FAKULTI TEKNOLOGI MAKLUMAT DAN KOMUNIKASI
SEMESTER 2 SESSION 2024/2025

BITP 3123 Distributed Application Development

Project 1
Hospital Appointment System Technical Report

LECTURER'S NAME
Ts. Faheem

Name	Matric Number
Muhammad Hadif Bin Shohaimi	B032420086
Muhammad Ammar Hafizin Bin Kamaruzzaman	B032420021

Table of Content

1. Introduction

1.1. Overview

Hospital Appointment System is developed for the patients and hospital staff to create/make appointments in hospitals via this platform. It is a network based system which adopts a client server architecture for communication between various parts, such as for example a client interface, a backend service, and database. The purpose is to provide a solid and easy to use application, which makes hospital processes easier, and also amplify the patient experience.

1.2. Objective

1. To allow hospital receptionists to easily book patient's appointments through a friendly interface.
2. To allow hospital staff to manage hospital appointments in real-time.
3. To implement a secure back-end system that handles appointment and communication between client and server components.
4. To develop an easy way for doctors to approve and schedule appointments.

1.3. Scope

1. Appointment creation, editing and deletion.
2. Real-time updating and tracking of appointment
3. Doctor approval and scheduling interface.
4. Validation of appointment.

1.4. Technologies and Tools Used

1. Eclipse IDE
2. Xampp
3. PHP
4. PDO
5. MySQL Database
6. Java SE
7. Apache
8. JSON
9. HTTP Protocol

2. System Analysis

2.1. Problem Statement

In most hospitals appointment management is still operated manually, on some machines are also used with the following problems: double booking, record missing, patient long waiting times and not updating recording etc. These problems may influence patient satisfaction and hospital operational efficiency. Therefore there is a need for an online Hospital Appointment system that handles appointments efficiently, in real time and as well as guarantees reliable communication between the users-to-backend -to database.

2.2. Requirements

The system must support multiple user roles (receptionist, doctor, staff) and enable the creation, updating, and tracking of hospital appointments. It must be secure, reliable, and accessible via a distributed client-server architecture.

2.3. User Requirement

1. Hospital receptionist should be able to create patient's appointment
2. Doctors should be able to view, approve and schedule patient's appointments.
3. Hospital staff should be able to track the appointment status in real-time.

2.4. Functional Requirement

1. The system should allow hospital receptionists to create new patient's appointment.
2. The system should allow doctors to view and approve appointments.
3. The system shall validate data.
4. The backend should process appointment data and store it into a database.
5. The system should provide user authentication for secure access.

2.5. Non-Functional Requirement

1. The system should have a user-friendly and responsive interface.
2. The system should be accessible locally.
3. The system should ensure data security and confidentiality.
4. The system should provide fast response time for user requests (<2 seconds).

2.6. Target Users

1. Hospital Receptionist: Responsible for creating new patient's appointments.
2. Doctor: View, approve and schedule appointments.

3. System Design
3.1. System Architecture

The system architecture is Three-Tier Client-Server architecture. Three-Tier Client-Server architecture is fitting as it is manageable and flexible by dividing the system into three layers (Presentation Layer, Application Layer, Data Layer).

Presentation Layer
This layer handles data display and user input in the shape of user interface. The user interface is created using Java Swing. The components for this layer are MainFrame and DoctorRequestWindow.

Application Layer
This layer handles the business logic, it processes user requests, performs computation and makes decisions.

Data Layer
This layer handles database operations and data retrieval. This storage layer is responsible for managing and storing data.

3.2. Entity-Relationship Diagram

appointments

PK	appointment_id
FK1	patient_id
FK2	doctor_id
	appointment_date
	appointment_time
	status
	notes
	created_at

patients

PK	patient_id
	first_name
	last_name
	phone
	email
	address
	date_of_birth
	created_at

doctors

PK	doctor_id
	first_name
	last_name
	specialization
	phone
	email
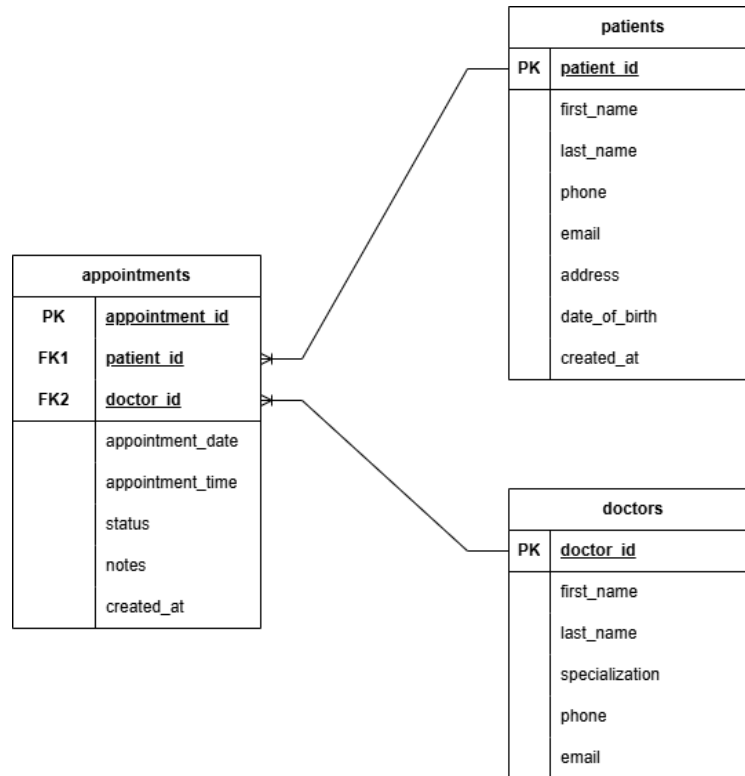
3.3. Backend Design

