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 Diagramt		
10.	Class Diagram		

1. PREPARING PROBLEM STATEMENT

Aim :

Hostel management faces challenges such as manual booking processes, leading to overbookings, and inefficient payment tracking that results in financial discrepancies. Communication barriers create misunderstandings between management and residents, while fragmented data storage hinders effective resource management. The goal is to develop an integrated management system that automates bookings, streamlines payment processing, facilitates maintenance requests, and improves communication, ultimately enhancing the overall resident experience.

Algorithm :

1. Initialize System
   - Load all necessary data (rooms, bookings, residents, maintenance requests).
   - Set up user roles (admin, staff, resident).
2. User Authentication
   - Prompt user to log in (admin, staff, or resident).
   - Validate credentials and direct user to appropriate dashboard.
3. Booking Management
   1)Search Availability
   - Input check-in and check-out dates.
   - Display available rooms.
   2)Create Booking
   - Collect resident details (name, contact, ID).
   - Confirm room selection.
   - Process payment (validate payment details).
   - Generate booking confirmation.
4. Payment Processing
   - Track payment status (pending, completed).
   - Send reminders for pending payments.
5. Maintenance Management
   1)Submit Request
   - Allow residents to submit maintenance requests.
   2)Track Request
   - Update status of maintenance requests (open, in progress, resolved).
6. Communication System
   - Enable messaging between residents and management.
   - Notify residents of important announcements (e.g., events, maintenance schedules).
7. Data Management and Reporting
   - Generate occupancy reports and financial summaries.
   - Analyze data for resource allocation and improvements.
8. User Logout
   - Log out user and secure session data.

INPUTS FOR THE HOSTEL MANAGEMENT SYSTEM:

1.User Authentication:
- Username
- Password

2. Booking Management:
- Check-in date
- Check-out date
- Number of guests
- Room type preference (if applicable)

3. Resident Information (for new bookings):
- Full name
- Contact number
- Email address
- Identification document (e.g., student ID, passport)

4. Payment Processing:
- Payment method (credit/debit card, online payment gateway)
- Card details (card number, expiry date, CVV)
- Billing address

5. Maintenance Management:
- Maintenance request description
- Room number
- Urgency level (low, medium, high)

6. Communication System:
- Message content (to management)
- Subject line (optional)
- Recipient (specific staff or general management)

7. Feedback/Reviews:
- Resident ID
- Feedback content
- Rating (e.g., 1 to 5 stars)

8. Search and Reporting:
- Date range for occupancy reports
- Room type or status for filtering reports

STAKEHOLDER PROBLEM STATEMENT:

Problem

Hostel management faces significant challenges due to inefficient processes in booking, payment handling, maintenance requests, and communication. These issues lead to overbookings, financial discrepancies, delayed maintenance responses, and poor resident satisfaction. The lack of a centralized system exacerbates these problems, making it difficult to manage operations effectively and meet stakeholder needs.

Background

Hostels are essential for providing affordable accommodation to students, travelers, and workers. However, the increasing demand for hostel services has highlighted the inadequacies of traditional management methods, which often rely on manual processes. Many hostels still use spreadsheets and paper-based systems, leading to errors, miscommunication, and a lack of real-time data. As competition in the hospitality sector grows, hostels must modernize their operations to enhance resident experience and streamline management tasks.

Resilience

To build resilience in the hostel management system, it is crucial to implement an integrated digital platform that automates key processes. This system should:

1. **Enhance Communication:** Establish clear channels for residents to provide feedback and for management to relay important information.
2. **Automate Operations:** Utilize technology to handle bookings, payments, and maintenance requests, reducing the potential for human error.
3. **Provide Real-time Data:** Implement analytics tools to monitor occupancy rates, financial health, and resident satisfaction, allowing for informed decision-making.
4. **Adapt to Change:** Create a flexible system that can quickly adapt to changing circumstances, such as fluctuating demand or emerging technologies.

Objectives

Automate Booking Processes: Streamline reservations with an online booking platform to minimize errors and overbookings.
Streamline Payment Handling: Implement secure payment processing for real-time tracking and reminders.
Enhance Maintenance Management: Centralize maintenance requests for timely issue resolution.
Improve Communication: Establish effective channels for management-resident communication and feedback.
Facilitate Data Management: Develop analytics tools for informed decision-making on occupancy and finances.
Boost Resident Satisfaction: Enhance user experience with personalized services and quick responses.
Ensure Security and Compliance: Protect sensitive data and ensure adherence to data protection regulations.
Support Scalability and Flexibility: Design the system for easy updates and adaptability to changing needs.

Result :

The problem statement was written successfully by following the steps described above.

Ex.NO. 2 : Write the software requirement specification document

Aim :

The aim of this Software Requirement Specification (SRS) document is to define the functional and non-functional requirements for a comprehensive Hostel Management System. This system is designed to automate and streamline the various operations involved in hostel management, improving efficiency and enhancing the overall experience for residents and management alike.

Algorithm :

1.Initialize System
- Load configuration settings and user data (rooms, bookings, residents, maintenance requests).
- Define user roles (admin, staff, resident).

2. User Authentication
- Prompt user to enter username and password.
- Validate credentials.
- If valid, direct user to their respective dashboard; if invalid, display an error message.

3. Booking Management

Search for Available Rooms
- Input check-in and check-out dates.
- Retrieve available rooms based on dates and guest count.
- Display available options to the user.

Create Booking
- Collect resident details (name, contact, ID).
- Confirm selected room and dates.
- Process payment (validate payment details).
- Generate and send booking confirmation to the resident.

View Bookings
- Allow residents and staff to view current and past bookings.

4. Payment Processing

Process Payment
- Capture payment details (method, card info).
- Validate payment and update booking status.
- Notify user of payment status (successful, failed).

Track Payments
- Maintain a record of all transactions for reporting.

5. Maintenance Management
- Submit Maintenance Request
  - Collect details about the issue (description, urgency, room number).
  - Log the request and assign it a unique ID.
- Track Maintenance Requests
  - Allow staff to update the status of requests (open, in progress, resolved).
  - Notify residents of status updates.

6. Communication System
Send Messages
- Enable residents to send messages to management (e.g., inquiries, feedback).
  - Allow management to send announcements or updates to residents.
View Messages
  - Provide a message inbox for residents and staff to view communications.

7. Reporting and Analytics
- Generate reports on occupancy rates, financial performance, and resident feedback.
- Provide visualization tools for data analysis.

8. User Logout
- Securely log out the user and clear session data.

TABLE OF CONTENT:

1. Introduction
- 1.1 Purpose
- 1.2 Scope
- 1.3 Audience

2. Overall Description
- 2.1 Product Perspective
- 2.2 User Classes

3. Functional Requirements
- 3.1 User Authentication
- 3.2 Booking Management
- 3.3 Payment Processing
- 3.4 Maintenance Management
- 3.5 Communication System
- 3.6 Reporting and Analytics

4. Non-Functional Requirements
   - 4.1 Performance
   - 4.2 Security
   - 4.3 Usability

5. System Architecture
   - 5.1 Overview
   - 5.2 Component Diagram

6. Use Cases
   - 6.1 Use Case Diagram
   - 6.2 Key Use Cases

7. User Interface Requirements
   - 7.1 Design Principles
   - 7.2 Mockups

8. Testing Requirements
   - 8.1 Test Strategy
   - 8.2 Test Cases Overview

9. Implementation Plan
   - 9.1 Development Methodology

10. Appendices
    - 10.1 Glossary
    - 10.2 Change Log

SAMPLE OUTPUT:

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a comprehensive overview of the functional and non-functional requirements for the Hostel Management System. This system is intended to automate and streamline hostel operations, improving efficiency and enhancing the overall experience for residents and management.

1.2 Scope
The scope of this SRS includes detailed descriptions of the system's functionalities, user interactions, and performance criteria. It covers key features such as booking management, payment processing, maintenance tracking, and communication tools, aiming to address the specific needs of all stakeholders involved in hostel management.

