**HOSTEL MANAGEMENT SYSTEM**

**MINI PROJECT REPORT**

**Submitted by**

# Sachin Chaudhary 230701402

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2024 - 25

# BONAFIDE CERTIFICATE

Certified that this project report “**HOSTEL MANAGEMENT SYSTEM**” is the bonafide work of

**“SACHIN CHAUDHARY (230701402)”** who carried out the project work under my supervision.

## Submitted for the Practical Examination held on

**SIGNATURE**

**Mrs.Dharshini.B.S**

**Assistant Professor**,

**Computer Science and Engineering,**

**Rajalakshmi Engineerin**g **College Thandalam, Chennai - 602 105**

## INTERNAL EXAMINER EXTERNAL EXAMINER

**ABSTRACT:**

The **Hostel Management System** is a comprehensive tool designed to streamline and optimize the management of hostel facilities and resident information. This system facilitates efficient tracking of student accommodation, room allocation, and other hostel-related records, providing a centralized solution for administrators to manage and monitor hostel activities. By offering a clear overview of resident details, room assignments, attendance, and other hostel information, the system enables informed decision-making for both hostel staff and residents.

Through its structured organization of hostel data and real-time information capabilities, the Hostel Management System ensures that hostel operations run smoothly and efficiently. This fosters better accommodation planning, aids in progress tracking, and encourages improved resident engagement. Users can add, update, and view resident details, allocate rooms, and generate reports, maintaining transparency and accountability within the hostel environment.

The system features an intuitive user interface that includes an admin dashboard, enabling administrators to oversee and manage resident accounts, monitor accommodation status, and analyze trends in room occupancy and hostel performance. With advanced security protocols and efficient database management, the system ensures that all resident information is accurate, secure, and up-to-date.

Overall, the Hostel Management System enhances the efficiency of hostel management processes and empowers hostel staff and residents to achieve greater comfort, satisfaction, and collaboration within the accommodation environment.

## TABLE OF CONTENTS

### Chapter 1 1 INTRODUCTION

1.1 Introduction ..................................................... 6-8

1.2 Objectives ....................................................... 9

1.3 Modules ......................................................... 10

### Chapter 2 2 SURVEY OF TECHNOLOGIES

2.1 Software Description .................................... 11-13

2.2 Languages .................................................... 14

2.2.1 Java .......................................................... 14-15

2.2.2 SQL ........................................................... 15

### Chapter 3 3 REQUIREMENTS AND ANALYSIS

3.1 Requirement Specification ............................ 16-19

3.1.1 Functional Requirements ......................... 19-21

3.2 Hardware and Software Requirements ........... 22-24

**Chapter 4 4 PROGRAM CODE**

4.1 Program Code ............................................ 26-35

**Chapter 5 5 RESULTS AND DISCUSSION**

5.1 Results and Discussion ................................ 36-42

**Chapter 6 6 CONCLUSION**

6.1 Conclusion ................................................... 43-44

**Chapter 7 7 REFERENCES**

7.1 References ................................................... 45-47

### Chapter 1: INTRODUCTION

### Effective hostel management is essential for academic institutions to ensure smooth operations, maintain accurate accommodation records, and provide a positive experience for residents and staff. The Hostel Management System offers a streamlined, user-friendly platform designed to securely and efficiently manage resident information. With a comprehensive suite of tools, administrators can monitor resident records, room assignments, and accommodation data with ease.

### The system provides core functionalities, including resident registration, secure login, and accommodation record management (room assignments, attendance, and fee details). Additionally, it maintains essential information such as personal details, accommodation history, and room status, ensuring all hostel-related data is accessible in one place. These features empower administrators and residents with a holistic view of accommodation information, fostering better decision-making and collaboration.

### Built with Java and MySQL, the Hostel Management System leverages Java for backend logic and an intuitive GUI developed using JFrame. MySQL supports data storage and retrieval, offering a secure and reliable foundation for managing resident records. The Java Swing-based interface enhances usability, delivering a modern and interactive experience for users.

### This report details the system's development, architecture, and technology integration, demonstrating how Java, MySQL, and NetBeans combine to create a secure and efficient hostel management solution. The system aims to provide a seamless, reliable user experience to address hostel management needs with industry-standard performance.

### 1.2 OBJECTIVES

### To develop a centralized database for securely managing resident profiles, room assignments, and accommodation records.

