UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Final Project: Software Architecture Design Section 04

Dr. Muhammad Khatib

Group Name :
Arch Dudes

Name	Matric No.
Khaled Osama Tahoon	A22EC4033
Omar Abdelmonem Hanfay	A22EC4012
Daffa Tangguh Ananda Kurniawan	A21EC4042
Feng Yiyang	A20EC4020
Zagidullin Ruslan	X24EC0005

Software Design Description (SDD)

Hospital Management System (HMS)

---

1. Introduction

1.1 Purpose

The purpose of this document is to provide a detailed design description of the **Hospital Management System (HMS)**. The system is designed to streamline hospital operations, including patient registration, appointment scheduling, medical record management, billing, and administrative tasks.

1.2 Scope

The HMS will support the following stakeholders:

- **Receptionists**: Register patients and manage appointments.
- **Patients**: Schedule visits and view medical records.
- **Doctors**: Update medical records and prescribe medications.
- **Admins**: Manage staff, generate reports, and update system configurations.

1.3 Stakeholders

- **Receptionists**
- **Patients**
- **Doctors**
- **Admins**

---

2. System Overview

2.1 High-Level Architecture

The HMS follows a **Layered Architecture** with three main layers:

1. **Presentation Layer**: Handles user interfaces for Receptionists, Patients, Doctors, and Admins.
2. **Business Logic Layer**: Manages core functionalities like patient registration, appointment scheduling, billing, and medical record management.
3. **Data Layer**: Stores and manages data securely, including patient records, appointments, and invoices.

