

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



Lab Record

Object-Oriented Modeling

Submitted in partial fulfillment for the 5th Semester Laboratory

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

C Mohan Babu
1BM23CS402

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

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND

ENGINEERING



CERTIFICATE

This is to certify that the Object-Oriented Modeling (22CS5PCOOM) laboratory has been carried out by C Mohan Babu (1BM23CS402) during the 5th Semester Oct 24-Jan 2025.

Signature of the Faculty Incharge:

Prof. Anusha S

Department of Computer Science and Engineering
B.M.S. College of Engineering, Bangalore

Table of Contents

1. Hotel Management System
2. Credit Card Processing
3. Library Management System
4. Stock Maintenance System
5. Passport Automation System

1. Hotel Management System

Problem Statement: Hotels often face challenges in managing their daily operations efficiently, such as handling reservations, managing room availability, billing, staff management, and customer service. Traditional systems or manual methods are time-consuming, error-prone, and do not scale well with growing customer demands.

1.1 SRS-Software Requirements Specification

METRO	
Page:	1 / 1
→ Hotel Management System :-	
Problem Statement :- They require efficient management to handle daily operations, such as which involve managing reservations, customer service availability of room, arranging the staff, scheduling the work, over room booking, delayed checkout, proper management of billings, proper management in parking.	
Goal is to develop some required operations like :-	
→ Customer Management	
→ Billing and Payment	
→ Room bookings	
→ Highly automated parking management	
→ Food management	
→ Scope :-	
→ Reservation management:- Handle booking, reservation, cancellation & modifications.	
→ Staff management:- Proper Training to the staff & handle & maintain them.	
→ Room management:- Managing rooms availability & maintenance & pricing.	
→ Billing & Payment processing:- Handle invoice payments & manage transactions.	
Functional Requirements :-	
→ User Management:- Create, delete & update accounts	
→ Role based access for different type of user (Manager)	
→ Reservation Management:- Create, delete & modify reservations	
→ Send mails to the booking guests.	

→ Billing & Payment: Generate Invoice of guest,
manage payment

→ Check in / check out on time management.

Non-Functional Requirements

→ Performance: The system should handle multiple users at the same time without any degradation in its efficiency.

→ Availability: System should available to all customers even if crowded at the same time.

→ Security: It should validate customers data & keep secure from any threats & scams & outgoing the user information.

→ Backup & Recovery: The system should have backup & recovery of user data when the system should crash.

Domain Requirements

Booking Policies: Management should implement the policies for the user bookings, check-in / check-out & for other services that include booking.

1.2 Class Diagram

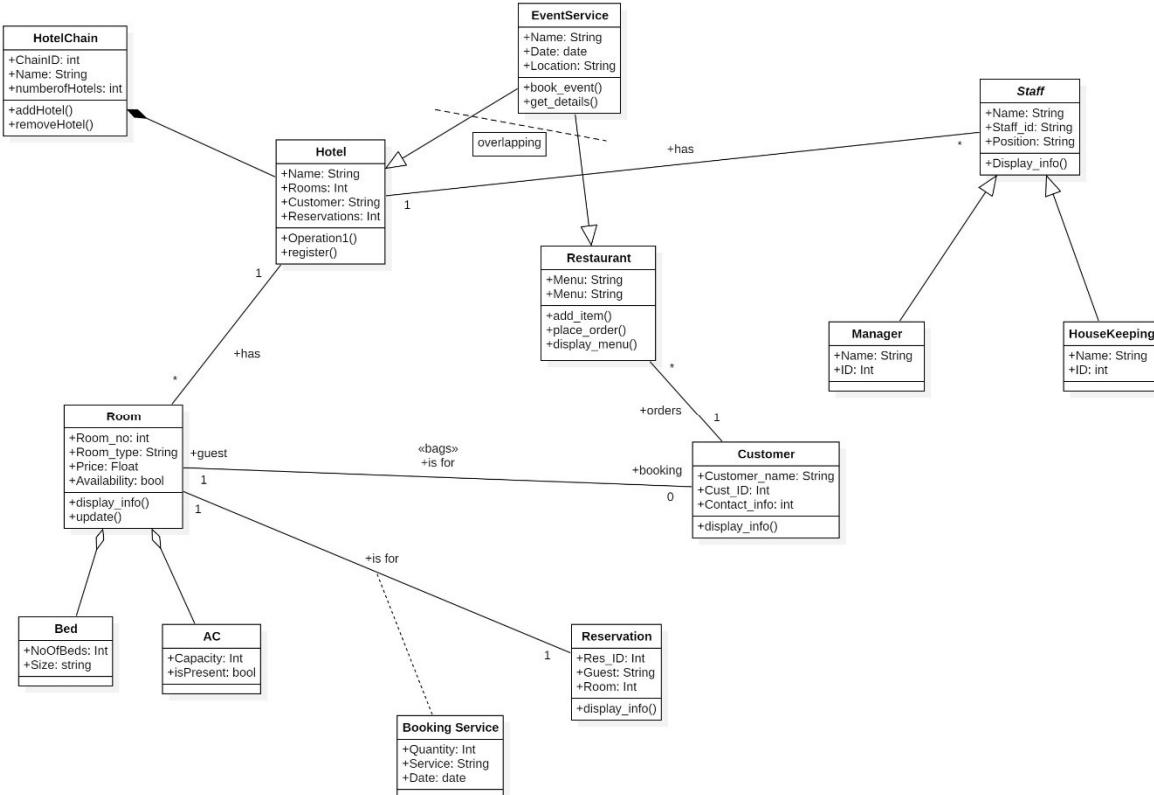


Fig 1.2.1

The class diagram represents a **Hotel Management System**, outlining key entities and their interactions. Below are the major components:

Core Classes:

- **HotelManagement**: Manages high-level operations with attributes like HotelName, NoOfEmployees, and Location. The Open() method initializes hotel operations.
- **Booking**: Handles reservations with attributes like BookingId, NoOfRooms, CheckIn, and CheckOut, and methods book() and cancel(). Linked to **Customer** and associated with **Rooms**.
- **Rooms**: Represents hotel rooms with properties such as RoomId, Type, RoomNo, and Cost. Aggregates **Beds** to define room structure.
- **Payment**: Tracks payment details using attributes like PaymentId and Status (enumeration: Confirmed, NotConfirmed, Processing) and methods confirmPayment() and cancelPayment().

- **Invoice:** Stores billing information (InvoiceId, Bill) and generates an invoice view using the display() method.

Specialized Entities:

- **Person:** Base class for **Employee** and **Customer**.
 - **Employee:** Adds EmployeeId and Salary.
 - **Customer:** Includes CustomerId and is linked to **Booking**.
- **HotelChain:** Represents a chain of hotels with attributes HotelName, ChainID, and NoOfHotels. Methods AddHotel() and RemoveHotel() manage the chain.

Utility Classes and Enumeration:

- **Beds, AC, TV:** Support room features, e.g., **Beds** defines NoOfBeds and BedSize.
- **Status:** Enumerates payment states (e.g., Confirmed, NotConfirmed).

1.3 State Diagram

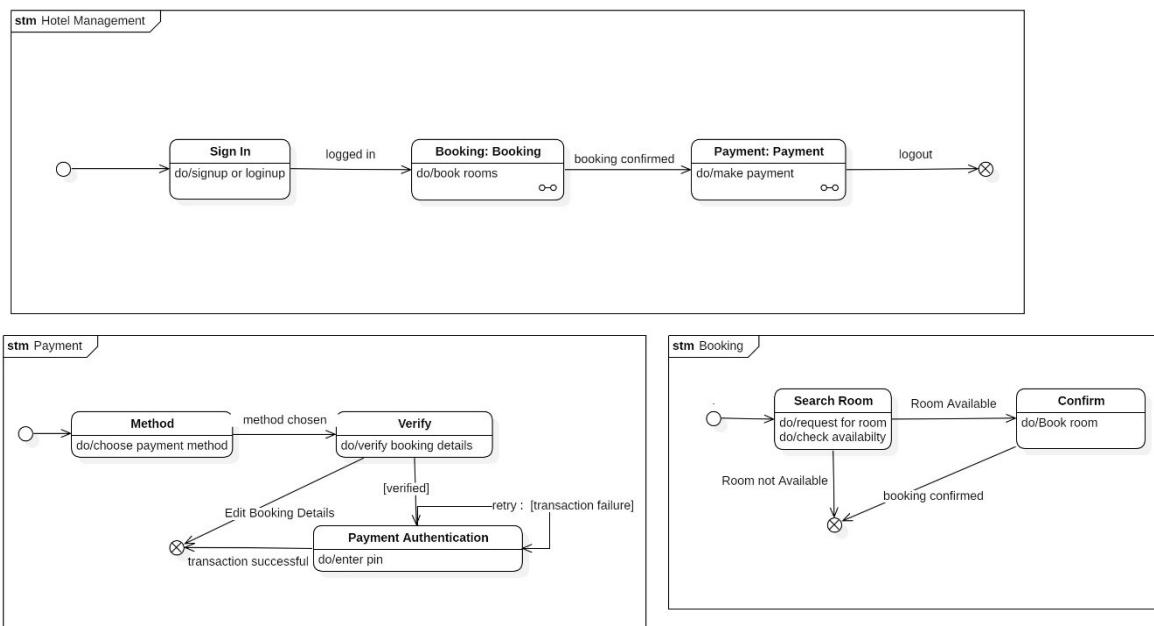


Fig 1.3.1 State Diagram

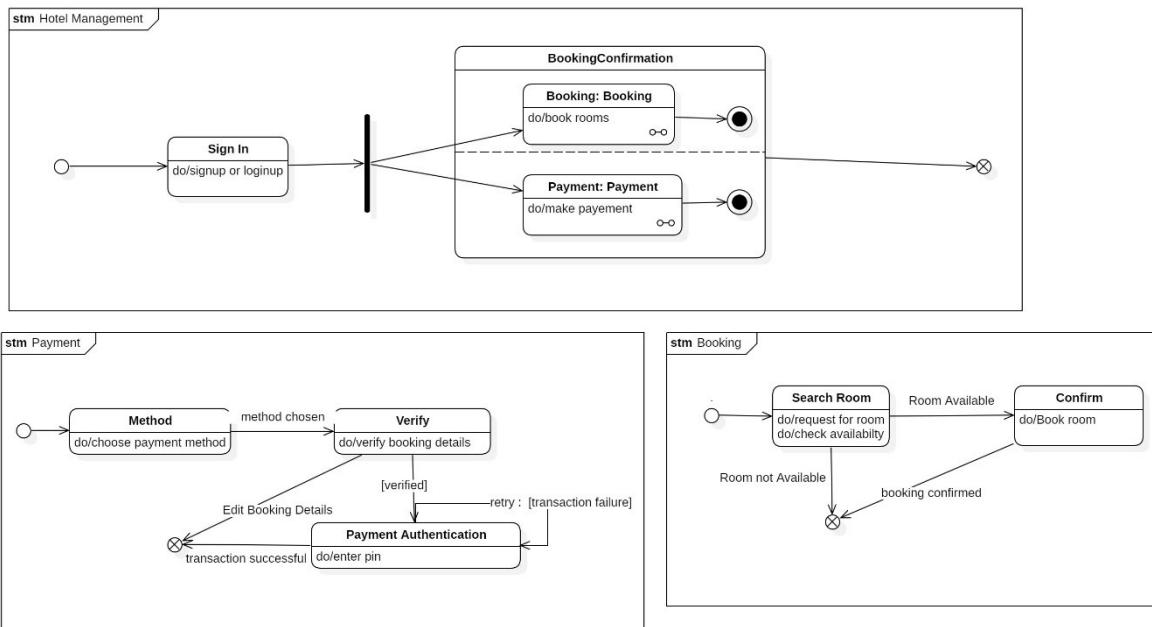


Fig 1.3.2 Advanced State Diagram

The state machine diagrams illustrate the dynamic behavior and transitions between states in the **Hotel Management System**. The key models include the **Hotel Management System**, **Payment Process**, and **Booking Process**.

State Machine: Hotel Management

- **Initial State:** The process begins at the **Sign In** state.
- **Sign In:** Handles user authentication, allowing guests or employees to log in or sign up.
- **BookingConfirmation:** A composite state with two parallel operations:
 - **Booking:** Performs actions to doBook rooms for confirmed reservations.
 - **Payment:** Executes doMake payment to complete the booking process.
- **Completion:** The final state is reached after successful booking and payment.

State Machine: Payment Process

- **Initial State:** Begins with selecting a **Payment Method** (e.g., card, cash, digital wallet).
- **Verify:** Confirms booking details for accuracy.
 - **If verification fails:** Users can edit booking details and retry.

- **Payment Authentication:** Requires entering a PIN or equivalent credentials for payment authorization.
 - **Transaction Success:** Proceeds to the final state, marking payment completion.
 - **Transaction Failure:** Redirects users to retry or cancel the payment.

State Machine: Booking Process

- **Initial State:** Begins with the **Search Room** action.
 - **Search Room:** Processes the user's room availability request.
 - **If Room Not Available:** The search ends, or alternatives are suggested.
- **Room Available:** Transitions to the **Confirm** state, where booking details are finalized.
 - **Confirm:** Executes the doBook room action, completing the reservation.
- **Final State:** The process concludes when the booking is confirmed.

1.4 Use Case Diagram

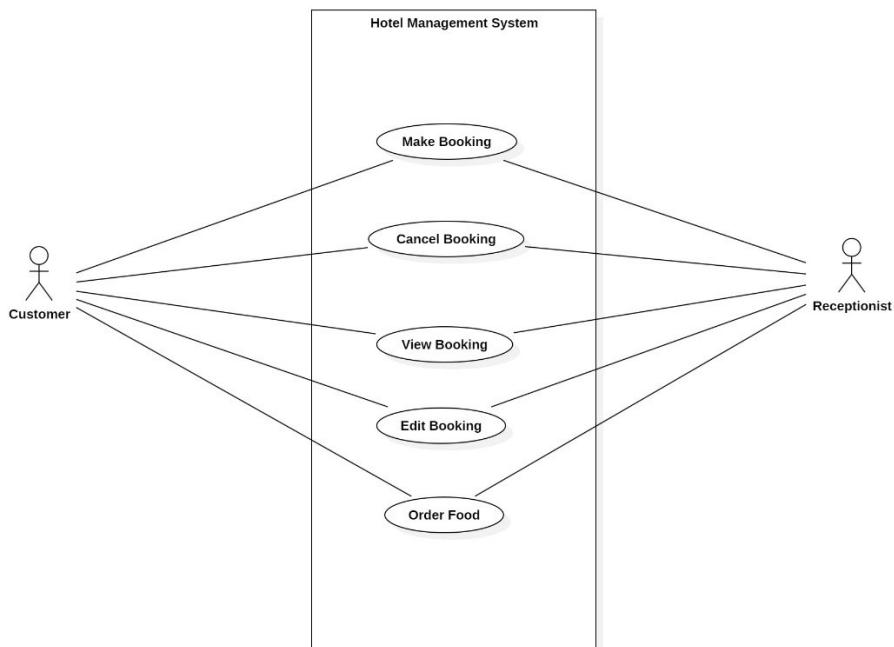


Fig 1.4.1 Use Case Diagram

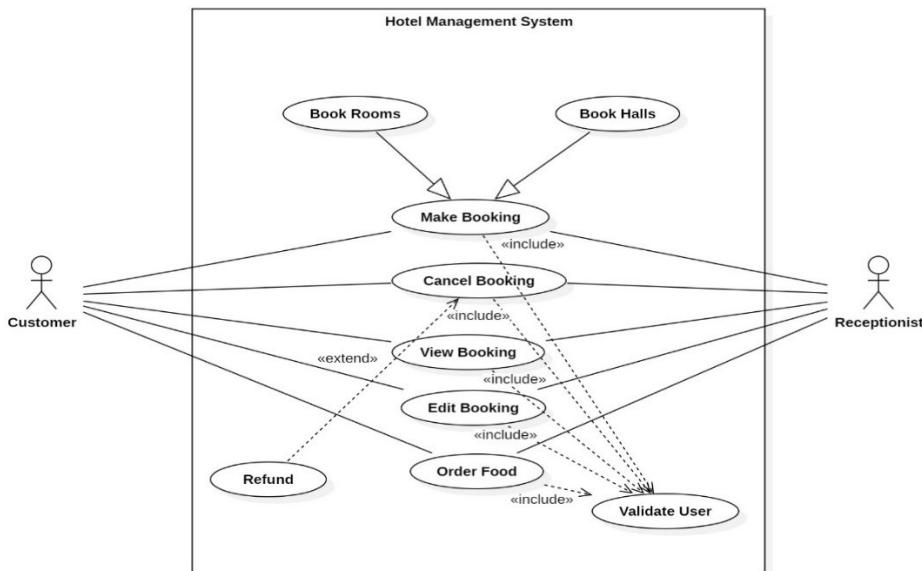


Fig 1.4.2 Advanced Use Case Diagram

This use case diagram illustrates the interactions between actors (Customer and Receptionist) and the hotel management system. Key use cases and their relationships are outlined below:

Actors:

1. **Customer:** Interacts with the system to manage bookings and related activities.
2. **Receptionist:** Handles administrative tasks and validates user actions.

Use Cases:

1. **Book Rooms:** Customers can book rooms for their stay.
2. **Book Halls:** Customers can reserve halls for events or meetings.
3. **Make Booking:** A general use case encompassing room or hall bookings.
4. **Cancel Booking:** Allows the cancellation of bookings by the customer or receptionist.
 - o **Includes:** Validate user to ensure authorization for cancellation.
5. **View Booking:** Customers can view their current bookings.
6. **Edit Booking:** Enables customers to modify booking details.
 - o **Includes:** User validation to confirm authorization.
7. **Order Food:** Lets customers order food during their stay.
 - o **Extends:** Edit Booking for adding meal preferences to existing reservations.

8. **Extend Stay:** Allows customers to prolong their stay at the hotel.
 - **Extends:** Edit Booking for extending reservation duration.

9. **Refund:** Processes refunds for canceled bookings.
 - **Extends:** Cancel Booking when applicable.

1.5 Sequence Diagram

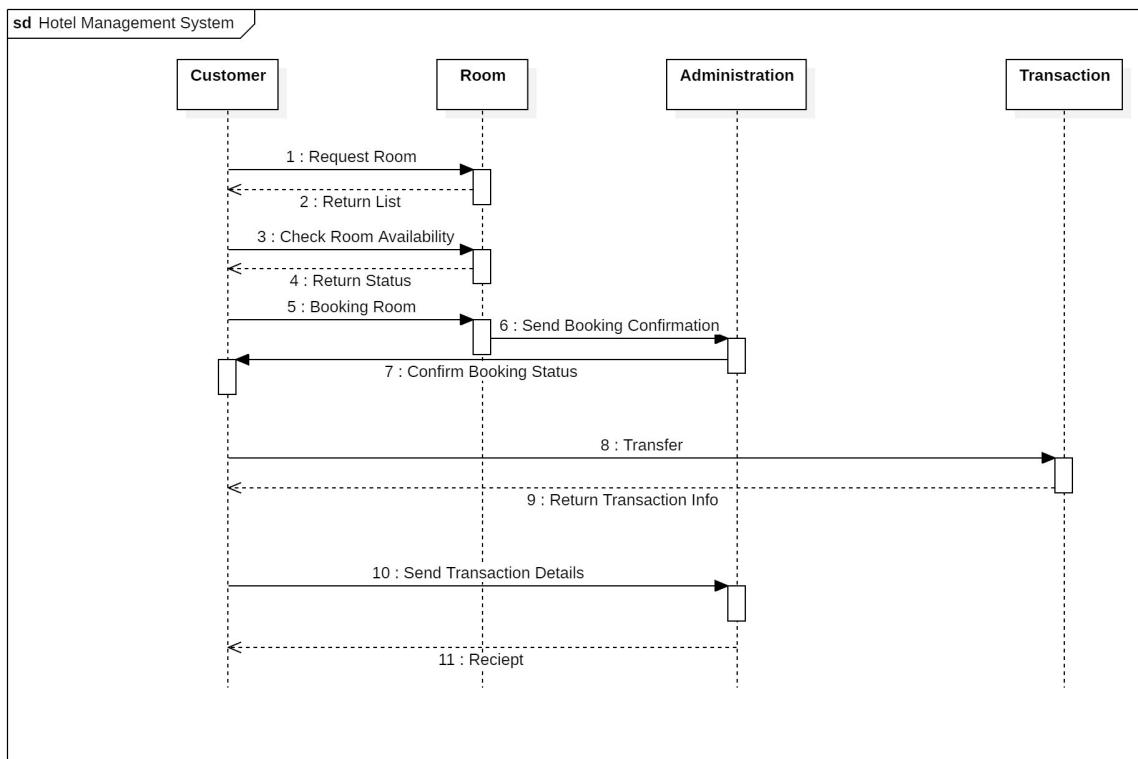


Fig 1.5.1 Sequence Diagram

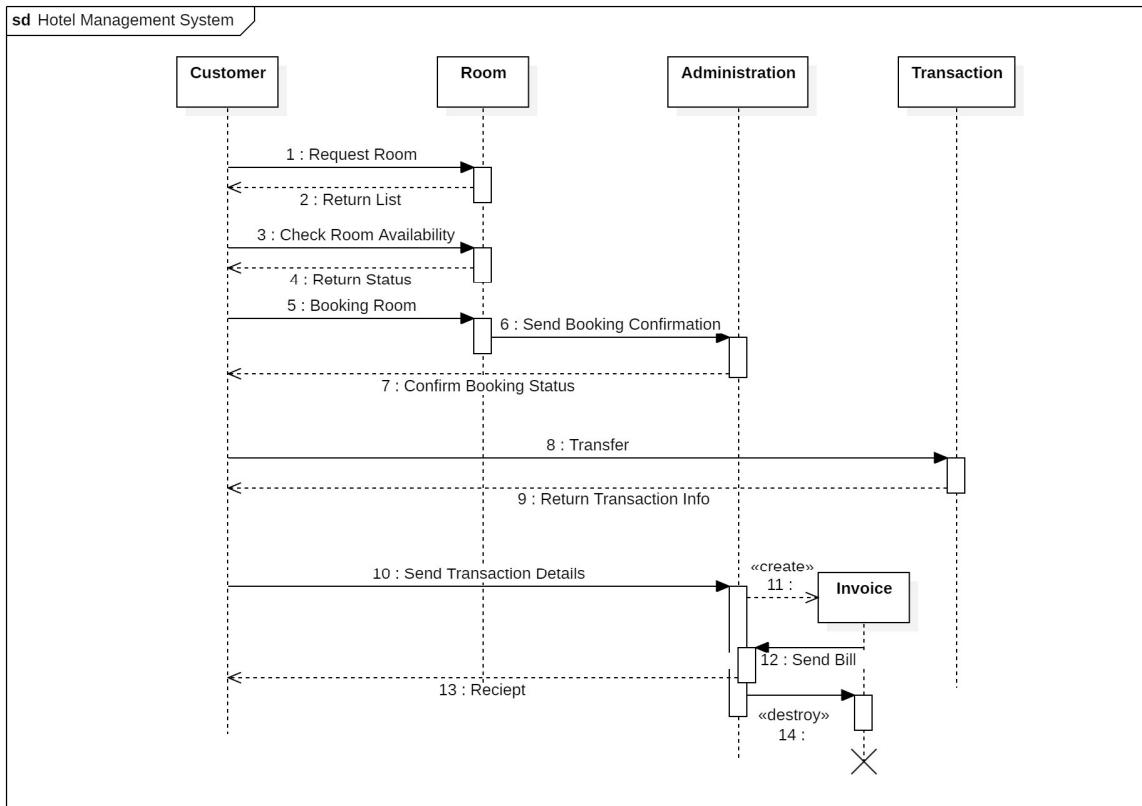


Fig 1.5.2 Advanced Sequence Diagram

- **Request Room (Customer → Room):** The customer initiates the process by requesting room details.
- **Return List (Room → Customer):** The system provides the customer with a list of available rooms.
- **Check Room Availability (Customer → Room):** The customer selects a room, and the system verifies its availability.
- **Return Status (Room → Customer):** The system informs the customer of the room's availability.
- **Booking Room (Customer → Room):** If available, the customer books the room.
- **Send Booking Confirmation (Room → Administration):** The room system communicates the booking to the administration for approval.
- **Confirm Booking Status (Administration → Customer):** The administration notifies the customer of the confirmed booking.
- **Transfer Payment (Customer → Transaction):** The customer proceeds with payment via the transaction system.

