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:

ANJAN C
1BM22CS044

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

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 Anjan C (1BM22CS044) during the 5th Semester Oct 24-Jan2025.

Signature of the Faculty Incharge:

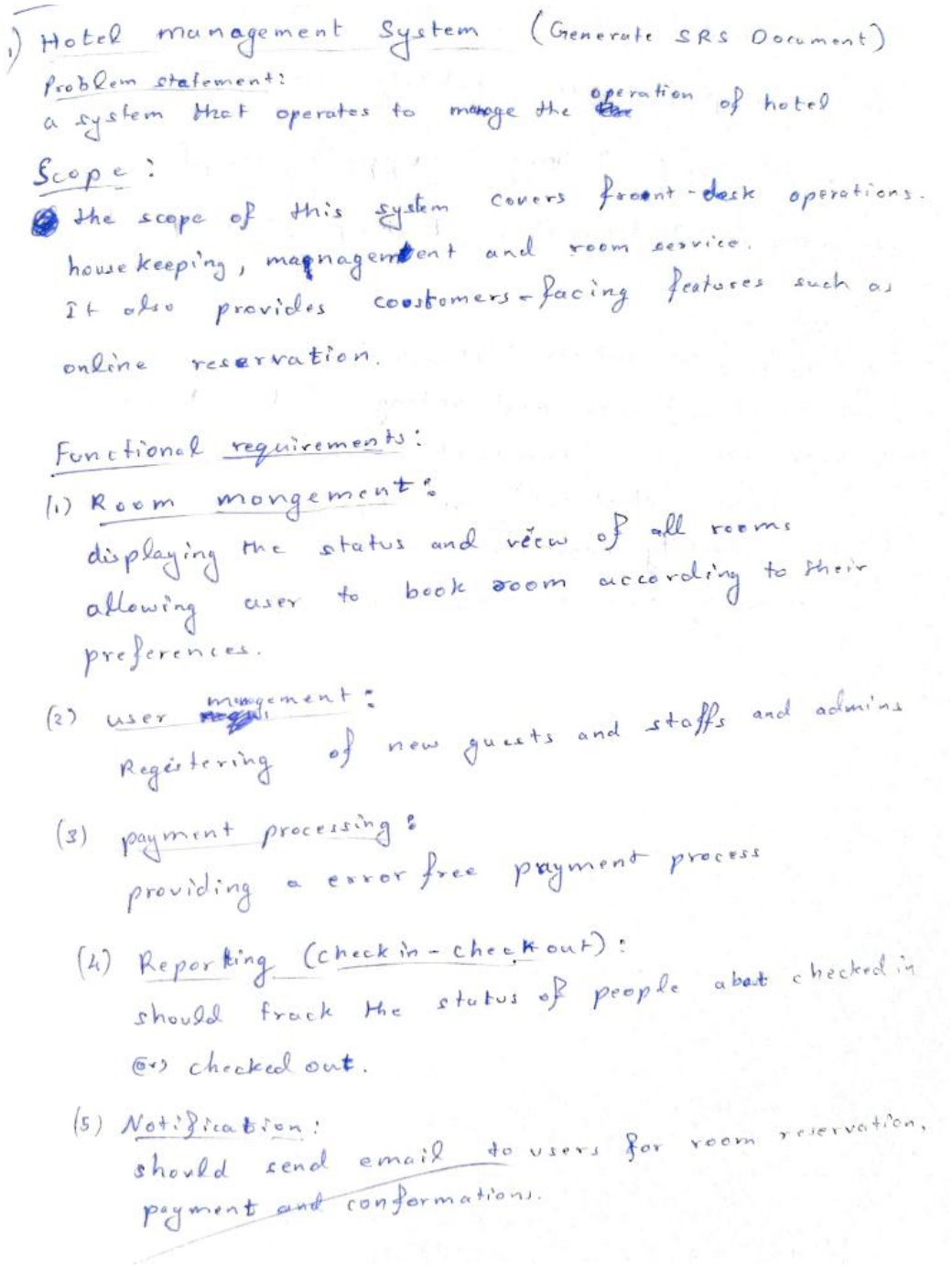
Dr Laxmi Neelima
Department of Computer Science and Engineering
B.M.S. College of Engineering, Bangalore

Table of Contents

1. Hotel Management System

1.1 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.

1.2 SRS-Software Requirements Specification


1) Hotel management System (Generate SRS Document)

Problem statement:
a system that operates to manage the operation of hotel

Scope :
the scope of this system covers front-desk operations. housekeeping, management and room service. It also provides coustomers-facing features such as online reservation.

Functional requirements:

(1) Room mongement:
displaying the status and view of all rooms
allowing user to book room according to their preferences.

(2) user management:
Registering of new guests and staffs and admins

(3) payment processing:
providing a error free payment process

(4) Reporting (check in - check out):
should track the status of people about checked in & checked out.

(5) Notification:
should send email to users for room reservation, payment and conformations.

Fig 1.2.1

Non-functional:

(i) product: performance, usability, scalability

(ii) Organizational: Training, support, compliance

(iii) External: Integration, Backup, interoperability

Domain:

(i) Comply with industrial standards:
Room classification and ratings (star rating)

(ii) user specifics industrial concept:
"check-in", "check-out", "room type".

(iii) meet safety and security standards:
Fire safety and emergency response plans

Fig 1.2.2

1.3 Class Diagram

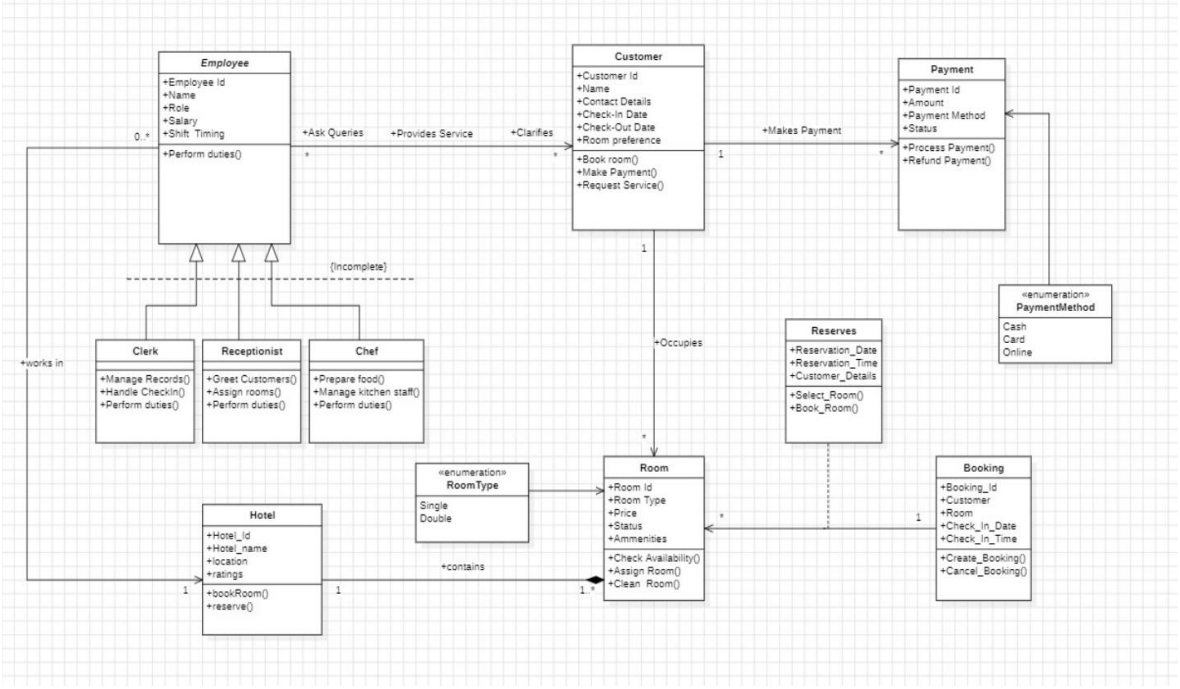


Fig 1.3.1

The class diagram represents the structure of a **Hotel Management System**, detailing the relationships and interactions among various entities. The primary entities include Employee, Customer, Room, Booking, Reserves, Payment, and Hotel. The Employee class serves as the parent for roles such as Clerk, Receptionist, and Chef each having specific duties like managing records, assigning rooms, preparing food, assisting guests, and maintaining room cleanliness. The Customer class holds details such as name, contact information, check-in/out dates, and room preferences, along with methods for booking rooms, making payments, and requesting services. The Room class defines attributes like room type (Single or Double), price, status, and amenities, and includes methods for checking availability, assigning, and cleaning rooms.

The Booking class manages the room booking process with details such as booking ID, customer, room, and check-in time, while the Reserves class handles room reservations and facilitates the selection and booking of rooms. The Payment class manages financial transactions, supporting various payment methods (cash, card, or online) and processes such as payment and refund handling.

1.4 State Diagram

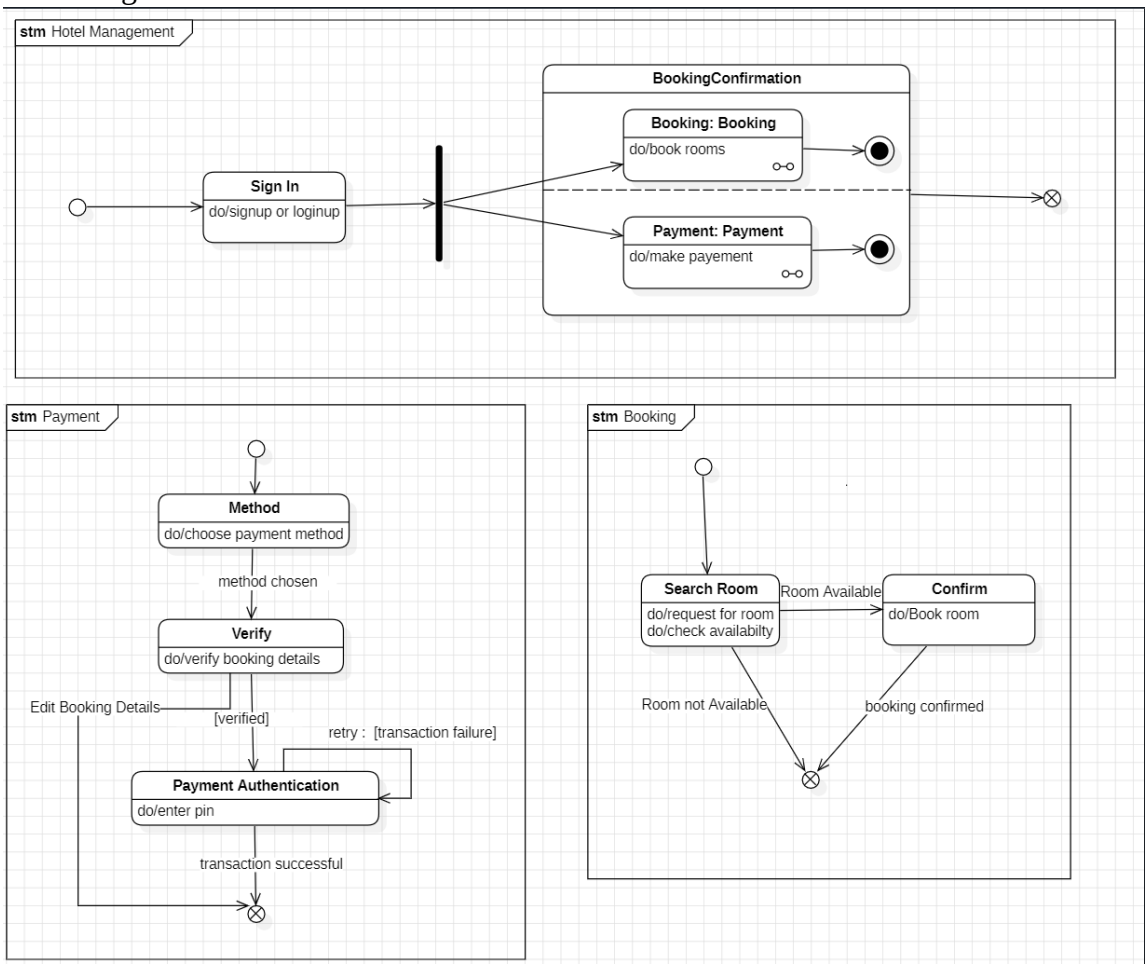


Fig 1.4.1

This state machine diagram represents the workflow of a **Hotel Management System**, focusing on the processes of **booking** and **payment**. The process begins with the **Sign In** state, where users either sign up or log in to access the system. Once authenticated, the workflow branches into two parallel processes—**Booking** and **Payment**—which are managed within the **BookingConfirmation** composite state. The **Booking** process allows users to **Search Room**, check availability, and **Confirm** the booking if a room is available. If no rooms are available, the process terminates. Simultaneously, the **Payment** process involves selecting a payment **Method**, **Verifying** booking details, and proceeding to **Payment Authentication**, where users enter a PIN to complete the transaction. In case of failure, the system allows retries or editing booking details. The workflow concludes when the payment is successful, finalizing the booking confirmation. This diagram effectively models the interactions and transitions between states, covering possible scenarios such as retries, failures, and successful completions within the hotel booking system.