1.3 Audience
This document is intended for use by project stakeholders, including:

- **Developers:** To understand the system requirements for implementation.
- **Project Managers:** To ensure alignment with project goals and timelines.
- **Quality Assurance Teams:** To design test cases based on specified requirements.
- **End Users (Residents and Management):** To provide insights into the functionalities they can expect from the system.

2. Overall Description

2.1 Product Perspective
The Hostel Management System is a web-based application designed to automate and optimize the operational processes of hostels. It integrates various functionalities such as booking management, payment processing, maintenance requests, and communication tools into a single platform. The system will replace manual processes, enhancing efficiency and providing a better experience for both residents and management.

2.2 Product Functions
Key functionalities of the Hostel Management System include:

- **User Authentication:** Secure login for residents and staff with role-based access control.
- **Booking Management:** Allow residents to search for available rooms, make reservations, and manage their bookings.
- **Payment Processing:** Facilitate secure online payments and track payment statuses.
- **Maintenance Management:** Enable residents to submit maintenance requests and track their status.
- **Communication System:** Provide channels for residents and management to exchange messages and announcements.
- **Reporting and Analytics:** Generate reports on occupancy, financial performance, and resident feedback to support data-driven decisions.

2.3 User Classes and Characteristics

The system will serve the following user classes:

- **Residents:** Users seeking accommodation who will use the system to book rooms, make payments, and submit maintenance requests.
- **Management Staff:** Admin users responsible for overseeing operations, managing bookings, processing payments, and addressing maintenance issues.
- **Support Staff:** Employees who handle maintenance requests and assist residents with inquiries.

2.4 Operating Environment

The Hostel Management System will be a web-based application accessible via standard web browsers (e.g., Chrome, Firefox, Safari) on various devices, including desktops, laptops, tablets, and smartphones. It will be hosted on a secure server with appropriate backups and data protection measures.

2.5 Design and Implementation Constraints

- The system must comply with data protection regulations (e.g., GDPR) to ensure user privacy.
- Integration with existing financial systems may be required for payment processing.
- The application must be developed using scalable technologies to accommodate future growth.
-

2.6 User Documentation

Comprehensive user documentation will be provided, including user manuals, online help, and training materials to assist users in navigating and utilizing the system effectively.

3.System Features

1. User Management
   - Admin dashboard, guest profiles, role-based access.

2. Room Management
   - Room listings, real-time booking, status tracking.

3. Check-in/Check-out
   - Self check-in kiosks, express check-out, ID verification.

4. Billing and Payments
   - Multiple payment options, invoice generation, payment history.

5. Reporting and Analytics
   - Occupancy analytics, financial reports, guest demographics.

6. Communication Tools
   - In-app messaging, email notifications, guest feedback system.

7. Amenities Management
   - Service directory, booking additional services, amenity feedback.

8. Maintenance Management
   - Issue reporting, scheduled maintenance, vendor coordination.

9. Security Features
   - Access control, data encryption, emergency protocols.

10. Mobile Accessibility
    - Responsive design, dedicated guest mobile app.

11.Integration Capabilities
    - Third-party integrations, API availability.

12. Multi-Language and Currency Support
    - Localization options, currency conversion.

13. Customer Relationship Management (CRM)
    - Guest profiles, loyalty programs, targeted marketing.

14. Event Management
    - Event scheduling, RSVP system, promotional campaigns.

4. External Interface Requirements

1.User Interface (UI)
- Web Application: A responsive and user-friendly web interface for both admin and guests.
- Mobile Application: Native or hybrid app for iOS and Android to manage bookings and services on-the-go.

2. API Interfaces
- Booking API: Interface for integrating with external booking platforms (e.g., OTAs).
- Payment Gateway Integration: Support for various payment gateways (e.g., PayPal, Stripe) to handle online payments securely.
- CRM Integration: Connect with third-party CRM systems for customer data management.

3. Database Interfaces
- Database Access: Secure access to a relational database (e.g., MySQL, PostgreSQL) for data storage and retrieval.
- Backup and Recovery: Automated backup processes with interfaces for data recovery.

4. Communication Interfaces
- Email Service Integration: SMTP or third-party email services (e.g., SendGrid) for sending notifications and confirmations.
- Messaging Services: Integration with messaging platforms (e.g., Twilio) for SMS notifications.

5. Reporting and Analytics
- Data Export: Capability to export reports in various formats (CSV, PDF) for external analysis.
- Business Intelligence Tools: Interfaces for connecting with BI tools (e.g., Tableau, Power BI) for advanced analytics.

6. Security Interfaces
- Authentication Protocols: Support for OAuth, JWT, or other authentication standards for secure access.
- Encryption Standards: Implementation of SSL/TLS for secure data transmission.

7. External System Integration
- Social Media Integration: APIs for social media platforms for sharing promotions and guest engagement.
- Event Management Systems: Interfaces to connect with external event management tools.

5. **Non-Functional Requirements**

1. Performance

- **Response Time**: The system should respond to user actions within 2 seconds.
- **Scalability**: Capable of handling up to 1,000 simultaneous users without performance degradation.

2. Reliability

- **Availability**: The system must be available 99.9% of the time, excluding scheduled maintenance.
- **Error Recovery**: The system should automatically recover from failures within 5 minutes.

3. Security

- **Data Protection**: All sensitive data must be encrypted in transit and at rest.
- **User Authentication**: Implement strong password policies and multi-factor authentication for user accounts.

4. Usability

- **User Interface**: The UI should be intuitive, with a maximum of three clicks required to access any feature.
- **Accessibility**: The system should comply with WCAG 2.1 standards to ensure usability for all users, including those with disabilities.

5. Maintainability

- **Code Modularity**: The codebase should be modular to facilitate updates and maintenance.
- **Documentation**: Comprehensive documentation must be provided for system architecture and API endpoints.

6. Compatibility

- **Browser Support**: The web application must support the latest versions of major browsers (Chrome, Firefox, Safari, Edge).
- **Mobile Compatibility**: The mobile app should function on the latest versions of iOS and Android.

7. Interoperability

- **Third-Party Integration**: The system should be able to integrate seamlessly with external APIs for payment processing, CRM, and reporting tools.

8. Localization

- **Multi-Language Support**: The system should support at least five languages for user interfaces.
- **Currency Conversion**: It must accommodate multiple currencies for international guests.

9. Data Management

- **Backup Frequency**: Data backups must occur daily, with the ability to restore to any point within the last 30 days.
- **Data Retention**: Guest data must be retained for a minimum of 5 years to comply with regulations.

10. Legal and Compliance

- **GDPR Compliance**: The system must comply with the General Data Protection Regulation (GDPR) for handling personal data.
- **Local Regulations**: Ensure compliance with local laws regarding data privacy and financial transactions.

Result:

The SRS was made successfully by following the steps described above.

3. ENTITY RELATIONSHIP MODEL

Aim :

TO DRAW AN ENTITY RELATIONAL MODEL FOR HOSTEL MANAGEMENT

Algorithm :

Step 1: Mapping of Regular Entity Types
- Entities: Identify the main entities in the system.
  - Guest: Attributes: GuestID (PK), Name, Email, Phone, Address.
  - Room: Attributes: RoomID (PK), RoomType, Price, Capacity, Status.
  - Staff: Attributes: StaffID (PK), Name, Role, Email, Phone.
  - Booking: Attributes: BookingID (PK), GuestID (FK), RoomID (FK), CheckInDate, CheckOutDate.

Step 2: Mapping of Weak Entity Types
- Weak Entities: Identify any weak entities that depend on a strong entity for their existence.
  - Payment: Attributes: PaymentID (PK), BookingID (FK), Amount, PaymentDate.
    - Dependency: The Payment entity depends on the Booking entity.

