B.M.S. COLLEGE OF ENGINEERING BENGALURU

Autonomous Institute, Affiliated to VTU

Lab Record

Object-Oriented Modelling

Submitted in partial fulfillment for the 5th Semester Laboratory

Bachelor of Engineering in
Computer Science and Engineering

Submitted by:

Harsha B (1BM23CS107)

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 Analysis and Design(22CS6PCSEO) laboratory has been carried out by Harsha B (1BM23CS107) during the 5th Semester August 2025-December 2025

.

Signature of the Faculty Incharge

Name of the Your Batch Incharge and designation

Roopashree S

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

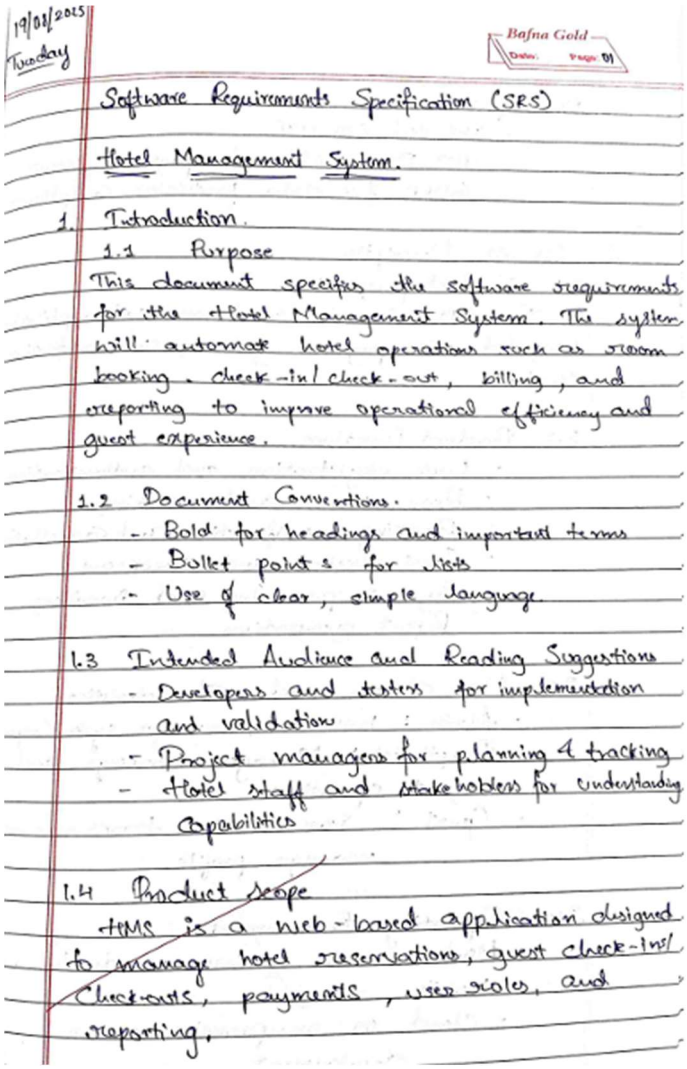
Table of Contents

1. Hotel Management System

Problem Statement

In many hotels, daily operations such as room booking, customer check-in/check-out, billing, and room availability tracking are still handled manually or using multiple unconnected systems. This often leads to data inconsistency, booking errors, time delays, and difficulty in generating accurate reports.

SRS:



19/08/2025
Tuesday

Bafna Gold

Software Requirements Specification (SRS)

Hotel Management System.

1. Introduction.

1.1 Purpose

This document specifies the software requirements for the Hotel Management System. The system will automate hotel operations such as room booking, check-in/check-out, billing, and reporting to improve operational efficiency and guest experience.

1.2 Document Conventions.

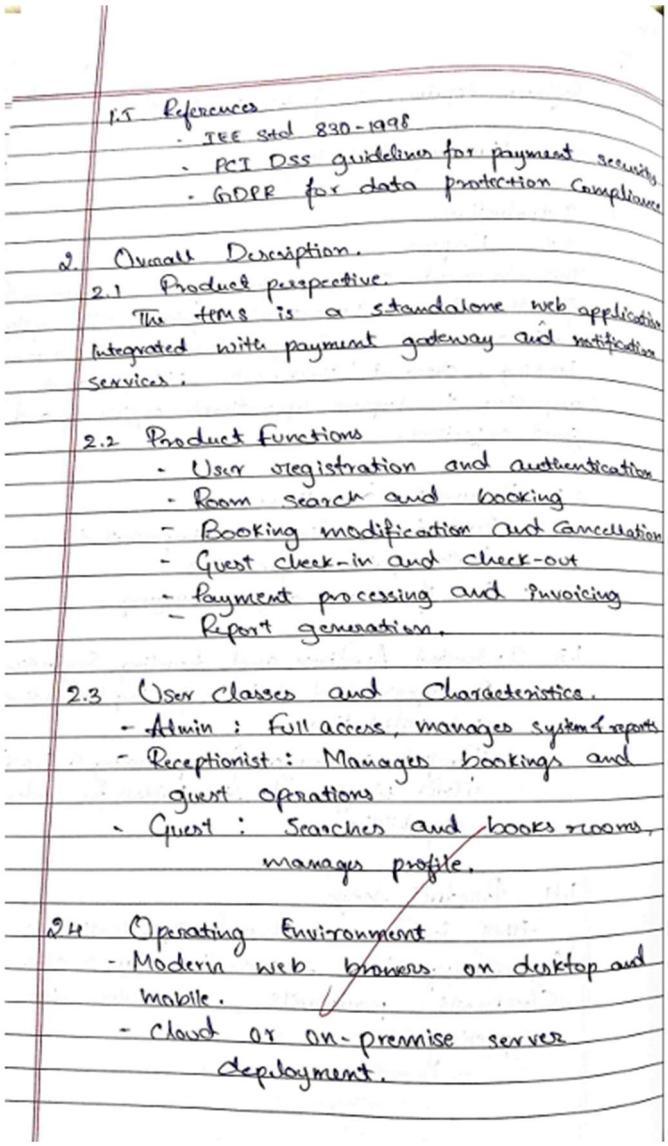
- Bold for headings and important terms
- Bullet points for lists
- Use of clear, simple language.

1.3 Intended Audience and Reading Suggestions

- Developers and testers for implementation and validation
- Project managers for planning & tracking
- Hotel staff and stakeholders for understanding capabilities

1.4 Product Scope

HMS is a web-based application designed to manage hotel reservations, guest check-ins/check-outs, payments, user roles, and reporting.



1.5 References

- IEEE std 830-1998
- PCI DSS guidelines for payment security
- GDPR for data protection Compliance

2. Overall Description.

2.1 Product perspective.

The HMS is a standalone web application integrated with payment gateway and notification services.

2.2 Product Functions

- User registration and authentication
- Room search and booking
- Booking modification and cancellation
- Guest check-in and check-out
- Payment processing and invoicing
- Report generation.

2.3 User classes and Characteristics.

- Admin : Full access, manages system & reports
- Receptionist : Manages bookings and guest operations
- Guest : Searches and books rooms, manages profile.

2.4 Operating Environment

- Modern web browsers on desktop and mobile.
- Cloud or on-premise server deployment.

2.5 Design and Implementation Constraints
- Secure payment processing (PCI DSS compliance)
- Data encryption in transit and at rest
- Role-based access control.

2.6 User Documentation
- User manuals and online help
- Admin and staff training guides.

2.7 Assumptions and Dependencies
- Internet connection for payment and notifications
- Third-party API's availability.

3. Specific Requirements
3.1 Functional Requirements
- Secure user registration and login
- Role-based access control.
- Room availability search by date, type and price.
- Book, modify and cancel reservations.
- Check-in and check-out processing
- Process payments and generate receipts
- Manage user profiles.
- Admin reports on bookings, revenue, and occupancy.
- Notifications via email / SMS

3.2 External Interface Requirements
- User-friendly web UI
- Payment gateway integration (Stripe, PayPal)
- Email and SMS notification APIs
- Secure HTTPS communication

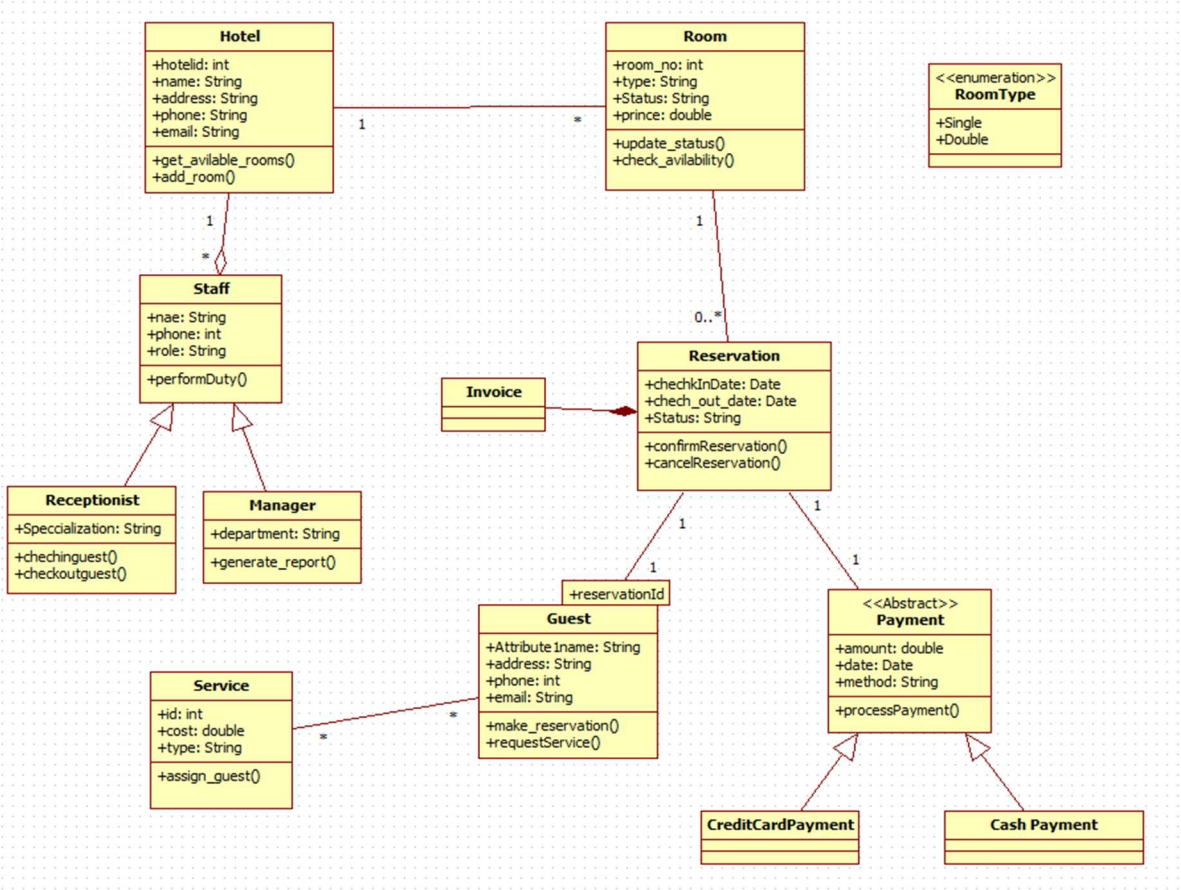
3.3 System Features
- Room Management: add / update / delete rooms and availability
- Booking Management: full booking lifecycle handling
- User Management: roles and access controls
- Payment Processing and Billing
- Reporting Dashboard.

3.4 Non-Functional Requirements
- Support 100+ concurrent users
- Response time under 2 seconds
- Data security and encryption
- 99.9% system uptime
- Scalable architecture for future expansion.
- Intuitive and responsive UI

4. Appendices
4.1 Glossary
- Booking: Reservation of room(s)
- Check-in: Guest arrival process
- Check-out: Guest departure and payment finalization

4.2 Future Enhancements
- Mobile apps
- Door lock system integration
- Loyalty programs
- Multi-language support

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_aviability() - 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

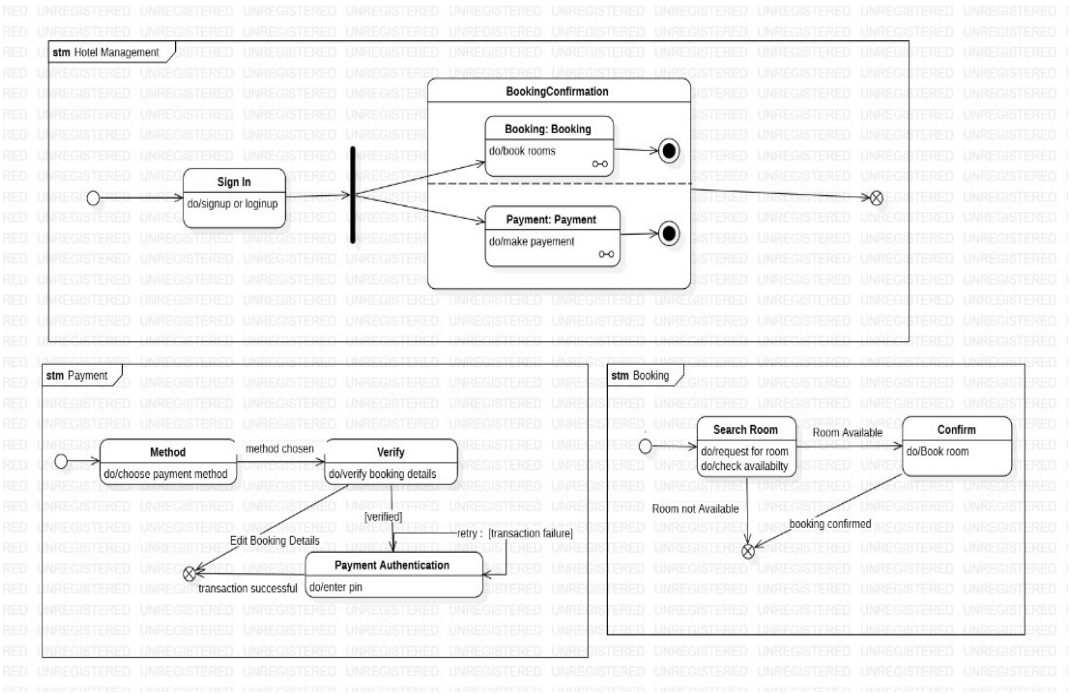
Payment <<Abstract>>: Abstract base class for all payment transactions.

- 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)

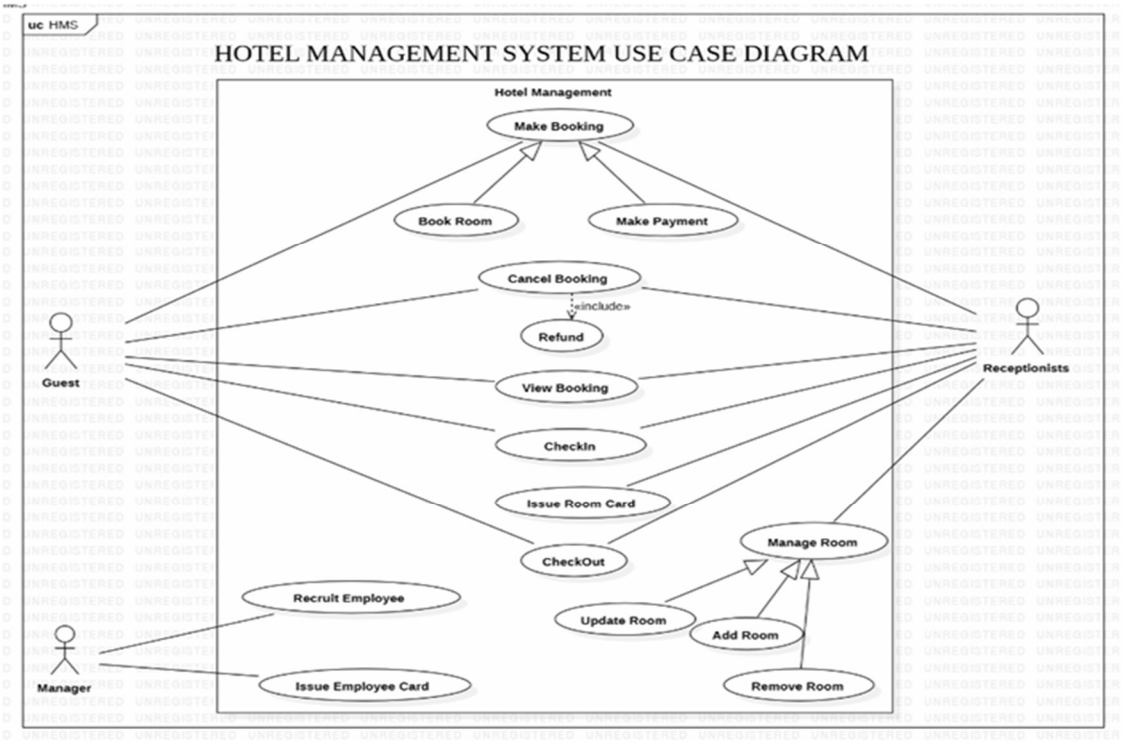
Search Room (Initial State): Room availability inquiry phase.

