B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

Laboratory Record

OBJECT-ORIENTED MODELLING

Submitted in partial fulfilment for the 5th Semester Laboratory

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

LAKSHITHA.L

1BM22CS134

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
September 2024- January 2025

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Object-Oriented Modelling (23CS5PCOOM) laboratory has been carried out by Lakshitha.L (1BM22CS134) during the 5th Semester September 2024- January 2025.

Signature of the Faculty in Charge
Saritha A. N
Assistant Professor,
Department of Computer Science and Engineering
B.M.S. College of Engineering, Bangalore

TABLE OF CONTENTS

CHAPTER - 1

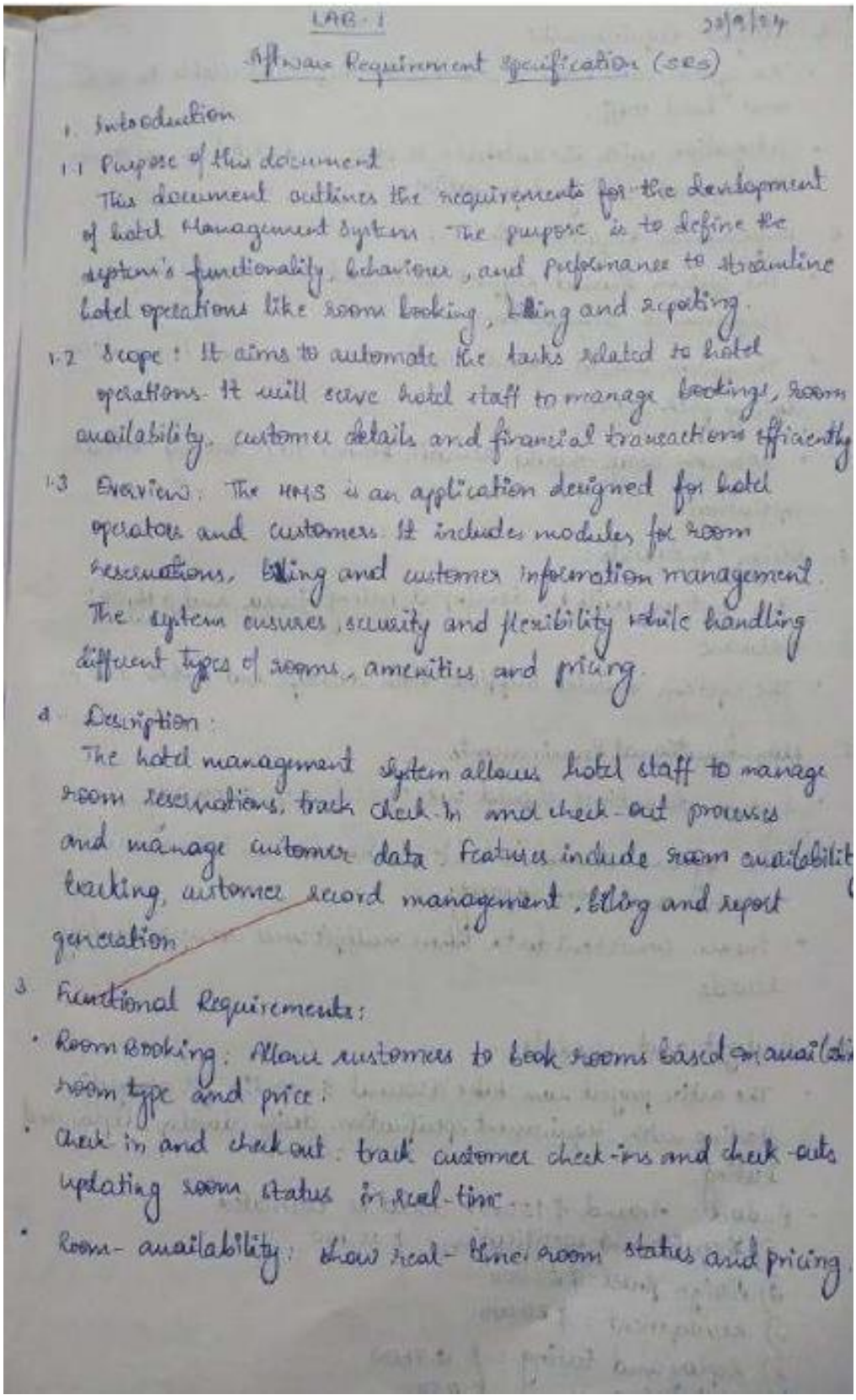
HOTEL MANAGEMENT SYSTEM

PROBLEM STATEMENT

The objective of this project is to develop a comprehensive **Hotel Management System** to manage hotel operations and enhance the guest experience. The proposed system aims to address the following critical areas:

1. **Reservation Management:** Enable online booking with real-time availability checks, secure payment processing, and automated booking confirmations to enhance convenience and operational efficiency.
2. **Services:** Simplify check-in and check-out processes, handle guest requests efficiently, and provide detailed information about hotel amenities to ensure a seamless and satisfying experience.
3. **Inventory Management:** Provide tools to monitor room availability, dynamically manage room rates, assign rooms effectively, and oversee maintenance activities to optimize resource utilization.
4. **Financial Management:** Enable accurate invoice generation, secure payment processing, effective account management, and the creation of comprehensive financial reports to support financial oversight and decision-making.
5. **Employee Management:** Streamline employee scheduling, track attendance records, and manage access permissions to enhance workforce efficiency and ensure operational security.

SOFTWARE REQUIREMENTS SPECIFICATION



LAB - 1 22/8/24

Software Requirement Specification (SRS)

1. Introduction

1.1 Purpose of this document

This document outlines the requirements for the development of hotel Management System. The purpose is to define the system's functionality, behaviour, and performance to streamline hotel operations like room booking, billing and reporting.

1.2 Scope: It aims to automate the tasks related to hotel operations. It will serve hotel staff to manage bookings, room availability, customer details and financial transactions efficiently.

1.3 Overview: The HMS is an application designed for hotel operators and customers. It includes modules for room reservations, billing and customer information management. The system ensures security and flexibility while handling different types of rooms, amenities and pricing.

2. Description:

The hotel management system allows hotel staff to manage room reservations, track check-in and check-out processes and manage customer data. Features include room availability tracking, customer record management, billing and report generation.

3. Functional Requirements:

- Room Booking: Allow customers to book rooms based on availability, room type and price.
- Check in and check out: track customer check-ins and check-outs updating room status in real-time.
- Room-availability: show real-time room status and pricing.

4. Interface Requirements:
- The system will have a web based interface available to customers and hotel staff.
- Integration with the database to store and retrieve customer, room and booking information.

5. Performance Requirements:
- The system should support concurrent user access without performance degradation.
- The system should handle up to 500 concurrent transactions during peak times.
- Memory usage should remain below 70% during normal operation.

Design Constraints:
- The system will be developed using java and a MySQL database.
- The system should support both desktop and mobile devices.

Non-functional Requirements:
- System downtime should not exceed 1% annually.
- The system should be able to handle an increase in customer and room capacity.
- Ensure consistent data when multiple users access & modify records.

Budget and schedule:
- The entire project can take around 6 months for completion starting with requirement specification, design, develop, deploy and testing.
- Budget: Around $150000 could be estimated
  1) Requirement specification: $ 15000
  2) Design phase: $ 25000
  3) Development: $ 80000
  4) Deploy and testing: $ 27500
  5) Maintainance: $ 8500.

UML DIAGRAMS

CLASS DIAGRAM

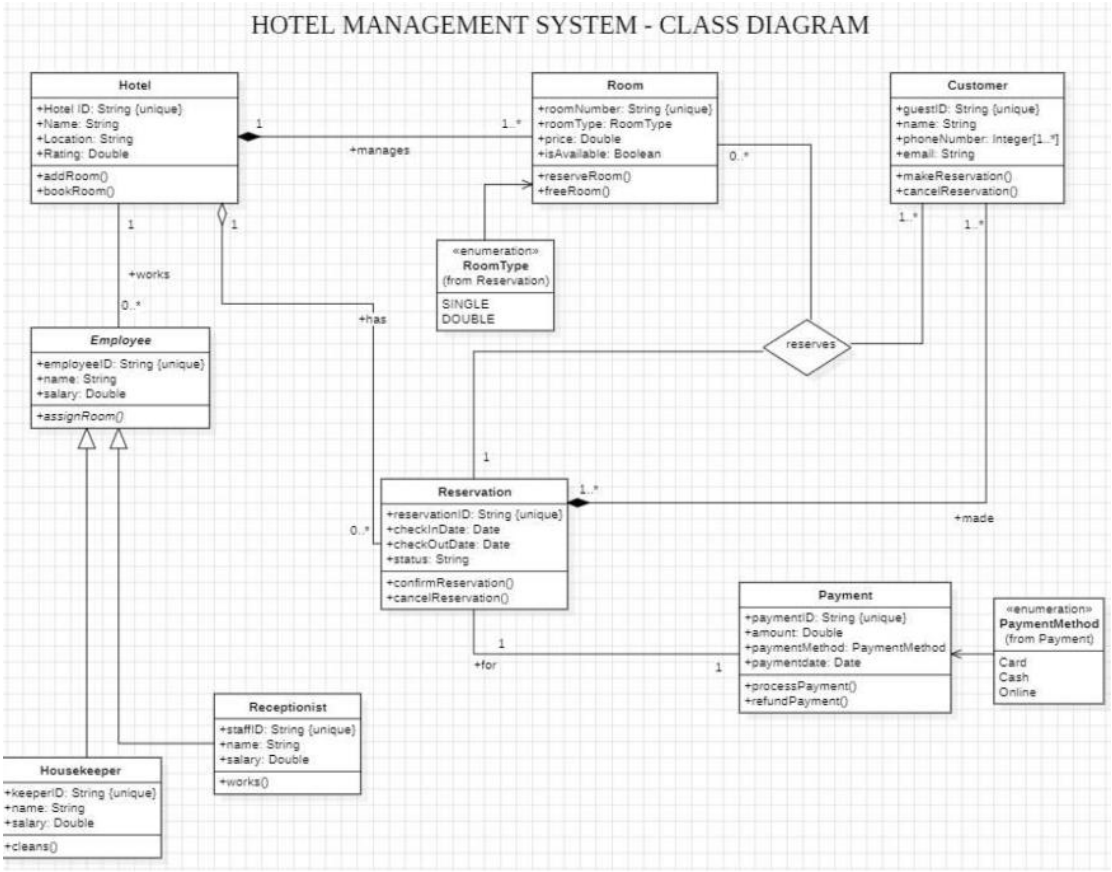


Figure 1.1: Class Diagram

Classes:

- **Hotel:** Represents a hotel with attributes like Hotel ID, Name, Location, and Rating. It has methods to add rooms and book rooms.
- **Room:** Represents a room in the hotel with attributes like room number, type, availability, and price. It has methods to reserve and free rooms.
- **Customer:** Represents a customer with attributes like customer ID, name, phone number, and email. It has methods to make and cancel reservations.
- **Employee:** Represents an employee with attributes like employee ID, name, and salary. It has a method to assign rooms.
- **Manager:** Represents a manager, inheriting from Employee. It has a method to manage the hotel.

- **Receptionist:** Represents a receptionist, inheriting from Employee. It has a method to work.
- **Housekeeper:** Represents a housekeeper, inheriting from Employee. It has a method to clean rooms.
- **Reservation:** Represents a reservation with attributes like reservation ID, check-in and check-out dates, and status. It has methods to confirm and cancel reservations.
- **Payment:** Represents a payment with attributes like payment ID, amount, payment method, and date. It has methods to process and refund payments.

Associations:

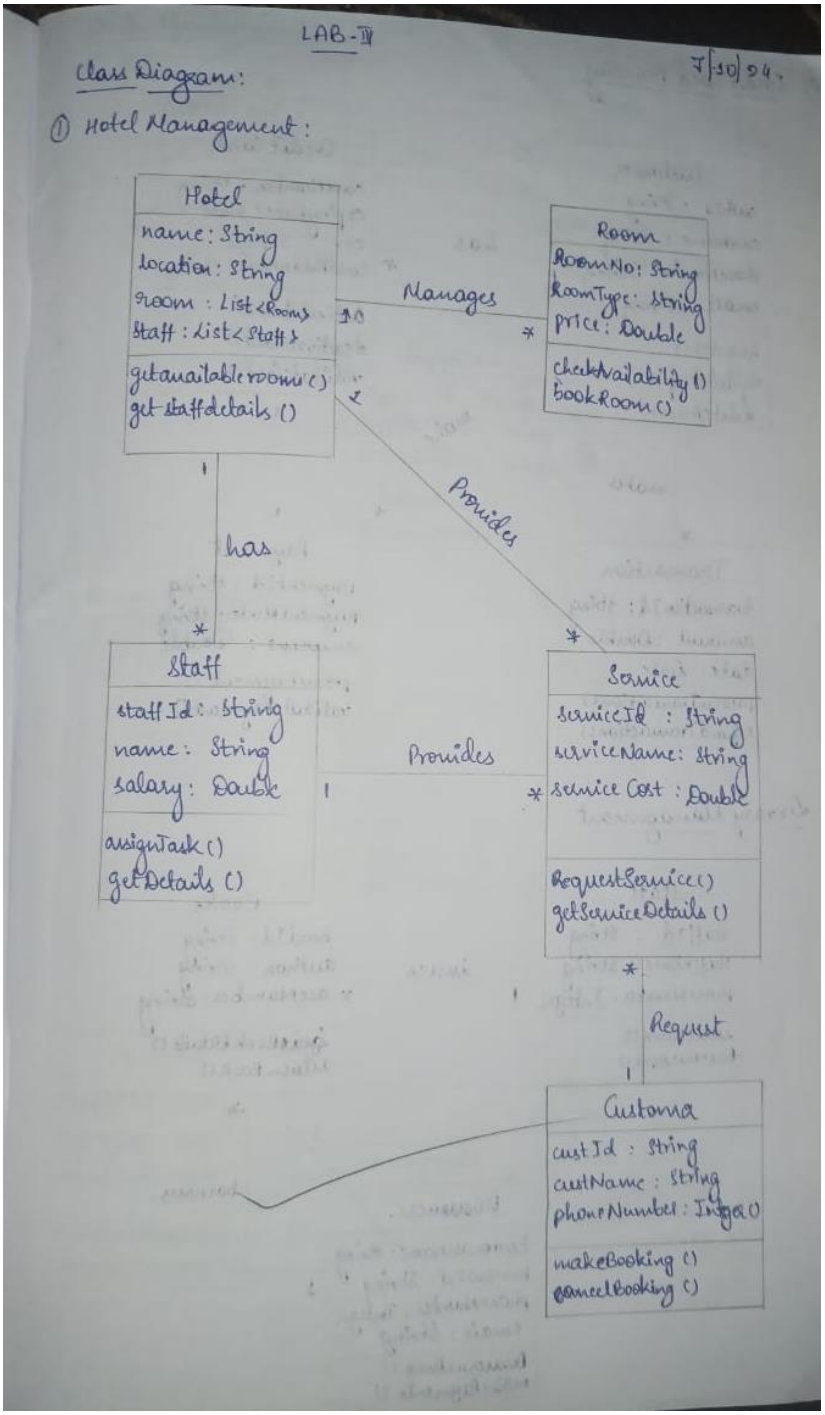
- **Hotel manages Room:** One hotel manages many rooms.
- **Customer reserves Room:** One customer can reserve many rooms, and one room can be reserved by many customers.
- **Employee works for Hotel:** Many employees work for one hotel.
- **Manager manages Hotel:** One manager manages one hotel.
- **Receptionist works for Hotel:** Many receptionists work for one hotel.
- **Housekeeper works for Hotel:** Many housekeepers work for one hotel.
- **Reservation for Room:** One reservation is for one room.
- **Payment for Reservation:** One payment is for one reservation.

Multiplicity:

- **Hotel manages Room:** 1..* (One hotel manages one or more rooms)
- **Customer reserves Room:** 0..* (One customer can reserve zero or more rooms)
- **Employee works for Hotel:** 0..* (Many employees can work for one hotel)
- **Manager manages Hotel:** 1 (One manager manages one hotel)
- **Receptionist works for Hotel:** 0..* (Many receptionists can work for one hotel)
- **Housekeeper works for Hotel:** 0..* (Many housekeepers can work for one hotel)
- **Reservation for Room:** 1 (One reservation is for one room)
- **Payment for Reservation:** 1 (One payment is for one reservation)

Other Notable Elements:

- **Enumeration RoomType:** Defines different types of rooms (SINGLE, DOUBLE, SUITE).
- **Enumeration PaymentMethod:** Defines different payment methods (Cash, Card, Online).



STATE DIAGRAM

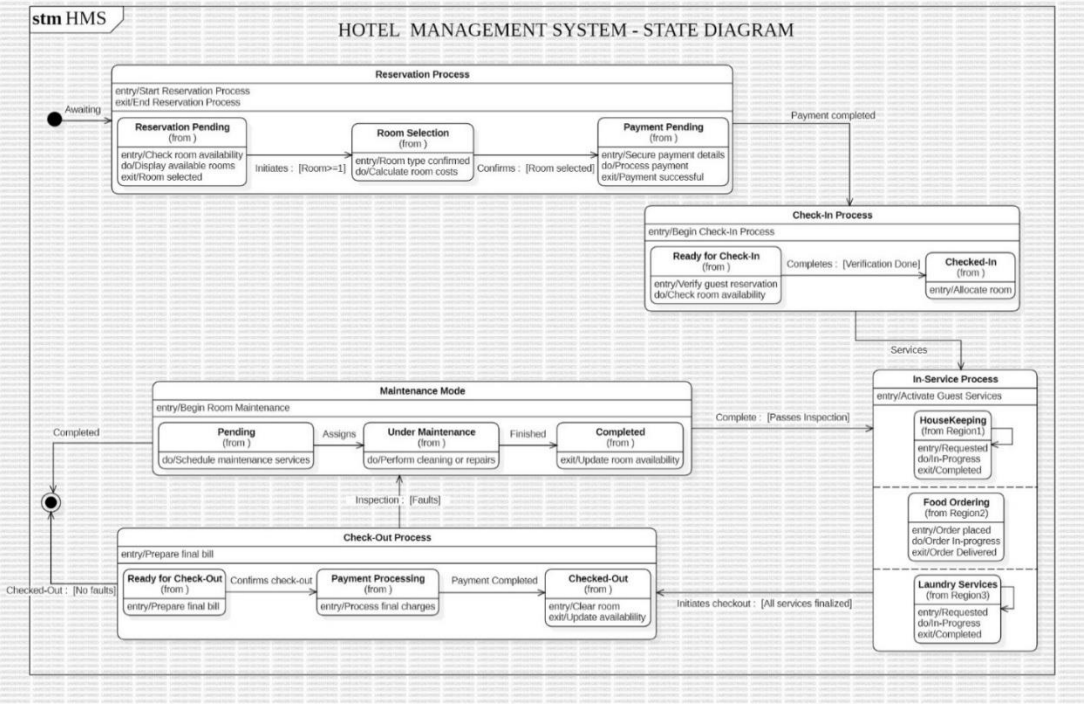


Figure 1.2: State Diagram

States:

- **Awaiting Reservation:** The initial state where the system is ready to accept a new reservation request.
- **Reservation Pending:** The system is processing the reservation request, likely checking room availability.
- **Room Selection:** The available rooms are displayed to the customer for selection.
- **Payment Pending:** The customer is prompted to make a payment for the selected room.
- **Payment Completed:** The payment is processed successfully, and the reservation is confirmed.
- **Checked-In:** The customer has checked in to the hotel and the room is assigned.

Transitions:

- **Reservation Pending:** Triggered by the customer initiating a reservation request.
- **Room Selection:** Triggered after checking room availability in the Reservation Pending state.
- **Payment Pending:** Triggered when the customer selects a room.
- **Payment Completed:** Triggered when the payment is successfully processed.
- **Checked-In:** Triggered when the customer arrives at the hotel and checks in.

2. Advanced State Diagram (Nested State Diagram: In-Service Process)

This part of the diagram focuses on the services provided to guests after they check in.

Nested States (within In-Service Process):

- **Housekeeping:** This nested state handles housekeeping services like cleaning and maintenance.
- **Food Ordering:** This nested state handles food and beverage orders.
- **Laundry Services:** This nested state handles laundry requests.

States within each nested state:

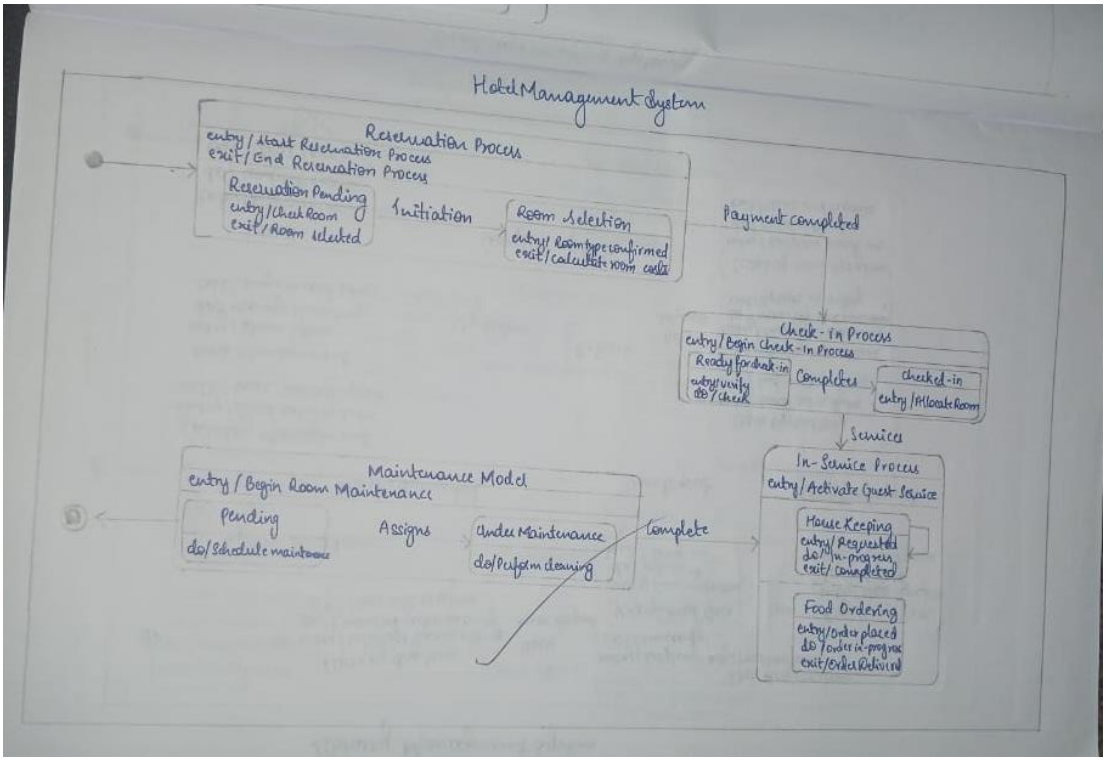
- **Requested:** The service is requested by the guest.
- **In-Progress:** The service is being processed.
- **Completed:** The service is completed.

Transitions:

- **Requested:** Triggered when a guest requests a service.
- **In-Progress:** Triggered when the service is initiated.
- **Completed:** Triggered when the service is completed.

Overall Description:

This state diagram provides a high-level overview of the hotel management system, focusing on the key processes: reservation, check-in, check-out, and in-service processes. The nested state diagram within the In-Service Process demonstrates a more detailed view of the services provided to guests.


Hotel Management System
Reservation Process
entry / start Reservation Process
exit / End Reservation Process
Reservation Pending
entry / Check Room
exit / Room Idented
Initiation
Room Selection
entry / Room type confirmed
exit / calculate room costs
Payment completed
Check - in Process
entry / Begin check - in Process
Ready for check-in
entry / verify
do / check
Completes
Checked-in
entry / Allocate Room
Services
In-Service Process
entry / Activate Guest Service
House Keeping
entry / Requested
do / in-progress
exit / completed
Food Ordering
entry / order placed
do / order in-progress
exit / order delivered
Maintenance Model
entry / Begin Room Maintenance
Pending
do / Schedule maintenance
Assigns
under Maintenance
do / Perform cleaning
Complete

USE CASE DIAGRAM

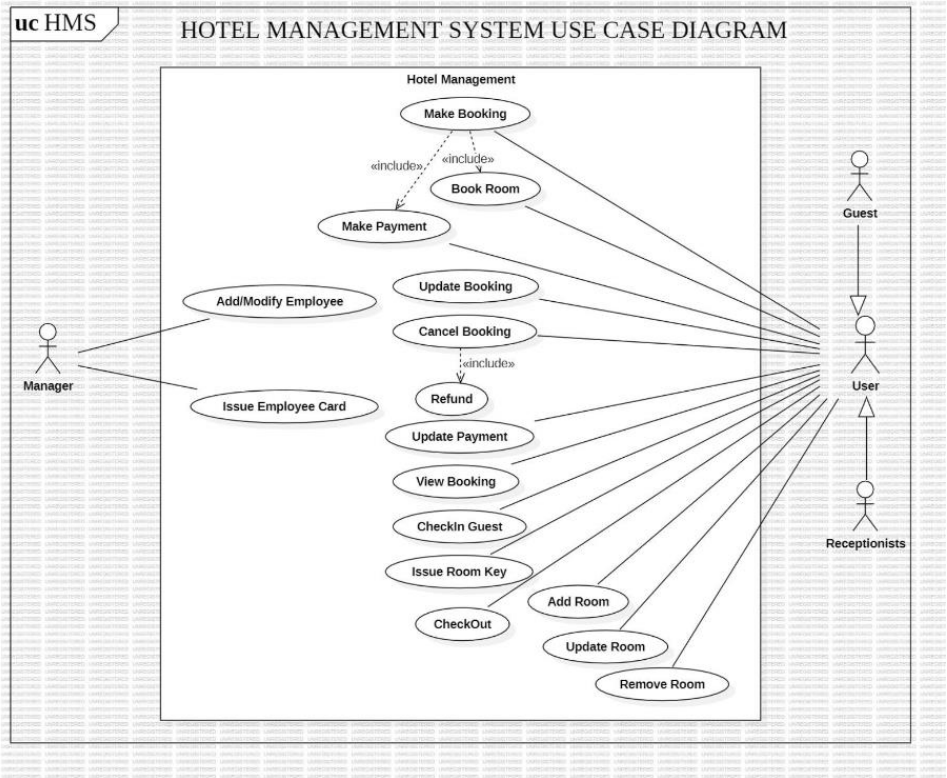


Figure 1.3: Use Case Diagram

Actors:

- **Manager:** Responsible for managing employees and hotel operations.
- **Guest:** Represents the customers staying at the hotel.
- **Receptionist:** Staff members responsible for managing guest interactions, check-in/check-out, and room assignments.

Use Cases:

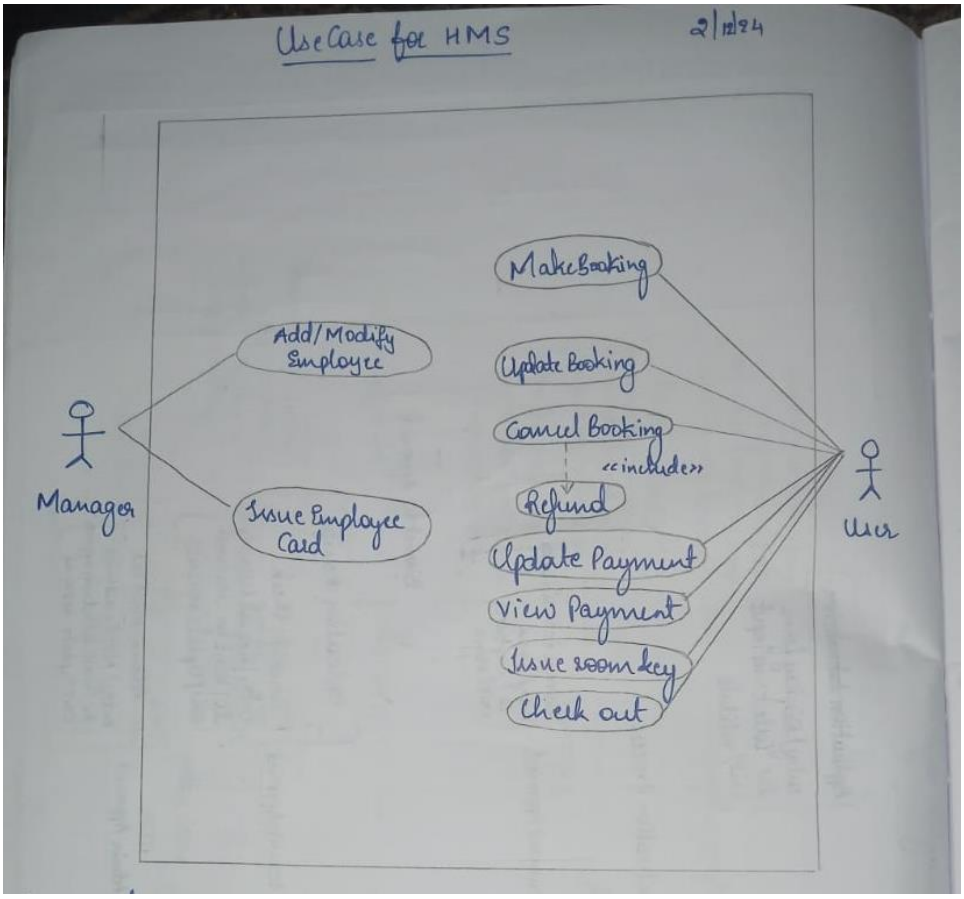
- **Hotel Management:** This is a parent use case that includes other use cases related to managing the hotel.
- **Make Booking:** Use case for guests to reserve rooms in the hotel.
- **Book Room:** A child use case of Make Booking, specifically for reserving a particular room.
- **Make Payment:** Use case for guests to pay for their bookings.
- **Update Booking:** Use case for modifying existing bookings (e.g., changing dates, adding guests).