Appointment.java

```java
82      // Serialize to JSON with snake_case keys
83      public JSONObject toJSON() {
84          JSONObject json = new JSONObject();
85          json.put("appointment_id", appointmentId);
86          json.put("patient_id", patientId);
87          json.put("doctor_id", doctorId);
88          json.put("appointment_date", appointmentDate != null ? new java.text.SimpleDateFormat("yyyy-MM-dd'
89          json.put("appointment_time", appointmentTime);
90          json.put("status", status);
91          json.put("notes", notes);
92          return json;
93      }
94
95      // Deserialize from JSON with snake_case keys
96      public static Appointment fromJSON(JSONObject json) {
97          Appointment appointment = new Appointment();
98          appointment.setAppointmentId(json.optInt("appointment_id"));
99          appointment.setPatientId(json.optInt("patient_id"));
100         appointment.setDoctorId(json.optInt("doctor_id"));
101         String dateStr = json.optString("appointment_date", "");
102         if (!dateStr.isEmpty() && !dateStr.equals("null")) {
103             try {
104                 appointment.setAppointmentDate(java.sql.Date.valueOf(dateStr));
105             } catch (Exception e) {
106                 e.printStackTrace();
107             }
108         }
109         appointment.setAppointmentTime(json.optString("appointment_time"));
110         appointment.setStatus(json.optString("status"));
111         appointment.setNotes(json.optString("notes"));
112         appointment.setPatientName(json.optString("patient_name"));
113         appointment.setDoctorName(json.optString("doctor_name"));
114         return appointment;
115     }
116 }
117
```

Doctor.java

```java
35      // Return JSON representation
36      public JSONObject toJSON() {
37          JSONObject json = new JSONObject();
38          json.put("doctor_id", doctorId);
39          json.put("first_name", firstName);
40          json.put("last_name", lastName);
41          json.put("specialization", specialization);
42          json.put("phone", phone);
43          json.put("email", email);
44          return json;
45      }
46
47      // Convenience method for displaying full name
48      public String getFullName() {
49          return firstName + " " + lastName;
50      }
51
52      public static Doctor fromJSON(JSONObject json) {
53          Doctor doctor = new Doctor();
54          doctor.setDoctorId(json.optInt("doctor_id"));
55          doctor.setFirstName(json.optString("first_name"));
56          doctor.setLastName(json.optString("last_name"));
57          doctor.setSpecialization(json.optString("specialization"));
58          doctor.setPhone(json.optString("phone"));
59          doctor.setEmail(json.optString("email"));
60          return doctor;
61      }
62
63      // For combo boxes or debug printing
64      @Override
65      public String toString() {
66          return getFullName(); // Or use this: "Dr. " + getFullName() + " (" + specialization + ")";
67      }
68 }
```

Patient.java

```java
public JSONObject toJSON() {
    JSONObject json = new JSONObject();
    json.put("patient_id", patientId);

        json.put("firstName", firstName);    // ☑ Correct key
        json.put("lastName", lastName);      // ☑ Correct key
        json.put("phone", phone);
        json.put("email", email);
        json.put("address", address);
        if (dateOfBirth != null) {
            String dobStr = new SimpleDateFormat("yyyy-MM-dd").format(dateOfBirth);
            json.put("dateOfBirth", dobStr);
        } else {
            json.put("dateOfBirth", JSONObject.NULL);
        }
        return json;
```

```java
public JSONObject toJSON() {
    JSONObject json = new JSONObject();
    json.put("patient_id", patientId);

        json.put("firstName", firstName);    // ☑ Correct key
        json.put("lastName", lastName);      // ☑ Correct key
        json.put("phone", phone);
        json.put("email", email);
        json.put("address", address);
        if (dateOfBirth != null) {
            String dobStr = new SimpleDateFormat("yyyy-MM-dd").format(dateOfBirth);
            json.put("dateOfBirth", dobStr);
        } else {
            json.put("dateOfBirth", JSONObject.NULL);
        }
        return json;
```
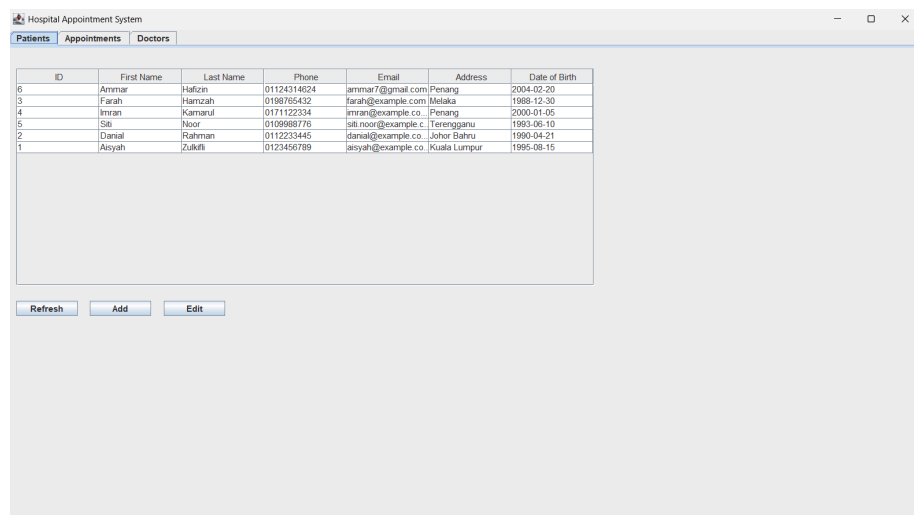
3.4. Frontend Design

