



**M.Kumarasamy
College of Engineering**

NAAC Accredited Autonomous Institution

Approved by AICTE & Affiliated to Anna University

ISO 9001:2015 Certified Institution

Thalavapalayam, Karur - 639 113, TAMILNADU.



A Project Report

on

HOUSE RENTAL SYSTEM

Submitted in partial fulfillment of requirements for the award of the course

of

CGB1201 – JAVA PROGRAMMING

Under the guidance of

Mrs. I.Karthika M.E.,

Assistant Professor / CSE

Submitted By

INDHUJA S R (927622BCS036)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

M.KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous)

KARUR – 639 113

DECEMBER 2024



M. KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to Anna University, Chennai)

KARUR – 639 113

BONAFIDE CERTIFICATE

Certified that this project report on “ **HOUSE RENTAL SYSTEM**” is the bonafide work of **INDHUJA S R (927622BCS036)** who carried out the project work during the academic year 2024 - 2025 under my supervision.

Signature

Mrs. I. KARTHIKA M.E.,

SUPERVISOR,

Department of Computer Science and
Engineering,

M.Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

Signature

Dr. D. PRADEEP M.E.,Ph.D.,

HEAD OF THE DEPARTMENT,

Department of Electronics and Communication
Engineering,

M.Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

Submitted for the End semester practical Examination held on 05.12.2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education

MISSION OF THE INSTITUTION

- Produce smart technocrats with empirical knowledge who can surmount the global challenges
- Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students
- Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations

VISION OF THE DEPARTMENT

To achieve education and research excellence in Computer Science and Engineering

MISSION OF THE DEPARTMENT

- To excel in academic through effective teaching learning techniques
- To promote research in the area of computer science and engineering with the focus on innovation
- To transform students into technically competent professionals with societal and ethical responsibilities

PROGRAM EDUCATIONAL OBJECTIVES (PEOS)

PEO 1: Graduates will have successful career in software industries and R&D divisions through continuous learning.

PEO 2: Graduates will provide effective solutions for real world problems in the key domain of computer science and engineering and engage in lifelong learning.

PEO 3: Graduates will excel in their profession by being ethically and socially responsible.

PROGRAM OUTCOMES

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.



PROGRAM SPECIFIC OUTCOMES (PSOs)

- **PSO1: Professional Skills:** Ability to apply the knowledge of computing techniques to design and develop computerized solutions for the problems.
- **PSO2: Successful career:** Ability to utilize the computing skills and ethical values in creating a successful career.

ABSTRACT

The House Rental System is a web – based application which is designed to facilitate the rental of properties between landlords and tenants. This system allows landlords to list their properties with details such as rent, location, while tenants can search for available rentals based on these criteria. The application is the user friendly interface where the tenants can search for the location and rent and view property information. Tenants can also request to book a property by submitting a rental request to the landlord. The system includes an admin panel that allows administrators to manage user accounts, including creating and deleting landlord and tenant accounts. Additionally, administrators can create, update, and delete rental listings, as well as track and manage rental requests.

ABSTRACT WITH POs AND PSOs MAPPING

ABSTRACT	POs MAPPED	PSOs MAPPED
<p>The House Rental System is a web – based application which is designed to facilitate the rental of properties between landlords and tenants. This system allows landlords to list their properties with details such as rent, location, while tenants can search for available rentals based on these criteria. The application is the user friendly interface where the tenants can search for the location and rent and view property information. Tenants can also request to book a property by submitting a rental request to the landlord. The system includes an admin panel that allows administrators to manage user accounts, including creating and deleting landlord and tenant accounts. Additionally, administrators can create, update, and delete rental listings, as well as track and manage rental requests.</p>	<p>PO 1(3) PO 2(3) PO 3(3) PO 4(3) PO 5(3) PO 6(3) PO 7(3) PO 8(3) PO 9(3) PO 10(2) PO 11(2) PO 12(3)</p>	<p>PSO 1(3) PSO 2(3)</p>

