

# **HOSPITAL MANAGEMENT SYSTEM**

## **MINI PROJECT REPORT**

Submitted by

**TARUNKUMAR S [ RA2311003012062]  
RITHISH BARATH N [ RA2311003012062]  
RITHISH KARTHIKEYAN [ RA2311003012062]  
SHAN NEERAJ M [ RA2311003012062]**

Under the Guidance of

**Dr. Viveka S**

**21CSC203P – ADVANCED PROGRAMMING PRACTICES**

**DEPARTMENT OF COMPUTING TECHNOLOGIES**



**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR**

**NOVEMBER 2024**

# **SRM INSTITUTION OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

## **BONAFIDE CERTIFICATE**

Certified that the 21CSC203P Advance Programming Practice course project report titled **“Hospital Management System”** is the bonafide work done by **S Tarunkumar [RA231100301062]** of **II Year/III Sem B.Tech(CSE)** who carried out the mini project under my supervision.

### **SIGNATURE**

**Faculty In-Charge**

**Dr. Viveka S**

**Assistant Professor**

Department of Computing Technologies,

School of Computing,

SRM Institute of Science and Technology

Kattankulathur

### **SIGNATURE**

**Dr. NIRANJANA. G**

**Head of the Department**

Professor and Head

Department of Computing Technologies,

School of Computing,

SRM Institute of Science and Technology

Kattankulathur

## **ABSTRACT**

The Hospital Management System (HMS) is a robust application designed to streamline the administrative and operational functions of healthcare institutions. Developed using Java for the frontend and MySQL for the backend, the HMS centralizes and automates key processes such as patient registration, data management, appointment scheduling, staff information, billing, and inventory control. The system ensures that these processes are accessible through a user-friendly interface while providing secure and efficient data handling.

Key features of the HMS include a centralized database, enabling seamless access to real-time patient information, medical histories, and treatment records. This improves healthcare delivery by allowing medical staff quick access to critical patient information, which is especially beneficial in emergencies. The automated appointment scheduling and billing modules significantly reduce administrative workloads and minimize human errors, thus contributing to better time management and financial accuracy. Additionally, the inventory management module aids in tracking medical supplies, reducing waste, and ensuring that essential items are always available.

By integrating diverse hospital functions into a cohesive digital system, the HMS fosters improved workflows, enhances patient care, and supports data-driven decision-making. It minimizes manual tasks, reduces wait times, and ensures an organized and accessible repository of information, ultimately contributing to a more efficient, reliable, and patient-centered healthcare environment.

## ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavours. We would like to express my warmth of gratitude to our **Registrar Dr. S. PONNUSAMY**, for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V. GOPAL**, for bringing out novelty in all executions. We would like to express my heartfelt thanks to our Chairperson, School of Computing **Dr. REVATHI VENKATARAMAN**, for imparting confidence in completing my course project.

We extend our gratitude to our Associate Chairperson **Dr. PUSHPALATHA. M, Professor** and our **HOD Dr. NIRANJANA. G, Professor and Head, Department of Computing Technologies** for their Support.

We wish to express our sincere thanks to **Course Audit Professors Dr. Vadivu.G, Professor, Department of Data Science and Business Systems** and **Course Coordinators Dr. MANJULA. R, Assistant Professor and Dr. N.A.S VINOTH, Assistant Professor, Department of Computing Technologies** for their constant encouragement and support.

We are highly thankful to our Course project Faculty **Dr S Viveka, Assistant Professor, Department of Computing Technologies** for her assistance, timely suggestion and guidance throughout the duration of this course project.

Finally, we thank our parents and friends and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

## TABLE OF CONTENTS

Sr. No.	Title	Page No.
1.	Introduction	6
2.	Literature Survey	7
3.	Requirement Analysis	11
4.	Architecture and Design	15
5.	Implementation	17
6.	Experiment Result and Analysis	58
7.	Future Scope	60
8.	Conclusion	63
9.	References	63

# INTRODUCTION

**Overview of the Project:** The Hospital Management System (HMS) is a software application designed to automate and streamline the key administrative and operational tasks in hospitals. This includes patient management, appointments, billing, medical records, inventory management, and more. The primary goal of this system is to improve the quality and efficiency of healthcare services while minimizing manual tasks and human errors.

**Motivation and Need:** Hospitals face challenges such as maintaining large amounts of patient data, scheduling appointments, managing medical records, and processing billing efficiently. These processes often involve heavy paperwork, manual calculations, and coordination between multiple departments. An automated system can solve these challenges by reducing paperwork, ensuring faster access to information, and improving patient care.

**Scope:** The scope of the HMS includes patient registration, doctor scheduling, inventory management, billing, and medical record maintenance. It is designed to cater to various healthcare settings, ranging from small clinics to large hospitals.

## Objectives:

- ❖ Improve hospital workflow and reduce administrative overhead.
- ❖ Enhance patient care through accurate, real-time medical records.
- ❖ Reduce human errors in appointments, billing, and inventory management.
- ❖ Ensure data security and privacy, complying with healthcare regulations (e.g., HIPAA).

## LITERATURE SURVEY

The development of Hospital Management Systems (HMS) has been a topic of ongoing research and practical implementation due to the critical role these systems play in improving healthcare efficiency, accuracy, and accessibility. This literature survey covers existing HMS solutions, explores advancements in relevant technologies, and highlights ongoing challenges in healthcare information systems.

### 1. Existing Systems

- ❖ **Legacy Hospital Management Systems:** Traditional HMS solutions, such as MediTech and Cerner, have been widely used in healthcare settings. These systems provide comprehensive patient information management, billing, appointment scheduling, and other essential features. However, due to their complexity, these systems often require significant customization, which increases costs and deployment time. Additionally, older systems frequently lack integration with emerging technologies such as mobile apps and cloud services, reducing accessibility for modern healthcare environments.
- ❖ **Open-Source Hospital Management Software:** Open-source HMS like OpenMRS (Open Medical Record System) has gained popularity, especially in developing countries. OpenMRS provides a flexible framework for managing medical records and is highly customizable. Its adaptability makes it a popular choice for non-profit organizations and research institutions. However, it often lacks the out-of-the-box features needed for hospital administration, such as inventory management and billing modules, which must be developed or integrated separately.
- ❖ **Cloud-Based Hospital Management Systems:** Cloud-based HMS solutions like Athenahealth and AdvancedMD offer advantages such as scalability, remote access, and lower infrastructure costs. These systems facilitate real-time access to patient data, support telemedicine services, and enable healthcare providers to collaborate effectively across locations. However, concerns about data privacy, latency, and compliance with healthcare regulations like HIPAA (Health Insurance Portability and Accountability Act) are challenges to adoption.

