

**RAJALAKSHMI ENGINEERING
COLLEGE**
RAJALAKSHMI NAGAR, THANDALAM – 602 105



**RAJALAKSHMI
ENGINEERING COLLEGE**

**CB23332
SOFTWARE ENGINEERING LAB**

Laboratory Record Note Book

Name :

Year / Branch / Section :

Register No. :

Semester :

Academic Year :

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)
RAJALAKSHMI NAGAR, THANDALAM – 602-105

BONAFIDE CERTIFICATE

NAME: _____ **REGISTER NO.:** _____

ACADEMIC YEAR: 2024-25 **SEMESTER:** III **BRANCH:** _____ B.E/B.Tech

This Certification is the bonafide record of work done by the above student in the

CB23332-SOFTWARE ENGINEERING - Laboratory during the year 2024 – 2025.

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on _____

Internal Examiner

External Examiner

INDEX

S.No.	Name of the Experiment	Expt. Date	Faculty Sign
1.	Preparing Problem Statement		
2.	Software Requirement Specification (SRS)		
3.	Entity-Relational Diagram		
4.	Data Flow Diagram		
5.	Use Case Diagram		
6.	Activity Diagram		
7.	State Chart Diagram		
8.	Sequence Diagram		
9.	Collaboration Diagram		
10.	Class Diagram		

EX NO:1	WRITE THE COMPLETE PROBLEM STATEMENT
DATE:	

AIM:

To prepare PROBLEM STATEMENT for **Hostel Management System**

ALGORITHM:

1. Initialize the system and set up the database.
2. Admin User logs in and sets up hostel details (rooms, fees, etc.).
3. Student User creates an account or logs in:
 - Searches for and requests room allocation.
 - Views accommodation details and notices.
 - Submits complaints or issues.
4. Hostel Warden User:
 - Logs in and reviews room allocation requests.
 - Approves or denies student requests.
 - Updates student details, complaints, and attendance.
5. Accountant User:
 - Logs in to manage student fee transactions.
 - Records payments and generates financial reports.
6. Admin generates periodic reports on occupancy, fees, and other statistics.
7. End program once all transactions and data entries are updated.

INPUT:

1. **Student Information:** Name, ID, contact details, room preferences.
2. **Room Information:** Room numbers, occupancy status, fee details.
3. **Payment Details:** Transaction IDs, payment date, amount.
4. **Complaints/Issues:** Student-submitted grievances or requests.
5. **Attendance Records:** Student attendance logs (updated by warden).

Objectives:

1. **Efficient Room Allocation:** Automate the process of allocating rooms to students based on their preferences and room availability.
2. **Fee Management:** Provide a module to facilitate fee transactions and tracking for students and accountants.
3. **Complaint Handling:** Enable students to submit complaints and allow the warden to monitor and address them.
4. **Student and Attendance Tracking:** Maintain student profiles, and track attendance for hostel management purposes.
5. **Report Generation:** Allow the admin to generate reports on room occupancy, fee status, and other hostel metrics.

Background:

Hostel management systems are software solutions designed to simplify and streamline the administrative tasks of managing hostels, dormitories, or accommodations for students, workers, or travelers. Traditional hostel management methods relied heavily on manual paperwork, spreadsheets, or basic computer tools, leading to inefficiencies, errors, and time consumption. With advancements in technology, hostel management systems have evolved into comprehensive platforms offering features such as room allocation, fee tracking, maintenance logging, and occupancy management.

These systems are increasingly becoming essential, especially in educational institutions, to provide a seamless experience for students and staff. They also enhance transparency and communication by integrating digital tools like online payment portals, automated notifications, and report generation.

Relevance:

Hostel management systems are highly relevant in today's fast-paced world, where efficiency and organization are paramount. Some key reasons for their relevance include:

1. **Digital Transformation in Education:**
Educational institutions are embracing technology to digitize processes. Hostel management systems fit into this trend, replacing manual methods with smart, automated solutions.
2. **Rising Demand for Student Housing:**
The increasing number of students enrolling in schools, colleges, and universities has led to a greater need for organized and scalable hostel management.
3. **Streamlined Operations:**
These systems eliminate repetitive tasks, reduce errors, and enable staff to focus on more critical responsibilities, thereby improving overall management efficiency.
4. **Enhanced User Experience:**
They provide a better experience for students and parents by enabling online bookings, fee payments, and real-time updates.
5. **Data Analytics and Reporting:**
Hostel management systems offer tools to track trends, occupancy rates, and maintenance issues, aiding in better decision-making and resource allocation.
6. **Compliance and Record Keeping:**
Such systems ensure adherence to rules and regulations by maintaining accurate and easily retrievable records.

Objectives:

1. **Efficient Room Allocation:** Automate the process of allocating rooms to students based on their preferences and room availability.
2. **Fee Management:** Provide a module to facilitate fee transactions and tracking for students and accountants.
3. **Complaint Handling:** Enable students to submit complaints and allow the warden to monitor and address them.
4. **Student and Attendance Tracking:** Maintain student profiles, and track attendance for hostel management purposes.
5. **Report Generation:** Allow the admin to generate reports on room occupancy, fee status, and other hostel metrics.

Result:

EX NO:2	WRITE THE SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT
DATE:	

AIM:

To do requirement analysis and develop Software Requirement Specification Sheet
(SRS) for Hostel Management System

ALGORITHM:

The algorithm below provides a basic outline of operations for the Hostel Management System:

1. **Start**
2. **Login System:** Verify user credentials for administrators, staff, and students.
3. **Manage Student Information:**
 - Add, edit, or delete student records.
 - Maintain contact details, room allocations, and fees.
4. **Room Allocation:**
 - Display available rooms.
 - Allocate rooms to students based on preference or availability.
5. **Fee Management:**
 - Record fee payments.
 - Track payment status and generate invoices.
6. **Attendance Tracking:**
 - Log student attendance.
 - Generate attendance reports.
7. **Report Generation:**
 - Generate reports for occupancy, fees, and student records.
8. **End**

1. Introduction**Purpose**

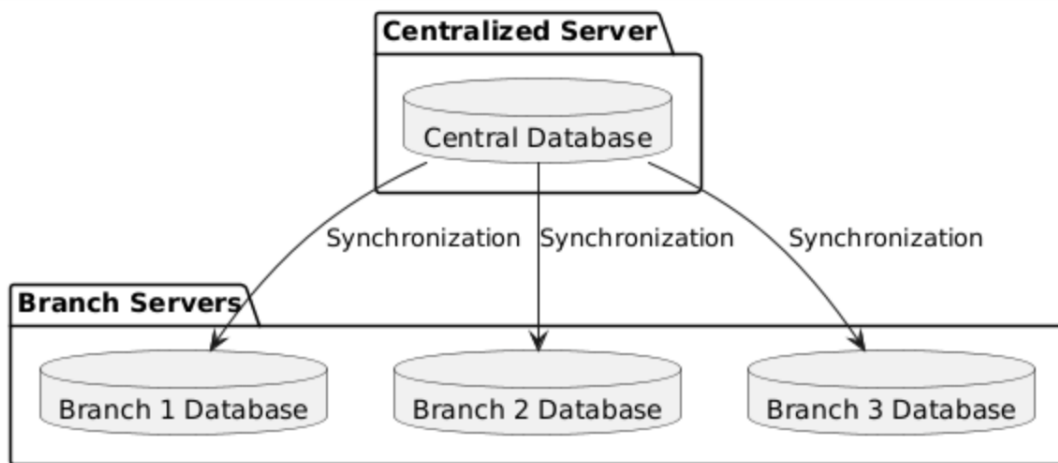
The purpose of this document is to outline the requirements and specifications for developing a *Hostel Management System (HMS)*. This system will provide a centralized platform for managing hostel operations such as room allocation, fee collection, maintenance tracking, and resident information, ensuring efficient and transparent administration.

Scope

The Hostel Management System will cater to educational institutions, hostels, and other residential facilities, automating tasks like room assignments, payment tracking, and generating reports. It will benefit hostel administrators, staff, and residents by reducing manual effort, improving data accuracy, and offering real-time updates.

Definitions, Acronyms, and Abbreviations

- HMS: Hostel Management System



- Admin: Administrator responsible for managing hostel operations
- Resident: A student, worker, or traveler staying in the hostel
- UI: User Interface
- API: Application Programming Interface

References

- Software Development Lifecycle Guidelines
- Educational Institution Management Standards
- Hostel Management Software Manuals

2. Overall Description

Product Perspective

The HMS will function as a web-based or desktop application with a user-friendly interface. It will integrate with existing systems, such as financial management software, and support multi-device access (desktop, mobile, tablets).

Product Functions

1. Room Allocation and Management
2. Fee Payment and Billing System
3. Maintenance Request Tracking
4. Occupancy and Vacancy Reports
5. Resident Information Database
6. Notifications and Alerts for dues or events

User Characteristics

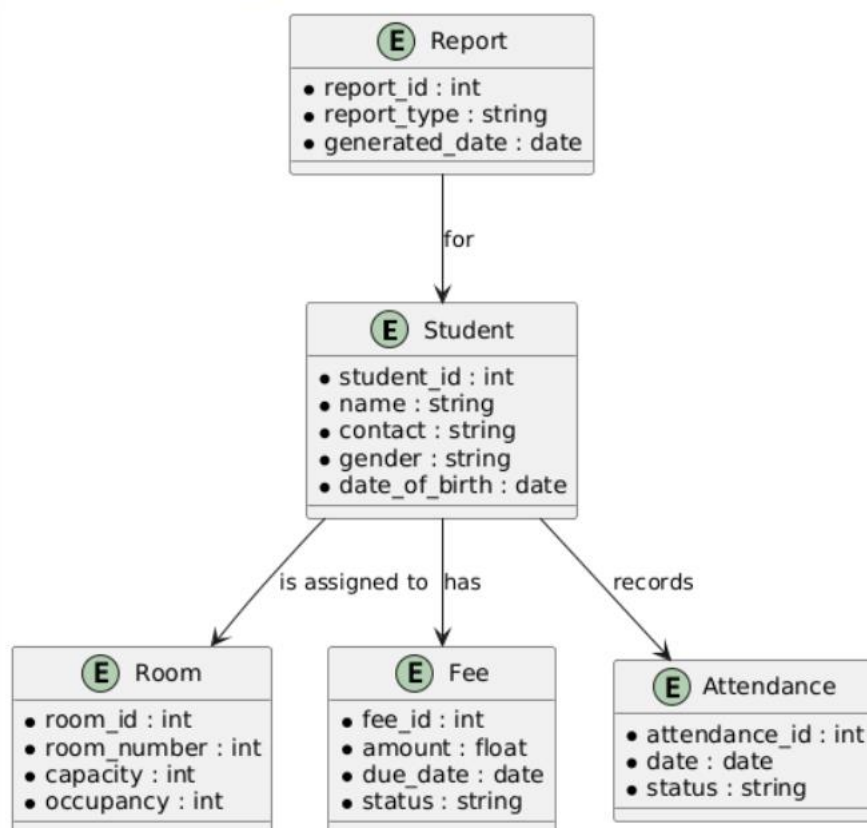
- Administrators: Knowledgeable in hostel operations; require access to manage all modules.
- Residents: Basic computer or smartphone usage skills; use for payments and updates.
- Maintenance Staff: Limited access for managing repair requests.

