

**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 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 (Matriculation Number)** | **Course** | **Lab Group** | **Signature /Date** |
| AUNG AUNG PYAE PHYO (U2320039C) | SC2002 | SCEA | Aung 19/11/2024 |
| GUDA CHAAITRA JOSEPH (U2323304K) | SC2002 | SCEA | Chaaitra 19/11/2024 |
| LI JIALING (U2231890D) | SC2002 | SCEA | Jialing 19/11/2024 |
| PATEL DHAIRYA NAYANBHAI (U2221896A) | SC2002 | SCEA | Dhairya 19/11/2024 |
| YUAN HAOER (U2310565C) | SC2002 | SCEA | Haoer 19/11/2024 |

# **Table of Contents**

# 

1. **Design Considerations**

**1.1 Approach**

**1.2 Assumptions**

**1.3.1 SOLID Design Principles**

**1.3.2 Additional Design Principles**

**1.4 Object-Oriented Programming Principles**

**1.5 Additional Features**

1. **UML Class Diagram**

**2.1 Explanation**

1. **Test Case Demonstration**

**3.1 Patient Test Cases**

**3.2 Doctor Test Cases**

**3.3 Pharmacist Test Cases**

**3.4 Administrator Test Cases**

**3.5 System Test Cases**

**3.6 Other Test Cases**

1. **Reflection**
2. **Work Breakdown and Contributions**
3. **References**

## **1. Design Considerations**

**1.1 Approach**

The Hospital Management System (HMS) is designed to automate the management of hospital functions, such as patient administration, scheduling appointments, managing staff, and controlling medication inventory. The application aims to improve the efficient use of hospital resources, enhance the quality of patient care, and simplify administrative tasks. The system is separated into packages and subpackages such as boundary, controller, enums, entity, interfaces, services, repository, and utils. We have also employed several design patterns to achieve low coupling and high cohesion in the system.

**1.2 Assumptions**

The following assumptions are made while designing the system:

1. There will be at least one administrator account at any point in time.

**1.3. Design Principles**

**1.3.1 SOLID Design Principles**

The SOLID design principles consist of the following:

* **Single responsibility principle:**

This principle states that "A class should have one and only one reason to change", that is, each class should have a singular responsibility. This is observed in the Medicine class, which handles only medicine-related operations and focuses on inventory management with methods serving specific purposes for the state of medicines.

* **Open-closed principle:**

This principle states that “Entities (such as classes, modules, functions, etc.) should be open to extension, while being closed to modification”, that is, one should be able to extend a class’s functionality without altering it. This is observed in the User class, wherein new user types can be added without changing existing code, each implementing specific behavior.

* **Liskov substitution principle:**

This principle states that “Child or sub classes must be substitutable for their parent or base classes“, that is, any class that is the child of a parent class should be usable in place of its parent without any unexpected behaviour. This is observed in the Repository interface, wherein all repository implementations are interchangeable and derived classes hold the base contract.

* **Interface segregation principle:**

This principle states that “No client should be forced to depend on interfaces it does not use”, and is observed in the Controller interfaces in controller.interfaces package, wherein services have focused interfaces, controllers implement relevant ones, and clients depend only on what they need.

* **Dependency inversion principle:**

This principle states that “High-level modules should not depend on low-level modules, and that both should depend on abstractions”, that is, the major functionalities should not be influenced by the details of the implementation. It is primarily observed in DoctorUI class, wherein dependencies are injected via the constructor, which enhances testability and reduce direct dependencies between components.

**1.3.2 Additional Design Principles**

Furthermore, we made use of the following design patterns in our management system:

* **Singleton Pattern:**

This pattern ensures that a class has only one instance, while providing a global access point to this instance. This is employed in the repositories to ensure a single instance of data access.

* **Model-View-Controller (MVC) Pattern:**

Through this pattern, the entire application is split into three parts:   
i. ‘Model’ handles data (i.e. Entity classes like Appointment, Medicine);   
ii. ‘View’ is what users see (i.e. UI classes like DoctorUI, PharmacistUI);   
iii. ‘Controller’ connects the above two (i.e. Controller classes that implement service interfaces)

* **Repository Pattern:**

This pattern serves as a structured way to abstract data access, by providing a collection-like interface to access domain objects.

* **Factory Pattern:**

Using this pattern, construction of objects is done without having to define the exact class of object to be created. This is used in DataImportManager for creating different types of entities.

**1.4 Object-Oriented Programming Principles**

The 4 major principles of Object-Oriented Programming are:

* **Encapsulation:**

Encapsulation refers to building a “barrier” to protect the private data of an object, with access only provided through public get and set methods in the class of the object. This is implemented in our entities, with all data being made private and only modifiable through public set methods.