- **Return Transaction Info (Transaction → Administration):** The transaction system sends payment details to the administration.
- **Send Transaction Details (Administration → Customer):** The administration updates the customer with transaction details.
- **Create Invoice (Administration → Invoice):** The invoice system generates a billing invoice for the transaction.
- **Send Bill (Invoice → Customer):** The generated bill is sent to the customer.
- **Receipt (Customer → Invoice):** The customer acknowledges receipt of the bill.
- **Destroy Invoice:** The invoice is destroyed after the process is successfully completed.

1.6 Activity Diagram

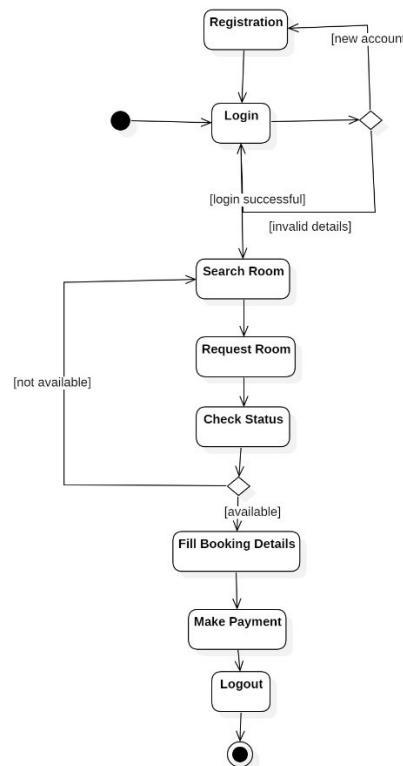


Fig 1.6.1 Activity Diagram

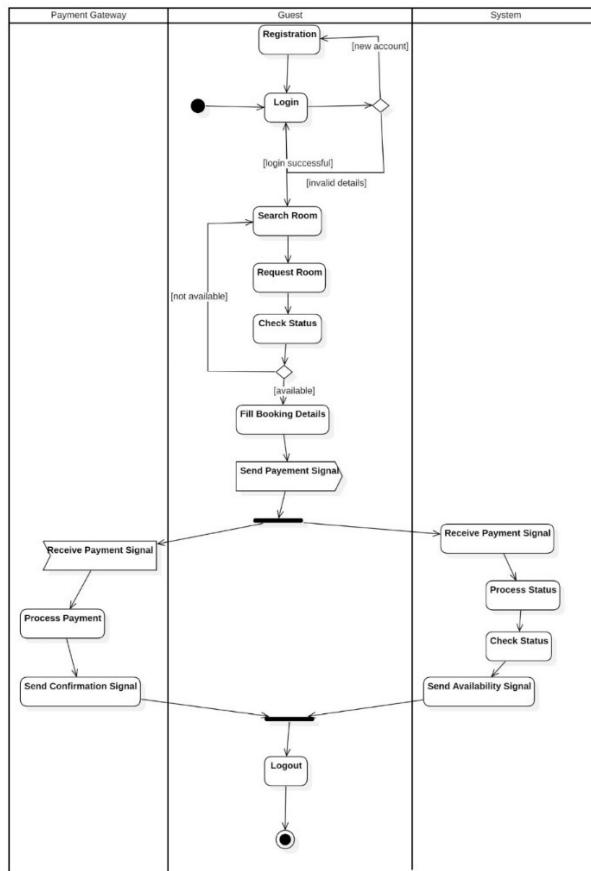


Fig 1.6.2 Advanced Activity Diagram

An activity diagram typically represents the workflows within a system, focusing on the sequence of activities and decision points. Here's a breakdown of the activity diagram for the Hotel Management System:

Start Point:

- The process begins when a customer initiates an action, such as searching for a room or performing other tasks.

Key Activities:

1. Search for Rooms or Halls:

- The customer searches for available options in the system, prompting it to display a list of choices.

2. Select Room or Hall:

- Based on preferences, the customer selects a specific room or hall.

3. Check Availability:

- The system verifies the availability of the selected option for the specified date and time to allow the process to proceed.

4. Make a Booking:

- The customer enters personal and booking details, such as check-in and check-out dates, and confirms the booking request.

5. Process Payment:

- The system transitions to payment processing, where the customer completes the transaction using credit cards, debit cards, or other payment gateways.

6. Confirm Booking:

- Upon successful payment, the booking is confirmed, a notification is sent to the customer, and system records are updated.

Decision Points:

- Is the Room or Hall Available?**

- *Yes*: The process proceeds to booking.
- *No*: The customer is prompted to select a different option or modify criteria.

- Is Payment Successful?**

- *Yes*: The booking is confirmed.
- *No*: The customer may retry payment or cancel the process.

Additional Features:

- Cancel Booking:**

- Customers can cancel bookings, triggering the system to process refunds (if applicable).

- Order Food or Extend Stay:**

- Customers may perform additional activities like ordering food or extending their stay, which can be integrated into the workflow.

End Point:

- The process concludes when the booking is confirmed, canceled, or a related action (like retrying payment) is completed.

2. Credit Card Processing System

Problem Statement: Financial institutions and merchants face challenges in securely and efficiently managing credit card transactions. The process involves verifying customer details, processing payments, fraud detection, and ensuring regulatory compliance. An unreliable or inefficient system can lead to transaction delays, increased risks of fraud, and poor customer experience.

2.1 SRS-Software Requirements Specification

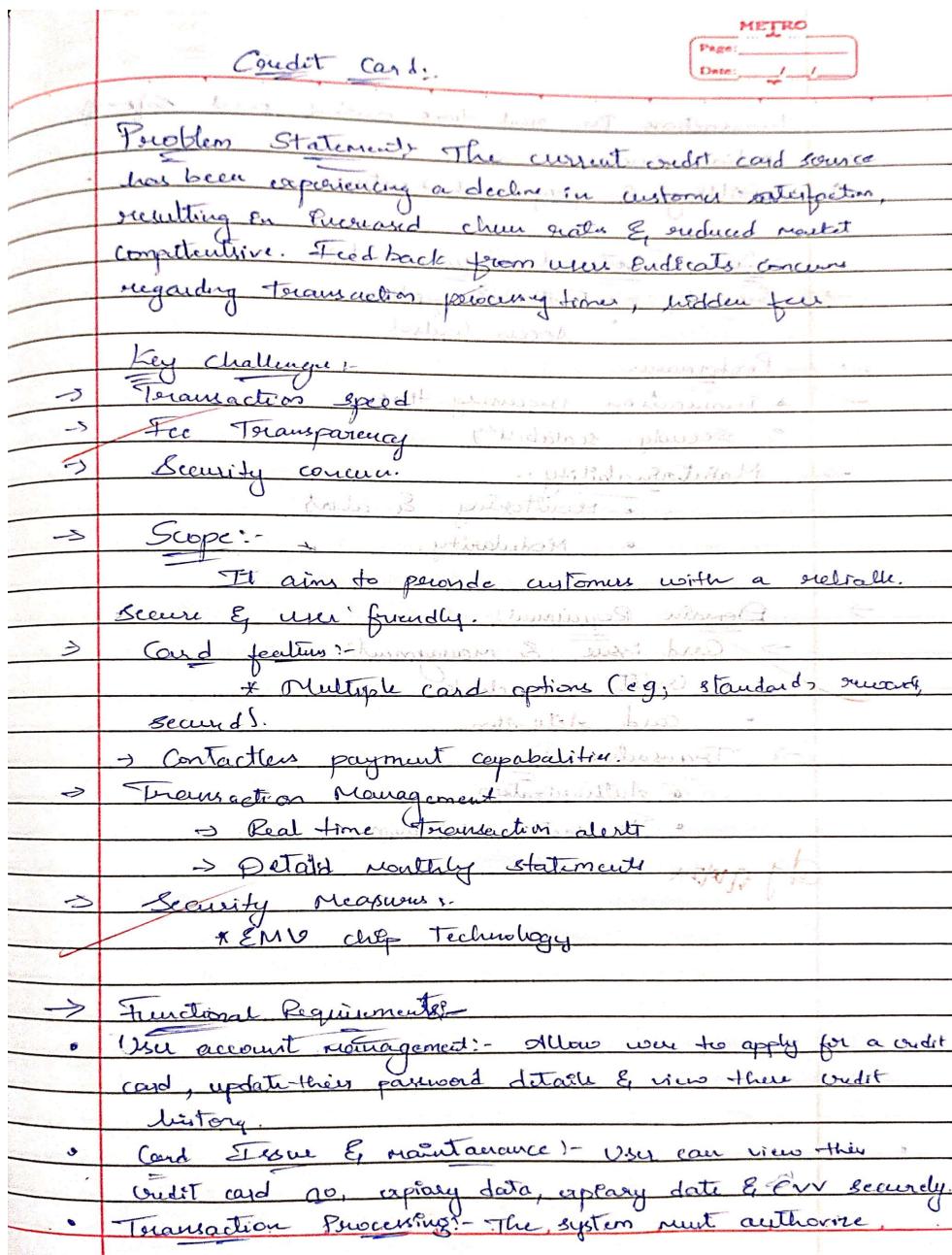


Fig 2.1.1

- ~~• Business Requirements~~
 - ~~• Functional Requirements~~
 - ~~• Non-Functional Requirements~~
 - ~~→ Security :-~~
 - Data Encryption
 - Access Control
 - ~~→ Performance:-~~
 - Transaction Processing time
 - Security scalability
 - ~~→ Maintainability:-~~
 - Monitoring & alert
 - Modularity.
 - ~~→ Domain Requirements~~
 - ~~→ Card issue & management:-~~
 - Credit card check
 - card Activation.
 - ~~→ Transaction Process~~
 - Authorization
 - Transaction validation
- ~~dry notes~~

Fig 2.1.2

2.2 Class Diagram

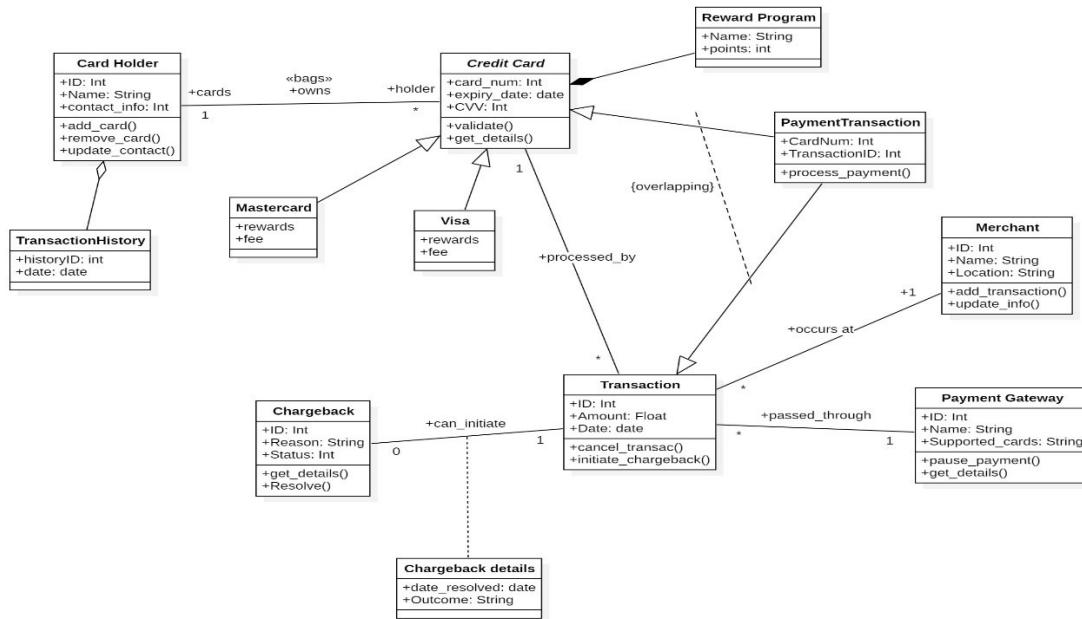


Fig 2.1.1

The class diagram represents a **Credit Card Processing System** that encompasses key entities and their interactions. Below are the major elements:

Core Classes:

- **Customer**: Represents the cardholder with attributes such as cust_id (Customer ID), card_no (Card Number), and methods like:
 - makePayment(): Initiates the payment process.
 - check_card(): Validates the associated card.
- **Card**: Represents the customer's credit card with attributes such as card_no (Card Number), expiry (Expiry Date), and balance. It includes methods like:
 - checkBalance(): Retrieves the card's balance.
 - payment(): Processes a payment.
- **Account**: Manages banking details associated with the customer and card. Attributes include accountNumber, accountType, and balance. Methods:
 - deposit(): Adds funds to the account.
 - withdraw(): Deducts funds.
- **Employee**: Represents staff members involved in the credit card process, with attributes like emp_id (Employee ID) and emp_type (Role). It includes methods such as:
 - check_customers(): Verifies customer details.

- issueCard(): Issues a new card to the customer.
- **Transaction_log:** Tracks all payment activities with attributes like log_id, month, and pay_status (Successful, NotSuccessful). Methods include:
 - generate_report(): Creates a detailed report of transactions.
- **Bill:** Represents invoices or payment dues with attributes such as bill_id, amount, and invoice_no. It includes the payment() method to settle bills.

Specialized Entities:

- **ValidPayment:** Validates payments with attributes like valid (Boolean) and a check() method to confirm payment status.
- **RewardPrograms:** Tracks rewards linked to the customer's card with attributes such as name and points.

Utility Classes:

- **Credit_balance:** Checks the current balance against bills, with methods like:
 - check(card_no, bill_amount): Validates whether sufficient credit is available.
- **Check_card:** Manages card validity with attributes like card_no and cust_id. The check_held_cards() method validates all cards under a customer.

Relationships:

- **Inheritance:**
 - Customer and Employee inherit from Person, which serves as a base class with attributes like p_name (Person Name), p_gender, and p_address.
- **Aggregation:**
 - Customer aggregates Card, indicating that a customer may have multiple cards.
 - Card aggregates RewardPrograms, representing card-related rewards.
- **Dependency:**
 - Transaction_log depends on ValidPayment to determine transaction outcomes.
 - Bill depends on Transaction_log for maintaining payment records.
- **Associations:**
 - Customer is associated with Card through a one-to-many relationship.
 - Bill interacts with Card and Credit_balance for payment processing.

Enumeration:

- **Payment Status:** Includes statuses like Successful and NotSuccessful to manage and track transaction outcomes effectively.

2.3 State Diagram

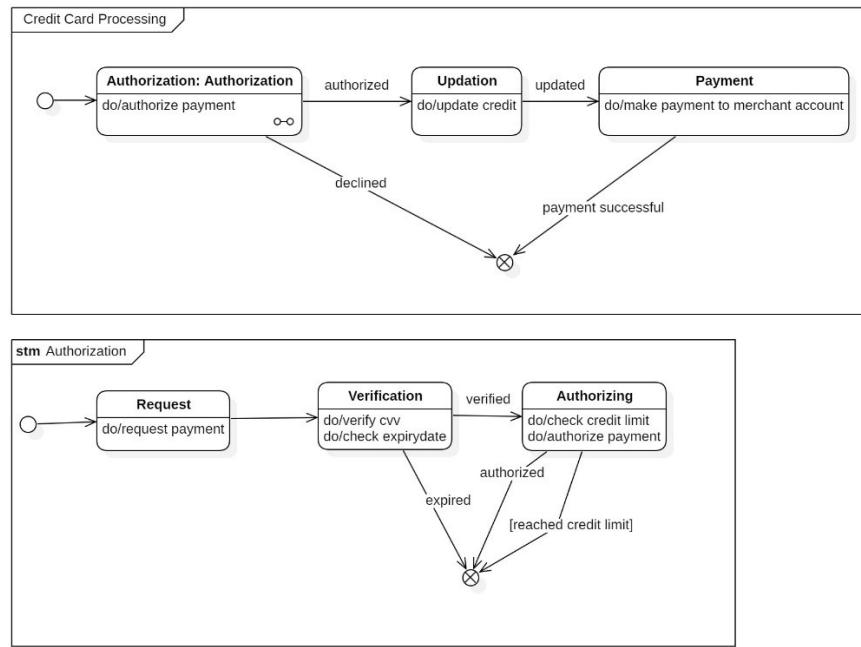


Fig 2.3.1 State Diagram

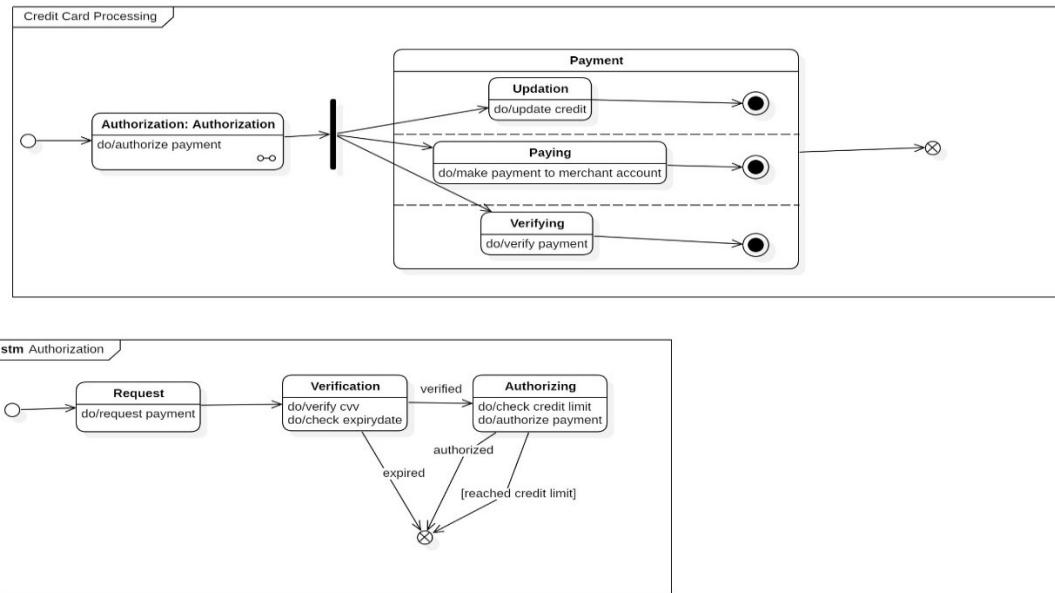


Fig 2.3.2 Advanced State Diagram

The state machine diagrams illustrate the dynamic behaviour and transitions between states in the **Credit Card Processing System**. The key models include **Credit Card Processing**, **Payment Authorization**, and **Payment Process**.

State Machine: Credit Card Processing

- **Initial State:** The process begins at the Authorization state.

- **Authorization:** Represents the initial step of validating the payment request.
 - Actions:
 - do/authorize payment: Initiates the payment authorization process.
- **Payment:** A composite state that integrates three parallel operations:
 - **Updation:** Updates credit information.
 - Action: do/update credit.
 - **Paying:** Processes payment transfer to the merchant account.
 - Action: do/make payment to merchant account.
 - **Verifying:** Confirms payment details.
 - Action: do/verify payment.
- **Completion:** The final state is reached after successful payment processing.

State Machine: Payment Authorization

- **Initial State:** Starts with the Request state.
 - Action: do/request payment.
- **Verification:** Ensures the payment credentials (e.g., CVV, expiry date) are correct.
 - Actions:
 - do/verify CVV.
 - do/check expiry date.
 - Transitions:
 - If verified, transitions to the Authorizing state.
 - If expired, moves to a termination state.
- **Authorizing:**
 - Checks credit limit and processes payment authorization.
 - Actions:
 - do/check credit limit.
 - do/authorize payment.
 - Transitions:
 - If authorized, moves to the final state.
 - If credit limit reached, transitions to termination or retry states.
- **Final State:** Reached upon successful payment authorization.

State Machine: Payment Process

- **Initial State:** Begins with choosing the Payment Method.
 - Action: do/select payment method (e.g., card, digital wallet).