Step 3: Mapping of Binary 1:1 Relation Types
- Relations: Identify relationships where one entity is related to one and only one instance of another entity.
  - Staff to Room Assignment: Each room is assigned to one staff member for management.
    - Relationship: Room (RoomID) 1:1 Staff (StaffID)

Step 4: Mapping of Binary 1:N Relationship Types
- Relationships: Identify relationships where one entity is related to many instances of another entity.
  - Guest to Booking: One guest can make multiple bookings.
    - Relationship: Guest (GuestID) 1:N Booking (BookingID)
  - Room to Booking: One room can be booked multiple times, but not concurrently.
    - Relationship: Room (RoomID) 1:N Booking (BookingID)

Step 5: Mapping of Binary M:N Relationship Types
- Relationships: Identify many-to-many relationships.
  - Guest to Amenities: Guests can use multiple amenities, and each amenity can be used by multiple guests.
    - Mapping: Create a junction table.
      - Junction Table: GuestAmenities: Attributes: GuestID (FK), AmenityID (FK).
        - Relationships: Guest (GuestID) M:N Amenity (AmenityID).

Step 6: Mapping of Multivalued Attributes
- Attributes: Identify attributes that can hold multiple values for a single entity.
  - Room Amenities: A room can have multiple amenities (e.g., Wi-Fi, TV).
    - Mapping:
      - Separate Entity: Amenities: Attributes: AmenityID (PK), AmenityName.
      - Junction Table: RoomAmenities: Attributes: RoomID (FK), AmenityID (FK).

Input :

Step 1: Mapping of Regular Entity Types

- **Guest**: A guest has a unique ID, name, email, and phone number.
- **Room**: A room has a unique ID, type, price, capacity, and availability status.
- **Staff**: A staff member has a unique ID, name, role, and contact information.
- **Booking**: A booking has a unique ID, references a guest and a room, and includes check-in and check-out dates.

Step 2: Mapping of Weak Entity Types

- **Payment**: A payment is linked to a specific booking and includes an amount and payment date.

Step 3: Mapping of Binary 1:1 Relationship Types

- **Staff to Room Assignment**: Each room is assigned to one staff member for management.

Step 4: Mapping of Binary 1
Relationship Types

- **Guest to Booking**: A guest can have multiple bookings.
- **Room to Booking**: A room can be booked for multiple periods but not concurrently.

Step 5: Mapping of Binary M
Relationship Types

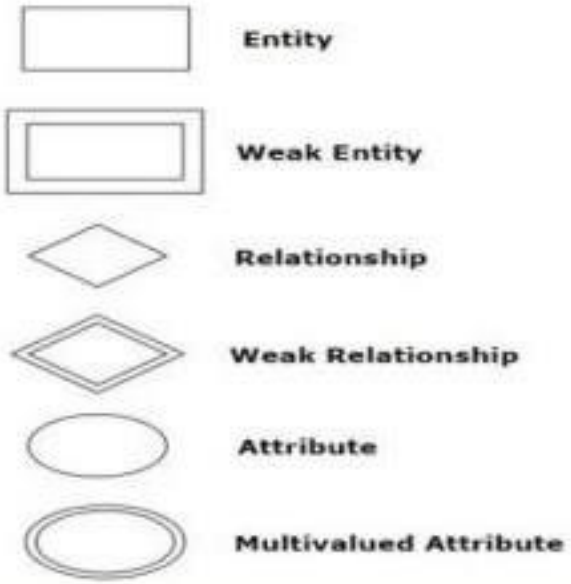
- **Guest to Amenities**: A guest can use multiple amenities, and each amenity can be used by multiple guests.

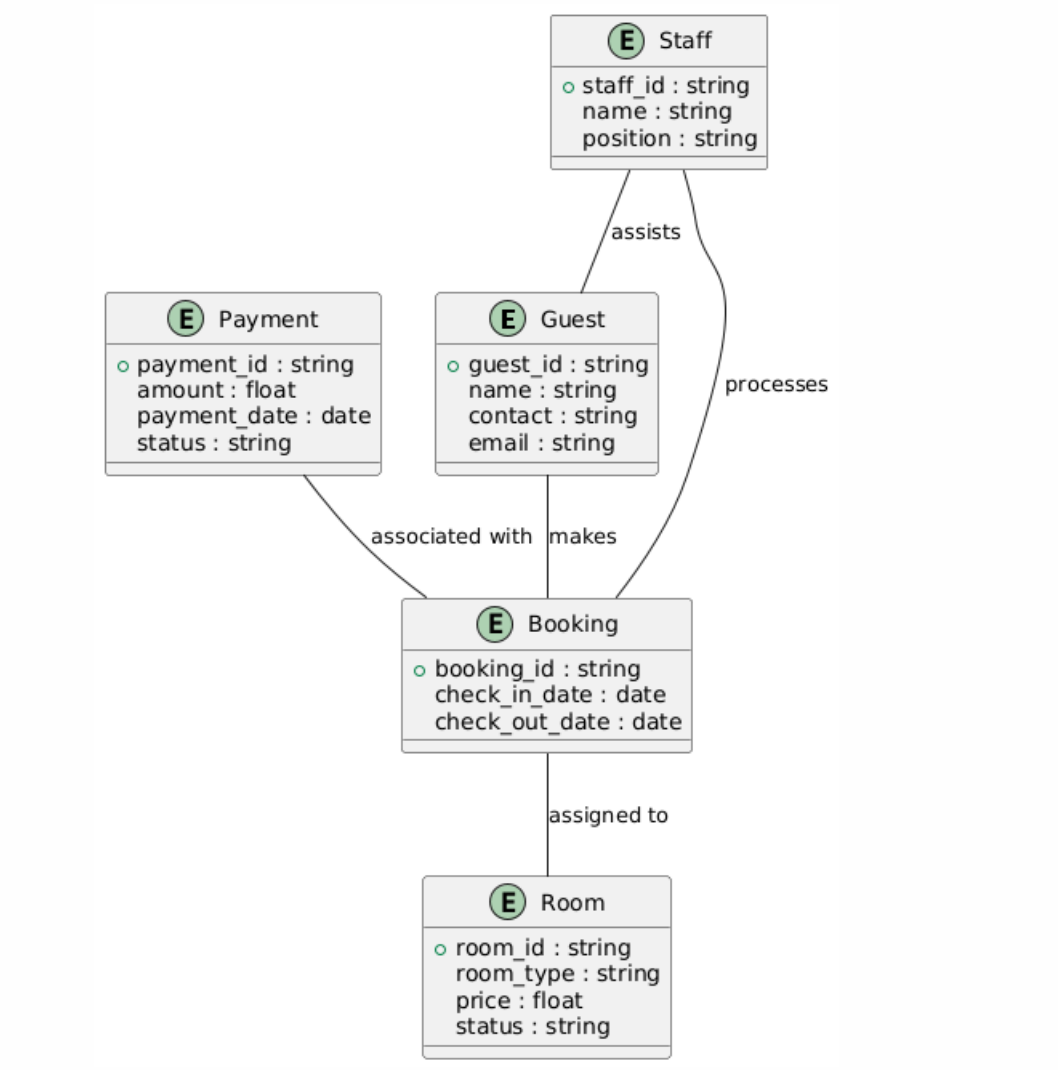
Step 6: Mapping of Multivalued Attributes

- **Room Amenities**: A room can have multiple amenities (e.g., Wi-Fi, Air Conditioning), so a room is linked to several amenities.

ER DIAGRAM:

SYMBOLS:





Result: The entity relationship diagram was made successfully by following the steps describedabove.

.

4. DATA FLOW DIAGRAM

Aim :

To Draw the Data Flow Diagram for Hostel management system and List the Modules in the Application.