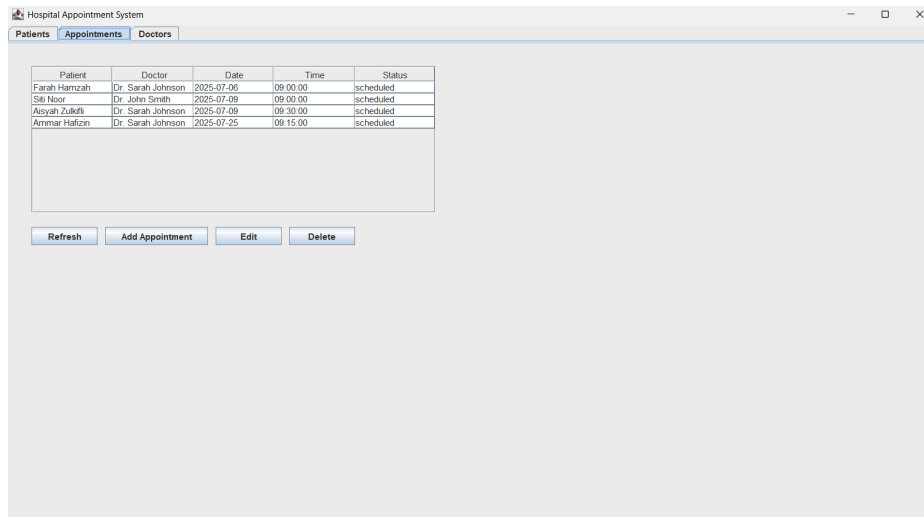Figure 3.1 - MainFrame (Patient Panel)
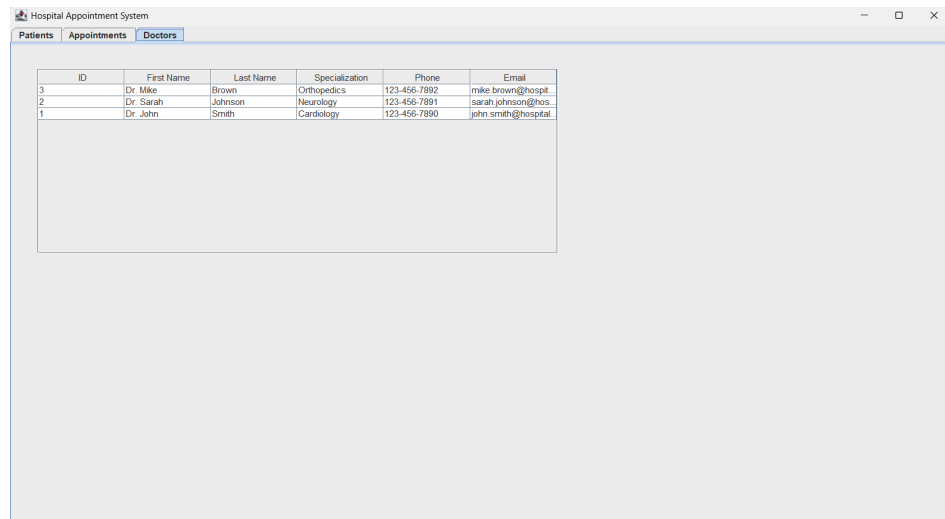
Figure 3.2 - MainFrame (Appointment Panel)

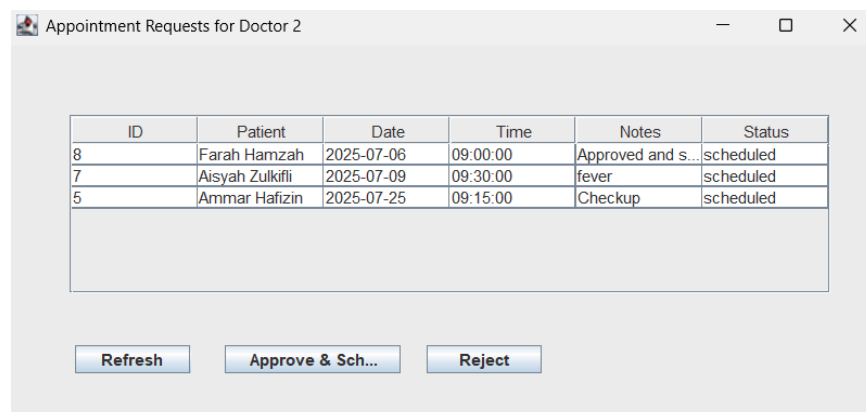Figure 3.3 - MainFrame (Doctor List)

Figure 3.4 - DoctorRequestWindow

4. Application Development

4.1. Backend Application

4.1.1. Technologies and Frameworks Used

1. PHP
2. PDO
3. MySQL Database
4. Apache HTTP Server
5. RESTful Architecture
6. HTTP Protocol
7. JSON
8. CORS (Cross-Origin Resource Sharing)

4.1.2. API Designs and Sockets (REST/SOAP/Socket)

The system implements a RESTful API architecture with the following characteristics:

1. Resource-Based URLs: Each entity has its own endpoint
2. HTTP Methods: Proper use of GET, POST, PUT, and DELETE for CRUD operations
3. JSON Communications: All data exchange uses JSON format
4. Stateless: Each request contains all necessary information

API Endpoint Structure
```
Base URL: http://localhost/hospital_management/php_backend
Patients API:
- GET /patients_api.php – Get all patients
- GET /patients_api.php/{id} – Get specific patient
- POST /patients_api.php – Create new patient
- PUT /patients_api.php/{id} – Update patient
- DELETE /patients_api.php/{id} – Delete patient

Doctors API:
- GET /doctors_api.php – Get all doctors
- GET /doctors_api.php/{id} – Get specific doctor
- POST /doctors_api.php – Create new doctor
- PUT /doctors_api.php/{id} – Update doctor
- DELETE /doctors_api.php/{id} – Delete doctor

Appointments API:
- GET /appointments_api.php – Get all appointments
```

```
- GET /appointments_api.php/{id} - Get specific appointment
- POST /appointments_api.php - Create new appointment
- PUT /appointments_api.php/{id} - Update appointment
- DELETE /appointments_api.php/{id} - Delete appointment
```

