

TABLE OF CONTENTS

| | |
|--|----------------------|
| DECLARATION | I |
| CERTIFICATE | II |
| ACKNOWLEDGMENT | III |
| ABSTRACT | IV |
| | |
| 1. INTRODUCTION | Page 4 To 8 |
| 1.1 Overview of HostelBuddy | 04 |
| 1.2 Motivation Behind Hostel Management Automation | 05 |
| 1.3 Objectives of the Project | 06 |
| 1.4 Core Features and Architecture | 07 |
| | |
| 2. TECHNOLOGIES USED | Page 9 To 12 |
| 2.1 Java Swing for GUI | 09 |
| 2.2 MySQL for Database Management | 10 |
| 2.3 NetBeans IDE | 11 |
| 2.4 Supporting Libraries and Tools | 11 |
| | |
| 3. SYSTEM ANALYSIS | Page 13 To 16 |
| 3.1 Existing Manual System | 13 |
| 3.2 Limitations of Manual Management | 14 |
| 3.3 Proposed System Features | 14 |
| 3.4 Benefits of HostelBuddy | 15 |
| | |
| 4. SYSTEM DESIGN | Page 17 To 19 |

| | |
|-------------------------------------|----------------------|
| 4.1 System Architecture Overview | 17 |
| 4.2 Entity Relationship Diagram | 18 |
| 4.3 Flow of Control Between Modules | 18 |
| 4.4 GUI Navigation Flow | 19 |
| 5. DATABASE DESIGN | Page 20 To 24 |
| 5.1 Tables Overview | 20 |
| 5.2 Student Table Schema | 21 |
| 5.3 Room Table Schema | 21 |
| 5.4 Employee and Fees Table | 22 |
| 5.5 Sample Queries and Transactions | 23 |
| 6. MODULES DESCRIPTION | Page 25 To 80 |
| 6.1 Login and Access Control | 25 |
| 6.2 New Student Registration | 29 |
| 6.3 Room Allocation and Management | 37 |
| 6.4 Fee Collection and History | 47 |
| 6.5 Employee Management | 60 |
| 6.6 Report Generation and Export | 74 |
| 7. USER INTERFACE SCREENS | Page 81 To 87 |
| 7.1 Login Screen | 81 |
| 7.2 Home Dashboard | 81 |
| 7.3 Add Student Form | 82 |
| 7.4 Fee Management Panel | 83 |
| 7.5 Employee Panel | 84 |
| 7.6 Sample Reports | 85 |

| | |
|---|------------------------|
| 8. CODE SNIPPETS AND LOGIC | Page 88 To 90 |
| 8.1 Database Connectivity | 88 |
| 8.2 Login Validation | 88 |
| 8.3 Insertion and Update Queries | 89 |
| 8.4 Export to CSV Functionality | 89 |
| 8.5 Form Events and GUI Actions | 90 |
| 9. TESTING AND OUTPUT | Page 91 To 106 |
| 9.1 Functional Testing of Modules | 91 |
| 9.2 Input Validations | 91 |
| 9.3 Sample Outputs | 92 |
| 9.4 Exception Handling | 106 |
| 10. CONCLUSION AND FUTURE ENHANCEMENTS | Page 108 To 109 |
| 10.1 Summary of Achievements | 108 |
| 10.2 Challenges Faced | 108 |
| 10.3 Scope for Improvement | 109 |
| 10.4 Potential Future Features | 109 |
| 11. REFERENCES | Page 110 |

1. INTRODUCTION

In today's dynamic educational institutions, hostel management plays a vital role in ensuring the comfort, discipline, and accountability of students residing on campus. Managing hundreds of students manually through traditional record-keeping methods such as registers, spreadsheets, or informal systems often leads to inconsistencies, delays, and administrative bottlenecks.

To overcome these issues, a more intelligent, digital, and automated approach is necessary. The project HostelBuddy has been developed with this objective in mind. It is a desktop-based application that brings together key functionalities required for running a hostel—right from room allotment, student registration, and fee tracking to employee management and report generation—all under a single platform.

HostelBuddy offers an efficient and user-friendly interface built using Java Swing and maintains its data integrity through a MySQL backend. This blend of technologies ensures the system is scalable, fast, and easy to maintain. HostelBuddy provides a real-world software engineering experience by integrating GUI, backend databases, user input validation, exception handling, and modular system architecture.

1.1 Overview of HostelBuddy

HostelBuddy is a comprehensive hostel management software solution designed for colleges and universities to automate daily hostel operations. The system has been built keeping in mind the practical difficulties faced by wardens and hostel administrators. HostelBuddy not only reduces human effort but also improves data accuracy and availability.

The system has the following features:

- **Login System:** A secure way to access the application, allowing only authenticated administrators.
- **Student Management:** Register new students, update or delete their records, and track whether they are currently staying in the hostel, on temporary leave, or have permanently left.
- **Room Allocation:** Assign rooms dynamically and update room status based on occupancy. (1 student per room or configurable as needed).
- **Fee Tracking:** Maintain monthly payment history, generate fee receipts, and export data to CSV files.
- **Employee Management:** Record staff data, track working or left status, and manage their salary history.
- **Dashboard Navigation:** A modern Home Page UI with card-based layout for module navigation.
- **Database Integration:** All records are stored and retrieved in real-time using MySQL, ensuring consistency and reliability.

These features provide a robust framework for any institution to manage its hostel infrastructure efficiently.

1.2 Motivation Behind Hostel Management Automation

Traditional hostel management systems are typically plagued with several issues:

- **Manual Errors:** In room allocation, fee collection, and attendance logs.
- **Paperwork Overload:** Multiple registers and files are difficult to maintain and retrieve.
- **Lack of Real-Time Updates:** No instant tracking of room status or student availability.

- **Inaccessible Historical Data:** Difficulty in pulling old records or understanding patterns in student or employee behavior.
- **Inefficiency in Report Generation:** Tedious and slow process of summarizing data manually.

HostelBuddy was conceived to address these challenges head-on. The need for automation in hostel management has increased dramatically as educational institutions scale up. HostelBuddy simplifies hostel operations through its centralized platform. It ensures:

- **Efficiency:** Speeds up registration, room assignment, and payment processing.
- **Accuracy:** Reduces chances of data duplication and mismatches.
- **Transparency:** Every operation is logged and accessible.
- **Accessibility:** Information is quickly retrievable using filters and search.
- **Scalability:** Easy to extend with future features like biometric access or mobile notifications.

This motivation stems not only from a technical challenge but also from an effort to make campus life smoother and more organized.

1.3 Objectives of the Project

The main objectives of the HostelBuddy project are:

1. To develop a user-friendly and efficient hostel management application using Java Swing and MySQL.
2. To automate the process of student registration, room allotment, and fee tracking, reducing the workload on hostel staff.

3. To maintain an up-to-date record of all students, including those currently staying, those temporarily away, and those who have permanently left.
4. To provide a platform to manage employee details, including their role, work status, and payment information.
5. To enable monthly fee payment tracking and generate payment histories and reports for administrative use.
6. To allow real-time search and filter features that quickly locate student/employee records based on names, room numbers, or status.
7. To create exportable CSV files for reports and backups.
8. To allow flexible room management, including setting capacity and current occupancy.
9. To provide a dashboard interface that improves ease of access to every module.
10. To offer a scalable backend that can be adapted to growing hostel requirements.

These objectives collectively ensure that HostelBuddy is not just a simple CRUD application but a full-fledged hostel administration toolkit.

1.4 Core Features and Architecture

HostelBuddy is structured using a layered architecture, which separates the application into distinct modules to ensure clean code, easy debugging, and future maintainability. The core layers are:

- **Presentation Layer:** Java Swing GUI, containing forms, buttons, card views, and layout managers.
- **Logic Layer:** Contains form validation, event handlers, and business rules.

- **Data Access Layer:** Interacts with the MySQL database using JDBC for all CRUD operations.

Major Functional Modules Include:

- **Authentication Module:** Validates user login using hardcoded or database-stored credentials.
- **Student Management:** Register, update, and delete student information. Track status (living, temp leave, leaved).
- **Room Management:** Add/edit rooms, track current occupancy, and auto-change status from 'Not Booked' to 'Booked'.
- **Fee Management:** Record monthly fee data, check pending payments, export payment history.
- **Employee Management:** Add/update employees, track if they are currently working or have left, and maintain salary details.
- **Reports and Export:** Export student fee history and employee payment records into CSV format for administrative use.

User-Interface:

The application opens with a splash screen followed by a login screen. On successful login, the user is presented with a Home Page that uses a card-based interface to navigate through all major modules. Each card is clickable and visually displays its purpose using an icon and title.

Scalability-&-Extensibility:

Future additions like mobile app support, biometric authentication, online fee payments, and room-sharing options can easily be incorporated into the system, thanks to its modular structure.

2. TECHNOLOGIES USED

HostelBuddy integrates various modern development tools and platforms to offer a smooth and efficient hostel management experience. The selection of technologies was made to balance performance, ease of development, maintainability, and user interface responsiveness.

This section details the key technologies employed in building the HostelBuddy system.

2.1 Java Swing for GU

Java Swing is a part of Java Foundation Classes (JFC) used to create window-based applications. It is a lightweight, platform-independent component-based framework for building Graphical User Interfaces (GUIs) in Java.

In HostelBuddy, Java Swing was used extensively to design the entire user interface of the application. The choice of Swing ensures:

- Cross-platform compatibility
- Highly customizable components like JFrame, JPanel, JTable, JButton, and more
- Support for rich UI elements such as icons, tooltips, and layout managers
- Ease of event-driven programming with built-in listeners

Swing components were used to create:

- Login and Welcome screens
- Home Page dashboard with a modern card-based interface
- Forms for student and employee entry

- Tables for viewing and managing hostel data

To improve user experience, layout managers like `BoxLayout`, `GridLayout`, and absolute positioning (where necessary) were used to ensure consistency and responsiveness.

2.2 MySQL for Database Management

MySQL is an open-source relational database management system (RDBMS). It is widely used for web and application development due to its speed, flexibility, and support for SQL.

In HostelBuddy, MySQL was used to create and manage all persistent data, including:

- Student records
- Room information
- Employee details
- Monthly fee transactions

The database schema was designed to enforce referential integrity and optimize performance. Common SQL operations such as `INSERT`, `SELECT`, `UPDATE`, and `DELETE` are executed through Java Database Connectivity (JDBC) APIs.

The benefits of using MySQL include:

- Seamless integration with Java applications
 - High performance for concurrent access
 - Wide availability of tools like MySQL Workbench for database design and query testing
 - Easy export/import for backups and reports
-

2.3 NetBeans IDE

NetBeans is a popular open-source Integrated Development Environment (IDE) for Java application development. It offers intelligent code completion, drag-and-drop GUI design, version control integration, and in-built project management.

For HostelBuddy, NetBeans was chosen due to:

- Powerful GUI builder for designing Swing interfaces
- Real-time error checking and suggestions
- Easy project structuring and file management
- Support for multiple Java versions and libraries

NetBeans allowed rapid development of forms and modules by offering visual UI design, which is particularly useful in projects involving many interfaces.

2.4 Supporting Libraries and Tools

Besides the core technologies, additional tools and libraries were used to enhance functionality and productivity:

- **JDBC (Java Database Connectivity):** For establishing connections with MySQL, executing SQL queries, and handling results.
- **CSV Export Utility:** A custom utility class (ExportUtil.java) was created to export tables like fee history or employee payments to .csv files for offline analysis.
- **FlatLaf (optional):** A modern look-and-feel library that can be integrated for a clean material-style GUI.

- **MySQL Workbench:** Used for database schema creation, SQL execution, and visual ER diagram generation.
- **File I/O APIs:** Used for future extensibility like document uploads, backups, etc.

Each of these tools contributed to making HostelBuddy a complete, user-friendly, and feature-rich application.

3. SYSTEM ANALYSIS

This section presents a detailed analysis of both the traditional hostel management system and the improvements brought in by HostelBuddy. By examining the current challenges in manual processes and identifying the pain points experienced by administrators and hostel managers, we can clearly justify the need for an automated solution.

3.1 Existing Manual System

The manual system commonly used in hostels relies on physical registers or spreadsheet files to record and manage data. The process involves pen-and-paper entries, verbal confirmations, and paper receipts. Hostel staff must perform time-consuming tasks daily, such as checking room occupancy manually, verifying fee payment status, updating student records when someone leaves or arrives, and preparing summary reports at the end of each month.

Some examples of processes in a manual system include:

- Using register books to assign rooms to students.
- Keeping handwritten logs of students currently residing in the hostel.
- Manually issuing fee receipts and updating the payment status.
- Tracking employee working days and calculating their salary.

This outdated approach is not scalable or reliable, especially for institutions with hundreds of students and dozens of employees.

3.2 Limitations of Manual Management

The major limitations and drawbacks of a manual hostel management system include:

- **Data Inconsistency:** Manual data entries are highly prone to human errors such as duplication, misplacement, or mismatching of records.
- **Time-Consuming:** Every transaction like room assignment, fee tracking, and employee payments takes considerable time and manual verification.
- **Lack of Real-Time Information:** Staff do not have immediate access to updated data, leading to confusion and miscommunication.
- **Data Retrieval Difficulty:** Searching through registers or spreadsheets to find a particular student's history or fee record is inefficient.
- **Security Risks:** Paper records can be easily lost, damaged, or tampered with.
- **Report Generation Hassle:** Creating accurate reports requires manually calculating figures, which is slow and can be inaccurate.

These limitations cause unnecessary workload for staff and lower the overall efficiency and accountability of hostel operations.

3.3 Proposed System Features

The proposed system, HostelBuddy, is a modern software solution that directly addresses the drawbacks of manual systems through automation, data centralization, and smart interfaces. HostelBuddy enables hostel administrators to manage all hostel operations from a single digital platform.

Key features of the proposed system include:

- **Digital Room Allocation:** Track room availability and assign rooms with a click.
 - **Student Record Management:** Register, update, delete, and search student records quickly.
 - **Status Tracking:** Mark students as 'Living', 'Temporarily Away', or 'Leaved'.
 - **Fee Management:** Record and view fee payments, generate monthly reports.
 - **Employee Management:** Maintain staff details and salary payment history.
 - **Automated Calculations:** Real-time updating of room status and financial summaries.
 - **CSV Report Exporting:** Export data tables to files for offline use and backups.
 - **GUI-Based Navigation:** Easy-to-use card-style menu for navigating system modules.
 - **Security and Accuracy:** Ensures only authorized access and reduces data entry mistakes.
-

3.4 Benefits of HostelBuddy

Implementing HostelBuddy provides numerous benefits for institutions, including:

- **Operational Efficiency:** Streamlines day-to-day hostel tasks, saving time and effort.
- **Centralized Data:** All student, employee, room, and payment records are stored and accessible from one place.
- **Enhanced Accuracy:** Reduced chances of human error due to validations and guided input forms.
- **Real-Time Information Access:** Instantly check room occupancy, student details, and fee statuses.

- **Improved Reporting:** Generate detailed reports with a few clicks instead of hours of manual work.
- **Scalability:** Easily expandable as the number of students and employees grows.
- **User-Friendly Interface:** Visual icons, buttons, and clean design simplify usage even for non-technical staff.
- **Data Backup & Security:** Digital records are safer and easier to back up compared to physical logs.

HostelBuddy not only modernizes hostel management but also empowers the administrative staff with tools that improve service quality and reduce stress.

4. SYSTEM DESIGN

System design plays a crucial role in the development of any software application. It ensures the logical flow of operations, modularity, clarity in architecture, and smooth integration between different system components. HostelBuddy has been designed with simplicity, modularity, and extensibility in mind. The system is structured using a multi-layered architecture that clearly separates user interface, business logic, and data access components.

4.1 System Architecture Overview

HostelBuddy follows a layered architecture model, which divides the application into distinct functional layers. This approach enhances code clarity, debugging ease, and future scalability. The main layers are:

- **Presentation Layer:** The front-end interface built using Java Swing. It includes all GUI components such as forms, buttons, input fields, navigation menus, and card-based layout.
- **Logic/Control Layer:** Handles the application's business logic. This includes form validation, user input processing, decision-making (e.g., checking room availability), and triggering appropriate data operations.
- **Data Access Layer:** Interfaces with the MySQL database using JDBC. Executes CRUD operations, manages connections, and handles SQL exceptions.