- **Verify:** Validates the booking or transaction details for accuracy.
 - Transitions:
 - If valid, proceeds to Payment Authentication.
 - If invalid, allows the user to edit details and retry.
- **Payment Authentication:**
 - Requires entering a PIN or equivalent credentials to authorize payment.
 - Transitions:
 - **Transaction Success:** Leads to the final state, marking payment completion.
 - **Transaction Failure:** Redirects users to retry or cancel the payment.
- **Final State:** Reached upon successful completion of the payment

2.4 Use Case Diagram

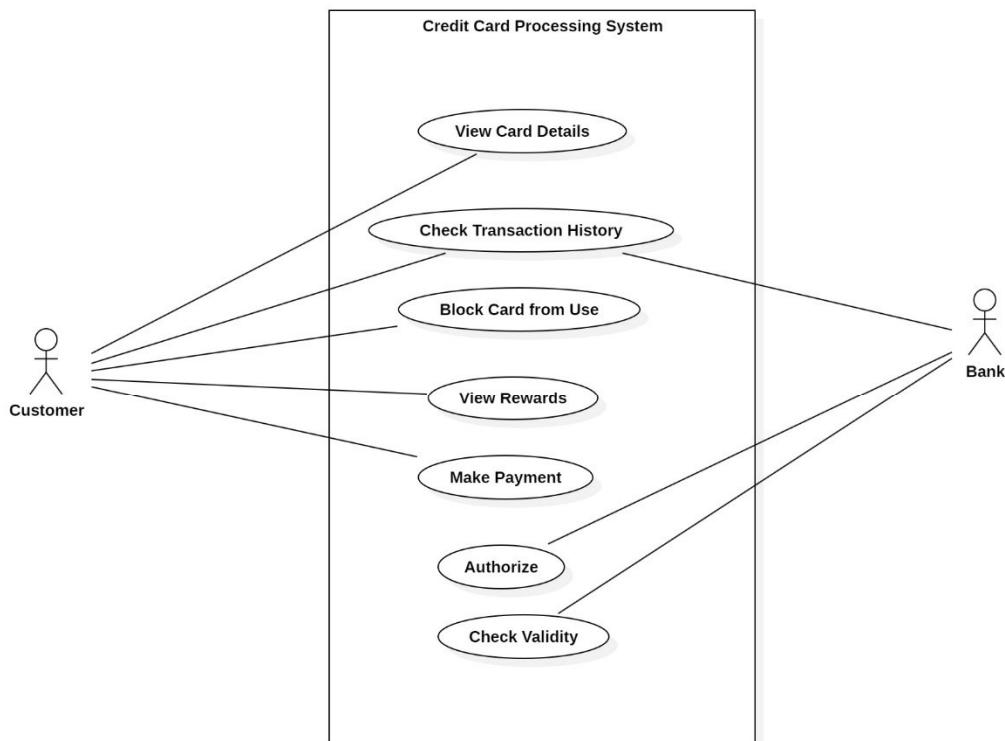


Fig 2.4.1 Use Case Diagram

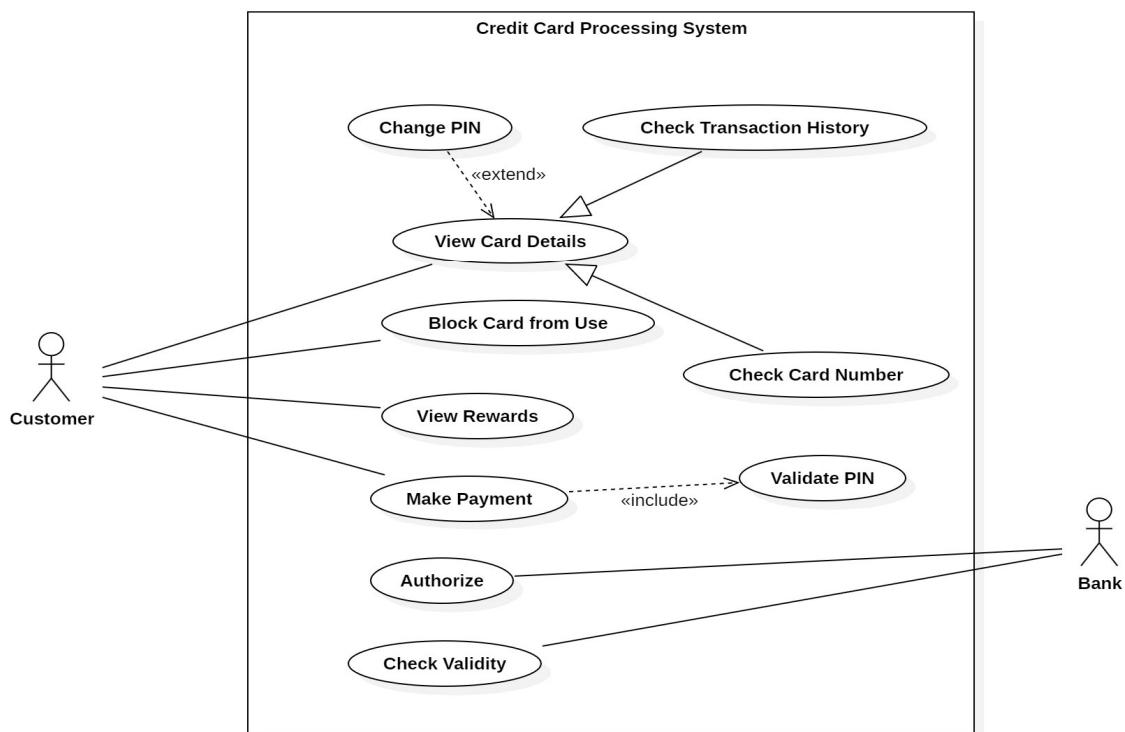


Fig 2.4.2 Advanced Use Case Diagram

This use case diagram provides an overview of the interactions between actors (Customer and Bank) and the **Credit Card Processing System**. Key use cases and their relationships are as follows:

Actors:

1. **Customer**: Interacts with the system to manage credit card-related tasks.
2. **Bank**: Handles backend validation and credit management processes.

Use Cases:

1. **View Card Details**: The customer retrieves information about their credit card.
 - o Includes:
 - **Validate PIN**: Ensures the customer is authorized to access card details.
 - o Extends:
 - **View Transaction History**: Displays past transactions for the card.
 - **View Card Status**: Shows the current status of the card (e.g., active, blocked).
 - o Associated with:
 - **View Rewards**: Displays the reward points earned by the customer.
2. **Block Card From Use**: The customer or bank blocks a card due to loss, theft, or fraud.

- Includes:
 - **Validate PIN:** Confirms customer identity before blocking the card.
3. **Make Payment:** Allows the customer to process payments using the card.
- Includes:
 - **Authorize:** Verifies the transaction by checking validity, credit limit, and expiry date.
 - **Check Validity:** Confirms the card's details.
 - **Check Credit Limit:** Ensures sufficient credit is available.
 - **Check Expiry Date:** Validates that the card is not expired.
 - Extends:
 - **Apply Discount:** Applies eligible discounts to the transaction.
4. **Change PIN:** Enables the customer to modify their credit card PIN.
- Includes:
 - **Validate PIN:** Ensures security before allowing changes.
5. **Refund Payment:** The system processes refunds for incorrect or canceled transactions.
- Extends:
 - **Make Payment:** Optional use case to reverse a completed transaction.

Relationships:

- **Includes:** Represents essential tasks that are part of a broader use case.
 - Example: Make Payment includes Authorize, which further includes Check Validity, Check Credit Limit, and Check Expiry Date.
- **Extends:** Represents optional or conditional tasks that extend the base use case.
 - Example: Apply Discount extends Make Payment when eligible discounts are available.

2.5 Sequence Diagram

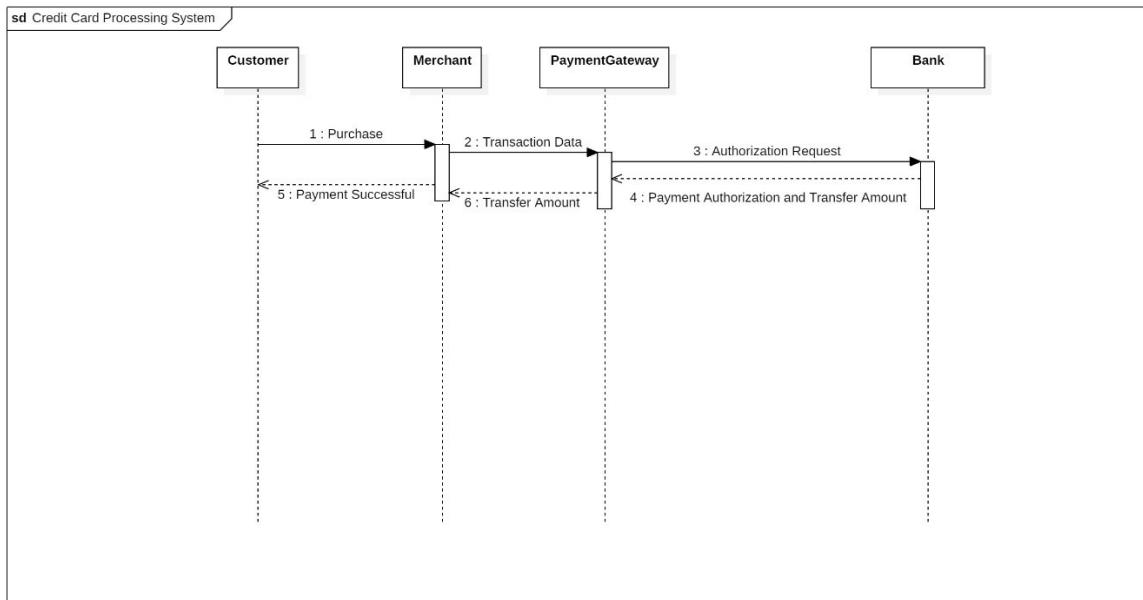


Fig 2.5.1 Sequence Diagram

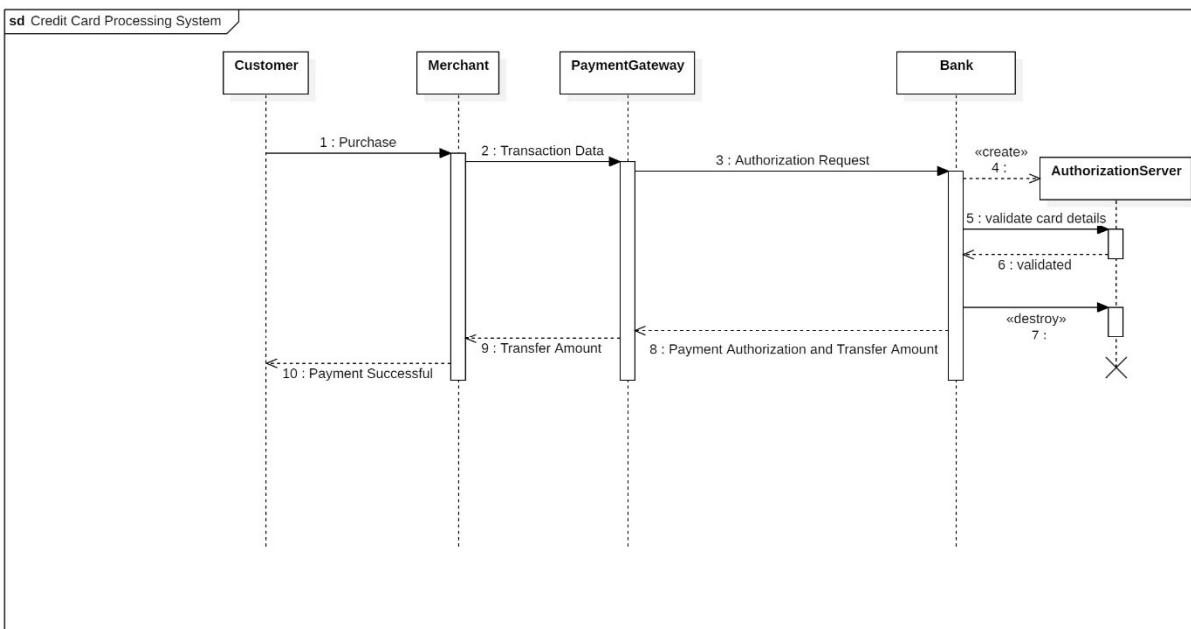


Fig 2.5.2 Advanced Sequence Diagram

This sequence diagram illustrates the flow of interactions between various components in a **Credit Card Processing System**, namely: Customer, Payment Gateway, Issuer Bank, and Merchant. Here's a detailed breakdown:

1. Enter Card Details (Customer → Payment Gateway):

- The customer initiates the transaction by entering their card details into the payment system.

2. Validate Payment Details (Payment Gateway):

- The payment gateway validates the provided card details (e.g., card number, expiry date, and CVV).
- If the validation fails, the system displays a **Payment Failure** message to the customer.

3. Send Request to Issuer Bank (Payment Gateway → Issuer Bank):

- If the details are valid, the payment gateway sends a request to the issuer bank to check for sufficient credit and authorization.

4. Check Credit (Issuer Bank → Payment Gateway):

- The issuer bank verifies the credit limit and authorizes or denies the transaction.
- **If authorized:**
 - The transaction proceeds.
- **If not authorized:**
 - The system sends a **Payment Failure** message to the customer.

5. Transfer Amount to Merchant (Issuer Bank → Merchant):

- Upon successful authorization, the issuer bank transfers the amount to the merchant's account.

6. Payment Successful (Payment Gateway → Customer):

- The payment gateway notifies the customer of a successful transaction.

Additional Details:

- **Error Handling:**
 - If any step (e.g., validation or credit check) fails, the process halts, and the customer is informed.
- **Final State:**
 - The sequence concludes with either a successful payment confirmation or a failure notification.

2.6 Activity Diagram

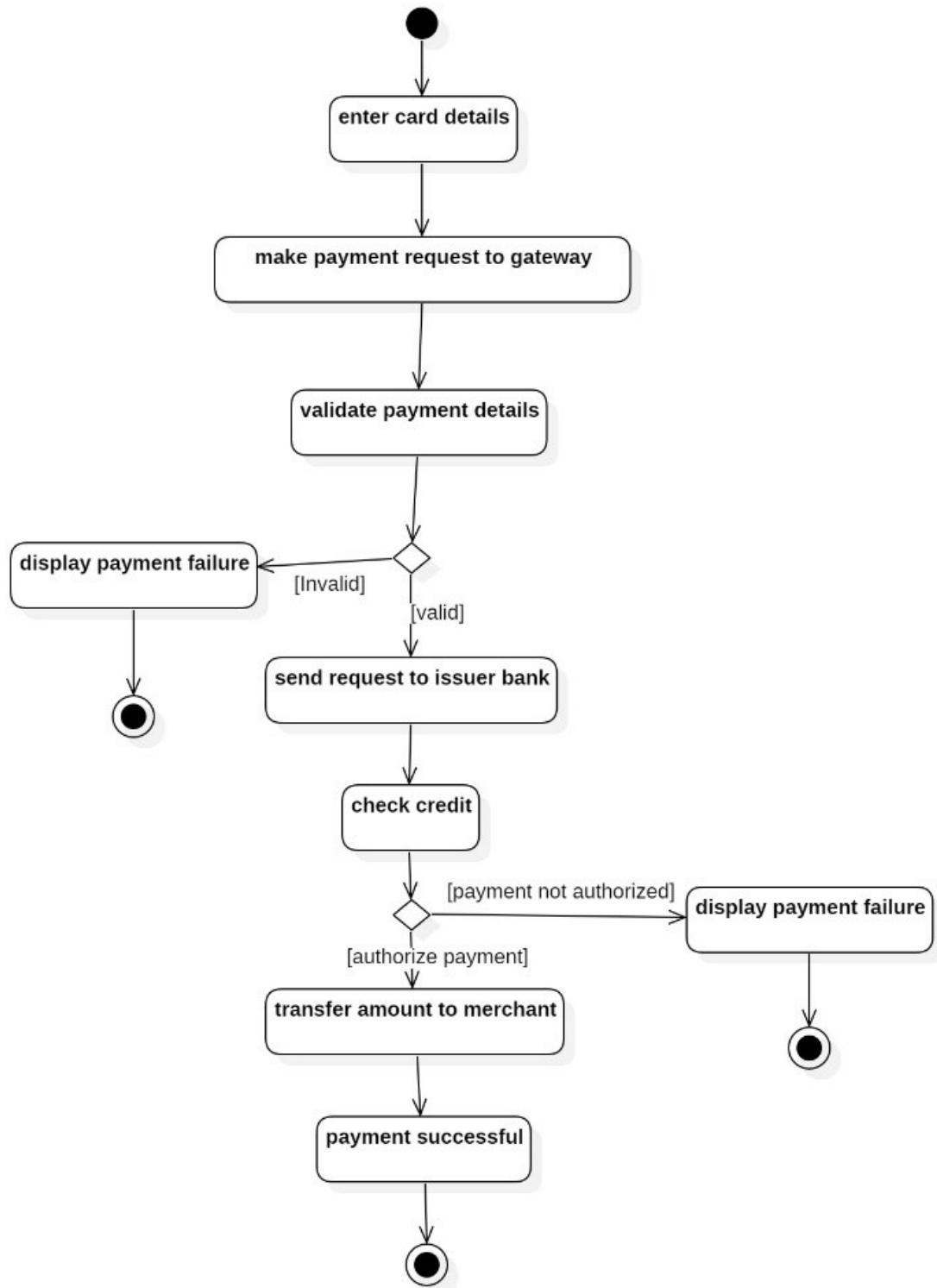


Fig 2.6.1 Activity Diagram

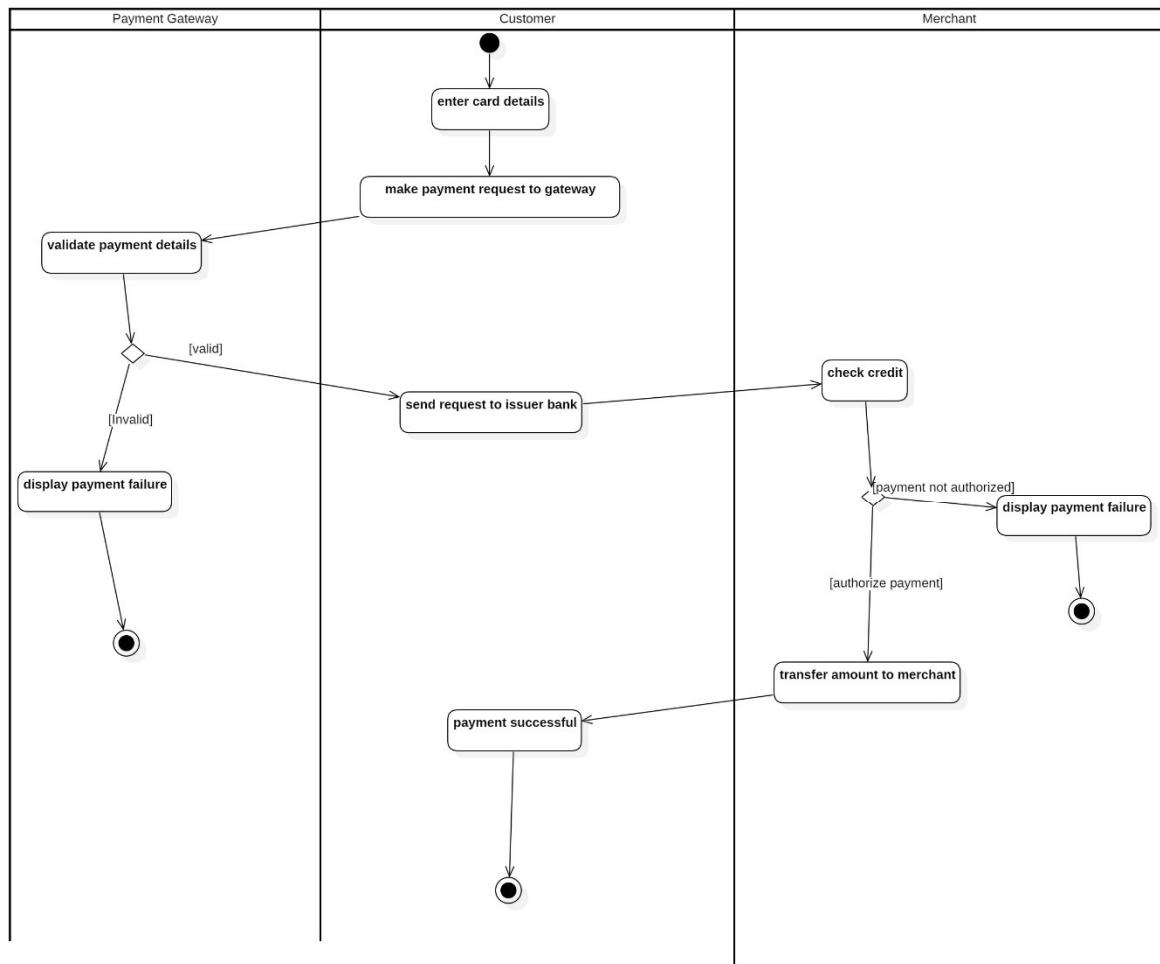


Fig 2.6.2 Advanced Activity Diagram

The activity diagram for the **Credit Card Processing System** illustrates the step-by-step workflow of credit card payment processing. Below is the detailed breakdown:

Start Point:

- The process begins with the **Customer** entering their card details into the payment system.

Key Activities:

1. Enter Card Details:

- The customer provides their card details, including card number, expiry date, and CVV.

2. Make Payment Request:

- The payment request is sent from the customer to the **Payment Gateway**.

3. Validate Payment Details (Payment Gateway):

- The system validates the card details for accuracy and completeness.
- **Decision Point:**
 - **If valid:** Proceed to send the request to the **Issuer Bank**.
 - **If invalid:** Display **Payment Failure** to the customer and terminate the process.

4. Send Request to Issuer Bank:

- The payment gateway forwards the request to the issuer bank for credit verification and authorization.

5. Check Credit (Issuer Bank):

- The issuer bank verifies if the customer has sufficient credit to process the transaction.
- **Decision Point:**
 - **If authorized:** Proceed with the payment.
 - **If not authorized:** Inform the **Payment Gateway**, which displays **Payment Failure** to the customer.

6. Transfer Amount to Merchant (Issuer Bank → Merchant):

- If the transaction is authorized, the issuer bank transfers the amount to the merchant's account.

7. Confirm Payment:

- The **Payment Gateway** notifies the customer of the successful payment transaction.

Decision Points:

- **Is Card Valid?**
 - If valid, proceed with sending the request to the issuer bank.
 - If invalid, display payment failure to the customer.
- **Is Payment Authorized?**

- If authorized, the amount is transferred to the merchant.
- If not authorized, display payment failure to the customer.

End Point:

- The activity concludes when:
 - The payment is successfully processed, and a confirmation is sent to the customer.
 - Alternatively, the process ends with a payment failure notification.

3. Library Management System

Problem Statement: Managing the operations of a library manually, such as tracking borrowed books, overdue returns, and inventory updates, is time-consuming and prone to errors. A robust system is needed to streamline these operations, improve efficiency, and enhance the experience for library users.

3.1 SRS-Software Requirements Specification

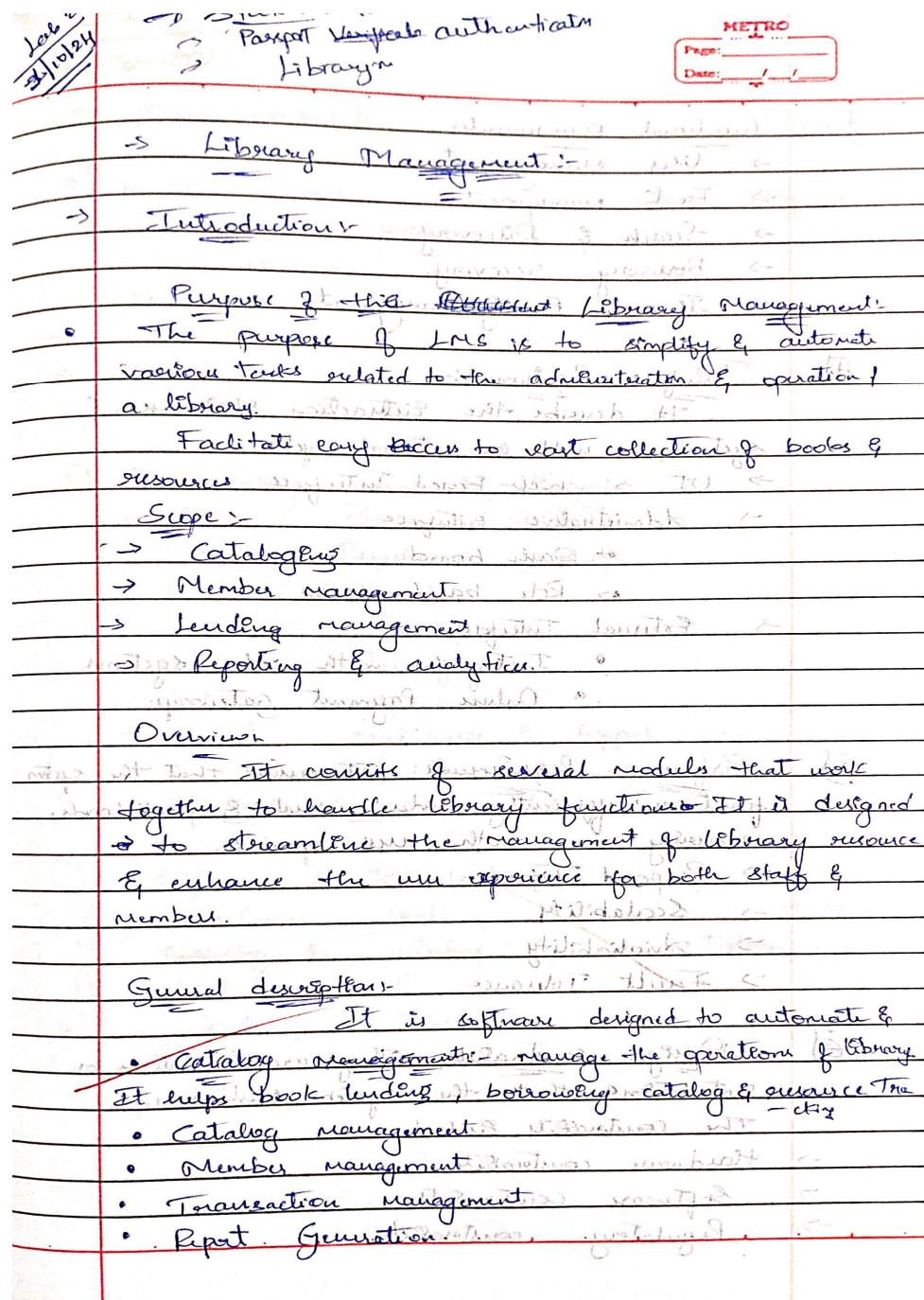


Fig 3.1.1

- 3) → Functional Requirements
→ User management
→ Book management
→ Search & Discovery
→ Borrowing recovery.
→ Fine & Penalty management

4) Interface Requirements :- It describe the Extraction SW user?

system, as well as external system.