1.5 Use Case Diagram

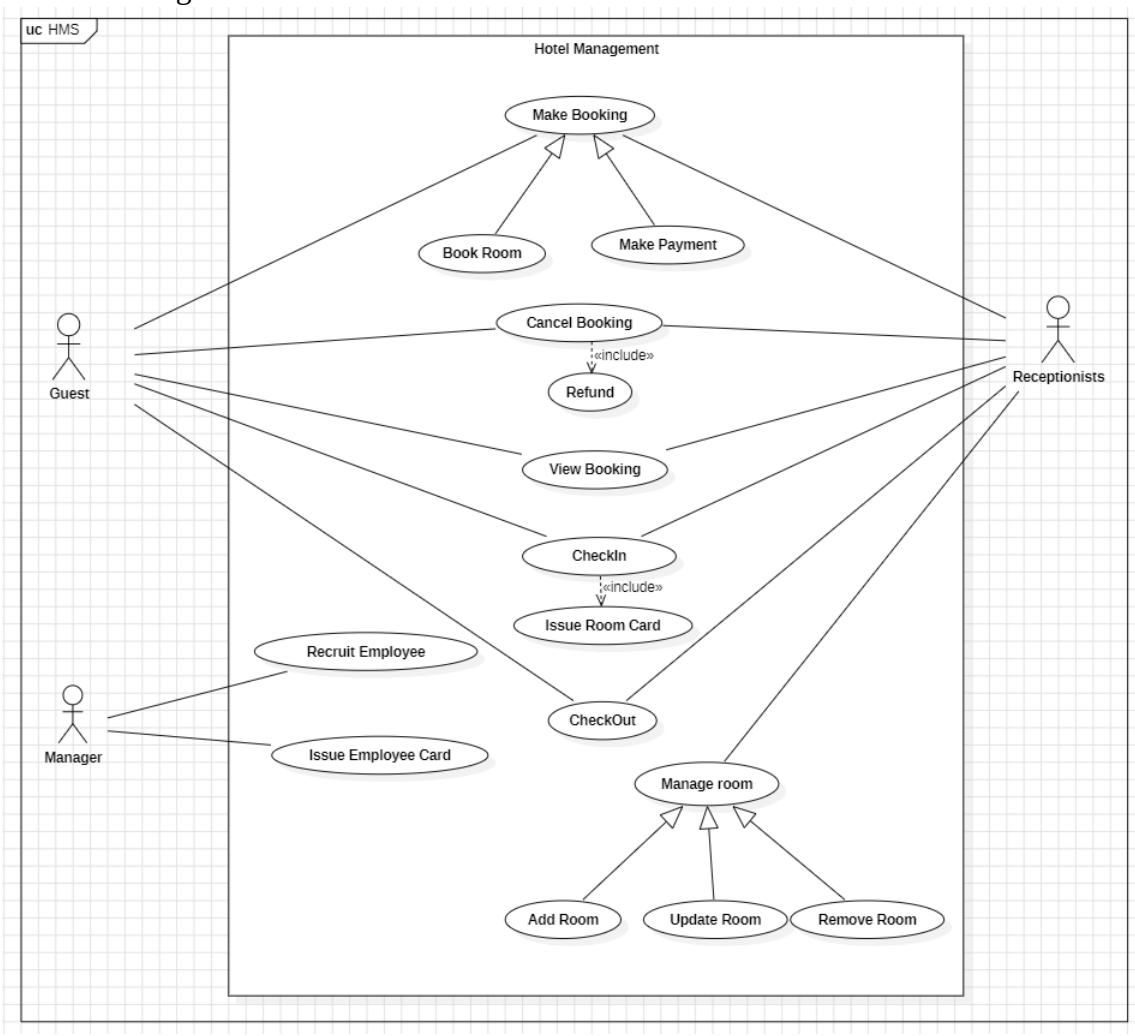


Fig 1.5.1

The use case diagram illustrates the functionality of a Hotel Management System by detailing the interactions between different actors and system processes. The primary actors include **Guests**, **Receptionists**, and a **Manager**. Guests can perform actions such as booking rooms, making payments, canceling bookings, requesting refunds, viewing bookings, checking in, and checking out. Receptionists facilitate these operations by assisting with room bookings, issuing room cards, and managing the check-in and check-out processes. The **Manager** has additional responsibilities like recruiting employees and issuing employee cards. The system is designed to handle cancellations and refunds effectively while maintaining efficient guest services. This diagram provides a clear overview of the roles and responsibilities within the hotel management workflow.

1.6 Sequence Diagram

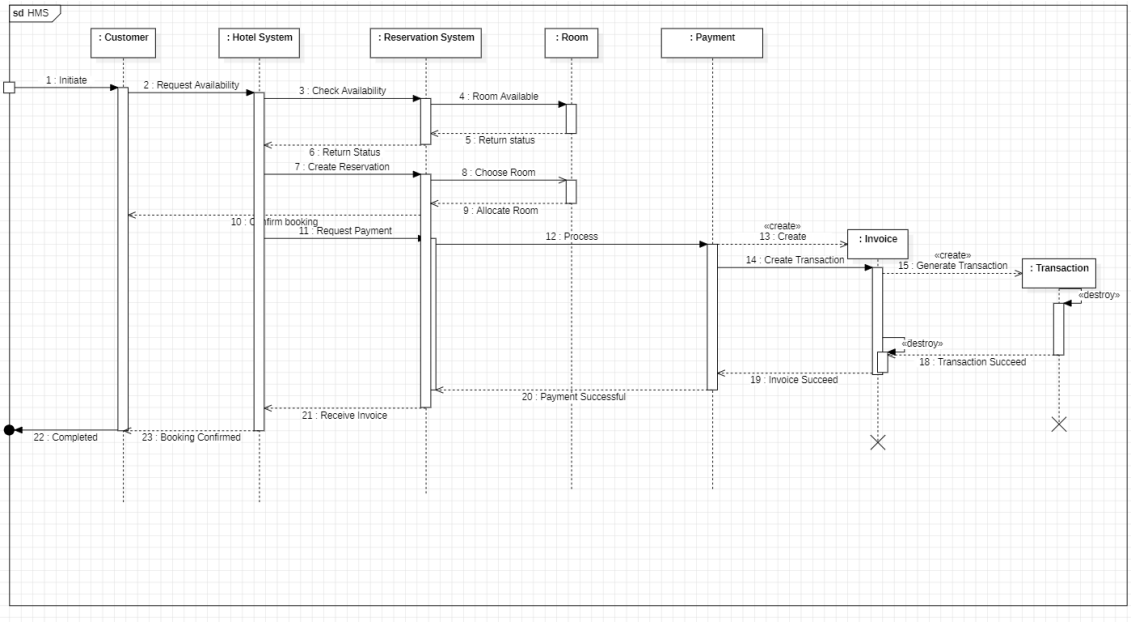


Fig 1.6.1

This sequence diagram illustrates the workflow of a **Hotel Management System (HMS)**, detailing the interactions between various components involved in the booking and payment process. The process begins with the **Customer** initiating a request to check room availability, which is forwarded by the **Hotel System** to the **Reservation System**. The **Reservation System** verifies availability by consulting the **Room** system and returns the status. If rooms are available, a reservation is created, and a room is allocated. The **Hotel System** then confirms the booking and sends a payment request to the **Payment** component. The **Payment** system processes the transaction by generating an **Invoice** and creating a **Transaction**. Once the transaction is successfully completed, both the **Invoice** and **Transaction** objects are destroyed, and the payment status is updated. The **Invoice** is sent back to the **Hotel System**, which informs the **Customer** of the booking confirmation. The process concludes with the system marking the operation as **Completed**. This diagram effectively captures the sequence of actions, including availability checks, reservation creation, payment handling, and final confirmation, ensuring a streamlined booking process.

1.7 Activity Diagram

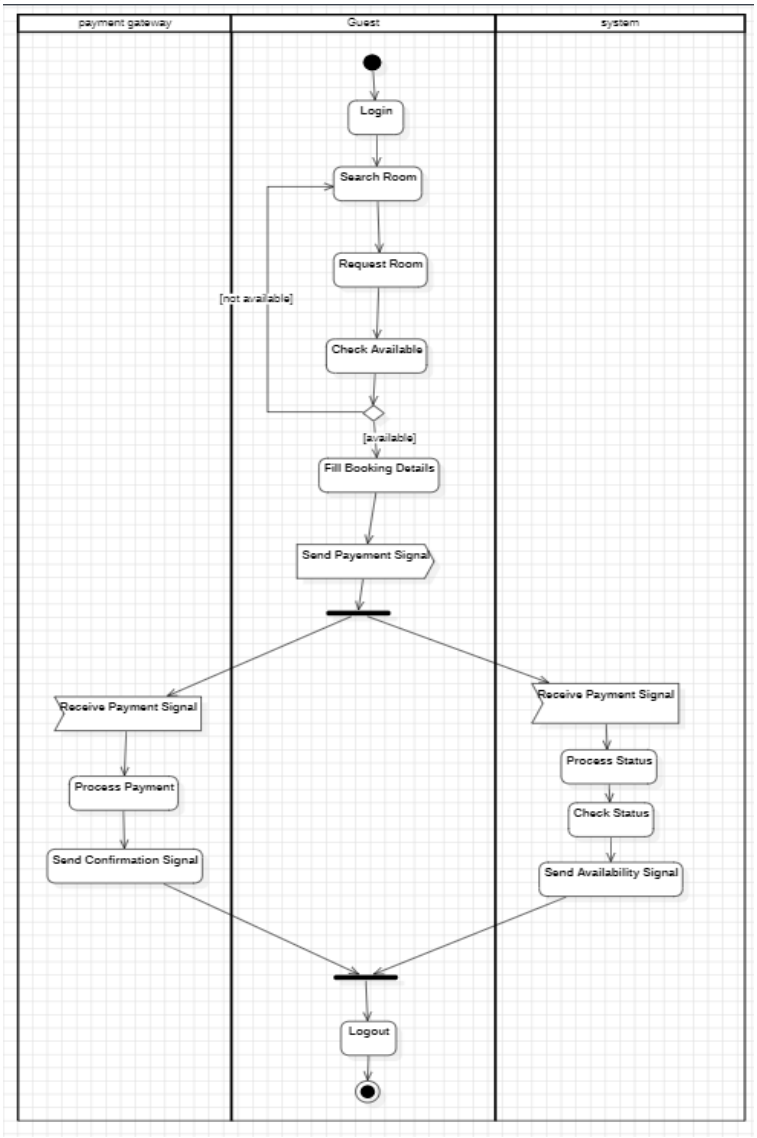


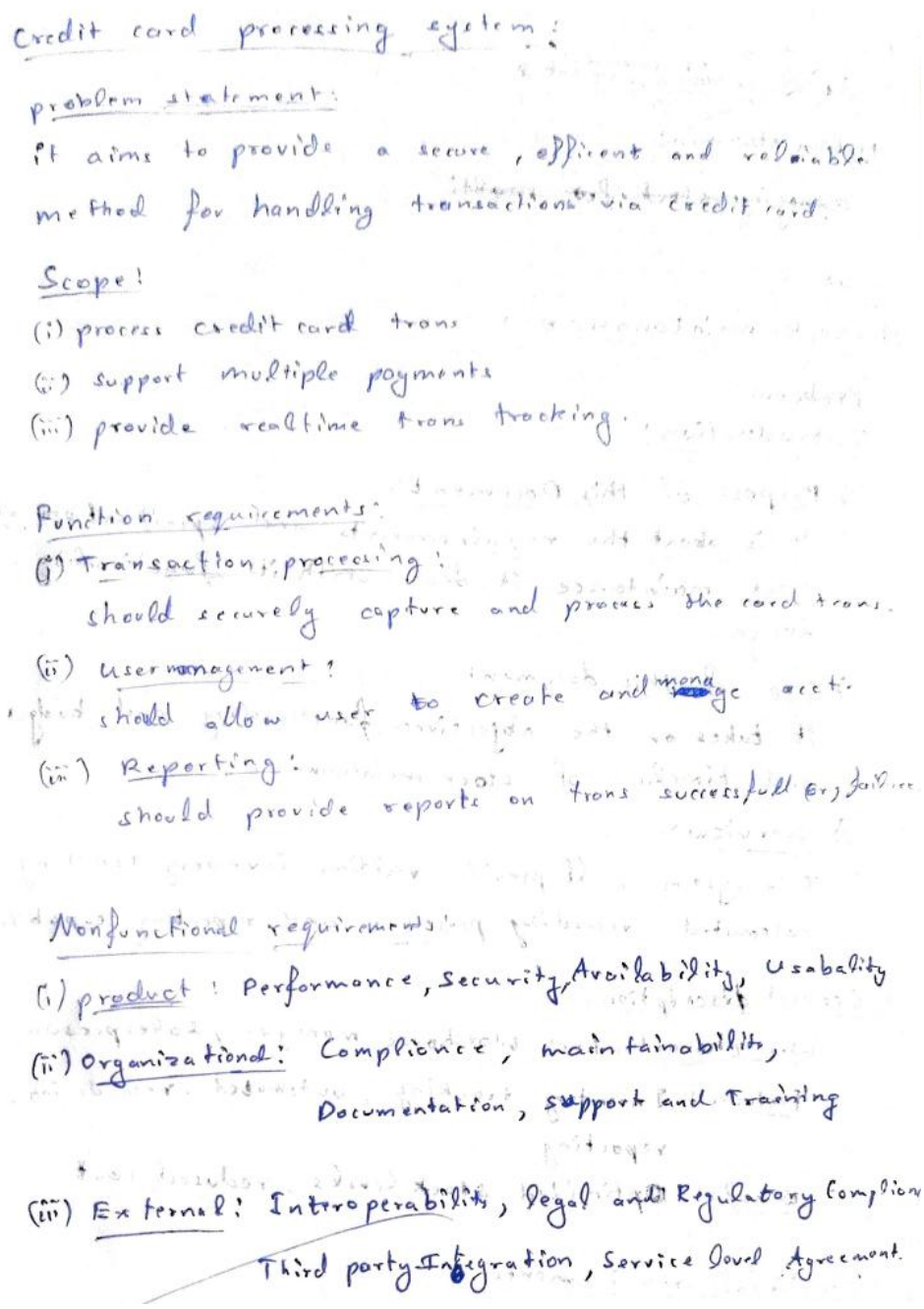
Fig 1.7.1

This activity diagram depicts the activity flow for a hotel room booking process. It involves three participants: the guest, the system, and the payment gateway. The guest begins by logging in and searching for available rooms. Upon finding a suitable room, they request it, and the system checks its availability. If available, the guest fills in booking details and sends a payment signal. This triggers parallel actions: the payment gateway processes the payment and sends a confirmation signal if successful, while the system simultaneously changes the room's status to booked and sends an availability signal. Finally, upon receiving both signals, the process concludes with the guest logging out. This diagram illustrates the sequential and parallel activities involved in a typical online hotel room booking scenario.

2. Credit Card Processing System

2.1 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.

2.2 SRS-Software Requirements Specification



Credit card processing system:

problem statement:
it aims to provide a secure, efficient and reliable method for handling transactions via credit card.

Scope:
(i) process credit card trans
(ii) support multiple payments
(iii) provide realtime trans tracking.

Function requirements:
(i) Transaction processing:
should securely capture and process the card trans.
(ii) user management:
should allow user to create and manage acct.
(iii) Reporting:
should provide reports on trans successfull or, fail...

Nonfunctional requirements:
(i) product: Performance, Security, Availability, Usability
(ii) Organizational: Compliance, maintainability, Documentation, support and Training
(iii) External: Intreoperability, legal and Regulatory Complian Third party Integration, Service level Agreement.

