

HOSPITAL MANEGEMENT SYSTEM  
A MINI PROJECT REPORT

Submitted by

GANESAN G

230701089

HARSHA VARDHANAN V

230701108

In partial fulfillment for the award of the degree of  
BACHELOR OF ENGINEERING  
IN  
COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)  
THANDALAM  
CHENNAI-602105

2023 - 24

RAJALAKSHMI ENGINEERING COLLEGE  
CHENNAI

BONAFIDE CERTIFICATE

Certified that this project report “ HOSPITAL MANAGEMENT SYSTEM ” is  
the bonafide work of GANESAN G (230701089), HARSHAVARDHANAN V  
(230701108)” who carried out the project work under my supervision.

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

SIGNATURE

Mrs.Divya.M  
Assistant Professor,  
Computer Science and Engineering,  
Rajalakshmi Engineering College,  
Thandalam,Chennai 602105

SIGNATURE

Mr.Ragu  
Assistant Professor,  
Computer Science and Engineering,  
Rajalakshmi Engineering College,  
Thandalam,Chennai 602105

INTERNAL EXAMINER

EXTERNAL EXAMIER

## ABSTRACT:

The Hospital Management System is a comprehensive, user-friendly graphical interface application designed to streamline and automate various aspects of hospital administration and patient care management. This system effectively manages essential hospital operations including patient registration, appointment scheduling, room allocation, and billing processes. Through an intuitive tabbed interface, the system provides seamless access to patient information, hospital services, and financial management tools. The system features robust patient management capabilities, allowing staff to add new patients, update their information, and maintain detailed medical records including symptoms, admission dates, and discharge information. The hospital services module efficiently handles doctor-patient appointments, ensuring optimal resource allocation and preventing scheduling conflicts through automated availability checks. Additionally, the system incorporates a sophisticated room management system that tracks room assignments and availability in real-time. The billing module provides comprehensive financial management tools, enabling automated bill generation, payment processing, and financial record-keeping. The system maintains detailed billing histories and supports multiple payment methods, ensuring efficient financial operations. By integrating these various components into a single, cohesive interface, the Hospital Management System significantly improves operational efficiency, reduces administrative overhead, and enhances the quality of patient care delivery. The system's database-driven architecture ensures data consistency and reliability, while its modular design allows for easy maintenance and future expansions. This modernized approach to hospital management helps healthcare providers focus more on patient care while maintaining accurate and accessible administrative records.

## TABLE OF CONTENTS

### Chapter 1

#### 1 INTRODUCTION

1.1	INTRODUCTION	6
1.2	OBJECTIVES	7
1.3	MODULES	7

### Chapter 2

#### 2 SURVEY OF TECHNOLOGIES

2.1	SOFTWARE DESCRIPTION	9
2.2	LANGUAGES	9
2.2.1	JAVA	9
2.2.2	MYSQL	9

### Chapter 3

#### 3 REQUIREMENTS AND ANALYSIS

3.1	REQUIREMENT SPECIFICATION	11
3.1.1	FUNCTIONAL REQUIREMENTS	11
3.1.2	NON FUNCTIONAL REQUIREMENTS	12
3.2	HARDWARE AND SOFTWARE REQUIREMENTS	13
3.3	ARCHITECTURE DIAGRAM	14
3.4	ER DIAGRAM	15
3.5	NORMALIZATION	16

### Chapter 4

#### 4 PROGRAM CODE

4.1	PROGRAM CODE	18
-----	--------------	----

### Chapter 5

#### 5 RESULTS AND DISCUSSION

5.1	RESULTS AND DISCUSSION	27
-----	------------------------	----

## Chapter 6

6 CONCLUSION \_\_\_\_\_

6.1 CONCLUSION-----31

## Chapter 7

7 REFERENCES \_\_\_\_\_

7.1 REFERENCES-----32

## 1.1 INTRODUCTION

Hospital management is a complex and critical aspect of healthcare delivery, requiring efficient coordination of various departments, staff, patients, and resources. The Hospital Management System is a sophisticated software solution developed to address these challenges by automating and streamlining hospital administrative operations, patient care management, and financial processes. This system serves as a comprehensive platform that integrates various hospital functions into a single, cohesive interface, ensuring smooth operations and enhanced patient care delivery.

The Hospital Management System offers an extensive range of features designed to meet the diverse needs of hospital staff and administrators. At its core, the system maintains a centralized database of patient information, including medical histories, admission details, treatment records, and billing information. This organized repository enables healthcare providers to access and update patient information efficiently, leading to better-informed medical decisions and improved patient care.

Developed using Java and Swing framework for the graphical user interface, along with SQL for database management, the system leverages these technologies to create a robust and user-friendly platform. The Java programming language provides the foundation for the application's logic and functionality, while Swing components create an intuitive and responsive user interface. SQL handles the backend database operations, ensuring efficient storage and retrieval of hospital data, including patient records, appointment schedules, room allocations, and financial transactions.

The system is structured into three main modules: Patient Management, Hospital Services, and Billing. The Patient Management module handles patient registration, information updates, and record maintenance. The Hospital Services module manages doctor appointments, room allocations, and other hospital resources. The Billing module oversees financial transactions, including bill generation, payment processing, and financial record-keeping. These modules work in concert to provide a seamless experience for hospital staff while maintaining data accuracy and accessibility.

This report will elaborate on the development process, system architecture, and the technologies employed in creating the Hospital Management System. The objective is to demonstrate how the integration of these technologies results in a comprehensive solution that effectively addresses the challenges of modern hospital management, ultimately leading to improved operational efficiency and enhanced patient care delivery.

## 1.2 OBJECTIVES

- To create a centralized database for managing patient information, appointments, and medical records.
- To enable seamless communication between patients, doctors, and medical staff, improving coordination and response times.
- To provide real-time information on the availability of rooms, medical equipment, and healthcare resources across various departments.
- To ensure compliance with health regulations, medical standards, and patient privacy laws (such as HIPAA).
- To improve operational efficiency by automating key processes like appointment scheduling, billing, and patient admission, reducing manual errors.
- To enhance the overall patient experience by providing easy access to medical services, appointments, and healthcare updates through a user-friendly interface.

## 1.3 MODULES

### 1. Patient Registration Module

The Patient Registration Module is designed to capture and store comprehensive information about each patient entering the hospital system. This module collects essential data including the patient's personal details (name, age, gender), contact information, emergency contacts, medical history, insurance details, and previous hospital visits. It ensures that all patient information is accurately recorded and securely stored in the database for future reference and continuity of care.

### 2. Patient Search and Room Allocation Functionality

The Patient Search and Room Allocation Functionality is a critical component for efficiently managing patient locations and room assignments within the hospital. This module enables hospital staff to quickly locate patients and available rooms based on various criteria. It provides real-time updates on room occupancy and patient status, helping administrators manage hospital resources effectively and ensuring appropriate accommodation for all patients.

### 3. Doctor-Patient Communication Module

The Doctor-Patient Communication Module facilitates seamless interaction between healthcare providers and patients. It manages all communication channels and appointment scheduling, enabling efficient coordination of medical services. This module supports various communication methods and maintains detailed records of all patient interactions.