### To enable efficient handling of room allocation, attendance, and fees, providing real-time updates for residents and administrators.

### To provide a secure login system for authentication, ensuring the confidentiality of resident data.

### To allow users to easily access and view essential resident details, including personal information, accommodation status, and room assignments, on a streamlined interface.

### To ensure compliance with data security standards for information storage and management.

### 1.3 MODULES

### Resident Registration Module This module captures essential resident information, such as name, email, password, and room allocation. Data is securely stored in the database. This module ensures that only authorized users can access resident records by implementing credential storage with secure encryption and validation mechanisms.

### Profile Management Module This module provides a centralized view for users to access and manage resident profiles. Administrators can view critical information such as resident IDs, allocated rooms, fee status, and attendance. The system ensures that data is accurate and secure with robust authentication mechanisms.

### Room Allocation Module This module facilitates room assignment and tracking. Administrators can assign rooms to residents, and residents can view their room details in real-time.

### Attendance Management Module This module facilitates attendance tracking for residents. Administrators can update attendance records, and residents can view their attendance status in real-time.

### Fee Management Module This module records fee details and payment status for each resident. Administrators can update fee records, while residents can view their fee details and payment history.

### Admin Dashboard Module The dashboard provides administrators with tools to oversee resident records, monitor accommodation status, and generate reports on room occupancy, attendance, and fees. Role-based access ensures data security and restricts unauthorized access.

### Database Management Module This module is responsible for securely storing and retrieving all resident, room, and accommodation data. MySQL serves as the backend database, ensuring efficient data management and high reliability.

### Chapter 2: SURVEY OF TECHNOLOGIES

### 2.1 SOFTWARE DESCRIPTION

### The Hostel Management System utilizes a combination of technologies to ensure robust and efficient functionality. The backend is supported by a relational database management system (RDBMS), while the frontend features an interactive and user-friendly interface built using Java Swing. Middleware technologies enable seamless communication between the backend and frontend.

### 2.1.1 Java

### Role: Java serves as the primary programming language for both backend logic and GUI development.

### Usage:

### Backend: Handles operations like resident registration, room allocation, attendance, and fee management.

### Frontend: A JFrame-based GUI provides an intuitive and interactive user experience.

### Middleware: Java Database Connectivity (JDBC) ensures seamless communication with the MySQL database.

### Advantages:

### Platform independence for cross-platform compatibility.

### Built-in security features to protect sensitive resident data.

### 2.1.2 MySQL

### Role: MySQL is used as the relational database for storing and managing all resident-related information.

### Usage:

### Stores resident profiles, room assignments, attendance, fee records, and accommodation history.

### Efficient SQL queries enable quick retrieval and management of large datasets.

### Advantages:

### Reliable, open-source database management.

### Ensures data integrity and supports complex queries for accommodation reporting.

### Chapter 3: REQUIREMENTS AND ANALYSIS

### 3.1 REQUIREMENT SPECIFICATION

### 3.1.1 Functional Requirements

### User Authentication and Authorization

### Enable secure resident registration and login.

### Maintain session details for logged-in users.

### Resident Registration and Profile Management

### Allow creation and updating of resident profiles with personal and accommodation information.

### Provide a home screen displaying resident details, such as name, resident ID, allocated room, and attendance.

### Room Allocation and Attendance Management

### Enable real-time tracking of room allocations and resident attendance.

### Update and display attendance immediately after input by administrators.

### Fee Management

### Enable tracking of fees and payment status for each resident.

### Provide a real-time view of payment details and history.

### Database Records Management

### Use unique identifiers (e.g., resident ID) for managing records.

### Separate tables for resident profiles, room assignments, attendance, fees, and other hostel data for efficient management.

