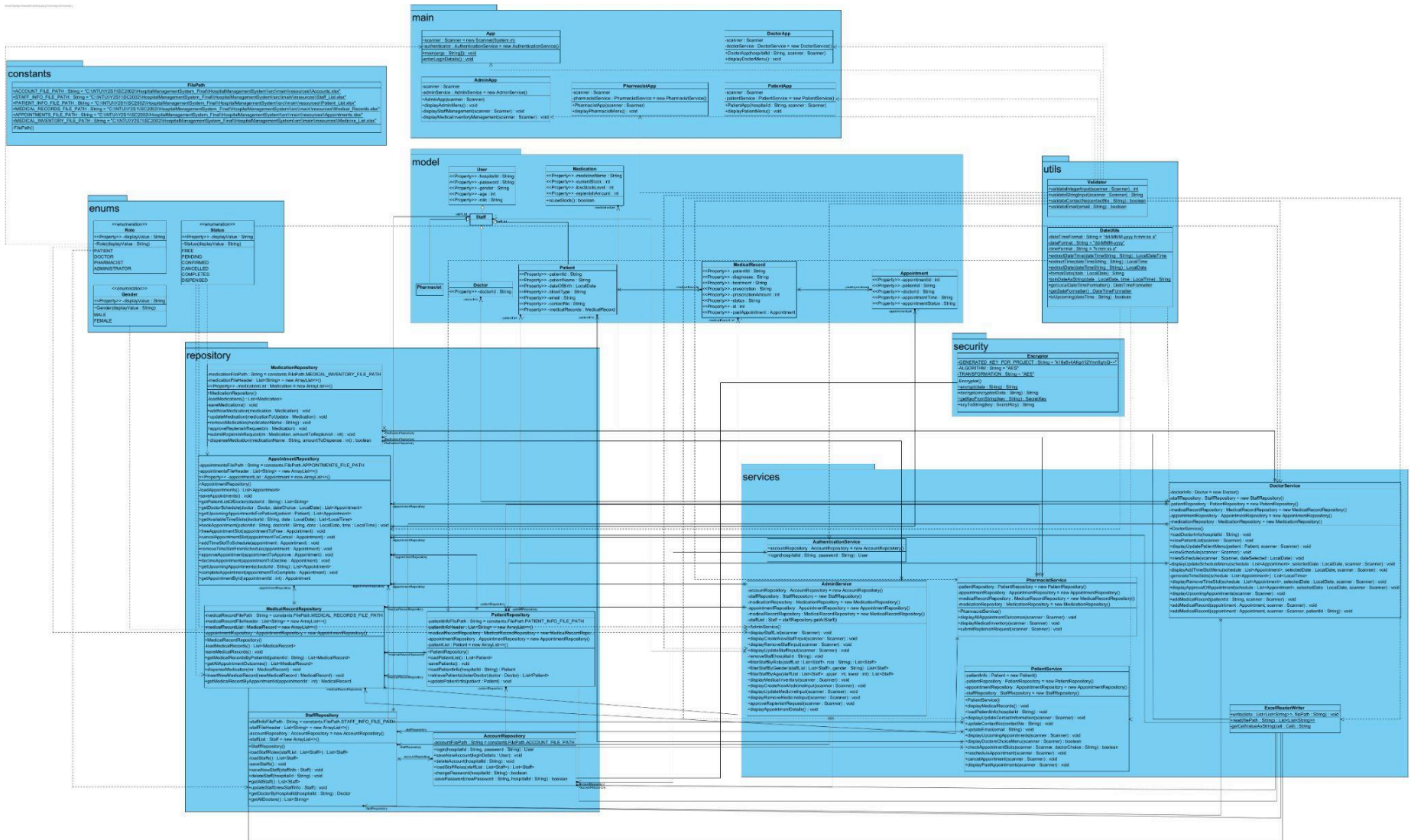
1. The default password: “password” is only temporary for new accounts and is not allowed to be set as the users’ password when prompted to change the password on first login.
2. All Users have fixed HospitalIDs.
3. There is only 1 administrator.
4. There are only 3 patients.
5. The administrator can only update the staff’s age, gender and role.
6. The addition, removal, update of stock levels and update of low alert level for the medications are all under managing the inventory of medication.
7. The Excel files are the databases, used to store and retrieve information.
8. Patients only need to view upcoming appointments that are either in “PENDING” or “CONFIRMED” status.
9. Patients can only view available time slots when attempting to book or reschedule an appointment.
10. Type of Service provided from the doctor are the treatments.
11. Consultation notes from the doctor are the diagnosis.