- **Cancel Booking:** Use case for guests to cancel their bookings.
- **Refund:** Use case for processing refunds for canceled bookings.
- **View Booking:** Use case for guests to view their booking details.
- **Checkin Guest:** Use case for receptionists to check guests into their rooms.
- **Issue Room Key:** Use case for receptionists to provide room keys to guests.
- **Checkout:** Use case for guests to check out of the hotel.
- **Add Room:** Use case for adding new rooms to the hotel inventory.
- **Update Room:** Use case for modifying room details (e.g., type, price).
- **Remove Room:** Use case for removing rooms from the hotel inventory.
- **Add/Modify Employee:** Use case for managers to add or modify employee records.
- **Issue Employee Card:** Use case for managers to issue employee ID cards.

Relationships:

- **Includes:** Indicates that one use case includes the functionality of another use case. For example, "Make Booking" includes "Book Room."
- **Inheritance:** Indicates that one actor inherits the roles and responsibilities of another actor. For example, "Guest" inherits from "User."

Overall, this use case diagram provides a high-level overview of the key functionalities and interactions within the Hotel Management System. It helps to visualize the different roles of actors and how they interact with the system to perform various tasks.


UseCase for HMS
2/12/24
Make Booking
Add/Modify Employee
Update Booking
Cancel Booking
<<include>>
Refund
Manager
Issue Employee Card
Update Payment
View Payment
Issue room key
Check out
User

SEQUENCE DIAGRAM

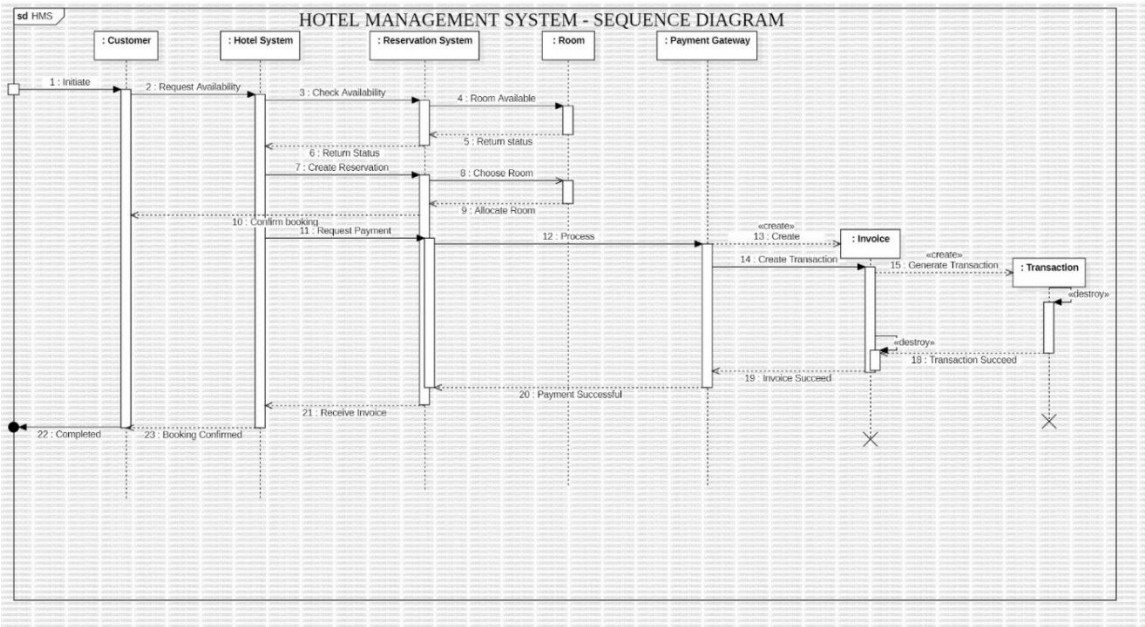


Figure 1.4: Sequence Diagram

Scenario 1: Customer Makes a Reservation

1. **Customer:** Sends a request to the Hotel System to check room availability for specific dates.
2. **Hotel System:** Receives the request and checks room availability in the Reservation System.
3. **Reservation System:** Checks room availability based on the requested dates and returns the status to the Hotel System.
4. **Hotel System:** Receives the availability status and sends it back to the Customer.
5. **Customer:** Reviews the available rooms and chooses a preferred room.
6. **Customer:** Sends the chosen room selection to the Hotel System.
7. **Hotel System:** Creates a reservation for the chosen room in the Reservation System.
8. **Reservation System:** Confirms the reservation and allocates the room.
9. **Hotel System:** Informs the Customer about the confirmed reservation.
10. **Hotel System:** Requests payment details from the Customer.
11. **Customer:** Provides payment details to the Hotel System.

12. **Payment Gateway:** Processes the payment and sends an invoice to the Hotel System.
13. **Hotel System:** Receives the invoice and creates a transaction record.
14. **Payment Gateway:** Generates a transaction receipt and sends it to the Customer.
15. **Customer:** Receives the invoice and payment success notification.
16. **Customer:** Completes the booking process.
17. **Hotel System:** Confirms the booking to the Customer.

Scenario 2: Customer Cancels a Reservation

1. **Customer:** Sends a request to the Hotel System to cancel an existing reservation.
2. **Hotel System:** Receives the cancellation request and checks the reservation status in the Reservation System.
3. **Reservation System:** Retrieves the reservation details and updates the status to "Cancelled."
4. **Hotel System:** Informs the Customer that the reservation has been canceled.

Simple Sequence Diagram

Objects:

- **User**
- **Authentication System**
- **Database**
- **Application**
- **UI**

Interactions:

1. **User** sends a login request to the **Authentication System**.
2. **Authentication System** verifies the credentials and sends a request to the **Database**.
3. **Database** retrieves user information and sends it back to the **Authentication System**.

4. **Authentication System** authenticates the user and sends a success message to the **Application**.
5. **Application** displays the user interface (**UI**) to the **User**.

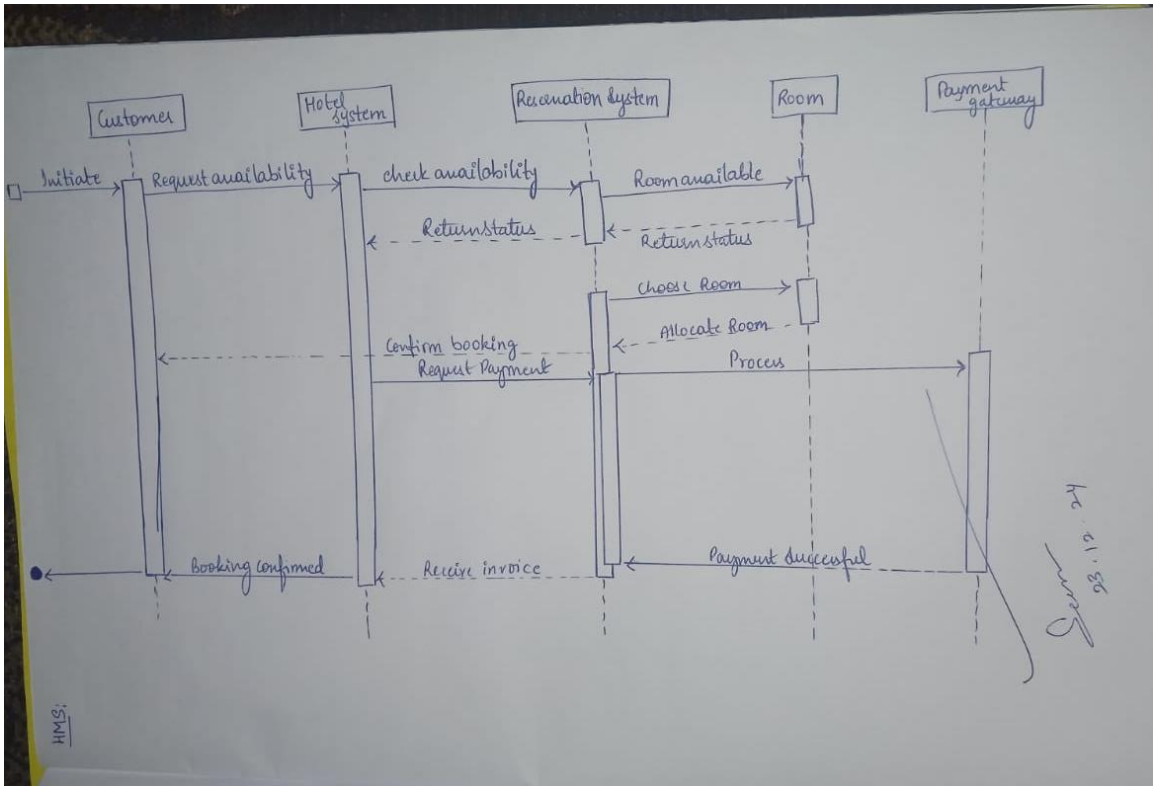
Advanced Sequence Diagram with Passive and Transient Objects

Objects:

- **Order** (Transient)
- **Order Line (Transient)**
- **Product** (Passive)
- **Inventory** (Passive)
- **Order System**

Interactions:

1. **User** places an order with the **Order System**.
2. **Order System** creates an **Order** object (transient).
3. **Order System** creates multiple **Order Line** objects (transient) for each product in the order.
4. **Order Line** interacts with **Product** (passive) to retrieve product information (price, quantity).
5. **Order Line** interacts with **Inventory** (passive) to check product availability.
6. **Order System** processes the order and updates the **Inventory** accordingly.
7. **Order** and **Order Line** objects are destroyed after the order is processed.


Customer
Hotel system
Reservation system
Room
Payment gateway
Initiate
Request availability
check availability
Room available
Return status
Return status
Choose Room
Allocate Room
Confirm booking
Request Payment
Process
Booking confirmed
Receive invoice
Payment successful
HMS:
23.10.24

ACTIVITY DIAGRAM

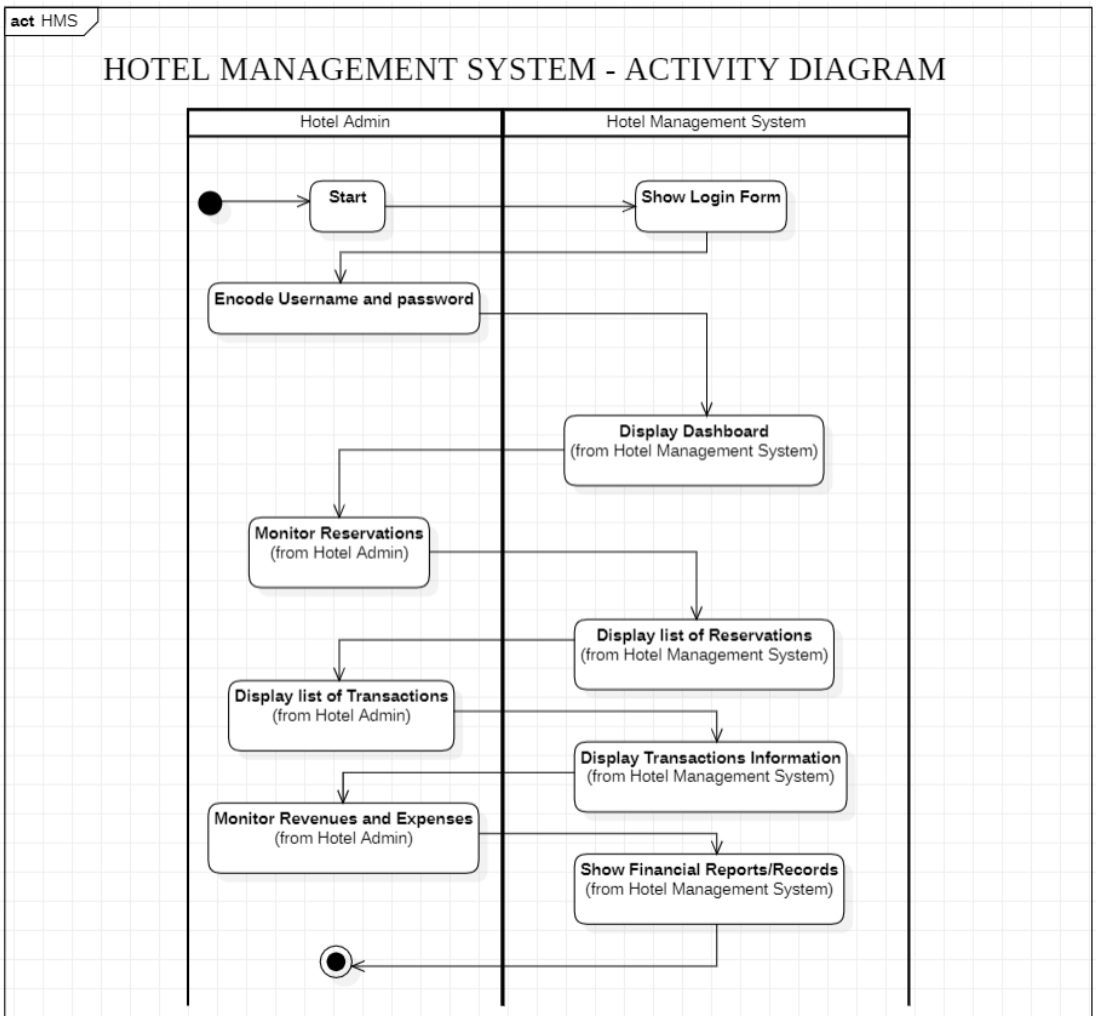
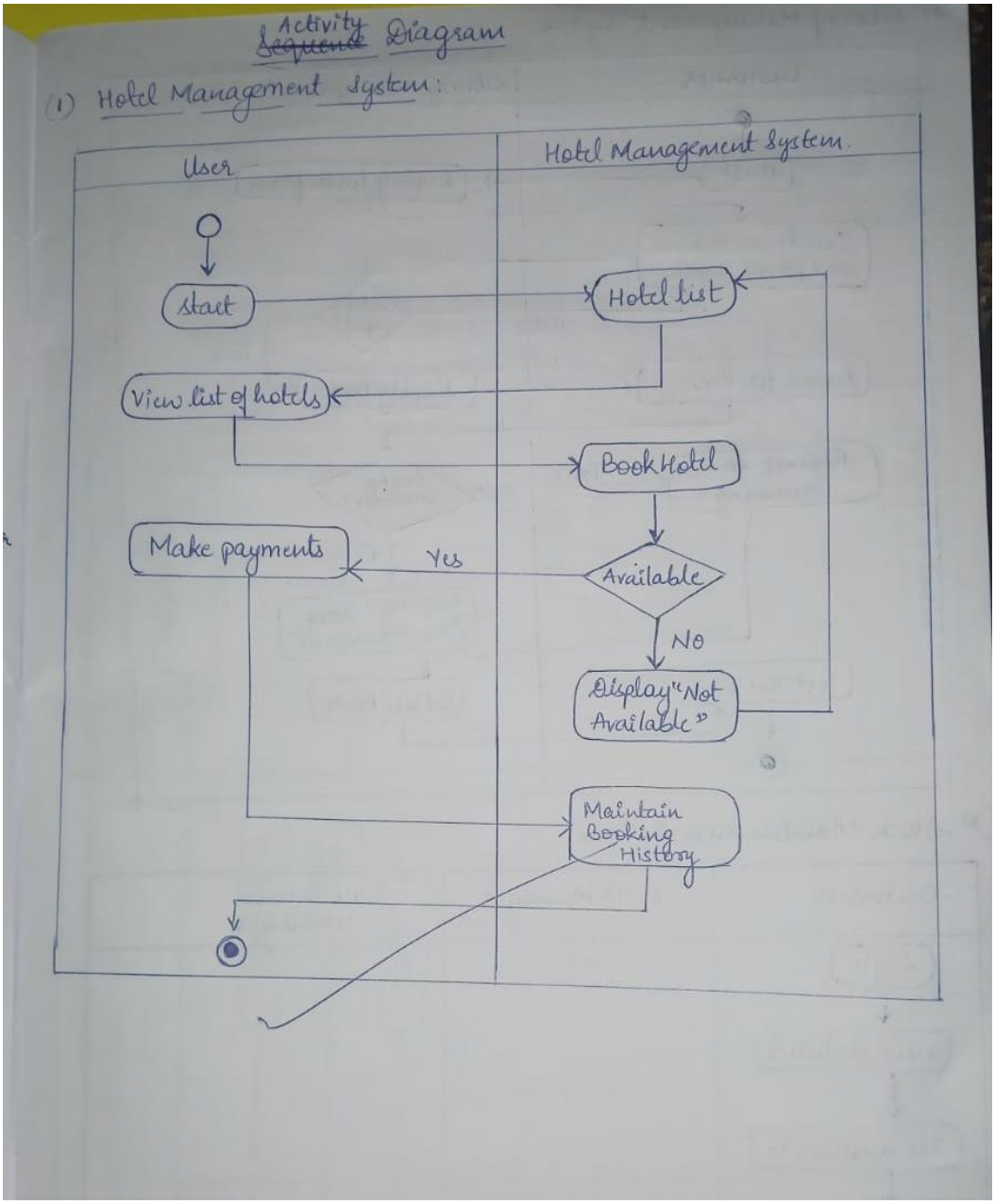


Figure 1.5: Activity Diagram

Swimlanes:

1. **Hotel Admin:** This swimlane represents the actions performed by the hotel administrator. It encompasses activities like:
    - **Start:** The initial state of the system.
    - **Encode Username and Password:** The hotel admin enters their credentials for login.
    - **Monitor Reservations:** The admin checks the list of reservations.
    - **Check list of Transactions:** The admin reviews the transaction records.
    - **Monitor Revenues and Expenses:** The admin analyzes the financial performance of the hotel.

2. **Hotel Management System:** This swimlane encompasses the actions performed by the Hotel Management System itself, which includes:
    - **Show Login Form:** The system displays the login form to the admin.
    - **Display "Wrong Inputs":** The system provides an error message if the admin enters incorrect login credentials.
    - **Display Dashboard:** The system displays the main dashboard with key information.
    - **Show list of Reservations:** The system presents the list of reservations to the admin.
    - **Display Transactions Information:** The system displays detailed information about transactions.
    - **Show Financial Reports/Records:** The system generates and displays financial reports and records.


Activity Diagram
(1) Hotel Management System:
User
Hotel Management System.
start
Hotel list
View list of hotels
Book Hotel
Make payments
Yes
Available
No
Display "Not Available"
Maintain Booking History

CHAPTER – 2
CREDIT CARD PROCESSING SYSTEM

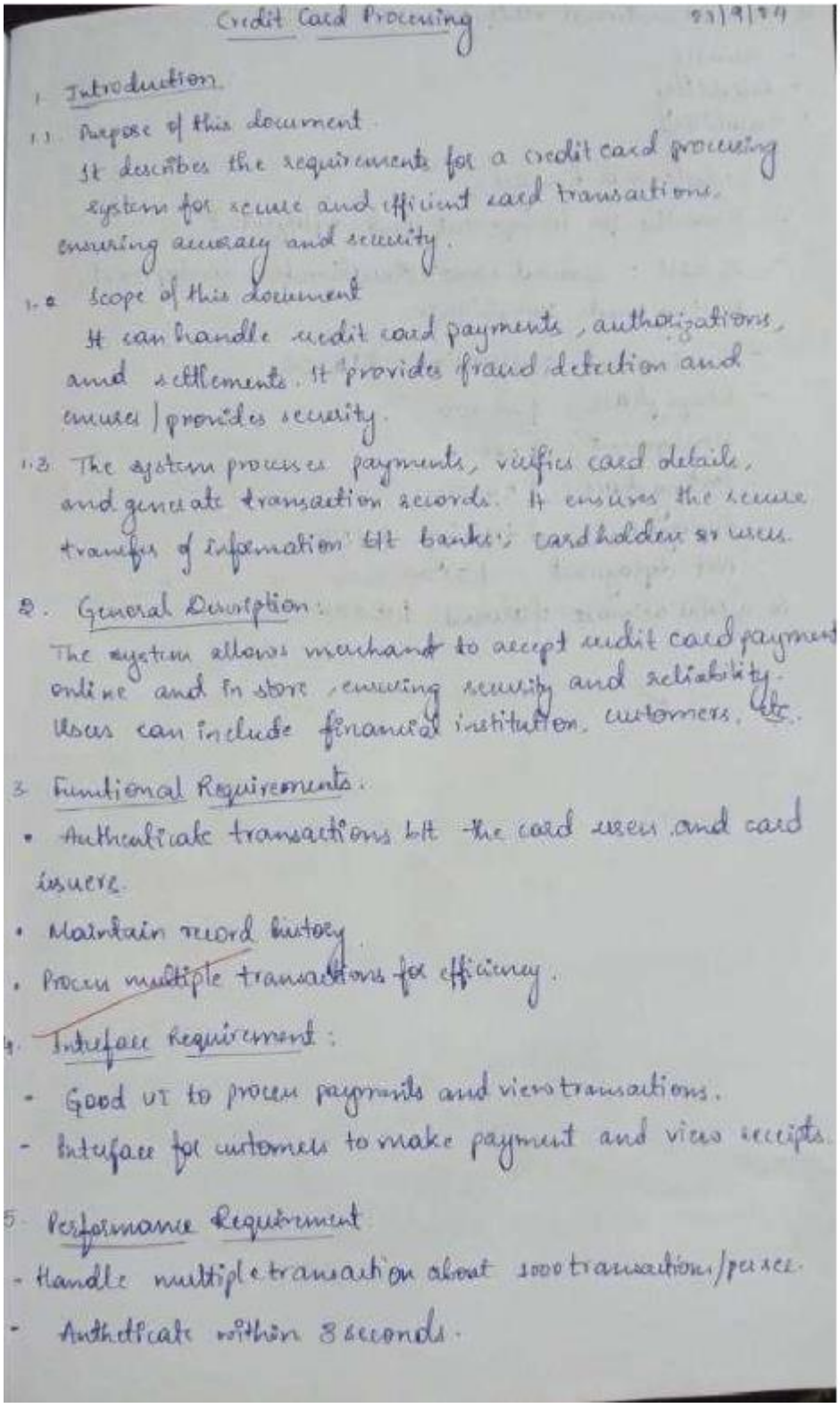
PROBLEM STATEMENT

The objective of this project is to design and implement a robust **Credit Card Processing System** to ensure secure, efficient, and accurate handling of credit card transactions. The proposed system aims to address the following critical functionalities:

1. **Transaction Processing:** Facilitate real-time authorization, capture, and settlement of credit card payments, ensuring fast and reliable transaction handling.
2. **Fraud Detection and Prevention:** Integrate advanced algorithms to monitor transactions for suspicious activities, implement security protocols, and reduce fraudulent activities effectively.
3. **Account Management:** Provide tools for managing user accounts, including updating cardholder information, viewing transaction histories, and generating account statements.
4. **Dispute Resolution:** Support efficient handling of disputes and chargebacks by enabling detailed tracking of transaction records and providing automated workflows for resolution.
5. **Compliance and Security:** Ensure adherence to industry standards such as PCI DSS (Payment Card Industry Data Security Standard) and implement strong encryption and tokenization mechanisms to safeguard sensitive data.
6. **Integration Capabilities:** Allow seamless integration with banking systems, payment gateways, and merchant platforms to enhance interoperability and scalability.

This system will centralize and automate credit card processing, minimizing errors, enhancing transaction security, and providing a seamless user experience for both merchants and cardholders.

SOFTWARE REQUIREMENTS SPECIFICATION



Credit Card Processing 23/9/24

1. Introduction

1.1 Purpose of this document

It describes the requirements for a credit card processing system for secure and efficient card transactions. ensuring accuracy and security.

1.2 Scope of this document

It can handle credit card payments, authorizations, and settlements. It provides fraud detection and ensures / provides security.

1.3 The system processes payments, verifies card details, and generate transaction records. It ensures the secure transfer of information bet banks, cardholders & users.

2. General Discription:

The system allows merchant to accept credit card payment online and in store, ensuring security and reliability. Users can include financial institution, customers, etc.

3 Funtional Requirements:

- Authenticate transactions bt the card users and card issuers.
- Maintain record history
- Process multiple transactions for efficiency.

4. Interface Requirement:

- Good UI to process payments and view transactions.
- Interface for customers to make payment and view receipts.

5 Performance Requirement

- Handle multiple transaction about 1000 transaction/process.
- Authenticate within 3 seconds.

7. Non-Functional Attributes
- Security
- Reliability
- Scalability.

8. Schedule and Budget:
- 6 months for development and deployment.
- Budget: around some estimation for. development, testing and compliance.
  - Requirement specification: $15,000
  - Design phase: $25 000
  - Development: $80000
  - Testing phase: $20,000
  - Deployment and training: $7500
  - Post deployment: $2500

So, a total estimate of around $150000.

UML DIAGRAMS

CLASS DIAGRAM

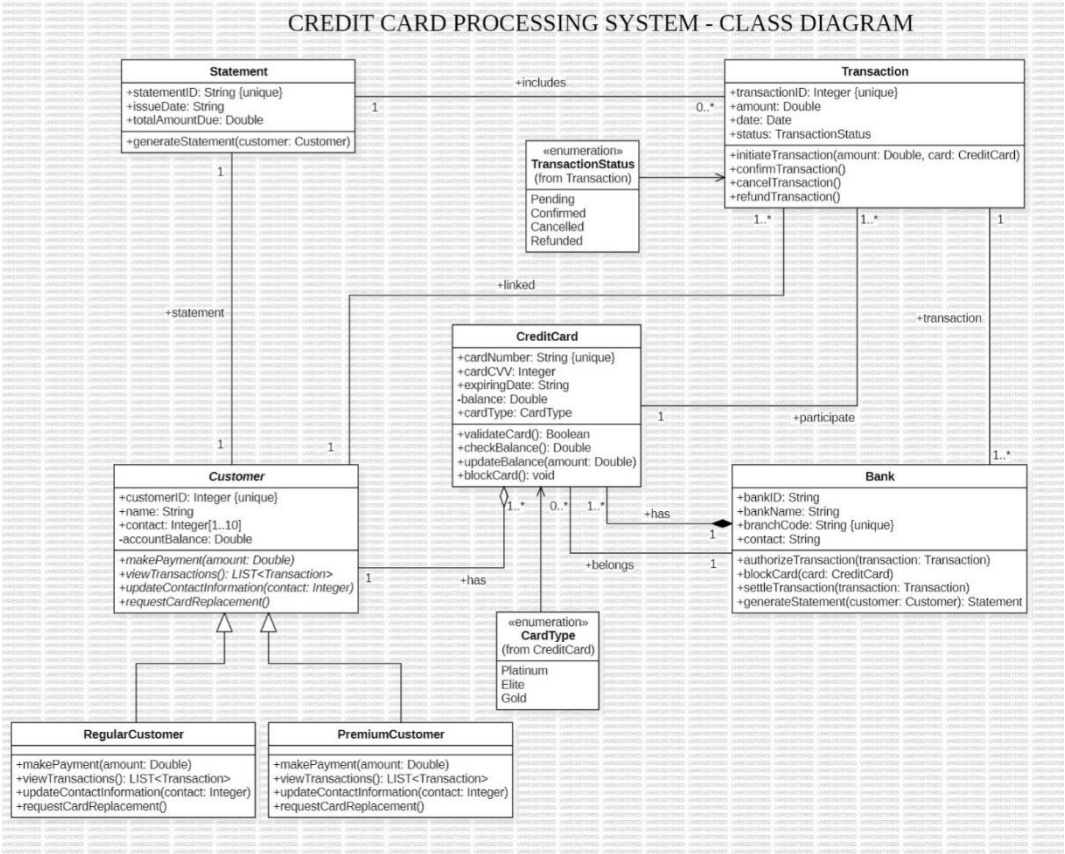


Figure 2.1: Class Diagram

Classes:

- **Statement:** Represents a statement with attributes like statement ID, issue date, and total amount due. It has a method to generate a statement for a customer.
- **Transaction:** Represents a transaction with attributes like transaction ID, date, amount, status, and associated credit card. It has methods to initiate, confirm, cancel, and refund transactions.
- **CreditCard:** Represents a credit card with attributes like card number, cardholder name, expiration date, card type, and balance. It has methods to validate the card, check the balance, and block the card.
- **Customer:** Represents a customer with attributes like customer ID, name, contact information, and account balance. It has methods to make payments, view transactions, update contact information, and request card replacement.