4.1.3.   Backend Functionalities

1. Patient Management
   a. Create Patients: Register new patients with personal information
   b. Read Patients: Retrieve patients details
   c. Update Patients: Modify patients information
   d. Delete Patients: Remove patients record
   e. Data Fields: First name, last name, phone, email, address, date of birth

2. Doctor Management
   a. Create Doctor: Register new doctor with specialization
   b. Read Doctor: Retrieve doctor details
   c. Update Doctor: Modify doctor details
   d. Delete Doctor: Delete doctor record
   e. Data Fields: First name, last name, email, phone, specialization

3. Appointment Management
   a. Create Appointments: Register new appointments between patient and doctor
   b. Read Appointment: Retrieve Appointments details
   c. Update Appointment: Modify appointment details
   d. Delete Appointment: Delete appointment records
   e. Data Fields: Patient ID, doctor ID, appointment date, time, status, notes

4. Data Relationships
   a. JOIN Operations: Appointments API performs SQL.JOINs to fetch patients names, doctor names, and specialization
   b. Foreign Key Relationships: Appointments reference patient ID and doctor ID
   c. Data Integrity: Proper handling of NULL values and required fields

5. Java Client Integration
    a. RestClient Class: Java HTTP client for consuming the PHP APIs
    b. HTTP Methods: Implement GET, POST, PUT, and DELETE operations
    c. JSON Parsing: Converts between Java objects and JSON
    d. Error Handling: Comprehensive exception handling and logging

4.1.4.   Security Mechanism

1. Input Validation
    a. JSON Validation:
    b. Required Field Validation:
    c. Data Type Validation

2. SQL Injection Prevention
    a. Prepared Statement:
    b. Parameter Binding:
    c. No Direct SQL Concatenations:

3. Error Handling
    a. Exception Handling:
    b. Error Suppression:
    c. Structured Error Messages:

4. CORS Configuration
    a. Cross-Origin Headers:
    b. Method Restrictions:
    c. Content Type Headers:

4.2. Frontend Application 1
4.2.1. Technologies and Frameworks Used

1. Eclipse IDE
2. Java Frontend
3. RestClient
4. Apache HttpClient
5. Swing

4.2.2. Features

Application 1 is a main application for hospital receptionists to register new patients and request an appointment. Other than that, application 1 can view appointments and doctor lists, and update appointments.

Patient Panel

1. Display list of registered patients inside a table
2. Create or register new patient
3. Update patient data
4. Refresh patient list

Appointment Panel

1. Display list of appointments inside a table
2. Create or register new appointment
3. Update or edit appointment
4. Delete appointment
5. Refresh appointment list

Doctor Panel

1. Display list of doctors inside a table

4.2.3. User Interface Design
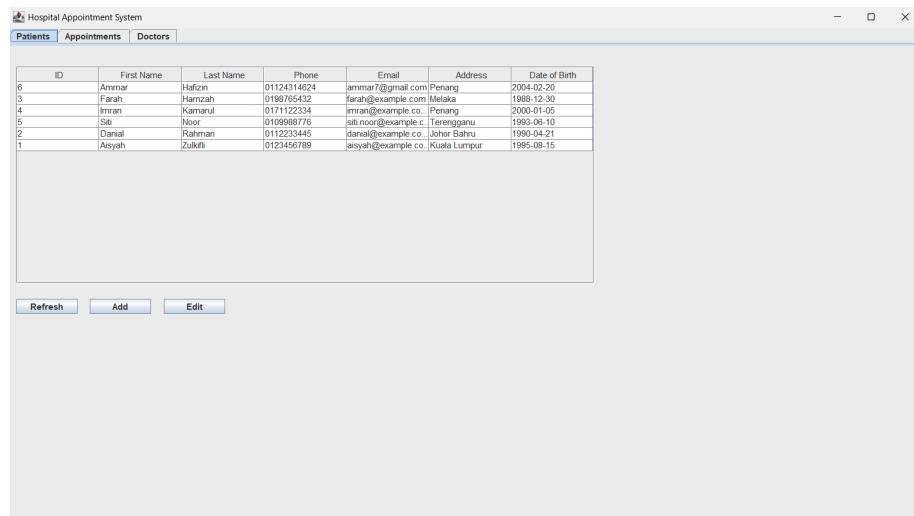
Figure 3.1 - Application 1 (Patient Panel)
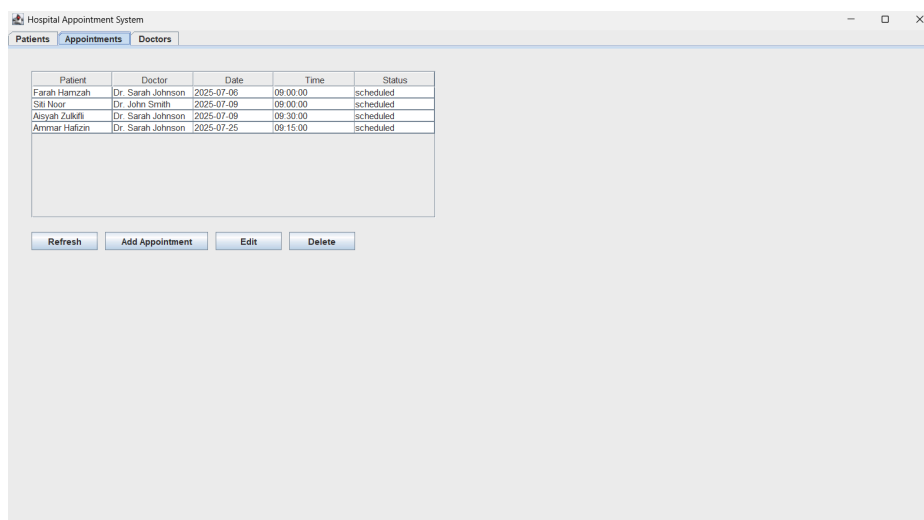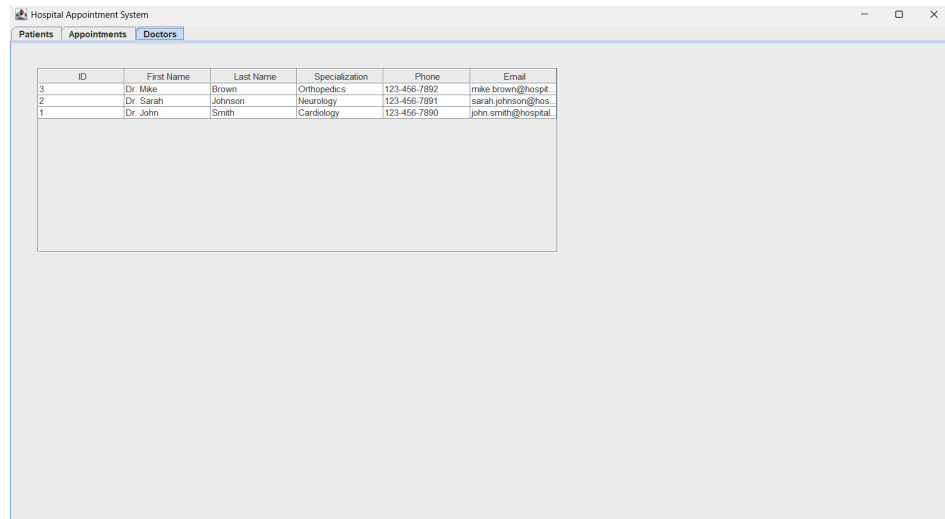
Figure 3.2 - Application 1 (Appointment Panel)

