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

Lab Record

Software Engineering and Object-Oriented Modeling

Submitted in partial fulfillment for the 5th Semester Laboratory

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

Abhinav Raghu

1BM22CS005

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

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Object-Oriented Analysis and Design(22CS6PCSEO) laboratory has been carried out by Abhinav Raghu (1BM22CS005) during the 5th Semester Oct 2024- Jan2025.

Signature of the Faculty Incharge:

Dr. Latha N R

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

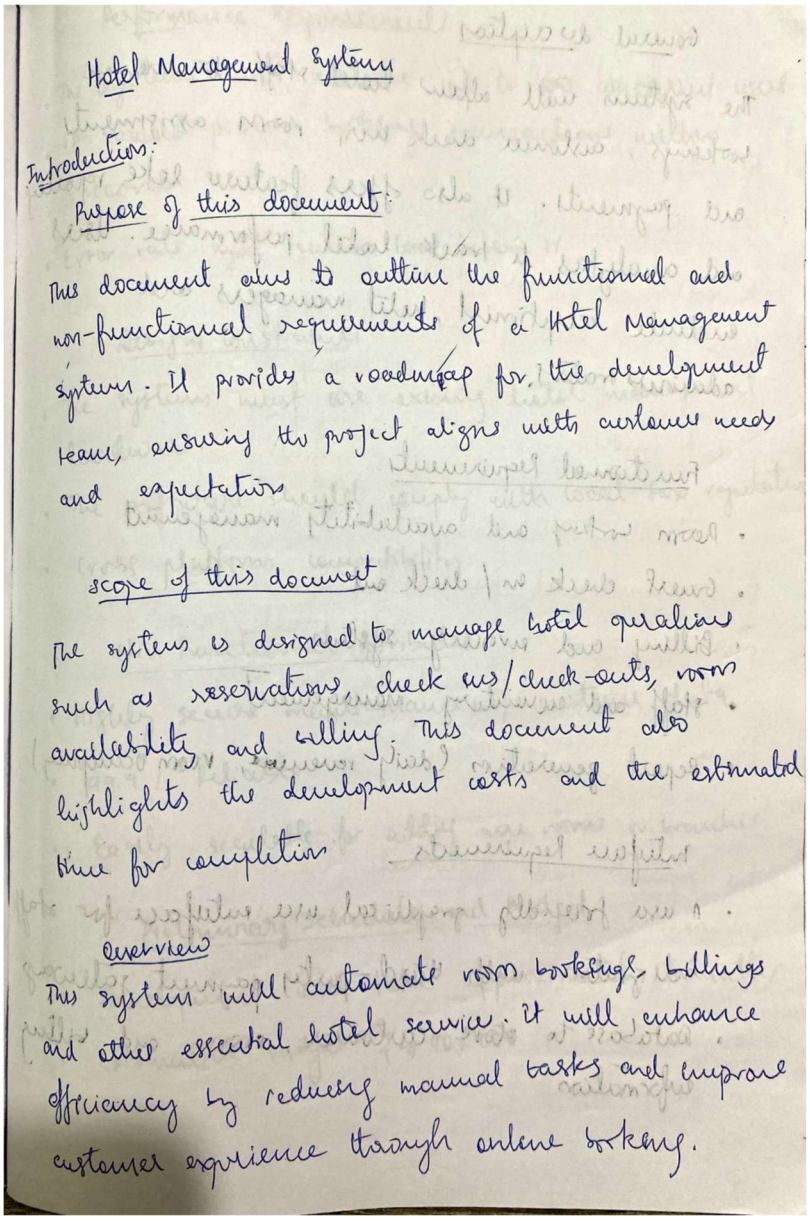
Table of Contents

Hotel Management System

1.1 Problem Statement

Software system to efficiently manage hotel operations, including customer bookings, room assignments, staff schedules, payment processing, and amenities. The system should also support inventory tracking for housekeeping and maintain connectivity across multiple hotel branches

1.2 Software Requirements Specification Document



Hotel Management System

Introduction:

Purpose of this document:

This document aims to outline the functional and non-functional requirements of a Hotel Management system. It provides a roadmap for the development team, ensuring the project aligns with customer needs and expectations

Scope of this document

The system is designed to manage hotel operations such as reservations, check ins/check-outs, room availability, and billing. This document also highlights the development costs and the estimated time for completion

Overview

This system will automate room bookings, billings and other essential hotel service. It will enhance efficiency by reducing manual tasks and improve customer experience through online booking.

Fig 1.1

General description

The system will allow hotel staff to manage bookings, customer check ins, room assignments and payments. It also offers features like reporting and analytics to track hotel performance. Users include receptionist, hotel managers and administrators.

Functional Requirements

- Room booking and availability management
- Guest check in / check out
- Billing and invoicing system
- Staff and inventory management
- Report generation (daily revenue, room occupancy)

Interface Requirements

- A user friendly graphical user interface for staff
- Integration with third-party payment gateways
- Database to store customer, room and billing informations

Fig 1.2

Performance Requirements

- The system must handle up to 500 concurrent users.
- It should process booking transactions within 3 seconds
- Error rate must remain ~~below~~ below 1%.

Design constraints

- The system must use existing hotel management hardware.
- The software should comply with local tax regulations
- Cross platform compatibility.

Non Functional Attributes

- Highly secure maintainance of customer data
- 99.9% Reliable
- Easily scalable for adding more rooms or branches

Preliminary schedule and Budget

- Development will take approximately 6 months
- Estimated budget: ₹4,00,000

Fig 1.3

1.3 Class Diagram

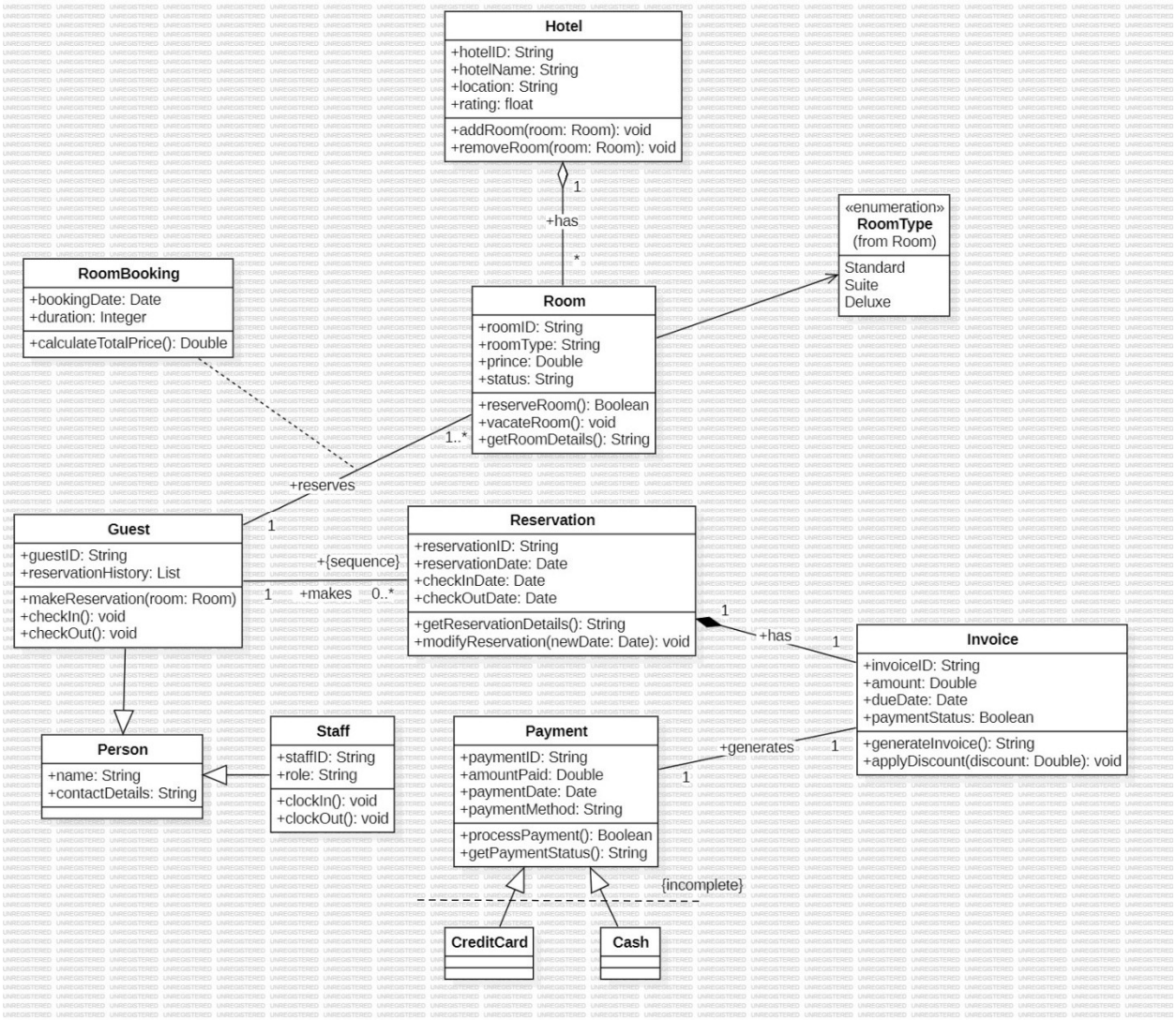


Fig 1.4

Description:

Key Classes:

- **Hotel:** Represents the hotel itself, with attributes like ID, name, location, and rating. It manages a collection of Rooms.
- **Room:** Represents a room in the hotel, with attributes like ID, type (Standard, Suite, Deluxe), price, and status (e.g., available, occupied).
- **Guest:** Represents a hotel guest, with attributes like ID, name, contact details, and a reservation history.
- **Staff:** Represents hotel staff, with attributes like ID, name, role, and clock-in/clock-out functionality.