### 4. Patient Registration and Doctor Coordination Module

The Patient Registration and Doctor Coordination Module is designed to register patients seeking medical care and assign them to appropriate doctors. This module collects detailed information about patients, including their medical history, symptoms, and personal details. It then uses this information to match them with suitable specialists or general practitioners from the available pool of doctors. The module also tracks and schedules patient appointments, ensuring that patients are seen in a timely manner based on their condition's severity. By streamlining patient registration and ensuring proper doctor coordination, this module helps enhance the quality of care and improves hospital workflow efficiency.

### 5 Admin Dashboard Module

The Admin Dashboard Module serves as a centralized hub for administrators to manage all aspects of the blood donation system. This module provides a comprehensive overview of donor registrations, blood inventory levels, recipient information, and ongoing communications. It includes tools for generating reports, monitoring system performance, and tracking key metrics such as donation rates and blood usage. The dashboard is designed with an intuitive interface, allowing administrators to easily navigate through different sections, access detailed records, and perform administrative tasks efficiently. It also features role-based access control to ensure that sensitive information is only accessible to authorized personnel.

### 6 Database Module

The Database Module serves as the backbone of the Hospital Management System, responsible for storing, managing, and securing all data related to hospital operations, including patients, staff, appointments, room allocations, billing, and medical records. This module ensures data integrity, scalability, and efficient retrieval through structured queries and optimized indexing.



## Chapter 2 SURVEY OF TECHNOLOGIES

### 2.1 SOFTWARE DESCRIPTION

The Hospital Management System leverages a combination of cutting-edge technologies to deliver a comprehensive, efficient, and user-friendly solution. The backend is powered by a robust relational database management system (RDBMS) to store and manage critical data securely. The frontend is designed with an intuitive graphical user interface (GUI) to facilitate seamless interaction for end-users. Middleware technologies are employed to ensure smooth communication and data exchange between the backend and frontend components.

### 2.2 LANGUAGES

The system is primarily developed using Java as the core programming language for backend processing, enabling robust functionality and seamless integration with the database. Java Swing is utilized for creating a dynamic and user-friendly graphical user interface (GUI), while SQL is employed for efficient database management and queries.

#### 2.2.1 JAVA

**Role:** Java serves as the primary programming language used for backend logic and interface development in the Hospital Management System.

**Usage:** Java is used in conjunction with Swing to build a dynamic and user-friendly graphical user interface (GUI). The application interacts seamlessly with the database using SQL for real-time data retrieval, updates, and management. Java's platform-independent nature ensures smooth execution across various operating systems.

**Advantages:**

- **Database Connectivity :** Java provides robust support for database operations through JDBC, enabling seamless integration with SQL databases for efficient data management.
- **Object-Oriented Design:** Java's object-oriented nature facilitates the development of modular, scalable, and maintainable code.

#### 2.2.2 SQL

**Role:** SQL (Structured Query Language) is used for managing and manipulating the relational database that stores all data.

**Usage:** All information related to donors, blood inventory, and transactions in the system are stored and implemented as a relational schema using SQL.

**Advantages:**

- **Efficient Data Management :** SQL allows for efficient querying, updating, and management of large datasets.

### 2.2.3 Java Swing

**Role:** Java Swing is the graphical user interface (GUI) toolkit used to design and create the user interface of the Hospital Management System.

**Usage:** Swing structures the interface, enabling the embedding of components such as buttons, forms, tables, and menus. It provides a flexible framework for creating interactive and user-friendly desktop applications.

**Advantages:**

- **Compatibility:** Java Swing is platform-independent, ensuring that the application can run on any operating system with Java Runtime Environment (JRE).
- **Customization:** Swing offers a wide range of customizable components and layouts, allowing developers to create visually appealing and functional interfaces.

### 2.2.4 JDBC (Java Database Connectivity)

**Role:** JDBC (Java Database Connectivity) is used to establish a seamless connection between the Java-based application and the SQL database.

**Usage:** JDBC facilitates communication between the application and the database, enabling real-time execution of SQL queries for retrieving, updating, and managing data such as patient records, appointments, and billing details.

**Advantages:**

- **Efficient Data Handling:** JDBC provides an efficient mechanism for executing complex queries and handling large data sets.
- **Cross-Platform Support:** As part of Java, JDBC ensures compatibility with multiple database systems, offering flexibility in deployment.
- **Security:** JDBC supports secure database transactions by leveraging Java's robust security features.

## Chapter 3      REQUIREMENTS AND ANALYSIS

### 3.1 REQUIREMENT SPECIFICATION

#### 3.1.1 Functional Requirements

##### User Authentication and Authorization

- User Registration and Login: Allow hospital staff and administrators to register, create accounts, and log in securely.
- Role-Based Access Control: Assign specific permissions based on user roles (e.g., doctor, nurse, admin, receptionist).

##### Patient Management Module

- Patient Registration and Profile Management: Enable staff to register patients and update their personal and medical information.
- Medical History Tracking: Record and update detailed medical histories, including allergies, previous treatments, and ongoing medications.

##### Doctor Appointment Scheduling

- Schedule Creation and Management: Allow doctors to set availability and staff to book, reschedule, or cancel patient appointments.
- Notifications: Send reminders for upcoming appointments to patients and doctors.

##### Room and Resource Allocation

- Room Assignment: Enable staff to allocate hospital rooms based on patient needs and availability.
- Resource Management: Track the availability of medical equipment and other hospital resources.

##### Billing and Finance Module

- Invoice Generation: Automatically create detailed invoices for treatments and services provided.
- Payment Processing: Allow secure handling of payments and maintain transaction history.

## Admin Dashboard Module

- **Comprehensive Overview:** Provide an overview of patient registrations, room occupancy, appointment schedules, and financial summaries.
- **Management Tools:** Include tools for generating operational reports, monitoring system performance, and tracking key metrics such as patient inflow and resource utilization.

### 3.1.2 Non-Functional

#### Requirements Security

- **Data Encryption:** Ensure sensitive patient and hospital data is encrypted both during transmission and while stored in the database.
- **Compliance:** Adhere to applicable regulations and standards for health data privacy and security, such as HIPAA or GDPR.

#### Performance

- **Scalability:** Design the system to efficiently support an increasing number of patients, staff, and operational transactions.
- **Response Time:** Ensure quick response times for user interactions, database queries, and system operations.

#### Reliability

- **Availability:** Guarantee high system uptime, with failover mechanisms to minimize downtime.
- **Data Backup:** Implement regular data backup procedures to safeguard against data loss and support seamless recovery in case of failures.

#### Usability

- **User-Friendly Interface:** Provide an intuitive, user-friendly interface suitable for users of varying technical expertise, including doctors, nurses, and administrative staff.
- **Accessibility:** Design the system to meet web accessibility standards (e.g., WCAG), ensuring it is usable by individuals with disabilities.

#### Maintainability

- **Modular Design:** Develop the system with a modular architecture to simplify updates, maintenance, and integration of new features.
- **Comprehensive Documentation:** Provide detailed technical documentation for developers and user manuals for end-users to facilitate ease of use and system upgrades.

#### Interoperability

- **System Integration:** Ensure compatibility with external systems such as laboratory information systems (LIS), pharmacy management systems, and medical equipment tracking systems.