Constraints

- The system must be accessible 24/7 with minimal downtime.
- Must comply with data privacy regulations (e.g., GDPR).
- Limited budget for hardware upgrades in some institutions.

Assumptions and Dependencies

- Internet access is available to all users.
- Users have basic computer literacy.
- Third-party payment gateways for online fee collection are reliable.



3. System Features

System Module Requirements

1. Room Management Module

- Add, update, or delete room details.
- Allocate rooms to residents.

2. Fee Management Module

- Record payment details.
- Generate receipts and reminders.

3. Resident Information Module

- Maintain personal and contact details.
- Track residency history.

Functionality Requirements

- Search functionality for residents and rooms.
- Generate reports for fees, occupancy, and maintenance.
- Automated notifications for room assignments and payment deadlines.

4. External Interface Requirements

User Interfaces

- Intuitive dashboard for administrators.
- Mobile-friendly portal for residents to view details and make payments.

Hardware Interfaces

- Compatible with standard devices (PCs, smartphones, printers).

Software Interfaces

- Integration with third-party payment systems (e.g., PayPal, Stripe).
- Synchronization with institution databases (if applicable).

Communication Interfaces

- Email and SMS notification services for alerts.
- Support for LAN/Wi-Fi connectivity for internal operations.

5. Non-functional Requirements

Performance Requirements

- The system must handle simultaneous access by 500+ users.
- Ensure response time for user queries within 2 seconds.

Security Requirements

- Data encryption for sensitive information like payments.
- Role-based access to modules to protect data integrity.

Usability Requirements

- Easy-to-navigate interface with minimal training required.
- Multi-language support for diverse user bases.

Reliability Requirements

- 99.9% uptime with backups performed daily.
- Automatic recovery mechanisms in case of failures.

6. Other Requirements

Legal and Compliance Requirements

- Adhere to GDPR for data privacy in Europe.
- Comply with FERPA for student data protection in the U.S. (if applicable).
- Ensure accessibility compliance (e.g., WCAG standards).
- Legal agreements with third-party services like payment gateways.

This structured approach ensures the Hostel Management System aligns with user needs, regulatory standards, and technological advancements.

Result:

EX NO:3	DRAW THE ENTITY RELATIONSHIP DIAGRAM
DATE:	

AIM:

To Draw the Entity Relationship Diagram Hostel Management Systems Project

ALGORITHM:

1. **Start**
2. **Identify Entities:** Determine entities such as Student, Room, Fee, Attendance, and Report.
3. **Define Attributes** for each entity:
 - Student: student_id, name, contact, gender, etc.
 - Room: room_id, room_number, capacity, etc.
 - Fee: fee_id, amount, due_date, etc.
 - Attendance: attendance_id, date, status, etc.
4. **Define Relationships** between entities:
 - Student is assigned to a Room.
 - Student has associated Fee records.
 - Student has Attendance records.
 - Report is generated for a Student.
5. **Specify Cardinalities** for each relationship (one-to-many, many-to-one, etc.).
6. **End**

INPUT:

The primary inputs for the ER Diagram are:

- Entities: The objects in the system, e.g., Student, Room.
- Attributes: Properties of each entity, e.g., name, contact.
- Relationships: Connections between entities, e.g., a student is assigned a room.
- Constraints: Cardinalities or limitations on relationships.

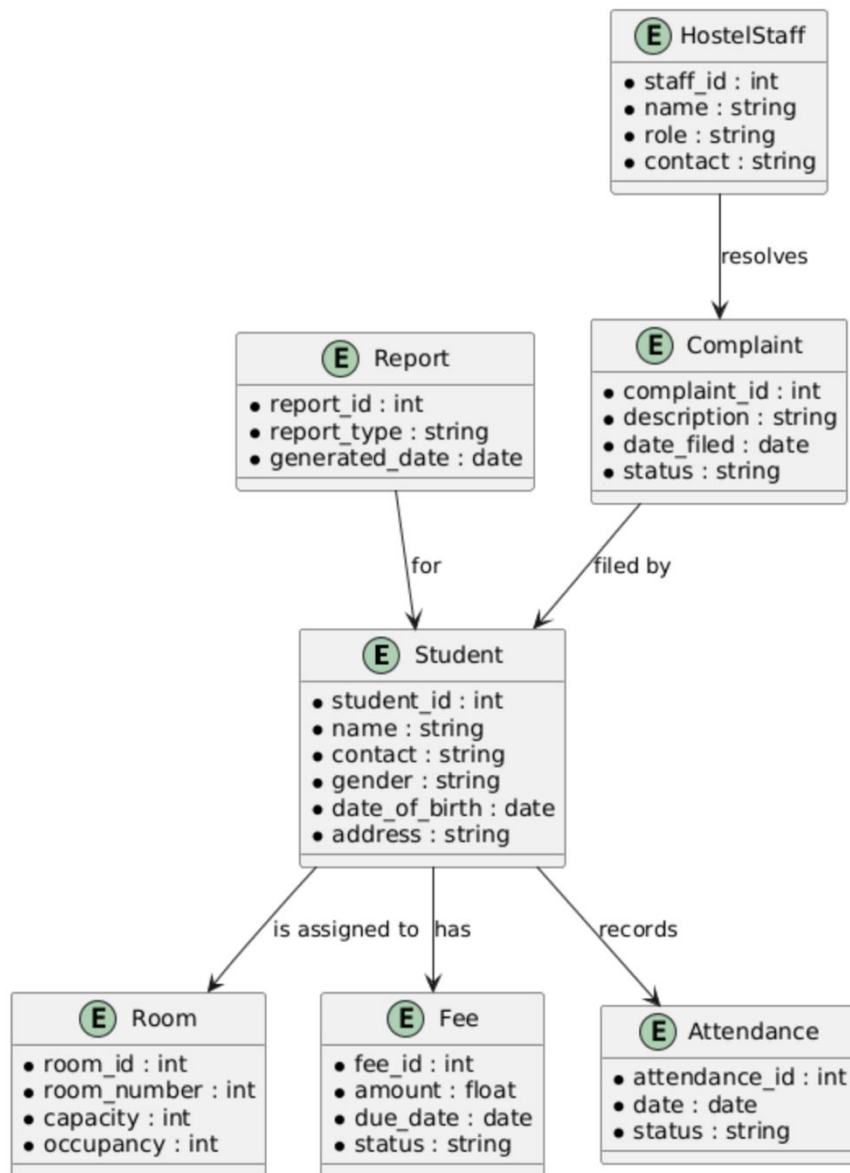
2. Attributes for Each Entity

1. Student

- student_id (Primary Key): Unique identifier for each student.
- name: Full name of the student.
- contact: Contact number for the student.
- email: Email address of the student.
- gender: Gender of the student.
- date_of_birth: Date of birth.
- address: Residential address.
- course: The course the student is enrolled in.
- joining_date: Date when the student joined the hostel.
- room_id (Foreign Key): ID of the assigned room.

2. Room

- room_id (Primary Key): Unique identifier for each room.
- room_number: The number of the room.
- capacity: Maximum capacity of the room.
- occupancy: Current occupancy count (i.e., number of students currently in the room).



- type: Type of room (e.g., single, double, shared).
- floor_number: Floor where the room is located.
- status: Status of the room (e.g., available, full, under maintenance).

3. Fee

- fee_id (Primary Key): Unique identifier for each fee record.
- student_id (Foreign Key): ID of the student the fee is associated with.
- amount: Fee amount to be paid.
- due_date: Date by which the fee should be paid.
- status: Payment status (e.g., pending, paid, overdue).
- payment_date: Date when the fee was paid.
- receipt_number: Receipt number for the payment.

4. Attendance

- attendance_id (Primary Key): Unique identifier for each attendance record.
- student_id (Foreign Key): ID of the student.
- date: Date of the attendance record.
- status: Attendance status (e.g., present, absent).
- remarks: Optional comments regarding the attendance for that day.

5. Report

- report_id (Primary Key): Unique identifier for each report.
- student_id (Foreign Key): ID of the student the report is generated for.
- report_type: Type of report (e.g., fee summary, attendance, disciplinary).
- generated_date: Date the report was generated.
- description: Brief summary of the report.
- generated_by: Name or ID of the staff who generated the report.

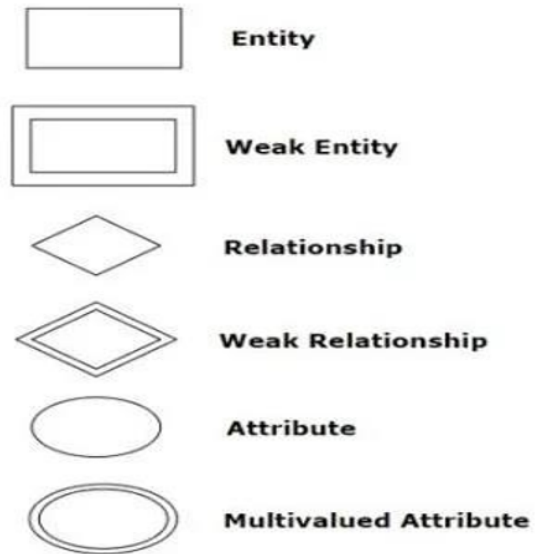
6. HostelStaff

- staff_id (Primary Key): Unique identifier for each staff member.
- name: Name of the staff member.
- role: Role or designation (e.g., warden, caretaker, admin).
- contact: Contact number of the staff member.
- email: Email address of the staff member.
- date_of_joining: Date the staff joined the hostel.
- address: Residential address of the staff.