The system also uses helper classes and utility functions to handle CSV exports, input validation, and modular event handling.

4.2 Entity Relationship Diagram

An Entity Relationship (ER) Diagram describes the database structure used in HostelBuddy. It outlines the primary entities and the relationships among them:

- **Students:** Each student is linked to a room and maintains fields such as name, mobile number, college, status, and fee records.
- **Rooms:** Represents available rooms, their status (Booked/Not Booked), and current occupancy.
- **Employees:** Contains details of hostel staff such as name, role, contact, and salary records.
- **Fees:** Tracks monthly payments by students including amount, date, and status.

Insert Entity Relationship Diagram here (ERD showing tables and relations)

4.3 Flow of Control Between Modules

The flow of control in HostelBuddy follows an event-driven pattern. Each user interaction (button click, form submission) triggers a series of actions that navigate the system across modules. A typical flow might look like:

1. **User logs in** via the login screen → Validation through control layer.
2. **HomePage** opens → User selects an operation (e.g., New Student).
3. **Student Form** appears → User fills in details and submits.
4. **Logic Layer** validates input → Control passes to data layer.
5. **Database Operation** executes (e.g., INSERT student into DB).

6. **Success/Failure message** is shown on the GUI.

This clear separation of interaction, validation, and data manipulation allows for modular testing and debugging.

4.4 GUI Navigation Flow

The GUI of HostelBuddy is built using Java Swing with a focus on clarity, ease of use, and modern design. Upon login, the user is presented with a Home Page that features a card-style layout, where each card represents a specific function.

Navigation flow:

- **WelcomeScreen** → Launch animation/splash (optional)
- **LoginScreen** → Admin enters credentials
- **HomePage** → Dashboard opens with navigation cards:
 - Manage Room
 - New Student / All Students / Leaved Students
 - Student Fees / Fee History / Monthly Report
 - New Employee / Update Employee / Employee Payment
- Clicking a card opens its respective form or table view
- All forms follow consistent layout and design
- Logout button returns user to LoginScreen

Each component is aligned to offer minimum cognitive load to the user and ensures actions are intuitive. The consistent structure also ensures that newly added modules (in the future) can follow the same navigation logic without disrupting the user experience.

5. DATABASE DESIGN

The backbone of any robust software system is its database design. A well-planned database schema ensures consistent data storage, efficient retrieval, minimal redundancy, and data integrity. HostelBuddy employs a MySQL-based relational database model to handle all backend operations involving students, rooms, fees, and employees.

This section provides a detailed breakdown of the core tables used in the system, their structure, relationships, and some key SQL operations.

5.1 Tables Overview

The HostelBuddy database includes the following key tables:

1. **students** – Holds information about each student including personal details, room allocation, and current status.
2. **rooms** – Maintains data about hostel rooms such as room number, status (Booked/Not Booked), and current occupants.
3. **employees** – Records employee profiles and whether they are currently working or have left.
4. **fees** – Stores monthly fee payments for each student, including amount paid and payment status.
5. **employee_payments** – Logs salary payments for employees.

These tables are interlinked using primary and foreign key constraints where appropriate.

5.2 Student Table Schema

```
CREATE TABLE students (  
    student_id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(100) NOT NULL,  
    mobile_no VARCHAR(15),  
    father_name VARCHAR(100),  
    mother_name VARCHAR(100),  
    email VARCHAR(100),  
    college_name VARCHAR(200),  
    aadhaar_no VARCHAR(20),  
    room_number VARCHAR(10),  
    status ENUM('Living', 'Leaved') DEFAULT 'Living',  
    FOREIGN KEY (room_number) REFERENCES rooms(room_number)  
);
```

This table tracks student identity, contact, and room-related data. Status helps determine if the student is actively residing or has permanently left.

5.3 Room Table Schema

```
CREATE TABLE rooms (  
    room_number VARCHAR(10) PRIMARY KEY,  
    status ENUM('Booked', 'Not Booked') DEFAULT 'Not Booked',  
    capacity INT DEFAULT 2,
```

```
current_occupants INT DEFAULT 0,  
activate BOOLEAN DEFAULT TRUE);
```

Each room has a unique identifier, availability status, and counters for managing dual occupancy scenarios.

5.4 Employee and Fees Table

- **Employee Table:**

```
CREATE TABLE employees (  
    employee_id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(100),  
    mobile_no VARCHAR(15),  
    email VARCHAR(100),  
    designation VARCHAR(100),  
    status ENUM('Working', 'Leaved') DEFAULT 'Working'  
);
```

- **Employee Payment Table:**

```
CREATE TABLE employee_payments (  
    payment_id INT PRIMARY KEY AUTO_INCREMENT,  
    employee_id INT,  
    month VARCHAR(20),  
    amount DECIMAL(10,2),  
    payment_date DATE,
```

```
FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);
```

- **Fee Table:**

```
CREATE TABLE fees (
    fee_id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT,
    month VARCHAR(20),
    amount DECIMAL(10,2),
    payment_date DATE,
    FOREIGN KEY (student_id) REFERENCES students(student_id)
);
```

5.5 Sample Queries and Transactions

- **Insert a new student:**

```
INSERT INTO students (name, mobile_no, college_name, room_number, status)
VALUES ('Ravi Kumar', '9876543210', 'KIT Kanpur', 'A101', 'Living');
```

- **Update room occupancy:**

```
UPDATE rooms SET current_occupants = current_occupants + 1 WHERE room_number = 'A101';
```

- **Record a monthly fee:**

```
INSERT INTO fees (student_id, month, amount, payment_date)
VALUES (1, 'April 2025', 5000.00, CURDATE());
```

- **Mark student as left:**

```
UPDATE students SET status = 'Leaved' WHERE student_id = 1;
```

```
UPDATE rooms SET current_occupants = current_occupants - 1 WHERE room_number = 'A101';
```

- **Retrieve fee history:**

```
SELECT * FROM fees WHERE student_id = 1 ORDER BY payment_date DESC;
```

6. MODULES DESCRIPTION

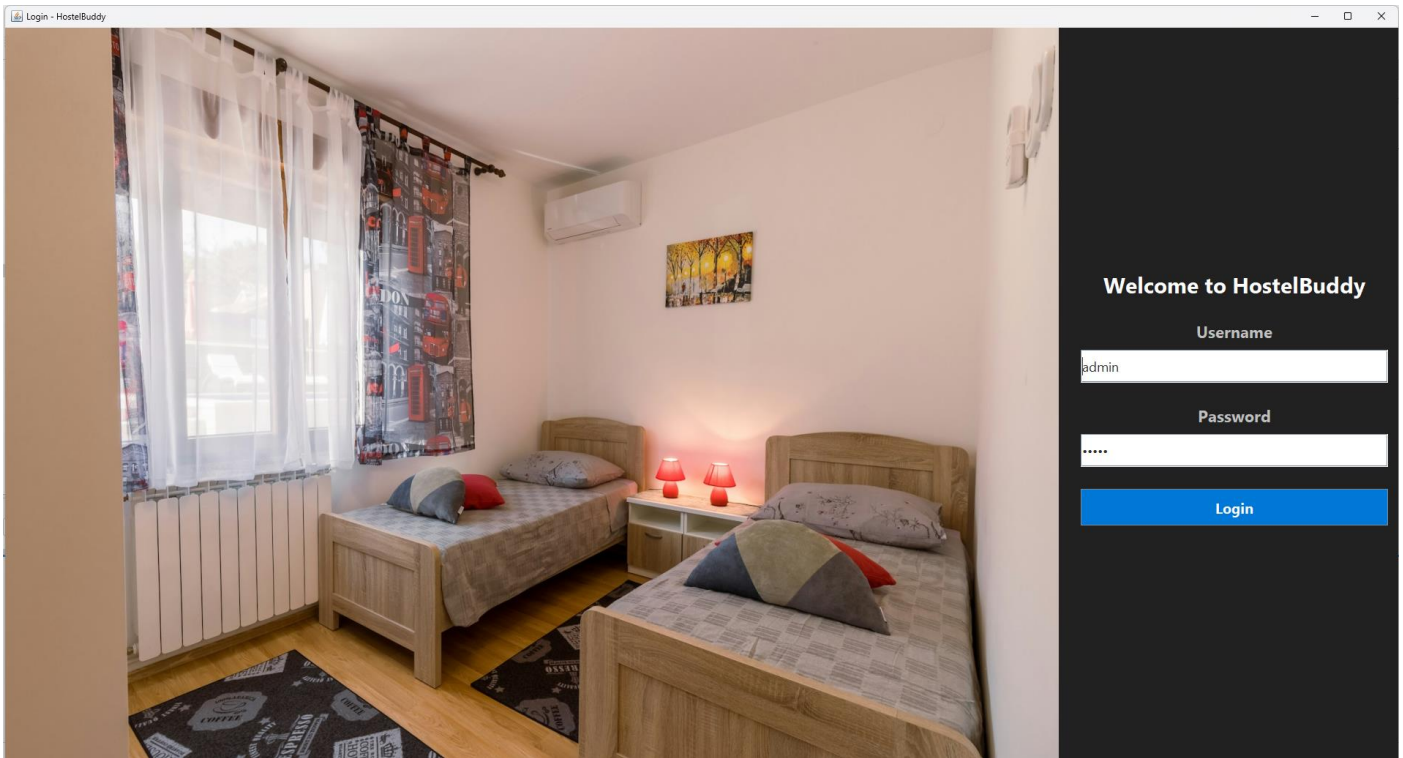
This section provides a detailed breakdown of each module implemented in the HostelBuddy application. Each module is responsible for a specific set of tasks and interacts with others through the shared database and graphical user interface.

6.1 Login and Access Control

The login module ensures that only authorized users (administrators) can access the system. It verifies the entered credentials and grants access to the HomePage if validation succeeds. Invalid logins are blocked with an error message.

Key functionalities:

- Input fields for username and password
- Login button with event listener
- Validation of hardcoded or database-stored credentials



```
package hostelbuddy.ui;
import javax.swing.*.*;
import java.awt.*.*;

public class LoginScreen extends JFrame {

    public LoginScreen() {
        setTitle("Login - HostelBuddy");
        setSize(1920, 1080); // Full HD
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(null);

        int width = getWidth();
        int height = getHeight();

        int wallpaperPanelWidth = (int) (width * 0.75); // 75%
        int loginPanelWidth = width - wallpaperPanelWidth; // 25%

        // ==== Wallpaper Panel ====
```

```

JLabel wallpaperLabel = new JLabel();
wallpaperLabel.setBounds(0, 0, wallpaperPanelWidth, height);

try {
    ImageIcon wallpaper = new
ImageIcon(getClass().getResource("/resources/LoginScreenBG.jpg"));
    Image img = wallpaper.getImage().getScaledInstance(wallpaperPanelWidth, height,
Image.SCALE_SMOOTH);
    wallpaperLabel.setIcon(new ImageIcon(img));
} catch (Exception e) {
    wallpaperLabel.setText("Wallpaper not found");
    wallpaperLabel.setHorizontalAlignment(SwingConstants.CENTER);
    wallpaperLabel.setFont(new Font("Segoe UI", Font.BOLD, 24));
}

add(wallpaperLabel);

// ==== Login Panel ====
JPanel loginPanel = new JPanel(null);
loginPanel.setBounds(wallpaperPanelWidth, 0, loginPanelWidth, height);
loginPanel.setBackground(new Color(33, 33, 33));
add(loginPanel);

int fieldWidth = loginPanelWidth - 60;
int xCenter = (loginPanelWidth - fieldWidth) / 2;

// ==== Center Container for Login Fields ====
JPanel centerPanel = new JPanel(null);
centerPanel.setOpaque(false);
int centerPanelHeight = 420; // More height for extra spacing
centerPanel.setBounds(0, (height - centerPanelHeight) / 2, loginPanelWidth,
centerPanelHeight);
loginPanel.add(centerPanel);

int y = 0;
int gap = 30; // More vertical space

```

```
JLabel title = new JLabel("Welcome to HostelBuddy");
title.setFont(new Font("Segoe UI", Font.BOLD, 30));
title.setForeground(Color.WHITE);
title.setBounds(xCenter, y, fieldWidth, 40);
title.setHorizontalAlignment(SwingConstants.CENTER);
centerPanel.add(title);
```

```
y += 40 + gap;
```

```
JLabel userLabel = new JLabel("Username");
userLabel.setFont(new Font("Segoe UI", Font.BOLD, 22));
userLabel.setForeground(Color.LIGHT_GRAY);
userLabel.setBounds(xCenter, y, fieldWidth, 30);
userLabel.setHorizontalAlignment(SwingConstants.CENTER);
centerPanel.add(userLabel);
```

```
y += 30 + 10;
```

```
JTextField userField = new JTextField("admin");
userField.setBounds(xCenter, y, fieldWidth, 45);
userField.setFont(new Font("Segoe UI", Font.PLAIN, 18));
centerPanel.add(userField);
```

```
y += 45 + gap;
```

```
JLabel passLabel = new JLabel("Password");
passLabel.setFont(new Font("Segoe UI", Font.BOLD, 22));
passLabel.setForeground(Color.LIGHT_GRAY);
passLabel.setBounds(xCenter, y, fieldWidth, 30);
passLabel.setHorizontalAlignment(SwingConstants.CENTER);
centerPanel.add(passLabel);
```

```
y += 30 + 10;
```

```
JPasswordField passField = new JPasswordField("admin");
passField.setBounds(xCenter, y, fieldWidth, 45);
passField.setFont(new Font("Segoe UI", Font.PLAIN, 18));
```

```

centerPanel.add(passField);

y += 45 + gap;

JButton loginBtn = new JButton("Login");
loginBtn.setFont(new Font("Segoe UI", Font.BOLD, 20));
loginBtn.setBackground(new Color(0, 120, 215));
loginBtn.setForeground(Color.WHITE);
loginBtn.setFocusPainted(false);
loginBtn.setBounds(xCenter, y, fieldWidth, 50);
centerPanel.add(loginBtn);

// ==== Button Action ====
loginBtn.addActionListener(e -> {
    String username = userField.getText();
    String password = new String(passField.getPassword());

    if (username.equals("admin") && password.equals("admin")) {
        dispose();
        new HomePage(); // Replace with your HomePage class
    } else {
        JOptionPane.showMessageDialog(this, "Invalid username or password.");
    }
});

setVisible(true);
}

public static void main(String[] args) {
    new LoginScreen();
}
}

```

6.2 New Student Registration

This module allows the administrator to register new students and assign them to available rooms. The interface collects student information including name, contact, college name, and Aadhaar number.

