# 

SC2002 Object Oriented Design & Programming

Hospital Management System

AY24/25 Semester 1 | SCS4 | Group 6

ADITI VASUDEVAN : U2323375J

GAY MING KAI : U2320596C

LEE HAN HUA : U2322351B

LIANG QIAOYUN : U2322330D

Github Repository Link: : <https://github.com/AuroraVane/SC2002_Project.git>

Project Video Link: <http://youtube.com>

**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 (SC2002) | Lab Group | Signature/Date |
| --- | --- | --- | --- |
| Aditi Vasudevan | SC2002 | SCS4 |  |
| Gay Ming Kai | SC2002 | SCS4 |  |
| Lee Han Hua | SC2002 | SCS4 |  |
| Liang Qiaoyun | SC2002 | SCS4 |  |

# **1. Design Considerations**

## **1.1. Our Approach**

Our HMS system is designed with its classes separated into controller, boundary, and entity categories to ensure high cohesion and loose coupling . When a user interacts with the system, they interact with the boundary classes, which in turn interact with controller classes to complete operations. These controllers, in turn, interact with the entity classes to manage and persist data. This layered architecture promotes a clean separation of concerns, making the system more maintainable and scalable.

Our system also operates in a continuous loop that makes it self sustainable - once initiated, it remains active until manually terminated**.** This allows for multiple users to log in and out sequentially without requiring the entire system to restart.

## **1.2. Assumptions**

The HMS system is designed to operate in a single-user, single-instance environment. This assumption ensures data integrity and prevents conflicts that might arise from concurrent access, such as multiple users editing the same file simultaneously. Additionally, we assume that users can follow instructions to give logical and correct formatted inputs, and that there's no limit to the storage size of the system.

## **1.3. Applied Design Principles**

## **1.3.1. Single Responsibility Principle(SRP)**

Single Responsibility Principle states that each class assumes a single well-defined responsibility such that it does not have more than one reason for it to be changed. By assigning single responsibility to classes with Entity, Boundary, Controller, and more user specific roles we can minimise the ripple effects of changes, and improve code modularity and allow for easier testing and modification.

## **1.3.2. Open/Closed Principle(OCP)**

Open Closed Principle states that software entities (classes, modules, functions, etc.) should be open for extension but closed for modification, meaning new features can be added without modifying existing code. OCP can be applied via abstraction, inheritance and polymorphism.

In our project, we applied OCP by creating an abstract class “User<Entity>” that is extended to different types of Users such as “Staff” and “Patient”. The subclasses override getRole(), displayUI(), allowing us to implement role specific displayUIs for new users without modifying the existing code. Hence, increasing the flexibility and maintainability of our HMS system.

## **1.3.3. Liskov Substitution Principle(LSP)**

Liskov Substitution Principle states that the subtypes must be substitutable for the base types. In our system, ‘Staff’ class objects are substitutable by all its subclasses ‘Doctor’, ‘Administrator’ and ‘Pharmacist’, all while ensuring the methods behave correctly. When trying to get the role of different users, we use different instances of requests to call ‘getRole()’, and a method in the subclass will be called and returns the correct role of the user. This promotes code reusability and reduces the likelihood of introducing errors when adding new subclasses.

## **1.3.4. Interface Segregation Principle(ISP)**

Interface segregation principle states that many specific interfaces are better than one general interface. In accordance with the Interface Segregation Principle (ISP), we decomposed our entity interfaces into ‘UserUI’ and ‘AppointmentUI’ interfaces. This strategy mitigates the risk of bloated interfaces and ensures that entity classes implement only the methods necessary for their respective functionalities. By promoting modularity and reducing coupling, we improve the overall maintainability and extensibility of our system.

## **1.3.5. Dependency Injection Principle(DIP)**

Dependency injection principle suggests that higher modules must not depend on lower modules, but both should depend on abstraction. This means that we should depend on interfaces rather than directly depending on concrete classes to perform any same operations, this makes the system more maintainable because we do not have to individually edit every concrete class.

In our system, when navigating and displaying the menu for each type of user, we depend on the <UserUI> interface instead of individual concrete classes of ‘PatientUI’, ‘DoctorUI’ etc. Therefore, allowing us to introduce more UIs for different users with minimal effort later, making our system extendable.