## 2. Technologies Used

- ❖ **Relational Database Management Systems (RDBMS):** Most HMS systems, including those built with MySQL, PostgreSQL, and Oracle, rely on RDBMS due to the structured nature of healthcare data. These databases support ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring data integrity and reliability. However, the vast and growing volume of healthcare data often requires optimization strategies, such as indexing and data partitioning, to manage performance effectively.
- ❖ **Frontend Development Technologies:** Hospital management systems need intuitive and user-friendly interfaces to reduce training times for hospital staff. Desktop-based systems often rely on Java Swing or .NET, while web-based systems increasingly use modern JavaScript frameworks like React, Angular, or Vue.js. The use of responsive design principles ensures that systems are accessible across devices, including tablets and smartphones.
- ❖ **Backend Development Technologies:** Java, .NET, and PHP are widely used for backend logic in HMS. Recently, microservices architectures have been explored for modular systems, enabling independent development and scaling of different system components (e.g., patient records, billing). RESTful APIs and GraphQL facilitate data retrieval and interoperability with other healthcare systems, such as EHR (Electronic Health Record) systems and lab management software.
- ❖ **Data Security and Privacy:** Ensuring data security is paramount in HMS, as systems store sensitive information such as medical records, insurance details, and billing information. Technologies such as HTTPS, TLS encryption, secure authentication, and access control are critical. Compliance with regulations like HIPAA (USA), GDPR (EU), and the Indian Personal Data Protection Bill (India) is necessary to protect patient privacy and secure data transfers across systems.



### 3. Emerging Trends and Technologies in HMS

- ❖ **Artificial Intelligence (AI) and Machine Learning (ML):** AI and ML applications in HMS are rapidly evolving. They can assist in predictive analytics, which identifies patient admission trends, analyzes patient health data, and optimizes resource allocation. For instance, ML algorithms can predict emergency department visits, helping hospitals allocate resources more efficiently. Additionally, natural language processing (NLP) aids in processing unstructured medical data, such as doctor's notes, which can be integrated into patient records for more comprehensive health information.
- ❖ **Blockchain for Secure Data Management:** Blockchain technology has been proposed as a solution to enhance the security and integrity of healthcare data. By creating immutable records of transactions and access events, blockchain can safeguard patient data against tampering and unauthorized access. However, challenges such as scalability, regulatory approval, and the need for specialized infrastructure limit its immediate applicability in hospital management.
- ❖ **Cloud Computing and IoT Integration:** Cloud platforms like AWS, Microsoft Azure, and Google Cloud offer storage and computing power that can scale based on hospital needs. This enables hospitals to reduce infrastructure costs and provides remote access to medical information, which is valuable in telemedicine and rural healthcare. The Internet of Things (IoT) further enables HMS to integrate with wearable devices and medical sensors, providing real-time patient monitoring and data collection.

## 4. Challenges in Existing Hospital Management Systems

- ❖ **Data Fragmentation and Interoperability:** Many healthcare facilities use disparate systems that do not communicate with each other effectively. This lack of interoperability results in data silos, making it challenging to provide comprehensive patient care. Although standards like HL7 (Health Level 7) and FHIR (Fast Healthcare Interoperability Resources) have been developed to improve interoperability, many systems still struggle with seamless data exchange.
- ❖ **High Implementation and Maintenance Costs:** Implementation costs for HMS can be high, especially for proprietary and large-scale systems. Hospitals must invest in hardware, software licensing, data migration, and user training, which can strain their budgets. Cloud-based HMS can mitigate some costs, but subscription fees and data privacy concerns remain.
- ❖ **Scalability and Adaptability:** Hospitals are dynamic environments that require scalable systems to handle varying workloads. Traditional HMS may not scale well when hospitals grow or when new features are required. Modular and microservices-based architectures offer scalability and flexibility, but transitioning legacy systems to these architectures can be complex.
- ❖ **User Training and Resistance to Change:** Many healthcare professionals face a learning curve when transitioning from paper-based systems or outdated software to modern HMS. Training is essential, but some users may resist adopting new technologies, which impacts the system's efficiency. Designing an intuitive interface with clear instructions and providing adequate training can help mitigate this issue.

## **REQUIREMENT ANALYSIS**

The Hospital Management System (HMS) is designed to optimize the day-to-day operations within a healthcare facility by streamlining the management of patient records, staff information, appointments, billing, inventory, and other essential functions. This requirement analysis identifies and categorizes the system requirements necessary to build an efficient, secure, and user-friendly HMS.

### **Requirement Analysis**

The Hospital Management System (HMS) is designed to optimize the day-to-day operations within a healthcare facility by streamlining the management of patient records, staff information, appointments, billing, inventory, and other essential functions. This requirement analysis identifies and categorizes the system requirements necessary to build an efficient, secure, and user-friendly HMS.

### **1. Functional Requirements**

Functional requirements define the core functionalities that the system must provide to meet its objectives.

#### **❖ Patient Management**

- Store and retrieve patient demographic details (name, age, gender, contact, address, etc.).
- Maintain medical history records, including diagnoses, treatments, and prescriptions.
- Allow real-time updates to patient records by authorized healthcare staff.
- Support data search functionality for quick retrieval of patient records.

### ❖ **Staff Management**

- Store details of hospital staff, including doctors, nurses, and administrative staff.
- Manage staff schedules, shifts, and availability.
- Maintain role-based access control for system features, based on staff roles and responsibilities.
- Generate and manage staff payroll based on shifts, hours worked, and roles,

### ❖ **Appointment Scheduling**

- Facilitate appointment booking, rescheduling, and cancellation by patients or administrative staff.
- Enable appointment reminders via SMS or email for patients.
- Provide a doctor availability overview to help with scheduling.
- Track patient attendance for appointments and update status accordingly.

### ❖ **Billing and Payments**

- Generate invoices for medical services, consultations, treatments, and procedures.
- Support different payment methods (cash, card, online payments).
- Track payment history for each patient, with a summary of outstanding balances.
- Provide insurance claim processing and integration with third-party insurers, if applicable.

## **2. Non-Functional Requirements**

Non-functional requirements describe system qualities, such as performance, usability, reliability, and security.

### ❖ **Performance**

- Ensure the system can handle concurrent users without lag, especially during peak hours.

- Optimize database queries for fast retrieval and update of patient records.
- Minimize load times for each module, aiming for less than 2 seconds per transaction.

#### ❖ **Usability**

- Design an intuitive, user-friendly interface for both technical and non-technical users.
- Provide role-based views (e.g., doctors, nurses, administrative staff) with easy access to the specific features they need.
- Ensure the system is accessible on desktop and mobile devices to allow remote access.

#### ❖ **Reliability**

- Implement system logging to track user actions and system changes for traceability.
- Ensure system uptime of at least 99.5% with minimal service disruptions.
- Back up patient and hospital data daily to prevent data loss.

#### ❖ **Security**

- Encrypt sensitive data, including patient records and payment information.
- Use multi-factor authentication (MFA) for access to sensitive features and data.
- Implement role-based access control to restrict access to certain system modules based on user roles.
- Comply with relevant healthcare data privacy regulations (e.g., HIPAA, GDPR).