- 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** (⊗)

Use Case Diagram:



Actors:

1. Guest - Hotel customers who book and use hotel services
2. Receptionist - Front desk staff who manage guest services and room operations
3. Manager - Administrative staff who handle employee and room management

Use Cases by Actor:

Guest Can:

- Make Booking - Reserve a room at the hotel
- Book Room - Specific room reservation (part of booking process)
- Make Payment - Pay for services and accommodations
- Cancel Booking - Cancel an existing reservation
- Refund - Request money back for cancelled bookings
- View Booking - Check existing reservation details
- CheckIn - Register arrival at the hotel

- CheckOut - Complete stay and leave the hotel

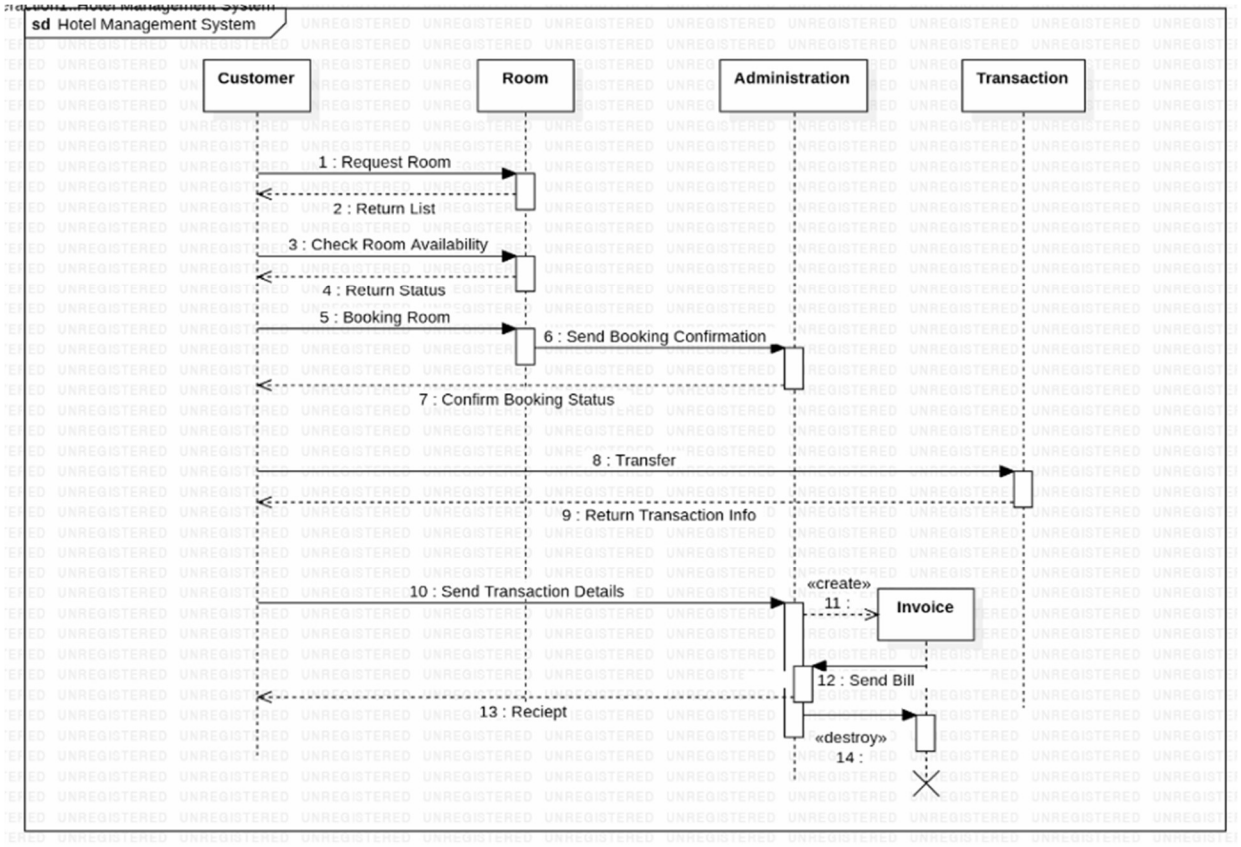
Receptionist Can:

- Make Booking - Create reservations on behalf of guests
- View Booking - Access and review booking information
- CheckIn - Process guest arrivals
- Issue Room Card - Provide key cards to guests
- CheckOut - Process guest departures
- Manage Room - Oversee room status and operations
- Update Room - Modify room information and status
- Add Room - Register new rooms in the system
- Remove Room - Delete rooms from the system

Manager Can:

- Recruit Employee - Hire new staff members
- Issue Employee Card - Provide identification/access cards to employees
- Manage Room - Supervise room inventory and operations
- Update Room - Modify room details
- Add Room - Add new rooms to inventory
- Remove Room - Remove rooms from the system

Sequence Diagram:



This is a **UML Sequence Diagram** for a **Hotel Management System** that illustrates the step-by-step interactions between different components during a room booking process.

Participants (Objects):

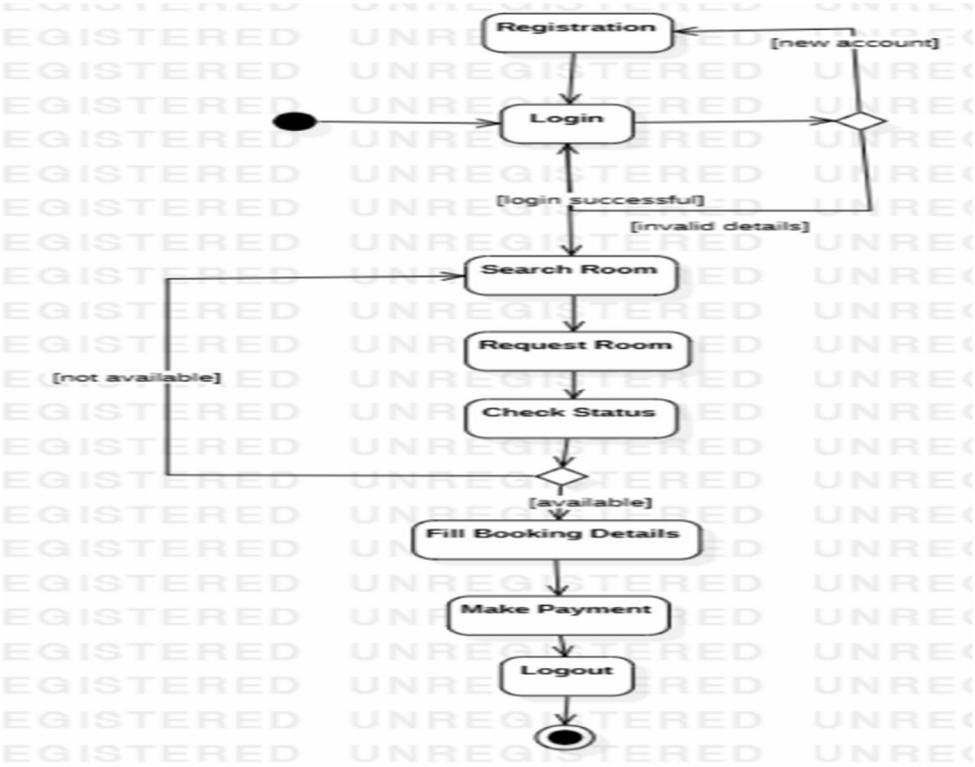
1. Customer - The guest making a booking
2. Room - The room inventory system
3. Administration - The administrative/booking management system
4. Transaction - The payment processing system
5. Invoice - A document object created during the process

Sequence Flow:

1. Request Room - Customer initiates a room search
2. Return List - Room system returns available rooms list
3. Check Room Availability - Customer checks specific room availability

4. Return Status - Room system confirms availability status
5. Booking Room - Customer books the selected room
6. Send Booking Confirmation - Room system notifies Administration
7. Confirm Booking Status - Administration confirms the booking back to Customer
8. Transfer - Customer initiates payment to Transaction system
9. Return Transaction Info - Transaction system returns payment confirmation
10. Send Transaction Details - Transaction sends payment details to Administration
11. «create» - Administration creates a new Invoice object
12. Send Bill - Invoice is sent back to Administration
13. Receipt - Customer receives the final receipt
14. «destroy» - Invoice object is destroyed (marked with X symbol)

Activity Diagram



1. Registration & Login Phase:

- Start (black filled circle) - Process begins

- Registration - New users create an account
    - [new account] - Path for first-time users
- Login - Users authenticate themselves
    - [login successful] - Proceeds if credentials are valid
    - [invalid details] - Loops back to Registration if login fails
- Decision diamond - Routes flow based on login success

2. Room Search & Selection:

- Search Room - User browses available rooms
- Request Room - User selects a specific room
- Check Status - System verifies room availability
    - [available] - Room is available, continues to booking
    - [not available] - Returns to Search Room to try another option

3. Booking & Payment:

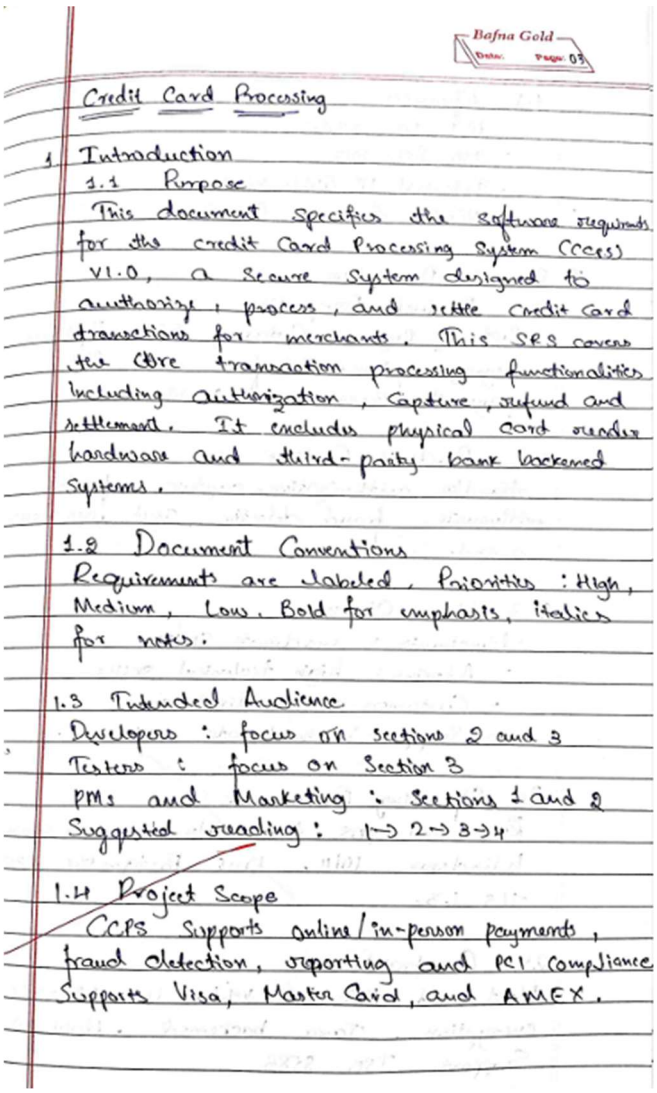
- Fill Booking Details - User enters reservation information (dates, guest details, etc.)
- Make Payment - User completes payment transaction
- Logout - User exits the system
- End (black filled circle with ring) - Process terminates

2. Credit Card Processing

Problem Statement

Traditional credit card management systems face significant challenges including manual processing inefficiencies, inadequate fraud detection, limited customer self-service capabilities, and poor real-time transaction tracking. These issues lead to increased operational costs, security vulnerabilities, customer dissatisfaction, and delayed payment processing. There is a critical need for an automated, secure, and user-friendly Credit Card Management System that enables real-time transaction monitoring, fraud prevention, seamless payment processing, and comprehensive account management for both cardholders and financial institutions.

SRS:



Bafna Gold

Credit Card Processing

1. Introduction

1.1 Purpose

This document specifies the software requirements for the credit Card Processing System (CCPS) V1.0, a secure system designed to authorize, process, and settle credit card transactions for merchants. This SRS covers the core transaction processing functionalities including authorization, capture, refund and settlement. It excludes physical card reader hardware and third-party bank backened systems.

1.2 Document Conventions

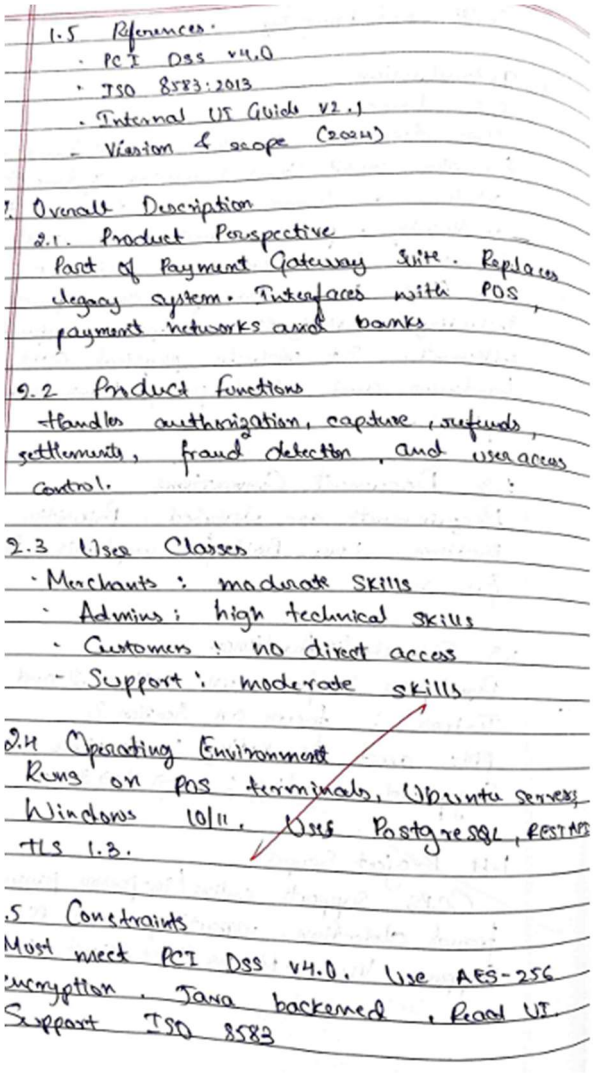
Requirements are labeled. Priorities: High, Medium, Low. Bold for emphasis, italics for notes.

1.3 Intended Audience

Developers: focus on sections 2 and 3
Testers: focus on Section 3
PMs and Marketing: Sections 1 and 2
Suggested reading: 1 → 2 → 3 → 4

1.4 Project Scope

CCPS supports online/in-person payments, fraud detection, reporting and PCI compliance. Supports Visa, Master Card, and AMEX.



1.5 References:

- PCI DSS v4.0
- ISO 8583:2013
- Internal UI Guide v2.1
- Vision & scope (2024)

2. Overall Description

2.1 Product Perspective

Part of Payment Gateway Suite. Replaces legacy system. Interfaces with POS, payment networks and banks

2.2 Product functions

Handles authorization, capture, refunds, settlements, fraud detection, and user access control.