- **Reservation:** Represents a booking made by a guest, with attributes like ID, dates (reservation, check-in, check-out), and associated Room.
- **Invoice:** Represents a bill for a reservation, with attributes like ID, amount, due date, and payment status.
- **Payment:** Represents a payment made by a guest, with attributes like ID, amount paid, date, and method (Cash or CreditCard).

Relationships:

- **Hotel - Room (Composition):** A Hotel *has* many Rooms. The diamond on the Hotel side indicates composition, meaning that if the Hotel is deleted, the Rooms are also deleted.
- **Room - RoomBooking (Aggregation):** A Room *can be part of* many RoomBookings. The empty diamond on the Room side indicates aggregation, meaning that the Room can exist independently of the RoomBooking. The multiplicity 1..* on the RoomBooking side indicates that a RoomBooking must involve at least one Room.
- **Guest - Reservation (Association):** A Guest *makes* zero or more Reservations.
- **Reservation - Room (Association):** A Reservation *is for* one Room.
- **Reservation - Invoice (Association):** A Reservation *has* one Invoice.
- **Invoice - Payment (Association):** An Invoice *generates* one Payment.
- **Person - Guest (Inheritance):** Guest *is a* Person (inherits common attributes like name and contact details).
- **Person - Staff (Inheritance):** Staff *is a* Person.
- **Payment - Cash/CreditCard (Generalization/Specialization):** Payment can be either Cash or CreditCard. This shows an inheritance relationship where Cash and CreditCard are specialized types of Payment.

Other Notable Features:

- **RoomType (Enumeration):** Defines the possible types of rooms (Standard, Suite, Deluxe).
- **Methods:** The diagram includes methods (operations) for each class, such as addRoom(), reserveRoom(), checkIn(), generateInvoice(), and processPayment().
- **Multiplicity:** The numbers on the relationship lines (e.g., 1, 0..*, 1..*) indicate the multiplicity (how many instances of one class can be related to instances of another class).

1.4 State Diagram

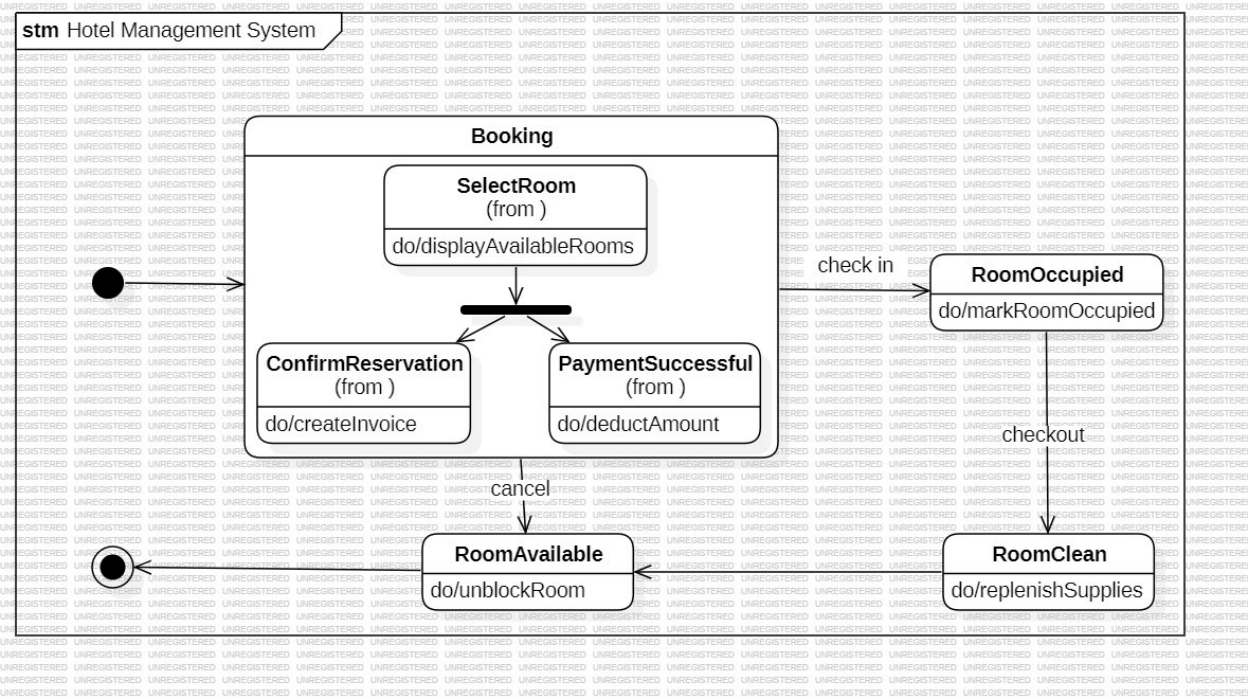


Fig 1.5

Description:

This state diagram models the lifecycle of a room in a hotel. It starts with the room being available, goes through the booking process (which includes selecting the room, confirming the reservation, and processing the payment), then to occupied when the guest checks in. After checkout, the room is cleaned, and it becomes available again. The diagram also handles the cancellation of a booking. The use of a composite state for "Booking" adds more detail to this process..

- **States :**
  1. **Booking:** This is a composite state, meaning it contains other states. It represents the process of making a reservation for the room.
     - **SelectRoom:** The room is selected by the guest.
     - **ConfirmReservation:** The reservation is confirmed, and an invoice is created.
     - **PaymentSuccessful:** The payment for the reservation is successful.
  2. **RoomOccupied:** The room is currently occupied by a guest.
  3. **RoomAvailable:** The room is available for booking.
  4. **RoomClean:** The room has been cleaned and is ready for the next guest.
- **Activities (within states):**

- **Booking.SelectRoom:** do/displayAvailableRooms - Displays the available rooms to the guest.
- **Booking.ConfirmReservation:** do/createInvoice - Creates an invoice for the reservation.
- **Booking.PaymentSuccessful:** do/deductAmount - Deducts the payment amount.
- **RoomOccupied:** do/markRoomOccupied - Marks the room as occupied in the system.
- **RoomAvailable:** do/unblockRoom - Makes the room available for booking in the system.
- **RoomClean:** do/replenishSupplies - Replenishes supplies in the room.

 Events (transitions between states):

- check in: Transitions from Booking to RoomOccupied.
- checkout: Transitions from RoomOccupied to RoomClean.
- cancel: Transitions from Booking to RoomAvailable.

 Guard Conditions (Not explicitly shown but implied):

- The transition from Booking to RoomOccupied (check in) likely has a guard condition: the reservation must be confirmed and paid for.
- The transition from Booking to RoomAvailable (cancel) likely has a guard condition: the time limit for cancelling the reservation has not passed.

 Concurrency: The Booking state is a composite state with concurrent substates. Specifically, after SelectRoom, the process can either proceed to ConfirmReservation and then PaymentSuccessful or be cancelled, going to RoomAvailable. This represents a form of implicit concurrency, as the reservation and payment steps are part of the broader booking process.

1.5 Use case diagram

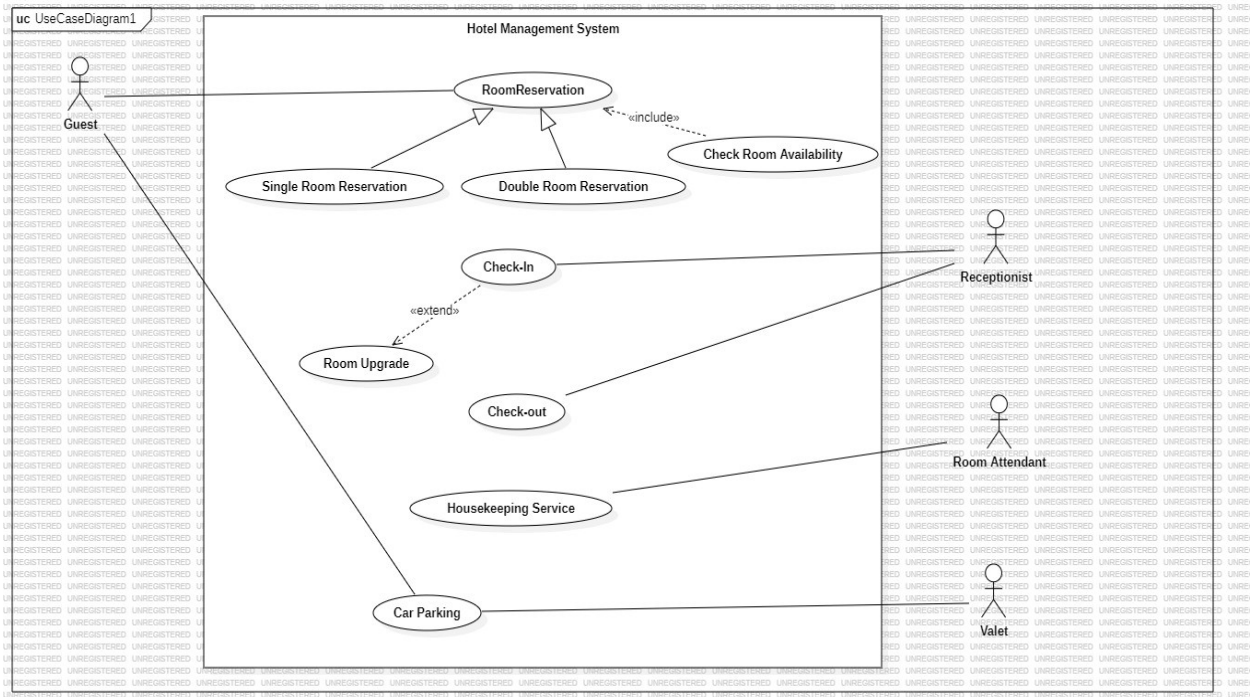


Fig 1.6

Description:

- A guest can make a room reservation, which involves checking the availability of rooms. The guest can choose between a single or double room reservation.
- A receptionist handles the check-in and check-out processes. During check-in, a guest *may* request a room upgrade, which extends the basic check-in procedure.
- A room attendant is responsible for housekeeping services.
- A valet provides car parking services.

 Use Cases:

1. **Room Reservation:** This is a general use case that includes checking room availability.
2. **Single Room Reservation:** A specific type of room reservation.
3. **Double Room Reservation:** Another specific type of room reservation.
4. **Check-In:** The process of a guest arriving and receiving their room.

5. **Check-Out:** The process of a guest leaving the hotel.
6. **Housekeeping Service:** Requesting and performing cleaning services for a room.
7. **Car Parking:** Using the hotel's parking facilities.
8. **Room Upgrade:** Upgrading the reserved room.

 Actors:

- **Guest:** The person making reservations and staying at the hotel.
- **Receptionist:** The hotel staff member handling check-in and check-out.
- **Room Attendant:** The staff member responsible for cleaning rooms.
- **Valet:** The staff member responsible for parking cars.

1.6 Sequence Diagram

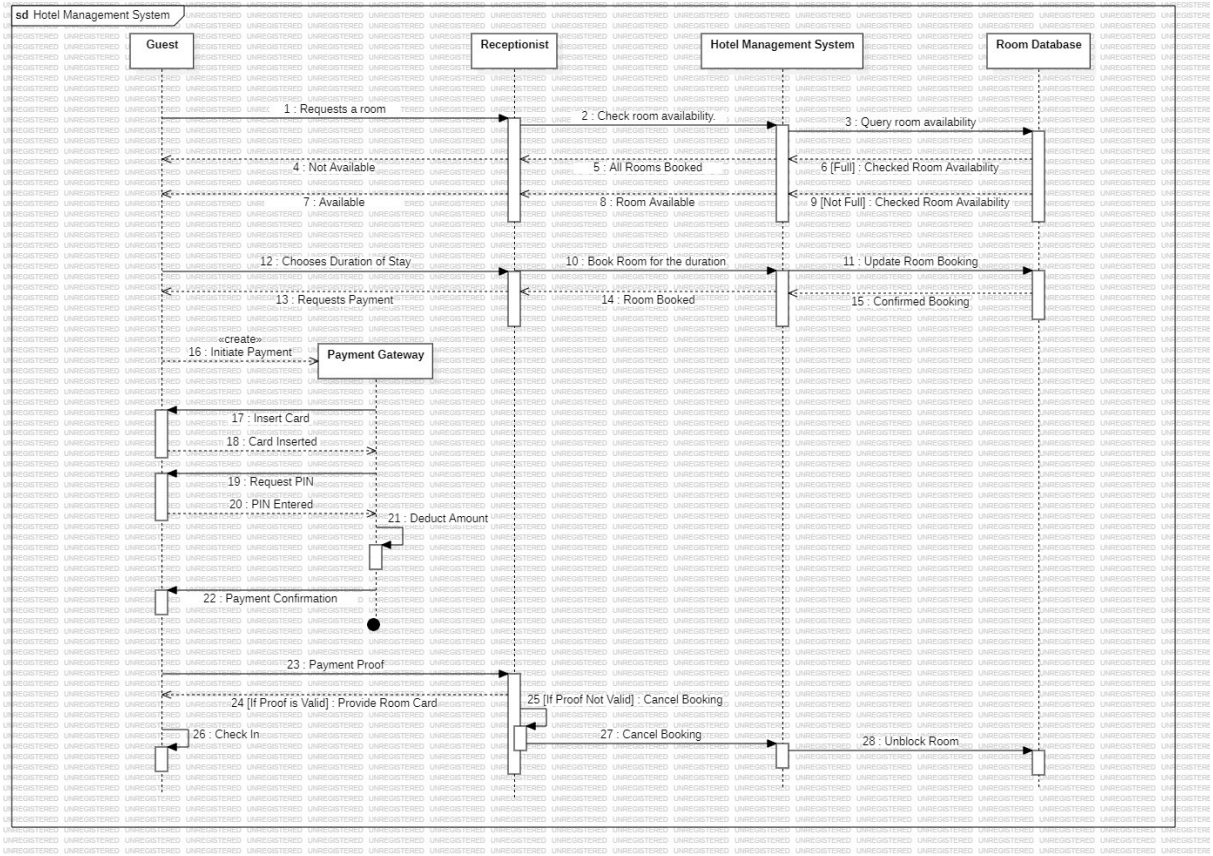


Fig 1.7

Description:

1. Use Case: Check-In

Scenario: A guest, John Smith, arrives at the hotel with a reservation under his name for a standard room from July 20th to July 23rd. He approaches the reception desk and presents his confirmation. The receptionist verifies the reservation details in the system, confirms the room is available, assigns room 204, provides John with a key card, and marks him as checked in. In a variant scenario, John requests an upgrade to a suite if available, which the receptionist checks and grants if possible.

2. Use Case: Housekeeping Service

Scenario: A guest in room 302 calls the reception desk to request housekeeping service. The receptionist logs the request in the system, specifying room 302 and the type of service needed

(general cleaning). The system notifies the assigned room attendant, who then proceeds to clean the room. After completing the cleaning, the room attendant updates the room status in the system.

Brief Explanation

This sequence diagram illustrates the flow of a room reservation, encompassing both successful and unsuccessful scenarios. The Guest interacts with the Receptionist, who communicates with the Hotel Management System and the Room Database to check availability and book rooms. The Payment Gateway is used for payment processing. The diagram uses combined fragments (implied by the branching after message 22) to show the different outcomes based on payment success or failure. In a successful reservation (scenario 1), the guest receives a room card and checks in. In a failed reservation (scenario 2), the booking is canceled, and the room is made available again. The Room Database is a passive object, while the Payment Gateway can be considered passive in this context as well, as it is invoked by the Hotel Management System. No transient objects are explicitly shown in this diagram, but in a more detailed version, a "Booking" or "Payment Transaction" object could be introduced.