7. Complaint

- complaint_id (Primary Key): Unique identifier for each complaint.
- student_id (Foreign Key): ID of the student who filed the complaint.
- staff_id (Foreign Key): ID of the staff member assigned to resolve the complaint.
- description: Description of the complaint.
- date_filed: Date the complaint was filed.
- status: Status of the complaint (e.g., open, in progress, resolved).
- resolution_date: Date when the complaint was resolved.
- remarks: Additional comments or remarks regarding the complaint.

Symbols Used:



Cardinality:

- 1:1 - One-to-One Relationship
- 1:N - One-to-Many Relationship
- N:M - Many-to-Many Relationship

Result:

EX NO:4	DRAW THE DATA FLOW DIAGRAMS AT LEVEL 0 AND LEVEL 1
DATE:	

AIM:

To Draw the Data Flow Diagram for Hostel Management Systems Project and List the Modules in the Application.

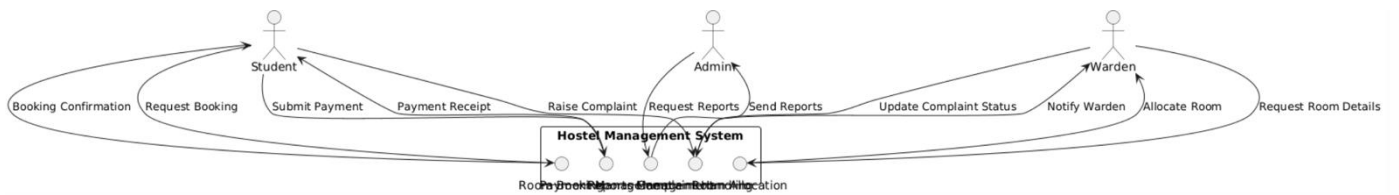
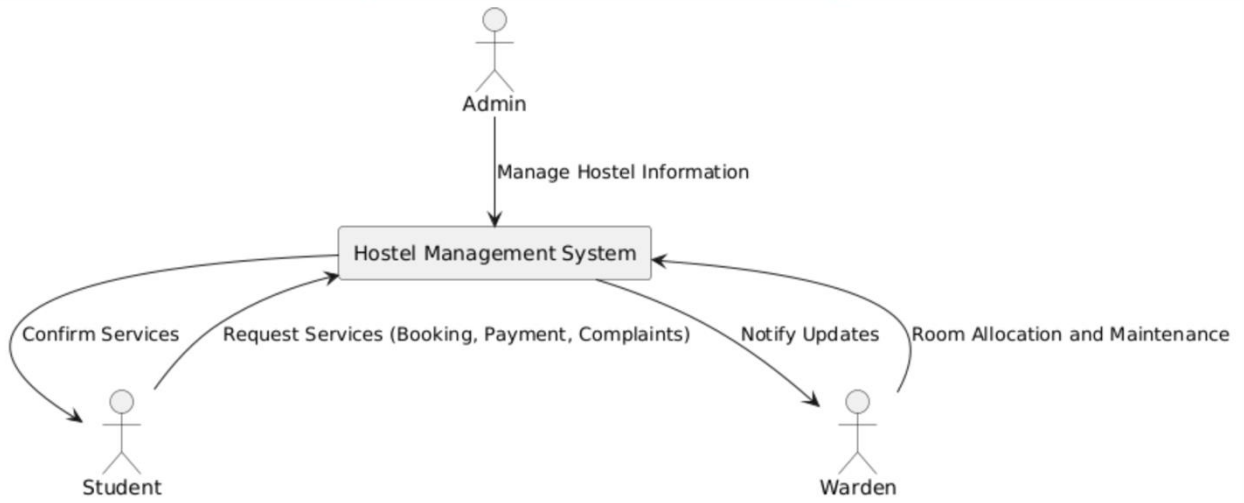
ALGORITHM:

1. **Start**
2. **Student Management:**
 - Receive inputs from the user (add, update, delete student records).
 - Store data in the Student data store.
3. **Room Allocation:**
 - Retrieve available rooms from the Room data store.
 - Allocate rooms to students and update occupancy in the Room data store.
4. **Fee Management:**
 - Record fee payments for each student.
 - Update fee status in the Fee data store.
5. **Attendance Management:**
 - Capture student attendance data.
 - Store attendance details in the Attendance data store.
6. **Complaint Management:**
 - Record complaints filed by students.
 - Assign complaints to hostel staff and update status in the Complaint data store.
7. **Reporting and Analytics:**
 - Generate reports based on data from various data stores.
 - Display or export reports for analysis.
8. **End**





INPUT:

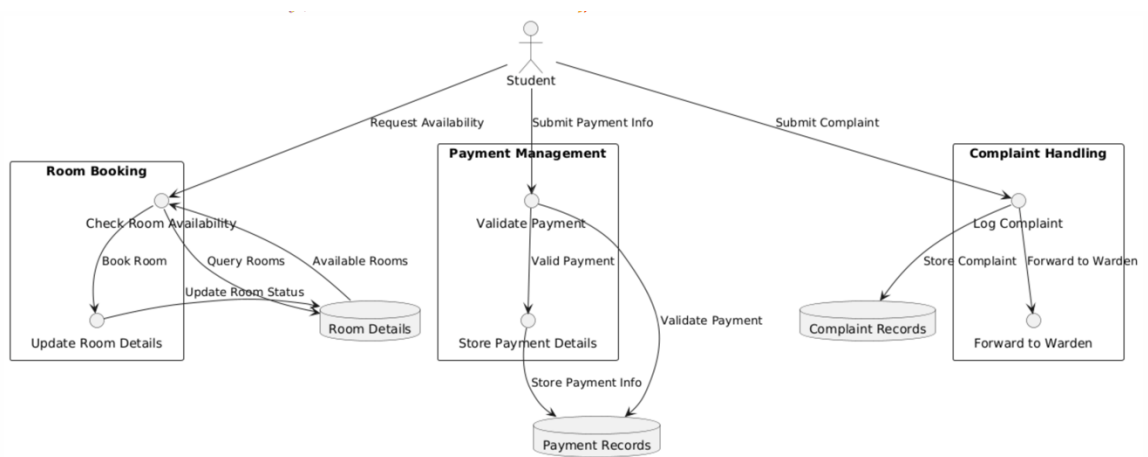
The primary inputs for the DFD are as follows:

- **External Entities:**
 - Admin: Adds or updates student details, manages rooms, handles fees, monitors attendance, and generates reports.
 - Student: Views room allocation, pays fees, checks attendance, and files **complaints**.
- **Processes:**
 - Student Management
 - Room Allocation
 - Fee Management
 - Attendance Management
 - Complaint Management
 - Report Generation
- **Data Stores:**
 - Student Data Store
 - Room Data Store
 - Fee Data Store
 - Attendance Data Store
 - Complaint Data Store



Symbols Used:

	dataflow	Arrows showing direction of flow
	process	circles
	file	horizontal pair of lines
	data- source, sink	rectangular box



Result:

EX NO:5	DRAW USE CASE DIAGRAM
DATE:	

AIM:

To Draw the Use Case Diagram for Hostel Management Systems Project

ALGORITHM:

1. **Identify Actors:**
 - Determine all the users (both human and external systems) that interact with the Hostel Management System, such as **Admin, Student, and Hostel Staff.**
2. **Identify Use Cases:**
 - List the primary actions or use cases each actor can perform within the system, such as **Manage Students, Allocate Rooms, Pay Fees, Track Attendance, and Generate Reports.**
3. **Define Relationships:**
 - Establish relationships between actors and use cases.
 - **Include:** For use cases that are always a part of a larger function.
 - **Extend:** For optional or conditional use cases that extend core use cases.
4. **Draw the Diagram:**
 - Place actors outside the system boundary.
 - Draw use cases inside the system boundary.
 - Use association lines to connect actors with their respective use cases.

INPUTS:

The main inputs for the Use Case Diagram are as follows:

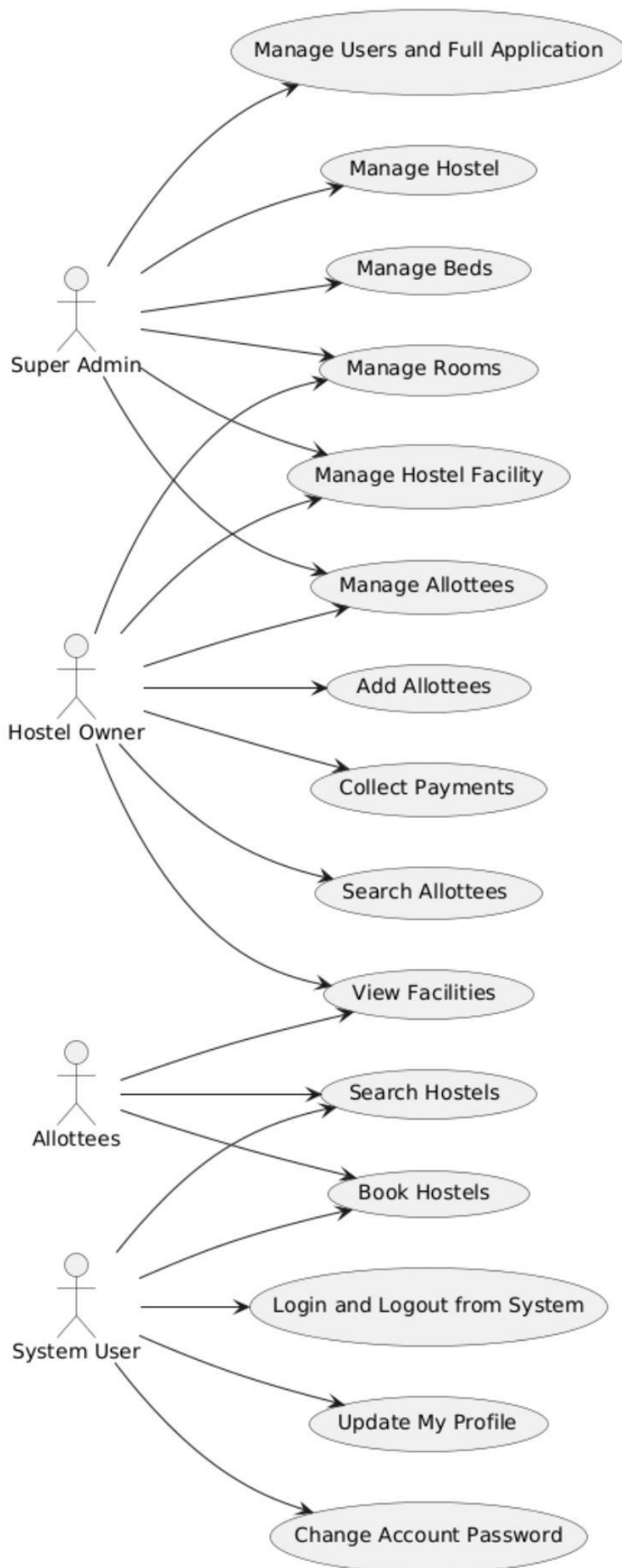
- **Actors:** Key users who interact with the system (e.g., Admin, Student, Hostel Staff).
- **Use Cases:** Functionalities or actions each actor can perform within the system.
- **Relationships:** Connections between actors and use cases, including association, inclusion, and extension relationships.