- UI → web Based Interface
- Administrative Interface
 - Dash board
 - Role based access
- External Interface
 - Integration with other systems.
 - Online Payment gateway.

5) Performance Requirements :- It ensure that the system functions efficiently under normal & peak loads, ensuring a smooth user experience.

- Response Time
- Scalability
- Availability
- Fault Tolerance

6) Design constraints - They are illustrations of restrictions that the system must develop.

The constraints include:

- Hardware constraints
- Software constraints
- Regulatory constraints

Fig 3.1.2

→ Time constraints

q) Non-functional - It refers to the quantity attr
- Entity of the system.

They focus on the multiple qualities of the system.

→ Performance

→ Security

→ Maintainability

→ Usability.

8) Preliminary Schedule & Budget :-

→ Week 1 :- Project Initiation

→ Week 2 :- Requirements Gathering

→ Week 3 :- Design

→ Week 4 :- Develop

→ Week 5 :- Testing

→ Week 6 :- Deployment

→ Week 7 :- Maintenance & Support.

Estimate Budget breakdown

I) Software & Tool cost 10,000

II) Hardware cost 15,000

III) Training & Traveling & Documentation 5000

IV) Maintenance cost 5000

V) Contingency Fund 11,000

Total estimated budget for the project 41,36,000.

Fig 3.1.3

3.2 Class Diagram

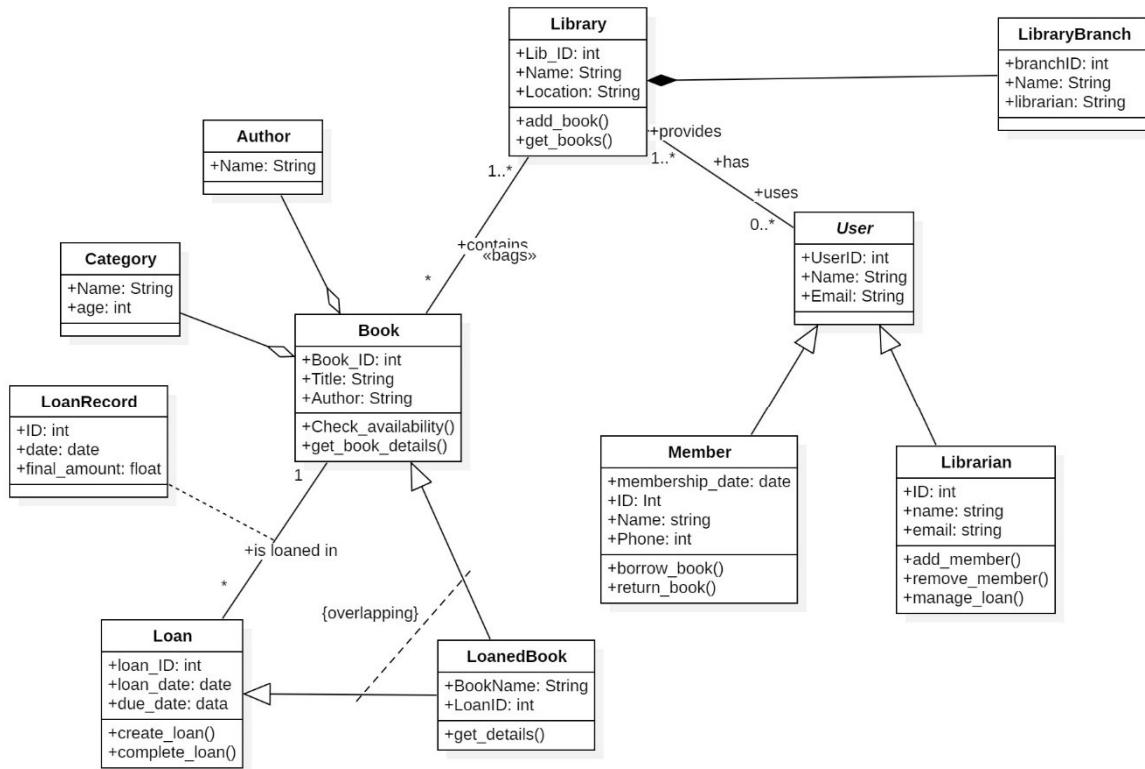


Fig 3.2.1

Core Classes:

1. **Book**: Represents a book in the library with attributes and methods to manage book-related activities.

- o **Attributes**:

- bookName: Name of the book.
- bookID: Unique identifier for the book.
- noOfPages: Total number of pages in the book.

- o **Methods**:

- IssueBook(): Issues a book to a member.
- ReturnBook(): Handles the return of a book.
- OrderBook(): Processes the procurement of a new book.

2. **Member:** Represents a library member with attributes and methods for membership management.

- o **Attributes:**

- memberID: Unique identifier for the member.
- membership: Type of membership (e.g., Student, Professor).
- memberDetails: Additional details about the member.

- o **Methods:**

- createAccount(): Creates a new member account.
- deleteAccount(): Deletes an existing member account.
- manageAccountDetails(): Updates member account details.

3. **LibraryCard:** Represents the library card issued to members.

- o **Attributes:**

- cardID: Unique ID of the library card.
- customerDetails: Information about the cardholder.
- booksIssued: Number of books issued on this card.

- o **Methods:**

- returnBook(): Facilitates the return of books.
- printIssuedBooks(): Displays details of books issued on the card.

4. **Library:** Represents the library entity and its operations.

- o **Attributes:**

- libraryID: Identifier for the library.
- name: Name of the library.
- timings: Operational hours of the library.

- o **Methods:**

- addBook(): Adds a book to the inventory.

- removeBook(): Removes a book from the inventory.
- listBooksByCategory(): Displays books based on category.

5. **Publisher:** Represents publishers associated with the library's books.

○ **Attributes:**

- publisherID: Unique identifier for the publisher.
- name: Name of the publisher.
- contactNo: Contact details of the publisher.

○ **Methods:**

- listPublisherDetails(): Retrieves publisher information.
- getPublishedBooks(): Lists books published by the publisher.

6. **Librarian:** Represents the librarian managing the library system.

○ **Attributes:**

- librarianID: Unique ID for the librarian.
- name: Librarian's name.
- email: Contact email of the librarian.

○ **Methods:**

- assistMember(): Assists members with inquiries.
- manageInventory(): Manages the library's book inventory.
- generateReport(): Prepares reports on library activities.

Specialized Entities:

1. **Student:** Inherits from the Member class, representing student members.

○ **Attributes:**

- ageGroup: Specifies the student's age range.
- grade: Represents the student's grade.
- school: Name of the school the student attends.

- **Methods:**
 - getDetails(): Retrieves details specific to the student.
- 2. **Professor:** Inherits from the Member class, representing professor members.
 - **Attributes:**
 - field: The professor's field of expertise.
 - occupation: Designation or role in the institution.
 - **Methods:**
 - getDetails(): Retrieves details specific to the professor.
- 3. **Assistance:** Represents help provided by the library staff.
 - **Attributes:**
 - date: Date the assistance was provided.
 - purpose: Reason or purpose for the assistance.
 - **Methods:**
 - getDetails(): Provides details about the assistance.
- 4. **IssueCard:** Tracks the issue and return of books.
 - **Attributes:**
 - issueDate: Date of book issue.
 - returnDate: Expected return date for issued books.
 - **Methods:**
 - display(): Displays issue card details.
 - calculateFine(): Calculates fines for overdue books.

3.3 State Machine Diagram

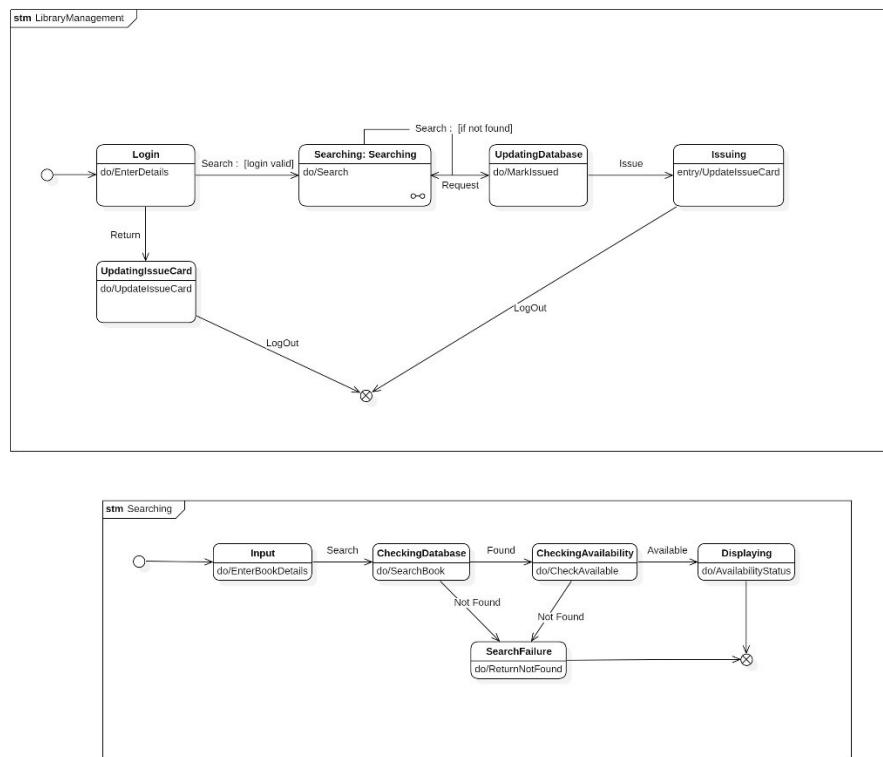


Fig 3.3.1 State Diagram

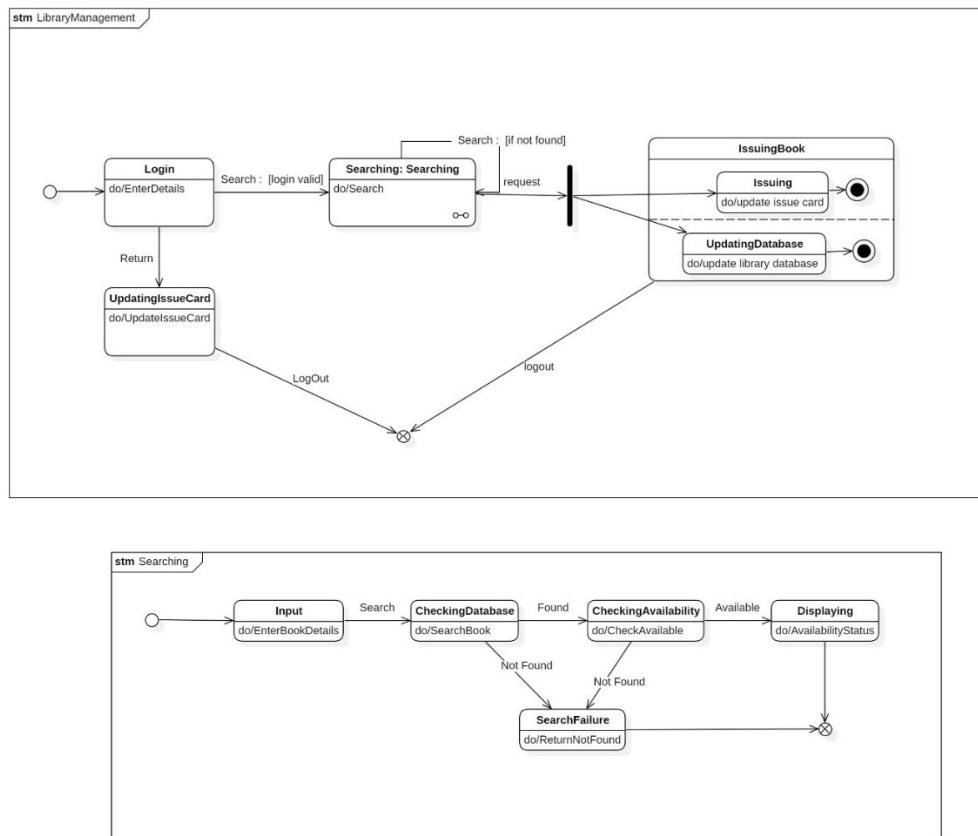


Fig 3.3.2 Advanced State Diagram

State Machine: Book Borrowing

- **Initial State:** The process begins at the "Request Book" state.
- **Request Book:** Represents the initial step where a library member requests a book.
 - **Actions:** do/request book: Records the book request.
 - **Transitions:**
 - If the book is available, transitions to the "Check Member Eligibility" state.
 - If unavailable, transitions to the "Waitlist Book" state.
- **Check Member Eligibility:** Verifies the borrowing limit and membership validity.
 - **Actions:**
 - do/check borrowing limit.
 - do/validate membership.
 - **Transitions:**
 - If eligible, transitions to the "Issue Book" state.
 - If ineligible (e.g., expired membership), transitions to a termination or retry state.
- **Issue Book:** Issues the requested book to the member.
 - **Actions:**
 - do/update inventory.
 - do/register issued book.
 - **Transitions:** Leads to the "Borrowed" state.
- **Borrowed:** Marks the book as borrowed.
 - **Actions:** do/track due date.
 - **Transitions:** Ends at the final state after the book has been successfully issued.
- **Final State:** Reached upon successful completion of the book borrowing process.

State Machine: Book Return

- **Initial State:** The process begins at the "Initiate Return" state.
 - **Action:** do/return book request.
- **Verify Return:** Confirms the details of the returned book.
 - **Actions:**
 - do/check book condition.
 - do/validate return date.
 - **Transitions:**
 - If valid, transitions to the "Process Return" state.
 - If invalid (e.g., damaged book), transitions to the "Fine Calculation" state.
- **Fine Calculation:** Calculates late fees or damage penalties.
 - **Actions:**
 - do/calculate overdue fine.
 - do/calculate damage fee.
 - **Transitions:**
 - If fines are paid, proceeds to the "Process Return" state.
 - If unpaid, allows retry or transitions to a termination state.
- **Process Return:** Updates the inventory and member account details.
 - **Actions:**
 - do/update inventory.
 - do/clear borrowed record.
 - **Transitions:** Leads to the "Returned" state.
- **Returned:** Marks the book as returned.
 - **Actions:** do/notify member of successful return.
 - **Transitions:** Ends at the final state.

- **Final State:** Reached upon successful return of the book.

State Machine: Membership Management

- **Initial State:** The process begins at the "Create Account" state.
 - **Action:** do/register member.
- **Validate Details:** Verifies the personal details of the member.
 - **Actions:**
 - do/validate name.
 - do/validate ID proof.
 - **Transitions:**
 - If valid, moves to the "Activate Membership" state.
 - If invalid, transitions to a retry or termination state.
- **Activate Membership:** Activates the membership and generates a library card.
 - **Actions:**
 - do/generate library card.
 - do/update membership database.
 - **Transitions:** Leads to the "Membership Active" state.
- **Membership Active:** Marks the membership as active and ready for use.
 - **Actions:** do/send confirmation to member.
 - **Transitions:** Ends at the final state.
- **Final State:** Reached upon successful activation of the membership.

3.4 Use Case Diagram

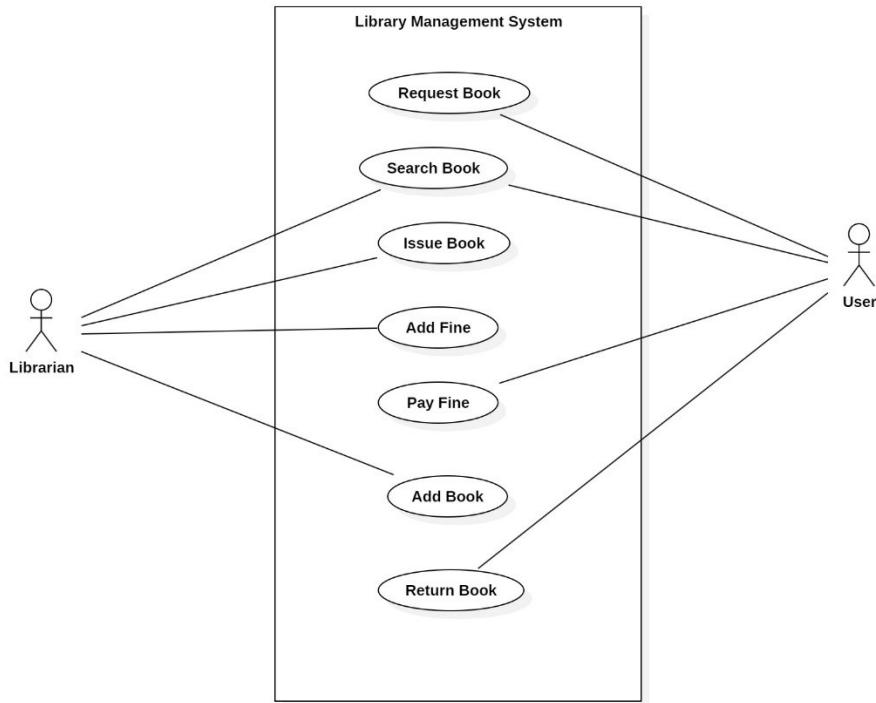


Fig 3.4.1 Use Case Diagram

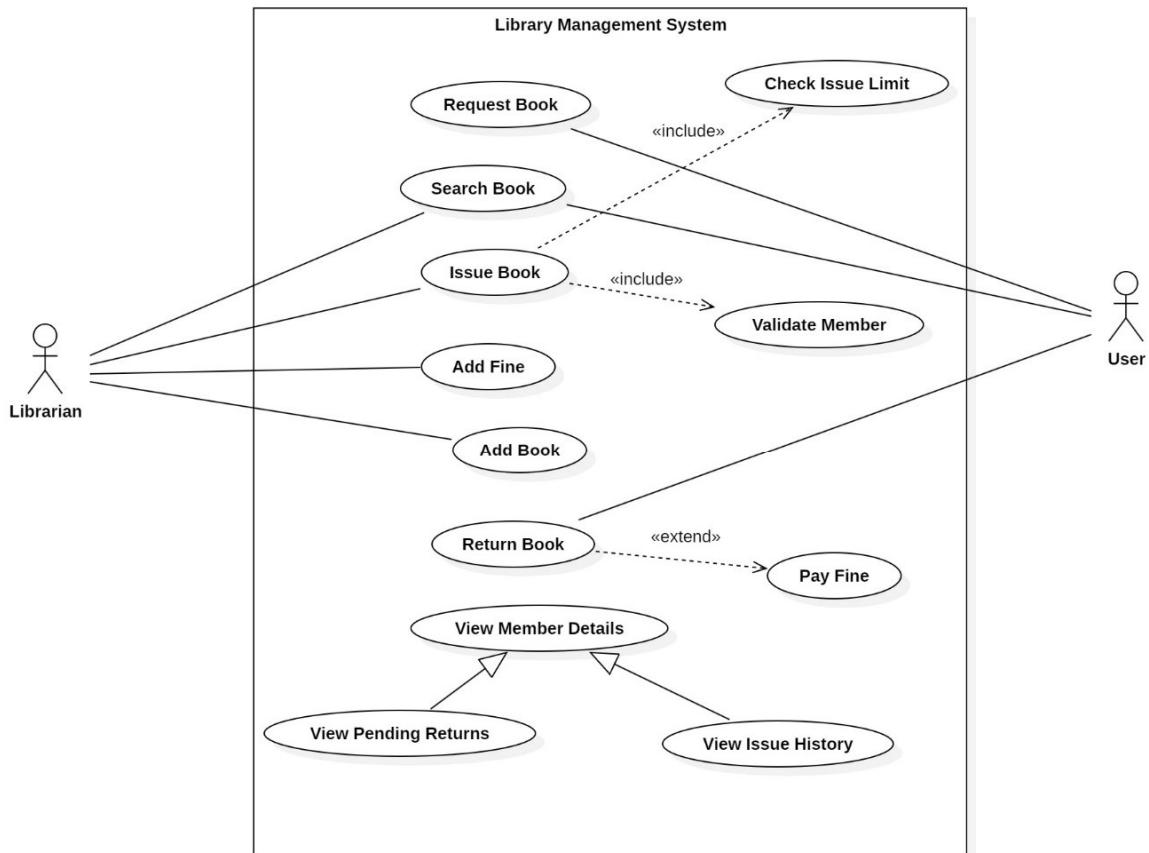


Fig 3.4.2 Advanced Use Case Diagram

Actors:

1. **Library Member:** Interacts with the system to perform book-related and membership-related activities.
2. **Librarian:** Manages the operational tasks of book lending, returns, and inventory updates.
3. **System Admin:** Oversees system configurations and user account management.

Use Cases:

1. **Borrow Book:** The library member borrows a book from the library.
 - **Includes:**
 - **Check Book Availability:** Ensures the requested book is available for borrowing.
 - **Verify Member Eligibility:** Confirms that the member has an active membership and hasn't exceeded the borrowing limit.
 - **Extends:**
 - **Reserve Book:** Allows the member to reserve the book if unavailable.
 - **Associated with:**
 - **Update Borrow History:** Records the borrowed book in the member's account.
2. **Return Book:** The library member returns a borrowed book to the library.
 - **Includes:**
 - **Verify Return Details:** Checks the book condition and return date.
 - **Extends:**
 - **Calculate Late Fee:** Applies overdue charges if the book is returned after the due date.
 - **Notify Librarian:** Sends a notification to the librarian for any book damage or overdue returns.
3. **Search Catalog:** The library member searches for books in the library database.
 - **Includes:**
 - **Filter Results:** Narrows search results based on genre, author, or publication year.
 - **View Book Details:** Displays details such as availability, location, and a brief summary of the book.
4. **Renew Membership:** Enables the library member to extend their membership validity.
 - **Includes:**

- **Validate Member Account:** Ensures the account is eligible for renewal.
 - **Process Payment:** Handles the payment process for membership renewal.
5. **Manage Inventory:** Allows the librarian to add, update, or remove books in the library database.
- **Includes:**
 - **Add New Book:** Registers a new book in the system.
 - **Update Book Details:** Edits book information such as title, author, or availability.
 - **Remove Book:** Deletes outdated or unavailable books from the inventory.
6. **Register Member:** The librarian or system admin registers a new library member.
- **Includes:**
 - **Validate Member Details:** Ensures that the provided details (e.g., name, ID proof) are accurate.
 - **Generate Membership ID:** Issues a unique membership ID for the new user.
7. **Generate Reports:** Allows the librarian or system admin to create reports on library usage.
- **Includes:**
 - **Generate Borrowing Statistics:** Summarizes borrowing trends over a specific period.
 - **Generate Overdue Reports:** Lists members with overdue books.

Relationships:

- **Includes:** Represents essential tasks that are part of a broader use case.
 - **Example:** Borrow Book includes Check Book Availability and Verify Member Eligibility, both of which are necessary steps.
- **Extends:** Represents optional or conditional tasks that extend the base use case.
 - **Example:** Calculate Late Fee extends Return Book when the book is returned late.