2.3 User Classes

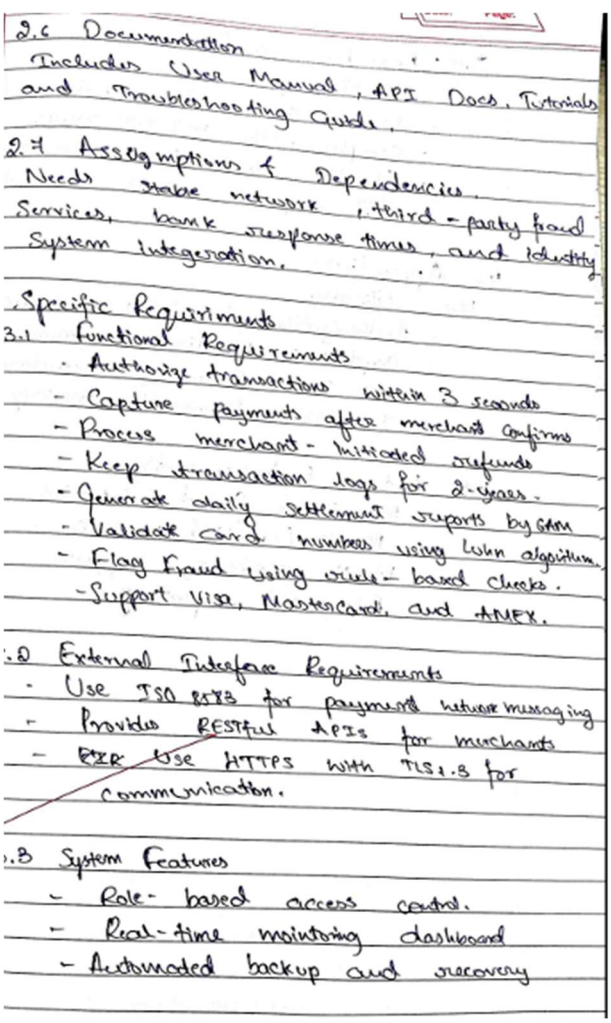
- Merchants: moderate skills
- Admins: high technical skills
- Customers: no direct access
- Support: moderate skills

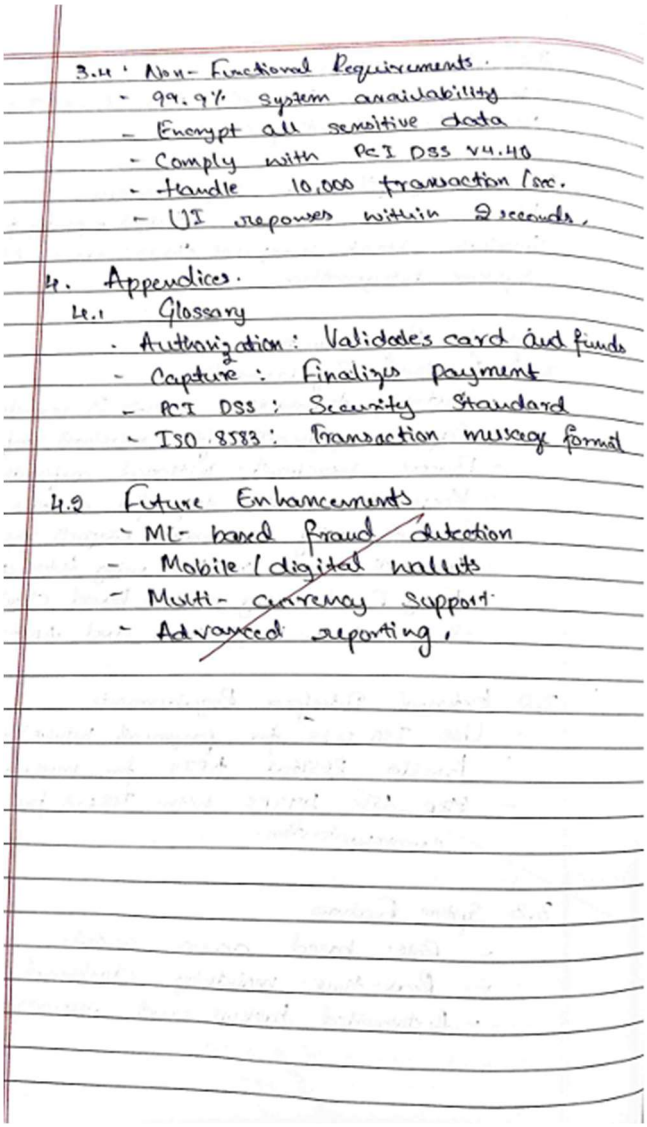
2.4 Operating Environment

Runs on POS terminals, Ubuntu servers, Windows 10/11. Uses Postgresql, RESTful TLS 1.3.

.5 Constraints

Must meet PCI DSS v4.0. Use AES-256 encryption. Java backened. Read UI. Support ISO 8583

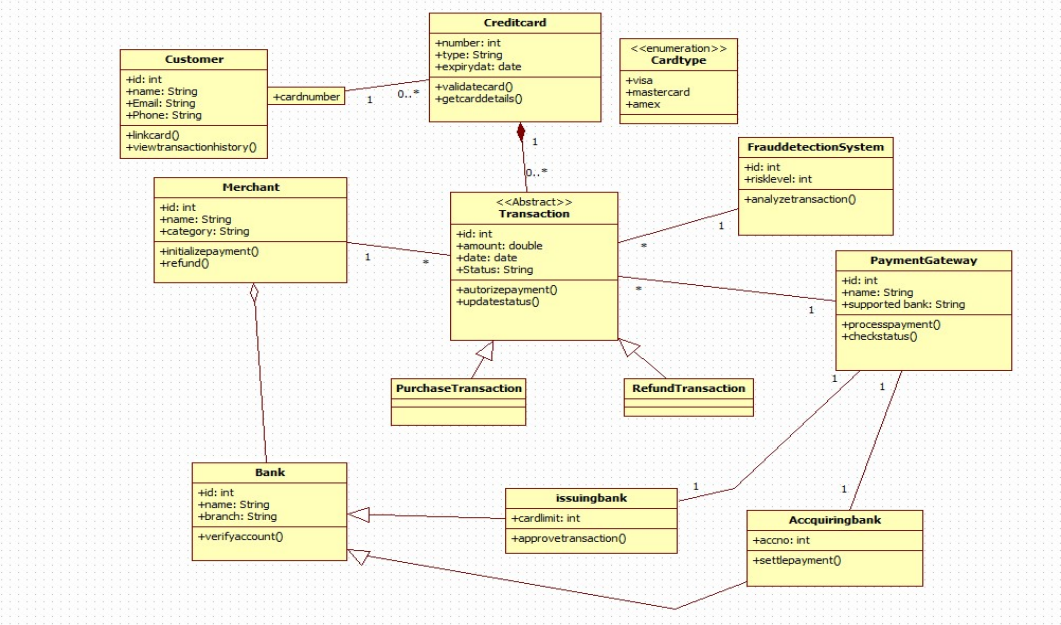




Class Diagram:

1. Customer

- Attributes: id, name, address, phone, email
- Methods: linkcard(), viewTransactionHistory()
- Relationship: Has 0 or more CreditCards (1 to 0..*)



2.CreditCard

- Attributes: number, type, expiryDate, cvv
- Methods: validatecard(), blockcard()
- Relationship: Linked to Customer and CardType enumeration
- Has 0 or more Transactions (1 to *)

3. CardType (<<enumeration>>)

- Values: visa, mastercard, amex
- Defines the type of credit card

4. Transaction (<<Abstract>>)

- Abstract base class for all transactions
- Attributes: id, amount, date, status
- Methods: authorizePayment(), updateStatus()
- Specialized into two types:
    - PurchaseTransaction - Regular purchases
    - RefundTransaction - Return/refund transactions

5. Merchant

- Attributes: id, name, category, location
- Methods: initiatePayment(), refund()
- Relationship: Processes multiple Transactions (1 to *)

6. FrauddetectionSystem

- Attributes: id, name, riskLevel
- Methods: analyzetransaction()
- Relationship: Monitors Transactions (1 to 1)

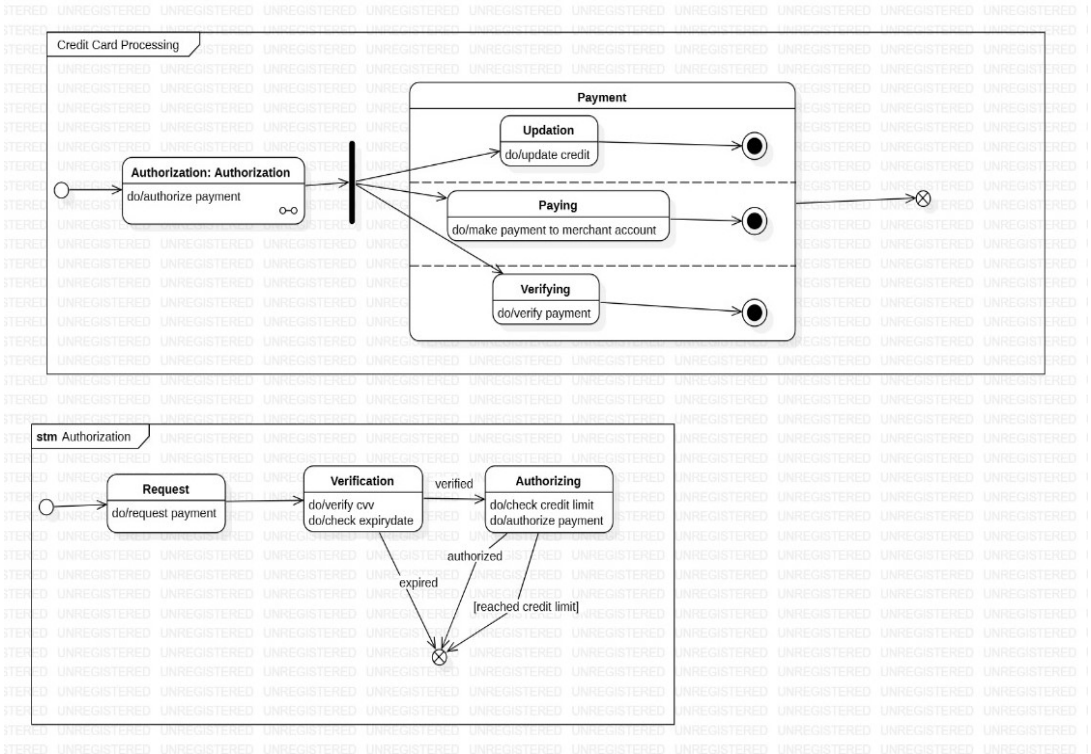
7. PaymentGateway

- Attributes: id, name, supportedBanks
- Methods: processpayment(), checkStatus()
- Relationship: Processes Transactions (1 to 1)

8. Bank

- Attributes: id, name, branch, swift
- Methods: verifyaccount()
- Two specialized types:
    - Issuingbank - Issues credit cards (cardlimit attribute, approveTransaction() method)
    - Acquiringbank - Processes merchant payments (acqno attribute, settlepayment() method)

State Diagrams:



State Machine 1: Credit Card Processing (Main Workflow)

Authorization: Entry point for the payment processing system that handles payment authorization.

- do/authorize payment - Validates and authorizes the payment transaction
- This is a composite state that contains its own internal state machine (detailed in stm Authorization below)
- Represented with rounded corners indicating it's a subprocess
- After authorization completes → Fork (synchronization bar) leading to three parallel regions

Payment (Composite State with Parallel Regions): After authorization, the system enters three concurrent sub-states that execute simultaneously:

Updation: Updates credit status in the system.

- do/update credit - Modifies credit account balance and transaction records
- Transitions to flow final (filled circle ●) when update complete

Paying: Processes fund transfer to merchant.

- do/make payment to merchant account - Transfers authorized amount to merchant's account
- Transitions to flow final (filled circle ●) when payment complete

Verifying: Confirms transaction integrity.

- do/verify payment - Validates successful completion of transaction
- Transitions to flow final (filled circle ●) when verification complete

Final State: Represented by circled X (⊗)

- Reached after all three parallel activities (Updation, Paying, Verifying) complete successfully
- Indicates successful end of credit card processing

State Machine 2: Authorization (stm Authorization)

This state machine details the internal workings of the Authorization composite state.

Request (Initial State): Initial payment request submission.

- do/request payment - Submits payment request with card details to payment gateway
- Automatic progression → Verification

Verification: Validates payment information and checks account status.

- do/verify cvv - Validates Card Verification Value
- do/check expirydate - Confirms card hasn't expired
- Event: verified → Authorizing (all validations passed)
- Event: expired → Final State (⊗) (card expired, authorization denied)

Authorizing: Final authorization decision state.

- do/check credit limit - Verifies sufficient credit available
- do/authorize payment - Grants or denies authorization
- Event: authorized → Final State (⊗) (authorization successful)
- Event: [reached credit limit] → Final State (⊗) (insufficient credit, authorization denied)

Use Case Diagram

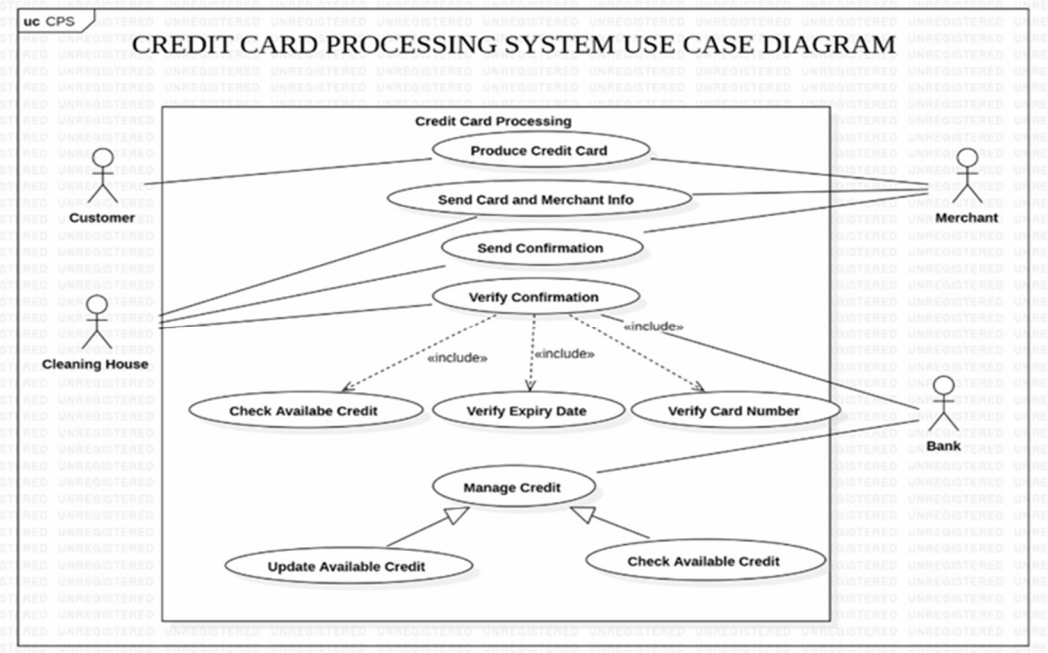
Actors (External Entities)

The diagram shows four main actors positioned around the system boundary:

1. **Customer** - Initiates credit card transactions
2. **Merchant** - Receives card and merchant information for processing payments
3. **Cleaning House** - Acts as an intermediary in the payment clearing process
4. **Bank** - Validates and authorizes credit card transactions

Use Cases (System Functions)

The diagram depicts several key use cases within the Credit Card Processing system:



Primary Transaction Flow:

- Produce Credit Card - Customer initiates by providing their credit card
- Send Card and Merchant Info - Information is transmitted between Customer and Merchant
- Send Confirmation - System sends confirmation back to the Customer
- Verify Confirmation - The Cleaning House verifies the transaction confirmation

Verification Process (with «include» relationships): The "Verify Confirmation" use case includes three mandatory sub-processes:

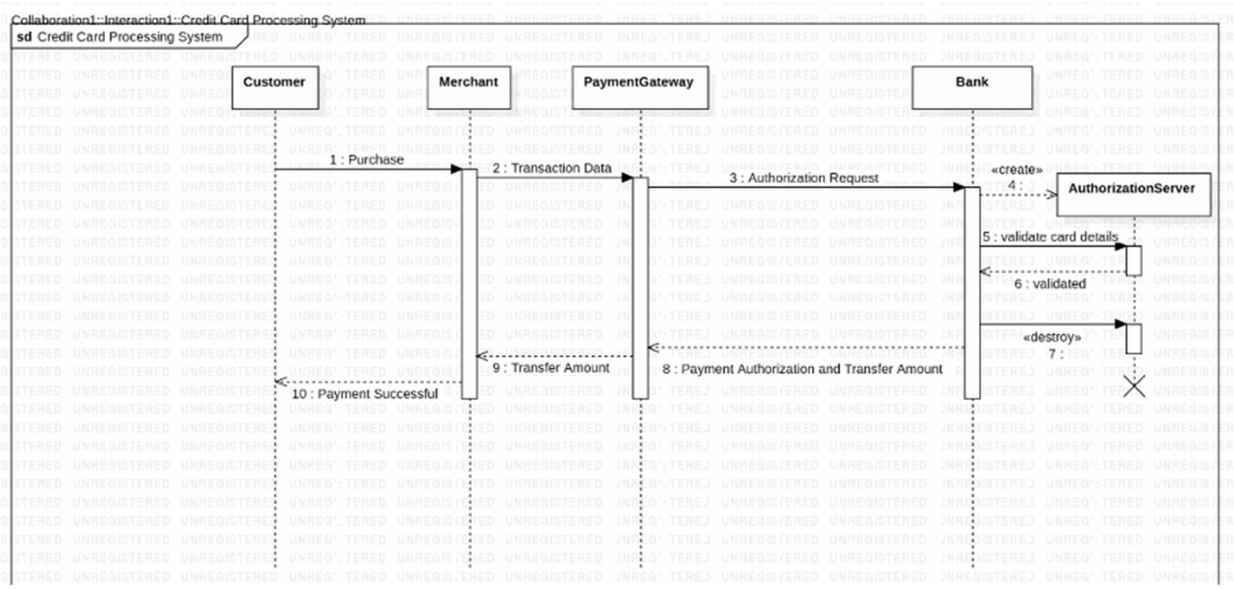
- Check Available Credit - Validates sufficient credit is available
- Verify Expiry Date - Confirms the card hasn't expired
- Verify Card Number - Validates the card number is legitimate

These verification steps connect to the **Bank** actor, which performs the actual validation.