Relations between Actors and Use Cases:

1. **Admin:**
 - Can manage students, rooms, fees, attendance, and complaints.
 - Can generate reports.
2. **Student:**
 - Can view room allocation, pay fees, view attendance, and file complaints.
3. **Hostel Staff:**
 - **Can view and resolve complaints filed by students.**

Use Cases and Relationships

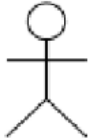
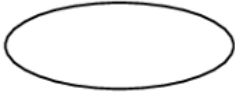
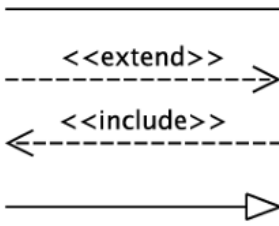
1. Manage Students: Admin can add, update, or delete student records.
2. Allocate Rooms: Admin can allocate rooms to students and check room availability.
3. Pay Fees: Students can pay fees, and the system will update their payment status.
4. Track Attendance: Admin can record student attendance.
5. File Complaint: Students can file complaints regarding hostel facilities or issues.
6. Resolve Complaint: Hostel Staff can view and resolve complaints.



7.

Generate Reports: Admin can generate reports on various aspects, like occupancy, fee collection, and attendance.

Symbols Used:

Symbol	Reference Name
	Actor
	Use case
	Relationship

Result:

EX NO:6	DRAW ACTIVITY DIAGRAM OF ALL USE CASES.
DATE:	

AIM:

To Draw the activity Diagram for Hostel Management Systems Project.

ALGORITHM:

1. **Identify Core Processes:**
 - Identify major processes in the system, such as **Student Registration, Room Allocation, Fee Payment, Attendance Management, and Complaint Handling.**
2. **Define Actions:**
 - Define the individual actions that each process entails (e.g., validating student details, checking room availability, updating fee records, etc.).
3. **Define Decision Points:**
 - Identify the decision points where different paths are possible (e.g., if a room is available, if fees are paid, if complaints are resolved).
4. **Draw the Diagram:**
 - Use a start node to indicate the beginning of the process.
 - Connect actions using arrows to indicate the flow of activities.

INPUTS:

The primary inputs for the Activity Diagram are as follows:

- **Actors:** The main users who interact with the system, such as **Admin, Student, and Hostel Staff.**
- **Processes and Actions:** Actions involved in each of the core processes.
- **Decision Points:** Conditions that determine different paths in the activity flow.

Key Activities in the Hostel Management System

1. **Student Registration:**
 - Register a new student, verify details, assign a room, and update the student record.
2. **Room Allocation:**
 - Check room availability, allocate room if available, and update room records.
3. **Fee Management:**
 - Check fee status, process fee payment, and update payment record.
4. **Attendance Management:**
 - Record attendance for each student and update attendance records.
5. **Complaint Handling:**
 - Receive complaints, assign to hostel staff, check if resolved, and close complaint.

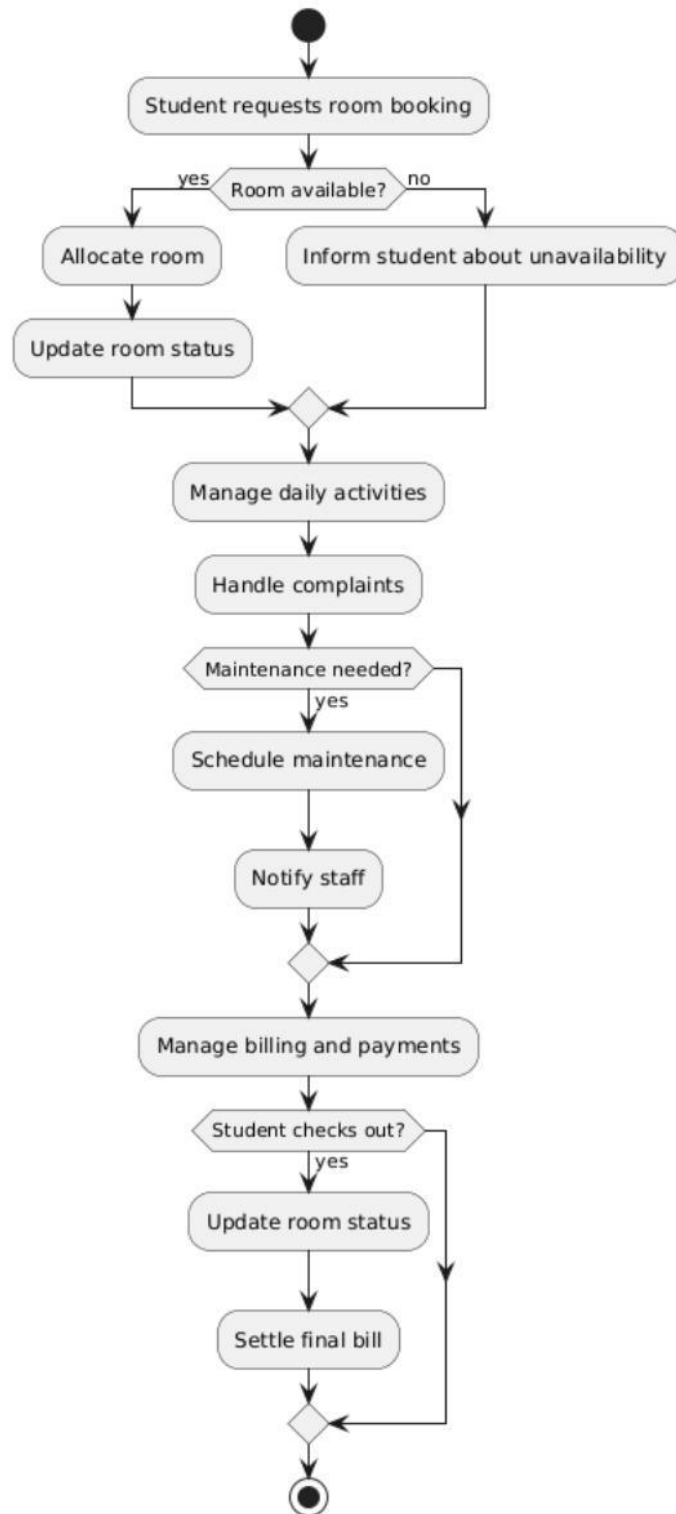
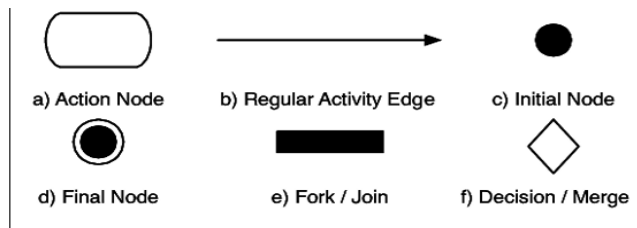


Diagram:



Result:

EX NO:7	DRAW STATE CHART DIAGRAM OF ALL USE CASES.
DATE:	

AIM:

To Draw the State Chart Diagram for Hostel Management Systems Project.

ALGORITHM:

1. Identify Key States:

- Define the main states that the student entity can occupy throughout its life cycle in the system, such as **Registered, Room Allocated, Fee Pending, Fee Paid, Complaint Filed, Complaint Resolved, Checked-Out.**

2. Define Events/Transitions:

- Identify the triggers that cause a state transition. For instance, **fee payment** may transition a student from the **Fee Pending** state to the **Fee Paid** state.

3. Draw the Diagram:

- Start with an initial state to represent the beginning of the student's journey in the system.
- Add arrows to indicate transitions between states based on specific events.
- End with a final state, representing the student's departure from the hostel (e.g., **Checked-Out**).