3.5 Sequence Diagram

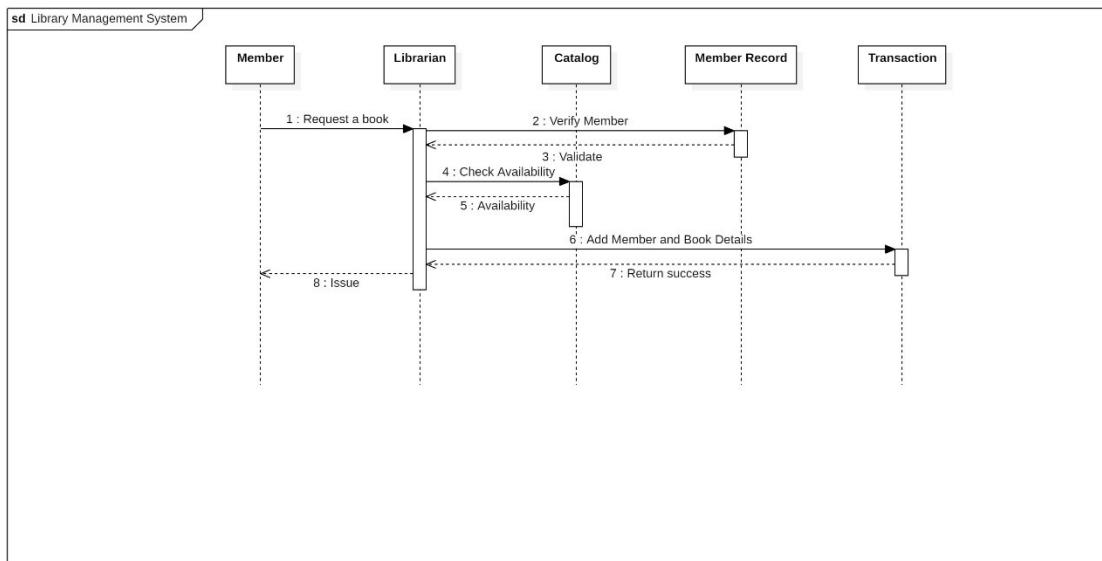


Fig 3.5.1 Sequence Diagram

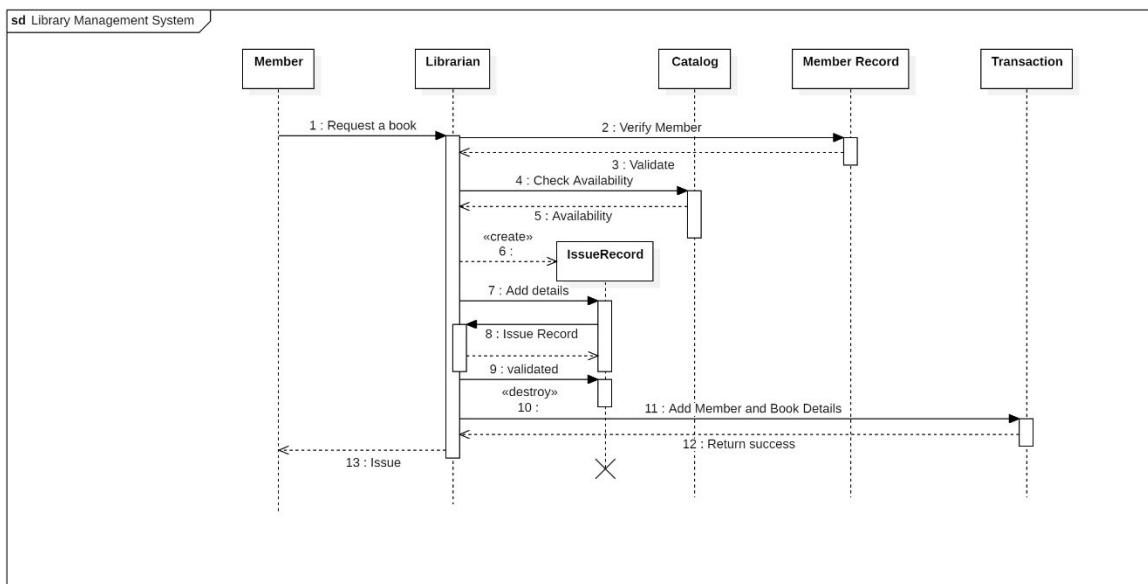


Fig 3.5.2 Advanced Sequence Diagram

This sequence diagram illustrates the flow of interactions between various components in a Library Book Borrowing System, namely: **Library Member**, **Library System**, **Librarian**, and **Inventory Database**. Here's a detailed breakdown:

1. Search for Book Availability (Library Member → Library System):

- The library member initiates the borrowing process by searching for the desired book in the library system.

2. Check Book Inventory (Library System → Inventory Database):

- The library system queries the inventory database to check for the availability of the requested book.
- If the book is unavailable:
 - The system notifies the member of the unavailability and offers the option to reserve the book.
- If the book is available:
 - The process proceeds to the next step.

3. Borrow Request Initiation (Library Member → Library System):

- The library member initiates the borrowing request through the system.

4. Verify Membership (Library System → Librarian):

- The system notifies the librarian to validate the member's eligibility (e.g., active membership and borrowing limit).
- If the membership is invalid:
 - The librarian informs the member, and the process ends.
- If the membership is valid:
 - The process continues.

5. Update Inventory and Borrow History (Library System → Inventory Database):

- Upon successful verification, the library system updates the inventory database to mark the book as borrowed.
- It also records the transaction in the member's borrowing history

6. Handover Book (Librarian → Library Member):

- The librarian hands over the physical book to the member, completing the borrowing process.

Additional Details:

- **Error Handling:**

- If any step fails (e.g., the book is unavailable, or the member's account is invalid), the process halts, and the system informs the member.
- **Final State:**
 - The sequence concludes with either a successful handover of the book or a notification of failure due to unmet conditions.

3.6 Activity Diagram

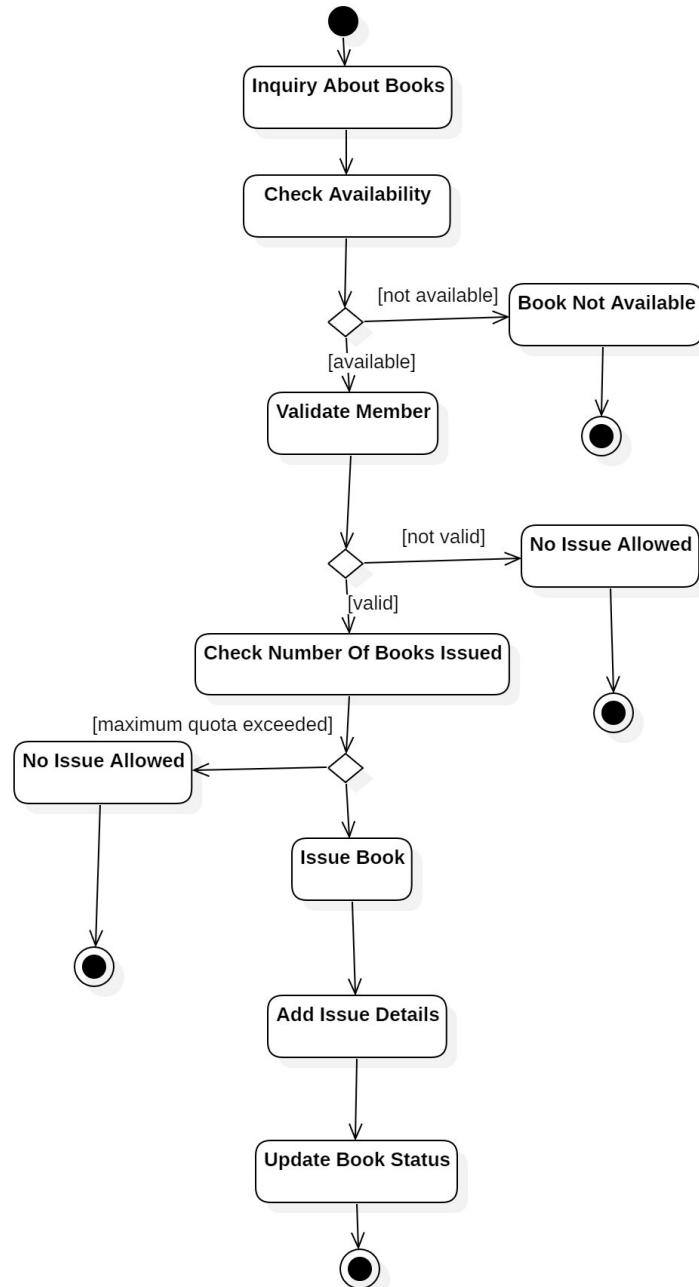


Fig 3.6.1 Activity Diagram

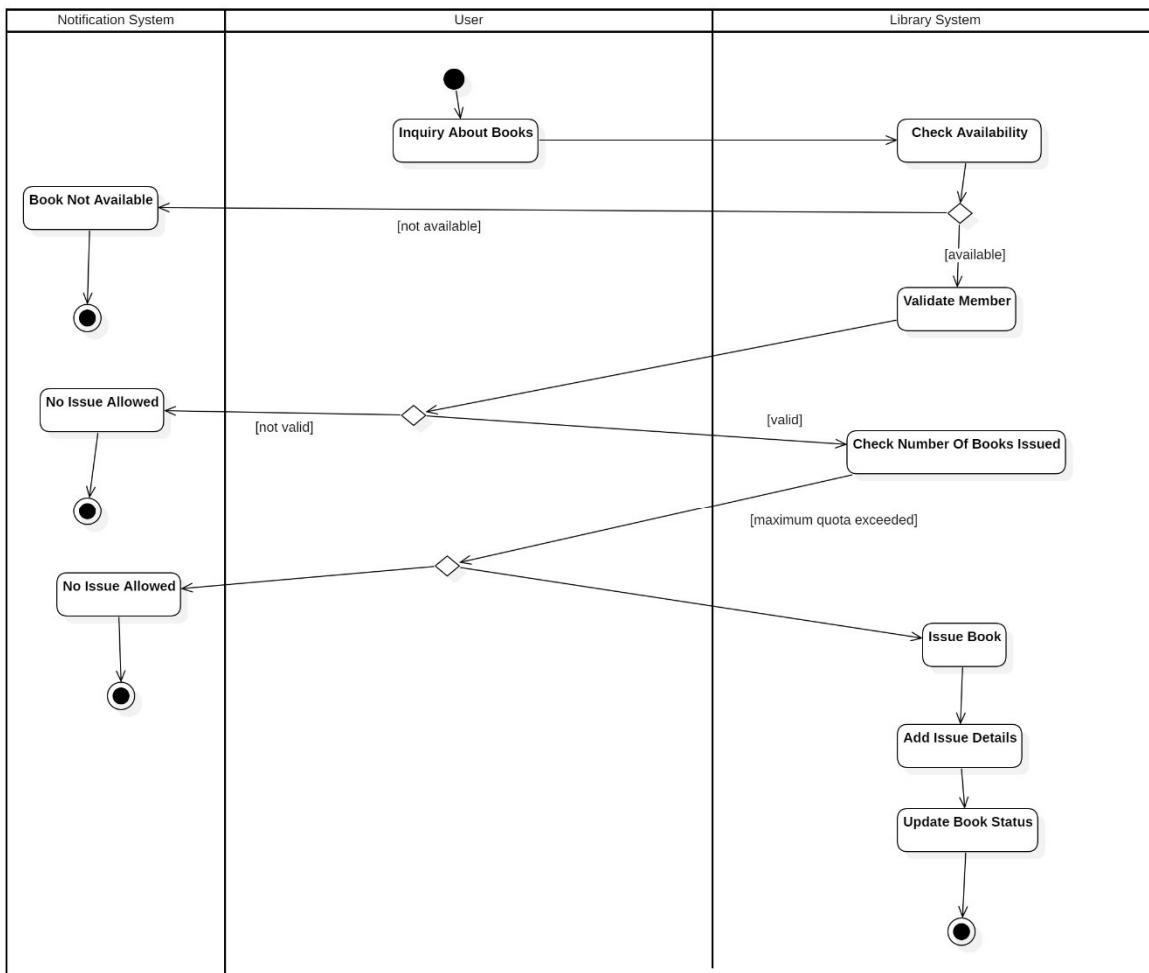


Fig 3.6.2 Advanced Activity Diagram

The activity diagram for the Library Book Borrowing System illustrates the step-by-step workflow of borrowing a book. Below is the detailed breakdown:

Start Point:

- The process begins when the Library Member initiates a book search in the library system.

Key Activities:

1. Search for Book:

- The member searches for the desired book by entering the title or author into the system.

2. Check Book Availability:

- The library system queries the inventory database to determine the book's availability.

3. Decision Point: Is the Book Available?

- **If available:** Proceed to initiate a borrow request.
- **If not available:** Display a message offering the option to reserve the book.

4. Initiate Borrow Request:

- If the book is available, the member submits a borrow request.

5. Verify Membership:

- The librarian verifies the member's eligibility (e.g., active membership and borrowing limit).

6. Decision Point: Is Membership Valid?

- **If valid:** Proceed to update inventory and borrowing history.
- **If invalid:** Display a message rejecting the borrow request and terminate the process.

7. Update Inventory and Borrow History:

- The library system updates the inventory database to mark the book as borrowed.
- It also logs the transaction in the member's borrowing history.

8. Handover Book:

- The librarian hands the physical book to the member, completing the borrowing process.

Decision Points:

1. Is the Book Available?

- **If yes:** Proceed with the borrowing process.
- **If no:** Display an option to reserve the book or end the process.

2. Is Membership Valid?

- **If yes:** Allow the borrowing request to proceed.

- **If no:** Reject the request and terminate the process.

End Point:

- The activity concludes when:
 - The book is successfully borrowed, and the member receives the book.
 - Alternatively, the process ends if the book is unavailable or the membership is invalid.

4. Passport Automation System

Problem Statement: Managing stock levels manually in a business is inefficient, prone to human error, and can lead to overstocking or understocking issues. A well-organized system is required to monitor inventory, track stock movements, and ensure the availability of items to meet customer demands efficiently.

4.1 SRS-Software Requirements Specification

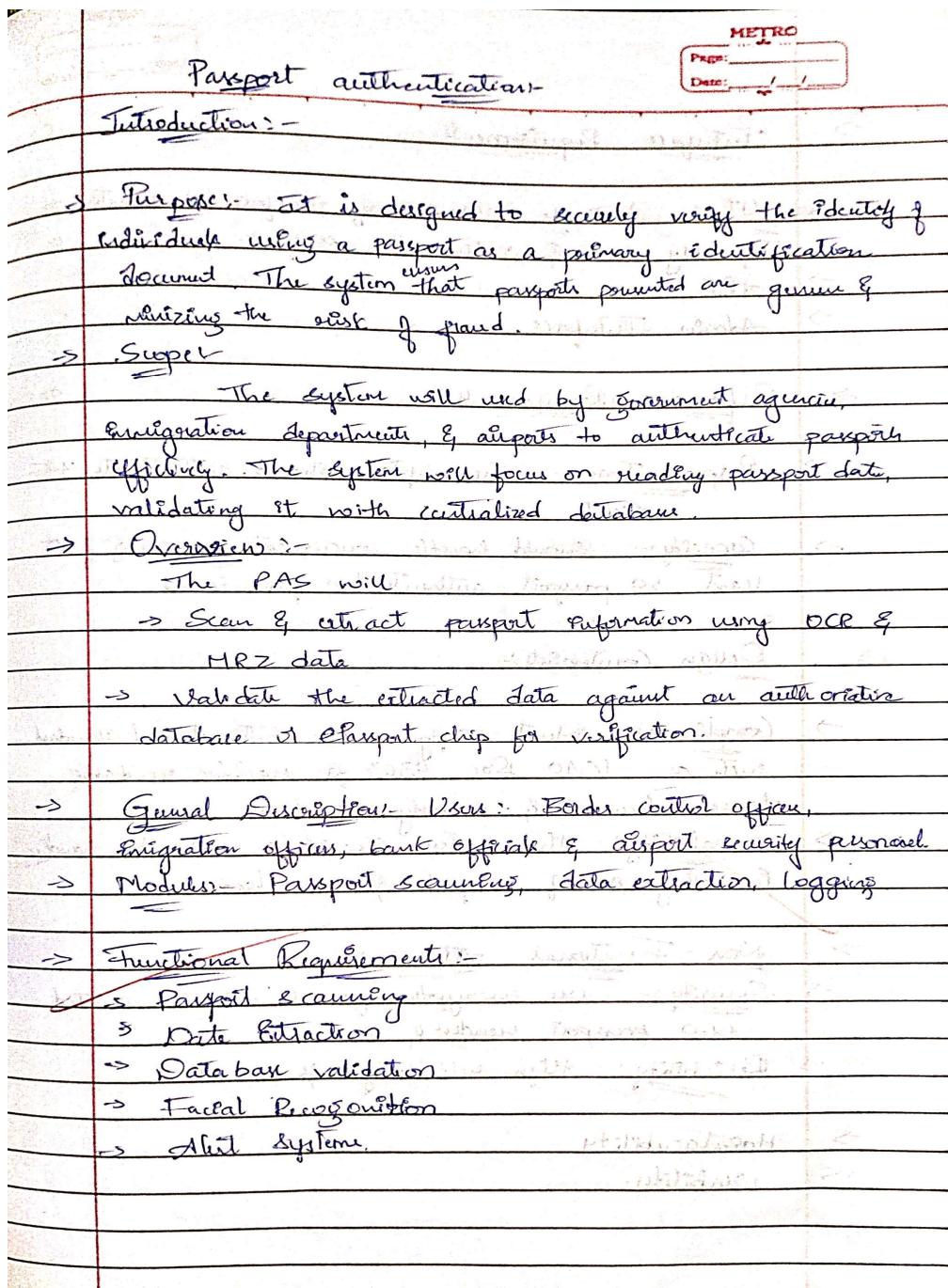


Fig 4.1.1

	Page: 340 Date: 21/10/2023	
→	<u>Interface Requirements:-</u>	→ Pers
→	UI :- A simple, user-friendly interface for operators to quickly scan & authenticate passports.	→ Plan
→	API	→ Do
→	Admin Interface.	→ Fe
→	<u>Performance Requirements:-</u>	→ To
→	Response Time :- The system should authenticate 3-5 seconds.	→ Be
→	Capacity :- Should handle concurrent processing of at least 50 passport authentication per second.	→ E
→	<u>Design Constraints:-</u>	→ C
→	Compliance :- Must comply with international standards such as ICAO Doc 9303 for machine-readable travel documents & security standards.	→ To
→	Scalability :- The system should be able to handle increasing no. of passports & user loads.	→ Obj
→	<u>Non-Functional Attributes:-</u>	
→	Security :- Use encryption for all data transmitted via passport readers & servers.	
→	Reliability :- High availability is required.	
→	Usability	
→	Reliability	

Fig 4.1.2

Preliminary Scheduler

- Planning & Requirement Gathering : 2 weeks
- Design & Prototyping : 3-4 weeks
- Development - 8-10 w
- Testing & Quality Assurance - 3 w
- Deployment : 2 w.
- Total Estimated Duration - 18-20 w

Budget :-

- Development Team Costs :- \$ 80,000
- Hardware Costs : \$ 10,000
- SW Licenses : \$ 5,000
- Maintenance Costs : \$ 15,000

Total Project Budget Estimate : \$ 105,000

Aug 2020

Fig 4.1.3

4.2 Class Diagram

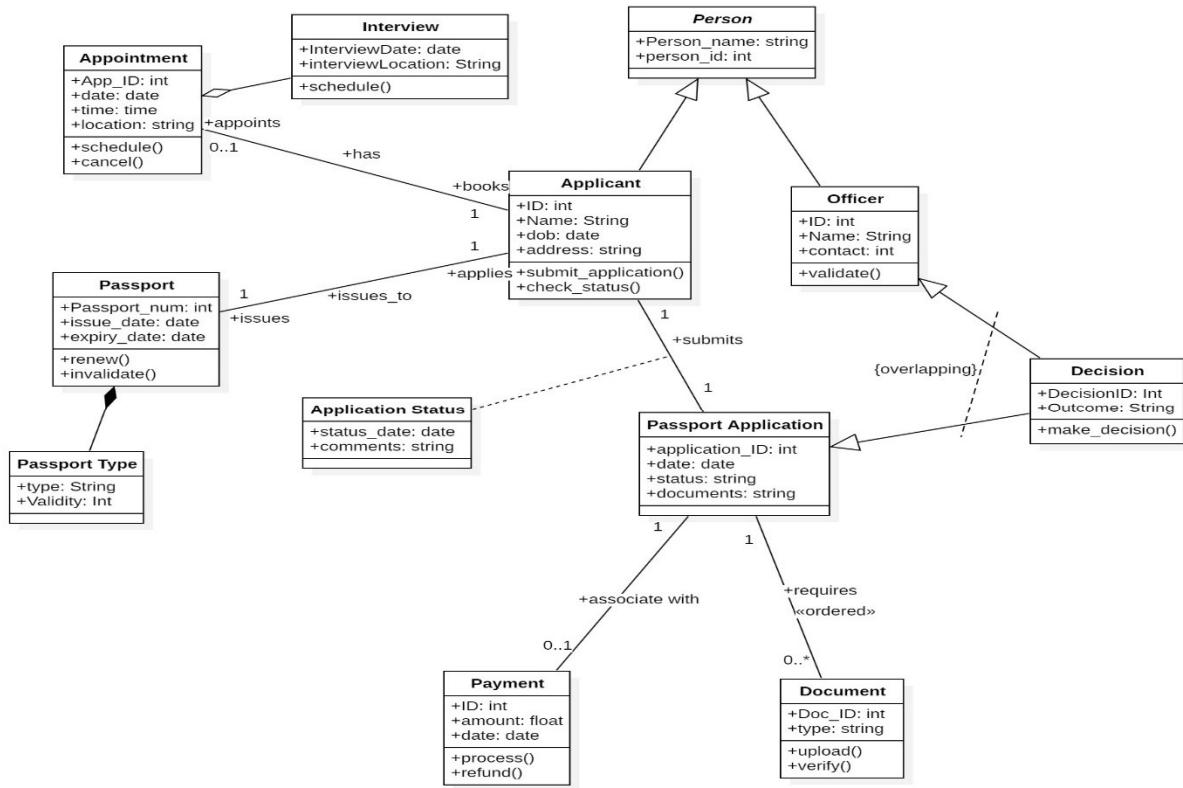


Fig 4.2.1

The class diagram represents a **Passport Automation System** that encompasses key entities and their interactions. Below are the major elements:

Core Classes:

- **Applicant**: Represents the individual applying for a passport. Attributes include:
 - applicant_id (unique ID of the applicant)
 - name, address, contact (applicant details)
 - Methods:
 - apply_for_passport(): Initiates the passport application process.
 - update_details(): Allows the applicant to modify their details.
- **Application**: Manages the passport application process. Attributes include:
 - application_id, status, submission_date, type (type of application: new, renewal, lost).

