HOSPITAL APPOINTMENT MANAGEMENT SYSTEM

Software Engineering Tools and Practices

JANUARY 20, 2026

ARBAMINCH UNIVERSITY

(AMU)

ARBA MINCH UNIVERSITY

INSTITUTE OF TECHNOLOGY (AMIT)

DEPARTMENT OF SOFTWARE ENGINEERING

Course title - Software Engineering Tools And Practices

Project Topic – Hospital Appointment Management System

Group members	**ID number**
1.Abenezer Tekalign	NSR/1115/16
2.Nathan Kaleb	NSR/T/011/16
3.Kaleb Hailu	NSR/531/16
4.Abenezer Amenu	NSR/T/006/16
5.Bernabas Derese	NSR/166/16

Submission-Date January 16, 2026

Submitted To: Mr. Yohannes Miniyilu

TABLE OF CONTENTS Pages

Introduction

Hospitals operate in an environment where efficiency, accuracy, and coordination are critical. One of the most essential operational processes in a hospital is the management of patient appointments. Appointment scheduling directly affects patient satisfaction, doctor workload, nurse coordination, and administrative efficiency. In many healthcare institutions, appointment management is still handled manually or through loosely integrated systems, resulting in frequent scheduling conflicts, long waiting times, poor record management, and increased pressure on hospital staff.

The Hospital Appointment Management System is proposed as a structured software solution that addresses these challenges through proper system analysis, object-oriented design, and UML-based modeling. The system provides a centralized approach for handling appointment-related activities, ensuring that patients, doctors, nurses, and administrators interact through clearly defined workflows. Rather than focusing on deployment, the project emphasizes design correctness, clarity, and completeness, which are fundamental goals in software engineering education.

This project demonstrates the application of software engineering principles learned in class, including requirement engineering, object-oriented analysis, system modeling, and documentation. By using Java as the conceptual implementation language and UML diagrams as the primary design tools, the system presents a realistic and well-organized representation of how a hospital appointment system can be designed in practice.

Problem Statement

The absence of a unified and well-structured appointment management system creates significant operational challenges in hospitals. Patients often struggle to book appointments with preferred doctors due to limited visibility into doctor availability. Manual scheduling methods increase the likelihood of double bookings, missed appointments, and inefficient use of medical staff time. These problems negatively impact both patient experience and healthcare service delivery.

Healthcare professionals also face difficulties due to fragmented systems. Doctors may not have a clear overview of their daily schedules, nurses may rely on paper-based or informal methods to record patient vitals, and administrators may lack centralized control over users and hospital departments. These issues result in data inconsistency, increased workload, and reduced operational transparency.

The Hospital Appointment Management System is designed to solve these problems by providing a logically organized, object-oriented solution that models real hospital workflows. The system ensures that appointment scheduling, doctor preferences, patient data handling, and administrative control are all managed within a single, coherent framework.

Project Objectives

The primary objective of this project is to design a Hospital Appointment Management System that streamlines appointment scheduling while ensuring clarity, accuracy, and proper role separation. The system aims to allow patients to register, authenticate, view available doctors, select doctor preferences based on specialization and availability, and book appointments efficiently.

Another major objective is to support healthcare staff in managing their responsibilities effectively. Doctors are provided with mechanisms to manage schedules, approve or reject appointment requests, and access patient-related information. Nurses are supported through structured processes for recording and updating patient vitals, ensuring that patient health data is accurate and up to date.

From an administrative perspective, the system aims to provide full control over user management, roles, permissions, and hospital departments. By achieving these objectives, the system reduces manual workload, minimizes errors, improves coordination among stakeholders, and enhances overall hospital workflow efficiency.

Scope of the System

The scope of the Hospital Appointment Management System is limited to appointment-related and administrative operations within a hospital environment. The system includes functionalities such as patient registration and authentication, appointment booking, doctor schedule management, nurse vital recording, and administrative user management.

The system does not include clinical diagnosis, billing, pharmacy management, or laboratory services. Additionally, since this is an academic project, system deployment, live database integration, and real-time notifications are outside the project scope. However, the system design is flexible enough to support future extensions if required.

Software Development Methodology

The project follows a **Waterfall-based** Object-Oriented Software Development methodology, which is appropriate for systems with clearly defined and stable requirements. The methodology consists of requirement analysis, system design, modeling, and conceptual implementation. Each phase is completed systematically before moving to the next, ensuring consistency and traceability.

Object-oriented principles form the foundation of the system design. Encapsulation is used to protect data integrity, inheritance is applied to promote reuse, and abstraction helps manage system complexity. UML diagrams are used extensively to validate both structural and behavioral aspects of the system.

This methodology was selected because it supports thorough documentation, clear evaluation criteria, and strong alignment with software engineering academic standards.