#### ❖ **Scalability**

- Design the system to be scalable to support an increasing number of patients, staff, and data without degrading performance.
- Enable horizontal and vertical scaling of the database and application server as the hospital expands.

#### ❖ **Maintainability**

- Use modular coding practices to simplify updates and bug fixes.
- Provide comprehensive documentation for system components to aid future maintenance.
- Implement error handling and logging to quickly identify and resolve issues.

#### ❖ **Interoperability**

- Ensure compatibility with other healthcare systems, such as Electronic Health Record (EHR) systems and lab management software, through APIs.
- Facilitate data exchange between the HMS and external systems using standard formats like HL7 or FHIR.

#### ❖ **Audit and Compliance**

- Maintain an audit log of all user activities to ensure transparency and accountability.
- Comply with data protection regulations (HIPAA, GDPR) by storing audit trails and ensuring access to sensitive data is properly managed.

### **3. Technical Requirements**

- ❖ **Database:** Use MySQL for storing structured data related to patient records, staff information, billing, and inventory.
- ❖ **Backend:** Java for backend development to ensure stability, security, and compatibility with enterprise-level applications.
- ❖ **Frontend:** A user-friendly frontend made using Swing in Java

## ARCHITECTURE AND DESIGN

### Client-Server Architecture:

- **Java Swing Client Application:** The system uses Java Swing for building the client-side application, providing a desktop-based interface for hospital staff, including administrators, doctors, and nurses. The application has forms, panels, and layouts designed in Swing to handle user interactions and display information.
- **Server and Database:** The client application connects to a centralized server, where all business logic and data processing occur. This server communicates with a MySQL database to handle operations like patient registration, appointment scheduling, and billing. Java's networking and database libraries (JDBC) manage data requests and responses.

### Modular Design:

- The HMS follows a **modular structure** where each component of the system is divided into separate modules:
  - **Patient Management Module:** Swing-based forms and tables for adding new patients, updating records, and viewing patient histories.
  - **Appointment Scheduling Module:** A user-friendly interface with date and time selectors, allowing easy booking and tracking of appointments.
  - **Billing Module:** A billing form with automatic calculations and an invoice generation feature.
  - **Inventory Module:** Manages medical supplies, including adding new inventory, updating quantities, and setting reorder levels.

### Database Layer:

- The system uses a MySQL database to store and manage data, including patient records, appointments, inventory, and billing information.

## 1. Table Overview:

- **Patients Table:** Stores patient information with fields like patient ID (primary key), name, age, address, and medical history.
- **Doctors Table:** Contains doctor details, with a primary key doctor ID and additional fields like specialization and department.
- **Appointments Table:** Links patients and doctors with a one-to-many relationship, containing fields such as appointment ID (primary key), patient ID (foreign key), doctor ID (foreign key), and appointment date.
- **Billing Table:** Stores billing details with fields for bill ID, patient ID, amount, and date of billing.

## User Interface Design:

### ❖ User Roles:

- Different roles are established, such as **Pharmacist**, **Doctor**, **Receptionist** each with different permissions.
- Receptionists/Admins have access to all modules, while other roles have restricted access according to their responsibilities.

### ❖ UI Elements:

- **Navigation Panel:** A side or top panel for easy access to different modules.
- **Forms:** Input fields and data entry forms for each module, designed with Swing components (like JTextFields, JButtons, JComboBoxes).
- **Tables:** JTable is used to display lists, such as patient records, appointment schedules, and inventory items, with sorting and filtering capabilities.
- **Dialog Boxes:** Swing dialog boxes are used for alerts, confirmations, and notifications, providing feedback to users and guiding them through the workflow.



# IMPLEMENTATION

## LOGIN.JAVA

```
package hospital.management.system;

import java.sql.*;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.*;

public class LoginPage extends javax.swing.JFrame {

    public LoginPage() {

        initComponents();

        Connect();

    }

    Connection con;

    PreparedStatement pst;

    ResultSet rs;

    public void Connect()

String URL="jdbc:mysql://localhost/hospitalmanagement";

String Username = "root"

String Password = "";

{
```

```

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    con = DriverManager.getConnection(URL, Username, Password);
} catch (SQLException | ClassNotFoundException ex) {
    Logger.getLogger(UserCreation.class.getName()).log(Level.SEVERE, null, ex);
}
}

private void exitActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    int response = JOptionPane.showConfirmDialog(null,"Do you want to exit the
application?","Exit Confirmation",JOptionPane.YES_NO_OPTION);

    if(response == JOptionPane.YES_OPTION){
        System.exit(0);
    }
}

private void loginActionPerformed(java.awt.event.ActionEvent evt) {
    String username = txtusername.getText();
    String password = txtpassword.getText();
    String userType = txtusertype.getSelectedItem().toString();
    Connect();
}

```

```
try {  
    pst = con.prepareStatement("select * from user where username = ? and password = ? and  
usertype = ?");  
    pst.setString(1,username);  
    pst.setString(2, password);  
    pst.setString(3, userType);  
  
    rs = pst.executeQuery();  
  
    if(rs.next()){  
        if(userType == "Receptionist")  
        {  
            int userid = rs.getInt("id");  
            this.setVisible(false);  
            new Main(userid,username,userType).setVisible(true);  
        }  
        else if(userType == "Doctor")  
        {  
            int userid = rs.getInt("id");  
            this.setVisible(false);  
            new DoctorHomePage(userid,username,userType).setVisible(true);  
        }  
    }  
}
```

```

    }

    else{

        JOptionPane.showMessageDialog(this, "Invalid Username or Password");

        txtusername.setText("");

        txtpassword.setText("");

        txtusertype.setSelectedIndex(-1);

        txtusername.requestFocus();

    }

} catch (SQLException ex) {

    Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);

}

}

private void showPassActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    if(showPass.isSelected()){

        txtpassword.setEchoChar((char) 0);

    }

    else{

        txtpassword.setEchoChar('*');

    }

}

```

```

private void signUpActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    UserCreation u = new UserCreation();

    u.setVisible(true);

}

private void jLabel15MouseClicked(java.awt.event.MouseEvent evt) {

    // TODO add your handling code here:

    int response = JOptionPane.showConfirmDialog(null,"Do you want to exit the
application?","Exit Confirmation",JOptionPane.YES_NO_OPTION);

    if(response == JOptionPane.YES_OPTION){

        System.exit(0);

    }

}

private void txtusernameKeyPressed(java.awt.event.KeyEvent evt) {

    // TODO add your handling code here:

    if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_TAB){

        txtpassword.requestFocus();

    }

}

private void txtpasswordKeyPressed(java.awt.event.KeyEvent evt) {

    // TODO add your handling code here:

    if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_TAB){

        signUp.requestFocus();

    }

}

```

```

    }
}

private void signUpKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:

    if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_TAB){
        login.requestFocus();
    }

    else if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_ENTER){
        signUp.doClick();
    }
}

private void loginKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:

    if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_TAB){
        exit.requestFocus();
    }

    else if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_ENTER){
        login.doClick();
    }
}

private void exitKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:

    if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_TAB){

