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

Lab Record

Software Engineering and Object-Oriented Modeling

Submitted in partial fulfillment for the 5th Semester Laboratory

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

Aditya Singh

1BM22CS022

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 Analysis and Design(23CS5PCOOM) laboratory has been carried out by Aditya Singh (1BM22CS022) during the 5th Semester Oct24-Jan2025.

Signature of the Faculty Incharge:

NAME OF THE FACULTY:Dr Sandhya.A.Kulkarni

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

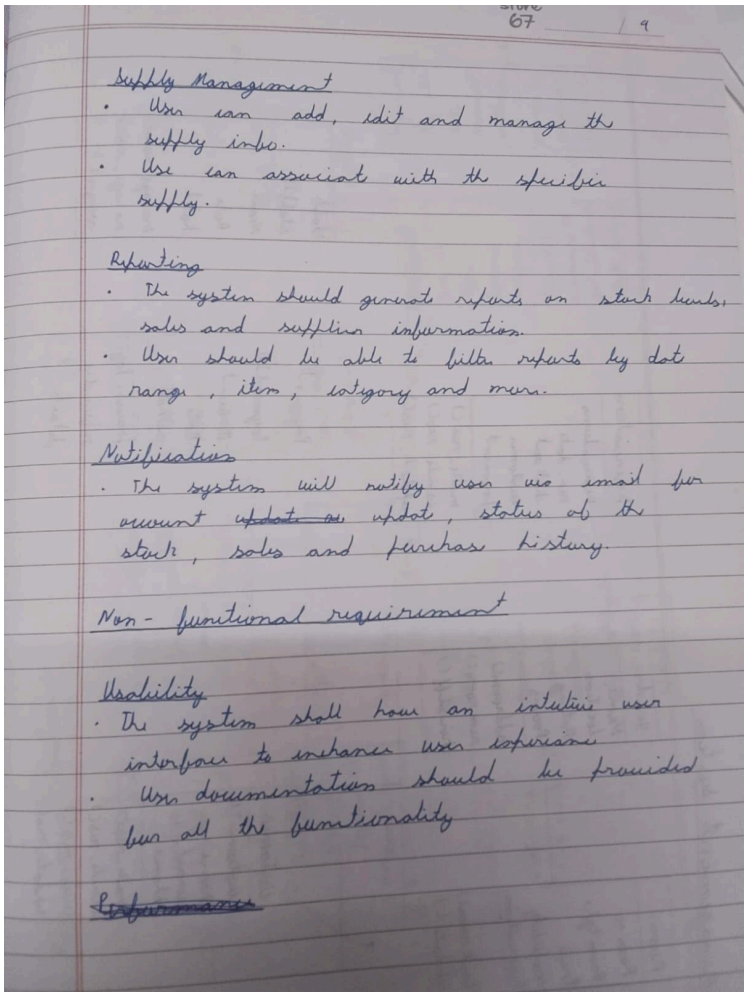
Table of Contents

Github:- https://github.com/Singh12Aditya/OOM__LAB

1. Hotel Management System

1.1 Problem Statement: The current manual processes in hotel management are time-consuming, error-prone, and inefficient. Managing room reservations, guest check-ins and check-outs, billing, and housekeeping coordination requires significant manual effort, often leading to errors and reduced customer satisfaction. There is a need for an automated, user-friendly system that can handle these tasks seamlessly, enhance operational efficiency, and improve guest experience.

1.2 SRS:



Supply Management
- User can add, edit and manage the supply info.
- Use can associate with the specific supply.

Reporting
- The system should generate reports on stock levels, sales and supplies information.
- User should be able to filter reports by date range, item, category and more.

Notifications
- The system will notify user via email for account ~~update~~ update, status of the stock, sales and purchase history.

Non-functional requirement

Usability
- The system shall have an intuitive user interface to enhance user experience.
- User documentation should be provided for all the functionality

~~Performance~~

for conformation should be given.

4) Check in and check out
- The hotel staff should be able to track to has checked in and who has checked out.
- They should be able to keep a record of check in time and date.

5) Payment
- The user should be able to make payment to the selected rooms and services securly.
- The hotel management should be able to verify the transaction status once the booking is done.

6) Maintenance Management
- The hotel staff should be able to track the things which needs to be repaired or changed.
- Once the changes are made, it should be reflected in the portal.

7) Staff Requirements / Management
- The admin should be able to maintain the staff details, their attendance and updat the changes.

8) Review System
- The users should be able to give rating the different services of the hotel.

store
67 / 3

Non functional requirements

1) Performance
   - User experiance should be smooth and secured.

2) Security
   - transactions and user authantification should be secure and data should not be misused.

3) Scalability
   - If more branches are open, we can modify to system to ~~read~~ ~~user~~ met user need.

4) Reliability
   - It website should provide 24×7 support system with authomated chat bots and customer support.

Domain Requirement

- Room inventory management
- Booking rules and policies
- Revenue management
- Integration with third-person platforms and services
- Fund and function management.

1.3 Class Diagram:

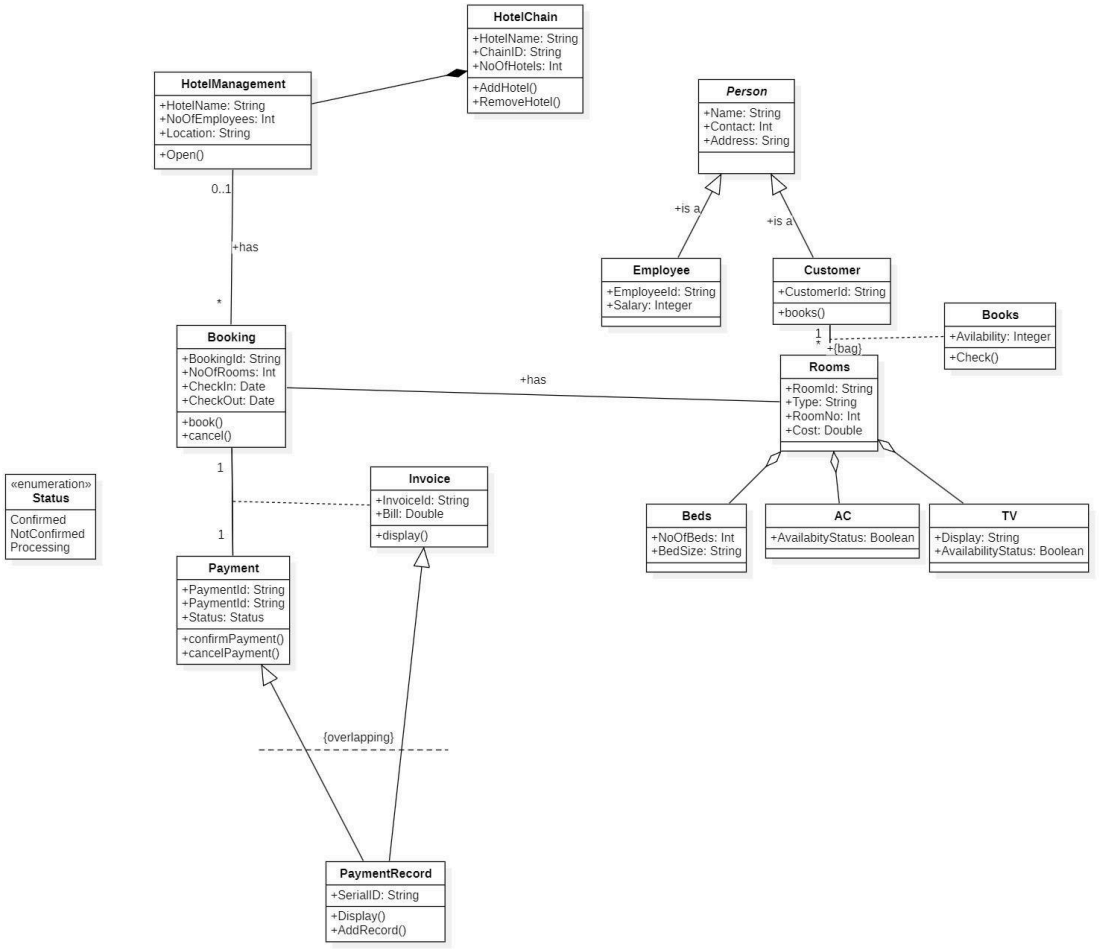


Fig 1.3.1: Class Diagram

Description:

The class diagram illustrates the structure of the system, highlighting key entities and their relationships.

- **Classes:**
    - **Booking Class**: Manages details of reservations such as booking ID, dates, room type, and status.
    - **PersonAccount Class**: Abstract class serving as a base for employees and customers.
        - **Customer Class**: Inherits from PersonAccount and includes attributes like customer ID, name, and contact details.
        - **Employee Class**: Inherits from PersonAccount and includes attributes like employee ID, role, and shift details.
    - **Room Class**: Represents the rooms in the hotel, with attributes such as room number, type, availability status, and price.
    - **Payment Class**: Manages payment details, including payment ID, method, status, and amount.
- **Relationships:**
    - The **Booking Class** is associated with both **Room Class** and **Customer Class**, signifying that customers make bookings for rooms.
    - The **Payment Class** is related to the **Booking Class** to process transactions for bookings.
    - **Employee Class** is involved in managing bookings and customer queries.

1.4 State Diagram:

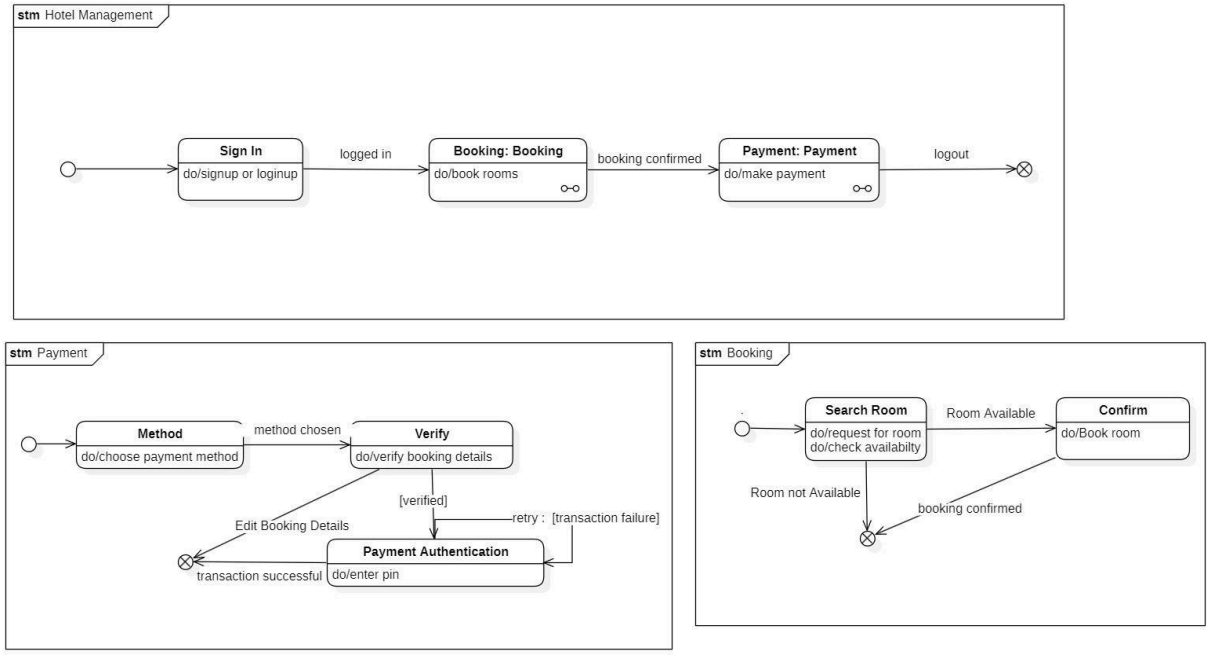


Fig 1.4.1: State Model Diagram

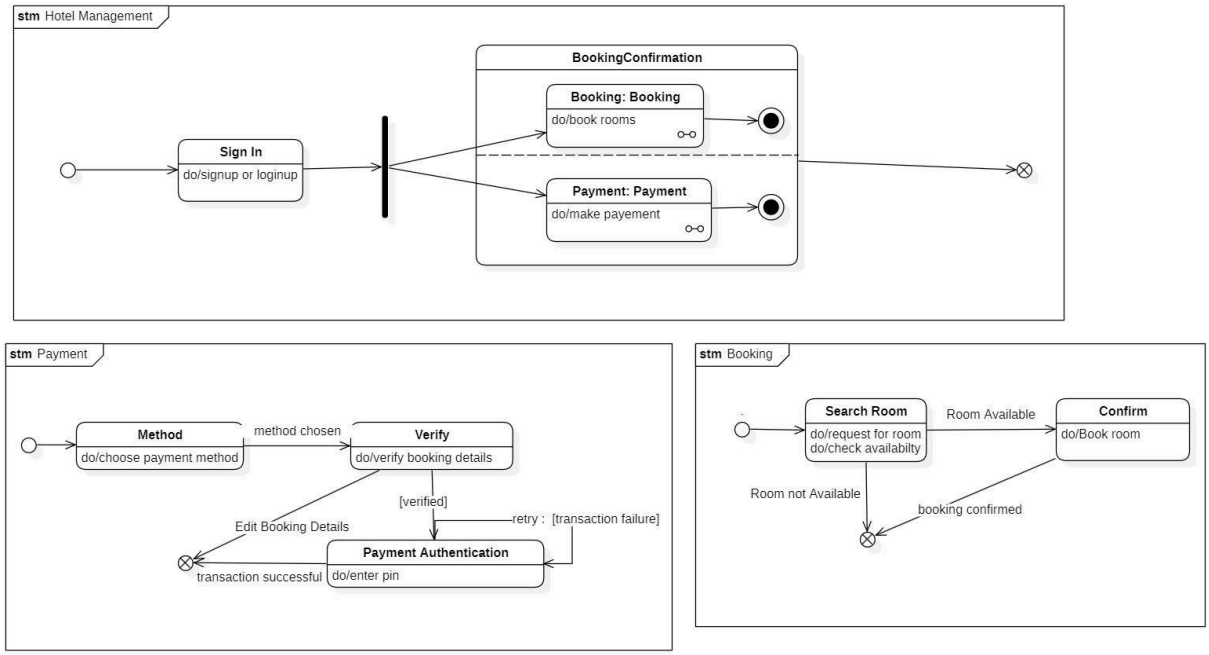


Fig 1.4.2: Advanced State Model Diagram

Description:

The state diagram shows the various states and transitions involved in the hotel management system.

- **States:**
    - **Sign-In State**: Represents when a user logs into the system (either customer or employee).
    - **Booking State**: Manages the reservation process, including selecting room type, dates, and confirming availability. Sub-states include:
        - Room Selection
        - Availability Check
    - **Payment State**: Handles payment processing for bookings. Sub-states include:
        - Payment Method Selection
        - Payment Confirmation
    - **Concurrent States:**
        - Booking and Payment processes are concurrent, allowing simultaneous management of reservations and transactions.

1.5 Use Case Diagram:

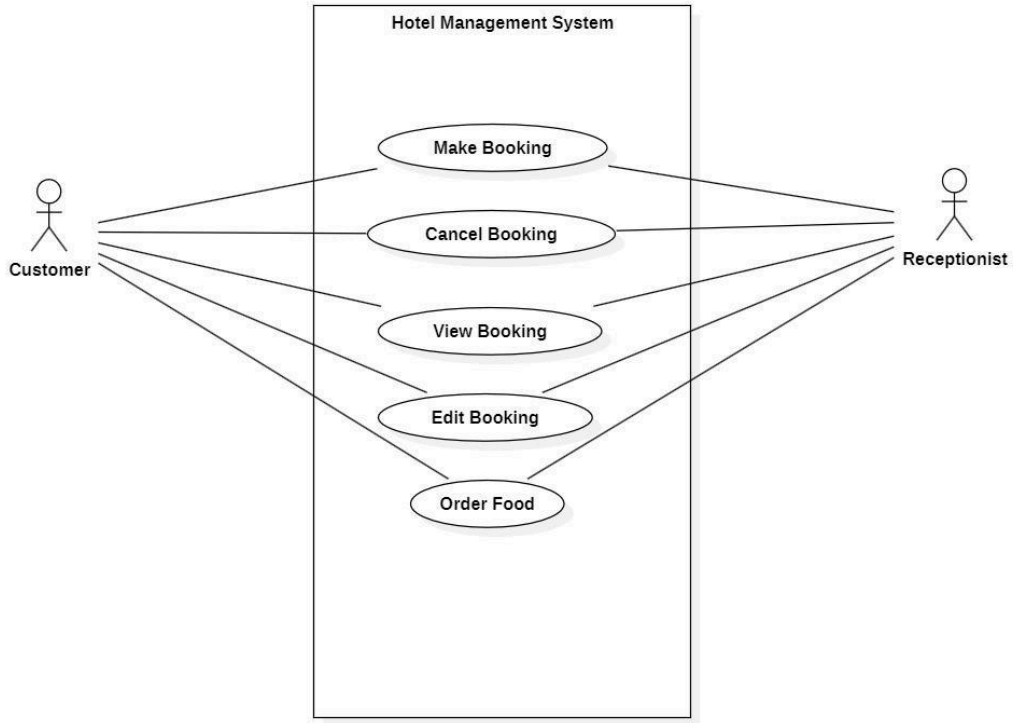


Fig 1.5.1: Use Case Diagram

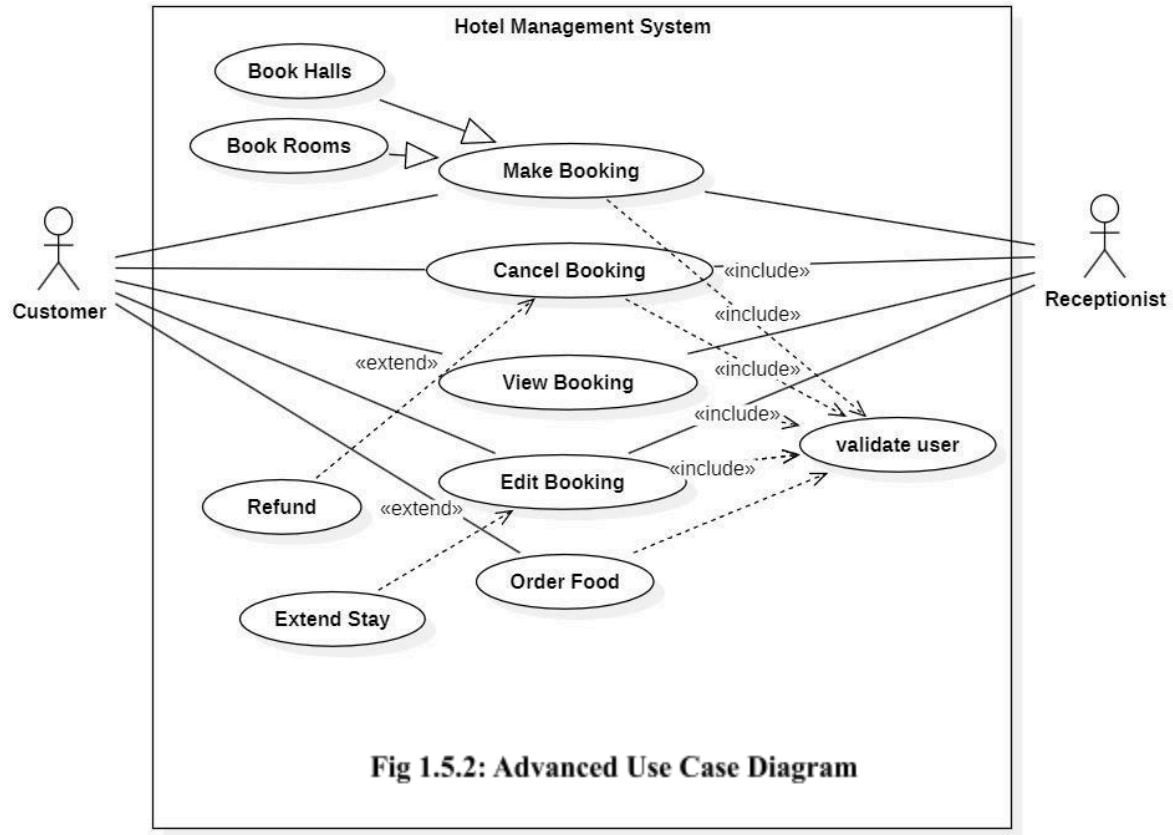


Fig 1.5.2: Advanced Use Case Diagram

Description:

The use case diagram defines interactions between actors and system functionalities.