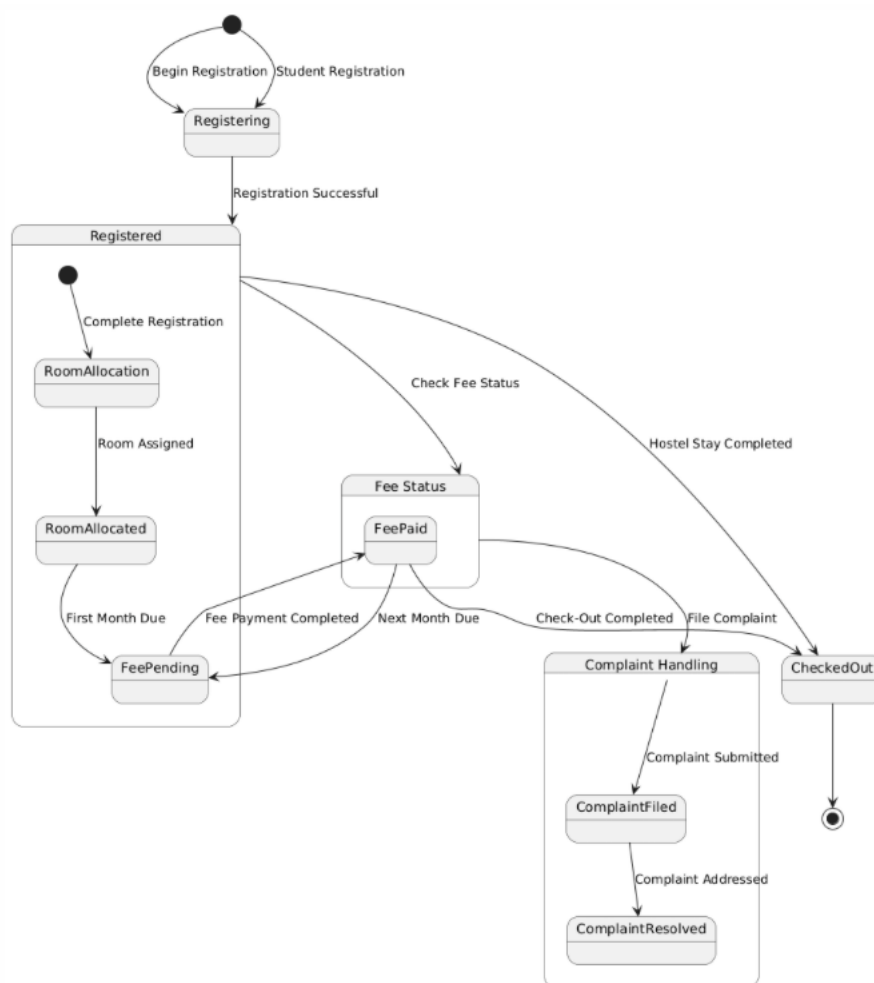
INPUTS:

The main inputs for the State Chart Diagram include:

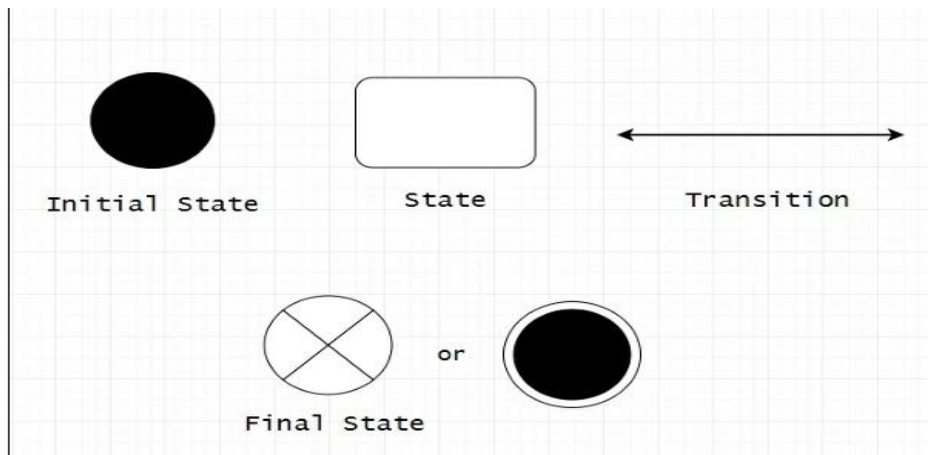
- States: Different statuses the student can be in (e.g., Registered, Room Allocated, Fee Pending).
- Transitions/Events: Actions that move the student from one state to another (e.g., paying fees, filing complaints, checking out).
- Actors/Entities: Admins, students, and hostel staff who trigger these transitions.

Key States for the Student in the Hostel Management System

1. Initial State: The starting point when a student is about to register.
2. Registered: The student is registered in the system.
3. Room Allocated: The student has been assigned a room.
4. Fee Pending: The student has an outstanding fee.
5. Fee Paid: The student has cleared all outstanding fees.
6. Complaint Filed: The student has lodged a complaint.
7. Complaint Resolved: The complaint has been addressed and closed.
8. Checked-Out: The student has completed their stay and checked out.



Diagram



Result:

EX NO:8	DRAW SEQUENCE DIAGRAM OF ALL USE CASES.
DATE:	

AIM:

To Draw the Sequence Diagram for Hostel management system project

ALGORITHM:

To create the Sequence Diagram for a Hostel Management System, follow these steps:

1. **Identify the Scenario:**
 - Select a specific process to illustrate, such as **Student Registration** or **Room Allocation**.
2. **Identify Actors and Objects:**
 - List the primary actors (e.g., **Admin, Student, Database**) and objects involved in the scenario.
3. **Define Messages:**
 - Outline the messages exchanged between actors and system components throughout the process. This includes requests, responses, and confirmations.
4. **Draw the Diagram:**
 - Arrange the actors and objects horizontally at the top of the diagram.
 - Draw vertical lifelines for each actor and object.
 - Use arrows to represent messages exchanged between them, with the order of messages flowing top to bottom.





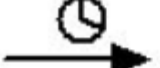
Inputs**The primary inputs for the Sequence Diagram are:**

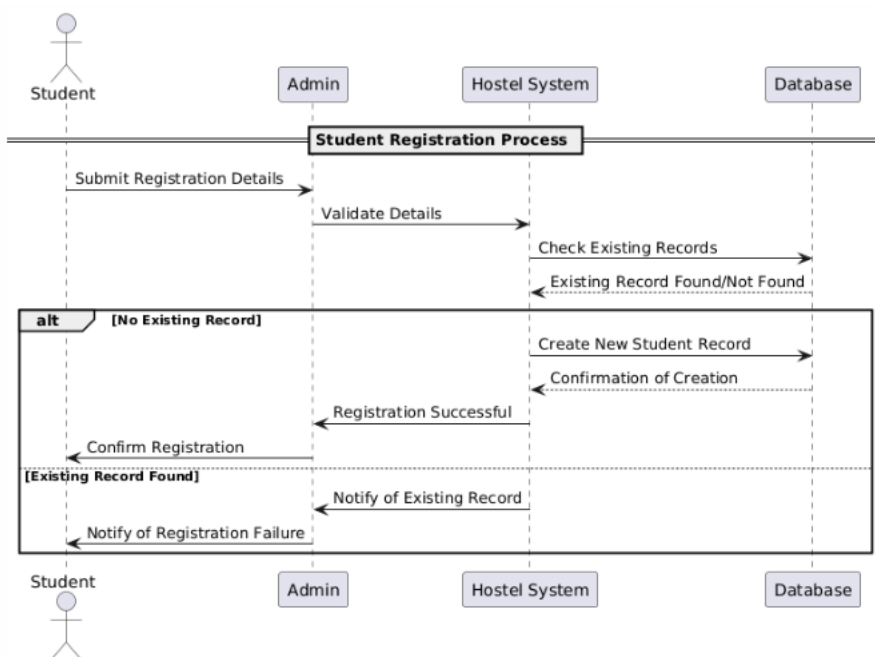
- Actors: Key users who interact with the system (e.g., Admin, Student).
- Objects: Components involved in the process (e.g., Database).
- Messages: Actions and communications between actors and objects (e.g., Register Student, Validate Details, Confirm Registration).

Key Scenario: Student Registration Process

1. Student submits registration details.
2. Admin validates the details.
3. System checks for existing records.
4. System creates a new student record.
5. Confirmation is sent to the student.

Symbols Used:

Arrow	Message type
	Simple
	Synchronous
	Asynchronous
	Balking
	Time out



Result:

EX NO:9	DRAW COLLABORATION DIAGRAM OF ALL USE CASES
DATE:	

AIM:

To Draw the Collaboration Diagram for Hostel management system project.

ALGORITHM:

1. **Identify Use Cases:**
 - Choose a specific use case scenario, such as **Room Allocation** or **Fee Payment**.
2. **Identify Objects:**
 - List the relevant objects involved in the chosen use case (e.g., Student, Room, Fee Management, Hostel Staff).
3. **Define Relationships:**
 - Establish how these objects are related to each other, such as associations, dependencies, and aggregation.
4. **Define Message Exchanges:**
 - Determine the sequence of messages that will be exchanged between the objects to complete the use case.
5. **Draw the Diagram:**
 - Represent each object as a box or a labeled circle.
 - Use lines to connect related objects, indicating their relationships.
 - Label the lines with messages exchanged, including the sequence number to indicate the order of messages.

Inputs

The primary inputs for the Collaboration Diagram are as follows:

- **Use Case Scenario:** The specific scenario being represented (e.g., Room Allocation).
- **Objects:** The main objects involved in the scenario (e.g., Student, Room, Fee Management, and Hostel Staff).
- **Message Exchanges:** The interactions or messages passed between the objects during the execution of the use case.

Example Use Case: Room Allocation

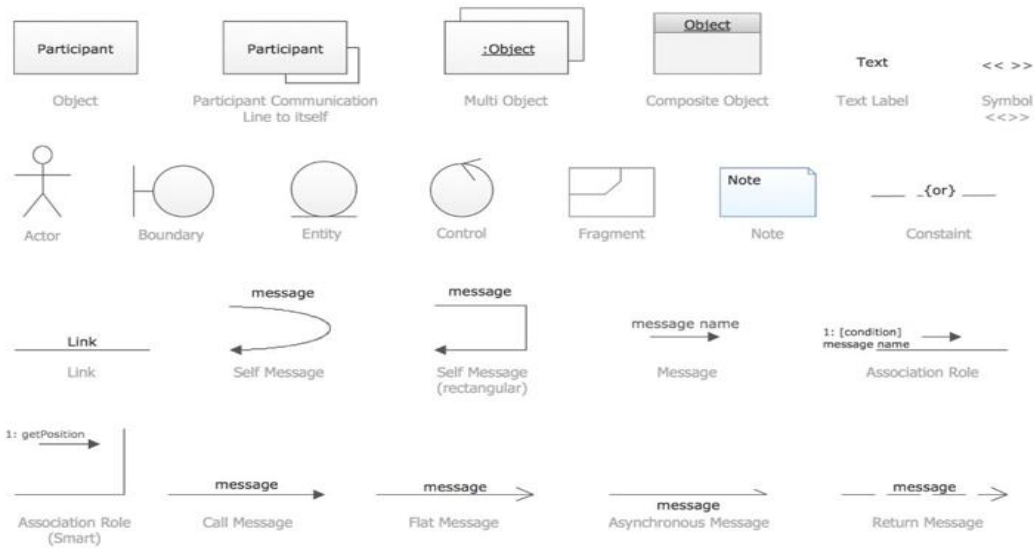
Objects Involved

1. **Student:** Represents the student seeking a room.
2. **Room:** Represents the available rooms in the hostel.
3. **Room Manager:** Represents the system responsible for room allocations.
4. **Database:** Represents the data store where room information is kept.