Tools and Technologies Used

Java is used as the conceptual implementation language due to its strong support for object-oriented programming, robustness, and widespread adoption in enterprise systems. Java’s class-based structure aligns well with UML modeling and allows clear mapping between design and implementation.

Modelio and Draw.io are used for UML diagram creation. These tools were selected because they provide comprehensive UML support, ease of modification, and professional-quality outputs suitable for academic submission. They enable clear visualization of system interactions, structures, and workflows.

No specialized hardware is required since the system is not deployed. Standard personal computers are sufficient for design and modeling activities.

Project Initiation and System Understanding

The project began with a detailed system understanding phase based on requirement engineering principles. This involved analyzing hospital appointment workflows, identifying stakeholders, and defining system boundaries. The goal was to ensure that the system design reflects real-world hospital operations.

System structure and behavior were identified using object-oriented analysis techniques. Core entities such as Patient, Doctor, Nurse, Admin, Appointment, and Booking were derived from the problem domain. Their responsibilities and interactions were carefully analyzed before design modeling began.

Object-Oriented Analysis and Design

Object-oriented analysis was used to identify system classes, attributes, methods, and relationships. Each class represents a real-world entity with a specific responsibility. This approach improves clarity, modularity, and maintainability.

Relationships such as associations and inheritance are used to represent interactions between entities. For example, patients are associated with appointments, doctors manage schedules, and administrators oversee system users. This structured design ensures logical consistency across the system.

UML Diagrams and Detailed Explanation

UML diagrams are central to validating the system design. Each diagram represents a different perspective of the system and collectively ensures correctness and completeness.

I. Use Case Diagram

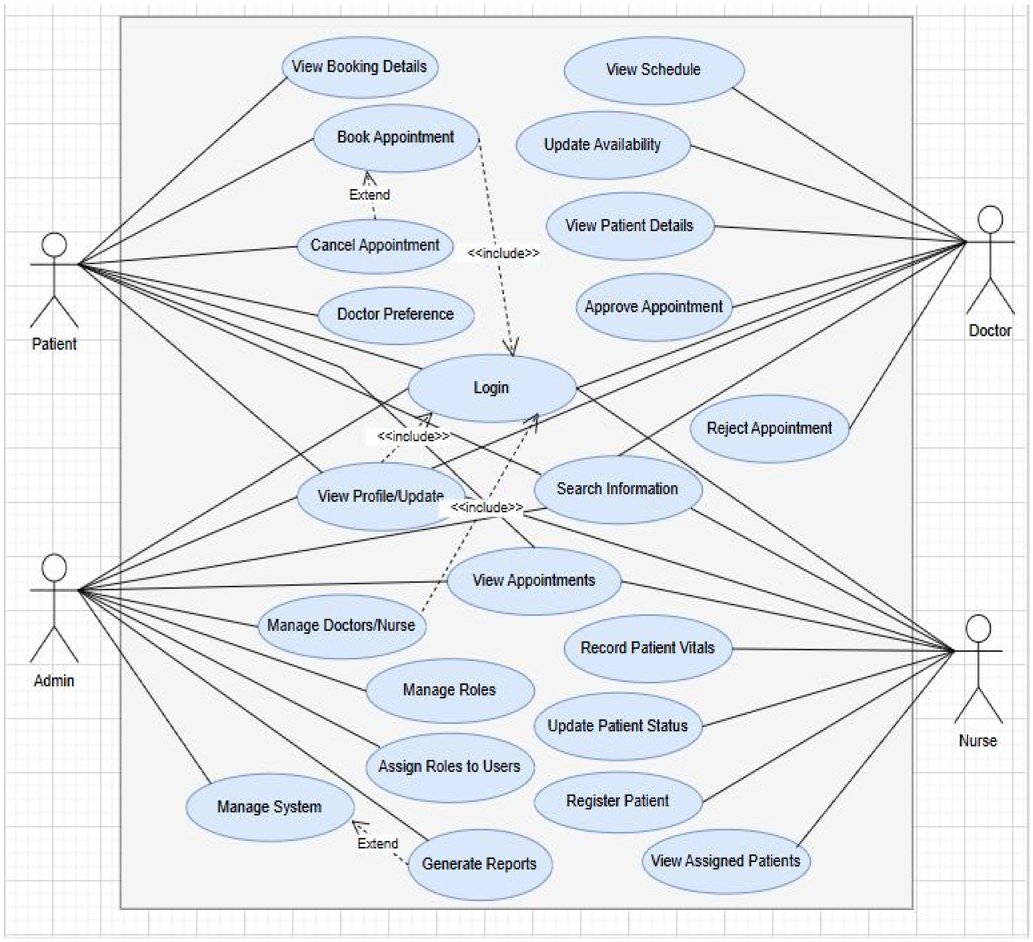

Figure 1: Use Case Diagram of Hospital Appointment Management System

The **use case diagram** illustrates the functional requirements of the Hospital Appointment Management System from the perspective of its users. It defines the system boundary and shows how different actors interact with the system to achieve specific goals. The diagram identifies four main actors: Patient, Doctor, Nurse, and Administrator.

