**HOSPITAL MANAGEMENT SYSTEM**

**MINI PROJECT REPORT**

SUBMITTED BY

Bernieo Fatim. A 230701051

P. Brijith Manikandan 230701058

*In partial fulfilment for the award of the degree*

BACHELOR OF ENGINEERING

IN COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE(AUTONOMOUS)

THANDALAM

CHENNAI-602105

2024- 2025

**BONAFIDE CERTIFICATE**

Certified that this project report

**“HOSPITAL MANAGEMENT SYSTEM”**

is the bonafide work of

**“Bernieo Fatim. A (230701051),**

**P. Brijith Manikandan (230701058)”**

who carried out the project work under my supervision.

Submitted for the Practical Examination held on 23.11.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ABSTRACT**

This project on Hospital Management System is designed to automate the administrative and management processes within a hospital. The system provides functionality to manage patient information, doctor details, appointment schedules, billing, and other essential hospital operations. It allows for storing, retrieving, and managing data related to hospital services and transactions. The system also enables the addition of records when a new patient is admitted or an appointment is scheduled. For data storage and retrieval, we use a MySQL Database, which allows us to manage a large number of records efficiently. The project "Hospital Management System" includes various components, each with specific roles:

* Hospital Administration
* Doctors and Medical Staff
* Patients/Visitors

Each of these accesses a database schema with corresponding tables to support their operations.

Language Used:- Java Core

Concept Used:- Swing

IDE Used:- Visual Studio Code

Database Used:- MySQL

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTERS** |  | **PAGE NO** |
| Chapter 1 | Introduction  1.1Problem Definition  1.2 Need | 1  2 |
| Chapter 2 | Requirements  2.1 Software Requirement Specifications  2.2 Hardware Requirement Specifications | 3    3 |
| Chapter 3 | Entity Relationship Diagram  3.1 Entity relationship diagram 5 | 4 |
| Chapter 4 | Schema Diagram 6  4.1 Schema diagram 7 |  |
| Chapter 5 | Implementation  5.1 Backend Implementation 8  5.2 Frontend implementation 9  5.3 Creating mainframe class 10-13 |  |
| Chapter 6 | Snapshots 14-19 |  |
|  | Conclusion |  |
|  | References |  |

**CHAPTER 1**

**INTRODUCTION**

The Hospital Management System is designed to improve and automate the management and administration of hospital operations. It involves activities that streamline hospital workflows and enhance patient care by centralizing information on medical and operational priorities. The system provides a comprehensive overview of hospital operations, creating awareness of factors influencing hospital management, such as resource allocation, patient flow, and financial aspects. This field of study also encompasses key principles of healthcare administration, emphasizing the operational, financial, and strategic factors that impact healthcare delivery. The system introduces automation to hospital management, contributing to efficiency and improved patient experiences, and reflects a commitment to adopting advanced technologies.

**1.1 Problem Definition**

This project on Hospital Management System automates the processes related to patient registration, appointment scheduling, and hospital records management. It provides information on patient details, doctor assignments, and treatment records, enabling efficient record addition, modification, and deletion based on patient needs. For data storage and retrieval, the system uses a MySQL database, integrated with a Java-based frontend, to ensure seamless updates across the system. Any changes made in the frontend are directly reflected in the backend database.

**1.2 Need**

The system provides an electronic means of handling patient and hospital records, enhancing accuracy, flexibility, reliability, and reducing human errors. Hospitals deliver a wide range of healthcare services, including patient consultations, treatments, and follow-ups. This system supports accurate information handling for patient admission, discharge, and records management. It aims to deliver a robust, efficient, and scalable solution capable of handling any number of records. The healthcare industry is evolving rapidly, and with it, the need for systems that can support high volumes of data while maintaining precision and efficiency.