## 3.2 HARDWARE AND SOFTWARE REQUIREMENTS

### Hardware Requirements

- Desktop PC or Laptop: A reliable PC or laptop to host and run the Hospital Management System.
- Processor: Intel® Core™ i3-6006U CPU @ 2.00GHz or an equivalent processor for efficient data processing and system operations.
- RAM: 4.00 GB of RAM to support simultaneous user access and database operations without performance degradation.
- System Architecture: A 64-bit operating system with an x64-based processor for optimal compatibility and performance.
- Monitor Resolution: 1024 x 768 resolution or higher for clear visualization of the graphical user interface.
- Input Devices: Standard keyboard and mouse for navigation and user input.
- Server: A server with sufficient processing power and storage capacity for managing hospital records, patient data, and operational logs.
- Reliable Network Infrastructure: A stable network connection to ensure uninterrupted access and communication between client and server.

### Software Requirements

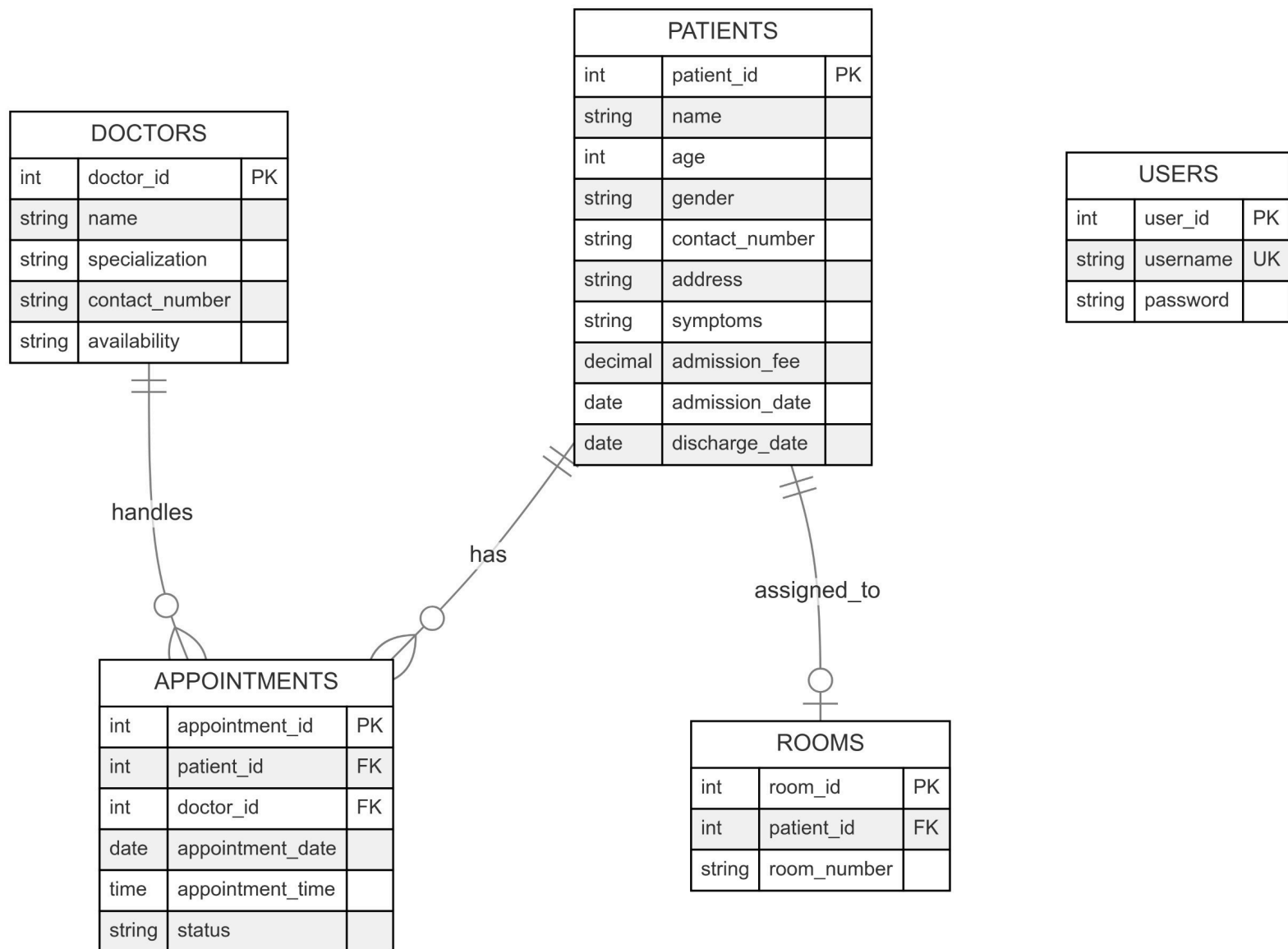
- Operating System: Windows 10 or higher for a stable environment.
- Code Editor: NetBeans or IntelliJ IDEA for Java development and debugging.
- Front End: Java Swing for the graphical user interface (GUI).
- Back End: MySQL for database management and connectivity with the Java backend.
- Middleware: JDBC (Java Database Connectivity) for communication between the Java application and the MySQL database.
- Version Control: Git for source code management and version control.

### 3.3 ARCHITECTURE DIAGRAM

A visual diagram that provides an overall view of the Hospital Management System, identifying the external entities that interact with the system and the major data flows between these entities and the system.



### 3.4 ER DIAGRAM



### 3.5 NORMALISATION

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing a database into two or more tables and defining relationships between the tables. The steps to normalize a database table for Blood Bank Management System are as follows.

ATTRIBUTE	DATATYPE	EXAMPLE VALUE
DOCTOR_ID	INT	1
NAME	VARCHAR(100)	DR. SMITH
AVAILABILITY	VARCHAR(20)	AVAILABLE
PATIENT_ID	INT	1
APPOINTMENT_ID	INT	3
USERNAME	VARCHAR(50)	ADMIN
ROOM_ID	INT	5
ROOM_NUMBER	VARCHAR(10)	101A
USERNAME	VARCHAR(50)	ADMIN
PASSWORD	VARCHAR(255)	PASSWORD123



### First Normal Form (1NF)

To achieve 1NF, we need to ensure that each cell contains only a single value and that there are no repeating groups or multi-valued attributes. We'll separate contact numbers into individual rows for each patient and doctor.

patient_id	name	contact_number	doctor_id
1	Alice	98137871278	101
2	Bob	89889221144	102
4	Charlie	63872992892	101

### Second Normal Form (2NF)

To achieve 2NF, we need to ensure that all non-key attributes are fully functionally dependent on the primary key. We'll decompose the table into multiple tables to remove partial dependencies.