- Methods:
 - submit_application(): Records the submission of a passport application.
 - update_application(): Updates application details based on applicant input.
- **Document:** Represents documents submitted for verification. Attributes include:
 - doc_type, issue_date, expiry_date, retrieved (indicates if retrieved).
 - Methods:
 - validate(): Checks the validity of the document.
- **Officer:** Represents the personnel verifying applications. Attributes include:
 - officer_id, name, department.
 - Methods:
 - conduct_verification(): Reviews and validates the application and documents.
 - update_status(): Updates the verification status of an application.
- **Verification:** Tracks the verification process. Attributes include:
 - verification_id, status (e.g., Pending, Verified, Rejected), and date.
 - Methods:
 - update_status(): Modifies the status of verification based on results.
- **Passport:** Represents the finalized passport issued to the applicant. Attributes include:
 - passport_number, issue_date, expiry_date.
 - Methods:
 - validate(): Ensures the passport details comply with requirements.
 - get_passport_details(): Retrieves passport information.

Specialized Entities:

- **Fee:** Represents payment information. Attributes include:
 - fee_id, amount, and payment_status.

- Methods:
 - pay_fee(): Processes application fees.
- **Embassy:** Represents the embassy's involvement. Attributes include:
 - embassy_id, contact, and address.
 - Methods:
 - process_delivery(): Manages passport delivery to applicants.

Utility Classes:

- **Status:** Manages the current state of applications. Attributes include:
 - status_type (e.g., Pending, Approved, Rejected).
 - Methods:
 - check_status(): Retrieves the status of an application.

Relationships:

- **Inheritance:**
 - Initial_Application, Renewal_Application, and Lost_Passport_Application inherit from Application.
- **Aggregation:**
 - Applicant aggregates Application and Document, indicating a one-to-many relationship.
- **Dependency:**
 - Verification depends on Officer to conduct the process.
 - Fee depends on Application for payment tracking.
- **Associations:**
 - Applicant is associated with Application through a one-to-many relationship.
 - Passport interacts with Embassy for delivery.

4.3 State Machine Model

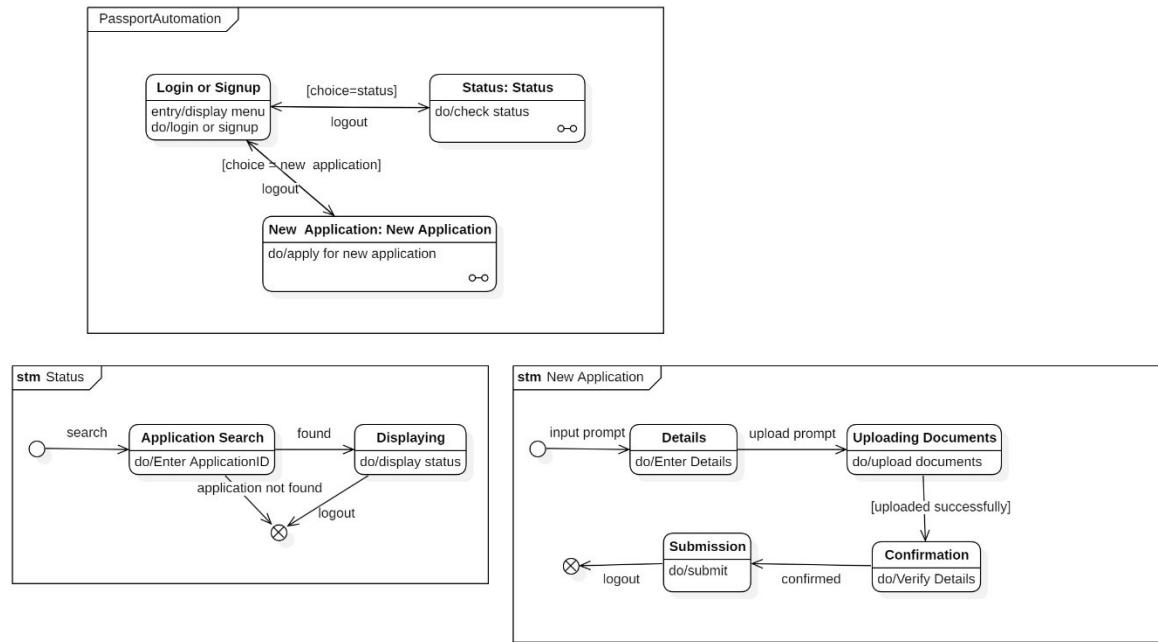


Fig 4.3.1 State Diagram

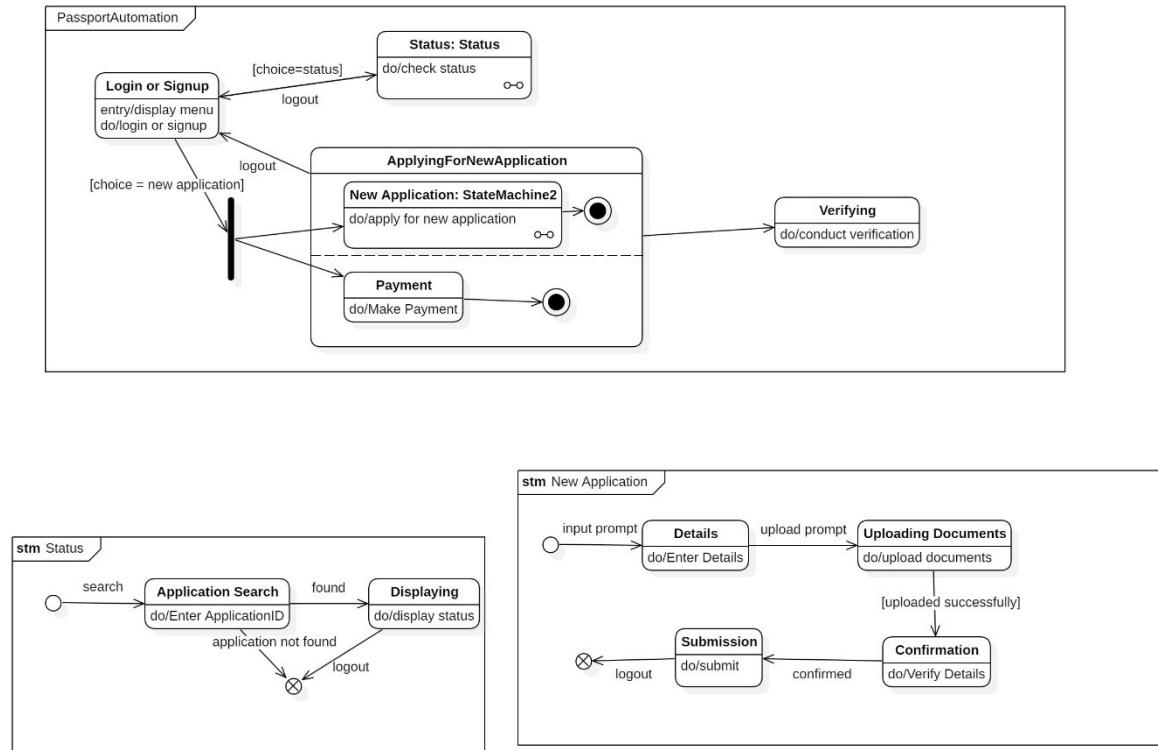


Fig 4.3.2 Advanced State Diagram

State Machine: Passport Application Process

- **Initial State:** The process begins at the **Application Submission** state.
- **Application Submission:** Represents the initial step of submitting the passport application.
 - **Actions:**
 - do/submit application: Records applicant details and application type.
 - **Transitions:**
 - Moves to the **Verification** state after submission.
- **Verification:** Ensures that the documents and application details are valid.
 - **Actions:**
 - do/verify documents: Reviews submitted documents.
 - do/check application details: Confirms the accuracy of applicant details.
 - **Transitions:**
 - If all documents are valid, transitions to the **Approval** state.
 - If documents are invalid, transitions to the **Correction** state.
- **Correction:** Allows applicants to update and resubmit details.
 - **Actions:**
 - do/request corrections: Informs applicants of required changes.
 - do/accept revised documents: Records resubmitted information.
 - **Transitions:**
 - If updated successfully, transitions back to the **Verification** state.
- **Approval:** The application is reviewed for final approval.
 - **Actions:**
 - do/review application: Conducts final checks by the officer.
 - **Transitions:**
 - If approved, moves to the **Fee Payment** state.
 - If rejected, transitions to a **Rejection** or **Termination** state.
- **Fee Payment:** Represents the payment of application fees.
 - **Actions:**
 - do/pay application fee: Processes payment for the application.
 - **Transitions:**
 - If successful, transitions to the **Passport Issuance** state.
 - If unsuccessful, allows retry or moves to a **Termination** state.
- **Passport Issuance:** Represents the printing and issuance of the passport.

- **Actions:**
 - do/print passport: Generates the passport.
 - do/deliver passport: Sends the passport to the applicant.
- **Final State:** Reached after the passport is successfully issued.

State Machine: Document Verification

- **Initial State:** Starts with the **Document Submission** state.
 - **Action:**
 - do/submit documents: Receives necessary documents for verification.
- **Validation:** Reviews each document for compliance and validity.
 - **Actions:**
 - do/validate ID proof: Confirms the authenticity of ID proof.
 - do/verify address proof: Ensures address details are accurate.
 - **Transitions:**
 - If all documents are valid, moves to the **Verification Completed** state.
 - If any document is invalid, transitions to **Correction Requested**.
- **Correction Requested:** Requests the applicant to correct and resubmit documents.
 - **Actions:**
 - do/request corrections: Indicates required changes.
 - **Transitions:**
 - If corrected successfully, returns to the **Validation** state.
- **Verification Completed:**
 - Represents the final state of document verification.

State Machine: Passport Delivery Process

- **Initial State:** Begins with the **Passport Dispatch** state.
 - **Action:**
 - do/dispatch passport: Sends the passport to the designated address.
- **In Transit:** Tracks the delivery process.
 - **Action:**
 - do/track delivery: Monitors the status of passport delivery.
 - **Transitions:**
 - If delivered, moves to the **Delivery Confirmed** state.
 - If delivery fails, transitions to **Reattempt Delivery**.
- **Reattempt Delivery:** Attempts to resolve delivery issues.

- **Actions:**
 - do/verify address: Confirms the recipient's details.
 - do/schedule reattempt: Plans a new delivery attempt.
- **Transitions:**
 - If successful, moves to the **Delivery Confirmed** state.
 - If unsuccessful, transitions to **Termination**.
- **Delivery Confirmed:**
 - The final state, reached upon successful delivery of the passport.

4.4 Use Case Diagram

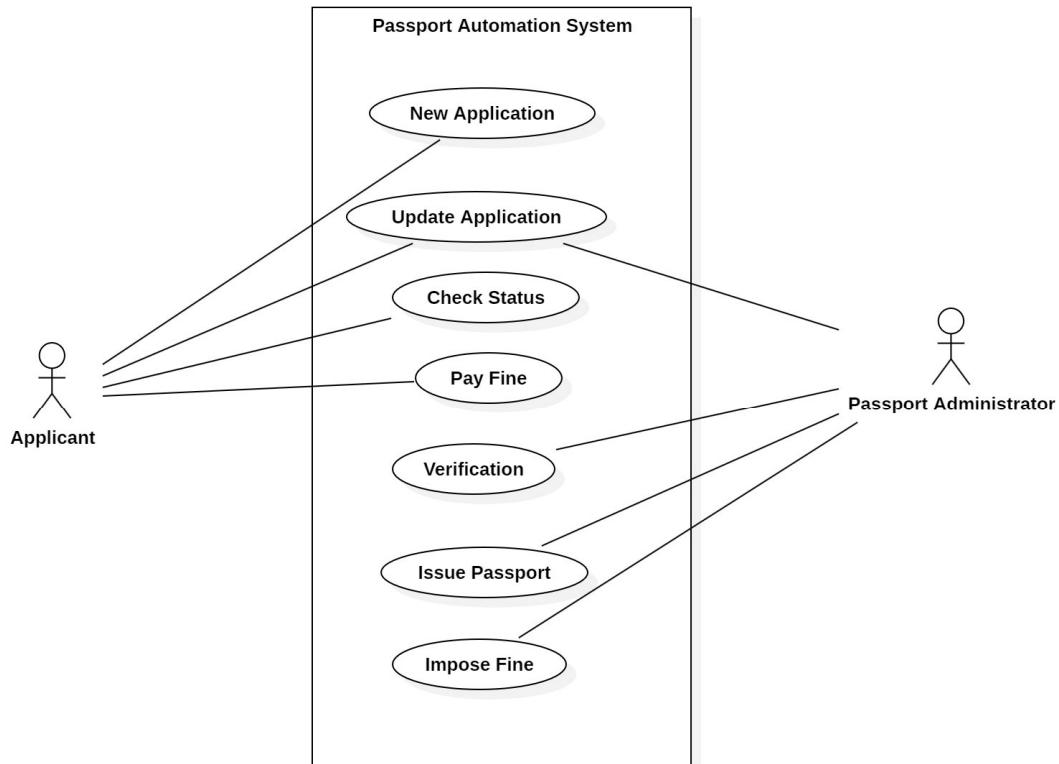


Fig 4.4.1 Use Case Diagram

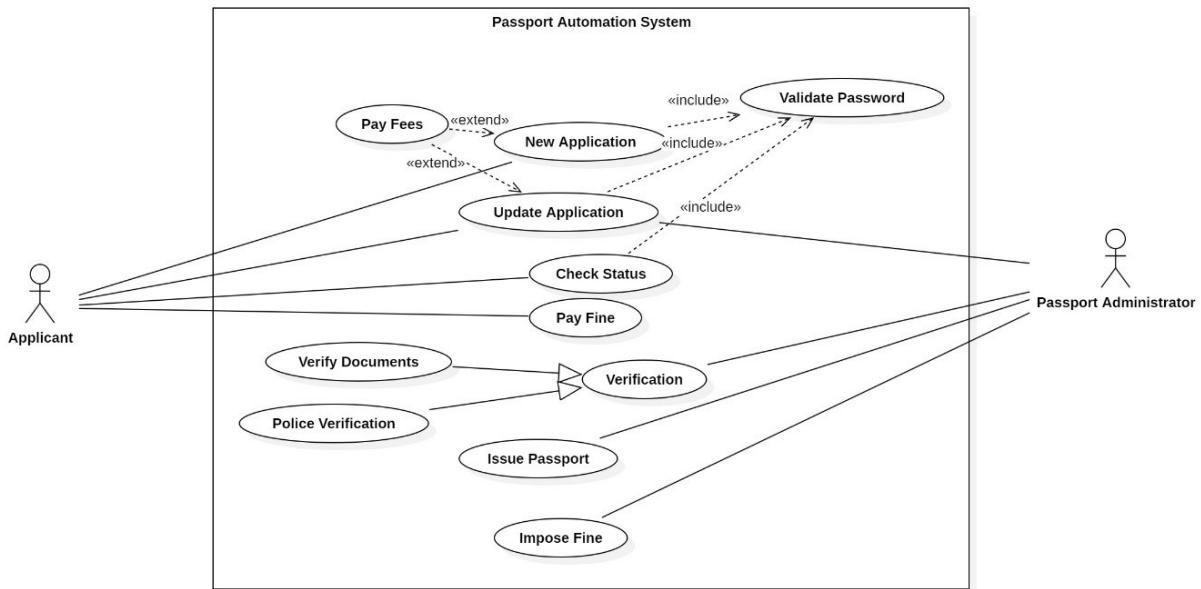


Fig 4.4.2 Advanced Use Case Diagram

Actors:

1. **Applicant:** Submits the application, tracks the status, and provides required details.
2. **Passport Authority:** Manages document verification, approval, and issuance of passports.

Use Cases:

1. **Submit Application:** The applicant submits the passport application.
 - **Includes:**
 - Validate Application Details: Ensures all required details are provided.
 - Validate Documents: Confirms that uploaded documents meet the required format.
 - **Extends:**
 - Track Application Status: Allows applicants to monitor their application's progress.
2. **Verify Documents:** The passport authority reviews submitted documents for accuracy.
 - **Includes:**
 - Validate ID Proof: Confirms the authenticity of identity documents.
 - Validate Address Proof: Verifies the applicant's address details.
 - **Extends:**
 - Request Corrections: Initiated if the submitted documents are invalid.

3. **Approve Application:** The passport authority approves valid applications.
 - **Includes:**
 - Review Application: Conducts a final review of the application.
 - Issue Approval Letter: Provides official confirmation to the applicant.
4. **Make Payment:** The applicant pays the required application fee.
 - **Includes:**
 - Validate Payment Details: Checks for accurate payment credentials.
 - **Extends:**
 - Retry Payment: Optional use case for unsuccessful transactions.
5. **Issue Passport:** The passport authority issues the passport after approval.
 - **Includes:**
 - Generate Passport: Prepares the physical or digital passport.
 - Dispatch Passport: Sends the passport to the applicant.
 - **Extends:**
 - Track Delivery: Allows applicants to monitor the delivery status.
6. **Track Application Status:** The applicant tracks the progress of their application.
 - **Includes:**
 - Check Verification Status: Provides updates on document verification.
 - Check Approval Status: Displays the final decision on the application.
7. **Request Corrections:** The applicant corrects and resubmits invalid application details.
 - **Includes:**
 - Edit Application Details: Allows updates to submitted information.
 - Reupload Documents: Facilitates resubmission of corrected documents.
 - **Extends:**
 - Submit Application: The revised details are resubmitted for review.

4.5 Sequence Diagram

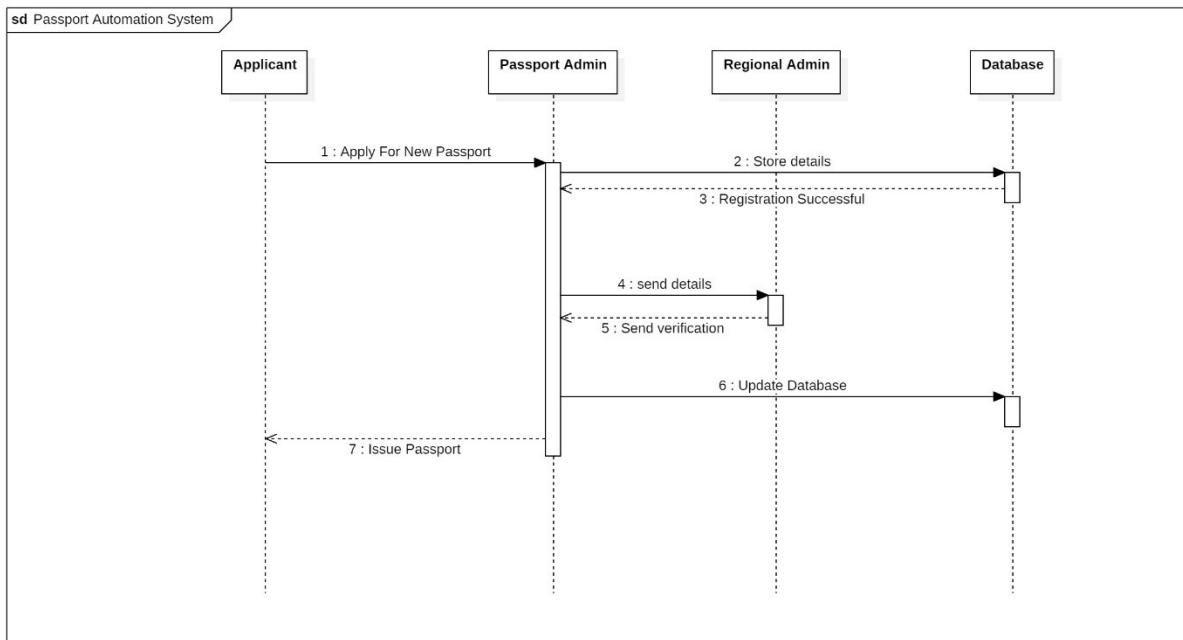


Fig 4.5.1 Sequence Diagram

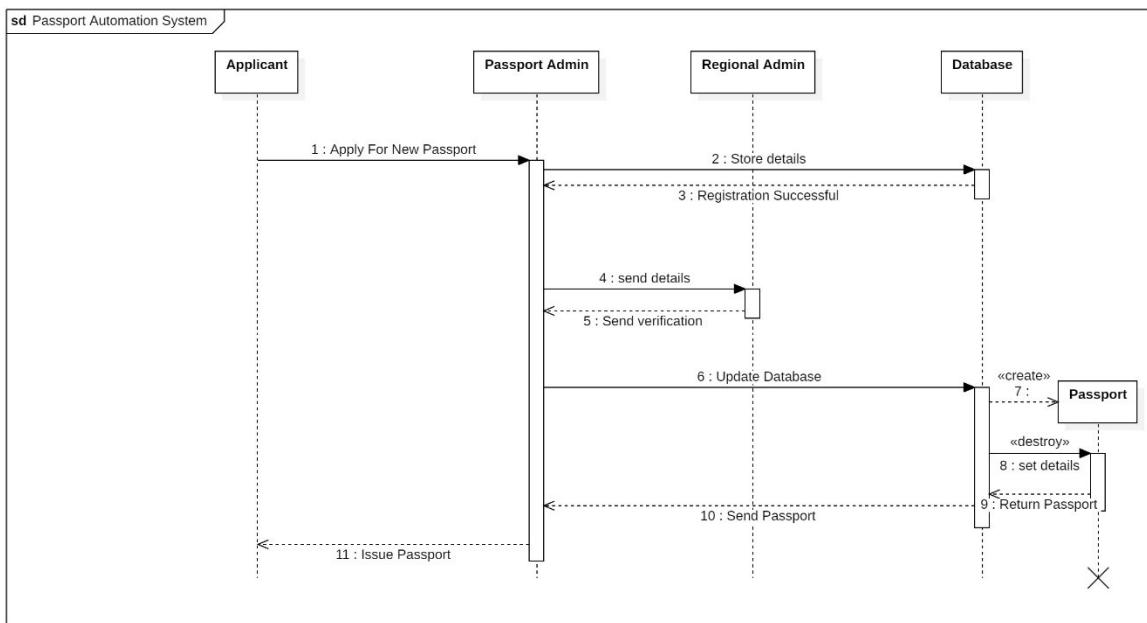


Fig 4.5.2 Advanced Sequence Diagram

This sequence diagram illustrates the flow of interactions between various components in the Passport Automation System, namely: Applicant, System, Passport Authority, and Delivery Service. Here's a detailed breakdown:

1. Submit Application (Applicant → System):

- The applicant initiates the process by submitting the passport application through the system.

2. Validate Application Details (System):

- The system validates the provided details, ensuring completeness and format compliance.
- **If validation fails:**

- The system notifies the applicant to correct and resubmit the application.

3. Upload Documents (Applicant → System):

- After successful validation, the applicant uploads the required supporting documents (e.g., ID proof, address proof).

4. Verify Documents (System → Passport Authority):

- The system forwards the application and documents to the Passport Authority for verification.
 - **If verification fails:**
- The system sends a notification to the applicant to correct or resubmit the required documents.

5. Approve Application (Passport Authority → System):

- The Passport Authority reviews and approves the verified application.
 - **If the application is rejected:**
- The system notifies the applicant with reasons for rejection and next steps.

6. Make Payment (Applicant → System):

- The applicant makes the required payment through the system.
 - **If payment fails:**
- The system allows the applicant to retry the transaction.

7. Generate Passport (System → Passport Authority):

- After successful payment, the Passport Authority generates the passport.

8. Dispatch Passport (Passport Authority → Delivery Service):

- The Passport Authority forwards the generated passport to the delivery service for dispatch.

9. Track Delivery (Applicant → System):

- The applicant tracks the delivery status of the passport using the system.

10. Passport Delivered (Delivery Service → Applicant):

- The delivery service completes the process by delivering the passport to the applicant.

Additional Details:

• Error Handling:

- If any step (e.g., application validation, document verification, or payment) fails, the system halts the process and notifies the applicant for resolution.

• Final State:

- The sequence concludes with either successful passport delivery or resolution of errors.