### 3.2 HARDWARE AND SOFTWARE REQUIREMENTS

### Hardware Requirements

### Processor: Intel Core i3 or equivalent for smooth processing.

### RAM: 4 GB or higher to handle concurrent database operations.

### Storage: At least 500 MB for application files and database storage.

### Monitor Resolution: 1024 x 768 or higher.

### Software Requirements

### Operating System: Windows 10 or higher.

### Frontend: Java Swing (JFrame-based interface).

### Backend: MySQL for database management.

### IDE: NetBeans for development.

### Version Control: Git for code versioning and collaboration.

### Chapter 4 :PROGRAM CODE

#### 1. Login And Signup Page

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class LoginPage extends JFrame {

    private JLabel userLabel, passLabel, messageLabel;

    private JTextField userTextField;

    private JPasswordField passField;

    private JButton loginButton, signupButton;

    public LoginPage() {

        // Set up the JFrame

        setTitle("Login Page");

        setSize(400, 200);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLocationRelativeTo(null); // Center the window

        setLayout(new GridLayout(4, 2, 10, 10)); // Grid layout

        // Create and add components

        userLabel = new JLabel("Username:");

        passLabel = new JLabel("Password:");

        messageLabel = new JLabel("", SwingConstants.CENTER);

        messageLabel.setForeground(Color.RED);

        userTextField = new JTextField();

        passField = new JPasswordField();

        loginButton = new JButton("Login");

        signupButton = new JButton("Signup");

        add(userLabel);

        add(userTextField);

        add(passLabel);

        add(passField);

        add(new JLabel()); // Empty space

        add(messageLabel);

        add(loginButton);

        add(signupButton);

        // Action listeners

        loginButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                String username = userTextField.getText();

                String password = String.valueOf(passField.getPassword());

                // Validate credentials

                if (LoginSignupApp.users.containsKey(username) &&

                    LoginSignupApp.users.get(username).equals(password)) {

                    messageLabel.setText("Login Successful!");

                    messageLabel.setForeground(Color.GREEN);

                    dispose(); // Close the Login Page

                    new HomePage(username); // Open Home Page

                } else {

                    messageLabel.setText("Invalid Username or Password!");

                    messageLabel.setForeground(Color.RED);

                }

            }

        });

        signupButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                dispose(); // Close the Login Page

                new SignupPage(); // Open Signup Page

            }

        });

        // Make the window visible

        setVisible(true);

    }

}

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class SignupPage extends JFrame {

    private JLabel userLabel, passLabel, confirmPassLabel, messageLabel;

    private JTextField userTextField;

    private JPasswordField passField, confirmPassField;

    private JButton signupButton, backButton;

    public SignupPage() {

        // Set up the JFrame

        setTitle("Signup Page");

        setSize(400, 300);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLocationRelativeTo(null); // Center the window

        setLayout(new GridLayout(5, 2, 10, 10)); // Grid layout

        // Create and add components

        userLabel = new JLabel("Username:");

        passLabel = new JLabel("Password:");

        confirmPassLabel = new JLabel("Confirm Password:");

        messageLabel = new JLabel("", SwingConstants.CENTER);

        messageLabel.setForeground(Color.RED);

        userTextField = new JTextField();

        passField = new JPasswordField();

        confirmPassField = new JPasswordField();

        signupButton = new JButton("Signup");

        backButton = new JButton("Back to Login");

        add(userLabel);

        add(userTextField);

        add(passLabel);

        add(passField);

        add(confirmPassLabel);

        add(confirmPassField);

        add(new JLabel()); // Empty space

        add(messageLabel);

        add(signupButton);

        add(backButton);

        // Action listeners

        signupButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                String username = userTextField.getText();

                String password = String.valueOf(passField.getPassword());

                String confirmPassword = String.valueOf(confirmPassField.getPassword());

                if (username.isEmpty() || password.isEmpty()) {

                    messageLabel.setText("Fields cannot be empty!");

                    messageLabel.setForeground(Color.RED);

                } else if (!password.equals(confirmPassword)) {

                    messageLabel.setText("Passwords do not match!");

                    messageLabel.setForeground(Color.RED);

                } else if (LoginSignupApp.users.containsKey(username)) {

                    messageLabel.setText("Username already exists!");

                    messageLabel.setForeground(Color.RED);

                } else {

                    // Save the new user

                    LoginSignupApp.users.put(username, password);

                    messageLabel.setText("Signup Successful!");

                    messageLabel.setForeground(Color.GREEN);

                }

            }

        });

        backButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                dispose(); // Close the Signup Page

                new LoginPage(); // Open Login Page

            }

        });

        // Make the window visible

        setVisible(true);

    }

}

#### 2. Home Page

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

public class HomePage extends JFrame {

    public HomePage(String username) {

        // Set up the JFrame

        setTitle("Home Page");

        setSize(600, 400); // Increased size for the table view

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLocationRelativeTo(null); // Center the window

        setLayout(new BorderLayout());