- **RegularCustomer:** Represents a regular customer, inheriting from Customer.
- **PremiumCustomer:** Represents a premium customer, inheriting from Customer.
- **Bank:** Represents a bank with attributes like bank ID, name, branch code, and contact information. It has methods to authorize, block, and settle transactions, as well as generate statements for customers.

Associations:

- **Statement includes Transaction:** One statement includes many transactions.
- **Transaction linked to CreditCard:** One transaction is linked to one credit card.
- **Customer has CreditCard:** One customer can have many credit cards.
- **Bank has CreditCard:** One bank can have many credit cards.
- **Bank participates in Transaction:** One bank participates in many transactions.

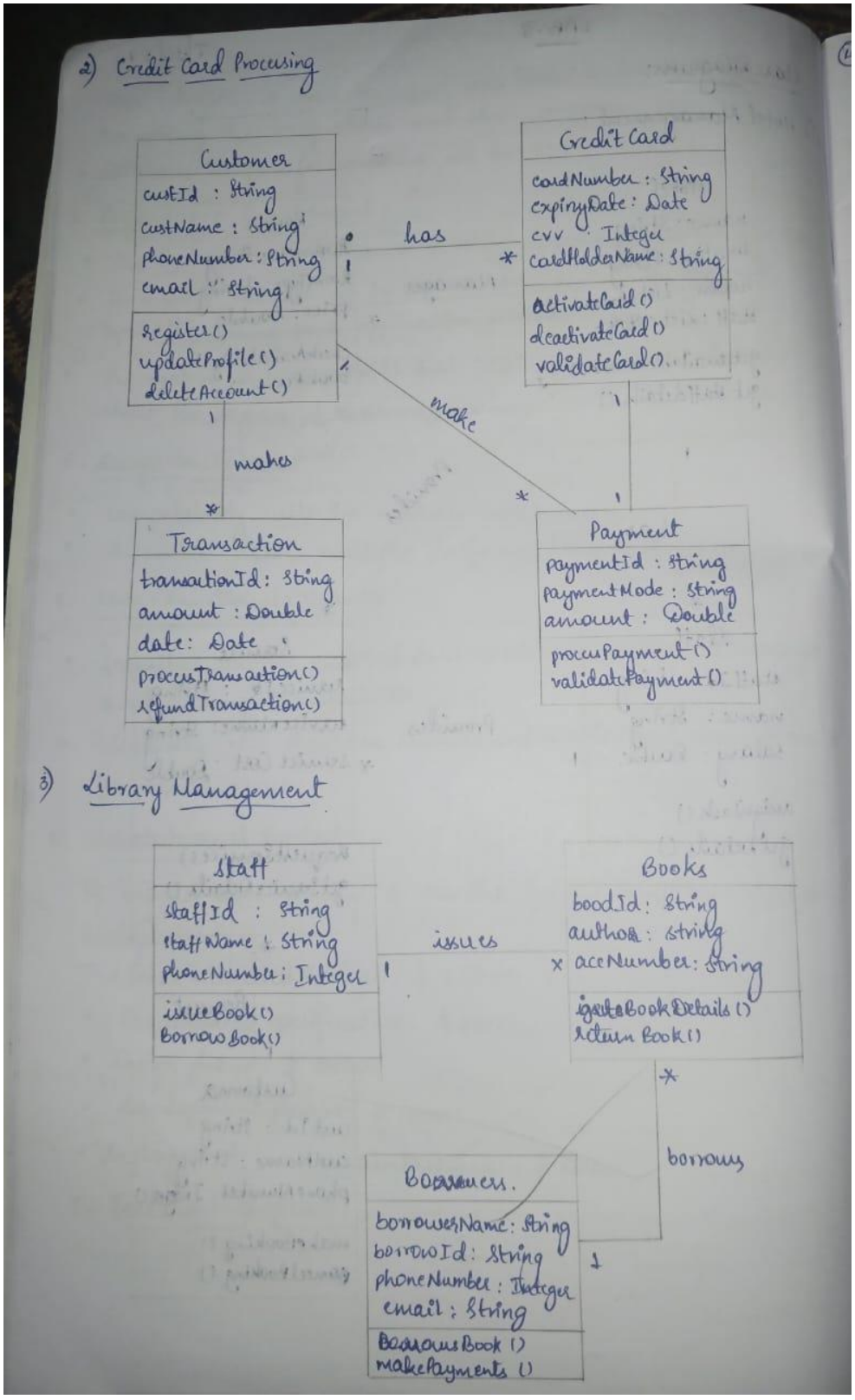
Multiplicity:

- **Statement includes Transaction:** 1..* (One statement includes one or more transactions)
- **Transaction linked to CreditCard:** 1 (One transaction is linked to one credit card)
- **Customer has CreditCard:** 0..* (One customer can have zero or more credit cards)
- **Bank has CreditCard:** 0..* (One bank can have zero or more credit cards)
- **Bank participates in Transaction:** 1..* (One bank participates in one or more transactions)

Other Notable Elements:

- **Enumeration TransactionStatus:** Defines different transaction statuses (Pending, Confirmed, Cancelled, Refunded).
- **Enumeration CardType:** Defines different card types (Platinum, Elite, Gold, Regular).

2) Credit Card Processing



3) Library Management

STATE DIAGRAM

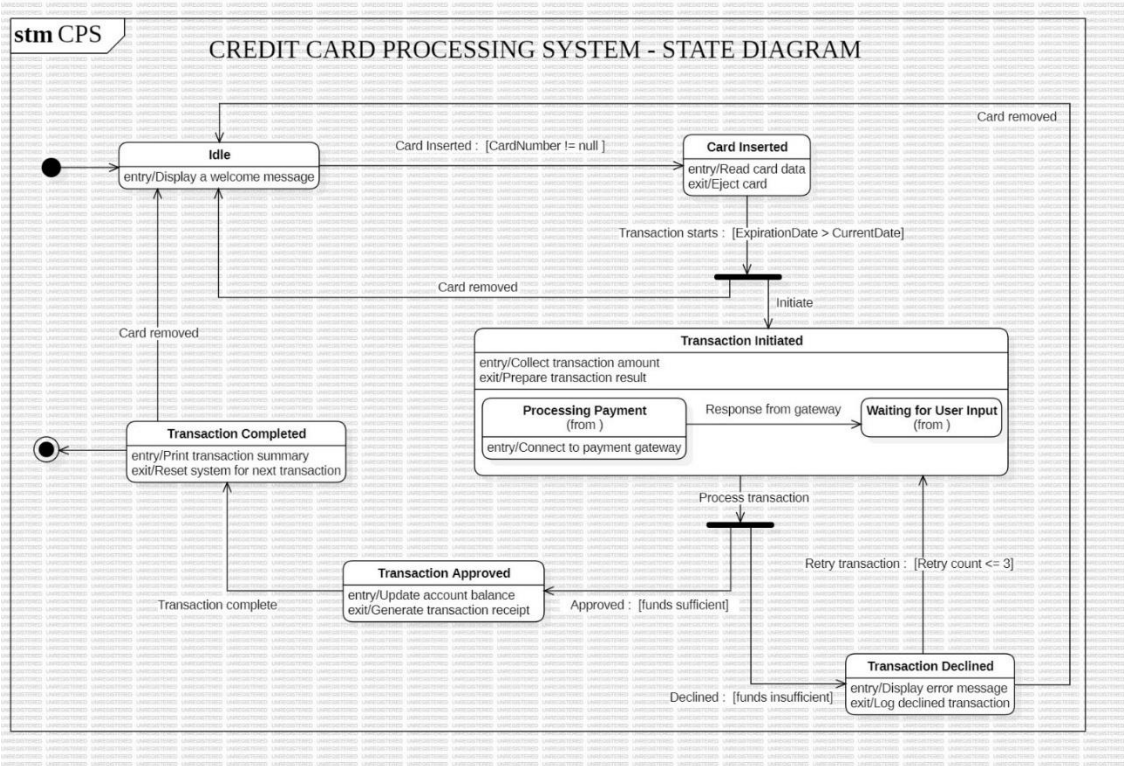


Figure 2.2: State Diagram

States:

- **Idle:** The initial state of the system, waiting for a card to be inserted.
- **Card Inserted:** The card is inserted into the machine.
- **Transaction Initiated:** The transaction process begins, including data collection and amount input.
- **Processing Payment:** The system is communicating with the payment gateway to process the transaction.
- **Waiting for User Input:** The system is waiting for user input, such as PIN entry or approval.
- **Transaction Approved:** The transaction is successfully processed, and the account balance is updated.
- **Transaction Declined:** The transaction is declined due to insufficient funds or other reasons.
- **Transaction Completed:** The transaction process is complete, and the system is ready for the next transaction.

Transitions:

- **Card Inserted:** Triggered when the card is inserted into the machine.
- **Transaction Initiated:** Triggered when the card is valid and the transaction amount is entered.
- **Processing Payment:** Triggered when the system starts communicating with the payment gateway.
- **Waiting for User Input:** Triggered if the payment gateway requires additional input from the user.
- **Transaction Approved:** Triggered if the transaction is approved by the payment gateway.
- **Transaction Declined:** Triggered if the transaction is declined by the payment gateway.
- **Transaction Completed:** Triggered after the transaction is either approved or declined and the system is reset for the next transaction.

Events:

- **Card Inserted:** The event that triggers the transition from the Idle state to the Card Inserted state.
- **Transaction Starts:** The event that triggers the transition from the Card Inserted state to the Transaction Initiated state.
- **Response from Gateway:** The event that triggers the transition from the Processing Payment state to either the Transaction Approved or Transaction Declined state.
- **Transaction Complete:** The event that triggers the transition from the Transaction Approved or Transaction Declined state to the Transaction Completed state.

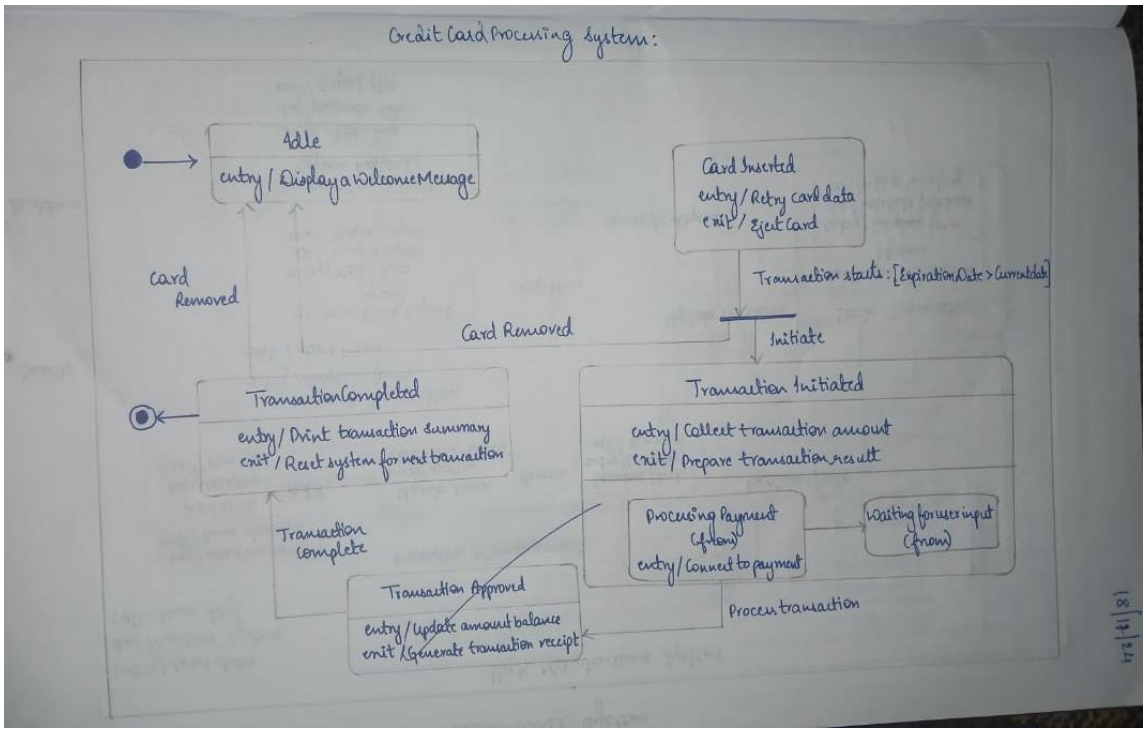
Activities:

- **Display a welcome message:** Activity performed in the Idle state.
- **Read card data:** Activity performed in the Card Inserted state.
- **Collect transaction amount:** Activity performed in the Transaction Initiated state.

- **Connect to payment gateway:** Activity performed in the Processing Payment state.
- **Update account balance:** Activity performed in the Transaction Approved state.
- **Display error message:** Activity performed in the Transaction Declined state.
- **Print transaction summary:** Activity performed in the Transaction Completed state.

Overall Description:

This state diagram illustrates the various states and transitions involved in a credit card processing system. It shows the flow of the transaction from the initial card insertion to the final completion or decline. The diagram also includes activities and events that trigger transitions between states.



USE CASE DIAGRAM

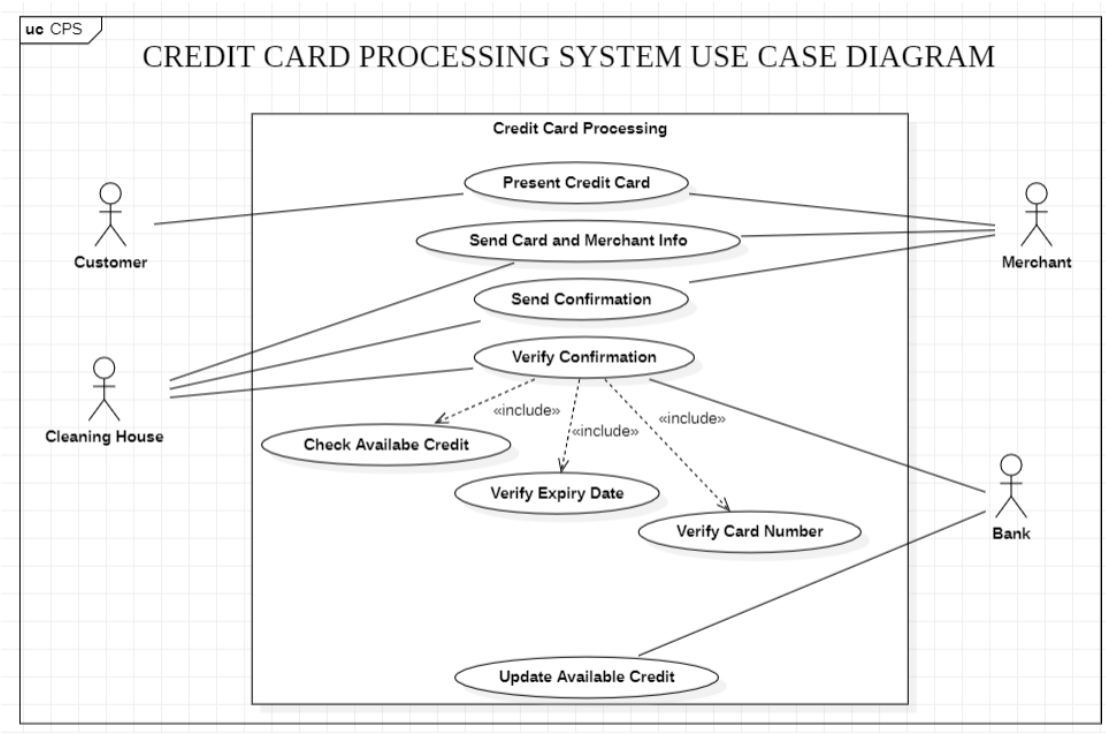


Figure 2.3: Use Case Diagram

Actors:

- **Customer:** Represents the cardholder who initiates a transaction. This includes different customer types:
    - **Regular:** Standard cardholders.
    - **Elite:** Customers with higher spending limits and potential benefits.
    - **Premium:** Customers with the highest level of privileges and benefits.
- **Merchant:** Represents the business receiving payment from the customer.
- **Payment Processor:** Represents the entity that handles the financial aspects of the transaction, such as authorization and settlement.
- **Bank:** The financial institution that issues the credit card and maintains the customer's account.

Use Cases:

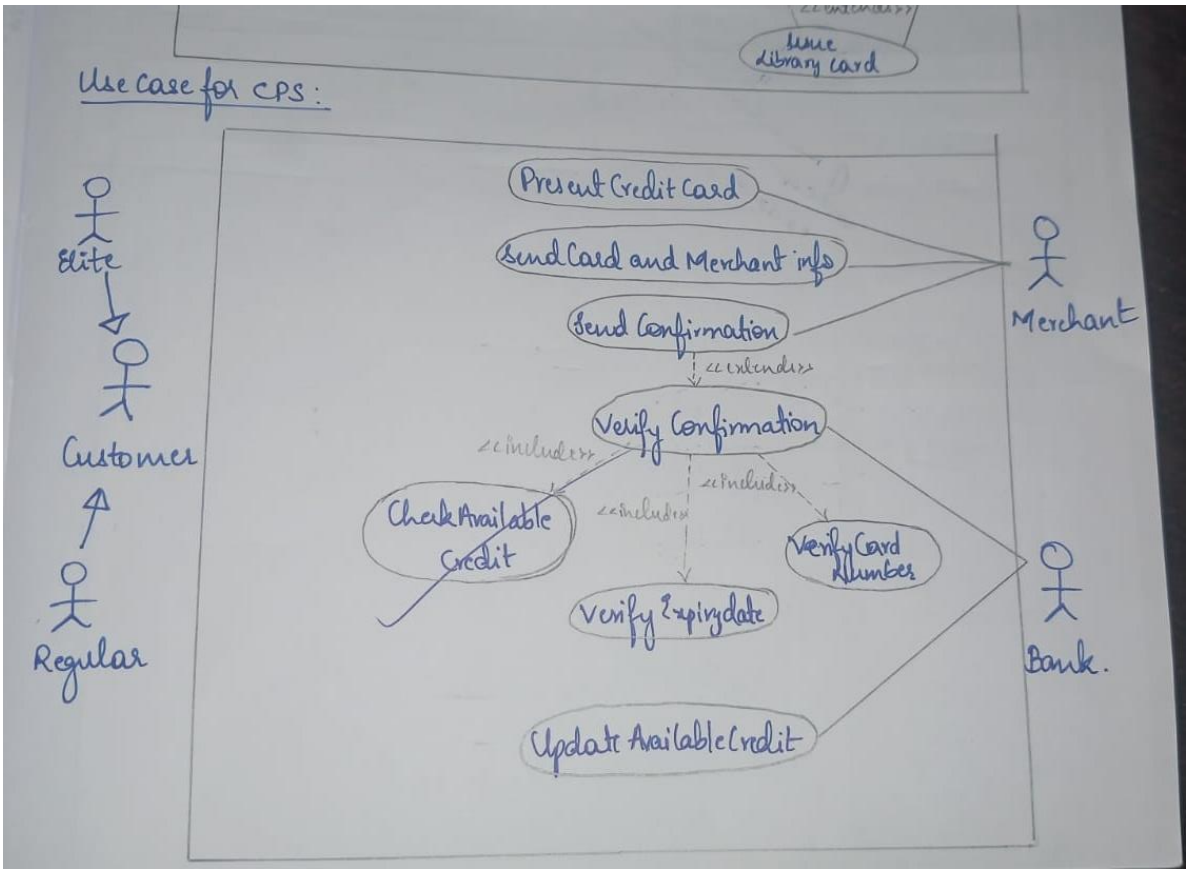
- **Credit Card Processing:** This is the main use case, encompassing all the steps involved in a credit card transaction.

    - **Present Credit Card:** The customer presents their credit card to the merchant.
    - **Send Card and Merchant Info:** The merchant sends the card details and transaction information to the payment processor.
    - **Send Confirmation:** The payment processor sends a confirmation to the merchant.
    - **Verify Confirmation:**
        - **Includes:**
            - **Verify Card Number:** The system checks if the card number is valid.
            - **Verify Expiry Date:** The system checks if the card is not expired.
        - **Extends:** This use case can be extended to include additional verification steps, such as fraud checks or security measures.
- **Check Available Credit:** The payment processor checks if the customer has sufficient credit available for the transaction.
- **Update Available Credit:** The payment processor updates the customer's credit limit after a successful transaction.

Relationships:

- **Includes:** Represents a dependency between use cases, where one use case includes the functionality of another. For example, "Verify Confirmation" includes "Verify Card Number" and "Verify Expiry Date."
- **Extends:** Represents an optional extension to a use case. For example, "Verify Confirmation" can be extended to include additional verification steps.
- **Inheritance:** Represents a hierarchical relationship between actors, where one actor inherits the characteristics of another. For example, "Elite" and "Premium" inherit from "Customer."

Overall, this use case diagram provides a high-level overview of the key functionalities and interactions involved in a credit card processing system. It helps to visualize the roles of different actors, the flow of information between them, and the various steps involved in a typical transaction.


Issue Library card
Use case for CPS:
Present Credit Card
Send Card and Merchant info
Send Confirmation
Verify Confirmation
Check Available Credit
Verify Card Numbers
Verify Expiry date
Update Available Credit
Elite
Customer
Regular
Merchant
Bank.

SEQUENCE DIAGRAM

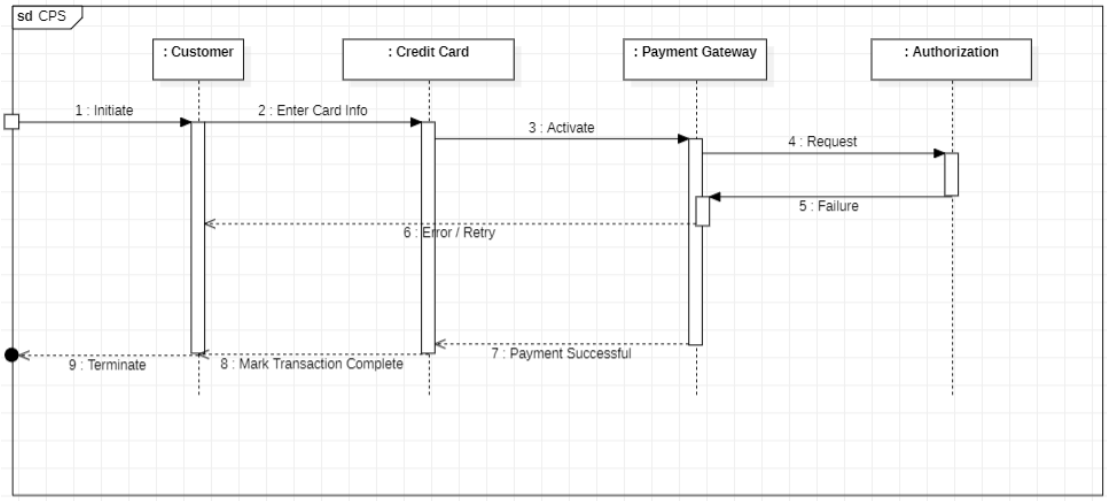


Figure 2.4: Sequence Diagram

Objects:

- **Customer:** The entity initiating the transaction.
- **Credit Card:** The payment instrument used in the transaction.
- **Payment Gateway:** The intermediary that processes the transaction between the customer and the bank.
- **Authorization:** The entity responsible for verifying the transaction and authorizing the payment.
- **Transaction:** The record created to represent the successful transaction.

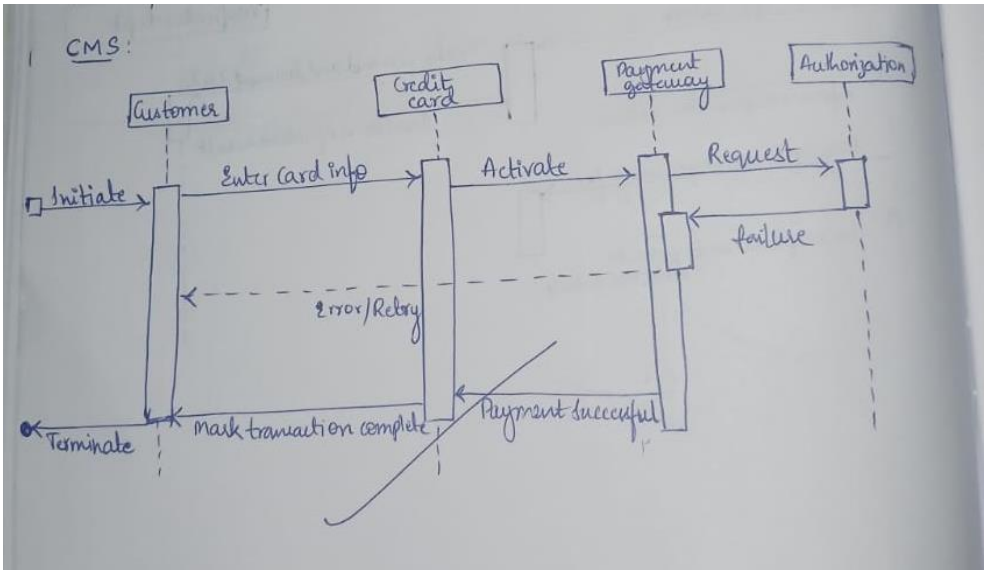
Interactions:

1. **Initiate:** The Customer initiates the transaction process.
2. **Enter Card Info:** The Customer enters the credit card information (number, expiry date, CVV, etc.).
3. **Activate:** The Credit Card is activated for use in the transaction.
4. **Request:** The Credit Card sends a request for authorization to the Payment Gateway.
5. **Failure:** In case of an error (e.g., invalid card information, insufficient funds), the Payment Gateway sends a failure response to the Credit Card.

6. **Authorization Granted:** If the authorization is successful, the Payment Gateway grants authorization to the Credit Card.
7. **Create Transaction:** A new Transaction object is created to record the successful transaction.
8. **Error/Retry:** If the authorization fails, the process may be retried (e.g., by entering card information again).
9. **Transaction Successful:** The Payment Gateway informs the Credit Card that the transaction was successful.
10. **Payment Successful:** The Credit Card informs the Customer that the payment was successful.
11. **Mark Transaction Complete:** The Credit Card marks the transaction as complete.
12. **Terminate:** The transaction process is terminated.

Overall:

This Sequence Diagram illustrates the flow of interactions between the Customer, Credit Card, Payment Gateway, and Authorization entities during a typical credit card transaction. It shows the sequence of messages exchanged and the different states the system goes through, from initiation to completion or failure.