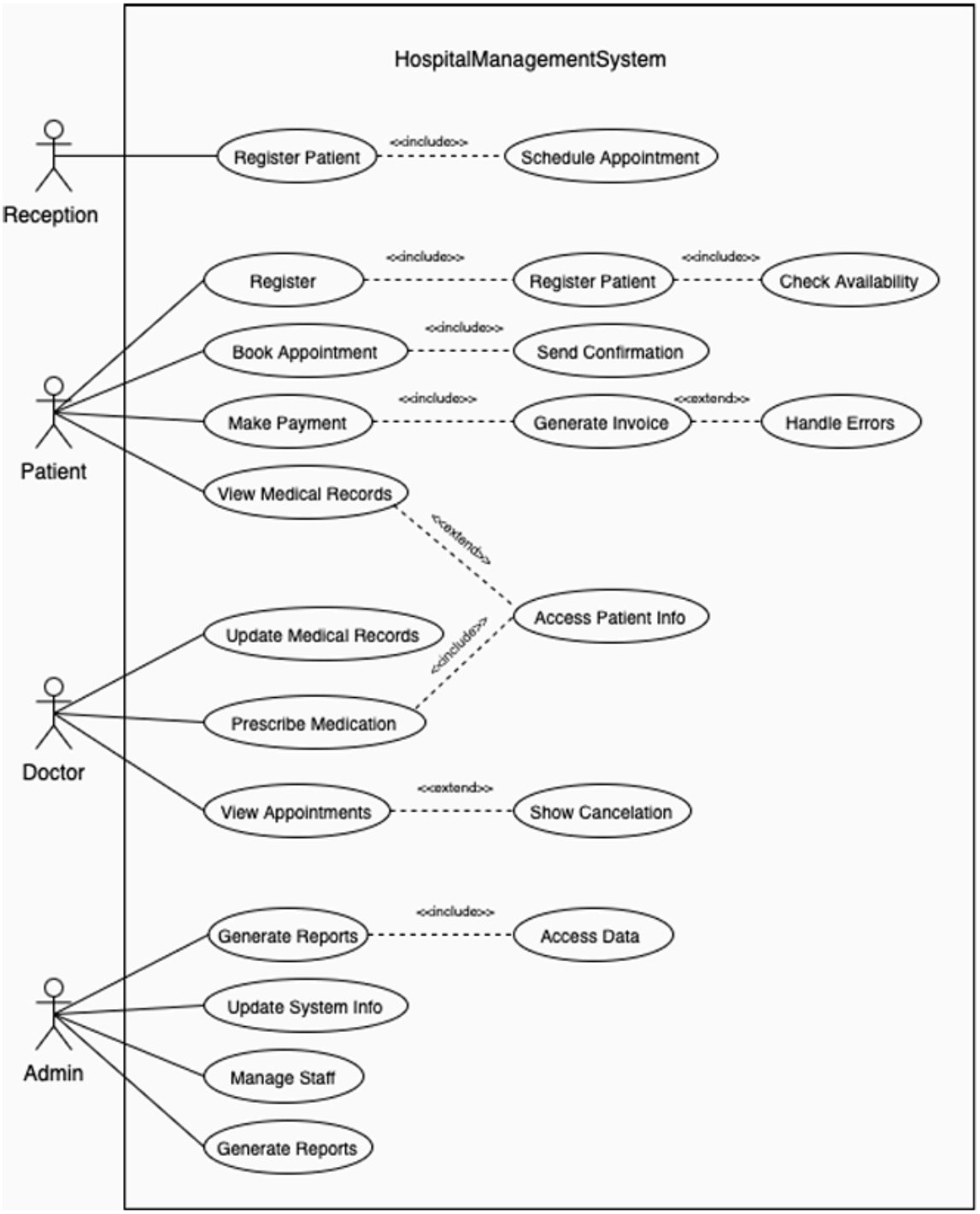
2.2 Components

- **Patient Registration Module**
- **Appointment Scheduling Module**
- **Billing and Payment Module**
- **Medical Records Module**
- **Admin Dashboard**

---

3. Use Case Diagrams and Specifications

3.1 Use Case Diagram



3.2 Use Case Specifications

UC001: Register Patient

Use Case ID	**UC001**
Use Case Name	Register Patient
Actor	Receptionist
Brief Description	The receptionist registers new patients into the system.
Pre-conditions	The system is operational; the receptionist is logged in.
Normal Flow	1. Receptionist selects “Register Patient.” 2. System prompts for patient details (name, age, gender, contact info, medical history). 3. Receptionist enters data. 4. System validates data and creates a new patient profile. 5. System displays a confirmation message.
Post-conditions	A new patient profile is created and stored in the database.
Alternative Flows	If mandatory data is missing, the system prompts the receptionist to correct the information.

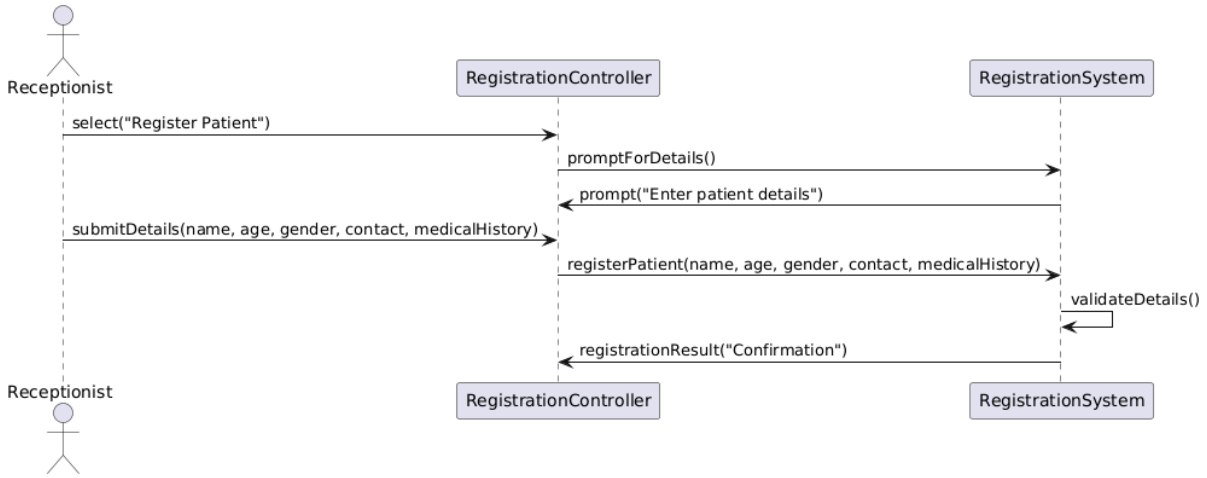
UC002: Book Appointment:

Use Case ID	**UC002**
Use Case Name	Book Appointment
Actor	Patient
Brief Description	Allows patients to book appointments with available doctors.
Pre-conditions	The patient is registered; the doctor’s schedule is available.
Normal Flow	1. Patient logs in and selects “Book Appointment.” 2. System displays available doctors and time slots. 3. Patient selects a preferred time slot and doctor. 4. System confirms the appointment and sends a confirmation.
Post-conditions	The appointment is successfully booked.
Alternative Flows	If the chosen time slot is unavailable, the system prompts the patient to choose another slot.