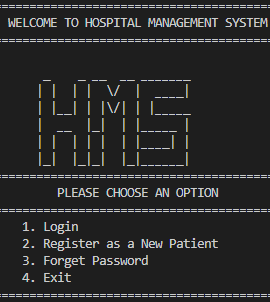
## **1.4. Boundary Controller Entity Framework (BCE)**

BCE is a design pattern that organises software into 3 layers: Boundary (Handle Interactions), Controller, (Handle Logic) and Entity (Core Data Model). We made use of it by decoupling our classes to Boundary, Entity and Controller classes so that the classes would not be too cluttered and adhere to SRP and allow room for DIP.

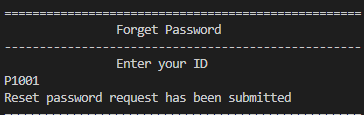
## **1.5. Factory Pattern**

The Factory Pattern specifically Factory Method is a design pattern that provides a way to create objects without specifying their exact class. We employed it in our project's login process to encapsulate object creation- the authenticate method can process different user types without explicit knowledge of their specific subclasses. This decoupling allows for a more flexible and extensible authentication system.

# **2. Additional Features & Functionality Implemented**



| **Register New Patient** : *Users* can self register at the Hospital start page |
| --- |



| **Forget Password** : A reset password request can be made anytime if *Users* forget their password, an *Admin* can then approve the request, |
| --- |

| **Password Hashing** - SHA3-256 : Passwords are encrypted with SHA3-256 algorithm to enhance security |
| --- |

| **Billing** - When an appointment is completed, the patient is billed based on appointment and medicine. |
| --- |

| **Backtracking** - Allows *users* to return back to the menu at any point in time |
| --- |

| **Colour Code -** Every different role has a different colour associated to its UI |
| --- |

# **3. Reflection**

The initial phase of our Hospital Management System (HMS) project was marked by several challenges. One significant hurdle was the development of a comprehensive Unified Modeling Language (UML) diagram, as our team members had varying levels of understanding of object-oriented programming (OOP) concepts and the Boundary, Control, Entity (BCE) framework.

To overcome this, we opted for a pragmatic approach, focusing on building a functional skeleton first. This allowed us to iterate on the design and incorporate OOP principles as we progressed.

Another challenge was the distribution of work among team members. While dividing tasks based on roles seemed straightforward, the interdependencies between different components required careful coordination and collaboration. To address this, we implemented a pair-programming approach, which facilitated knowledge sharing and improved code quality.

In the second phase of the project, we focused on incorporating OOP principles such as inheritance, polymorphism, and the SOLID principles. While we faced some challenges in applying these principles strictly, we made significant strides in improving the design and maintainability of our code. However I feel that one thing we discovered should be addressed. Prior to JDK 8, which the syllabus is based on, interface cannot have static methods while for JDK 8 onwards interface can have static methods. Since we are taking OODP in the form of JDK 7, in the end we chose to not add static methods into interfaces as a form of respect to the syllabus.

Overall, this project has been a valuable learning experience. We have gained a deeper understanding of OOP concepts, software design principles, and teamwork. We believe that our project, while still under development, has the potential to be a robust and user-friendly hospital management system.

# 

# **4. UML Diagram**

Outside of report per Dr Li Fang Lecture on week12\_lect2\_fri (Timestamp: 40.21)

# **5. Testing**

**5.1 Login System and Password Management**

**5.1.1 First-Time Login & Password Change**

| Upon entering the system, a user may login with their ID & password. A newly registered user may login with the default password: “password”. Here we use new patient Alice, as an example. |
| --- |
| The user may then change their default password. The user is then able to login again with the updated password |

**5.1.2 Login with Incorrect Credentials**

| When a user attempts to log in with an incorrect password. The system displays an error message indicating invalid credentials, and login is denied. |
| --- |

**5.2 Patient Actions**

**5.2.1 View Medical Record**

| When a patient logs in they view the following menu, and choose from 10 domains. Here they can view their own medical record. The system displays the Patient's ID, Name, Date of Birth, Gender, Contact Information, Blood Type, and Past Diagnoses and Treatments. |
| --- |