- The **Patient** actor interacts with the system to manage appointment-related activities. Patients can log into the system, view and update their profiles, search for information, select doctor preferences, book appointments, cancel appointments, and view booking details. The use case "Doctor Preference" allows patients to choose doctors based on specialization or availability, enhancing user convenience and flexibility.
- The **Doctor** actor uses the system to manage medical schedules and patient interactions. Doctors can view their schedules, update availability, view patient details, approve appointment requests, and reject appointments when necessary. These use cases ensure that doctors maintain control over their workload and consultation schedules.
- The **Nurse** actor supports patient care activities within the system. Nurses can view assigned patients, record patient vitals, and update patient status. These use cases ensure accurate documentation of patient health information and smooth coordination with doctors during consultations.
- The **Administrator** actor is responsible for overall system management. Administrators can manage doctors and nurses, assign roles to users, manage system operations, generate reports, and oversee system functionality. The "Generate Reports" use case extends the "Manage System" use case, indicating that report generation is an optional administrative function.

The Login use case is a common functionality shared by all actors and is included in multiple use cases using the <<include>> relationship. This indicates that authentication is mandatory before accessing protected system features. The use of <<Extend>> relationships represents optional or conditional behaviors, such as appointment cancellation and report generation.

Overall, the use case diagram provides a clear and organized view of system functionality, actor responsibilities, and interaction flow. It ensures that all functional requirements of the Hospital Appointment Management System are clearly defined and aligned with real-world hospital operations.

II. Class Diagram

The class diagram represents the static structure of the Hospital Appointment Management System by showing the main classes, their attributes, methods, and relationships. It is designed using object-oriented principles to reflect real-world hospital operations in a clear and modular manner.