4.6 Activity Diagram

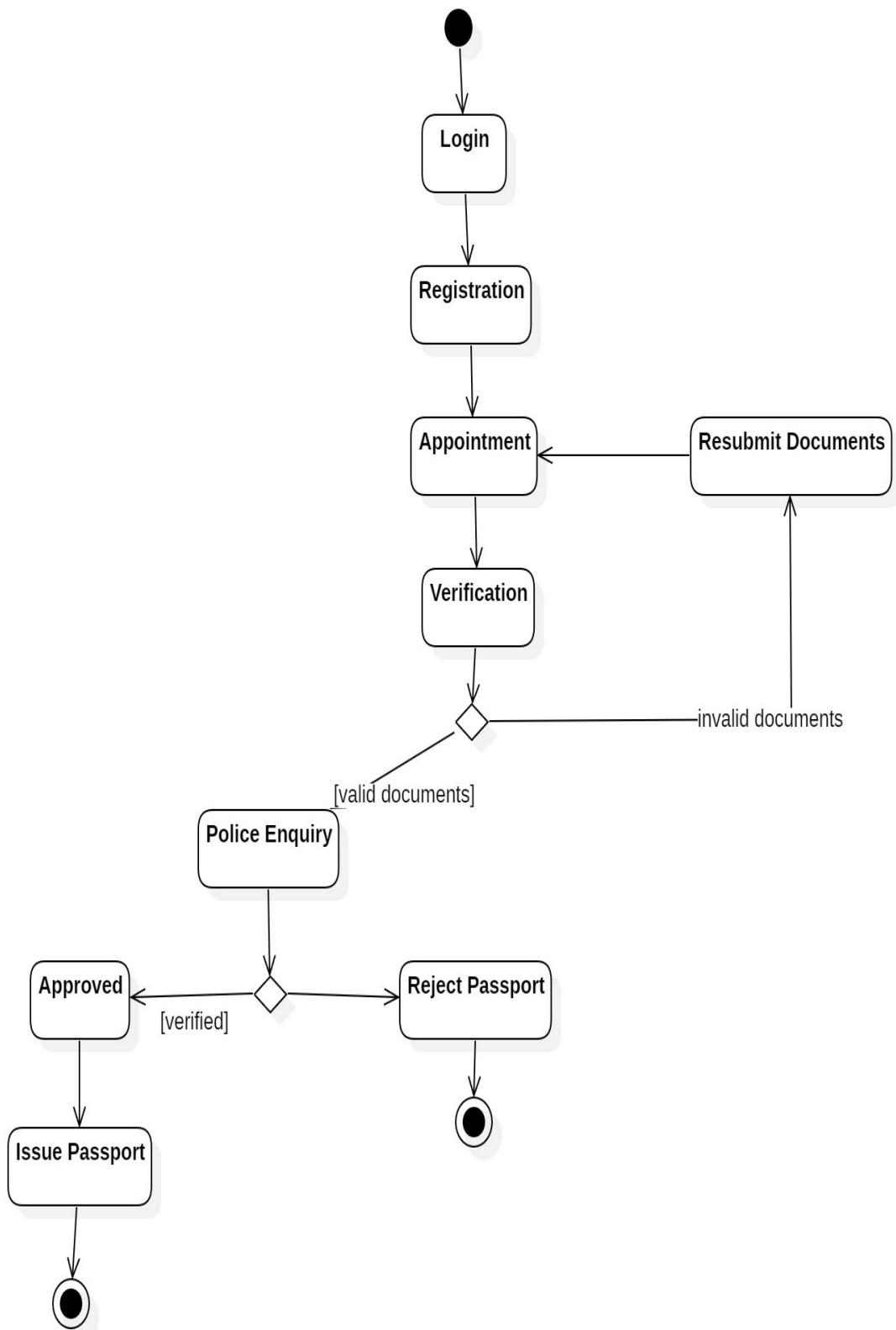


Fig 4.6.1 Activity Diagram

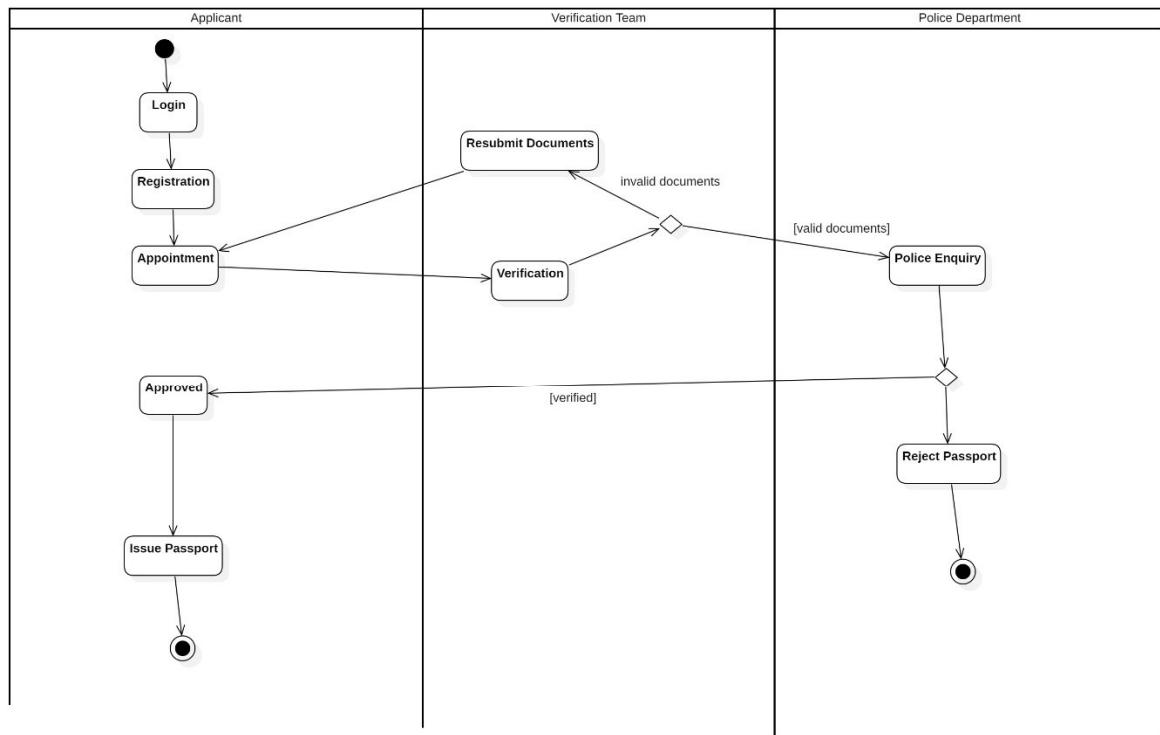


Fig 4.6.2 Advanced Activity Diagram

The activity diagram for the Passport Automation System illustrates the step-by-step workflow of passport application, processing, and delivery. Below is the detailed breakdown:

Start Point:

- The process begins with the applicant initiating the passport application.

Key Activities

1. Initiate Application:

- The applicant starts by filling out the application form online.

2. Submit Application:

- The completed application is submitted through the system.

3. Validate Application Details (System):

- The system validates the submitted details for completeness and accuracy.

◦ Decision Point:

- **If valid:** Proceed to document upload.

- **If invalid:** Notify the applicant and terminate the process until corrected.

4. Upload Documents:

- The applicant uploads required documents such as ID proof, address proof, and photographs.

5. Verify Documents (System → Passport Authority):

- The system forwards the documents to the Passport Authority for verification.

- **Decision Point:**

- **If verified:** Proceed to application approval.
- **If rejected:** Notify the applicant to resubmit correct documents.

6. Approve Application (Passport Authority):

- The Passport Authority reviews and approves the application after verification.

- **Decision Point:**

- **If approved:** Proceed to payment.
- **If rejected:** Notify the applicant with reasons for rejection.

7. Make Payment (Applicant):

- The applicant makes the payment for the passport processing fee.

- **Decision Point:**

- **If payment successful:** Proceed to passport generation.
- **If failed:** Allow the applicant to retry.

8. Generate Passport (Passport Authority):

- The Passport Authority generates the passport after payment confirmation.

9. Dispatch Passport (Passport Authority → Delivery Service):

- The generated passport is dispatched to the applicant's address through the delivery service.

10. Track Delivery (Applicant):

- The applicant can track the passport delivery status via the system.

11. Deliver Passport (Delivery Service → Applicant):

- The delivery service delivers the passport to the applicant's registered address.

Decision Points:

1. Is Application Valid?

- If valid, proceed to document upload.
- If invalid, notify the applicant to make corrections.

2. Are Documents Verified?

- If verified, proceed to application approval.
- If rejected, notify the applicant for resubmission.

3. Is Payment Successful?

- If successful, proceed to passport generation.
- If failed, allow retry.

End Point:

- The activity concludes when:
 - The passport is successfully delivered to the applicant.
 - Alternatively, the process ends with the applicant being notified of a failure or rejection at any stage.

5. Stock Management System

Problem Statement: Managing stock transactions manually in a stock trading system is inefficient, prone to human error, and can lead to delays, incorrect transactions, or missed opportunities. A well-organized system is required to monitor stock availability, validate user actions, track transactions, and ensure seamless buying and selling of stocks to meet user demands efficiently.

5.1 SRS-Software Requirements Specification

Stock Maintenance System (SMS)	
METRO	Page: _____ Date: _____
<u>Introduction:</u>	
→ Purpose :-	The purpose of the SMS is to streamline & automate the process of tracking & managing inventory levels, orders, & sales in real-time.
→ Scope:-	The system will be used by retail stores, warehouses & e-commerce platforms to maintain stock levels & track products from suppliers to consumers. It will key functions such as stock monitoring, order processing, & report generation.
→ Overview:-	SMS will be web or desktop-based application that allow users to: → Add, remove & update inventory data. → Track stock levels in real-time. → Create reports for stock analysis, monitoring, & sale. → Handle stock transfer b/w different locations.
→ General Description:-	<ul style="list-style-type: none">Users:- Store manager, warehouse operator & Inventory clerk.Inventory management, Order processing, Reporting.Technology stack:- Frontend MERN stack (MongoDB, Angular) for web-based & Java for desktop-based systems.Security:- Role-based access control (Admin, Manager).
→ Functional Requirements:-	<ul style="list-style-type: none">Inventory management:- Ability to add, edit & delete items from stock.Stock monitoring:- Real-time update on stock availability & alert when inventory levels are low.User Roles:- Different roles with specific access levels (Admin, Manager, Clerk).

Fig 5.1.1

→ Interface Requirements:-

- UI :- A user-friendly, responsive web-based interface with simple navigation for non-technical users.
- Admin Interface :- More detailed, with functionalities for stock adjustments, report generation, & supplier management.
- API :- RESTful API's for integrating the stock system with 3rd party platforms such as e-commerce or accounting S/W.

→ Performance Requirements:-

- Response Time :- The system should respond within 2 seconds for inventory queries & order generation.
- Availability :- The system should have 99.9% uptime to ensure availability for stock management.

→ Design Constraints:-

- Scalability :- The system should be scalable to accommodate additional users & stock items as the business grows.
- Data Storage :- Relational or NoSQL database should be able to handle large datasets & fast retrieval.

→ Non-Functional Attributes:-

- Security :- Use encryption for sensitive data like supplier details & financial records.
- Reliability :- System must be reliable with minimal downtime.
- Usability :- The system should be intuitive & easy to use, requiring minimal training.

Fig 5.1.2

→ Preliminary Schedule:-

- Planning & Requirements Gathering: 2 weeks
- Design & Prototyping: 3 weeks
- Development: 6-8 weeks
- Testing & QA: 3 weeks
- Deployment: 1 week
- Total Estimated Duration: 15-17 weeks.

Budget:-

→ Development Team Costs: \$50,000

→ HW costs: \$5000

→ SW ~~soft~~ Licenses: \$2000

→ Maintenance Costs: \$10,000

Total Initial Budget Estimate: \$67000

Fig 5.1.3

5.2 Class Diagram

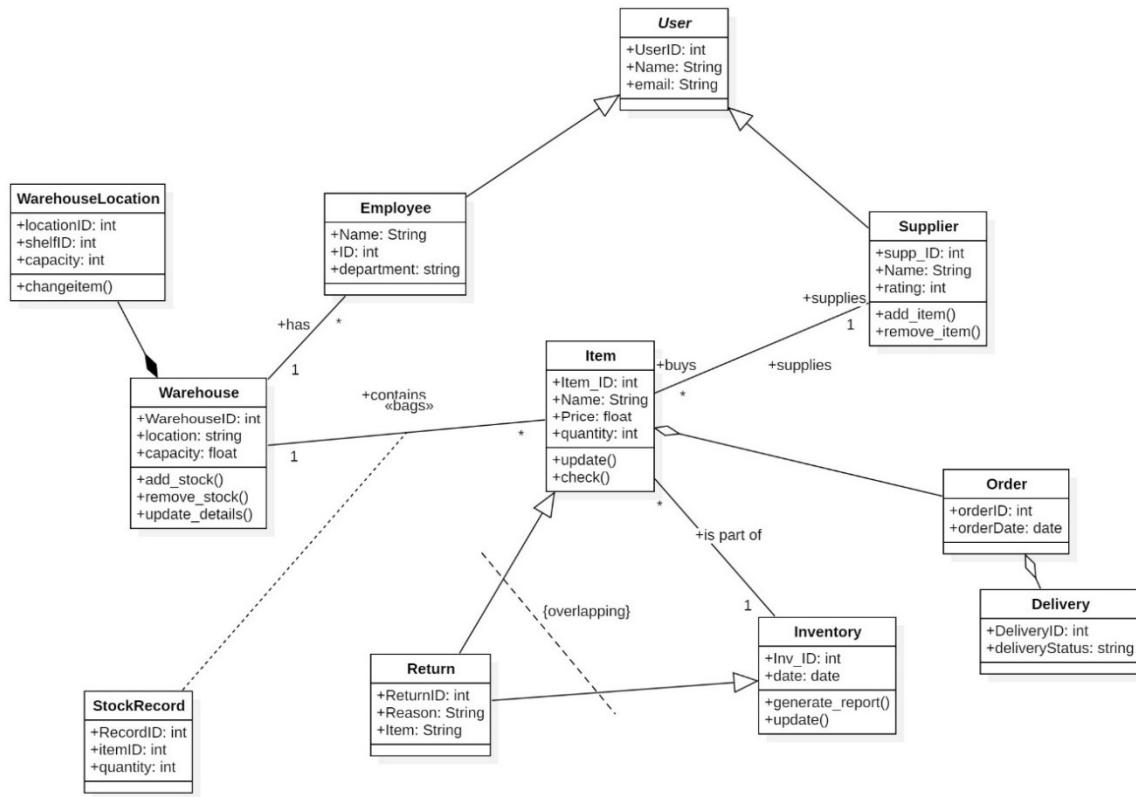


Fig 5.2.1

Classes and Their Roles:

1. Warehouse:

- **Attributes:**

- WarehouseID: Unique identifier for a warehouse.
- location: Address of the warehouse.
- capacity: Total storage capacity of the warehouse.

- **Operations:**

- add_stock(): Adds stock to the warehouse.
- remove_stock(): Removes stock from the warehouse.
- update_details(): Updates warehouse information.

2. WarehouseLocation:

- **Attributes:**

- locationID: Unique identifier for the location.
- shelfID: Shelf identifier within the warehouse.

- capacity: Storage capacity of the shelf.

- **Operations:**

- changeitem(): Changes or relocates an item in the shelf.

3. Employee:

- **Attributes:**

- Name: Name of the employee.
- ID: Employee ID.
- department: Department of the employee.

- Relationship: Employees are associated with warehouses.

4. Item:

- **Attributes:**

- Item_ID: Unique identifier for an item.
- Name: Name of the item.
- Price: Price of the item.
- Quantity: Number of items available.

- **Operations:**

- update(): Updates item details.
- check(): Checks item availability.

5. Supplier:

- **Attributes:**

- supp_ID: Unique supplier ID.
- Name: Supplier's name.
- rating: Supplier's rating.

- **Operations:**

- add_item(): Adds items supplied by the supplier.
- remove_item(): Removes items supplied by the supplier.

- Relationship: Suppliers supply items to the system.

6. User:

- **Attributes:**

- UserID: Unique user identifier.

- Name: Name of the user.
- email: Email of the user.
- Relationship: Users purchase items.

7. Order:

- **Attributes:**
 - orderID: Unique order identifier.
 - orderDate: Date of the order.

8. Delivery:

- **Attributes:**
 - DeliveryID: Unique delivery identifier.
 - deliveryStatus: Status of the delivery.

9. Inventory:

- **Attributes:**
 - Inv_ID: Inventory ID.
 - date: Date of inventory record.
- **Operations:**
 - generate_report(): Generates inventory reports.
 - update(): Updates inventory details.

10. Return:

- **Attributes:**
 - ReturnID: Identifier for returned items.
 - Reason: Reason for the return.
 - Item: Returned item information.

11. StockRecord:

- **Attributes:**
 - RecordID: Record identifier.
 - itemID: ID of the item in stock.
 - quantity: Quantity of the item in stock.

Relationships:

1. Warehouse ↔ WarehouseLocation:

- **1-to-1 Relationship:** Each warehouse has a specific location.
2. **Warehouse ↔ Employee:**
- **1-to-Many Relationship:** A warehouse is managed by multiple employees.
3. **Warehouse ↔ Item:**
- **Aggregation Relationship:** A warehouse contains multiple items.
4. **Item ↔ Supplier:**
- **Many-to-Many Relationship:** Multiple suppliers supply multiple items.
5. **User ↔ Item:**
- **Many-to-Many Relationship:** Users purchase multiple items.
6. **Order ↔ Item:**
- **Association:** An order consists of multiple items.
7. **Order ↔ Delivery:**
- **1-to-1 Relationship:** Each order corresponds to a delivery.
8. **Inventory ↔ Item:**
- **Association:** Inventory maintains records of items.
9. **Return ↔ Item:**
- **Association:** A return record is associated with items.
10. **StockRecord ↔ Warehouse:**
- **Association:** A warehouse maintains stock records.

5.3 State Machine Model

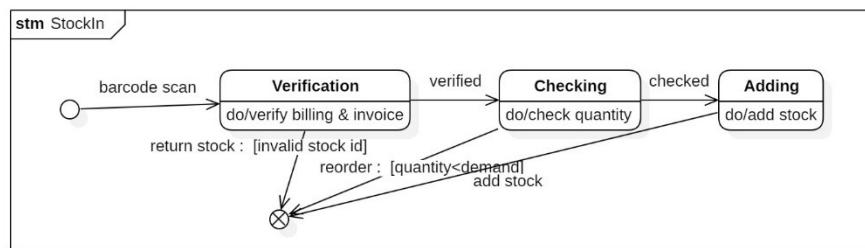
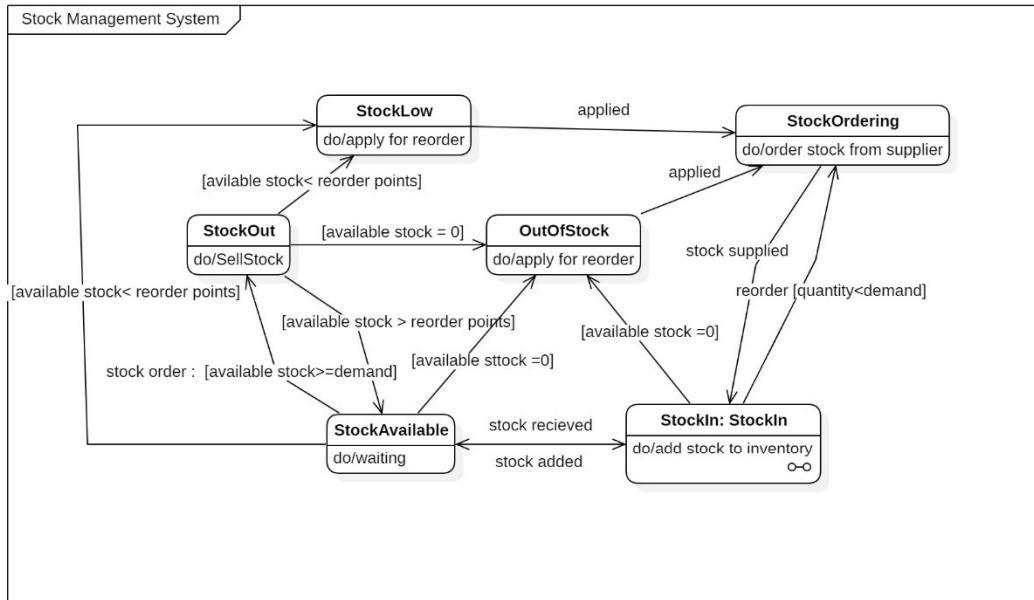
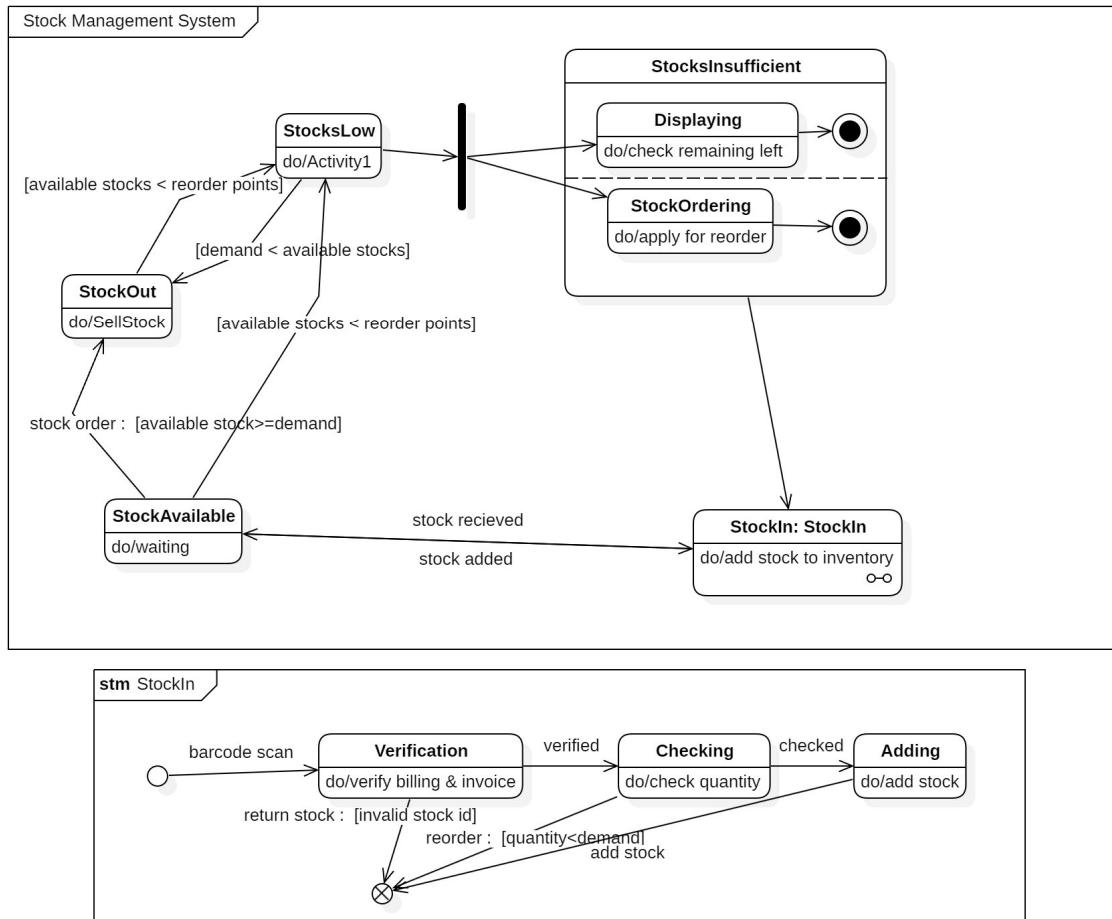


Fig 5.3



.1

State Diagram

Fig 5.3.2 Advanced State Diagram

Main Components:

1. States and Sub-States:

- **StocksLow**: Triggered when available stocks fall below the reorder point.
 - Activity: **do/Activity1** indicates the system performs a task when in this state.
- **StockOut**: Occurs when demand exceeds available stock.
 - Activity: **do/SellStock** handles stock selling.
- **StockAvailable**: Represents sufficient stock availability.
 - Activity: **do/waiting** indicates waiting for further events.
- **StocksInsufficient (Composite State)**: Contains two sub-states:
 - **Displaying**: Checks remaining stock (**do/check remaining left**).