Messages Exchanged

1. Student requests room availability.
2. Room Manager queries the database for available rooms.
3. Database returns the list of available rooms.
4. Room Manager displays available rooms to the student.
5. Student selects a room.
6. Room Manager updates the room status in the database.
7. Database confirms the room allocation.
8. Room Manager notifies the student of successful allocation.

Symbols Used:



Result:

EX NO:10	ASSIGN OBJECTS IN SEQUENCE DIAGRAM TO CLASSES AND MAKE CLASS DIAGRAM.
DATE:	

AIM:

To Draw the Class Diagram for any Hostel management system project.

Algorithm:**Room Booking Algorithm**

To allow a guest to book a room.

- Guest ID
- Room Number
- Check-in Date
- Check-out Date
- Booking ID
- Payment Amount

Input:

Guest ID

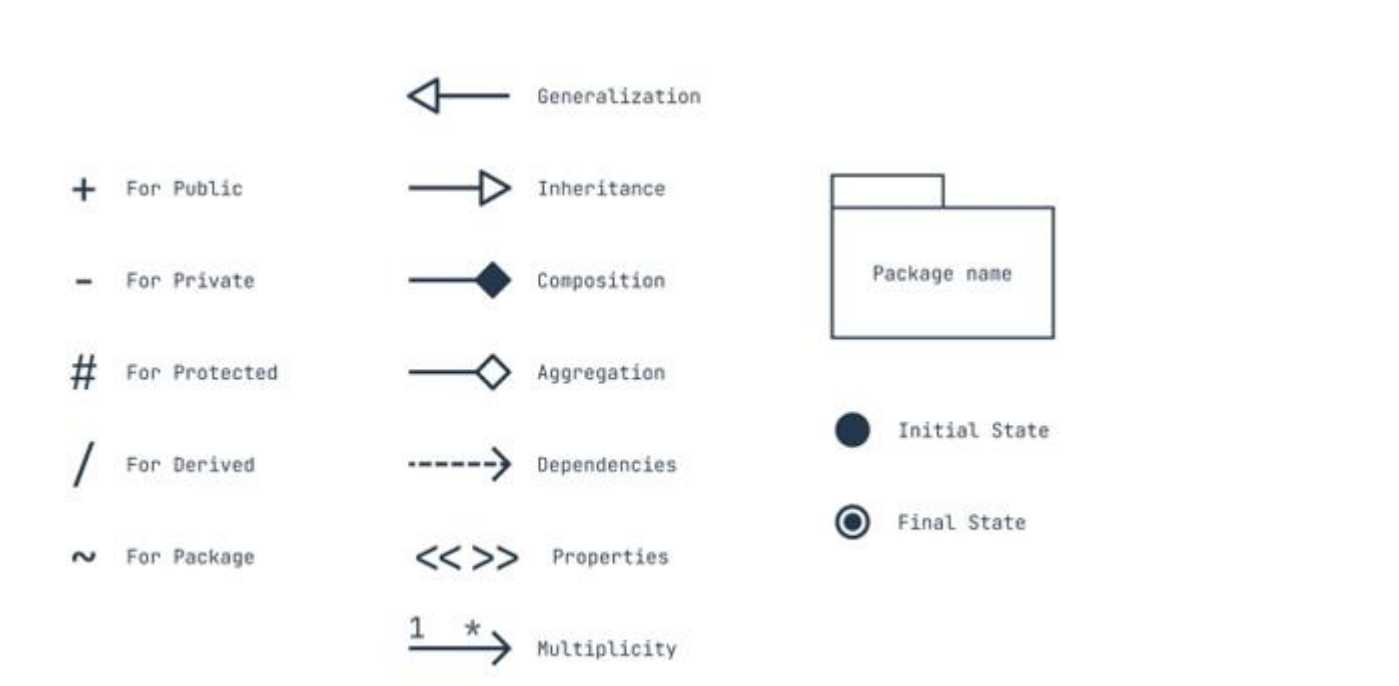
- **Attributes:**

- 1. Start
- 2. Retrieve guest using Guest ID
- 3. Retrieve room using Room Number
- 4. If room is not available (isBooked == true):
 - a. Display "Room is not available"
 - b. End
- 5. Create a new Booking object with:
 - a. Booking ID (generate unique ID)
 - b. Guest (retrieved guest)
 - c. Room (retrieved room)
 - d. Check-in Date
 - e. Check-out Date
- 6. Call room.bookRoom() to mark room as booked
- 7. Store the Booking object in a list of bookings
- 8. Display "Booking confirmed"
- 9. End

- **Check-In Algorithm**

- +1. Start
- 2. Retrieve guest using Guest ID
- 3. Retrieve room using Room Number
- 4. Find the booking for the guest and room
- 5. If booking does not exist:
 - a. Display "No booking found"
 - b. End
- 6. If room is already checked in (isBooked == false):
 - a. Display "Room is already checked in"
 - b. End
- 7. Call Hostel.checkIn() method with guest and room
- 8. Update room status (isBooked = false)
- 9. Display "Check-in successful"
- 10. End

Diagram:

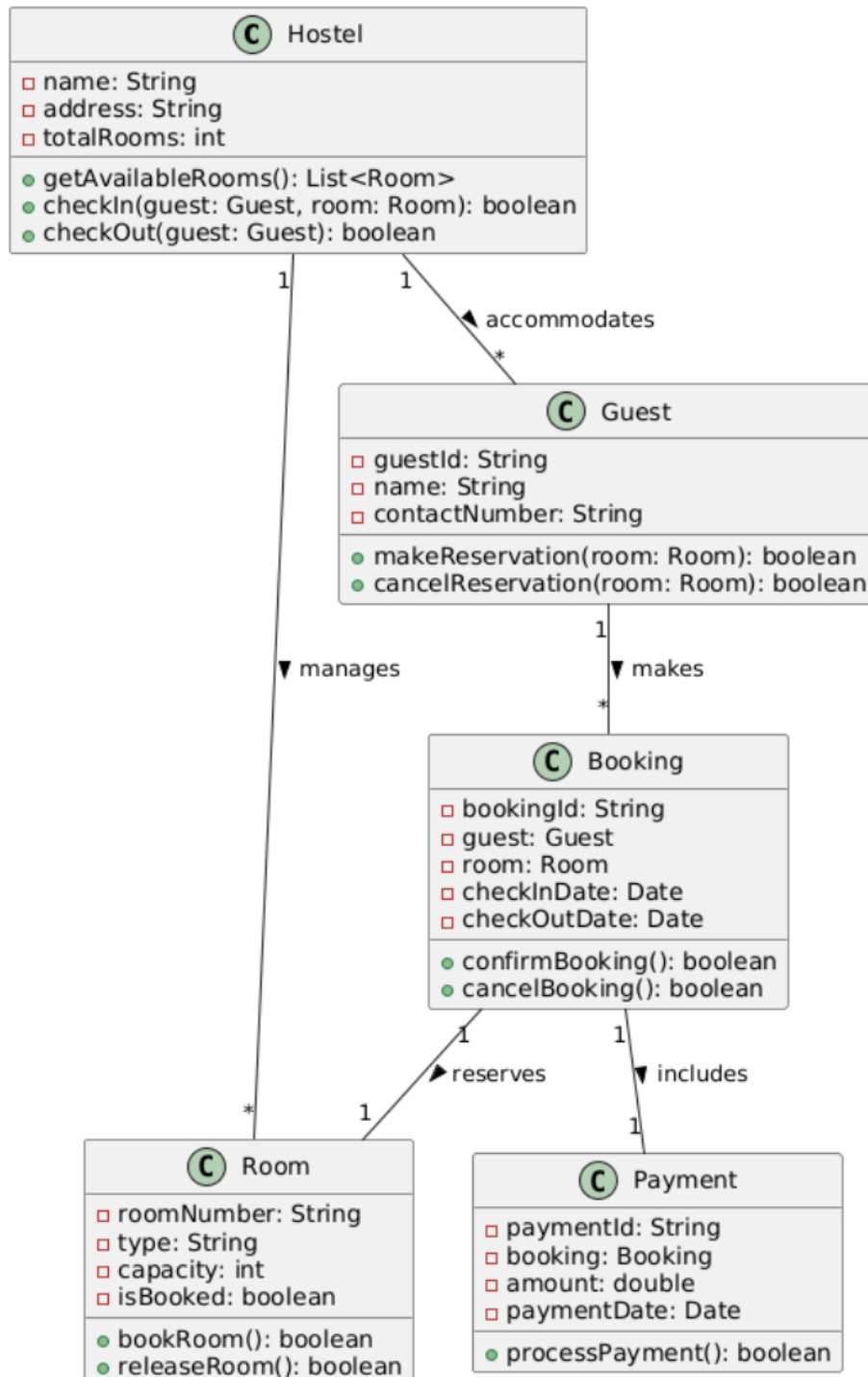


- **Check-Out Algorithm:**

- 1. Start
- 2. Retrieve guest using Guest ID
- 3. Retrieve room using Room Number
- 4. Find the booking for the guest and room
- 5. If booking does not exist:
 - a. Display "No booking found"
 - b. End
- 6. Call Hostel.checkOut() method with guest
- 7. Update room status (isBooked = true)
- 8. Display "Check-out successful"
- 9. End

Payment Processing Algorithm

- 1. Start
- 2. Retrieve booking using Booking ID
- 3. If booking does not exist:
 - a. Display "No booking found"
 - b. End
- 4. Create a new Payment object with:
 - a. Payment ID (generate unique ID)
 - b. Booking (retrieved booking)
 - c. Amount (input amount)
 - d. Payment Date (current date)
- 5. Call Payment.processPayment() method
- 6. Update booking status to paid
- 7. Display "Payment processed successfully"
- 8. End



RESULT:

EX NO:11	MINI PROJECT ON HOSTEL MANAGEMENT SYSTEMS
DATE:	

Aim

To design and implement a Hostel Management System using Python and Streamlit that facilitates the management of student data, attendance records, fee payments, and complaints in a streamlined and interactive web-based application.