Key functionalities:

- Validate input fields before submission
- Auto-update the current occupancy of rooms
- Automatically set room status to 'Booked' when capacity is reached

New Student Registration

[Back](#)

Register New Student

Student ID:

Name:

Mobile Number:

Email:

Father's Name:

Mother's Name:

Address:

College Name:

Aadhaar Number:

Room Number:

[Register](#) [Clear](#)

package hostelbuddy.ui;

```

import hostelbuddy.database.DBConnection;

import javax.swing.*;
import java.awt.*;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.HashMap;

public class NewStudent extends JFrame {
    private HashMap<String, JComponent> fields;

    public NewStudent(JFrame parent) {
        setTitle("New Student Registration");
        setSize(800, 620);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setLayout(null);
        getContentPane().setBackground(new Color(230, 201, 168));

        JLabel heading = new JLabel("Register New Student");
        heading.setFont(new Font("Segoe UI", Font.BOLD, 24));
        heading.setBounds(260, 20, 400, 30);
        add(heading);

        // Field definition: label + type
        String[][] fieldData = {
            {"Student ID", "text"},
            {"Name", "text"},

```

```

{"Mobile Number", "text"},
{"Email", "text"},
{"Father's Name", "text"},
{"Mother's Name", "text"},
{"Address", "area"},
{"College Name", "text"},
{"Aadhaar Number", "text"},
{"Room Number", "combo"}
};

```

```
fields = new HashMap<>();
```

```
int y = 80;
```

```

for (String[] fd : fieldData) {
    JLabel label = new JLabel(fd[0] + ":");
    label.setBounds(100, y, 150, 25);
    add(label);

```

```
JComponent input;
```

```

switch (fd[1]) {
    case "combo":
        input = new JComboBox<>();
        loadAvailableRooms((JComboBox<String>) input);
        break;
    case "area":
        input = new JTextArea();
        ((JTextArea) input).setLineWrap(true);
        ((JTextArea) input).setWrapStyleWord(true);

```



```
JScrollPane scroll = new JScrollPane(input);
scroll.setBounds(250, y, 300, 60);
add(scroll);
fields.put(fd[0], input);
y += 70;
continue;
default:
    input = new JTextField();
}
```

```
input.setBounds(250, y, 300, 25);
add(input);
fields.put(fd[0], input);
```

```
y += 40;
}
```

```
JButton btnRegister = new JButton("Register");
btnRegister.setBounds(250, y + 20, 120, 30);
add(btnRegister);
```

```
JButton btnClear = new JButton("Clear");
btnClear.setBounds(390, y + 20, 100, 30);
add(btnClear);
```

```
JButton btnBack = new JButton("Back");
btnBack.setBounds(20, 20, 100, 25);
add(btnBack);
```

```

btnBack.addActionListener(e -> {
    dispose();
    parent.setEnabled(true);
    parent.toFront();
});

btnClear.addActionListener(e -> clearForm());

btnRegister.addActionListener(e -> registerStudent());

setVisible(true);
}

private void loadAvailableRooms(JComboBox<String> roomBox) {
    try (Connection con = DBConnection.getConnection();
        PreparedStatement ps = con.prepareStatement("SELECT room_number FROM rooms
WHERE status = 'Not Booked' AND activate = TRUE");
        java.sql.ResultSet rs = ps.executeQuery()) {
        while (rs.next()) {
            roomBox.addItem(rs.getString("room_number"));
        }
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Room loading error: " + ex.getMessage());
    }
}

private void clearForm() {
    for (String key : fields.keySet()) {
        JComponent comp = fields.get(key);

```

```

        if (comp instanceof JTextField) ((JTextField) comp).setText("");
        if (comp instanceof JTextArea) ((JTextArea) comp).setText("");
        if (comp instanceof JComboBox) ((JComboBox<?>) comp).setSelectedIndex(0);
    }
}

```

```

private void registerStudent() {
    try (Connection con = DBConnection.getConnection()) {
        String id = getText("Student ID");
        String name = getText("Name");
        String mobile = getText("Mobile Number");
        String email = getText("Email");
        String father = getText("Father's Name");
        String mother = getText("Mother's Name");
        String address = getText("Address");
        String college = getText("College Name");
        String aadhaar = getText("Aadhaar Number");
        String room = (String) ((JComboBox<?>) fields.get("Room
Number")).getSelectedItem();

        if (id.isEmpty() || name.isEmpty() || mobile.isEmpty() || email.isEmpty() || room ==
null) {
            JOptionPane.showMessageDialog(this, "Please fill all required fields.");
            return;
        }
    }
}

```

```

        PreparedStatement ps = con.prepareStatement("INSERT INTO students (student_id,
name, mobile_number, email, father_name, mother_name, address, college_name,
aadhaar_number, room_number, status) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, 'Living')");

```

```

ps.setString(1, id);
ps.setString(2, name);
ps.setString(3, mobile);
ps.setString(4, email);
ps.setString(5, father);
ps.setString(6, mother);
ps.setString(7, address);
ps.setString(8, college);
ps.setString(9, aadhaar);
ps.setString(10, room);

int inserted = ps.executeUpdate();

if (inserted > 0) {
    // Mark room as booked
    PreparedStatement roomUpdate = con.prepareStatement("UPDATE rooms SET
status = 'Booked' WHERE room_number = ?");
    roomUpdate.setString(1, room);
    roomUpdate.executeUpdate();

    JOptionPane.showMessageDialog(this, "Student registered successfully!");
    clearForm();
} else {
    JOptionPane.showMessageDialog(this, "Failed to register student.");
}

} catch (Exception ex) {
    JOptionPane.showMessageDialog(this, "Registration error: " + ex.getMessage());
}

```

```
}
```

```
private String getText(String key) {  
    JComponent comp = fields.get(key);  
    if (comp instanceof JTextField) return ((JTextField) comp).getText().trim();  
    if (comp instanceof JTextArea) return ((JTextArea) comp).getText().trim();  
    return "";  
}
```

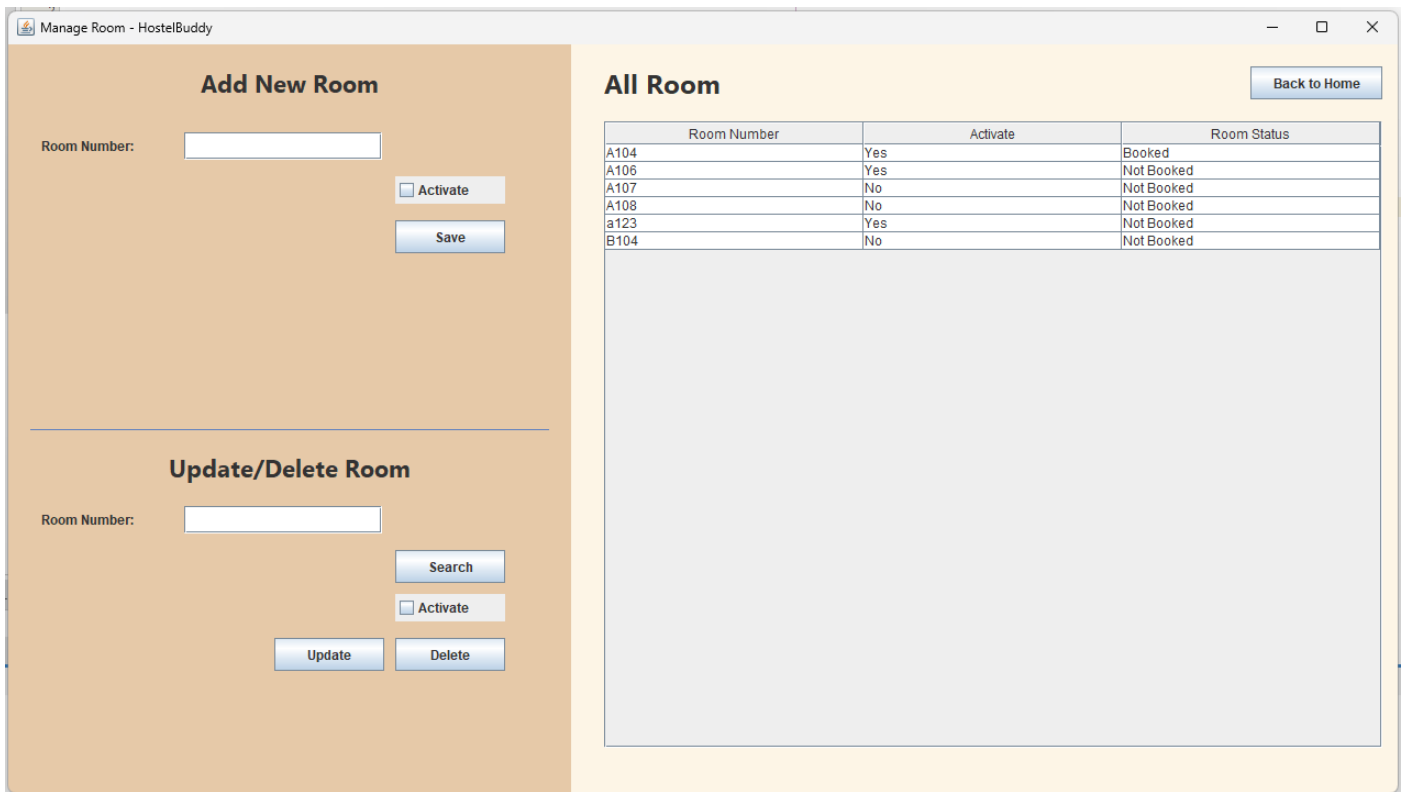
```
public static void main(String[] args) {  
    new NewStudent(null);  
}  
}
```

6.3 Room Allocation and Management

This module manages all aspects of room availability and assignment. It allows the admin to add, update, and deactivate rooms.

Key functionalities:

- Track current occupants per room
- View and manage room status ('Booked' / 'Not Booked')
- Activate or deactivate rooms



```
package hostelbuddy.ui;
```

```
import hostelbuddy.database.DBConnection;
```

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

```
import javax.swing.table.DefaultTableModel;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

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

```
public class ManageRoom extends JFrame {
    private JTable roomTable;
    private DefaultTableModel model;
    private JTextField roomField, updateRoomField;
    private JCheckBox activateCheckbox, updateActivateCheckbox;
```

```

public ManageRoom() {
    setTitle("Manage Room - HostelBuddy");
    setSize(1280, 720);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    setLayout(null);

    int leftWidth = (int) (getWidth() * 0.4);
    int rightWidth = getWidth() - leftWidth;
    int fullHeight = getHeight();

    JPanel leftPanel = new JPanel(null);
    leftPanel.setBounds(0, 0, leftWidth, fullHeight);
    leftPanel.setBackground(new Color(230, 201, 168));
    add(leftPanel);

    int halfHeight = fullHeight / 2;

    JLabel addTitle = new JLabel("Add New Room", SwingConstants.CENTER);
    addTitle.setFont(new Font("Segoe UI", Font.BOLD, 22));
    addTitle.setBounds(0, 20, leftWidth, 30);
    leftPanel.add(addTitle);

    JLabel roomLabel = new JLabel("Room Number:");
    roomLabel.setBounds(30, 80, 120, 25);
    leftPanel.add(roomLabel);

    roomField = new JTextField();
    roomField.setBounds(160, 80, 180, 25);

```

```
leftPanel.add(roomField);
```

```
activateCheckbox = new JCheckBox("Activate");
```

```
activateCheckbox.setBounds(leftWidth - 160, 120, 100, 25);
```

```
leftPanel.add(activateCheckbox);
```

```
JButton saveBtn = new JButton("Save");
```

```
saveBtn.setBounds(leftWidth - 160, 160, 100, 30);
```

```
leftPanel.add(saveBtn);
```

```
saveBtn.addActionListener(e -> saveRoom());
```

```
JSeparator separator = new JSeparator();
```

```
separator.setBounds(20, halfHeight - 10, leftWidth - 40, 1);
```

```
leftPanel.add(separator);
```

```
int startY = halfHeight + 10;
```

```
JLabel updateTitle = new JLabel("Update/Delete Room", SwingConstants.CENTER);
```

```
updateTitle.setFont(new Font("Segoe UI", Font.BOLD, 22));
```

```
updateTitle.setBounds(0, startY, leftWidth, 30);
```

```
leftPanel.add(updateTitle);
```

```
JLabel updateRoomLabel = new JLabel("Room Number:");
```

```
updateRoomLabel.setBounds(30, startY + 50, 120, 25);
```

```
leftPanel.add(updateRoomLabel);
```

```
updateRoomField = new JTextField();
```

```
updateRoomField.setBounds(160, startY + 50, 180, 25);
```



```
leftPanel.add(updateRoomField);
```

```
JButton searchBtn = new JButton("Search");
```

```
searchBtn.setBounds(leftWidth - 160, startY + 90, 100, 30);
```

```
leftPanel.add(searchBtn);
```

```
searchBtn.addActionListener(e -> searchRoom());
```

```
updateActivateCheckbox = new JCheckBox("Activate");
```

```
updateActivateCheckbox.setBounds(leftWidth - 160, startY + 130, 100, 25);
```

```
leftPanel.add(updateActivateCheckbox);
```

```
JButton updateBtn = new JButton("Update");
```

```
updateBtn.setBounds(leftWidth - 270, startY + 170, 100, 30);
```

```
leftPanel.add(updateBtn);
```

```
updateBtn.addActionListener(e -> updateRoom());
```

```
JButton deleteBtn = new JButton("Delete");
```

```
deleteBtn.setBounds(leftWidth - 160, startY + 170, 100, 30);
```

```
leftPanel.add(deleteBtn);
```

```
deleteBtn.addActionListener(e -> deleteRoom());
```

```
JPanel rightPanel = new JPanel(null);
```

```
rightPanel.setBounds(leftWidth, 0, rightWidth, fullHeight);
```

```
rightPanel.setBackground(new Color(253, 245, 230));
```

```
add(rightPanel);
```

```

JLabel allRoomTitle = new JLabel("All Room");
allRoomTitle.setFont(new Font("Segoe UI", Font.BOLD, 24));
allRoomTitle.setBounds(30, 20, 200, 30);
rightPanel.add(allRoomTitle);

JButton backBtn = new JButton("Back to Home");
backBtn.setBounds(rightWidth - 150, 20, 120, 30);
rightPanel.add(backBtn);
backBtn.addActionListener(e -> dispose());

String[] columns = {"Room Number", "Activate", "Room Status"};
model = new DefaultTableModel(columns, 0);
roomTable = new JTable(model);

JScrollPane scrollPane = new JScrollPane(roomTable);
scrollPane.setBounds(30, 70, rightWidth - 60, fullHeight - 150);
rightPanel.add(scrollPane);

loadAllRooms();
setVisible(true);
backBtn.addActionListener(e -> dispose());
}

private void loadAllRooms() {
    model.setRowCount(0);
    try (Connection con = DBConnection.getConnection();
        PreparedStatement stmt = con.prepareStatement("SELECT * FROM rooms");
        ResultSet rs = stmt.executeQuery()) {

```

```

while (rs.next()) {
    String number = rs.getString("room_number");
    String activate = rs.getBoolean("activate") ? "Yes" : "No";
    String status = rs.getString("status");
    model.addRow(new Object[]{number, activate, status});
}
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Failed to load rooms: " + e.getMessage());
}
}

```

```

private void saveRoom() {

```

```

    String roomNumber = roomField.getText().trim();
    boolean isActive = activateCheckbox.isSelected();

```

```

    if (roomNumber.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Room Number is required");
        return;
    }

```

```

    try (Connection con = DBConnection.getConnection()) {
        PreparedStatement checkStmt = con.prepareStatement("SELECT * FROM rooms
WHERE room_number = ?");
        checkStmt.setString(1, roomNumber);
        ResultSet rs = checkStmt.executeQuery();
        if (rs.next()) {
            JOptionPane.showMessageDialog(this, "Room already exists!");
            return;
        }
    }

```

```

        PreparedStatement stmt = con.prepareStatement("INSERT INTO rooms
(room_number, activate, status) VALUES (?, ?, ?)");
        stmt.setString(1, roomNumber);
        stmt.setBoolean(2, isActive);
        stmt.setString(3, "Not Booked");
        stmt.executeUpdate();

        JOptionPane.showMessageDialog(this, "Room added successfully");
        roomField.setText("");
        activateCheckbox.setSelected(false);
        loadAllRooms();
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
    }
}

private void searchRoom() {
    String room = updateRoomField.getText().trim();
    if (room.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Enter Room Number to search");
        return;
    }

    try (Connection con = DBConnection.getConnection()) {
        PreparedStatement stmt = con.prepareStatement("SELECT * FROM rooms WHERE
room_number = ?");
        stmt.setString(1, room);
        ResultSet rs = stmt.executeQuery();
    }
}

```

```

        if (rs.next()) {
            updateActivateCheckbox.setSelected(rs.getBoolean("activate"));
        } else {
            JOptionPane.showMessageDialog(this, "Room not found");
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
    }
}

```

```

private void updateRoom() {

```

```

    String room = updateRoomField.getText().trim();

```

```

    boolean isActive = updateActivateCheckbox.isSelected();

```

```

    if (room.isEmpty()) {

```

```

        JOptionPane.showMessageDialog(this, "Room Number is required");

```

```

        return;
    }

```

```

    try (Connection con = DBConnection.getConnection()) {

```

```

        PreparedStatement stmt = con.prepareStatement("UPDATE rooms SET activate = ?
WHERE room_number = ?");

```