Figure 3.3 - Application 1 (Doctor List)

4.3. Frontend Application 2
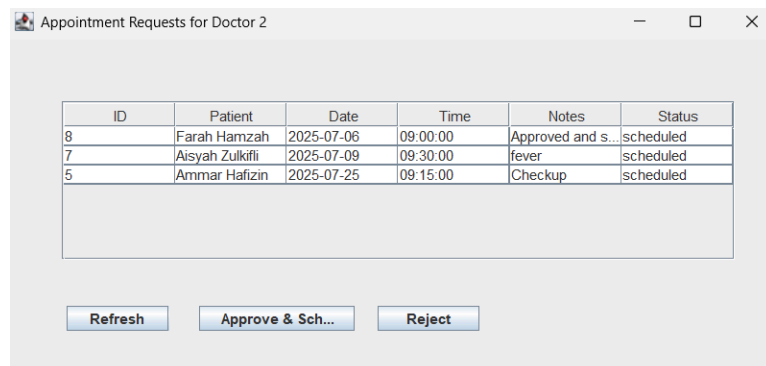
4.3.1. Technologies and Frameworks Used

1. Eclipse IDE
2. Java Frontend
3. Restclient
4. Apache HttpClient
5. Swing

4.3.2. Features

1. Display list of appointment inside a table
2. Approve and schedule an appointment
3. Reject an appointment
4. Refresh appointment list

4.3.3. User Interface Design

Figure 3.4 - DoctorRequestWindow

5. Integration and Data Flow
   5.1. Communication Between Backend and Frontend

The system uses a Client-Server Architecture with HTTP-based REST API communication between the JAVA frontend and PHP backend.
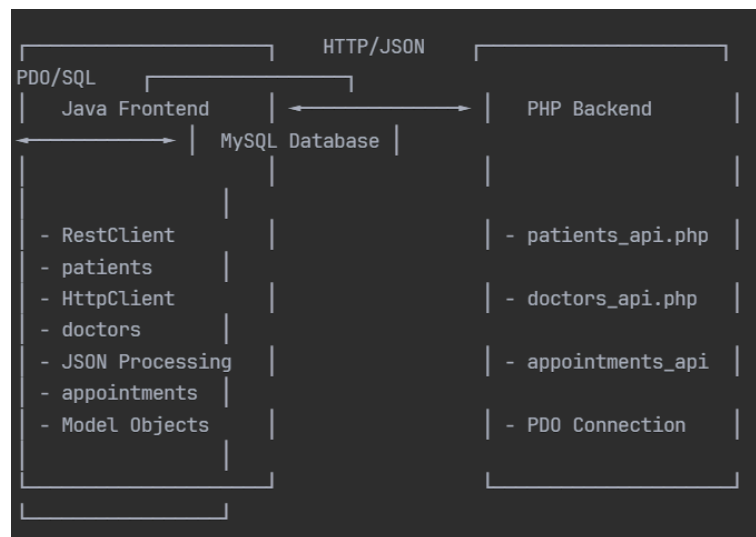
Communication Flow
1. Java Client creates HTTP request with JSON payload
2. Apache HttpClient handles HTTP communication
3. PHP Backend receives request and processes them
4. PDO executes SQL queries against MySQL
5. JSON responses are sent back to the client
6. Java Client parses JSON

5.2. Example API Request and Response

1. Patient Management Examples

<u>Create Patient Request</u>

```http
http
POST /hospital_management/php_backend/patients_api.php
Content-Type: application/json; charset=UTF-8
{
"firstName": "John",
"lastName": "Doe",
"phone": "555-1234",
"email": "john.doe@email.com",
"address": "123 Main St, City, State",
"dateOfBirth": null
}
```