Algorithm

1. Initialize the Application

- Import necessary libraries (streamlit, sqlite3, pandas, datetime).
- Define a function to initialize the SQLite database and create the required tables (students, attendance, fees, complaints).

2. Database Operations

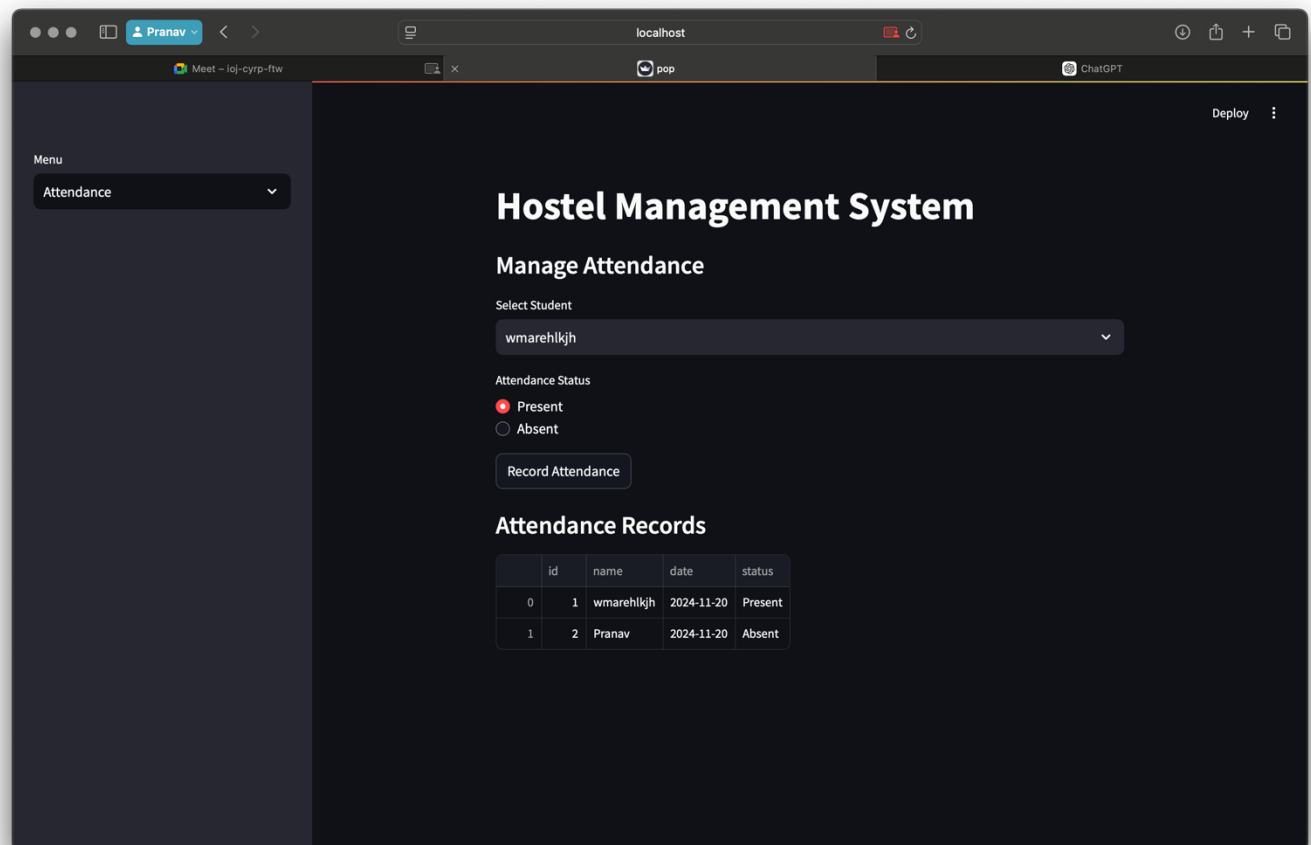
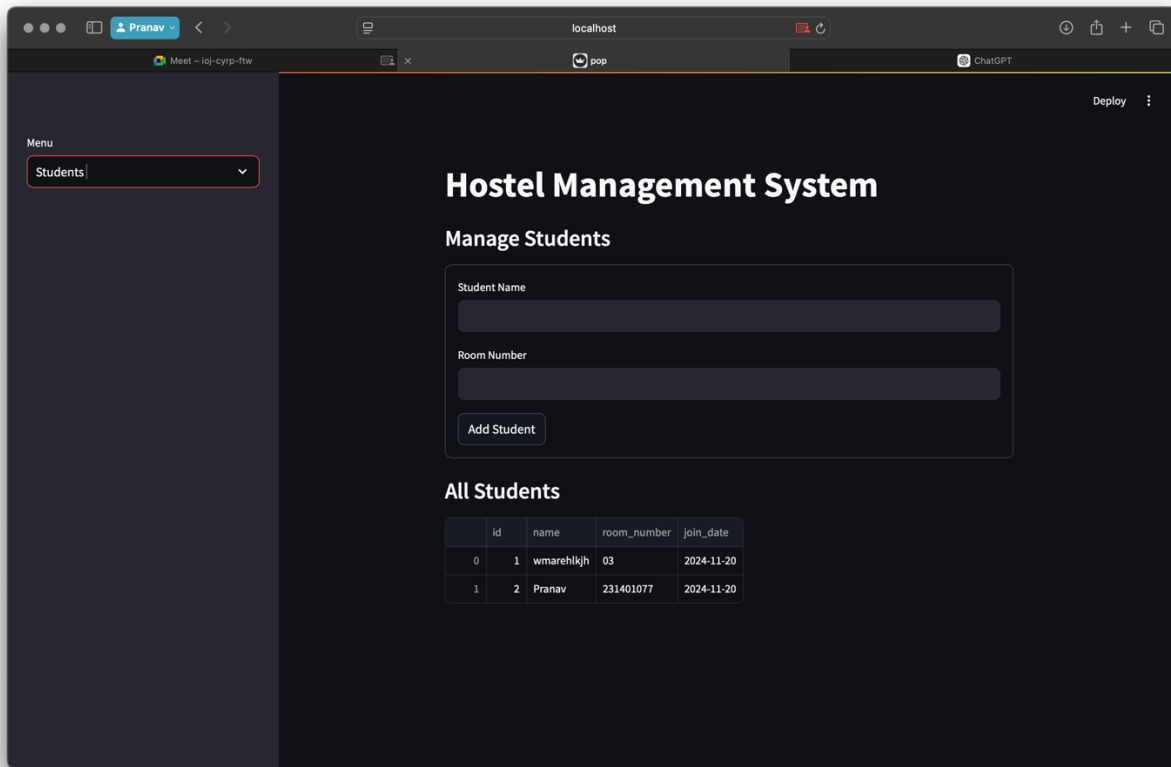
- Create functions to perform CRUD operations:
 - Add students (add_student).
 - Fetch all students (get_students).
 - Record attendance (record_attendance).
 - Retrieve attendance records (get_attendance).
 - Record fee payments (record_fee).
 - Retrieve fee records (get_fees).
 - Add complaints (add_complaint).
 - Retrieve complaints (get_complaints).

3. Design the Streamlit User Interface

- Use st.sidebar.selectbox to create a menu for the different functionalities: **Students**, **Attendance**, **Fees**, and **Complaints**.

4. Module Implementation

- **Students Module:**
 - Display a form to add a new student with fields for name and room number.
 - Use the get_students function to fetch and display the list of all students in a table.
- **Attendance Module:**
 - Provide a dropdown to select a student by name.
 - Display options to mark attendance as "Present" or "Absent."
 - Record attendance and fetch the attendance records for display.
- **Fees Module:**



- Provide a dropdown to select a student by name.
- Input field to enter the amount paid.
- Record the fee payment and display all fee records.

- **Complaints Module:**

- Provide a dropdown to select a student by name.
- Input field to submit a complaint.
- Display all complaints submitted by students.

5. Add Data Validation

- Validate user inputs in forms to ensure no empty or invalid data is submitted.

6. Connect UI with Database Operations

- Call the appropriate database functions based on user actions:
 - Add Student → add_student
 - Record Attendance → record_attendance
 - Record Payment → record_fee
 - Submit Complaint → add_complaint