Algorithm :

Open the Tool: Open Visual Paradigm or Lucidchart and choose a DFD template.
Name the Diagram: Name it "Hostel Management System DFD."
Add External Entities: Add entities like **Guest** and **Staff**.
Add Processes: Add key processes like **Room Booking** and **Payment Processing**.
Add Data Stores: Include data stores like **Room Inventory** and **Guest Database**.
Add Data Flows: Draw arrows to show data flow between entities, processes, and stores.
Name the Data Flows: Label data flows like **Booking Details** and **Payment Information**.
Customize: Adjust colors, fonts, and labels for clarity.
Title and Share: Add a title and share the DFD

Input :

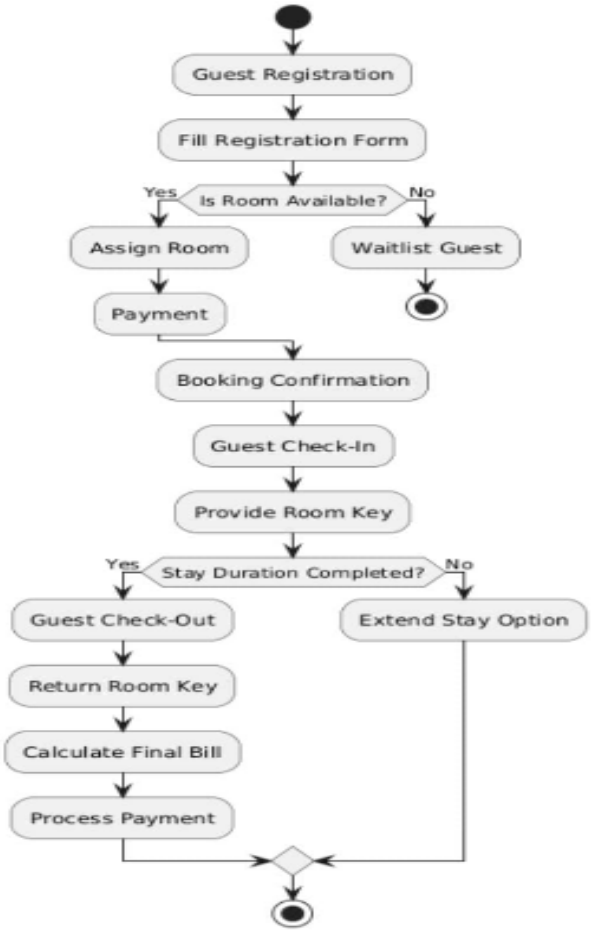
1. External Entities:
- Guest: Guest ID, Name, Booking Request, Payment Info.
- Staff: Staff ID, Room Assignments.

2. Processes:
- Room Booking: Booking Request (Room Type, Dates).
- Payment Processing: Payment Info (Amount, Method).

3. Data Stores:
- Room Inventory: Room Details (ID, Availability, Price).
- Guest Database: Guest Info (ID, Name, History).
- Payment Records: Payment Details (Booking ID, Amount).

4. Data Flows:
- Guest → Room Booking: Booking Info.
- Room Booking → Room Inventory: Room Assignment.
- Guest → Payment Processing: Payment Info.
- Payment Processing → Payment Records: Payment Confirmation.

5. Customization:
- Title: "Hostel Management System DFD."
- Colors/Fonts: Distinguish Processes, Data Stores, and Entities.



Result: The Data Flow diagram was made successfully by following the steps described above.

5. USE CASE DIAGRAM

Aim :

To Draw the Use Case Diagram for Hostel management system

Algorithm :

Step 1: Identify Actors
- Actors: Identify the key external users interacting with the system.
  - Guest
  - Staff
  - Admin

Step 2: Identify Use Cases
- Use Cases: Define the key functionalities the system should provide.
  - For Guest: Book Room, Make Payment, View Room Availability.
  - For Staff: Assign Room, Check-in Guest, Manage Bookings.
  - For Admin: Manage Rooms, View Reports, Update Staff Details.

Step 3: Connect Actors and Use Cases
- Draw lines to connect each actor with the appropriate use cases.
  - Guest: Connected to Book Room, Make Payment.
  - Staff: Connected to Assign Room, Check-in Guest.
  - Admin: Connected to Manage Rooms, View Reports.

Step 4: Add System Boundary
- Draw a boundary box around the system to represent the scope.
  - The system boundary defines which use cases belong to the Hostel Management System.

Step 5: Define Relationships
- Define relationships such as Include or Extend if applicable.
  - Make Payment may include Check-out Process.
  - View Room Availability may extend Book Room.

Step 6: Review and Refine
- Review the diagram to ensure that all major actors and use cases are captured.
  - Refine use case names and relationships as needed.

Step 7: Validate
- Validate the diagram with stakeholders to ensure the system covers all required functionality.
  - Ensure that the actors' needs and system's use cases align.

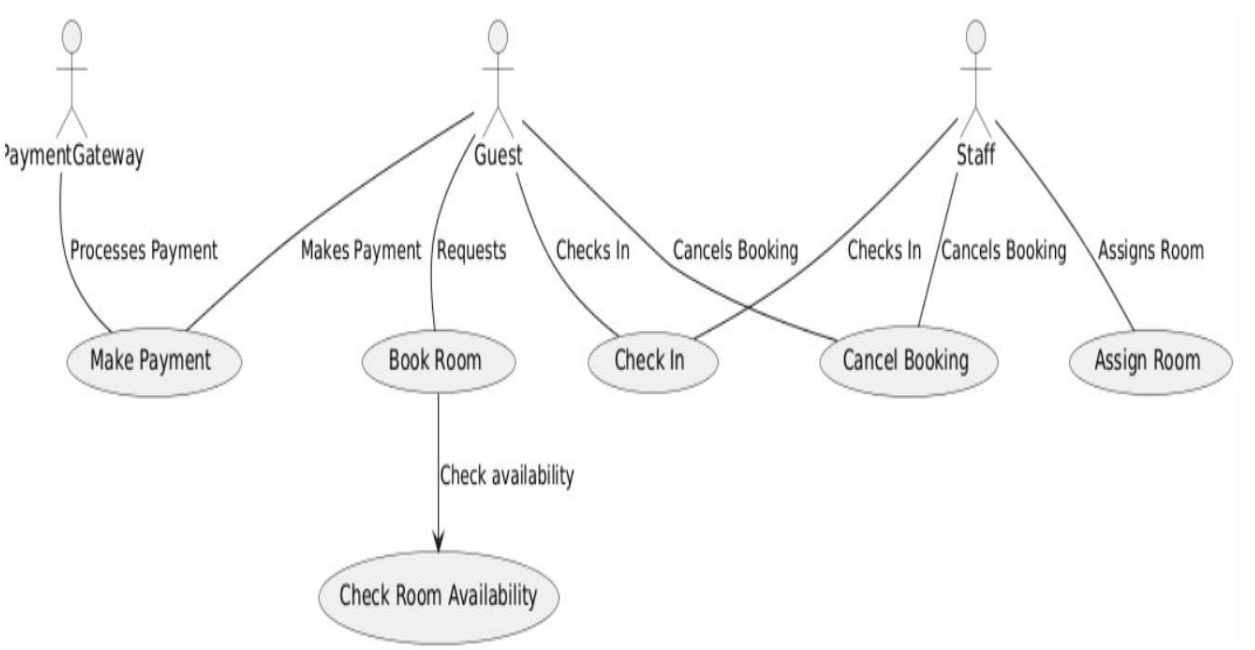
Input :

1. Actors
- Guest
- Staff
- Admin

2. Use Cases
- Guest
  - Book Room
  - Make Payment
  - View Room Availability
- Staff
  - Assign Room
  - Check-in Guest
  - Manage Bookings
- Admin
  - Manage Rooms
  - View Reports
  - Update Staff Details

3. Relations
- Guest → Book Room
- Guest → Make Payment
- Staff → Assign Room
- Staff → Check-in Guest
- Admin → Manage Rooms
- Admin → View Reports
- Admin → Update Staff Details

Output :


PaymentGateway
Guest
Staff
Processes Payment
Makes Payment
Requests
Checks In
Cancels Booking
Checks In
Cancels Booking
Assigns Room
Make Payment
Book Room
Check In
Cancel Booking
Assign Room
Check availability
Check Room Availability

Result :

The use case diagram has been created successfully by following the steps given.

6. ACTIVITY DIAGRAM

Aim:

To Draw the activity Diagram FOR HOSTEL MANAGEMENT SYSTEM

Algorithm :

Step 1: Identify the Initial State and Final States
- Initial State: The system starts in the "Idle" state (no bookings or actions).
- Final State: The system ends in the "Booking Completed" or "Payment Confirmed" state.

Step 2: Identify the Intermediate Activities Needed
- Booking State: Guest selects a room.
- Payment State: Guest makes a payment.
- Check-in State: Staff checks in the guest after payment is confirmed.