ACTIVITY DIAGRAM

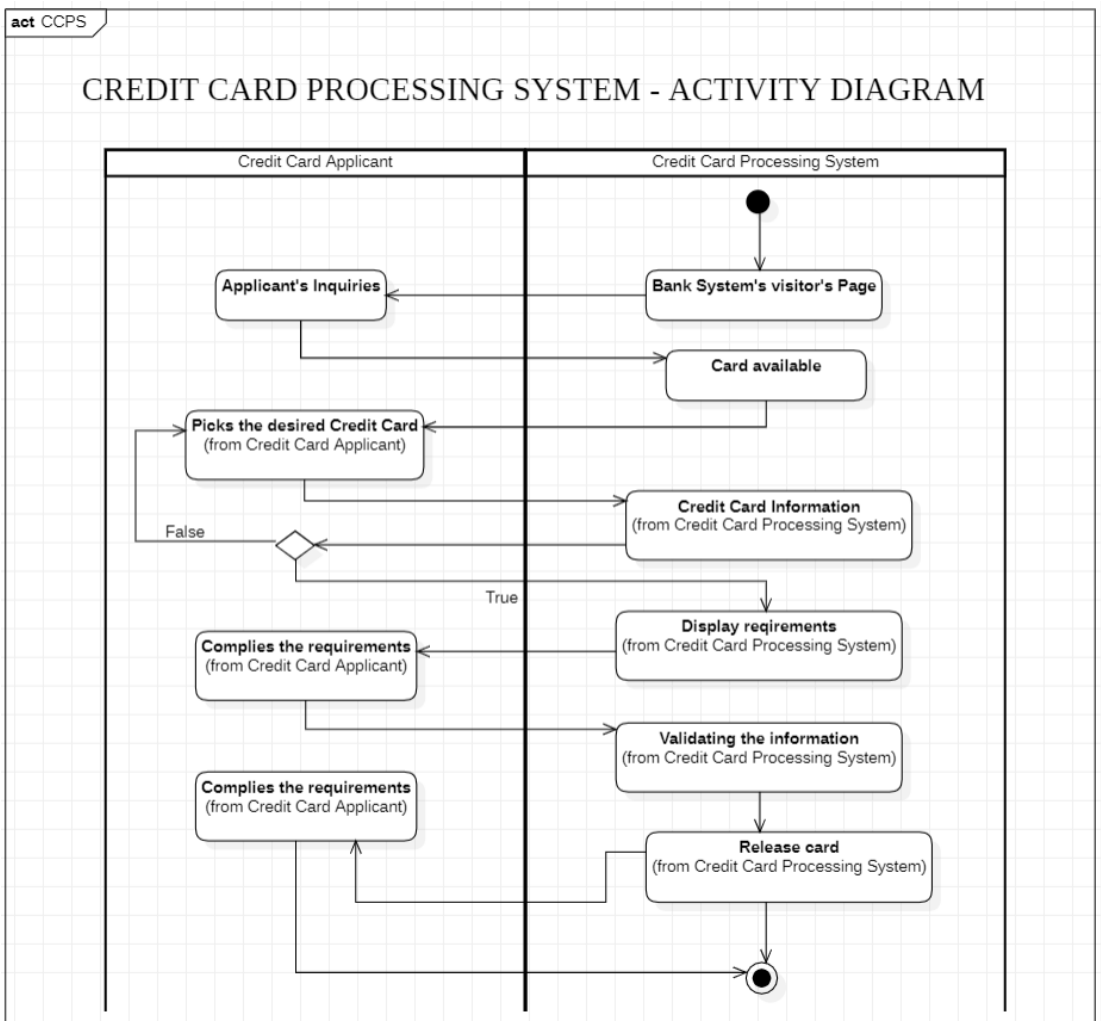
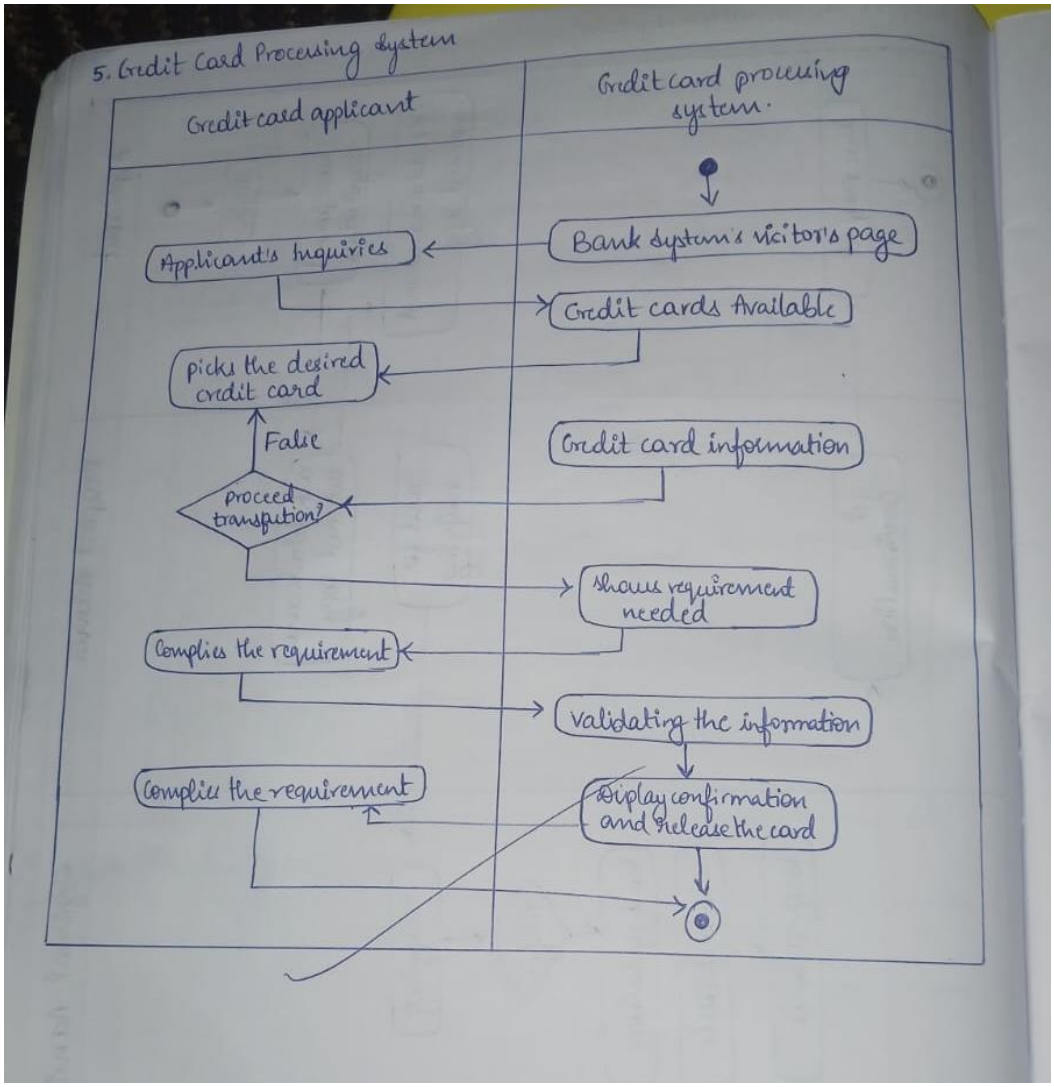


Figure 2.5: Activity Diagram

Swimlanes:

1. **Credit Card Applicant:** This swimlane represents the actions performed by the applicant who is applying for a credit card. These actions include:
    - **Applicant's Inquiries:** The applicant inquires about available credit cards.
    - **Picks the desired Credit Card:** The applicant selects a specific credit card from the options presented.
    - **Complies the requirements:** The applicant provides the necessary information and documents to fulfill the credit card application requirements.

2. **Credit Card Processing System:** This swimlane encompasses the actions performed by the credit card processing system, which includes:
    - **Bank System's visitor's Page:** The system displays the available credit card options to the applicant.
    - **Credit Card Available:** The system presents information about the selected credit card to the applicant.
    - **Credit Card Information:** The system provides details about the credit card and its features.
    - **Shows the requirements needed:** The system displays the requirements that the applicant needs to fulfill to apply for the card.
    - **Validating the information:** The system validates the information provided by the applicant.
    - **Display the confirmed application and release the card:** Upon successful validation, the system confirms the application and informs the applicant about the card issuance process.


5. Credit Card Processing System
Credit card applicant
Credit card processing system.
Applicant's inquiries
Bank system's visitor's page
Credit cards Available
picks the desired credit card
False
Credit card information
proceed transaction?
shows requirement needed
Complies the requirement
validating the information
Complies the requirement
Display confirmation and release the card

CHAPTER – 3
LIBRARY MANAGEMENT SYSTEM

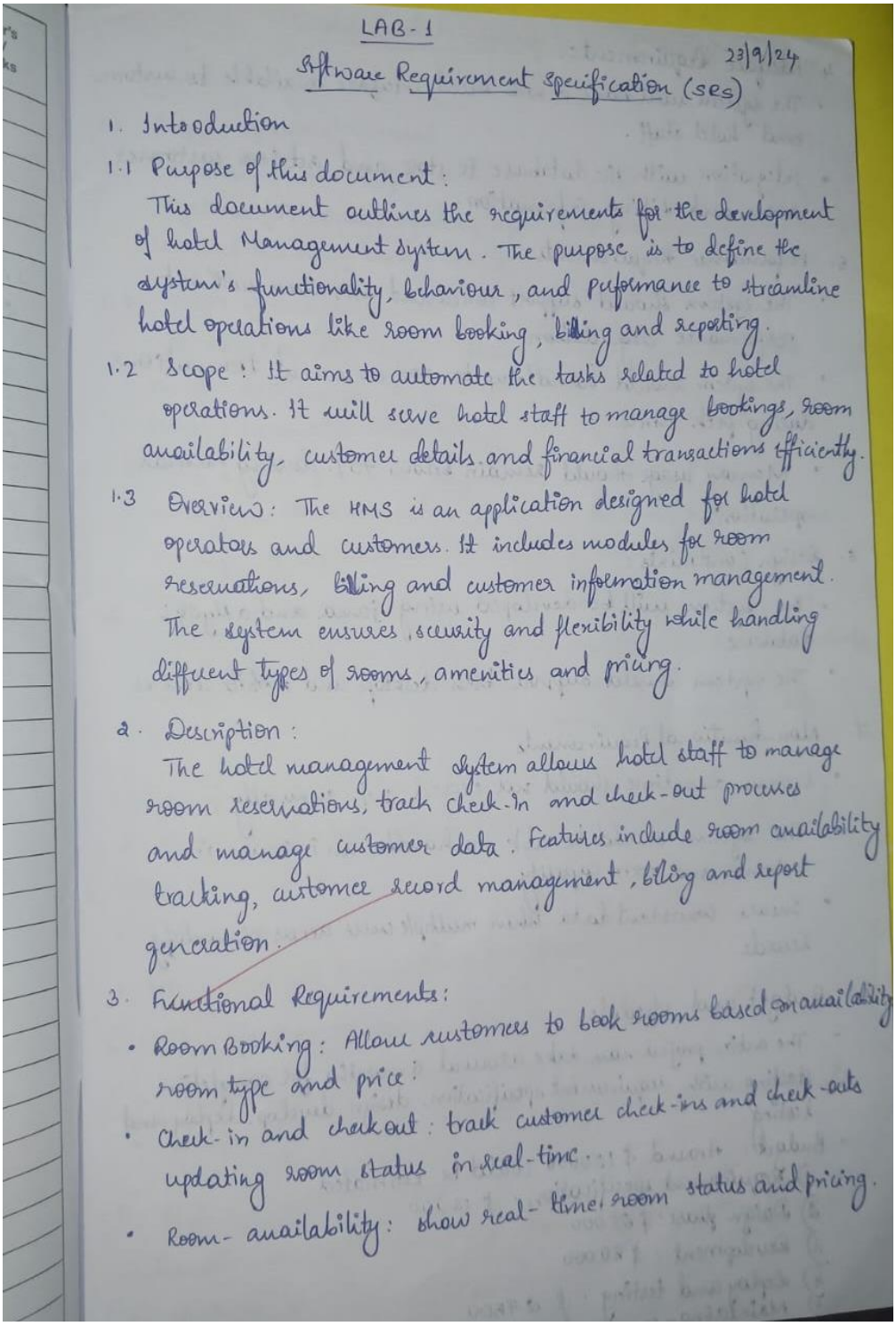
PROBLEM STATEMENT

The objective of this project is to develop an efficient and user-friendly **Library Management System** to streamline library operations and enhance user experience. The proposed system aims to address the following critical functionalities:

1. **Book Management:** Facilitate the addition, removal, and categorization of books, along with real-time tracking of book availability to ensure efficient inventory management.
2. **User Management:** Enable registration and management of users, including librarians, members, and administrators, with appropriate access permissions and roles.
3. **Borrowing and Return Management:** Provide seamless processes for issuing and returning books, including automated due date tracking, fine calculation, and reminders for overdue books.
4. **Catalog Search:** Implement advanced search functionalities, such as keyword-based, genre-based, and author-based search, to help users locate books quickly.
5. **Reservation System:** Allow users to reserve books currently unavailable, with automated notifications when the reserved items become available.
6. **Reports and Analytics:** Generate detailed reports on library usage, borrowing trends, and book inventory to aid in decision-making and resource allocation.
7. **Security and Data Integrity:** Ensure secure access to the system, maintain data accuracy, and implement backup mechanisms to protect against data loss.

This system will integrate these functionalities into a cohesive platform, reducing manual effort, improving operational efficiency, and enhancing the overall user experience for library members and staff.

SOFTWARE REQUIREMENTS SPECIFICATION



LAB-1

23/9/24

Software Requirement Specification (SRS)

1. Introduction

1.1 Purpose of this document:
This document outlines the requirements for the development of hotel Management system. The purpose is to define the system's functionality, behaviour, and performance to streamline hotel operations like room booking, billing and reporting.

1.2 Scope: It aims to automate the tasks related to hotel operations. It will serve hotel staff to manage bookings, room availability, customer details and financial transactions efficiently.

1.3 Overview: The HMS is an application designed for hotel operators and customers. It includes modules for room reservations, billing and customer information management. The system ensures security and flexibility while handling different types of rooms, amenities and pricing.

2. Description:
The hotel management system allows hotel staff to manage room reservations, track check-in and check-out processes and manage customer data. Features include room availability tracking, customer record management, billing and report generation.

3. Functional Requirements:
- Room Booking: Allow customers to book rooms based on availability room type and price.
- Check-in and check out: track customer check-ins and check-outs updating room status in real-time.
- Room-availability: show real-time room status and pricing.

4. Interface Requirements:

- The system will have a web-based interface available to customers and hotel staff.
- Integration with the database to store and retrieve customer, room and billing information.

5. Performance Requirements:

- The system should support concurrent user access without performance degradation.
- The system should handle up to 500 concurrent transactions during peek times.
- Memory usage should remain below 70% during normal operation.

6. Design Constraints:

- The system will be developed using java and a MySQL database.
- The system should support both desktop and mobile devices.

7. Non-functional Requirements:

- System downtime should not exceed 1% annually.
- The system should be able to handle an increase in customer and room capacity.
- Ensure consistent data when multiple users access or modify records.

8- Budget and schedule:

- The entire project can take around 6 months for completion starting with requirement specification, design, develop, deploy and testing.
- Budget: Around $150000 could be estimated
  1) Requirement specification: $ 15000
  2) Design phase: $ 25000
  3) Development: $ 80000
  4) Deploy and testing: $ 27500
  5) Maintainance: $ 2500.

UML DIAGRAMS

CLASS DIAGRAM

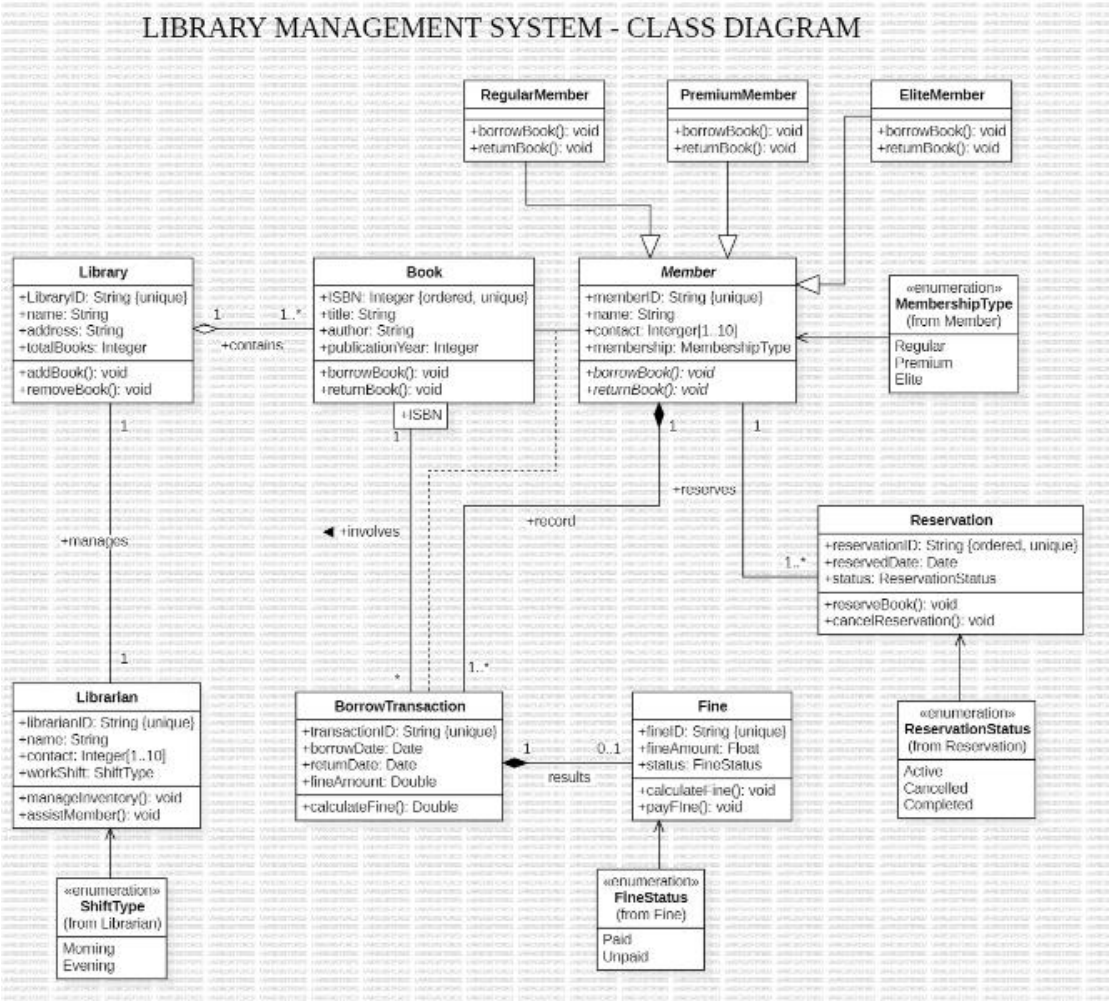


Figure 3.1: Class Diagram

Classes:

- **Library:** Represents a library with attributes like Library ID, name, address, and total books. It has methods to add and remove books.
- **Book:** Represents a book with attributes like ISBN (ordered, unique), title, author, and publication year. It has methods to borrow and return books.
- **Member:** Represents a library member with attributes like member ID, name, contact information, and membership type. It has methods to borrow and return books.
- **RegularMember:** Represents a regular member, inheriting from Member.

- **PremiumMember:** Represents a premium member, inheriting from Member.
- **EliteMember:** Represents an elite member, inheriting from Member.
- **Librarian:** Represents a librarian with attributes like librarian ID, name, contact information, and work shift. It has methods to manage inventory and assist members.
- **BorrowTransaction:** Represents a transaction when a book is borrowed with attributes like transaction ID, borrow date, return date, and fine amount. It has a method to calculate the fine.
- **Reservation:** Represents a reservation with attributes like reservation ID, reserved date, and status. It has methods to reserve and cancel reservations.
- **Fine:** Represents a fine with attributes like fine ID, fine amount, and status. It has methods to calculate and pay the fine.

Associations:

- **Library contains Book:** One library contains many books.
- **Member borrows Book:** One member can borrow many books, and one book can be borrowed by many members.
- **Member reserves Book:** One member can reserve many books, and one book can be reserved by many members.
- **Librarian manages Library:** One librarian manages one library.
- **BorrowTransaction involves Book:** One borrow transaction involves one book.
- **BorrowTransaction results in Fine:** One borrow transaction can result in one fine (if applicable).
- **Reservation records BorrowTransaction:** One reservation can record one borrow transaction.

Multiplicity:

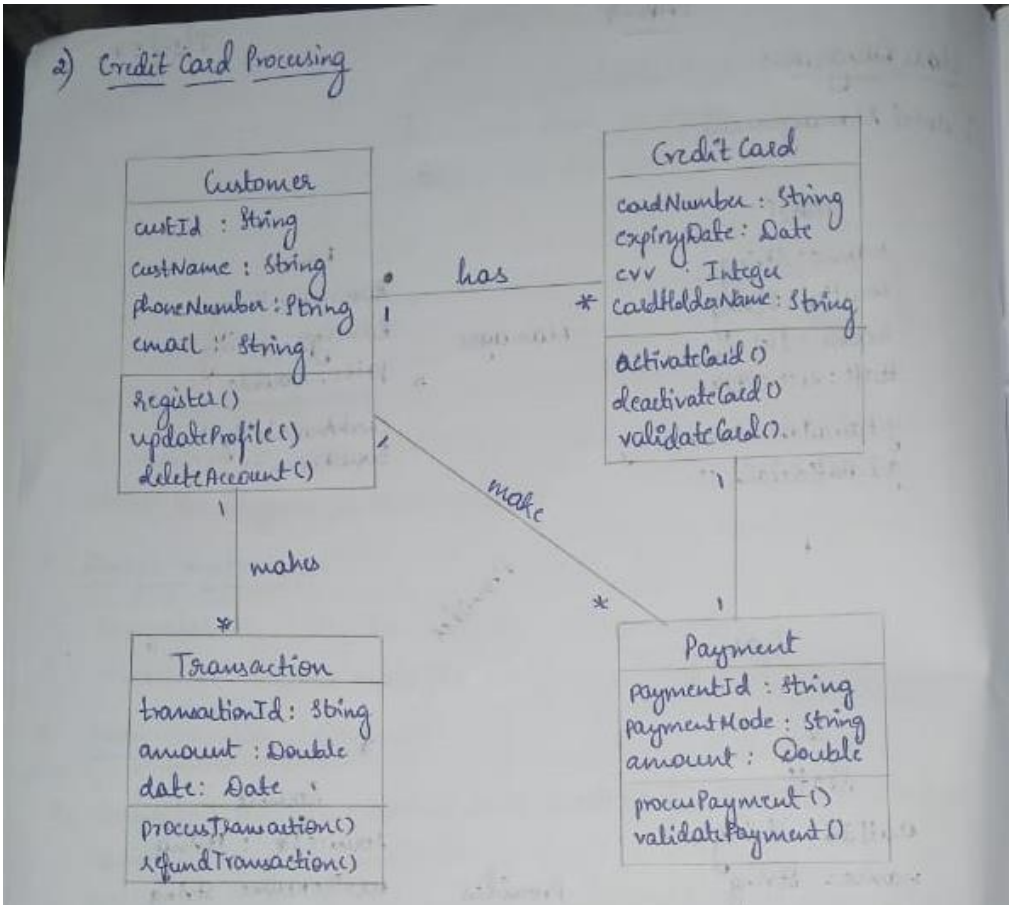
- **Library contains Book:** 1..* (One library contains one or more books)
- **Member borrows Book:** 0..* (One member can borrow zero or more books)
- **Member reserves Book:** 0..* (One member can reserve zero or more books)
- **Librarian manages Library:** 1 (One librarian manages one library)

- **BorrowTransaction involves Book:** 1 (One borrow transaction involves one book)
- **BorrowTransaction results in Fine:** 0..1 (One borrow transaction can result in zero or one fine)
- **Reservation records BorrowTransaction:** 1 (One reservation can record one borrow transaction)

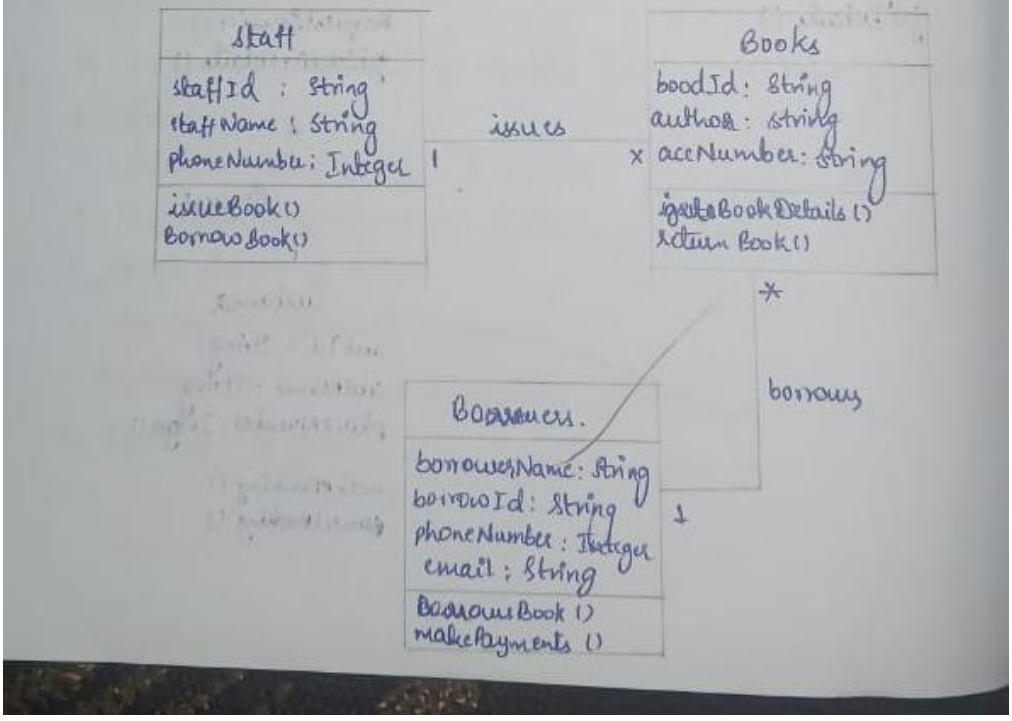
Other Notable Elements:

- **Enumeration MembershipType:** Defines different membership types (Regular, Premium, Elite).
- **Enumeration ShiftType:** Defines different librarian shift types (Morning, Evening).
- **Enumeration ReservationStatus:** Defines different reservation statuses (Active, Cancelled, Completed).
- **Enumeration FineStatus:** Defines different fine statuses (Paid, Unpaid).

2) Credit Card Processing



3) Library Management



STATE DIAGRAM

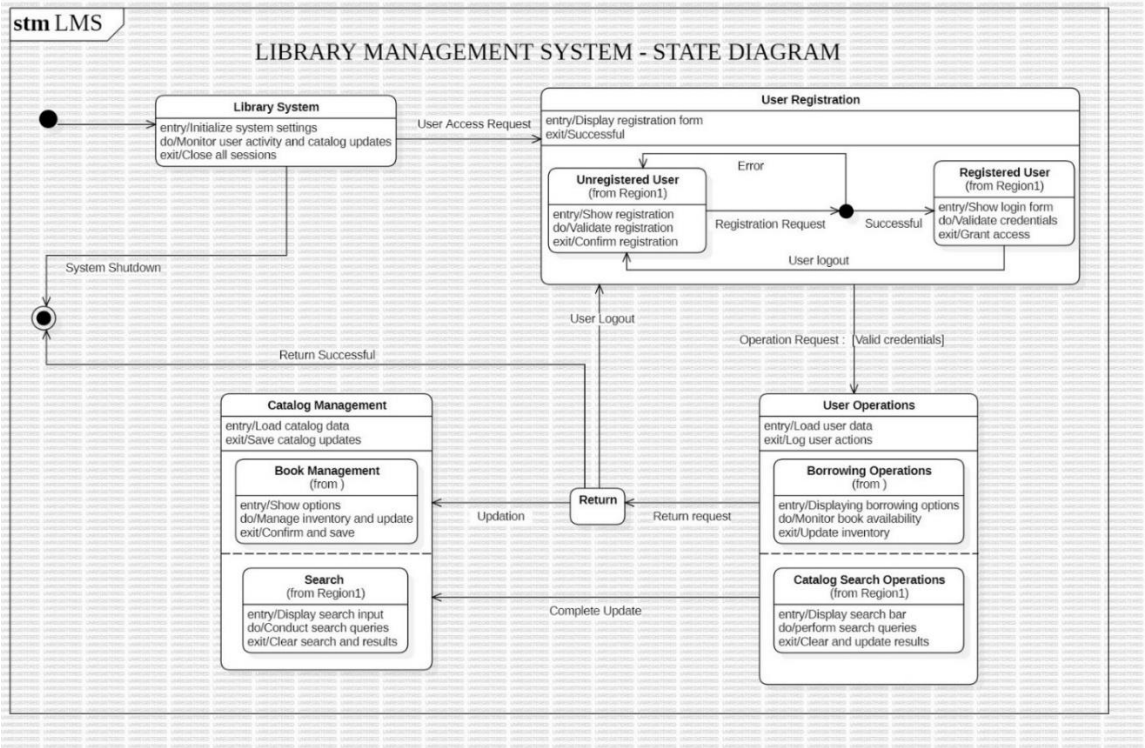


Figure 3.2: State Diagram

States:

- **Library System:** The initial state of the system, where it is initialized and ready for user interaction.
- **User Registration:** The state where a new user is being registered.
- **Unregistered User:** The state for users who have not yet registered.
- **Registered User:** The state for users who have successfully registered and logged in.
- **Catalog Management:** The state where the library catalog is managed, including adding, removing, and updating books.
- **Book Management:** The state where specific book operations are performed, such as borrowing, returning, and reserving.
- **Search:** The state where users can search the library catalog.
- **User Operations:** The state where users can perform various operations, such as borrowing books, returning books, and checking their accounts.
- **Return:** A temporary state during the process of returning a book.

- **Error:** A state indicating an error condition, such as invalid login credentials or failed registration.