Key reasons for developing the system include:

1. Faster system

2. Accuracy

3. Reliability

4. Informative

5. Remote patient appointments and updates

**CHAPTER 2**

**REQUIREMENTS**

**2.1 Software Requirement Specifications**

Operating System Front End Back End Server Documentation : Windows 10

Frontend Software: Java NetBeans 8.2 : JDK 8

Backend Software: MySQL

**2.2 Hardware Requirement Specifications**

Computer Processor Core i3 Processor Speed 2.3 GHz Processor Hard Disk 400 GB or more RAM Min 2GB

**CHAPTER 3**

**ENTITY RELATIONSHIP DIAGRAM**

An entity-relationship (ER) diagram is a specialized graphic that illustrates the interrelationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes. If the application is primarily a database application, the entity-relationship approach can be used effectively for modeling some parts of the problem. The main focus in ER modeling is the Data Items in the system and the relationship between them. It aims to create conceptual scheme for the Data from the user’s perspective. The model thus created is independent of any database model. The ER models are frequently represented as ER diagram. Here we present the ER diagram of the above mentioned project.



**CHAPTER 4**

**SCHEMA DIAGRAM**

**4.1 SCHEMA DIAGRAM**

A database schema is the skeleton structure that represents the logical view of the entire database. A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It’s the database designers who design the schema to help programmers understand the database and make it useful.

A database schema can be divided broadly into two categories −

* Physical Database Schema − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
* Logical Database Schema − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

**CHAPTER 5**

**IMPLEMENTATION**

**5.1 Backend Implementation**

**MYSQL**

MySQL is an open-source relational database management system (RDBMS). A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

**Table Hospital:**

CREATE TABLE hospital (

patient\_id INT PRIMARY KEY AUTO\_INCREMENT,

patient\_name VARCHAR(100) NOT NULL,

age INT NOT NULL,

diagnosis VARCHAR(255),

doctor\_name VARCHAR(100),

specialty VARCHAR(50)

);

**5.2 Frontend Implementation**

**Java Core**

Core Java is the part of Java programming language that is used for creating or developing a general-purpose application. It uses only one tier architecture that is why it is called as ‘stand alone’ application.Core java programming covers the swings, socket, awt, thread concept, collection object and classess.

**Swings**

**Swing** is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC) – an API for providing a graphical user interface (GUI) for Java programs.

Swing provides a look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

**5.3 Creating mainframe class**

import javax.swing.\*;

import java.awt.\*;

import java.sql.\*;

public class HospitalManagement extends JFrame {

private static final String URL = "jdbc:mysql://localhost:3306/test";

private static final String USER = "root";

private static final String PASSWORD = "Brijith123$$$";

private JTextField patientNameField, ageField, diagnosisField, doctorNameField, specialtyField;

private JTextArea outputArea;

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

try {

new HospitalManagement().setVisible(true);

} catch (SQLException e) {

e.printStackTrace();

}

});

}

public HospitalManagement() throws SQLException {

setTitle("Hospital Management System");

setSize(600, 600);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setLayout(new BorderLayout());

JPanel inputPanel = new JPanel(new GridLayout(5, 2));

// Input fields

patientNameField = new JTextField();

ageField = new JTextField();

diagnosisField = new JTextField();

doctorNameField = new JTextField();

specialtyField = new JTextField();

inputPanel.add(new JLabel("Patient Name:"));

inputPanel.add(patientNameField);

inputPanel.add(new JLabel("Age:"));

inputPanel.add(ageField);

inputPanel.add(new JLabel("Diagnosis:"));

inputPanel.add(diagnosisField);

inputPanel.add(new JLabel("Doctor Name:"));

inputPanel.add(doctorNameField);

inputPanel.add(new JLabel("Specialty:"));

inputPanel.add(specialtyField);

// Buttons for CRUD operations

JPanel buttonPanel = new JPanel();

JButton addButton = new JButton("Add Record");

JButton viewButton = new JButton("View Records");

JButton deleteButton = new JButton("Delete Record");

JButton updateButton = new JButton("Update Record"); // Renamed Update Button

buttonPanel.add(addButton);

buttonPanel.add(viewButton);

buttonPanel.add(deleteButton);

buttonPanel.add(updateButton); // Add renamed Update Button to panel

// Output area for results

outputArea = new JTextArea(10, 40);

outputArea.setEditable(false);

JScrollPane scrollPane = new JScrollPane(outputArea);

add(inputPanel, BorderLayout.NORTH);

add(buttonPanel, BorderLayout.CENTER);

add(scrollPane, BorderLayout.SOUTH);

addButton.addActionListener(e -> addRecord());

viewButton.addActionListener(e -> viewRecords());

deleteButton.addActionListener(e -> deleteRecord());

updateButton.addActionListener(e -> updateRecord()); // Add ActionListener for renamed Update Button