Step 3: Identify the Conditions or Constraints
- Condition: A guest can only be checked in after a successful payment.
- Constraint: Rooms can only be booked if available.

Step 4: Draw the Diagram with Appropriate Notations
- Draw states as rectangles with rounded corners.
- Use arrows to show transitions between states based on conditions or actions (e.g., from "Booking" to "Payment").
- Label the transitions with the conditions or events triggering them (e.g., "Payment Success").

Inputs :

1. Activities:
- Booking Room: Guest selects room and provides booking details.
- Making Payment: Guest enters payment details and completes payment.
- Room Assignment: Staff assigns a room to the guest.
- Check-in Process: Staff checks the guest into the room after payment confirmation.

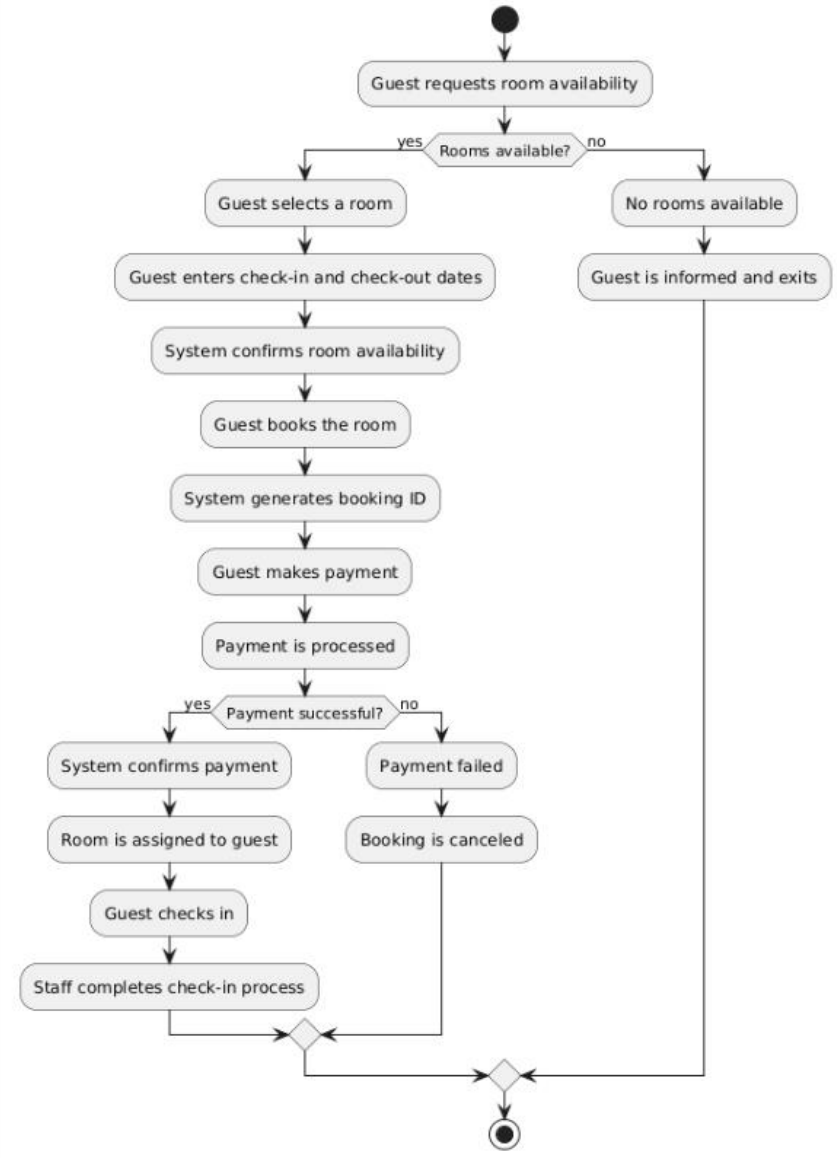
2. Decision Points:
- Payment Success?: After the guest makes a payment, the system decides if the payment is successful or failed.
- Room Availability?: Check if the selected room is available before booking.

3. Guards:
- Payment Validity: Guard to ensure that payment is successful before proceeding with check-in.
- Room Availability: Guard to ensure that the room is available before booking can proceed.

4. Parallel Activities:
- Booking and Payment: These can be handled in parallel (e.g., room availability check can happen while the guest is making the payment).
- Room Assignment and Check-in: While the staff assigns a room, the guest can proceed with the check-in process after successful payment.

5. Conditions:
- Successful Payment: Transition from Booking to Check-in happens only if the payment is successful.
- Available Room: Booking can only proceed if a room is available for the guest.

Output :



Result : The Activity diagram has been created successfully by following the steps given.

7. STATE CHART DIAGRAM

Aim :

To Draw the State Chart Diagram for HOSTEL MANAGEMENT SYSTEM

Algorithm :

Step 1: Identify the Important Objects to Be Analyzed
- Guest: Tracks the guest's booking and check-in status.
- Room: Tracks the room availability, booking, and occupancy status.
- Staff: Tracks the status of staff activities (e.g., room assignment, check-in).

Step 2: Identify the States
- Guest States:
  - Idle: No booking made.
  - Room Booked: Booking confirmed, but not yet paid.
  - Checked-in: Guest has paid and checked into the room.

- Room States:
  - Available: Room is free and can be booked.
  - Booked: Room is booked but not yet occupied.
  - Occupied: Room is currently being used by a guest.

- Staff States:
  - Idle: No current assignments.
  - Assigned: Staff is managing the room or booking process.
  - On Duty: Staff is actively checking in guests or handling bookings.

Step 3: Identify the Events
- For Guest:
  - Book Room: Guest makes a booking.
  - Make Payment: Guest completes payment.
  - Check-in: Guest is checked into the room.

- For Room:
  - Room Booked: Room status changes after a booking.
  - Room Occupied: Room becomes occupied after check-in.

- For Staff:
  - Assign Room: Staff assigns a room to the guest.
  - Process Payment: Staff confirms the payment.
  - Check-in Guest: Staff checks in the guest after payment confirmation.

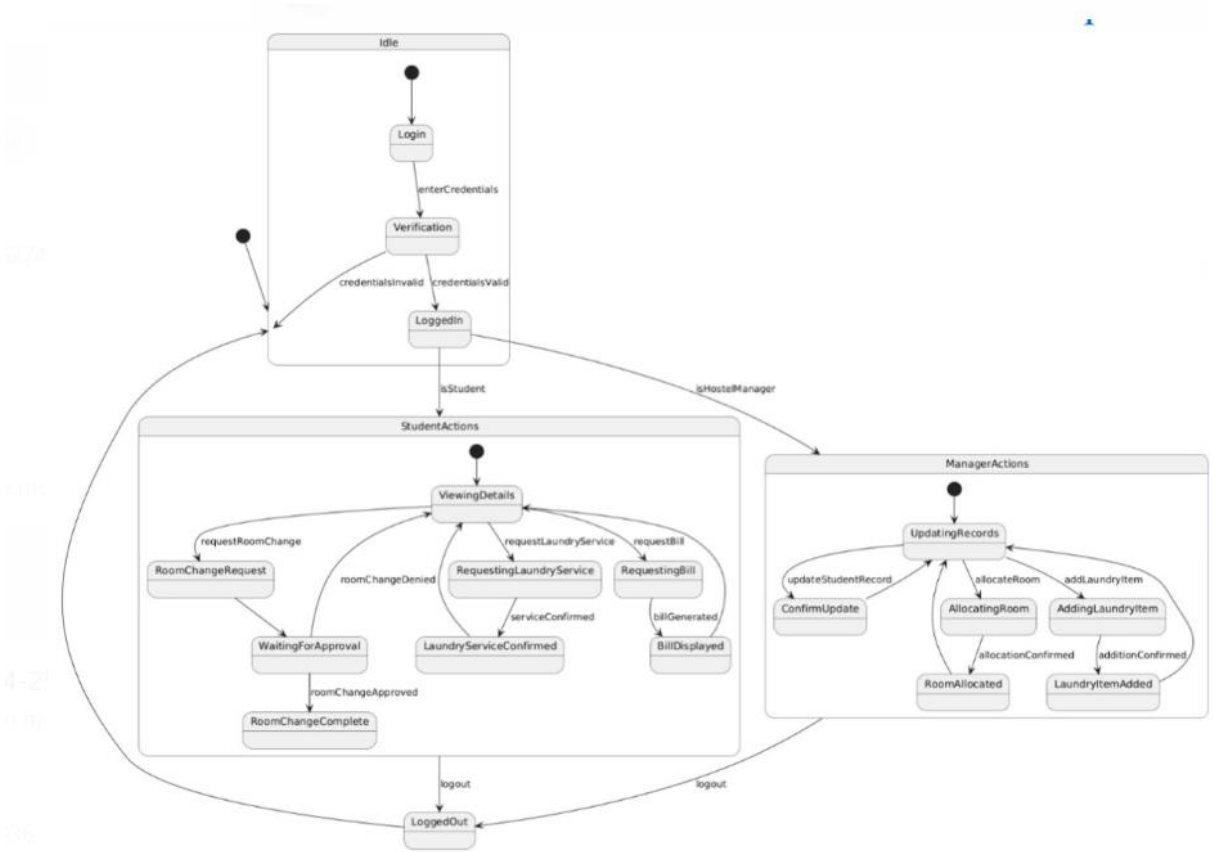
Inputs :