Transitions:

- **User Access Request:** Triggered when a user attempts to access the system.
- **Registration Request:** Triggered when a user requests to register.
- **Successful Registration:** Triggered when the registration process is successful.
- **User Logout:** Triggered when a user logs out of the system.
- **Operation Request:** Triggered when a user requests to perform an operation, such as borrowing a book.
- **Return Successful:** Triggered when a book is successfully returned.
- **Complete Update:** Triggered when an update to the catalog or user data is completed.
- **Return Request:** Triggered when a user requests to return a book.

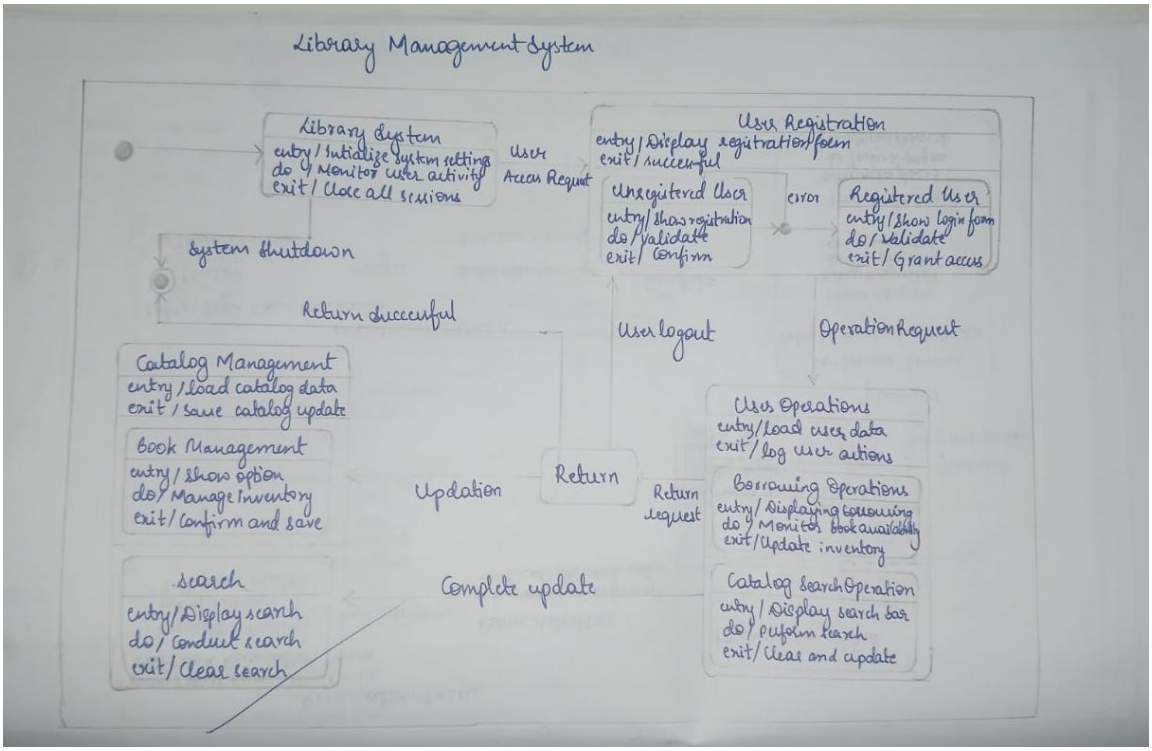
Activities:

- **Initialize system settings:** Activity performed in the Library System state.
- **Display registration form:** Activity performed in the User Registration state.
- **Validate registration:** Activity performed in the Unregistered User state.
- **Confirm registration:** Activity performed in the Unregistered User state.
- **Show login form:** Activity performed in the Registered User state.
- **Validate credentials:** Activity performed in the Registered User state.
- **Grant access:** Activity performed in the Registered User state.
- **Load catalog data:** Activity performed in the Catalog Management state.
- **Save catalog updates:** Activity performed in the Catalog Management state.
- **Show options:** Activity performed in the Book Management state.
- **Manage inventory and update:** Activity performed in the Book Management state.
- **Confirm and save:** Activity performed in the Book Management state.
- **Display search input:** Activity performed in the Search state.
- **Conduct search queries:** Activity performed in the Search state.
- **Clear search and results:** Activity performed in the Search state.
- **Load user data:** Activity performed in the User Operations state.

- **Log user actions:** Activity performed in the User Operations state.
- **Displaying borrowing options:** Activity performed in the Borrowing Operations state.
- **Monitor book availability:** Activity performed in the Borrowing Operations state.
- **Update inventory:** Activity performed in the Borrowing Operations state.

Overall Description:

This state diagram provides a high-level overview of the Library Management System, illustrating the various states and transitions that occur during user interaction and system operation. It covers user registration, login, catalog management, book operations, and search functionalities.



USE CASE DIAGRAM

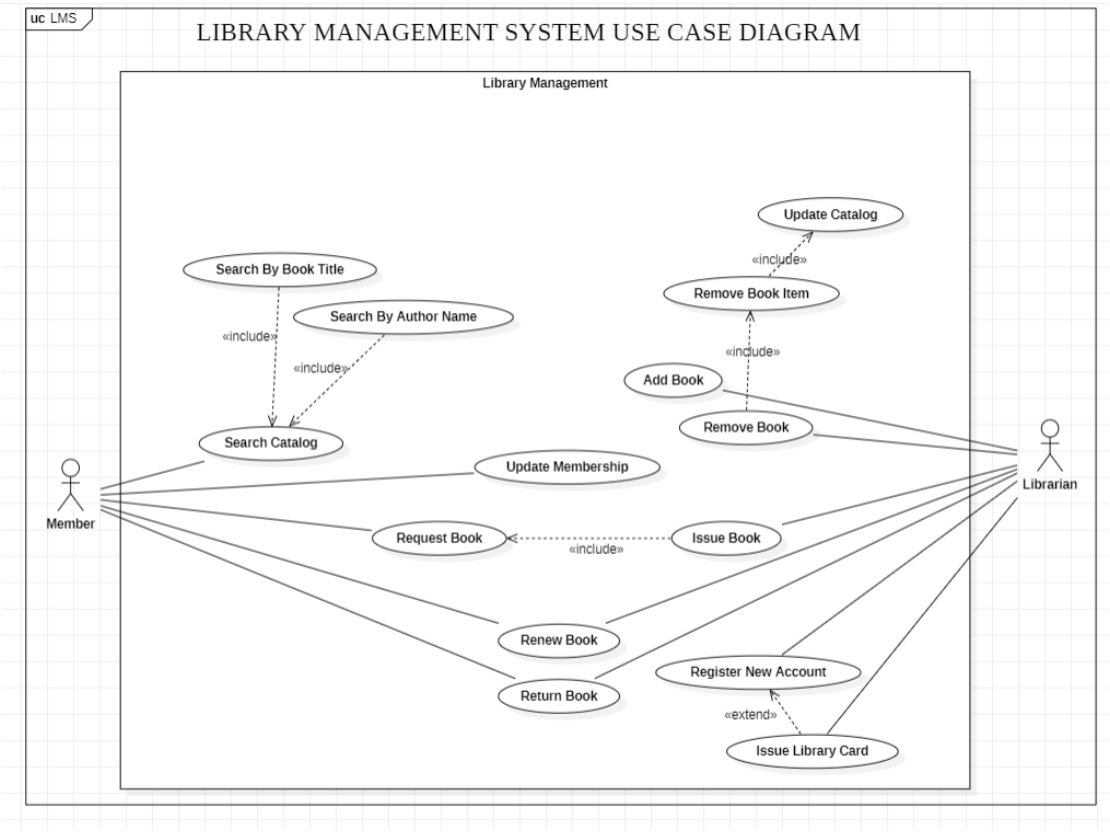


Figure 3.3: Use Case Diagram

Actors:

- **Member:** Represents library users, which can be of different types:
  - **Regular:** Standard members with basic borrowing privileges.
  - **Elite:** Members with enhanced borrowing privileges (e.g., longer loan periods, higher borrowing limits).
  - **Premium:** Members with the highest level of privileges and benefits.
- **Librarian:** Represents the library staff responsible for managing the library operations.

Use Cases:

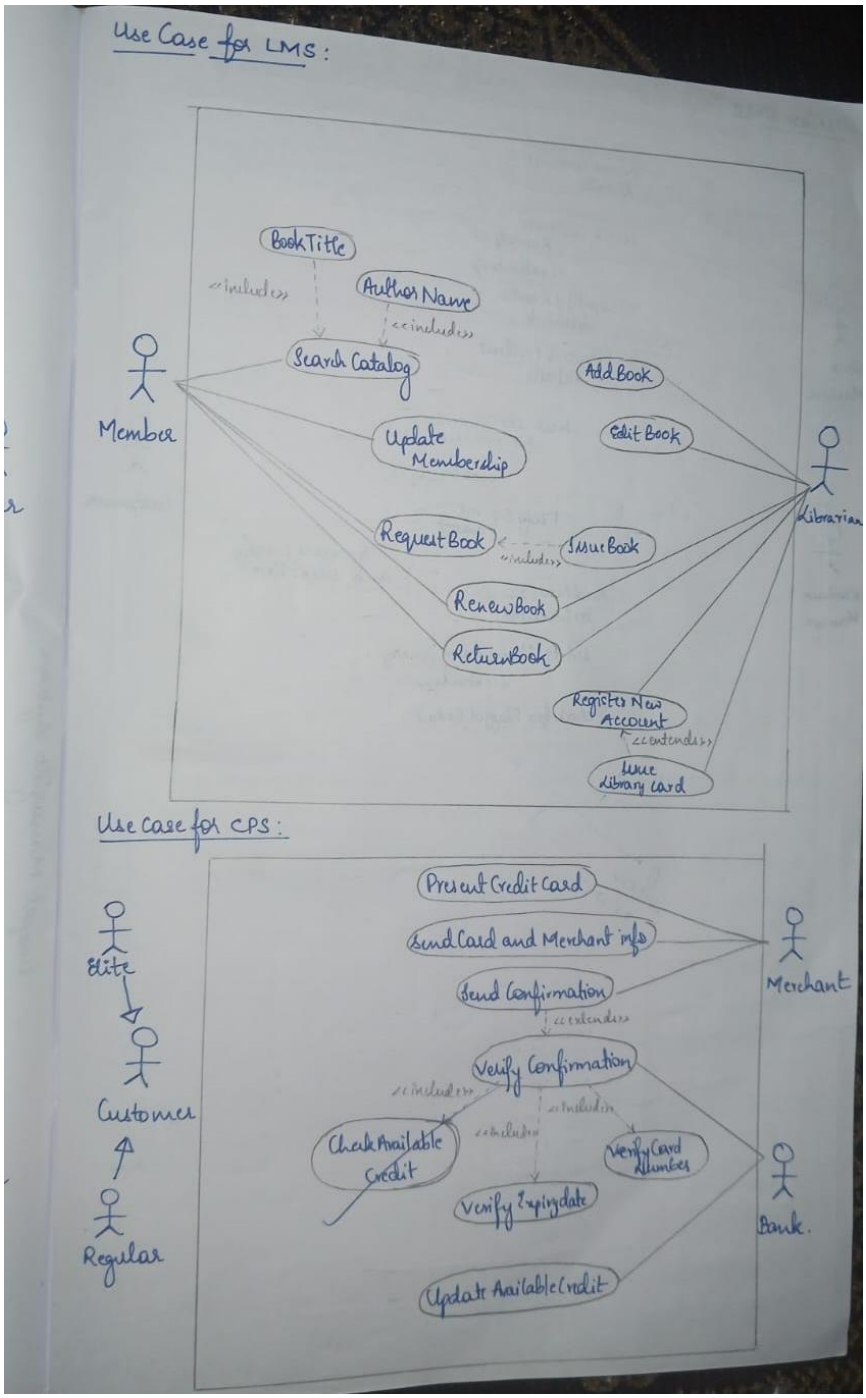
- **Library Management:** This is the main use case, encompassing all the functions related to managing the library.
  - **Update Catalog:** Use case for updating the library catalog.

        - **Includes:**
            - **Add Book Item:** Use case for adding new book items to the catalog.
            - **Remove Book Item:** Use case for removing book items from the catalog.
            - **Edit Item:** Use case for editing the details of a book item.
    - **Add Book:** Use case for adding a new book title to the catalog.
    - **Remove Book:** Use case for removing a book title from the catalog.
    - **Update Membership:** Use case for updating member information or membership types.
- **Search Catalog:** Use case for searching the library catalog.
    - **Includes:**
        - **Search By Book Title:** Use case for searching books by title.
        - **Search By Author Name:** Use case for searching books by author name.
        - **Search By Subject Name:** Use case for searching books by subject name.
- **Request Book:** Use case for members to request a book that is currently unavailable.
- **Renew Book:** Use case for members to renew a borrowed book.
- **Return Book:** Use case for members to return a borrowed book.
- **Issue Book:** Use case for librarians to issue books to members.
- **Register New Account:** Use case for registering new library members.
- **Issue Library Card:** Use case for issuing library cards to new members.

Relationships:

- **Includes:** Represents a dependency between use cases, where one use case includes the functionality of another. For example, "Update Catalog" includes "Add Book Item," "Remove Book Item," and "Edit Item."
- **Inheritance:** Represents a hierarchical relationship between actors, where one actor inherits the characteristics of another. For example, "Elite" and "Premium" inherit from "Member."

Overall, this use case diagram provides a high-level overview of the key functionalities and interactions within the Library Management System. It helps to visualize the roles of different actors, the flow of information between them, and the various services offered by the library.



SEQUENCE DIAGRAM

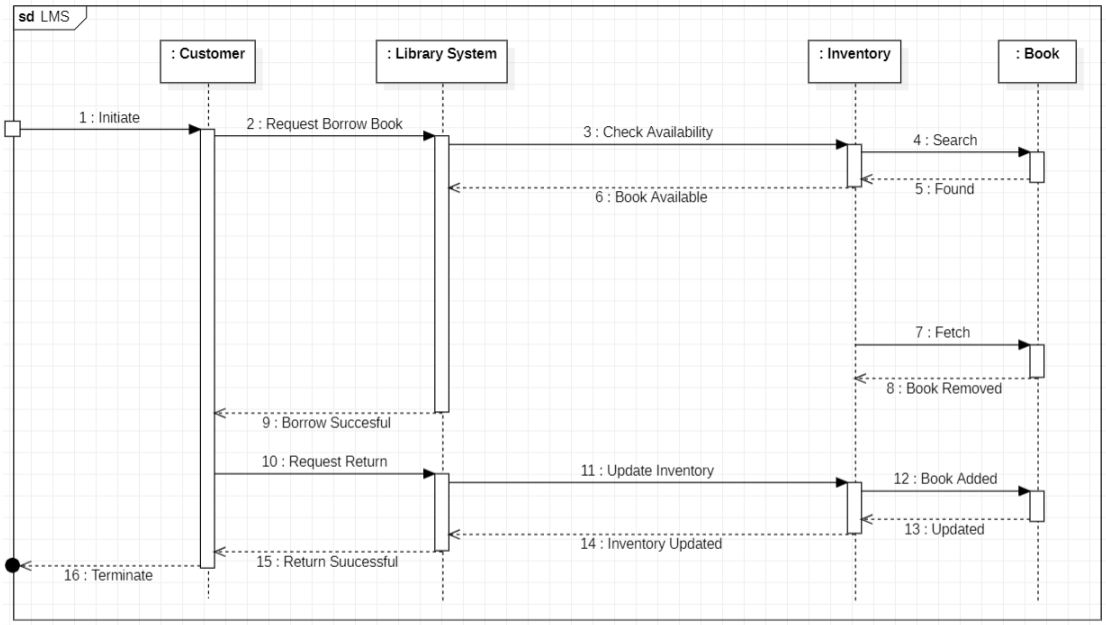


Figure 3.4: Sequence Diagram

Objects:

- **Customer:** The user who interacts with the system.
- **Library System:** The core system that manages library operations.
- **Inventory:** The system that maintains the list of books and their availability.
- **Book:** Represents an individual book in the library.

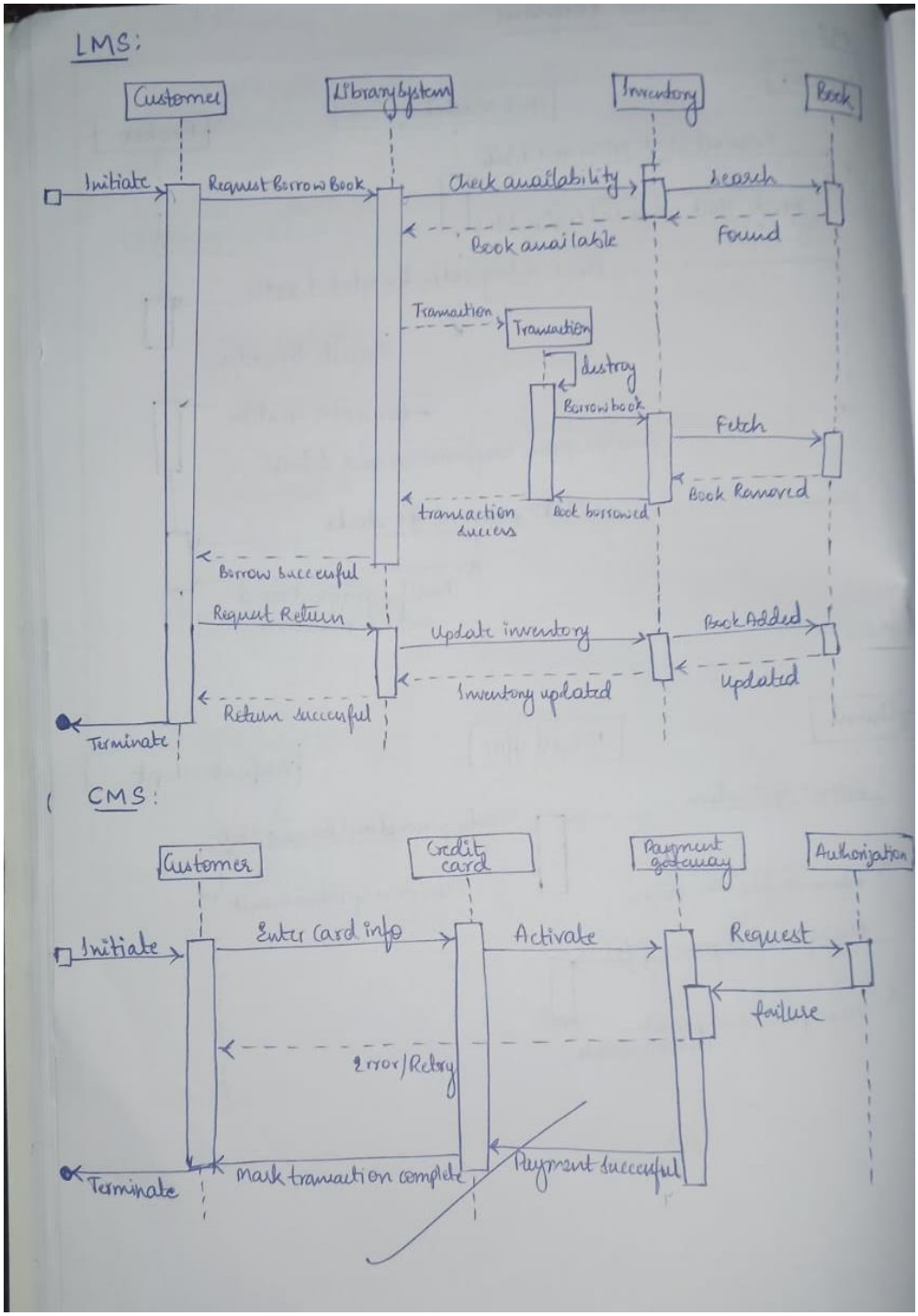
Interactions:

1. **Initiate:** The Customer initiates the interaction by requesting to borrow a book.
2. **Request Borrow Book:** The Customer sends a request to the Library System to borrow a specific book.
3. **Check Availability:** The Library System checks the availability of the requested book with the Inventory.
4. **Search:** The Inventory searches for the requested book.
5. **Found:** The Inventory informs the Library System that the book was found in the inventory.
6. **Book Available:** The Library System informs the Customer that the book is available for borrowing.

7. **Create Transaction:** A new Transaction object is created to record the borrowing process.
8. **Borrow Book:** The Library System sends a request to the Inventory to borrow the book.
9. **Fetch:** The Inventory fetches the book from its storage location.
10. **Book Removed:** The Inventory updates its records to reflect that the book has been borrowed.
11. **Book Borrowed:** The Inventory informs the Library System that the book has been successfully borrowed.
12. **Borrow Successful:** The Customer receives confirmation that the book has been borrowed successfully.
13. **Request Return:** The Customer requests to return the borrowed book.
14. **Update Inventory:** The Library System updates the Inventory to reflect that the book has been returned.
15. **Book Added:** The Inventory adds the returned book back to its available stock.
16. **Inventory Updated:** The Inventory informs the Library System that the inventory has been updated.
17. **Updated:** The Library System acknowledges the inventory update.
18. **Return Successful:** The Library System informs the Customer that the book has been successfully returned.
19. **Terminate:** The interaction is completed.

Overall:

This Sequence Diagram illustrates the flow of interactions between the Customer, Library System, Inventory, and Book objects during the process of borrowing and returning a book. It highlights the creation and destruction of the Transaction object, which is used to manage the borrowing process.


LMS:
Customer
Library system
Inventory
Book
Initiate
Request Borrow Book
Check availability
Search
Book available
Found
Transaction
Transaction
destroy
Borrow book
Fetch
Book Removed
transaction success
Book borrowed
Borrow successful
Request Return
Update inventory
Book Added
Updated
Inventory updated
Return successful
Terminate
CMS:
Customer
Credit card
Payment gateway
Authorization
Initiate
Enter Card info
Activate
Request
failure
Error/Retry
Terminate
Mark transaction complete
Payment successful

ACTIVITY DIAGRAM

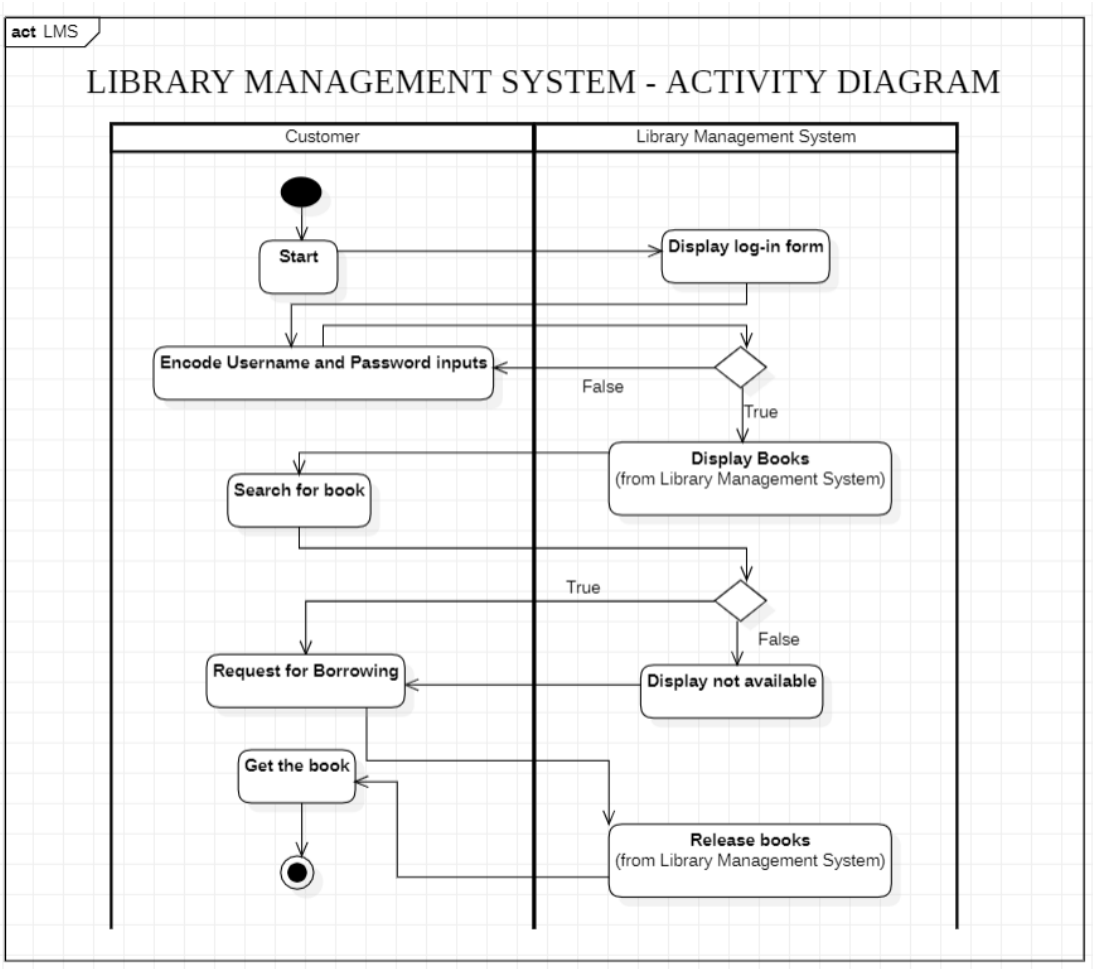
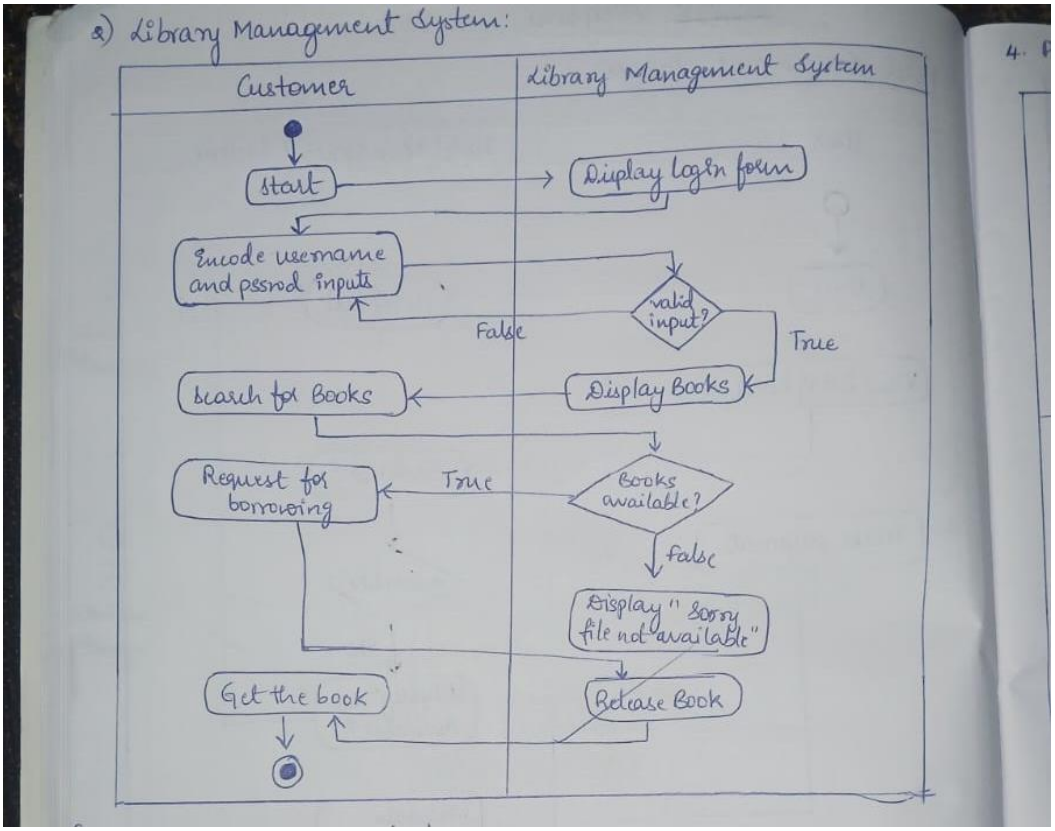


Figure 3.5: Activity Diagram

Swimlanes:

1. **Customer:** This swimlane represents the actions performed by the library user. These actions include:
    - **Start:** The initial state of the user's interaction with the system.
    - **Encode Username and Password Inputs:** The user enters their login credentials.
    - **Search for book:** The user searches for a specific book in the library catalog.
    - **Request for Borrowing:** The user requests to borrow the desired book.
    - **Get the book:** The user receives the borrowed book from the library.

2. **Library Management System:** This swimlane encompasses the actions performed by the library management system itself. These actions include:
    - **Display log-in form:** The system presents the login form to the user.
    - **Display Books:** The system displays the available books to the user after successful login.
    - **Display "Sorry, it's not available":** The system displays an error message if the requested book is unavailable.
    - **Release books:** The system handles the process of returning borrowed books.