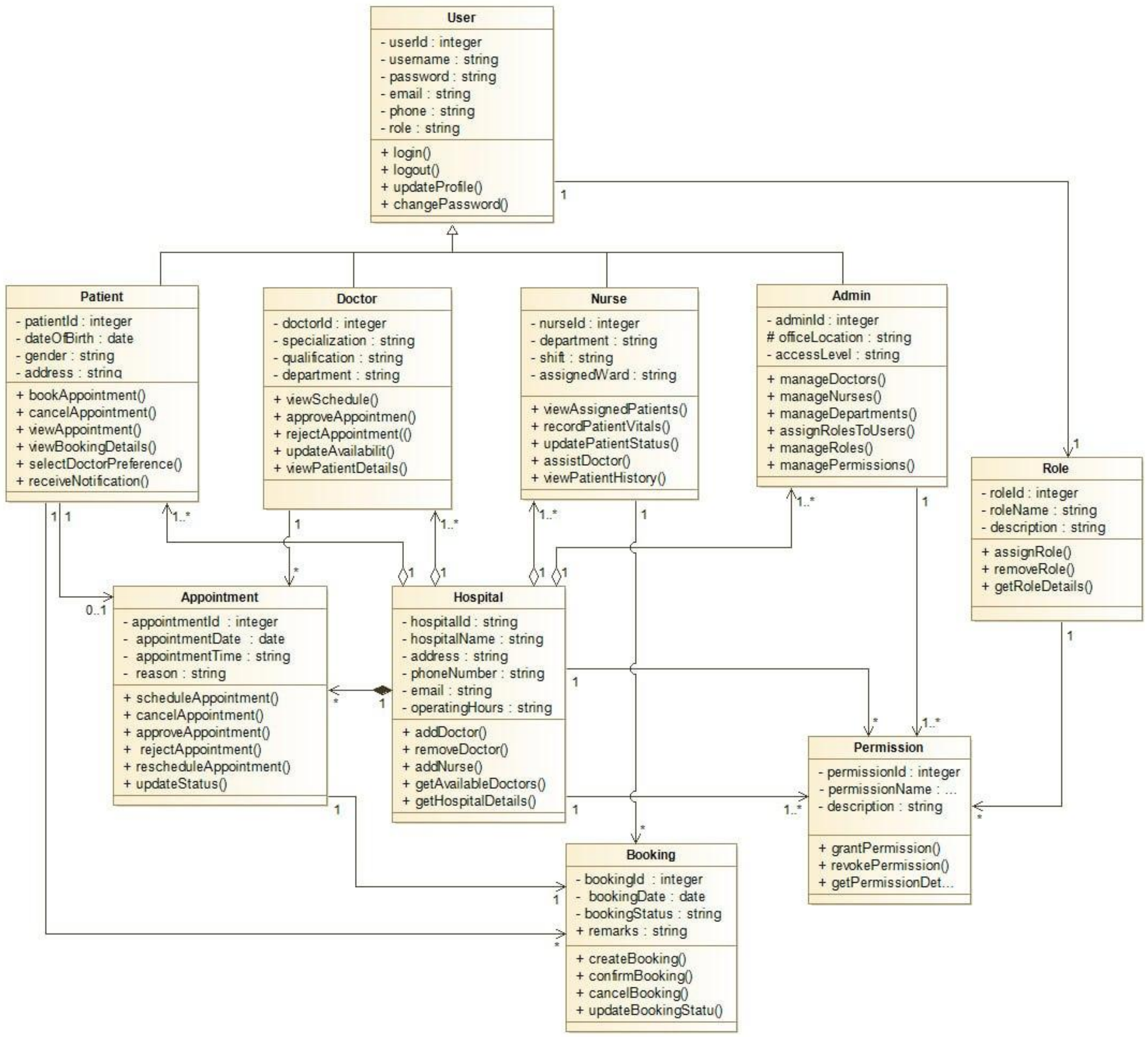

Figure 2: Hospital Appointment Management System – Class Diagram using **Modelio**

User and Inheritance Structure

At the top of the hierarchy is the User class, which serves as a generalized superclass for system users. It contains common attributes such as user_id, user_role_id, user_name, user_email, and user_address, along with basic operations like adding, editing, deleting, and searching users. This design promotes code reuse and avoids duplication.

The Patient, Doctor, Nurse, and Admin classes inherit from the User class. This inheritance relationship reflects the real-world concept that all these roles are system users but with specialized responsibilities.

Patient Class

The Patient class stores patient-specific information such as patient ID, gender, and date of birth. It provides operations for adding, editing, and viewing patient details. Patients are associated with appointments and bookings, representing their ability to request and manage hospital visits. This class is central to the appointment scheduling process.

Doctor Class

The Doctor class represents medical professionals in the system. It includes attributes such as doctor ID, specialization, and availability. The methods allow doctors to update their availability and view schedules. Doctors are associated with appointments and bookings, emphasizing their role in approving or rejecting appointment requests and managing consultation schedules.

Nurse Class

The Nurse class supports clinical assistance activities. It includes nurse-specific identifiers and provides methods such as recording patient vitals and updating patient status. This class interacts indirectly with appointments and patients, ensuring that patient health information is properly maintained during the care process.

Admin Class

The Admin class is responsible for overall system administration. It includes operations for managing doctors, nurses, appointments, departments, and generating reports. The administrator also has access to system logs, ensuring proper governance and monitoring of system activities. This class plays a crucial role in system control and maintenance.

Hospital Class

The Hospital class represents the organizational entity. It contains attributes such as hospital ID, name, type, and description. The hospital class is associated with users, appointments, and bookings, serving as a central context within which all operations occur. Methods are provided to add, edit, delete, and search hospital records.

Appointment Class

The Appointment class manages appointment-related data, including appointment ID, date, time, reason, and status. It includes methods to schedule, reschedule, cancel, approve, or reject appointments. This class plays a critical role in coordinating interactions between patients and doctors.

Booking Class

The Booking class represents the confirmed reservation linking patients, doctors, and appointments. It includes booking date and time and references related patient, doctor, and appointment objects. Methods allow bookings to be created, updated, deleted, and viewed, ensuring structured management of appointment confirmations.

Role and Permission Classes

The Role class defines different system roles and their descriptions. It provides methods for role creation, editing, deletion, and assignment to users. The Permission class defines access rights associated with roles, specifying modules and permissions. Together, these classes support role-based access control, ensuring secure and controlled system usage.

Relationships and Design Justification

- Inheritance is used between User and its subclasses to promote reuse and clarity.
- Associations represent real-world interactions, such as patients booking appointments and doctors managing schedules.
- Encapsulation ensures that data and behavior are grouped within appropriate classes.
- The diagram ensures high cohesion and low coupling, making the system easier to understand and extend.

Overall, this class diagram provides a solid blueprint for the Hospital Appointment Management System. It clearly defines system entities, their responsibilities, and their interactions, ensuring that the system design is logically consistent, scalable, and aligned with object-oriented software engineering principles.