```

```

        txtusertype.requestFocus();
    }
    else if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_ENTER){
        exit.doClick();
    }
}

boolean cyclingDropdown = false;

private void txtusertypeKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if (evt.getKeyCode() == java.awt.event.KeyEvent.VK_ENTER) {
        // If not cycling yet, start cycling mode
        if (!cyclingDropdown) {
            cyclingDropdown = true; // Enable cycling mode
        } else {
            // If already in cycling mode, exit it and move focus to the next component
            cyclingDropdown = false; // Stop cycling mode
            txtusername.requestFocus(); // Move focus to the "login" button, or next
            component
        }
    }
}

else if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_TAB){
    int currentIndex = txtusertype.getSelectedIndex();
    int itemCount = txtusertype.getItemCount();

```

```
        if(currentIndex < itemCount - 1){

            txtusertype.setSelectedIndex(currentIndex + 1);

        }

        else{

            txtusertype.setSelectedIndex(0);

        }

    }

}

private void showPassKeyPressed(java.awt.event.KeyEvent evt) {

    // TODO add your handling code here:

    if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_ENTER){

        showPass.setSelected(!showPass.isSelected());

    }

    if (showPass.isSelected()) {

        txtpassword.setEchoChar((char) 0); // Show password

    } else {

        txtpassword.setEchoChar('*'); // Hide password

    }

}
```



## MAIN.JAVA

```
package hospital.management.system;

import javax.swing.JOptionPane;

public class Main extends javax.swing.JFrame {

    public Main() {

        initComponents();

    }

    int newid;

    String uname;

    String utype;


    public Main(int id, String username, String usertype) {

        initComponents();


        this.uname = username;

        jLabel4.setText(uname);

        this.utype = usertype;

        jLabel5.setText(utype);

        this.newid = id;

    }

    private void userCreateActionPerformed(java.awt.event.ActionEvent evt) {

        UserCreation u = new UserCreation();

        u.setVisible(true);}
```

```

private void logoutActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    int response = JOptionPane.showConfirmDialog(null,"Do you want to
logout?","Logout Confirmation",JOptionPane.YES_NO_OPTION);

    if(response == JOptionPane.YES_OPTION){

        this.setVisible(false);

        LoginPage l = new LoginPage();

        l.setVisible(true);

    }

}

```

```

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    Doctor d = new Doctor();

    d.setVisible(true);

}

```

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    Patient p = new Patient();

    p.setVisible(true);

}

```

```

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    Appointments ap = new Appointments();

    ap.setVisible(true);

}

private void jLabel6MouseClicked(java.awt.event.MouseEvent evt) {

    // TODO add your handling code here:

    int response = JOptionPane.showConfirmDialog(null,"Do you want to
logout?","Logout Confirmation",JOptionPane.YES_NO_OPTION);

    if(response == JOptionPane.YES_OPTION){

        this.setVisible(false);

        LoginPage l = new LoginPage();

        l.setVisible(true);

    }

}

private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    viewDoc obj = new viewDoc();

    obj.setVisible(true);

}

```

## **DOCTOR.JAVA**

```
package hospital.management.system;
```

```
import java.sql.*;
```

```
import java.util.Vector;
```

```
import java.util.logging.Level;
```

```
import java.util.logging.Logger;
```

```
import javax.swing.*;
```

```
import javax.swing.table.*;
```

```
public Doctor() {
```

```
    initComponents();
```

```
    Connect();
```

```
    AutoID();
```

```
    doctor_table();
```

```
}
```

```
Connection con;
```

```
PreparedStatement pst;
```

```
ResultSet rs;
```

```
public void Connect()
```

```
{
```

```

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

            con
            DriverManager.getConnection("jdbc:mysql://localhost/hospitalmanagement","root", "");
        } catch (ClassNotFoundException | SQLException ex) {

            java.util.logging.Logger.getLogger(Doctor.class.getName()).log(java.util.logging.Level.
            SEVERE, null, ex);

        }

    }

    public void AutoID(){

        try {

            Statement s = con.createStatement();

            rs = s.executeQuery("select MAX(doctorno)from doctor");

            rs.next();

            if(rs.getString("MAX(doctorno)")== null){

                doctorid.setText("D001");

            }

            else{

                long
                id
                Long.parseLong(rs.getString("MAX(doctorno)").substring(2,rs.getString("MAX(doctorn
                o)").length()));

                id++;
            }
        }
    }

```

```

        doctorid.setText("D"+String.format("%03d", id));
    }
} catch (SQLException ex) {
    Logger.getLogger(Doctor.class.getName()).log(Level.SEVERE, null, ex);
}
}

```

```

public void doctor_table(){

```

```

    try {
        pst = con.prepareStatement("select * from doctor");
        rs = pst.executeQuery();
        ResultSetMetaData Rsm = rs.getMetaData();
        int c;
        c = Rsm.getColumnCount();

        DefaultTableModel df = (DefaultTableModel) jTable1.getModel();
        df.setRowCount(0);
        while(rs.next()){
            Vector v2 = new Vector();

```

```

        for(int i = 1; i<=c; i++)
        {
            v2.add(rs.getString("doctorno"));
            v2.add(rs.getString("name"));
            v2.add(rs.getString("specialization"));
            v2.add(rs.getString("qualification"));
            v2.add(rs.getString("doctorfee"));
            v2.add(rs.getString("phone"));
            v2.add(rs.getString("roomno"));
        }
        df.addRow(v2);
    }

} catch (SQLException ex) {

    Logger.getLogger(Doctor.class.getName()).log(Level.SEVERE, null, ex);
}

}

private void addDoctorActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    String dno = doctorid.getText();

    String dname = txtdoctorname.getText();

    String dspecialization = txtdoctorspecialization.getText();

```

```

String dqualification = txtdoctorqualification.getText();

String dfees = txtdoctorfees.getText();

String dphone = txtdoctorphone.getText();

String droomno = roomno.getValue().toString();

if (dphone.length() != 10) {

    JOptionPane.showMessageDialog(null, "Phone number must be exactly 10
digits long.");

    return;

}

try {

    pst = con.prepareStatement("insert into doctor(doctorno, name, specialization,
qualification, doctorfee, phone, roomno) values(?,?,?,?,?,?,?)");

    pst.setString(1, dno);

    pst.setString(2, dname);

    pst.setString(3, dspecialization);

    pst.setString(4, dqualification);

    pst.setString(5, dfees);

    pst.setString(6, dphone);

    pst.setString(7, droomno);

    pst.executeUpdate();

    JOptionPane.showMessageDialog(null, "Doctor Inserted Successfully!");

    AutoID();

    txtdoctorname.setText("");

```



```

        txtdoctorspecialization.setText("");
        txtdoctorqualification.setText("");
        txtdoctorfees.setText("");
        txtdoctorphone.setText("");
        roomno.setValue(0);
        txtdoctorname.requestFocus();
        doctor_table();