Patient Table

patient_id	patient_name	admission_fee	admission_date	discharge_date
3	John	100.00	2024-11-01	2024-11-05
8	Jane	150.00	2024-11-02	2024-11-06

Appointment Table

appointm ent_id	patient_id	doctor_id	appointment _date	appointment _time	status
1	1	1	2024-11-22	10:00:00	Scheduled
2	2	3	2024-11-22	14:30:00	Completed
3	3	2	2024-11-24	11:00:00	Pending

# PROGRAM CODE

## Chapter 4

### Java Code:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class HospitalManagementSystemGUI extends JFrame {
    private Connection connection;
    private JTextArea textArea;
    private JTextField nameField, ageField, genderField, contactField, addressField, symptomsField,
    admissionFeeField, admissionDateField, dischargeField;
    private JButton addPatientButton, updatePatientButton, showAllPatientsButton,
    showAllDoctorsButton, bookAppointmentButton, showAllAppointmentsButton, assignRoomButton,
    showAllRoomsButton;
    private JButton generateBillButton, viewBillHistoryButton, makePaymentButton;

    public HospitalManagementSystemGUI() {
        try {
            connection = DatabaseConnection.getConnection();
            System.out.println("Database connected successfully!");
        } catch (Exception e) {
            System.out.println("Database connection error: " + e.getMessage());
        }

        setTitle("Hospital Management System");
        setSize(900, 850);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout());
        setBackground(new Color(245, 245, 245));

        JTabbedPane tabbedPane = new JTabbedPane();
        tabbedPane.setFont(new Font("SansSerif", Font.BOLD, 15));
        tabbedPane.setBackground(new Color(60, 100, 150));
        tabbedPane.setForeground(Color.WHITE);

        tabbedPane.addTab("Patients", createPatientPanel());
        tabbedPane.addTab("Hospital Services", createServicesPanel());
        tabbedPane.addTab("Billing", createBillingPanel());

        textArea = new JTextArea();
        textArea.setEditable(false);
        textArea.setFont(new Font("SansSerif", Font.PLAIN, 14));
        JScrollPane scrollPane = new JScrollPane(textArea);
        scrollPane.setPreferredSize(new Dimension(850, 300));
        scrollPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        add(tabbedPane, BorderLayout.NORTH);
        add(scrollPane, BorderLayout.CENTER);

        UIManager.put("TabbedPane.selected", new Color(72, 120, 192));
    }
}
```

```

private JPanel createPatientPanel() {
    JPanel panel = new GradientPanel(new Color(230, 240, 255), new Color(190, 210,
240));

    JPanel patientInfoPanel = new JPanel(new GridLayout(9, 2, 8, 8));
    patientInfoPanel.setOpaque(false);
    patientInfoPanel.setBorder(BorderFactory.createTitledBorder(
BorderFactory.createEmptyBorder(), "Patient Information",
0, 0, new Font("SansSerif", Font.BOLD, 16), new Color(80, 120, 180)));

    nameField = new JTextField(); ageField = new JTextField(); genderField = new
JTextField(); contactField = new JTextField();
    addressField = new JTextField(); symptomsField = new JTextField();
admissionFeeField = new JTextField();
    admissionDateField = new JTextField(); dischargeField = new JTextField();

    addLabelAndField(patientInfoPanel, "Name:", nameField);
    addLabelAndField(patientInfoPanel, "Age:", ageField);
    addLabelAndField(patientInfoPanel, "Gender:", genderField);
    addLabelAndField(patientInfoPanel, "Contact Number:", contactField);
    addLabelAndField(patientInfoPanel, "Address:", addressField);
    addLabelAndField(patientInfoPanel, "Symptoms:", symptomsField);
    addLabelAndField(patientInfoPanel, "Admission Fee:", admissionFeeField);
    addLabelAndField(patientInfoPanel, "Admission Date:", admissionDateField);
    addLabelAndField(patientInfoPanel, "Discharge Date:", dischargeField);

    JPanel patientActionPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20,
20));
    patientActionPanel.setOpaque(false);
    addPatientButton = createStyledButton("Add Patient");
    updatePatientButton = createStyledButton("Update Patient");
    showAllPatientsButton = createStyledButton("View All Patients");

    addPatientButton.addActionListener(e -> addPatient());
    updatePatientButton.addActionListener(e -> updatePatient());
    showAllPatientsButton.addActionListener(e -> showAllPatients());

    patientActionPanel.add(addPatientButton);
    patientActionPanel.add(updatePatientButton);
    patientActionPanel.add(showAllPatientsButton);

    panel.setLayout(new BorderLayout(10, 10));
    panel.add(patientInfoPanel, BorderLayout.NORTH);
    panel.add(patientActionPanel, BorderLayout.SOUTH);

    19
    return panel;
}

```

```

private JPanel createServicesPanel() {
    JPanel panel = new GradientPanel(new Color(230, 240, 255), new Color(190, 210, 240));
    panel.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 20));

    showAllDoctorsButton = createStyledButton("View All Doctors");
    bookAppointmentButton = createStyledButton("Book Appointment");
    showAllAppointmentsButton = createStyledButton("View All Appointments");
    assignRoomButton = createStyledButton("Assign Room");
    showAllRoomsButton = createStyledButton("View All Rooms");

    showAllDoctorsButton.addActionListener(e -> showAllDoctors());
    bookAppointmentButton.addActionListener(e -> bookAppointment());
    showAllAppointmentsButton.addActionListener(e -> showAllAppointments());
    assignRoomButton.addActionListener(e -> assignRoom());
    showAllRoomsButton.addActionListener(e -> showAllRooms());

    panel.add(showAllDoctorsButton);
    panel.add(bookAppointmentButton);
    panel.add(showAllAppointmentsButton);
    panel.add(assignRoomButton);
    panel.add(showAllRoomsButton);

    return panel;
}

private JPanel createBillingPanel() {
    JPanel panel = new GradientPanel(new Color(230, 240, 255), new Color(190, 210, 240));
    panel.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 20));

    generateBillButton = createStyledButton("Generate Bill");
    viewBillHistoryButton = createStyledButton("View Bill History");
    makePaymentButton = createStyledButton("Make Payment");

    generateBillButton.addActionListener(e -> generateBill());
    viewBillHistoryButton.addActionListener(e -> viewBillHistory());
    makePaymentButton.addActionListener(e -> makePayment());

    panel.add(generateBillButton);
    panel.add(viewBillHistoryButton);
    panel.add(makePaymentButton);

    return panel;
}

```

```

private JButton createStyledButton(String text) {
    JButton button = new JButton(text);
    button.setFont(new Font("SansSerif", Font.PLAIN, 15));
    button.setBackground(new Color(72, 120, 192));
    button.setForeground(Color.WHITE);
    button.setFocusPainted(false);
    button.setBorder(BorderFactory.createEmptyBorder(10, 15, 10, 15));
    button.setCursor(new Cursor(Cursor.HAND_CURSOR));
    button.setPreferredSize(new Dimension(160, 50));
    return button;
}

private void addLabelAndField(JPanel panel, String labelText, JTextField textField) {
    JLabel label = new JLabel(labelText);
    label.setFont(new Font("SansSerif", Font.PLAIN, 15));
    label.setForeground(new Color(60, 60, 60));
    panel.add(label);
    panel.add(textField);
    textField.setBorder(BorderFactory.createCompoundBorder(
        BorderFactory.createLineBorder(new Color(190, 210, 240)),
        BorderFactory.createEmptyBorder(5, 10, 5, 10)));
}

private class GradientPanel extends JPanel {
    private Color color1, color2;
    public GradientPanel(Color color1, Color color2) {
        this.color1 = color1;
        this.color2 = color2;
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setPaint(new GradientPaint(0, 0, color1, 0, getHeight(), color2));
        g2d.fillRect(0, 0, getWidth(), getHeight());
    }
}

private void addPatient() {
    String name = nameField.getText();
    int age;
    try {
        age = Integer.parseInt(ageField.getText());
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Invalid age value.");
        return;
    }
    String gender = genderField.getText();
    String contact = contactField.getText();
    String address = addressField.getText();
    String symptoms = symptomsField.getText();
    double admissionFee;
    try {
        admissionFee = Double.parseDouble(admissionFeeField.getText());
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Invalid admission fee value.");
        return;
    }
    String admissionDate = admissionDateField.getText();
    String dischargeDate = dischargeField.getText().isEmpty() ? null : dischargeField.getText();
}

```