        // Welcome message

        JLabel welcomeLabel = new JLabel("Welcome, " + username + "!", SwingConstants.CENTER);

        // Panel for buttons

        JPanel buttonPanel = new JPanel();

        buttonPanel.setLayout(new GridLayout(2, 2, 10, 10));

        // Buttons

        JButton homeButton = new JButton("Home");

        JButton editButton = new JButton("Edit");

        JButton listButton = new JButton("List");

        JButton searchButton = new JButton("Search");

        buttonPanel.add(homeButton);

        buttonPanel.add(editButton);

        buttonPanel.add(listButton);

        buttonPanel.add(searchButton);

        // Add action listeners for buttons

        homeButton.addActionListener(e -> JOptionPane.showMessageDialog(this, "You are already on the Home Page!"));

        editButton.addActionListener(e -> {

            // Open the EditPage when the Edit button is clicked

            new EditPage();

        });

        listButton.addActionListener(e -> {

            // Open the ListPage when the List button is clicked

            new ListPage();

        });

        searchButton.addActionListener(e -> {

            // Open the SearchPage when the Search button is clicked

            new SearchPage();

        });

        // Logout button

        JButton logoutButton = new JButton("Logout");

        logoutButton.addActionListener(e -> {

            dispose(); // Close the Home Page

            new LoginPage(); // Open Login Page

        });

        // Add components

        add(welcomeLabel, BorderLayout.NORTH);

        add(buttonPanel, BorderLayout.CENTER);

        add(logoutButton, BorderLayout.SOUTH);

        // Make the window visible

        setVisible(true);

    }

}

#### 3. Edit Page

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class EditPage extends JFrame {

    private JTextField nameField, ageField, courseField, roomField;

    private JButton saveButton, cancelButton;

    public EditPage() {

        // Set up the JFrame

        setTitle("Edit Student Details");

        setSize(400, 300);

        setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

        setLocationRelativeTo(null); // Center the window

        setLayout(new GridLayout(5, 2, 10, 10)); // Grid layout for form

        // Create and add components

        JLabel nameLabel = new JLabel("Name:");

        nameField = new JTextField();

        JLabel ageLabel = new JLabel("Age:");

        ageField = new JTextField();

        JLabel courseLabel = new JLabel("Course:");

        courseField = new JTextField();

        JLabel roomLabel = new JLabel("Room Number:");

        roomField = new JTextField();

        saveButton = new JButton("Save");

        cancelButton = new JButton("Cancel");

        add(nameLabel);

        add(nameField);

        add(ageLabel);

        add(ageField);

        add(courseLabel);

        add(courseField);

        add(roomLabel);

        add(roomField);

        add(saveButton);

        add(cancelButton);

        // Action listeners

        saveButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                saveStudentDetails();

            }

        });

        cancelButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                dispose(); // Close the EditPage without saving

            }

        });

        // Make the window visible

        setVisible(true);

    }

    // Method to save student details into the database

    private void saveStudentDetails() {

        String name = nameField.getText();

        String age = ageField.getText();

        String course = courseField.getText();

        String roomNumber = roomField.getText();

        if (name.isEmpty() || age.isEmpty() || course.isEmpty() || roomNumber.isEmpty()) {

            JOptionPane.showMessageDialog(this, "All fields are required!", "Error", JOptionPane.ERROR\_MESSAGE);

            return;

        }

        // Database connection setup

        String dbUrl = "jdbc:mysql://localhost:3306/hostel"; // Change with your DB URL

        String dbUsername = "root"; // Replace with your DB username

        String dbPassword = "2105"; // Replace with your DB password

        try (Connection conn = DriverManager.getConnection(dbUrl, dbUsername, dbPassword)) {

            String sql = "INSERT INTO student (name, age, course, room\_number) VALUES (?, ?, ?, ?)";

            try (PreparedStatement stmt = conn.prepareStatement(sql)) {

                stmt.setString(1, name);

                stmt.setString(2, age);

                stmt.setString(3, course);

                stmt.setString(4, roomNumber);

                int rowsAffected = stmt.executeUpdate();

                if (rowsAffected > 0) {

                    JOptionPane.showMessageDialog(this, "Student details saved successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

                    dispose(); // Close the EditPage

                } else {

                    JOptionPane.showMessageDialog(this, "Failed to save student details.", "Error", JOptionPane.ERROR\_MESSAGE);