CHAPTER – 4

STOCK MAINTENANCE SYSTEM

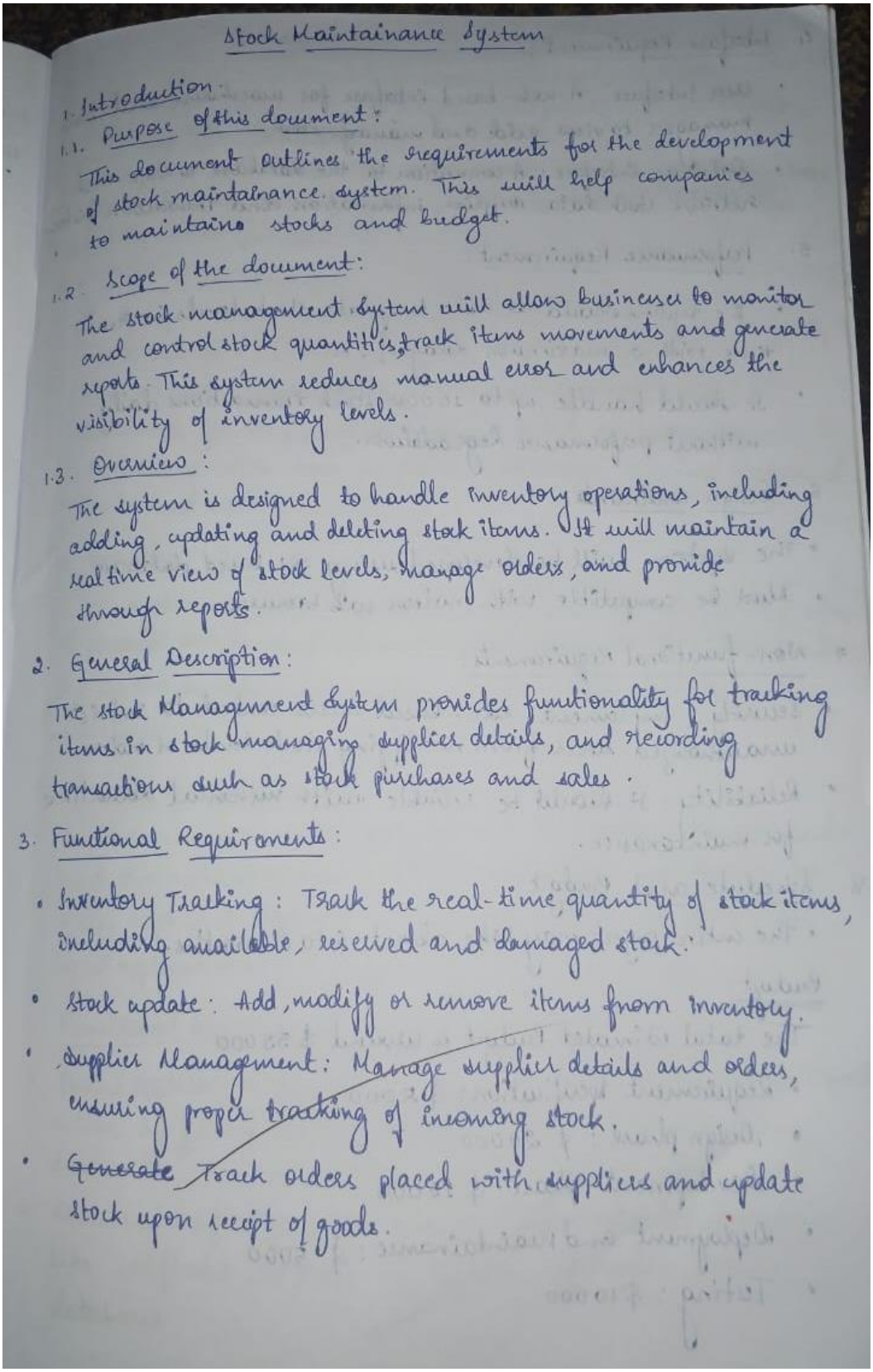
PROBLEM STATEMENT

The objective of this project is to develop a reliable and efficient **Stock Maintenance System** to streamline inventory management and enhance operational efficiency. The proposed system aims to address the following critical functionalities:

1. **Inventory Management:** Enable real-time tracking of stock levels, categorize items, and manage stock additions, deletions, and adjustments to ensure accurate inventory records.
2. **Supplier Management:** Provide tools to maintain supplier information, manage purchase orders, and track delivery schedules to ensure seamless stock replenishment.
3. **Stock Monitoring:** Implement mechanisms to monitor stock usage, set reorder thresholds, and generate alerts for low stock levels to prevent shortages.
4. **Sales and Distribution Tracking:** Facilitate tracking of stock movement related to sales and distribution, maintaining accurate records of outgoing stock and sales performance.
5. **Reporting and Analytics:** Generate detailed reports on inventory levels, stock turnover, and supplier performance, along with analytics to support strategic decision-making.
6. **Integration with Financial Systems:** Allow integration with financial systems to streamline cost management, calculate stock value, and manage accounts related to stock transactions.
7. **Security and Access Control:** Ensure secure access to the system, implement role-based access permissions, and maintain data integrity to prevent unauthorized actions.

This system will centralize and automate stock maintenance processes, reducing manual effort, improving accuracy, and supporting efficient stock management for businesses.

SOFTWARE REQUIREMENTS SPECIFICATION



Stock Maintainance System

1. Introduction:

1.1. Purpose of this document:

This document outlines the requirements for the development of stock maintainance system. This will help companies to maintain stocks and budget.

1.2 Scope of the document:

The stock management system will allow businesses to monitor and control stock quantities, track items movements and generate reports. This system reduces manual error and enhances the visibility of inventory levels.

1.3. Overview:

The system is designed to handle inventory operations, including adding, updating and deleting stock items. It will maintain a real time view of stock levels, manage orders, and provide through reports.

2. General Description:

The stock Management System provides funtionality for tracking items in stock managing supplier details, and recording transactions such as stock purchases and sales.

3. Functional Requirements:

- Inventory Tracking: Track the real-time quantity of stock items, including available, reserved and damaged stock.
- Stock update: Add, modify or remove items from inventory.
- Supplier Management: Manage supplier details and orders, ensuring proper tracking of incoming stock.
- ~~Generate~~ Track orders placed with suppliers and update stock upon receipt of goods.

4. Interface Requirement.

- User Interface: A web-based interface for warehouse and store managers to view, add and manage stock.
- Database interface: A connection to the database to store and retrieve stock data, supplier information and transaction history.

5. Performance Requirement:

- The system should be able to update stock levels in real time with a maximum delay of 1 sec.
- It should handle up to 10000 stock transactions daily without performance degradation.

6. Design Constraints:

- The system will be developed using SQL based database.
- Must be compatible with modern web browser.

7. Non-functional Requirements:

- Security: Implement role-based access control to restrict unauthorized users from modifying stock levels or orders.
- Reliability: It should be reliable with minimal downtime for maintenance.

8. Schedule and Budget:

- The entire project may take about 6 to 8 months.

Budget:

The total estimated Budget is around $ 55 000

- Requirement Specification: $ 8000
- Design phase: $ 20000
- Development phase: $ 12000
- Deployment and maintainance: $ 5000
- Testing: $ 10000

UML DIAGRAMS

CLASS DIAGRAM

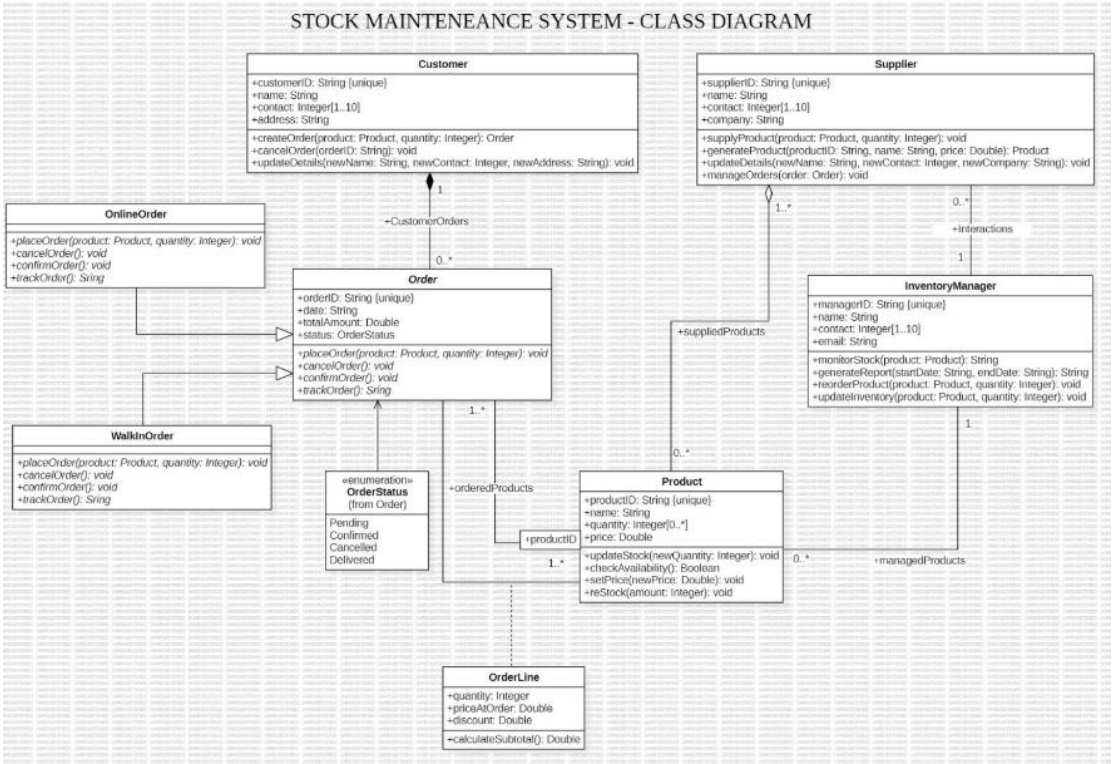


Figure 4.1: Class Diagram

Classes:

- **Customer:** Represents a customer with attributes like customer ID, name, contact, and address. It has methods to create, cancel, and update orders.
- **Supplier:** Represents a supplier with attributes like supplier ID, name, contact, and company. It has methods to supply products and update details.
- **Product:** Represents a product with attributes like product ID, name, quantity, and price. It has methods to update stock, check availability, set price, and restock.
- **Order:** Represents an order with attributes like order ID, total amount, and status. It has methods to place, cancel, and confirm orders.
- **OnlineOrder:** Represents an online order, inheriting from Order.
- **WalkinOrder:** Represents a walk-in order, inheriting from Order.

- **OrderLine:** Represents a line item within an order with attributes like product, quantity, price, discount, and subtotal. It has a method to calculate the subtotal.
- **CustomerOrders:** Represents a collection of orders placed by a customer.
- **InventoryManager:** Represents an inventory manager with attributes like manager ID, name, contact, and email. It has methods to monitor stock, generate reports, reorder products, and update inventory.

Associations:

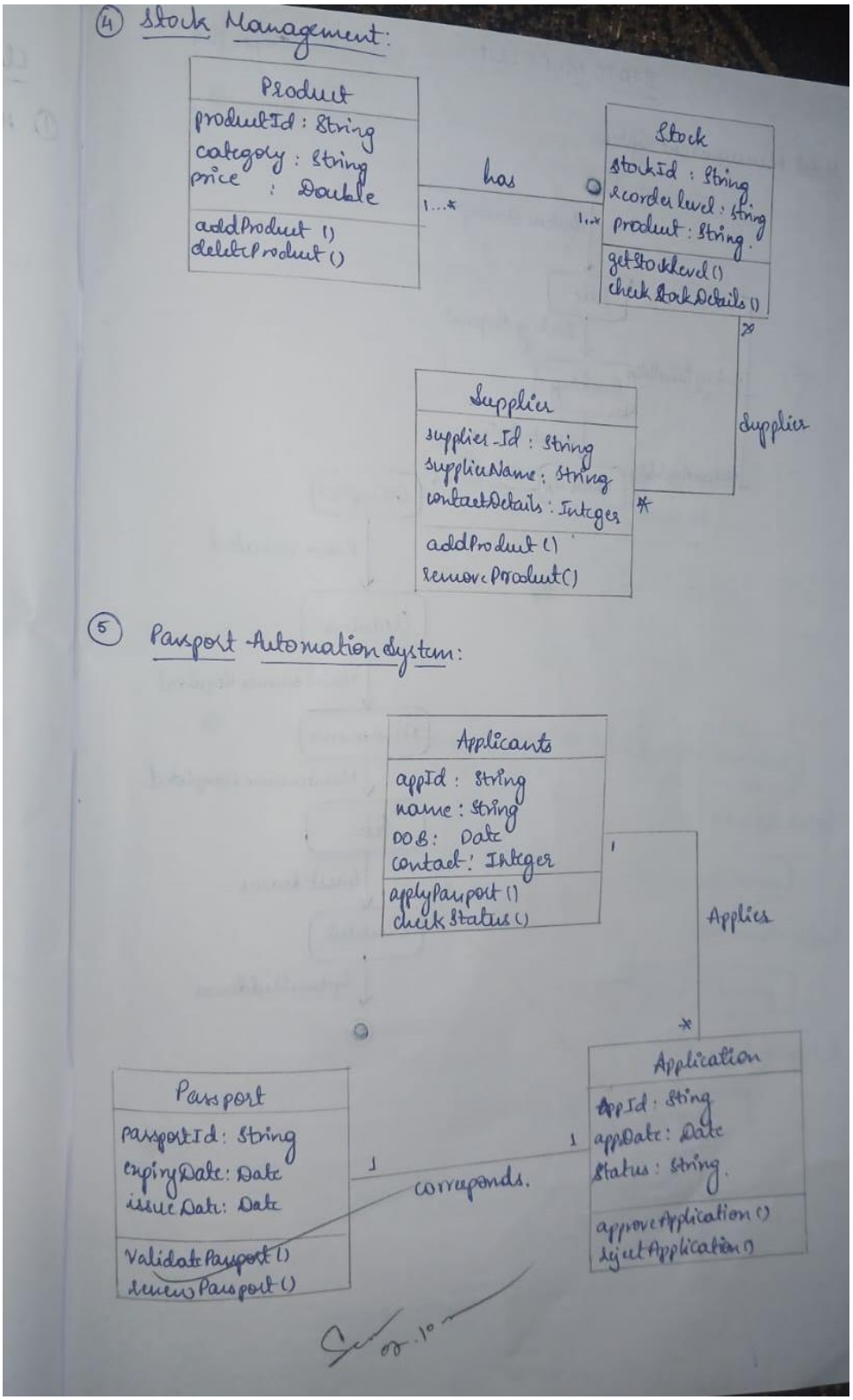
- **Customer places Order:** One customer can place many orders.
- **Supplier supplies Product:** One supplier can supply many products.
- **Order contains OrderLine:** One order can contain many order lines.
- **OrderLine references Product:** One order line references one product.
- **InventoryManager manages Product:** One inventory manager manages many products.
- **InventoryManager manages Order:** One inventory manager manages many orders.
- **Customer has CustomerOrders:** One customer has many customer orders.

Multiplicity:

- **Customer places Order:** 0..* (One customer can place zero or more orders)
- **Supplier supplies Product:** 0..* (One supplier can supply zero or more products)
- **Order contains OrderLine:** 1..* (One order can contain one or more order lines)
- **OrderLine references Product:** 1 (One order line references one product)
- **InventoryManager manages Product:** 1..* (One inventory manager manages one or more products)
- **InventoryManager manages Order:** 1..* (One inventory manager manages one or more orders)
- **Customer has CustomerOrders:** 1 (One customer has one collection of customer orders)

Other Notable Elements:

- **Enumeration OrderStatus:** Defines different order statuses (Pending, Confirmed, Cancelled, Delivered).



STATE DIAGRAM

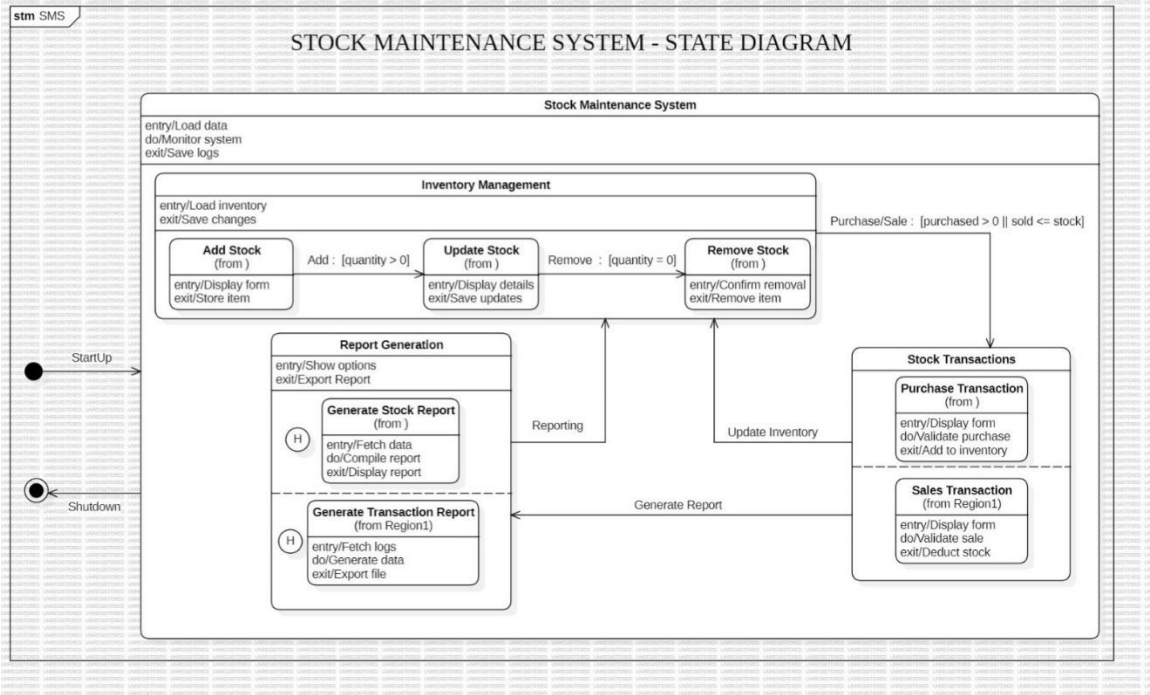


Figure 4.2: State Diagram

States:

- **Stock Maintenance System:** The initial state of the system, where it is initialized and ready for user interaction.
- **Inventory Management:** The state where inventory-related operations are performed, such as adding, removing, and updating stock.
- **Add Stock:** The state where a new stock item is being added.
- **Update Stock:** The state where the quantity of an existing stock item is being updated.
- **Remove Stock:** The state where an existing stock item is being removed.
- **Report Generation:** The state where reports are generated based on stock data.
- **Generate Stock Report:** The state where a stock report is being generated.
- **Generate Transaction Report:** The state where a transaction report is being generated.
- **Stock Transactions:** A superstate encapsulating both purchase and sales transactions.
- **Purchase Transaction:** The state where a purchase transaction is being processed.

- **Sales Transaction:** The state where a sales transaction is being processed.

Transitions:

- **Inventory Management:** Triggered when the system enters the Inventory Management state.
- **Add Stock:** Triggered when a user initiates adding a new stock item.
- **Update Stock:** Triggered when a user initiates updating the quantity of an existing stock item.
- **Remove Stock:** Triggered when a user initiates removing an existing stock item.
- **Report Generation:** Triggered when a user initiates generating a report.
- **Generate Stock Report:** Triggered when the user selects to generate a stock report.
- **Generate Transaction Report:** Triggered when the user selects to generate a transaction report.
- **Stock Transactions:** Triggered when a stock transaction (purchase or sale) is initiated.
- **Purchase Transaction:** Triggered when a purchase transaction is initiated.
- **Sales Transaction:** Triggered when a sales transaction is initiated.

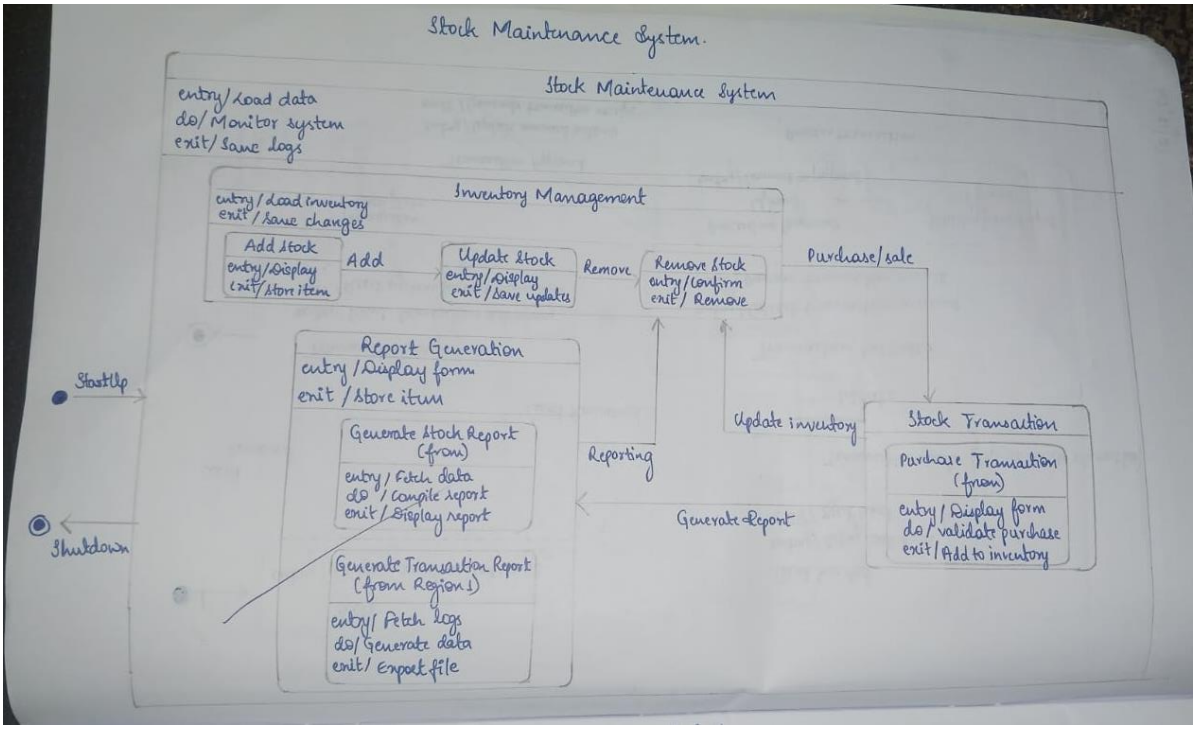
Activities:

- **Load data:** Activity performed in the Stock Maintenance System state.
- **Monitor system:** Activity performed in the Stock Maintenance System state.
- **Save logs:** Activity performed in the Stock Maintenance System state.
- **Load inventory:** Activity performed in the Inventory Management state.
- **Save changes:** Activity performed in the Inventory Management state.
- **Display form:** Activity performed in the Add Stock, Update Stock, and Remove Stock states.
- **Store item:** Activity performed in the Add Stock state.
- **Save updates:** Activity performed in the Update Stock state.
- **Confirm removal:** Activity performed in the Remove Stock state.
- **Remove item:** Activity performed in the Remove Stock state.
- **Show options:** Activity performed in the Report Generation state.

- **Fetch data:** Activity performed in the Generate Stock Report and Generate Transaction Report states.
- **Compile report:** Activity performed in the Generate Stock Report state.
- **Display report:** Activity performed in the Generate Stock Report and Generate Transaction Report states.
- **Export Report:** Activity performed in the Generate Stock Report and Generate Transaction Report states.
- **Display form:** Activity performed in the Purchase Transaction and Sales Transaction states.
- **Validate purchase:** Activity performed in the Purchase Transaction state.
- **Add to inventory:** Activity performed in the Purchase Transaction state.
- **Validate sale:** Activity performed in the Sales Transaction state.
- **Deduct stock:** Activity performed in the Sales Transaction state.

Overall Description:

This state diagram provides a high-level overview of the Stock Maintenance System, illustrating the various states and transitions that occur during inventory management and reporting. It covers stock addition, removal, updates, and generation of stock and transaction reports.


Stock Maintenance System.
Stock Maintenance System
entry/ Load data
do/ Monitor system
exit/ Save logs
Inventory Management
entry / Load inventory
exit / Save changes
Add Stock
entry/ Display
exit/ Store item
Add
Update Stock
entry/ Display
exit / Save updates
Remove
Remove Stock
entry/ Confirm
exit/ Remove
Purchase/ sale
Report Generation
entry / Display form
exit / Store item
Generate Stock Report (from)
entry / Fetch data
do / Compile report
exit / Display report
Generate Transaction Report (from Region)
entry/ Fetch logs
do/ Generate data
exit/ Export file
Reporting
Update inventory
Stock Transaction
Purchase Transaction (from)
entry / Display form
do / validate purchase
exit / Add to inventory
Generate Report
StartUp
Shutdown

USE CASE DIAGRAM

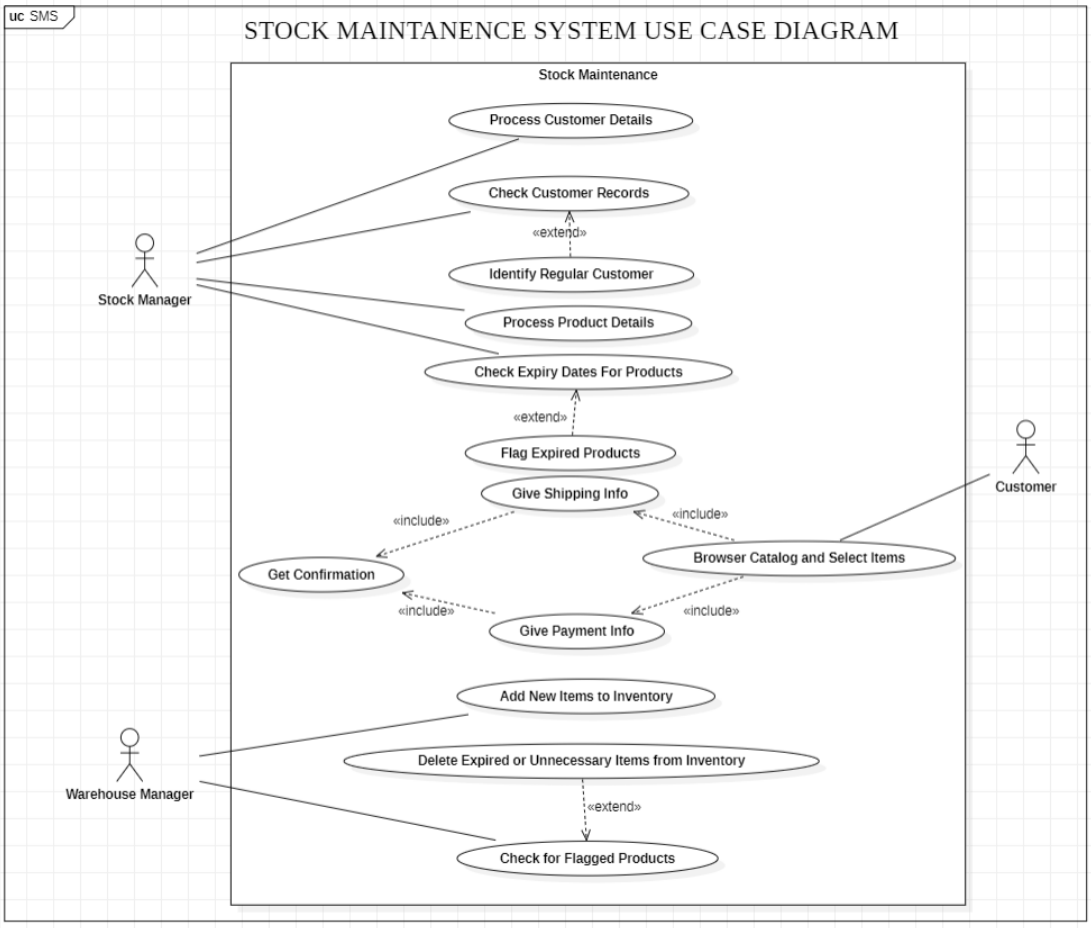


Figure 4.3: Use Case Diagram

Actors:

- **Stock Manager:** Responsible for overseeing the overall stock management process.
- **Control Manager:** Responsible for controlling and managing the stock flow.
- **Warehouse Manager:** Responsible for managing the physical storage and movement of stock within the warehouse.
- **Inventory Manager:** Responsible for maintaining accurate inventory records.
- **Customer:** Represents the end-user who interacts with the system to purchase products.

