

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



Lab Record

Object-Oriented Modeling – 23CS5PCOOM

Submitted in partial fulfillment for the 5th Semester Laboratory

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

NEHA SAJJANAR
1BM23CS209

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

B.M.S. COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING



CERTIFICATE

This is to certify that the Object-Oriented Modeling(23CS5PCOOM) laboratory has been carried out by NEHA SAJJANAR(1BM23CS209)during the 5th Semester August 2025-December 2025

Signature of the Faculty Incharge:

NAME OF THE Your Batch Incharge and designation: Sunayana S

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

Table of Contents

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

1. Hotel Management System

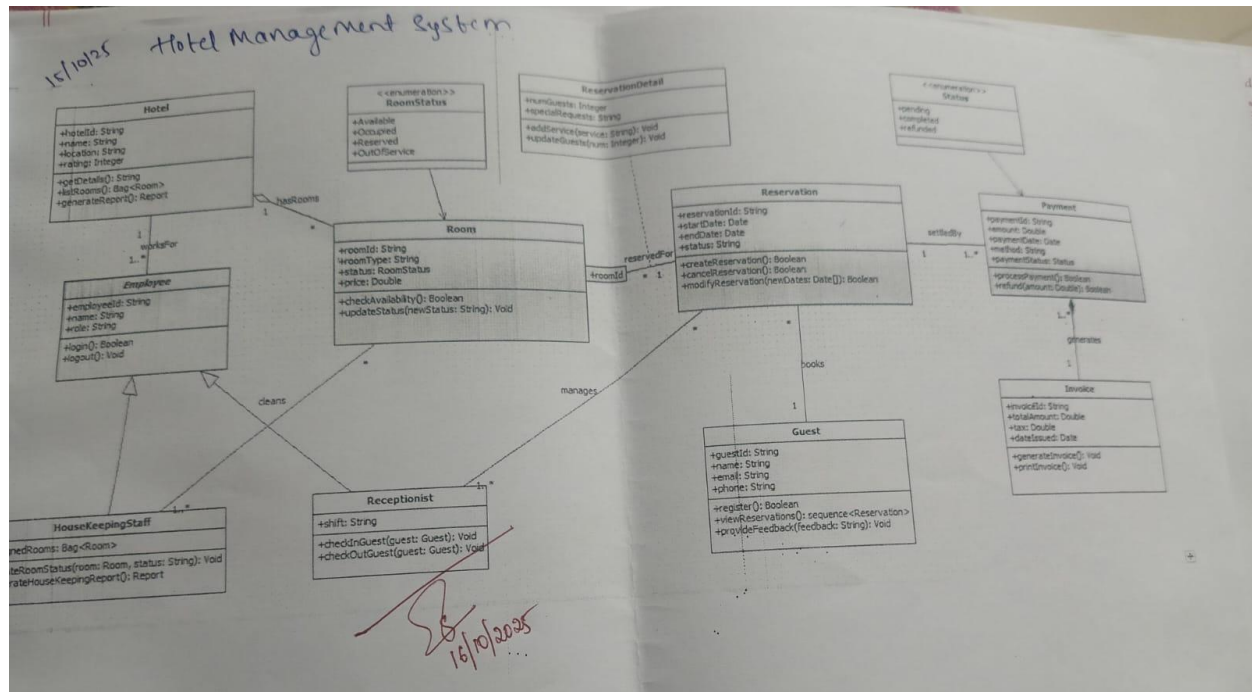
Problem Statement
SRS-Software Requirements Specification
<p>Class Diagram</p> <p>Requirements: Minimum 7 classes, 4 attributes and 4 operations in each class, associations, association name, association end names, multiplicity, association class, enumeration, qualified association, aggregation, composition, generalization, ordered, sequencing, multiplicity of attribute, brief description of the Class diagram</p>
<p>State Diagram: one simple state diagram, one advance state diagram (either Sub Machine or Nested State, include any one of the concurrency in your design)</p> <p>Requirements: minimum of 6 states for both the state diagram with state name, do activities, events, guard condition, brief description of State diagram</p>
<p>Use-Case Diagram: Minimum 5 use cases to be identified, and design one simple use case diagram and one advanced use case diagram(ie using include,extend and generalization relationship) and explanation as there in the solution manual.</p>
<p>Sequence Diagram: Write the scenario for any two use case transaction completely. and design the simple sequence diagram with minimum of 5 objects and communication between them.</p> <p>Design advanced sequence diagram using passive and transient objects. Brief explanation for each</p>
<p>Activity Diagram: Design the simple activity diagram showing the algorithm or workflow. Design the advanced activity diagram with swimlanes showing the responsible person for swimlane and also write the brief explanation for both.</p>

1. Hotel Management System

SRS-Software Requirements Specification

<p>and housekeeping, while providing real-time data and improving overall guest experience.</p>	<p>performance and design constraints. Finally, a preliminary schedule and budget estimation are included.</p>
<p><u>Software Requirements Specification (SRS)</u> for Hotel Management System:</p>	<p>2. General Description:</p>
<p>1. Introduction:</p>	<p>The Hotel Management System will be a web-based platform designed to streamline hotel operations. It will allow receptionists to manage bookings, housekeeping staff to update room status, and managers to generate reports. The system will provide a secure login for different user roles and offer real-time access to room availability, customer data, and transactions.</p>
<p>1.1 Purpose of the Document</p>	<p>3. Functional Requirements</p>
<p>The purpose of this document is to define the Software Requirements for the Hotel Management System (HMS). It outlines the functional and non-functional requirements, interface details, performance criteria, and constraints. This document serves as a reference for the development team, stakeholders, and clients.</p>	<p>1. Reservation Management: The System shall allow: 1) Room Booking: The System shall allow users to book rooms based on availability and preferences. 2) Guest Management: The System shall manage guest information, including check-in and check-out details. 3) Payment Processing: The System shall handle various payment methods and generate invoices. 4) Housekeeping Management: The System shall track room cleaning status and assign tasks. 5) Reporting: The System shall generate reports on occupancy rates, revenue and other key metrics.</p>
<p>1.2 Scope of the Document:</p>	<p>6. User Management: The System shall manage user accounts with different access levels.</p>
<p>This document describes the design and functionality of a Hotel Management System that automates hotel operations. Such as room bookings, check-ins / check-outs, billing, housekeeping, staff management, and reporting. The System will support staff and management users, with different access levels.</p>	<p>4. Interface Requirements: 1) User Interface (UI): Web-based responsive interface for different screen sizes. 2) Database Interface: MySQL or PostgreSQL database integration for data storage and retrieval. 3) Third-party Integration: Payment gateway, email notifications and possibly external booking APIs.</p>
<p>1.3 Overview:</p>	<p>5. Performance Requirements:</p>
<p>This SRS provides a detailed description of the System's expected behavior. Section 2 provides general system information. Section 3 lists the functional requirements, followed by non-functional requirements.</p>	<p>1. The System should handle up to 500 concurrent users. 2. Data retrieval and updates should complete within 2 seconds under normal conditions. 3. System uptime should be 99.5% monthly.</p>
<p>6. Design Constraints</p>	<p>8. Preliminary Schedule and Budget Schedule (For a 6-month project) Month 1: Requirements gathering and system design. Month 2: Database design and backend development. Month 3: Frontend development. Month 4: Integration of modules (booking, billing, housekeeping). Month 5: Testing (Unit, Integration, System). Month 6: Deployment and user training.</p>
<p>7. Non-functional Attributes:</p>	<p>9. Budget (Estimated):</p>
<p>Security: Role-based access control, encrypted data storage.</p>	<p>Development: 200000 Rs. Testing and QA: 20000 Rs. Infrastructure (Servers, DB, Hosting): 20000 Rs. Licencing and payment gateway: 50000 Rs. Total Estimated Cost: 1,140,000 Rs.</p>
<p>8. Usability: Intuitive and user-friendly interface.</p>	<p>Scalability: Must support future expansion.</p>
<p>9. Maintainability: Modular code and clear documentation.</p>	<p>Portability: Should be deployable on both Windows and Linux servers.</p>
<p>10. Reliability: High availability and disaster recovery plans.</p>	<p></p>
<p>11. Compliance: Adherence to data protection regulations (e.g., GDPR).</p>	<p></p>
<p>12. Accessibility: Compliant with WCAG guidelines.</p>	<p></p>
<p>13. Interoperability: Ability to integrate with existing hotel systems.</p>	<p></p>
<p>14. Security: Regular security audits and updates.</p>	<p></p>
<p>15. Performance: Optimize database queries and server response times.</p>	<p></p>
<p>16. Documentation: Comprehensive user manuals and system documentation.</p>	<p></p>
<p>17. Training: Provide training sessions for staff and management.</p>	<p></p>
<p>18. Support: Offer ongoing technical support and maintenance.</p>	<p></p>
<p>19. Backup and Recovery: Implement robust backup and recovery procedures.</p>	<p></p>
<p>20. Audit and Reporting: Implement audit logs and generate compliance reports.</p>	<p></p>
<p>21. Scalability: Design for horizontal scaling to handle future growth.</p>	<p></p>
<p>22. Flexibility: Allow for easy integration of new features and modules.</p>	<p></p>
<p>23. Security: Implement strong authentication and authorization mechanisms.</p>	<p></p>
<p>24. Performance: Optimize system performance through caching and load balancing.</p>	<p></p>
<p>25. Usability: Ensure the system is easy to use for all intended users.</p>	<p></p>
<p>26. Maintainability: Design for easy maintenance and updates.</p>	<p></p>
<p>27. Reliability: Ensure high system availability and uptime.</p>	<p></p>
<p>28. Compliance: Adhere to all relevant industry standards and regulations.</p>	<p></p>
<p>29. Interoperability: Ensure the system can work with other hotel systems.</p>	<p></p>
<p>30. Security: Implement robust security measures to protect data.</p>	<p></p>

Class Diagram



Hotel: Represents the hotel establishment and manages its rooms and staff.

- hotelId: int - Unique identifier for the hotel
- name: String - Hotel name
- address: String - Physical location of the hotel
- phone: String - Contact phone number
- email: String - Contact email address
- get_available_rooms() - Returns a list of currently available rooms
- add_room() - Adds a new room to the hotel inventory
- Has a composition relationship with Staff (1 to many) - Hotel manages multiple staff members
- Has an aggregation relationship with Room (1 to many) - Hotel contains multiple rooms

Staff: Represents hotel employees with different roles.

- name: String - Employee name
- phone: int - Contact number
- role: String - Job position/designation
- performDuty() - Execute role-specific responsibilities

Receptionist - Front desk staff handling guest check-in/check-out

- specialization: String - Area of expertise
- checkinGuest() - Process guest arrival
- checkoutGuest() - Process guest departure

Manager - Management personnel overseeing operations

- department: String - Managed department
- generate_report() - Create operational reports
- Associated with Service (many to many) - Staff members can provide multiple services

Room: Represents individual accommodation units in the hotel.

- room_no: int - Unique room identifier
- type: String - Room category (references RoomType enumeration)
- status: String - Current availability status
- price: double - Room rate per night
- update_status() - Change room availability status
- check_availability() - Verify if room can be booked
- Associated with RoomType enumeration (defines Single/Double categories)
- Part of Reservation (0 or more reservations per room)

RoomType <<enumeration>>: Defines available room categories.

- Single - Single occupancy room
- Double - Double occupancy room

Reservation: Represents a booking made by a guest.

- checkInDate: Date - Scheduled arrival date
- check_out_date: Date - Scheduled departure date
- status: String - Reservation state (confirmed, pending, cancelled)
- reservationId - Unique booking identifier
- confirmReservation() - Finalize and validate the booking
- cancelReservation() - Terminate the reservation
- Associated with Room (1 reservation to 0 or more rooms)
- Associated with Guest (1 guest to 1 or more reservations)
- Associated with Payment (1 reservation to 1 payment)
- Associated with Invoice (generates billing documentation)

Guest: Represents customers booking and staying at the hotel.

- name: String - Guest full name
- address: String - Residential address
- phone: int - Contact number
- email: String - Email address
- make_reservation() - Create a new booking
- requestService() - Order hotel services
- Associated with Reservation (1 guest can have multiple reservations)

- Associated with Service (many to many) - Guests can request multiple services

Service:Represents additional amenities and services offered by the hotel.

- id: int - Unique service identifier
- cost: double - Service charge
- type: String - Service category (room service, laundry, spa, etc.)
- assign_guest() - Allocate service to a specific guest
- Associated with Guest (many to many)
- Associated with Staff (many to many) - Multiple staff can provide services

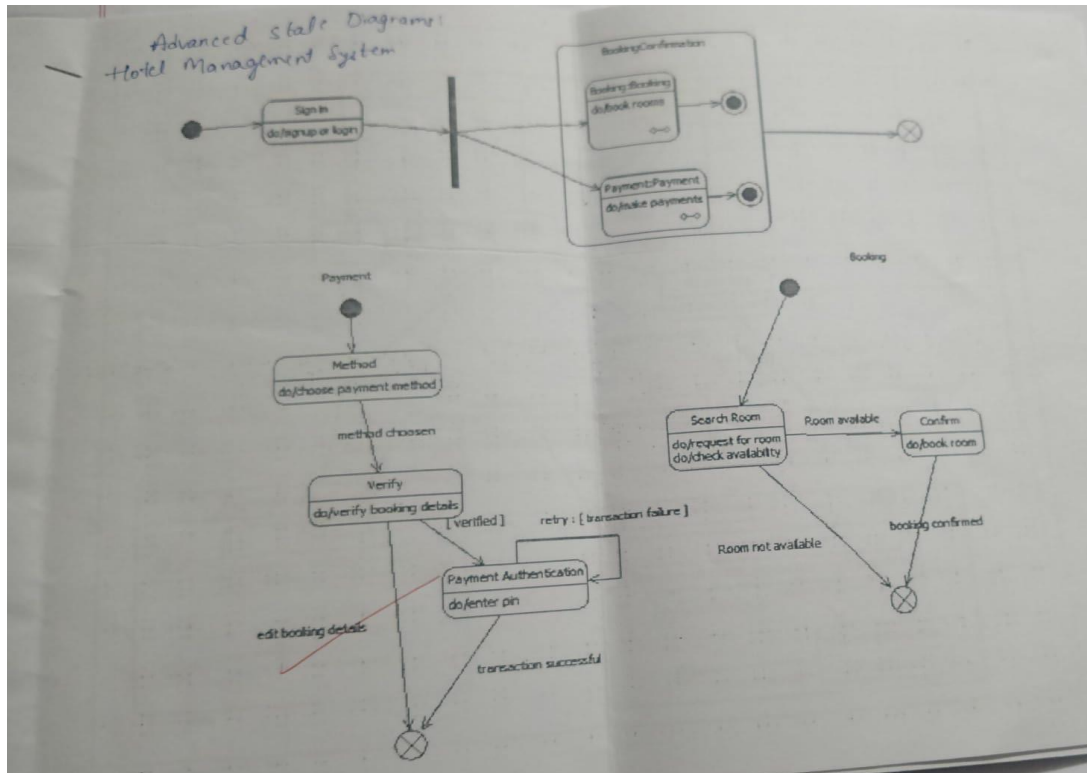
Payment <<Abstract>>:Abstract base class for all payment transactions.