                }

            }

        } catch (SQLException e) {

            JOptionPane.showMessageDialog(this, "Database error: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

}

#### 4. Student List Page

import javax.swing.\*;

import java.awt.\*;

import java.sql.\*;

import javax.swing.table.DefaultTableModel;

public class ListPage extends JFrame {

    private JTable studentTable;

    private DefaultTableModel model;

    public ListPage() {

        // Set up the JFrame

        setTitle("Student List");

        setSize(600, 400); // Adjust size as needed

        setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

        setLocationRelativeTo(null); // Center the window

        // Create the table model to hold student data

        model = new DefaultTableModel();

        model.addColumn("ID");

        model.addColumn("Name");

        model.addColumn("Age");

        model.addColumn("Course");

        model.addColumn("Room Number");

        // Create the JTable and add it to a JScrollPane

        studentTable = new JTable(model);

        JScrollPane scrollPane = new JScrollPane(studentTable);

        studentTable.setFillsViewportHeight(true);

        // Fetch the student data from the database

        fetchStudentData();

        // Create a panel for buttons at the bottom

        JPanel buttonPanel = new JPanel();

        buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER));

        // Create the "Delete" button

        JButton deleteButton = new JButton("Delete");

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

        // Create the "Back" button

        JButton backButton = new JButton("Back");

        backButton.addActionListener(e -> {

            dispose(); // Close the ListPage

            new HomePage("admin"); // Open HomePage (pass the username if necessary)

        });

        // Add the buttons to the button panel

        buttonPanel.add(deleteButton);

        buttonPanel.add(backButton);

        // Add components to the JFrame

        add(scrollPane, BorderLayout.CENTER);

        add(buttonPanel, BorderLayout.SOUTH); // Add buttons at the bottom

        // Make the window visible

        setVisible(true);

    }

    private void fetchStudentData() {

        // Database credentials

        String dbUrl = "jdbc:mysql://localhost:3306/hostel"; // Change this with your DB URL

        String dbUsername = "root"; // Replace with your DB username

        String dbPassword = "2105"; // Replace with your DB password

        // SQL query to retrieve student data

        String query = "SELECT id, name, age, course, room\_number FROM student";

        try (Connection conn = DriverManager.getConnection(dbUrl, dbUsername, dbPassword);

             Statement stmt = conn.createStatement();

             ResultSet rs = stmt.executeQuery(query)) {

            // Populate the table with data from the database

            while (rs.next()) {

                model.addRow(new Object[]{

                    rs.getInt("id"),

                    rs.getString("name"),

                    rs.getInt("age"),

                    rs.getString("course"),

                    rs.getString("room\_number")

                });

            }

        } catch (SQLException e) {

            JOptionPane.showMessageDialog(this, "Error fetching student data: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    private void deleteStudent() {

        // Get the selected row in the JTable

        int selectedRow = studentTable.getSelectedRow();

        if (selectedRow == -1) {

            JOptionPane.showMessageDialog(this, "Please select a student to delete", "Error", JOptionPane.ERROR\_MESSAGE);

            return;

        }

        // Get the student ID from the selected row

        int studentId = (int) model.getValueAt(selectedRow, 0); // Assuming ID is in the first column

        // Database credentials

        String dbUrl = "jdbc:mysql://localhost:3306/hostel"; // Change this with your DB URL

        String dbUsername = "root"; // Replace with your DB username

        String dbPassword = "2105"; // Replace with your DB password

        // SQL query to delete the student record

        String deleteQuery = "DELETE FROM student WHERE id = ?";

        try (Connection conn = DriverManager.getConnection(dbUrl, dbUsername, dbPassword);

             PreparedStatement ps = conn.prepareStatement(deleteQuery)) {

            // Set the ID parameter for the query

            ps.setInt(1, studentId);

            // Execute the delete query

            int rowsAffected = ps.executeUpdate();

            if (rowsAffected > 0) {

                // Remove the row from the table

                model.removeRow(selectedRow);

                JOptionPane.showMessageDialog(this, "Student deleted successfully", "Success", JOptionPane.INFORMATION\_MESSAGE);

            } else {

                JOptionPane.showMessageDialog(this, "Error deleting student", "Error", JOptionPane.ERROR\_MESSAGE);

