RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI ENGINEERING COLLEGE

CS23333 OOPS Using Java

Laboratory Record Note Book

Name :

Year / Branch / Section :

University Register No. : ..

College Roll No. : .

Semester : .

Academic Year : .

RAJALAKSHMI ENGINEERING COLLEGE

An Autonomous Institution

BONAFIDE CERTIFICATE

Name: ……………………………………………………………………………

Academic Year: …………… **Semester:** …………… **Branch:** ………………

Register No.

Certified that this is the bonafide record of work done by the above student in the...Laboratory during the academic year 2025- 2026

Signature of Faculty in-charge

Submitted for the Practical Examination held on……………………………

Internal Examiner **External Examiner**

INDEX

HOSPITAL PATIENT MANAGEMENT SYSTEM

A MINI-PROJECT REPORT

Submitted by

SARANYA V 240701620

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

NOVEMBER

BONAFIDE CERTIFICATE

Certified that this project **"HOSPITAL PATIENT MANAGEMENT SYSTEM"** is the bonafide work of **"SARANYA V"** who carried out the project work under my supervision.

SIGNATURE

B.DEEPA

ASSISTANT PROFESSOR SG

Dept. of Computer Science and Engg,
Rajalakshmi Engineering College
Chennai

This mini project report is submitted for the viva voce examination to be held on

____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

:

ABSTRACT

In our state, healthcare plays a major role. Even though there are various large hospital chains that operate on a larger scale, many local clinics and smaller hospitals are bereft of any such hassle-free application system for patient management. To address this drawback in the local healthcare system, our team developed a database system to help the local facilities maintain patient data and organize it efficiently. The main objective of this project is to manage patient appointments and records according to the patient's requirement. This system helps to maintain the confidentiality and details of patients in a local hospital or clinic. This will allow the facility to improve patient care by providing efficient service.

ACKNOWLEDGEMENT

We express our sincere thanks to our beloved and honorable chairman

MR. S. MEGANATHAN and the chairperson **DR. M.THANGAM MEGANATHAN** for their timely support and encouragement.

We are greatly indebted to our respected and honorable principal **Dr. S.N. MURUGESAN** for his able support and guidance.

No words of gratitude will suffice for the unquestioning support extended to us by our Head Of The Department **Dr. E.M. MALATHY** and our Deputy Head Of The Department **Dr. J. MANORANJINI** for being an ever supporting force during our project work.

We also extend our sincere and hearty thanks to our internal guide **B.DEEPA ,** for her valuable guidance and motivation during the completion of this project.

Our sincere thanks to our family members, friends and other staff members of computer science engineering.

SARANYA V

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The Hospital Patient Management System (HPMS) is a web-based application designed to digitize and streamline the process of managing patient and doctor records in a hospital or clinic. In an era where data is critical, moving from manual, paper-based systems to a secure, centralized digital system is essential for efficient healthcare delivery. This project helps hospital staff quickly access patient information, register new patients, and manage doctor lists, all through a user-friendly web interface.

1.2 SCOPE OF THE WORK

The system is a Java-based web application built on Servlet and JSP technology. It provides a secure login for administrators, who can then perform full CRUD (Create, Read, Delete) operations for patient records and (Create, Read) operations for doctor records. The data is stored in a persistent SQLite database, making it a complete, end-to-end solution for basic hospital administration.

1.3 PROBLEM STATEMENT

Many small-to-medium-sized hospitals and clinics still rely on manual, paper-based ledgers or standalone spreadsheets to manage patient information. This method is inefficient, prone to human error, insecure, and makes data retrieval extremely difficult. There is no central system for tracking patient history or staff records, leading to delays and potential data loss.

1.4 AIM AND OBJECTIVES OF THE PROJECT

The main aim of this project is to develop a secure, centralized, and user-friendly web application to manage hospital patient and doctor data.

The key objectives are:

- To create a secure admin login portal to prevent unauthorized access.
- To design a dynamic web dashboard for easy navigation.
- To implement modules for adding new patients, searching for patients, and deleting patient records.
- To create a module for adding new doctors and viewing all doctors.
- To ensure all data is stored persistently in a relational (SQLite) database.
- To use Java Servlets for backend logic and JSP for dynamically rendering the front-end.