Note: 1- Low, 2-Medium, 3- High

SUPERVISOR

HEAD OF THE DEPARTMENT

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	ABSTRACT	
1	INTRODUCTION	1
	1.1 Objective	1
	1.2 Overview	1
	1.3 Java Programming concepts	2
2	PROJECT METHODOLOGY	2
	2.1 Proposed Work	2
	2.2 Block Diagram	2
3	MODULE DESCRIPTION	3
	3.1 User Management Module	3
	3.2 Property Listing Module	3
	3.3 Search and Filter Module	3
	3.4 Booking and Request Module	3
	3.5 Admin Panel Module	4
4	RESULTS AND DISCUSSION	5
5	CONCLUSION	10
	REFERENCES	11
	APPENDIX	12

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of this project is to design and implement a House Rental System, a Java-based application that facilitates property rental transactions between landlords and tenants. The system provides a platform where:

- Landlords can list properties with details such as rent, location, and availability.
- Tenants can search for properties based on criteria and send booking requests.
- Administrators manage user accounts, rental listings, and track rental requests.

1.2 Overview

The House Rental System is a user-friendly application designed for efficiency and simplicity. This application includes:

- A landlord interface for managing properties and viewing tenant booking requests.
- A tenant interface for searching properties and requesting bookings.
- An admin panel for system-wide management of users and properties.

The system employs Java Swing and AWT for graphical interfaces and adheres to modular programming principles.

1.3 Java Programming Concepts

This project utilizes the following Java concepts:

- Object-Oriented Programming (OOP): Classes, objects, inheritance, and polymorphism to model landlords, tenants, properties, and admins.
- Java Swing and AWT: Building graphical interfaces.
- File Handling/Database Connectivity: For storing and retrieving user and property information.
- Event Handling: Responding to user actions on the UI.
- Collections Framework: Managing lists of properties and users.

CHAPTER 2

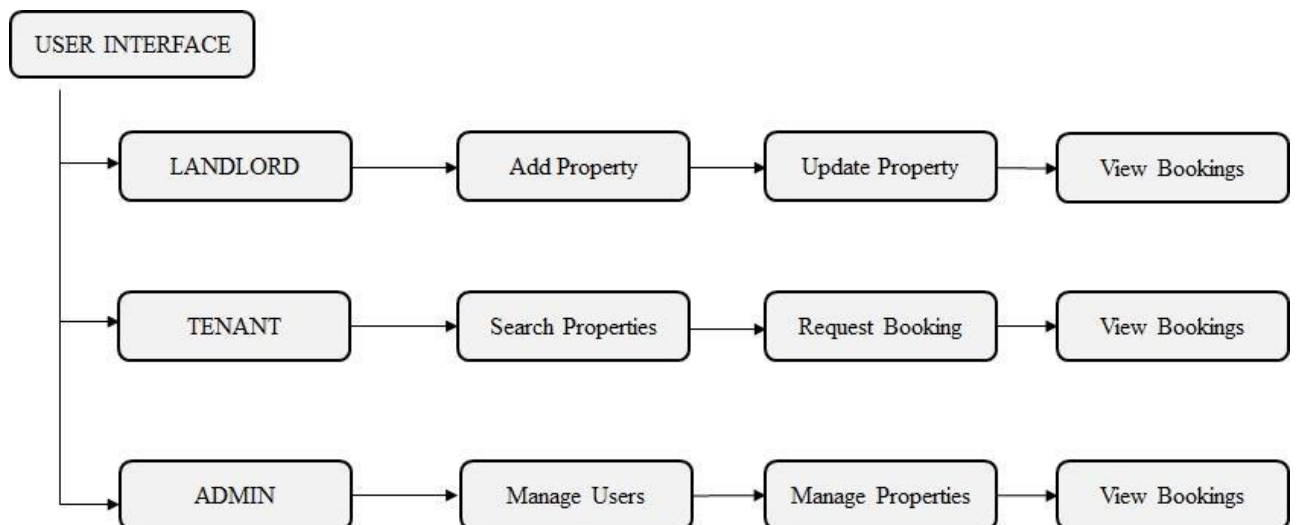
PROJECT METHODOLOGY

2.1 Proposed Work

The project proposes to create a modular Java-based system using the following approach:

- User Management Module: Secure login for landlords, tenants, and admins.
- Property Listing Module: Enable landlords to manage property details.
- Search and Filter Module: Provide tenants with an intuitive search interface.
- Booking and Request Module: Facilitate communication between tenants and landlords for booking requests.
- Admin Panel Module: Enable administrators to manage users, properties, and requests.