- **Actors:**
    - **Customer**: Can make a booking, cancel a booking, view existing bookings, edit bookings, and order food.
    - **Receptionist**: Can assist customers by managing bookings, updating information, and processing food orders.
- **Use Cases:**
    - **Make Booking**: Allows a customer to reserve a room.
    - **Cancel Booking**: Enables cancellation of an existing booking.
    - **View Booking**: Provides booking details to the customer or receptionist.
    - **Edit Booking**: Allows modifications to an existing booking.
    - **Order Food**: Lets the customer request room service or meals during their stay.

1.6 Sequence Diagram:

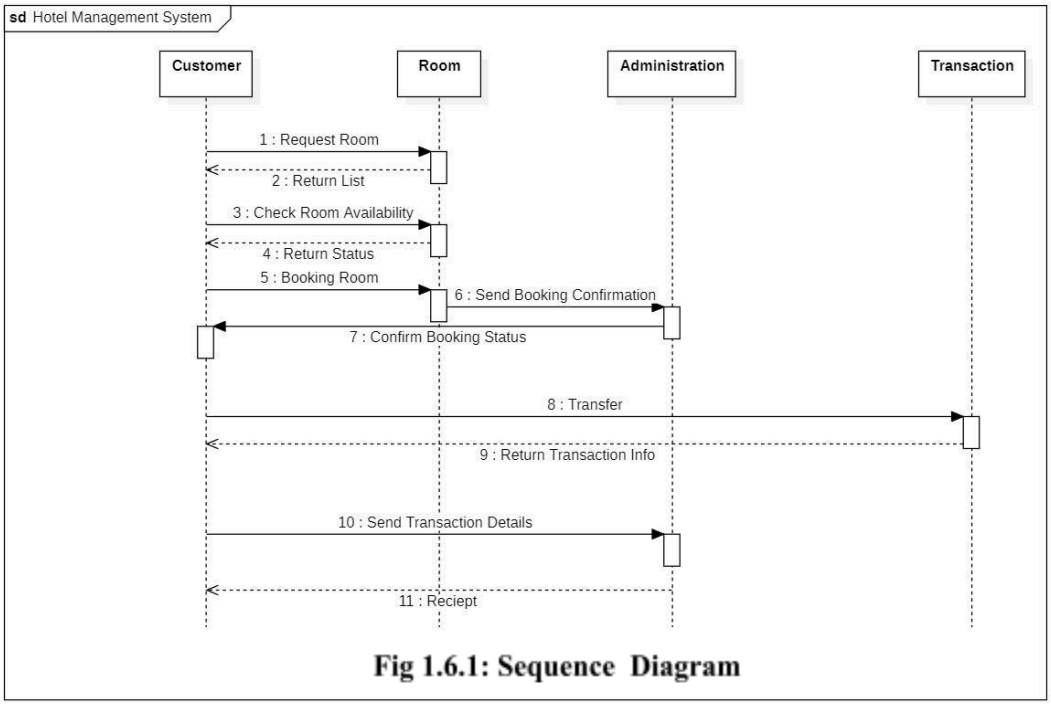


Fig 1.6.1: Sequence Diagram

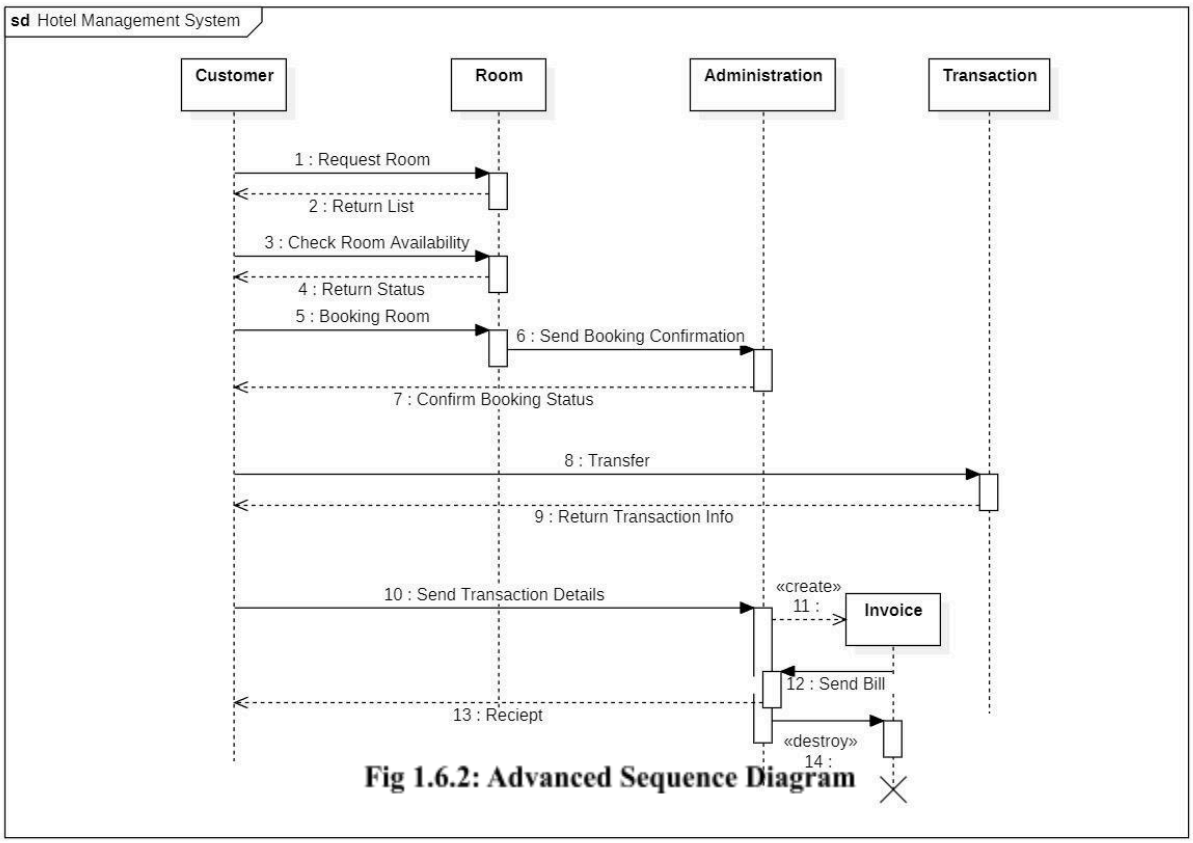


Fig 1.6.2: Advanced Sequence Diagram

Description:

The sequence diagram details the interactions between objects to perform a booking operation.

- **Objects:**
    - **Customer**: Initiates the booking process.
    - **Room**: Provides room availability and details.
    - **Administration**: Validates the booking and room availability.
    - **Transaction**: Processes the payment for the booking.
    - **Invoice (Transient Object)**: Created after payment confirmation, representing the receipt of the transaction.
- **Flow:**
    0. Customer requests a room booking.
    1. System checks room availability via the Room object.
    2. Administration validates the details.
    3. Payment is processed via the Transaction object.
    4. Invoice is generated and sent to the Customer.

1.7 Activity Diagram:

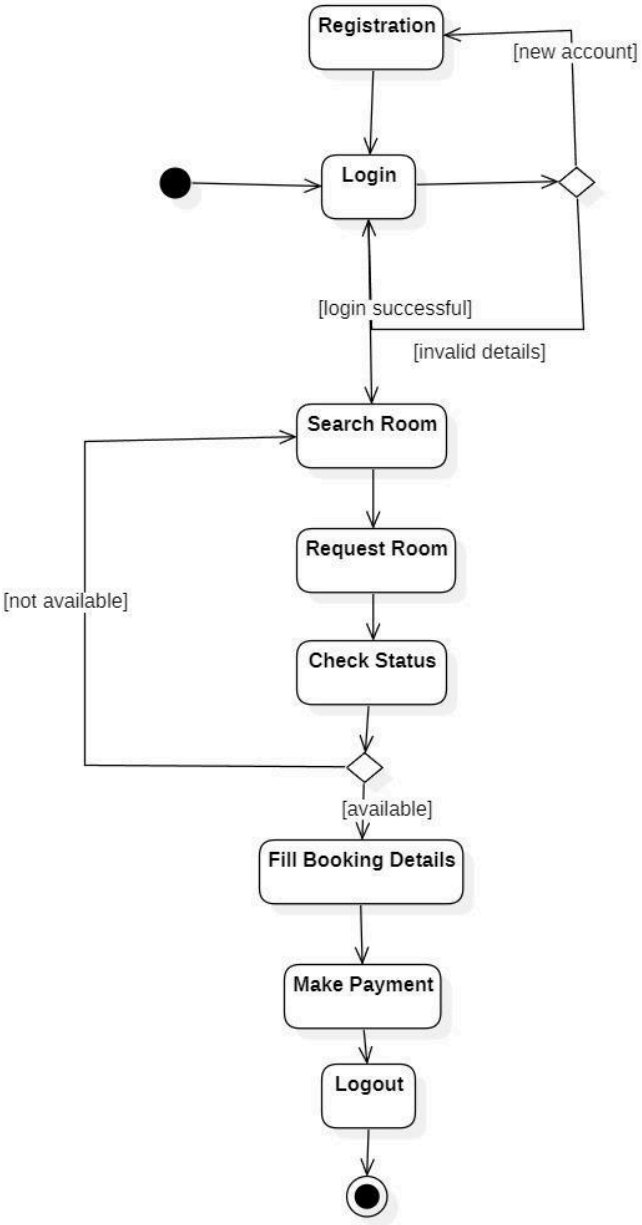


Fig 1.7.1 : Activity Diagram

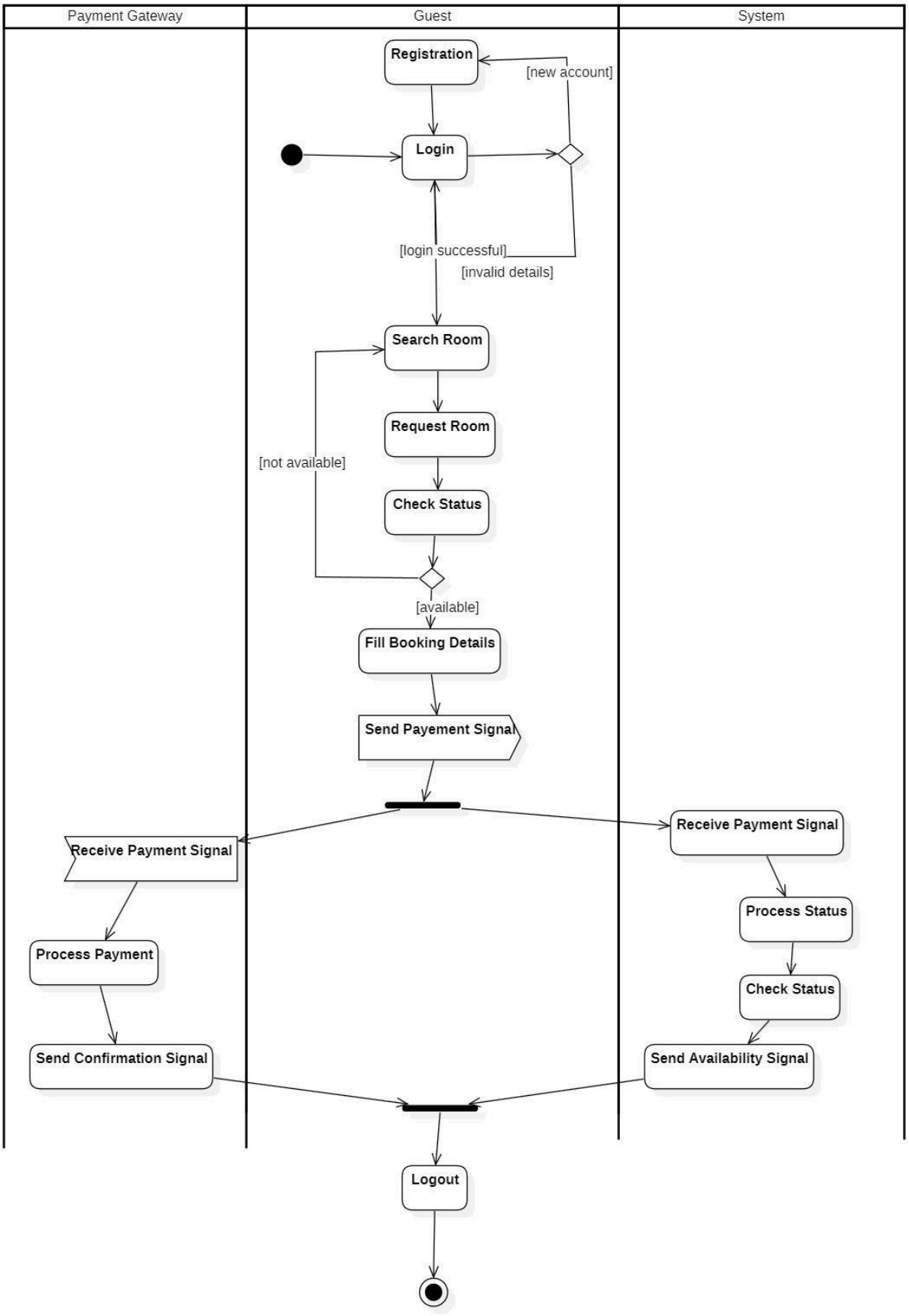


Fig 1.7.2 : Advanced Activity Diagram

Description:

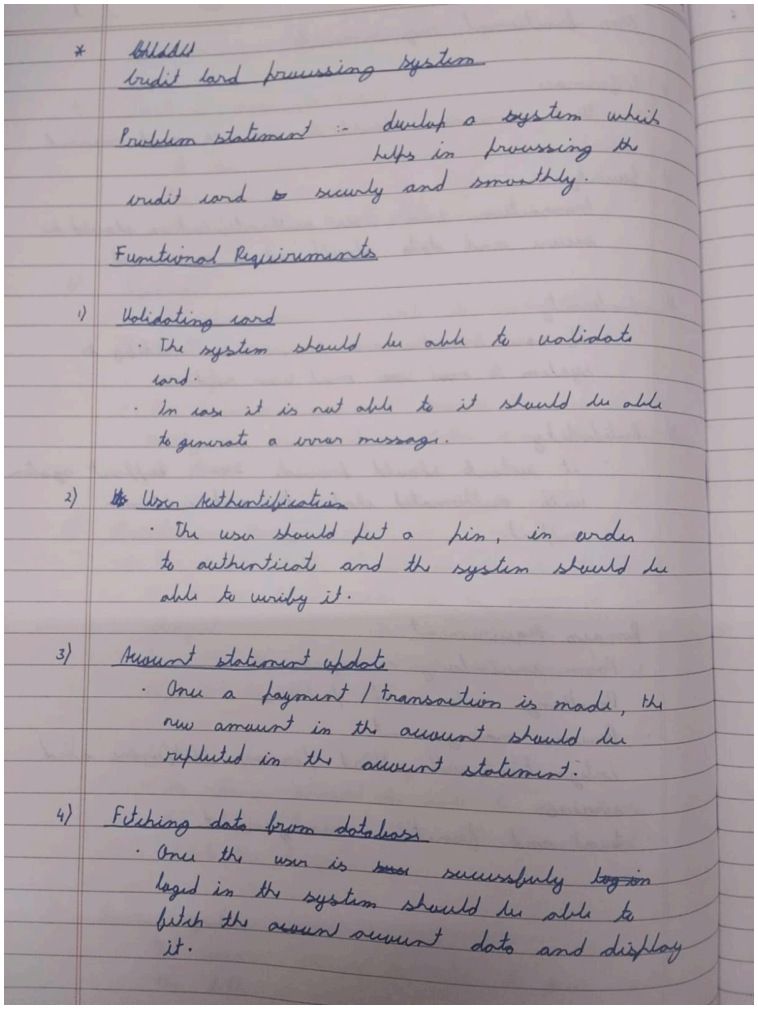
The activity diagram represents the flow of activities for making a payment, divided into swimlanes for clarity.

- **Swimlanes:**
    - **Payment Gateway**: Processes the payment details, including authentication and transaction completion.
    - **Guest**: Performs actions like selecting the room, confirming booking details, and initiating payment.
    - **System**: Validates input, updates booking records, and generates invoices.
- **Flow:**
    0. Guest selects room and initiates payment.
    1. System validates the booking and redirects to the Payment Gateway.
    2. Payment Gateway processes the payment and confirms success.
    3. System updates the booking status and generates an invoice for the guest.

2. Credit Card Processing System

2.1 Problem Statement: With the increasing volume of credit card transactions, financial institutions face challenges in ensuring secure, accurate, and real-time processing of payments. Current systems often struggle with fraud detection, transaction delays, and compliance with industry regulations. A robust, scalable, and secure system is required to process credit card payments efficiently while providing real-time fraud monitoring and adhering to regulatory standards.

2.2 SRS:



* ~~bhhddd~~
credit card processing system

Problem statement :- develop a system which helps in processing the credit card securly and smoothly.

Functional Requirements

1) Validating card
- The system should be able to validate card.
- In case it is not able to it should be able to generate a error message.

2) ~~bb~~ User Authentification
- The user should put a pin, in order to authenticate and the system should be able to verify it.

3) Account statement update
- Once a payment / transaction is made, the new amount in the account should be reflected in the account statement.

4) Fetching data from database
- Once the user is ~~xxx~~ successfully ~~login~~ loged in the system should be able to fetch the account account data and display it.

store
67 / 5

Non functional Requirement

1) Fast and secure authentification
2) Fast and secure transaction
3) The interface should be minimal and user friendly for smooth use.
4) Scalibility
5) Performance

Domain Requirement

- Multiple credit cards can use used.
- The software can be used on multiple devices.
- IT requirements
- Multiple language support

2.3 Class Diagram:

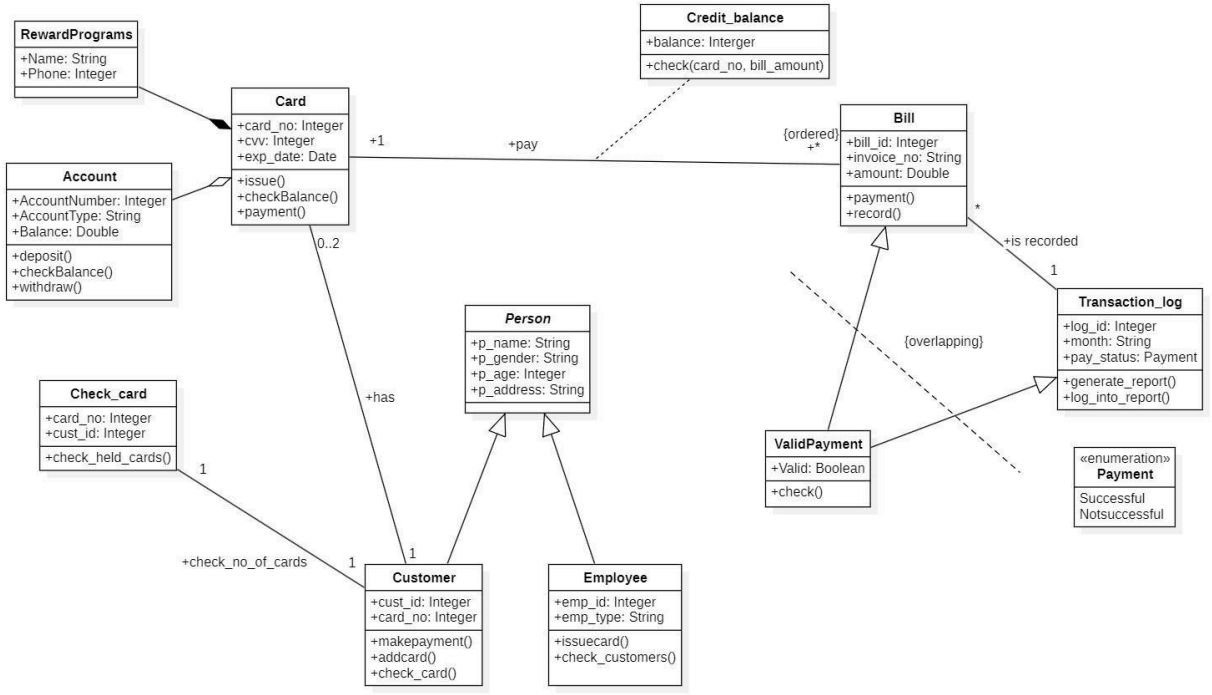


Fig 2.3.1: Class Diagram

Description:

The class diagram represents the static structure of the credit card processing system, highlighting the main entities and their relationships.

- **Classes:**
    - **Card Class**: Represents a credit card with attributes like card number, expiration date, CVV, and status (active/inactive).
    - **Account Class**: Represents the customer's account associated with the credit card, with attributes like account ID, balance, credit limit, and holder details.
    - **Bill Class**: Manages billing details, including billing ID, due date, amount due, and payment status.
    - **Transaction Class**: Tracks individual transactions with attributes like transaction ID, timestamp, amount, merchant details, and status (approved/declined).
- **Relationships:**

- **Card Class** is associated with the **Account Class** since each card is tied to a specific account.
- **Transaction Class** is linked to both **Card Class** and **Account Class**, representing purchases made with the card and their effect on the account.
- **Bill Class** is associated with the **Account Class** to manage periodic payments and outstanding balances.