- amount: double - Total payment amount
- date: Date - Transaction date
- method: String - Payment type identifier
- processPayment() - Execute the payment transaction
- CreditCardPayment - Card-based payments (attributes/methods not detailed in diagram)
- Cash Payment - Cash transactions (attributes/methods not detailed in diagram)
- Associated with Reservation (1 payment per reservation)

Invoice:Represents billing documentation for reservations.

- Associated with Reservation (generates invoice for bookings)

State Diagram



State Machine 1: Hotel Management (Main Workflow)

Sign In :Initial authentication state where users access the system.

- do/signup or loginup - User performs registration or login
- After successful authentication → Fork (synchronization bar) leading to parallel regions

BookingConfirmation (Composite State):Contains two concurrent sub-states that execute in parallel:

Sub-state 1: Booking

- Activities: do/book rooms - Process room reservation
- Completion: Transitions to flow join (filled circle) after booking complete

Sub-state 2: Payment

- Activities: do/make payment - Process payment transaction
- Completion: Transitions to flow join after payment complete

Exit Condition: Both booking AND payment must complete before proceeding to final state

Final State: Reached after successful completion of both booking and payment

State Machine 2: Payment (stm Payment)

Method (Initial State):Payment method selection phase.

- do/choose payment method - User selects payment option (credit card, cash, etc.)
- Event: method chosen → Verify

Verify: Validation of payment details.

- do/verify booking details - System validates transaction information
- Event: [verified] → Payment Authentication
- Event: retry : [transaction failure] → Returns to Method (allows user to choose different payment method)

Payment Authentication:Secure payment processing state.

- do/enter pin - User provides authentication credentials
- Event: transaction successful → Final State (⊗)
- Event: Edit Booking Details → Final State (user opts to modify booking instead)

State Machine 3: Booking (stm Booking)

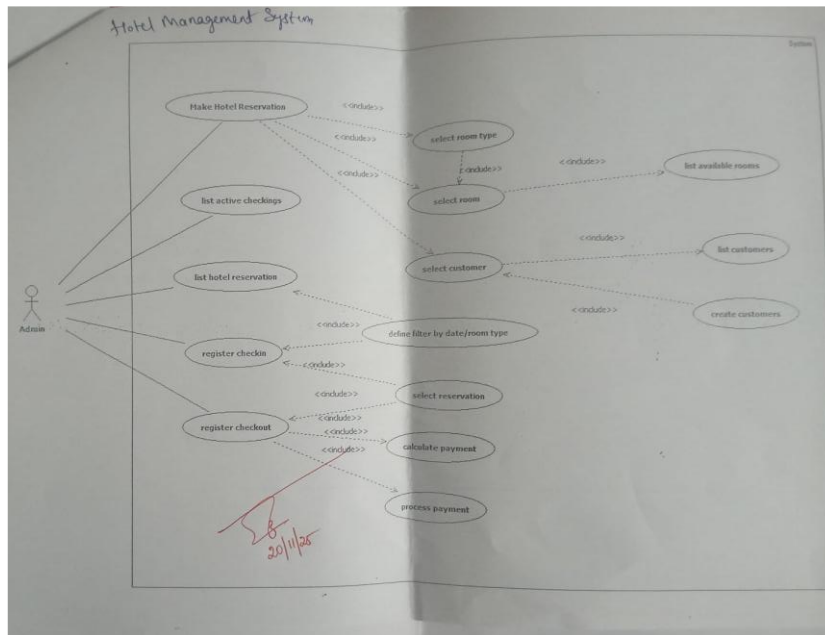
Search Room (Initial State):Room availability inquiry phase.

- do/request for room - User searches for available rooms
- do/check availability - System queries room inventory
- Guard: [Room Available] → Confirm
- Guard: [Room not Available] → Final State (⊗) - Booking cannot proceed

Confirm:Reservation finalization state.

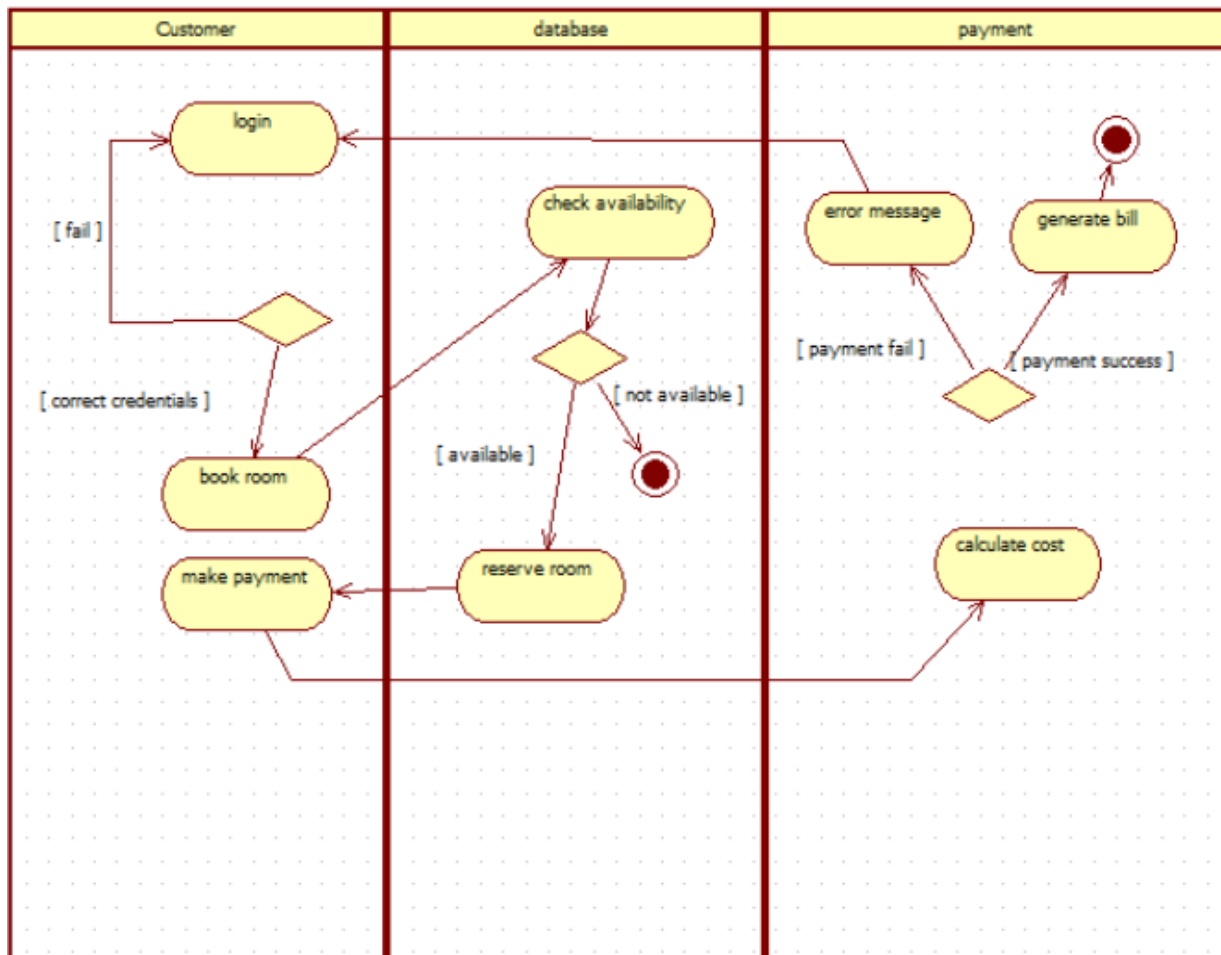
- do/Book room - System creates reservation record
- Event: booking confirmed → **Final State** (⊗)

USECASE



- The diagram shows the main Admin actor who interacts with the hotel system.
- The admin can perform tasks such as Make Hotel Reservation, List Active Check-ins, and List Hotel Reservations.
- For making a reservation, the admin selects a room type, views available rooms, and chooses a specific room.
- The admin can also select customers from the customer list.
- If needed, new customers can be created using the “create customer” use case.
- Reservations can be filtered by date or room type.
- The admin can register check-ins when guests arrive.
- The admin can register checkouts when guests leave.
- The admin can also select reservations to modify or confirm them.
- Lastly, the admin handles payments, including selecting payment options and processing them.

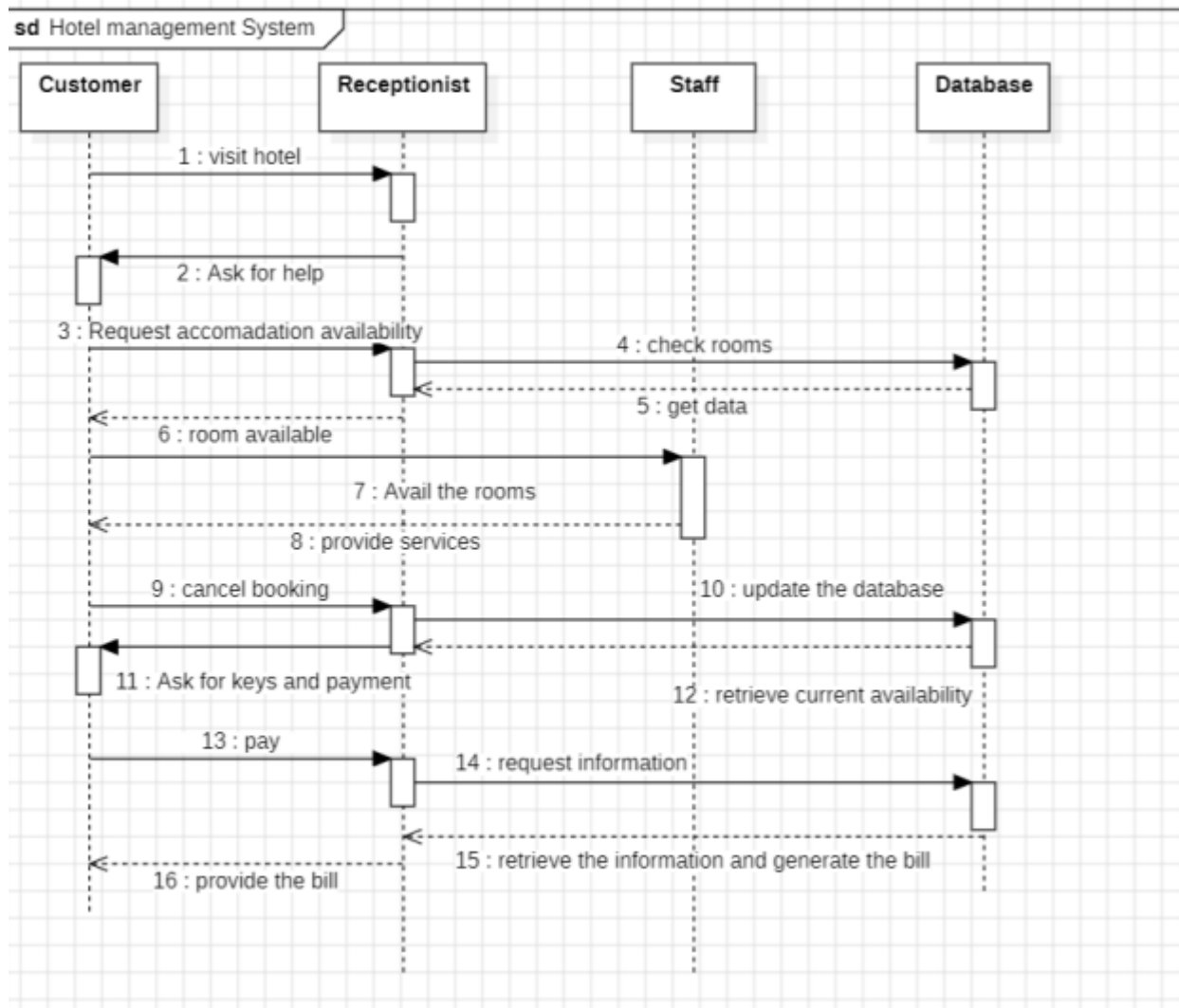
Activity Diagram



- The diagram has three swimlanes: Customer, Database, and Payment, showing who performs which action.
- The process starts with the customer logging in to the system.
- If the login credentials are correct, the customer proceeds; otherwise, they retry.
- The customer requests to check the availability of rooms.
- The Database swimlane handles the availability check and returns results.
- If rooms are available, the customer decides to book a room.
- After booking, the system moves to the payment process.
- Payment is processed — either success or failure paths exist.