1.7 Activity Diagram

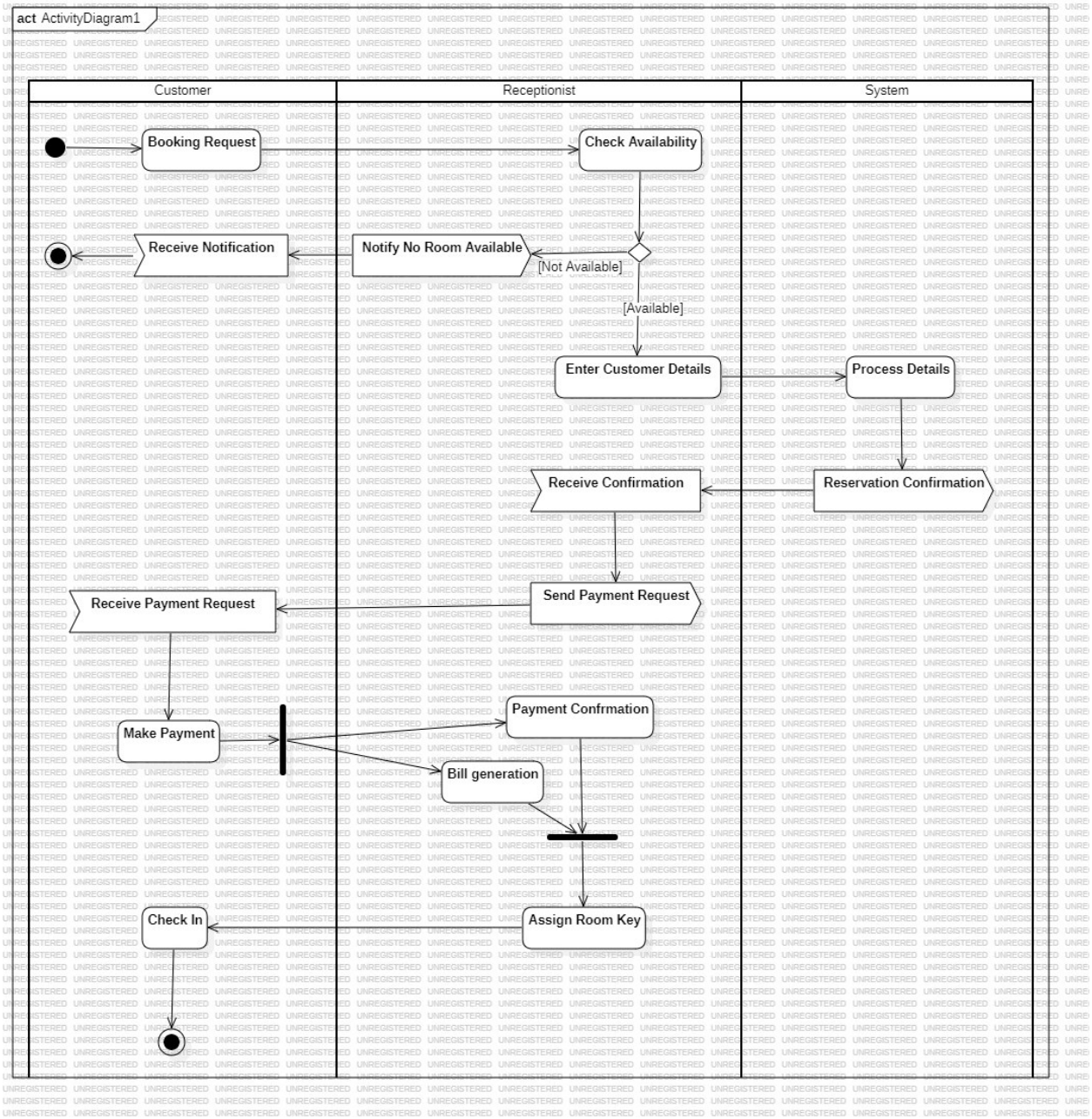


Fig 1.8

Description:

Swimlanes: The diagram is divided into three vertical lanes representing the different actors involved in the process:

- **Customer:** This lane represents the actions performed by the customer.
- **Receptionist:** This lane represents the actions performed by the hotel receptionist.
- **System:** This lane represents the actions performed by the hotel's computer system.

Brief explanation:

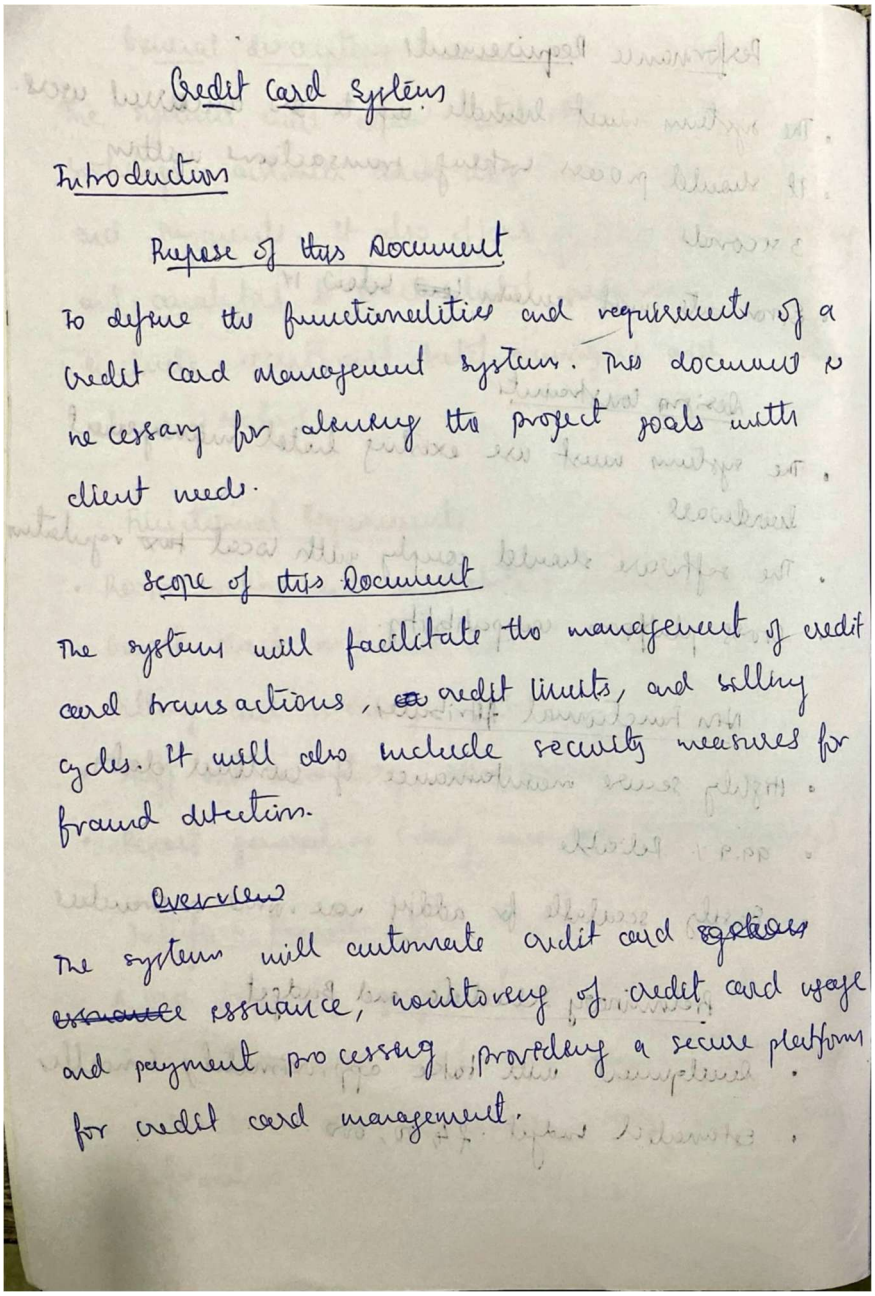
This activity diagram illustrates the process of booking a hotel room. It begins with the Customer making a Booking Request. The Receptionist then checks availability with the System. A decision point follows: if no rooms are available, the Receptionist notifies the Customer. If rooms are available, the Receptionist enters the Customer's details, which the System processes, generating a Reservation Confirmation. The System then sends a Payment Request to the Customer, who makes the payment. Upon Payment Confirmation by the System, a Bill is generated (this happens concurrently with the next step), and the Receptionist assigns a Room Key. Finally, the Customer checks in, marking the end of the process. The diagram clearly shows the flow of actions between the Customer, Receptionist, and the System, highlighting the decision point for availability and the concurrent processes of payment and bill generation.

Credit Card Processing System

2.1 Problem Statement

A system for secure credit card processing that handles customer transactions, fraud detection, credit limit validations, and billing cycles. The system should also integrate with multiple payment gateways and provide detailed transaction histories for both users and administrators.

2.2 SRS Document



Credit Card System

Introduction

Purpose of this Document

To define the functionalities and requirements of a credit card management system. This document is necessary for aligning the project goals with client needs.

Scope of this Document

The system will facilitate the management of credit card transactions, credit limits, and billing cycles. It will also include security measures for fraud detection.

Overview

The system will automate credit card issuance, monitoring of credit card usage and payment processing, providing a secure platform for credit card management.

Fig 2.1

General description

The credit card system will allow users to manage card applications, monitor transactions, handle bill payments and check credit limits. User groups include customers, bank staff and administrators.

Functional Requirements:

- Credit card application and approval
- Transaction tracking and statement generation
- Fraud detection and reporting
- Credit limit management

Interface Requirements

- User interfaces for card holders, bank staff, admins
- Integration with banking core systems
- Secure API for 3rd party payment systems.

Performance Requirements

- Process transactions within 2 seconds
- Handle 100,000 daily transactions with no downtime
- Maintain Error rate below 0.5%.

Fig 2.2

Design Contracts

- limited to bank-approved security algorithms
- must work legacy banking systems.

Non-Functional Attributes

- High-level security with encryption for data storage
- 24/7 availability
- Fault-tolerant for zero transactions loss.