Fig 2.2.1

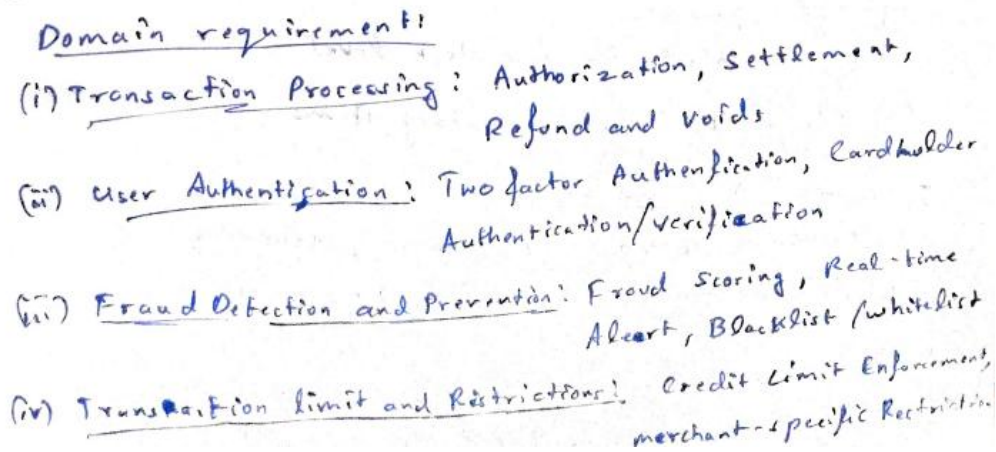


Fig 2.2.2

2.3 Class Diagram

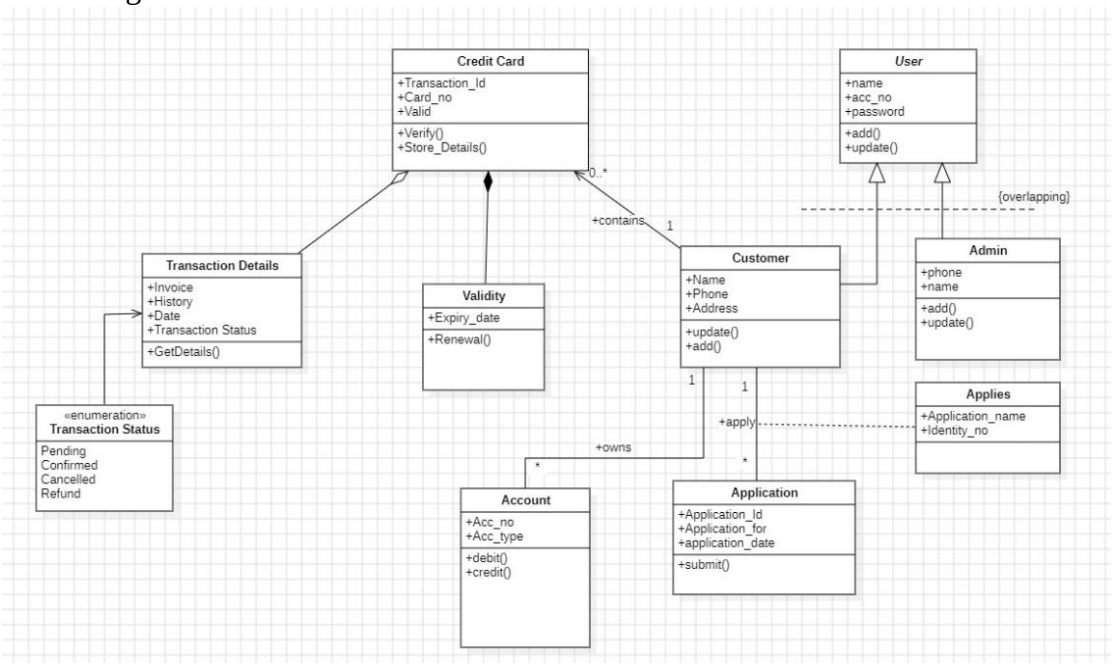


Fig 2.3.1

This UML class diagram models a credit card processing system. It shows that a Transaction uses a Credit Card and contains Transaction Details. Each Credit Card has a Validity status. A Customer can have multiple Accounts and submit Applications for credit cards, which are also linked to an Account. An Admin manages Customers and their Applications. The diagram also lists attributes for each class, like card number for Credit Card and transaction date for Transaction, as well as operations like verify() for Credit Card, debit() and credit() for Account, and submit() for Application. This structure defines the core components and their interactions within the credit card processing system.

2.4 State Diagram

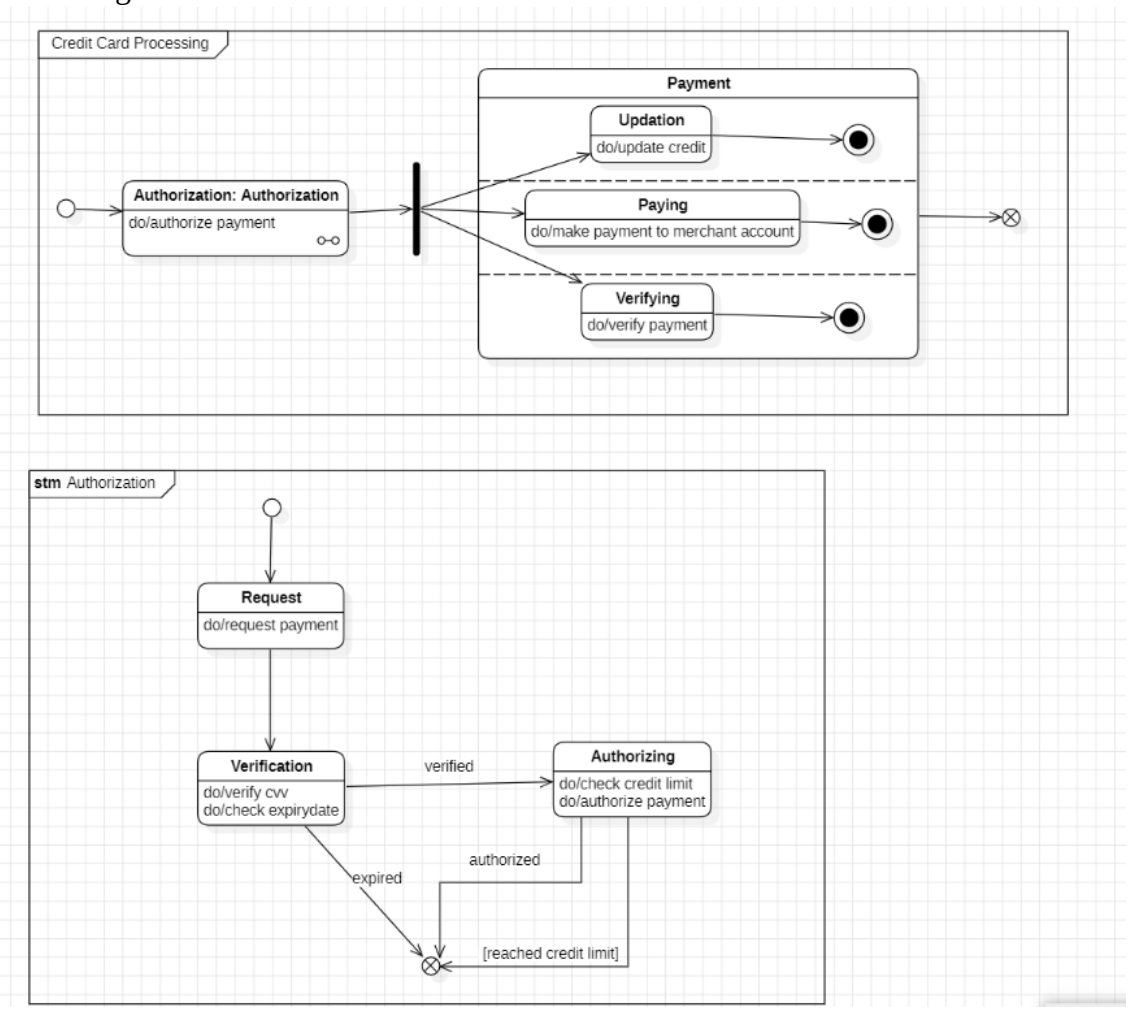


Fig 2.4.1

This state diagram models user interaction with a credit card system. Users begin by logging in, then proceed as either a Customer or Admin. Customers can "Make Payment" (involving card reading, PIN entry, and amount confirmation, leading to either successful payment or transaction denial) or "View Application." Admins can "View Application," which includes searching for applications and subsequently approving or rejecting them, updating the application status. Actions like reading card details or displaying payment invoices are associated with state transitions. The diagram clearly outlines the different paths and possible outcomes for both customer and admin users within the system.

2.5 Use Case Diagram

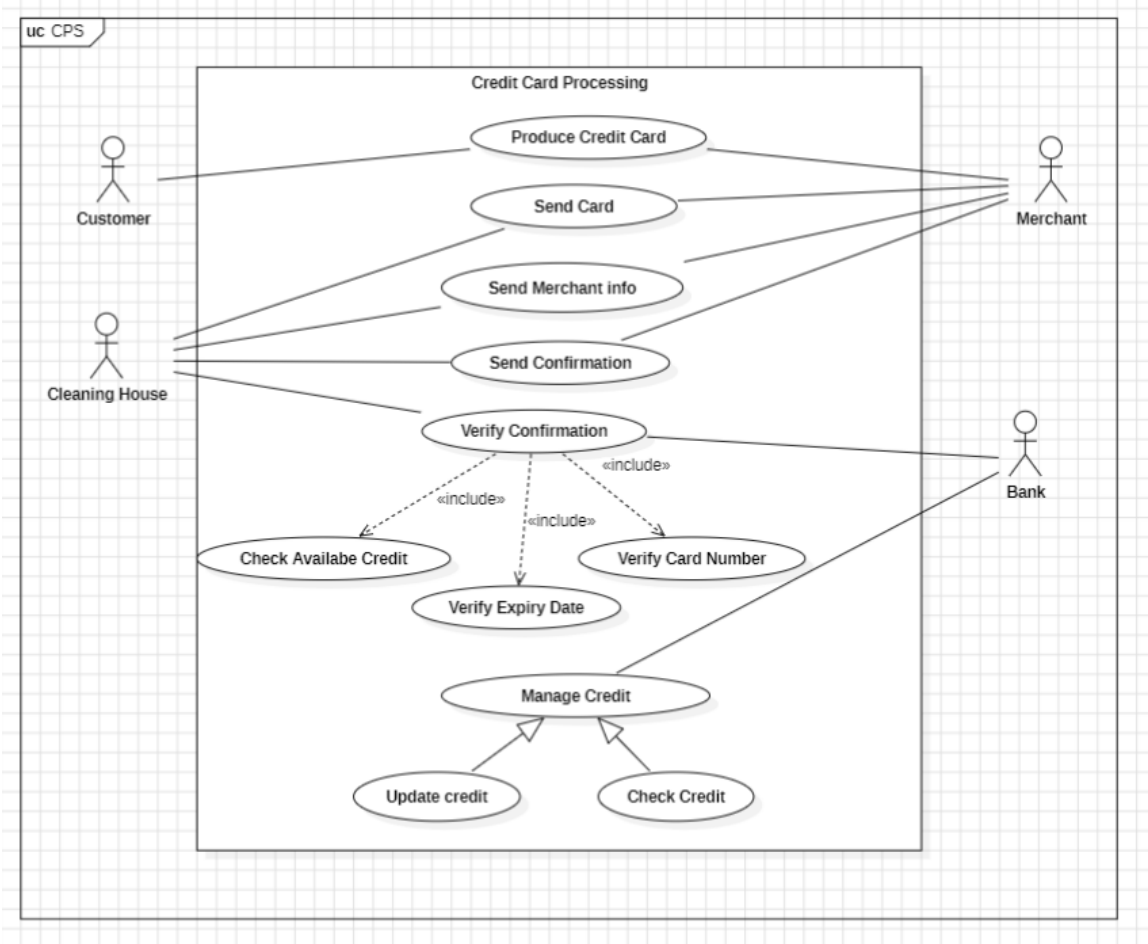


Fig 2.5.1

This use case diagram illustrates interactions within a Credit Card Processing System (CPS). It shows four actors: Customer, Merchant, Cleaning House (likely for transaction reconciliation), and Bank. The primary use case is "Credit Card Processing," which involves several included use cases. The Customer can "Create a Credit Card." The Merchant "Sends Confirmation" of a transaction, which triggers the "Verify Confirmation" use case. This verification process *includes* three sub-processes handled by the Bank: "Check Credit," "Verify Expiry Date," and "Verify Card Number." Finally, the Bank also "Updates Credit" after a transaction. The diagram highlights the dependencies and interactions between the different actors and their respective roles in the credit card transaction process.

2.6 Sequence Diagram

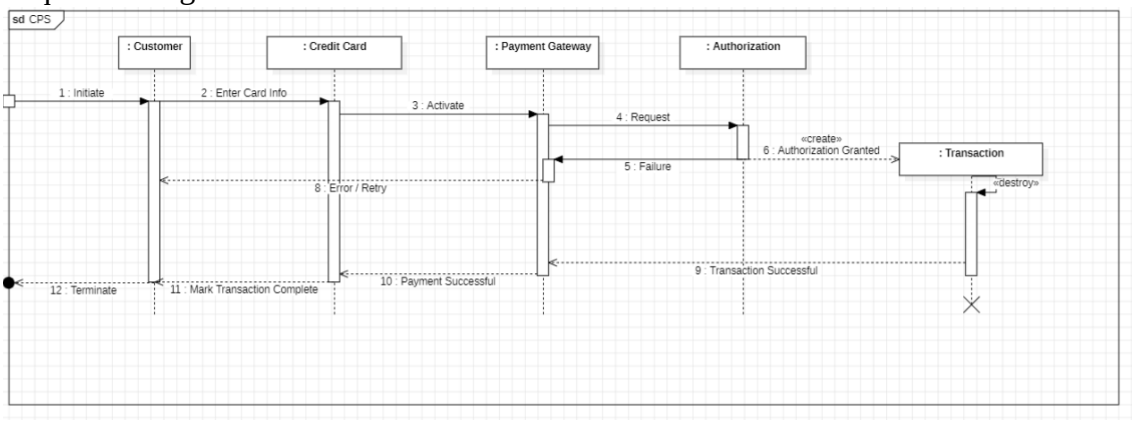


Fig 2.6.1

This sequence diagram represents the process flow of a **Credit Card Payment System (CPS)**, showcasing interactions between various components involved in a payment transaction. The process begins with the **Customer** initiating the transaction (Step 1) and entering their credit card information (Step 2). The **Credit Card** component then activates the payment request (Step 3) and forwards it to the **Payment Gateway** for further processing. The **Payment Gateway** sends a request for authorization (Step 4) to the **Authorization** system.

If the authorization request fails (Step 5), the process provides an option for **Error/Retry** (Step 8). However, if the authorization is granted (Step 6), a **Transaction** object is created to manage the payment process. The **Payment Gateway** confirms the transaction's success (Step 9), and the **Credit Card** system verifies that the payment was successful (Step 10). Following this, the **Customer** is informed that the transaction is complete (Step 11), and the process is terminated (Step 12). Finally, the **Transaction** object is destroyed to indicate the end of its lifecycle.

This diagram effectively outlines the communication between systems, error handling, and transaction finalization, ensuring a secure and streamlined payment process.