Use Cases:

- **Stock Maintenance:** This is the main use case, encompassing all the functions related to stock management.
    - **Process Customer Details:** Use case for managing customer information and orders.
        - **<<extend>>** : **Check Customer Records:** An extension to check customer records, potentially for loyalty programs or order history.
        - **<<extend>>** : **Identify Regular Customer:** An extension to identify regular customers for potential discounts or promotions.
    - **Process Product Details:** Use case for managing product information, including inventory levels and pricing.
        - **<<extend>>** : **Check Expiry Dates For Products:** An extension to check for products with upcoming expiry dates.
        - **<<extend>>** : **Flag Expired Products:** An extension to flag products that are expired or nearing their expiry date.
    - **Give Shipping Info:** Use case for providing shipping information to customers.
    - **Get Confirmation:** Use case for obtaining confirmation from customers regarding orders or other actions.
        - **<<include>>** : **Give Payment Info:** A sub-use case included in "Get Confirmation," where customers provide payment information.
        - **<<include>>** : **Browser Catalog and Select Items:** A sub-use case included in "Get Confirmation," where customers browse the catalog and select items for purchase.
    - **Add New Items to Inventory:** Use case for adding new products to the inventory.
    - **Delete Expired or Unnecessary Items from Inventory:** Use case for removing expired or obsolete products from inventory.
        - **<<extend>>** : **Check for Flagged Products:** An extension to check for products that were previously flagged (e.g., for expiry) and remove them accordingly.

Relationships:

- **<<include>>:** Represents a dependency between use cases, where one use case includes the functionality of another. For example, "Get Confirmation" includes "Give Payment Info" and "Browser Catalog and Select Items."
- **<<extend>>:** Represents an optional extension to a use case. For example, "Process Customer Details" can be extended to "Check Customer Records" and "Identify Regular Customer."

Overall, this use case diagram provides a high-level overview of the stock maintenance system. It helps to visualize the roles of different actors, the flow of information between them, and the various functions involved in managing stock inventory and customer interactions.

SEQUENCE DIAGRAM

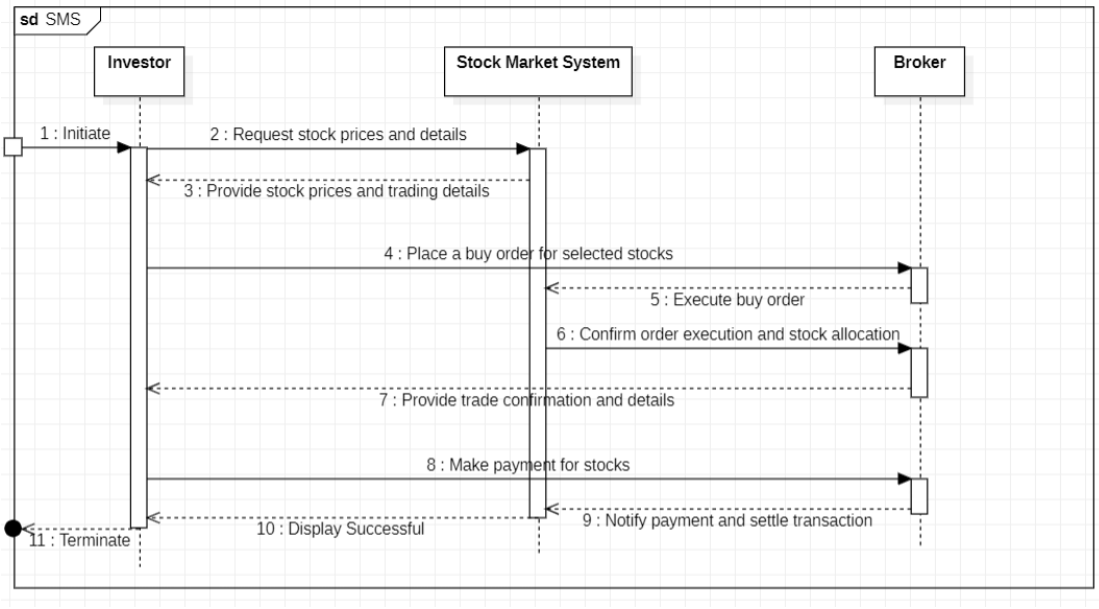


Figure 4.4: Sequence Diagram

Objects:

- **Investor:** The entity initiating the stock trading process.
- **Stock Market System:** The system that manages stock orders and transactions.
- **Broker:** The intermediary that facilitates the trade between the Investor and the Stock Market System.

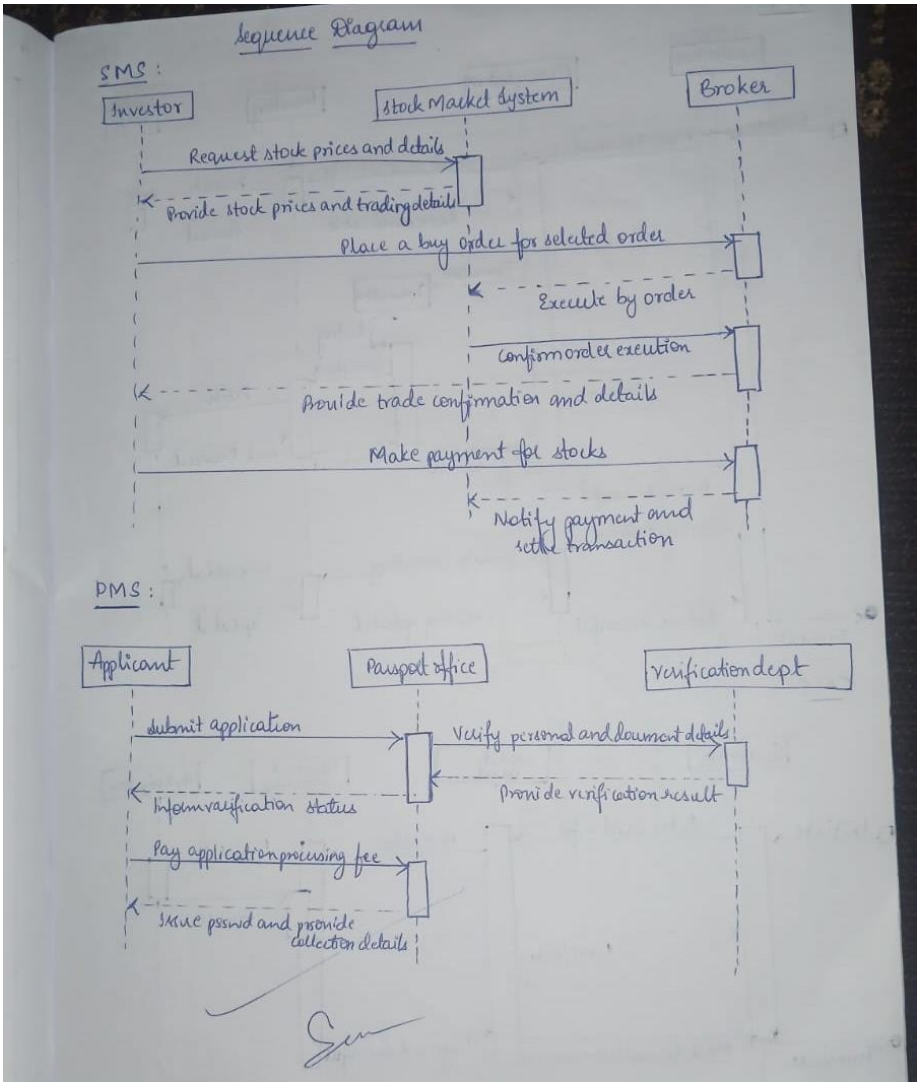
Interactions:

1. **Initiate:** The Investor initiates the stock trading process.
2. **Request stock prices and details:** The Investor requests information about stock prices and trading details from the Stock Market System.
3. **Provide stock prices and trading details:** The Stock Market System provides the requested information to the Investor.
4. **Place a buy order for selected stocks:** The Investor places a buy order for the selected stocks through the Broker.
5. **Execute buy order:** The Broker executes the buy order on behalf of the Investor through the Stock Market System.

6. **Confirm order execution and stock allocation:** The Stock Market System confirms the order execution and allocates the purchased stocks to the Investor's account.
7. **Provide trade confirmation and details:** The Stock Market System provides the Investor with a confirmation of the trade and details of the transaction.
8. **Make payment for stocks:** The Investor makes payment for the purchased stocks to the Broker.
9. **Notify payment and settle transaction:** The Broker notifies the Stock Market System of the payment and settles the transaction.
10. **Display Successful:** The Stock Market System displays a message indicating that the transaction was successful.
11. **Terminate:** The transaction process is terminated.

Overall:

This Sequence Diagram illustrates the flow of interactions between the Investor, Stock Market System, and Broker during a stock purchase transaction. It shows the sequence of messages exchanged and the different states the system goes through, from the initial request for information to the final confirmation of the transaction.


Sequence Diagram
SMS :
Investor
Stock Market System
Broker
Request stock prices and details
Provide stock prices and trading details
Place a buy order for selected order
Execute by order
confirm order execution
Provide trade confirmation and details
Make payment for stocks
Notify payment and settle transaction
PMS :
Applicant
Passport office
verification dept
Submit application
Verify personal and document details
Provide verification result
Inform verification status
Pay application processing fee
Issue pswd and provide collection details

ACTIVITY DIAGRAM

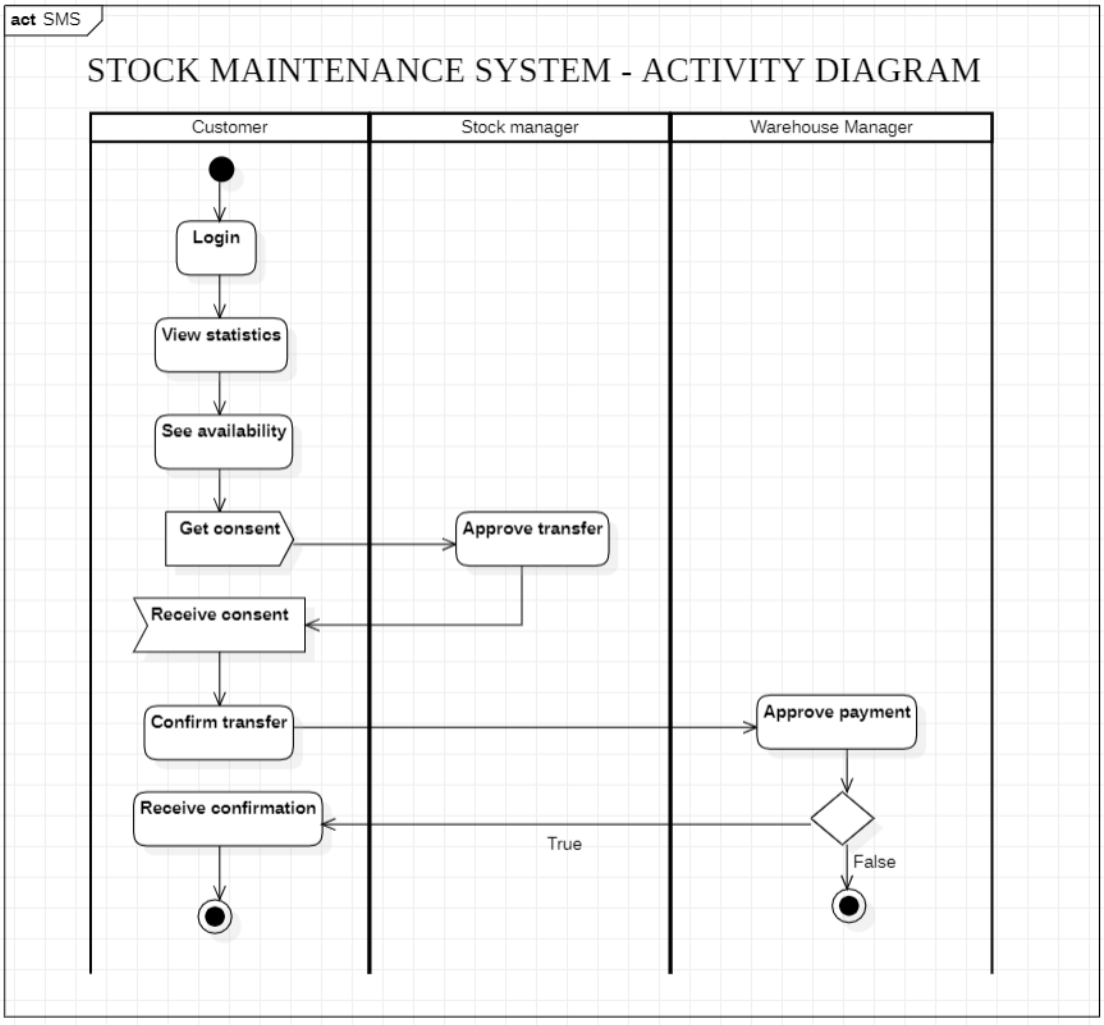


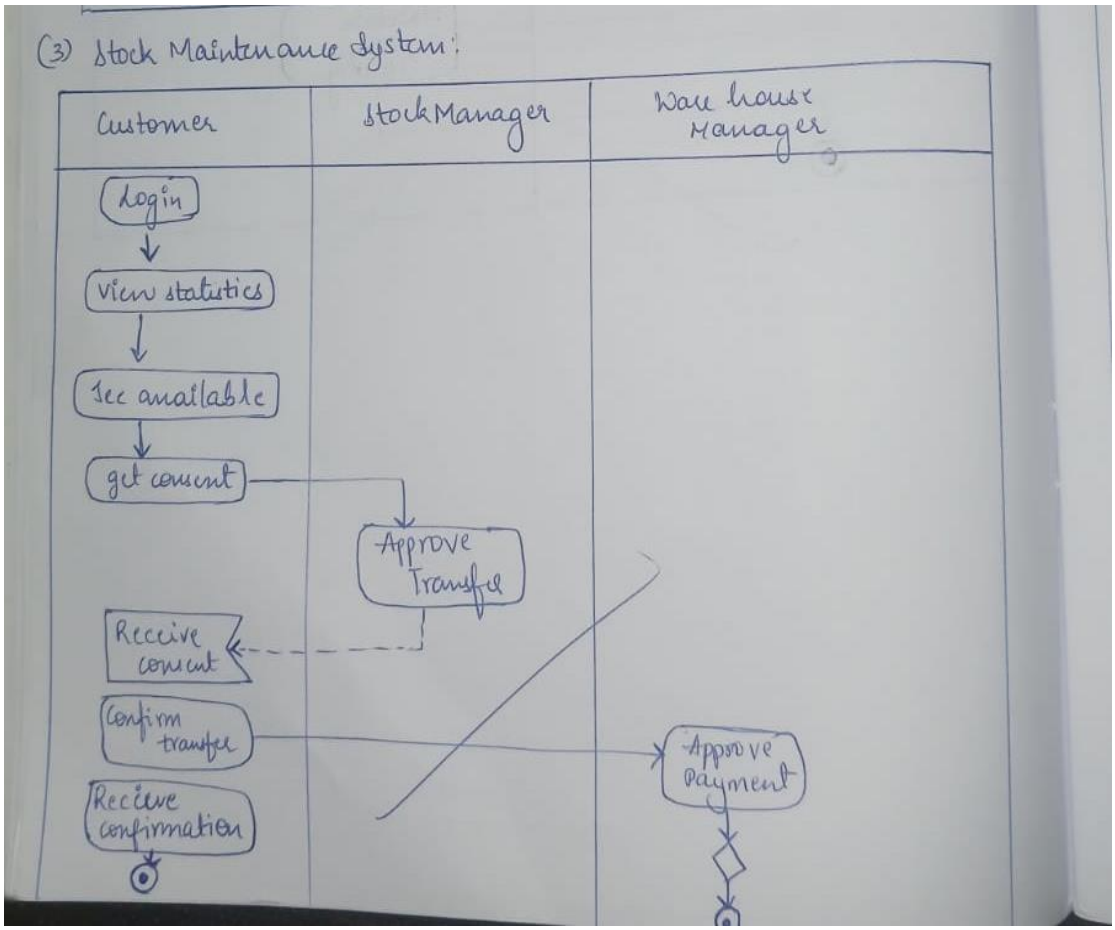
Figure 4.5: Activity Diagram

Swimlanes:

1. **Customer:** This swimlane represents the actions performed by a customer interacting with the system. These actions include:
    - **Login:** The customer logs into the system.
    - **View statistics:** The customer views relevant stock statistics.
    - **See availability:** The customer checks the availability of a particular stock.
    - **Get consent:** The customer obtains consent for a stock transfer.
    - **Receive consent:** The customer receives confirmation of the consent from the relevant parties.

- **Confirm transfer:** The customer confirms the stock transfer.
- **Receive confirmation:** The customer receives confirmation that the stock transfer has been completed.

2. **Stock Manager:** This swimlane represents the actions performed by the stock manager. These actions include:
    - **Approve transfer:** The stock manager approves the requested stock transfer.
3. **Warehouse Manager:** This swimlane represents the actions performed by the warehouse manager. These actions include:
    - **Approve payment:** The warehouse manager approves the payment for the stock transfer



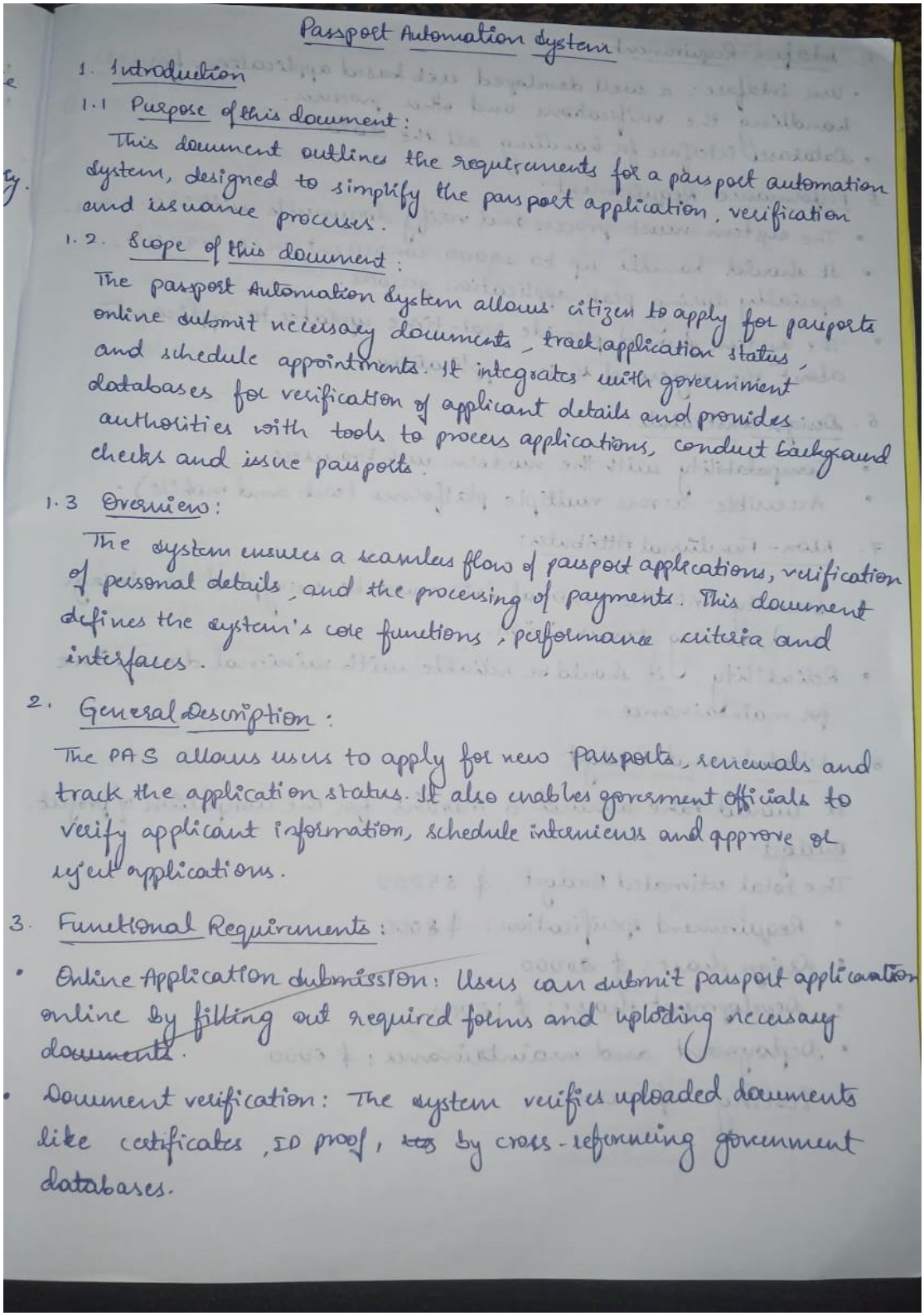
CHAPTER – 5
PASSPORT AUTOMATION SYSTEM

PROBLEM STATEMENT

The objective of this project is to design and implement a comprehensive **Passport Automation System** to streamline the process of passport issuance, renewal, and verification. The proposed system aims to address the following critical functionalities:

1. **Application Management:** Provide an online platform for applicants to submit passport applications, update personal details, and upload required documents.
2. **Appointment Scheduling:** Facilitate real-time scheduling of appointments for document verification, biometrics, and interviews at passport offices.
3. **Document Verification and Validation:** Automate the verification of submitted documents and cross-check applicant information with government databases to ensure accuracy and authenticity.
4. **Processing and Status Tracking:** Enable efficient processing of applications and provide real-time status updates to applicants at every stage of the process.
5. **Passport Issuance and Renewal:** Streamline the issuance and renewal process by automating workflows, generating unique passport numbers, and printing authorized passports.
6. **Security and Fraud Detection:** Implement robust security measures to prevent identity theft, detect fraudulent activities, and safeguard sensitive applicant data.
7. **Reporting and Analytics:** Generate reports on application trends, processing times, and operational performance to aid in decision-making and policy formulation.

SOFTWARE REQUIREMENTS SPECIFICATION



Passport Automation System

1. Introduction

1.1 Purpose of this document:

This document outlines the requirements for a passport automation system, designed to simplify the passport application, verification and issuance processes.

1.2. Scope of this document:

The passport Automation System allows citizen to apply for passports online submit necessary documents, track application status, and schedule appointments. It integrates with government databases for verification of applicant details and provides authorities with tools to process applications, conduct background checks and issue passports.

1.3 Overview:

The system ensures a seamless flow of passport applications, verification of personal details, and the processing of payments. This document defines the system's core functions, performance criteria and interfaces.

2. General Description:

The PAS allows users to apply for new passports, renewals and track the application status. It also enables government officials to verify applicant information, schedule interviews and approve or reject applications.

3. Functional Requirements:

- Online Application Submission: Users can submit passport application online by filling out required forms and uploding necessary documents.
- Document verification: The system verifies uploaded documents like certificates, ID proof, ~~etc~~ by cross-referencing government databases.

4. Interface Requirement:

- User interface: a well developed web based application for handling the verifications and other processes.
- Database interface for handling all the data.

5. Performance Requirement:

- The system must process and verify documents within 3-5 sec.
- It should handle up to 10000 concurrent user requests, specially during peak application periods.
- The system should provide real-time updates to applicants about the progress of their applications.

6. Design constraints:

- Compatability with the modern web browsers.
- Accesible across multiple platforms (web and mobile).

7. Non-Funtional Attributes:

- Security: All the personal data must be encrypted in storage and during transmission.
- Reliability: it should be reliable with minimal downtime for maintainance.

8. Schedule and Budget:

It would take around 6 months for the completion of project.

Budget:

The total estimated Budget: $ 55000

- Requirement specification: $ 8000
- Design phase: $ 20000
- Development phase: $ 12000
- Deployment and maintainance: $ 5000
- Testing: $ 10000

UML DIAGRAMS

CLASS DIAGRAM

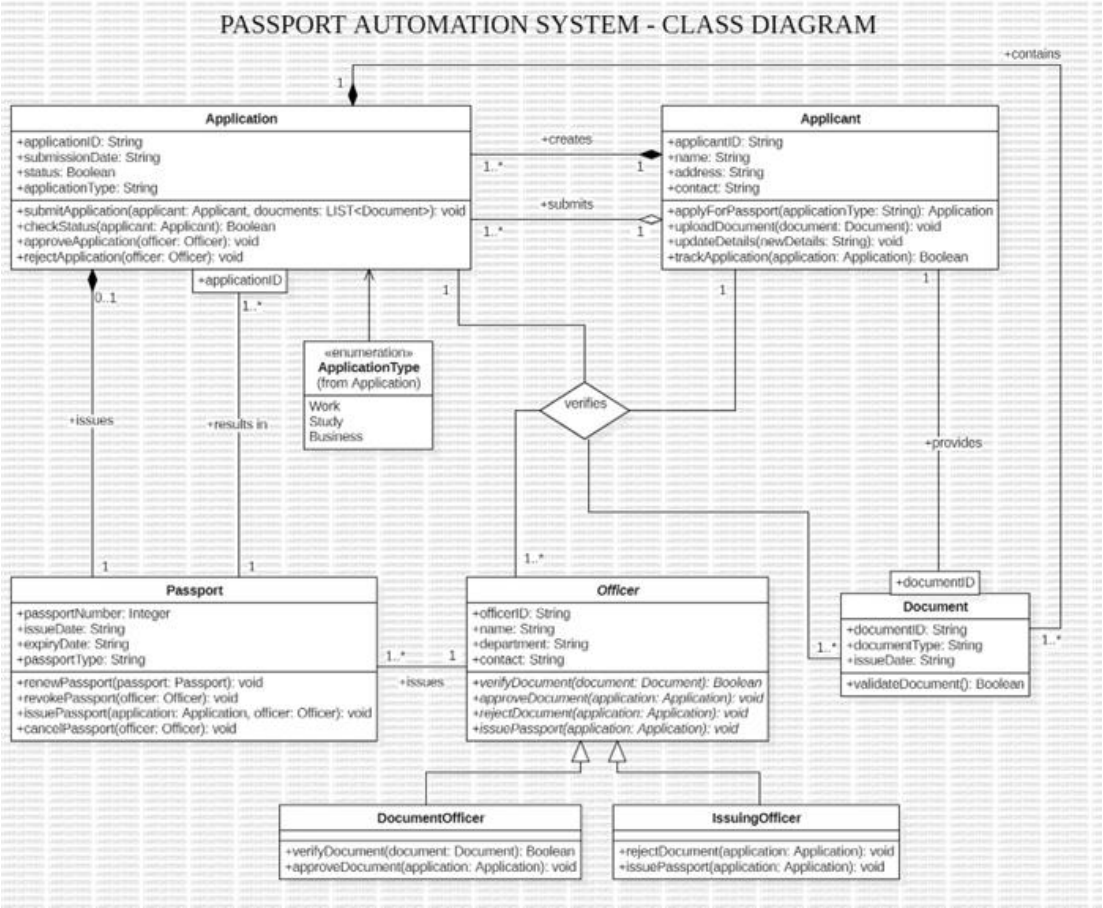


Figure 5.1: Class Diagram

Classes:

- **Application:** Represents an application for a passport with attributes like application ID, submission date, status, and application type. It has methods to submit, check status, approve, and reject applications.
- **Applicant:** Represents an applicant with attributes like applicant ID, name, address, and contact. It has methods to apply for a passport, update details, upload documents, and track the application status.
- **Passport:** Represents a passport with attributes like passport number, issue date, expiry date, and passport type. It has methods to renew, revoke, and cancel passports.

- **Officer:** Represents an officer with attributes like officer ID, name, department, and contact. It has methods to verify documents, approve documents, reject applications, issue passports, and revoke passports.
- **Document:** Represents a document with attributes like document ID, document type, and issue date. It has a method to validate the document.
- **DocumentOfficer:** Represents an officer responsible for verifying documents, inheriting from Officer.
- **IssuingOfficer:** Represents an officer responsible for issuing passports, inheriting from Officer.

Associations:

- **Applicant creates Application:** One applicant can create many applications.
- **Application contains Document:** One application can contain many documents.
- **ApplicationType is an enumeration of Application:** The ApplicationType enumeration is associated with the Application class.
- **Applicant submits Application:** One applicant can submit many applications.
- **Officer verifies Document:** One officer can verify many documents.
- **Officer approves Document:** One officer can approve many documents.
- **Officer rejects Application:** One officer can reject many applications.
- **Officer issues Passport:** One officer can issue many passports.
- **Officer revokes Passport:** One officer can revoke many passports.