            }

        } catch (SQLException e) {

            JOptionPane.showMessageDialog(this, "Error deleting student: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

}

#### 5. Java/MySQL Connectivity

import java.sql.Connection; import java.sql.DriverManager; import java.sql.SQLException;

public class DBConnection { private static final String URL = "jdbc:mysql://localhost:/student\_management"; private static final String USER = "root"; // Your MySQL username

private static final String PASSWORD = "2105"; // Your MySQL password private static Connection connection;

public static Connection getConnection() {

try {

// Ensure MySQL driver is loaded

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

// Create a new connection if none exists or if the connection is closed if (connection == null || connection.isClosed()) { connection = DriverManager.getConnection(URL, USER, PASSWORD);

System.out.println("Database connection established successfully.");

}

} catch (ClassNotFoundException e) {

System.out.println("MySQL JDBC Driver not found.");

e.printStackTrace();

} catch (SQLException e) {

System.out.println("SQL Exception: " + e.getMessage());

System.out.println("Error Code: " + e.getErrorCode()); System.out.println("SQL State: " + e.getSQLState());

e.printStackTrace();

}

return connection;

}

public static void closeConnection() {

try {

if (connection != null && !connection.isClosed()) { connection.close();

System.out.println("Database connection closed.");

}

} catch (SQLException e) {

System.out.println("Error closing database connection.");

e.printStackTrace();

}

}

}

import java.sql.Connection; // Import the Connection class from java.sql

public class App { public static void main(String[] args) {

// Example usage of the DBConnection class

Connection conn = DBConnection.getConnection(); // Use DBConnection, not dbconnection

if (conn != null) {

System.out.println("Database connection established successfully.");

} else {

System.out.println("Failed to establish database connection.");

}

// Closing the connection when done

DBConnection.closeConnection(); // Use DBConnection, not dbconnection

}

}

**6. Search Page**

import javax.swing.\*;

import java.awt.\*;

import java.sql.\*;

import javax.swing.table.DefaultTableModel;

public class SearchPage extends JFrame {

    private JTable studentTable;

    private DefaultTableModel model;

    private JTextField searchField;

    public SearchPage() {

        // Set up the JFrame

        setTitle("Search Students");

        setSize(600, 400);

        setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

        setLocationRelativeTo(null); // Center the window

        // Create the table model to hold student data

        model = new DefaultTableModel();

        model.addColumn("ID");

        model.addColumn("Name");

        model.addColumn("Age");

        model.addColumn("Course");

        model.addColumn("Room Number");

        // Create the JTable and add it to a JScrollPane

        studentTable = new JTable(model);

        JScrollPane scrollPane = new JScrollPane(studentTable);

        studentTable.setFillsViewportHeight(true);

        // Create a panel for the search input and button

        JPanel searchPanel = new JPanel();

        searchPanel.setLayout(new FlowLayout(FlowLayout.LEFT));

        // Create the search field

        searchField = new JTextField(20);

        searchPanel.add(new JLabel("Search: "));

        searchPanel.add(searchField);

        // Create the search button

        JButton searchButton = new JButton("Search");

        searchButton.addActionListener(e -> searchStudent());

        searchPanel.add(searchButton);

        // Create the back button

        JButton backButton = new JButton("Back");

        backButton.addActionListener(e -> {

            dispose(); // Close the SearchPage

            new HomePage("admin"); // Open HomePage (pass the username if necessary)

        });

        // Add components to the JFrame

        add(searchPanel, BorderLayout.NORTH);

        add(scrollPane, BorderLayout.CENTER);

        JPanel buttonPanel = new JPanel();

        buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER));

        buttonPanel.add(backButton);

        add(buttonPanel, BorderLayout.SOUTH);

        // Make the window visible

        setVisible(true);

    }

    private void searchStudent() {

        String searchQuery = searchField.getText().trim();

        if (searchQuery.isEmpty()) {

            JOptionPane.showMessageDialog(this, "Please enter a search term", "Error", JOptionPane.ERROR\_MESSAGE);

            return;

        }

        // Database credentials

        String dbUrl = "jdbc:mysql://localhost:3306/hostel"; // Change this with your DB URL