Credit Management:

- Manage Credit - Central function for credit operations
- Update Available Credit - Modifies credit balance after transactions
- Check Available Credit - Queries current available credit

Sequence Diagram:



1. Customer - The person making a purchase
2. Merchant - The seller/vendor processing the sale
3. PaymentGateway - The intermediary system handling payment processing
4. Bank - The financial institution managing the transaction
5. AuthorizationServer - A component that validates card details

Message Flow (Sequence of Interactions)

The transaction proceeds through the following steps:

1. Purchase - Customer initiates a purchase with the Merchant

2. Transaction Data - Merchant sends transaction information to the PaymentGateway

3. Authorization Request - PaymentGateway forwards the authorization request to the Bank

4. «create» - Bank creates an AuthorizationServer instance (shown with dotted arrow)

5. Validate card details - Bank sends card information to AuthorizationServer for validation

6. Validated - AuthorizationServer returns validation confirmation to the Bank

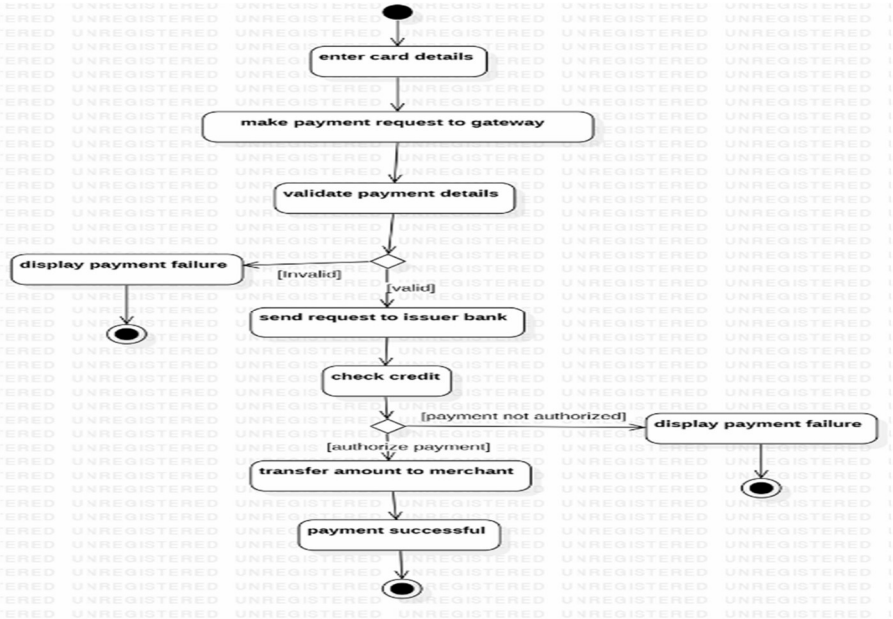
7. «destroy» - AuthorizationServer instance is terminated (shown with X symbol at the end of its lifeline)

8. Payment Authorization and Transfer Amount - Bank sends authorization approval and transfers the amount back to PaymentGateway

9. Transfer Amount - PaymentGateway notifies the Merchant of the successful fund transfer

10. Payment Successful - Merchant confirms successful payment to the Customer

Activity Diagram



Activities (Rounded Rectangles):

1. enter card details - Initial step where card information is provided
2. make payment request to gateway - System sends payment data to the payment gateway
3. validate payment details - Gateway validates the entered information

Decision Node (Diamond) - First validation checkpoint:

- [invalid] branch → leads to display payment failure → ends at termination node
- [valid] branch → continues to next step

4. send request to issuer bank - Validated request is forwarded to the card-issuing bank
5. check credit - Bank verifies available credit/funds

Second Decision Node (Diamond) - Authorization checkpoint:

- [payment not authorized] branch → leads to display payment failure → ends at termination node
- [authorize payment] branch → continues to success path

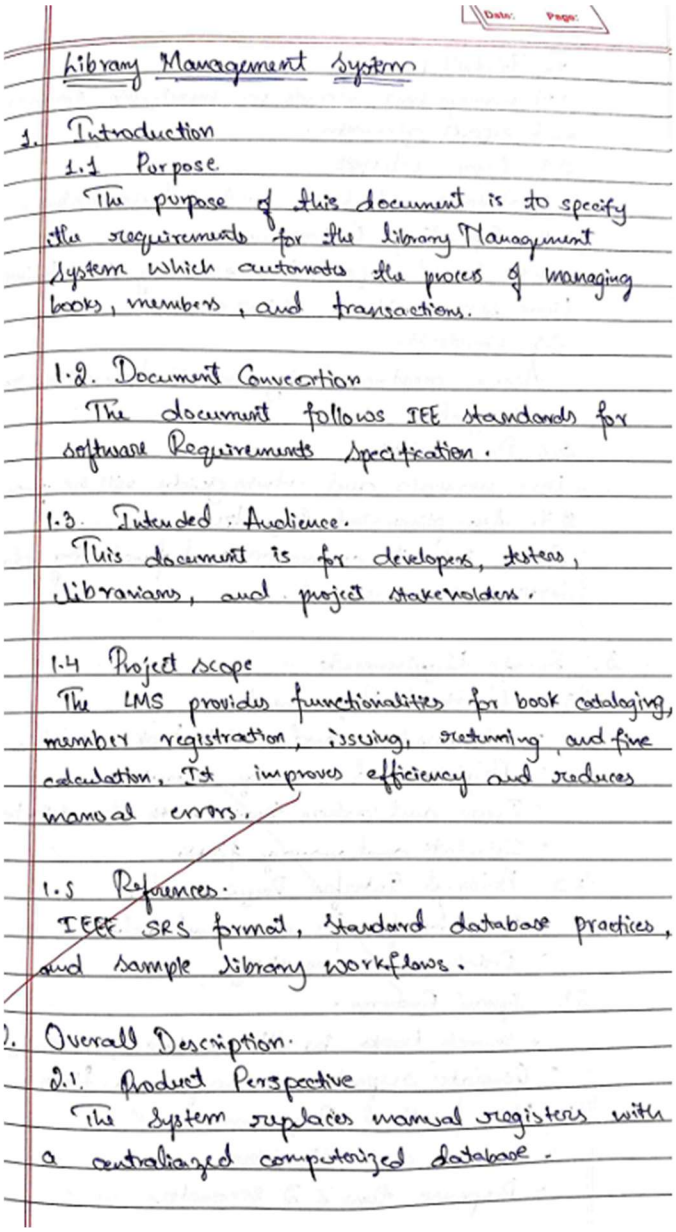
6. transfer amount to merchant - Funds are transferred upon successful authorization
7. payment successful - Final confirmation of completed transaction

Library Management System

Problem Statement

In many libraries, activities such as book issuing, returning, student record maintenance, and book availability checking are still managed manually or using semi-automated systems. This leads to problems like data redundancy, human errors, time delays, and difficulty in tracking books and users efficiently.

SRS:



Library Management System

1. Introduction
1.1 Purpose
The purpose of this document is to specify the requirements for the library Management System which automates the process of managing books, members, and transactions.

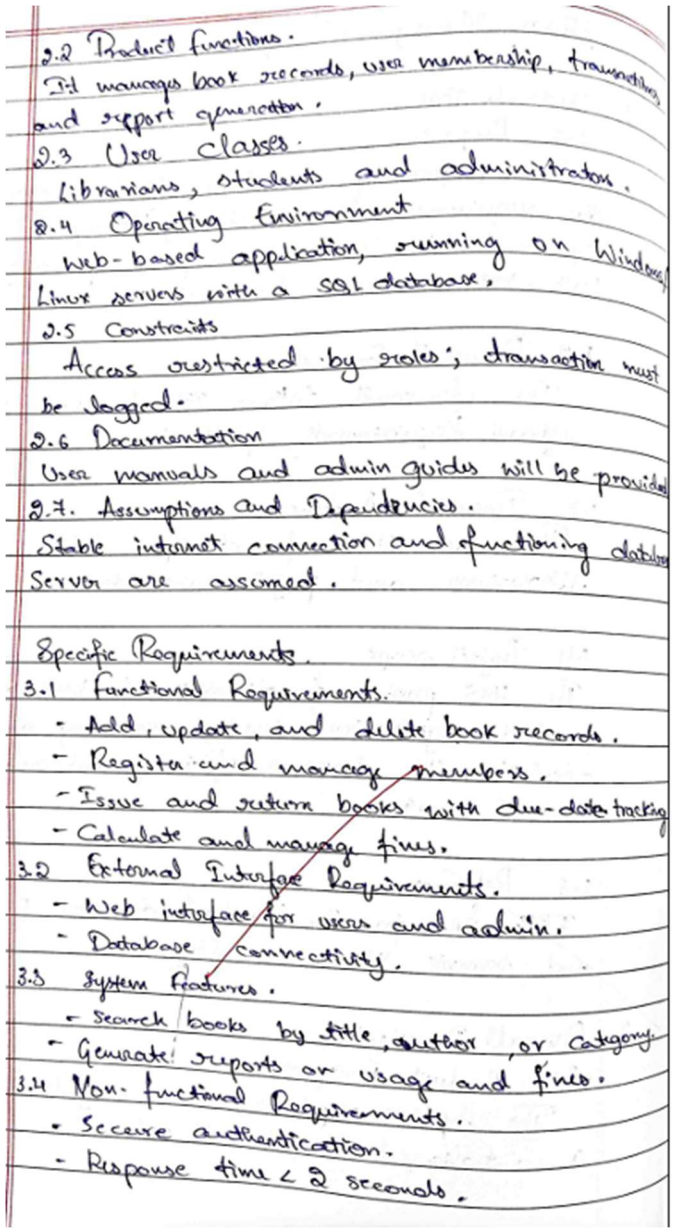
1.2. Document Convention
The document follows IEEE standards for software Requirements Specification.

1.3. Intended Audience.
This document is for developers, testers, librarians, and project stakeholders.

1.4 Project Scope
The LMS provides functionalities for book cataloging, member registration, issuing, returning, and fine calculation. It improves efficiency and reduces manual errors.

1.5 References.
IEEE SRS format, standard database practices, and sample library workflows.

2. Overall Description.
2.1. Product Perspective
This system replaces manual registers with a centralized computerized database.



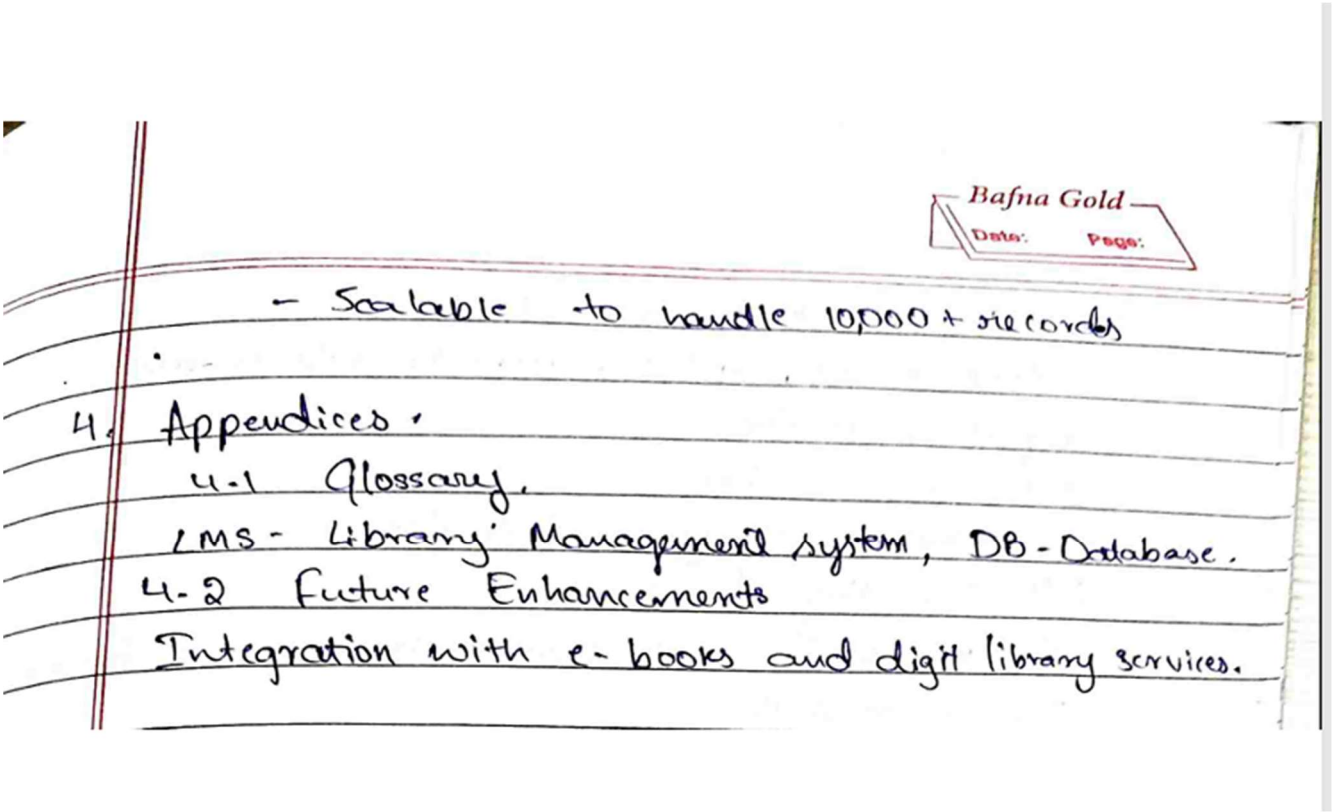
2.2 Product functions.
It manages book records, user membership, transactions, and report generation.
2.3 User classes.
Librarians, students and administrators.
2.4 Operating Environment
Web-based application, running on Windows/Linux servers with a SQL database.
2.5 Constraints
Access restricted by roles; transaction must be logged.
2.6 Documentation
User manuals and admin guides will be provided.
2.7. Assumptions and Dependencies.
Stable internet connection and functioning database server are assumed.

Specific Requirements.
3.1 Functional Requirements.
- Add, update, and delete book records.
- Register and manage members.
- Issue and return books with due-date tracking.
- Calculate and manage fines.

3.2 External Interface Requirements.
- Web interface for user and admin.
- Database connectivity.

3.3 System Features.
- Search books by title, author, or category.
- Generate reports on usage and fines.

3.4 Non-functional Requirements.
- Secure authentication.
- Response time < 2 seconds.



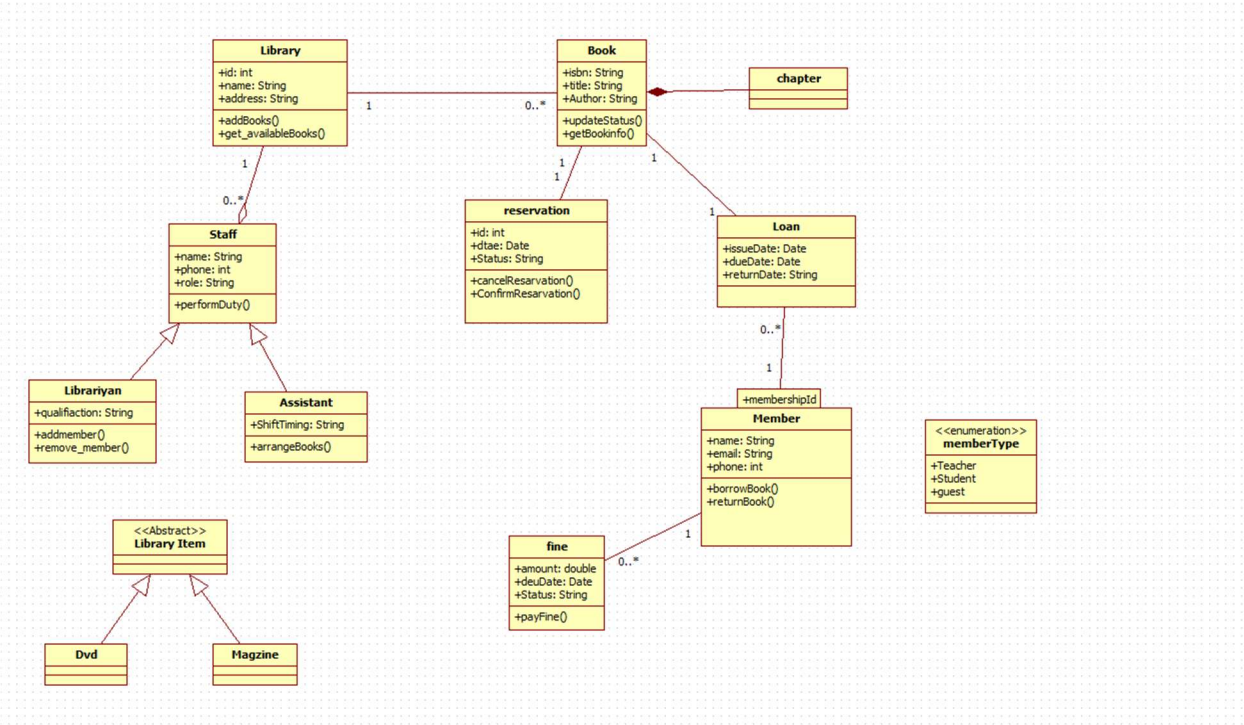
Class Diagram

Library

- Attributes: name: String, address: String
- Methods: addBooks(), get_availableBooks()
- Relationship: Has a one-to-many relationship with Book (1 to 0..*)

Book

- Attributes: isbn: String, title: String, author: String
- Methods: updateStatus(), getBookInfo()



Library

- Attributes: name: String, address: String
- Methods: addBooks(), get_availableBooks()
- Relationship: Has a one-to-many relationship with Book (1 to 0..*)