2.4 State Diagram

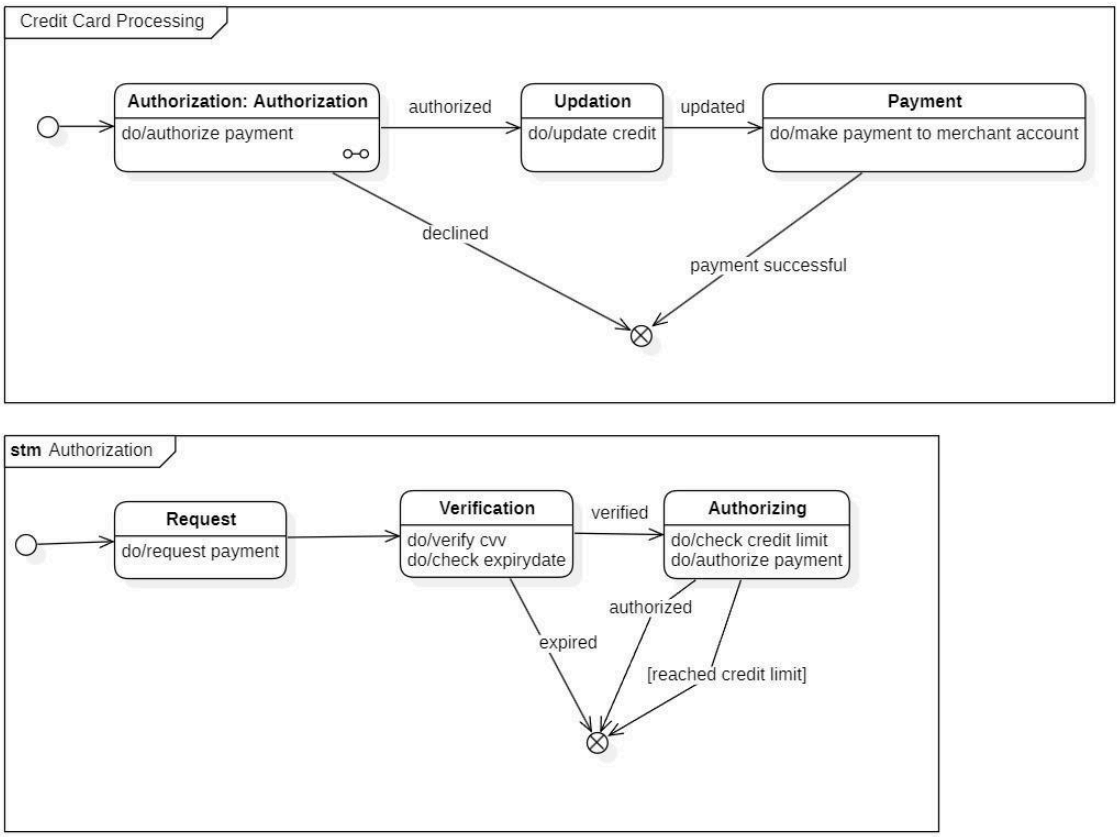


Fig 2.4.1: State Diagram

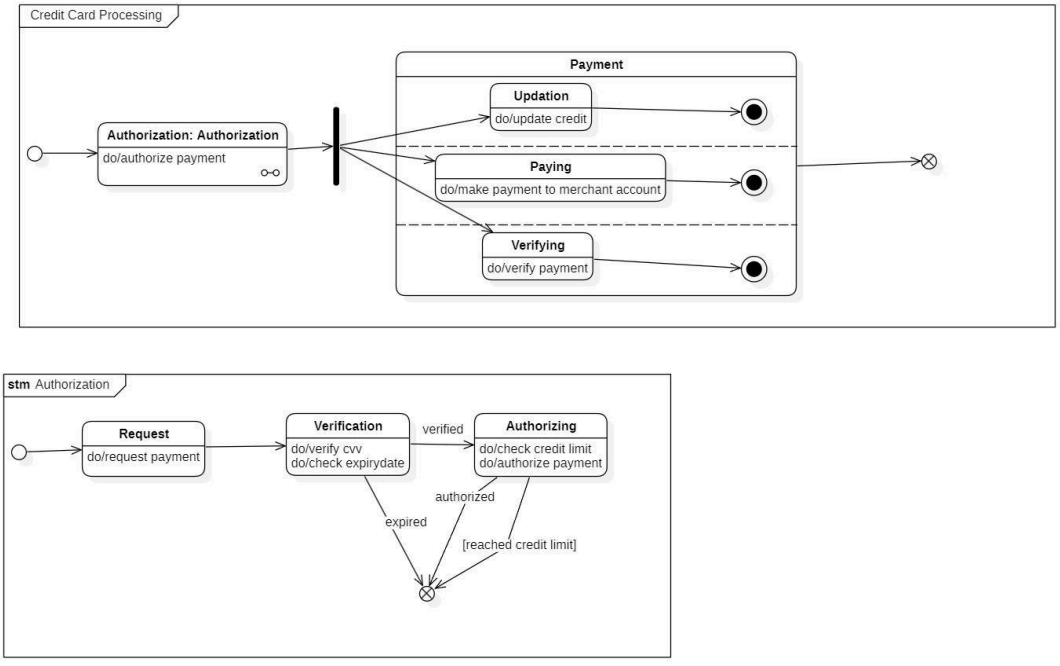


Fig 2.4.2: Advanced State Diagram

Description:

The state diagram showcases the dynamic behavior of the system, focusing on the transitions between different states.

- **States:**
    - **Authorization State**: Validates card details, checks account status, and approves or declines the transaction.
        - Sub-states:
            - Card Validation
            - Fraud Detection
            - Approval/Rejection
    - **Payment State**: Handles the payment process, involving concurrent sub-states:
        - **Updation of Credit**: Updates the available credit after a successful transaction.
        - **Paying**: Deducts the amount from the account and marks the transaction as paid.
        - **Verifying**: Confirms that the payment is correctly processed and logged.

2.5 Use Case Diagram:

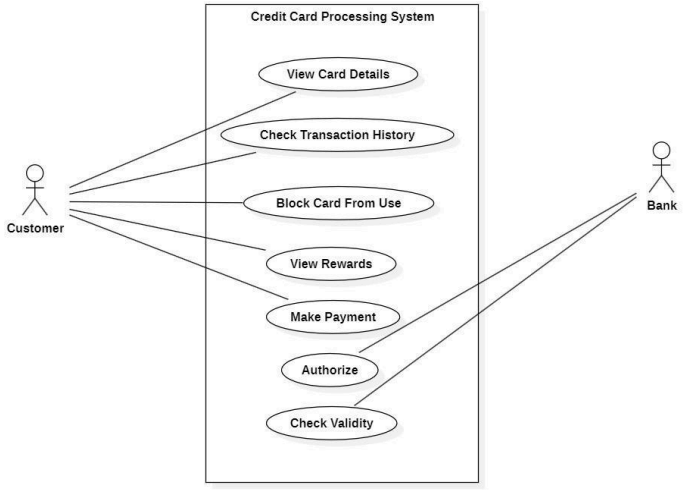


Fig 2.5.1: Use Case Diagram

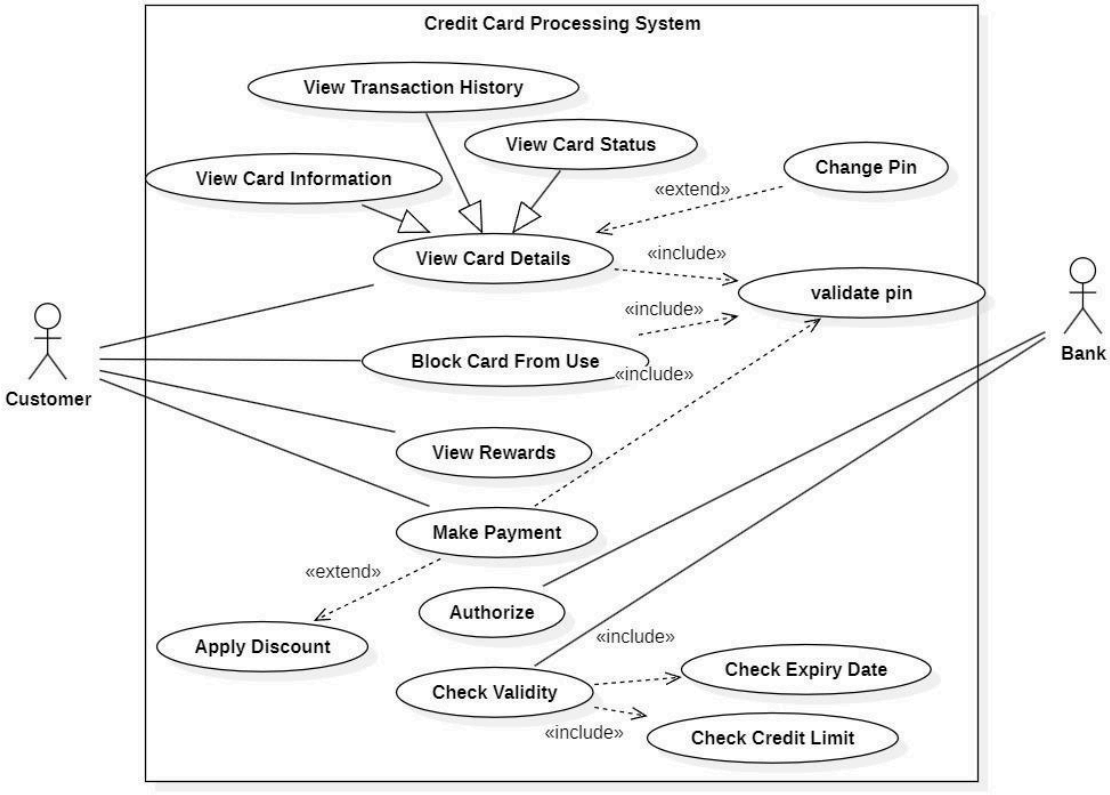


Fig 2.5.2: Advanced Use Case Diagram

Description:

The use case diagram identifies the interactions between actors and the system functionalities.

- **Actors:**
    - **Customer**: Manages card details, transactions, and rewards.
    - **Merchant**: Initiates transactions and payments.
    - **System**: Authorizes payments and manages account information.
- **Use Cases:**
    - **View Card Details**: Customers can check their credit card information.
    - **Check Transaction History**: Provides a log of past transactions for review.
    - **Block Card from Use**: Allows customers to block a lost or stolen card.
    - **View Rewards**: Displays accrued rewards or cashback for the customer.
    - **Authorize**: Validates a transaction initiated by the merchant.
    - **Make Payment**: Processes bill payments for the credit card.

2.6 Sequence Diagram:

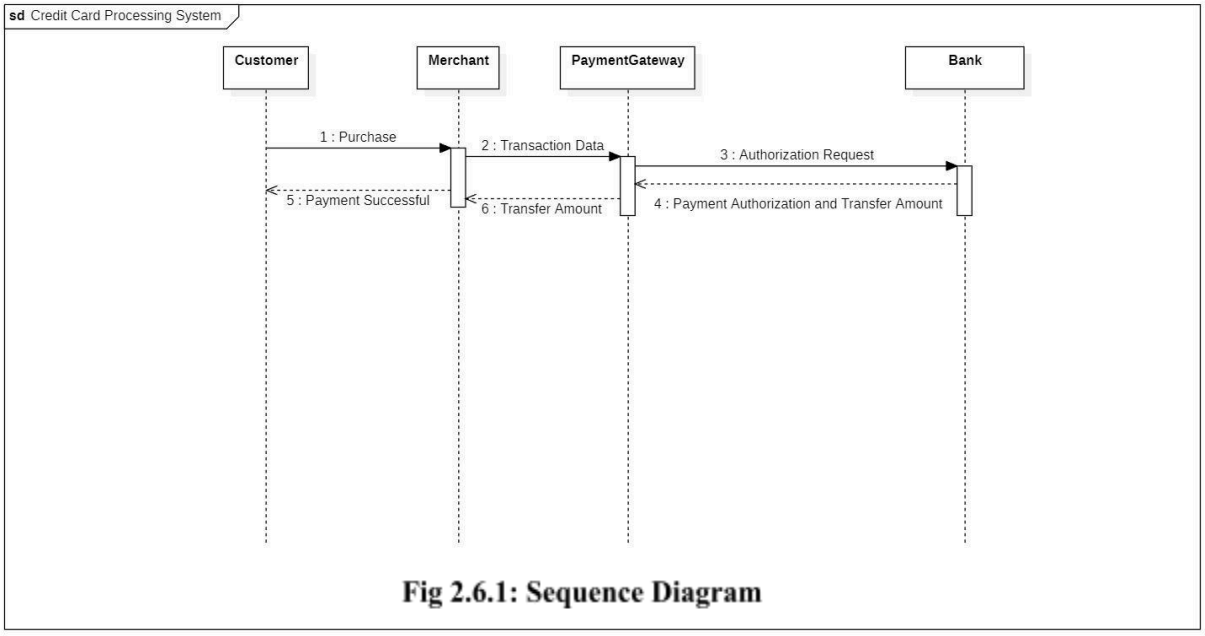


Fig 2.6.1: Sequence Diagram

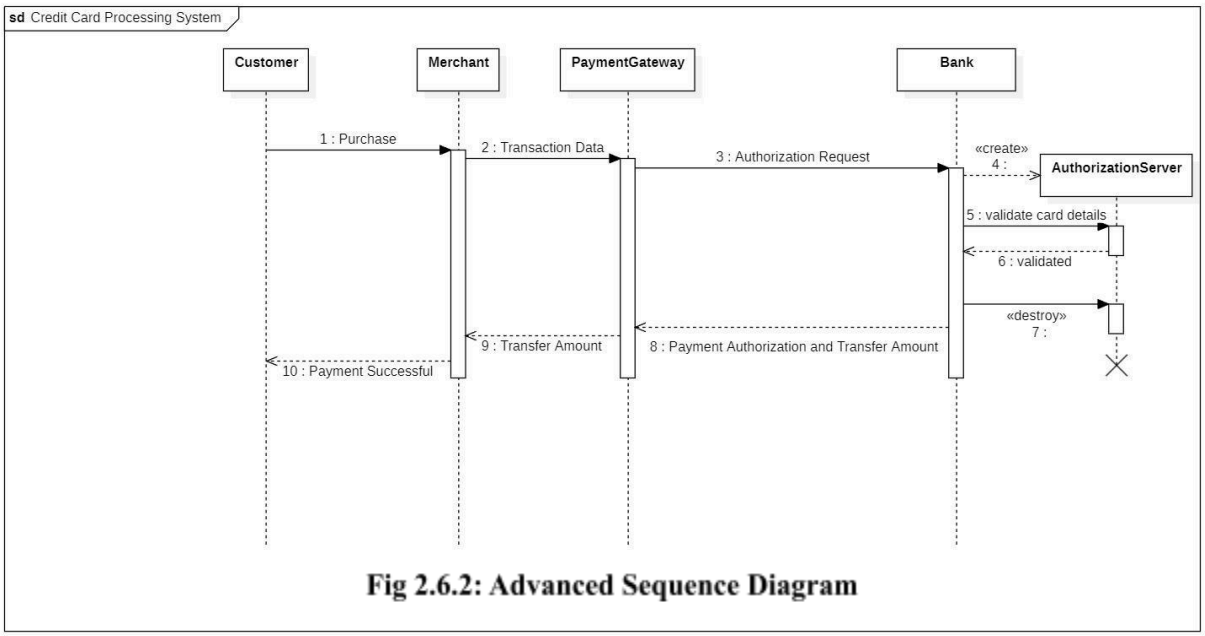


Fig 2.6.2: Advanced Sequence Diagram

Description:

The sequence diagram represents the interaction flow during a transaction.

- **Objects:**

- **Customer**: Initiates the transaction by providing card details.
- **Merchant**: Sends the payment request to the system.
- **Payment Gateway**: Mediates between the merchant and the bank, ensuring secure processing.
- **Bank**: Validates the card and processes the transaction.
- **Authorization (Transient Object)**: Created temporarily to validate the transaction.

- **Flow:**

0. Customer provides card details to the Merchant.
1. Merchant forwards the details to the Payment Gateway.
2. Payment Gateway requests transaction approval from the Bank.
3. Bank validates card and account details, creating an Authorization object.
4. Authorization object verifies and responds with an approval/decline status.
5. Payment Gateway informs the Merchant, who finalizes the transaction.

2.7 Activity Diagram

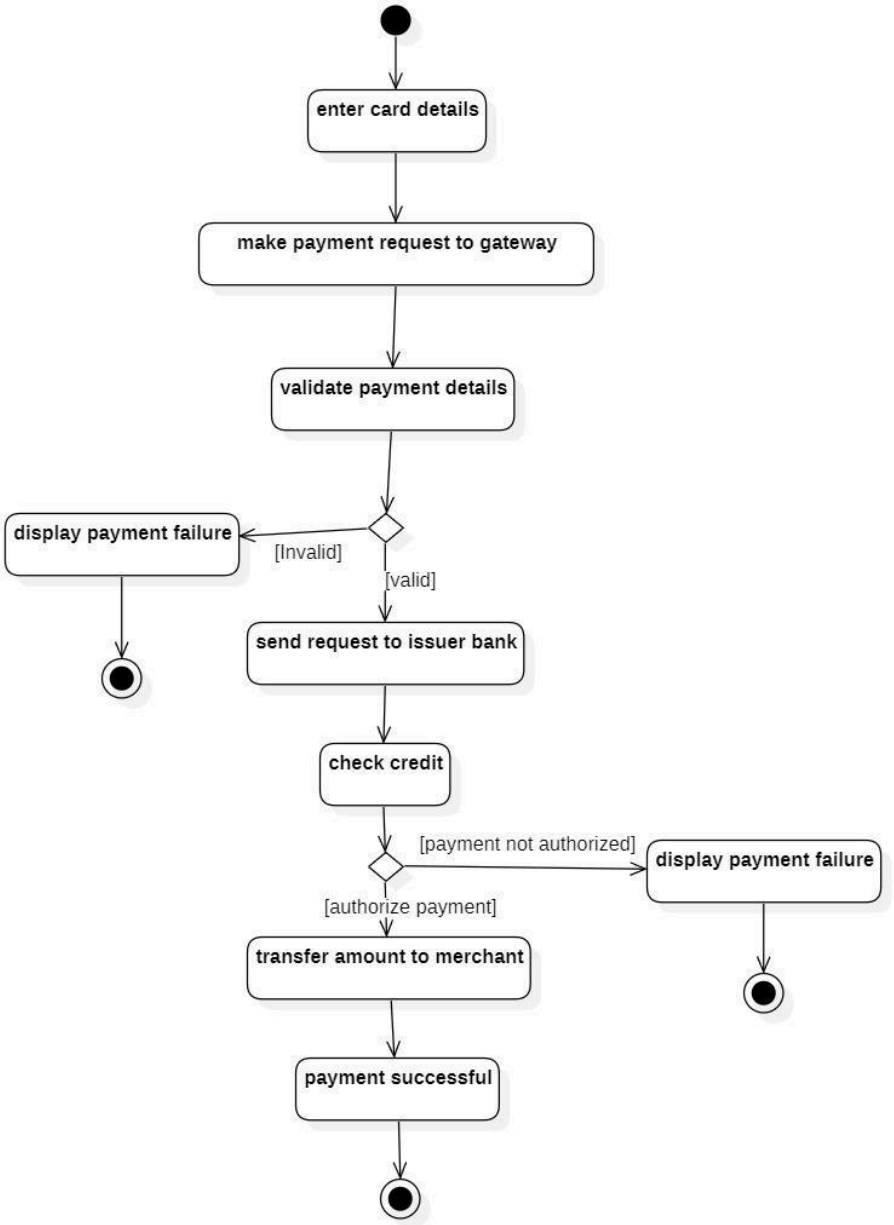


Fig 2.7.1: Activity Diagram

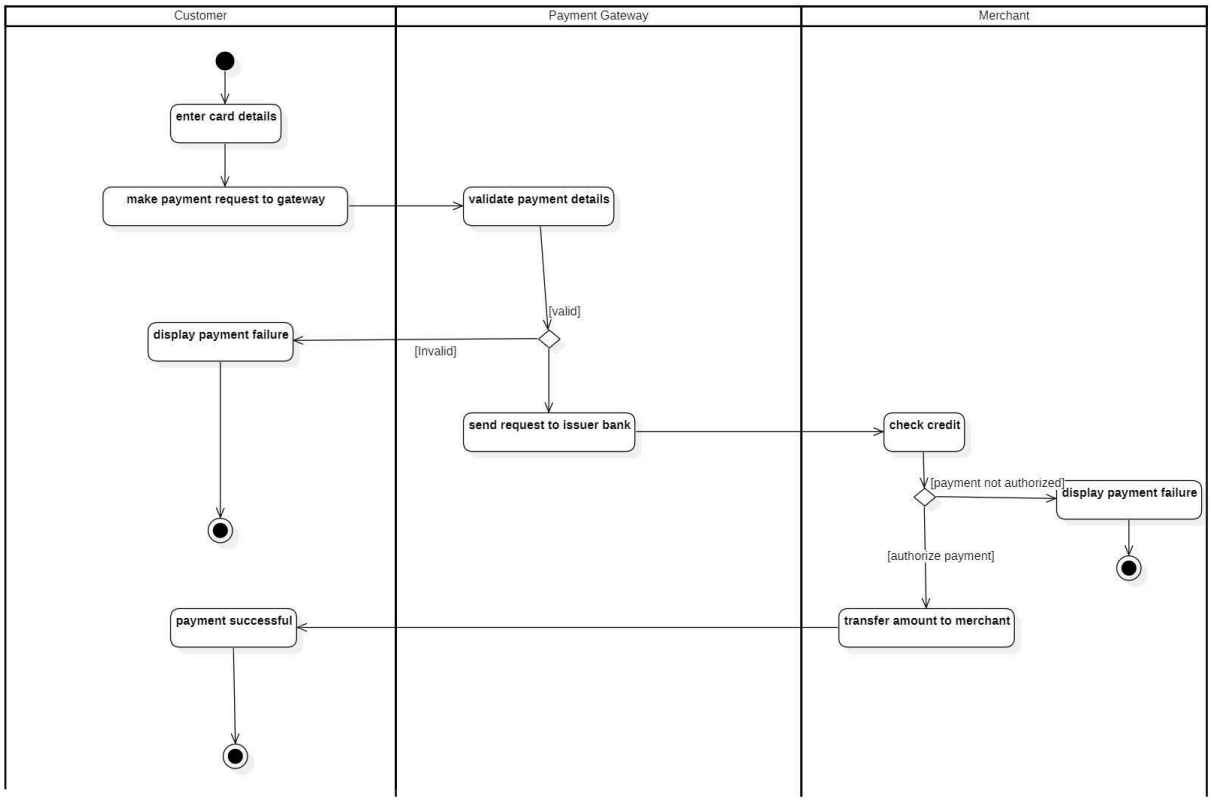


Fig 2.7.2: Advanced Activity Diagram

Description:

The activity diagram outlines the process flow during a payment, divided into swimlanes for clarity.