Preliminary Schedule and Budget

- Estimated development time: 8 months
- Budget: ₹9,00,000

Fig 2.3

2.3 Class Diagram

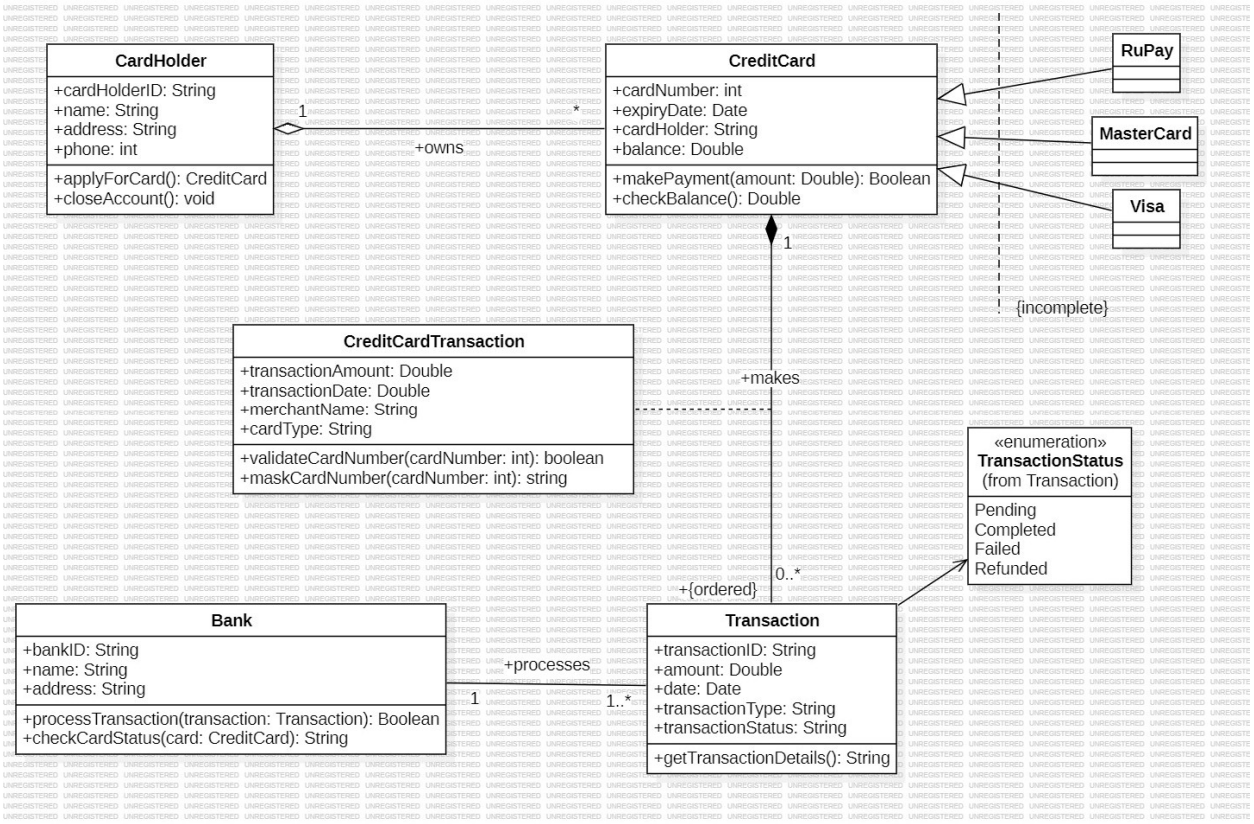


Fig 2.4

Description:

Classes

- **CardHolder:**
  - Attributes: cardHolderID (String), name (String), address (String), phone (int)
  - Operations: applyForCard() (returns CreditCard), closeAccount() (void)
- **CreditCard:**
  - Attributes: cardNumber (int), expiryDate (Date), cardHolder (String), balance (Double)
  - Operations: makePayment(amount: Double) (returns Boolean), checkBalance() (returns Double)
- **Bank:**
  - Attributes: bankID (String), name (String), address (String)
  - Operations: processTransaction(transaction: Transaction) (returns Boolean), checkCardStatus(card: CreditCard) (returns String)
- **Transaction:**

- Attributes: transactionID (String), amount (Double), date (Date), transactionType (String), transactionStatus (String)
- Operations: getTransactionDetails() (returns String)

 CreditCardTransaction: (Association Class)

- Attributes: transactionAmount (Double), transactionDate (Double), merchantName (String), cardType (String)
- Operations: validateCardNumber(cardNumber: int) (returns boolean), maskCardNumber(cardNumber: int) (returns string)

 RuPay, MasterCard, Visa: (Specializations of CreditCard)

 TransactionStatus: (Enumeration)

- Values: Pending, Completed, Failed, Refunded

This class diagram models a credit card transaction system. It defines the key entities involved, including cardholders, credit cards (with different types like RuPay, MasterCard, and Visa), banks, and transactions. The CreditCardTransaction association class captures details specific to credit card usage within a transaction. The diagram shows how cardholders own credit cards, how banks process transactions, and how transactions are linked to specific credit card transactions. The use of generalization, composition, and enumeration enhances the model's expressiveness. The ordering of transactions related to a credit card is also specified.

2.4 State Diagram

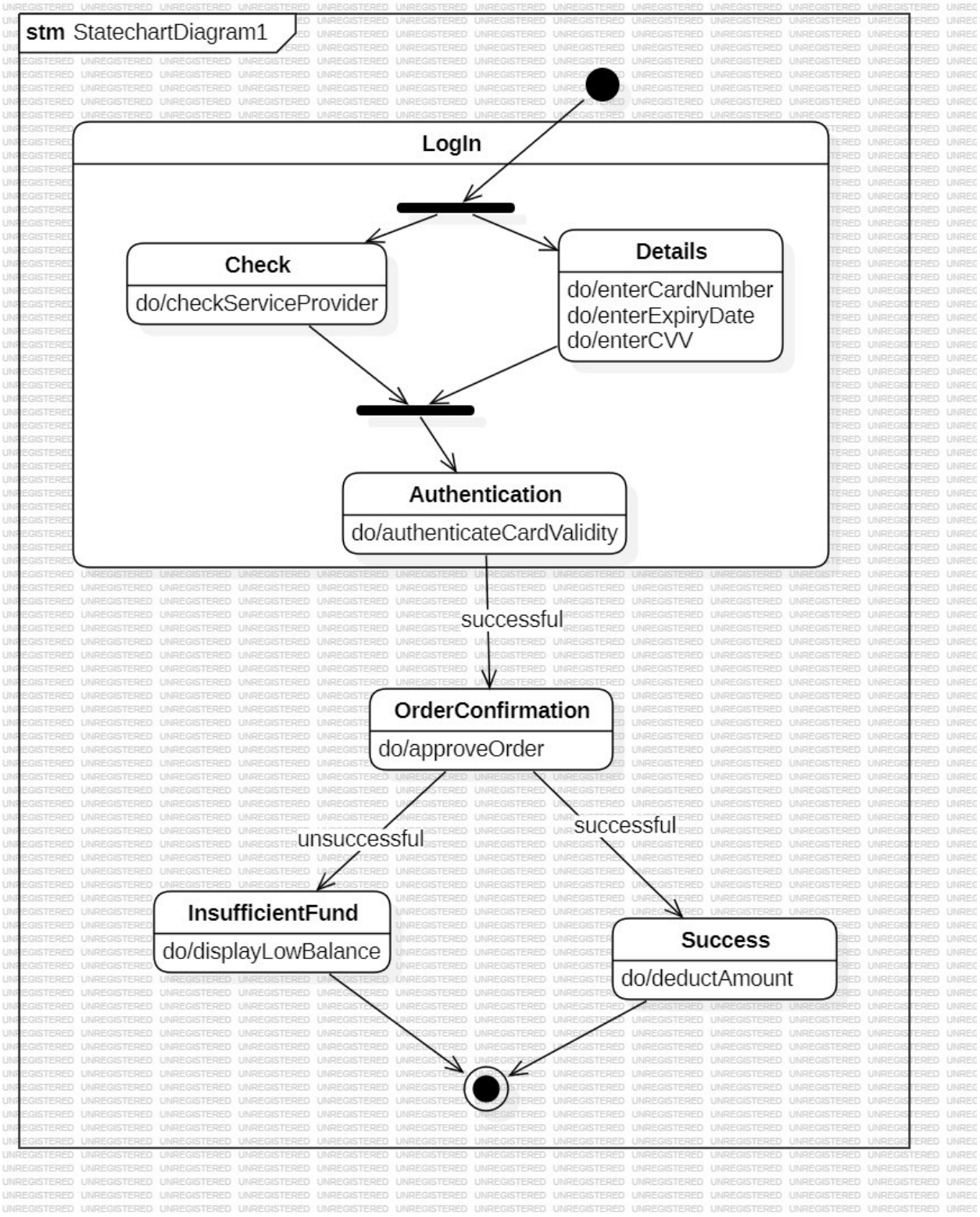


Fig 2.5

Description:

his state diagram models the process of online payment. It starts with a login process where the system checks the service provider and the user enters their card details. The card's validity is then authenticated. If successful, the order is confirmed with the user. If the authentication fails or the order is not confirmed, the system goes to the InsufficientFund state. If the order is confirmed, the payment amount is deducted, and the process ends in the Success state. The composite state Login and the implied concurrency within it are key aspects of this diagram. The guard conditions on the transitions control the flow based on the success or failure of the preceding steps.