    } catch (SQLException ex) {

        Logger.getLogger(Doctor.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:

    DefaultTableModel d1 = (DefaultTableModel)jTable1.getModel();
    int selectedIndex = jTable1.getSelectedRow();

    doctorid.setText(d1.getValueAt(selectedIndex, 0).toString());
    txtdoctorname.setText(d1.getValueAt(selectedIndex, 1).toString());
    txtdoctorspecialization.setText(d1.getValueAt(selectedIndex, 2).toString());
    txtdoctorqualification.setText(d1.getValueAt(selectedIndex, 3).toString());
    txtdoctorfees.setText(d1.getValueAt(selectedIndex, 4).toString());
    txtdoctorphone.setText(d1.getValueAt(selectedIndex, 5).toString());

    try {

```

```

        roomno.setValue(Integer.valueOf(d1.getValueAt(selectedIndex, 6).toString()));
    } catch (NumberFormatException e) {

        // Handle the case where room number is not a valid integer

        JOptionPane.showMessageDialog(null, "Invalid Room No");
    }

    addDoctor.setEnabled(false);
}

private void updateDoctorActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    String dno = doctorid.getText();

    String dname = txtdoctorname.getText();

    String dspecialization = txtdoctorspecialization.getText();

    String dqualification = txtdoctorqualification.getText();

    String dfees = txtdoctorfees.getText();

    String dphone = txtdoctorphone.getText();

    String droomno = roomno.getValue().toString();

    if (dno.isEmpty() || dname.isEmpty() || dspecialization.isEmpty() ||
dqualification.isEmpty() || dfees.isEmpty() || dphone.isEmpty() || droomno.isEmpty()) {

        JOptionPane.showMessageDialog(null, "Please fill all fields.", "Input Error",
JOptionPane.ERROR_MESSAGE);

        Return;

        try {

```

```
pst=con.prepareStatement("update doctor set name=?,specialization=?,qualification=?,  
doctorfee=?, phone=?, roomno=? where doctorno = ?");
```

```
pst.setString(1, dname);
```

```
pst.setString(2, dspecialization);
```

```
pst.setString(3, dqualification);
```

```
pst.setString(4, dfees);
```

```
pst.setString(5, dphone);
```

```
pst.setString(6, droomno);
```

```
pst.setString(7, dno);
```

```
int rowsAffected = pst.executeUpdate();
```

```
if(rowsAffected>0){
```

```
JOptionPane.showMessageDialog(null, "Doctor Updated Successfully!");
```

```
AutoID();
```

```
txtdoctorname.setText("");
```

```
txtdoctorspecialization.setText("");
```

```
txtdoctorqualification.setText("");
```

```
txtdoctorfees.setText("");
```

```
txtdoctorphone.setText("");
```

```
roomno.setValue(0);
```

```
txtdoctorname.requestFocus();
```

```
doctor_table();
```

```

        addDoctor.setEnabled(true);

    }

    else {

        JOptionPane.showMessageDialog(null, "No doctor found with the specified ID.",
"Update Error", JOptionPane.ERROR_MESSAGE);

    }

} catch (SQLException ex) {

    Logger.getLogger(Doctor.class.getName()).log(Level.SEVERE, null, ex);

}}

private void deleteDoctorActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    String dno = doctorid.getText();

    try {

        pst = con.prepareStatement("delete from doctor where doctorno = ?");

        pst.setString(1, dno);

        pst.executeUpdate();

        JOptionPane.showMessageDialog(null, "Doctor Deleted Successfully!");

```

```
AutoID();

txtdoctorname.setText("");

txtdoctorspecialization.setText("");

txtdoctorqualification.setText("");

txtdoctorfees.setText("");

txtdoctorphone.setText("");

roomno.setValue(0);

txtdoctorname.requestFocus();

doctor_table();

addDoctor.setEnabled(true);

} catch (SQLException ex) {

    Logger.getLogger(Doctor.class.getName()).log(Level.SEVERE, null, ex);

}

}

private void exitActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    this.setVisible(false);

}
```

## **PATIENT.JAVA**

```
package hospital.management.system;

import com.mysql.cj.jdbc.exceptions.MysqlDataTruncation;
import java.sql.*;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;
import javax.swing.table.*;

public class Patient extends javax.swing.JFrame {

    public Patient() {

        initComponents();

        Connect();

        AutoID();

        patient_table();

    }

    Connection con;

    PreparedStatement pst;

    ResultSet rs;
```

```

public void Connect()
{
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");

        con
        DriverManager.getConnection("jdbc:mysql://localhost/hospitalmanagement","root", "");
    } catch (ClassNotFoundException | SQLException ex) {

java.util.logging.Logger.getLogger(Patient.class.getName()).log(java.util.logging.Level.
SEVERE, null, ex);

    }
}

public void AutoID(){
    try {
        Statement s = con.createStatement();

        rs = s.executeQuery("select MAX(patientno)from patient");

        rs.next();

        if(rs.getString("MAX(patientno)")== null){

            jLabel5.setText("PS001");

        }
    }
}

```

```

else{

long id =
Long.parseLong(rs.getString("MAX(patientno)").substring(2,rs.getString("MAX(patient
no)").length()));

        id++;

        jLabel5.setText("PS"+String.format("%03d", id));

    }

} catch (SQLException ex) {

    Logger.getLogger(Patient.class.getName()).log(Level.SEVERE, null, ex);

}

}

public void patient_table(){

    try {

        pst = con.prepareStatement("select * from patient");

        rs = pst.executeQuery();

        ResultSetMetaData Rsm = rs.getMetaData();

        int c;

        c = Rsm.getColumnCount();

        DefaultTableModel df = (DefaultTableModel) jTable1.getModel();

        df.setRowCount(0);

```



```

while(rs.next()){

    Vector v2 = new Vector();

    for(int i = 0; i<=c; i++)

    {

        v2.add(rs.getString("patientno"));

        v2.add(rs.getString("name"));

        v2.add(rs.getString("phone"));

        v2.add(rs.getString("address"));

    }

    df.addRow(v2);

}

} catch (SQLException ex) {

    Logger.getLogger(Patient.class.getName()).log(Level.SEVERE, null, ex);

}

}

```

```

private void addPatientActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    String pno = jLabel5.getText();

    String pname = txtpatientname.getText();

    String phone = txtpatientphone.getText();

```

```

String address = txtpatientaddress.getText();

try {
    if (phone.length() > 10) {
        JOptionPane.showMessageDialog(null, "Phone number is too long! Please enter
a valid phone number.");
        return;
    }

    pst = con.prepareStatement("insert into patient(patientno, name, phone, address)
values(?,?,?,?)");

    pst.setString(1, pno);
    pst.setString(2, pname);
    pst.setString(3, phone);
    pst.setString(4, address);
    pst.executeUpdate();

    JOptionPane.showMessageDialog(null, "Patient Inserted Successfully!");