2.7 Activity Diagram

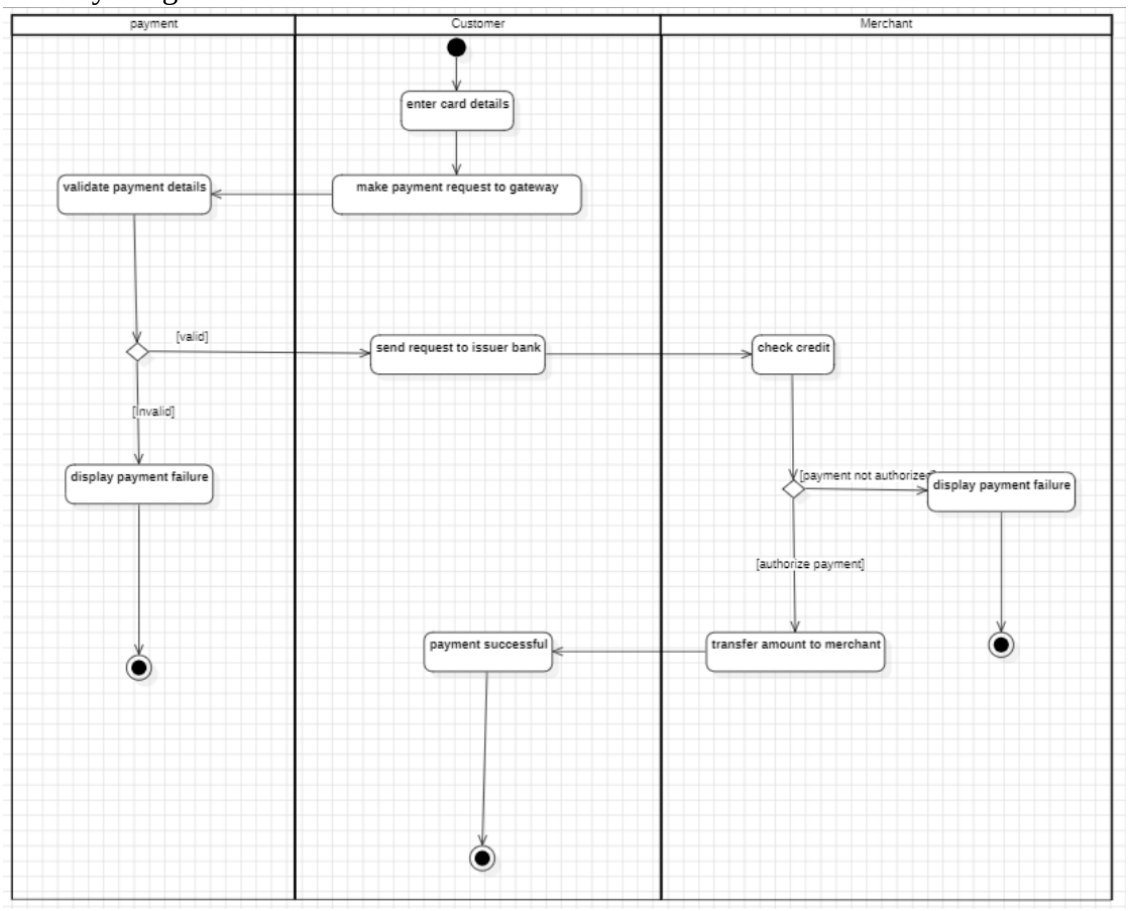


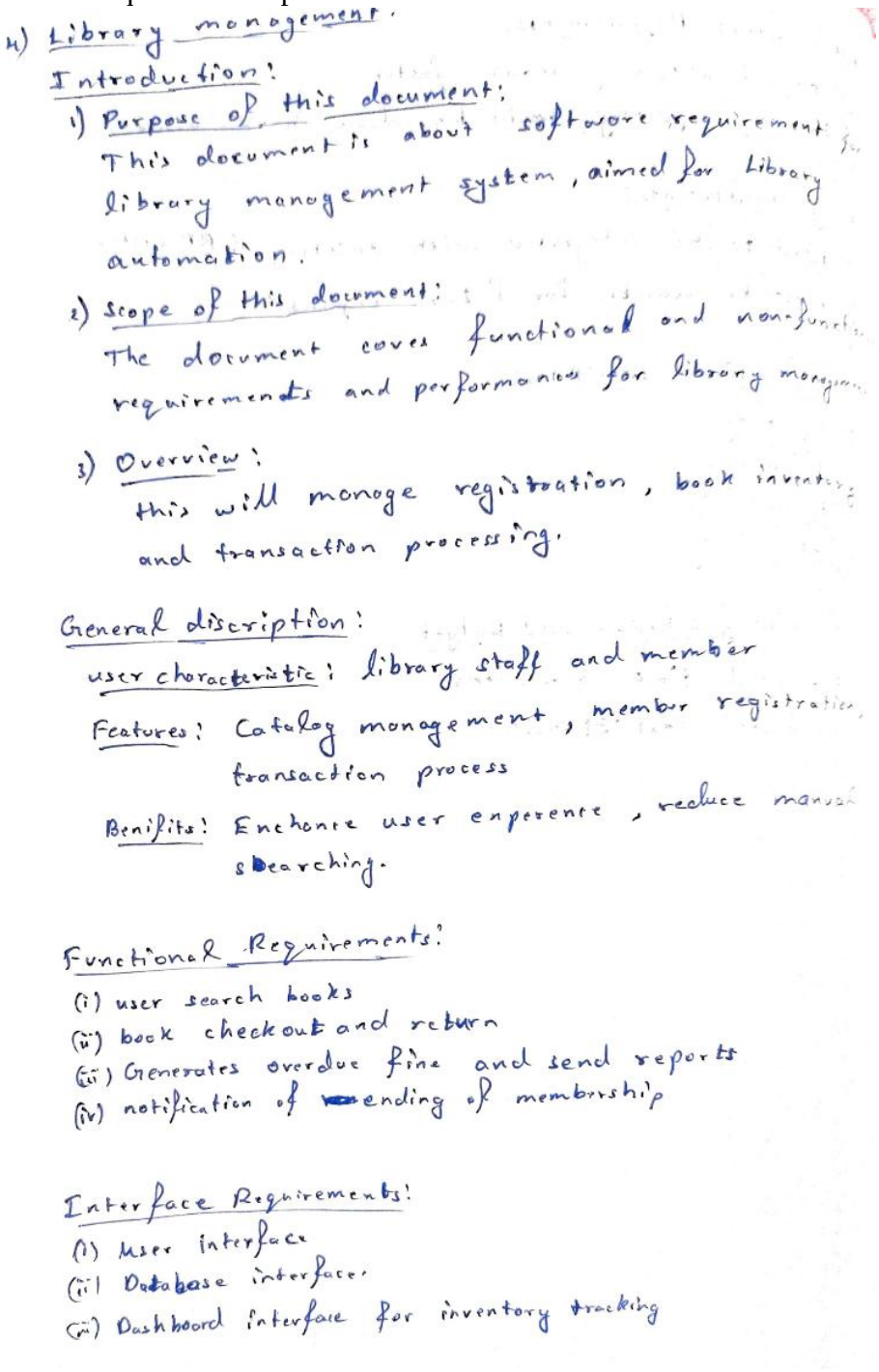
Fig 2.7.1

This activity diagram depicts the flow of a credit card transaction involving a Customer, Payment Gateway, and Merchant. The Customer begins by entering card details and making a payment request. The Payment Gateway then validates these details. If invalid, a payment failure is displayed to the Customer, and the process ends. If the details are valid, the Payment Gateway sends a request to the issuer bank, represented here by the Merchant's action of checking credit availability. If the payment is not authorized, a payment failure is displayed to the Merchant, and the process ends. However, if the payment is authorized, the amount is transferred to the Merchant, and a "payment successful" message is sent back through the Payment Gateway, completing the transaction. This diagram clearly shows the sequential steps and decision points in a credit card transaction, highlighting the interactions between the three parties involved.

3. Library Management System

3.1 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.

3.2 SRS-Software Requirements Specification



4) Library management.

Introduction:

1) Purpose of this document:
This document is about software requirement library management system, aimed for Library automation.

2) Scope of this document:
The document covers functional and non-funct... requirements and performances for library manag...

3) Overview:
this will manage registration, book invent... and transaction processing.

General discription:

user characteristic: library staff and member

Features: Catalog management, member registrati... transaction process

Benifits: Enchance user experence, reduce manu... searching.

Functional Requirements:

(i) user search books
(ii) book checkout and return
(iii) Generates overdue fine and send reports
(iv) notification of ending of membership

Interface Requirements:

(i) User interface
(ii) Database interface.
(iii) Dashboard interface for inventory tracking

Fig 3.2.1

Performance requirements:

(i) responce for searching should be fast
(ii) less error rate
(iii) efficient usage of memory

Design Constraints:

(i) use of open-source database systems
(ii) should cooperate on low resource service

Non functional:

(i) usability
(ii) Portability
(iii) scalability

Preliminary Schedule and Budget:

schedule time: 6 months

estimated cost: ₹ 10 lakh

Fig 3.2.2

3.3 Class Diagram

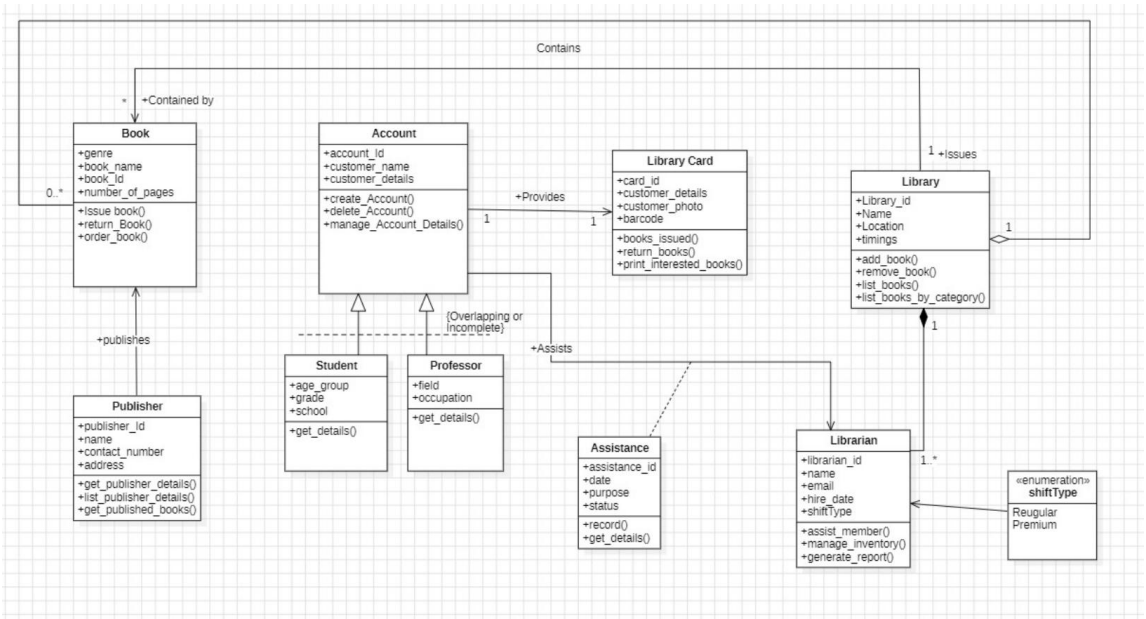


Fig 3.3.1

This UML depicts a class diagram for a Library Management System. It outlines the key entities involved and their relationships. The core entities include Book (with attributes like genre, name, ID, and number of pages, and operations like issuing, returning, and ordering), Account (with customer details and account management operations), Library Card (linking accounts to the library and tracking issued books), and Library (containing books, managing inventory, and handling book lists). These are further connected to Publisher (responsible for publishing books) and two types of library members: Student and Professor, both inheriting from Account. Finally, a Librarian assists members and manages the library, potentially with different ShiftType (Regular or Premium), and an Assistance entity tracks the details of assistance provided by librarians. The diagram illustrates how these entities interact, providing a structured view of the system's data and functionality.

3.4 State Diagram

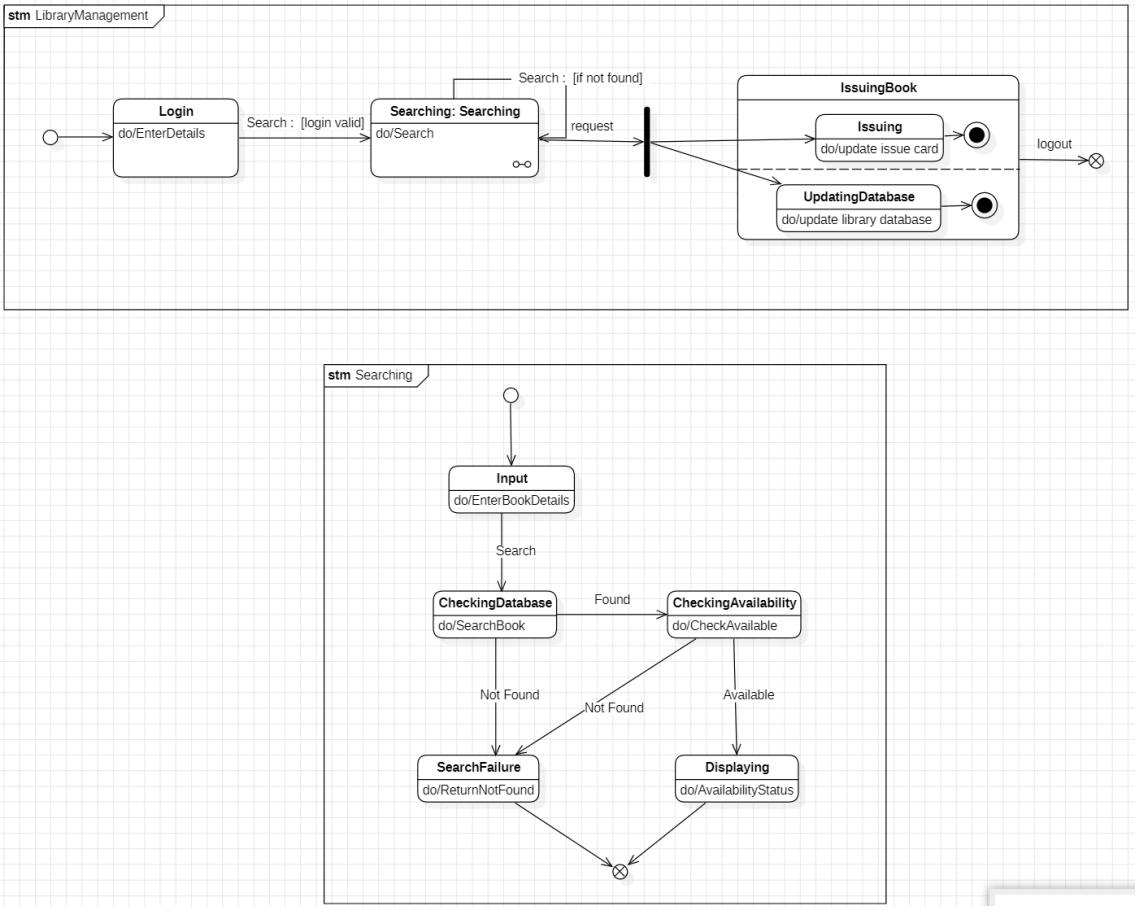


Fig 3.4.1

This state diagram for a Library Management System illustrates the system's dynamic behavior and user interactions. Starting with a login state requiring valid login details, the system branches into two main user roles: Admin and regular User. The Admin Menu allows actions like viewing books, which can lead to removing a specific book (requiring confirmation) or adding a new book via an "Add Book Form". Both removal and addition ultimately return to displaying the updated book list. The User Menu offers options to view the catalog, borrow a book (involving book selection and borrow confirmation), or return a book (with return confirmation). Both borrowing and returning books transition to a "Book Borrowed" or "Book Returned" state, respectively, before returning to the catalog display. Finally, both Admin and User roles can log out, transitioning the system to a logout state and completing the process. This diagram effectively visualizes the various paths users can take within the system and the corresponding state changes.