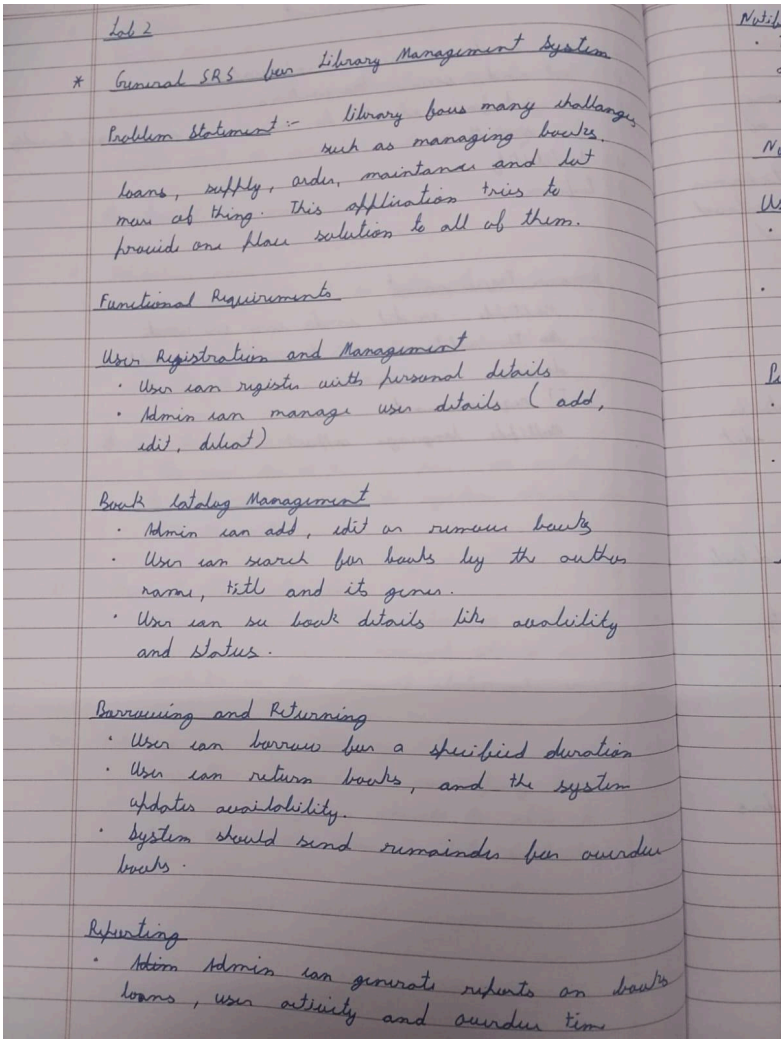
- **Swimlanes:**
    - **Payment Gateway**: Manages secure processing and communication with the bank.
    - **Customer**: Initiates the transaction and confirms payment.
    - **Merchant**: Processes the payment request and completes the transaction.
- **Flow:**
    0. Customer initiates payment by providing card details to the Merchant.
    1. Merchant sends payment details to the Payment Gateway.
    2. Payment Gateway validates details and communicates with the Bank.

3. Bank processes the transaction and confirms success/failure to the Payment Gateway.
4. Payment Gateway updates the Merchant, who completes the process and informs the Customer.

3. Library Management System

3.1 Problem Statement: Traditional library management systems rely heavily on manual record-keeping, which is inefficient and susceptible to errors. Tasks like book issuance, returns, fine calculations, and inventory management are time-consuming and difficult to track. There is a need for an automated system that can streamline these operations, provide accurate records, and improve user accessibility to library resources.

3.2 SRS:



Lab 2

* General SRS for Library Management System

Problem Statement :- Library have many challenges such as managing books. loans, supply, order, maintance and lot more of thing. This application tries to provide one place solution to all of them.

Functional Requirements

User Registration and Management
- User can registr with personal details
- Admin can manage user details (add, edit, delet)

Book Catalog Management
- Admin can add, edit or remove books
- User can search for books by the author name, title and its genre.
- User can se book details like availibity and status.

Borrowing and Returning
- User can borrow for a specified duration
- User can return books, and the system updates availability.
- System should send remainder for overdue books.

Reporting
- Adm Admin can genrate reports on books loans, user activity and overdue tim

Notification

- The system will notify user via email bus account updates overdue remainds, and reservation availility.

Non-functional Requirements

Usability

- The system shall have an intuitive user interface to enhance user experience.
- User documentation should be provided bus all the functionality

Performance

- The system should support hundread concurrent users at the same time.
- Book transaction should be ~~been~~ processed very fast.

Security

- User data should be stored securely with end to end encryption.
.

Reliability

- System should have high ~~update time~~ Uptime
- Data recovery and and back up mechanisms must be in place

Interface Requirement

- UI
- User login / registration
- home page
- Book search and detail page

3.3 Class Diagram:

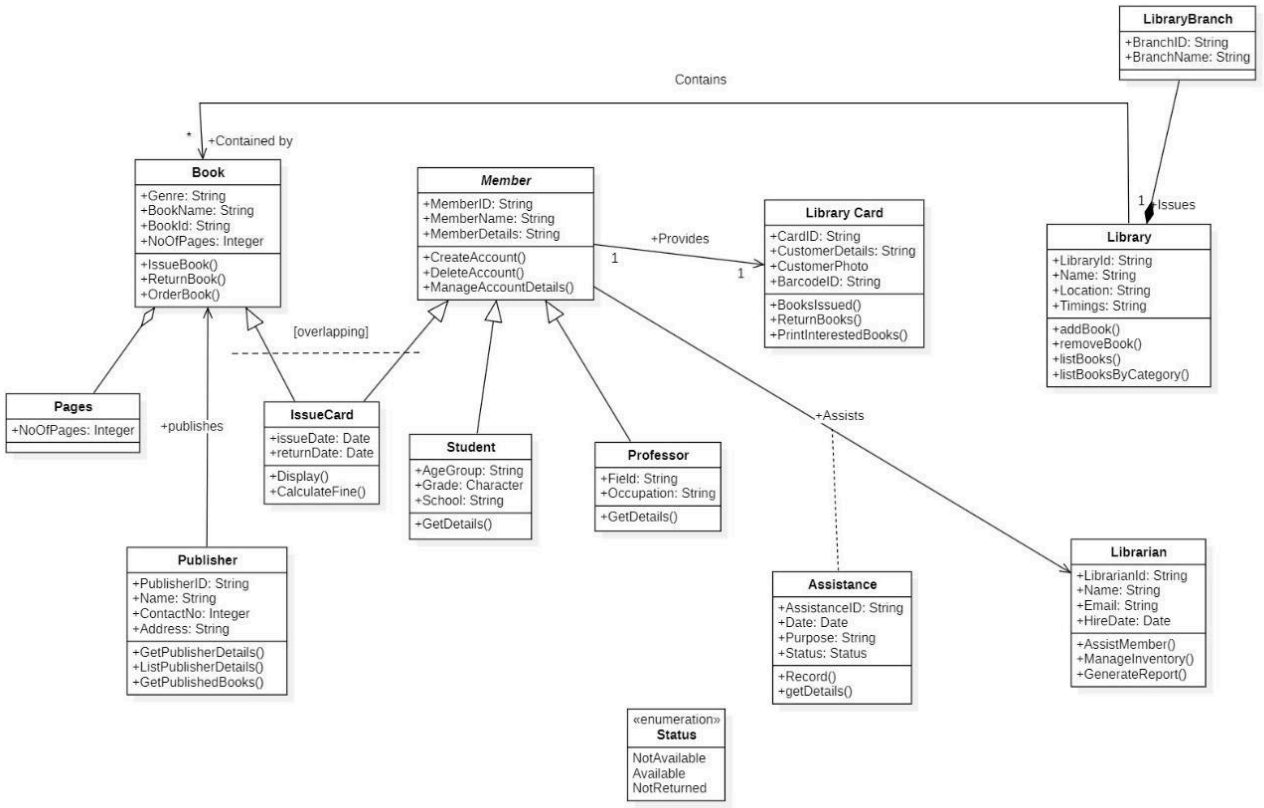


Fig 3.3.1: Class Diagram

Description:

The class diagram outlines the structure of the library management system, highlighting key entities and their relationships.

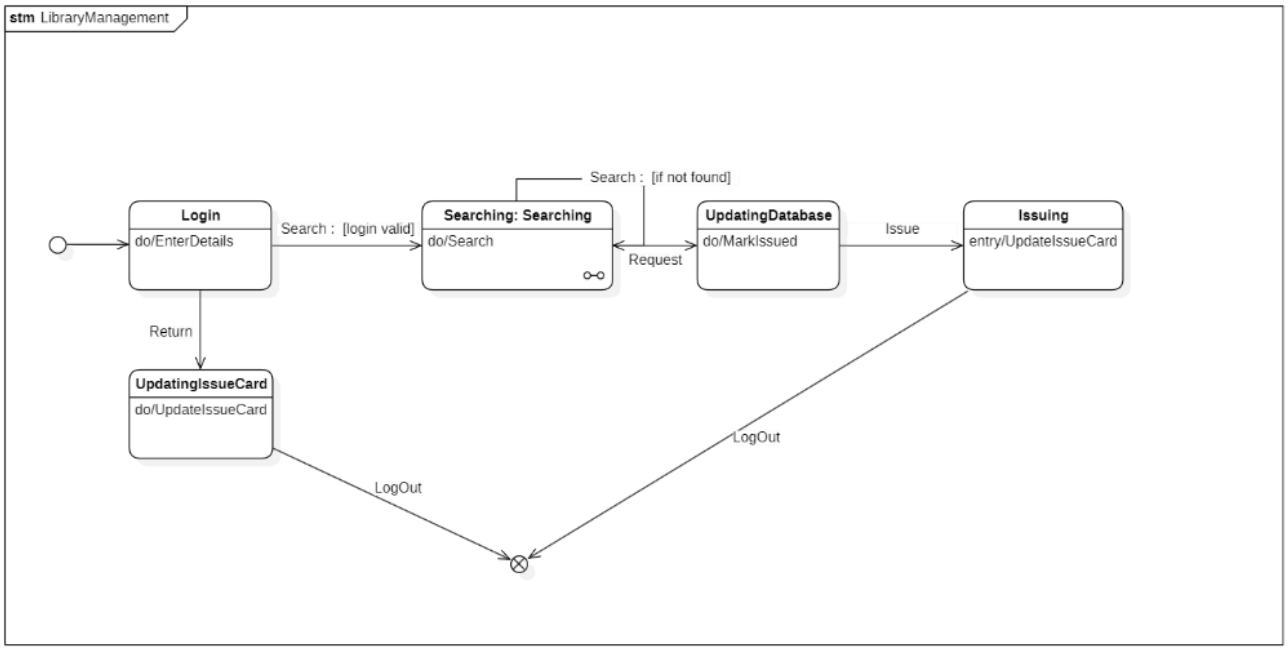
- **Classes:**
    - **Book Class**: Represents a book with attributes like book ID, title, author, genre, and availability status.
    - **Member Class**: Abstract class representing library members, generalized into:
        - **Student Class**: A specific type of member with attributes like student ID and grade.
        - **Professor Class**: A specific type of member with attributes like professor ID and department.

- **LibraryCard Class**: Represents a unique card assigned to members to track borrowing activity.
- **IssueCard Class**: Represents a temporary overlap of **Book** and **Member**, tracking the details of issued books, including issue date, due date, and status.
- **Library Class**: Manages the overall library system, composed of:
  - **LibraryBranch Class**: Represents individual branches with attributes like branch ID, name, and address.

- **Relationships:**
  - The **Member Class** and **LibraryCard Class** have an association since each member holds a library card.
  - The **IssueCard Class** overlaps the **Book Class** and **Member Class**, representing borrowed books.
  - **Library Class** is composed of multiple **LibraryBranch Classes** for a distributed library system.

‘

3.4 State Diagram:



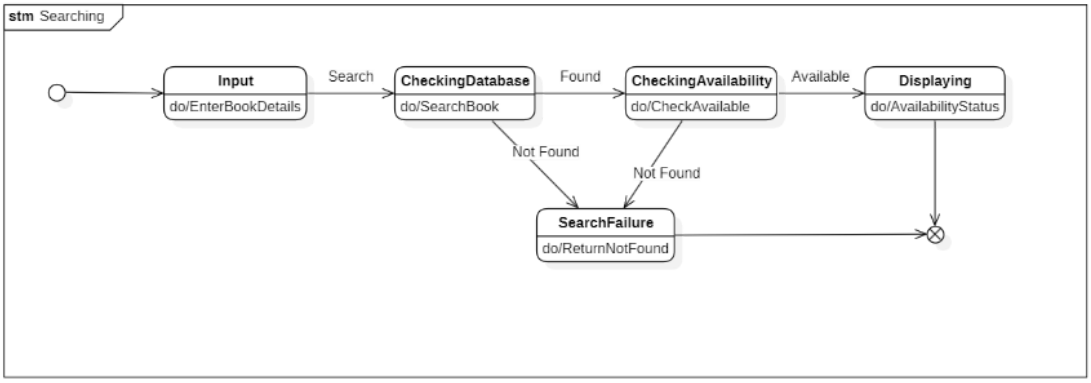
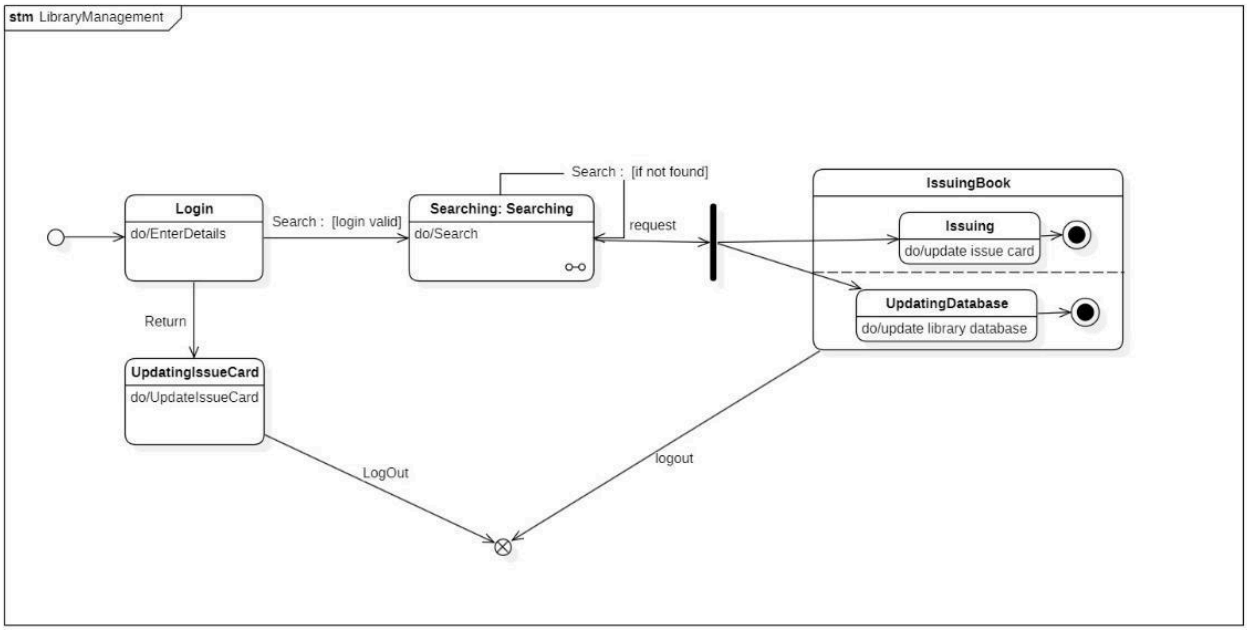


Fig 3.4.1: State Diagram



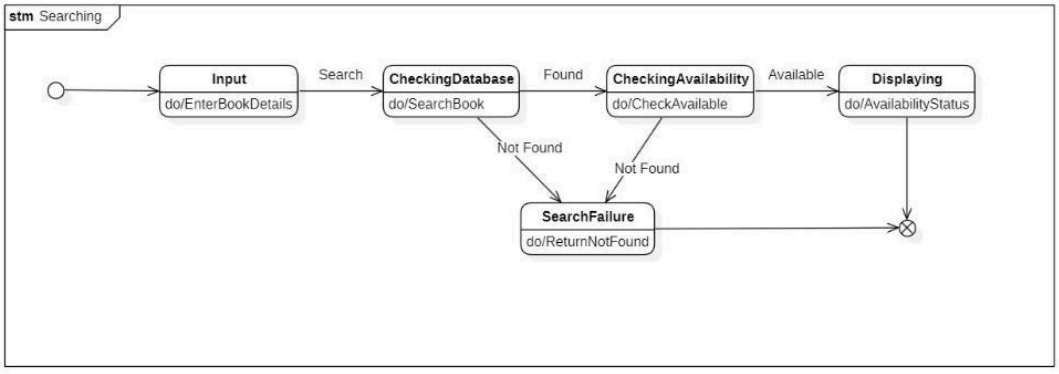


Fig 3.4.2: Advanced State Diagram

Description:

The state diagram illustrates the transitions between various states in the library management system.

- **States:**
    - **Login State**: Represents when users or librarians log into the system.
    - **Searching State**: Handles searching for books based on criteria like title, author, or genre.
        - Sub-states:
            - Keyword Search

        - Advanced Search (e.g., by publication date or availability)

    - **Updating Database**: Handles the addition, deletion, or modification of records (e.g., book details or member details).
        - Concurrent with **Issuing**.
    - **Issuing State**: Manages the borrowing process, including verifying book availability and member status.
    - **Updating Issue Card**: Updates records in the **IssueCard Class** to track borrowed books.

- **Concurrency:**
    - **Issuing** and **Updating Database** occur simultaneously.