CHAPTER 2

SYSTEM SPECIFICATIONS

2.1 HARDWARE SPECIFICATIONS

Processor	:	Intel i5
Memory Size	:	8GB (Minimum)
HDD	:	1 TB (Minimum)

2.2 SOFTWARE SPECIFICATIONS

TYPE	**SPECIFICATION**
Front – End	HTML5, CSS3, JavaScript, JavaServer Pages (JSP)
Back - End	Java
Data base	SQLite
Web Server	Apache Tomcat
IDE	Apache NetBeans

CHAPTER 3: MODULE DESCRIPTION

1. Authentication Module (`LoginServlet.java`)

This module handles the security and session management for the application.

- It receives the `username` and `password` from the `index.html` login form.
- It performs a check to verify the admin credentials (hardcoded as `admin` / `admin123`).
- On success, it creates a new `HttpSession`, stores the username, and redirects the user to the main `dashboard.jsp`.
- On failure, it redirects back to the login page with an error flag so the user can try again.

2. Patient Management Module (CRUD)

This is the core module for managing all patient data, using servlets and the `Patient.java` data model.

- **`AddPatientServlet.java`** – Receives new patient details (name, age, diagnosis, etc.) from the `add_patient.html` form. It uses a `PreparedStatement` to safely `INSERT` this data into the `patients` table in the database.

- **`PatientHistoryServlet.java`** – Fetches all records from the `patients` table using a `SELECT *` query. It stores these records in a `List<Patient>` and forwards this list to the `patient_history.jsp` page to be displayed in an HTML table.
- **`SearchPatientServlet.java`** – Receives a patient ID and name from the `search_patient.html` form. It queries the database using a `WHERE`

clause to find specific matches and forwards the resulting list to the same `patient_history.jsp` page for display.

- **`DeletePatientServlet.java`** – Receives a patient `id` as a URL parameter (when the user clicks "Delete" in the list). It executes a `DELETE` command using a `PreparedStatement` to safely remove the patient from the database and then redirects back to the main patient list.

3. Doctor Management Module

This module handles the staff records for doctors, using the `Doctor.java` data model.

- **`AddDoctorServlet.java`** – Receives a doctor's name and specialization from the `add_doctor.jsp` form and `INSERT`s them into the `doctors` table.
- **`DoctorRecordsServlet.java`** – Fetches all records from the `doctors` table, stores them in a `List<Doctor>`, and forwards the list to `doctor_records.jsp` to be displayed in a table.

4. Database Connection Module (`DBConnection.java`)

This is the foundational module that manages all database interactions.

- It contains the `getConnection()` method, which uses the SQLite JDBC driver to connect to the `hospital.db` file.
- Most importantly, it includes a `setupTables()` method that automatically runs `CREATE TABLE IF NOT EXISTS` for both `patients` and `doctors`. This ensures the database and tables are always ready, even on a new computer.
- It also includes a helper function to populate the `doctors` table with sample data if it's empty on startup.

CHAPTER 4: SAMPLE CODING

This chapter provides key code samples that demonstrate the core functionality of the application, including database connection, writing data, and reading data.

Sample 1: DBConnection.java (Database & Table Setup)

This code is the core of the application. It not only connects to the SQLite database but also automatically creates the patients and doctors tables if they don't already exist, making the application highly portable and easy to run.

JAVA

```
package com.mycompany.hpms;



import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.io.File;



public class DBConnection {



    // This path works for the local machine

    private static final String DB_PATH =
"C:/Users/kphar/Desktop/HPMS_DB/hospital.db";

    private static final String DB_URL = "jdbc:sqlite:" + DB_PATH;
```

```
public static Connection getConnection() {
    Connection conn = null;
    try {
        // ... (Folder creation) ...

        // Load SQLite driver
        Class.forName("org.sqlite.JDBC");
        conn = DriverManager.getConnection(DB_URL);

        if (conn != null) {
            System.out.println("Database connected successfully at: " + DB_PATH);
            // Call helper method to create tables if they don't exist
            setupTables(conn);
        }

    // ... (catch blocks) ...
    return conn;
}
```

```
private static void setupTables(Connection conn) {
    try (Statement stmt = conn.createStatement()) {

        // SQL to create patients table if it's missing
        String createPatients = "CREATE TABLE IF NOT EXISTS patients (" +
                "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "name TEXT NOT NULL, " +
                "age INTEGER, " +
                "gender TEXT, " +
                "diagnosis TEXT, " +
                "admission_date TEXT" +
                ");";

        // SQL to create doctors table if it's missing
        String createDoctors = "CREATE TABLE IF NOT EXISTS doctors (" +
                "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "name TEXT NOT NULL, " +
                "specialization TEXT" +
                ");";
```

```
            stmt.executeUpdate(createPatients);

            stmt.executeUpdate(createDoctors);



            System.out.println(" Verified: Tables 'patients' and 'doctors' exist.");

        } catch (SQLException e) {

            System.err.println("⚠️Table creation check failed: " + e.getMessage());

        }

    }

}
```