}

private Connection getConnection() throws SQLException {

return DriverManager.getConnection(URL, USER, PASSWORD);

}

private void addRecord() {

String patientName = patientNameField.getText();

int age = Integer.parseInt(ageField.getText());

String diagnosis = diagnosisField.getText();

String doctorName = doctorNameField.getText();

String specialty = specialtyField.getText();

String sql = "INSERT INTO hospital (patient\_name, age, diagnosis, doctor\_name, specialty) VALUES (?, ?, ?, ?, ?)";

try (Connection conn = getConnection(); PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, patientName);

stmt.setInt(2, age);

stmt.setString(3, diagnosis);

stmt.setString(4, doctorName);

stmt.setString(5, specialty);

stmt.executeUpdate();

outputArea.setText("Record added successfully!");

} catch (SQLException e) {

e.printStackTrace();

outputArea.setText("Error adding record.");

}

}

private void viewRecords() {

String sql = "SELECT \* FROM hospital";

StringBuilder output = new StringBuilder();

try (Connection conn = getConnection(); Statement stmt = conn.createStatement(); ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

output.append("Patient ID: ").append(rs.getInt("patient\_id")).append(", ")

.append("Name: ").append(rs.getString("patient\_name")).append(", ")

.append("Age: ").append(rs.getInt("age")).append(", ")

.append("Diagnosis: ").append(rs.getString("diagnosis")).append(", ")

.append("Doctor Name: ").append(rs.getString("doctor\_name")).append(", ")

.append("Specialty: ").append(rs.getString("specialty")).append("\n");

}

outputArea.setText(output.toString());

} catch (SQLException e) {

e.printStackTrace();

outputArea.setText("Error retrieving records.");

}

}

private void deleteRecord() {

String patientId = JOptionPane.showInputDialog(this, "Enter Patient ID to delete:");

String sql = "DELETE FROM hospital WHERE patient\_id = ?";

try (Connection conn = getConnection(); PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, Integer.parseInt(patientId));

int rowsAffected = stmt.executeUpdate();

if (rowsAffected > 0) {

outputArea.setText("Record deleted successfully!");

} else {

outputArea.setText("No record found with Patient ID " + patientId);

}

} catch (SQLException e) {

e.printStackTrace();

outputArea.setText("Error deleting record.");

}

}

// New method to update doctor name for a given patient ID

private void updateRecord() {

String patientId = JOptionPane.showInputDialog(this, "Enter Patient ID to update:");

String newDoctorName = JOptionPane.showInputDialog(this, "Enter new Doctor Name:");

String sql = "UPDATE hospital SET doctor\_name = ? WHERE patient\_id = ?";

try (Connection conn = getConnection(); PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, newDoctorName);

stmt.setInt(2, Integer.parseInt(patientId));

int rowsAffected = stmt.executeUpdate();

if (rowsAffected > 0) {

outputArea.setText("Doctor name updated successfully for Patient ID " + patientId);

} else {

outputArea.setText("No record found with Patient ID " + patientId);

}

} catch (SQLException e) {

e.printStackTrace();

outputArea.setText("Error updating record.");