```

String sql = "INSERT INTO patients (name, age, gender, contact_number, address,
symptoms, admission_fee, admission_date, discharge_date) VALUES (?, ?, ?, ?, ?, ?, ?,
?, ?)";
try (PreparedStatement stmt = connection.prepareStatement(sql)) {
    stmt.setString(1, name);
    stmt.setInt(2, age);
    stmt.setString(3, gender);
    stmt.setString(4, contact);
    stmt.setString(5, address);
    stmt.setString(6, symptoms);
    stmt.setDouble(7, admissionFee);
    stmt.setString(8, admissionDate);
    stmt.setString(9, dischargeDate);
    stmt.executeUpdate();
    JOptionPane.showMessageDialog(this, "Patient added successfully!");
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Error adding patient: " + e.getMessage());
}

private void updatePatient() {
    // Ask for the patient name
    String patientName = JOptionPane.showInputDialog(this, "Enter patient name to
update:");
    if (patientName == null || patientName.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Patient name is required.");
        return;
    }

    try {
        // Check if patient exists in the database
        int patientId = getPatientIdByName(patientName);
        if (patientId == -1) {
            JOptionPane.showMessageDialog(this, "Patient not found: " +
patientName);
            return;
        }

        // Ask the user for the field to update
        String[] fields = {"Name", "Age", "Gender", "Contact Number", "Address",
"Symptoms", "Admission Fee", "Admission Date", "Discharge Date"};
        String fieldToUpdate = (String) JOptionPane.showInputDialog(
            this,
            "Select the field to update:",
            "Update Patient",
            JOptionPane.QUESTION_MESSAGE,
            null,
            fields,
            fields[0]
        );
    }
}

```

```

// SQL query to update the specific field
String sql = "UPDATE patients SET " + getDatabaseFieldName(fieldToUpdate) + " = ? WHERE
patient_id = ?";
try (PreparedStatement stmt = connection.prepareStatement(sql)) {
    // Convert and set the appropriate data type for the field
    switch (fieldToUpdate) {
        case "Age":
            stmt.setInt(1, Integer.parseInt(newValue));
            break;
        case "Admission Fee":
            stmt.setDouble(1, Double.parseDouble(newValue));
            break;
        case "Admission Date":
        case "Discharge Date":
            stmt.setDate(1, java.sql.Date.valueOf(newValue));
            break;
        default:
            stmt.setString(1, newValue);
    }
    stmt.setInt(2, patientId);
    int rowsUpdated = stmt.executeUpdate();

    if (rowsUpdated > 0) {
        JOptionPane.showMessageDialog(this, "Patient " + fieldToUpdate + " updated
successfully!");
    } else {
        JOptionPane.showMessageDialog(this, "Failed to update patient " + fieldToUpdate + ".");
    }
}
} catch (SQLException | NumberFormatException e) {
    JOptionPane.showMessageDialog(this, "Error updating patient: " + e.getMessage());
}
}

// Helper method to map GUI fields to database column names
private String getDatabaseFieldName(String fieldName) {
    switch (fieldName) {
        case "Name": return "name";
        case "Age": return "age";
        case "Gender": return "gender";
        case "Contact Number": return "contact_number";
        case "Address": return "address";
        case "Symptoms": return "symptoms";
        case "Admission Fee": return "admission_fee";
        case "Admission Date": return "admission_date";
        case "Discharge Date": return "discharge_date";
        default: throw new IllegalArgumentException("Unknown field: " + fieldName);
    }
}

private int getPatientIdByName(String name) throws SQLException {
    String sql = "SELECT patient_id FROM patients WHERE name = ?";
    try (PreparedStatement stmt = connection.prepareStatement(sql)) {
        stmt.setString(1, name);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            return rs.getInt("patient_id");
        } else {
            return -1;
        }
    }
}
}

```

```

private void showAllPatients() {
    String sql = "SELECT * FROM patients";
    try (Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {
        StringBuilder patientsList = new StringBuilder();
        while (rs.next()) {
            patientsList.append("ID: ").append(rs.getInt("patient_id")).append("\n")
                .append("Name: ").append(rs.getString("name")).append("\n")
                .append("Age: ").append(rs.getInt("age")).append("\n")
                .append("Gender: ").append(rs.getString("gender")).append("\n")
                .append("Contact: ").append(rs.getString("contact_number")).append("\n")
                .append("Address: ").append(rs.getString("address")).append("\n")
                .append("Symptoms: ").append(rs.getString("symptoms")).append("\n")
                .append("Admission Fee: ").append(rs.getDouble("admission_fee")).append("\n")
                .append("Admission Date: ").append(rs.getDate("admission_date")).append("\n")
                .append("Discharge Date: ")
            ").append(rs.getDate("discharge_date")).append("\n\n");
        }
        textArea.setText(patientsList.toString());
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Error retrieving patients: " + e.getMessage());
    }
}

private void showAllDoctors() {
    String sql = "SELECT * FROM doctors";
    try (Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {
        StringBuilder doctorsList = new StringBuilder();
        while (rs.next()) {
            doctorsList.append("ID: ").append(rs.getInt("doctor_id")).append("\n")
                .append("Name: ").append(rs.getString("name")).append("\n")
                .append("Specialization: ")
            ").append(rs.getString("specialization")).append("\n")
                .append("Contact: ").append(rs.getString("contact_number")).append("\n")
                .append("Availability: ")
            ").append(rs.getString("availability")).append("\n\n");
        }
        textArea.setText(doctorsList.toString());
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Error retrieving doctors: " + e.getMessage());
    }
}

private boolean isDoctorAvailable(int doctorId, String appointmentDate) throws SQLException {
    String sql = "SELECT availability FROM doctors WHERE doctor_id = ?";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setInt(1, doctorId);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            // Check if doctor is available
            String availability = rs.getString("availability");
            if (!"Available".equals(availability)) {
                return false;
            }
        }
    }
}

```



```

// Check if doctor already has appointment at that time
sql = "SELECT COUNT(*) FROM appointments WHERE doctor_id = ? AND appointment_date = ?";
try (PreparedStatement apptStmt = connection.prepareStatement(sql)) {
    apptStmt.setInt(1, doctorId);
    apptStmt.setString(2, appointmentDate);
    ResultSet apptRs = apptStmt.executeQuery();
    if (apptRs.next()) {
        return apptRs.getInt(1) == 0;
    }
}
return false;
}

// Add this method to check if patient already has a room
private boolean isPatientAssignedRoom(int patientId) throws SQLException {
    String sql = "SELECT COUNT(*) FROM rooms WHERE patient_id = ?";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setInt(1, patientId);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return rs.getInt(1) > 0;
        }
    }
    return false;
}

private void bookAppointment() {
    String patientName = JOptionPane.showInputDialog(this, "Enter patient name:");
    if (patientName == null || patientName.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Patient name is required.");
        return;
    }

    try {
        int patientId = getPatientIdByName(patientName);
        if (patientId == -1) {
            JOptionPane.showMessageDialog(this, "Patient not found.");
            return;
        }