III. Sequence Diagram

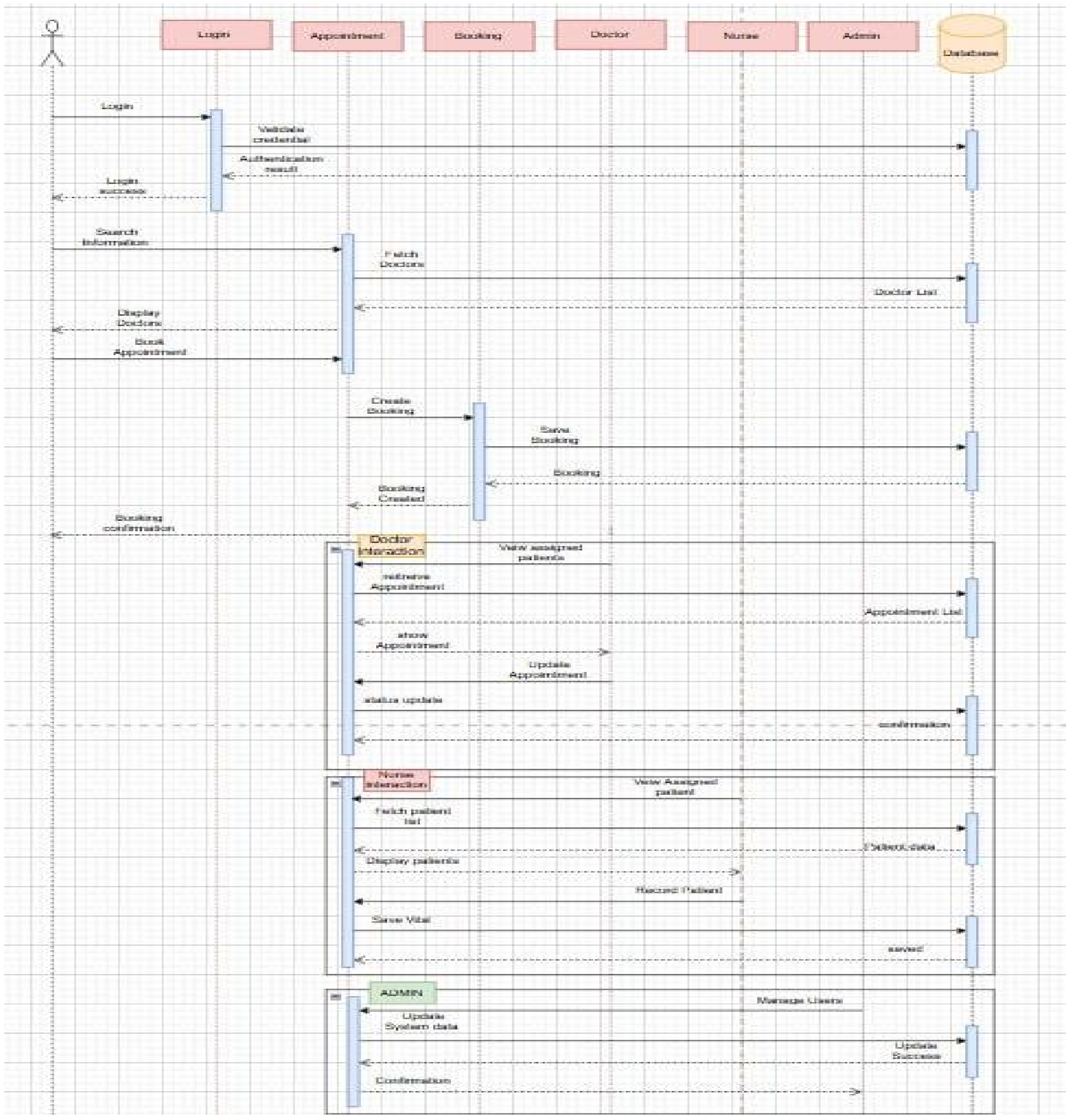

Figure 3: Sequence Diagram for Appointment Booking Process using Draw io

The sequence diagram illustrates the dynamic interaction between objects during the appointment booking process. It shows the order of message exchanges between the patient, system components, and doctor.

The diagram begins with the patient initiating an appointment request. The system checks doctor availability, processes the booking request, and updates appointment status. The doctor then approves or rejects the appointment, and the system notifies the patient. This diagram validates the logical flow of operations and ensures correct interaction sequencing.

1. The Login Phase

An actor (the user) starts by entering their details.

- ✧ The **Login** page asks the **Database** if the username and password are correct.
- ✧ If the database says "Yes," the user is logged in successfully.

2. Booking an Appointment

The user wants to see a doctor.

- ✧ **Search:** The user searches for info. The **Appointment** system pulls a list of doctors from the database and shows them to the user.
- ✧ **Booking:** The user picks a doctor and hits "Book." The **Booking** system saves this information in the database and sends a confirmation back to the user.

3. Staff Interactions (The Boxes)

The lower half of the diagram shows what happens once the staff gets involved. It is divided into three main roles:

The Doctor

- ✧ The doctor checks their assigned patients.
- ✧ The system pulls the appointment list from the database.
- ✧ The doctor can update the appointment (e.g., marking it as "completed") and the database saves that change.

The Nurse

- ✧ The nurse views the patient list.
- ✧ They record "vitals" (like blood pressure or temperature).
- ✧ These details are sent to the database to be saved in the patient's file.