2.2 Block Diagram





CHAPTER 3

MODULE DESCRIPTION

3.1 User Management Module

- **Login and Registration:**
Separate login pages for landlords, tenants, and admins using Java Swing. Authentication ensures secure access to specific functionalities.
- **User Role Management:**
Admins can create, update, and delete accounts for landlords and tenants.

3.2 Property Listing Module

- **Add Property:**
Landlords can list properties with details like location, rent, and availability status.
- **Update/Delete Property:**
Landlords can modify or remove their listings as required.
- **View Properties:**
A feature for landlords to review their listed properties.

3.3 Search and Filter Module

- **Search Properties:**
Tenants can search for properties based on location, rent, or availability.
- **Filter Options:**
Advanced filters for narrowing down search results.
- **Property Details View:**
Displays property details, including photos and descriptions.

3.4 Booking and Request Module

- **Request Booking:**
Tenants can send booking requests directly to landlords.



- **Manage Requests:**
Landlords can accept or reject booking requests.
- **Request Status Tracking:**
Tenants can view the status of their rental requests.

3.5 Admin Panel Module

- **User Management:**
Admins can create, modify, or delete landlord and tenant accounts.
- **Property Management:**
Admins can manage all property listings across the system.
- **Booking Overview:**
Admins can track and review all rental requests and their statuses.

CHAPTER 4

RESULTS AND DISCUSSION

Registration Page

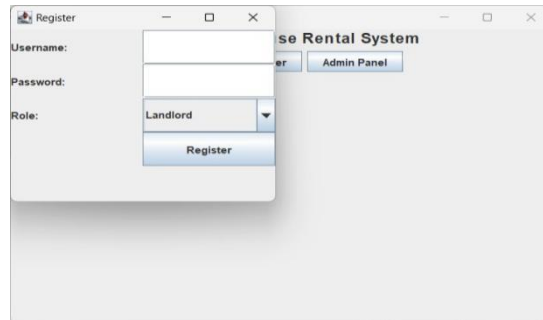


Fig.4.1 Registration Page

The registration form in the House Rental System is a critical component that enables users to create accounts and access role-specific functionalities. It features input fields for a username, password, and role selection (Landlord, Tenant, or Admin) through a dropdown menu, ensuring role-based access control. Implemented using Java Swing, the form validates inputs for security, such as checking username uniqueness and enforcing password rules. This design ensures a smooth user experience while maintaining system integrity by directing users to their specific roles after registration. Enhancements like error messages for invalid inputs or a password strength indicator could further improve usability and security.

Login Page

The login page in the House Rental System is designed to authenticate users securely and provide access to role-specific functionalities. It includes fields for a username and password, along with a login button, implemented using Java Swing components like `JTextField` and `JPasswordField`. The system validates user credentials against stored data to ensure secure access. Role-based redirection ensures that landlords, tenants, and admins are directed to their respective dashboards upon successful login. This simple and intuitive interface promotes ease of use while maintaining security. Enhancements like password recovery options or login attempt limits could further improve the system.

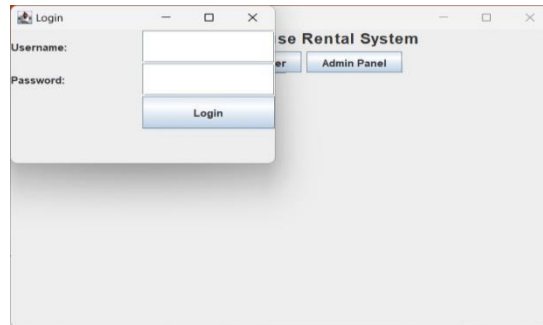


Fig.4.2 Login Page

Booking Requests

The booking request functionality allows tenants to send requests to landlords for properties they wish to rent. Landlords can then view these requests on their dashboard, which includes tenant details and request status. Landlords can either accept or reject requests using action buttons, providing real-time feedback to tenants. This streamlined process ensures effective communication between landlords and tenants, fostering a transparent rental workflow.