* **Abstraction:**

Abstraction refers to hiding unnecessary information and displaying only necessary information to the users interacting. This is implemented through our interfaces, where users can only access relevant functions based on their roles, increasing system security and simplicity.

* **Polymorphism:**

Polymorphism means many forms. It refers to the ability of an object reference being referred to different types; knowing which method to apply depends on where it is in the inheritance hierarchy.

* **Inheritance:**

Inheritance is an essential feature that allows for new classes to be derived from existing classes by inheriting their attributes and behaviours while adding new capabilities to the new classes. This was implemented through a base user class, with other roles inheriting from the user class.

**1.5 Additional Features**

* **Administrator Role:**

We added more functionalities to the administrator role, allowing administrators to filter through appointments (by date, doctor, etc.) when viewing for easier accessibility and clarity. Additionally, they can view inventory actions, which displays history logs of inventory actions.

* **Doctor Role:**

We implemented further encapsulation to the doctor role, specifically limiting to the most relevant options to the user. For example, the doctor can only view pending appointments or prescribe medication that is only currently available in the inventory rather than all appointments, including accepted ones, and unavailable medicine. This allows for more clarity and simplicity between the user (doctors) and the system.

**2. UML Class Diagram**

**2.1 Explanation**

Separating packages and subpackages as shown in our UML class diagram below such as boundary, controller, enums, entity, interfaces, services, repository, and utils promotes modularity, readability, maintainability, and scalability of the code. Each package adheres to Single Responsibility Principle and Encapsulation of information, which helps to keep the code organized and manageable.

## 

The jpg of the UML Class Diagram is attached along with this report.

## **3. Test Case Demonstration**

**3.1 Patients Test Cases**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Expected Outcome** | **Result** |
| 1. Update personal information | Users can update their contact information (phone number and email)  **[Exception Handling]**  Phone numbers must be digits only. |  |
| 2. Schedule an appointment | Patient can select an availability slot for an appointment with the doctor.  **[Exception Handling]**  Slot numbers entered out of range will be rejected and date format must be in YYYY-MM-DD. |  |
| 3. View scheduled appointments | Patients can see their scheduled appointments after doctors accept their request. Status is changed from pending to confirmed once the appointment request is accepted. |  |

**3.2 Doctor Test Cases**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Expected Outcome** | **Result** |
| 4. Update patient medical records | Doctors can update the patient’s medical record after administering diagnosis.  **[Exception Handling]**  Patient’s ID must exist. |  |
| 5. View personal schedule | Doctors can view personal schedule which includes availabilities and records of appointments made |  |
| 6. Set availability for appointments | Doctors are able to set appointment availability for patients to book.  **[Exception Handling]**  The availability date must be entered within 7 days from that day, time must be in 24-hour format and end time must be after start time. |  |
| 7. Accept/ Decline appointment requests | Doctors can accept or decline appointment requests from patients.  **[Exception Handling]**  Out of range selection will be rejected |  |

**3.3 Pharmacist Test Cases**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Expected Outcome** | **Result** |
| 8. View appointment outcome record | Pharmacists can see appointment records of patients with their doctors. The details of the prescriptions will be used to administer medicines for the patients accordingly. |  |
| 9. Update prescription status | To update prescriptions status, pharmacists need the appointment ID to select appointments and update if the medications have been given.  **[Exception Handling]**  Invalid appointmentID and out of range selections will be rejected. |  |
| 10. Submit replenishment request | If stocks are critically low (i.e. below the low stock alert level), pharmacists can submit the replenishment request which will be sent to the administrator.  **[Exception Handling]**  Only low stocks will be shown. Out of range selection will be rejected. |  |

**3.4 Administrator Test Cases**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Expected Outcome** | **Result** |
| 11. View and manage hospital staff | Administrators can do the following:   * View their staff, categorized by roles * Add new staff members * Update staff member * Remove staff   **[Exception Handling]**  If a staff member is not found, an error will be displayed. |  |
| 12. View appointment details | Administrators can view appointment details with their statuses. The detailed records can be sorted by the dates, doctor or status.  **[Exception Handling]**  Out of range selection will be rejected. |  |
| 13. View and manage medication inventory | Administrators can view inventory, update stock level, update low stock alert level and view inventory actions history log.  **[Exception Handling]**  Out of range selection will be rejected. |  |
| 14. Approve replenishment requests | Administrators can approve or deny replenishment requests sent by pharmacists.  **[Exception Handling]**  Incorrect or out of range selection will be rejected. |  |

**3.5 System Test Cases**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Expected Outcome** | **Results** |
| 15. First time login and password change | Users will be prompted to change password upon first login which must be at least 8 characters.  **[Exception Handling]**  Upon entering a new password that is less than 8 characters, an error message will be displayed and the user needs to login again. |  |
| 16. Login with incorrect credentials | Users must enter the correct password or username to login to their domain.  **[Exception Handling]**  If the password is incorrect/ username doesn't exist, the new password equals the old password or incorrect entry 2 times, an error message will be displayed. |  |

**3.6 Other Test Cases**

For test cases that are not found in the report can be accessed here.