Sample 2: AddPatientServlet.java (Handling Form Data)

This servlet handles the POST request from the add_patient.html form 1111. It demonstrates how to retrieve form parameters 2and use a PreparedStatement to safely INSERT data into the database 3333, which prevents SQL Injection attacks.

```
// Inside AddPatientServlet.java

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse
response)

        throws ServletException, IOException {


    // 1. Get all the patient details from the HTML form

    String name = request.getParameter("patient_name"); [cite: 7]

    int age = Integer.parseInt(request.getParameter("patient_age")); [cite: 8]

    String gender = request.getParameter("patient_gender"); [cite: 9]
    String diagnosis = request.getParameter("patient_diagnosis"); [cite: 10]

    String admissionDate = request.getParameter("admission_date"); [cite: 11]


    Connection conn = DBConnection.getConnection(); [cite: 13]


    try {

        // 2. Create a safe SQL INSERT statement with placeholders (?)

        String sql = "INSERT INTO patients (name, age, gender, diagnosis, admission_date) VALUES (?, ?, ?, ?, ?)"; [cite: 17]


        // 3. Use PreparedStatement to safely insert data

        PreparedStatement pstmt = conn.prepareStatement(sql); [cite: 20]

        pstmt.setString(1, name); [cite: 21]

        pstmt.setInt(2, age); [cite: 22]

        pstmt.setString(3, gender); [cite: 23]

        pstmt.setString(4, diagnosis); [cite: 24]

        pstmt.setString(5, admissionDate); [cite: 25]


        // 4. Execute the statement

        pstmt.executeUpdate(); [cite: 28]

        conn.close(); [cite: 30]


    } catch (SQLException e) {

        e.printStackTrace(); [cite: 33]
```

```
   }

   // 5. Redirect back to the dashboard after success

   response.sendRedirect("dashboard.jsp"); [cite: 37]

}
```

Sample 3: PatientHistoryServlet.java (Forwarding Data to JSP)

This code shows the Model-View-Controller (MVC) pattern. The servlet (Controller) gets data from the database (Model), packages it as a List, and forwards it to the patient_history.jsp (View) to be displayed.

```
// Inside PatientHistoryServlet.java

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse
response)

      throws ServletException, IOException {


   List<Patient> patientList = new ArrayList<>();

   Connection conn = DBConnection.getConnection();


   try {

      // 1. Create a simple statement to select all patients

      Statement stmt = conn.createStatement();

      String sql = "SELECT * FROM patients";

      ResultSet rs = stmt.executeQuery(sql);


      // 2. Loop through the results and create Patient objects
```

```
        while(rs.next()){

            int id = rs.getInt("id");

            String name = rs.getString("name");

            int age = rs.getInt("age");

            String gender = rs.getString("gender");

            String diagnosis = rs.getString("diagnosis");

            String admissionDate = rs.getString("admission_date");


            // Add new Patient object to the list

            Patient patient = new Patient(id, name, age, gender, diagnosis,
admissionDate);

            patientList.add(patient);

        }

        conn.close();

    } catch (Exception e) {

        e.printStackTrace();

    }


    // 3. Set the list as an attribute for the JSP page to use

    request.setAttribute("patientList", patientList);


    // 4. Forward the request (with the data) to the JSP page

    RequestDispatcher dispatcher =
request.getRequestDispatcher("patient_history.jsp");

    dispatcher.forward(request, response); }
```