```

        stmt.setBoolean(1, isActive);

```

```

        stmt.setString(2, room);

```

```

        int rows = stmt.executeUpdate();

```

```

        if (rows > 0) {

```

```

            JOptionPane.showMessageDialog(this, "Room updated");

```

```

            loadAllRooms();
        }
    }
}

```

```

    } else {
        JOptionPane.showMessageDialog(this, "Room not found");
    }
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
}
}

```

```

private void deleteRoom() {
    String room = updateRoomField.getText().trim();
    if (room.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Enter Room Number to delete");
        return;
    }

```

```

    int confirm = JOptionPane.showConfirmDialog(this, "Are you sure to delete this room?",
"Confirm", JOptionPane.YES_NO_OPTION);
    if (confirm != JOptionPane.YES_OPTION) return;

```

```

    try (Connection con = DBConnection.getConnection()) {
        PreparedStatement stmt = con.prepareStatement("DELETE FROM rooms WHERE
room_number = ?");
        stmt.setString(1, room);
        int rows = stmt.executeUpdate();

        if (rows > 0) {
            JOptionPane.showMessageDialog(this, "Room deleted");
            updateRoomField.setText("");
            updateActivateCheckbox.setSelected(false);

```

```

        loadAllRooms();
    } else {
        JOptionPane.showMessageDialog(this, "Room not found");
    }
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
}
}

public static void main(String[] args) {
    new ManageRoom();
}
}

```

6.4 Fee Collection and History

The fee module enables monthly fee entry and tracks historical payments by each student. It also allows the admin to export fee records.

Key functionalities:

- Select student and enter monthly fee
- Store payment amount and date
- View and filter fee history
- Export to CSV format

Student Fee Management

Student Fee Payment

Back

Student ID:

Search

Name:

Email:

Room:

Month:

January

▼

Amount:

Save Payment

Clear

Fee History Viewer

Fee History Viewer

Back

Student ID:

Month:

All

▼

Search

Export to CSV

| Student ID | Name | Email | Room | Month | Amount |
|------------|---------------------|------------------|------|----------|----------|
| 223153 | Shashwat Singh Gaur | 223153@kit.ac.in | A107 | January | 35000.00 |
| 223153 | Shashwat Singh Gaur | 223153@kit.ac.in | A107 | July | 35000.00 |
| 321654 | Ravi Gupta | 321654@gmail.com | A104 | January | 64000.00 |
| 321654 | Ravi Gupta | 321654@gmail.com | A104 | December | 64000.00 |

(1) StudentFees.java

```
package hostelbuddy.ui;

import hostelbuddy.database.DBConnection;

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

public class StudentFees extends JFrame {
    private JTextField idField, nameField, emailField, roomField, amountField;
    private JComboBox<String> monthDropdown;

    public StudentFees(JFrame parent) {
        setTitle("Student Fee Management");
        setSize(700, 500);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setLayout(null);

        Color bgColor = new Color(250, 240, 230);
        getContentPane().setBackground(bgColor);

        JLabel title = new JLabel("Student Fee Payment");
        title.setFont(new Font("Segoe UI", Font.BOLD, 24));
        title.setBounds(200, 20, 300, 30);
```

```
add(title);
```

```
JLabel idLabel = new JLabel("Student ID:");  
idLabel.setBounds(50, 80, 100, 25);  
add(idLabel);
```

```
idField = new JTextField();  
idField.setBounds(160, 80, 200, 25);  
add(idField);
```

```
JButton searchBtn = new JButton("Search");  
searchBtn.setBounds(380, 80, 100, 25);  
add(searchBtn);
```

```
searchBtn.addActionListener(e -> searchStudent());
```

```
JLabel nameLabel = new JLabel("Name:");  
nameLabel.setBounds(50, 120, 100, 25);  
add(nameLabel);
```

```
nameField = new JTextField();  
nameField.setBounds(160, 120, 320, 25);  
nameField.setEditable(false);  
add(nameField);
```

```
JLabel emailLabel = new JLabel("Email:");  
emailLabel.setBounds(50, 160, 100, 25);  
add(emailLabel);
```

```
emailField = new JTextField();  
emailField.setBounds(160, 160, 320, 25);  
emailField.setEditable(false);  
add(emailField);
```

```
JLabel roomLabel = new JLabel("Room:");  
roomLabel.setBounds(50, 200, 100, 25);  
add(roomLabel);
```

```
roomField = new JTextField();  
roomField.setBounds(160, 200, 320, 25);  
roomField.setEditable(false);  
add(roomField);
```

```
JLabel monthLabel = new JLabel("Month:");  
monthLabel.setBounds(50, 240, 100, 25);  
add(monthLabel);
```

```
monthDropdown = new JComboBox<>(new String[] {  
    "January", "February", "March", "April", "May", "June",  
    "July", "August", "September", "October", "November", "December"  
});  
monthDropdown.setBounds(160, 240, 200, 25);  
add(monthDropdown);
```

```
JLabel amountLabel = new JLabel("Amount:");  
amountLabel.setBounds(50, 280, 100, 25);  
add(amountLabel);
```

```

amountField = new JTextField();
amountField.setBounds(160, 280, 200, 25);
add(amountField);

JButton payBtn = new JButton("Save Payment");
payBtn.setBounds(160, 330, 140, 30);
add(payBtn);

JButton clearBtn = new JButton("Clear");
clearBtn.setBounds(320, 330, 100, 30);
add(clearBtn);

JButton backBtn = new JButton("Back");
backBtn.setBounds(540, 20, 100, 25);
add(backBtn);

clearBtn.addActionListener(e -> clearForm());
backBtn.addActionListener(e -> {
    dispose();
    parent.setEnabled(true);
    parent.toFront();
});

payBtn.addActionListener(e -> savePayment());

setVisible(true);
}

private void searchStudent() {

```

```

String id = idField.getText().trim();
if (id.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please enter a student ID.");
    return;
}

try (Connection con = DBConnection.getConnection()) {
    PreparedStatement stmt = con.prepareStatement("SELECT name, email,
room_number FROM students WHERE student_id = ?");
    stmt.setString(1, id);
    ResultSet rs = stmt.executeQuery();
    if (rs.next()) {
        nameField.setText(rs.getString("name"));
        emailField.setText(rs.getString("email"));
        roomField.setText(rs.getString("room_number"));
    } else {
        JOptionPane.showMessageDialog(this, "Student not found.");
    }
} catch (Exception ex) {
    JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
}

}

private void savePayment() {
    String id = idField.getText().trim();
    String name = nameField.getText().trim();
    String email = emailField.getText().trim();
    String room = roomField.getText().trim();
    String month = monthDropdown.getSelectedItem().toString();

```

```

String amount = amountField.getText().trim();

if (id.isEmpty() || name.isEmpty() || email.isEmpty() || room.isEmpty() ||
amount.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please fill all fields before saving.");
    return;
}

try (Connection con = DBConnection.getConnection()) {
    PreparedStatement checkStmt = con.prepareStatement(
        "SELECT * FROM fees WHERE student_id = ? AND month = ?");
    checkStmt.setString(1, id);
    checkStmt.setString(2, month);
    ResultSet rs = checkStmt.executeQuery();

    if (rs.next()) {
        JOptionPane.showMessageDialog(this, "Payment already made for this month.");
        return;
    }

    PreparedStatement stmt = con.prepareStatement(
        "INSERT INTO fees (student_id, name, email, room_number, month, amount)
VALUES (?, ?, ?, ?, ?, ?)");
    stmt.setString(1, id);
    stmt.setString(2, name);
    stmt.setString(3, email);
    stmt.setString(4, room);
    stmt.setString(5, month);
    stmt.setString(6, amount);
}

```

```

        stmt.executeUpdate();
        JOptionPane.showMessageDialog(this, "Payment saved successfully.");
        clearForm();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
    }
}

```

```

private void clearForm() {
    idField.setText("");
    nameField.setText("");
    emailField.setText("");
    roomField.setText("");
    amountField.setText("");
    monthDropDown.setSelectedIndex(0);
}

```

```

public static void main(String[] args) {
    new StudentFees(null);
}
}

```

(2) FeeHistoryViewer.java

```

package hostelbuddy.ui;

```

```

import hostelbuddy.database.DBConnection;
import hostelbuddy.utils.ExportUtil;

```

```

import javax.swing.*.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*.*;
import java.sql.*.*;

public class FeeHistoryViewer extends JFrame {
    private JTextField studentIdField;
    private JComboBox<String> monthDropdown;
    private DefaultTableModel model;
    private JTable table;

    public FeeHistoryViewer(JFrame parent) {
        setTitle("Fee History Viewer");
        setSize(950, 600);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setLayout(null);

        getContentPane().setBackground(new Color(255, 250, 240));

        JLabel title = new JLabel("Fee History Viewer");
        title.setFont(new Font("Segoe UI", Font.BOLD, 26));
        title.setBounds(320, 20, 400, 30);
        add(title);

        JLabel studentLabel = new JLabel("Student ID:");
        studentLabel.setBounds(50, 80, 100, 25);
        add(studentLabel);
    }
}

```



```
studentIdField = new JTextField();  
studentIdField.setBounds(150, 80, 150, 25);  
add(studentIdField);
```

```
JLabel monthLabel = new JLabel("Month:");  
monthLabel.setBounds(320, 80, 60, 25);  
add(monthLabel);
```

```
monthDropdown = new JComboBox<>(new String[] {  
    "All", "January", "February", "March", "April", "May", "June",  
    "July", "August", "September", "October", "November", "December"  
});  
monthDropdown.setBounds(380, 80, 150, 25);  
add(monthDropdown);
```

```
JButton searchBtn = new JButton("Search");  
searchBtn.setBounds(550, 80, 100, 25);  
add(searchBtn);
```

```
JButton exportBtn = new JButton("Export to CSV");  
exportBtn.setBounds(660, 80, 150, 25);  
add(exportBtn);
```

```
JButton backBtn = new JButton("Back");  
backBtn.setBounds(820, 20, 100, 25);  
add(backBtn);
```

```
searchBtn.addActionListener(e -> fetchRecords());
```

```

exportBtn.addActionListener(e -> {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle("Save CSV");
    int result = fileChooser.showSaveDialog(this);

    if (result == JFileChooser.APPROVE_OPTION) {
        String path = fileChooser.getSelectedFile().getAbsolutePath();
        if (!path.endsWith(".csv")) {
            path += ".csv";
        }
        ExportUtil.exportTableToCSV(table, path);
    }
});

```

```

backBtn.addActionListener(e -> {
    dispose();
    parent.setEnabled(true);
    parent.toFront();
});

```

// Table Setup

```

String[] columns = { "Student ID", "Name", "Email", "Room", "Month", "Amount" };
model = new DefaultTableModel(columns, 0);
table = new JTable(model);

JScrollPane scrollPane = new JScrollPane(table);
scrollPane.setBounds(50, 130, 850, 400);
add(scrollPane);

```

```

setVisible(true);
}

private void fetchRecords() {
    model.setRowCount(0); // clear table

    String studentId = studentIdField.getText().trim();
    String month = monthDropDown.getSelectedItem().toString();

    String query = "SELECT * FROM fees WHERE 1=1";
    if (!studentId.isEmpty()) query += " AND student_id = ?";
    if (!month.equals("All")) query += " AND month = ?";

    try (Connection con = DBConnection.getConnection();
        PreparedStatement stmt = con.prepareStatement(query)) {

        int paramIndex = 1;
        if (!studentId.isEmpty()) stmt.setString(paramIndex++, studentId);
        if (!month.equals("All")) stmt.setString(paramIndex, month);

        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            model.addRow(new Object[] {
                rs.getString("student_id"),
                rs.getString("name"),
                rs.getString("email"),
                rs.getString("room_number"),
                rs.getString("month"),
                rs.getString("amount")
            });
        }
    }
}

```

```

        });
    }

    if (model.getRowCount() == 0) {
        JOptionPane.showMessageDialog(this, "No records found.");
    }

} catch (SQLException ex) {
    JOptionPane.showMessageDialog(this, "Database error: " + ex.getMessage());
}
}

public static void main(String[] args) {
    new FeeHistoryViewer(null);
}
}

```

6.5 Employee Management

This module handles the data related to hostel staff. The admin can add new employees, update existing records, and manage salary history.

Key functionalities:

- Add/update/delete employee records
- Track employee work status ('Working' / 'Leaved')
- Log monthly salary payments

New Employee Registration

Register New Employee

Employee ID:

Name:

Gender:

Mobile Number:

Email:

Aadhaar Number:

Address:

Role:

Joining Date:

(1) NewEmployee.java

```
package hostelbuddy.ui;
```

```
import hostelbuddy.database.DBConnection;
```

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

```
import java.awt.*.*;
```

```
import java.sql.Connection;
```

```
import java.sql.PreparedStatement;
```

```
import java.time.LocalDate;
```

```
import java.util.HashMap;
```

```
public class NewEmployee extends JFrame {
```

```
private HashMap<String, JComponent> fields;
```

```
public NewEmployee(JFrame parent) {  
    setTitle("New Employee Registration");  
    setSize(800, 600);  
    setLocationRelativeTo(null);  
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);  
    setLayout(null);  
    getContentPane().setBackground(new Color(245, 255, 250));
```

```
    JLabel heading = new JLabel("Register New Employee");  
    heading.setFont(new Font("Segoe UI", Font.BOLD, 24));  
    heading.setBounds(250, 20, 400, 30);  
    add(heading);
```

```
// Field labels and their types
```

```
String[][] fieldData = {  
    {"Employee ID", "text"},  
    {"Name", "text"},  
    {"Gender", "combo"},  
    {"Mobile Number", "text"},  
    {"Email", "text"},  
    {"Aadhaar Number", "text"},  
    {"Address", "area"},  
    {"Role", "combo-role"},  
    {"Joining Date", "date"}  
};
```

```
fields = new HashMap<>();
```

```

int y = 80;

for (String[] fd : fieldData) {
    JLabel label = new JLabel(fd[0] + ":");
    label.setBounds(100, y, 150, 25);
    add(label);

    JComponent input;

    switch (fd[1]) {
        case "combo":
            input = new JComboBox<>(new String[]{"Male", "Female", "Other"});
            break;
        case "combo-role":
            input = new JComboBox<>(new String[]{"Warden", "Clerk", "Cleaner", "Cook",
"Security"});
            break;
        case "area":
            input = new JTextArea();
            ((JTextArea) input).setLineWrap(true);
            ((JTextArea) input).setWrapStyleWord(true);
            JScrollPane scroll = new JScrollPane(input);
            scroll.setBounds(250, y, 300, 60);
            add(scroll);
            fields.put(fd[0], input);
            y += 70;
            continue;
        case "date":
            input = new JTextField(LocalDate.now().toString());

```

```
        break;
    default:
        input = new JTextField();
    }
}
```

```
input.setBounds(250, y, 300, 25);
add(input);
fields.put(fd[0], input);
```

```
y += 40;
}
```

```
 JButton registerBtn = new JButton("Register");
registerBtn.setBounds(250, y + 20, 120, 30);
add(registerBtn);
```

```
 JButton clearBtn = new JButton("Clear");
clearBtn.setBounds(390, y + 20, 100, 30);
add(clearBtn);
```

```
 JButton backBtn = new JButton("Back");
backBtn.setBounds(20, 20, 100, 25);
add(backBtn);
```

```
backBtn.addActionListener(e -> {
    dispose();
    parent.setEnabled(true);
    parent.toFront();
});
```



```

clearBtn.addActionListener(e -> clearForm());

registerBtn.addActionListener(e -> registerEmployee());

setVisible(true);
}

private void clearForm() {
    for (String key : fields.keySet()) {
        JComponent comp = fields.get(key);
        if (comp instanceof JTextField) ((JTextField) comp).setText("");
        if (comp instanceof JTextArea) ((JTextArea) comp).setText("");
        if (comp instanceof JComboBox) ((JComboBox<?>) comp).setSelectedIndex(0);
    }
    ((JTextField) fields.get("Joining Date")).setText(LocalDate.now().toString());
}

private void registerEmployee() {
    String id = ((JTextField) fields.get("Employee ID")).getText().trim();
    String name = ((JTextField) fields.get("Name")).getText().trim();
    String gender = (String) ((JComboBox<?>) fields.get("Gender")).getSelectedItem();
    String mobile = ((JTextField) fields.get("Mobile Number")).getText().trim();
    String email = ((JTextField) fields.get("Email")).getText().trim();
    String aadhaar = ((JTextField) fields.get("Aadhaar Number")).getText().trim();
    String address = ((JTextArea) fields.get("Address")).getText().trim();
    String role = (String) ((JComboBox<?>) fields.get("Role")).getSelectedItem();
    String joinDate = ((JTextField) fields.get("Joining Date")).getText().trim();

```