7. Display Data

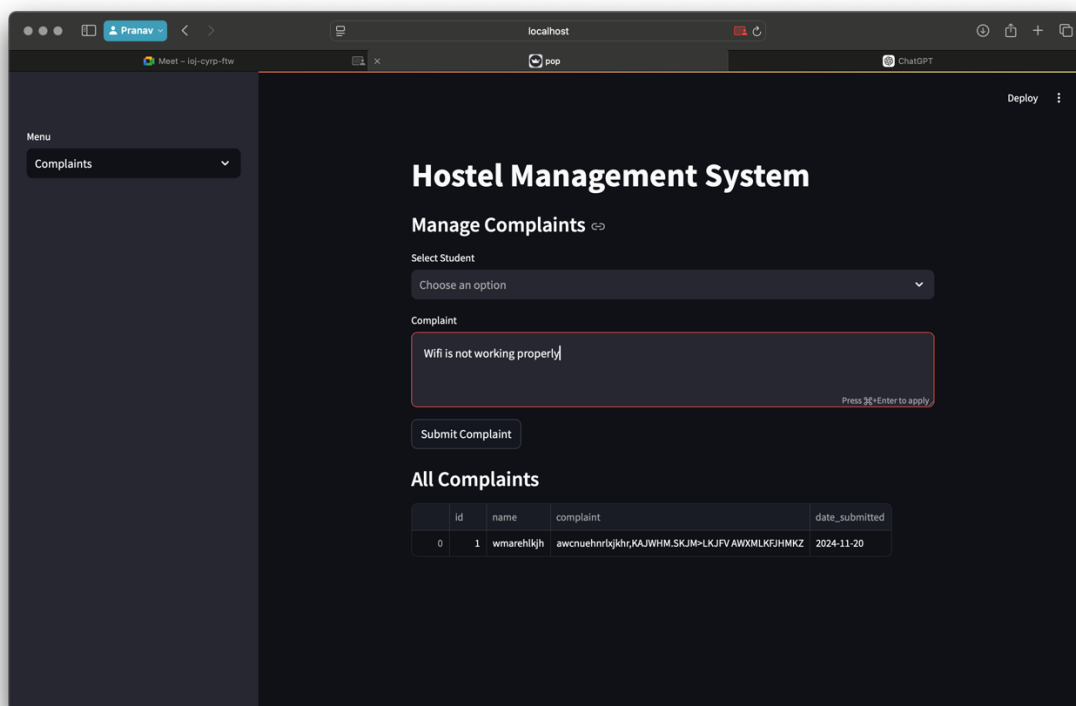
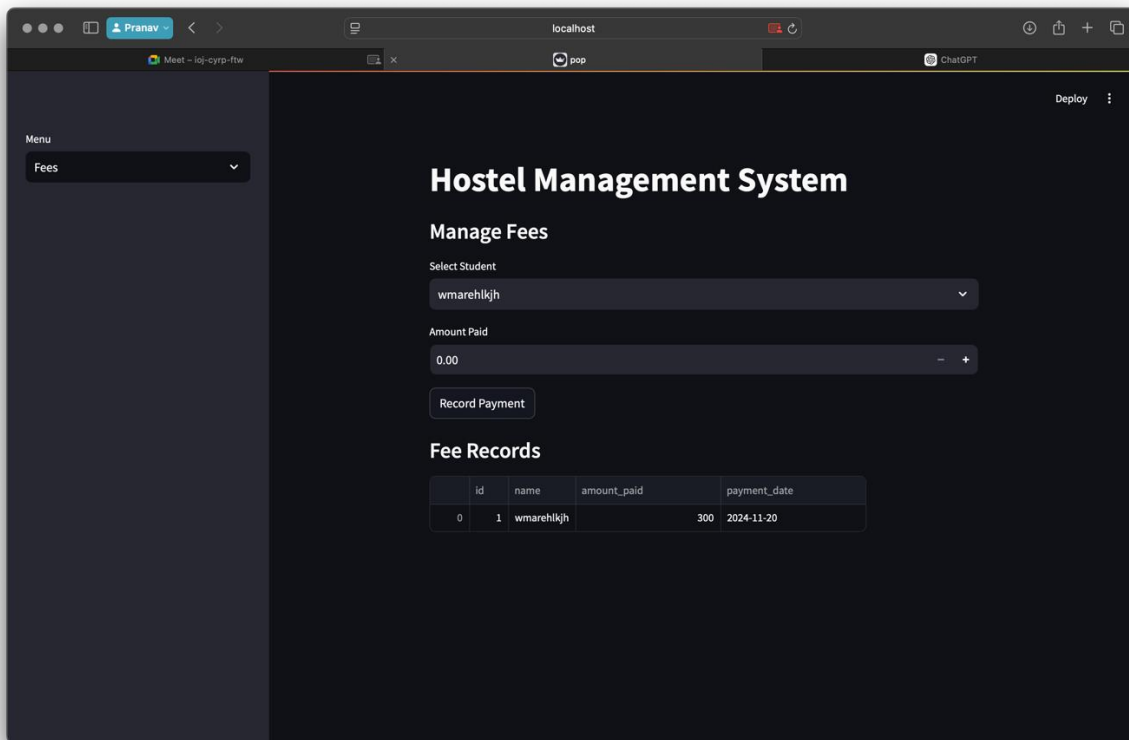
- Use st.dataframe to render tables for students, attendance, fees, and complaints.

CODE:

```
import streamlit as st
import sqlite3
import pandas as pd
from datetime import datetime

# Initialize database
def init_db():
    conn = sqlite3.connect("hostel_management.db")
    cursor = conn.cursor()

    # Student Table
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS students (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
```



```
name TEXT NOT NULL,  
    room_number TEXT NOT NULL,  
    join_date TEXT NOT NULL  
)  
""")
```

Attendance Table

```
cursor.execute("""  
    CREATE TABLE IF NOT EXISTS attendance (  
        id INTEGER PRIMARY KEY AUTOINCREMENT,  
        student_id INTEGER,  
        date TEXT NOT NULL,  
        status TEXT NOT NULL,  
        FOREIGN KEY(student_id) REFERENCES students(id)  
    )  
""")
```

```
cursor.execute("""  
    CREATE TABLE IF NOT EXISTS fees (  
        id INTEGER PRIMARY KEY AUTOINCREMENT,  
        student_id INTEGER,  
        amount_paid REAL,  
        payment_date TEXT NOT NULL,  
        FOREIGN KEY(student_id) REFERENCES students(id)  
    )  
""")
```

Complaints Table

```
cursor.execute("""  
    CREATE TABLE IF NOT EXISTS complaints (  
        id INTEGER PRIMARY KEY AUTOINCREMENT,  
        student_id INTEGER,  
        complaint TEXT NOT NULL,  
        date_submitted TEXT NOT NULL,  
        FOREIGN KEY(student_id) REFERENCES students(id)
```



```
)  
""")
```

```
conn.commit()  
  
return conn
```

```
# Add student
```

```
def add_student(conn, name, room_number):  
  
    cursor = conn.cursor()  
  
    cursor.execute("""  
        INSERT INTO students (name, room_number, join_date)  
        VALUES (?, ?, ?)  
        """, (name, room_number, datetime.now().strftime("%Y-%m-%d")))  
  
    conn.commit()
```

```
# Get students
```

```
def get_students(conn):  
  
    return pd.read_sql("SELECT * FROM students", conn)
```

```
# Record attendance
```

```
def record_attendance(conn, student_id, status):  
  
    cursor = conn.cursor()  
  
    cursor.execute("""  
        INSERT INTO attendance (student_id, date, status)  
        VALUES (?, ?, ?)  
        """, (student_id, datetime.now().strftime("%Y-%m-%d"), status))  
  
    conn.commit()
```

```
# Get attendance
```

```
def get_attendance(conn):  
  
    return pd.read_sql("""  
        SELECT a.id, s.name, a.date, a.status  
        FROM attendance a  
        JOIN students s ON a.student_id = s.id  
        """, conn)
```


Record fee payment

```
def record_fee(conn, student_id, amount_paid):  
    cursor = conn.cursor()  
    cursor.execute("""  
        INSERT INTO fees (student_id, amount_paid, payment_date)  
        VALUES (?, ?, ?)  
        """, (student_id, amount_paid, datetime.now().strftime("%Y-%m-%d")))  
    conn.commit()
```

Get fee records

```
def get_fees(conn):  
    return pd.read_sql("""  
        SELECT f.id, s.name, f.amount_paid, f.payment_date  
        FROM fees f  
        JOIN students s ON f.student_id = s.id  
        """, conn)
```

Add complaint

```
def add_complaint(conn, student_id, complaint):  
    cursor = conn.cursor()  
    cursor.execute("""  
        INSERT INTO complaints (student_id, complaint, date_submitted)  
        VALUES (?, ?, ?)  
        """, (student_id, complaint, datetime.now().strftime("%Y-%m-%d")))  
    conn.commit()
```

Get complaints

```
def get_complaints(conn):  
    return pd.read_sql("""  
        SELECT c.id, s.name, c.complaint, c.date_submitted  
        FROM complaints c  
        JOIN students s ON c.student_id = s.id  
        """, conn)
```



```
# Streamlit app

def main():

    st.title("Hostel Management System")


    # Initialize database

    conn = init_db()


    # Sidebar menu

    menu = ["Students", "Attendance", "Fees", "Complaints"]
    choice = st.sidebar.selectbox("Menu", menu)


    # Manage Students

    if choice == "Students":

        st.subheader("Manage Students")

        with st.form("add_student_form"):

            name = st.text_input("Student Name")
            room_number = st.text_input("Room Number")
            add_button = st.form_submit_button("Add Student")

        add_button:

            if name and room_number:

                add_student(conn, name, room_number)
                st.success(f"Student '{name}' added successfully!")

            else:

                st.error("Please fill all fields.")

        st.subheader("All Students")

        students = get_students(conn)
        st.dataframe(students)


    # Attendance Management

    elif choice == "Attendance":

        st.subheader("Manage Attendance")

        students = get_students(conn)

        if not students.empty:
```



```

    student_id = st.selectbox("Select Student", students["id"].tolist(), format_func=lambda x:
students.loc[students["id"] == x, "name"].values[0])

    status = st.radio("Attendance Status", ["Present", "Absent"])

    if st.button("Record Attendance"):

        record_attendance(conn, student_id, status)

        st.success("Attendance recorded successfully!")

st.subheader("Attendance Records")

attendance = get_attendance(conn)

st.dataframe(attendance)

# Fee Management

elif choice == "Fees":

    st.subheader("Manage Fees")

    students = get_students(conn)

    if not students.empty:

        student_id = st.selectbox("Select Student", students["id"].tolist(), format_func=lambda x:
students.loc[students["id"] == x, "name"].values[0])

        amount_paid = st.number_input("Amount Paid", min_value=0.0, step=100.0)

        if st.button("Record Payment"):

            record_fee(conn, student_id, amount_paid)

            st.success("Fee payment recorded successfully!")

        st.subheader("Fee Records")

        fees = get_fees(conn)

        st.dataframe(fees)

# Complaint Management

elif choice == "Complaints":

    st.subheader("Manage Complaints")

    students = get_students(conn)

    if not students.empty:

        student_id = st.selectbox("Select Student", students["id"].tolist(), format_func=lambda x:
students.loc[students["id"] == x, "name"].values[0])

        complaint = st.text_area("Complaint")

        if st.button("Submit Complaint"):

            add_complaint(conn, student_id, complaint)

            st.success("Complaint submitted successfully!")

```



```
st.subheader("All Complaints")

complaints = get_complaints(conn)

st.dataframe(complaints)

if __name__ == "__main__":
    main()
```

Conclusion:

The Hostel Management System, developed using **Streamlit** and **SQLite**, provides an efficient and user-friendly platform for managing hostel operations. This system streamlines critical functionalities such as room allocation, fee tracking, attendance monitoring, meal plan management, and complaint resolution. With its intuitive interface and centralized data management, the system empowers administrators, students, and staff to operate collaboratively and effectively, ensuring a smooth and organized hostel experience for all stakeholders.