Book

- Attributes: isbn: String, title: String, author: String
- Methods: updateStatus(), getBookInfo()
- Relationships:
    - Composed of chapters (aggregation)
    - Associated with reservations (1 to 1)
    - Associated with Loans (1 to 1)

chapter

- Simple class associated with Book

Staff (Superclass)

- Attributes: name: String, phone: int, address: String
- Methods: performDuty()
- Relationship: Has two specialized subclasses (generalization/inheritance)

Librarian (inherits from Staff)

- Attributes: qualification: String
- Methods: addmember(), remove_member()

Assistant (inherits from Staff)

- Attributes: ShiftTiming: String
- Methods: arrangeBooks()

reservation

- Attributes: id: int, date: Date, Status: String
- Methods: cancelReservation(), ConfirmReservation()

Loan

- Attributes: issueDate: Date, dueDate: Date, returnDate: String
- Relationship: Connected to Member and has fine associations

Member

- Attributes: name: String, email: String, phone: int
- Methods: borrowBook(), returnBook()
- Relationships:
    - Has a membership type (enumeration)
    - Associated with Loans (1 to 0..*)
    - Connected to thesisorship(s)

<<enumeration>> memberType

- Values: Teacher, Student, guest

<<Abstract>> Library Item

- Abstract superclass for different item types
- Subclasses: **Dvd** and **Magzine** (shown with inheritance arrows)

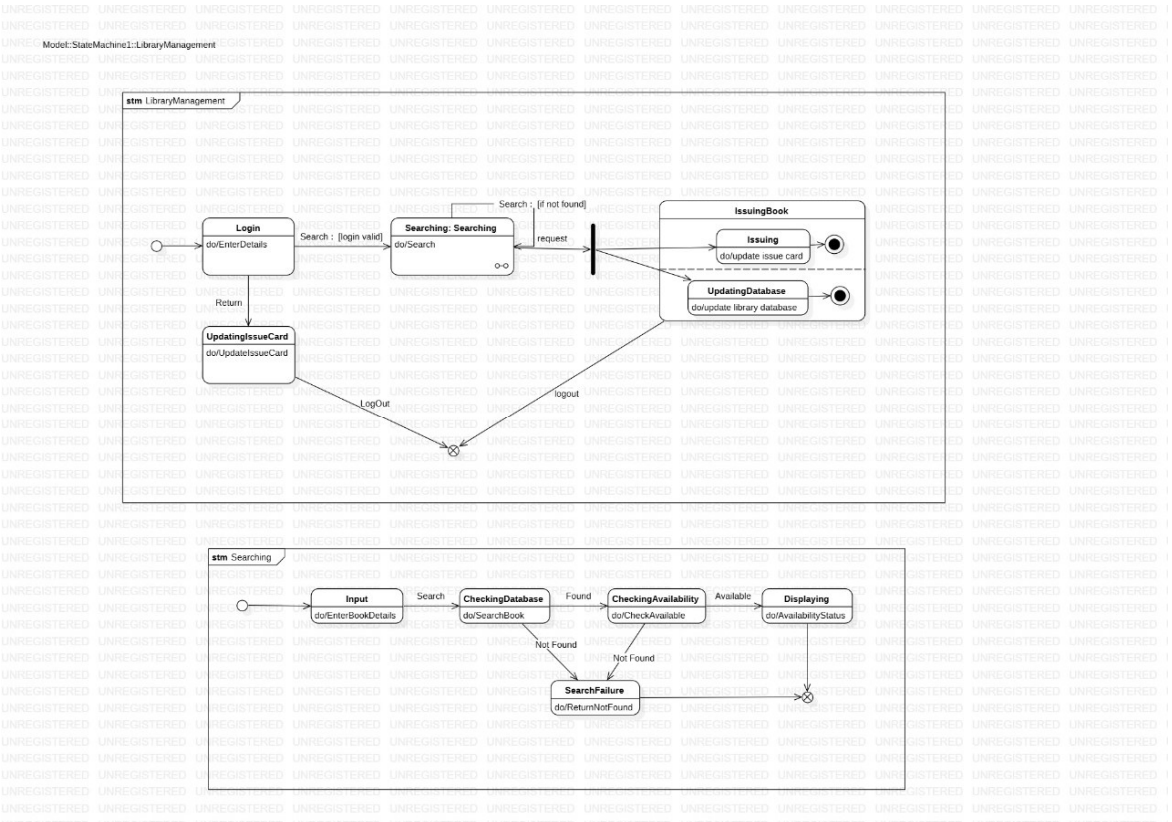
fine

- Attributes: amount: double, dueDate: Date, Status: String
- Methods: payFine()
- Relationship: Associated with Loan (1 to 0..*)

thesisorship(s)

- Connected to Member class

State Diagram



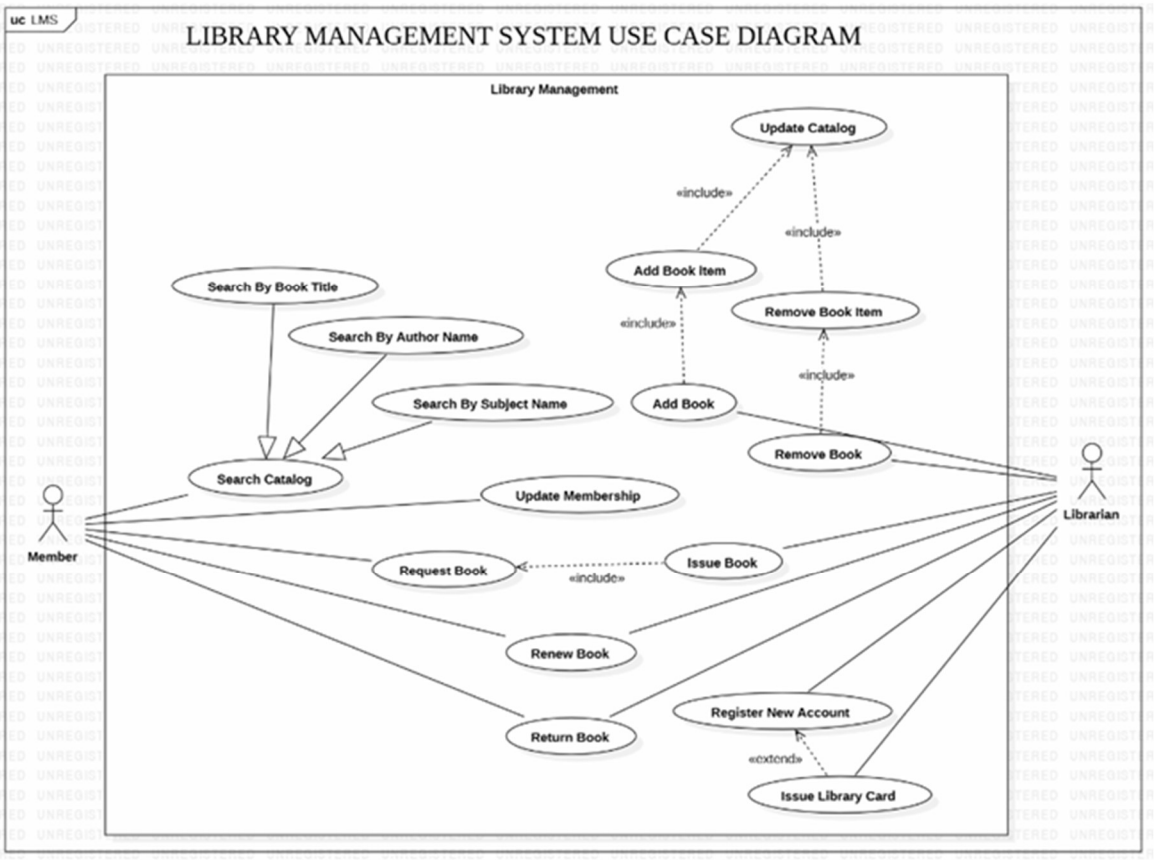
The diagram represents the **State Machine model of a Library Management System** focusing on the process of searching and issuing books. It begins with the **Login state**, where the user enters login details. If the login is valid, the user is allowed to proceed to the **Searching state**. In this state, the user searches for a required book by entering the book details.

The system then checks the database to verify whether the book exists or not. If the book is **not found**, the system moves to the **Search Failure state** and displays a message indicating that the book is unavailable. If the book is found, the system checks the **availability status** of the book.

If the book is available, the user proceeds to the **Issuing Book state**, where the issuing process is carried out. After the issue process, two important operations occur: **updating the issue card** and **updating the library database**. These updates ensure that the issued book and user details are properly stored.

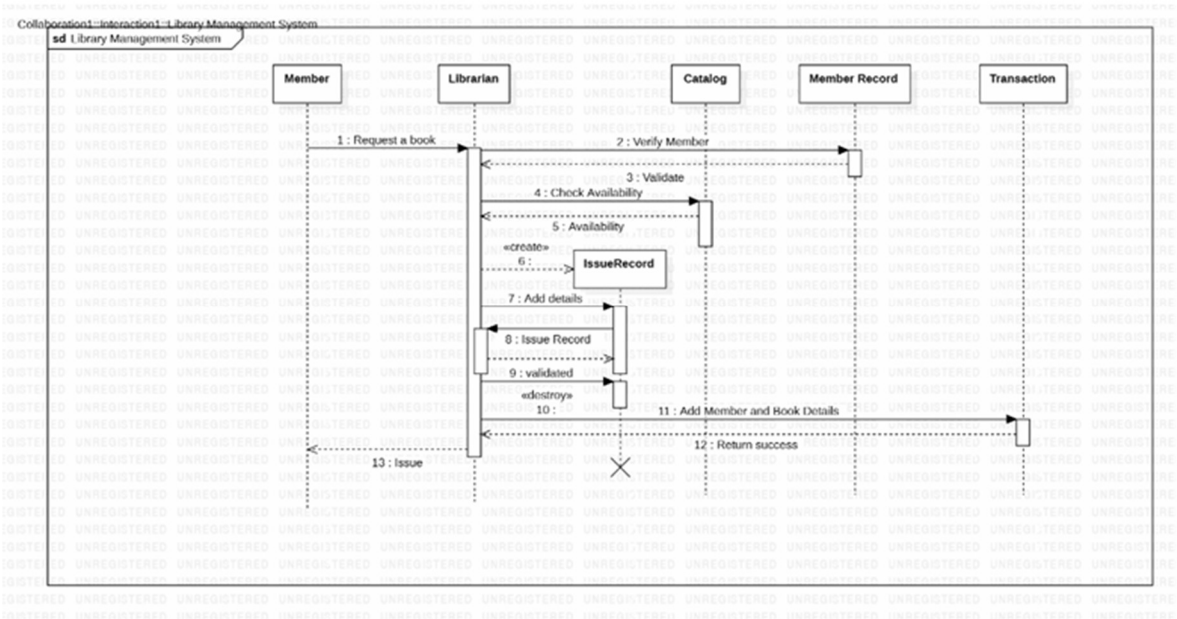
Finally, once the operations are completed, the user logs out of the system. The diagram clearly shows how different states and transitions are connected, helping in understanding the complete workflow of the library management process.

Use Case Diagram:



- The Member can search for books in the catalog using different methods such as Search by Book Title, Search by Author Name, and Search by Subject Name.
- The member can also Request a Book, Renew a Book, and Return a Book.
- The Librarian is responsible for major system operations like Issue Book, Add Book, Remove Book, and Update Membership.
- The use case Issue Book is connected with the book request process, where a request leads to the issue operation.
- The librarian can manage books through Add Book Item and Remove Book Item, which include updating the catalog.
- The Update Catalog use case ensures that any book added or removed is reflected in the library database.
- The librarian can also Register New Accounts, which extends to issuing a Library Card.

Sequence Diagram:

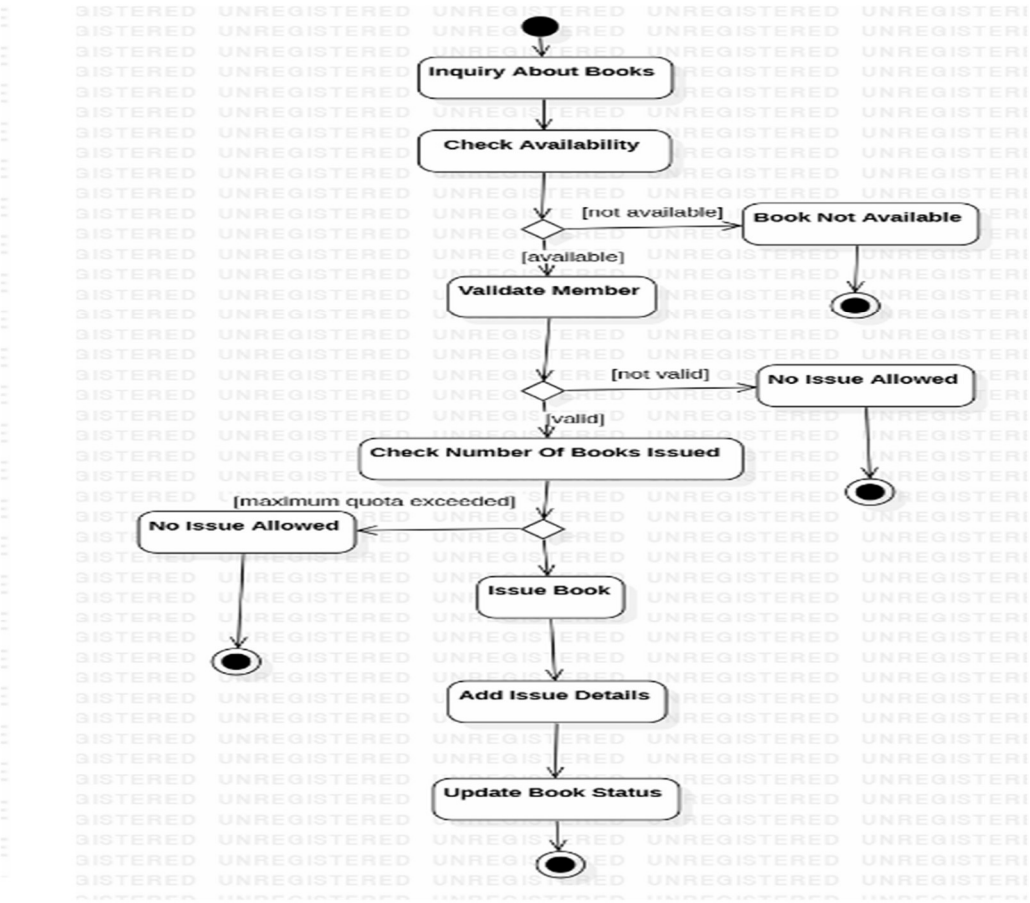


The diagram shows the process of issuing a book in a library system. First, the Member sends a request to the Librarian to issue a book. The librarian verifies the member by checking the Member Record. After successful verification, the librarian checks the book availability in the Catalog.

If the book is available, the system creates an Issue Record and adds the necessary details such as member information and book details. The librarian then updates the Transaction system with the issue information. Once all records are updated successfully, the system sends a confirmation message back stating issue success.

If at any stage the book is unavailable or the member validation fails, the issue process is terminated. Finally, the process ends after all necessary records are updated and confirmation is given.

Activity Diagram



This diagram represents the activity flow for issuing a book in a Library Management System. The process starts with an inquiry about books, after which the system checks the availability of the book. If the book is not available, the process ends with *Book Not Available*.

If the book is available, the system then validates the member. If the member is not valid, the system displays *No Issue Allowed* and ends. If the member is valid, the system checks the number of books already issued to the member. If the maximum quota is exceeded, issuing is denied.

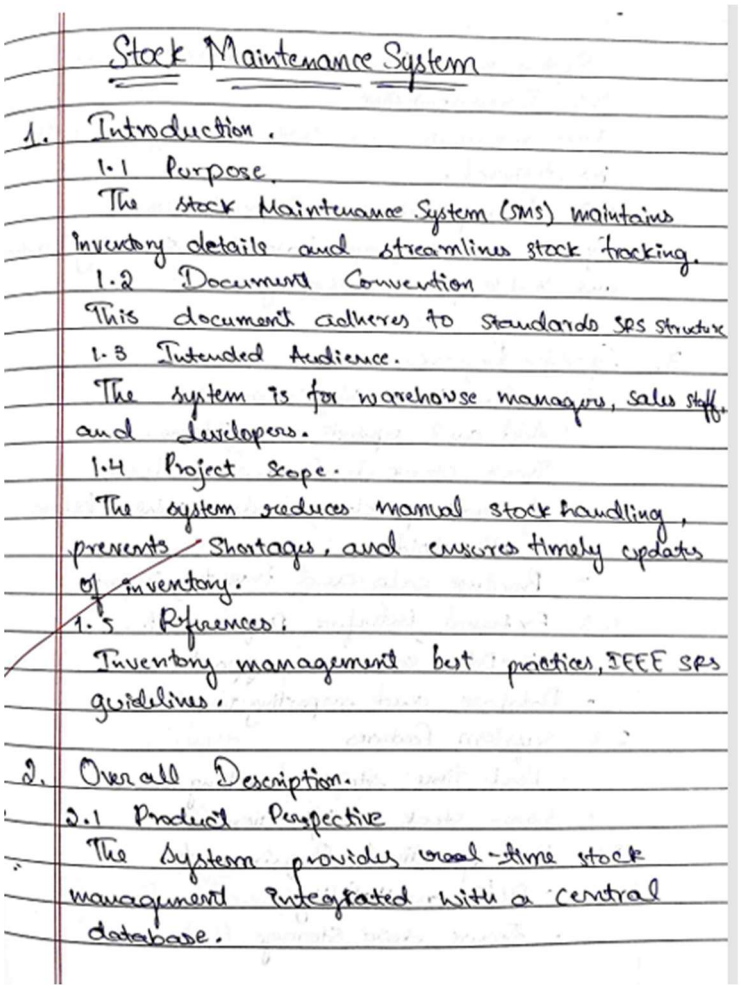
If all conditions are satisfied, the system proceeds to Issue the Book, then adds issue details, and finally updates the book status in the database. After completing these steps, the process ends successfully.