1. Objects:
- Guest
- Room
- Staff

2. States:
- Guest: Idle, Room Booked, Checked-in
- Room: Available, Booked, Occupied
- Staff: Idle, Assigned, On Duty

3. Events:
- Guest: Make Booking, Make Payment, Check-in
- Room: Room Booked, Room Occupied
- Staff: Assign Room, Process Payment, Check-in Guest

Output :



Result : The State Chart diagram has been created successfully by following the steps given.

8. SEQUENCE DIAGRAM

Aim :

To Draw the Sequence Diagram for HOSTEL MANAGEMENT SYSTEM

Algorithm :

Step 1: Identify the Scenario
- Define a specific scenario like Guest Booking a Room or Staff Checking-in a Guest.

Step 2: List the Participants
- Guest
- System
- Staff
- Payment Gateway

Step 3: Define Lifelines
- Create vertical dashed lines for each participant (Guest, Staff, System, Payment Gateway).

Step 4: Arrange Lifelines
- Place participants in the order of interaction, from left to right.

Step 5: Add Activation Bars
- Draw activation bars (rectangles) to show when an object is active in the process.

Step 6: Draw Messages
- Add horizontal arrows to show messages exchanged between participants, like Book Room or Process Payment.

Step 7: Include Return Messages
- Add dashed arrows for return messages (e.g., Payment Confirmation, Room Assigned).

Step 8: Indicate Timing and Order
- Ensure that the sequence of interactions is shown chronologically from top to bottom.

Step 9: Include Conditions and Loops
- Use if conditions or loops (e.g., "If payment successful, then check-in").

Step 10: Consider Parallel Execution
- If activities occur in parallel, represent them with separate lifelines.

Step 11: Review and Refine
- Check that all interactions are accurately captured and in correct sequence.

Step 12: Add Annotations and Comments
- Include brief comments or notes to clarify specific actions.

Step 13: Document Assumptions and Constraints
- Record any assumptions or constraints affecting the diagram (e.g., "Payment must be completed before check-in").

Step 14: Use a Tool to Create a Neat Sequence Diagram
- Use a diagramming tool like Lucidchart, Visual Paradigm, or UMLet to create a polished diagram.

Inputs :

1. Objects Taking Part in the Interaction:
- Guest
  - Interacts with the system to book a room, make payments, and check in.

- System
  - Manages room availability, bookings, payment processing, and check-in operations.

- Staff
  - Manages room assignments, guest check-ins, and other hostel-related tasks.

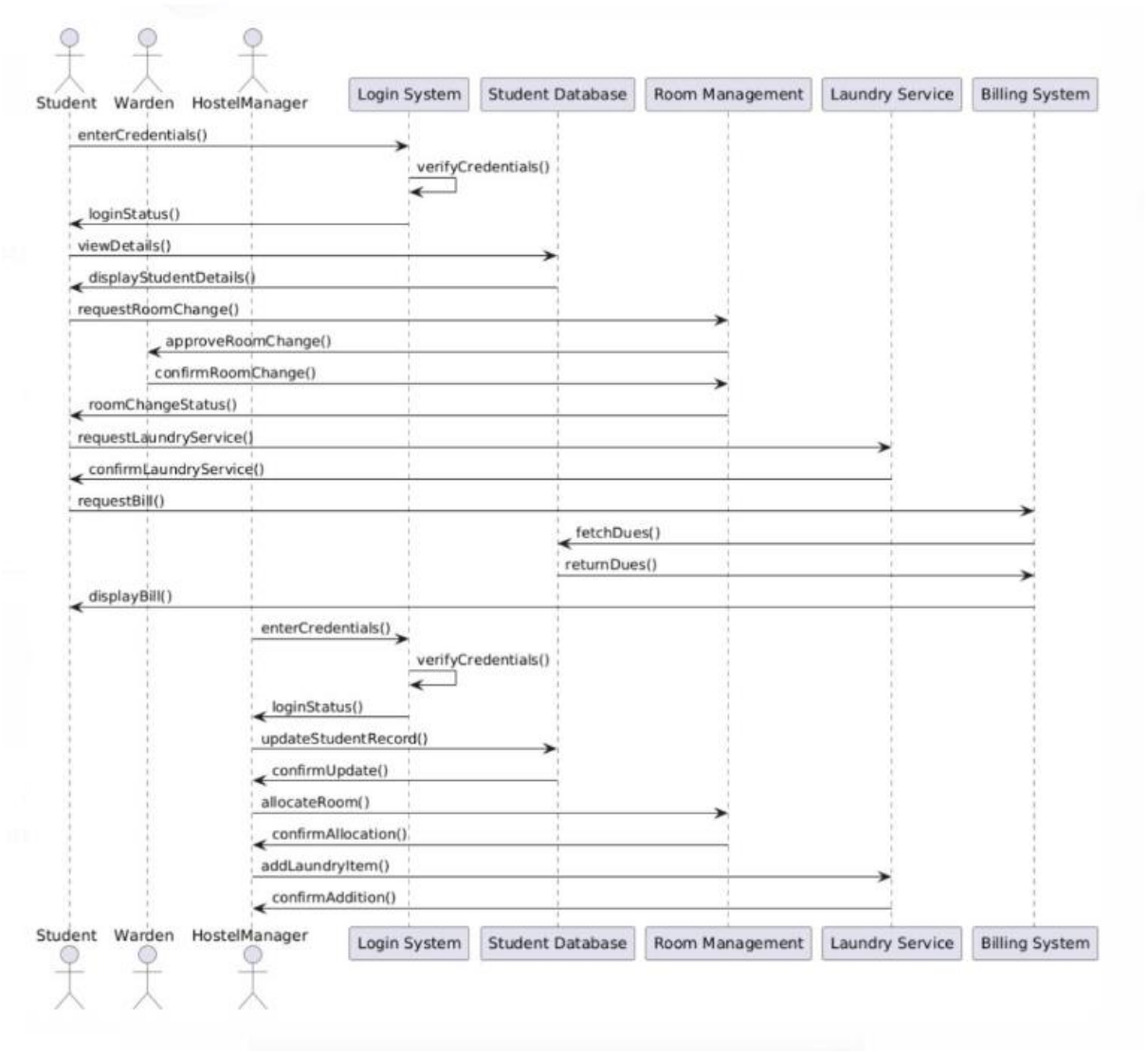
- Payment Gateway
  - Processes payment transactions.