3.5 Use Case Diagram:

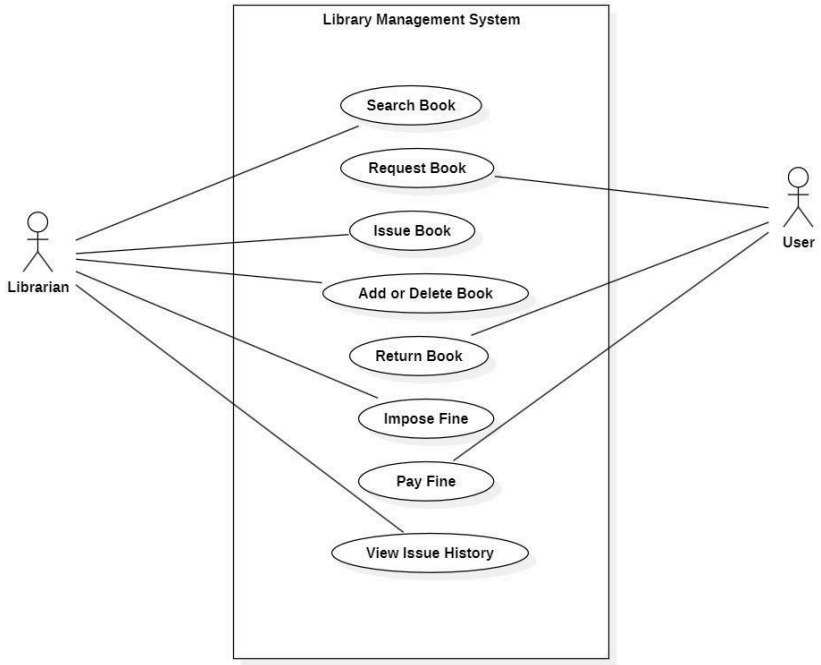


Fig 3.5.1: Use Case Diagram

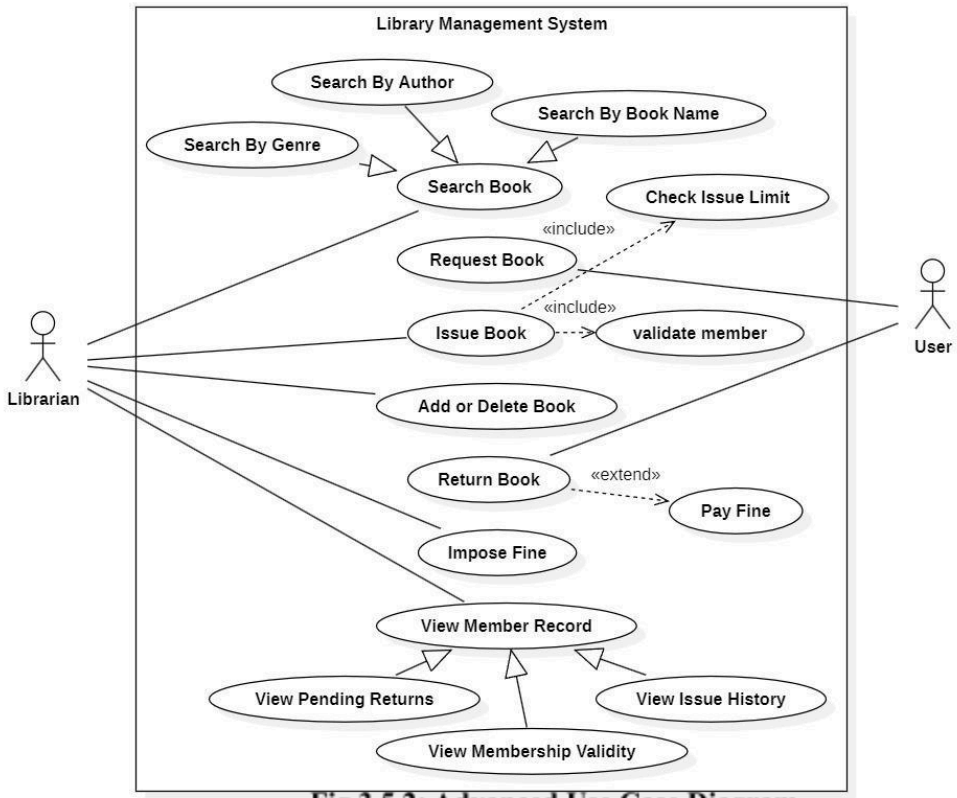


Fig 3.5.2: Advanced Use Case Diagram

Description:

The use case diagram defines the interactions between actors and system functionalities.

- **Actors:**
    - **Librarian**: Manages library operations such as adding/deleting books and imposing fines.
    - **Member**: Includes both students and professors, who interact with the system for book-related tasks.
- **Use Cases:**
    - **Search Book**: Members search for books in the library catalog.
    - **Request Book**: Members can place a request for unavailable books.
    - **Issue Book**: Books are issued to members after verification.
    - **Add or Delete Book**: Librarians manage the catalog by adding or removing books.
    - **Return Book**: Members return borrowed books, and the system updates the database.
    - **Impose Fine**: Librarians impose fines for overdue returns.
    - **Pay Fine**: Members pay the imposed fines through the system.
    - **View Issue History**: Members can view their borrowing history.

3.6 Sequence Diagram:

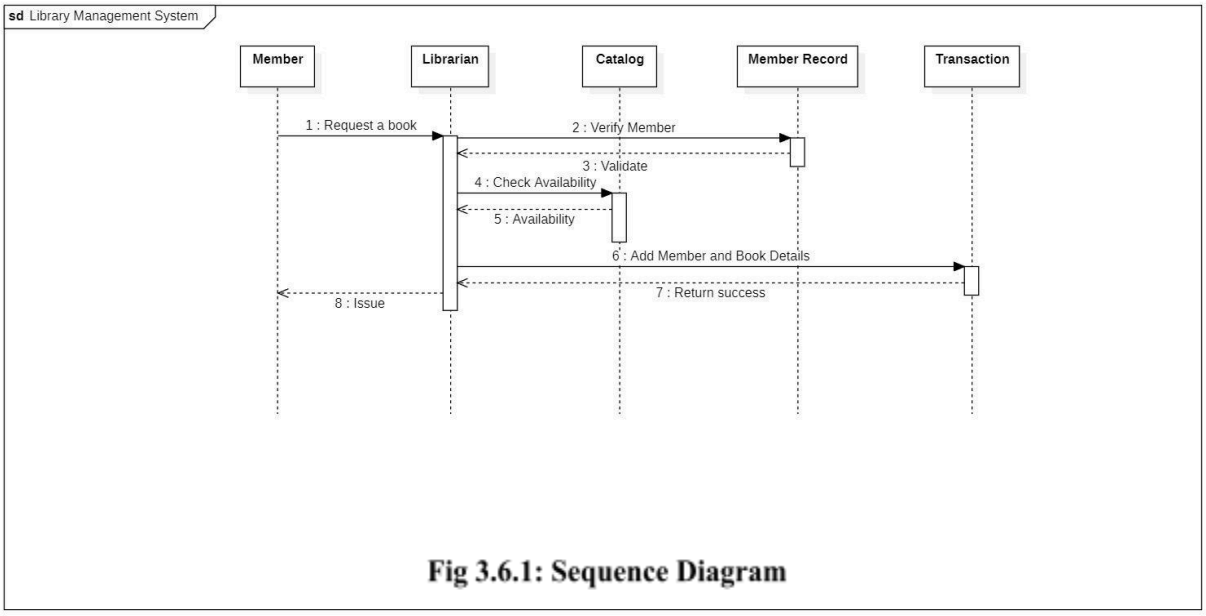


Fig 3.6.1: Sequence Diagram

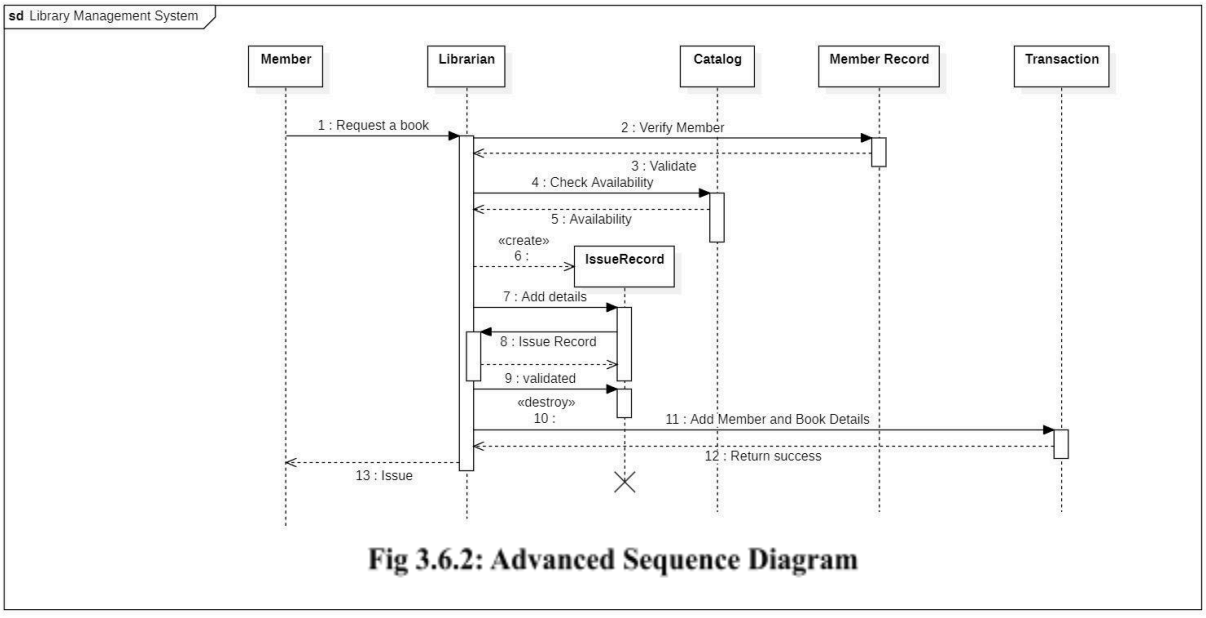


Fig 3.6.2: Advanced Sequence Diagram

Description:

The sequence diagram illustrates the interaction flow for issuing a book.

- **Objects:**
    - **Member**: Initiates a book issue request.
    - **Librarian**: Verifies the member's credentials and book availability.
    - **Catalog**: Checks the availability of the requested book.
    - **Member Record**: Updates the member's borrowing details after the book is issued.
    - **Transaction**: Represents the book-issuing transaction, tracking details like issue and due dates.
- **Flow:**
    0. Member searches for a book and requests to issue it.
    1. Librarian checks the Member Record and verifies the book in the Catalog.
    2. Catalog confirms book availability.
    3. Transaction object records the issue details.
    4. Member Record is updated with the new transaction.

3.7 Activity Diagram

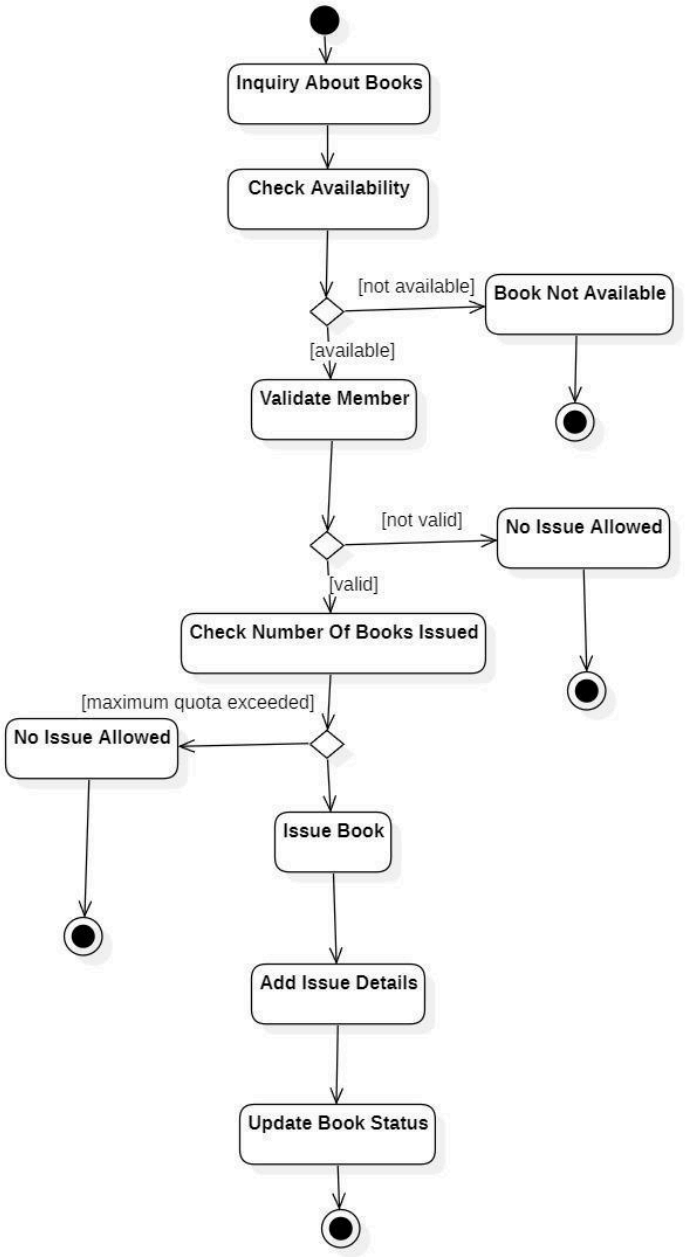


Fig 3.7.1: Activity Diagram

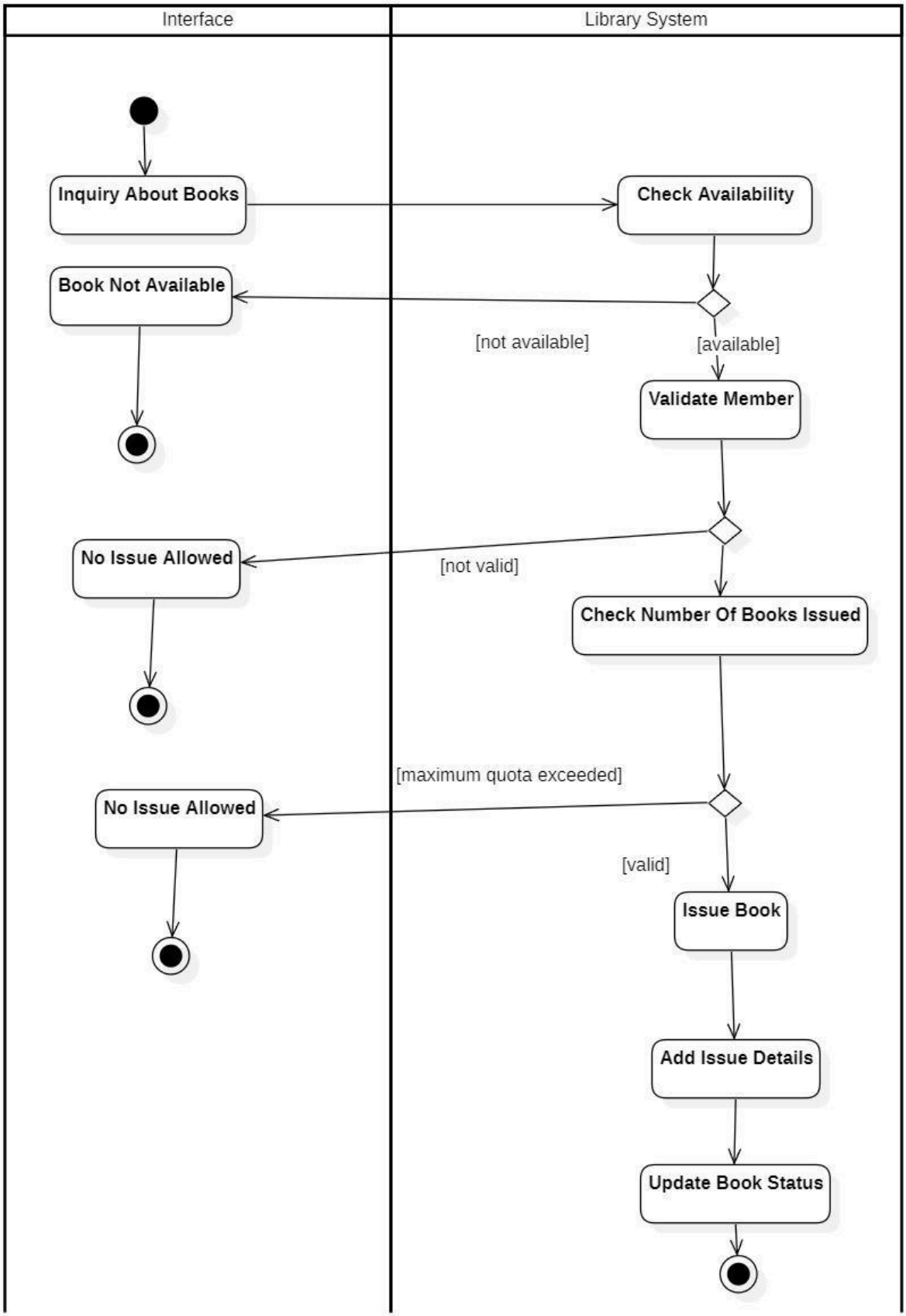


Fig 3.7.2: Advanced Activity Diagram

Description:

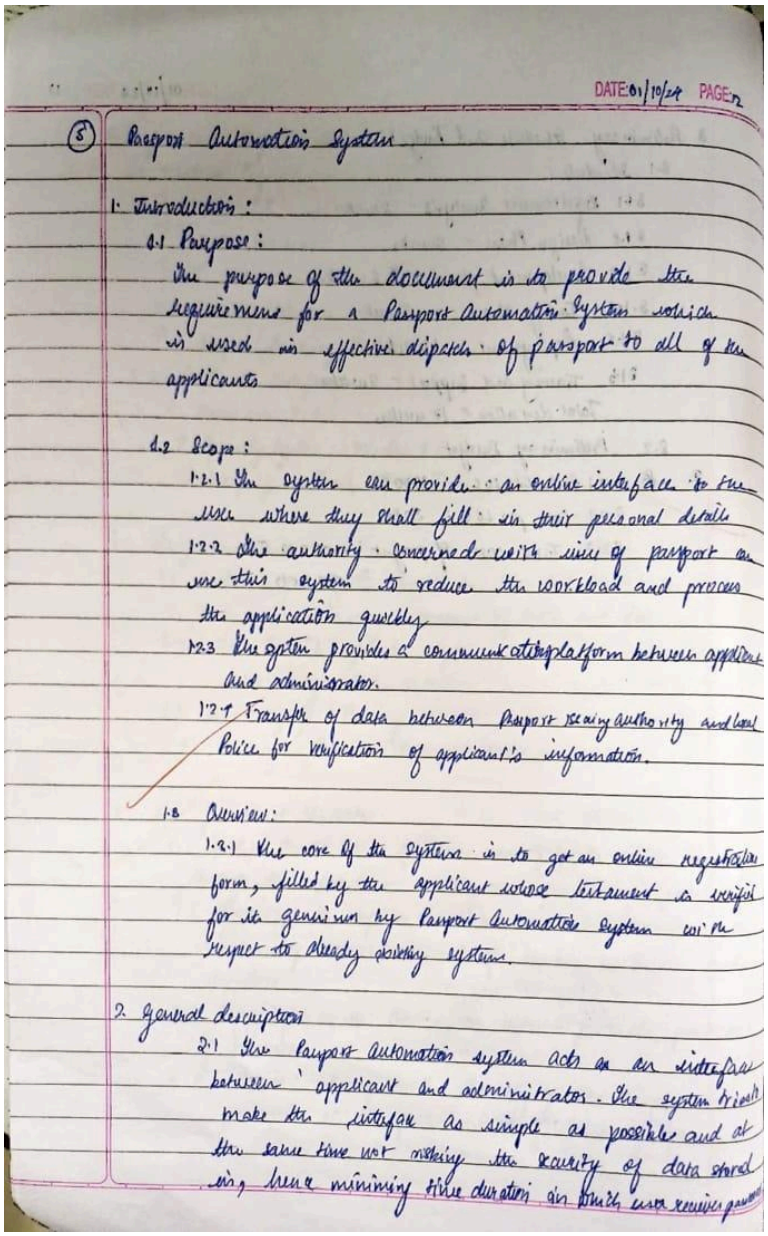
The activity diagram shows the process of issuing a book, divided into swimlanes for clarity.

- **Swimlanes:**
    - **Interface**: Represents the system's user interface where members and librarians interact.
    - **Library Management**: Handles backend operations like book availability checks, member validation, and transaction processing.
- **Flow:**
    0. Member logs in and searches for a book via the Interface.
    1. Library Management validates the request and checks book availability.
    2. If available, the book is issued, and the transaction is recorded.
    3. Interface updates the member's borrowing details and confirms the issue.

4. Passport Automation System

4.1 Problem Statement: The manual process of passport issuance and renewal is cumbersome, time-consuming, and prone to delays due to inefficiencies in document verification and record management. Applicants often face challenges in tracking the status of their applications. An automated passport management system is needed to streamline the entire process, from application submission to passport issuance, while ensuring data security and transparency.