CHAPTER 5

SCREEN SHOTS

Fig 5.1 Introduction page

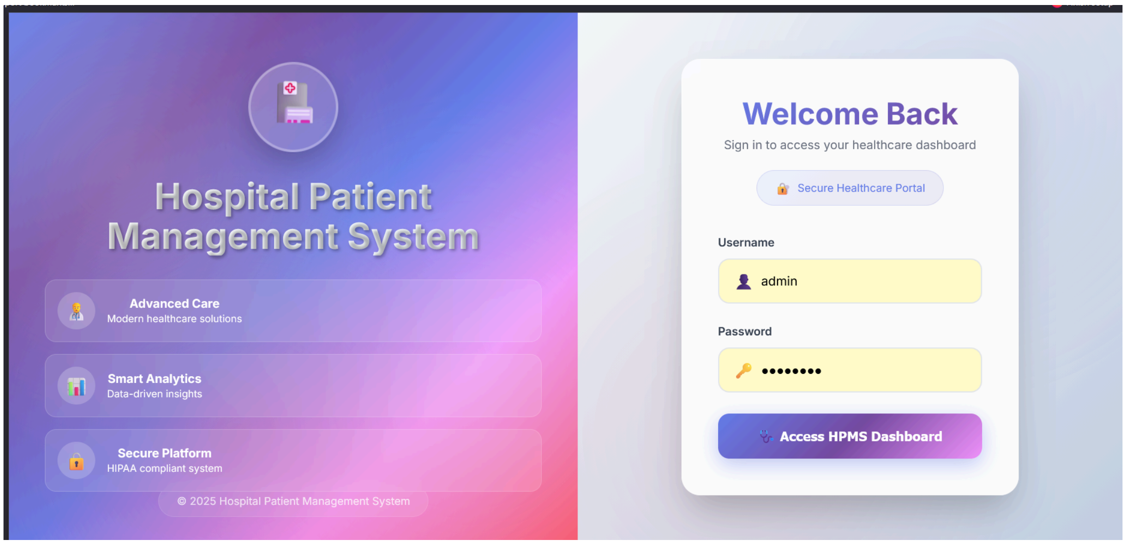


Fig 5.2 Main Dashboard

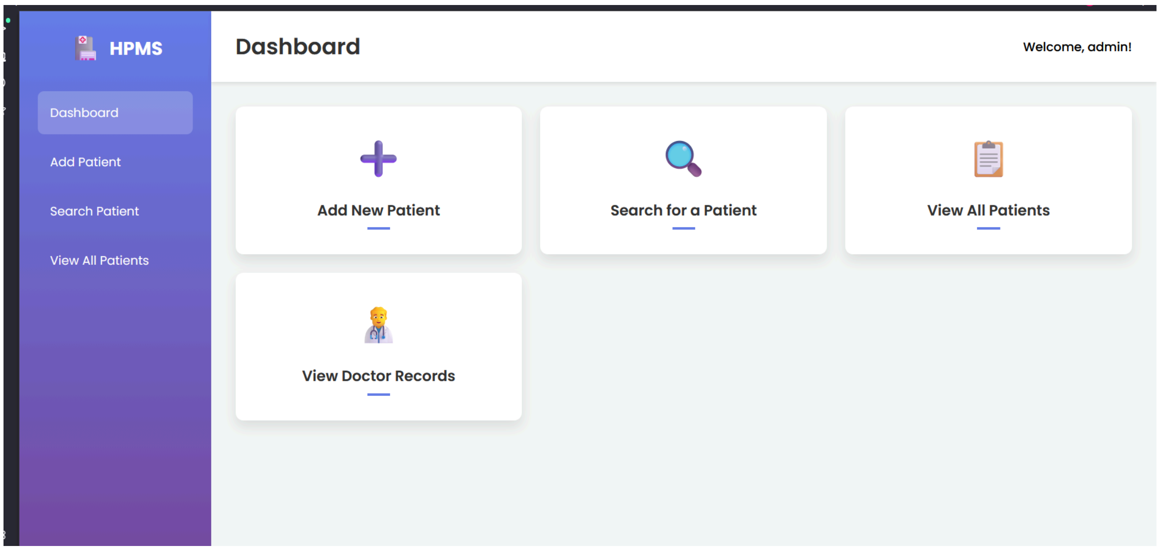
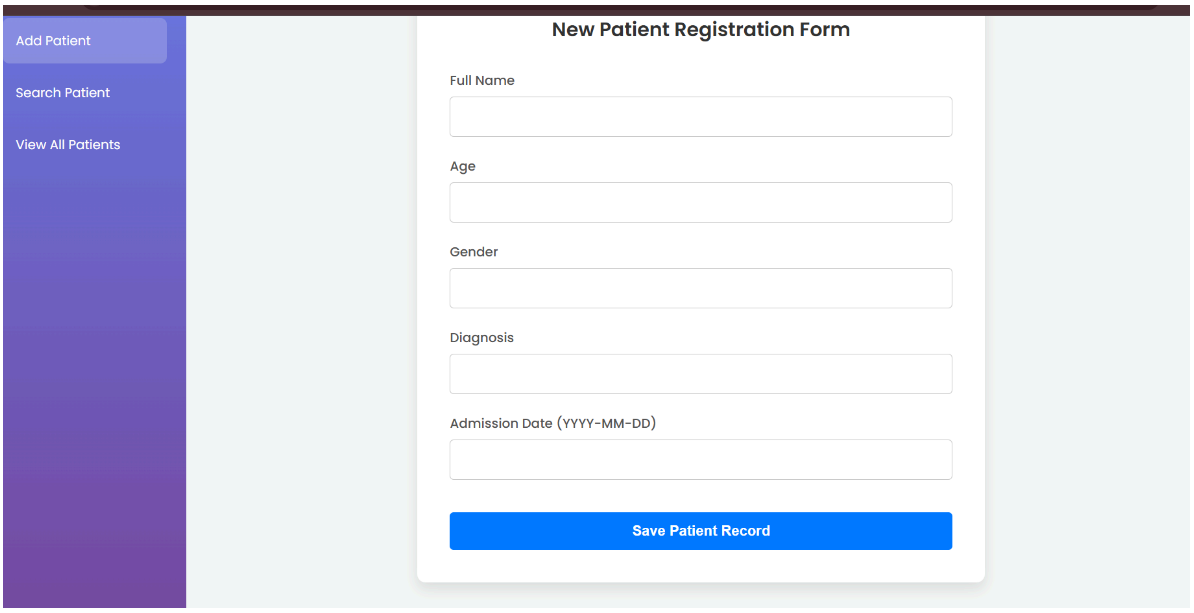