- If successful, a bill is generated in the Payment swimlane.
- Finally, the system ends with the **room being reserved** and booking confirmed

Sequence Diagram



- There are four lifelines: Customer, Receptionist, Staff, and Database.
- The sequence begins with the customer visiting the hotel.
- The customer asks for help and requests room availability. The Receptionist sends a message to Staff to check rooms.
- Staff checks rooms and sends the results to the receptionist.
- Receptionist also queries the database for room availability.

- If available, the receptionist informs the customer and completes booking.
- Customer may cancel booking, and the receptionist updates the database.
- Customer later asks for keys and payment processing.
- Receptionist retrieves data from database, finalizes billing, and provides the bill to the customer.

2. Credit Card Processing System

SRS-Software Requirements Specification

3. Credit Card Processing System

Problem Statement:
As we can see that many businesses rely heavily on credit card transactions, but processing them securely and efficiently is still a challenge due to issues like fraud, compliance requirements, and performance demands. And this system mainly works on ensuring secure, fast and reliable credit card payments.

SRS Document:

- 1. Introduction:**
 - 1.1 Purpose of Document:**
The purpose of this document is to clearly define the requirements and design considerations for developing a secure and efficient credit card processing system. It serves as a common reference for developers, testers, project managers, ensuring alignment and minimizing confusions during the development lifecycle.
 - 1.2 Scope of Document:**
The scope of this system is to handle end-to-end credit card transactions, including authorization, capture, settlement, refunds and chargebacks. It will keep the validation of user identity, check card details, process payments in real time, and provide seamless integration with external payment gateways and financial institutions.
- 2. External Interfaces:** Seamless integration with third-party gateways and financial networks.
- 3. Performance Requirements:**
 - At least most of transactions should complete in less time.
 - It should support many concurrent users without any delays.
- 4. Design Constraints:**
 - The system must comply with security standards for handling sensitive payment data.
 - Deployment should be such a way that it is scalable and flexible.
 - And also open source technologies should be used to reduce cost of making.
- 5. Non Functional Attributes:**
 - Security:** End-to-end encryption, secure storage of sensitive information.
 - Reliability:** Automatic failover mechanisms and also strategies for recovery from disaster.
 - Usability:** Easy to use and responsive user interface for both customers and admins.
 - Maintainability:** A system that allows easy updates and fixes.
 - Scalability:** Ability to handle increased transaction as demand grows.
- 6. Preliminary Schedule and Budget:**
 - 6.1 Schedule:**
 - Step 1: Requirement and Design - 2 weeks
 - Step 2: Development - 6 weeks
 - Step 3: Testing - 2 weeks
 - Step 4: Deployment and Monitoring - 2 weeks.

2. General Description:
The credit card processing system will support all key stages of an electronic transaction: Authorization, Checking with the bank if funds are available, Capture, Settlement, Refunds and chargebacks.
The System will authenticate users, validate card details, and communicate with banking networks to complete payment securely.

3. Functional Requirements:

- FR1: The System shall authenticate users before processing payments.
- FR2: The System shall validate credit card number, expiration, date and CVV.
- FR3: The System shall process transactions in real time.
- FR4: The System shall generate transaction logs and provide receipts to customers.
- FR5: The System shall support refunds and chargebacks.
- FR6: The System shall integrate with external payment gateways and banking networks.

4. Interface Requirements:

- User Interface:** A Simple and intuitive web or mobile interface for customers to make payments, and dashboard for administrators to monitor and manage transactions.
- API Interface:** RESTful APIs for integrating payments, checking transaction status and handling disputes.

Budget:

Category	Amount (Rs)
Requirements and Design	5,00,000
Development	5,00,000
Testing and Quality Assurance	2,00,000
Deployment and Infrastructure	3,00,000
Maintenance	1,00,000
Total	16,00,000

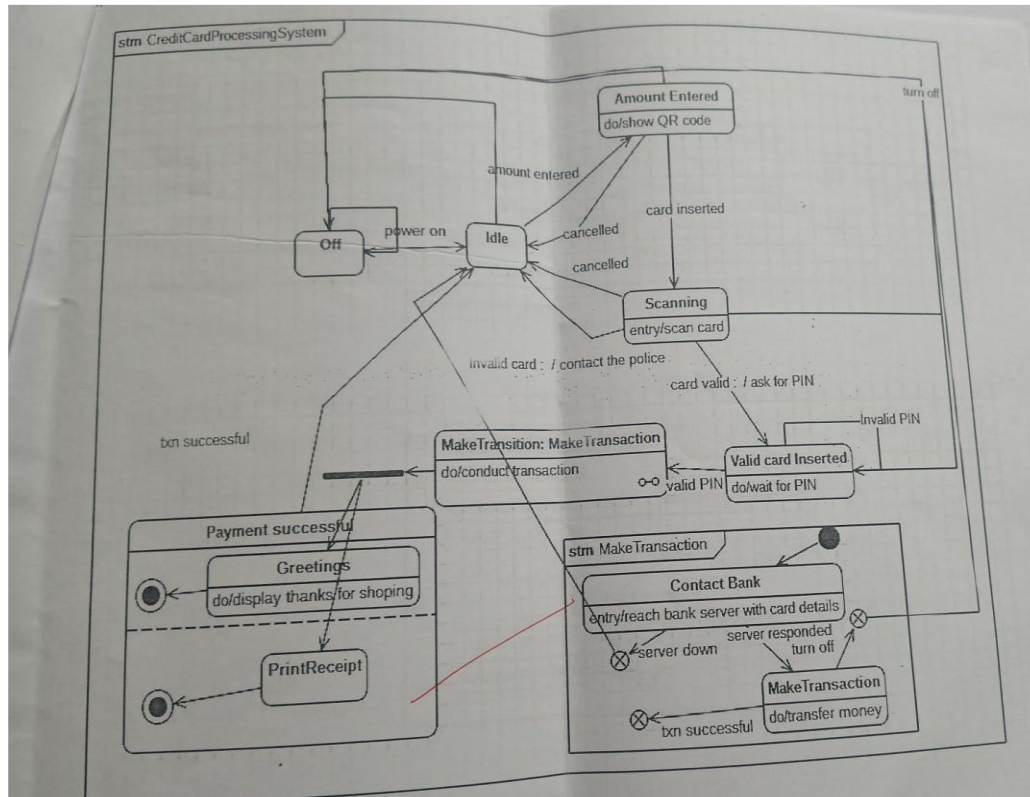
15/10/21

```
classDiagram
    class Merchant {
        nameId: String
        email: String
        password: String
        address: String
        +merchant(transaction: Transaction): Boolean
        +getMerchantReport(): Report
    }
    class ChargeCard {
        nameId: String
        emailCode: String
        +chargeCard: Date
        +makeChargeCard() void
    }
    class Transaction {
        transactionId: String
        amount: Double
        currency: String
        status: TransactionStatus
        +transaction: Date
        +refund(): Boolean
        +reverse(): Boolean
    }
    class Card {
        +cardNumber: String
        +expDate: Date
        +expYear: String
        +cardName: String
        +validate(): Boolean
        +makeChargeCard() String
    }
    class CardNetwork {
        networkId: String
        name: String
        +network()
        +validateNetwork(transaction: Transaction): Boolean
        +accumulateChargeFee(transaction: Transaction): Double
    }
    class Bank {
        +bankId: String
        +name: String
        +country: String
        +getDetails(): String
        +connectPayment(): Boolean
    }
    class PaymentGateway {
        +gatewayId: String
        +name: String
        +supportedCurrencies: List<String>
        +forwardTransaction(transaction: Transaction): Boolean
        +notifyMerchant(transaction: Transaction): void
    }
    class AcquirerBank {
        +acquirerCurrency: String
        +routeTransaction(transaction: Transaction): Boolean
        +verifyTransactionPayment(): void
    }
    class IssuerBank {
        +issuerCode: String
        +authorizeTransaction(transaction: Transaction): Boolean
        +voidCardCard: Card() void
    }
    class BackEngine {
        +engineId: String
        +name: String
        +makeChargeCard: String
        +approveTransaction(transaction: Transaction): Double
        +rejectTransaction(transaction: Transaction): Boolean
    }
    Merchant "1" -- "1..*" Transaction : +refuses
    Transaction "1..*" -- "1..*" CardNetwork : +networks
    Transaction "1" -- "1..*" PaymentGateway : +processes
    Transaction "1" -- "1..*" AcquirerBank : +acquires
    Transaction "1" -- "1..*" Bank : +requests
    Transaction "1" -- "1..*" IssuerBank : +requests
    Transaction "1" -- "1..*" BackEngine : +requests
    CardNetwork "1..*" -- "1..*" Bank : +requests
    CardNetwork "1..*" -- "1..*" IssuerBank : +requests
    CardNetwork "1..*" -- "1..*" BackEngine : +requests
    Bank "1..*" -- "1..*" IssuerBank : +requests
    Bank "1..*" -- "1..*" BackEngine : +requests
    IssuerBank "1..*" -- "1..*" BackEngine : +requests
```

16/10/2025

- This diagram displays object-oriented classes related to a digital payment processing system.
- Classes include Merchant, Card, Bank, Transaction, ChargeBack, and PaymentGateway.
- Each class contains attributes (such as card number, amount, ID) and methods (like authorize(), refund(), process()).
- The Merchant class interacts with the PaymentGateway to forward transactions.
- The Transaction class links multiple entities such as cards, banks, and gateways.
- IssuingBank and AcquiringBank specialize the Bank class, showing inheritance.
- The CardCheck class validates card information before a transaction proceeds.
- The ChargeBack class handles dispute or reversal scenarios.
- The RiskEngine class evaluates transaction safety and fraud detection.
- Associations, multiplicities, and direction arrows indicate communication flow between system components.

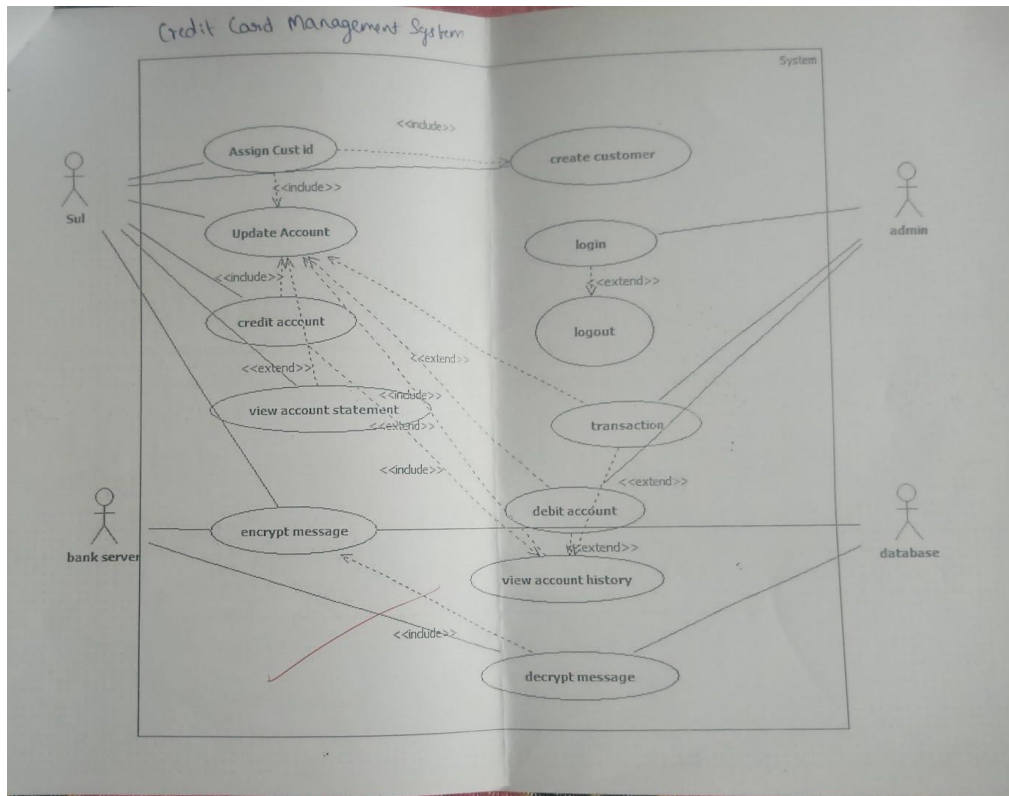
State Diagram



- This diagram shows the life cycle of a payment terminal, from power-off to transaction completion.
- The machine starts in the Off state and moves to Idle after powering on.
- From Idle, a user can enter an amount, which leads to the Amount Entered state.
- When a card is inserted, the system transitions to the Scanning state.
- If the card is invalid, the system issues an alert (e.g., “contact the police”).
- A valid card proceeds to Valid Card Inserted, awaiting a PIN entry.
- Upon entering a valid PIN, the machine shifts to Make Transaction.
- Inside this state, the Contact Bank sub-state attempts to reach the bank server.
- If the bank approves, the transaction is successful and the system goes to Payment Successful.