4.2 SRS:



Passport Automation System

1. Introduction :

1.1 Purpose :

The purpose of the document is to provide the requirements for a Passport Automation System which is used in effective dispatch of passport to all of the applicants

1.2 Scope :

1.2.1 The system can provide an online interface to the user where they shall fill in their personal details

1.2.2 The authority concerned with issue of passport can use this system to reduce the workload and process the application quickly

1.2.3 The system provides a communication platform between applicant and administrator.

1.2.4 Transfer of data between Passport issuing authority and local Police for verification of applicant's information.

1.3 Overview :

1.3.1 The core of the system is to get an online registration form, filled by the applicant whose testament is verified for its genuineness by Passport Automation System with respect to already existing systems.

2. General description

2.1 The Passport Automation system acts as an interface between applicant and administrator. The system tries to make the interface as simple as possible and at the same time not missing the security of data stored in, hence minimizing time duration in which user receives pass

2.2 It aims at improving the efficiency in the issue of Passport and reduces the complexities involved in it to maximum extent possible.

3. Functional Requirements:

3.1 Passport Information: This module is helpful for the people to move to other states or countries for their work.

3.2 Applying for Passport: This module is used for show to apply for the passport and which certificates are needed to submit for the particular types of passport.

3.3 Payment: This module is used to pay fees in different ways.

3.4 Document Uploading: To submit the document.

3.5 Verification Module: To verify all certifications online and change the status.

3.6 Authentication Module: This module is used to check whether user is valid or not.

3.7 Status verification and feedback: To check status of verification process and delivery of passport.

4. Interface Requirements: The interface should be easy to use and user friendly. It should have

→ User Input Form fields
→ Status DashBoard
→ Notifications
→ File Upload
→ File Size and Format Validation
→ Payment Gateway
→ Profile Management
→ FAQ section

4.3 Class Diagram:

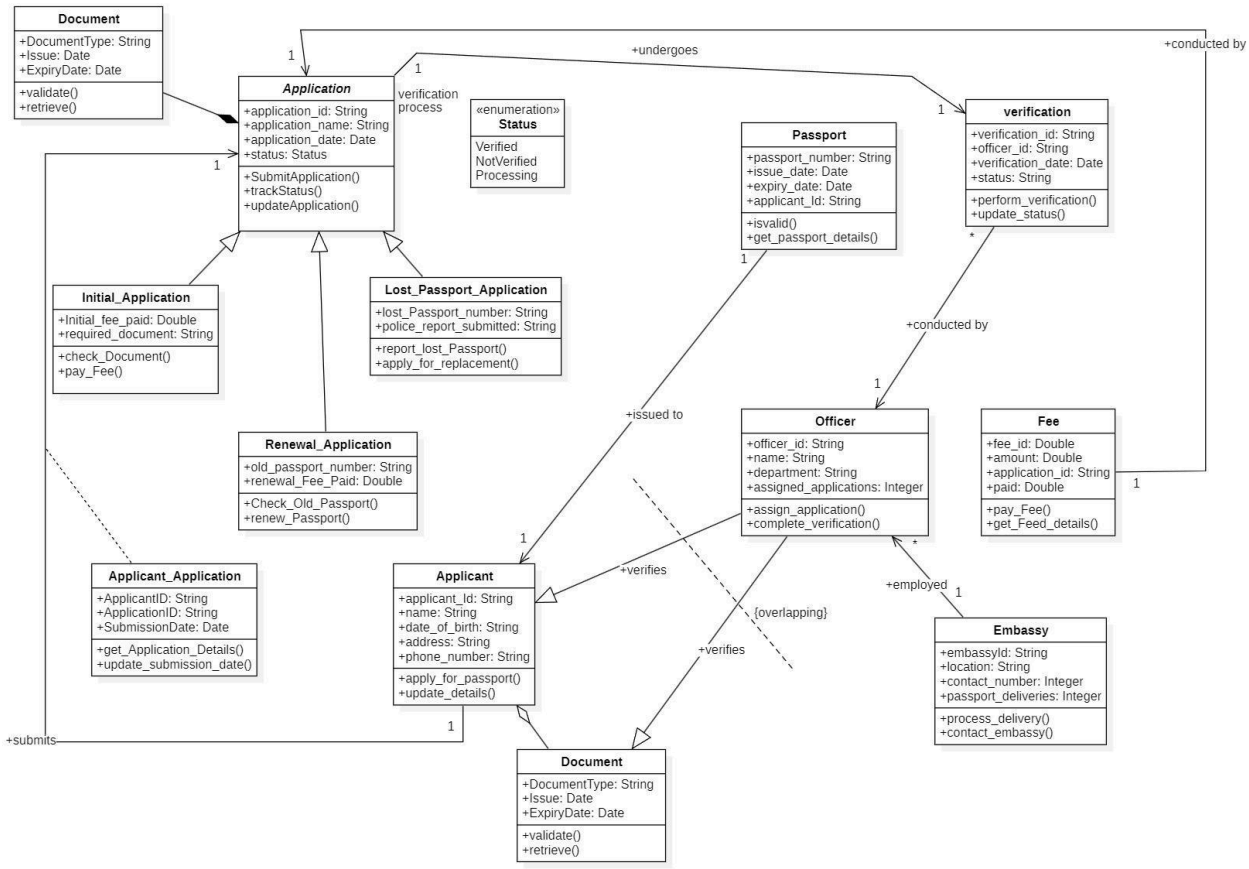


Fig 4.3.1: Class Diagram

Description:

The class diagram represents the static structure of the system, highlighting main entities and their relationships.

- **Classes:**
    - **Application Class**: Represents the overall process of passport applications, with generalizations:
        - **Initial Application**: For first-time passport applicants, containing attributes like birth certificate details and ID proof.
        - **Renewal Application**: For applicants renewing expired or soon-to-expire passports, with renewal-specific attributes.

        - **Lost Passport Application**: For applicants reporting and replacing lost passports, including police complaint details.
    - **Passport Class**: Represents a passport with attributes like passport number, issue date, expiration date, and applicant details.
    - **Verification Class**: Manages the verification process, with attributes like verification ID, type (police, regional admin), and status (approved/rejected).
- **Relationships:**
    - **Application Class** is associated with **Passport Class** to represent the issuance process.
    - **Verification Class** is linked to **Application Class** to ensure every application goes through proper checks.

4.4 State Diagram:

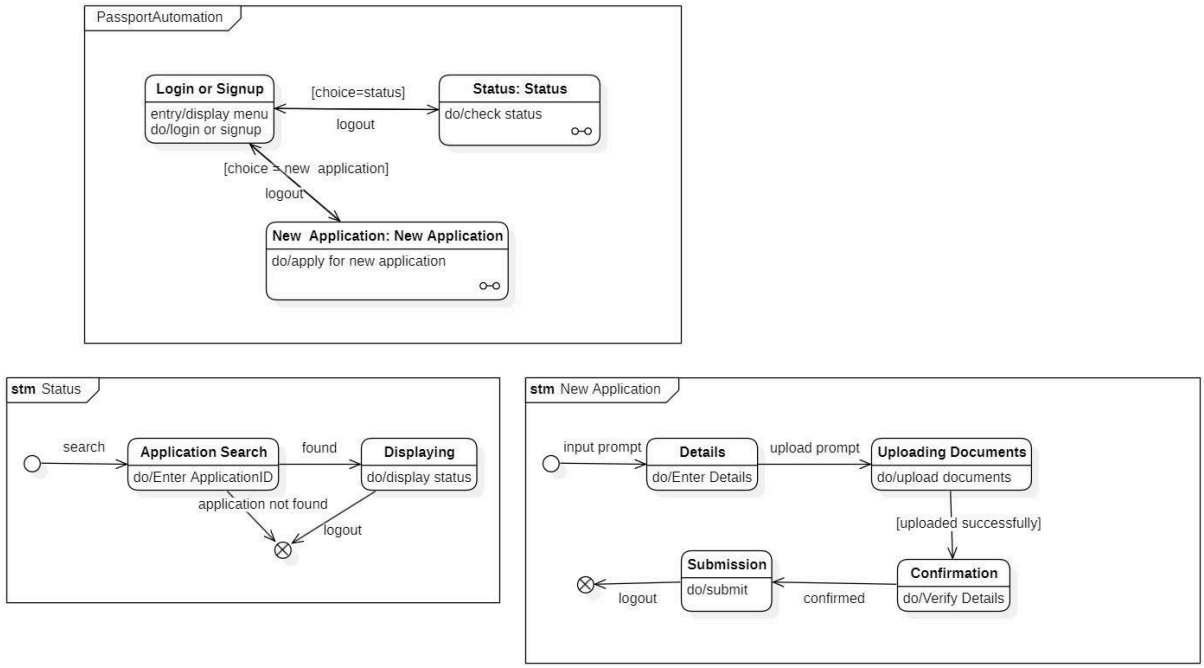


Fig 4.4.1: State Diagram

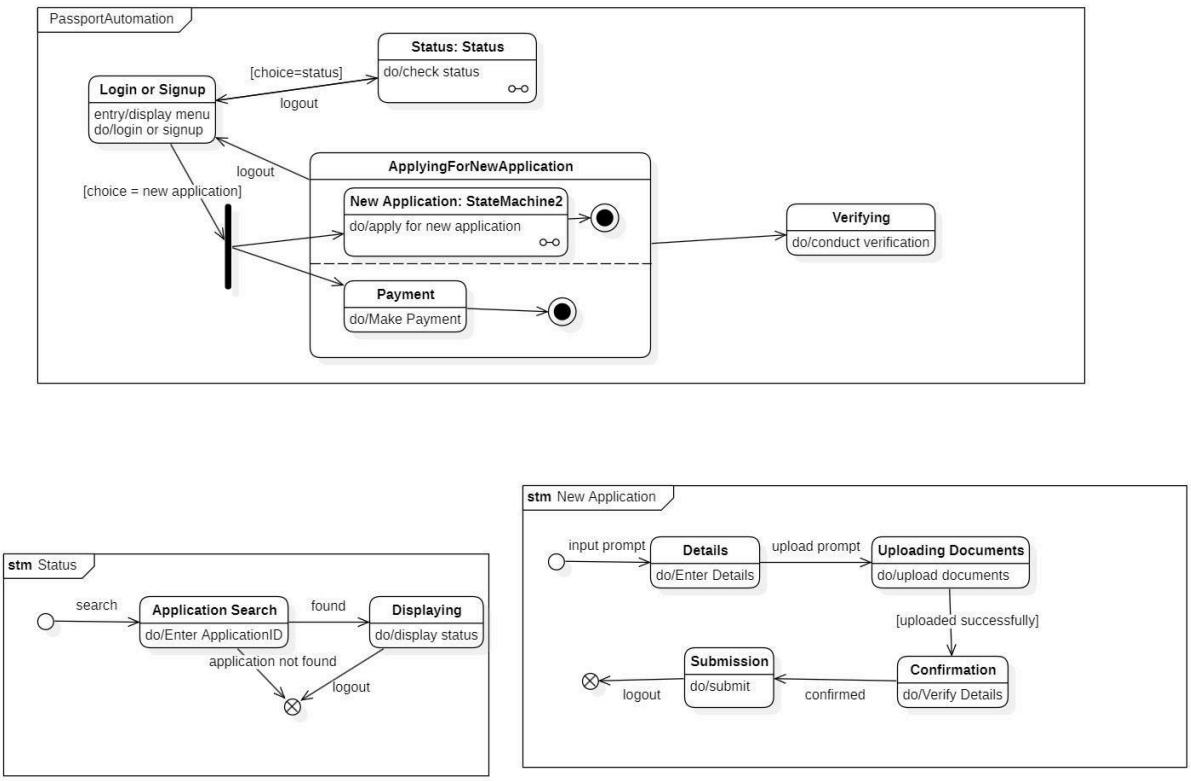


Fig 4.4.2: Advanced State Diagram

Description:

The state diagram outlines the transitions between different states in the passport automation system.

- **States:**
    - **Login State**: Represents user authentication into the system.
    - **Status State**: Enables applicants to check their application status.
    - **Verifying State**: Manages the verification of applicant details by authorities.
    - **Applying for New Application State**: Handles the process of submitting a new application.
        - **Concurrent States**:
            - **New Application**: Submits applicant details and required documents.

        - **Payment**: Manages the payment of fees for the application.

- **Submachines**:

    - **New Application**: Includes steps like entering details and uploading documents.
    - **Status**: Tracks the progress of the application.

4.5 Use Case Diagram:

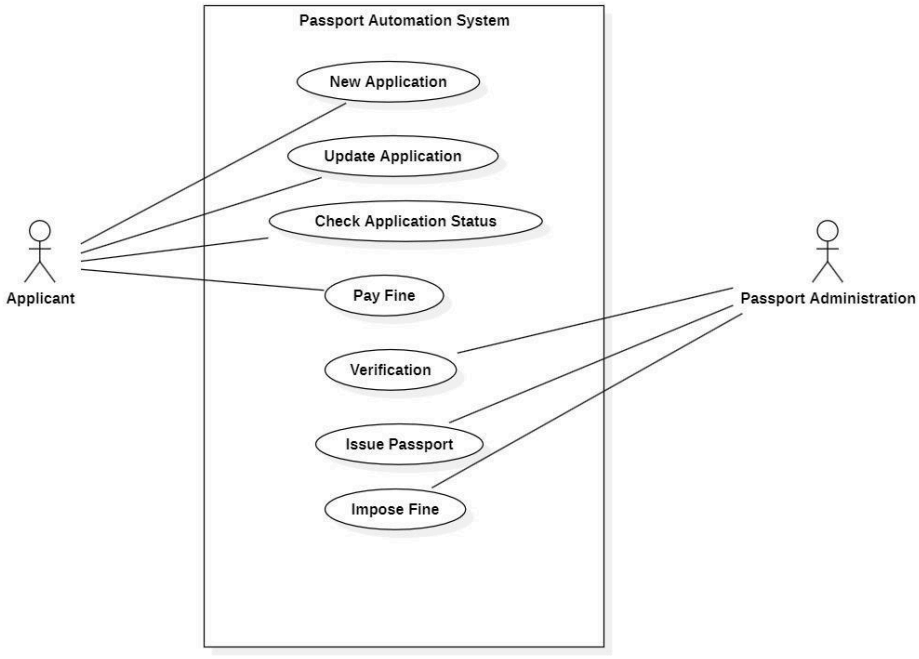


Fig 4.5.1: Use Case Diagram

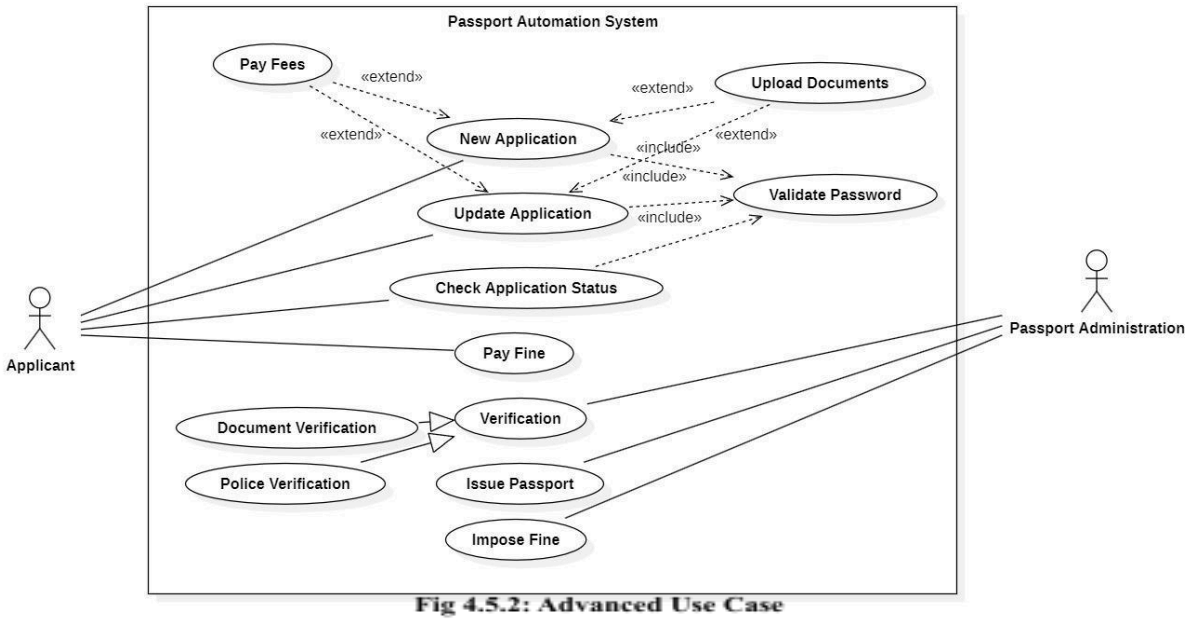


Fig 4.5.2: Advanced Use Case Diagram

Description:

The use case diagram illustrates interactions between users (actors) and system functionalities.

- **Actors:**
    - **Applicant**: Submits applications, checks status, and pays fines.

    - **Passport Admin**: Processes applications and imposes fines.

- **Use Cases:**
    - **New Application**: Submit a fresh application for a passport.
    - **Update Application**: Make changes to an ongoing or existing application.
    - **Check Application Status**: Track the progress of an application.
    - **Verification**: Authorities verify applicant details.
    - **Pay Fine**: Handles fines for late renewals or errors.
    - **Issue Passport**: Final step where the passport is issued to the applicant.

Impose Fine: Admin imposes penalties for discrepancies or delays.

4.6 Sequence Diagram:

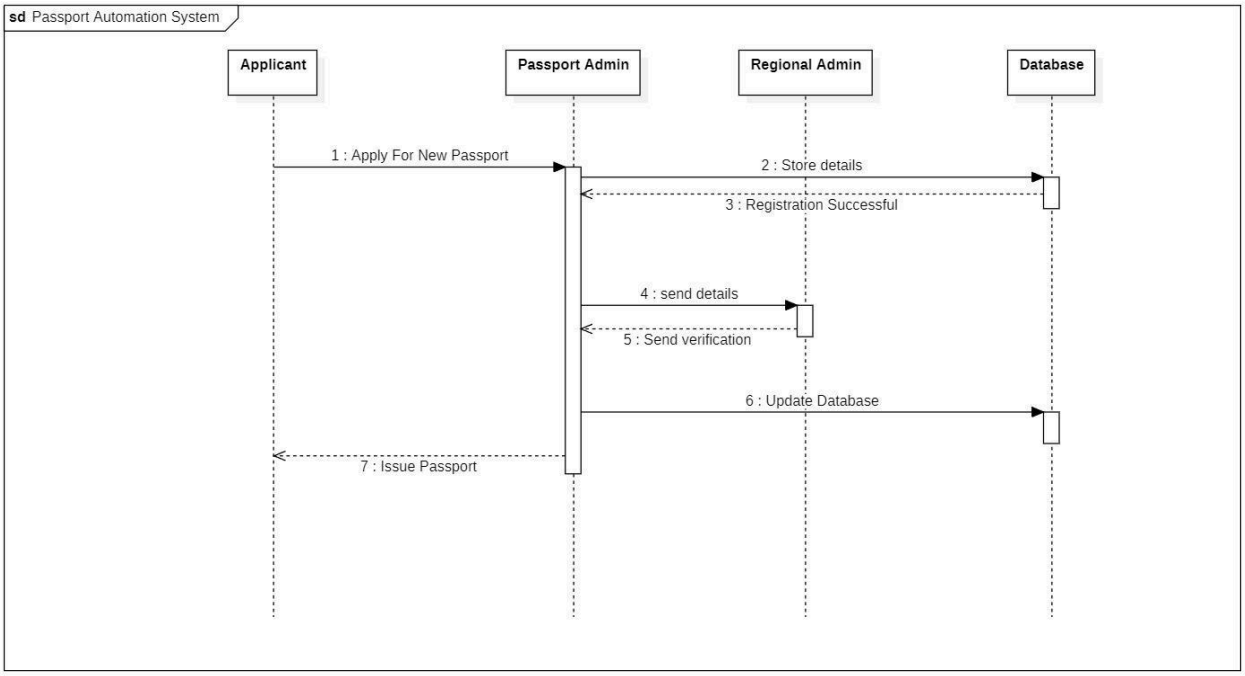


Fig 4.6.1: Sequence Diagram

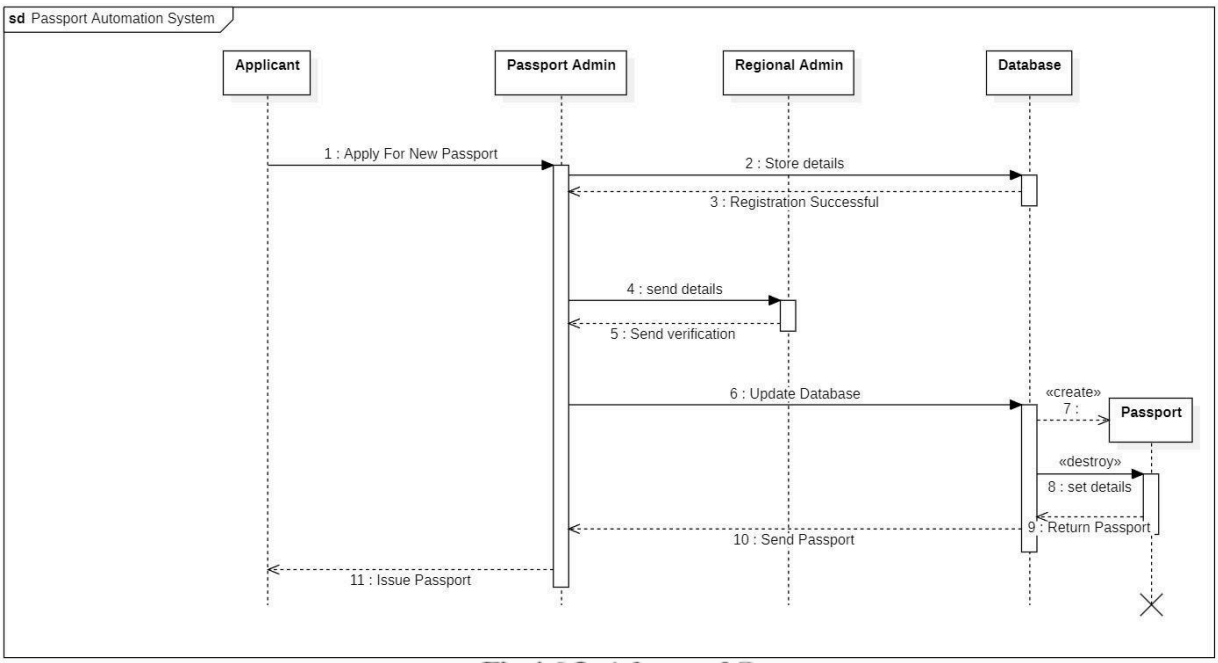


Fig 4.6.2: Advanced Sequence Diagram

Description:

The sequence diagram showcases the flow of interactions during the application process.