        String doctorIdStr = JOptionPane.showInputDialog(this, "Enter doctor ID for appointment:");
        if (doctorIdStr == null || doctorIdStr.isEmpty()) {
            return;
        }
        int doctorId = Integer.parseInt(doctorIdStr);

        String date = JOptionPane.showInputDialog(this, "Enter appointment date (yyyy-MM-dd):");
        if (date == null || date.isEmpty()) {
            return;
        }
    }
}

```

```

        // Check doctor availability
        if (!isDoctorAvailable(doctorId, date)) {
            JOptionPane.showMessageDialog(this, "Doctor is not available for the selected date.");
            return;
        }

        String time = JOptionPane.showInputDialog(this, "Enter appointment time (HH:mm):");
        if (time == null || time.isEmpty()) {
            return;
        }

        String sql = "INSERT INTO appointments (patient_id, doctor_id, appointment_date,
appointment_time, status) VALUES (?, ?, ?, ?, 'Scheduled')";
        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setInt(1, patientId);
            pstmt.setInt(2, doctorId);
            pstmt.setString(3, date);
            pstmt.setString(4, time);
            pstmt.executeUpdate();
            JOptionPane.showMessageDialog(this, "Appointment booked successfully!");
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Error booking appointment: " + e.getMessage());
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Invalid doctor ID format.");
    }
}

private void showAllAppointments() {
    String sql = "SELECT * FROM appointments";
    try (Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {
        StringBuilder appointmentsList = new StringBuilder();
        while (rs.next()) {
            appointmentsList.append("Appointment ID:
").append(rs.getInt("appointment_id")).append("\n")
                .append("Patient ID: ").append(rs.getInt("patient_id")).append("\n")
                .append("Doctor ID: ").append(rs.getInt("doctor_id")).append("\n")
                .append("Date: ").append(rs.getDate("appointment_date")).append("\n")
                .append("Time: ").append(rs.getTime("appointment_time")).append("\n")
                .append("Status: ").append(rs.getString("status")).append("\n\n");
        }
        textArea.setText(appointmentsList.toString());
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Error retrieving appointments: " + e.getMessage());
    }
}

private void assignRoom() {
    String patientName = JOptionPane.showInputDialog(this, "Enter patient name:");
    if (patientName == null || patientName.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Patient name is required.");
        return;
    }

    try {
        int patientId = getPatientIdByName(patientName);
        if (patientId == -1) {
            JOptionPane.showMessageDialog(this, "Patient not found.");
            return;
        }
    }
}

```

```

// Check if patient already has a room
if (isPatientAssignedRoom(patientId)) {
    JOptionPane.showMessageDialog(this, "Patient is already assigned to a room.");
    return;
}

String roomNumber = JOptionPane.showInputDialog(this, "Enter room number:");
if (roomNumber == null || roomNumber.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Room number is required.");
    return;
}

if (isRoomAvailable(roomNumber)) {
    String sql = "INSERT INTO rooms (patient_id, room_number) VALUES (?, ?)";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setInt(1, patientId);
        pstmt.setString(2, roomNumber);
        pstmt.executeUpdate();
        JOptionPane.showMessageDialog(this, "Room assigned successfully!");
    }
} else {
    JOptionPane.showMessageDialog(this, "Room is already occupied.");
}
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Error assigning room: " + e.getMessage());
}
}

private void showAllRooms() {
    String sql = "SELECT p.name, r.room_number FROM patients p JOIN rooms r ON p.patient_id = r.patient_id";
    try (Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {
        StringBuilder roomsList = new StringBuilder();
        while (rs.next()) {
            roomsList.append("Patient Name: ").append(rs.getString("name")).append("\n")
                .append("Room Number: ")
                .append(rs.getString("room_number")).append("\n\n");
        }
        textArea.setText(roomsList.toString());
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Error retrieving rooms: " + e.getMessage());
    }
}
}

```

```

private boolean isRoomAvailable(String roomNumber) throws SQLException {
    String sql = "SELECT COUNT(*) FROM rooms WHERE room_number = ?";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setString(1, roomNumber);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return rs.getInt(1) == 0;
        }
    }
    return false;
}

private void generateBill() {
    String patientName = JOptionPane.showInputDialog(this, "Enter patient name:");
    if (patientName == null || patientName.isEmpty()) {
        return;
    }

    try {
        int patientId = getPatientIdByName(patientName);
        if (patientId == -1) {
            JOptionPane.showMessageDialog(this, "Patient not found.");
            return;
        }

        // Get room charges and doctor fees for reference
        double roomCharges = calculateRoomCharges(patientId);
        double doctorFees = calculateDoctorFees(patientId);

        // Show the calculated charges as reference
        JOptionPane.showMessageDialog(this,
            "Calculated Charges (For Reference):\n" +
            "Room Charges: $" + roomCharges + "\n" +
            "Doctor Fees: $" + doctorFees + "\n" +
            "Total Calculated: $" + (roomCharges + doctorFees));

        // Ask for manual entry of charges
        String roomChargesStr = JOptionPane.showInputDialog(this, "Enter room charges amount:");
        String doctorFeesStr = JOptionPane.showInputDialog(this, "Enter doctor fees amount:");

        if (roomChargesStr == null || doctorFeesStr == null) {
            return;
        }

        double manualRoomCharges = Double.parseDouble(roomChargesStr);
        double manualDoctorFees = Double.parseDouble(doctorFeesStr);
        double totalAmount = manualRoomCharges + manualDoctorFees;

        // Create bill in database
        String sql = "INSERT INTO billing (patient_id, total_amount, paid_amount, balance_amount, bill_date, payment_status) VALUES (?, ?, 0, ?, CURDATE(), 'PENDING')";
        try (PreparedStatement pstmt = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
            pstmt.setInt(1, patientId);
            pstmt.setDouble(2, totalAmount);
            pstmt.setDouble(3, totalAmount);
            pstmt.executeUpdate();

```

```

        // Get the generated bill_id
        ResultSet rs = pstmt.getGeneratedKeys();
        if (rs.next()) {
            int billId = rs.getInt(1);
            // Add billing items with manual amounts
            addBillingItem(billId, "Room Charges", manualRoomCharges);
            addBillingItem(billId, "Doctor Fees", manualDoctorFees);
        }
    }

    JOptionPane.showMessageDialog(this, "Bill generated successfully!\nTotal Amount: $" + totalAmount);
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Error generating bill: " + e.getMessage());
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(this, "Invalid amount format. Please enter valid numbers.");
}
}

private void addBillingItem(int billId, String description, double amount) throws SQLException {
    String sql = "INSERT INTO billing_items (bill_id, item_type, description, quantity, unit_price, total_price,
date_added) VALUES (?, ?, ?, 1, ?, ?, CURDATE())";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setInt(1, billId);
        pstmt.setString(2, description);
        pstmt.setString(3, description);
        pstmt.setDouble(4, amount);
        pstmt.setDouble(5, amount);
        pstmt.executeUpdate();
    }
}

private double calculateRoomCharges(int patientId) throws SQLException {
    // Get room details and calculate charges
    String sql = "SELECT DATEDIFF(COALESCE(discharge_date, CURDATE()), admission_date) as days FROM patients WHERE
patient_id = ?";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setInt(1, patientId);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            int days = rs.getInt("days");
            // Assuming room rate is $100 per day
            return days * 100.0;
        }
    }
    return 0.0;
}

private double calculateDoctorFees(int patientId) throws SQLException {
    // Calculate total doctor fees from appointments
    String sql = "SELECT COUNT(*) as visits FROM appointments WHERE patient_id = ?";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setInt(1, patientId);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            int visits = rs.getInt("visits");
            // Assuming doctor fee is $50 per visit
            return visits * 50.0;
        }
    }
    return 0.0;
}
}

```