2. Message Flows Among the Objects:
- Guest → System: Request Room Availability
- System → Guest: Return Available Rooms
- Guest → System: Book Room
- System → Staff: Assign Room
- Guest → Payment Gateway: Make Payment
- Payment Gateway → System: Payment Confirmation
- System → Staff: Confirm Booking and Check-in
- Staff → Guest: Room Assigned and Check-in Completed

3. The Sequence in Which the Messages Are Flowing:
- 1st: Guest → System: Request room availability.
- 2nd: System → Guest: Return available rooms.
- 3rd: Guest → System: Book the room.
- 4th: System → Staff: Assign room to the guest.
- 5th: Guest → Payment Gateway: Make payment.
- 6th: Payment Gateway → System: Confirm payment.
- 7th: System → Staff: Confirm room booking and check-in.
- 8th: Staff → Guest: Complete check-in and provide room details.

4. Object Organization:
- Left to Right Arrangement:
  - Guest (leftmost), followed by System, then Staff, and Payment Gateway (rightmost).
- Lifelines: Vertical dashed lines represent the timeline for each object, and activation bars (rectangles) show the active period for each object during the process.

Output :



Result :

The Sequence diagram has been created successfully by following the steps given.

9. COLLABORATION DIAGRAM

Aim :

To Draw the Collaboration Diagram for HOSTEL MANAGEMENT SYSTEM

Algorithm :

Step 1: Identify Objects/Participants
- Guest
- System
- Staff
- Payment Gateway

Step 2: Define Interactions
- Guest → System: Request Room Availability
- System → Guest: Return Available Rooms
- Guest → System: Book Room
- System → Staff: Assign Room
- Guest → Payment Gateway: Make Payment
- Payment Gateway → System: Payment Confirmation
- System → Staff: Confirm Booking and Check-in
- Staff → Guest: Complete Check-in

Step 3: Add Messages
- Assign numbered messages to each interaction:
  - Message 1: Request Room Availability
  - Message 2: Return Available Rooms
  - Message 3: Book Room
  - Message 4: Assign Room
  - Message 5: Make Payment
  - Message 6: Payment Confirmation
  - Message 7: Confirm Booking and Check-in
  - Message 8: Complete Check-in

Step 4: Consider Relationships
- Draw lines connecting the objects (e.g., Guest → System or System → Staff).
- Indicate message flow with numbered arrows between the objects.

Step 5: Document the Collaboration Diagram with Explanations
- Label each message clearly with descriptions.
- Add annotations where necessary to clarify the flow of interactions or to explain specific business rules (e.g., "Payment must be confirmed before check-in").

Inputs :

1. Objects Taking Part in the Interaction:
- Guest
  - The user who interacts with the system to book rooms, make payments, and check in.

- System
  - Manages room availability, bookings, payment processing, and check-ins.

- Staff
  - The staff member who assigns rooms, checks in guests, and manages bookings.

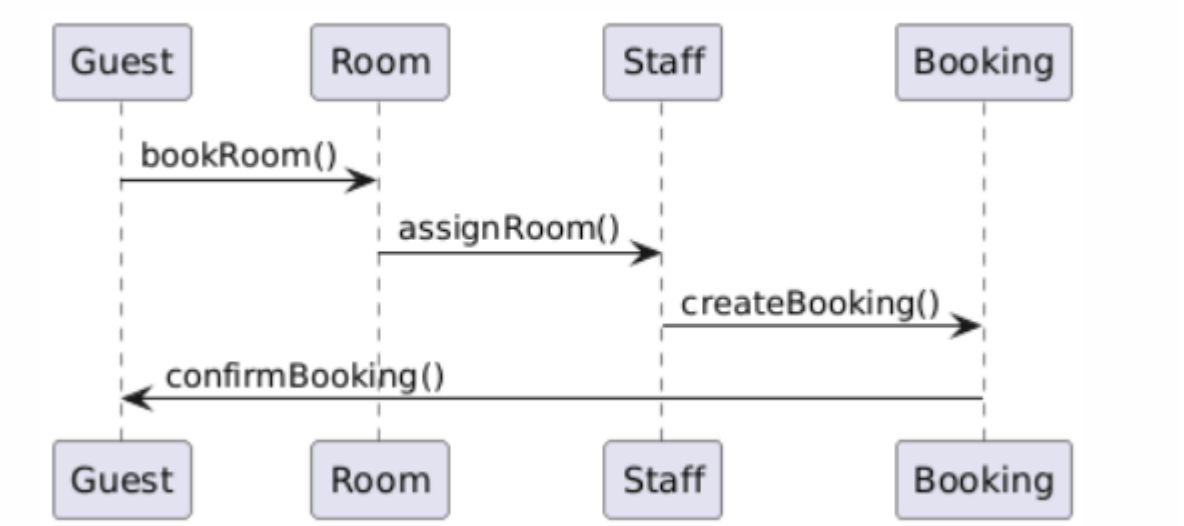
- Payment Gateway
  - Handles the payment transactions for bookings.

2. Message Flows Among the Objects:
- Guest → System: Request Room Availability
- System → Guest: Return Available Rooms
- Guest → System: Book Room
- System → Staff: Assign Room
- Guest → Payment Gateway: Make Payment
- Payment Gateway → System: Payment Confirmation
- System → Staff: Confirm Booking and Check-in
- Staff → Guest: Complete Check-in

3. The Sequence in Which the Messages Are Flowing:
1. Message 1: Guest → System: Request Room Availability
2. Message 2: System → Guest: Return Available Rooms
3. Message 3: Guest → System: Book Room
4. Message 4: System → Staff: Assign Room
5. Message 5: Guest → Payment Gateway: Make Payment
6. Message 6: Payment Gateway → System: Payment Confirmation
7. Message 7: System → Staff: Confirm Booking and Check-in
8. Message 8: Staff → Guest: Complete Check-in

4. Object Organization:
- Left to Right Arrangement:
  - Guest (leftmost), followed by System, then Staff, and Payment Gateway (rightmost).
- Lifelines: Vertical dashed lines represent the timeline for each object.
- Message Flow: Draw horizontal arrows between objects to show the flow of messages.
- Numbering: Number the messages to indicate the order of interaction

Output :



Result :

The Collaboration diagram has been created successfully by following the steps given.

10. CLASS DIAGRAM

Aim:

To Draw the Class Diagram for any project

Algorithm :

Step 1: Identify Classes
- Guest
- Room
- Staff
- Booking
- Payment

Step 2: List Attributes and Methods
- Guest:
  - Attributes: guestID, name, contact, bookingHistory
  - Methods: bookRoom(), makePayment(), checkIn()

- Room:
  - Attributes: roomID, roomType, availability, price
  - Methods: assignRoom(), checkAvailability()

- Staff:
  - Attributes: staffID, name, position
  - Methods: assignRoom(), checkInGuest()

- Booking:
  - Attributes: bookingID, roomID, guestID, checkInDate, checkOutDate
  - Methods: createBooking(), cancelBooking()

- Payment:
  - Attributes: paymentID, amount, paymentDate, paymentStatus
  - Methods: processPayment(), confirmPayment()

Step 3: Identify Relationships
- Guest → Booking: A guest makes a booking.
- Booking → Room: A booking is associated with a room.
- Staff → Guest: Staff assists the guest with check-in.
- Booking → Payment: A booking is linked to a payment.

Step 4: Create Class Boxes
- Draw rectangular boxes for each class, label them accordingly (Guest, Room, Staff, etc.).

Step 5: Add Attributes and Methods

- Inside each class box, list the attributes and methods.
Step 6: Draw Relationships
- Use lines to connect classes that have relationships (e.g., Guest → Booking, Booking → Room).
Step 7: Label Relationships
- Add appropriate relationship labels (e.g., "has", "makes", "assigned to").
Step 8: Review and Refine
- Ensure all relevant classes, attributes, methods, and relationships are captured correctly.
Step 9: Use Tools for Digital Drawing
- Use diagramming tools like Lucidchart, Visual Paradigm, or UMLet to create a clean, professional class diagram.