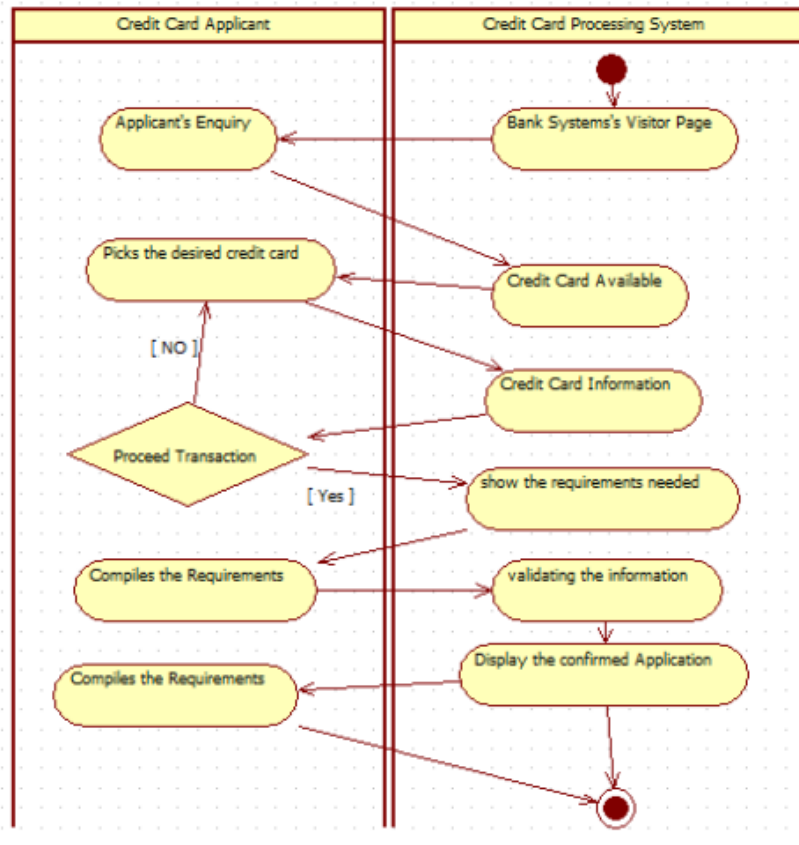
- Finally, the system prints a receipt and displays a thank-you message before returning to Idle.

Use Case Diagram:



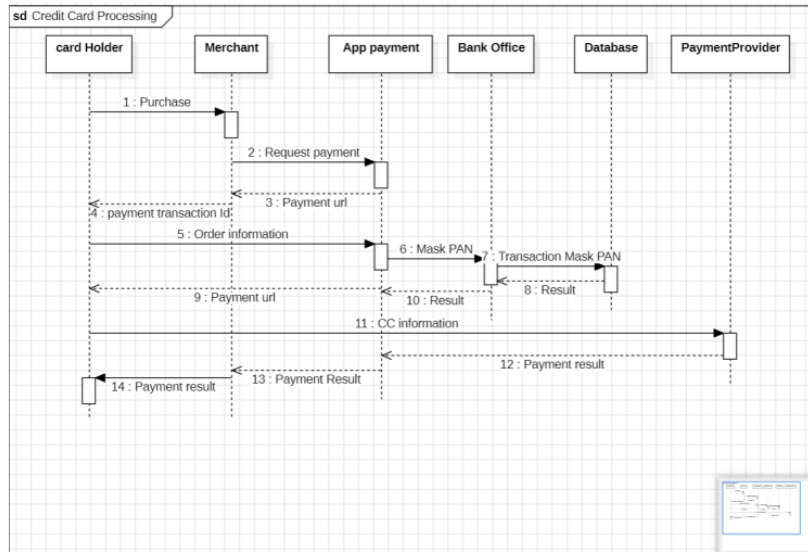
- The diagram identifies actors: Admin, database, staf, bank server.
- The Applicant interacts with the system to apply for a credit card.
- The applicant must submit details, upload documents, and fill the application form.
- The system allows checking application status.
- The applicant can also request for transaction details or credit card information.
- The Bank Administrator manages back-end activities.
- The admin can verify applicant details and validate submitted documents.
- The admin also approves or rejects the credit card request.
- Admin can update the credit card database and records.
- The use case diagram gives a high-level overview of all system-level interactions.

Activity Diagram



- The activity diagram has two swimlanes: Credit Card Applicant and Credit Card Processing System.
- The workflow starts when the applicant submits data or enters the home page.
- The applicant begins by providing personal and employment details.
- The system then retrieves the credit card information.
- The applicant proceeds to submit required documents.
- The system checks whether all documents and data are complete.
- If incomplete, the process returns to the applicant to fulfill missing requirements.
- If complete, the system performs verification and adds information to internal records.
- The system then evaluates the application for approval or rejection.
- Finally, the activity ends with the system displaying the processed application status.

Sequence Diagram



- The process begins when the Card Holder initiates a purchase with the Merchant.
- The Merchant then sends a Request for Payment to the App Payment system.
- The App Payment system responds with a Payment URL, which the Merchant forwards to the Card Holder.
- The Merchant also sends Order Information and Transaction ID for further processing.
- The App Payment system requests the Bank Office to mask the PAN (card number).
- The Bank Office sends the masked PAN to the Database for verification.
- The Database processes and returns the Transaction Masked PAN back to the Bank Office.
- The Bank Office then responds to the App Payment system with the Verification Result.
- The App Payment system sends the Payment URL to the Merchant for confirmation.
- After the payment is processed, the result flows from the Payment Provider → App Payment → Merchant → Card Holder as a Payment Result.

3. Library Management System

SRS-Software Requirements Specification

Management system. It outlines the systems functionalities, constraints and also performance expectations to ensure efficient development and maintenance.

1.2 Scope of Document:

This document describes the Software System designed to manage library operations such as book issuing, returns, catalog management, and member services. It is also intended for use by developers, testers and stakeholders to understand what the system should achieve.

1.3 Overview:

This document is organized into sections covering the general system description, functional and non-functional requirements, user design, system performance and project planning.

2. General Description:

The Library management system is a application that is designed to automate routine tasks in a library such as issuing, returning, cataloging and reserving books. It will also manage our accounts, fines and inventory. The system will support administrators, librarians through different access level and roles.

3. Functional Requirements:

FR1: Register and manage user accounts.

FR2: Add, update, and delete book records.

FR3: Issue and return books with due date tracking.

FR4: Calculate and manage late return fines.

FR5: Search and reserve books by title, author or category.

FR6: Generate reports.

FR7: Authentication and role based access control.

4. Interface Requirements:

- User Interface: Web-based GUIs for librarians, admin and students.
- Database Interface: Relational database for storing user and book data.
- External Interface: Optional integration with barcode.

5. Performance Requirements:

System must handle many concurrent users.

Search results should display fastly.

Daily backup of library database must be performed automatically.

6. Design Constraints:

- Development using open source tools.
- Must comply with institutional data privacy policies.
- Accessible via web browser on desktop and mobile devices.

3. Library Management System:

Problem Statement:

Libraries are important part of educational and professional institutions, but managing them manually is often inefficient and may be prone to errors. And many issues like misplaced records, difficulty in tracking issued books and also delays in due date. And this System helps in improving these factors and also in reducing the administrative burden staff.

1. Introduction:

1.1 Purpose of document:

The purpose of document is to define the requirements for the development of a library

7. Non Functional Requirements:

- a. Security: User login, encrypted passwords and access control.
- b. Reliability: Minimal downtime and consistent data availability.
- c. Usability: Easy to navigate interface for all user levels.
- d. Maintainability: Modular architecture for easy update.
- e. Portability: Compatible with major browsers and operating system.

8. Preliminary Schedule and Budget.

phase 1: Requirement Gathering and Planning (1 week)

phase 2: System Design (1 week)

phase 3: Development (4 weeks)

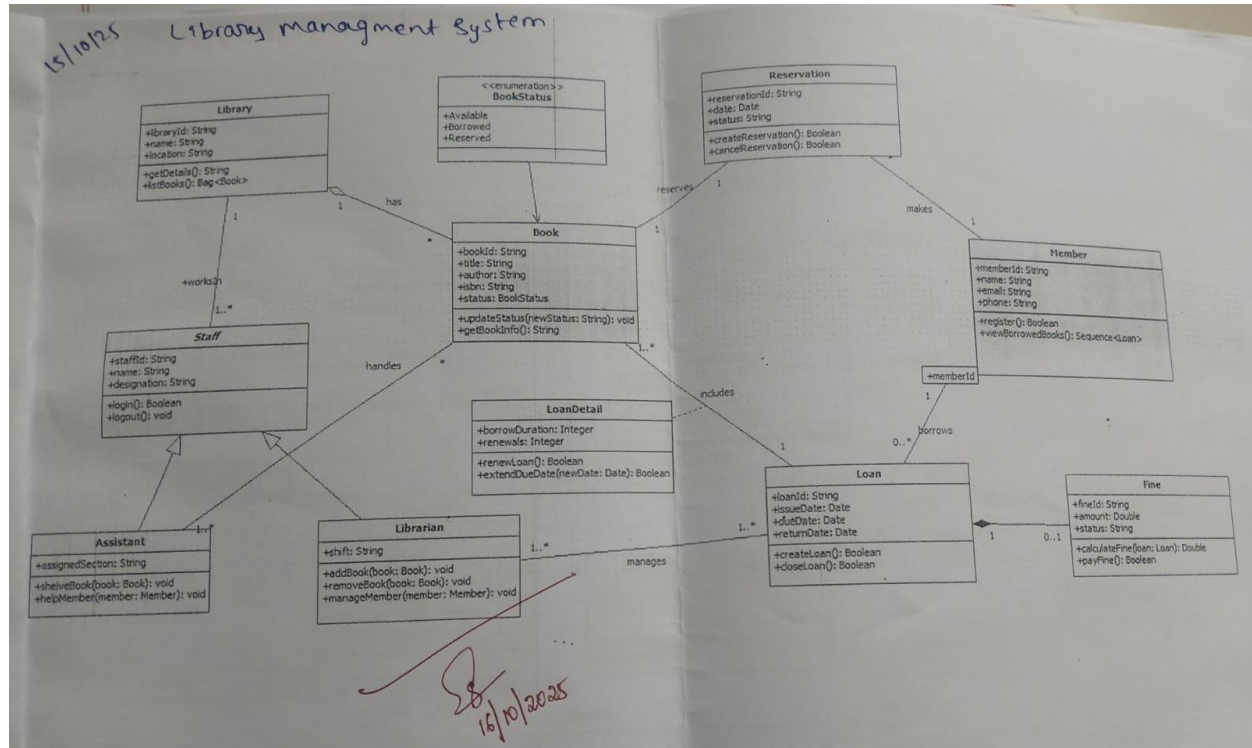
phase 4: Testing and Debugging (2 weeks)

phase 5: Deployment and Training (1 week)

Budget:

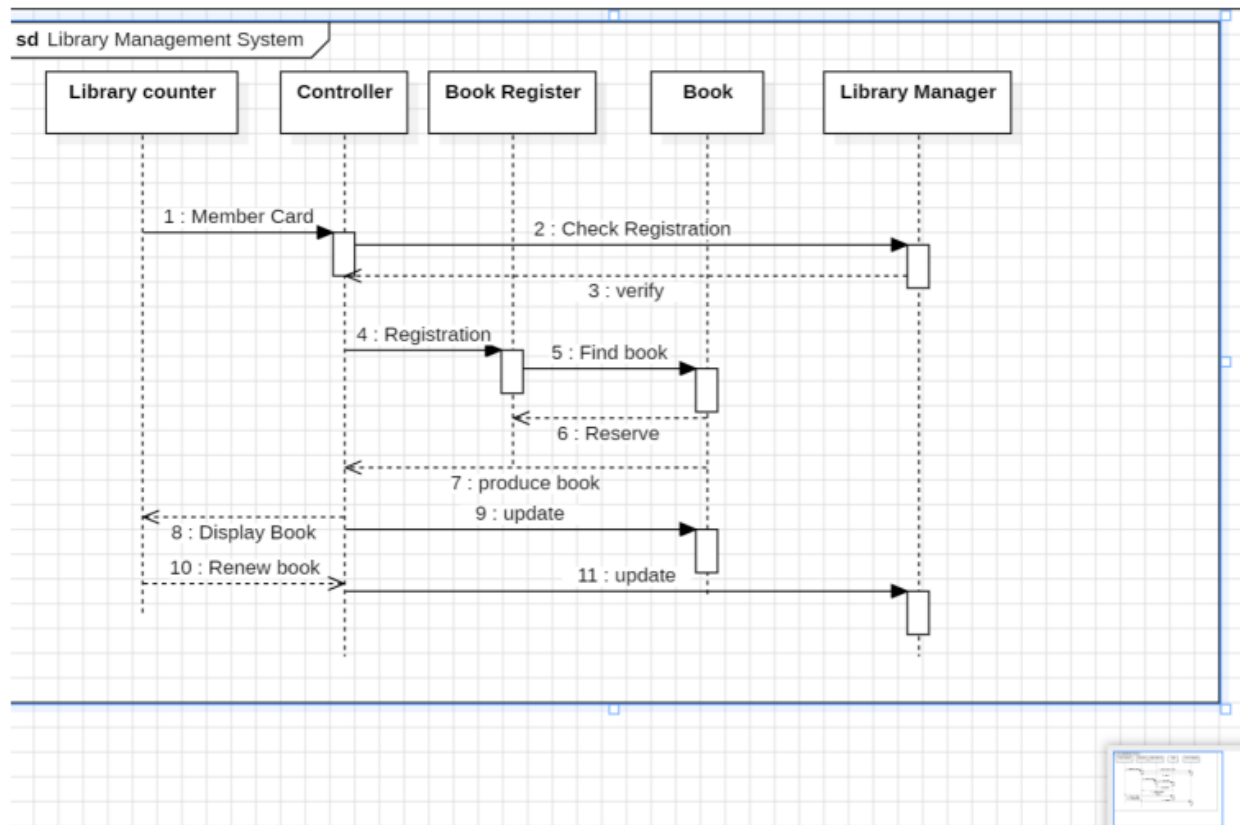
Requirement and Design	600,000 ₹
Development	400,000 ₹
Testing and Quality Assurance	250,000 ₹
Deployment and infrastructure maintenance	30,000 ₹
	100,000 ₹
Total:	1,380,000 ₹

Class Diagram



- This is a Class Diagram showing the structure of the library system.
- It defines main classes like User, Librarian, Book, Loan, Author, Publisher, Category etc.
- Each class contains its attributes (data fields) and sometimes methods.
- The User class represents students or members who borrow books.
- The Librarian class is responsible for managing books and issuing them.
- The Book class stores details like title, ISBN, author, and category.
- The Loan class records issuing details such as issue date and return date.
- There are relations like Aggregation (Book–Author), Association (User–Loan), and Inheritance.
- Multiplicity (1..*, 0..1) describes how many objects relate to each other.
- The diagram provides the static structure and database blueprint of the library system.