        String dbUsername = "root"; // Replace with your DB username

        String dbPassword = "2105"; // Replace with your DB password

        // SQL query to search for students by name or ID

        String query = "SELECT id, name, age, course, room\_number FROM student WHERE name LIKE ? OR id = ?";

        try (Connection conn = DriverManager.getConnection(dbUrl, dbUsername, dbPassword);

             PreparedStatement ps = conn.prepareStatement(query)) {

            // Set the search parameters

            ps.setString(1, "%" + searchQuery + "%"); // Search by name

            try {

                ps.setInt(2, Integer.parseInt(searchQuery)); // Try to search by ID (if it's a valid number)

            } catch (NumberFormatException e) {

                ps.setNull(2, java.sql.Types.INTEGER); // If not a valid number, search by name only

            }

            // Execute the query

            ResultSet rs = ps.executeQuery();

            // Clear the previous search results

            model.setRowCount(0);

            // Populate the table with the search results

            while (rs.next()) {

                model.addRow(new Object[]{

                        rs.getInt("id"),

                        rs.getString("name"),

                        rs.getInt("age"),

                        rs.getString("course"),

                        rs.getString("room\_number")

                });

            }

            // If no records are found

            if (model.getRowCount() == 0) {

                JOptionPane.showMessageDialog(this, "No results found for \"" + searchQuery + "\"", "No Results", JOptionPane.INFORMATION\_MESSAGE);

            }

        } catch (SQLException e) {

            JOptionPane.showMessageDialog(this, "Error searching student data: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

}

### MySQL CODE

CREATE DATABASE hostel\_management;

USE hostel\_management;

CREATE TABLE users (

username VARCHAR(50) PRIMARY KEY,

password VARCHAR(255) NOT NULL

);

CREATE TABLE residents (

id VARCHAR(10) PRIMARY KEY,

name VARCHAR(100),

room\_number VARCHAR(10),

admission\_date DATE,

fee\_status VARCHAR(20)

);

CREATE TABLE rooms (

room\_number VARCHAR(10) PRIMARY KEY,

capacity INT,

occupancy INT,

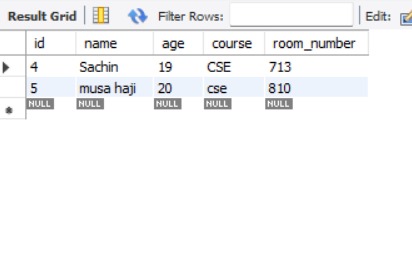
status VARCHAR(20)

);