<u>Create Patient Response (Success)</u>

```json
json
{
"success": true,
"patient_id": 15,
"message": "Patient created successfully"
}
```

<u>Create Patient Response (Error)</u>

```json
json
{
"error": "Missing required patient data"
}
```

<u>Get All Patient Request</u>

```http
http
GET /hospital_management/php_backend/patients_api.php
```

<u>Get All Patient Response</u>

```json
[
    {
            "patient_id": "1",
            "first_name": "John",
            "last_name": "Doe",
            "phone": "555-1234",
            "email": "john.doe@email.com",
            "address": "123 Main St, City, State",
            "date_of_birth": null
    },
    {
            "patient_id": "2",
            "first_name": "Jane",
            "last_name": "Smith",
            "phone": "555-5678",
            "email": "jane.smith@email.com",
            "address": "456 Oak Ave, City, State",
            "date_of_birth": "1990-05-15"
    }
]
```

2. Doctor Management Examples

<u>Create Doctor Request</u>

```http
POST /hospital_management/php_backend/doctors_api.php
Content-Type: application/json; charset=UTF-8

{
    "firstName": "Dr. Sarah",
    "lastName": "Johnson",
    "email": "sarah.johnson@hospital.com",
    "phone": "555-9876",
    "specialization": "Cardiology"
}
```

Create Doctor Response

```json
{
    "success": true,
    "doctor_id": 8
}
```

Get All Doctor Response

```json
[
    {
        "doctor_id": "1",
        "first_name": "Dr. Sarah",
        "last_name": "Johnson",
        "email": "sarah.johnson@hospital.com",
        "phone": "555-9876",
        "specialization": "Cardiology"
    },
    {
        "doctor_id": "2",
        "first_name": "Dr. Michael",
        "last_name": "Brown",
        "email": "michael.brown@hospital.com",
        "phone": "555-4321",
        "specialization": "Pediatrics"
    }
]
```

3. Appointment Management Examples

Create Appointment Request

```http
POST /hospital_management/php_backend/appointments_api.php
Content-Type: application/json; charset=UTF-8

{
    "patientId": "1",
    "doctorId": "1",
    "appointmentDate": "2024-12-15",
    "appointmentTime": "14:30:00",
    "notes": "Regular checkup appointment"

}
```

Create Appointment Response

```http
POST /hospital_management/php_backend/appointments_api.php
Content-Type: application/json; charset=UTF-8

{
    "patientId": "1",
    "doctorId": "1",
    "appointmentDate": "2024-12-15",
    "appointmentTime": "14:30:00",
    "notes": "Regular checkup appointment"

}
```

Get All Appointments Response (with JOINs)

```json
json
[
    {
        "appointment_id": "1",
        "patient_id": "1",
        "doctor_id": "1",
        "appointment_date": "2024-12-15",
        "appointment_time": "14:30:00",
        "status": "scheduled",
        "notes": "Regular checkup appointment",
        "patient_name": "John Doe",
        "doctor_name": "Dr. Sarah Johnson",
        "specialization": "Cardiology"
    },
    {
        "appointment_id": "2",
        "patient_id": "2",
        "doctor_id": "2",
        "appointment_date": "2024-12-16",
        "appointment_time": "09:00:00",
        "status": "completed",
        "notes": "Follow-up visit",
        "patient_name": "Jane Smith",
        "doctor_name": "Dr. Michael Brown",
        "specialization": "Pediatrics"
    }

]
```

Update Appointment Status Request

```http
PUT /hospital_management/php_backend/appointments_api.php/1
Content-Type: application/json; charset=UTF-8

{
    "patientId": "1",
    "doctorId": "1",
    "appointmentDate": "2024-12-15",
    "appointmentTime": "14:30:00",
    "status": "completed",
    "notes": "Patient arrived on time. Vital signs normal."
}
```
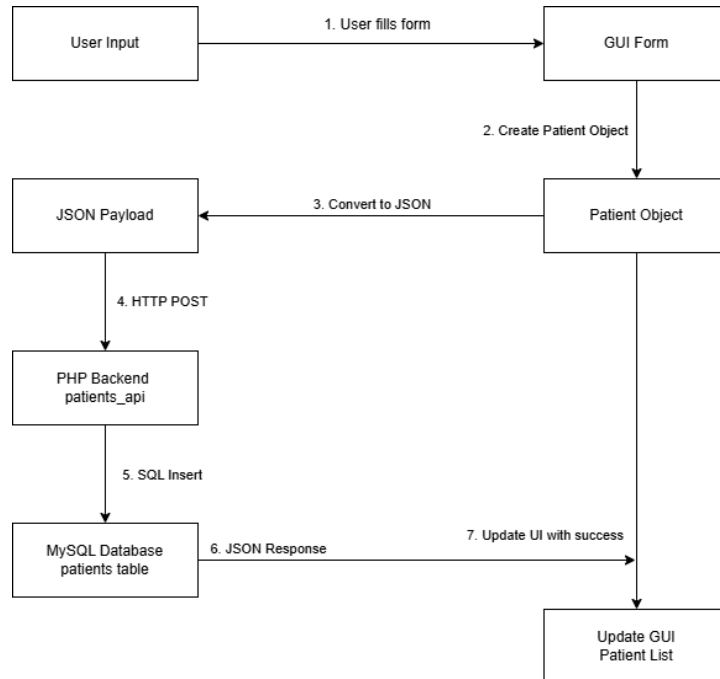
Update Appointment Response

```json
{
    "success": true,
    "message": "Appointment updated successfully"
}
```