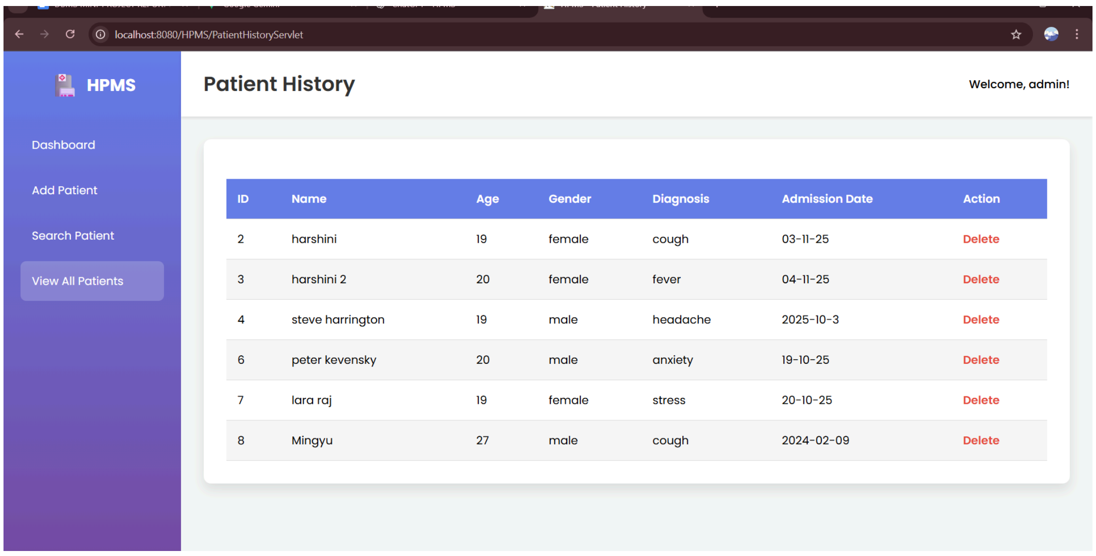
Fig 5.3: Add New Patient Form



New Patient Registration Form

Full Name	
Age	
Gender	
Diagnosis	
Admission Date (YYYY-MM-DD)	