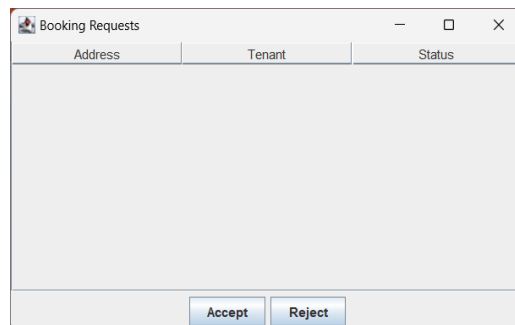
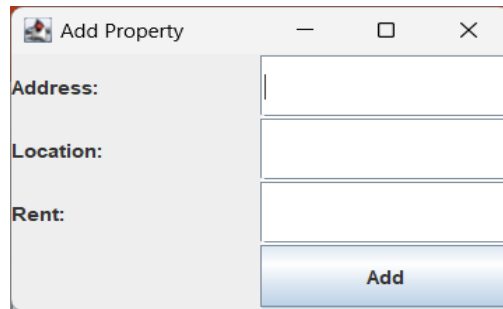


Fig.4.3 Booking Requests

Add Property

The **Add Property** feature is accessible to landlords, enabling them to list new properties by providing details such as location, rent, and availability. The input form is straightforward, utilizing Java Swing components like `JTextField` for text input and dropdowns for selection options. Once submitted, the property details are saved into the database or a file. This feature simplifies property management for landlords, ensuring their listings are quickly made available to tenants.

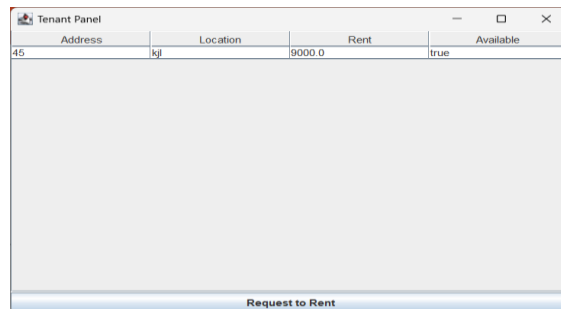


A dialog box titled "Add Property" with three input fields labeled "Address:", "Location:", and "Rent:". Below the fields is a blue button labeled "Add".

Fig.4.4 Add Property

Tenant Property

The tenant page in the House Rental System allows tenants to search for available properties based on criteria like location and rent. Using a simple and intuitive interface built with Java Swing, tenants can view property details and send booking requests to landlords. It also includes a section for tenants to track the status of their submitted booking requests (e.g., pending, accepted, or rejected). This feature ensures seamless interaction and transparency in the rental process, enhancing the user experience.



A window titled "Tenant Panel" displaying a table with property listings. The table has columns: Address, Location, Rent, and Available. The first row shows a property with Address "45", Location "kjl", Rent "9000.0", and Available "true". Below the table is a button labeled "Request to Rent".

Address	Location	Rent	Available
45	kjl	9000.0	true

Fig.4.5 Tenant Property

Admin Panel

The admin panel serves as the control center for managing the House Rental System. Admins can oversee all user accounts, property listings, and booking requests. Functionalities include creating or deleting user accounts, updating property details, and tracking booking statuses. Designed with a simple interface, the admin panel ensures the smooth operation of the system and provides centralized control to maintain data consistency and system integrity.



Address	Location	Rent	Landlord	Available
45	kjl	9000.0	yazh	true

Fig.4.6 Admin Panel



CHAPTER 5

CONCLUSION

In conclusion, the House Rental System successfully provides a streamlined and user-friendly platform for managing rental properties. With distinct functionalities for landlords, tenants, and admins, the system ensures efficient property listing, booking, and user account management. Features like role-based access, booking requests, and the admin panel promote transparency and organization, addressing the needs of all users. Implemented using Java Swing and AWT, the application demonstrates a robust design, scalability, and practicality. This project lays a strong foundation for future enhancements, such as integrating payment systems and advanced analytics, to further improve the rental experience.