2.5 Use case diagram

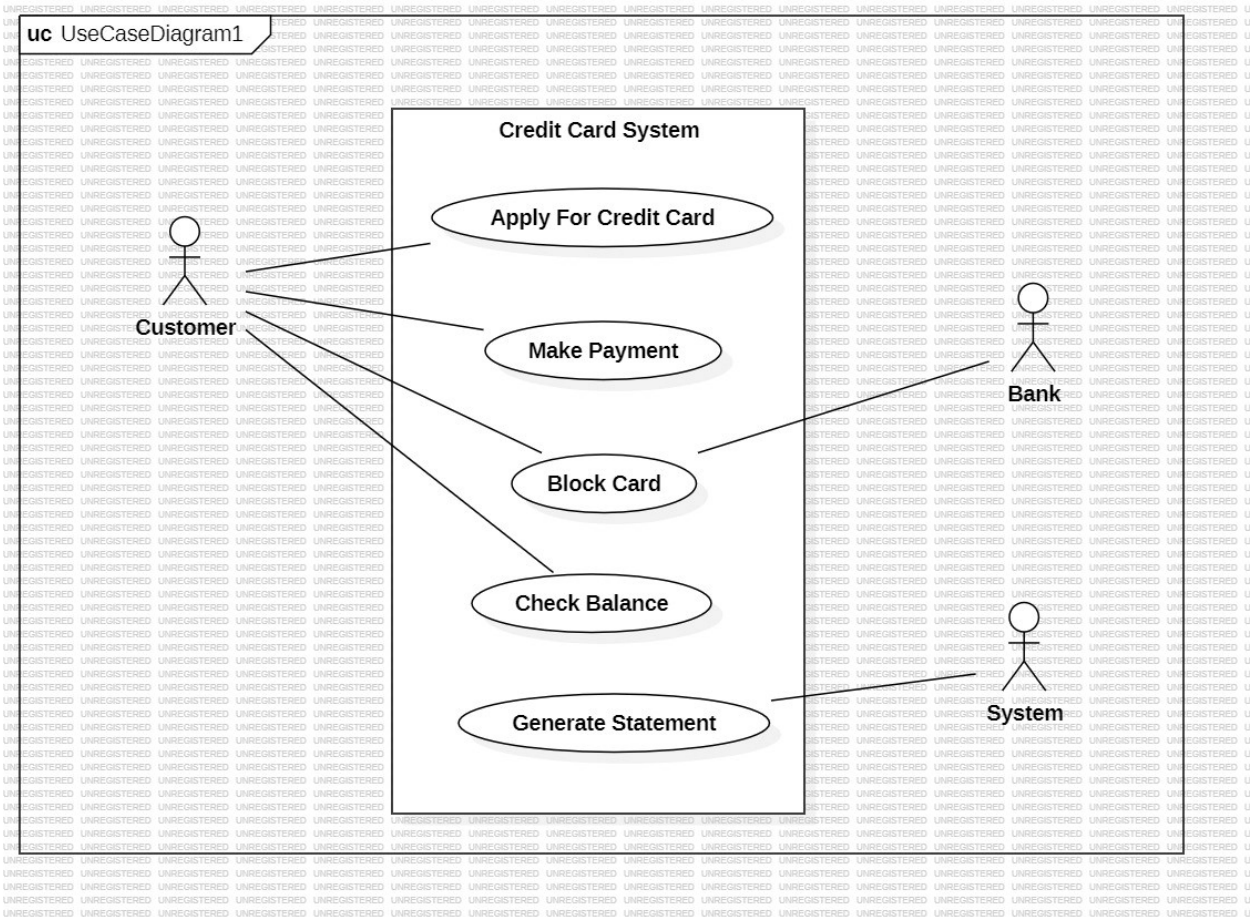


Fig 2.6

Description:

1. **Apply For Credit Card:** A customer applies for a new credit card.

2. **Make Payment:** A customer uses their credit card to make a payment.
3. **Block Card:** A customer or the bank blocks a credit card (e.g., due to loss or suspected fraud).
4. **Check Balance:** A customer checks their credit card balance.
5. **Generate Statement:** The system generates a credit card statement.

 Actors:

- **Customer:** The person who owns and uses the credit card.
- **Bank:** The financial institution issuing the credit card.
- **System:** The automated system responsible for managing credit card accounts and transactions.

 Relationships: There are no explicit include, extend, or generalization relationships shown in this diagram. All relationships are simple associations indicating that an actor *participates in* or *initiates* a use case.

 Explanation:

- A Customer can apply for a credit card, make payments with their card, block their card (or have it blocked by the Bank), and check their balance.
- The Bank can also block a card (independently of customer action, for instance, in case of fraud detection).
- The System is responsible for generating credit card statements.

2.6 Sequence Diagram

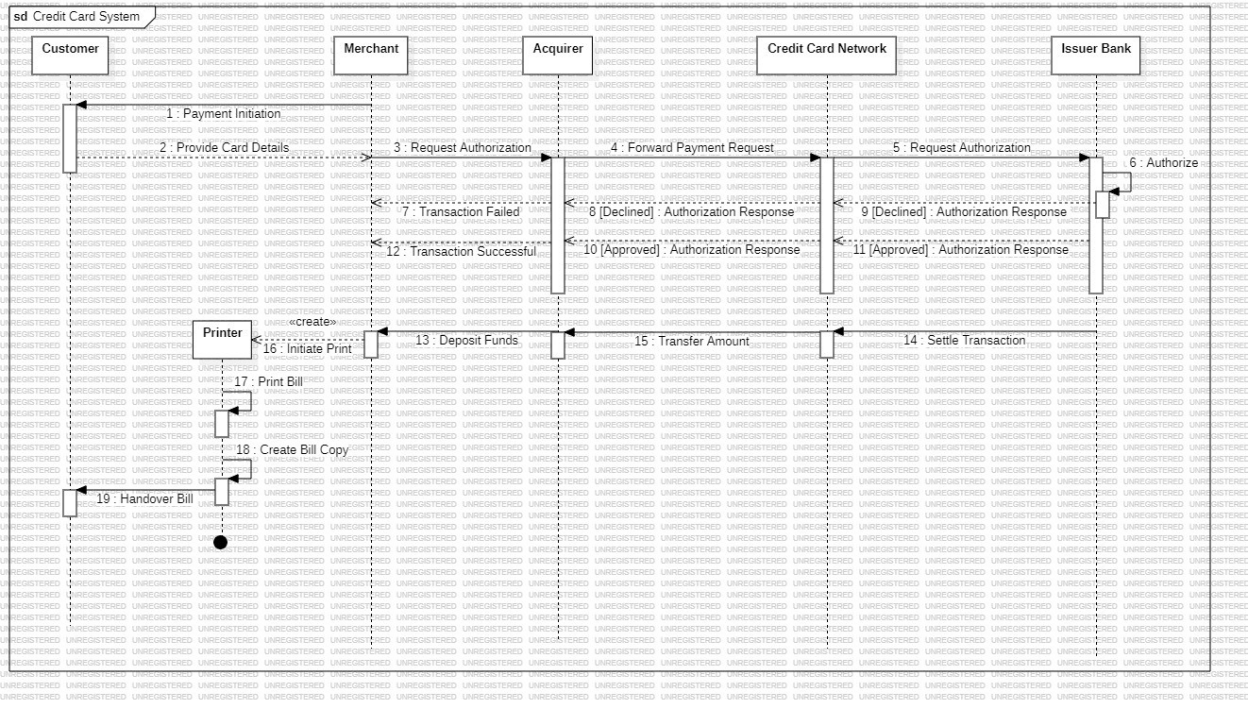


Fig 2.7

Description:

1. Use Case: Make Payment (as depicted in the provided diagram)

Scenario: A customer wishes to purchase goods at a merchant's store using their credit card. They present their card to the merchant, who initiates the payment process through their point-of-sale system. The system communicates with the acquirer (the merchant's bank), which in turn contacts the credit card network. The network routes the authorization request to the issuer bank (the customer's bank). The issuer bank approves or declines the transaction based on the customer's available funds and card status. The response travels back through the same chain. If approved, the funds are transferred, and a receipt (bill) is generated for the customer. If declined, the customer is informed of the declined transaction.

2. Use Case: Check Balance

Scenario: A customer wants to check their current credit card balance. They access their bank's online banking website or mobile app and navigate to the credit card section. They select the option to view their balance. The bank's system retrieves the balance information from its database and displays it to the customer.

Brief Description of the Provided Sequence Diagram:

The provided sequence diagram illustrates the "Make Payment" use case. It shows the interaction between the Customer, Merchant, Acquirer, Credit Card Network, and Issuer Bank during a credit card transaction. The diagram depicts the flow of messages for both successful and failed transactions. It starts with the customer initiating the payment and providing their card details to the merchant. The merchant then requests authorization through the acquirer and the credit card network to the issuer bank. The issuer bank authorizes or declines the transaction, and the response is sent back through the same path. If the transaction is approved, funds are transferred, and a bill is printed. If declined, the transaction ends. The Printer object is created during the transaction to print the bill. The diagram clearly shows the complex interactions involved in a credit card payment, including the handling of both successful and declined transactions. The use of conditional logic (shown by the bracketed messages like "[Declined]" and "[Approved]") illustrates the different paths the process can take.