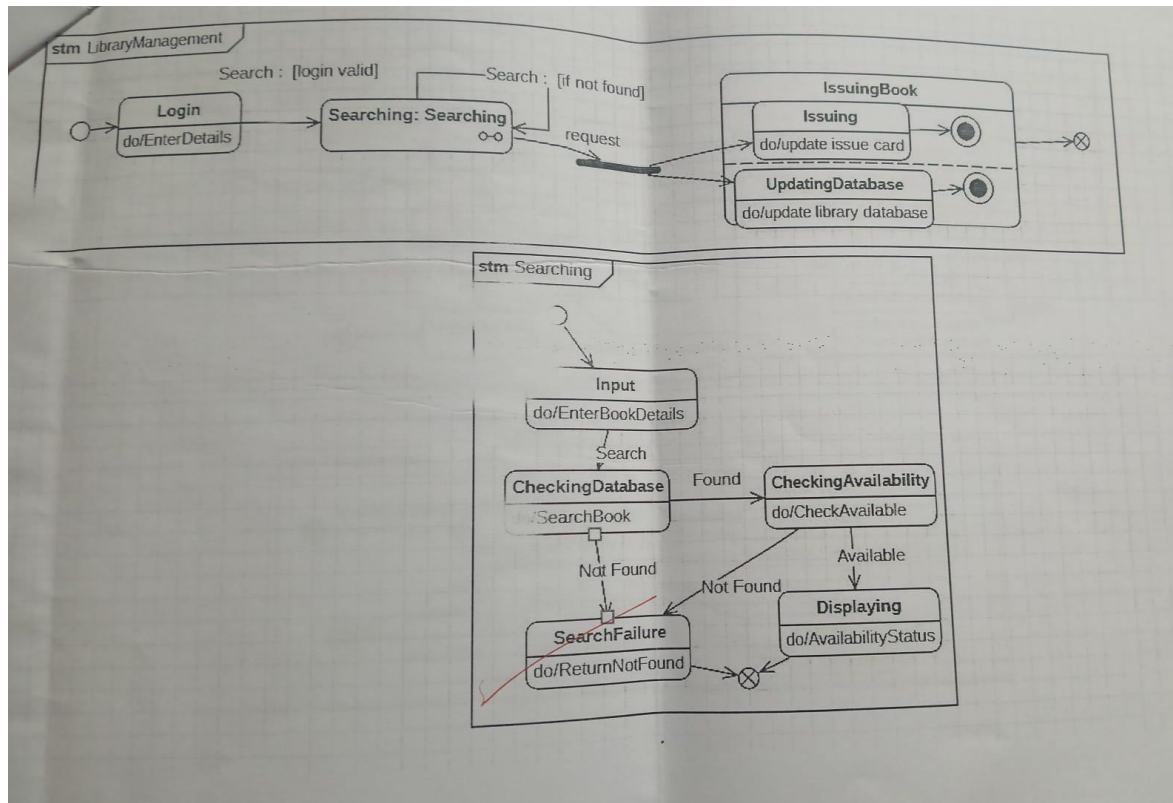
Sequence Diagram



- This Sequence Diagram shows communication among Library Counter, Controller, Book Register, Book, and Library Manager.
- The library counter begins by reading the Member Card.
- The controller checks member registration status with the Book Register.
- The register verifies and returns result.
- The user requests to find a book, and the controller forwards the request.
- The Book component sends the book details and confirms reservation.
- The Book Register updates the reservation entry.
- The Library Manager also updates the overall library database.
- The sequence ends with displaying the book or renewing it.

- The diagram shows the exact order of messages during book issue or renewal.

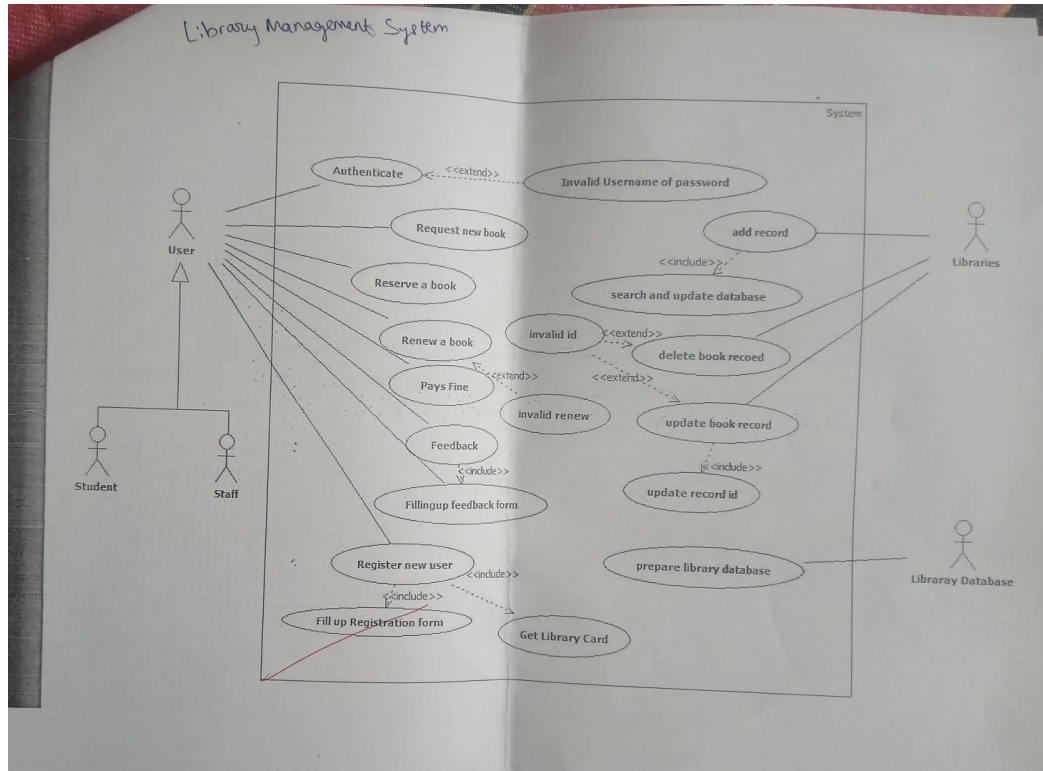
State Diagram



- This Activity Diagram shows the workflow of searching and issuing a book.
- The process starts with a login activity, verifying user credentials.
- After login, the user enters the book details to search.
- The system checks availability by querying the database.
- If the book is found, availability is displayed to the user.
- If not found, the system displays a "Search Failure" message.
- If available, the process moves to the Issuing Book use case.
- Issuing book updates the Library Database.

- The diagram highlights both successful and unsuccessful paths.
- It demonstrates the full cycle of search → check → issue → update.

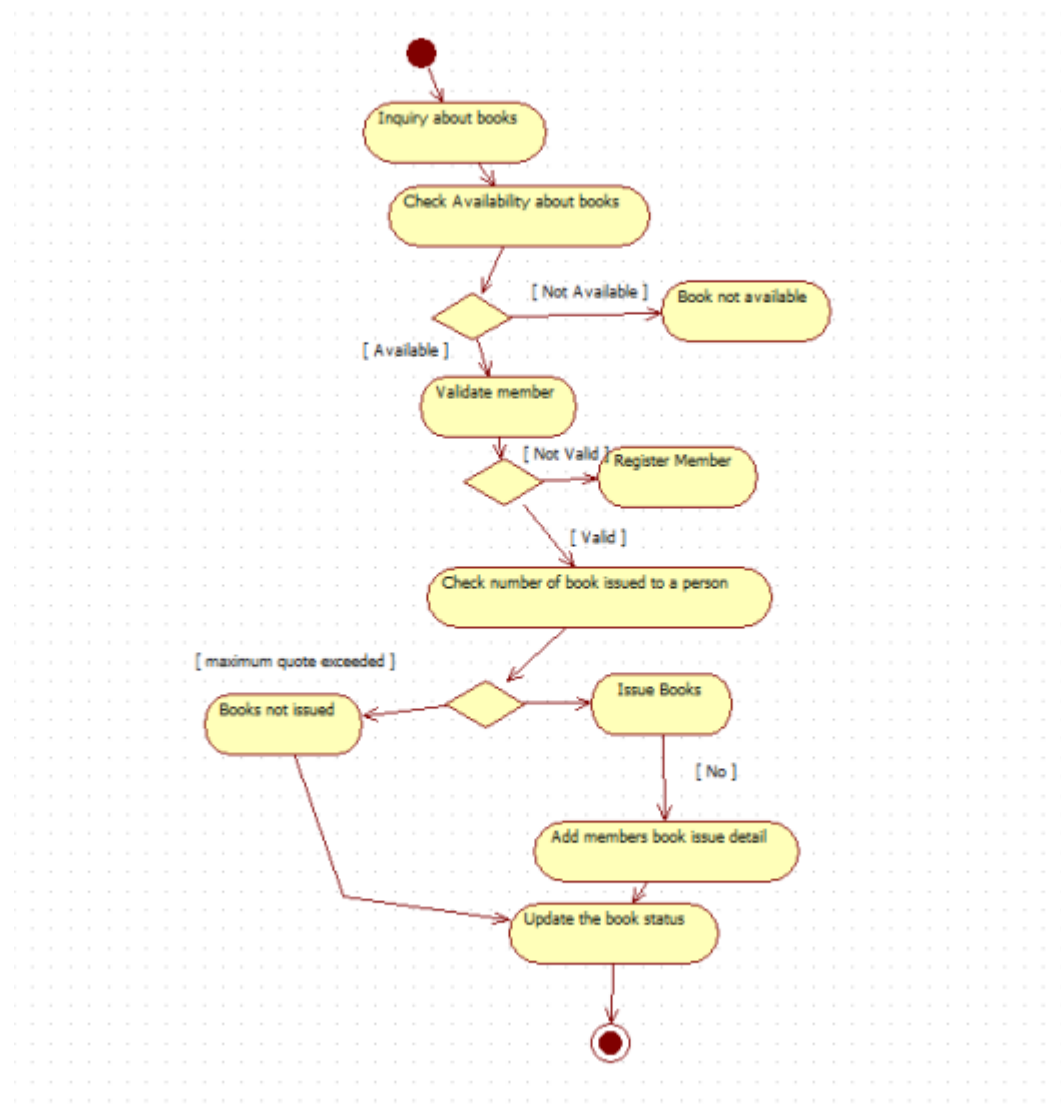
UseCase Diagram



- This Use Case Diagram has two actors: Student and Librarian.
- Students can perform use cases like Search Books, Borrow Books, Return Books, View Account Details etc.
- They can also request for a library card, renew books, and give feedback.
- The Librarian actor manages the backend operations of the library.
- Librarian handles adding new books, updating inventory, and deleting books.
- They also maintain the library database, manage fines, and handle authentication.
- The diagram shows communication lines between actors and each use case.
- Some use cases are connected (extend/include), showing dependency.

- It represents the functional requirements of the system.
- Overall, this diagram gives a full functional view of what the library system must support.

Activity Diagram



- The activity diagram begins when a user makes an inquiry about books.
- The system first checks the availability of the requested book.
- If the book is not available, the process ends with a “Book not available” message.
- If the book is available, the system proceeds to validate the member.

- If the member is not valid, the system registers the person as a new member.
- If the member is valid, the system checks how many books the person has already borrowed.
- A decision point determines whether the member has exceeded the maximum borrowing limit.
- If the limit is exceeded, the process moves to “Books not issued.”
- If the limit is not exceeded, the system issues the books and records the issuance in the member’s details.
- Finally, the book status is updated in the system, completing the process.

4. Stock management System

SRS-Software Requirements Specification

4. Stock Maintenance System

Problem Statement:
Managing stock manually often leads to errors, overstocking or stockout, which can impact negatively on sales, increase costs and cause customer dissatisfaction. So this system automates inventory control, improves accuracy, and ensures optimal stock levels across the organisation.

1. Introduction

1.1 Purpose of Document:
The purpose of document is to define the requirements for stock maintenance system that outlines the system core functionalities, interface, performance expectations and constraints to guide the design, development and deployment phase.

1.2 Scope of Document:
This document covers the system that manages inventory tracking, stock updates, item registration, supplier details and reporting. The system is intended for use in retail shops, warehouse or any organisation that manages product inventory. It does not give overall look over the system.

1.3 Overview:
This document includes general system description, functional and non-functional requirements, interface specifications, performance design constraints, and preliminary schedule and budget.

2. General Description:
The Stock Maintenance System automates inventory control by tracking the product quantities, locations, transactions and supplier information. This system will allow users to update stock levels, generate reports, set minimum stock levels and analyse product movements.

3. Functional Requirements:

- FR1: Add, update and delete product entries.
- FR2: Record stock-in and stock-out transactions.
- FR3: Track supplier details and purchase history.
- FR4: Set minimum stock levels and send low stock alerts.
- FR5: Generate inventory, transaction and reminder reports.
- FR6: Role-based access for admin and stock managers.
- FR7: Maintain audit logs for stock changes.

Interface Requirements:

- User Interface: Web-based dashboard for Stock Managers and Admins.
- Database Interface: Centralised inventory database with real-time updates.

5. Performance Requirements:

- a) Handle many products and concurrent transactions.
- b) Transaction update time should be fast i.e. less than second.
- c) And also time should be real-time report and also fast data check.

6. Design Constraints:

- a) Should be developed using standard technologies.
- b) System must have accessibility with the help of web browser.
- c) It should align with organisational security and audit policies.