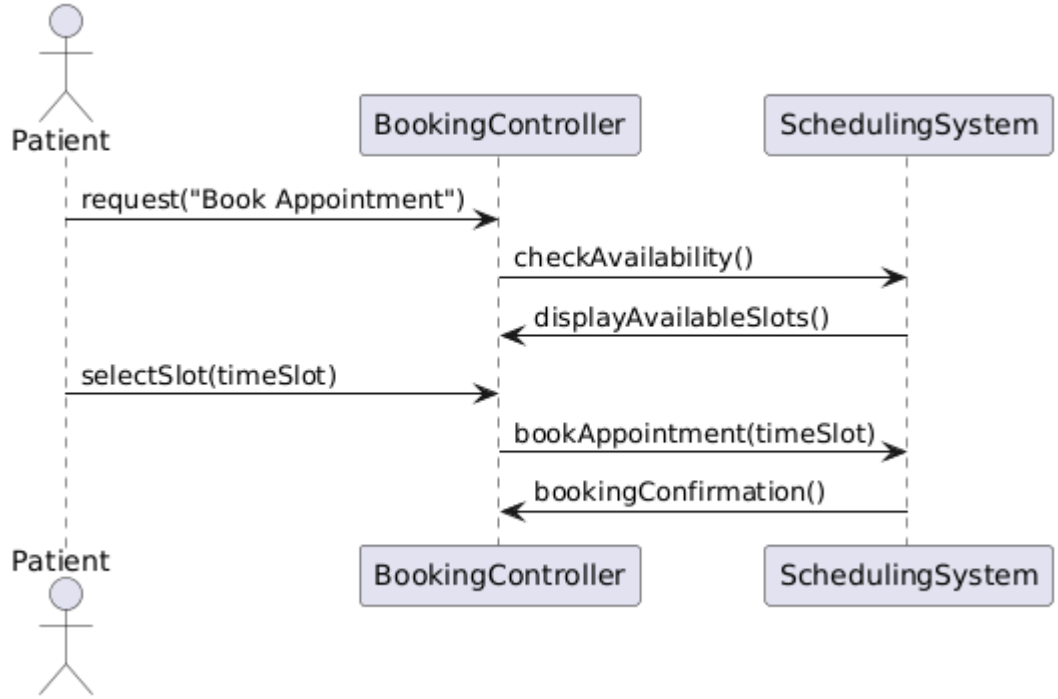
---

4. Sequence Diagrams

4.1 UC001: Register Patient

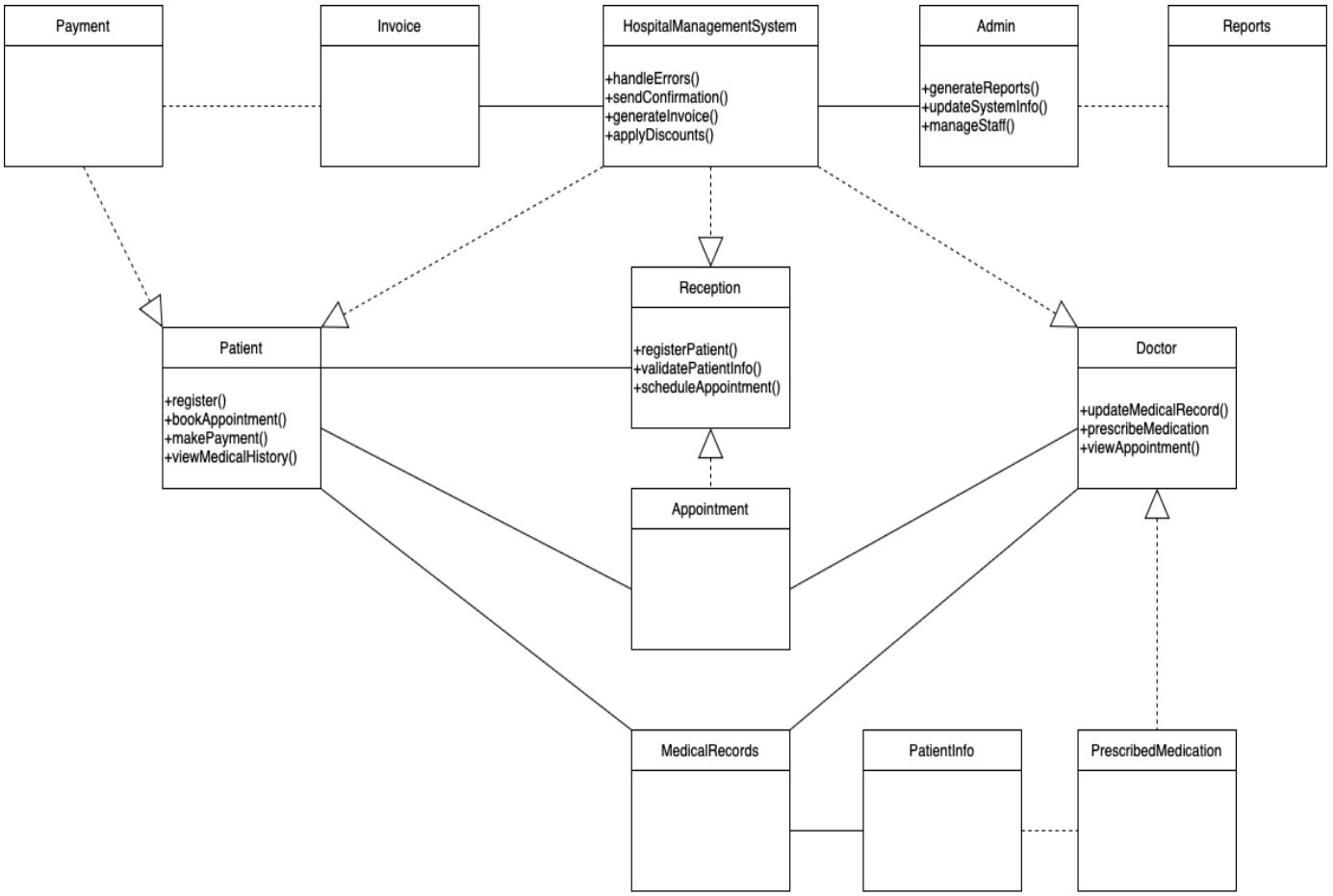


4.2 UC002: Book Appointment



---

5. Class Diagrams



6. Design Patterns:

Factory:

```cpp
#include <iostream>
#include <string>
using namespace std;

// Abstract Product
class Appointment {
public:
    string id;
    string patient;
    string doctor;
    string date;
    string time;
public:
    Appointment(string id, string patient, string doctor, string date, string time) {
        cout << "Appointment Details: " << endl;
        cout << "ID: " << id << ", Patient: " << patient << ", Doctor: " << doctor << ", Date: "
<< date << ", Time: " << time << endl;
    }
};

// Concrete Factory
class ReceptionFactory : public HospitalManagementSystem {
public:
    Appointment* createObject() override {
        return new Appointment("A101", "John Doe", "Dr. Smith", "2024-12-31", "10:30 AM");
    }
};

// Usage
int main() {
    ReceptionFactory receptionFactory;
    Appointment* appointment = receptionFactory.createObject();
    appointment->displayDetails();
    delete appointment;
    return 0;
}
```

Observer:

```cpp
#include <iostream>
#include <list>
#include <string>

using namespace std;

// Observer interface
class Observer {
public:
    virtual void update(const string& message) = 0;
};

// Subject
class Subject {
    list<Observer*> observers;
public:
    void attach(Observer* obs) {
        observers.push_back(obs);
    }

    void detach(Observer* obs) {
        observers.remove(obs);
    }

    void notify(const string& message) {
        for (auto& obs : observers) {
            obs->update(message);
        }
    }
};
```