CREATE TABLE attendance (

resident\_id VARCHAR(10),

dat

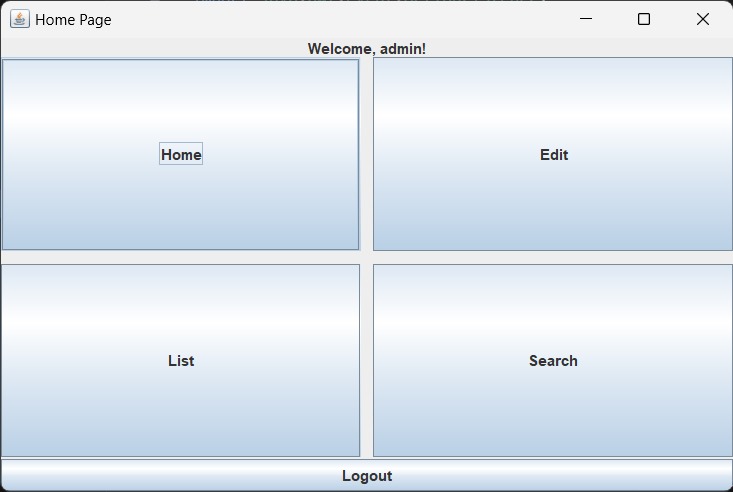


### Chapter 5: RESULT AND DISCUSSION

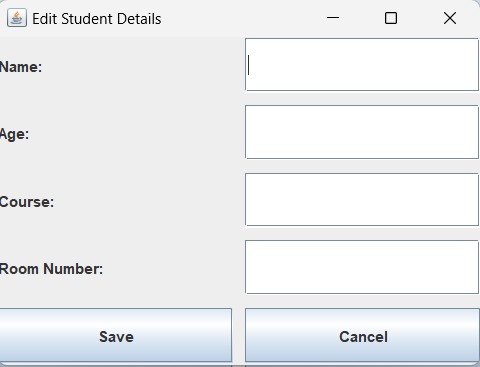
**1. Login Page:**



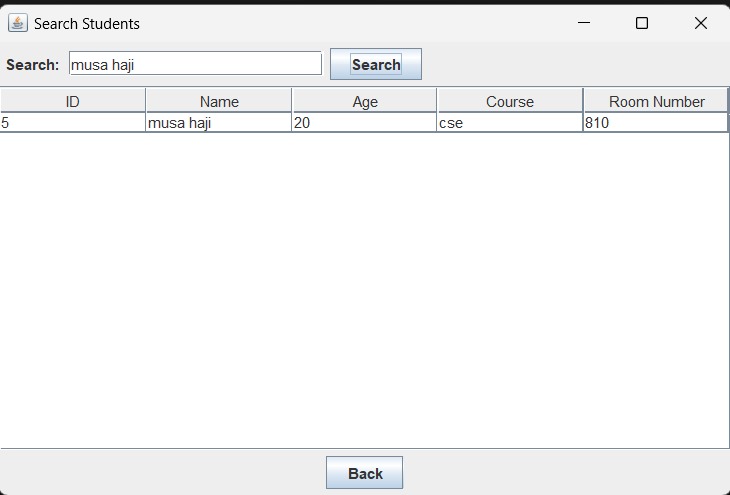
#### 2. Home Page



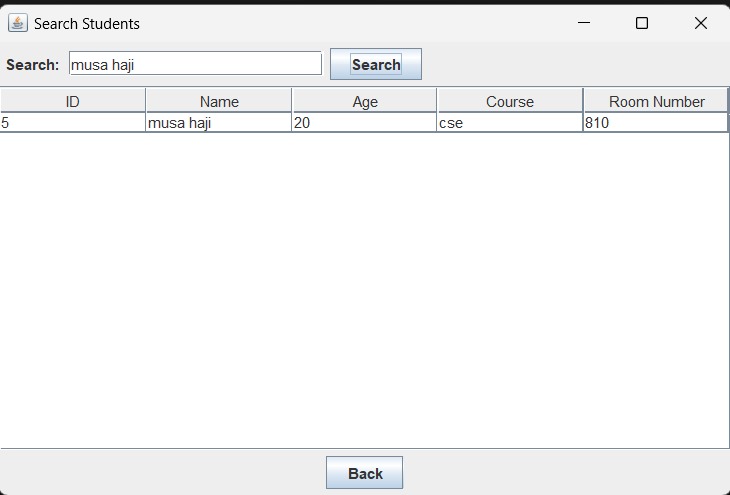
#### 3. Edit Page



#### 4. Student List Page



### 5. Search Students



### Chapter 6: Conclusion

#### 6.1 Conclusion

Here is the converted version of the information for a **Hostel Management System**:

The development of the **Hostel Management System** represents a significant advancement in accommodation record management, enhancing accessibility to essential resident and room information. This project successfully achieves its core objectives of simplifying the management of resident records, tracking room occupancy, and providing a comprehensive view of resident profiles, ultimately improving administrative efficiency and decision-making processes.

Built using Java with a JDBC connection to a MySQL database, the system is secure, robust, and efficient in handling real-time accommodation data. The use of Java JFrame ensures a user-friendly, sleek interface, making interactions such as resident registration, room allocation, and attendance tracking intuitive and streamlined. MySQL contributes to efficient database management, supporting high volumes of resident data with optimal performance and reliability.

With comprehensive functional capabilities, including user authentication, profile management, room allocation, and fee tracking, the system meets critical requirements for hostel management. Nonfunctional aspects such as security, performance, scalability, and maintainability further reinforce the system’s operational stability, ensuring administrators and residents experience uninterrupted, secure, and responsive service.

In conclusion, the **Hostel Management System** effectively addresses key challenges in accommodation management through an integrated approach, employing robust backend technologies and a user-focused design. It sets a standard for efficiency and security in accommodation record management, providing a scalable solution that can adapt to future enhancements while consistently delivering an exceptional user experience and reliable hostel management tools.

### Chapter 7: REFERENCE

#### 7.1 REFERENCES

1. https://stackoverflow.com
2. <https://www.youtube.com/watch?v=OGP2R29vzAw>
3. https://www.youtube.com/watch?v=jHSBrX8lLWk