Stock Maintenance System

Problem Statement:

In many organizations, stock or inventory is often maintained manually or using outdated systems. This leads to problems such as stock mismatches, overstocking, understocking, difficulty in tracking item movement, and errors in report generation due to human involvement.

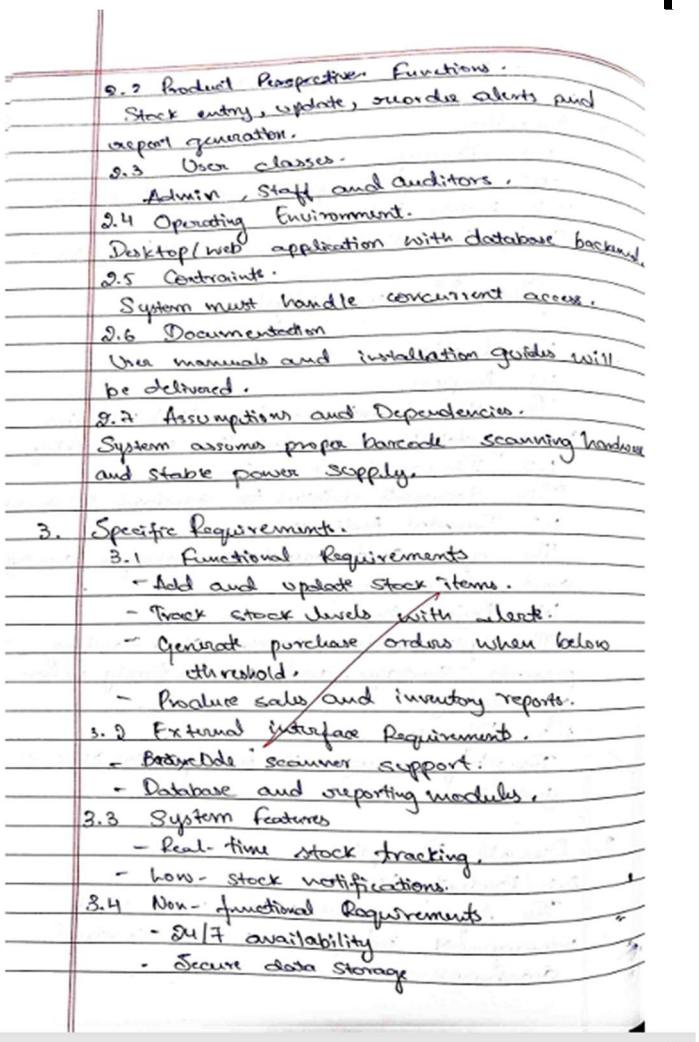
SRS:



Stock Maintenance System

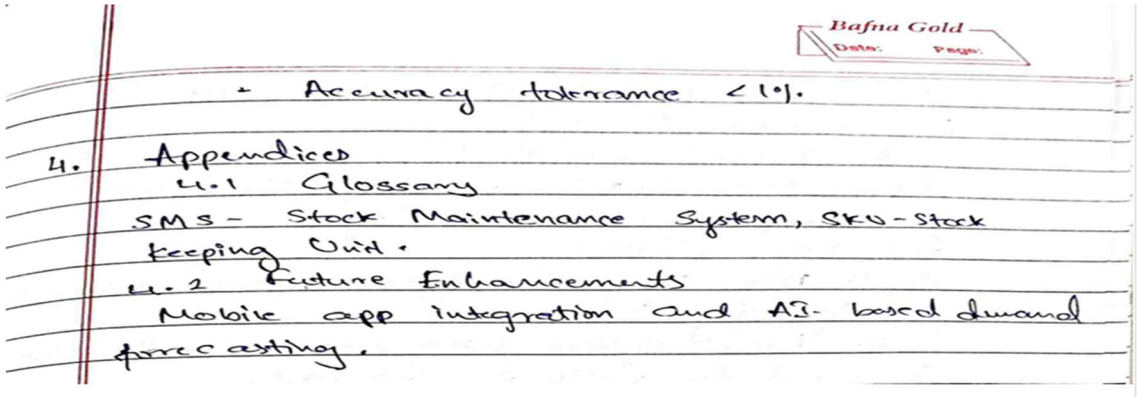
1. Introduction.
1.1 Purpose.
The stock Maintenance System (SMS) maintains inventory details and streamlines stock tracking.
1.2 Document Convention
This document adheres to standards SRS structure
1.3 Intended Audience.
The system is for warehouse managers, sales staff, and developers.
1.4 Project Scope.
The system reduces manual stock handling, prevents shortages, and ensures timely updates of inventory.
1.5 References:
Inventory management best practices, IEEE SRS guidelines.

2. Overall Description.
2.1 Product Perspective
The System provides real-time stock management integrated with a central database.



2.2 Product Perspective. Functions.
Stock entry, update, reorder alerts and report generation.
2.3 User classes.
Admin, Staff and auditors.
2.4 Operating Environment.
Desktop/web application with database backend.
2.5 Constraints.
System must handle concurrent access.
2.6 Documentation
User manuals and installation guides will be delivered.
2.7 Assumptions and Dependencies.
System assumes proper barcode scanning hardware and stable power supply.

3. Specific Requirements.
3.1 Functional Requirements
- Add and update stock items.
- Track stock levels with alerts.
- Generate purchase orders when below threshold.
- Produce sales and inventory reports.
3.2 External Interface Requirement.
- Barcode scanner support.
- Database and reporting modules.
3.3 System Features
- Real-time stock tracking.
- Low-stock notifications.
3.4 Non-functional Requirements
- 24/7 availability
- Secure data storage



Bafna Gold
Date: Page:

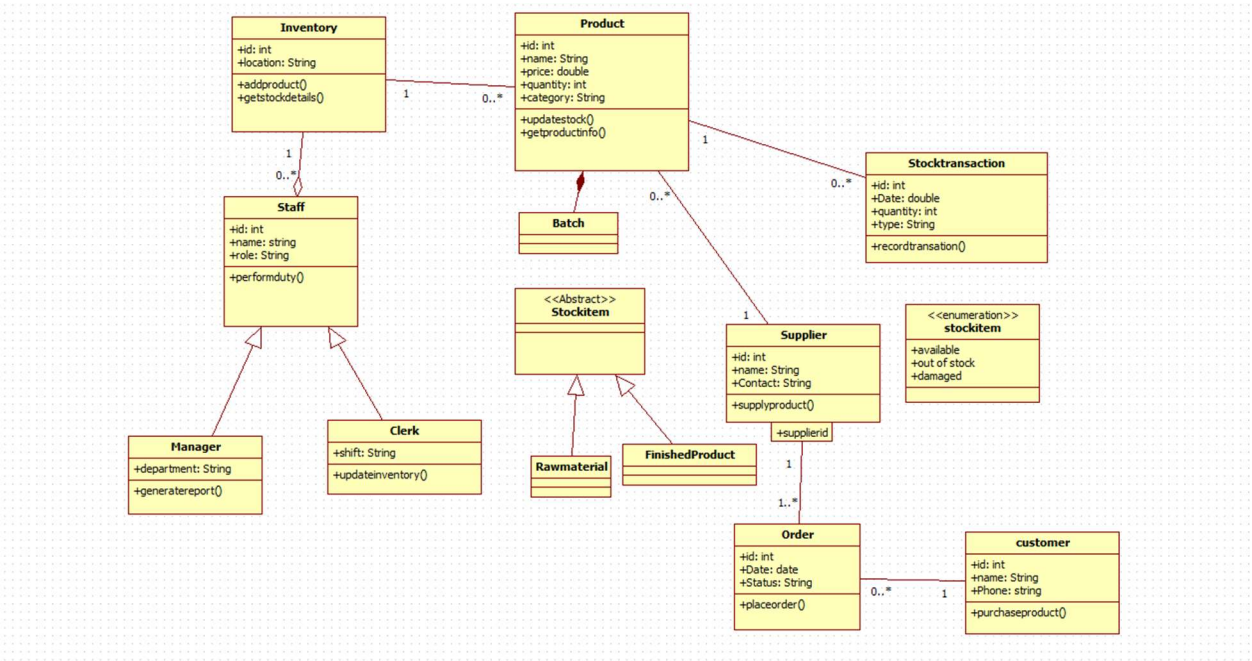
- Accuracy tolerance < 1%.

4. Appendices
4.1 Glossary
SMS - Stock Maintenance System, SKU - Stock Keeping Unit.
4.2 Future Enhancements
Mobile app integration and AI-based demand forecasting.

Class Diagram

Inventory

- Attributes: id, location
- Functions: addProduct(), getStockDetails()
- Purpose: Maintains overall stock details for a specific location.
- Relationship: One inventory can hold **multiple products**.



---

2. Product

- Attributes: id, name, price, quantity, category
- Functions: updateStock(), getProductInfo()
- Purpose: Represents an item stored in inventory.
- Relationship:
    - Connected to StockTransaction (one product can have many transactions).
    - Linked with Batch.
    - Supplied by one or more Suppliers.

3. Batch

- Represents product batches for tracking grouped stock.
- Connected with the Product class.

4. StockTransaction

- Attributes: id, date, quantity, type
- Function: recordTransaction()
- Purpose: Stores records of stock movements like stock-in and stock-out.

5. StockItem (Abstract Class)

- Parent class for stock types.
- Two child classes:
  - ☑ **RawMaterial**
  - ☑ **FinishedProduct**
- Helps in classifying types of stock.

6. StockItem Enumeration

Defines stock status:

- Available
- Out of Stock
- Damaged

Used to check the condition of stock.

7. Supplier

- Attributes: id, name, contact
- Function: supplyProduct()
- Purpose: Manages supplier details.

- Connected to products and orders (one supplier can supply multiple orders).

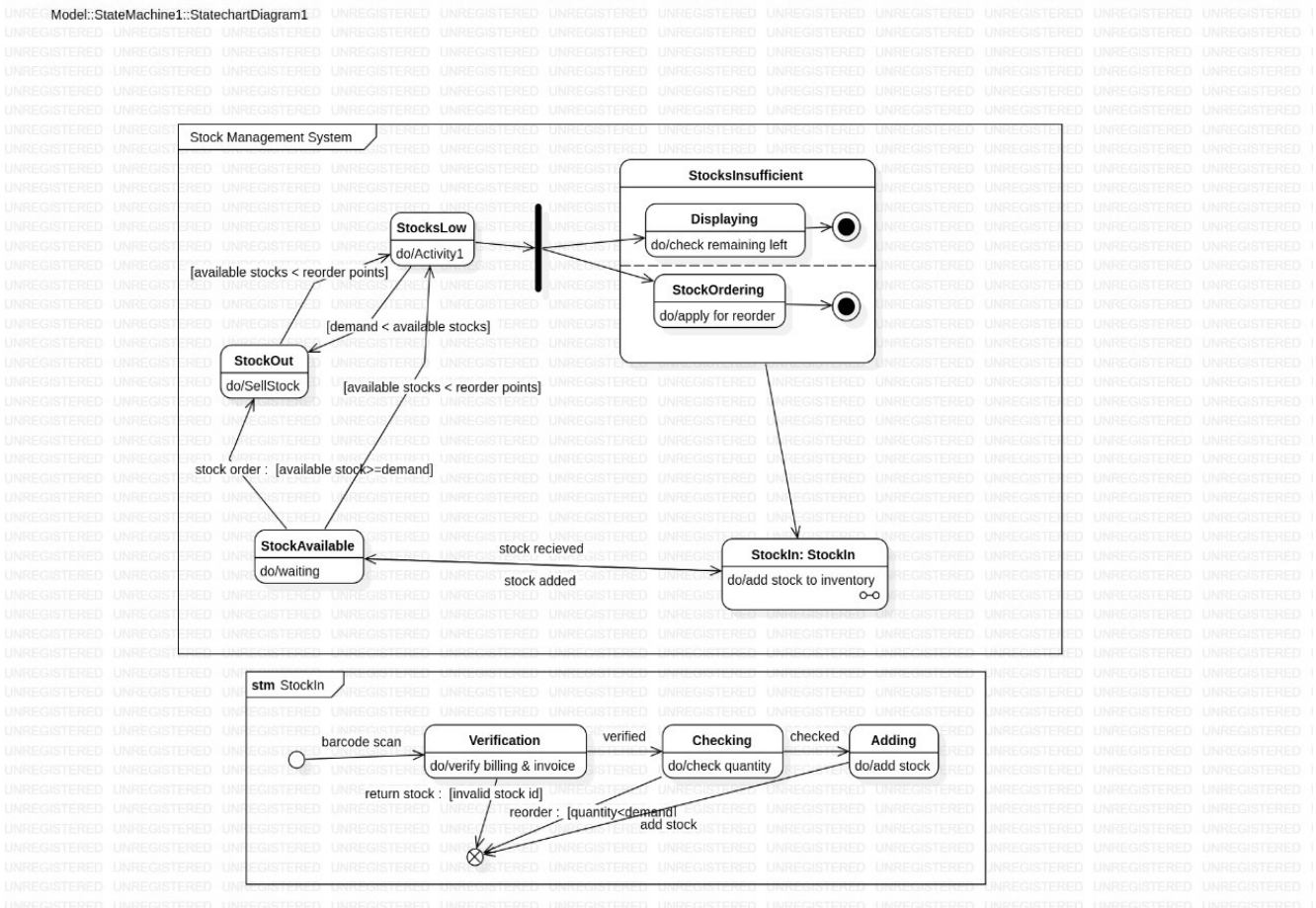
8. Order

- Attributes: id, date, status
- Function: placeOrder()
- Purpose: Manages orders placed by customers.

9. Customer

- Attributes: id, name, phone
- Function: purchaseProduct()
- Relationship:
    - One customer can place multiple orders.

State Diagram



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.

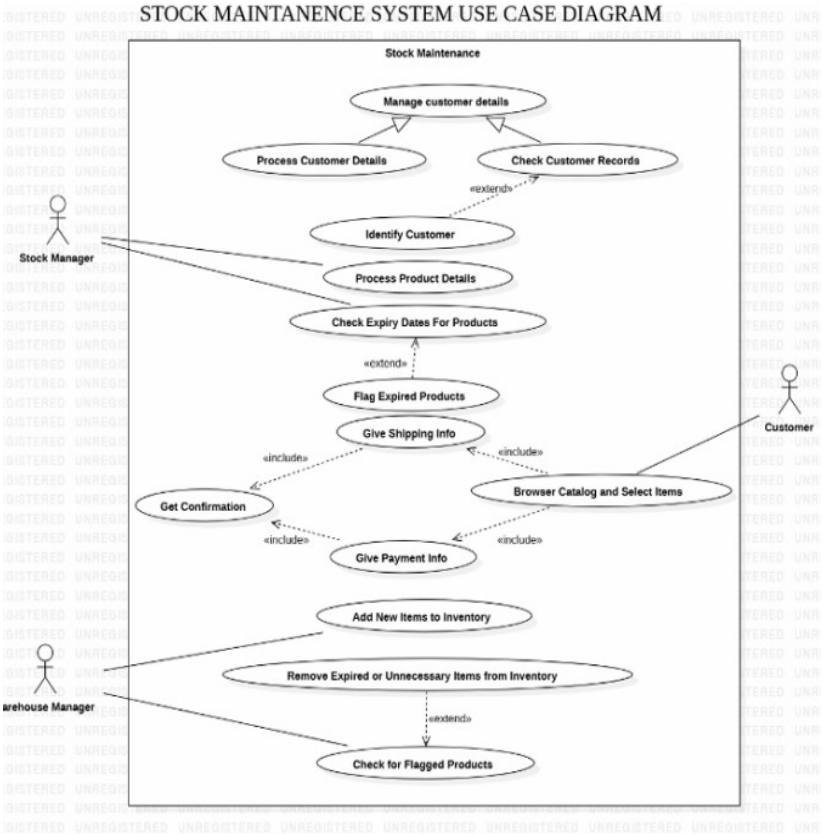
The diagram illustrates a comprehensive inventory management workflow that monitors stock levels, triggers reordering when necessary, and carefully processes incoming inventory through verification steps.

Use Case model:

Stock Manager handles customer management (processing details and checking records), identifies customers, processes product details, monitors expiry dates, flags expired products, and provides shipping information with confirmation.