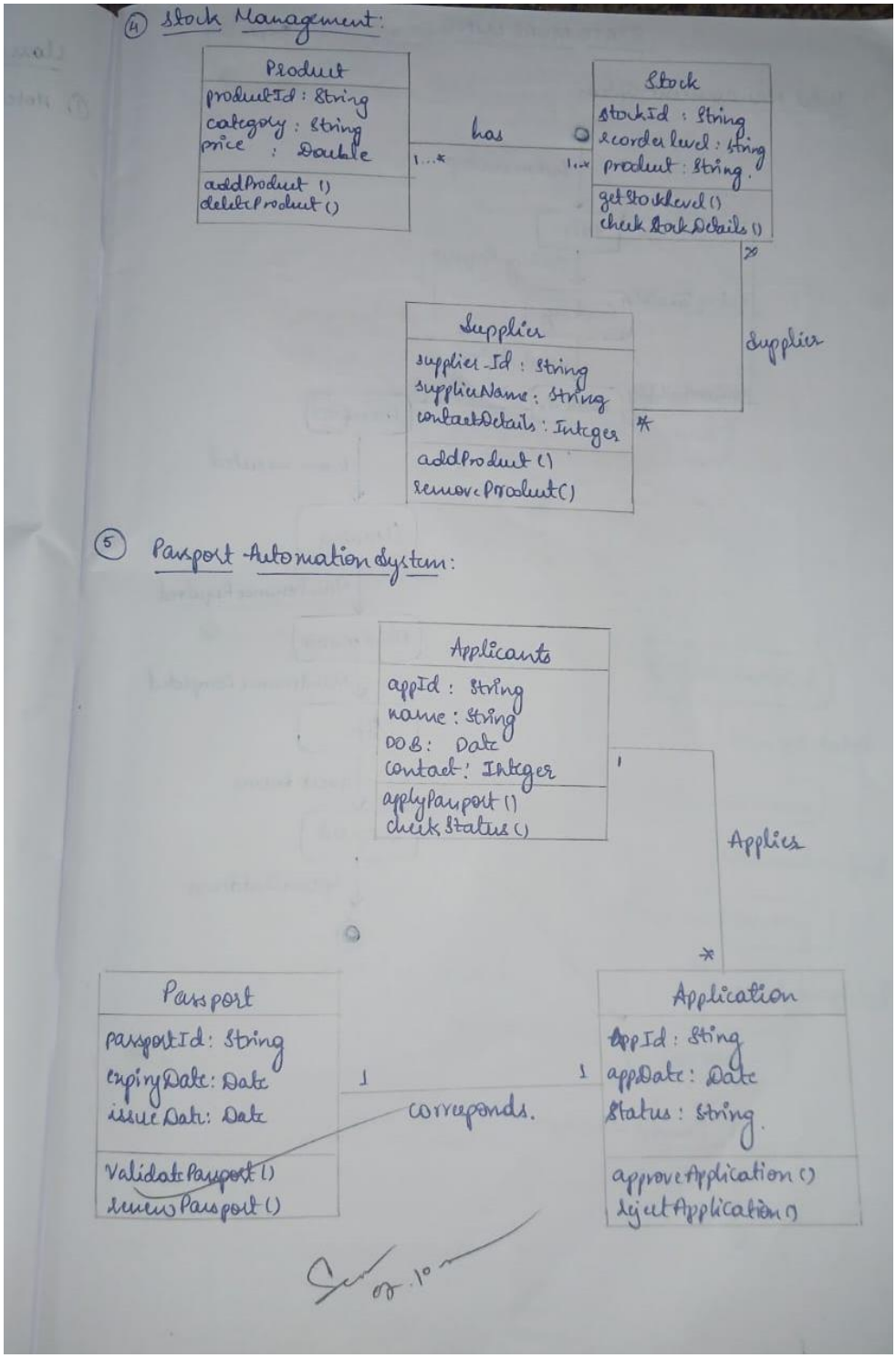
Multiplicity:

- **Applicant creates Application:** 1..* (One applicant can create one or more applications)
- **Application contains Document:** 0..* (One application can contain zero or more documents)
- **Applicant submits Application:** 1..* (One applicant can submit one or more applications)
- **Officer verifies Document:** 1..* (One officer can verify one or more documents)

- **Officer approves Document:** 1..* (One officer can approve one or more documents)
- **Officer rejects Application:** 1..* (One officer can reject one or more applications)
- **Officer issues Passport:** 1..* (One officer can issue one or more passports)
- **Officer revokes Passport:** 1..* (One officer can revoke one or more passports)

Other Notable Elements:

- **Enumeration ApplicationType:** Defines different types of passport applications (Work, Study, Business).


4 Stock Management:
Product
productId : String
category : String
price : Double
addProduct ()
deleteProduct ()
has
1..*
1..*
Stock
stockId : String
recorderLevel : String
product : String.
getStockLevel ()
checkStockDetails ()
Supplier
supplier_Id : String
supplierName : String
contactDetails : Integer
addProduct ()
removeProduct ()
Supplies
5 Passport Automation System:
Applicants
appId : String
name : String
DOB : Date
contact : Integer
applyPassport ()
checkStatus ()
Applies
Passport
passportId : String
expiryDate : Date
issueDate : Date
validatePassport ()
renewPassport ()
corresponds.
Application
AppId : String
appDate : Date
Status : String.
approveApplication ()
rejectApplication ()

STATE DIAGRAM

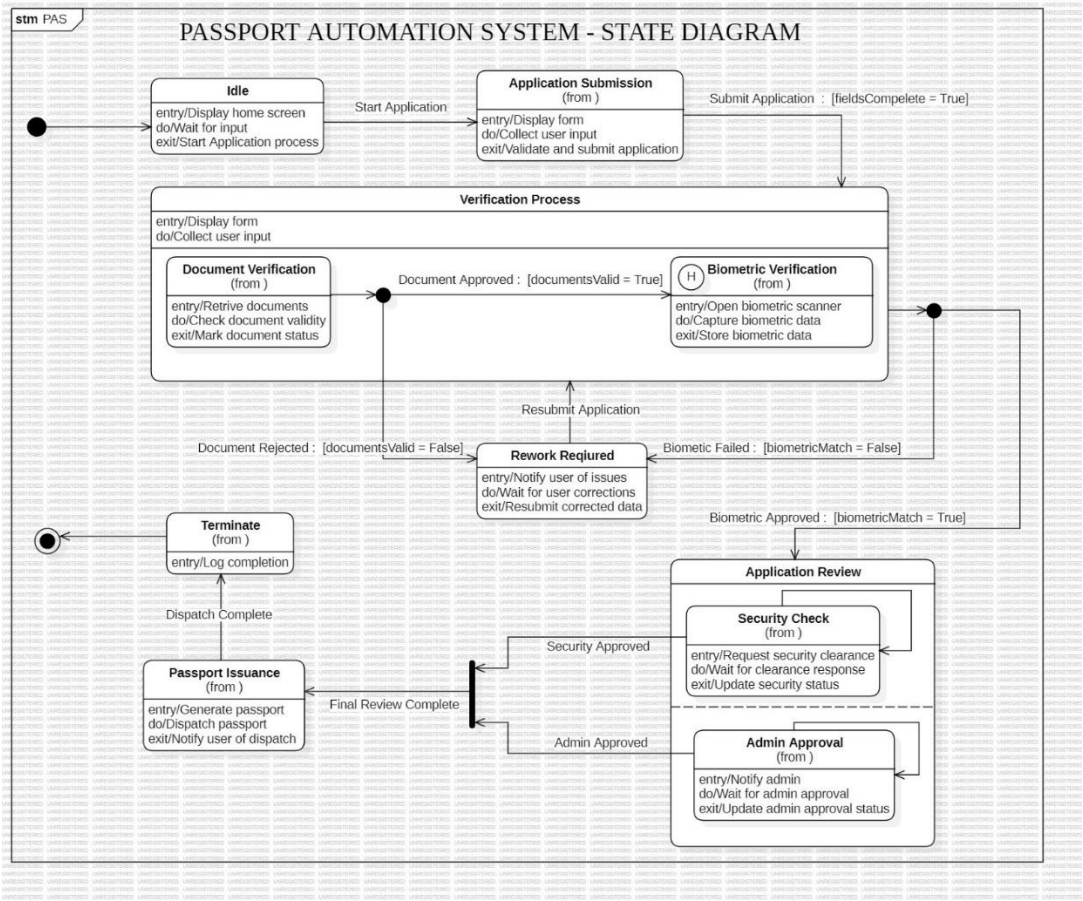


Figure 5.2: State Diagram

States:

- **Idle:** The initial state of the system, waiting for user interaction.
- **Application Submission:** The state where the user is submitting a passport application.
- **Verification Process:** The state where the submitted application is being verified.
- **Document Verification:** The state where submitted documents are being verified.
- **Biometric Verification:** The state where biometric data (fingerprints, facial recognition) is being captured and verified.
- **Rework Required:** The state where the applicant is required to resubmit documents or redo biometric verification.

- **Security Check:** The state where the application is undergoing a security clearance check.
- **Admin Approval:** The state where the application is waiting for administrative approval.
- **Final Review Complete:** The state where the application has passed all checks and is ready for passport issuance.
- **Passport Issuance:** The state where the passport is being generated and dispatched.
- **Dispatch Complete:** The final state indicating that the passport has been successfully dispatched to the applicant.

Transitions:

- **Start Application:** Triggered when a user initiates a passport application.
- **Submit Application:** Triggered when the user completes the application form and submits it.
- **Document Verification:** Triggered after the application is submitted, to begin document verification.
- **Document Approved:** Triggered if the submitted documents are valid.
- **Document Rejected:** Triggered if the submitted documents are invalid.
- **Biometric Verification:** Triggered if the documents are valid, to proceed with biometric verification.
- **Dispatch Complete:** Triggered when the passport is successfully dispatched to the applicant.

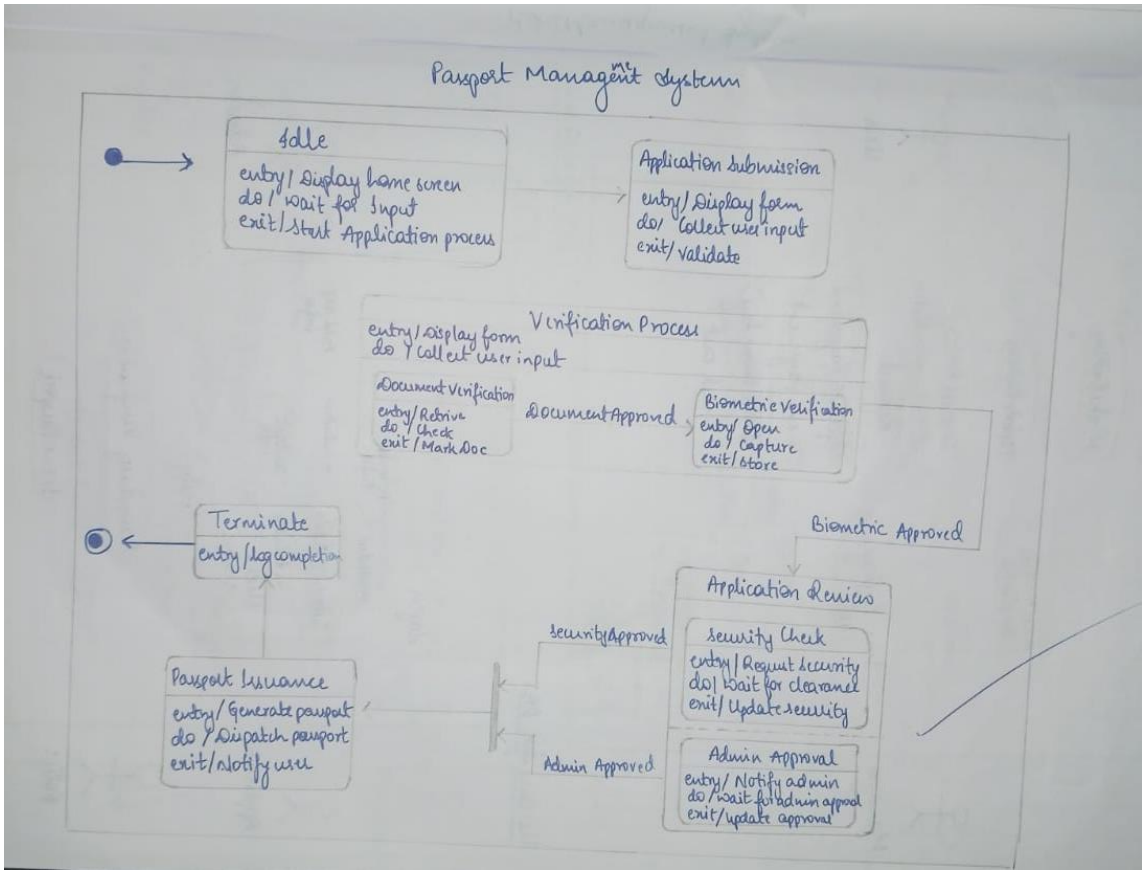
Activities:

- **Display home screen:** Activity performed in the Idle state.
- **Display form:** Activity performed in the Application Submission state.
- **Collect user input:** Activity performed in the Application Submission state.
- **Validate and submit application:** Activity performed in the Application Submission state.
- **Retrieve documents:** Activity performed in the Document Verification state.
- **Check document validity:** Activity performed in the Document Verification state.

- **Mark document status:** Activity performed in the Document Verification state.

Overall Description:

This state diagram provides a high-level overview of the passport application process, illustrating the various states and transitions that occur from the initial application submission to the final passport issuance. It includes activities and events that trigger transitions between states.



USE CASE DIAGRAM

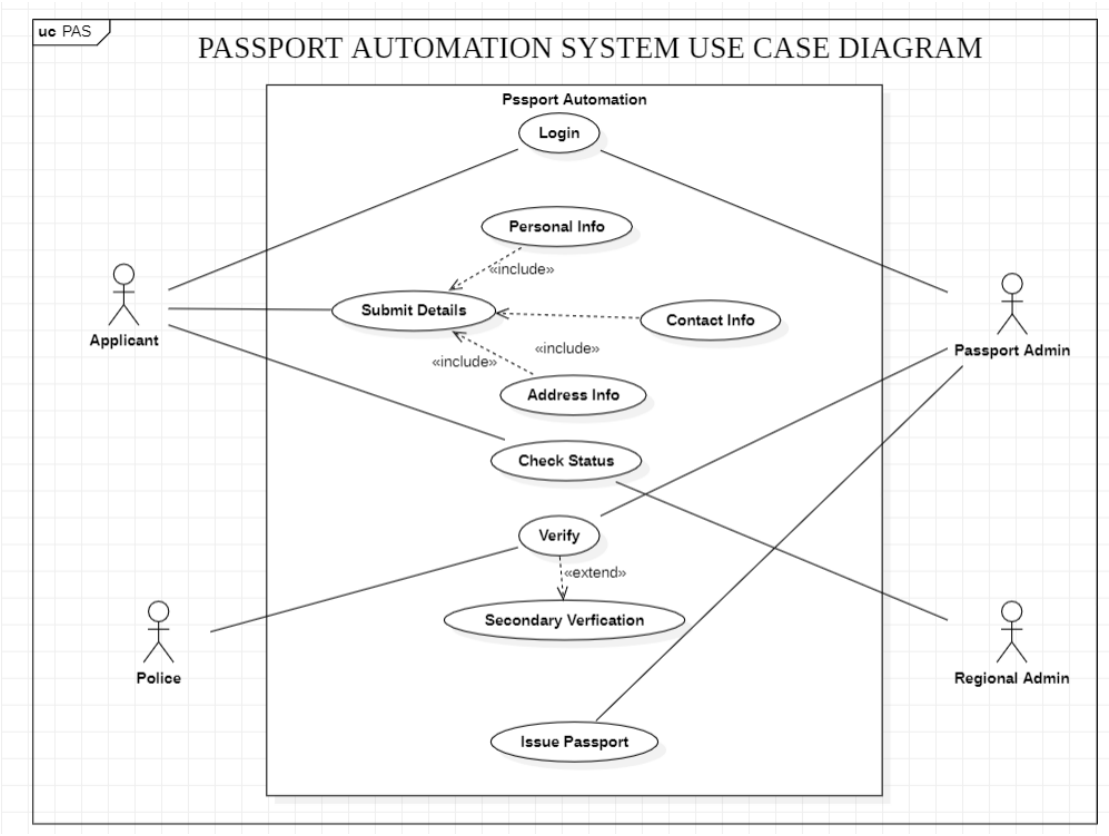


Figure 5.3: Use Case Diagram

Actors:

- **Applicant:** The person applying for a passport.
- **Passport Admin:** Responsible for managing the passport application process.
- **Admin:** The administrator of the system, with higher level access and control.
- **Regional Admin:** An administrator with regional level access and responsibilities.
- **Police:** Involved in verifying the applicant's background and address.

Use Cases:

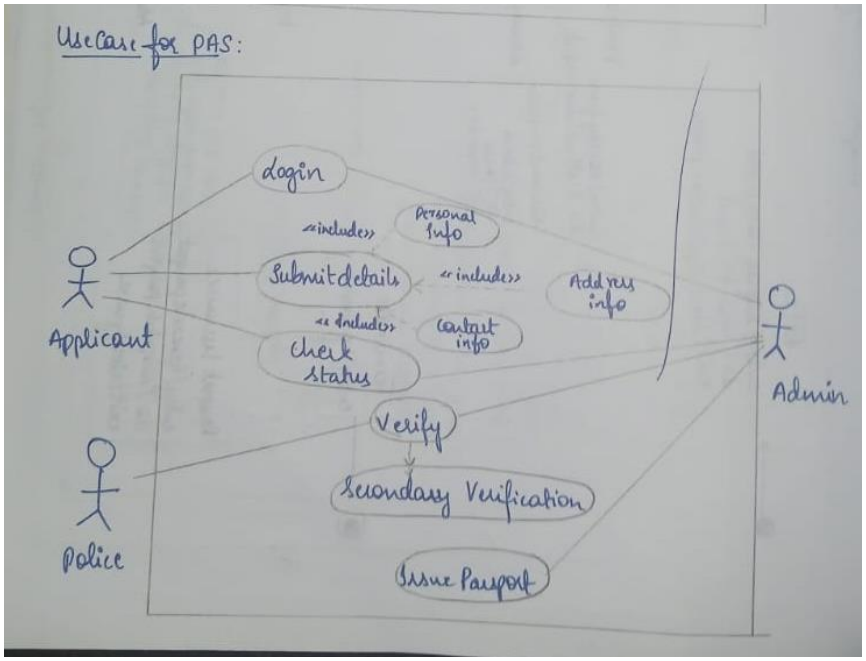
- **Passport Automation:** This is the main use case, encompassing all the functions related to passport application processing.
    - **Login:** The initial step for all users to access the system.
    - **Submit Details:**

        - **Includes:**
            - **Personal Info:** Entering personal information like name, date of birth, etc.
            - **Contact Info:** Entering contact information like address, phone number, etc.
            - **Address Info:** Entering address details.
    - **Check Status:** Checking the status of the passport application.
    - **Verify:**
        - **Extends:** This use case can be extended to include additional verification steps, such as background checks or document verification.
        - **Secondary Verification:** A specialized verification process, possibly involving additional agencies like the police.
    - **Issue Passport:** The final step where the passport is issued to the applicant.

Relationships:

- **Includes:** Represents a dependency between use cases, where one use case includes the functionality of another. For example, "Submit Details" includes "Personal Info," "Contact Info," and "Address Info."
- **Extends:** Represents an optional extension to a use case. For example, "Verify" can be extended to include "Secondary Verification."

Overall, this use case diagram provides a high-level overview of the passport application process. It helps to visualize the roles of different actors, the flow of information between them, and the various steps involved in obtaining a passport.


Usecase for PAS:
Login
<<includes>>
Personal Info
Submit details
<<includes>>
Address info
<<includes>>
Contact info
Check Status
Applicant
Admin
Verify
Secondary Verification
Police
Issue Passport

SEQUENCE DIAGRAM

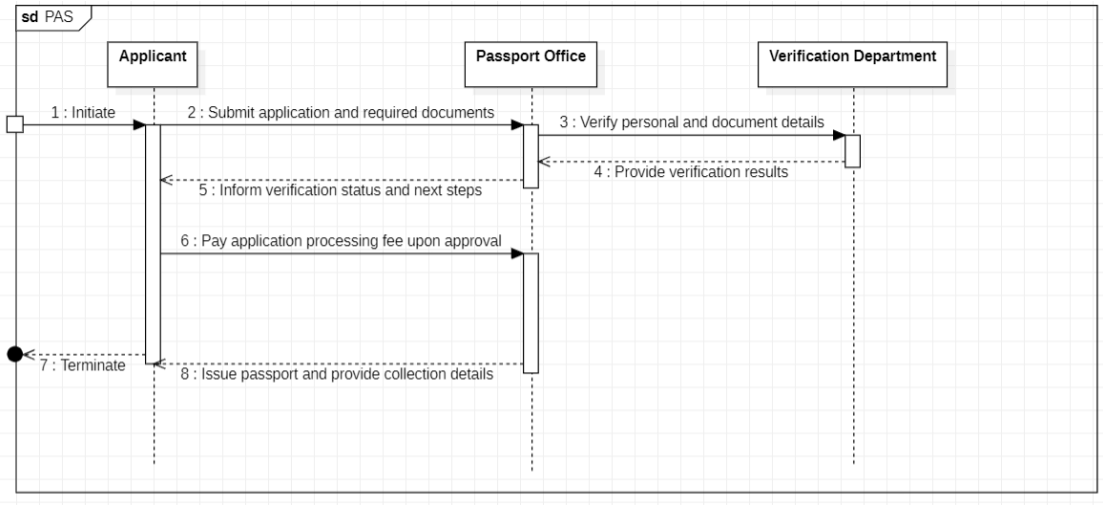


Figure 5.4: Sequence Diagram

Objects:

- **Applicant:** The person applying for the passport.
- **Passport Office:** The entity responsible for processing passport applications.
- **Verification Department:** The department responsible for verifying applicant details and documents.

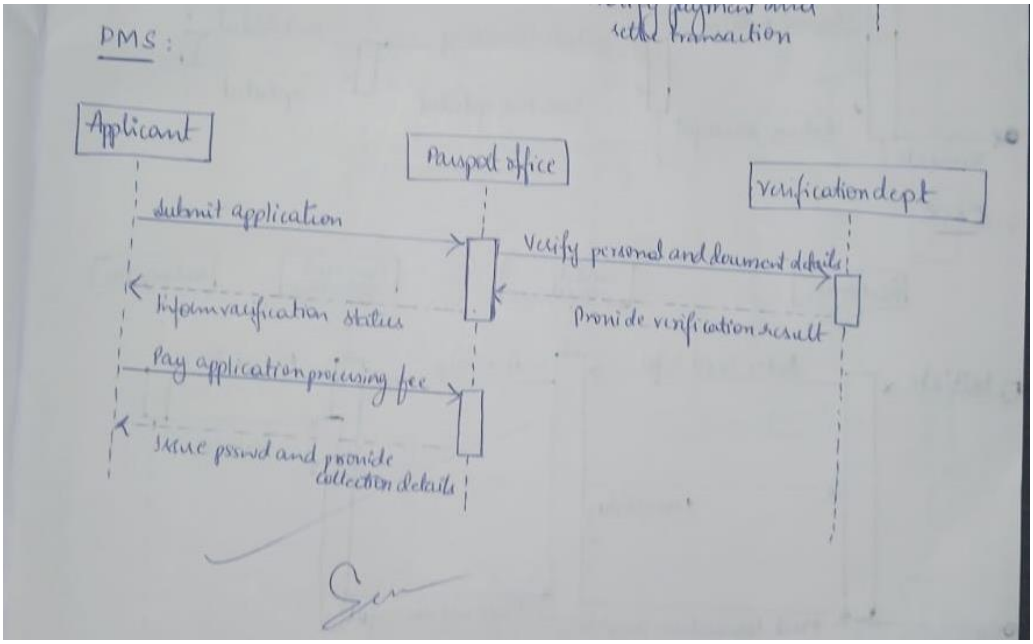
Interactions:

1. **Initiate:** The Applicant initiates the passport application process.
2. **Submit application and required documents:** The Applicant submits the application form and required documents to the Passport Office.
3. **Verify personal and document details:** The Passport Office sends the application and documents to the Verification Department for verification.
4. **Provide verification results:** The Verification Department verifies the information and sends the results back to the Passport Office.
5. **Inform verification status and next steps:** The Passport Office informs the Applicant about the verification status and the next steps in the process (e.g., payment, appointment scheduling).
6. **Pay application processing fee upon approval:** If the application is approved, the Applicant pays the processing fee to the Passport Office.

7. **Issue passport and provide collection details:** After payment and any necessary background checks, the Passport Office issues the passport and provides the Applicant with details on how to collect it.
8. **Terminate:** The passport application process is completed.

Overall:

This Sequence Diagram illustrates the flow of interactions between the Applicant, Passport Office, and Verification Department during the passport application process. It shows the sequence of messages exchanged and the different steps involved in obtaining a passport.



ACTIVITY DIAGRAM

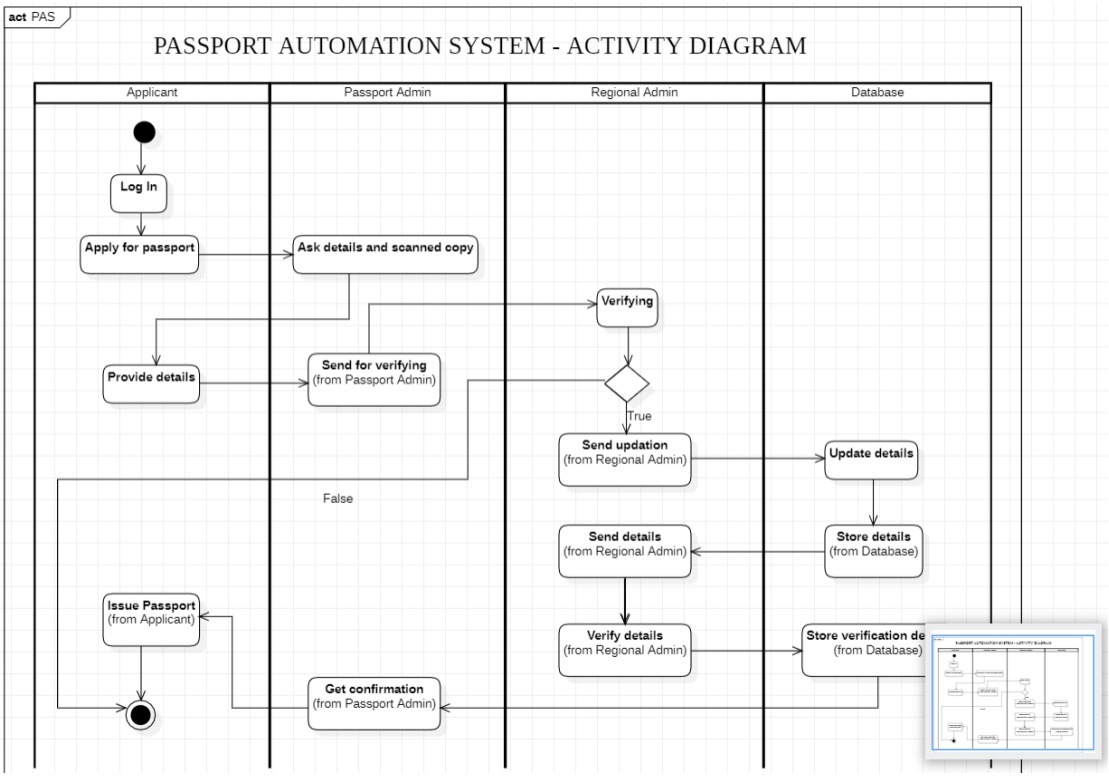


Figure 5.5: Activity Diagram

Swimlanes:

1. **Applicant:** This swimlane represents the actions performed by the applicant who is applying for a passport. These actions include:
    - **Log In:** The applicant logs into the system to initiate the application process.
    - **Apply for passport:** The applicant submits a request for a passport.
    - **Give details and send scanned copy:** The applicant provides personal information and uploads scanned copies of required documents.
    - **Issue Passport:** The applicant receives the issued passport.
2. **Passport Admin:** This swimlane represents the actions performed by the passport administrator. These actions include:
    - **Ask details and scanned copy:** The passport admin requests the necessary details and scanned documents from the applicant.
    - **Send for verifying:** The passport admin forwards the application and documents to the regional admin for verification.

- **Get confirmation:** The passport admin receives confirmation from the regional admin about the verification status.

3. **Regional Admin:** This swimlane represents the actions performed by the regional administrator. These actions include:
    - **Verifying:** The regional admin verifies the applicant's information and documents.
    - **Send updation:** The regional admin sends an update on the verification status to the passport admin.
    - **Send details:** The regional admin sends the verified details to the database.
    - **Verify details:** The regional admin verifies the details stored in the database.
4. **Database:** This swimlane represents the actions performed by the database system. These actions include:
    - **Store details:** The database stores the applicant's information and verification details.
    - **Store verification details:** The database stores the verification results provided by the regional admin.