3.5 Use Case Diagram

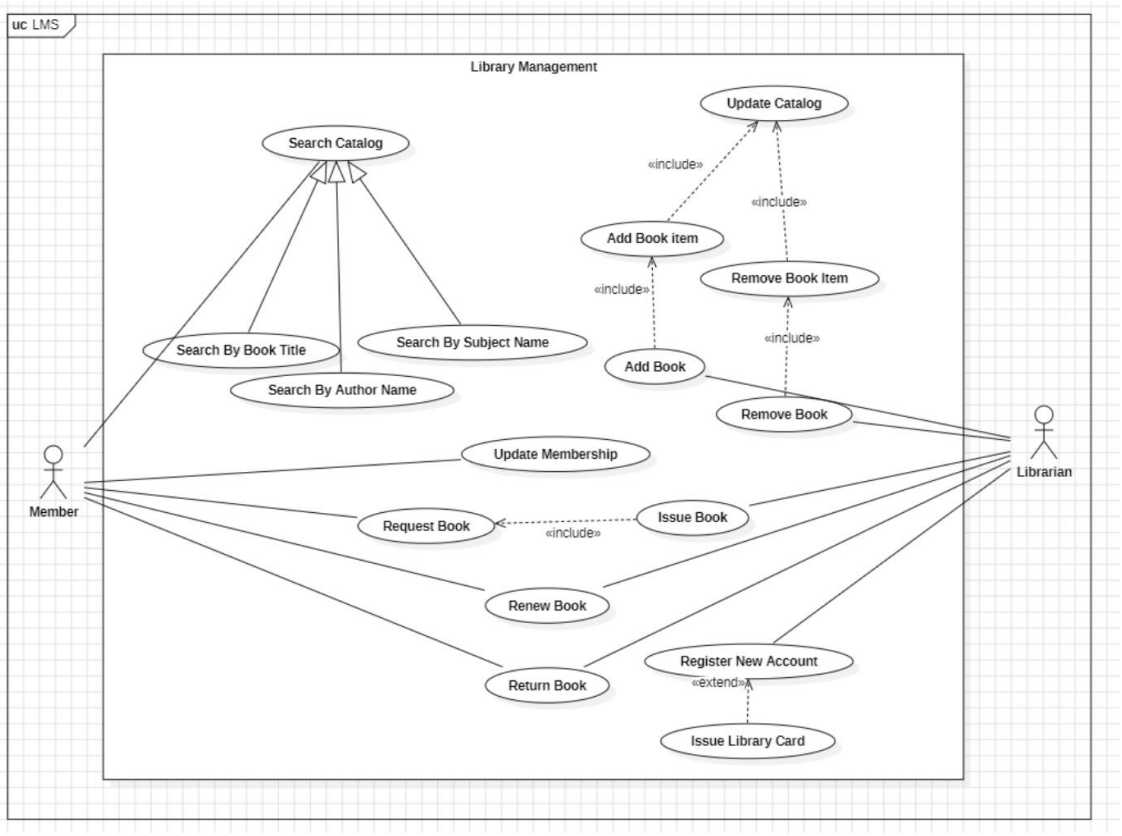


Fig 3.5.1

It represents a use case diagram for a Library Management System (LMS). It depicts the interactions between two actors, "Member" and "Librarian," and the system's functionalities represented as use cases (ovals). The "Member" actor can perform actions like searching the catalog (by book title, author name, or subject name), requesting a book, renewing a book, and returning a book. The "Librarian" actor is responsible for issuing books (which "includes" the "Request Book" use case), registering new accounts (which "extends" to "Issue Library Card"), and updating memberships. The "Search Catalog" use case is further detailed by "include" relationships with "Search By Book Title," "Search By Author Name," and "Search By Subject Name," indicating these are sub-functions of the broader search functionality. Overall, the diagram illustrates the system's scope and the services offered to members and librarians.

3.6 Sequence Diagram

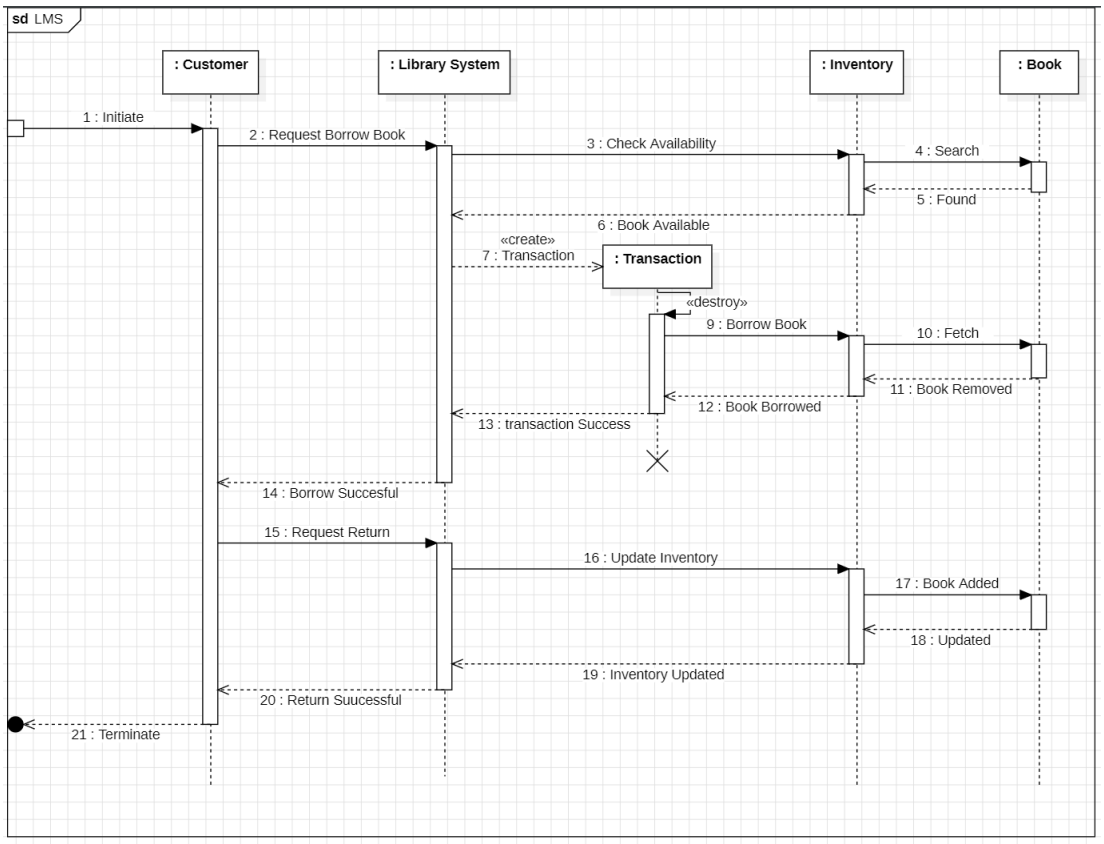


Fig 3.6.1

This UML sequence diagram provides a detailed representation of the borrowing and returning processes in a Library Management System (LMS). The sequence begins with the customer initiating a request to borrow a book. The Library System processes this request by checking the book's availability in the Inventory. If the book is found, the system creates a borrowing transaction, fetches the book from the Inventory, removes it from the available stock, and marks the transaction as successful. The customer is then notified that the borrowing process was completed successfully.

When the customer wants to return the book, they submit a return request to the Library System. The system processes the return by updating the Inventory, adding the book back to the available stock, and marking the inventory as updated. Finally, the customer is informed that the return process was successful. This sequence diagram highlights how the system manages book availability, tracks transactions, and ensures accurate inventory updates for efficient library operations.

3.7 Activity Diagram

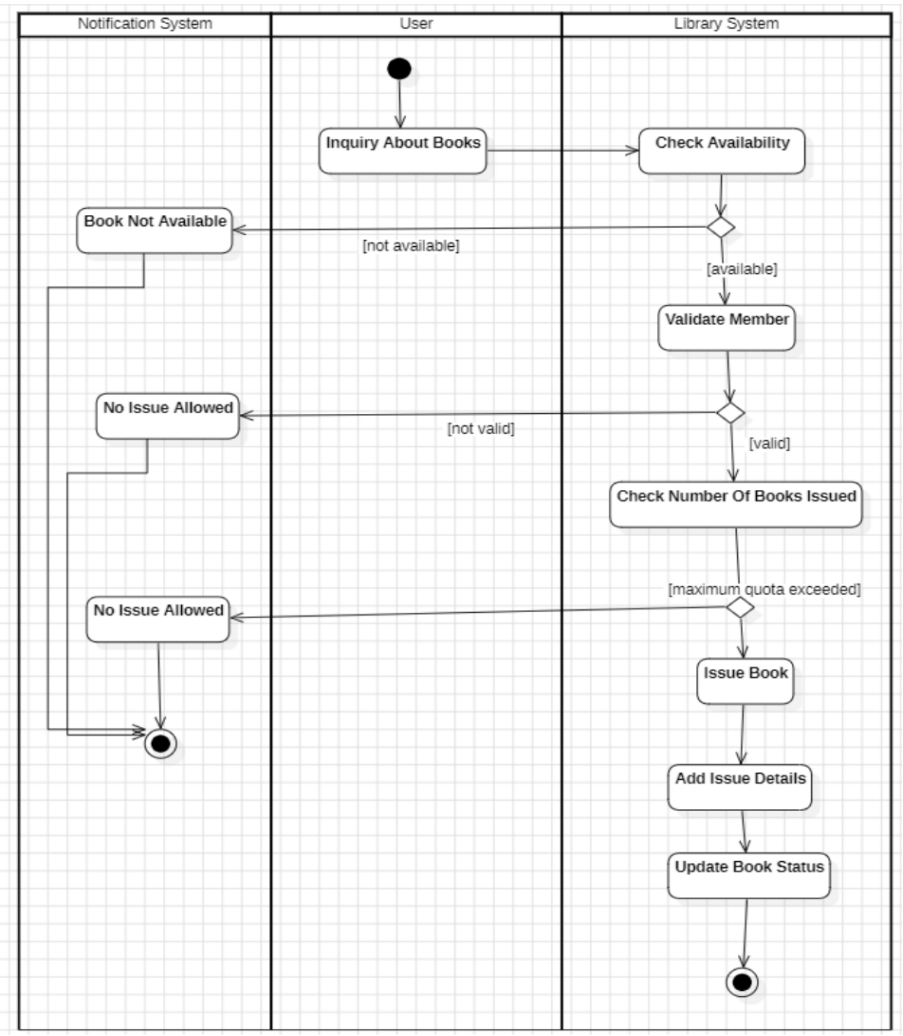


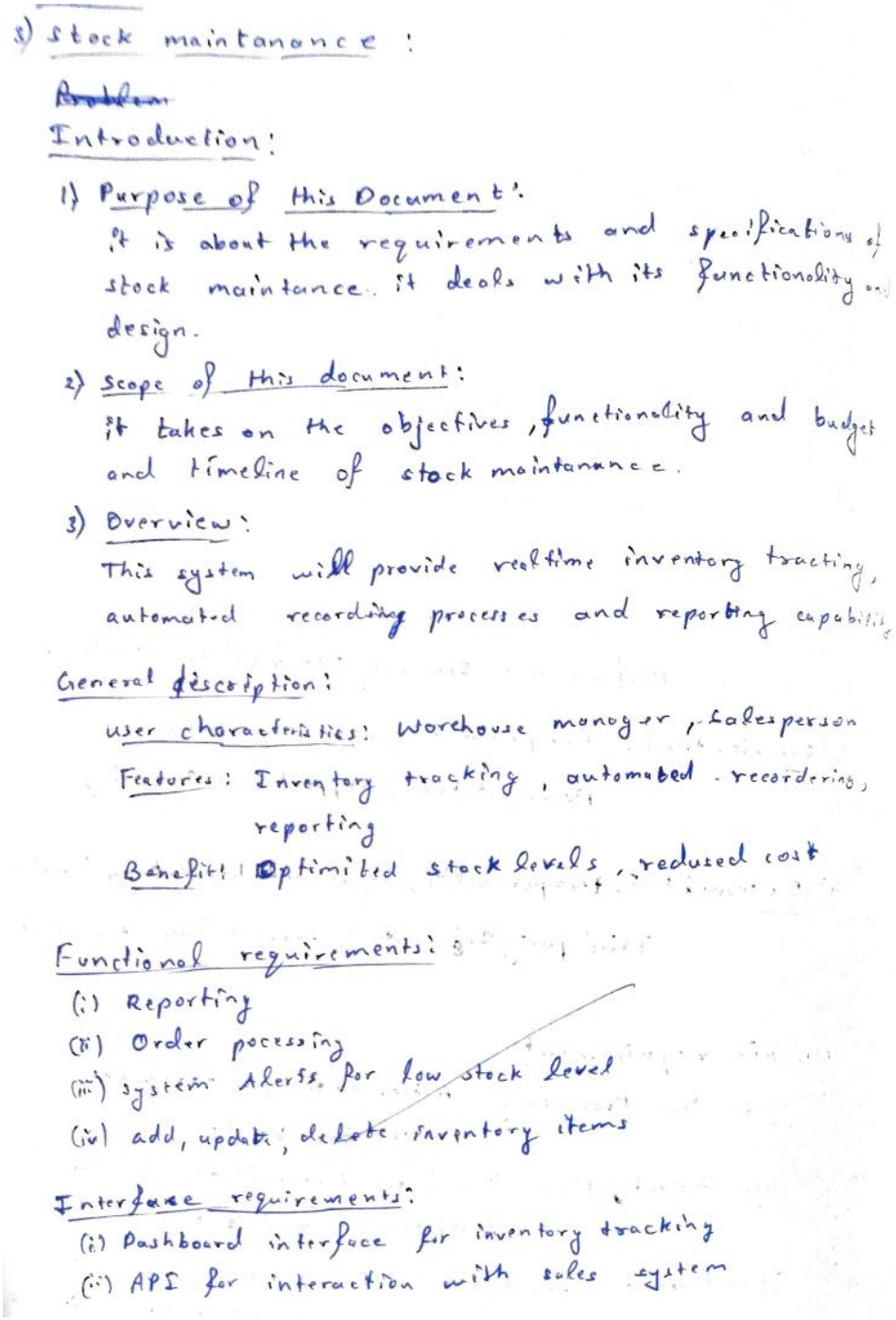
Fig 3.7.1

This is an activity diagram for a Library Management System, detailing the process of borrowing a book. It's divided into swimlanes representing the "User," "Library System," and "System Database." The process begins with the "User" requesting to borrow a book. The "Library System" then checks the book's availability in the "System Database." A decision diamond follows: if the book is "not available," a message is displayed to the user. If the book is "available," the "Library System" proceeds to "Issue Book," "Add Issue details," and updates the "book status" in the "System Database." Finally, the "Library System" "Lends the book" to the user, marking the end of the activity. The diagram clearly shows the flow of actions and responsibilities across different parts of the system during a book borrowing transaction, including the handling of book availability.

4. Stock Management System

4.1 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.

4.2 SRS-Software Requirements Specification



1) Stock maintanance :

~~Problem~~

Introduction:

1) Purpose of this Document:
it is about the requirements and specifications of stock maintance. it deals with its functionality and design.

2) Scope of this document:
it takes on the objectives, functionality and budget and timeline of stock maintanance.