7. Non Functional Attributes:

- Security: Role based authentication and secure data storage.
- Reliability: Automatic backup and minimal downtime.
- Usability: Clean intuitive interface for quick training and adoption.
- Maintainability: Modular codebase for easy maintenance.
- Interoperability: Support for future integration with billing.

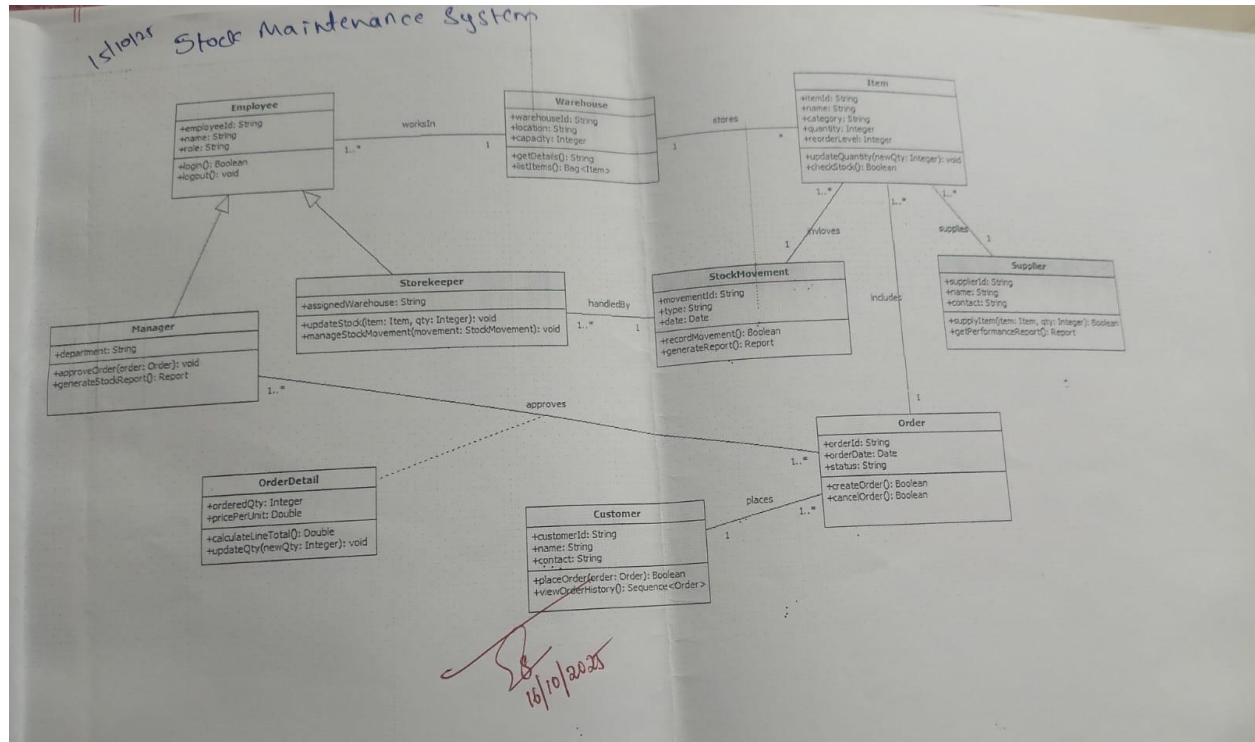
8. Preliminary Schedule and Budget:

Phase 1: Requirement Analysis	1 week
Phase 2: Design	1 week
Phase 3: Development	3 weeks
Phase 4: Testing	1 week
Phase 5: Deployment and Testing	1 week

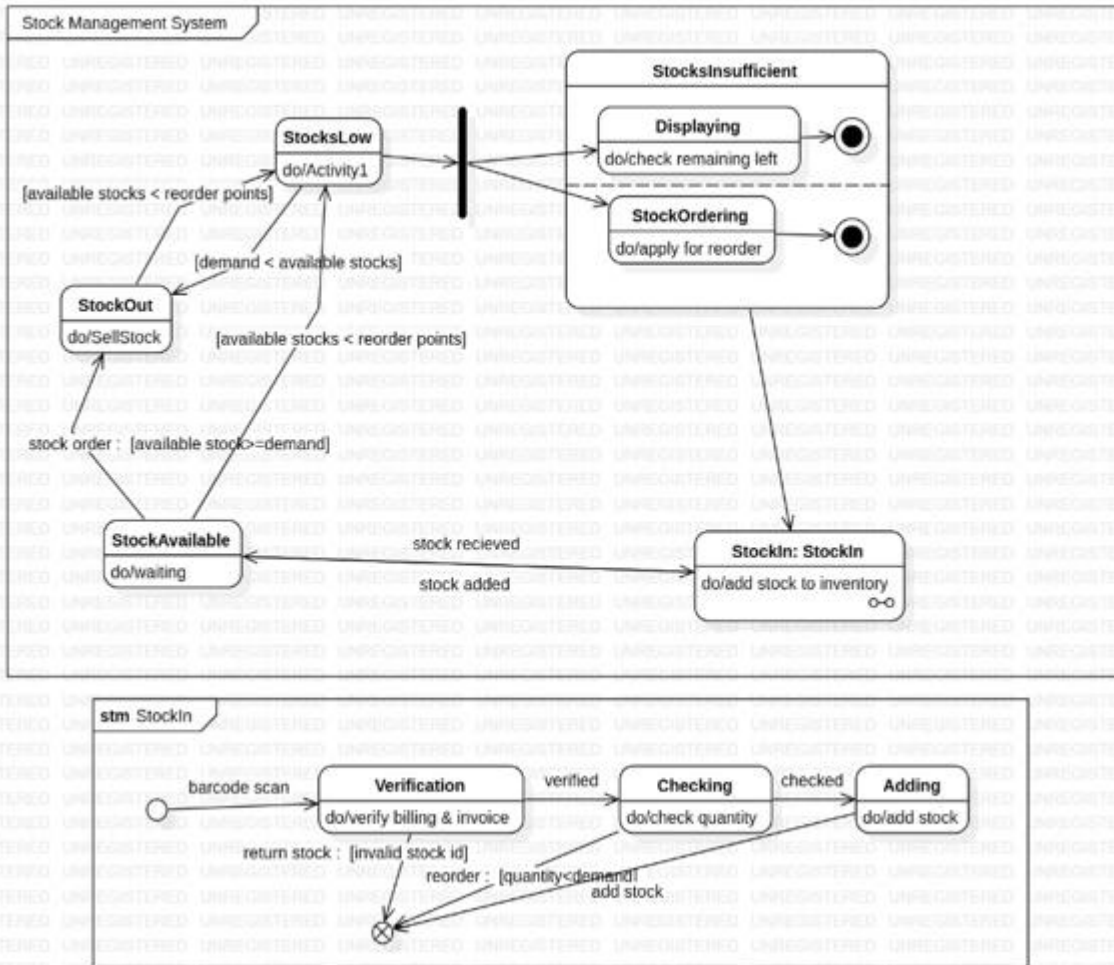
Budget:

Requirement and Design	500000
Development	400000
Testing and Quality Assurance	200000
Deployment and Infrastructure	200000
Maintenance	100000
Total	1800000

Class Diagram



- Employee is the parent class for Manager and Storekeeper.
- Manager approves orders and generates reports.
- Storekeeper manages stock and works inside a Warehouse.
- Warehouse stores multiple Items.
- Item has details like name, price, quantity.
- Supplier provides items and updates supply details.
- Customer places Orders.
- Order contains OrderDetails and tracks status.
- OrderDetail stores item quantity and line total.
- StockMovement tracks movement of items between warehouse and storekeeper



The system manages inventory levels and reordering processes through several interconnected states:

StockAvailable - The initial state where the system is waiting for activity. The system monitors available stock levels continuously.

StockOut - Triggered when a stock order occurs and available stock falls below demand (condition: [available stocks > demand]). The system performs a do/SellStock action in this state.

StocksLow - Activated when available stocks drop below the reorder point (condition: [available stocks < reorder points]). This state executes a do/Activity1 action and leads to the StocksInsufficient concurrent region.

StocksInsufficient - A composite state with two parallel regions that execute simultaneously:

- **Displaying:** Shows a message to check remaining stock left (do/check remaining left) before terminating

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

Both parallel activities must complete before the system can transition to the next state.

StockIn: StockIn - This submachine state handles the process of receiving new inventory. When stock is received, it transitions back to StockAvailable. When stock is added, it also returns to StockAvailable.

Submachine: StockIn

This detailed subprocess manages the stock receiving workflow through three sequential states:

Verification - Verifies billing and invoice documents (do/verify billing & invoice). If stock is invalid, it returns to the start; otherwise, proceeds when verified.

Checking - Checks the quantity of incoming stock (do/check quantity). Can trigger reorder if quantity doesn't match demand, or add stock if quantity matches demand.

Adding - Adds the verified stock to inventory (do/add stock), then completes the process.

The diagram illustrates a comprehensive inventory management workflow that monitors stock levels, triggers reordering when necessary, and carefully processes incoming inventory through verification steps

Stock Management System

SMS →

```

    usecaseDiagram
        actor Admin
        actor Shop
        actor Trader

        Admin --> UC_Login
        Admin --> UC_ValidateCredentials
        Admin --> UC_ForgotPassword
        Admin --> UC_ViewStockStatus
        Admin --> UC_PurchaseStock
        Admin --> UC_Report
        Admin --> UC_Logout
        Admin --> UC_TransferStock

        Shop --> UC_CheckStockAvailability
        Shop --> UC_FilterByCategory
        Shop --> UC_ApplyDiscount
        Shop --> UC_CancelPurchase
        Shop --> UC_Validation
        Shop --> UC_CheckStockStatus
        Shop --> UC_InventoryCheck
        Shop --> UC_UpdateDatabase
        Shop --> UC_NewStockRecord

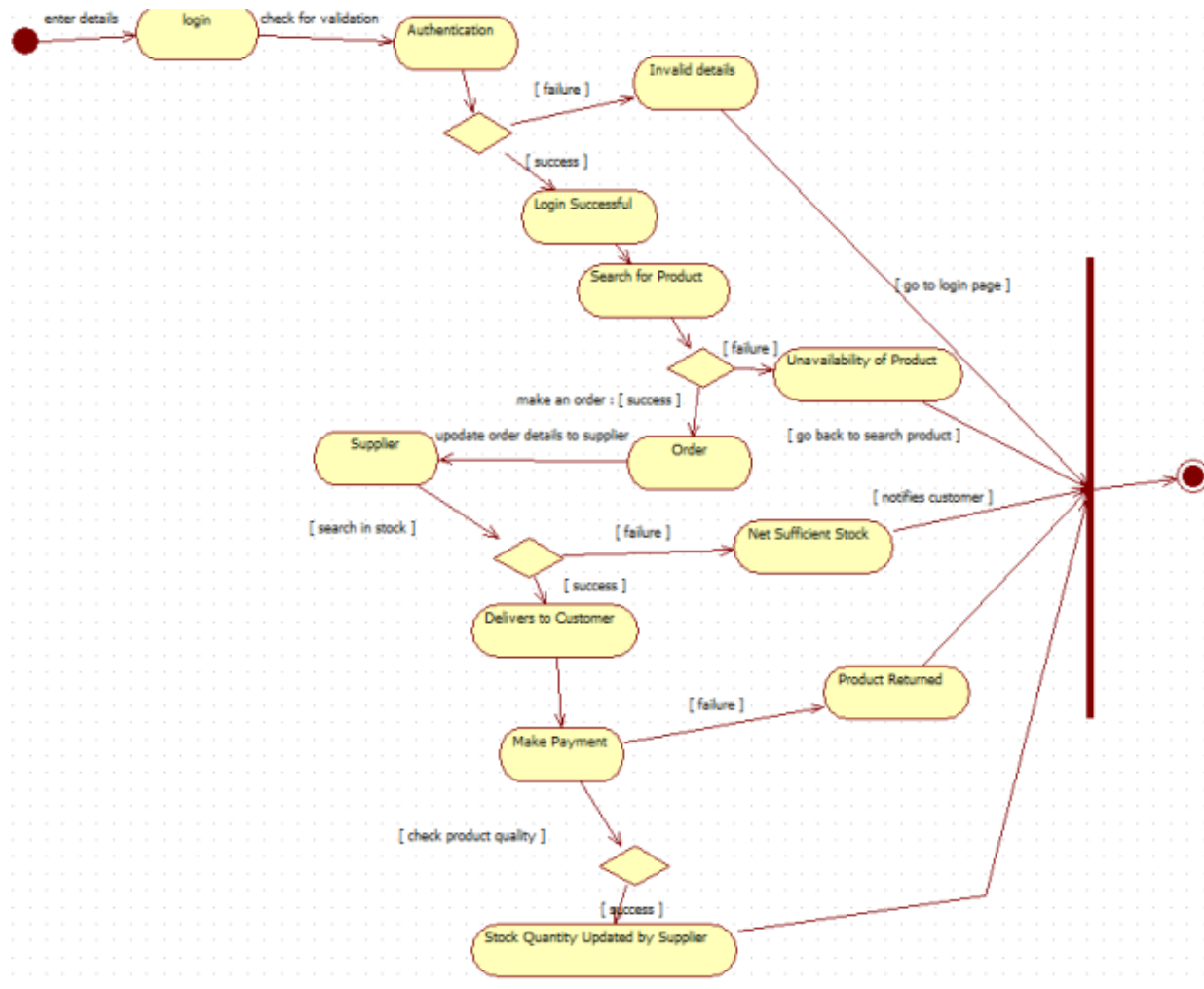
        Trader --> UC_SellStock
        Trader --> UC_ReturnOrderStock
        Trader --> UC_CancelOrder
        Trader --> UC_CheckOrderStockStatus

        UC_Login ..> UC_ValidateCredentials : <<include>>
        UC_Login ..> UC_ForgotPassword : <<include>>
        UC_ValidateCredentials ..> UC_ForgotPassword : <<include>>
        UC_ForgotPassword ..> UC_ViewStockStatus : <<include>>
        UC_ViewStockStatus ..> UC_PurchaseStock : <<include>>
        UC_PurchaseStock ..> UC_Report : <<include>>
        UC_PurchaseStock ..> UC_CancelPurchase : <<include>>
        UC_PurchaseStock ..> UC_Validation : <<include>>
        UC_Report ..> UC_CheckStockStatus : <<include>>
        UC_Report ..> UC_InventoryCheck : <<include>>
        UC_Report ..> UC_UpdateDatabase : <<include>>
        UC_Report ..> UC_NewStockRecord : <<include>>
        UC_Logout ..> UC_Login : <<include>>
        UC_TransferStock ..> UC_PurchaseStock : <<include>>
        UC_TransferStock ..> UC_Report : <<include>>
        UC_TransferStock ..> UC_SellStock : <<include>>
        UC_TransferStock ..> UC_ReturnOrderStock : <<include>>
        UC_TransferStock ..> UC_CancelOrder : <<include>>
        UC_TransferStock ..> UC_CheckOrderStockStatus : <<include>>
        UC_CheckStockAvailability ..> UC_FilterByCategory : <<include>>
        UC_FilterByCategory ..> UC_ApplyDiscount : <<include>>
        UC_ApplyDiscount ..> UC_CancelPurchase : <<include>>
        UC_CancelPurchase ..> UC_Validation : <<include>>
        UC_Validation ..> UC_CheckStockStatus : <<include>>
        UC_Validation ..> UC_InventoryCheck : <<include>>
        UC_Validation ..> UC_UpdateDatabase : <<include>>
        UC_Validation ..> UC_NewStockRecord : <<include>>
        UC_CheckStockStatus ..> UC_InventoryCheck : <<include>>
        UC_InventoryCheck ..> UC_UpdateDatabase : <<include>>
        UC_UpdateDatabase ..> UC_NewStockRecord : <<include>>
        UC_NewStockRecord ..> UC_SellStock : <<include>>
        UC_NewStockRecord ..> UC_ReturnOrderStock : <<include>>
        UC_NewStockRecord ..> UC_CancelOrder : <<include>>
        UC_NewStockRecord ..> UC_CheckOrderStockStatus : <<include>>
        UC_SellStock ..> UC_ReturnOrderStock : <<include>>
        UC_ReturnOrderStock ..> UC_CancelOrder : <<include>>
        UC_CancelOrder ..> UC_CheckOrderStockStatus : <<include>>
    
```