The Admin

- ✧ The admin manages the overall users of the system.
- ✧ They can update system data (like adding new doctors or changing schedules), which is then updated in the database.

IV. Activity Diagram

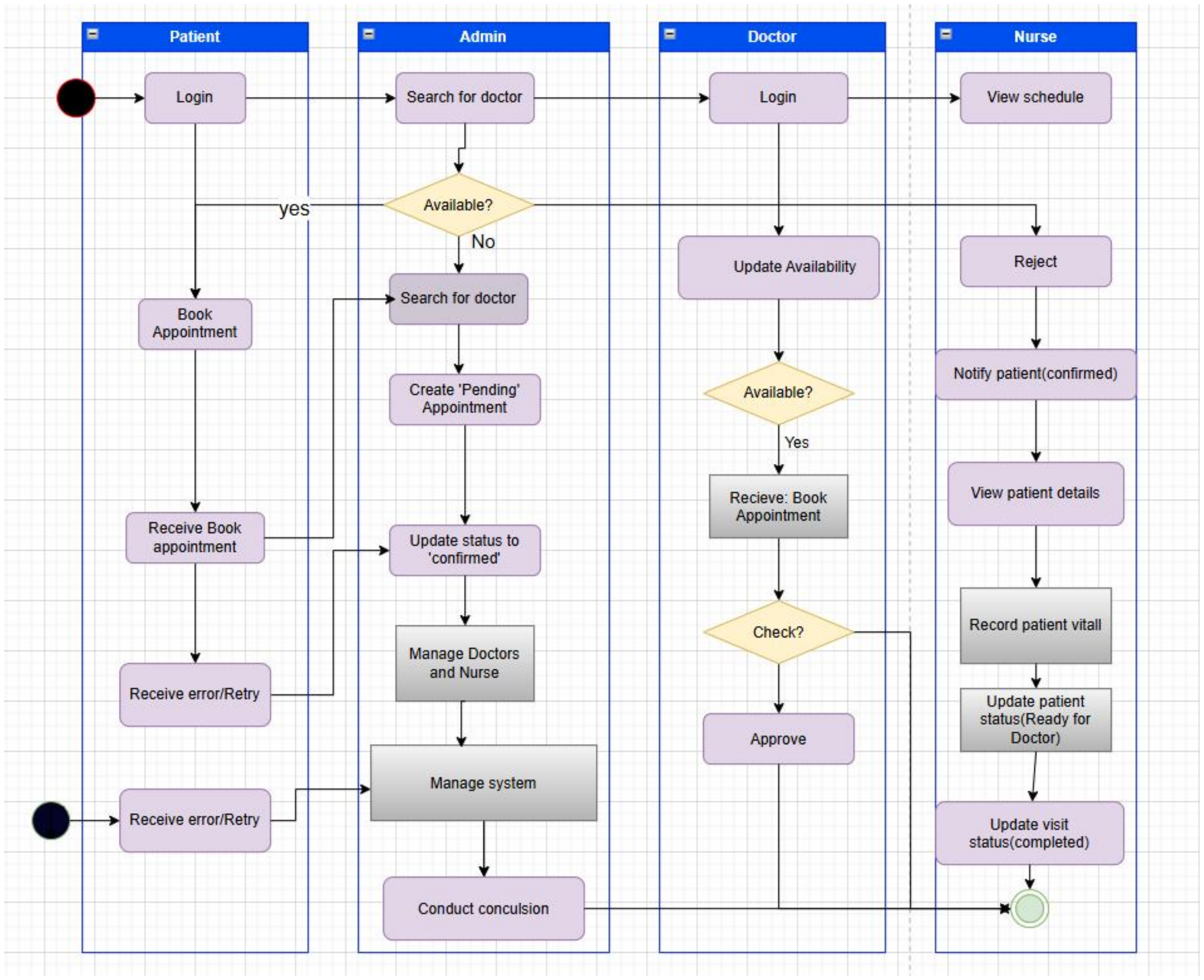

Figure 4: Activity Diagram for Hospital Appointment Workflow

The **activity diagram** illustrates the workflow of the Hospital Appointment Management System by showing how different actors—Patient, Admin, Doctor, and Nurse—interact during the appointment lifecycle.

The process starts when the **Patient logs into the system** and requests to **book an appointment**. The request is forwarded to the **Admin**, who **searches for a doctor** and checks the doctor's availability. If a doctor is **not available**, the system repeats the search process. When a doctor is available, the Admin **creates a pending appointment** and updates its status to **confirmed**.

The **Doctor logs in**, updates availability, and **receives the booking request**. After reviewing the appointment details, the Doctor performs a check and either **approves** or **rejects** the appointment. Approved appointments move forward for medical handling.