**5.2.2 Update Personal Information**

| Patients may update their email address. The patient's contact information is updated successfully, and the changes are reflected in the medical record. |
| --- |

**5.2.3 View Available Appointment Slots**

| Patient views available appointment slots with doctors. The system displays a list of available appointment slots, showing doctors' names, dates, and times. |
| --- |

**5.2.4 Schedule an Appointment & View Scheduled Appointments**

| Patients may schedule a new appointment. The appointment is scheduled successfully, pending approval by the doctor. Patients may view their list of scheduled & pending appointments. The system displays all upcoming appointments with details like doctor name, date, time, and status. |
| --- |
| The selected time slot becomes unavailable to other patients. Patients are prevented from booking a time slot that is booked. |

**5.2.5 Reschedule an Appointment**

| Patient reschedules an existing appointment to a new slot. The appointment is rescheduled successfully. We can see the previous time slot becomes available, and the new slot is reserved. |
| --- |

**5.2.6 Cancel an Appointment**

| Patients may cancel an existing appointment. The appointment is cancelled successfully, and the time slot becomes available for others. |
| --- |

**5.2.7 View Past Appointment Outcome Records**

| Patients may view past appointment record details including service, prescribed medications, and notes. |
| --- |

**5.3 Doctor Actions**

**5.3.1 View Patient Medical Records**

| When a Doctor logs in they see the following menu, where they can choose from 9 domains. Here they may view medical records of patients under their care. |
| --- |

**5.3.2 Update Patient Medical Records**

| Doctors may add a new diagnosis and treatment plan to a patient's medical record. The medical record is updated successfully, reflecting the new information. |
| --- |

**5.3.3 View Personal Schedule**

| Doctors may view their personal appointment schedule. The system displays the doctor's upcoming appointments and availability slots (null patients). |
| --- |

**5.3.4 Set Availability for Appointments**

| Doctors set or update their availability for patient appointments. The doctor's availability is updated, and patients can see the new slots when scheduling appointments. |
| --- |

**5.3.5 Accept or Decline Appointment Requests**

| Doctor accepts or declines an appointment request from a patient. 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. |
| --- |

**5.3.6 View Upcoming Appointments**

| Doctors may view all ‘Confirmed’ appointment details. |
| --- |

**5.3.7 Record Appointment Outcome**

| Doctors may record the outcome of a completed appointment, based on a selection of ‘CONFIRMED’ appointments. Appointment ID and appointment date can be transferred over without having to manually do so. |
| --- |

**5.4 Pharmacist Actions**

**5.4.1 View Appointment Outcome Record**

| When a pharmacist logs in the view the following menu, where they can choose from 6 domains. Here, they may view appointment outcome records to process prescriptions. The system displays the appointment outcome details, including prescribed medications |
| --- |

**5.4.2 Update Prescription Status**

| The pharmacist may update the status of a prescription to "dispensed." The prescription status is updated, and the change is reflected in the appointment outcome records. |
| --- |

**5.4.3 View Medication Inventory**

| Pharmacists may view the current medication inventory. The system displays a list of medications, including stock levels. |
| --- |

**5.4.4 Submit Replenishment Request**

| Pharmacists may view the current medication inventory including the list of medications and stock levels. By viewing replenishment requests we can verify the request has been submitted. |
| --- |

**5.5 Administrator Actions**

**5.5.1 View & Manage Hospital Staff**

| When an Administrator logs in the view the following menu, where they can choose from 7 domains. |
| --- |
| They may view the list of hospital staff, add a new staff member, update staff details and remove staff members. This change is verified when we view the staff list again below: |

**5.5.2 View Appointment Details**

| Administrators may view all appointments. The system displays a list of appointments including details like Patient ID, Doctor ID, status, and date/time |
| --- |

**5.5.3 View and Manage Medication Inventory**

| Administrators may update the stock level of a medication. By viewing the Medicine Inventory once again, we can verify that the medication's stock level is updated in the inventory. |
| --- |

**5.5.4 Approve Replenishment Requests**

| Administrator can approve the replenishment request from a pharmacist. By viewing the requests again, we verify that the request status changes to "approved," and the medication inventory is updated accordingly. |
| --- |