5.3.   Demonstration of Business Workflow

1.   Patient Registration Workflow

2.   Appointment Scheduling Workflow
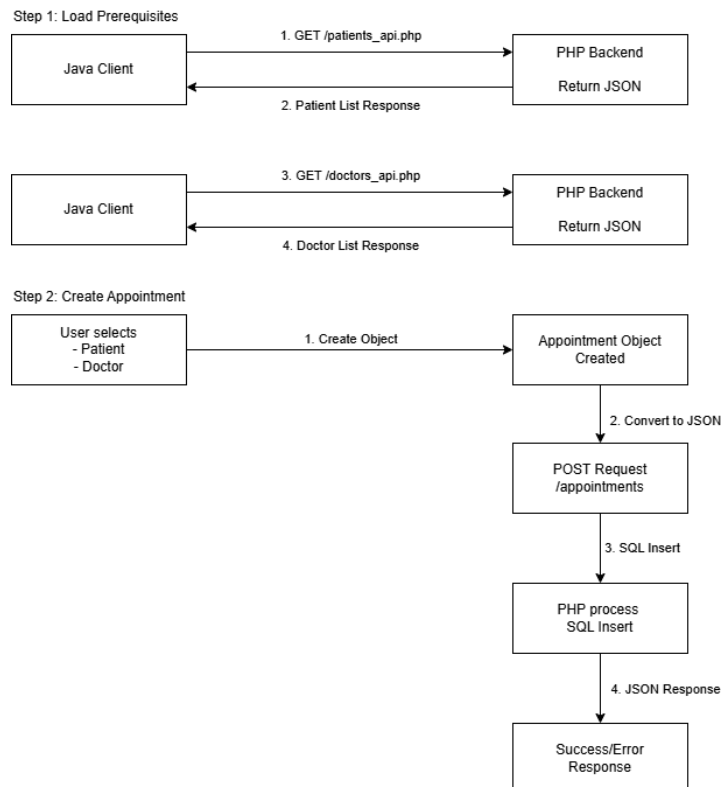
3. Appointment Management Workflow

View Appointments

```
┌─────────────────┐   1. GET /appointments_api.php   ┌─────────────────┐
│                 │ ───────────────────────────────► │  PHP Backend    │
│   Java Client   │                                  │                 │
│                 │ ◄─────────────────────────────── │   SQL JOIN      │
│                 │   2. Appointments with names     │   patients &    │
└─────────────────┘                                  │   doctors       │
         │                                           └─────────────────┘
         │ 3. Refresh Table
         ▼
┌─────────────────┐
│  Display Table  │
│ - Patient Name  │
│ - Doctor Name   │
│  - Date/Time    │
│   - Statuis     │
└─────────────────┘
```

Update Appointment Statis

```
┌─────────────────┐   1. Select Appointment   ┌─────────────────┐
│                 │ ────────────────────────► │                 │
│   User Action   │                           │   Get Details   │
│                 │                           │                 │
└─────────────────┘                           └─────────────────┘
                                                       │
                                                       │ 2. Fetch Current Data
                                                       ▼
                                              ┌─────────────────┐
                                              │  Get /appoint/ID │
                                              │ Retrieve full    │
                                              │ Appointment      │
                                              └─────────────────┘
                                                       │
                                                       │ 3. Update status
                                                       ▼
                                              ┌─────────────────┐
                                              │  PUT /appoint/ID │
                                              │ Full payload with│
                                              │   new status     │
                                              └─────────────────┘
```

6.   Database Design
6.1.   Database Schema

6.2. Tables and Attributes Description

<u>Appointments Table</u>

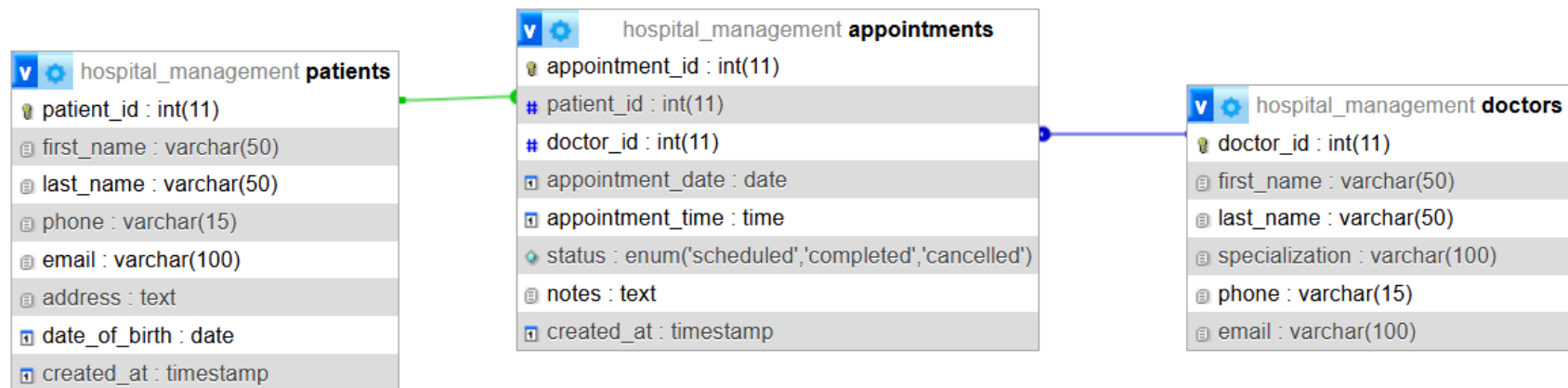| ATTRIBUTE NAME	CONTENTS	DATA TYPE AND SIZE	FORMAT	RANGE	REQUIRED?	DEFAULT VALUE	UNIQUE?	PK OR FK?	FK REFERENCED TABLE
appointment_id	Appointment's id	int(11)	N/A	-	YES	-	YES	PK	-
patient_id	Patient's Id	int(11)	N/A	-	NO	-	NO	FK	PATIENTS
doctor_id	Doctor's Id	int(11)	N/A	-	NO	-	NO	FK	DOCTORS
appointment_date	Appointment''s date	date	N/A	-	NO	-	NO	-	-
appointment_time	Appointment's time	time	N/A	-	NO	-	NO	-	-
status	Appointment's status	enum('scheduled', 'completed'. 'cancelled'')	N/A	-	NO	scheduled-	NO	-	-
notes	Doctor's notes	text	N/A	-	NO	-	NO	-	-
created_at	Time/Date the appointment is created	timestamp	N/A	-	YES	current_timestamp()	NO	-	-