The **Nurse views the schedule**, accesses **patient details**, and **records patient vitals**. The patient's status is updated to **ready for doctor**, allowing the consultation to proceed. After the consultation, the Nurse updates the **visit status to completed**.

If any error occurs during booking or processing, the system directs the Patient to a **retry or error-handling flow**, ensuring reliability. The activity ends once the visit is successfully completed or the issue is resolved

Design Validation Through Scenario-Based Analysis

As the Hospital Appointment Management System is an academic project that is not deployed, design validation is conducted through scenario-based analysis rather than execution-based testing. This approach evaluates whether the system design correctly satisfies functional requirements and accurately represents real hospital workflows. By analyzing typical system usage scenarios and tracing them through UML diagrams and object-oriented models, the logical correctness and completeness of the design are validated.

One primary validation scenario is the patient appointment booking process. In this scenario, a patient registers, logs into the system, views available doctors, selects a preferred doctor, and submits an appointment request. The system checks doctor availability, creates an appointment record, and updates the appointment status. This scenario confirms that the system correctly models patient interactions and that class relationships and message flows are logically consistent.

Another important scenario involves doctor appointment approval and schedule management. The doctor reviews pending appointment requests and either approves or rejects them based on availability. The system then updates the appointment status and reflects changes in the doctor's schedule. This scenario validates role responsibilities and ensures that decision-making processes are clearly represented in the system design.

A final validation scenario focuses on nurse interaction and administrative control. Nurses record and update patient vitals, while administrators manage users, roles, and hospital departments. These scenarios confirm proper role separation, access control, and coordination among system components. Overall, scenario-based analysis demonstrates that the system design is coherent, complete, and aligned with its stated objectives, even without deployment.

Code Generation Based on the System Models

After completing the system modeling phase, initial Java source code was generated from the UML class diagram developed in the design stage. This step demonstrates the practical transition from **conceptual system design** to **object-oriented software structure**, ensuring consistency between the design and implementation.

1. Purpose of Code Generation

The main objective of this phase was to transform the **structural design** of the system into Java code. Rather than implementing full functionality, the focus was on:

- ✓ Preserving the system architecture defined in the UML diagrams
- ✓ Ensuring that classes, relationships, and responsibilities are correctly reflected in code
- ✓ Providing a clean foundation for future implementation and extension

2. Use of the UML Class Diagram

The UML class diagram served as the **primary blueprint** for code generation. Each modeling element was mapped directly to Java constructs as follows:

- ✓ **UML Classes → Java Classes**
  Every class in the diagram (e.g., User, Patient, Doctor, Appointment) was converted into a corresponding Java class.
- ✓ **Attributes → Instance Variables**
  UML attributes were translated into Java fields with appropriate data types.
- ✓ **Operations → Method Signatures**
  UML operations were converted into method declarations without implementation logic, acting as placeholders for future development.
- ✓ **Inheritance →** extends **Keyword**
  Generalization relationships (e.g., Patient, Doctor, Nurse inheriting from User) were implemented using Java's inheritance mechanism.
- ✓ **Associations → Object References**
  Relationships between classes were represented using object references or collections, reflecting the associations defined in the diagram.

3. Scope of the Generated Code

The generated Java code represents **only the structural aspect of the system**, meaning:

- ❖ No business logic is implemented
- ❖ No database connectivity or persistence logic is included
- ❖ No user interface or deployment configuration is provided

This approach ensures that the generated code remains **clean, readable, and aligned with the UML design**, making it suitable for educational and architectural demonstration purposes.

4. Benefits of Model-Based Code Generation

Generating code directly from UML models provides several advantages:

- ❖ Reduces inconsistencies between design and implementation
- ❖ Improves maintainability by enforcing a clear structure
- ❖ Accelerates development by providing a ready-made project skeleton
- ❖ Enhances understanding of how UML models map to real programming constructs

5. Outcome of This Phase

As a result of this phase, a complete set of Java class files was produced, accurately reflecting the system's class diagram. These files serve as a **starting point** for implementing business logic, validations, and system behavior in subsequent development stages.

User Class

The User class is the superclass that represents all system users. It contains common attributes and operations shared by patients, doctors, nurses, and administrators.

```
public class User {
    protected int userId;
    protected String username;
    protected String password;
    protected String email;
    protected String phone;
    protected String role;

    public void login() {
    }

    public void logout() {
    }
    public void updateProfile() {
    }

    public void changePassword() {
    }
}
```

Patient Class

The Patient class represents patients in the system. It extends the User class and includes patient-specific information and operations related to patient management.

```
public class Patient extends User {
    private int patientId;
    private String dateOfBirth;
    private String gender;
    private String address;

    public void bookAppointment() {
    }
    public void cancelAppointment() {
    }
    public void viewAppointment() {
    }
    public void viewBookingDetails() {
    }
    public void selectDoctorPreference() {
    }
    public void receiveNotification() {
    }
}
```