2.7 Activity Diagram

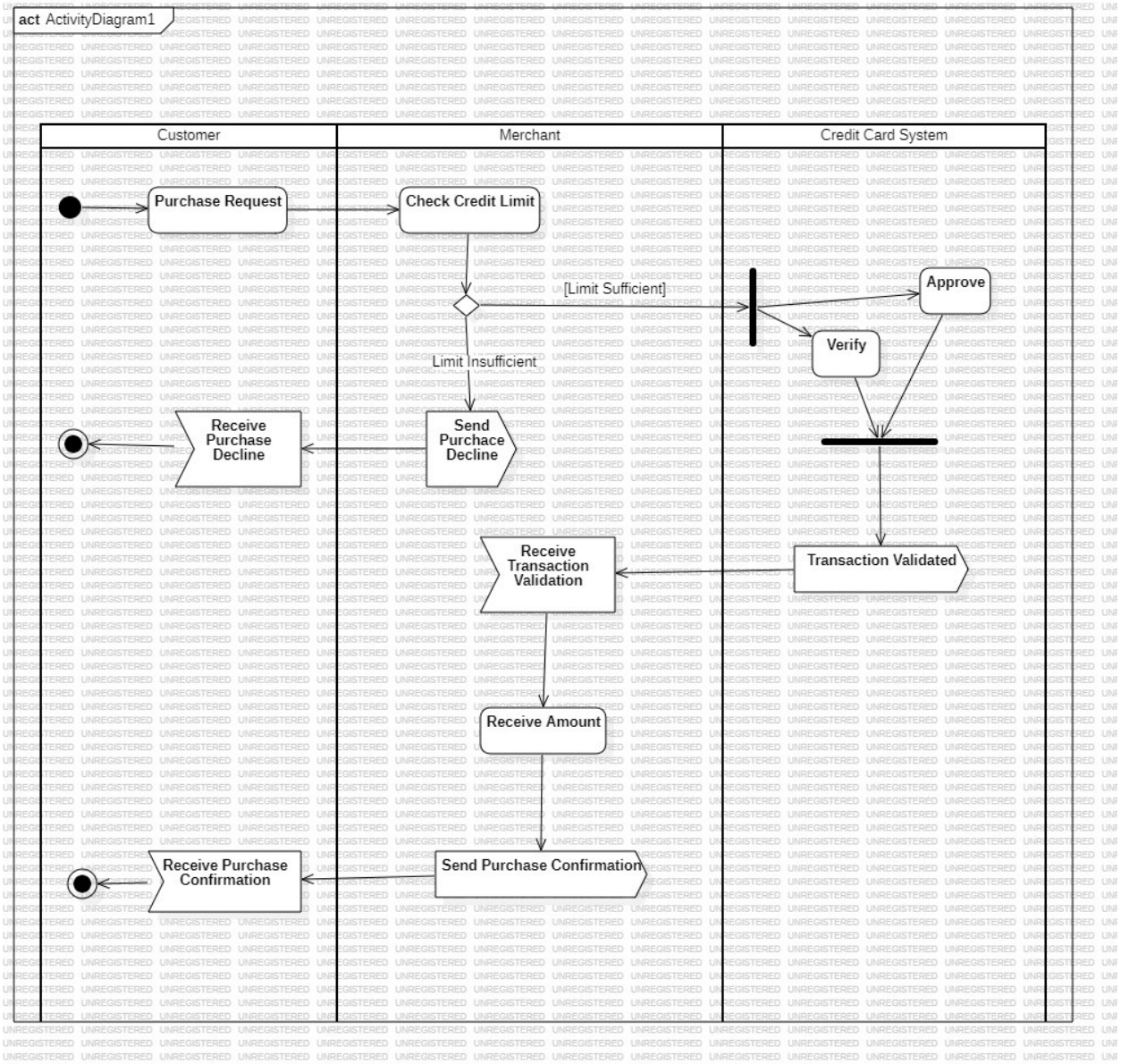


Fig 2.8

Description:

1. Swimlanes with responsible persons:

- **Customer:** Responsible for initiating the purchase request, receiving purchase confirmation or decline notifications.
- **Merchant:** Responsible for checking the credit limit, sending purchase confirmation or decline notifications, receiving transaction validation, and receiving the amount.

- **Credit Card System:** Responsible for verifying the transaction and approving or declining it based on the verification.

2. Brief explanation:

This activity diagram illustrates the process of a purchase using a credit card. The process begins with the Customer initiating a Purchase Request. The Merchant then checks the Credit Limit with the Credit Card System. A decision point follows:

- **Limit Sufficient:** If the limit is sufficient, the Credit Card System Verifies the transaction and either Approves it (leading to Transaction Validated) or declines it. The Merchant then receives Transaction Validation from Credit Card System and receives the amount. Finally, the Merchant sends a Purchase Confirmation to the Customer.
- **Limit Insufficient:** If the limit is insufficient, the Merchant sends a Purchase Decline notification to the Customer.

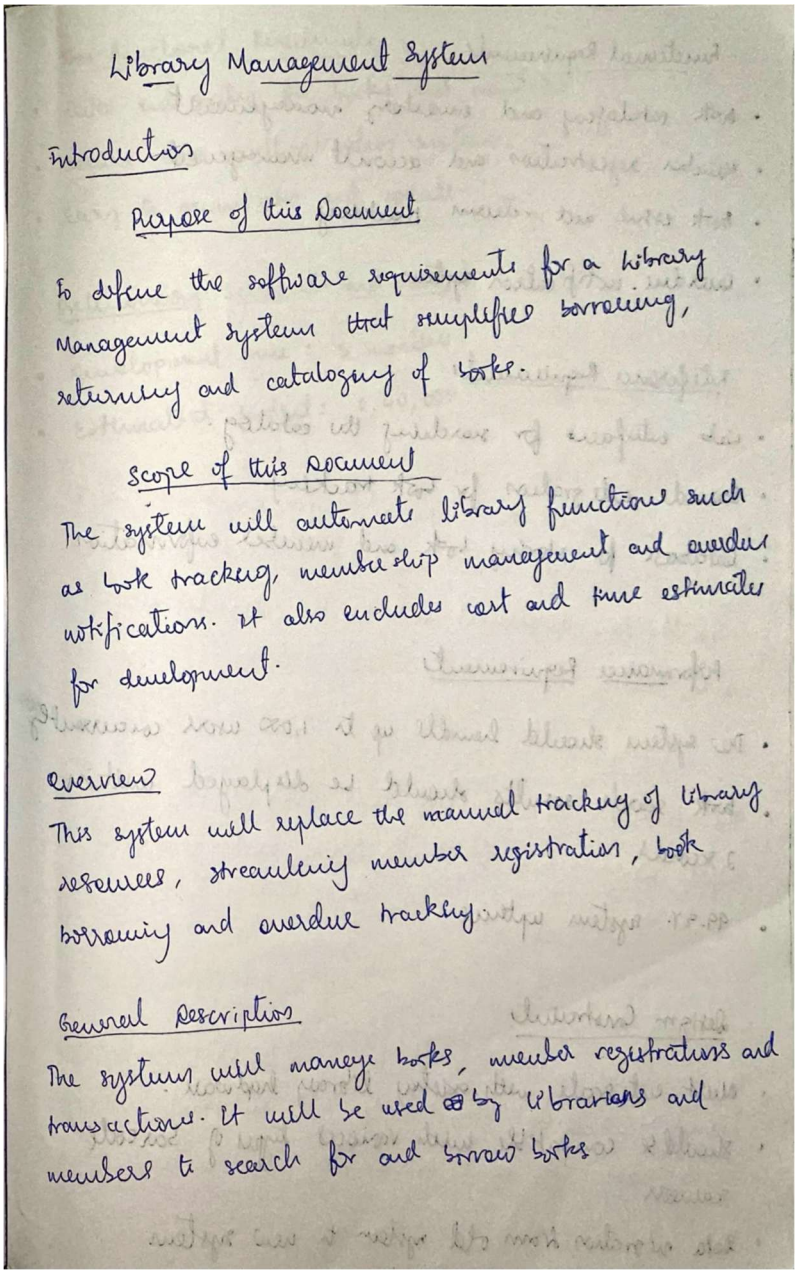
The process ends with the Customer either receiving a Purchase Confirmation (successful transaction) or a Purchase Decline (failed transaction). The diagram clearly shows the flow of actions between the Customer, Merchant, and the Credit Card System, highlighting the decision point based on the credit limit and the subsequent verification process.

Library Management System

3.1 Problem Statement

A system to manage library operations, including book inventory, member registrations, borrowing and returning of books, late fee calculations, and digital catalog search. The system should also enable administrators to track book availability and manage reservations.

3.2 Software Requirements Specification (SRS) Document



Library Management System

Introduction

Purpose of this Document

To define the software requirements for a library Management system that simplifies borrowing, returning and cataloging of books.

Scope of this Document

The system will automate library functions such as book tracking, membership management and overdue notifications. It also includes cost and time estimates for development.

Overview

This system will replace the manual tracking of library resources, streamlining member registration, book borrowing and overdue tracking.

General Description

The system will manage books, member registrations and transactions. It will be used by librarians and members to search for and borrow books

Fig 3.1

Functional Requirements

- Book cataloging and inventory management
- Member registration and account management
- Book issue and return tracking
- Overdue notification system.

Interference Requirements

- Web interface for searching the catalog.
- Barcode integration for book tracking.
- Database for storing book and member information.

Performance Requirement

- The system should handle up to 1,000 users concurrently
- Book search results should be displayed within 2 seconds
- 99.9% system uptime

Design Constraint

- Must integrate with existing library hardware.
- Should be compatible with various types of barcode scanners.
- Data migration from old system to new system

Fig 3.2

Non-Functional Requirements

- Data entegrity for books and members
- Secure user authentication system.
- Easy to maintain and update.