    AutoID();
    txtpatientname.setText("");
    txtpatientphone.setText("");
    txtpatientaddress.setText("");
    txtpatientname.requestFocus();

```

```

        patient_table();

    } catch (MysqlDataTruncation ex) {

        JOptionPane.showMessageDialog(null, "Phone number is too long! Please enter a
valid phone number.");

    } catch (SQLException ex) {

        Logger.getLogger(Patient.class.getName()).log(Level.SEVERE, null, ex);

    }

}

private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {

    // TODO add your handling code here:

    DefaultTableModel d1 = (DefaultTableModel)jTable1.getModel();

    int selectedIndex = jTable1.getSelectedRow();

    jLabel5.setText(d1.getValueAt(selectedIndex, 0).toString());

    txtpatientname.setText(d1.getValueAt(selectedIndex, 1).toString());

    txtpatientphone.setText(d1.getValueAt(selectedIndex, 2).toString());

    txtpatientaddress.setText(d1.getValueAt(selectedIndex, 3).toString());

    addPatient.setEnabled(false);

}

```

```

private void updateActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    String pno = jLabel5.getText();

    String pname = txtpatientname.getText();

    String phone = txtpatientphone.getText();

    String address = txtpatientaddress.getText();

    try {

        pst = con.prepareStatement("update patient set name=?,phone=?,address=? where
patientno = ?");

        pst.setString(1, pname);

        pst.setString(2, phone);

        pst.setString(3, address);

        pst.setString(4, pno);

        pst.executeUpdate();

        JOptionPane.showMessageDialog(null, "Patient Updated Successfully!");

        AutoID();

        txtpatientname.setText("");

        txtpatientphone.setText("");

        txtpatientaddress.setText("");

        txtpatientname.requestFocus();

        patient_table();

        addPatient.setEnabled(true);

    } catch (SQLException ex) {

```

```

        Logger.getLogger(Patient.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private void deleteActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    String pno = jLabel5.getText();

    try {

        pst = con.prepareStatement("delete from patient where patientno = ?");
        pst.setString(1, pno);
        pst.executeUpdate();

        JOptionPane.showMessageDialog(null, "Patient Deleted Successfully!");

        AutoID();

        txtpatientname.setText("");
        txtpatientphone.setText("");
        txtpatientaddress.setText("");
        txtpatientname.requestFocus();

        patient_table();

        addPatient.setEnabled(true);

    } catch (SQLException ex) {

        Logger.getLogger(Patient.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

## **APPOINTMENT.JAVA**

```
package hospital.management.system;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.ResultSetMetaData;

import java.sql.SQLException;

import java.sql.Statement;

import java.text.SimpleDateFormat;

import java.util.Vector;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.JOptionPane;

import javax.swing.table.DefaultTableModel;

public class Appointments extends javax.swing.JFrame {

    public Appointments() {

        initComponents();

        Connect();

        AutoID();

        loadDoctor();

        loadPatient();

        app_table();
```

```
}

Connection con;

PreparedStatement pst;

ResultSet rs;

String apno;

public class Doctor{

    String id;

    String name;

    public Doctor(String id, String name){

        this.id = id;

        this.name = name;

    }

    public String toString(){

        return name;

    }

}

public class Patient{

    String id;

    String name;

    public Patient(String id, String name){

        this.id = id;

        this.name = name;

    }

}
```

```

    public String toString(){
        return name;
    }
}

public void loadDoctor(){
    try {
        pst = con.prepareStatement("select * from doctor");
        rs = pst.executeQuery();
        doctorcb.removeAll();

        while(rs.next()){
            doctorcb.addItem(new Doctor(rs.getString(1), rs.getString(2)));
        }
    } catch (SQLException ex) {
        Logger.getLogger(Appointments.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public void loadPatient(){
    try {
        pst = con.prepareStatement("select * from patient");
        rs = pst.executeQuery();
        patientcb.removeAll();
    }
}

```



```

        while(rs.next()){

            patientcb.addItem(new Patient(rs.getString(1), rs.getString(2)));

        }
    } catch (SQLException ex) {

        Logger.getLogger(Appointments.class.getName()).log(Level.SEVERE, null, ex);

    }

}

public void Connect()

{

    try {

        Class.forName("com.mysql.cj.jdbc.Driver");

con =
DriverManager.getConnection("jdbc:mysql://localhost/hospitalmanagement","root", "");

        } catch (SQLException | ClassNotFoundException ex) {

            Logger.getLogger(UserCreation.class.getName()).log(Level.SEVERE, null, ex);

        }

    }

    public void AutoID(){

        try {

            Statement s = con.createStatement();

            rs = s.executeQuery("select MAX(appointmentno)from appointment");

            rs.next();

```

```

        if(rs.getString("MAX(appointmentno)")== null){

            appointmentnotxt.setText("AP001");

        }

        else{

long id =
Long.parseLong(rs.getString("MAX(appointmentno)").substring(2,rs.getString("MAX(a
ppointmentno)").length()));

            id++;

            appointmentnotxt.setText("AP"+String.format("%03d", id));

        }

    } catch (SQLException ex) {

        Logger.getLogger(Doctor.class.getName()).log(Level.SEVERE, null, ex);

    }

}

public void app_table(){

    try {

        pst = con.prepareStatement("select * from appointment");

        rs = pst.executeQuery();

        ResultSetMetaData Rsm = rs.getMetaData();

        int c;

        c = Rsm.getColumnCount();

        DefaultTableModel df = (DefaultTableModel) jTable1.getModel();

        df.setRowCount(0);

```

```

while(rs.next()){

    Vector v2 = new Vector();

    for(int i = 1; i<=c; i++)

    {

        v2.add(rs.getString("appointmentno"));

        v2.add(rs.getString("doctorname"));

        v2.add(rs.getString("patientname"));

        v2.add(rs.getString("roomno"));

        v2.add(rs.getString("date"));

    }

    df.addRow(v2);

}

} catch (SQLException ex) {

```

```

Logger.getLogger(hospital.management.system.Doctor.class.getName()).log(Level.SEVERE, null, ex);

```

```

}

}

```

```

private void addappActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    String appno = appointmentnotxt.getText();

    Doctor d = (Doctor) doctorcb.getSelectedItemAt();

    Patient p = (Patient) patientcb.getSelectedItemAt();

    String roomno = roomnosp.getValue().toString();

```

```

SimpleDateFormat dateformat = new SimpleDateFormat("yyyy-MM-dd");

String date = dateformat.format(txtdate.getDate());

try {

    pst = con.prepareStatement("insert into appointment(appointmentno, doctorname,
patientname , roomno, date) values(?,?,?,?,?)");

    pst.setString(1, appno);

    pst.setString(2, d.name);

    pst.setString(3, p.name);

    pst.setString(4, roomno);

    pst.setString(5, date);

    pst.executeUpdate();

    JOptionPane.showMessageDialog(null, "Appointment Created");

    AutoID();

    appointmentnotxt.setText("");

    doctorcb.setSelectedIndex(-1);

    patientcb.setSelectedIndex(-1);

    roomnosp.setValue(0);

    app_table();

} catch (SQLException ex) {

    Logger.getLogger(hospital.management.system.Doctor.class.getName()).log(Level.SEVERE, null, ex);

}

}