```

if (id.isEmpty() || name.isEmpty() || mobile.isEmpty() || email.isEmpty() ||
    aadhaar.isEmpty() || address.isEmpty() || joinDate.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please fill in all fields.");
    return;
}

try (Connection con = DBConnection.getConnection()) {
    PreparedStatement ps = con.prepareStatement(
        "INSERT INTO employees (employee_id, name, gender, mobile_number, email,
aadhaar_number, address, role, joining_date) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)"
    );
    ps.setString(1, id);
    ps.setString(2, name);
    ps.setString(3, gender);
    ps.setString(4, mobile);
    ps.setString(5, email);
    ps.setString(6, aadhaar);
    ps.setString(7, address);
    ps.setString(8, role);
    ps.setString(9, joinDate);

    int inserted = ps.executeUpdate();
    if (inserted > 0) {
        JOptionPane.showMessageDialog(this, "Employee registered successfully!");
        clearForm();
    } else {
        JOptionPane.showMessageDialog(this, "Failed to register employee.");
    }
}

```

```

    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
    }
}

public static void main(String[] args) {
    new NewEmployee(null);
}
}

```

The screenshot shows a Java Swing window titled "Employee Payment". The window contains a "Back" button in the top-left corner. The main content area is titled "Employee Payment" and contains the following fields:

- Select Employee:** A dropdown menu with "Sonu Varma (123456)" selected.
- Month:** A dropdown menu with "January" selected.
- Year:** A dropdown menu with "2025" selected.
- Amount Paid (₹):** An empty text input field.
- Payment Date:** A text input field containing "2025-05-25".
- Remarks (optional):** A large, empty text area.

At the bottom of the form, there are two buttons: "Submit Payment" and "Clear".

(2) EmployeePayment.java

```
package hostelbuddy.ui;
```

```
import hostelbuddy.database.DBConnection;
```

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

```
import java.awt.*;
```

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

```
import java.time.LocalDate;
```

```
import java.util.LinkedHashMap;
```

```
public class EmployeePayment extends JFrame {
```

```
    private JComboBox<String> employeeBox, monthBox, yearBox;
```

```
    private JTextField amountField, dateField;
```

```
    private JTextArea remarksArea;
```

```
    private LinkedHashMap<String, String> employeeMap = new LinkedHashMap<>(); // name
```

```
→ id
```

```
    public EmployeePayment(JFrame parent) {
```

```
        setTitle("Employee Payment");
```

```
        setSize(700, 550);
```

```
        setLocationRelativeTo(null);
```

```
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
```

```
        setLayout(null);
```

```
        getContentPane().setBackground(new Color(250, 255, 240));
```

```
        JLabel heading = new JLabel("Employee Payment");
```

```
        heading.setFont(new Font("Segoe UI", Font.BOLD, 24));
```

```
        heading.setBounds(240, 20, 300, 30);
```

```
        add(heading);
```

```
        int y = 80;
```

```

// Employee
addLabel("Select Employee:", 100, y);
employeeBox = new JComboBox<>();
employeeBox.setBounds(250, y, 300, 25);
add(employeeBox);
loadEmployees();

// Month
y += 40;
addLabel("Month:", 100, y);
monthBox = new JComboBox<>(new String[]{
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
});
monthBox.setBounds(250, y, 140, 25);
add(monthBox);

// Year
addLabel("Year:", 410, y);
yearBox = new JComboBox<>();
for (int yr = 2022; yr <= LocalDate.now().getYear() + 1; yr++) {
    yearBox.addItem(String.valueOf(yr));
}
yearBox.setSelectedItem(String.valueOf(LocalDate.now().getYear()));
yearBox.setBounds(460, y, 90, 25);
add(yearBox);

// Amount

```

```

y += 40;
addLabel("Amount Paid (₹):", 100, y);
amountField = new JTextField();
amountField.setBounds(250, y, 300, 25);
add(amountField);

// Payment Date
y += 40;
addLabel("Payment Date:", 100, y);
dateField = new JTextField(LocalDate.now().toString());
dateField.setBounds(250, y, 300, 25);
add(dateField);

// Remarks
y += 40;
addLabel("Remarks (optional):", 100, y);
remarksArea = new JTextArea();
remarksArea.setLineWrap(true);
remarksArea.setWrapStyleWord(true);
JScrollPane scroll = new JScrollPane(remarksArea);
scroll.setBounds(250, y, 300, 60);
add(scroll);

// Buttons
y += 80;
JButton btnPay = new JButton("Submit Payment");
btnPay.setBounds(250, y, 150, 30);
add(btnPay);

```

```
JButton btnClear = new JButton("Clear");  
btnClear.setBounds(410, y, 100, 30);  
add(btnClear);
```

```
JButton btnBack = new JButton("Back");  
btnBack.setBounds(20, 20, 100, 25);  
add(btnBack);
```

```
btnPay.addActionListener(e -> submitPayment());  
btnClear.addActionListener(e -> clearForm());  
btnBack.addActionListener(e -> {  
    dispose();  
    parent.setEnabled(true);  
    parent.toFront();  
});
```

```
setVisible(true);  
}
```

```
private void addLabel(String text, int x, int y) {  
    JLabel label = new JLabel(text);  
    label.setBounds(x, y, 140, 25);  
    add(label);  
}
```

```
private void loadEmployees() {  
    try (Connection con = DBConnection.getConnection();  
        PreparedStatement ps = con.prepareStatement("SELECT employee_id, name FROM  
employees WHERE status = 'Working'");
```

```

ResultSet rs = ps.executeQuery() {

while (rs.next()) {
    String id = rs.getString("employee_id");
    String name = rs.getString("name");
    String label = name + " (" + id + ")";
    employeeBox.addItem(label);
    employeeMap.put(label, id);
}

} catch (Exception ex) {
    JOptionPane.showMessageDialog(this, "Error loading employees: " +
ex.getMessage());
}
}

private void submitPayment() {
    String empLabel = (String) employeeBox.getSelectedItem();
    String empId = employeeMap.get(empLabel);
    String name = empLabel.split(" \\(")[0];
    String month = (String) monthBox.getSelectedItem();
    String year = (String) yearBox.getSelectedItem();
    String amount = amountField.getText().trim();
    String date = dateField.getText().trim();
    String remarks = remarksArea.getText().trim();

    if (empId == null || amount.isEmpty() || date.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Please fill in all required fields.");
        return;
    }
}

```



```

}

try (Connection con = DBConnection.getConnection();
    PreparedStatement ps = con.prepareStatement(
        "INSERT INTO employee_payments (employee_id, name, month, year, amount,
payment_date, remarks) VALUES (?, ?, ?, ?, ?, ?, ?)"
    )) {
    ps.setString(1, emplId);
    ps.setString(2, name);
    ps.setString(3, month);
    ps.setInt(4, Integer.parseInt(year));
    ps.setDouble(5, Double.parseDouble(amount));
    ps.setDate(6, Date.valueOf(date));
    ps.setString(7, remarks);

    int rows = ps.executeUpdate();
    if (rows > 0) {
        JOptionPane.showMessageDialog(this, "Payment recorded successfully!");
        clearForm();
    } else {
        JOptionPane.showMessageDialog(this, "Payment failed.");
    }
} catch (Exception ex) {
    JOptionPane.showMessageDialog(this, "Payment error: " + ex.getMessage());
}

}

private void clearForm() {
    employeeBox.setSelectedIndex(0);

```

```
monthBox.setSelectedIndex(LocalDate.now().getMonthValue() - 1);
yearBox.setSelectedItem(String.valueOf(LocalDate.now().getYear()));
amountField.setText("");
dateField.setText(LocalDate.now().toString());
remarksArea.setText("");
}

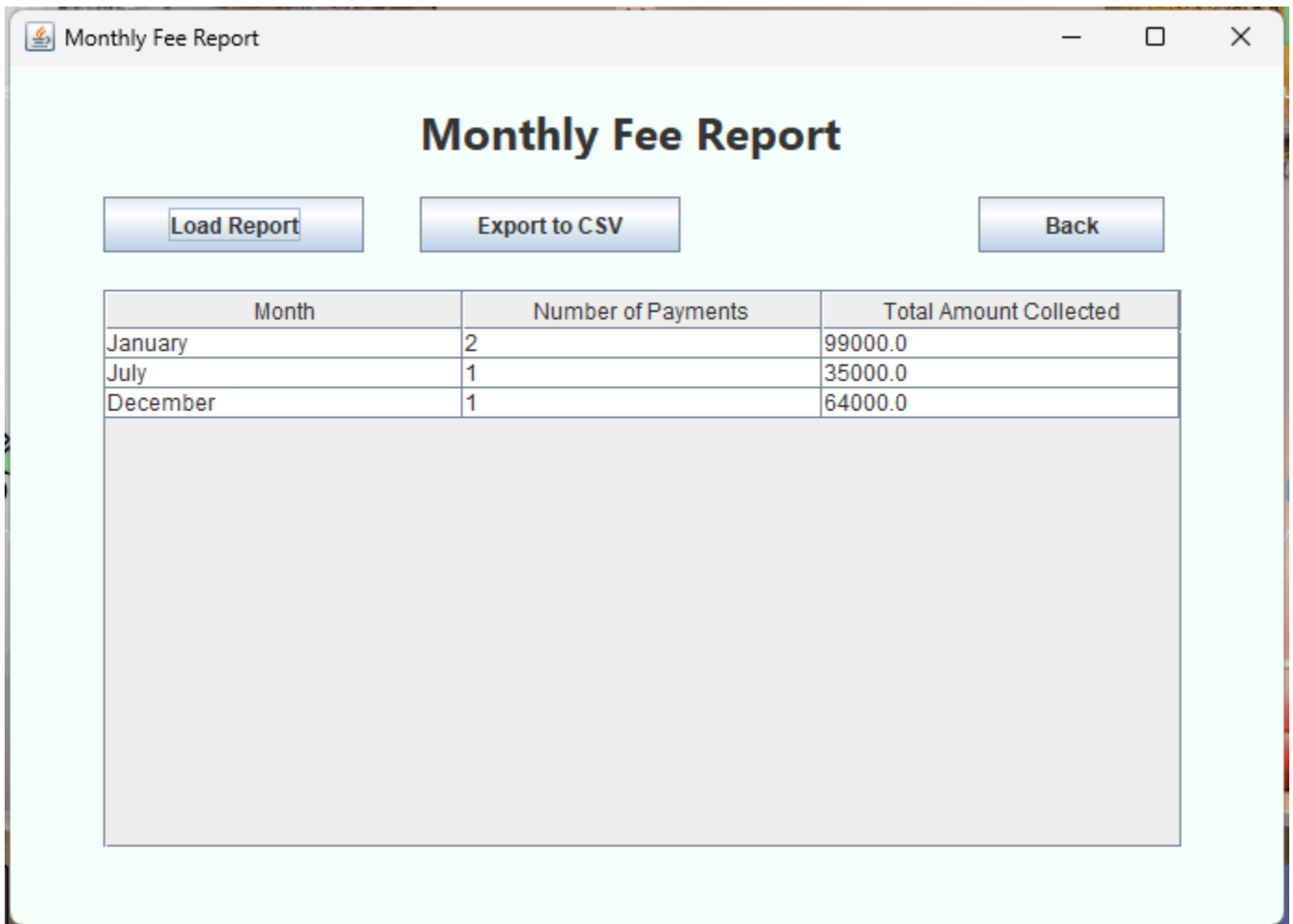
public static void main(String[] args) {
    new EmployeePayment(null);
}
}
```

6.6 Report Generation and Export

This module includes the reporting capabilities of HostelBuddy. It allows exporting student fee history and employee payment records to .csv format, which can be used for backups or audits.

Key functionalities:

- Generate monthly fee reports
- Export data in structured tabular format
- Save files for offline use



(1) MonthlyFeeReport.java

```
package hostelbuddy.ui;
```

```
import hostelbuddy.database.DBConnection;
```

```
import hostelbuddy.utils.ExportUtil;
```

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

```
import javax.swing.table.DefaultTableModel;
```

```

import java.awt.*;

import java.sql.*;

public class MonthlyFeeReport extends JFrame {

    private DefaultTableModel model;

    private JTable table;

    public MonthlyFeeReport(JFrame parent) {

        setTitle("Monthly Fee Report");

        setSize(700, 500);

        setLocationRelativeTo(null);

        setDefaultCloseOperation(DISPOSE_ON_CLOSE);

        setLayout(null);

        getContentPane().setBackground(new Color(245, 255, 250));

        JLabel title = new JLabel("Monthly Fee Report");
        title.setFont(new Font("Segoe UI", Font.BOLD, 24));
        title.setBounds(220, 20, 300, 30);
        add(title);

        JButton loadBtn = new JButton("Load Report");
        loadBtn.setBounds(50, 70, 140, 30);
        add(loadBtn);

        JButton exportBtn = new JButton("Export to CSV");
        exportBtn.setBounds(220, 70, 140, 30);
        add(exportBtn);
    }
}

```

```

JButton backBtn = new JButton("Back");
backBtn.setBounds(520, 70, 100, 30);
add(backBtn);

String[] columns = { "Month", "Number of Payments", "Total Amount Collected" };
model = new DefaultTableModel(columns, 0);
table = new JTable(model);

JScrollPane scrollPane = new JScrollPane(table);
scrollPane.setBounds(50, 120, 580, 300);
add(scrollPane);

loadBtn.addActionListener(e -> loadReport());

exportBtn.addActionListener(e -> {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle("Save CSV");
    int result = fileChooser.showSaveDialog(this);

    if (result == JFileChooser.APPROVE_OPTION) {
        String path = fileChooser.getSelectedFile().getAbsolutePath();
        if (!path.endsWith(".csv")) {
            path += ".csv";
        }
        ExportUtil.exportTableToCSV(table, path);
    }
});

backBtn.addActionListener(e -> {

```

```
dispose();
parent.setEnabled(true);
parent.toFront();
});
```

```
setVisible(true);
}
```

```
private void loadReport() {
    model.setRowCount(0);
```

```
    String query = "SELECT month, COUNT(*) as count, SUM(amount) as total FROM fees
GROUP BY month ORDER BY FIELD(month, " +
```

```
"'January','February','March','April','May','June','July','August','September','October','November','December')";
```

```
try (Connection con = DBConnection.getConnection());
    PreparedStatement stmt = con.prepareStatement(query);
    ResultSet rs = stmt.executeQuery()) {
```

```
while (rs.next()) {
    model.addRow(new Object[] {
        rs.getString("month"),
        rs.getInt("count"),
        rs.getDouble("total")
    });
}
```

```

        if (model.getRowCount() == 0) {
            JOptionPane.showMessageDialog(this, "No data available for report.");
        }

    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(this, "Error loading report: " + ex.getMessage());
    }
}

public static void main(String[] args) {
    new MonthlyFeeReport(null);
}
}

```

(2) ExportUtil.java

```

package hostelbuddy.utils;

import javax.swing.*.*;
import javax.swing.table.TableModel;
import java.io.*;

public class ExportUtil {

    public static void exportTableToCSV(JTable table, String filePath) {
        try (FileWriter csv = new FileWriter(filePath)) {
            TableModel model = table.getModel();

```

```

// Write header
for (int i = 0; i < model.getColumnCount(); i++) {
    csv.write(model.getColumnNames(i));
    if (i < model.getColumnCount() - 1) csv.write(",");
}
csv.write("\n");

// Write data
for (int row = 0; row < model.getRowCount(); row++) {
    for (int col = 0; col < model.getColumnCount(); col++) {
        csv.write(String.valueOf(model.getValueAt(row, col)));
        if (col < model.getColumnCount() - 1) csv.write(",");
    }
    csv.write("\n");
}