REFERENCES:

- Schildt, H. (2022). *Java: The Complete Reference*. McGraw Hill.
- Deitel, P. J., & Deitel, H. M. (2019). *Java How to Program: Early Objects*. Pearson.
- Sharma, V. (2018). *Development of Rental Management Systems Using Java*. *International Journal of Advanced Computer Science*.



APPENDIX

(Coding)

```
package house_rental_system;
```

```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.table.DefaultTableModel;  
import java.util.ArrayList;  
import java.util.List;
```

```
// User class to store user details
```

```
class User {  
    String username;  
    String password;  
    String role; // Landlord or Tenant
```

```
    User(String username, String password, String role) {  
        this.username = username;  
        this.password = password;  
        this.role = role;
```

```
    }  
}
```

```
// Property class to store rental details
```

```
class Property {  
    String address;  
    String location;  
    double rent;  
    String landlord; // Landlord username  
    boolean isAvailable;
```

```
    Property(String address, String location, double rent, String landlord) {  
        this.address = address;  
        this.location = location;  
        this.rent = rent;  
        this.landlord = landlord;  
        this.isAvailable = true;  
    }
```

```
@Override
```

```
public String toString() {  
    return "Address: " + address + ", Location: " + location + ", Rent: $" + rent + ", Available: " +  
    isAvailable;  
}
```



```
// Booking request class
class BookingRequest {
    Property property;
    String tenant; // Tenant username
    String status; // Pending, Approved, Rejected

    BookingRequest(Property property, String tenant) {
        this.property = property;
        this.tenant = tenant;
        this.status = "Pending";
    }

    @Override
    public String toString() {
        return "Property: " + property.address + ", Tenant: " + tenant + ", Status: " + status;
    }
}

public class HouseRentalSystem {
    private static List<User> users = new ArrayList<>();
    private static List<Property> properties = new ArrayList<>();
    private static List<BookingRequest> bookingRequests = new ArrayList<>();

    private static JFrame mainFrame;

    public static void main(String[] args) {
        SwingUtilities.invokeLater(HouseRentalSystem::createAndShowGUI);
    }

    private static void createAndShowGUI() {
        mainFrame = new JFrame("House Rental System");
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setSize(600, 400);
        mainFrame.setLayout(new BorderLayout());

        JLabel welcomeLabel = new JLabel("Welcome to House Rental System", JLabel.CENTER);
        welcomeLabel.setFont(new Font("Arial", Font.BOLD, 18));
        mainFrame.add(welcomeLabel, BorderLayout.NORTH);

        JPanel buttonPanel = new JPanel();
        JButton loginButton = new JButton("Login");
        JButton registerButton = new JButton("Register");
        JButton adminButton = new JButton("Admin Panel");

        loginButton.addActionListener(e -> showLoginPage());
        registerButton.addActionListener(e -> showRegisterPage());
        adminButton.addActionListener(e -> showAdminPanel());
    }
}
```



```

        buttonPanel.add(loginButton);
        buttonPanel.add(registerButton);
        buttonPanel.add(adminButton);

        mainFrame.add(buttonPanel, BorderLayout.CENTER);
        mainFrame.setVisible(true);
    }

    private static void showRegisterPage() {
        JFrame registerFrame = new JFrame("Register");
        registerFrame.setSize(300, 200);
        registerFrame.setLayout(new GridLayout(5, 2));

        JLabel userLabel = new JLabel("Username:");
        JTextField userField = new JTextField();
        JLabel passLabel = new JLabel("Password:");
        JPasswordField passField = new JPasswordField();
        JLabel roleLabel = new JLabel("Role:");
        JComboBox<String> roleComboBox = new JComboBox<>(new String[] { "Landlord",
        "Tenant" });
        JButton registerButton = new JButton("Register");

        registerButton.addActionListener(e -> {
            String username = userField.getText();
            String password = new String(passField.getPassword());
            String role = (String) roleComboBox.getSelectedItem();

            if (username.isEmpty() || password.isEmpty()) {
                JOptionPane.showMessageDialog(registerFrame, "All fields are required!");
            } else {
                users.add(new User(username, password, role));
                JOptionPane.showMessageDialog(registerFrame, "Registration successful!");
                registerFrame.dispose();
            }
        });

        registerFrame.add(userLabel);
        registerFrame.add(userField);
        registerFrame.add(passLabel);
        registerFrame.add(passField);
        registerFrame.add(roleLabel);
        registerFrame.add(roleComboBox);
        registerFrame.add(new JLabel(""));
        registerFrame.add(registerButton);

        registerFrame.setVisible(true);
    }