```

private void viewBillHistory() {
    String patientName = JOptionPane.showInputDialog(this, "Enter patient name:");
    if (patientName == null || patientName.isEmpty()) {
        return;
    }

    try {
        int patientId = getPatientIdByName(patientName);
        if (patientId == -1) {
            JOptionPane.showMessageDialog(this, "Patient not found.");
            return;
        }

        String sql = "SELECT b.*, p.name FROM billing b JOIN patients p ON b.patient_id = p.patient_id WHERE b.patient_id = ?";
        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setInt(1, patientId);
            ResultSet rs = pstmt.executeQuery();

            StringBuilder billHistory = new StringBuilder();
            billHistory.append("Bill History for Patient: ").append(patientName).append("\n\n");

            while (rs.next()) {
                billHistory.append("Bill ID: ").append(rs.getInt("bill_id")).append("\n")
                    .append("Date: ").append(rs.getDate("bill_date")).append("\n")
                    .append("Total Amount: $").append(rs.getDouble("total_amount")).append("\n")
                    .append("Paid Amount: $").append(rs.getDouble("paid_amount")).append("\n")
                    .append("Balance: $").append(rs.getDouble("balance_amount")).append("\n")
                    .append("Status: ").append(rs.getString("payment_status")).append("\n\n");
            }

            textArea.setText(billHistory.toString());
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error retrieving bill history: " + e.getMessage());
        }
    }

    private void makePayment() {
        String billIdStr = JOptionPane.showInputDialog(this, "Enter Bill ID:");
        if (billIdStr == null || billIdStr.isEmpty()) {
            return;
        }

        try {
            int billId = Integer.parseInt(billIdStr);

            // First get the current bill details
            String checkSql = "SELECT total_amount, paid_amount, balance_amount FROM billing WHERE bill_id = ?";

            double totalAmount = 0;
            double paidAmount = 0;
            double balanceAmount = 0;

            try (PreparedStatement checkStmt = connection.prepareStatement(checkSql)) {
                checkStmt.setInt(1, billId);
                ResultSet rs = checkStmt.executeQuery();
                if (rs.next()) {
                    totalAmount = rs.getDouble("total_amount");
                    paidAmount = rs.getDouble("paid_amount");
                    balanceAmount = rs.getDouble("balance_amount");
                } else {
                    JOptionPane.showMessageDialog(this, "Bill not found.");
                    return;
                }
            }
        }
    }
}

```

```

String amountStr = JOptionPane.showInputDialog(this, "Enter payment amount:");
if (amountStr == null || amountStr.isEmpty()) {
    return;
}

double amount = Double.parseDouble(amountStr);
String[] paymentMethods = {"Cash", "Credit Card", "Debit Card", "Insurance"};
String paymentMethod = (String) JOptionPane.showInputDialog(
    this,
    "Select payment method:",
    "Payment Method",
    JOptionPane.QUESTION_MESSAGE,
    null,
    paymentMethods,
    paymentMethods[0]
);

if (paymentMethod != null) {
    // Update the paid amount and balance amount
    double newPaidAmount = paidAmount + amount;
    double newBalanceAmount = totalAmount - newPaidAmount;

    // Update billing table
    String sql = "UPDATE billing SET paid_amount = ?, balance_amount = ?,
payment_status = CASE WHEN (paid_amount + ?) >= total_amount THEN 'PAID' ELSE 'PARTIAL' END WHERE
bill_id = ?";
    try (PreparedStatement updateStmt = connection.prepareStatement(sql)) {
        updateStmt.setDouble(1, newPaidAmount);
        updateStmt.setDouble(2, newBalanceAmount);
        updateStmt.setDouble(3, amount);
        updateStmt.setInt(4, billId);
        updateStmt.executeUpdate();
    }

    // Record payment
    sql = "INSERT INTO payments (bill_id, amount_paid, payment_date, payment_method,
payment_status) VALUES (?, ?, CURDATE(), ?, 'COMPLETED')";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setInt(1, billId);
        pstmt.setDouble(2, amount);
        pstmt.setString(3, paymentMethod);
        pstmt.executeUpdate();
    }

    JOptionPane.showMessageDialog(this, "Payment processed successfully!");
}
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Error processing payment: " + e.getMessage());
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(this, "Invalid number format.");
}
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        new HospitalManagementSystemGUI().setVisible(true);
    });
}
}

```

## 2 . LOGIN PAGE

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
public class LoginPage extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;
    public LoginPage() {
        setTitle("Hospital Management System - Login");
        setSize(800, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout());
        JPanel backgroundPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                ImageIcon icon = new
ImageIcon("C:/Users/harsh/Downloads/log.jpg");
(icon.getImage(), 0, 0, getWidth(), getHeight(), this);
            }
        };
        backgroundPanel.setLayout(new BorderLayout());
        add(backgroundPanel);
        JPanel loginPanel = new JPanel();
        loginPanel.setLayout(new BoxLayout(loginPanel, BoxLayout.Y_AXIS));
        loginPanel.setOpaque(false);
        loginPanel.setPreferredSize(new Dimension(300, 500));
        loginPanel.setBorder(BorderFactory.createEmptyBorder(30, 50, 50,
80)); /
        JLabel titleLabel = new JLabel("Login", SwingConstants.CENTER);
        titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
        titleLabel.setForeground(new Color(0, 51, 102));
        titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
```



```

// Username and password labels and fields
JLabel usernameLabel = new JLabel("Username:");
usernameLabel.setFont(new Font("Arial", Font.BOLD, 16));
usernameLabel.setForeground(new Color(0, 51, 102));

usernameField = new JTextField(10);
usernameField.setFont(new Font("Arial", Font.PLAIN, 14));
usernameField.setBorder(BorderFactory.createLineBorder(new Color(0,
1, 102), 1));
usernameField.setPreferredSize(new Dimension(200, 30));

JLabel passwordLabel = new JLabel("Password:");
passwordLabel.setFont(new Font("Arial", Font.BOLD, 16));
passwordLabel.setForeground(new Color(0, 51, 102));

passwordField = new JPasswordField(10);
passwordField.setFont(new Font("Arial", Font.PLAIN, 14));
passwordField.setBorder(BorderFactory.createLineBorder(new Color(0,
1, 102), 1));
passwordField.setPreferredSize(new Dimension(200, 30));

// Login button with improved styling
loginButton = new JButton("Login");
loginButton.setFont(new Font("Arial", Font.BOLD, 16));
loginButton.setBackground(new Color(0, 153, 76)); // Softer green
loginButton.setForeground(Color.WHITE);
loginButton.setFocusPainted(false);
loginButton.setBorder(BorderFactory.createEmptyBorder(10, 10, 10,
0));

loginButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
loginButton.addActionListener(new LoginActionListener());

// Adding components to login panel with spacing
loginPanel.add(Box.createVerticalStrut(10));
loginPanel.add(titleLabel);
loginPanel.add(Box.createVerticalStrut(10));
loginPanel.add(usernameLabel);
loginPanel.add(usernameField);
loginPanel.add(Box.createVerticalStrut(10));
loginPanel.add(passwordLabel);
loginPanel.add(passwordField);
loginPanel.add23(Box.createVerticalStrut(10));
loginPanel.add(loginButton);