JOptionPane.showMessageDialog(null, "CSV exported successfully to: " + filePath);
} catch (IOException e) {
    JOptionPane.showMessageDialog(null, "Export failed: " + e.getMessage());
}
}
}

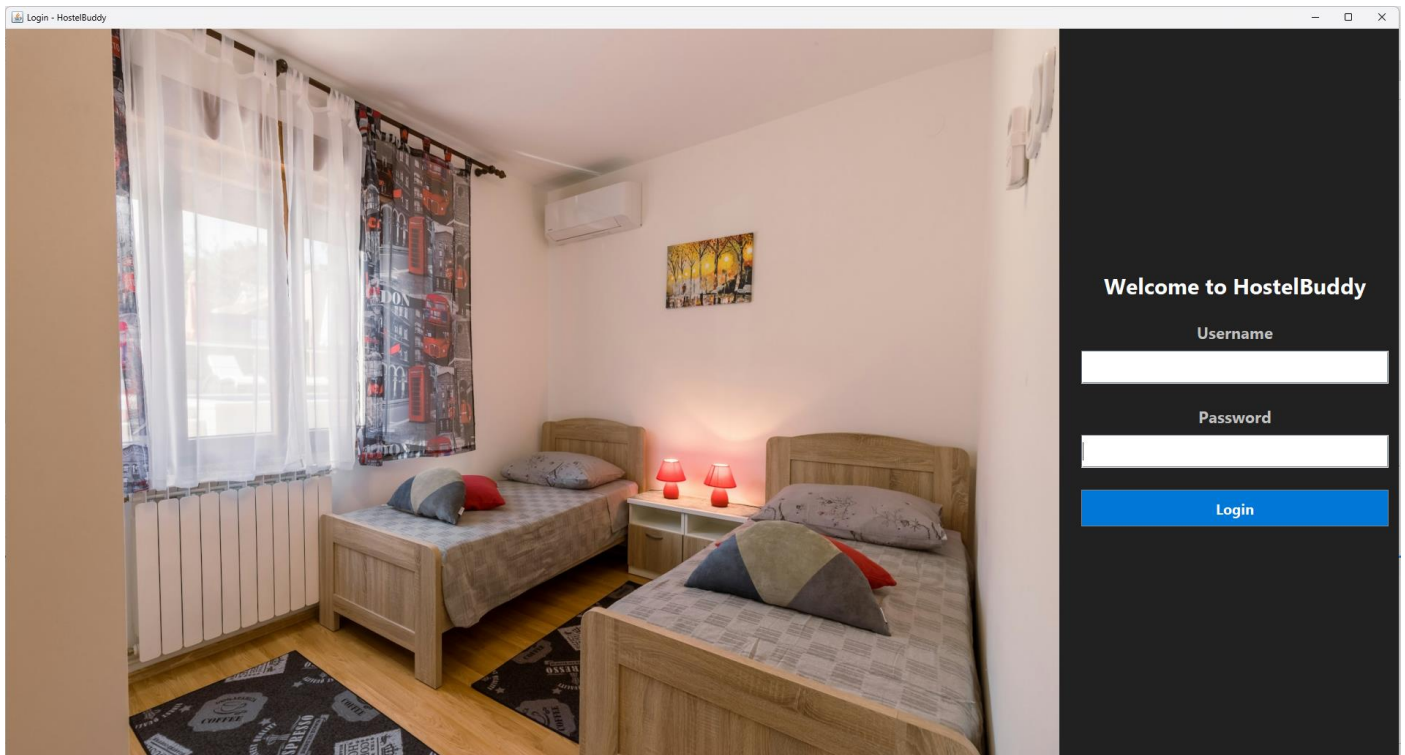
```

7.USER INTERFACE SCREENS

The visual experience plays a critical role in determining the usability and professionalism of any desktop application. HostelBuddy was developed with a focus on clean, intuitive, and well-aligned user interface screens using Java Swing. Below are the key user interface screens implemented in the project:

7.1 Login Screen

This is the entry point to the system. It contains input fields for username and password along with a login button. Basic validation and authentication logic is applied here.

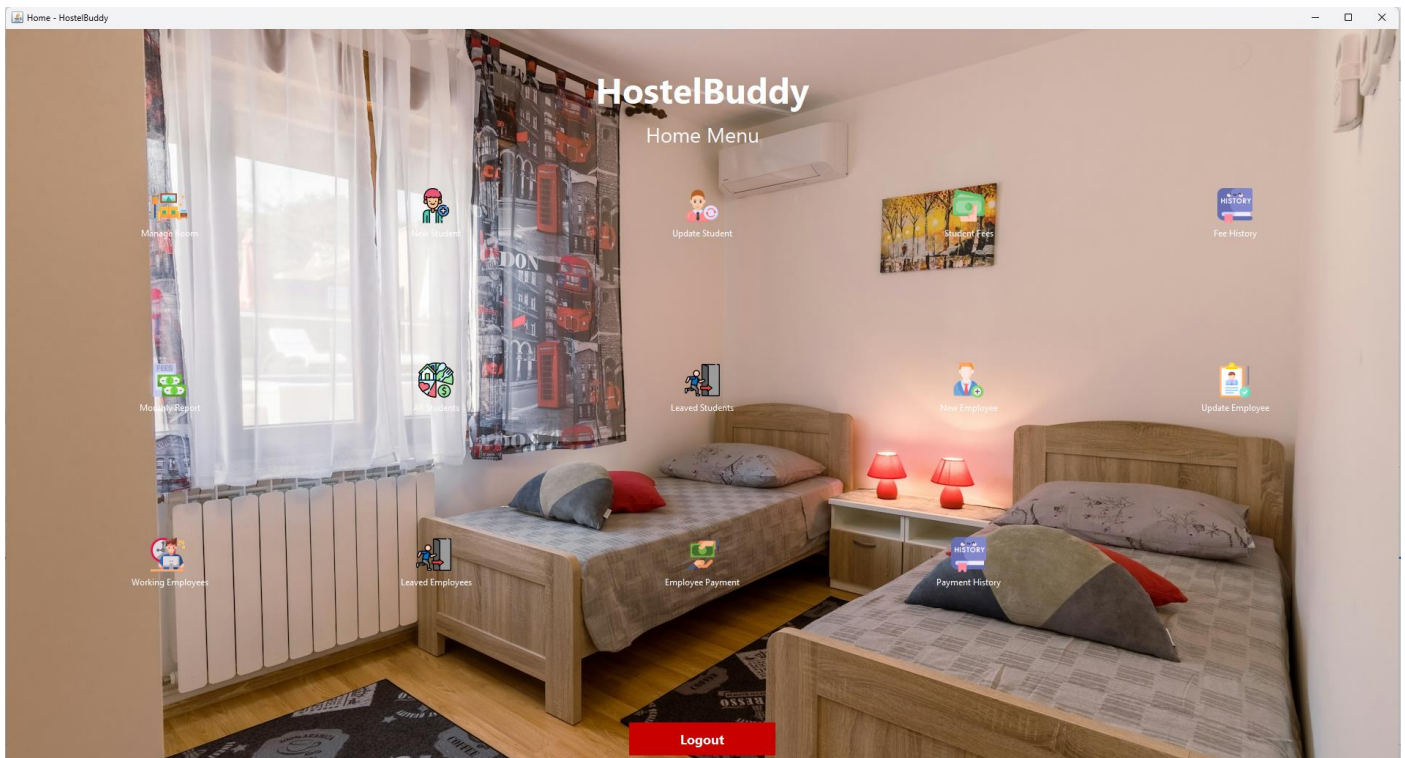


7.2 Home Dashboard

After a successful login, the user is directed to the Home Dashboard. This screen uses a card-based layout where each card represents a module of the system (e.g., Student, Room, Employee, Fee, Report).

Key features:

- Card icons with labels for intuitive navigation
- Visually separated sections
- Logout button at the bottom for easy exit



7.3 Add Student Form

This form allows administrators to enter new student information such as name, mobile number, college, Aadhaar number, and room allocation.

Key elements:

- Text fields and combo boxes
- Submit and reset buttons
- Room status updates on submission

New Student Registration

Back

Register New Student

Student ID:

Name:

Mobile Number:

Email:

Father's Name:

Mother's Name:

Address:

College Name:

Aadhaar Number:

Room Number:

Register Clear

7.4 Fee Management Panel

The fee panel supports fee entry for a selected student and displays their fee payment history. Admin can export data from this screen.

Includes:

- Fee entry form
- Fee history table with filters
- Export to CSV functionality

The screenshot shows a window titled "Student Fee Management" with standard window controls (minimize, maximize, close). The main content area has a light orange background and is titled "Student Fee Payment" in bold black text. In the top right corner of the form area is a "Back" button. The form contains the following fields and controls:

- Student ID:** A text input field followed by a "Search" button.
- Name:** A text input field.
- Email:** A text input field.
- Room:** A text input field.
- Month:** A dropdown menu currently showing "January".
- Amount:** A text input field.
- At the bottom of the form are two buttons: "Save Payment" and "Clear".

7.5 Employee Panel

This interface allows users to add, update, or delete employee records. It also shows their working status and provides access to payment history.

Key components:

- Employee registration form
- Table to view/edit employee data
- Salary payment entry

Update/Delete Employee

Back

Update/Delete Employee

Enter Employee ID:

Search

Employee ID:

Name:

Gender:

Male

Mobile Number:

Email:

Aadhaar Number:

Address:

Role:

Warden

Joining Date:

Update

Mark as Leaved


Clear

7.6 Sample Reports

HostelBuddy includes a section for viewing and exporting reports like monthly fee collections or employee payments. These reports are exportable in .csv format.

85

 Monthly Fee Report


 Monthly Fee Report


Load Report

Export to CSV

Back

| Month | Number of Payments | Total Amount Collected |
|----------|--------------------|------------------------|
| January | 2 | 99000.0 |
| July | 1 | 35000.0 |
| December | 1 | 64000.0 |

 Fee History Viewer

 Fee History Viewer

Student ID:

Month:

All

Search

Export to CSV

Back

| Student ID | Name | Email | Room | Month | Amount |
|------------|---------------------|------------------|------|----------|----------|
| 223153 | Shashwat Singh Gaur | 223153@kit.ac.in | A107 | January | 35000.00 |
| 223153 | Shashwat Singh Gaur | 223153@kit.ac.in | A107 | July | 35000.00 |
| 321654 | Ravi Gupta | 321654@gmail.com | A104 | January | 64000.00 |
| 321654 | Ravi Gupta | 321654@gmail.com | A104 | December | 64000.00 |

Employee Payment History

Employee Payment History

Back

Employee:

All

Month:

All

Year:

All

Search

Export to CSV

| ID | Employee ID | Name | Month | Year | Amount | Payment Date | Remarks |
|----|-------------|------------|----------|------|---------|--------------|--------------------|
| 1 | 123456 | Sonu Varma | January | 2025 | 30000.0 | 2025-04-23 | Salary Of January |
| 2 | 362514 | Jiya Singh | February | 2025 | 30000.0 | 2025-04-23 | Salary Of February |
| | | | | | | | |

8. CODE SNIPPETS AND LOGIC

The success of HostelBuddy lies in its well-structured and modular programming logic. This section provides an overview of the core logic implemented through code snippets that define how the application communicates with the database, handles user actions, and performs export operations.

8.1 Database Connectivity

HostelBuddy uses JDBC (Java Database Connectivity) to establish a connection with the MySQL database. A reusable DBConnection.java class is implemented for this purpose.

```
public class DBConnection {  
    public static Connection getConnection() {  
        try {  
            Class.forName("com.mysql.cj.jdbc.Driver");  
            return DriverManager.getConnection(  
                "jdbc:mysql://localhost:3306/hostelbuddy", "root", "your_password");  
        } catch (Exception e) {  
            JOptionPane.showMessageDialog(null, e);  
            return null;  
        }  
    }  
}
```

This method is called in all forms that require database operations.

8.2 Login Validation

The login screen validates the entered credentials. If valid, it redirects the user to the Home Page.


```
String username = txtUsername.getText();
String password = new String(txtPassword.getPassword());

if(username.equals("admin") && password.equals("admin")) {
    setVisible(false);
    new HomePage().setVisible(true);
} else {
    JOptionPane.showMessageDialog(null, "Incorrect username or password");
}
```

8.3 Insertion and Update Queries

Student and fee records are inserted using PreparedStatement for safety and performance.

```
Connection con = DBConnection.getConnection();
PreparedStatement ps = con.prepareStatement("INSERT INTO students (name, mobile_no,
room_number, status) VALUES (?, ?, ?, ?)");
ps.setString(1, name);
ps.setString(2, mobile);
ps.setString(3, room);
ps.setString(4, "Living");
ps.executeUpdate();
```

Room status and occupancy are updated accordingly.

```
PreparedStatement ps2 = con.prepareStatement("UPDATE rooms SET current_occupants =
current_occupants + 1 WHERE room_number = ?");
ps2.setString(1, room);
ps2.executeUpdate();
```

8.4 Export to CSV Functionality

A utility class (ExportUtil.java) is used to export table data into a CSV file.

```
public static void exportTableToCSV(JTable table, String path) {
    try (FileWriter fw = new FileWriter(path)) {
        for (int i = 0; i < table.getColumnCount(); i++) {
            fw.write(table.getColumnName(i) + (i < table.getColumnCount() - 1 ? "," : ""));
        }
    }
}
```

```

    }
    fw.write("
");
    for (int i = 0; i < table.getRowCount(); i++) {
        for (int j = 0; j < table.getColumnCount(); j++) {
            fw.write(table.getValueAt(i, j).toString() + (j < table.getColumnCount() - 1 ? "," :
""));
        }
        fw.write("
");
    }
} catch (Exception e) {
    JOptionPane.showMessageDialog(null, e);
}
}

```

8.5 Form Events and GUI Actions

All forms use Java Swing's ActionListener and MouseListener interfaces for user interaction.