Customer browses the catalog, selects items, provides payment information, and receives confirmation for purchases.

Warehouse Manager adds new items to inventory, removes expired or unnecessary items, and checks for flagged products.



The diagram uses **«includes»** relationships (mandatory sub-functions like getting confirmation) and **«extends»** relationships (optional functions like checking customer records or flagging expired products). The system provides comprehensive functionality for managing customer data, product inventory, order processing, and warehouse operations in an integrated stock maintenance environment.

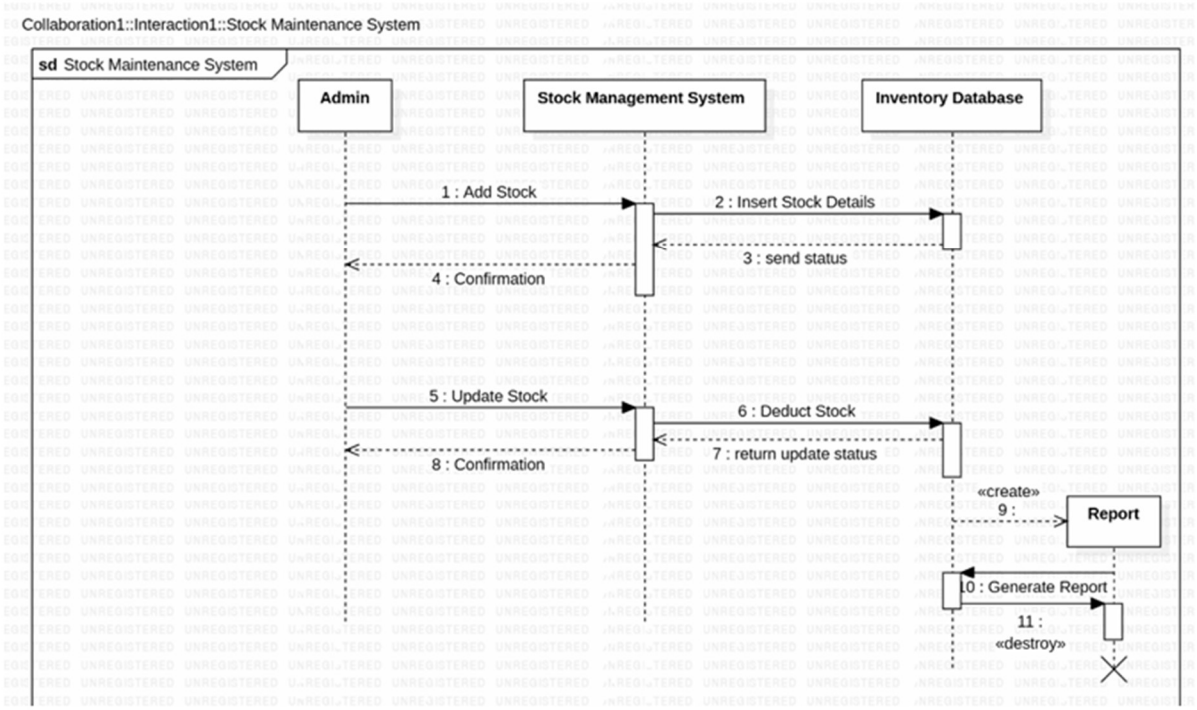
Sequence model:

Participants: Admin (user), Stock Management System (controller), Inventory Database (data store), and Report (output object).

Add Stock Flow: Admin sends an "Add Stock" command (1) to the Stock Management System, which inserts stock details (2) into the Inventory Database. The database sends back a status confirmation (3), and the Stock Management System returns a confirmation (4) to the Admin.

Update Stock Flow: Admin initiates "Update Stock" (5), the Stock Management System deducts stock (6) from the Inventory Database, receives an update status (7), and confirms the operation (8) back to the Admin.

Report Generation: After updates, the Inventory Database creates a Report object (9), generates the report content (10), and then destroys the report (11) upon completion.



The diagram uses solid lines for synchronous messages and dashed lines for return messages, illustrating the sequential interaction and data flow between system components during stock management operations.

Activity Model

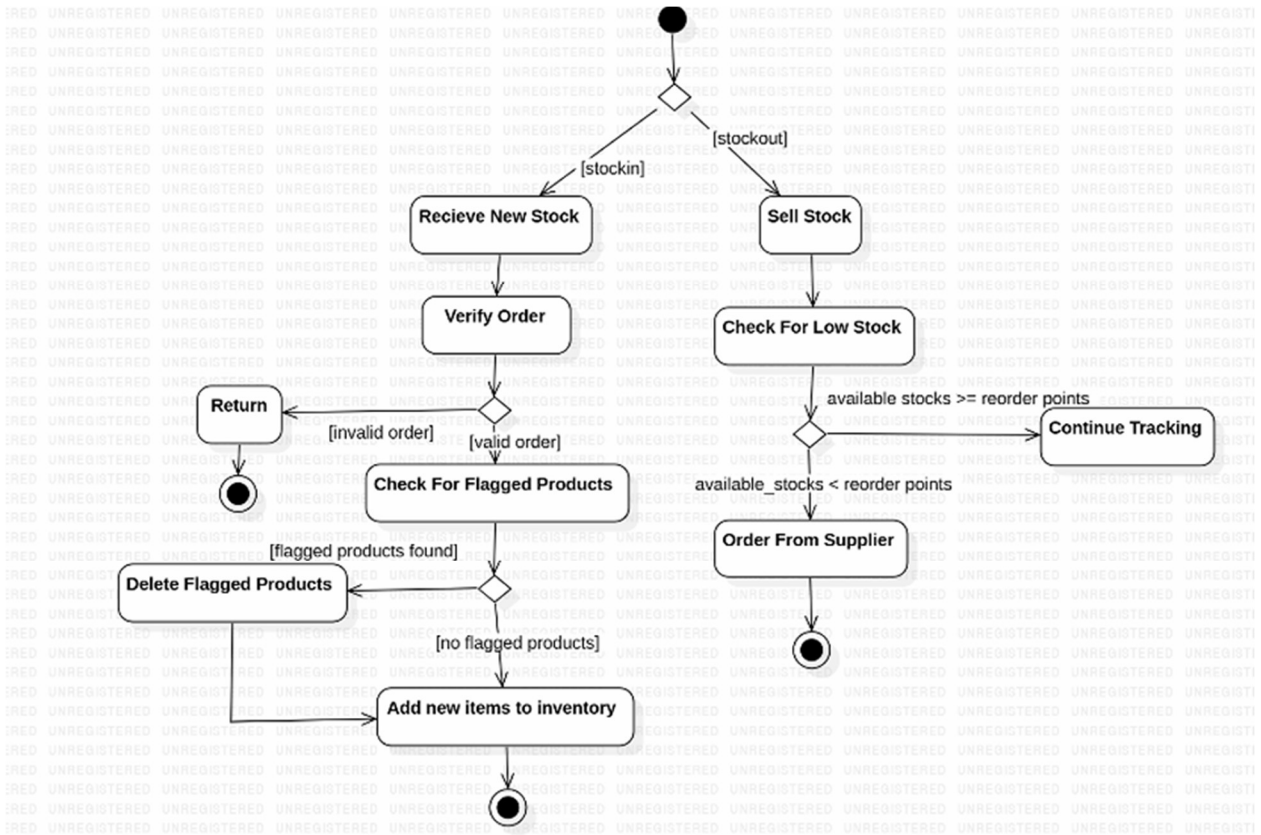
Initial Decision: The process starts with a decision node that branches into two paths: [stockin] for receiving new inventory and [stockout] for selling stock.

Stock-In Path: When receiving stock, the system executes "Receive New Stock" followed by "Verify Order." A decision checks if the order is valid—if invalid, the process returns and terminates; if valid, it proceeds to "Check For Flagged Products." Another decision determines if flagged products exist—if found, "Delete Flagged

Products" executes; if not, the flow continues. Both paths converge at "Add new items to inventory" before reaching the final state.

Stock-Out Path: When selling stock, "Sell Stock" is executed, then "Check For Low Stock" examines inventory levels. A decision evaluates whether available stocks are below reorder points—if yes, "Order From Supplier" is triggered and the process ends; if stocks are sufficient (available stocks >= reorder points), the system proceeds to "Continue Tracking" and ends.

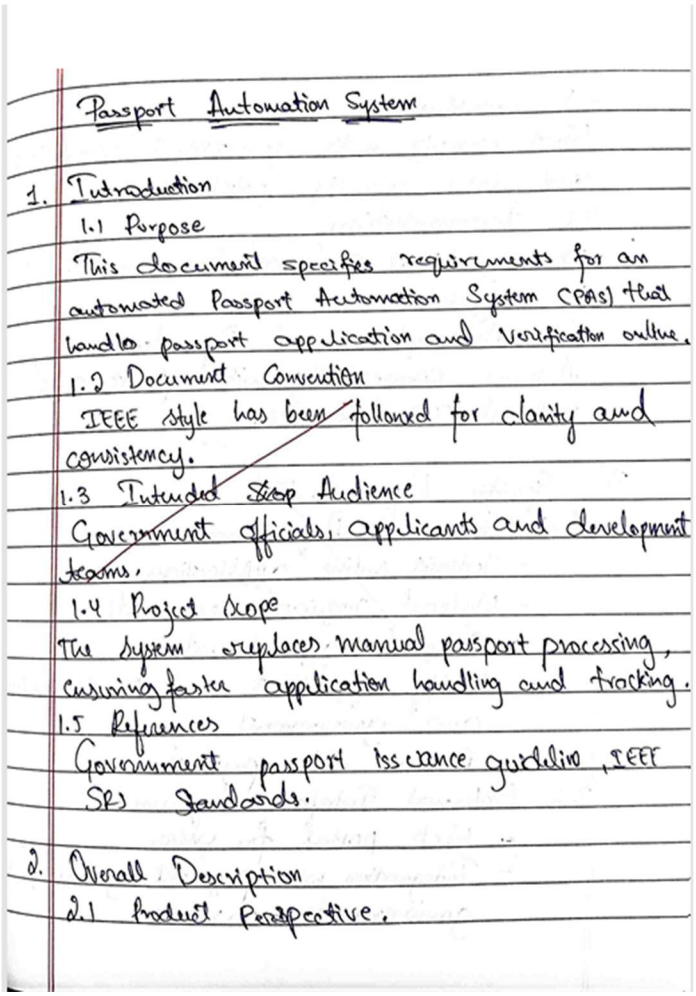
The diagram uses rounded rectangles for activities, diamonds for decisions with guard conditions in brackets, filled circles for start/end states, and arrows showing the flow direction between activities.



Passport Maintainace System

Problem Statement: The traditional passport application and processing system faces numerous inefficiencies that cause significant delays and frustration for applicants and administrative burden for passport offices.

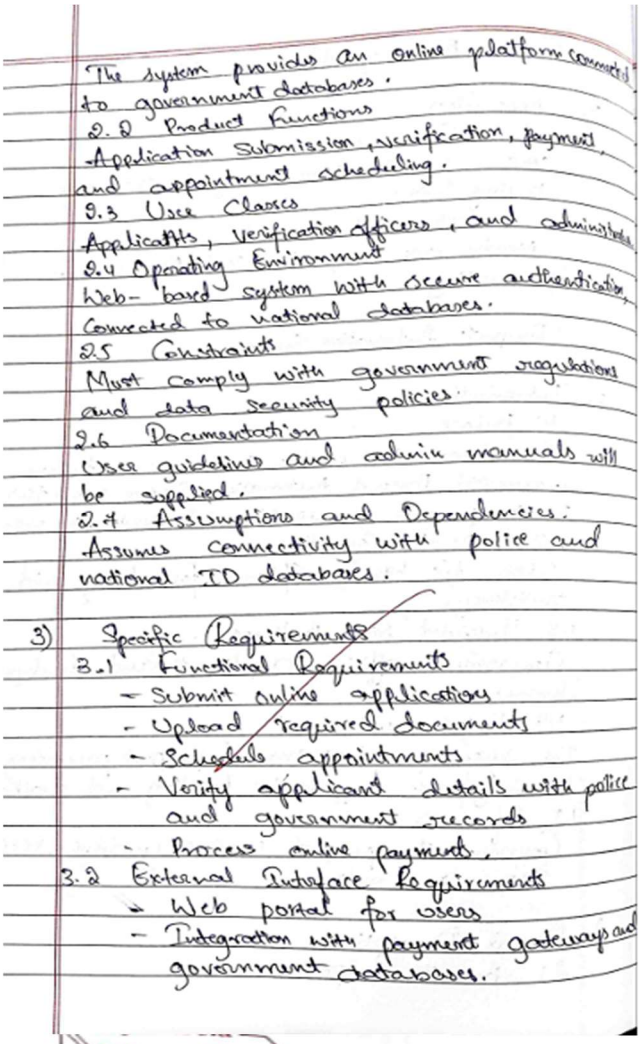
SRS:



Passport Automation System

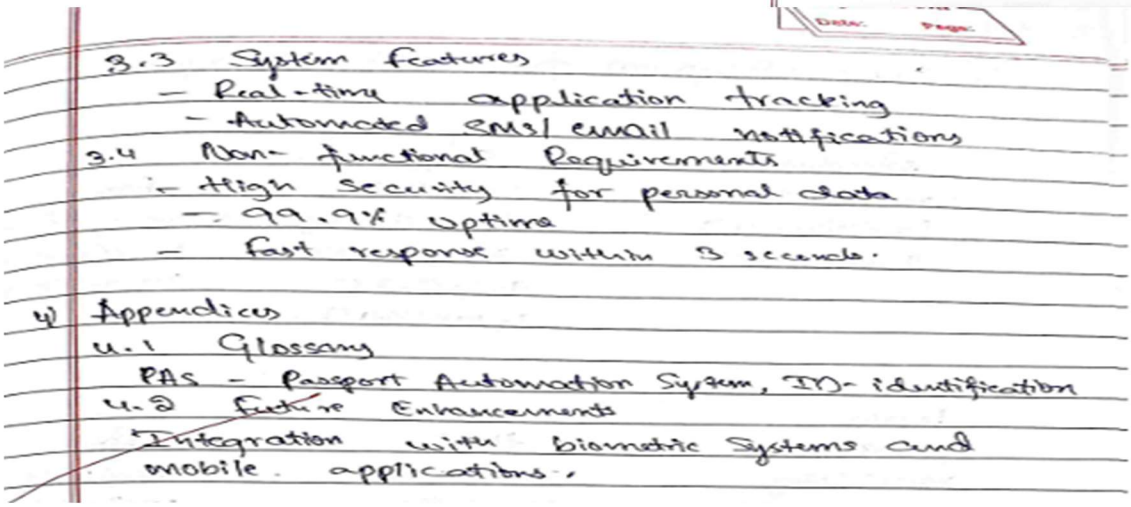
1. Introduction
1.1 Purpose
This document specifies requirements for an automated Passport Automation System (PAS) that handles passport application and verification online.
1.2 Document Convention
IEEE style has been followed for clarity and consistency.
1.3 Intended Audience
Government officials, applicants and development teams.
1.4 Project Scope
The system replaces manual passport processing, ensuring faster application handling and tracking.
1.5 References
Government passport issuance guideline, IEEE SRS standards.

2. Overall Description
2.1 Product Perspective.



The system provides an online platform connected to government databases.
2.2 Product Functions
-Application submission, verification, payment and appointment scheduling.
2.3 User Classes
Applicants, verification officers, and administrators
2.4 Operating Environment
Web-based system with secure authentication, connected to national databases.
2.5 Constraints
Must comply with government regulations and data security policies.
2.6 Documentation
User guidelines and admin manuals will be supplied.
2.7 Assumptions and Dependencies.
Assumes connectivity with police and national ID databases.

3) Specific Requirements
3.1 Functional Requirements
- Submit online applications
- Upload required documents
- Schedule appointments
- Verify applicant details with police and government records
- Process online payments.
3.2 External Interface Requirements
- Web portal for users
- Integration with payment gateways and government databases.