```



```
private static void showLoginPage() {
    JFrame loginFrame = new JFrame("Login");
    loginFrame.setSize(300, 200);
    loginFrame.setLayout(new GridLayout(4, 2));

    JLabel userLabel = new JLabel("Username:");
    JTextField userField = new JTextField();
    JLabel passLabel = new JLabel("Password:");
    JPasswordField passField = new JPasswordField();
    JButton loginButton = new JButton("Login");

    loginButton.addActionListener(e -> {
        String username = userField.getText();
        String password = new String(passField.getPassword());
        boolean isValid = false;

        for (User user : users) {
            if (user.username.equals(username) && user.password.equals(password)) {
                isValid = true;
                JOptionPane.showMessageDialog(loginFrame, "Welcome, " + user.role + "!");
                if (user.role.equals("Landlord")) {
                    showLandlordPanel(username);
                } else if (user.role.equals("Tenant")) {
                    showTenantPanel(username);
                }
                loginFrame.dispose();
                break;
            }
        }
        if (!isValid) {
            JOptionPane.showMessageDialog(loginFrame, "Invalid username or password!");
        }
    });

    loginFrame.add(userLabel);
    loginFrame.add(userField);
    loginFrame.add(passLabel);
    loginFrame.add(passField);
    loginFrame.add(new JLabel(""));
    loginFrame.add(loginButton);

    loginFrame.setVisible(true);
}

private static void showLandlordPanel(String username) {
    JFrame landlordFrame = new JFrame("Landlord Panel");
    landlordFrame.setSize(600, 400);
    landlordFrame.setLayout(new BorderLayout());
}
```



```
// Property Table
String[] columnNames = {"Address", "Location", "Rent", "Available"};
DefaultTableModel tableModel = new DefaultTableModel(columnNames, 0);
JTable propertyTable = new JTable(tableModel);
JScrollPane scrollPane = new JScrollPane(propertyTable);

// Refresh Button
JButton refreshButton = new JButton("Refresh");
refreshButton.addActionListener(e -> refreshPropertyTable(tableModel, username));

// Buttons
JButton addPropertyButton = new JButton("Add Property");
JButton handleRequestsButton = new JButton("Handle Requests");

addPropertyButton.addActionListener(e -> showAddPropertyDialog(username));
handleRequestsButton.addActionListener(e -> showBookingRequests(username));

JPanel buttonPanel = new JPanel();
buttonPanel.add(addPropertyButton);
buttonPanel.add(handleRequestsButton);
buttonPanel.add(refreshButton);

landlordFrame.add(scrollPane, BorderLayout.CENTER);
landlordFrame.add(buttonPanel, BorderLayout.SOUTH);
landlordFrame.setVisible(true);
refreshPropertyTable(tableModel, username); // Initial load
}

private static void refreshPropertyTable(DefaultTableModel tableModel, String username) {
    tableModel.setRowCount(0); // Clear existing rows
    for (Property property : properties) {
        if (property.landlord.equals(username)) {
            tableModel.addRow(new Object[]{property.address, property.location, property.rent,
property.isAvailable});
        }
    }
}

private static void showAddPropertyDialog(String username) {
    JFrame addPropertyFrame = new JFrame("Add Property");
    addPropertyFrame.setSize(300, 200);
    addPropertyFrame.setLayout(new GridLayout(4, 2));

    JLabel addressLabel = new JLabel("Address:");
    JTextField addressField = new JTextField();
    JLabel locationLabel = new JLabel("Location:");
    JTextField locationField = new JTextField();
    JLabel rentLabel = new JLabel("Rent:");
    JTextField rentField = new JTextField();
}
```



```

JButton addButton = new JButton("Add");

addButton.addActionListener(e -> {
    String address = addressField.getText();
    String location = locationField.getText();
    try {
        double rent = Double.parseDouble(rentField.getText());
        properties.add(new Property(address, location, rent, username));
        JOptionPane.showMessageDialog(addPropertyFrame, "Property added successfully!");
        addPropertyFrame.dispose();
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(addPropertyFrame, "Invalid rent amount!");
    }
});

addPropertyFrame.add(addressLabel);
addPropertyFrame.add(addressField);
addPropertyFrame.add(locationLabel);
addPropertyFrame.add(locationField);
addPropertyFrame.add(rentLabel);
addPropertyFrame.add(rentField);
addPropertyFrame.add(new JLabel(""));
addPropertyFrame.add(addButton);

addPropertyFrame.setVisible(true);
}

private static void showBookingRequests(String landlord) {
    JFrame requestsFrame = new JFrame("Booking Requests");
    requestsFrame.setSize(300, 200);

    DefaultTableModel tableModel
    = new DefaultTableModel(new String[]{"Address", "Tenant", "Status"}, 0);
    JTable requestsTable = new JTable(tableModel);

    for (BookingRequest request : bookingRequests) {
        if (request.property.landlord.equals(landlord)) {
            tableModel.addRow(new Object[]{request.property.address, request.tenant,
request.status});
        }
    }

    JButton acceptButton = new JButton("Accept");
    JButton rejectButton = new JButton("Reject");

    acceptButton.addActionListener(e -> {
        int selectedRow = requestsTable.getSelectedRow();
        if (selectedRow >= 0) {
            BookingRequest selectedRequest = bookingRequests.get(selectedRow);