- **Objects:**
    - **Applicant**: Initiates the application and provides details.
    - **Passport Admin**: Processes the application and coordinates verification.
    - **Regional Admin**: Handles escalations or approvals for specific cases.
    - **Database**: Stores and retrieves application and verification information.
    - **Passport (Transient Object)**: Temporarily represents the passport being issued.
- **Flow:**
    0. Applicant submits application details to the Passport Admin.
    1. Passport Admin stores the application in the Database and forwards it for verification.
    2. Regional Admin oversees the verification process and approves/rejects the application.
    3. Upon approval, the Database updates the application status and generates the Passport object.
    4. Applicant is notified of the issuance.

4.7 Activity Diagram

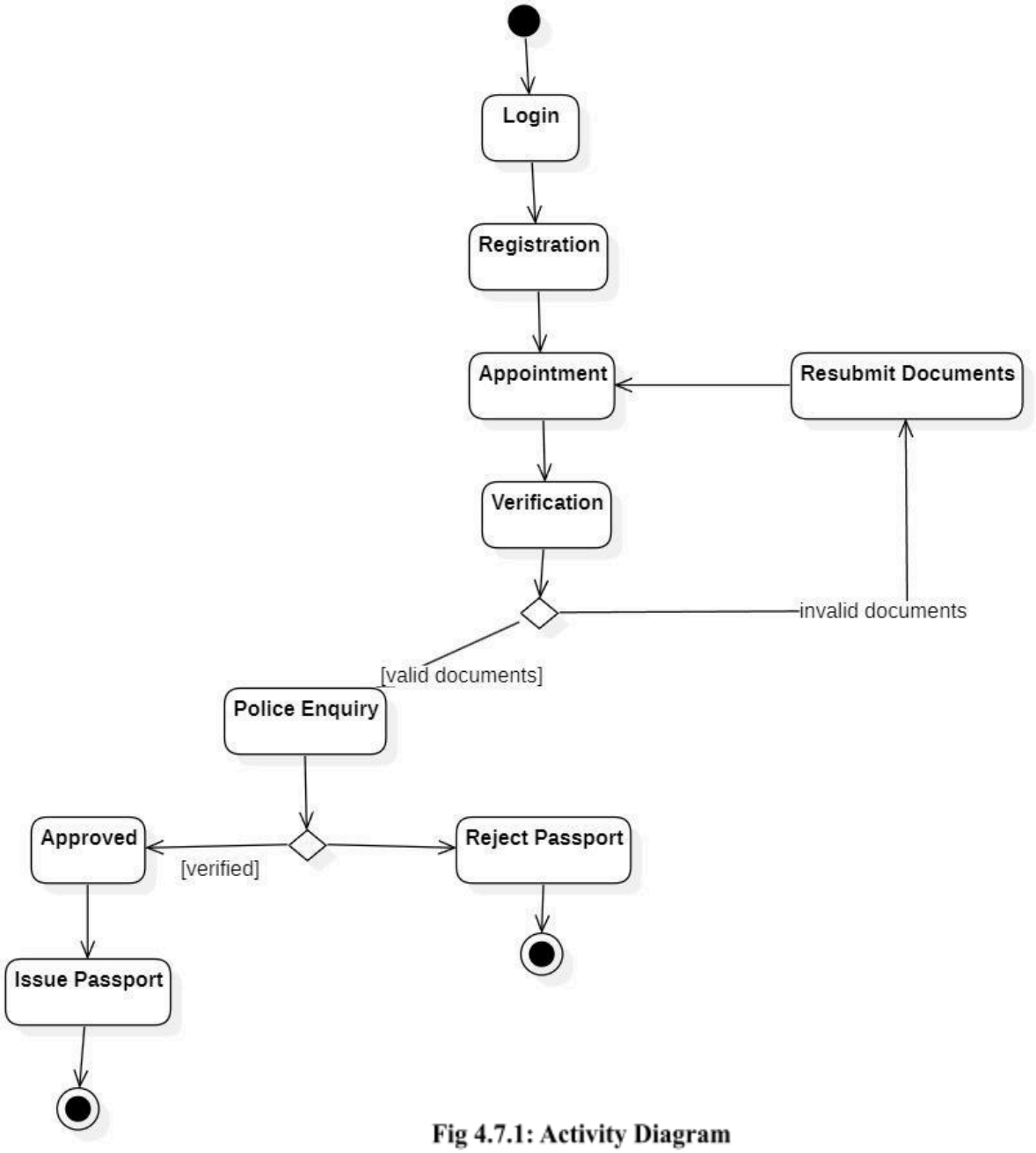


Fig 4.7.1: Activity Diagram

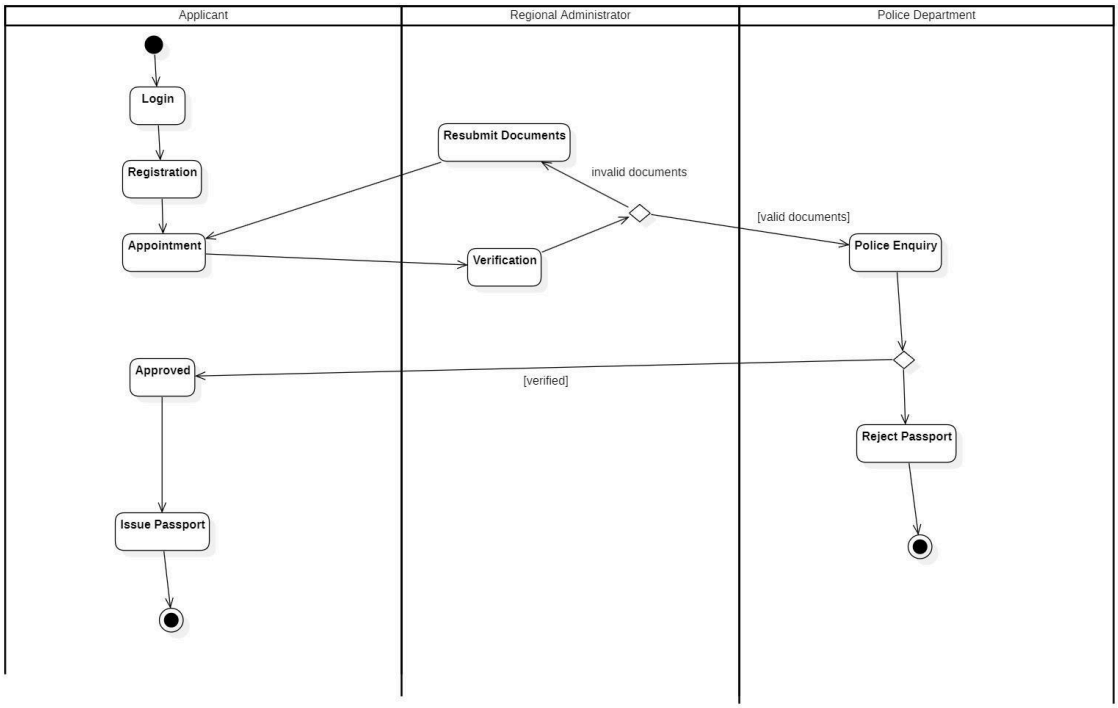


Fig 4.7.2: Advanced Activity Diagram

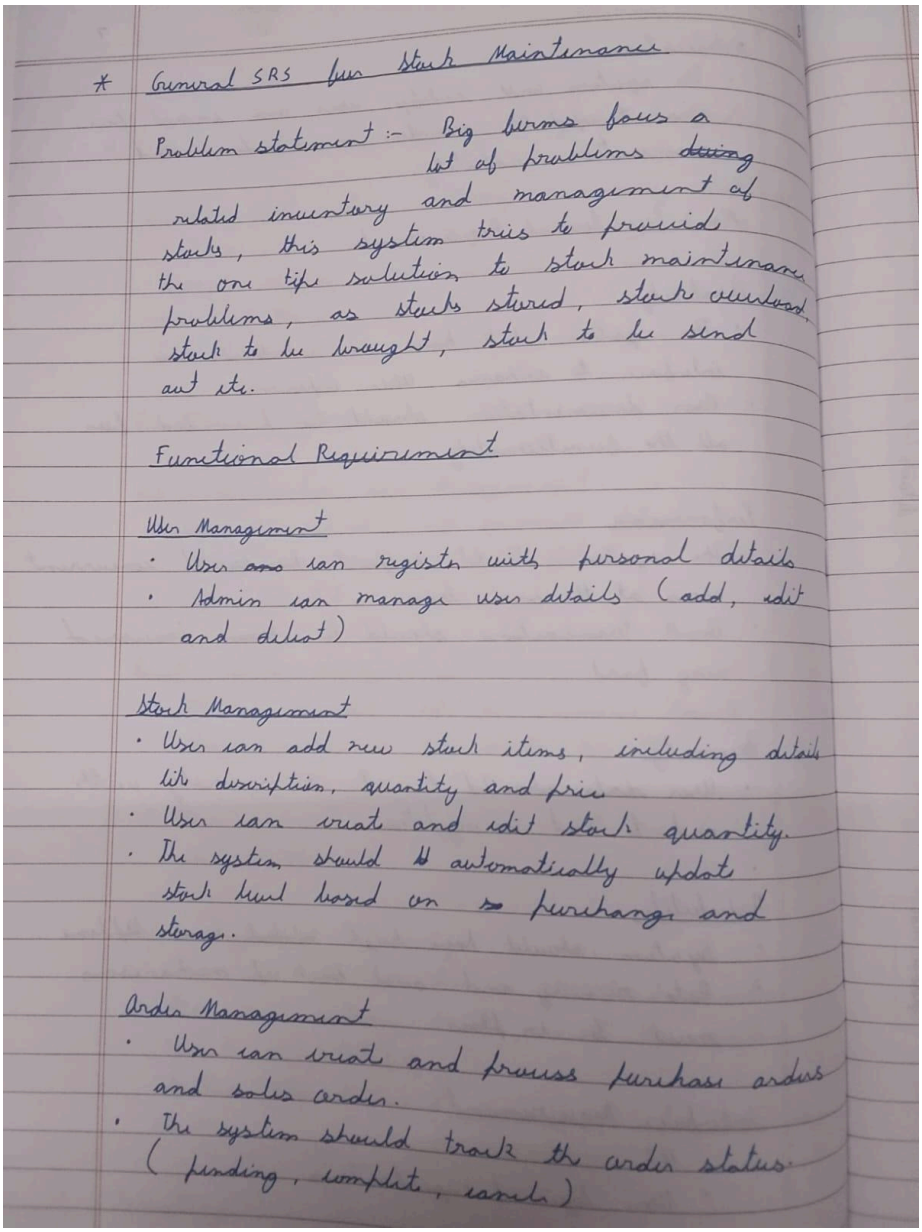
The activity diagram outlines the workflow for processing a passport application, divided into swimlanes for clarity.

- **Swimlanes:**
    - **Applicant**: Submits application, provides documents, and pays fees.
    - **Regional Administrator**: Verifies the application and documents.
    - **Police Department**: Conducts background checks and verification.
- **Flow:**
    0. Applicant logs into the system, submits the application, uploads documents, and makes payment.
    1. The Regional Administrator verifies the documents and forwards the application to the Police Department.
    2. The Police Department conducts a background check and updates the status.
    3. The Regional Administrator approves the application and notifies the Applicant.
    4. The passport is issued, and the Applicant is informed.

5. Stock Maintenance System

5.1 Problem Statement: Businesses face challenges in tracking inventory levels, forecasting demand, and minimizing stock wastage due to inefficiencies in manual stock management systems. Delays in stock replenishment often result in customer dissatisfaction or loss of revenue. An automated stock management system is required to monitor inventory in real time, optimize stock levels, and provide accurate reports to ensure efficient supply chain management.

5.2 SRS:



* General SRS for Stock Maintenance

Problem statement :- Big berms faces a lot of problems during related inventory and management of stocks, this system tries to provide the one tipe solution to stock maintenance problems, as stocks stored, stock overload, stock to be brought, stock to be send out etc.

Functional Requirement

User Management
- User can register with personal details
- Admin can manage user details (add, edit and delet)

Stock Management
- User can add new stock items, including details like description, quantity and price
- User can creat and edit stock quantity.
- The system should automatically updat stock level based on purchase and storage.

Order Management
- User can creat and process purchase orders and sales order.
- The system should track the order status. (pending, complete, cancle)

Suffly Management

- User can add, edit and manage the suffly info.
- Use can associat with the specific suffly.

Reporting

- The system should generate reports on stock levels, sales and suffling informations.
- User should be able to filter reports by dat range, item, category and more.

Notifications

- The system will notify user via email for account ~~updats as~~ updat, status of the stock, sales and purchas history.

Non-functional requirement

Usability

- The system shall have an intuitive user interface to enhance user experience
- User documentation should be provided for all the functionality

~~Performance~~

5.3 Class Diagram:

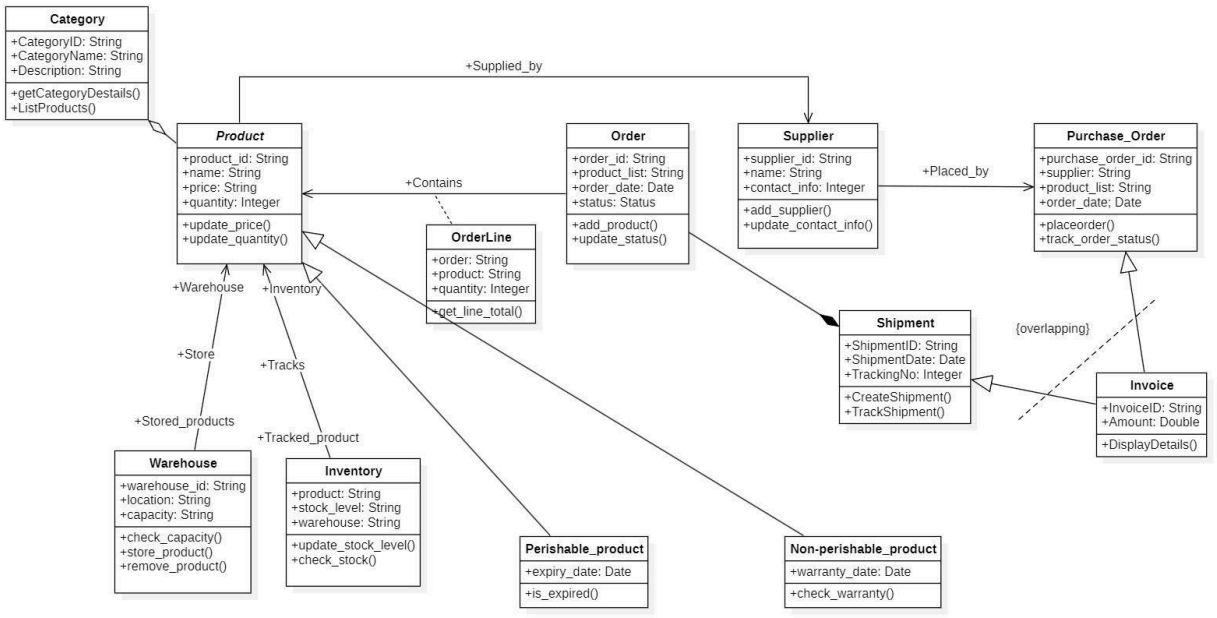


Fig 5.3.1: Class Diagram

Description:

This class diagram represents the key entities of the stock maintenance system and their relationships.

- **Classes:**
    - **Product Class**: Represents a product with attributes like product ID, name, price, and quantity.
        - **Perishable Product Class**: Subclass of Product with additional attributes like expiration date and storage requirements.
        - **Non-Perishable Product Class**: Subclass of Product without special storage needs.
    - **Category Class**: Represents the category of a product, with attributes like category ID and name.
    - **Warehouse Class**: Represents the storage location of products, with attributes like warehouse ID, location, and capacity.

- **Inventory Class**: Manages stock levels and tracks product details, associating **Product** with **Warehouse**.

- **Relationships:**
  - **Product** is aggregated with **Category**, **Warehouse**, and **Inventory** to represent its categorization, storage, and stock level management.

5.4 State Diagram:

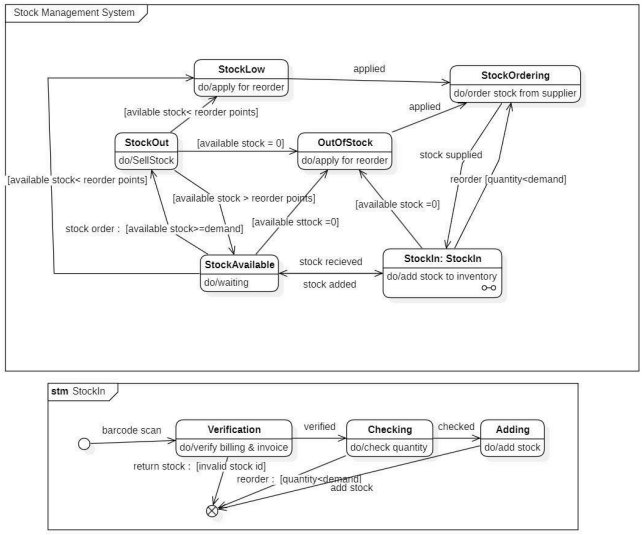


Fig 5.4.1: State Diagram

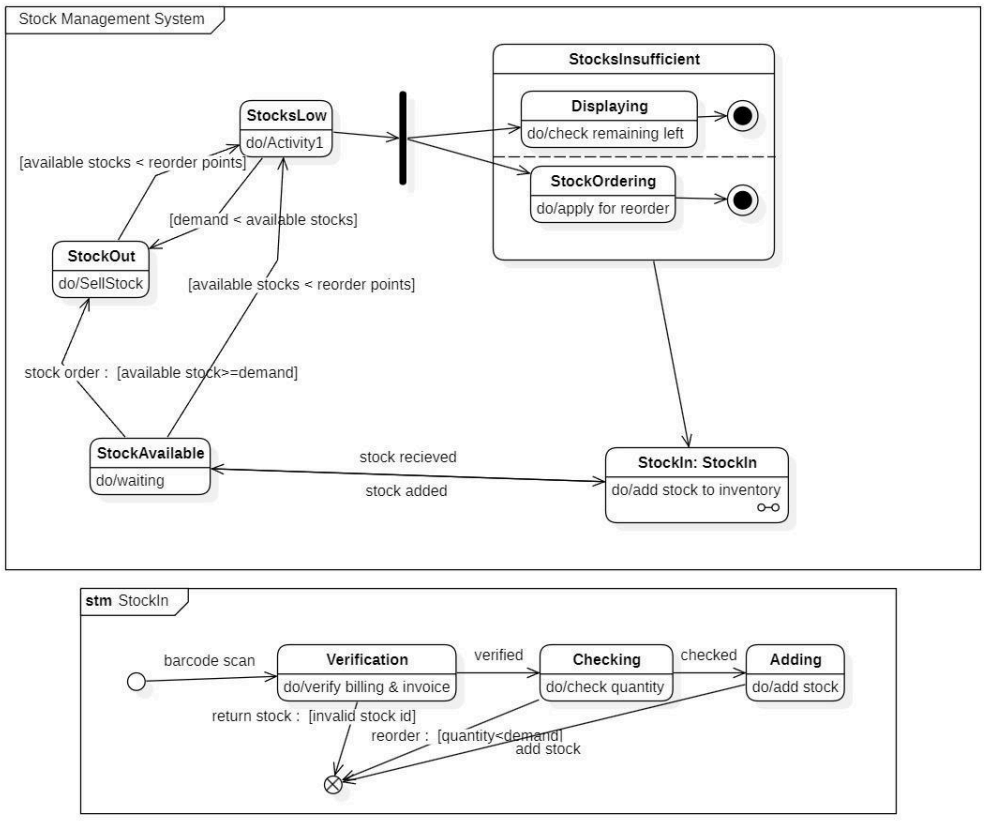


Fig 5.4.2: Advanced State Diagram

Description:

This state diagram highlights the transitions between various states in the stock lifecycle.