2.4. Additional Features

2.4.1. Encryption

Since we have to store the passwords of the user accounts in the database, it would be a security breach to store the passwords in plain text. Anyone that has access to the database, would be able to see all the passwords of the accounts, hence we implemented an encryption service using AES128. Users’ encrypted passwords are stored in the excel files instead of the plain text. During login, the imputed passwords are encrypted and matched with the encrypted passwords stored in the database to check if they are the same.

3. UML Diagram



4. Testing (Referring to the list of sample test cases provided in OOP ASSIGNMENT- HMS.pdf APPENDIX A)

Test Case	Output	Test Case	Output
Patients Menu	<pre> Welcome to Patient Menu! 1 View Medical Records 2 Update Contact Information 3 View Upcoming Appointments 4 Book Appointment 5 Reschedule Appointment 6 Cancel Appointment 7 View Appointment Outcome Records 8 Logout </pre>	3. View available appointment slot	<pre> 3 - No Upcoming Appointments! 3 --- Upcoming Appointments --- Appointment with D1001 at 21-Nov-2024 8:00:00 AM Status: PENDING Appointment with D1001 at 21-Nov-2024 12:00:00 PM Status: PENDING ----- 1 Reschedule 2 Back </pre>
1. Patient view medical record	<pre> 1 -- Medical Record -- Patient ID : P1001 Patient Name : Alice Brown Date of Birth : 14-May-1980 Gender : FEMALE Blood Type : A+ Contact No. : 91234567 Email : patient01@gmail.com ----- Diagnosis : Fever Treatment : More Rest Needed Prescription : 5x Paracetamol ----- Diagnosis : Diabetes Treatment : Insulin shot Prescription : 1x Paracetamol ----- </pre>	4. Schedule an appointment	<pre> 4 ---Doctor Choice Menu--- 1 - D1001 1 -- Available Appointment Slots--- 1 - 2024-11-21 2 - 2024-11-22 1 --- Time Slots Available for 2024-11-21--- 1 - 08:00 2 - 12:00 --- Book Slot for 2024-11-21--- 1 </pre>
2. Update personal Information	<pre> 2 -- Update Contact Information -- 1 Contact Number 2 Email 1 Enter New Contact Number: 88888888 Welcome to Patient Menu! 1 View Medical Records 2 Update Contact Information 3 View Upcoming Appointments 4 Book Appointment 5 Reschedule Appointment 6 Cancel Appointment 7 View Appointment Outcome Records 8 Logout 2 -- Update Contact Information -- 1 Contact Number 2 Email 2 Enter New Email Address: alicebrown@gmail.com -- Medical Record -- Patient ID : P1001 Patient Name : Alice Brown Date of Birth : 14-May-1980 Gender : FEMALE Blood Type : A+ Contact No. : 88888888 Email : alicebrown@gmail.com </pre>	5. Reschedule an appointment	<pre> ----- 1 Reschedule 2 Back 1 --- Upcoming Appointments --- 1 Appointment with D1001 at 21-Nov-2024 8:00:00 AM Status: PENDING 2 Appointment with D1001 at 21-Nov-2024 12:00:00 PM Status: PENDING Reschedule: 1 ---Doctor Choice Menu--- 1 - D1001 1 -- Available Appointment Slots--- 1 - 2024-11-21 2 - 2024-11-22 2 --- Time Slots Available for 2024-11-22--- 1 - 13:00 --- Book Slot for 2024-11-22--- 1 </pre>

Additional Test Cases Here: [SC2002-Additional-Test-Cases-SCS4_Group5](#)

5. Reflection

5.1. Difficulties encountered

5.1.1. Low Cohesion of the Initial Design Process

At the start of the project, we split the classes without much planning and everyone was given a role to code the classes (Admin, Doctor, Patient, Pharmacist & Integration). Soon after we realized that each class would be loaded with many functionalities (low cohesion) as there are system requirements for each of the roles.

Method to resolve: We revised the UML class diagram to identify the common different classes required, and further split the work by user interface + encryption + integrations, data handling and user functions.

By doing so, we achieved high cohesion in each class by providing them with a focused purpose, using methods only relevant to that particular function.

5.2. Further improvement suggestion

5.2.1. Appointments

Currently, our design only allows users to book appointments for tomorrow and the day after for simplicity's sake, but it could be improved to display all dates with available slots, of the chosen doctor. This can potentially improve user experience, as the current implementation for displaying the dates, might potentially not contain any available slots for the doctor of choice.

5.2.2. Receptionist

For now the application is unable to create new patients, which is an unlikely case for hospitals to have a fixed number of patients only. Hence, we could add a new receptionist role to facilitate new patients in creating new accounts.

5.2.3. Pharmacist with billing capabilities

In most cases, patients would need to collect medications first before payment in the hospitals. To better streamline the process, we could add another billing feature, where the application would not only return the type and amount of medication the patient needs but also the payment amount all in one. Patients would also be able to view the billing receipt when they view past appointments.