```

```

private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:

    DefaultTableModel d1 = (DefaultTableModel) jTable1.getModel();

    int selectedIndex = jTable1.getSelectedRow();

    apno = d1.getValueAt(selectedIndex, 0).toString();

    //JOptionPane.showMessageDialog(this, apno);
}

private void removebtnActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        pst = con.prepareStatement("delete from appointment where appointmentno=?");
        pst.setString(1, apno);
        pst.executeUpdate();

        JOptionPane.showMessageDialog(null, "Appointment Deleted");

        AutoID();

        appointmentnotxt.setText("");

        doctorcb.setSelectedIndex(-1);

        patientcb.setSelectedIndex(-1);

        roomnosp.setValue(0);

        app_table();

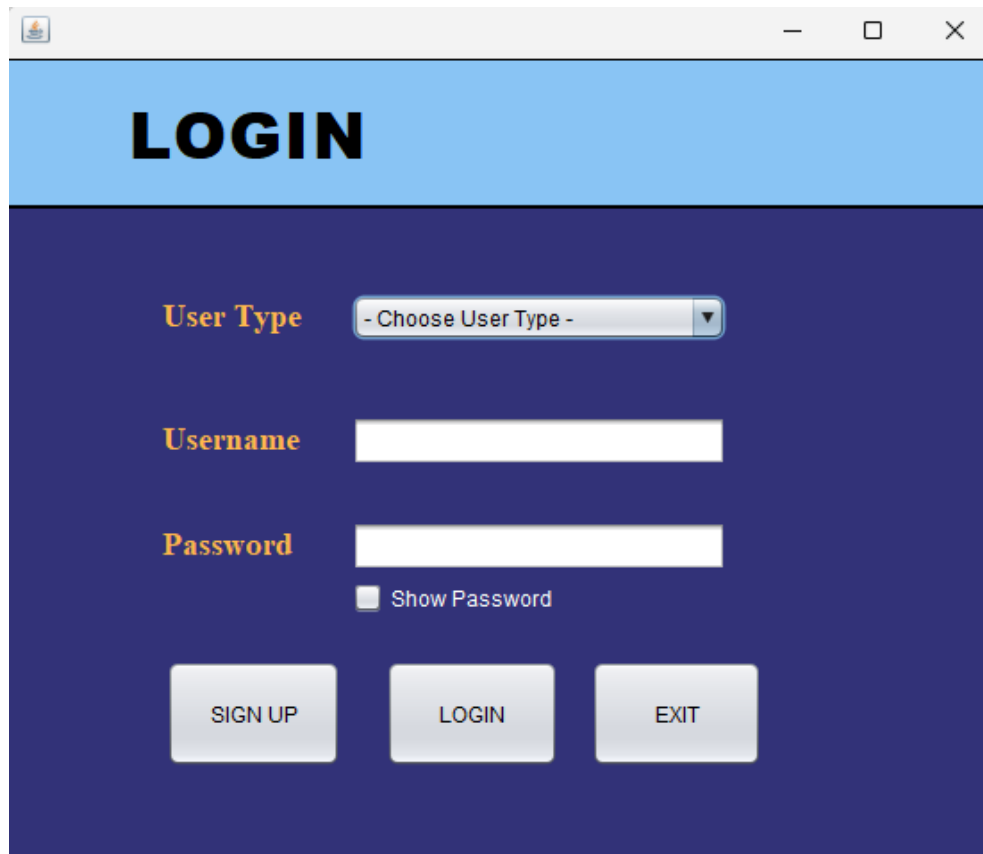
    } catch (SQLException ex) {

        Logger.getLogger(hospital.management.system.Doctor.class.getName()).log(Level.SEVERE, null, ex);

    }
}

```

## OUTPUT




A screenshot of a web application window titled "LOGIN". The window has a light blue header bar with the word "LOGIN" in bold black text. The main content area has a dark blue background. It contains three labels in orange: "User Type", "Username", and "Password". The "User Type" label is next to a dropdown menu showing "- Choose User Type -". The "Username" and "Password" labels are next to white text input fields. Below the "Password" field is a checkbox labeled "Show Password". At the bottom, there are three light gray buttons: "SIGN UP", "LOGIN", and "EXIT".



A screenshot of a web application window titled "USER CREATION". The window has a black header bar with the words "USER CREATION" in bold white text. The main content area has a light gray background. It contains three labels in black: "UserType", "Username", and "Password". The "UserType" label is next to a dropdown menu showing "Doctor". The "Username" label is next to a white text input field containing "Rithish". The "Password" label is next to a white text input field containing "\*\*\*\*". At the bottom, there are two light gray buttons: "Create" and "Back".

<div>Patient</div> <div>Doctor</div> <div>Schedule Appoinment</div> <div>Create User</div> <div>View Doctor</div> <div>Logout</div>	<div>HEALTHNET PRO - HM SYSTEM</div> <div><div>Username</div>abc</div> <div><div>User Type</div>Receptionist</div>
---	--



—

□

×

HOSPITAL MANAGEMENT SYSTEM

Username

Rithish

User Type

Doctor

Doctor

View Doctor

View Appointments

Patient

Logout

PATIENT REGISTRATION

Patient Registration

Patient No.

PS002

Patient Name

Tarun

Phone

7358165739

Address

Chennai

Patient No	Patient Name	Phone	Address
7358013490	Chennai		

Message

i

Patient Inserted Successfully!

OK

ADD PATIENT

UPDATE

DELETE

EXIT

PATIENT REGISTRATION

Patient Registration

Patient No.

PS001

Patient Name

Manish

Phone

7358013490

Address

Chennai

Patient No	Patient Name	Phone	Address
PS001	Manish	7358013490	Chennai

ADD PATIENT

UPDATE

DELETE

EXIT



DOCTOR REGISTRATION

Doctor Registration

Doctor No.

D001

Doctor Name

Rithish

Specialization

General

Qualification

MBBS

Doctor Fees

1000

Phone No.

1234567890

Room No.

2

Doctor No	Doctor Name	Specialization	Qualification	Doctor Fees	Phone No	Room No

Message

i

Doctor Inserted Successfully!