3) Overview:
This system will provide realtime inventory tracting, automated recording processes and reporting capabili

General description:
user characteristics: Warehouse manager, salesperson
Features: Inventory tracking, automated recordering, reporting
Benefits: Optimized stock levels, reduced cost

Functional requirements:
(i) Reporting
(ii) Order processing
(iii) system Alerts for low stock level
(iv) add, update, delete inventory items

Interface requirements:
(i) Dashboard interface for inventory tracking
(ii) API for interaction with sales system

Fig 4.2.1

Performance Requirements:

(i) Report should generate faster

(ii) Inventory updating should be done in real time.

Design Constraints:

(i) Need to be integrated with existing ERP system

(ii) must be scalable for future expansion.

non-functional Attributes:

(i) Security

(ii) Reliability

(iii) Reusability

(iv) Availability

Preliminary schedule and Budget:

Estimated time: 5 months

Estimated cost: ₹1 Million

Fig 4.2.2

4.3 Class Diagram

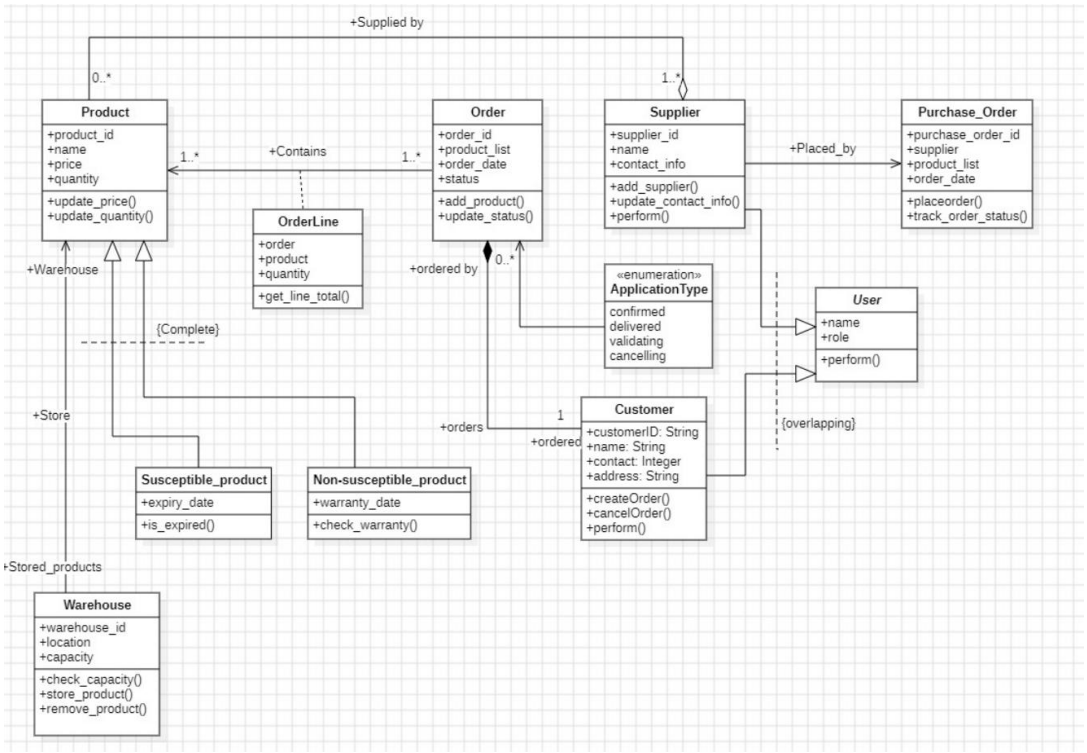


Fig 4.3.1

This represents a class diagram for a Stock Maintenance System. It models the relationships between various entities involved in managing stock, orders, and suppliers. Key entities include Product (with attributes like ID, name, price, and quantity, and operations to update price and quantity), Supplier (with name, contact information, and operations to add/update supplier details), Order (containing an order ID, product list, order date, and status, with operations to add products and update status), and Purchase_Order (linking suppliers to orders and tracking order status). An OrderLine class connects Order and Product, specifying the quantity of each product in an order. A Customer places orders and has attributes like ID, name, contact, and address. Warehouse stores products and has attributes for ID, location, and capacity, with operations to check capacity and manage stored products. The Inventory tracks the stock level of products in each warehouse. Finally, there are specialized product types: Susceptible_product (with an expiry date and an is_expired() operation) and Non-susceptible_product (with a warranty date and a check_warranty() operation), both inheriting from Product. The diagram shows the associations between these classes, illustrating the structure of the system and how different entities relate to each other in managing stock and orders.

4.4 State Diagram

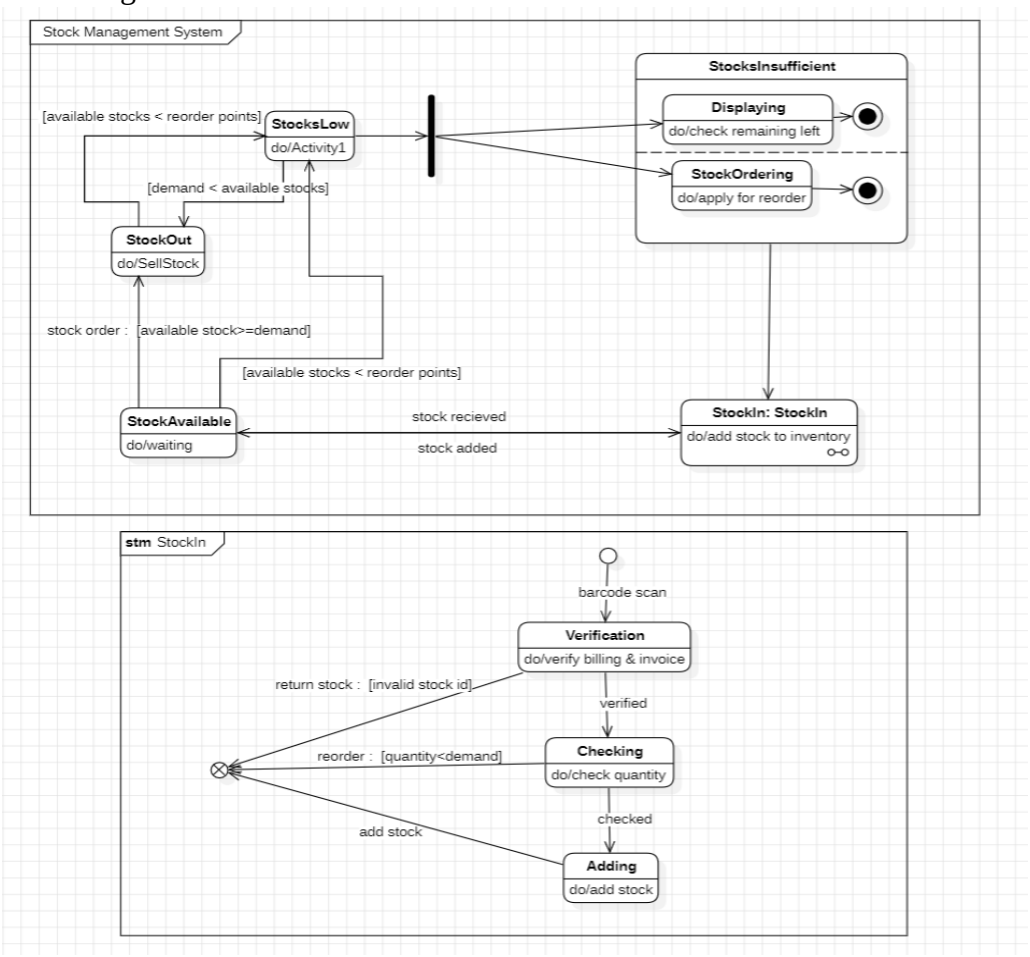


Fig 4.4.1

This state machine diagram represents the workflow of a stock maintenance system, detailing the various states and transitions involved in stock management. The process begins in an idle state, where the system waits for user login. Once authenticated, users can check the stock status, which involves fetching data from the stock database and displaying current stock levels and availability. After reviewing the stock, users can place orders by selecting the required items. Payment processing is handled through the payment system, with the outcome determining the next steps: a successful payment transitions to the successful state, while a failed payment leads to a waiting state where the system waits for the stock to arrive after retrying the payment. In the successful state, the system confirms the order by notifying the warehouse and updating internal stock records to reflect the new order. Finally, the system generates and prints a stock receipt for the ordered items, marking the end of the process. This diagram captures the seamless interaction between stock checking, ordering, payment, and stock record management

4.5 Use Case Diagram

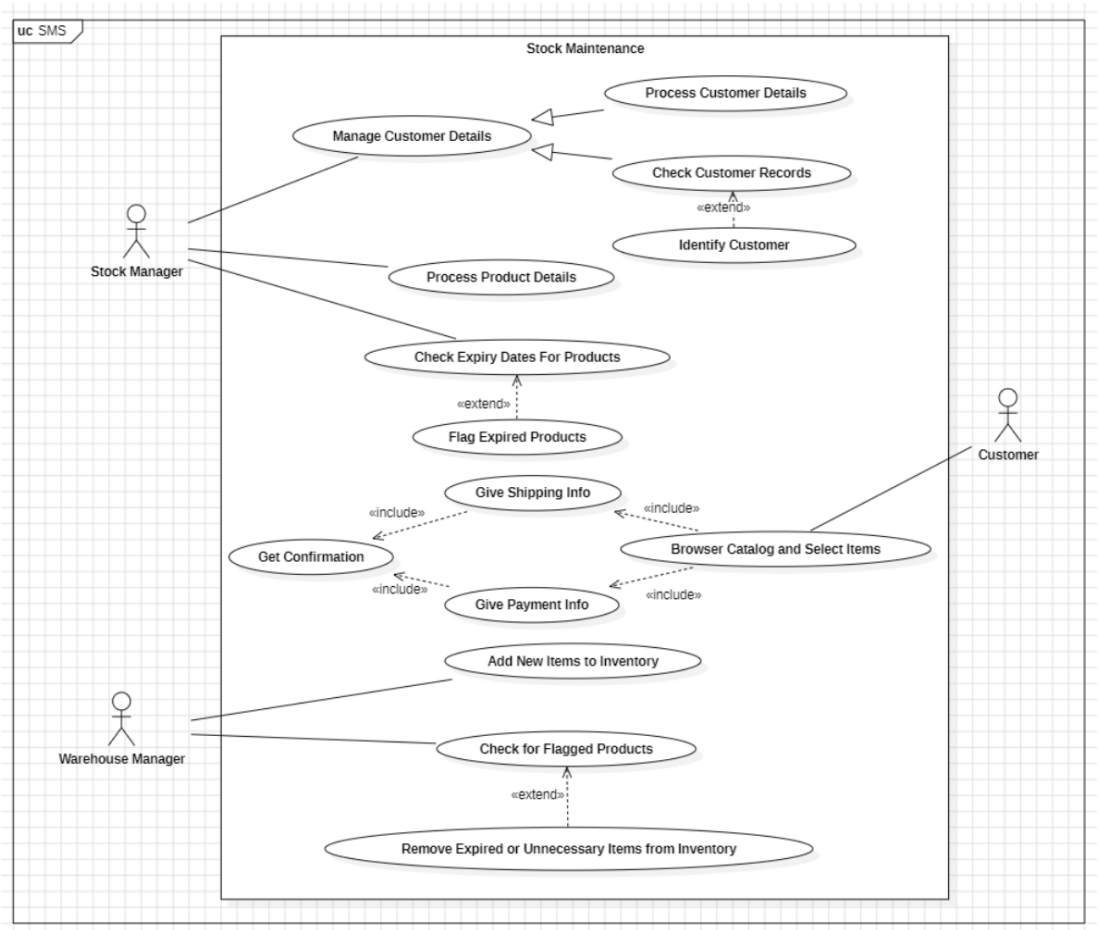


Fig 4.5.1

This UML use case diagram depicts the Stock Maintenance System (SMS) involving three actors: Stock Manager, Warehouse Manager, and Customer. The Stock Manager oversees customer and product details, processes records, checks product expiry dates, flags expired products, and adds new items to the inventory. The Warehouse Manager ensures inventory cleanliness by removing flagged or expired items. Meanwhile, the Customer interacts with the system to browse the catalog, select items, and provide shipping and payment details to complete purchases. The diagram highlights the key processes and interactions required for efficient inventory and customer management.

4.6 Sequence Diagram

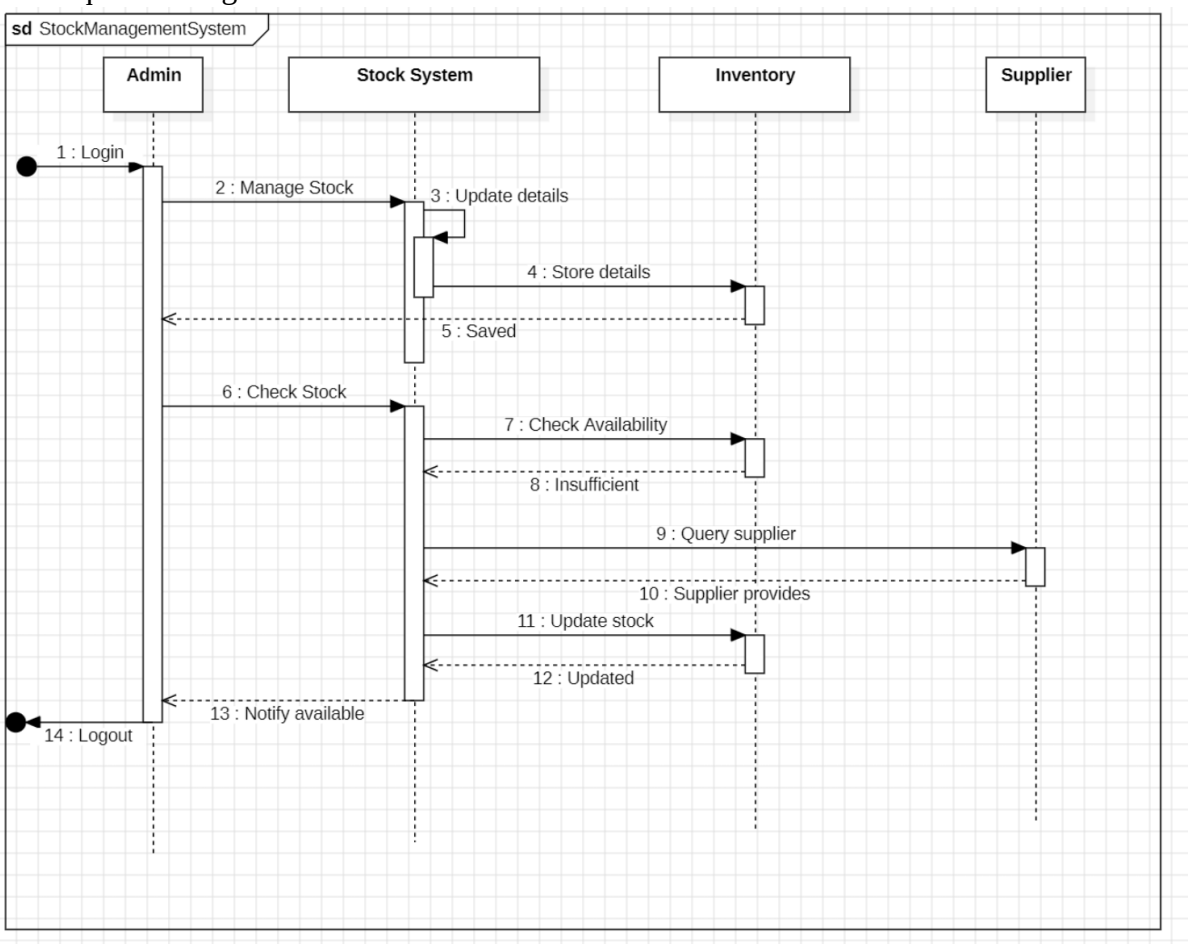


Fig 4.6.1

This sequence diagram illustrates the process flow in a stock management system involving three main entities: the customer, supplier, and warehouse. The interaction begins with the customer initiating a product inquiry, prompting the supplier to check product availability with the warehouse. The warehouse confirms availability to the supplier, who then shares the details with the customer. After reviewing the details, the customer places an order with the supplier. The supplier subsequently places the order with the warehouse, which updates stock records, confirms the order, and deducts the ordered quantity. Once the order is placed, the customer completes the payment, and the process proceeds to delivery. This diagram effectively captures the sequential actions and interactions ensuring smooth stock management.