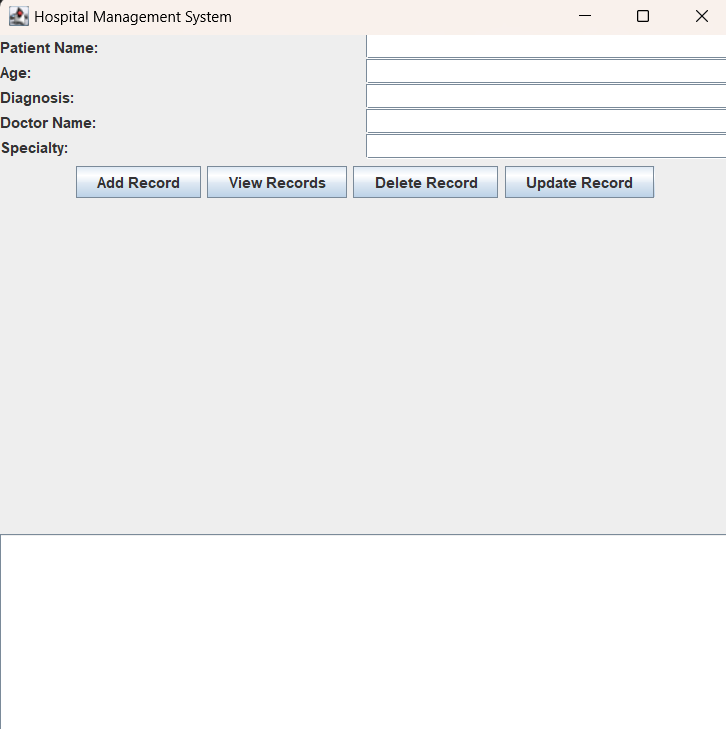
}

}

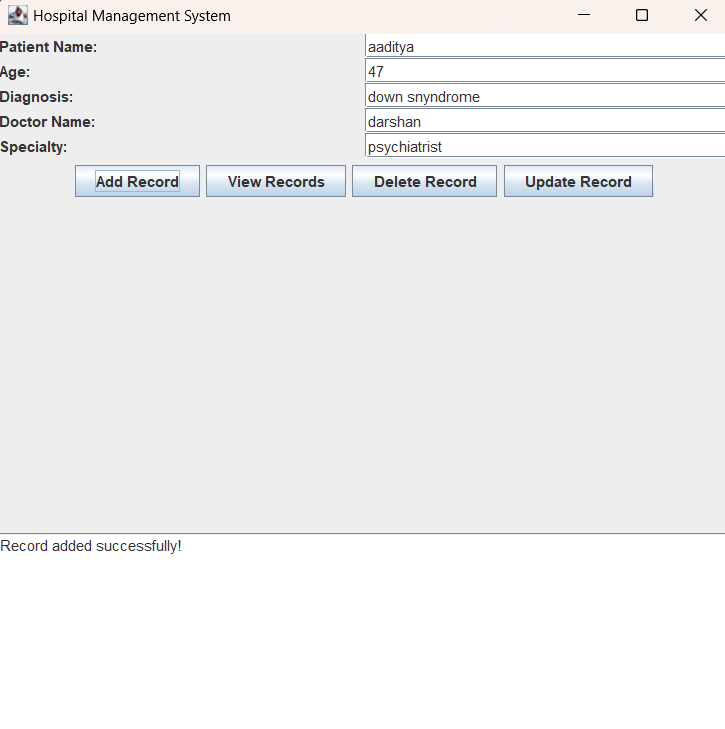
}

**CHAPTER 6**

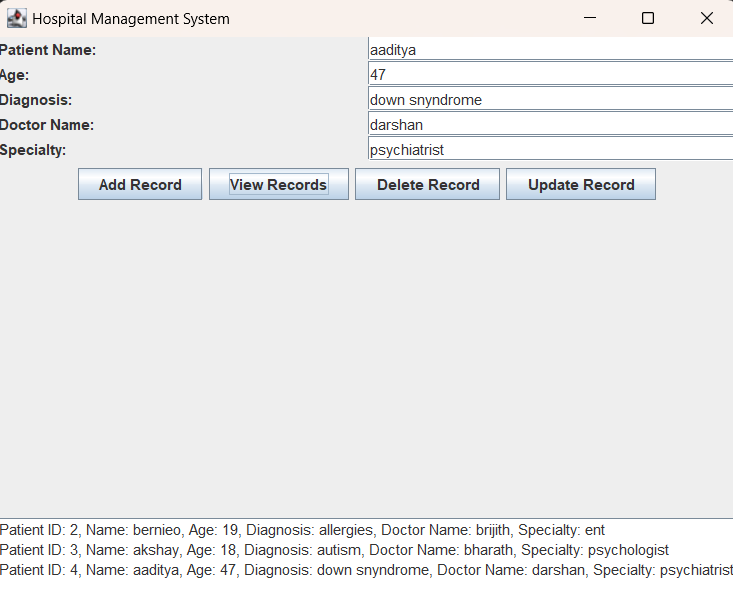
**SNAPSHOTS**



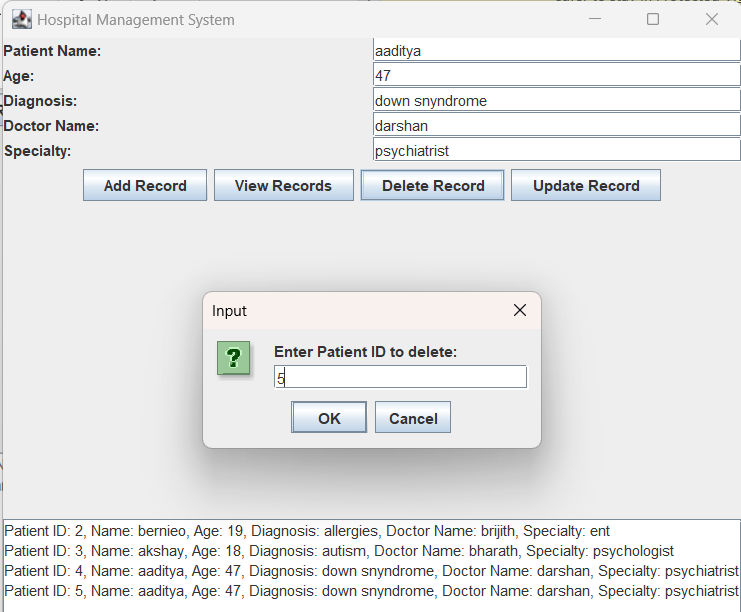
**Fig 6.1: Mainframe**

****

**Fig 6.2 Inserting Patient Details**

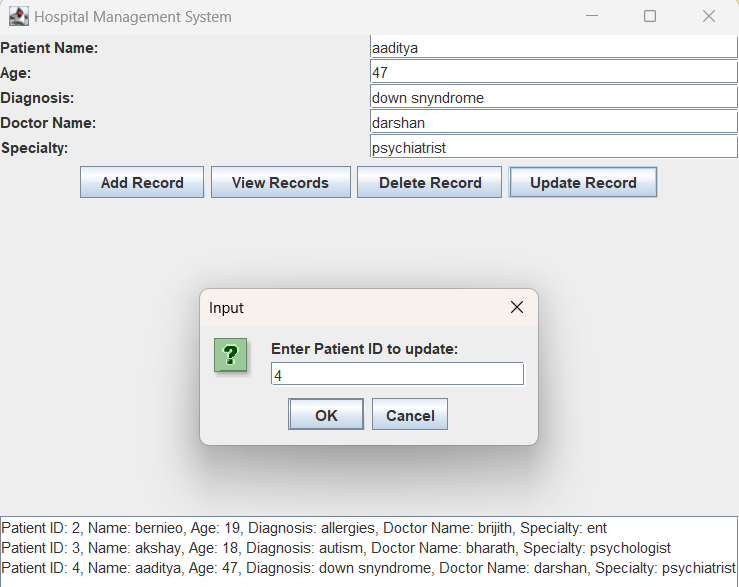
****

**Fig 6.3: Viewing Patient Details**

****

****

**Fig 6.3: Deleting Patient Records**

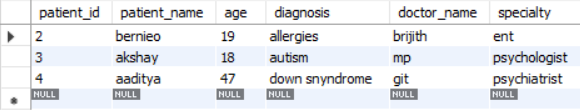
****

****

**Fig 6.4: Updating Records**

**CHAPTER 7**

**Backend (MySQL):**



**Fig 7.1: Updating Records**

**CONCLUSION**

This project on Hospital Management System automates the patient registration and management processes within a hospital setting. The system provides comprehensive information, including patient details, diagnosis records, assigned doctors, and medical specialties. It enables users to add records when a patient is admitted or when a new consultation is recorded. Additionally, it offers functionalities to view, delete, and update records based on patient requirements or medical updates. This project has enhanced our understanding of various aspects of computer science, especially the importance of developing robust applications for managing essential data in a healthcare environment.

**REFERENCES**

**Websites and Blogs:**

* GeeksforGeeks - Java Programming - https://www.geeksforgeeks.org/java/
* Stack Overflow for troubleshooting Java and SQL-related queries - [https://stackoverflow.com](https://stackoverflow.com/)

**YouTube Tutorials:**

* Java and JDBC Tutorial: [Java JDBC Tutorial for Beginners](https://www.youtube.com/watch?v=H1l2g8imMc8)
* MySQL Tutorials: [MySQL Database Management System Tutorials](https://www.youtube.com/playlist?list=PL4cUxeGkcC9gQrxTZHT4oBZHK_MsJOpRf)