- Admin, User, and Trader are main actors.
- Admin can log in, validate credentials, and manage inventory.
- User can search products, filter, apply discounts, and purchase stock.
- System checks stock availability before purchase.
- Payments and refunds are handled within the system.
- Trader adds new stock and checks order status.
- System supports inventory checks and stock validation.
- Users can view stock status anytime.

- System allows order cancellation.
- All actions are connected to the central Stock Management System.

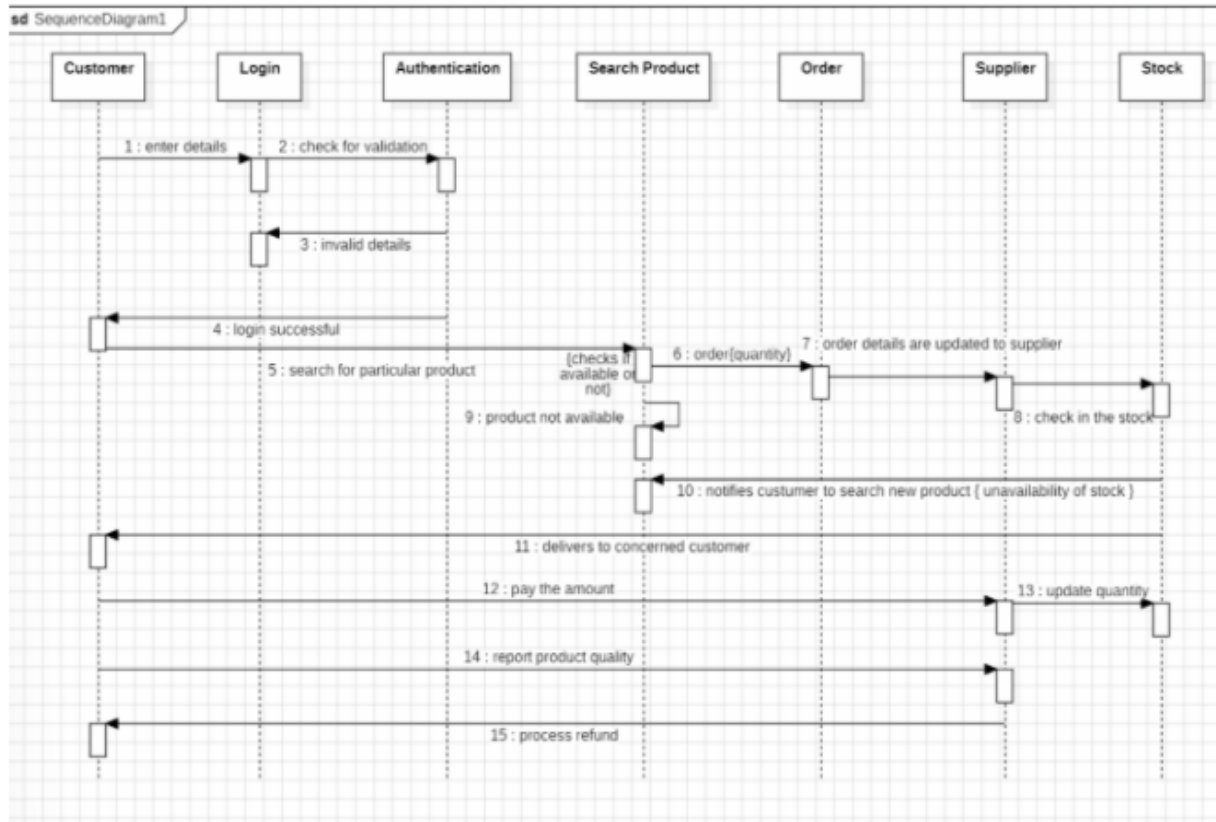
Activity Diagram



- User enters login details.
- System authenticates; success continues, failure ends.
- User searches for a product.
- If unavailable, user is notified.
- If available, user places an order.
- Supplier updates order details.
- Supplier delivers the product to customer.
- Customer makes payment.
- If product quality fails, it is returned.

- Supplier updates stock quantity after completion.

Sequence Diagram



- The sequence diagram models a customer's journey through login, product search, ordering, and post-purchase steps.
- It starts when the customer enters login details, which are passed to the Login module.
- The Login module requests the Authentication service to validate the credentials.
- If the credentials are invalid, an error message is returned to the customer.
- If the login succeeds, the customer proceeds to search for a product.
- The Search Product module checks product availability by querying the Stock component.
- If the product is unavailable, the customer is notified to choose another product.
- If available, the customer places an order specifying quantity, which is sent to the Order module.

- The Order module forwards order information to the Supplier, who then updates the stock.
- The process finishes with delivery, payment, quality reporting, and a refund option if required.

5.Passport Authentication Automation System

SRS-Software Requirements Specification

⑤ Passport Authentication Automation System :

Problem Statement :
The normal passport application processes are time-consuming and are more prone to manual errors and often require multiple visits to offices. So the goal of this system is to digitize and automate the end-to-end passport application workflow, ensuring faster processing, greater transparency and improved user experience for applicants and officers.

1. Introduction :

1.1 Purpose of Document
The purpose of Document is to specify the requirements for a Passport Automation System, which aims to streamline the process of applying for, verifying and issuing passports. It provides detailed information for system developers, stakeholders, and testers to ensure a secure, efficient, and user-friendly system.

1.2 Scope of Document :
The document outlines the development of a web-based passport automation system that handles user registration, document submission, application processing, verification, appointment scheduling, and status tracking. It does not include how the system works.

1.3 Overview :
This document presents the system's general description, functional and interface requirements, performance expectations, dependencies, and project planning details.

2. General Description :
The passport Automation System will allow citizens to apply for passports online, reducing manual paperwork and in-person visits. It will include user registration, form submission, payment, scheduling appointments, and real-time tracking of application status. The system will also support administrative roles for document verification, background checks, and passport issuing.

3. Functional Requirements :

FR1 : User Registration with unique ID and secure login.
FR2 : Online passport application form submission.
FR3 : Upload required documents.
FR4 : Online fee payment integration.
FR5 : Appointment scheduling for biometrics and verification processes.
FR6 : Admin interface for reviewing applications and updating status.

FR7 : Notification System.
FR8 : Report generation.

4. Interface Requirements :
User Interface : Responsive web UI for applicants and admin.
Database Interface : Secure backend for storing user data and documents.
External Interface : APIs for payment gateway and SMS/email service.

5. Performance Requirements :
a) System must handle many concurrent users.
b) Page should load under 2 seconds.
c) Application processing should be completed in less time.

6. Design Constraints :
a) Must follow government security and privacy standards.
b) Should be built using secure frameworks.
c) Must be browser-compatible and mobile responsive.

7. Non-Functional Attributes :
Security : Secure login, encrypted data storage, role-based database access.
Reliability : Automatic data backups and session recovery.
Usability : Simple and intuitive interface with minimal tech skills.
Maintainability : Modular code for future upgrades.
Compliance : Adherence to national digital governance guidelines.

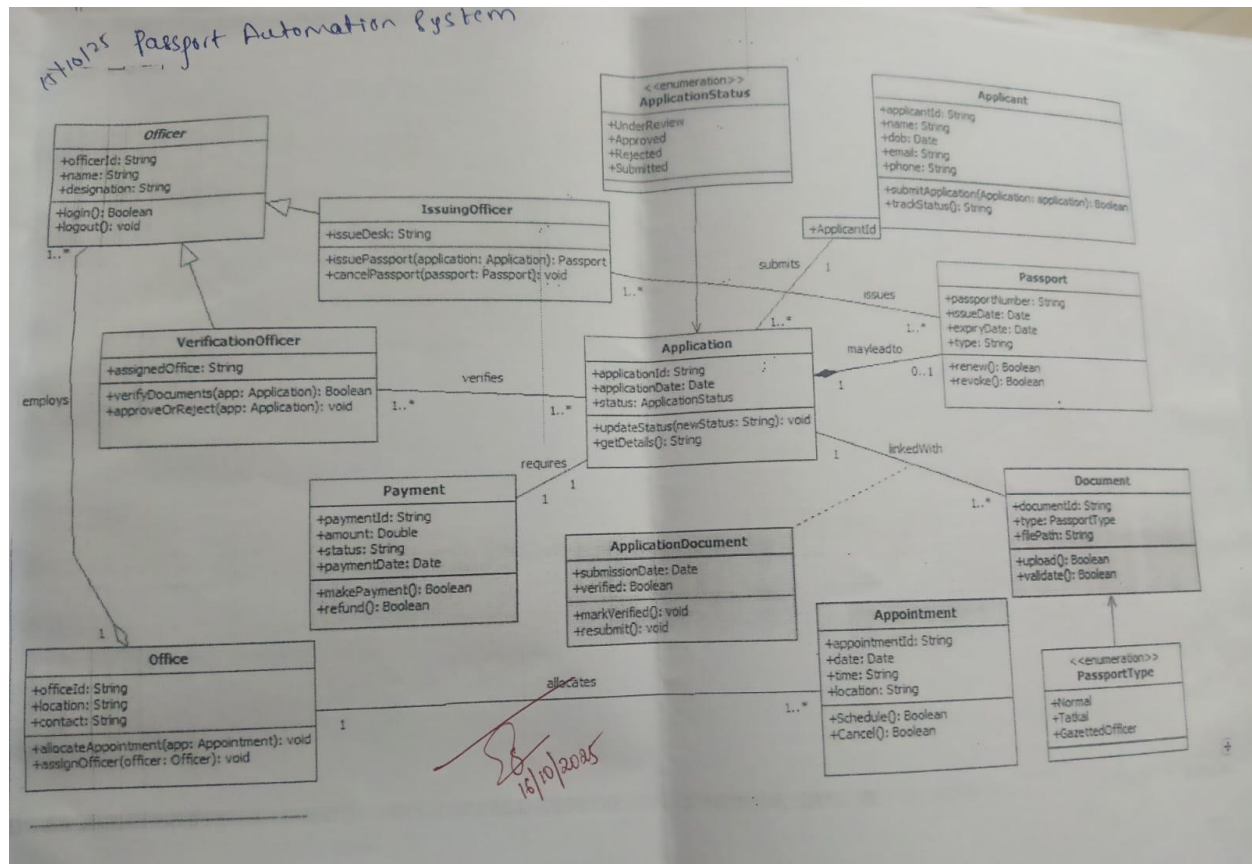
8. Preliminary Schedule and Budget :

Phase 1 : Requirement Gathering & Analysis 1 week
Phase 2 : UI/UX Design 1 week
Phase 3 : Development 4 weeks
Phase 4 : Testing & Security Audit 2 weeks
Phase 5 : Deployment & User Training 1 week

Budget :