- **StockOrdering:** Applies for a reorder (do/apply for reorder).

2. Transitions:

- From **StocksLow**:
 - To **StockOut**: When demand < available stocks.
 - To **StockAvailable**: After stock is received.
- From **StockOut**:
 - To **StocksLow**: If available stocks < reorder points.
 - To **StockAvailable**: If available stock \geq demand.
- From **StocksInsufficient**:
 - To **StockIn**: When stock is added to inventory.

3. StockIn (Sub-State Machine):

- **Verification**: Verifies the stock by scanning barcodes and checking invoices.
 - Transition:
 - To **Checking**: After successful verification.
 - To **Start**: If the stock ID is invalid or for returns.
- **Checking**: Confirms stock quantity (do/check quantity).
 - Transition: To **Adding** after quantity is checked.
- **Adding**: Adds verified stock to the inventory (do/add stock).

Key Processes:

1. Low Stock Handling:

- The system identifies low stocks and transitions to **StockOut** if demand exceeds available stock.
- Reorders are initiated via the **StocksInsufficient** composite state.

2. Stock Replenishment:

- When stock arrives, the system transitions to **StockIn** for verification, quantity checking, and adding stock.

3. Stock Availability:

- Upon successful replenishment, the system returns to **StockAvailable**, indicating normal stock levels.

5.4 Use Case Diagram

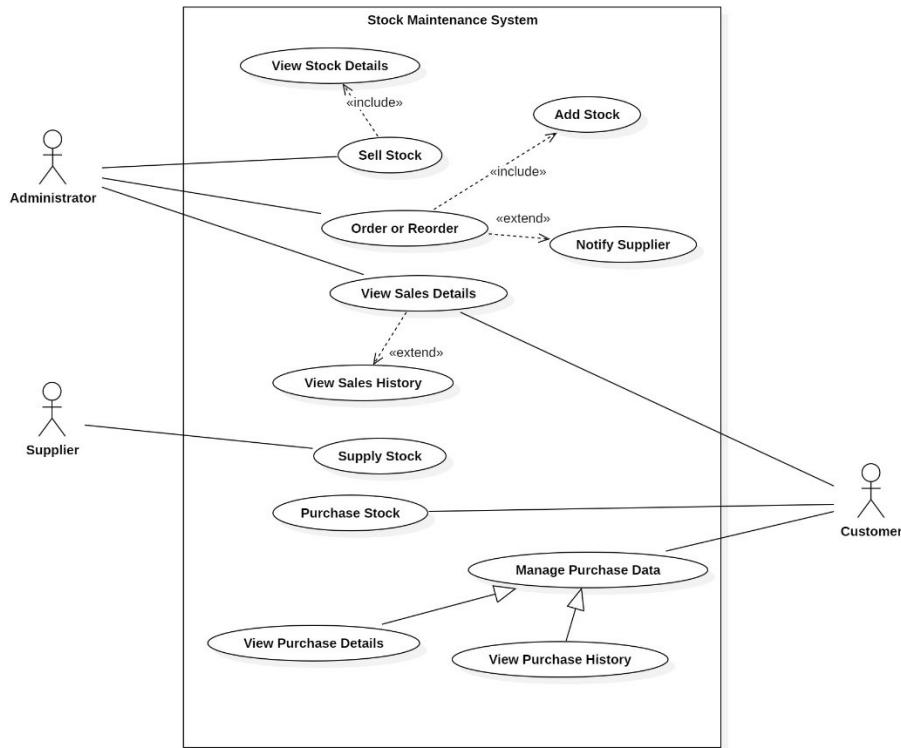


Fig 5.4.1 Use Case Diagram

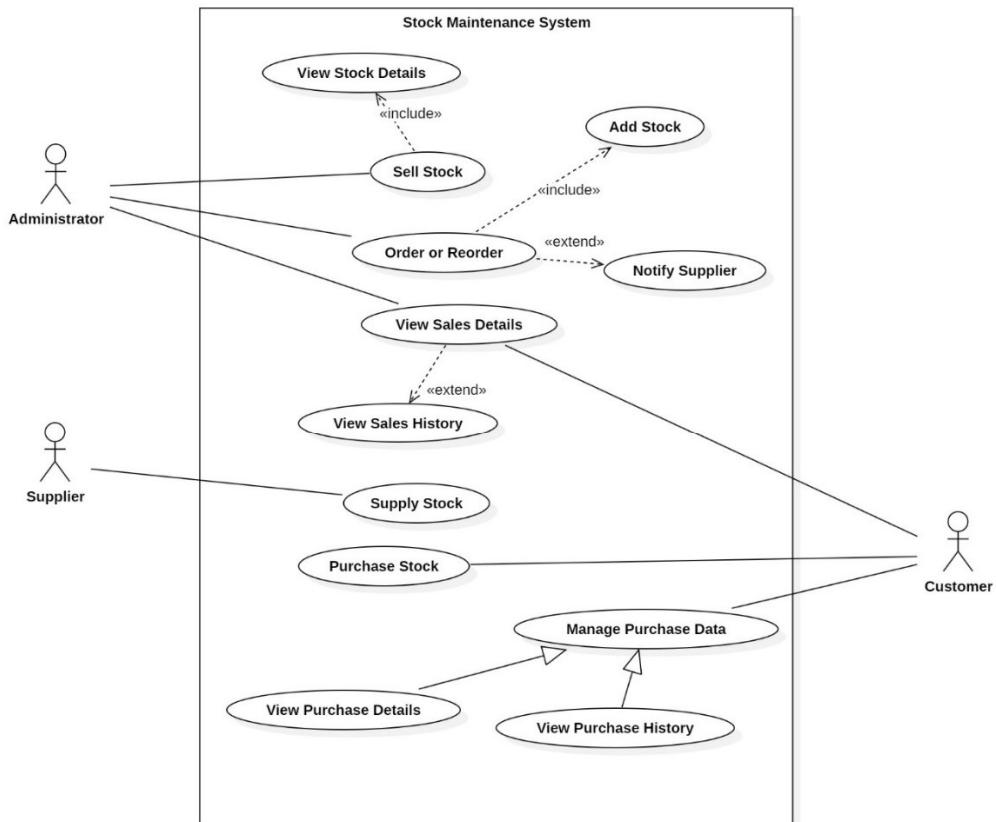


Fig 5.4.2 Advanced Use Case Diagram

Actors:

1. Administrator:

- Manages stock operations, orders, and sales details.

2. Supplier:

- Supplies stock to the system and purchases items.

3. Customer:

- Manages purchases and views purchase-related information.

Use Cases:

1. View Stock Details:

- Allows administrators to check stock availability.
- **Includes:** Used by other use cases like "Sell Stock" and "Add Stock" to ensure accurate stock information.

2. Sell Stock:

- Administrator sells stock based on customer demand.
- **Includes:** "View Stock Details" ensures updated stock availability before selling.

3. Order or Reorder:

- Administrator orders or reorders stock when levels are low.
- **Extends:** "Notify Supplier" triggers supplier notifications during stock reordering.

4. View Sales Details:

- Administrator views details of sales.
- **Extends:** "View Sales History" shows historical sales records for analysis.

5. Add Stock:

- Administrator adds new or replenished stock to the system.

6. Supply Stock:

- Supplier provides stock to the system, fulfilling orders or reorders.

7. Purchase Stock:

- Supplier purchases stock for their own needs, interacting with the system as a customer.

8. Manage Purchase Data:

- Customers track their purchases and manage purchase-related information.
- **Includes:**
 - "View Purchase Details" for specific transaction data.
 - "View Purchase History" for a broader view of past transactions.

Relationships:

1. Include:

- Ensures common functionality is reused across use cases, e.g., "View Stock Details" is shared between "Sell Stock" and "Add Stock."

2. Extend:

- Adds optional behavior, e.g., "View Sales History" extends "View Sales Details" for historical data, and "Notify Supplier" extends "Order or Reorder."

5.5 Sequence Diagram

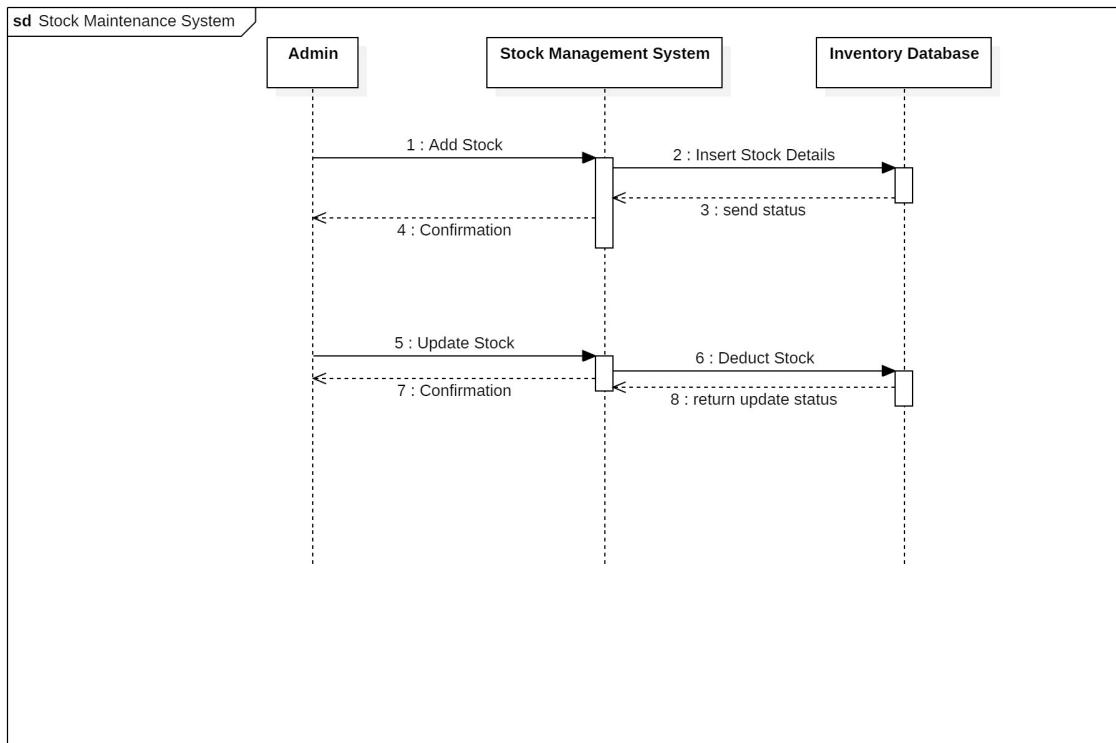


Fig 5.5.1 Sequence Diagram

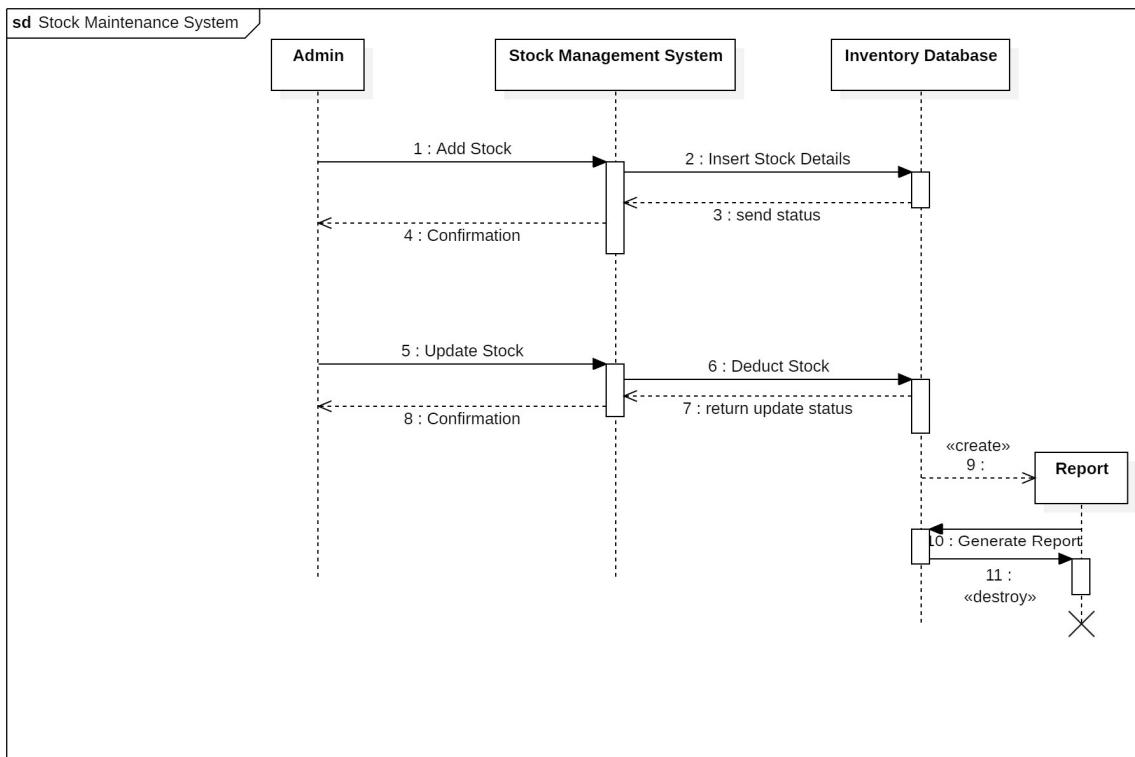


Fig 5.5.2 Advanced Sequence Diagram

Components:

1. **Admin:** Initiates actions like adding or updating stock.
2. **Stock Management System:** Manages requests and coordinates between the Admin and the Inventory Database.
3. **Inventory Database:** Stores and updates stock details.
4. **Report:** Handles the generation of reports.

Sequence Steps:

1. **Add Stock (Step 1 - 4):**
 - **Step 1:** The Admin sends a request to add stock to the Stock Management System.
 - **Step 2:** The Stock Management System processes the request and inserts the stock details into the Inventory Database.
 - **Step 3:** The Inventory Database sends a status update (e.g., success or failure) back to the Stock Management System.
 - **Step 4:** The Stock Management System confirms the addition of stock to the Admin.
2. **Update Stock (Step 5 - 8):**
 - **Step 5:** The Admin requests an update to the stock details via the Stock Management System.
 - **Step 6:** The Stock Management System deducts or adjusts stock quantities in the Inventory Database.
 - **Step 7:** The Inventory Database returns an update status (e.g., successful update) to the Stock Management System.
 - **Step 8:** The Stock Management System confirms the update to the Admin.
3. **Generate Report (Step 9 - 11):**
 - **Step 9:** The Admin initiates a request to generate a report, and the system creates the Report instance.
 - **Step 10:** The Report is generated with the relevant stock details and updates.

- **Step 11:** Once the report is generated, the Report instance is destroyed.

Notable Features:

- **Actors:**
 - **Admin** is the primary user interacting with the system.
 - Other components like the Inventory Database and Report act as background services.
- **Control Flow:** The flow moves sequentially between components, ensuring confirmation at every step.
- **Lifecycle of Report:** The Report object is created and destroyed dynamically, representing temporary use.

5.6 Activity Diagram

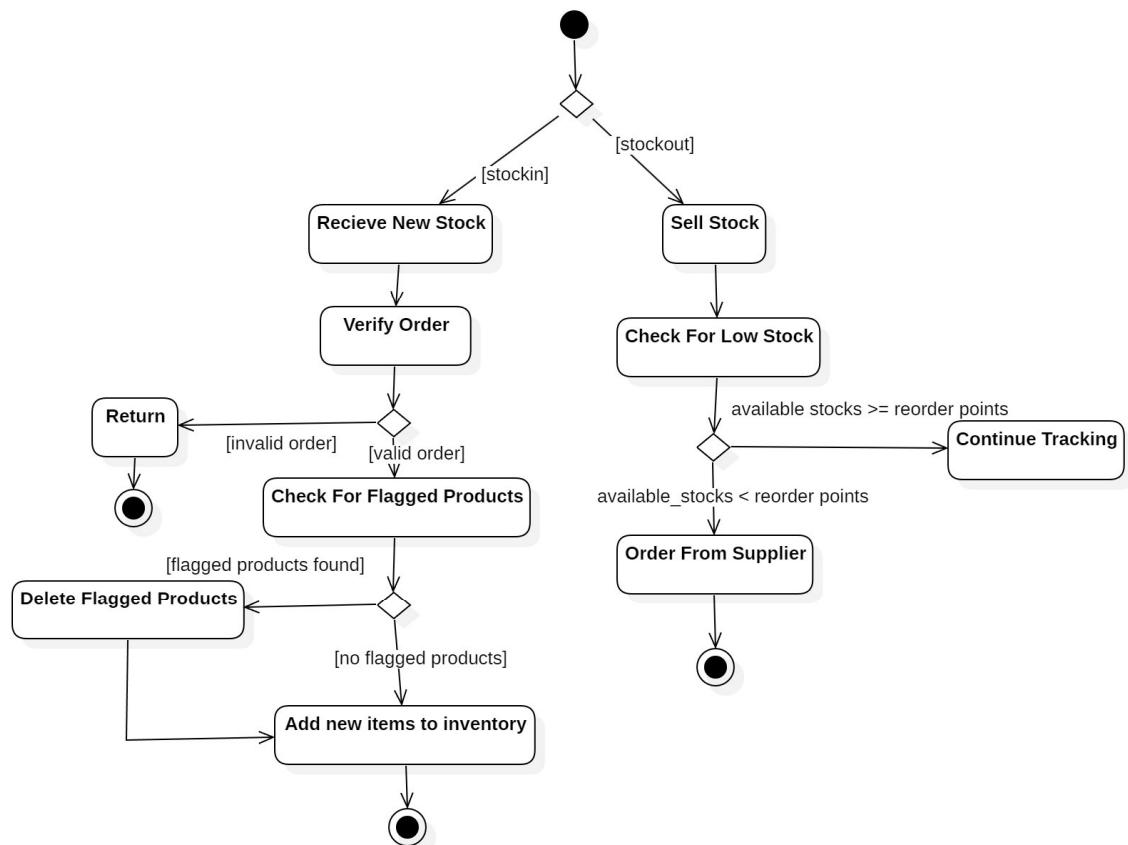


Fig 5.6.1 Activity Diagram

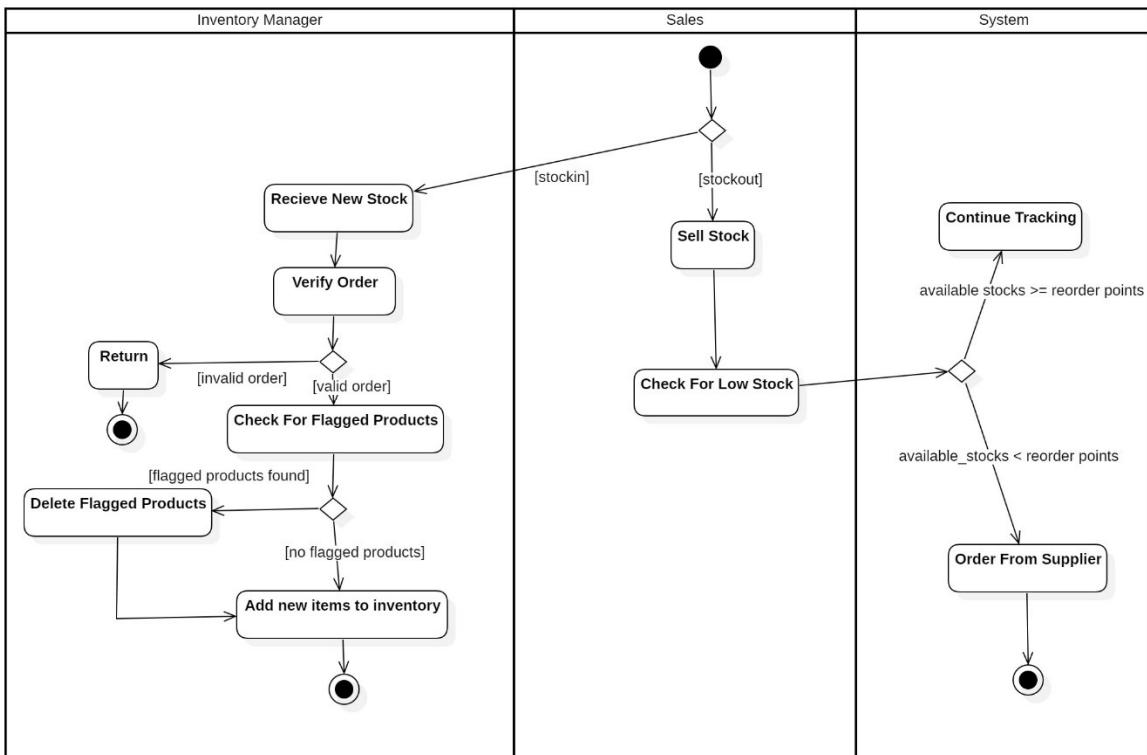


Fig 5.6.2 Advanced Activity Diagram

1. Inventory Manager Actions

- **Receive New Stock:**
 - The process begins with the Inventory Manager receiving new stock (indicated by the incoming stock flow).
- **Verify Order:**
 - The Inventory Manager verifies whether the received order is valid.
 - If the order is invalid, the process moves to the **Return** step, ending the workflow for invalid stock.
 - If the order is valid, the workflow proceeds.
- **Check for Flagged Products:**
 - The system checks the inventory for flagged products (e.g., defective, expired, or recalled items).
 - If flagged products are found:

- **Delete Flagged Products:** These flagged items are removed from inventory.
 - If no flagged products are found:
 - **Add New Items to Inventory:** The new stock is successfully added to the inventory.

2. Sales Actions

- **Sell Stock:**
 - This action represents sales transactions where stock is sold to customers.
- **Check for Low Stock:**
 - After selling stock, the system evaluates whether the stock levels are low (below the reorder point).

3. System Actions

- **Continue Tracking:**
 - If stock levels are sufficient (greater than or equal to reorder points), the system continues monitoring inventory without taking further action.
- **Order from Supplier:**
 - If stock levels are below the reorder point, the system triggers an order from the supplier to replenish stock.

Notable Features

1. **Decision Points:**
 - The diamonds in the diagram represent decision points, ensuring appropriate actions based on conditions (e.g., valid/invalid order, flagged products found, stock levels).
2. **Workflow Division:**
 - The actions are divided into three swimlanes, indicating responsibilities:
 - **Inventory Manager:** Handles physical stock and inventory verification.
 - **Sales:** Manages selling stock and triggering stock evaluations.

- **System:** Automates stock tracking and supplier orders.

3. End Points:

- The black circles indicate the end of respective workflows:
 - Inventory updates end with flagged product handling or adding items.
 - System processes end after ordering from suppliers.