OK

ADD DOCTOR

UPDATE

DELETE

EXIT

Appointments

Add Appointment

Appointment No

AP001

Doctor Name

Rithish

Patient Name

Manish

Room No

3

Appointment Date

14 Nov 2024

Add Appointmemnt

Remove Appointment

Appointment No	Doctor Name	Patient Name	Room No	Date

Message

i

Appointment Created

OK

# EXPERIMENT RESULT AND ANALYSIS

## Functional Testing:

1. **Patient Management:** Tested operations for adding, updating, and deleting patient records. All functions performed as expected, with data correctly stored and retrieved from the database.
2. **Appointment Scheduling:** Verified that appointments can be created, modified, and deleted. Conflicts in appointment times for the same doctor were effectively managed, preventing double bookings.
3. **Billing Module:** Assessed accuracy in generating invoices and calculating costs. The system successfully handled various billing scenarios, such as discounts, taxes, and refunds.
4. **Inventory Management:** Checked the addition and updating of items. Reorder notifications were correctly triggered when item levels fell below set thresholds.

## Performance Testing:

1. **Response Time:** Evaluated the response time for database transactions (retrieving patient data, booking appointments, generating bills). Average response time was under acceptable limits, with minimal delay in data loading for most queries.
2. **Scalability:** Simulated high loads by increasing the number of records and concurrent users. The system maintained stability, although response time slightly increased under maximum load.
3. **Memory Usage:** Monitored memory usage during typical operations. Memory usage remained within manageable levels, with occasional spikes during large transactions, which were promptly released after operation completion.

## **User Interface Usability Testing:**

1. **Navigation:** Users were able to navigate the Swing-based interface intuitively. Feedback indicated a positive user experience with clearly labeled buttons, tables, and forms.
2. **Role-Based Access:** Tested the system's ability to restrict access based on user roles (Administrator, Doctor, Receptionist, Nurse). All roles successfully accessed only their permitted functionalities.
3. **Error Handling and Feedback:** The system effectively displayed error messages and validations, such as alerting users for incorrect data formats or empty required fields.

## **Database Performance:**

1. **Query Optimization:** Checked database query performance. Optimizations such as indexing frequently accessed tables (Patients, Appointments) improved data retrieval times.
2. **Data Consistency:** The system maintained data consistency across multiple modules (e.g., a deleted patient record removed corresponding appointment data).

## **Results Summary:**

1. **Strengths:** The system performed well in terms of functionality, reliability, and data integrity. User feedback was positive regarding ease of use and role-based access control.
2. **Weaknesses:** Response times slightly increased under high loads, indicating potential need for further optimization as the system scales.
3. **Recommendations for Improvement:**
  - Consider implementing caching for frequently accessed data.
  - Explore potential upgrades in database indexing and optimization to handle larger data loads.

## **FUTURE SCOPE**

As the current HMS provides a robust foundation for managing various hospital operations, future enhancements and expansions can add significant value and improve functionality, scalability, and user experience. Below are some potential areas for future development:

### **Integration with Electronic Health Records (EHR):**

- Expanding the system to include EHR integration would allow the system to handle comprehensive medical records, including lab results, imaging, prescriptions, and clinical notes.
- EHR integration can support seamless data sharing across healthcare providers, improving patient outcomes and care continuity.

### **Cloud Deployment and Scalability:**

- Migrating the HMS to a cloud-based platform can enhance accessibility, allowing staff to access the system from any location and device.
- Cloud deployment will also improve scalability, making it easier to handle large amounts of data and support multiple hospital branches or clinics under the same system.

### **Mobile Application Development:**

- Developing a mobile application for patients and healthcare providers would increase convenience and accessibility.
- Features for patients could include appointment scheduling, notifications, viewing medical history, and receiving reminders. For healthcare providers, it could offer quick access to patient data and schedule management on the go.

### **Advanced Data Analytics and Reporting:**

- Adding an analytics module to generate detailed reports on patient flow, resource utilization, and financial performance can support data-driven decision-making.
- Predictive analytics could help in areas such as forecasting patient volume, inventory needs, and identifying patterns in patient care to improve hospital management.

### **Telemedicine and Remote Consultation Integration:**

- Incorporating telemedicine features would allow doctors to conduct remote consultations, which can be beneficial for follow-up visits, consultations, and non-emergency cases.
- Patients could book virtual appointments and access healthcare from home, increasing accessibility, especially for those in remote areas.

### **Patient Self-Service Portal:**

- Developing a self-service web or mobile portal for patients would enable them to view their medical records, schedule appointments, make payments, and communicate with healthcare providers directly.
- This could improve patient engagement and satisfaction by giving them more control over their healthcare experience.

### **Integration with Wearable Devices and IoT:**

- Integrating with wearable health devices and IoT (Internet of Things) technology would allow real-time monitoring of patient vitals, especially for chronic disease management.
- This data can be automatically sent to the HMS, providing healthcare providers with real-time insights into patient health.

### **Enhanced Security Measures:**

- As patient data privacy is critical, future updates could focus on strengthening security with multi-factor authentication, data encryption, and compliance with healthcare standards like HIPAA.
- Adding blockchain technology for secure and immutable data storage could also be explored to further enhance data security.

### **Artificial Intelligence (AI) and Machine Learning (ML) Integration:**

- AI and ML can be utilized to enhance diagnostic support, predict patient outcomes, and assist in decision-making.
- These technologies can also optimize scheduling, predict inventory needs, and personalize patient care based on historical data trends.

### **Multi-Language and Localization Support:**

- Adding support for multiple languages and localizations will make the HMS more accessible for users in different regions and countries.
- This could be especially useful for hospitals operating in multicultural areas or regions with diverse linguistic requirements.

### **Automated Workflows and RPA (Robotic Process Automation):**

- Implementing RPA for routine administrative tasks, such as billing, claims processing, and inventory restocking, can improve efficiency and reduce manual effort.
- Automated workflows could also streamline patient intake, discharge processes, and follow-up reminders.

### **Interoperability with Other Health Systems:**

- Future development could focus on making the HMS interoperable with other healthcare systems, such as insurance providers, laboratory systems, and pharmacies.
- Interoperability will improve collaboration across the healthcare ecosystem, enabling more comprehensive care for patients.

## CONCLUSION

The development of the Hospital Management System (HMS) represents a significant step towards improving hospital operations and patient care. By automating various processes such as patient management, appointment scheduling, billing, and inventory control, the system reduces human error and enhances the efficiency of daily hospital functions. The system's modular design ensures that different departments can work independently while sharing crucial data in real-time, thus improving overall workflow. Furthermore, the user-friendly interface ensures ease of access for hospital staff, making it easier for them to manage tasks efficiently.

Despite its robustness, there is significant potential for future growth and expansion. Integrating advanced technologies like cloud computing, telemedicine, AI, and machine learning can enhance the system's capabilities, improve patient care, and facilitate better decision-making. As the healthcare industry continues to evolve, the HMS can adapt and scale to meet the changing needs of healthcare providers and patients alike.

## REFERENCES

- ❖ Patel, V., & Thomas, P. (2020). **Hospital Information Systems: A Review and Analysis**. *International Journal of Computer Science and Information Technologies*, 11(1), 45-52.
- ❖ Gupta, S. (2021). **Development of Hospital Management System Using Java and MySQL**. *Journal of Computer Applications*, 38(7), 123-132.
- ❖ HealthIT.gov - <https://www.healthit.gov/>