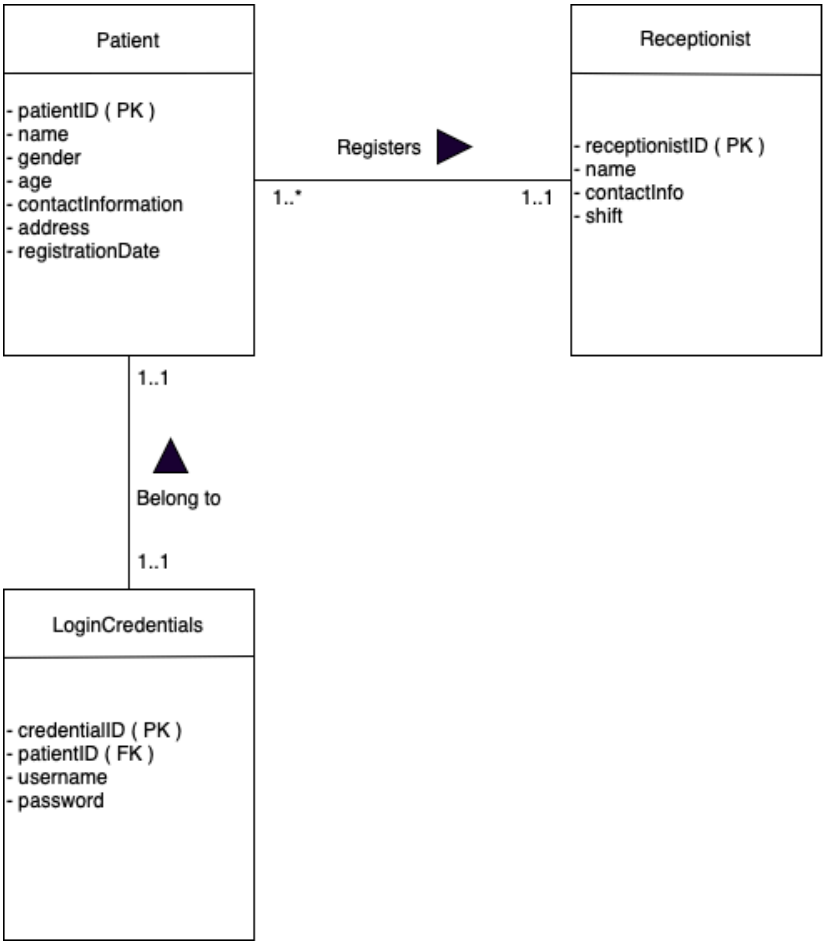
```cpp
// ConcreteSubject
class SystemNotifications : public Subject {
    string state;
public:
    void changeState(const string& newState) {
        state = newState;
        notify(state);
    }
};

// ConcreteObserver
class UserNotification : public Observer {
    string name;
public:
    UserNotification(const string& message) override {
        cout << "Notification for " << name << ". " << message << endl;
    }
};

// Usage example
int main() {
    SystemNotifications notifications;
    UserNotification doctor("Dr. Smith"), patient("John Doe");

    notifications.attach(&doctor);
    notifications.attach(&patient);

    notifications.changeState("Appointment Updated: New time 3 PM");

    notifications.detach(&patient);

    notifications.changeState("Medical Record Updated");

    return 0;
}
```

Facade:

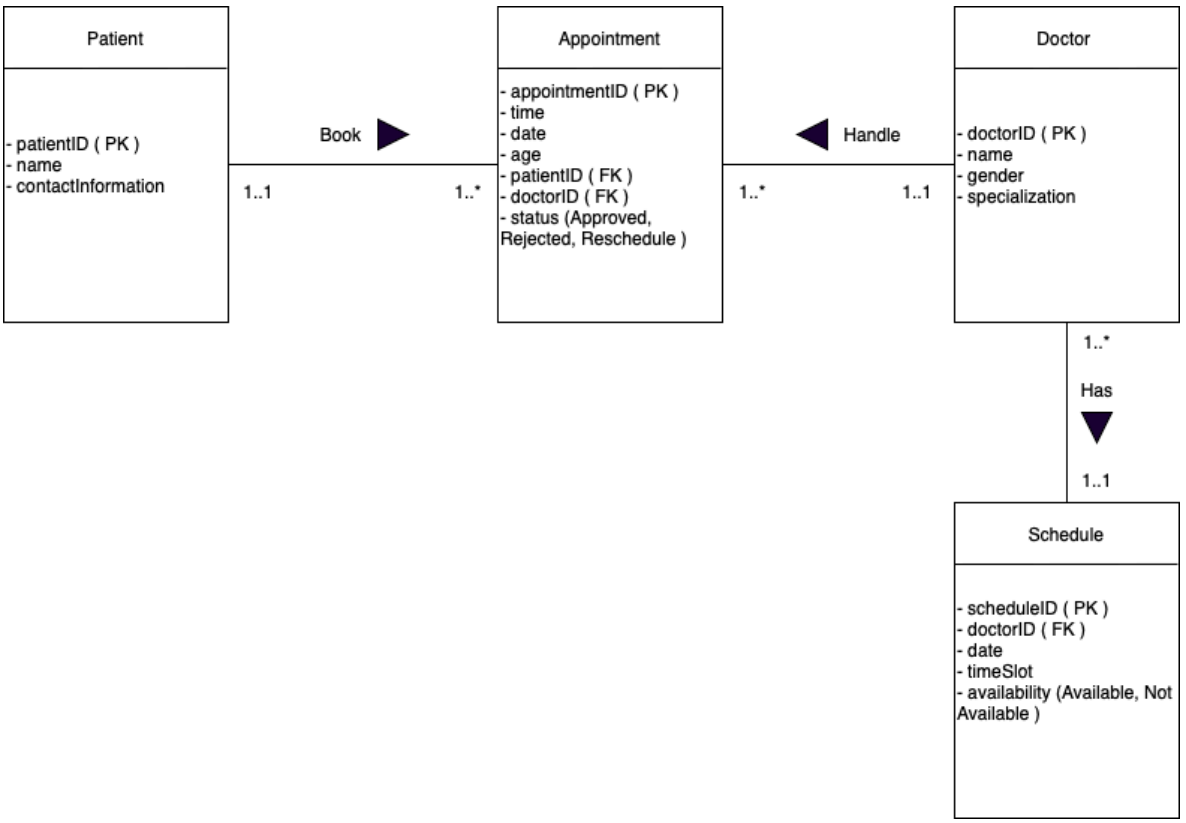
```cpp
// Subsystem classes
class RegistrationSystem {
public:
    void registerPatient(const string& patientName) {
        cout << "Registering patient: " << patientName << endl;
    }
};

class AppointmentSystem {
public:
    void bookAppointment(const string& date) {
        cout << "Booking appointment for: " << date << endl;
    }
};

class BillingSystem {
public:
    void processPayment(const string& patientName) {
        cout << "Processing payment for: " << patientName << endl;
    }
};

// Facade
class HospitalFacade {
    RegistrationSystem regSys;
    AppointmentSystem appSys;
    BillingSystem billSys;
public:
    void handleNewPatient(const string& patientName, const string& date) {
        regSys.registerPatient(patientName);
        appSys.bookAppointment(date);
        billSys.processPayment(patientName);
    }
};
```

7. Database Design (ERD)

7.1 Register Patient



7.2 Book Appointment



SQL

```sql
CREATE TABLE User (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL,
    Password VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    RoleID INT,
    FOREIGN KEY (RoleID) REFERENCES Role(RoleID)
);

CREATE TABLE Role (
    RoleID INT PRIMARY KEY,
    RoleName VARCHAR(50) NOT NULL
);

CREATE TABLE Permission (
    PermissionID INT PRIMARY KEY,
    PermissionName VARCHAR(50) NOT NULL
);

CREATE TABLE Permission (
    PermissionID INT PRIMARY KEY,
    PermissionName VARCHAR(50) NOT NULL
);

CREATE TABLE RolePermission (
    RoleID INT,
    PRIMARY KEY (RoleID,
    FOREIGN KEY (RoleID) REFERENCES Role(RoleID),
    FOREIGN KEY (RoleID) REFERENCES Permission(PermissionID)
);
```

8. Decision Table

8.1 Decision Table for User Authentication

Condition/Action	Rule 1	Rule 2	Rule 3	Rule 4
Valid Username	Yes	Yes	No	No
Valid Password	Yes	No	Yes	No
Grant Access	Yes	No	No	No
Deny Access	No	Yes	Yes	Yes

Explanation:

- **Rule 1**: Both username and password are valid → Grant access.
- **Rule 2**: Username is valid, but password is invalid → Deny access.
- **Rule 3**: Username is invalid, but password is valid → Deny access.
- **Rule 4**: Both username and password are invalid → Deny access.