```




```

        selectedRequest.status = "Approved";
        selectedRequest.property.isAvailable = false;
        tableModel.setValueAt("Approved", selectedRow, 2);
        JOptionPane.showMessageDialog(requestsFrame, "Request approved!");
    }
});

rejectButton.addActionListener(e -> {
    int selectedRow = requestsTable.getSelectedRow();
    if (selectedRow >= 0) {
        BookingRequest selectedRequest = bookingRequests.get(selectedRow);
        selectedRequest.status = "Rejected";
        tableModel.setValueAt("Rejected", selectedRow, 2);
        JOptionPane.showMessageDialog(requestsFrame, "Request rejected!");
    }
});

JPanel buttonPanel = new JPanel();
buttonPanel.add(acceptButton);
buttonPanel.add(rejectButton);

requestsFrame.add(new JScrollPane(requestsTable), BorderLayout.CENTER);
requestsFrame.add(buttonPanel, BorderLayout.SOUTH);
requestsFrame.setVisible(true);
}

private static void showTenantPanel(String username) {
    JFrame tenantFrame = new JFrame("Tenant Panel");
    tenantFrame.setSize(600, 400);

    DefaultTableModel tableModel = new DefaultTableModel(new String[] { "Address",
"Location", "Rent", "Available" }, 0);
    JTable propertyTable = new JTable(tableModel);

    for (Property property : properties) {
        if (property.isAvailable) {
            tableModel.addRow(new Object[] { property.address, property.location, property.rent,
property.isAvailable });
        }
    }

    JButton bookButton = new JButton("Request to Rent");
    JButton viewRequestsButton = new JButton("View Booking Requests");

    bookButton.addActionListener(e -> {
        int selectedRow = propertyTable.getSelectedRow();
        if (selectedRow >= 0) {
            Property selectedProperty = properties.get(selectedRow);
            bookingRequests.add(new BookingRequest(selectedProperty, username));
        }
    });
}