```

```

// Add login panel to the right side of the background panel
backgroundPanel.add(loginPanel, BorderLayout.EAST);

// Make the window visible
setVisible(true);
}

// Action listener for login button
private class LoginActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        String username = usernameField.getText();
        String password = new String(passwordField.getPassword());

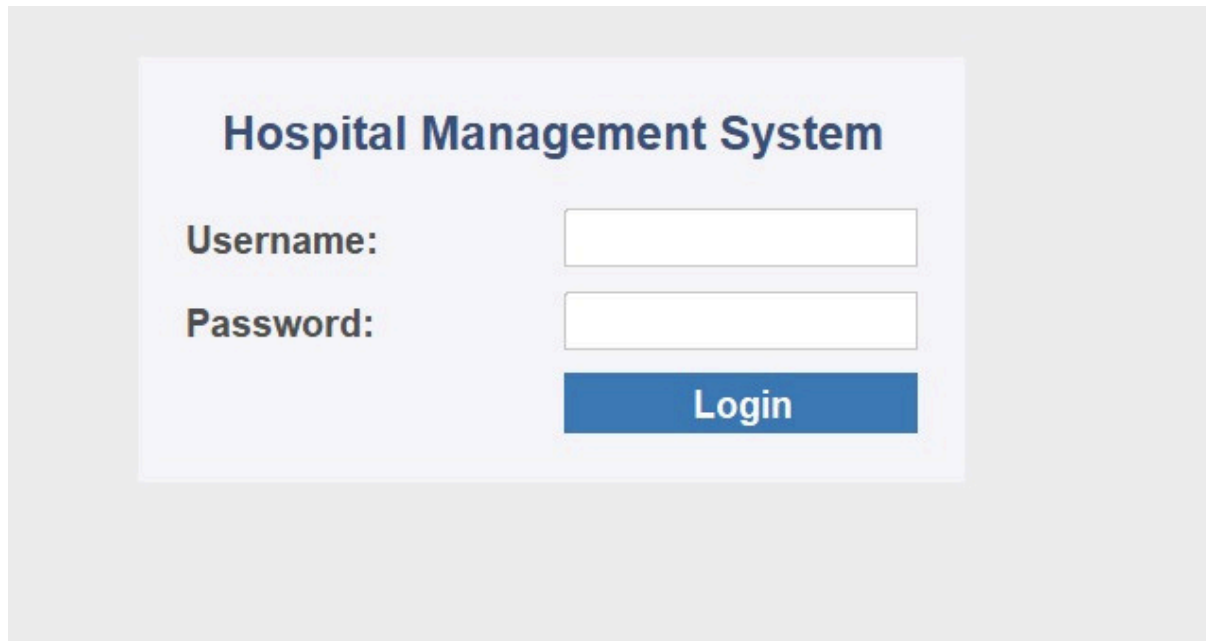
        if (validateLogin(username, password)) {
            JOptionPane.showMessageDialog(LoginPage.this, "Login
successful!");
            new HospitalManagementSystemGUI().setVisible(true);
            dispose(); // Close login page
        } else {
            JOptionPane.showMessageDialog(LoginPage.this, "Invalid
username or password", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private boolean validateLogin(String username, String password) {
        String sql = "SELECT * FROM users WHERE username = ? AND
password = ?";
        try (Connection connection =
DatabaseConnection.getConnection();
            PreparedStatement stmt =
connection.prepareStatement(sql)) {
            stmt.setString(1, username);
            stmt.setString(2, password);
            try (ResultSet rs = stmt.executeQuery()) {
                return rs.next();
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(LoginPage.this, "Database
error: " + ex.getMessage());
            return false;
        }
    }
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        new LoginPage().setVisible(true);
    });
}
}

```

## LOGIN PAGE



The login page features a light blue rectangular form centered on a gray background. At the top of the form, the text "Hospital Management System" is displayed in a bold, dark blue font. Below this, the labels "Username:" and "Password:" are positioned to the left of two white input fields with thin gray borders. A solid blue button with the word "Login" in white text is located at the bottom right of the form.

## PATIENT DASHBOARD



The dashboard interface includes a top navigation bar with three tabs: "Patients", "Hospital Services", and "Billing", with the "Billing" tab currently selected. Below the navigation bar, three blue buttons are arranged horizontally: "Generate Bill", "View Bill History", and "Make Payment". The main content area is a large, empty light blue rectangle. At the bottom of the dashboard, a scrollable list displays bill information. The first entry shows "Bill ID: 10", "Date: 2024-11-11", "Total Amount: \$140000.0", "Paid Amount: \$40000.0", "Balance: \$100000.0", and "Status: PARTIAL". The second entry shows "Bill ID: 11", "Date: 2024-11-12", "Total Amount: \$350.0", and "Paid Amount: \$350.0". A vertical scrollbar is visible on the right side of the list.

Bill ID	Date	Total Amount	Paid Amount	Balance	Status
10	2024-11-11	\$140000.0	\$40000.0	\$100000.0	PARTIAL
11	2024-11-12	\$350.0	\$350.0		

# HOSPITAL SERVICES DASHBOARD

PatientsHospital ServicesBilling

View All Doctors

Book Appointment

View All Appoint...

Assign Room

View All Rooms

Patient Name: Divyan  
Room Number: 4

Patient Name: AMMS  
Room Number: 5

Patient Name: Hari  
Room Number: 7

Patient Name: Haresh  
Room Number: 6

# BILLING DASHBOARD

PatientsHospital ServicesBilling

Generate Bill

View Bill History

Make Payment

Bill ID: 10  
Date: 2024-11-11  
Total Amount: \$140000.0  
Paid Amount: \$40000.0  
Balance: \$100000.0  
Status: PARTIAL

Bill ID: 11  
Date: 2024-11-12  
Total Amount: \$350.0  
Paid Amount: \$350.0  
Balance: \$0.0  
Status: PAID

The development of the Hospital Management System marks a significant improvement in optimizing hospital operations and enhancing patient care. By leveraging advanced technologies, the system streamlines essential processes such as appointment scheduling, patient management, doctor coordination, and resource allocation, thereby increasing efficiency and reducing operational costs. With the integration of Java and SQL, the system ensures secure and robust backend operations and efficient database management. This comprehensive solution facilitates key functionalities such as patient admission, appointment management, and room assignments, contributing to smoother hospital workflows and delivering superior patient care.

The user-friendly interface, powered by Java, ensures ease of access for both patients and hospital staff, contributing to a smoother and more efficient experience. With robust security features, including user authentication and role-based access control, the system safeguards sensitive patient data while ensuring compliance with industry regulations.

In summary, the Hospital Management System serves as a powerful tool in optimizing hospital operations, promoting better coordination among healthcare professionals, and enhancing patient satisfaction. It is an essential step towards modernizing healthcare institutions and supporting the delivery of high-quality medical services.

## 7.1 REFERENCES

- [1] <https://docs.oracle.com/javase/tutorial/jdbc/>
- [2] <https://dev.mysql.com/doc/connector-j/en/>
- [3] <https://leadsquared.com/industries/healthcare/hospital-management-system-hms/>