8.2 Decision Table for Register Patient

Condition/Action	Rule 1	Rule 2	Rule 3
Valid information	Yes	Yes	No
Unique ID	Yes	No	Yes/No
Register Patient	Yes	No	No

Reject Registration	No	Yes	Yes

Explanation:

- **Rule 1**: Both information and unique ID are valid → Register the patient.
- **Rule 2**: Information is valid, but ID is not unique → Reject the registration .
- **Rule 3**: Information is invalid → Reject the registration regardless of the unique ID.

8.3 Decision Table for Book Appointment

Condition/Action	Rule 1	Rule 2	Rule 3	Rule 4
Doctor Available	Yes	Yes	No	Yes/No
Time Slot Free	Yes	No	Yes/No	Yes/No
Patient Registered	Yes	Yes	Yes	No
Approve Appointment	Yes	No	No	No
Request Reschedule	No	Yes	No	No
Reject Appointment	No	No	Yes	Yes

Explanation:

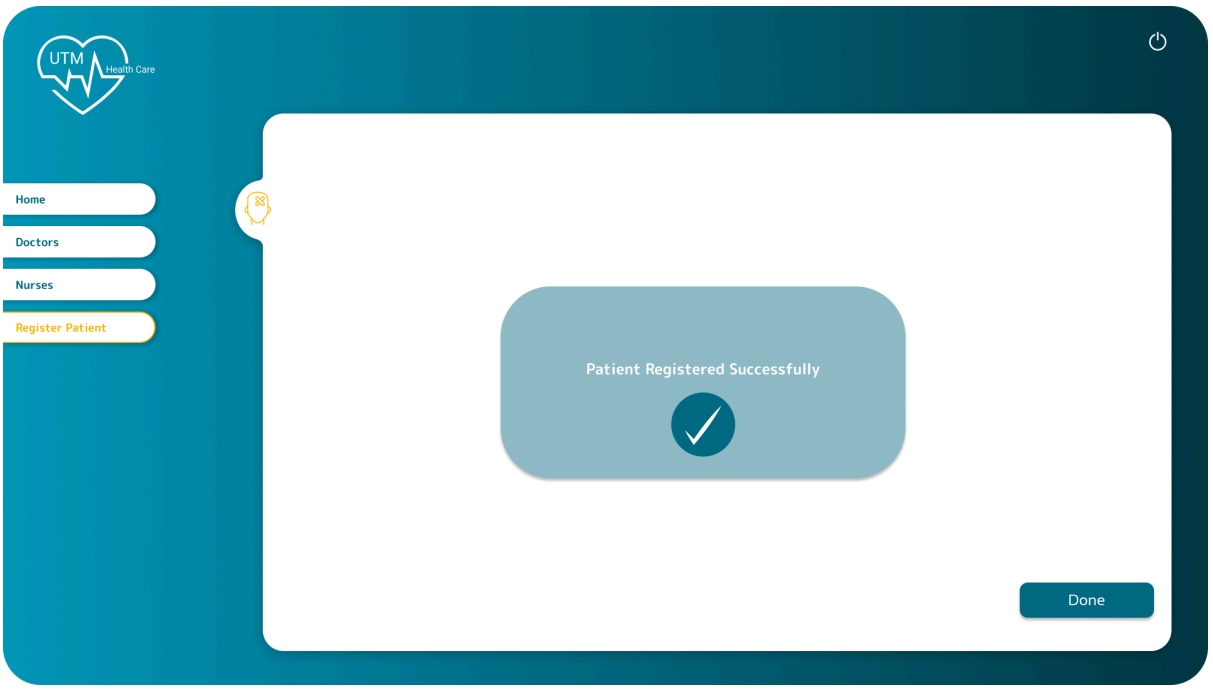
- **Rule 1**: All conditions are met → Approve the appointment.

- **Rule 2**: Doctor is available, but time slot is occupied → Request reschedule.
- **Rule 3**: Doctor is unavailable → Reject the appointment.
- **Rule 4**: Patient is not registered → Reject the appointment regardless of other conditions.

---

9. Wireframes

9.1 Register Patient



9.2 Book Appointment