```

btnSubmit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        saveStudentDetails();
    }
});

btnExport.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        ExportUtil.exportTableToCSV(jTable1, "students.csv");
    }
});

```

All these logical segments work together to make HostelBuddy a responsive and functionally rich application.

9. TESTING AND OUTPUT

Thorough testing ensures the stability, accuracy, and reliability of any software system. For HostelBuddy, various levels of testing were conducted to ensure that all modules perform as expected under both normal and exceptional conditions.

9.1 Functional Testing of Modules

Each functional module was tested independently and in combination with others to ensure correctness and integration:

- **Login Module:** Tested for valid and invalid credentials.
- **Student Module:** Verified record addition, update, and deletion.
- **Room Management:** Tested occupancy limits and automatic status change.
- **Fee Module:** Checked data entry, filtering, and historical tracking.
- **Employee Management:** Validated employee addition and salary logging.
- **CSV Export:** Verified correct structure and export path.

Test cases were created to simulate both correct usage and invalid scenarios.

9.2 Input Validations

Proper input validation is essential to ensure only valid data enters the system. All forms include the following checks:

- **Empty field checks** – Required fields cannot be submitted blank.
- **Data type validations** – Mobile numbers must be numeric; email must follow format.
- **Dropdown constraints** – Only allowed status/room values selectable.

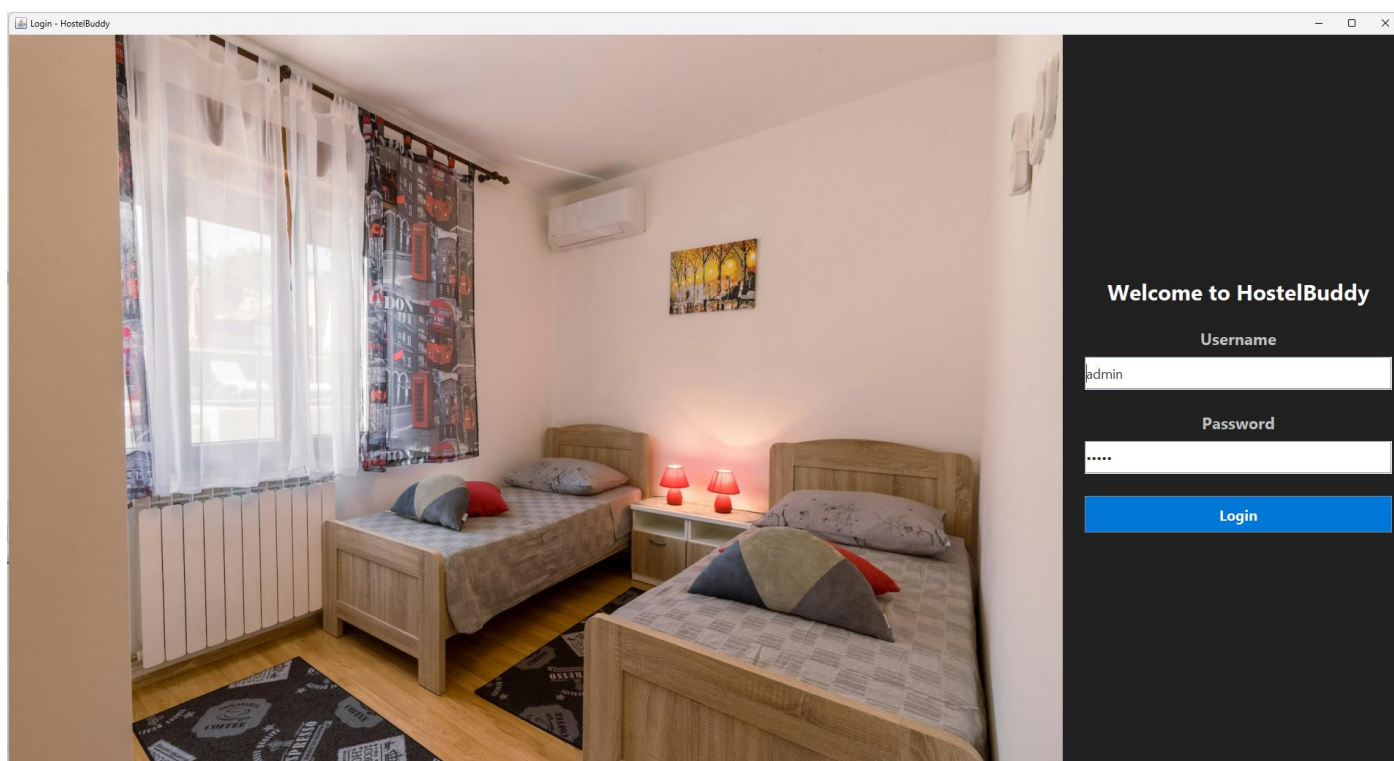
- **Unique constraints** – Duplicate entries (e.g., Aadhaar number) are restricted manually or via database logic.

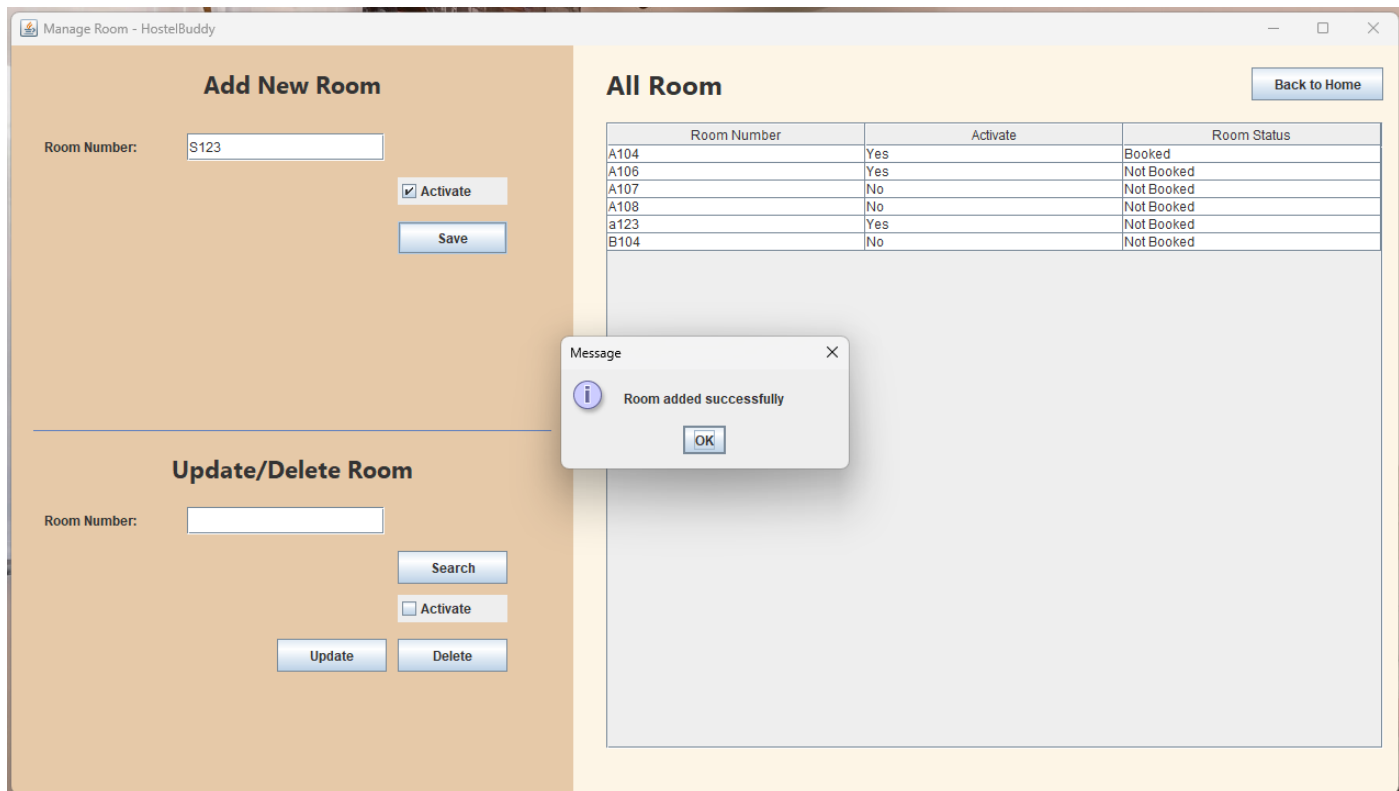
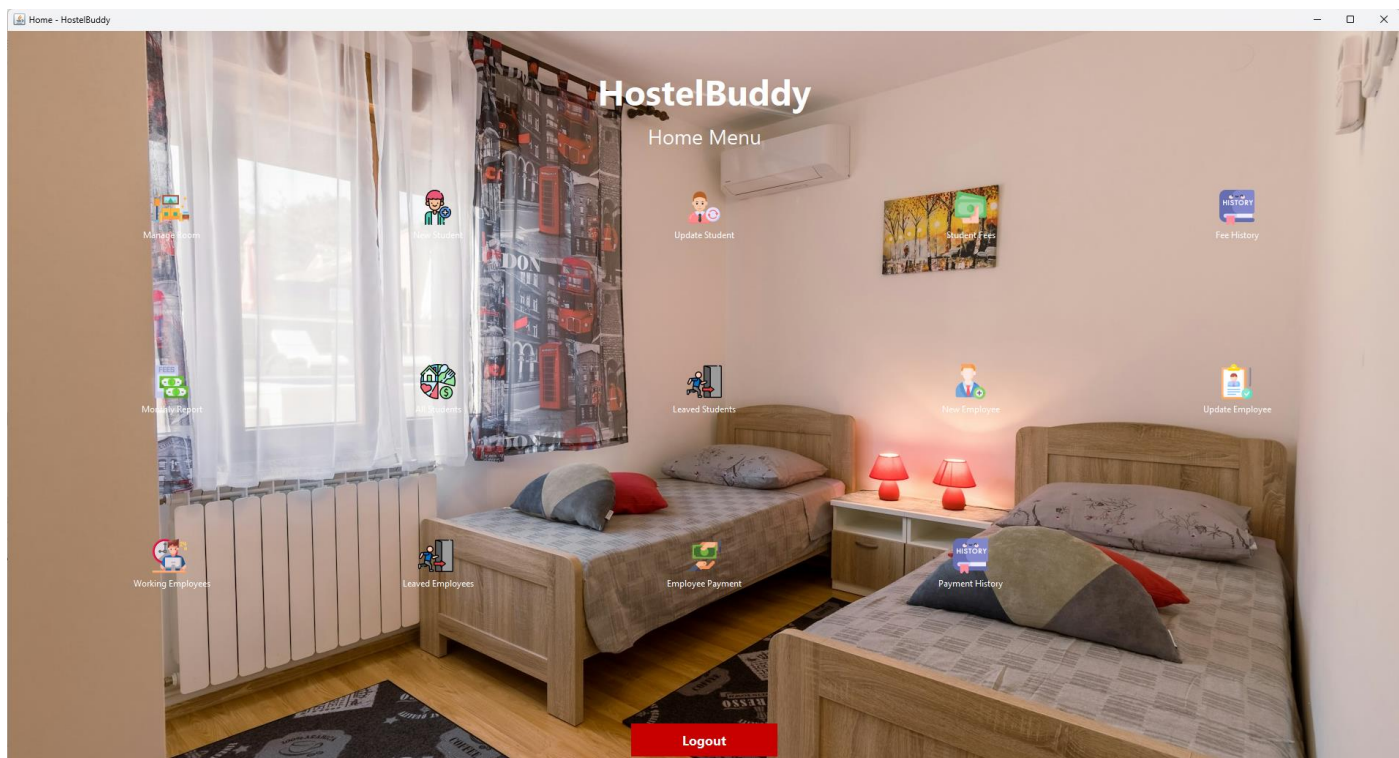
These checks help prevent data inconsistency and improve the user experience.

9.3 Sample Outputs

Below are examples of correct outputs generated by the system:

- **Successful login message**
- **New student record displayed in All Students table**
- **Fee history shown as monthly report**
- **Room status changing from 'Not Booked' to 'Booked' when capacity is reached**
- **CSV file exported with correct column headers and data rows**





Manage Room - HostelBuddy

Add New Room

Room Number:

☐ Activate

Save

Update/Delete Room

Room Number:

Search

☐ Activate

UpdateDelete

All Room

Back to Home

| Room Number | Activate | Room Status |
|-------------|----------|-------------|
| A104 | Yes | Booked |
| A106 | Yes | Not Booked |
| A107 | No | Not Booked |
| A108 | No | Not Booked |
| a123 | Yes | Not Booked |
| B104 | No | Not Booked |
| S123 | Yes | Not Booked |

MessageRoom updatedOK

New Student Registration

Back

Register New Student

Student ID:789456

Name:David Llewellyn Jones

Mobile Number:+917894561236

Email:David@Gmail.com

Father's Name:Jake Jones

Mother's Name:Sara Jones

Address:Delhi

College Name:XYZ Delhi

Aadhaar Number:789456789456

Room Number:S123

RegisterClear

Update/Delete Student

Back

Update/Delete Student

Enter Student ID:

789456

Search

Student ID / Roll No:

789456

Name:

David Llewellyn Jones

Mobile Number:

+917894561236

Email:

David@Gmail.com

Father's Name:

Jake Jones

Mother's Name:

Sara Jones

Address:

Delhi

College Name:

XYZ Delhi

Aadhaar Number:

789456789456

Room Number:

S123

Update

Mark as Leaved

Clear

Update/Delete Student

Back

Update/Delete Student

Enter Student ID:

789456

Search

Student ID / Roll No:

789456

Name:

David Llewellyn Jones

Mobile Number:

+917894561236

Email:

David@Gmail.com

Father's Name:

Mother's Name:

Address:

College Name:

XYZ Delhi

Aadhaar Number:

789456789456

Room Number:

S123

Update

Mark as Leaved

Clear

Message

Student marked as 'Leaved'. Room freed.

OK

Update/Delete Student

Back

Update/Delete Student

Enter Student ID:

789456

Search

Student ID / Roll No:

789456

Name:

David Llewellyn Jones

Mobile Number:

+917894561236

Email:

David@Gmail.com

Father's Name:

Ja

Mother's Name:

Sa

Address:

De...

College Name:

XYZ Delhi

Aadhaar Number:

789456789456

Room Number:

S123

Update

Mark as Leaved

Mark as Living

Clear

Message

Student reactivated successfully.

OK

Student Fee Management

Student Fee Payment

Back

Student ID:

Search

Name:

Email:

Room:

Month:

June

Amount:

Save Payment

Clear

Message

Payment saved successfully.

OK

Fee History Viewer

Fee History Viewer

Back

Student ID:

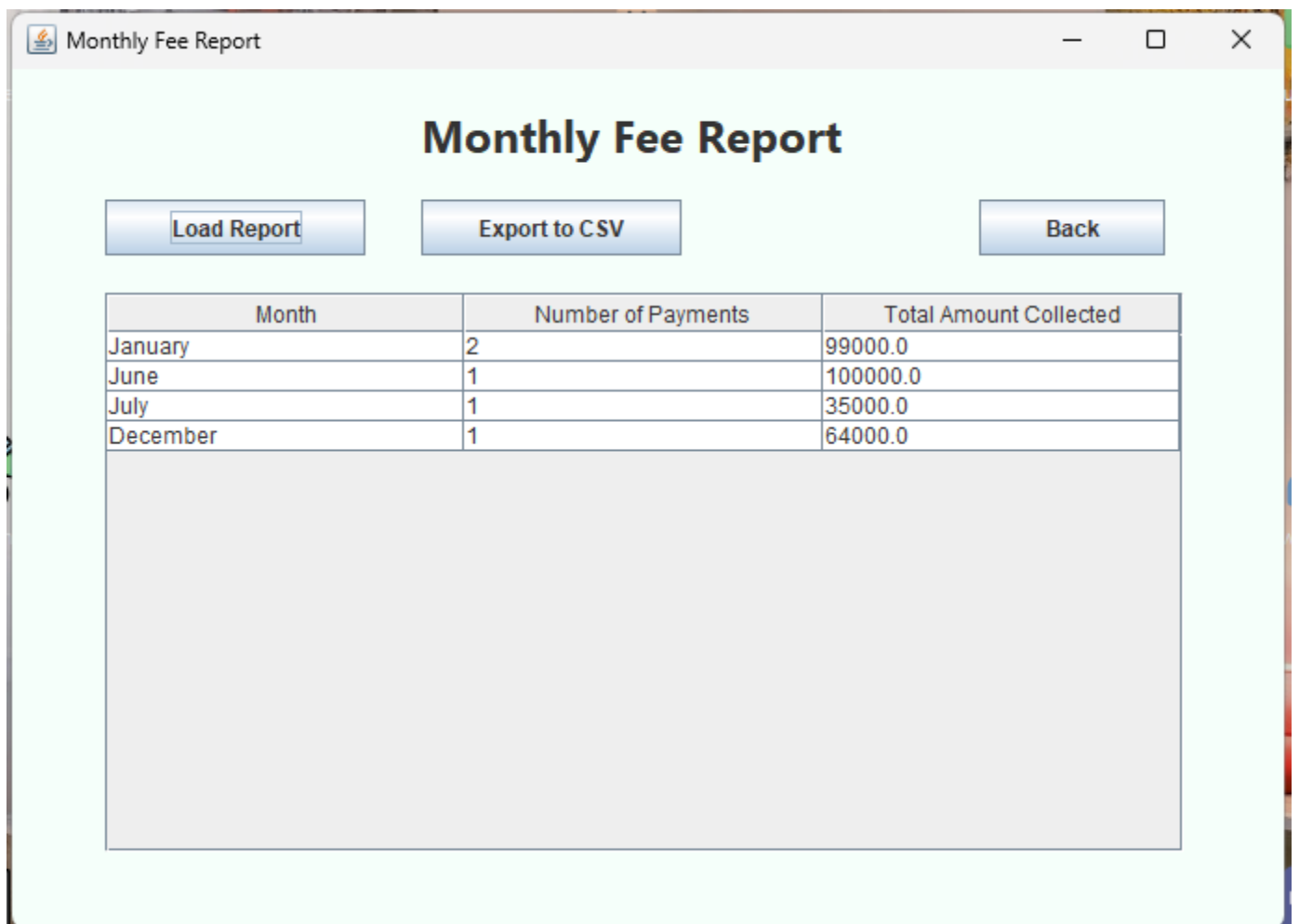
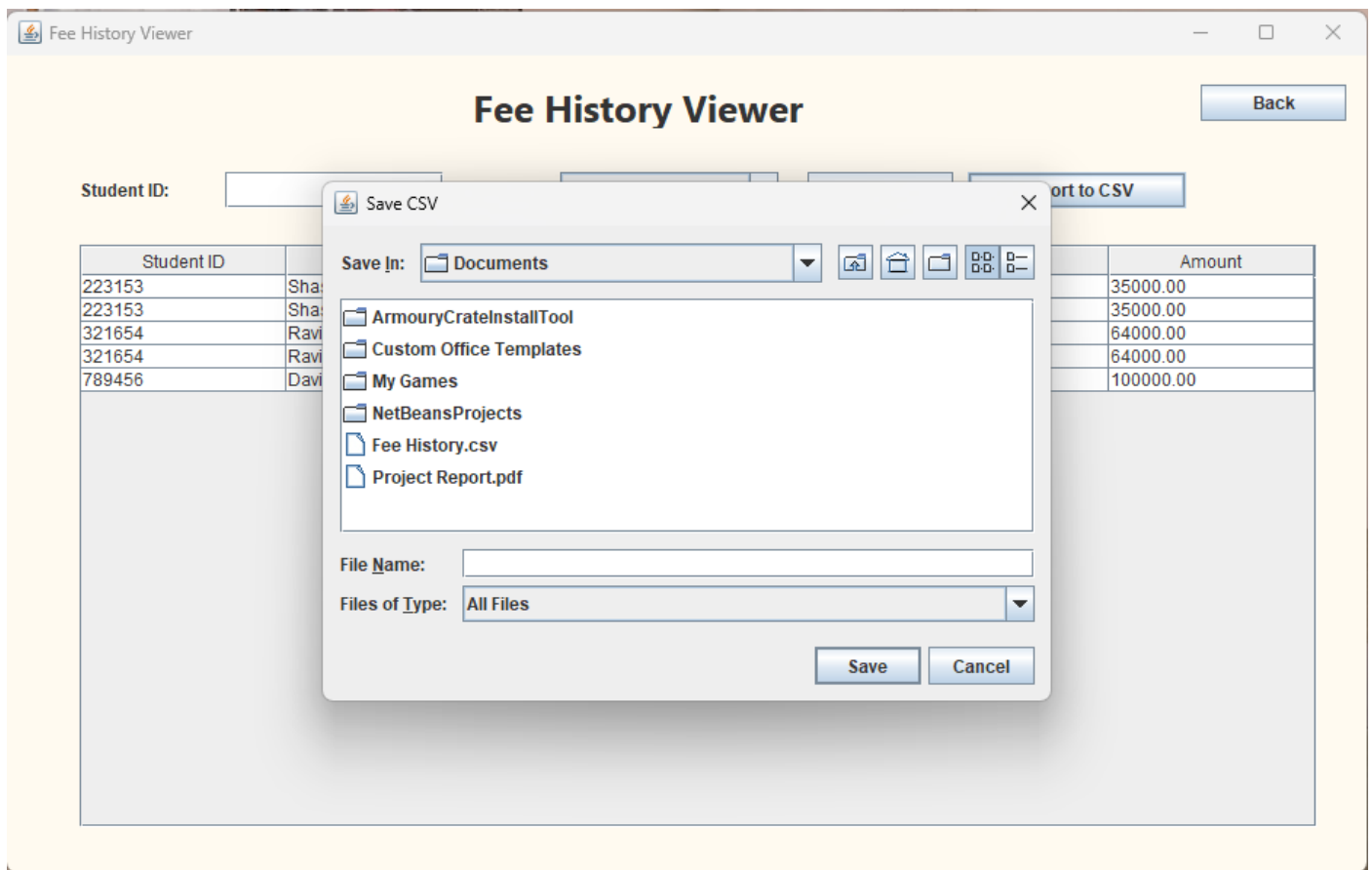
Month:

All

Search

Export to CSV

| Student ID | Name | Email | Room | Month | Amount |
|------------|-----------------------|------------------|------|----------|-----------|
| 223153 | Shashwat Singh Gaur | 223153@kit.ac.in | A107 | January | 35000.00 |
| 223153 | Shashwat Singh Gaur | 223153@kit.ac.in | A107 | July | 35000.00 |
| 321654 | Ravi Gupta | 321654@gmail.com | A104 | January | 64000.00 |
| 321654 | Ravi Gupta | 321654@gmail.com | A104 | December | 64000.00 |
| 789456 | David Llewellyn Jones | David@Gmail.com | S123 | June | 100000.00 |



All Students Living

All Students Currently Living

Back

Search (ID or Name):

Search

Reset

Export to CSV

| Student ID | Name | Mobile | Email | Father's Name | Mother's Name | Address | College | Aadhaar | Room |
|------------|-------------------|---------------|------------------|-------------------|---------------|---------|------------|----------------|------|
| 223153 | Shashwat Sin... | 9118113311 | 223153@kit.ac... | Anil Kumar Sin... | Rekha Singh | Kanpur | KIT Kanpur | 01112223334... | A106 |
| 321654 | Ravi Gupta | 987654321 | 321654@gmai... | Karan Gupta | Rani Gupta | Kanpur | KIT Kanpur | 11223344556... | A104 |
| 789456 | David Llewelly... | +917894561236 | David@Gmail... | Jake Jones | Sara Jones | Delhi | XYZ Delhi | 789456789456 | S123 |

Leaved Students


Students Who Have Left the Hostel

Back

Refresh

Export to CSV

| Student ID | Name | Email | Room | Left On (Aadhaar) | Mobile | Status |
|------------|---------------------|------------------|------|-------------------|------------|--------|
| 223153 | Shashwat Singh Gaur | 223153@kit.ac.in | A106 | 011122233344555 | 9118113311 | Leaved |

 New Employee Registration

Back

Register New Employee

Employee ID:

159753

Name:

Ramesh

Gender:

Male

Mobile Number:

+911425361425

Email:

Ramesh@Gmail.com

Aadhaar Number:

748596748596

Address:

Kanpur

Role:

Security

Joining Date:

2025-05-25

Register

Clear

Message



Employee registered successfully!

OK

Update/Delete Employee

Back

Update/Delete Employee

Enter Employee ID:

159753

Search

Employee ID:

159753

Name:

Ramesh

Gender:

Male

▼

Mobile Number:

+911425361425

Email:

Ramesh@Gmail.com

Aadhaar Number:

748596748596

Address:

Kanpur

Role:

Security

▼

Joining Date:

2025-05-25

Update

Mark as Leaved

Mark as Working

Clear

[Back](#)

[Export to CSV](#)

| Employee ID | Name | Gender | Mobile | Email | Aadhaar | Address | Role | Joining Date | Status |
|-------------|------------|--------|---------------|---------------|---------------|---------|----------|--------------|---------|
| 123456 | Sonu Varma | Male | 9876549876 | 123456@Gm... | 6655449988... | Kanpur | Warden | 2025-04-23 | Working |
| 159753 | Ramesh | Male | +911425361... | Ramesh@G... | 748596748596 | Kanpur | Security | 2025-05-25 | Working |
| 362514 | Jiya Singh | Female | 6543216543 | 362514@gmi... | 9638527419... | Kanpur | Cook | 2025-04-23 | Working |

Update/Delete Employee

Back

Update/Delete Employee

Enter Employee ID:

123456

Search

Employee ID:

123456

Name:

Sonu Varma

Gender:

Male

Mobile Number:

9876549876

Email:

123456@Gmail.com

Aadhaar Number:

66554499887733

Address:

Kanpur

Role:

Warden

Joining Date:

2025-04-23

Update

Mark as Leaved

Clear

Message

Employee marked as 'Leaved'.

OK

Leaved Employees

Employees Who Have Left

Back

Search (ID/Name):

Search

Refresh

Export to CSV

| Employee ID | Name | Gender | Mobile | Email | Aadhaar | Address | Role | Joining Date | Status |
|-------------|------------|--------|------------|--------------|---------------|---------|--------|--------------|--------|
| 123456 | Sonu Varma | Male | 9876549876 | 123456@Gm... | 6655449988... | Kanpur | Warden | 2025-04-23 | Leaved |

Employee Payment

Back

Employee Payment

Select Employee:

Ramesh (159753)

Month:

January

Year:

2025

Amount Paid (₹):

10000

Payment Date:

2025-05-25

Remarks (optional):

Salary

Submit Payment

Clear

Message

i

Payment recorded successfully!

OK

Employee Payment History

Back

Employee:

All

Month:

All

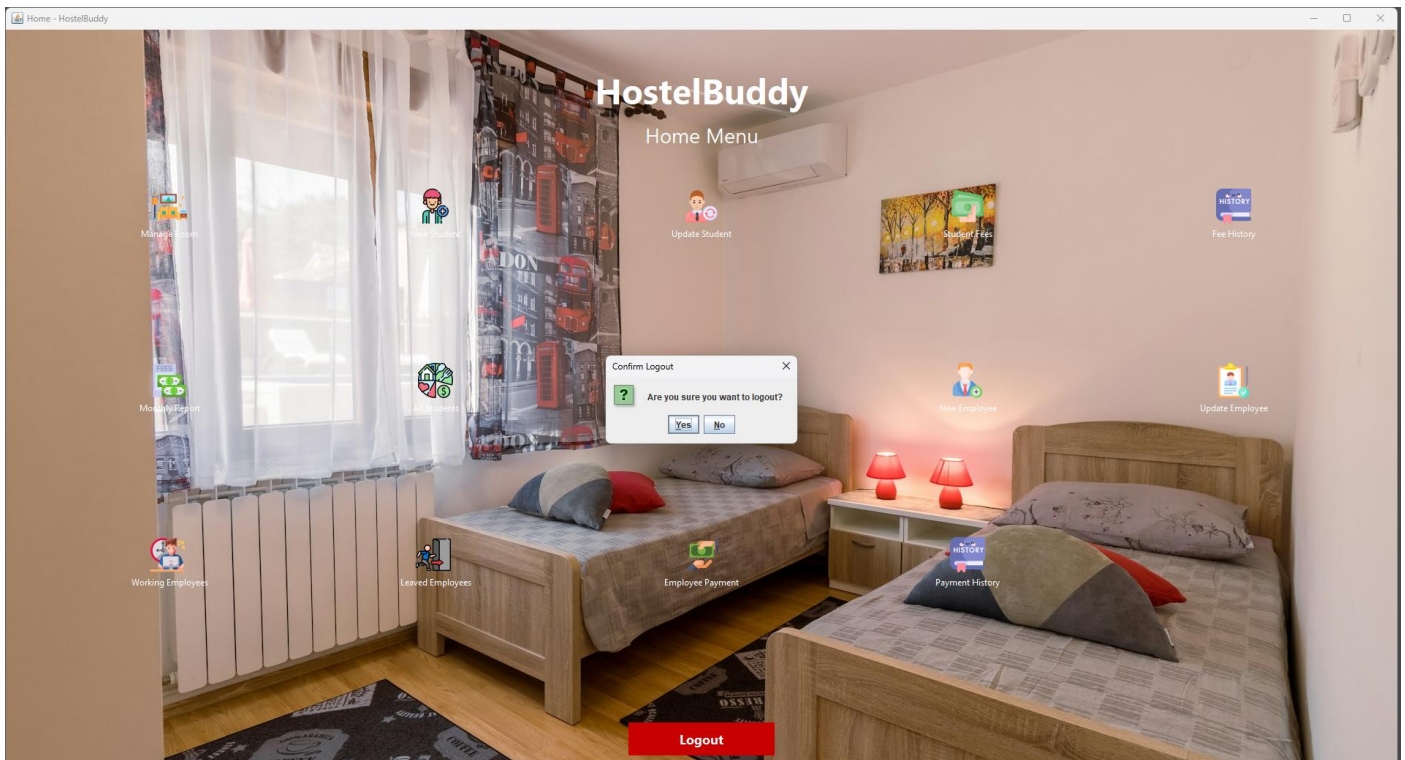
Year:

All

Search

Export to CSV

| ID | Employee ID | Name | Month | Year | Amount | Payment Date | Remarks |
|----|-------------|------------|----------|------|---------|--------------|--------------------|
| 1 | 123456 | Sonu Varma | January | 2025 | 30000.0 | 2025-04-23 | Salary Of January |
| 2 | 362514 | Jiya Singh | February | 2025 | 30000.0 | 2025-04-23 | Salary Of February |
| 3 | 159753 | Ramesh | January | 2025 | 10000.0 | 2025-05-25 | Salary |



9.4 Exception Handling

Exception handling is implemented to deal with unexpected inputs, database issues, or application errors.

Typical try-catch usage:

```
try {  
    Connection con = DBConnection.getConnection();  
    // SQL operations  
} catch (SQLException e) {  
    JOptionPane.showMessageDialog(null, "Database error: " + e.getMessage());  
}
```

Common exception types handled:

- Database connection failure
- SQL query syntax errors
- NullPointerException from missing inputs
- File writing errors during CSV export

Handled exceptions are displayed using Swing pop-ups to inform the user and help diagnose issues without crashing the application.

10. CONCLUSION AND FUTURE ENHANCEMENTS

This section summarizes the outcome of the HostelBuddy project, highlights key accomplishments, discusses challenges faced during development, and outlines future enhancements that can further improve the system.

10.1 Summary of Achievements

The HostelBuddy project successfully achieved its goal of providing a functional, reliable, and user-friendly hostel management system. Key achievements include:

- Developed a fully working **Java Swing-based desktop application**.
 - Designed a **modular structure** for handling students, rooms, employees, and fees.
 - Integrated **MySQL** backend using **JDBC** for seamless data storage and retrieval.
 - Implemented features for **student registration, room assignment, fee tracking, and employee salary management**.
 - Provided **CSV export** capabilities for generating offline reports.
 - Designed a modern **GUI with card-based dashboard**, responsive forms, and user interaction feedback.
-

10.2 Challenges Faced

During the development of HostelBuddy, several technical and design challenges were encountered:

- Ensuring smooth **JDBC connectivity** and handling exceptions related to database access.
- Validating form inputs in real-time to avoid runtime errors.
- Managing **dynamic room occupancy**, especially for scenarios with 2 students per room.
- Handling **UI alignment issues** across different screen resolutions.

- Structuring the application for **maintainability and scalability**.

All of these challenges were resolved through research, iterative testing, and by applying proper object-oriented design principles.

10.3 Scope for Improvement

While HostelBuddy is functionally complete, there are areas where future improvements can be made:

- Add support for **multi-user login with role-based access** (admin, warden, accountant).
 - Use **advanced form validation libraries** for stronger input handling.
 - Store user credentials securely using **hashed passwords**.
 - Improve performance on large datasets with **pagination and indexing**.
 - Provide support for **partial payments** or installment-based fee tracking.
-

10.4 Potential Future Features

Some future enhancements that can be implemented to evolve HostelBuddy further:

- **Mobile App Integration:** Develop a mobile companion app for students to view their room details, fee status, and notifications.
- **Biometric Authentication:** Add fingerprint or face ID login for additional security.
- **Online Payment Gateway:** Integrate UPI or card payment systems to allow fee payment directly from the software.
- **Cloud Database:** Migrate the backend to a cloud database for remote access and real-time synchronization.
- **Attendance Tracking:** Record daily in/out movement of students and employees.

These enhancements will not only increase system functionality but also improve its usability and real-world applicability in large-scale hostels or multi-campus environments.

11 References

- Oracle Java Documentation – <https://docs.oracle.com/javase/>
- MySQL Documentation – <https://dev.mysql.com/doc/>
- Java Swing Tutorials – <https://docs.oracle.com/javase/tutorial/uiswing/>
- JDBC API Guide – <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>
- NetBeans IDE Official Docs – <https://netbeans.apache.org/kb/>
- Stack Overflow – For issue resolution and logic suggestions
- iText PDF Library (optional for export features) – <https://itextpdf.com/>