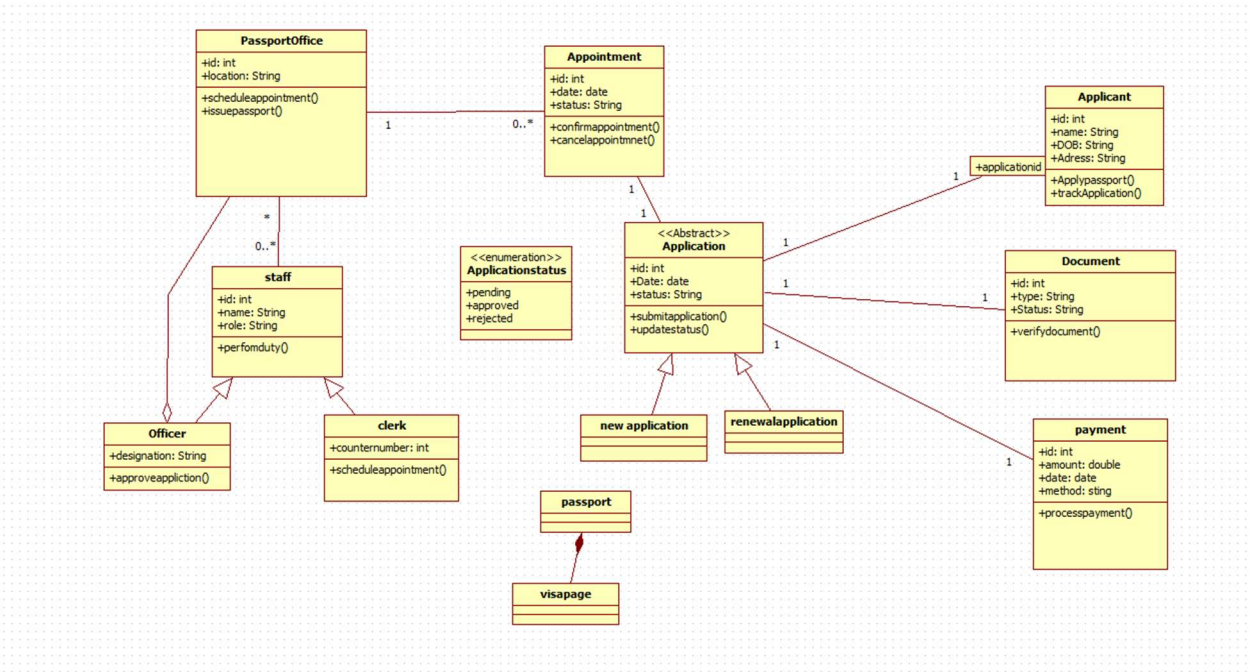
3.3 System Features
- Real-time application tracking
- Automated SMS/email notifications
3.4 Non-functional Requirements
- High security for personal data
- 99.9% uptime
- Fast response within 3 seconds.

4) Appendices
4.1 Glossary
PAS - Passport Automation System, ID- identification
4.2 Future Enhancements
Integration with biometric systems and mobile applications.

Class Model:

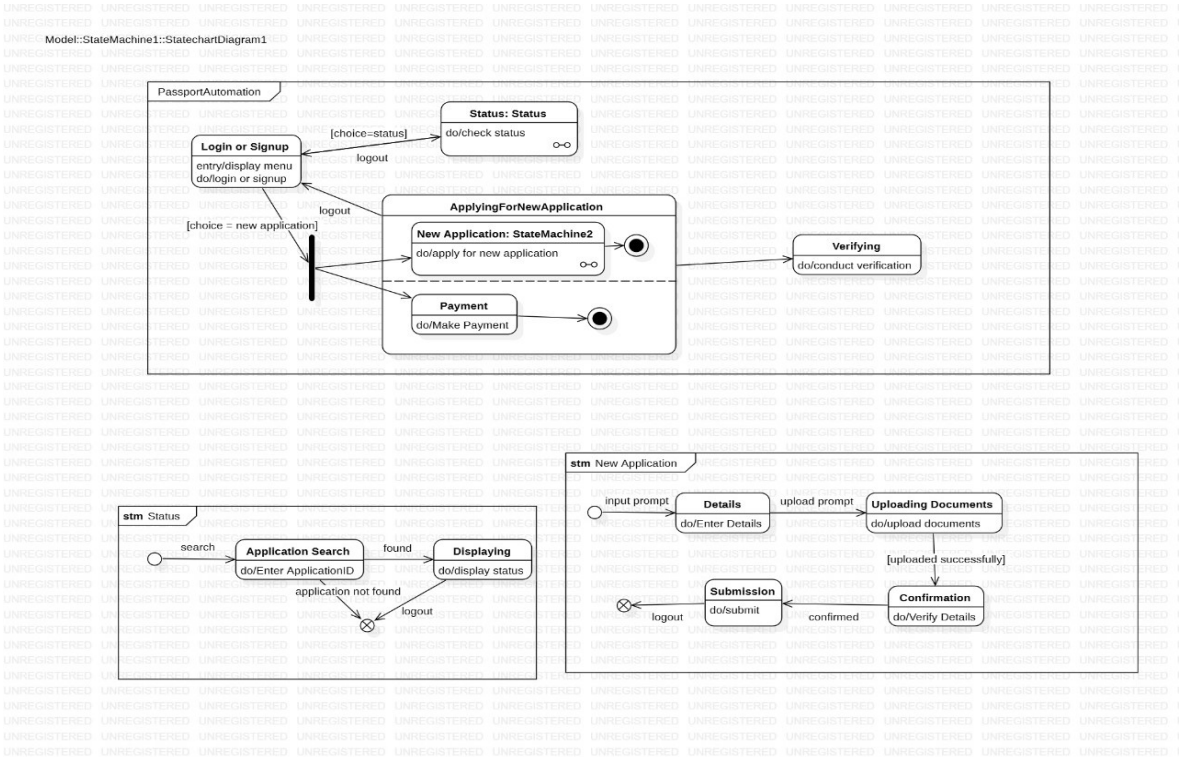
Core Classes: Application (abstract center class with id, date, and status) serves as the parent for three subclasses: new application, renewalapplication, and passport (which connects to visapage). Applicant (with name, address, and contact details) has a one-to-one relationship with Application and can apply/check application status. PassportOffice (with location attribute) manages multiple staff and appointments, and can schedule appointments and issue passports.

Staff Hierarchy: staff (abstract class with id, name, role) specializes into Officer (who can approve applications with designation attribute) and clerk (who schedules appointments with counter number). PassportOffice has a one-to-many relationship with staff members.



Supporting Classes: Appointment links applicants to passport offices with scheduling details (date, location) and includes confirm/cancel appointment methods. Document (with type and status) has a one-to-many relationship with Application for uploading required documents with verification capability. payment (with amount, date, method) connects one-to-one with Application to process payments.

State Model

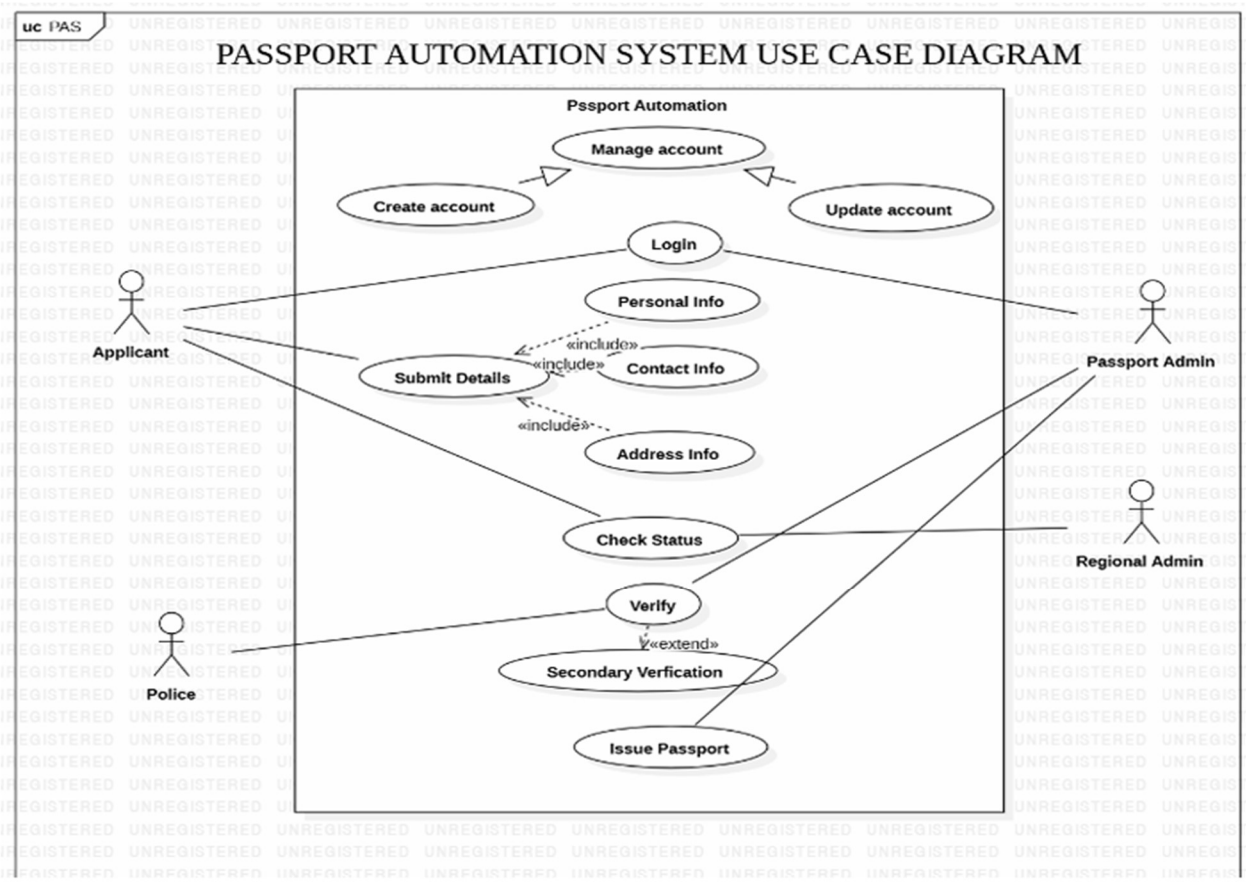


Main PassportAutomation State Machine: The process begins at "Login or Signup" where users access the menu. A choice point determines if this is a new application (logout triggers this path). The system enters "ApplyingForNewApplication" composite state containing two concurrent regions: "New Application: 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.

Use Case Model



Actors: Applicant (primary user), Passport Admin (system administrator), Regional Admin (regional authority), and Police (verification authority).

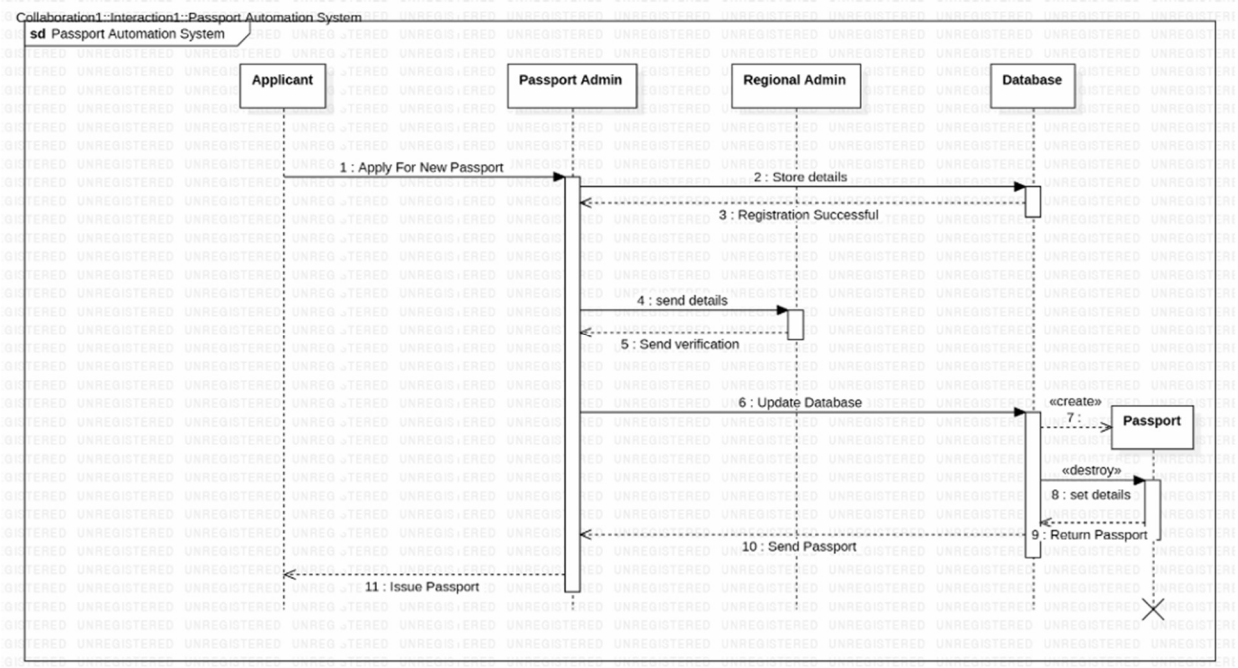
Account Management: "Manage account" serves as the parent use case with two included functions: "Create account" and "Update account" for user registration and profile management.

Application Process: Applicants "Login" to access the system and provide "Personal Info" through "Submit Details," which includes three sub-use cases: "Contact Info" and "Address Info" (both connected via «include» relationships). After submission, applicants can "Check Status" to track their application progress, which is also accessible to Passport Admin and Regional Admin.

Verification Workflow: The "Verify" use case is performed by Police, Passport Admin, and Regional Admin. It extends to "Secondary Verification" (shown with «extend» relationship) for additional scrutiny when required.

Passport Issuance: The final use case is "Issue Passport," executed by Passport Admin after successful verification, completing the application lifecycle.

Sequence Diagram



Participants: Applicant (end user), Passport Admin (system administrator), Regional Admin (verification authority), and Database (data storage), with a Passport object created during the process.

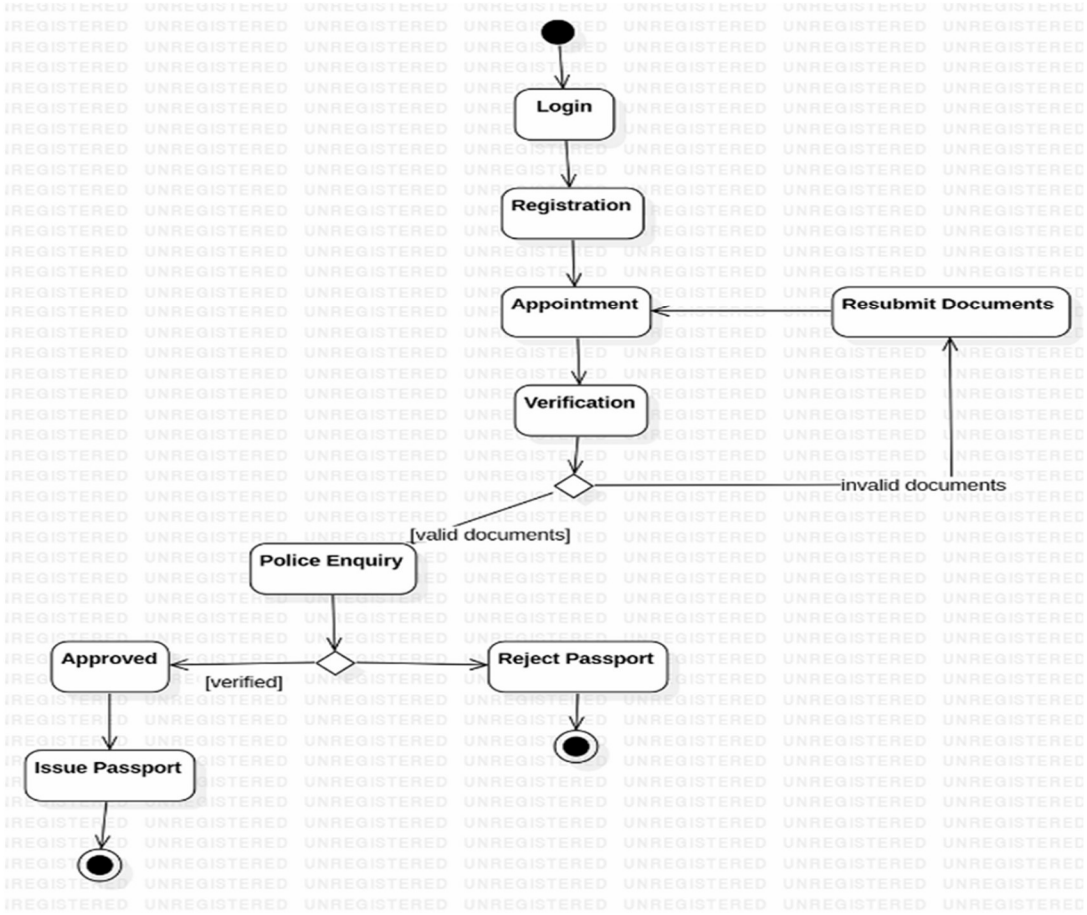
Application Flow: The Applicant initiates the process by sending "Apply For New Passport" (1) to Passport Admin, who stores the details (2) in the Database. The Database confirms "Registration Successful" (3) back to Passport Admin.

Verification Process: Passport Admin sends details (4) to Regional Admin for verification. Regional Admin reviews and sends verification (5) back to Passport Admin.

Database Update and Passport Creation: Passport Admin updates the Database (6) with verified information. The Database creates a Passport object (7) using the «create» stereotype, retrieves passport details (8), and later destroys the object (9) with «destroy» after processing is complete.

Passport Delivery: The Database returns the passport information (10) to Passport Admin, who then issues the passport (11) to the Applicant, completing the workflow.

Activity Model



The above diagram shows activity model for Passport managemnet System. From start It asks for login then to Register. After registering he get the Appointment order. It is verifies. If those documents are in valid they has to resubmit the docuents, else it goes to police enquiry. There also if it is not verified the passport got rejectes, else it is apprved then h=they can issue the passport.

It has both entry and two exits for succesful complete(issue passport) and error complete(reject passport).