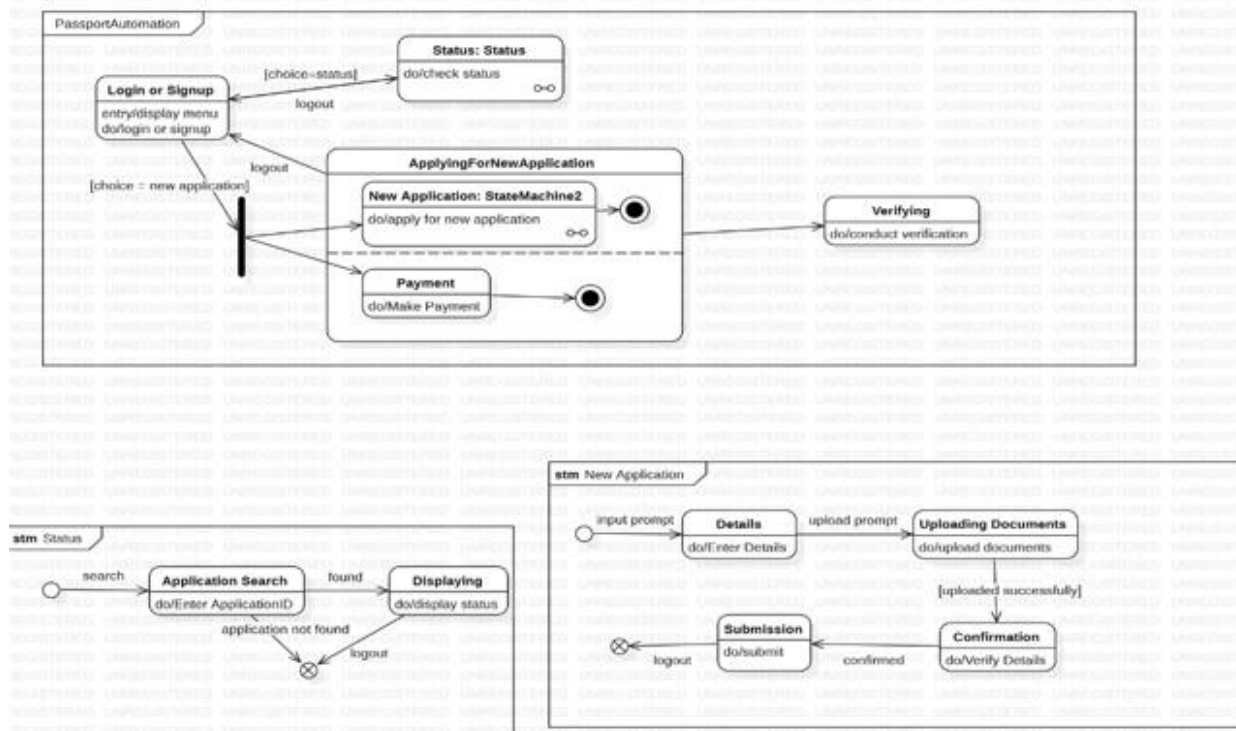
Requirement and Design	500,000 Rs
Development	400,000 Rs
Testing and Quality Assurance	350,000 Rs
Deployment and Infrastructure	250,000 Rs
Maintenance	150,000 Rs
Total	1,650,000 Rs

Class Diagram



- The class diagram represents all key components involved in automating a passport application process.
- The Applicant class stores personal information and initiates passport applications.
- The Application class records application ID, type, status, and related operations.
- Officers are modeled as classes such as Officer, VerificationOfficer, and IssuingOfficer.
- VerificationOfficer checks submitted documents and approves or rejects applications.
- IssuingOfficer handles issuing and canceling passports after verification.
- The Passport class maintains passport details like issue date, expiry, and status.
- ApplicationDocument records submitted documents and their verification status.
- The Payment class stores transaction details and includes methods for payment and refunds.
- Appointments and Documents are linked to the Application class, forming a complete workflow from submission to passport issuance.

State Diagram

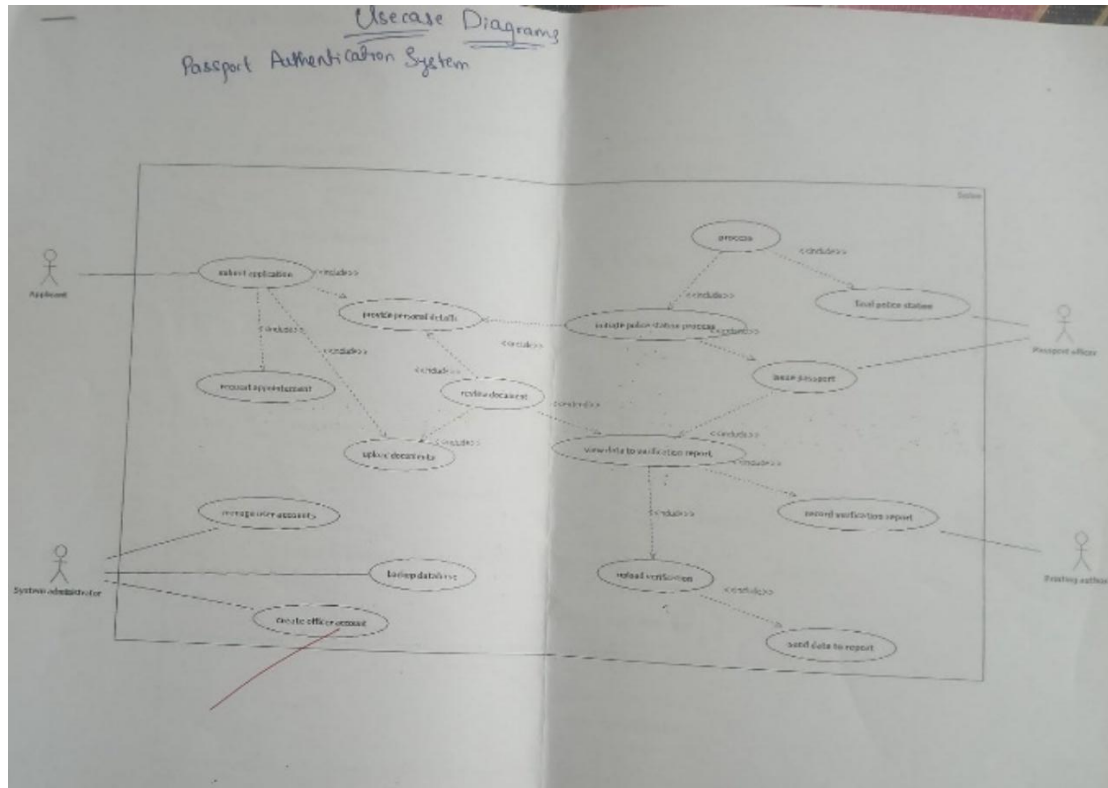


Main PassportAutomation State Machine: The process begins at "Login or Signup" where users access the menu. A choice point determines if this is a new application (logout triggers this path). The system enters "ApplyingForNewApplication" composite state containing two concurrent regions: "New Application: StateMachine2" (for application details) and "Payment" (for fee processing). Both must complete before reaching their final states. A "Status: Status" state allows checking application status and can loop back. The "Verifying" state conducts document verification after application submission.

stm Status Submachine: This handles application tracking. Starting from an initial state, users search for their application, transitioning to "Application Search" (entering ApplicationID). If found, the system moves to "Displaying" state (showing status); if not found, it returns to the initial state via a "logout" signal.

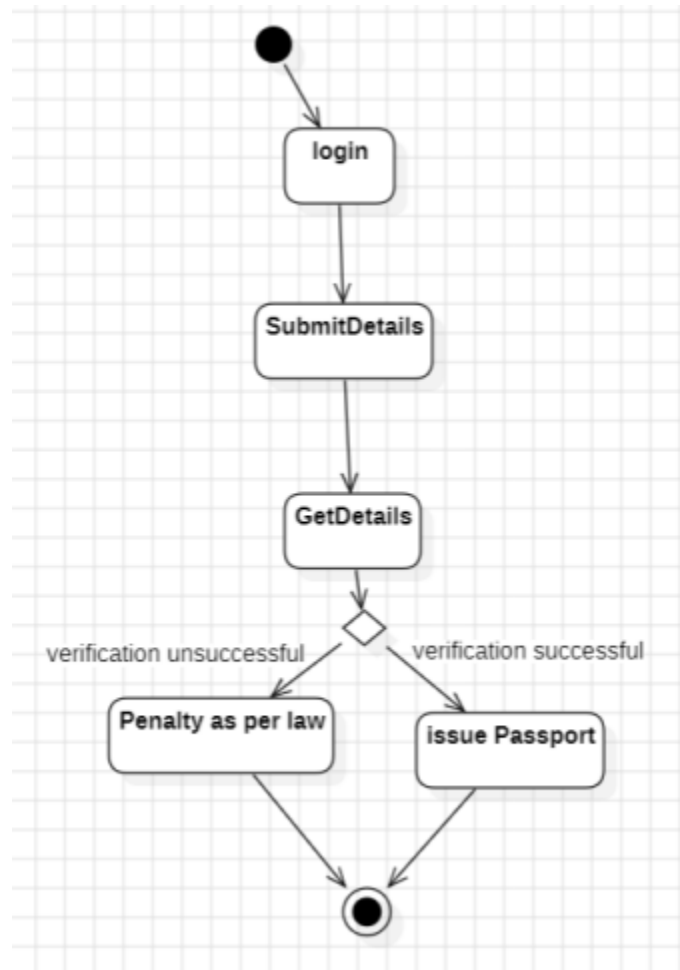
stm New Application Submachine: This manages the application creation process. From the input prompt, users enter "Details" (filling information), then receive an upload prompt to "Uploading Documents" (attaching required documents). Upon successful upload, the flow moves through a logout point to "Submission" (submitting data), then to "Confirmation" (verifying details) where confirmation completes the process.

UseCase Diagram



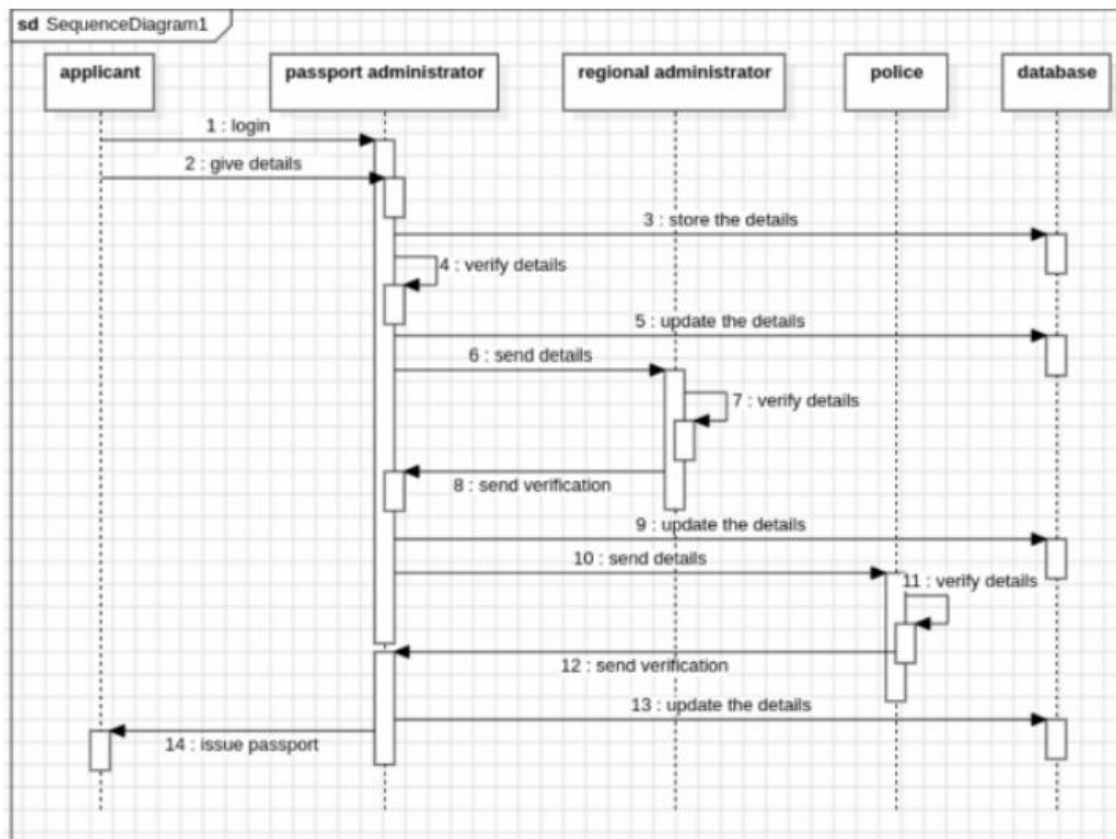
- The use-case diagram shows how different users interact with the passport automation system.
- The Applicant actor performs tasks such as submitting an application and booking an appointment.
- The Passport Officer handles processing the application and issuing the passport.
- The Police Authority performs background verification of the applicant.
- The System Administrator manages system updates and backend operations.
- Use cases include apply for passport, submit documents, schedule appointment, and receive passport.
- Police verification and document checking are additional important use cases.
- Extend relationships show optional steps like sending verification reports or requesting additional documents.
- Include relationships represent mandatory steps such as verifying identity before issuing a passport.
- The diagram gives an overview of the entire passport process and interactions among different actors.

Activity Diagram



- The activity diagram begins with the applicant starting the passport application process.
- The first action performed is logging into the system.
- After a successful login, the applicant proceeds to submit their details.
- The system receives these details and moves to the "GetDetails" activity.
- A decision node evaluates whether the submitted details are valid.
- The diagram then splits into two paths based on verification results.
- If verification is successful, the system proceeds to issue the passport.
- If verification is unsuccessful, a penalty is applied as per legal requirements.
- Both the successful and unsuccessful paths eventually merge back to the end node.
- The diagram overall represents a simple flow of login, submission, verification, and outcome.

Sequence Diagram



- The sequence diagram shows interactions among the applicant, passport administrator, regional administrator, police, and database.
- The applicant initiates the process by logging in.
- The applicant then provides details to the passport administrator.
- The passport administrator sends these details to the database for storage.
- The regional administrator retrieves the details and begins verification.
- After verifying, the regional administrator updates the database with the results.
- The passport administrator forwards the details to the police for further verification.
- The police verify the information and send the verification results back.
- The police also update the database with their verification outcome.
- After receiving all verifications, the passport administrator completes the process by issuing the passport.