4.7 Activity Diagram

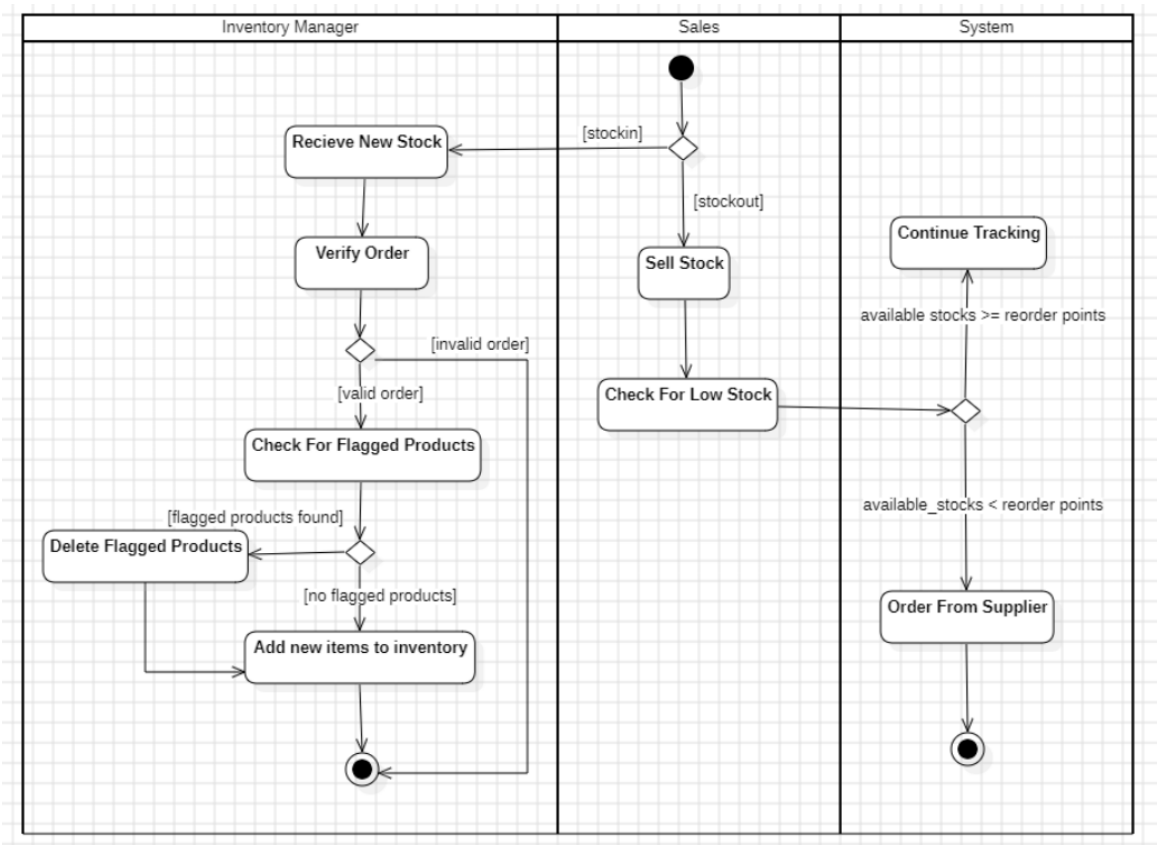


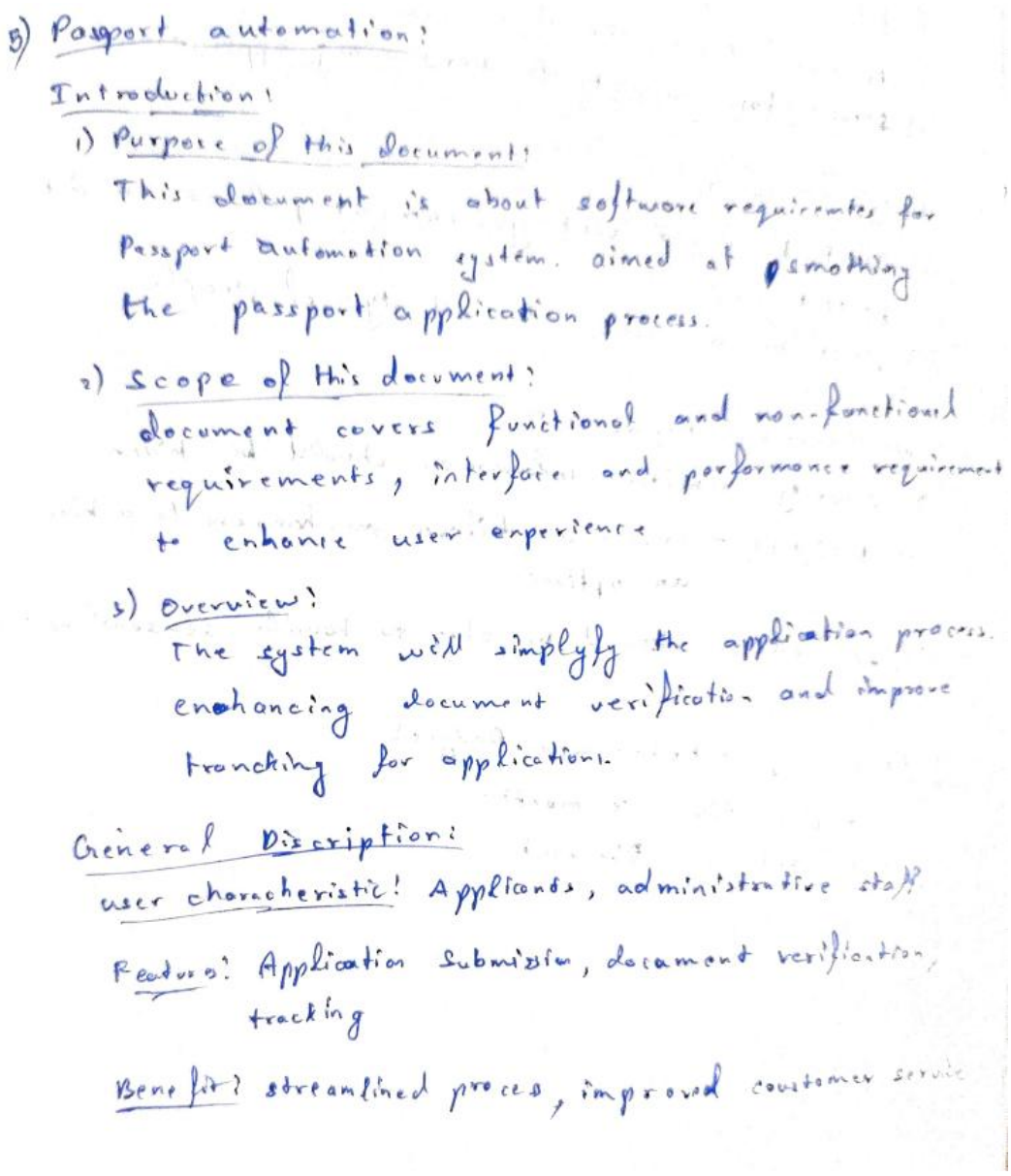
Fig 4.7.1

The diagram showcases a stock management process in a system with four key components: Admin, Inventory Database, Stock Maintenance System, and Supplier. It begins with the Admin logging into the system to manage stock. This triggers the Stock Maintenance System to check the stock levels in the Inventory Database. If the stock is sufficient, the system updates and stores the relevant details in the database, ensuring the Admin has the latest information. However, if the stock is found to be insufficient, the Stock Maintenance System communicates with the Supplier. This involves querying the Supplier to check availability and waiting for the supplies to be provided through the supplier's network. Once the Supplier fulfills the request and the stock is delivered, the system updates the inventory and notifies the Admin that the required stock is now available. This automated process ensures seamless coordination between different components for efficient inventory tracking and timely replenishment.

5. Passport Management System

5.1 Problem Statement: Managing passport applications and issuance is a complex process involving multiple stages, including document verification, application tracking, and appointment scheduling. A streamlined system is required to handle these tasks efficiently, minimize errors, and enhance user experience.

5.2 SRS-Software Requirements Specification


5) Passport automation:

Introduction:

1) Purpose of this document:

This document is about software requirements for Passport automation system. aimed at smoothing the passport application process.

2) Scope of this document:

document covers functional and non-functional requirements, interface and performance requirement to enhance user experience

3) Overview:

The system will simplify the application process. enahancing document verification and improve tranching for applications.

General Discription:

user characheristic: Applicants, administrative staff

Features: Application Submission, document verification, tracking

Benefit: streamlined proces, improved customer servic

Fig 5.2.1

Functional Requirments:

(i) User can submit applications online
(ii) System verifies dauments and track application status
(iii) Generates notification for updates

Interface Requirements:

(i) Backend API for admistrative tasks and document verifications
(ii) user friendly Interface for applicants

Performance Requirements:

(i) Processing time should be less
(ii) Should have capacity to hold many applications

Design Contraints:

(i) Must be legal wrt government polices on data protection
(ii) Must be accessable to all type of peoples

Non-Functional Requirements:

(i) Security - sensitive information should be hiden
(ii) Reliability - ensure a failover mechanism to achive an uptime.
(iii) Scalability - should be able to handle seasonal spikes

Preliminary schedule and Budget:

schedule time : 7 months
estimated cost : ₹15 lakh

Fig 5.2.2

5.3 Class Diagram

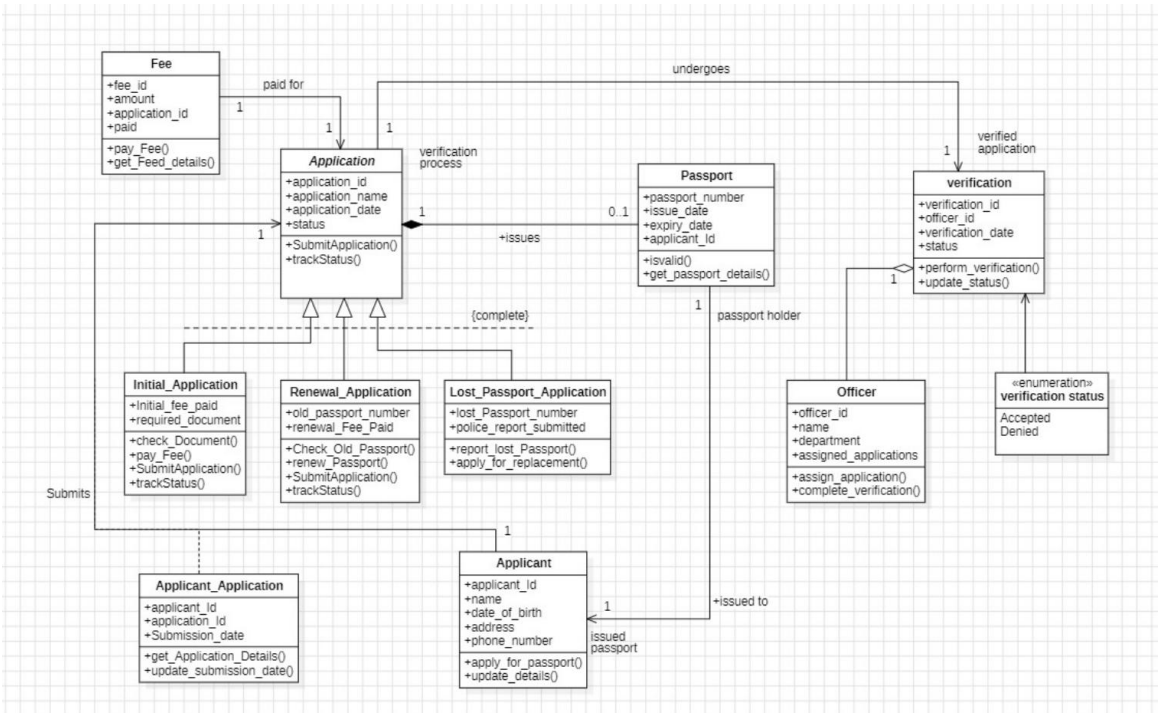


Fig 5.3.1

The Passport Automation System streamlines the end-to-end process of passport application, verification, and issuance by integrating multiple entities and workflows. Applicants can submit various application types, such as initial, renewal, or lost passport applications, each with specific requirements like document verification, fee payments, or police reports for lost passports. Applications are processed through a structured workflow where officers handle verification to ensure compliance. Approved applications result in passport issuance, with details such as passport number, issue, and expiry dates recorded. The system also tracks fee payments and facilitates passport delivery through the embassy. This automation ensures efficiency, transparency, and minimal delays in the passport issuance process.