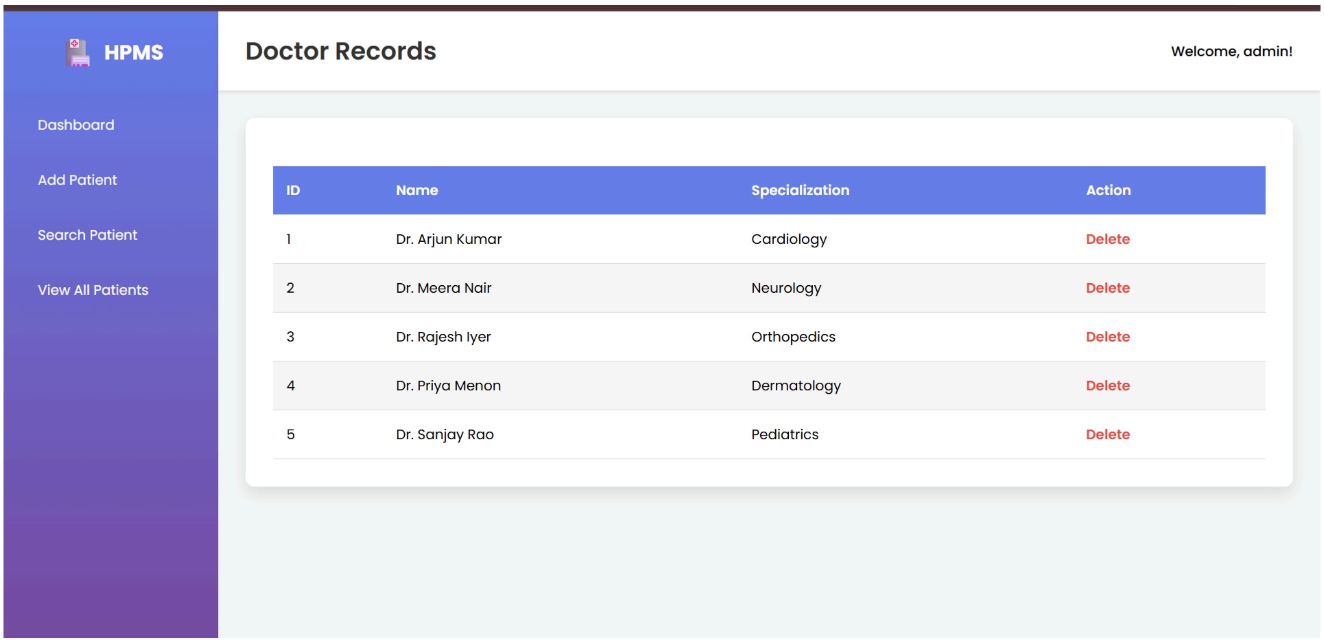
Fig 5.4 View All Patients Table



Patient History

ID	Name	Age	Gender	Diagnosis	Admission Date	Action
2	harshini	19	female	cough	03-11-25	Delete
3	harshini 2	20	female	fever	04-11-25	Delete
4	steve harrington	19	male	headache	2025-10-3	Delete
6	peter kevensky	20	male	anxiety	19-10-25	Delete
7	lara raj	19	female	stress	20-10-25	Delete
8	Mingyu	27	male	cough	2024-02-09	Delete