Doctor Class

The Doctor class represents doctors working in the hospital. It extends the User class and includes attributes related to specialization and availability.

```
public class Doctor extends User {


    private int doctorId;

    private String specialization;

    private String qualification;

    private String department;


    public void viewSchedule() {

    }


    public void approveAppointment() {

    }


    public void rejectAppointment() {

    }


    public void updateAvailability() {

    }


    public void viewPatientDetails() {

    }

}
```

Nurse Class

The Nurse class represents nurses who assist in patient care. It extends the User class and focuses on recording and updating patient health information.

```
public class Nurse extends User {

    private int nurseId;

    private String department;

    private String shift;

    private String assignedWard;


    public void viewAssignedPatients() {

    }

    public void recordPatientVitals() {

    }

    public void updatePatientStatus() {

    }

    public void assistDoctor() {

    }

    public void viewPatientHistory() {

    }

}
```

Admin Class

The Admin class represents system administrators. It extends the User class and provides operations for managing users, appointments, departments, and system reports.

```
public class Admin extends User {
    private int adminId;
    protected String officeLocation;
    protected String accessLevel;

public void manageDoctors() {
    }

    public void manageNurses() {
   }

    public void manageDepartments() {
   }

    public void assignRolesToUsers() {
   }

    public void manageRoles() {
    }

    public void managePermissions() {
    }
}
```

Hospital Class

The Hospital class represents the hospital entity. It stores hospital-related information and supports hospital management operations.

```
public class Hospital {

    private String hospitalId;

    private String hospitalName;

    private String address;

    private String phoneNumber;

    private String email;

    private String operatingHours;


    public void addDoctor() {

    }


    public void removeDoctor() {

    }


    public void addNurse() {

    }


    public void getAvailableDoctors() {

    }


    public void getHospitalDetails() {

    }

}
```

Appointment Class

The Appointment class manages appointment details such as date, time, reason, and status. It supports scheduling and appointment decision processes.

```
public class Appointment {
    private int appointmentId;
    private String appointmentDate;
    private String appointmentTime;
    private String reason;
    public void scheduleAppointment() {
    }

    public void cancelAppointment() {
    }

    public void approveAppointment() {
    }

    public void rejectAppointment() {
    }

    public void rescheduleAppointment() {
    }
    public void updateStatus() {
   }
}
```

Booking Class

The Booking class represents confirmed appointments. It links patients, doctors, and appointments together in a structured way.

```
public class Booking {


    private int bookingId;

    private String bookingDate;

    private String bookingStatus;

    private String remarks;


    public void createBooking() {

    }


    public void confirmBooking() {

    }


    public void cancelBooking() {

    }


    public void updateBookingStatus() {

    }

}
```

Role Class

The Role class defines different roles in the system. It helps manage access control by assigning roles to users.

```
public class Role {

    private int roleId;

    private String roleName;

    private String description;


    public void assignRole() {

    }


    public void removeRole() {

    }


    public void getRoleDetails() {

    }

}
```

Permission Class

The Permission class defines access permissions associated with roles. It controls which system modules can be accessed by users.

```
public class Permission {


    private int permissionId;

    private String permissionName;

    private String description;


    public void grantPermission() {

    }


    public void revokePermission() {

    }


    public void getPermissionDetails() {

    }

}
```

Conclusion and Future Work

The Hospital Appointment Management System demonstrates a well-organized and systematic software design that applies core software engineering principles to address real-world hospital appointment challenges. Through careful requirement analysis, object-oriented design, and UML-based modeling, the system effectively represents hospital workflows and clearly defines the roles and responsibilities of patients, doctors, nurses, and administrators. The use of multiple UML diagrams ensures that both the structural and behavioral aspects of the system are logically consistent, well-aligned, and easy to interpret.

As the project is design-focused and not intended for deployment, the primary emphasis is placed on design accuracy, modeling quality, and documentation clarity rather than runtime performance or system execution. This approach aligns well with academic objectives and highlights a strong understanding of system analysis, abstraction, and model-driven design. The project successfully illustrates how a complex real-life problem can be transformed into a structured and maintainable software design using standard modeling techniques.

Future work for this project is mainly conceptual and design-oriented. Potential enhancements include further refinement of UML diagrams to improve clarity and reduce redundancy, strengthening class relationships and constraints, and incorporating additional models such as database schemas and high-level user interface designs. If extended beyond the academic scope, the current system design could serve as a solid foundation for a fully implemented hospital management application. Within its defined scope, the project successfully achieves its objectives and stands as a complete, well-engineered academic software design.

References

- Software Engineering Tools & Practices – Group Project Submission Guideline UML
- Specification and Object-Oriented Design Principles
- Modelio Documentation

For more git hub link **https://abeni118.github.io/Hospital-Appointment-Management-System/**

Thank You!!!!!