## **4. Reflection**

**Difficulties Encountered and How They Were Overcame**

Initially, despite having practices in the Lab where smaller “applications” were made, we lacked experience in applying the programming principles learnt in the course to a larger scale, more complex system such as this hospital assignment. This made the assignment slightly more challenging, especially since we had to design all classes ourselves based on SOLID principles without a predefined template. As such, we reviewed the course material numerous times to strengthen our understanding of OOP and SOLID principles, referred to the Lab practices in the design process to get a better outline and researched online for additional design principles and examples to enhance our design.While we were clear about each role, it was challenging to visualize the structure of the relationships in the final program. Therefore, we decided to tackle the UML diagram first to visualize the system and refine the design iteratively. This provided a starting point to our coding process, as it gave us clarity on the essentials and how every entity was connected.

**Knowledge Learned from the Course**

As mentioned above, we were able to gain a deeper understanding of the core OOP principles of encapsulation, inheritance, polymorphism and data abstraction in Java. Through practice, we also gained practical knowledge on implementing SOLID principles and learned of additional design principles to ensure a separation of concerns and a modular design, making our application scalable and maintainable.

**Further Improvement Suggestions**

We believe an important “next step” would have been to create a graphical user interface (GUI) to allow for clearer communication and improved interaction between users and the application.

**Insights on Good Design and Implementation Practices**

Some insights we gained on good design were firstly, iteratively developing our application. Despite having already worked out the UML diagram, we tested each part before continuing, rather than coding everything at once. This allowed for better debugging during errors and refinement when necessary. Next, as we are working in a group setting, when passing code to one another, we ensured that changes or edits were clear through clean, well-commented code, enhancing efficiency as changes were easy to identify. Another insight we gained was in regards to code reusability and extensibility, by adhering to SOLID principles. Keeping this in mind, we created a generic user class that can be reused for all roles in the application. We also made separate classes such as Medicine and MedicineInventoryAction, to enhance adaptability for future functionalities in the application, promoting extensibility.

**Experience Gained from the Assignment**

Lastly, through the real-life application of OOP concepts, our key takeaway has to be the hands-on experience in creating a complex system. Working on a project simulating a real-world scenario like this hospital management assignment highlighted the importance of systematic design, modularity, and the SOLID principles. We also improved our understanding of Java language throughout the process.

## **5. Work Allocation and Breakdown**

|  |  |
| --- | --- |
| **Group Members** | **Work Allocation** |
| AUNG AUNG PYAE PHYO | - UML Diagram  - Implementation of Doctor and related classes  - Test cases  - Updating report  - Demo script  - Generating Javadoc |
| GUDA CHAAITRA JOSEPH | - UML Diagram  - Implementation of Administrator and related classes  - Updating report  - Demo script |
| ADM LI JIALING | - UML Diagram  - Implementation of Patient and related classes  - Updating report |
| PATEL DHAIRYA NAYANBHAI | - UML Diagram  - Implementation of Pharmacist and related classes  - Merging and organization of source code  - Debugging and fixing of source code |
| YUAN HAOER | - UML Diagram  - Implementation of User  - Updating report |

## **6. References:**

[1] GeeksforGeeks. (2024, Oct. 11). *SOLID principles in programming: Understand with real life examples* [Online]. Available: <https://www.geeksforgeeks.org/solid-principle-in-programming-understand-with-real-life-examples/>

[2] GeeksforGeeks. (2024, Oct. 11). *MVC design pattern* [Online]. Available: <https://www.geeksforgeeks.org/mvc-design-pattern/>

[3] GeeksforGeeks. (2024, Nov. 1). *Repository design pattern* [Online]. Available: <https://www.geeksforgeeks.org/repository-design-pattern/>

[4] GeeksforGeeks. (2024, Oct. 15). *Software design patterns tutorial* [Online]. Available: <https://www.geeksforgeeks.org/software-design-patterns/>

Link to our github repository: <https://github.com/AAPP02/SC2002Group6>