5.4 State Diagram

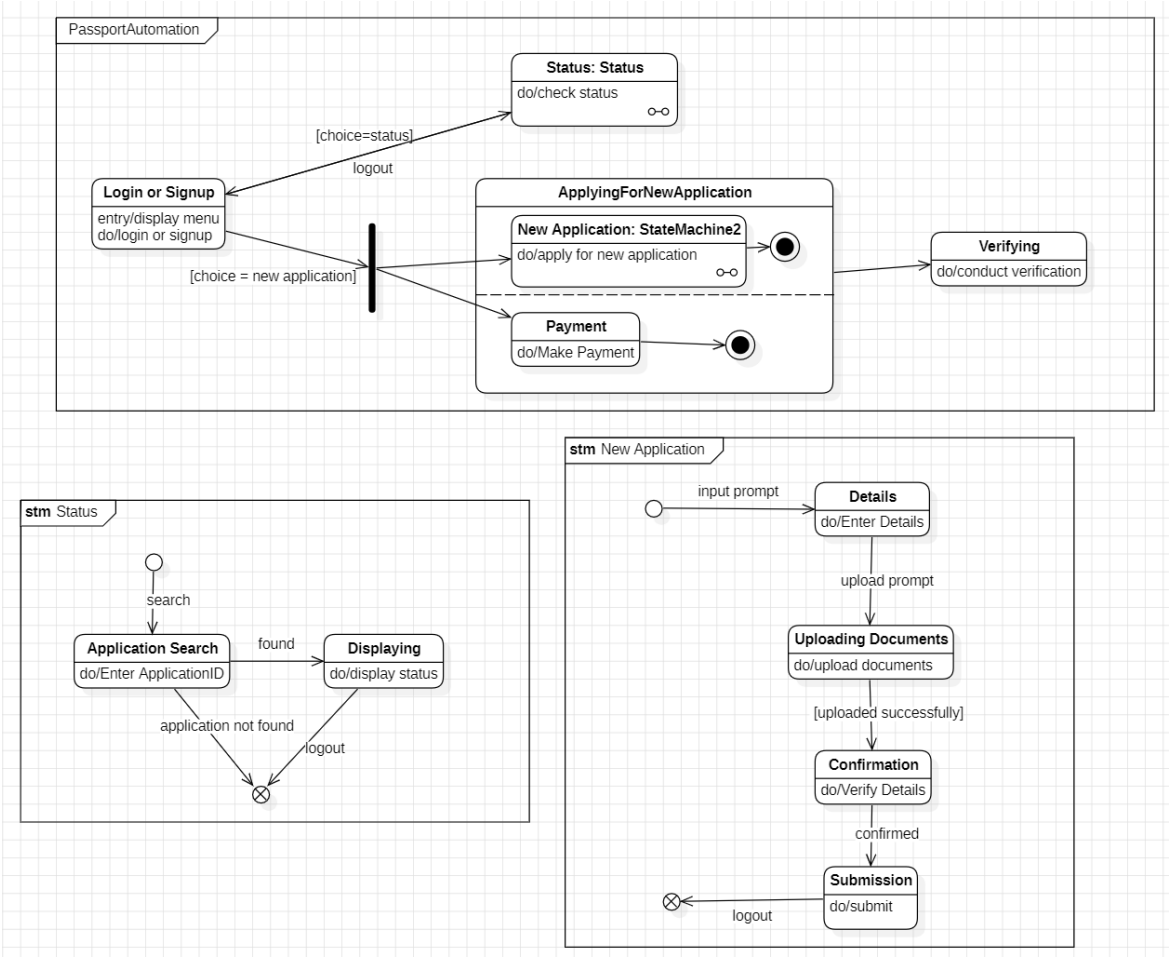


Fig 5.4.1

The diagram depicts the state model for a **Passport Management System** and its integrated **Payment Processing** system. The process begins with the initialization of the system, where users start the passport application process by submitting an application form. Following submission, the system transitions to payment processing, where funds are verified. Upon successful payment, the application proceeds to document submission and verification. If documents are verified successfully, the process moves forward to passport processing, printing, and preparing the passport for shipping. The final stage involves shipping the passport to the user and confirming delivery. In the Payment Processing subsystem, the flow handles various outcomes of payment transactions, including successful payment, failure, or cancellation, with corresponding system updates and user notifications. This state model ensures a structured, step-by-step approach to managing the passport application and delivery lifecycle.

5.5 Use Case Diagram

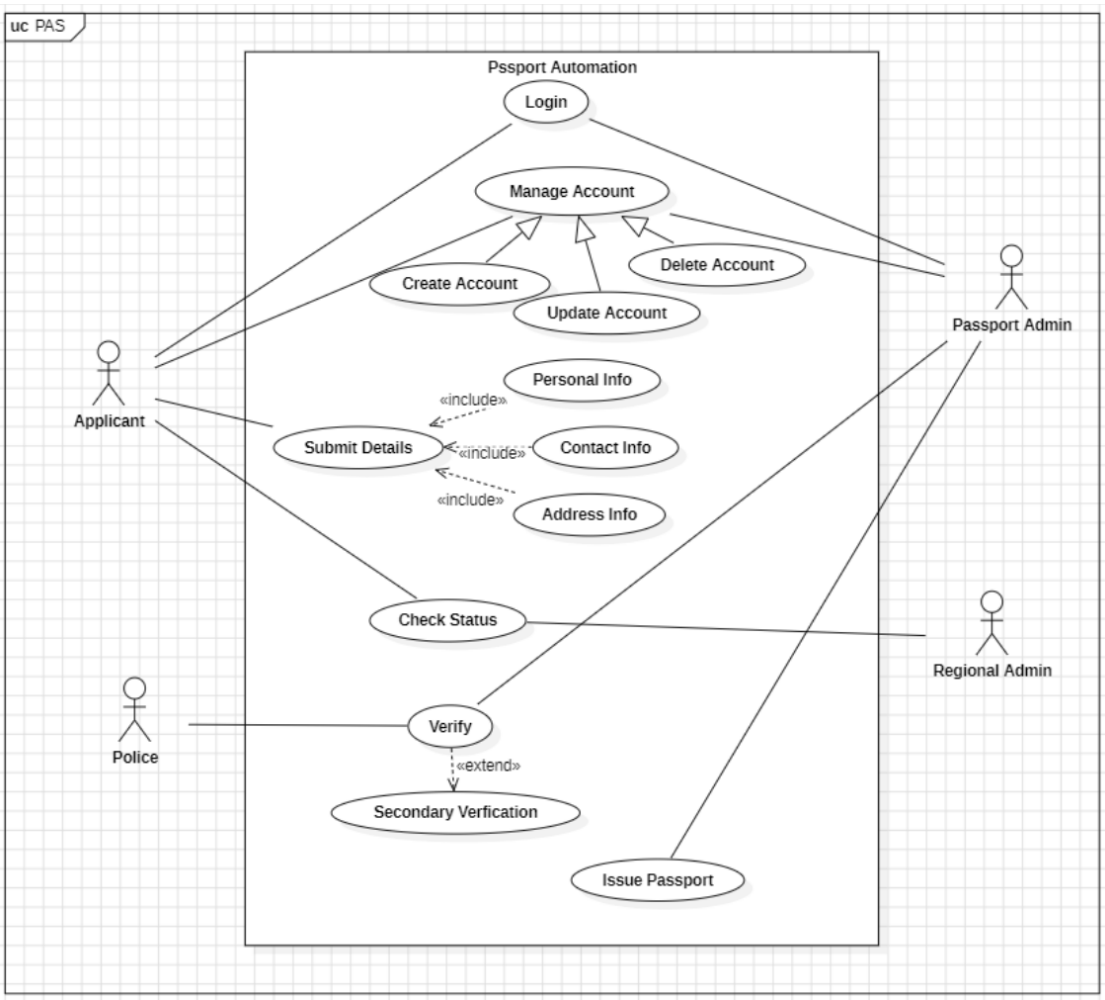


Fig 5.5.1

This use case diagram for a Passport Automation System. It illustrates the interactions between three actors: the Applicant, the Police, and the Passport Management System. The Applicant starts by logging into the system and proceeds to submit details, which include personal information, contact information, and address information. These are linked as separate use cases through the “include” relationship, signifying that they are essential steps in the process. The Applicant can also check the status of their application. The Police are involved in verifying the submitted details, a process connected to both the Applicant and the Passport Management System. Once the verification is completed, the system facilitates the issuance of the passport. The diagram effectively highlights the functional flow and dependency of activities within the automation system.

5.6 Sequence Diagram

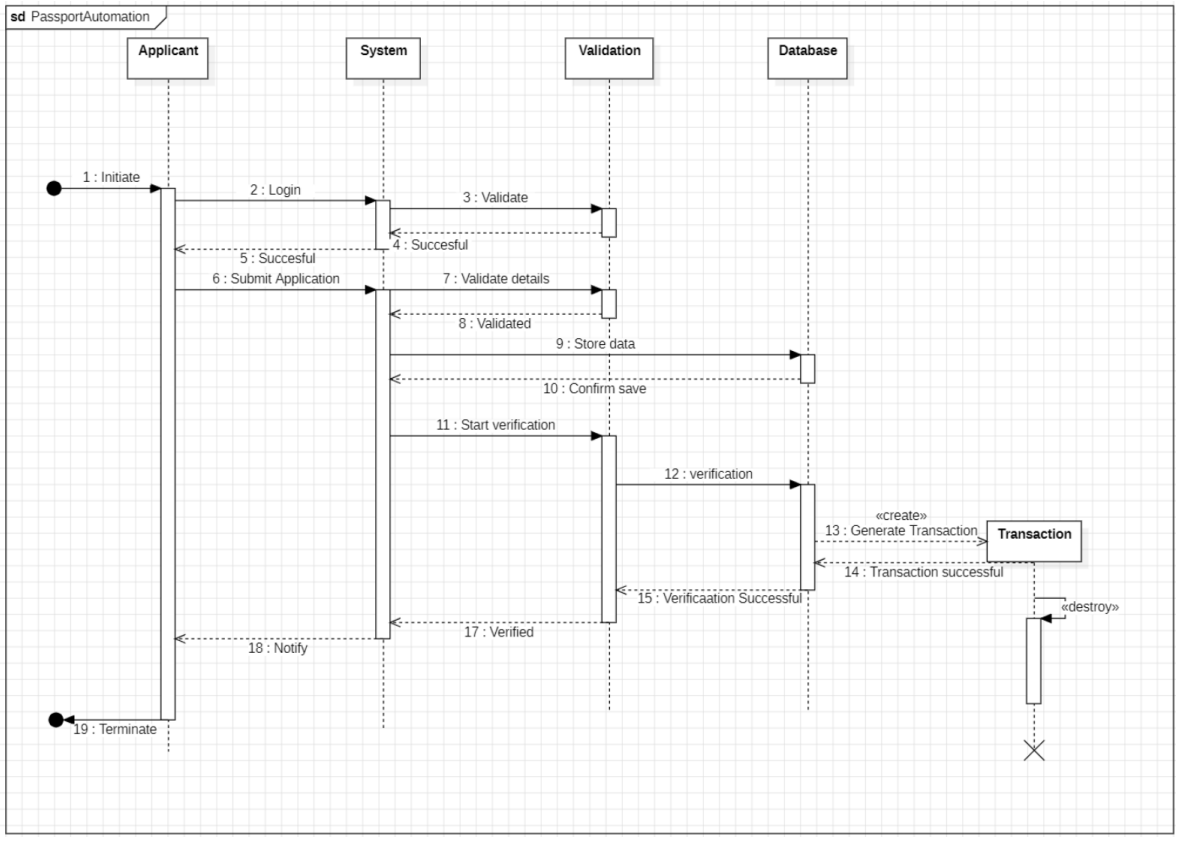


Fig 5.6.1

This UML sequence diagram outlines the process of a passport application through an automated system. The process begins when the Applicant initiates the system and logs in. Upon successful login, the Applicant submits their passport application, which is then validated by the System. The validated application is forwarded to the Validation module, where the details are verified and stored in the Database. After the data is successfully saved, the Database confirms the storage. The System then initiates the verification process, with the Validation module ensuring the correctness of the stored data in the Database. Once verification is complete, the System generates a successful Transaction, finalizing the process. Finally, the Applicant is notified that the application and verification process has been successfully completed. This diagram represents a seamless, step-by-step flow for handling passport applications, ensuring data validation, verification, and transaction management.

5.7 Activity Diagram

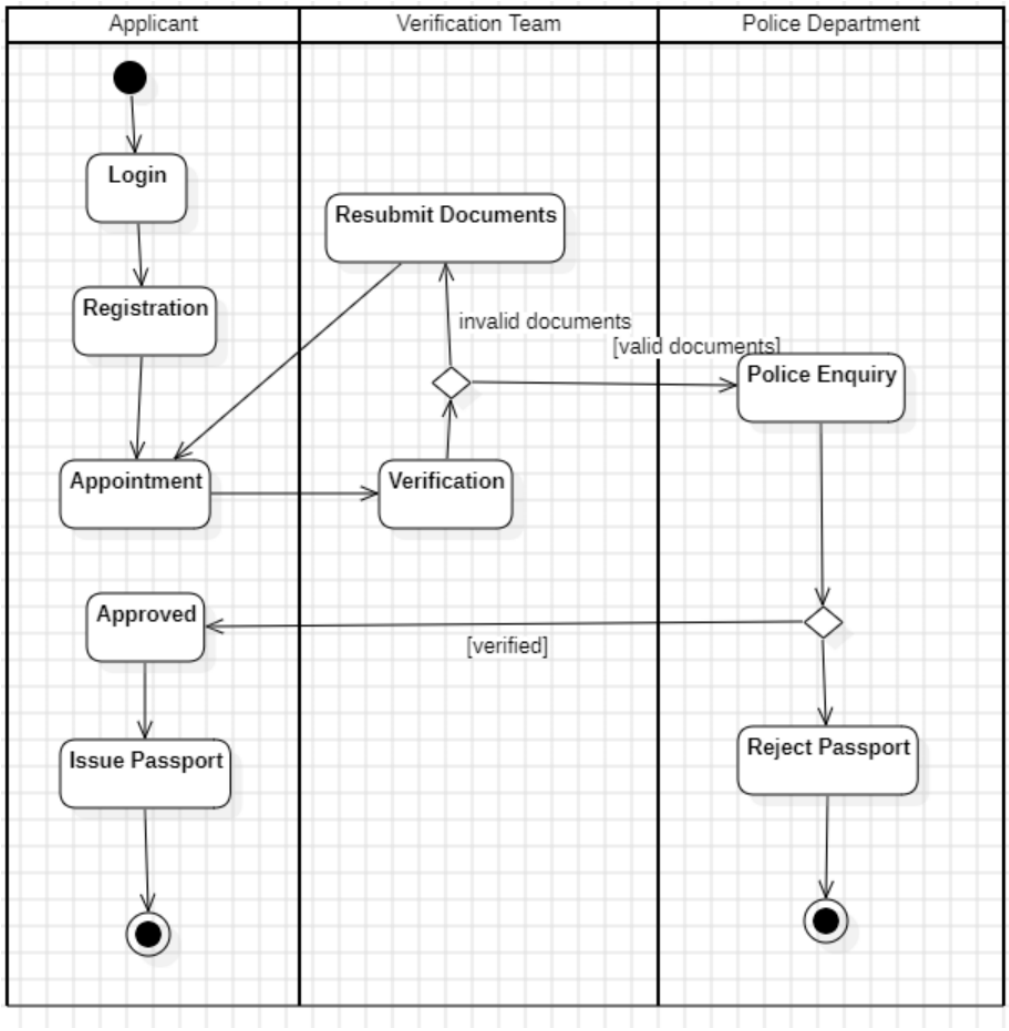


Fig 5.7.1

The activity diagram represents the workflow of a Passport Automation System, detailing the process across four primary components: Passport Automation System, Verification, Database, and Payment Gateway. It begins with a user decision between logging in (if an account exists) or signing up for a new account. Once authenticated through credential validation in the Database, the user submits an application. The Verification process involves two rounds of checks, including interaction with an authorities network, to ensure the details are accurate. After both rounds are completed, the application status moves to "Verification Completed." Simultaneously, the Database stores the data and confirms its integrity. In the Payment Gateway, a transaction is generated to complete the process, and upon successful payment, the application is officially submitted. This diagram clearly outlines the sequential and parallel actions, ensuring a comprehensive understanding of the system's operations.