Doctors Table

ATTRIBUTE NAME	CONTENTS	DATA TYPE AND SIZE	FORMAT	RANGE	REQUIRED?	DEFAULT VALUE	UNIQUE?	PK OR FK?	FK REFERENCED TABLE
doctor_id	Doctor's id	int(11)	N/A	-	YES	-	YES	PK	-
first_name	Doctor's first name	varchar(50)	N/A	-	YES	-	NO	-	-
last_name	Doctor's Last Name	varchar(50)	N/A	-	YES	-	NO	-	-
specialization	Doctor's specialization	varchar(100)	N/A	-	NO	-	NO	-	-
phone	Doctor's phone number	varchar(15)	N/A	-	NO	-	NO	-	-
email	Doctor's email	varchar(100)	N/A	-	NO	-	NO	-	-

Patients Table

ATTRIBUTE NAME	CONTENTS	DATA TYPE AND SIZE	FORMAT	RANGE	REQUIRED?	DEFAULT VALUE	UNIQUE?	PK OR FK?	FK REFERENCED TABLE
patient_id	Patient's id	int(11)	N/A	-	YES	-	YES	PK	-
first_name	Patient's first name	varchar(50)	N/A	-	YES	-	NO		
last_name	Patient's Last Name	varchar(50)	N/A	-	YES	-	NO		
phone	Patient's phone number	varchar(15)	N/A	-	NO	-	NO	-	-
email	Patient's email	varchar(100)	N/A	-	NO	--	NO	-	-
address	Patient's address	text	N/A	-	NO	-	NO	-	-
date_of_birth	Patient's date of birth	date	N/A	-	NO	-	NO	-	-
created_at	Time/Date the patient is registered	timestamp	N/A	-	YES	current_timestamp()	NO	-	-

7. Commercial Value

7.1. Market Value and Potential Impact

The market value for this system is believed to be highly significant as it targets the healthcare industry, a sector that is continuously in demand. With the global growing population and increasingly growing population, the need for accessible and efficient healthcare service is expected to rise in the coming years. Hospitals and clinics are under constant pressure to manage large volumes of patients while maintaining quality and timely services.

This system directly addresses a key operational bottleneck in healthcare facilities such as appointment scheduling and management. By digitizing the appointment process, the system can help reduce administrative workload, minimize scheduling conflicts, and decrease patient waiting times.This leads to improved operational efficiency, enhanced patient satisfaction, and better resource utilization within the healthcare infrastructure.

In addition to that, the system holds commercial potential for widespread adoption not only by hospitals but also by clinics, specialist centers, and telemedicine providers, especially in developing countries where digital transformation in healthcare is emerging.

8.   Conclusions

8.1.   Limitations

The limitations of the system is it is inaccessible with the internet. Patients need to go to their local hospital and register their appointment through the hospital receptionist. This is quite inconvenient as patients still have to take their time to go out to make their appointment, instead of making their appointment easily accessible through the web.

Other than that, the system lacks searchability options such as search bar or table filter. This limitation makes it difficult and time inducing for the user to find specific patients or appointments.

8.2.   Future Enhancements

From the limitations that have been addressed, future enhancements need to be made to address the issues. The system must be able to be accessed through the internet either through a website or application service. This solutions will make the system significantly more accessible by both patients and hospital staffs as administrative work can be done online.

Secondly, searchability features need to be added as a way to make finding specific items in specific options or criteria more simple and easy. This feature will reduce the time it takes for hospital staff to manage their workload, making healthcare system more robust and systematic.

8.3. Summary

In conclusion, the Hospital Appointment System is a practical and efficient solution to one of the most challenging challenges in the healthcare industry such as managing patients appointments. By implementing a distributed client-server architecture, the system allows seamless interaction between users, the backend service and the database, ensuring reliable data flow and real-time updates.

The system successfully achieves its core objectives by enabling hospital receptionists to easily register new appointments, allowing doctors to manage their schedule efficiently, and provide a secure backend to handle communication and data storage. This enhances workflow efficiency, reduces administrative burden, and improves the overall patient experience.

This project has not only fulfilled academic requirements but also provided valuable hands-on experience in designing, developing, and deploying a distributed application that addresses real world needs.

Reference

1. saurav (1 July 2015), SQLSTATE[HY000], [1045] Access denied for user 'username'@'localhost', StackOverflow - [php - SQLSTATE[HY000] [1045] Access denied for user 'username'@'localhost' - Stack Overflow](#)
2. user3406220 (20 March 2014), ERROR: SQLSTATE[HY000] [2002] No connection could be made because target machine actively refused it, Stack Overflow - [php - ERROR: SQLSTATE[HY000] [2002] No connection could be made because the target machine actively refused it - Stack Overflow](#)
3. Dani (18 July 2018), Incompatible Java Versions for WindowBuilder, Stack Overflow - [eclipse - Incompatible Java Versions for WindowBuilder - Stack Overflow](#)