```



```

        JOptionPane.showMessageDialog(tenantFrame, "Booking request sent!");
    }
});

viewRequestsButton.addActionListener(e -> showTenantRequests(username));

tenantFrame.add(new JScrollPane(propertyTable), BorderLayout.CENTER);
JPanel buttonPanel = new JPanel();
buttonPanel.add(bookButton);
buttonPanel.add(viewRequestsButton);
tenantFrame.add(buttonPanel, BorderLayout.SOUTH);
tenantFrame.setVisible(true);
}

private static void showTenantRequests(String tenant) {
    JFrame requestsFrame = new JFrame("My Booking Requests");
    requestsFrame.setSize(300, 200);

    DefaultTableModel tableModel = new DefaultTableModel(new String[] { "Property",
"Status" }, 0);
    JTable requestsTable = new JTable(tableModel);

    for (BookingRequest request : bookingRequests) {
        if (request.tenant.equals(tenant)) {
            tableModel.addRow(new Object[] { request.property.address, request.status });
        }
    }

    requestsFrame.add(new JScrollPane(requestsTable), BorderLayout.CENTER);
    requestsFrame.setVisible(true);
}

private static void showAdminPanel() {
    JFrame adminFrame = new JFrame("Admin Panel");
    adminFrame.setSize(600, 400);

    DefaultTableModel propertyTableModel = new DefaultTableModel(new String[] { "Address",
"Location", "Rent", "Landlord", "Available" }, 0);
    JTable propertyTable = new JTable(propertyTableModel);

    for (Property property : properties) {
        propertyTableModel.addRow(new Object[] { property.address, property.location,
property.rent, property.landlord, property.isAvailable });
    }

    JButton deleteButton = new JButton("Delete Property");
    JButton updateButton = new JButton("Update Property");

    deleteButton.addActionListener(e -> {

```



```

    int selectedRow = propertyTable.getSelectedRow();
    if (selectedRow >= 0) {
        properties.remove(selectedRow);
        propertyTableModel.removeRow(selectedRow);
        JOptionPane.showMessageDialog(adminFrame, "Property deleted successfully!");
    }
});

updateButton.addActionListener(e -> {
    int selectedRow = propertyTable.getSelectedRow();
    if (selectedRow >= 0) {
        Property selectedProperty = properties.get(selectedRow);
        showUpdatePropertyDialog(selectedProperty, propertyTableModel, selectedRow);
    }
});

JPanel buttonPanel = new JPanel();
buttonPanel.add(deleteButton);
buttonPanel.add(updateButton);

adminFrame.add(new JScrollPane(propertyTable), BorderLayout.CENTER);
adminFrame.add(buttonPanel, BorderLayout.SOUTH);
adminFrame.setVisible(true);
}

```

```

private static void showUpdatePropertyDialog(Property property, DefaultTableModel
tableModel, int rowIndex) {

```

```

    JFrame updatePropertyFrame = new JFrame("Update Property");
    updatePropertyFrame.setSize(300, 200);
    updatePropertyFrame.setLayout(new GridLayout(4, 2));

    JLabel addressLabel = new JLabel("Address:");
    JTextField addressField = new JTextField(property.address);
    JLabel locationLabel = new JLabel("Location:");
    JTextField locationField = new JTextField(property.location);
    JLabel rentLabel = new JLabel("Rent:");
    JTextField rentField = new JTextField(String.valueOf(property.rent));
    JButton updateButton = new JButton("Update");

```

```

    updateButton.addActionListener(e -> {
        String address = addressField.getText();
        String location = locationField.getText();
        try {
            double rent = Double.parseDouble(rentField.getText());
            property.address = address;
            property.location = location;
            property.rent = rent;
            tableModel.setValueAt(address, rowIndex, 0);
            tableModel.setValueAt(location, rowIndex, 1);

```



```
        tableModel.setValueAt(rent, rowIndex, 2);
        JOptionPane.showMessageDialog(updatePropertyFrame, "Property updated
successfully!");
        updatePropertyFrame.dispose();
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(updatePropertyFrame, "Invalid rent amount!");
    }
});

updatePropertyFrame.add(addressLabel);
updatePropertyFrame.add(addressField);
updatePropertyFrame.add(locationLabel);
updatePropertyFrame.add(locationField);
updatePropertyFrame.add(rentLabel);
updatePropertyFrame.add(rentField);
updatePropertyFrame.add(new JLabel(""));
updatePropertyFrame.add(updateButton);

updatePropertyFrame.setVisible(true);
}
}
```