Fig 5.5 View All Doctors Table


HPMS

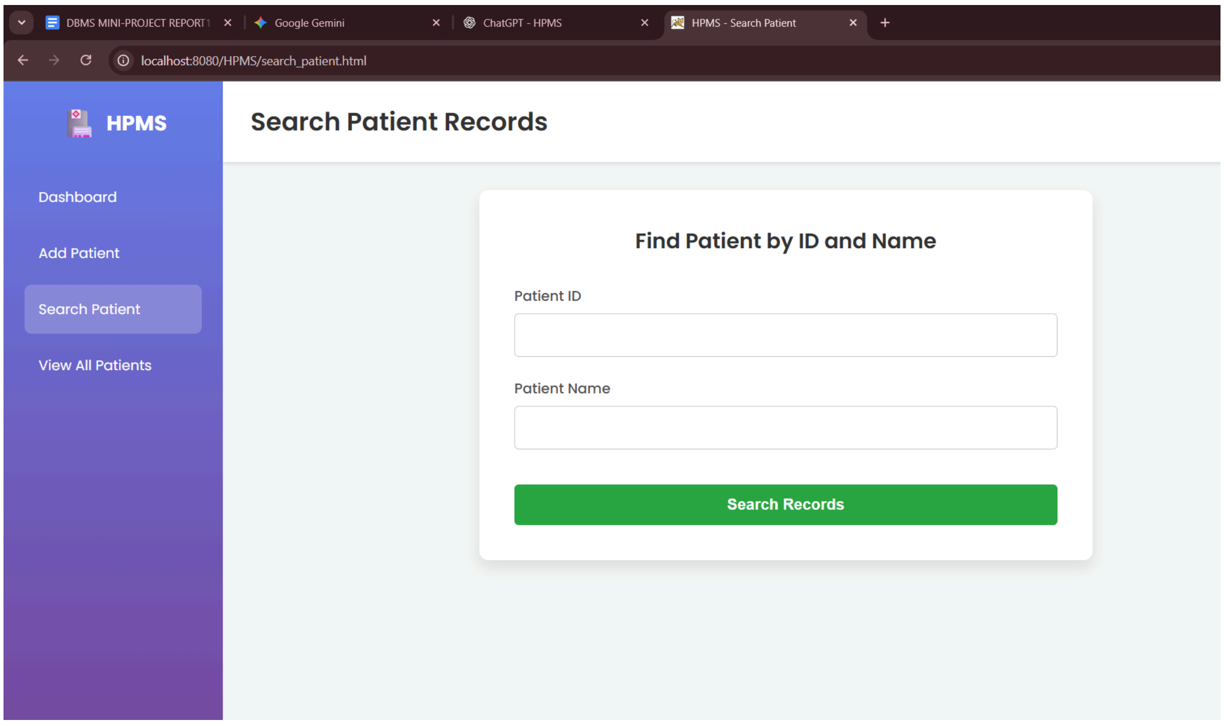
Dashboard
Add Patient
Search Patient
View All Patients

Doctor Records

Welcome, admin!

ID	Name	Specialization	Action
1	Dr. Arjun Kumar	Cardiology	Delete
2	Dr. Meera Nair	Neurology	Delete
3	Dr. Rajesh Iyer	Orthopedics	Delete
4	Dr. Priya Menon	Dermatology	Delete
5	Dr. Sanjay Rao	Pediatrics	Delete

Fig 5.6: Search Patient Page



HPMS

Dashboard
Add Patient
Search Patient
View All Patients

Search Patient Records

Find Patient by ID and Name

Patient ID

Patient Name

Search Records

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION

This project successfully developed a web-based Hospital Patient Management System using Java Servlets, JSP, and an SQLite database. The application runs on an Apache Tomcat server and provides a secure, centralized, and efficient way to manage patient and doctor records. By moving from a manual system to a digital one, this project solves the problem of data inaccuracy, slow retrieval, and insecurity. The system provides core administrative functions (CRUD) through a modern, user-friendly interface.

FUTURE ENHANCEMENT

While the core functionality is complete, the project can be expanded in several key areas:

Patient Login: Create a separate login for patients to view their own appointment history or book new appointments.

Enhanced Doctor Module: Allow admins to *delete* or *update* doctor records, not just add and view them.

Billing Module: Add a billing section to manage patient invoices and payment history.

Role-Based Access: Create different user roles (e.g., `admin`, `receptionist`, `doctor`) with different permissions.

Cloud Deployment: Migrate the database from a local file (SQLite) to a cloud-based database (like PostgreSQL or MySQL) and deploy the application on a public cloud host (like Heroku or AWS) to make it accessible from anywhere.

REFERENCES

- https://www.w3schools.com/html/
- https://www.w3schools.com/css/
- https://www.w3schools.com/java/
- https://docs.oracle.com/javaee/7/api/javax/servlet/package-summary.html
- https://www.tutorialspoint.com/jsp/index.htm
- https://www.sqlitetutorial.net/
- https://www.geeksforgeeks.org/java-servlet-technology/
- https://www.javatpoint.com/servlet-tutorial