Inputs :

1. Class Name:
- Guest
- Room
- Staff
- Booking
- Payment

2. Attributes:

- Guest:
  - guestID (String)
  - name (String)
  - contact (String)
  - bookingHistory (List of Bookings)

- Room:
  - roomID (String)
  - roomType (String)
  - availability (Boolean)
  - price (Float)

- Staff:
  - staffID (String)
  - name (String)
  - position (String)

- Booking:
  - bookingID (String)
  - roomID (String)
  - guestID (String)
  - checkInDate (Date)
  - checkOutDate (Date)

- Payment:
  - paymentID (String)
  - amount (Float)
  - paymentDate (Date)
  - paymentStatus (String)

3. Methods:

- Guest:
  - bookRoom()
  - makePayment()
  - checkIn()

- Room:
  - assignRoom()
  - checkAvailability()

- Staff:
  - assignRoom()
  - checkInGuest()

- Booking:
  - createBooking()
  - cancelBooking()

- Payment:
  - processPayment()
  - confirmPayment()

4. Visibility Notation:

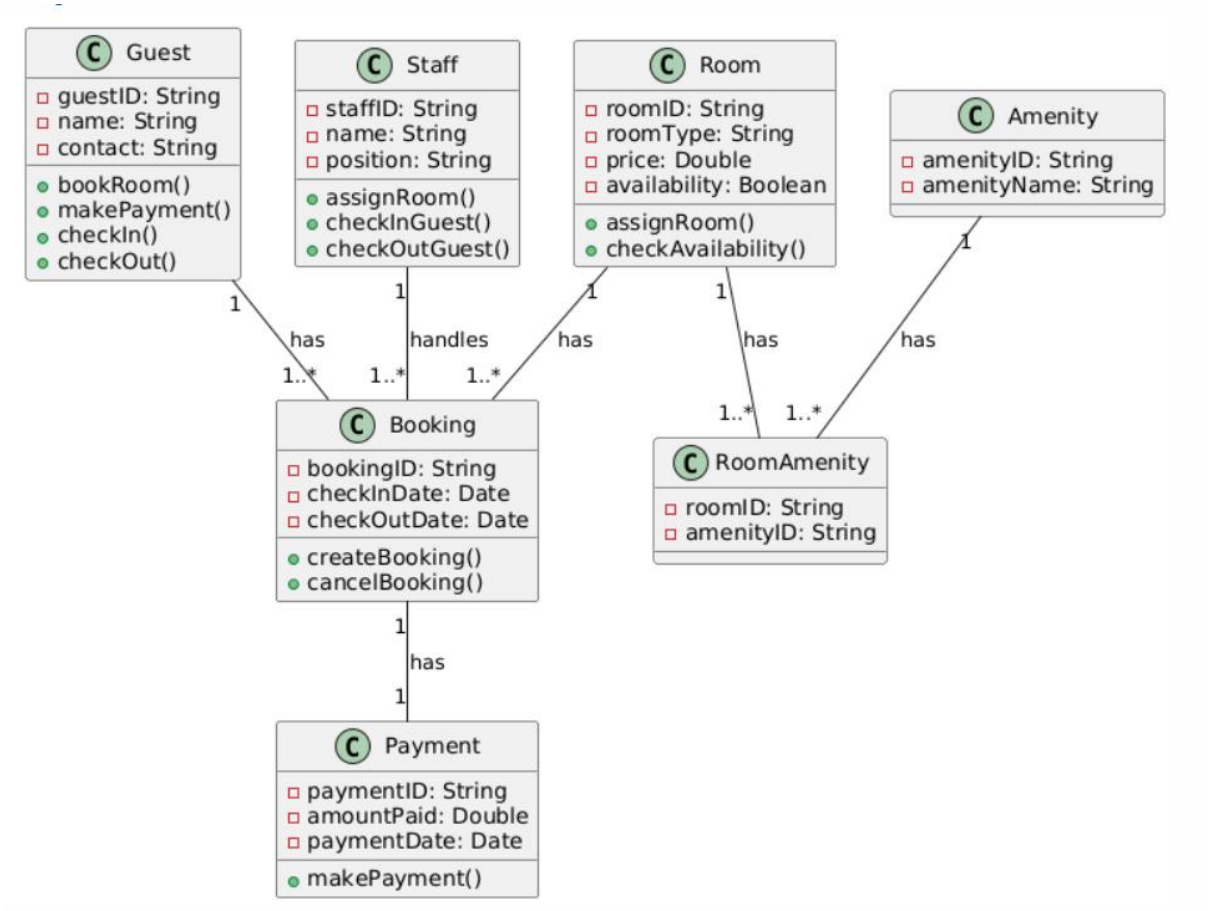
- Public: `+`
  - Accessible from any other class.
  - Example: bookRoom(), processPayment()

- Private: `-`
  - Only accessible within the class itself.
  - Example: paymentStatus (in Payment class)

- Protected: `#`
  - Accessible within the class and by subclasses.
  - Example: guestID (in Guest class)

- Package-Private (default): No symbol
  - Accessible within the same package.
  - Example: roomType (in Room class)

Sample Output :



Result:

The Class diagram has been created successfully by following the steps given.

CODE FOR MINI PROJECT

CODE:

IMPLEMENTATION CODE USING PYTHON

```
# Hostel Management System

class Student:
    def __init__(self, student_id, name, room_no, phone, check_in_date):
        self.student_id = student_id
        self.name = name
        self.room_no = room_no
        self.phone = phone
        self.check_in_date = check_in_date

    def __str__(self):
        return f"ID: {self.student_id}, Name: {self.name}, Room No: {self.room_no}, Phone: {self.phone}, Check-in Date: {self.check_in_date}"


class HostelManagementSystem:
    def __init__(self):
        self.students = []

    def add_student(self):
        student_id = input("Enter Student ID: ")
        name = input("Enter Student Name: ")
        room_no = input("Enter Room Number: ")
        phone = input("Enter Phone Number: ")
        check_in_date = input("Enter Check-in Date (YYYY-MM-DD): ")
        new_student = Student(student_id, name, room_no, phone, check_in_date)
        self.students.append(new_student)
        print("Student added successfully!\n")

    def view_students(self):
        if not self.students:
            print("No students found.\n")
        else:
            print("List of Students:\n")
            for student in self.students:
                print(student)
            print()

    def update_student(self):
```

```
        student_id = input("Enter the Student ID to update: ")
        found = False
        for student in self.students:
            if student.student_id == student_id:
                found = True
                student.name = input(f"Enter new name for {student.name} (leave blank to keep
current): ") or student.name
                student.room_no = input(f"Enter new room number for {student.room_no} (leave
blank to keep current): ") or student.room_no
                student.phone = input(f"Enter new phone number for {student.phone} (leave blank
to keep current): ") or student.phone
                student.check_in_date = input(f"Enter new check-in date for
{student.check_in_date} (leave blank to keep current): ") or student.check_in_date
                print("Student details updated successfully!\n")
                break
        if not found:
            print("Student ID not found!\n")

    def delete_student(self):
        student_id = input("Enter the Student ID to delete: ")
        found = False
        for student in self.students:
            if student.student_id == student_id:
                found = True
                self.students.remove(student)
                print("Student deleted successfully!\n")
                break
        if not found:
            print("Student ID not found!\n")

    def main_menu(self):
        while True:
            print("Hostel Management System")
            print("1. Add Student")
            print("2. View Students")
            print("3. Update Student")
            print("4. Delete Student")
            print("5. Exit")
            choice = input("Enter your choice: ")

            if choice == '1':
                self.add_student()
            elif choice == '2':
                self.view_students()
            elif choice == '3':
                self.update_student()
            elif choice == '4':
```

```
            self.delete_student()
        elif choice == '5':
            print("Exiting Hostel Management System.")
            break
        else:
            print("Invalid choice! Please try again.")

# Running the hostel management system
if __name__ == "__main__":
    hms = HostelManagementSystem()
    hms.main_menu()
```

INPUT:

```
Hostel Management System
1. Add Student
2. View Students
3. Update Student
4. Delete Student
5. Exit
```

OUTPUT:
Enter your choice: 1
Enter Student ID: 23
Enter Student Name: SAHANA
Enter Room Number: 100
Enter Phone Number: 6378186044
Enter Check-in Date (YYYY-MM-DD):
12.11.2024
Student added successfully!