- **States:**
    - **Low Stock**: Indicates that product stock is below the reorder threshold.
    - **Stock Ordering**: Represents the process of ordering new stock.
    - **Stock Out**: Indicates that stock is being dispatched or sold.
    - **Stock In**: Represents the arrival and addition of stock to the inventory.
    - **Out of Stock**: Occurs when the product is unavailable for sale or dispatch.
    - **Stock Available**: Normal state when adequate stock is present in the inventory.

5.5 Use Case Diagram:

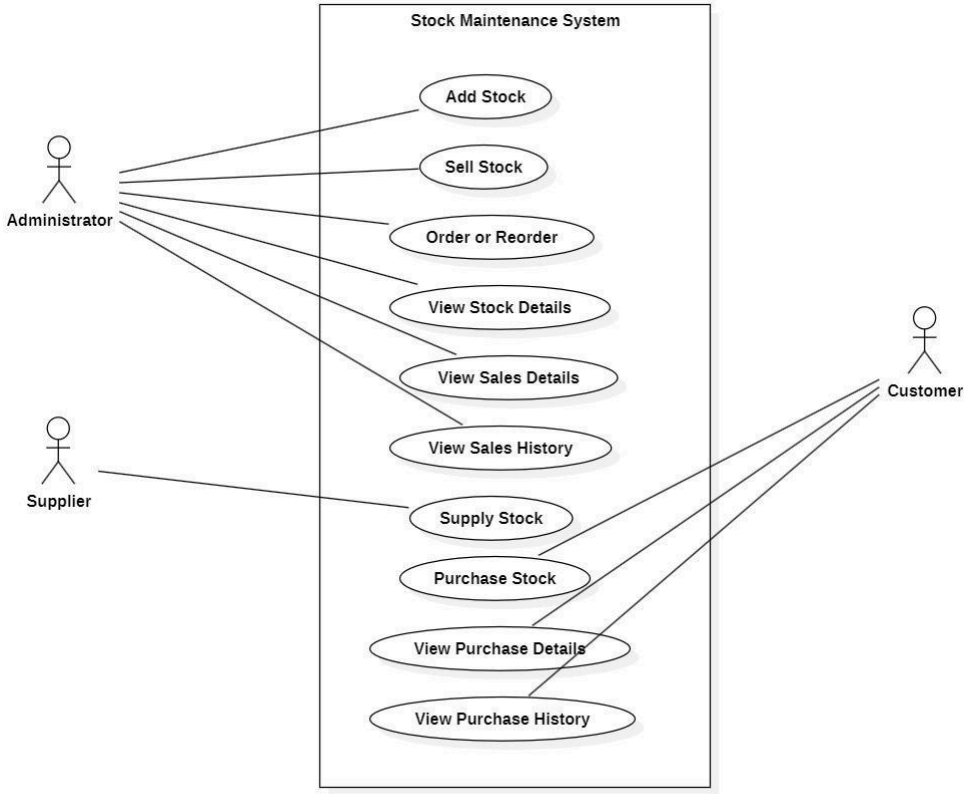


Fig 5.5.1: Use Case Diagram

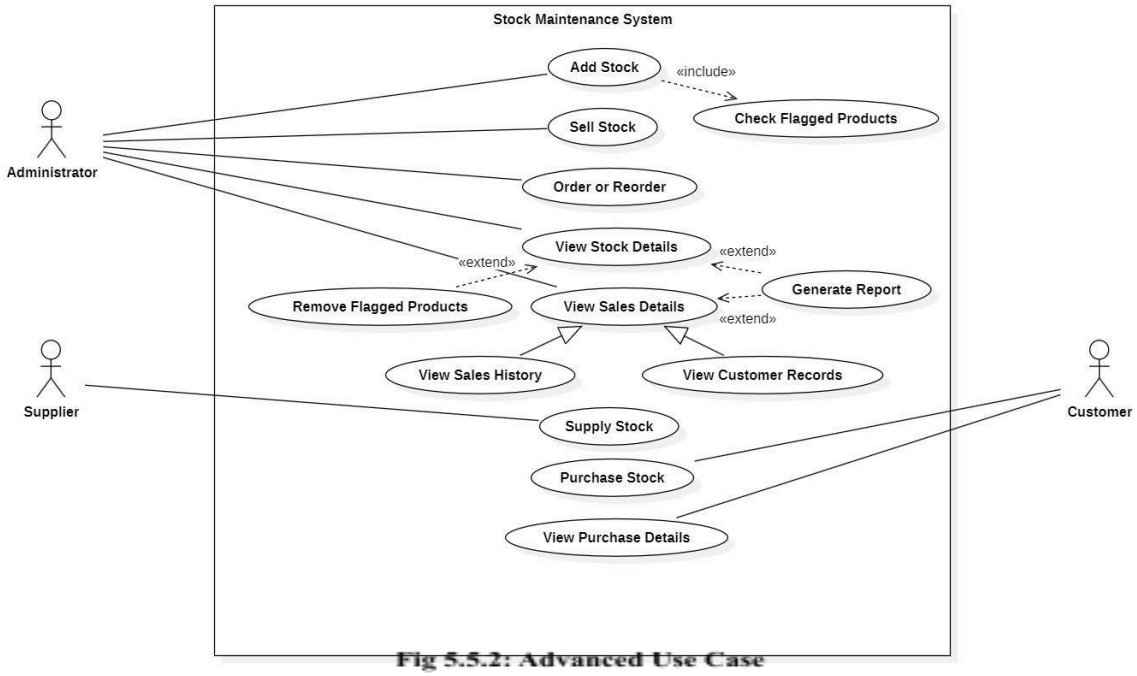


Fig 5.5.2: Advanced Use Case Diagram

Description:

The use case diagram illustrates the interactions between actors and system functionalities.

- **Actors:**
    - **Supplier**: Supplies stock to the warehouse.
    - **Admin**: Manages inventory and oversees operations like viewing stock and sales.
    - **Customer**: Purchases stock or products.
- **Use Cases:**
    - **Add Stock**: Admin adds new stock to the inventory.
    - **Sell Stock**: Stock is sold to customers.
    - **Order Stock**: Admin places an order for new stock with suppliers.
    - **View Stock Details**: Admin views detailed stock information, including quantities and categories.
    - **View Sales**: Admin tracks sales performance.
    - **Supply Stock**: Supplier delivers stock to the warehouse.
    - **Purchase Stock**: Customer buys products from the inventory.

5.6 Sequence Diagram:

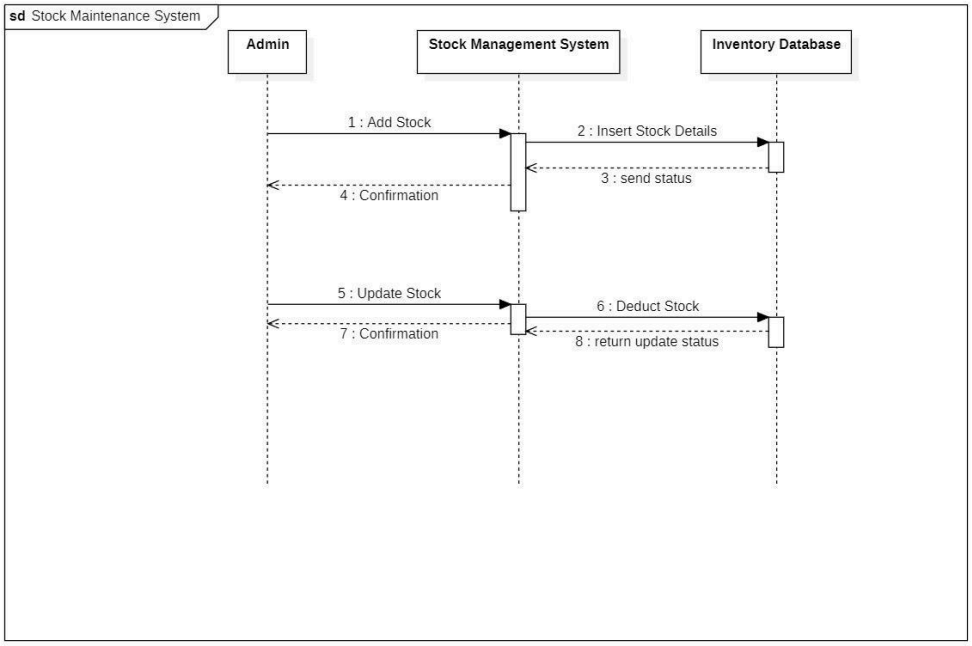


Fig 5.6.1: Sequence Diagram

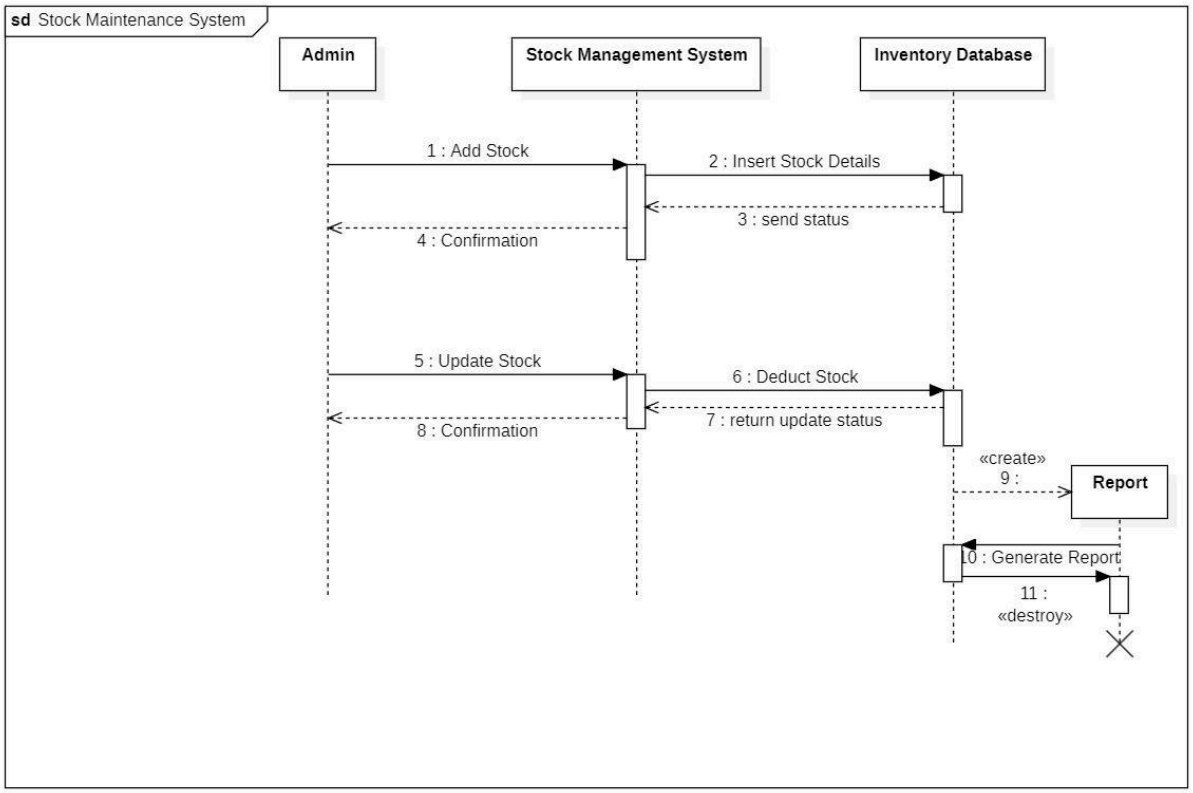


Fig 5.6.2: Advanced Sequence Diagram

Description:

The sequence diagram illustrates the flow of interactions for generating a stock report.

- **Objects:**
    - **Admin**: Initiates the request for stock management or reporting.
    - **Stock Maintenance System**: Processes requests related to inventory.
    - **Inventory Database**: Stores all stock and transaction details.
    - **Report (Transient Object)**: Temporarily represents the generated stock or sales report.
- **Flow:**
    0. Admin requests a stock or sales report via the system.
    1. Stock Maintenance System queries the Inventory Database for relevant data.
    2. Inventory Database retrieves the requested information.
    3. Stock Maintenance System generates a transient **Report** object.
    4. Admin views or exports the report.

5.7 Activity Diagram

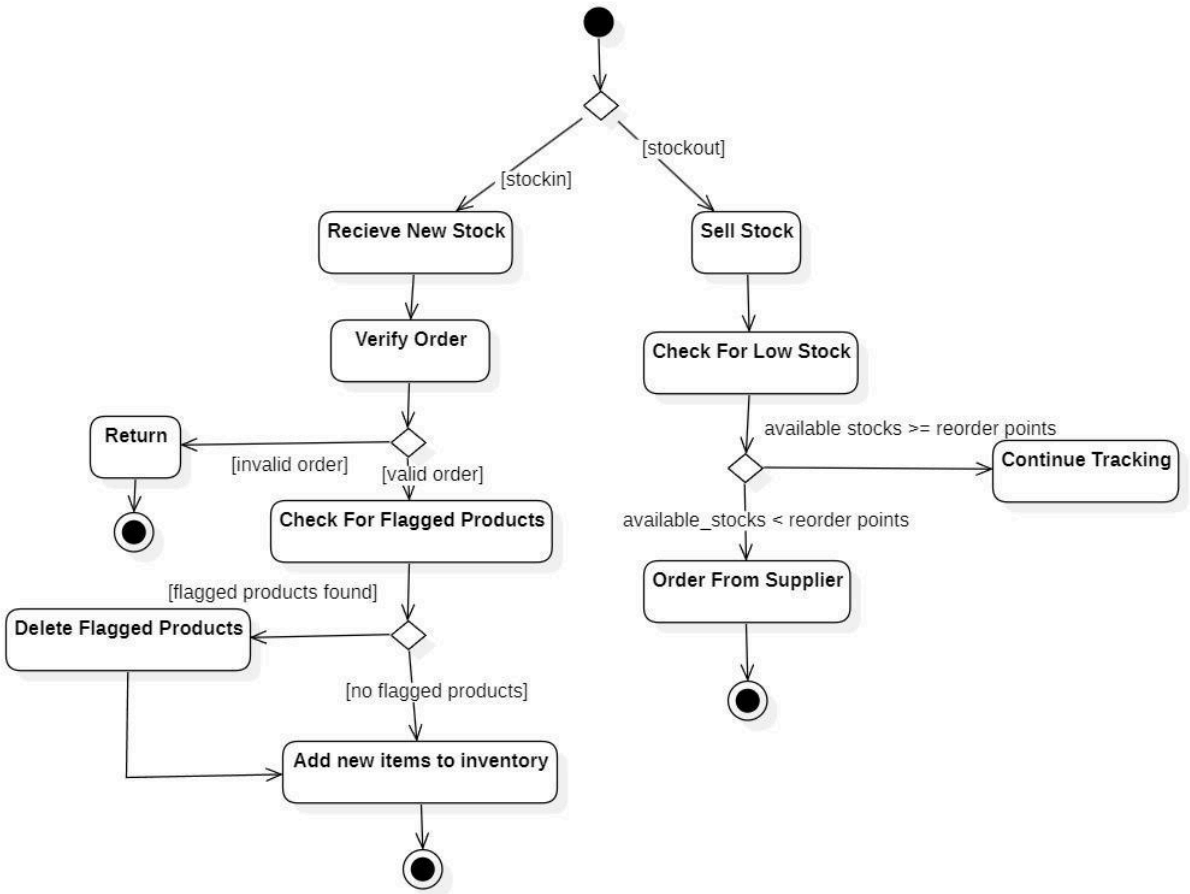


Fig 5.7.1: Activity Diagram

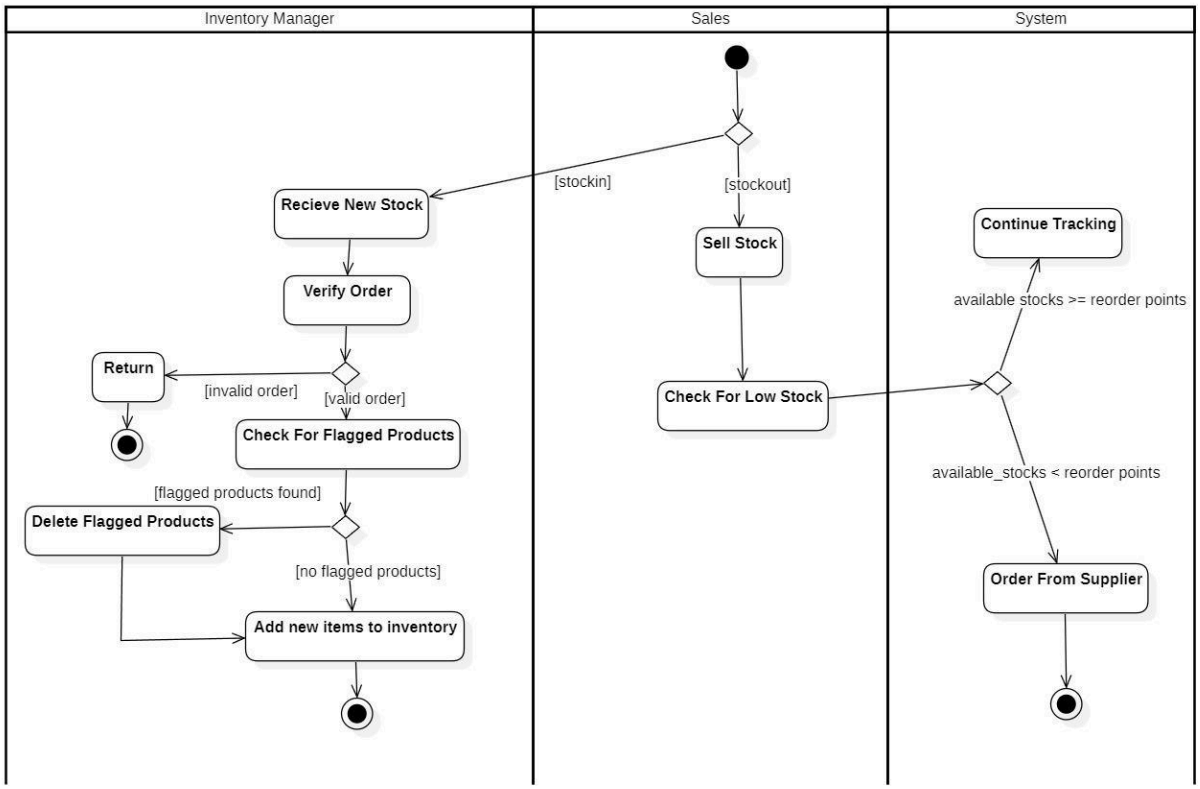


Fig 5.7.2: Advanced Activity Diagram

Description:

The activity diagram represents the workflow for managing inventory and sales.

- **Swimlanes:**
    - **Inventory Manager**: Handles stock updates, reorders, and inventory monitoring.
    - **Sales**: Processes sales transactions and updates stock levels.
    - **System**: Automates stock monitoring, order placements, and report generation.
- **Flow:**
    0. Inventory Manager logs into the system and views stock levels.
    1. If stock is low, a reorder is placed, and new stock is added upon arrival.
    2. Sales transactions are processed, reducing stock in the inventory.
    3. System generates sales and stock reports, which the Inventory Manager reviews.