Preliminary Schedule and Budget

- Development time : 5 months
- Estimated budget: 2,40,000

Fig 3.3

3.3 Class Diagram

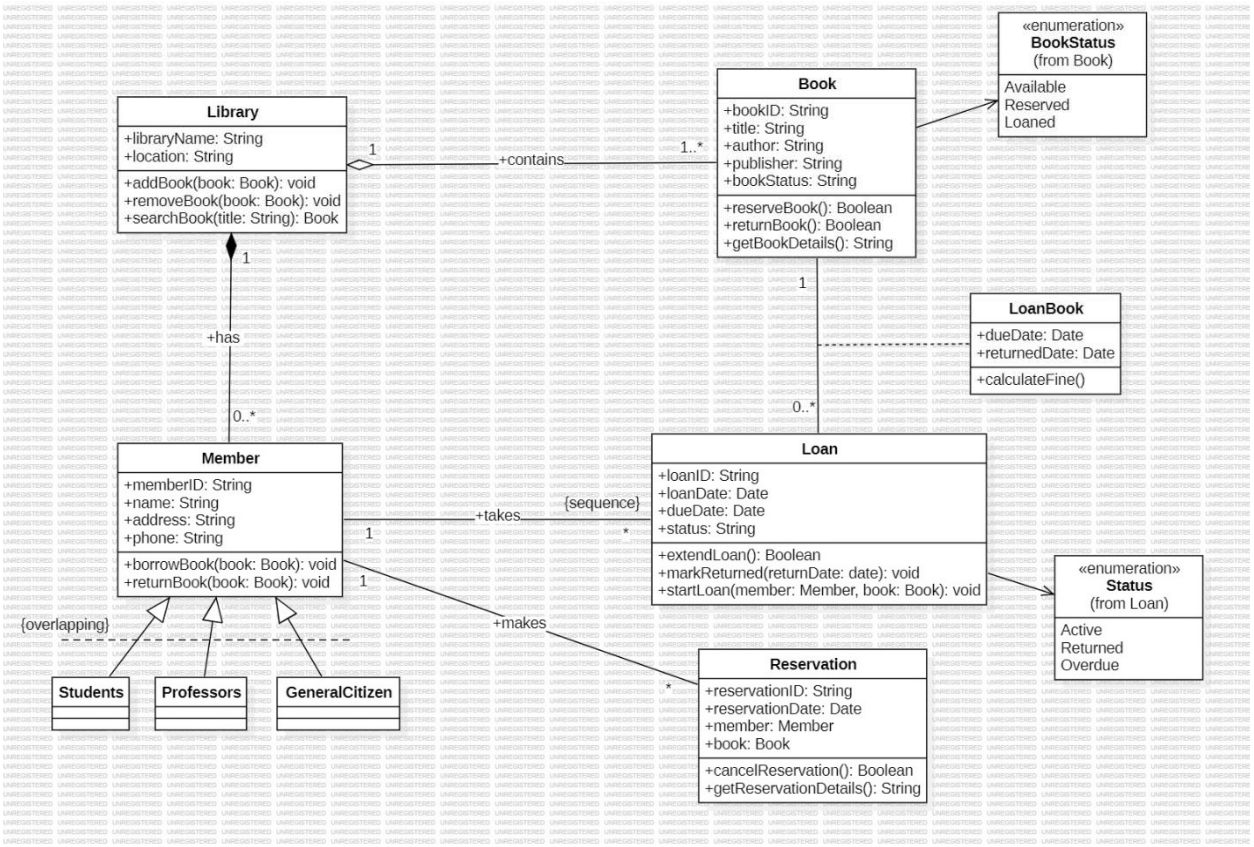


Fig 3.4

Description:

Classes:

- **Book:**
    - Attributes: bookID, title, author, publisher, status (enum: Available, Reserved, Loaned)
    - Operations: reserveBook(), returnBook(), getBookDetails()
- **Library:**
    - Attributes: libraryName, location
    - Operations: addBook(book: Book), removeBook(book: Book), searchBook(title: String)
- **LoanBook:**
    - Attributes: loanID, dueDate, returnedDate
    - Operations: calculateFine()
- **Loan:**

    - Attributes: loanID, loanDate, dueDate, status (enum: Active, Returned, Overdue)
    - Operations: extendLoan(), markReturned(returnDate: date), startLoan(member: Member, book: Book)
- **Member:**
    - Attributes: memberID, name, address, phone
    - Operations: borrowBook(book: Book), returnBook(book: Book)
- **Student**, **Professor**, **GeneralCitizen:** (Specializations of Member)

Relationships:

- **Library contains many Book:** Composition.
- **LoanBook has a Loan:** Composition.
- **Loan has a Member and a Book:** Association.
- **Reservation has a Member and a Book:** Association.
- **Student, Professor, GeneralCitizen are specializations of Member:** Generalization.

Brief Description:

This class diagram represents a library management system. It shows the entities involved, including books, libraries, members, loans, reservations, and different types of members. The key relationships are highlighted, such as the composition relationships between libraries and books, and the associations between loans, members, and books. The diagram also captures the state of books (Available, Reserved, Loaned) and loans (Active, Returned, Overdue).

3.4 State Diagram

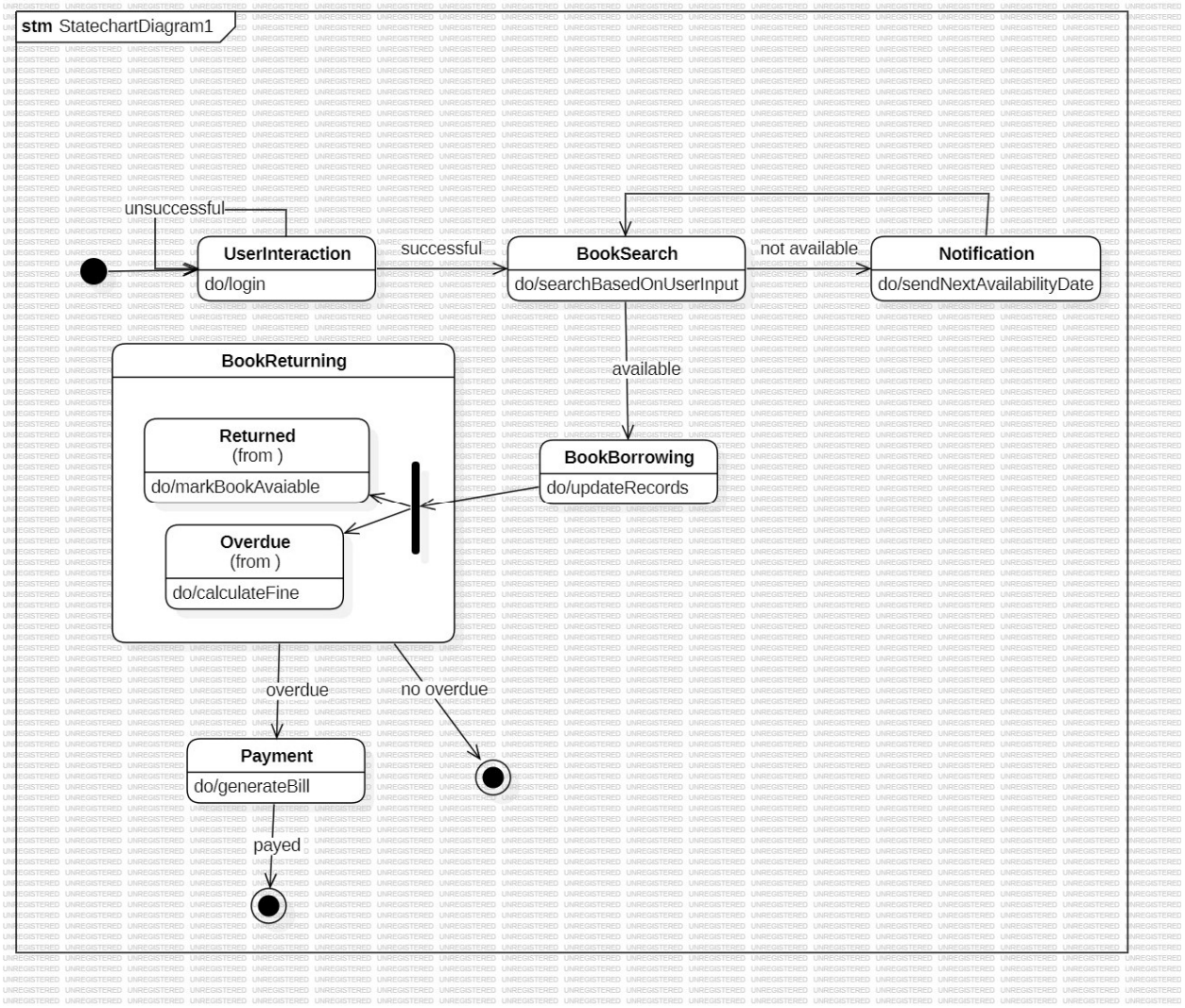


Fig 3.5

Description:

States:

1. **UserInteraction:** do/login - Handles user login.
2. **BookSearch:** do/searchBasedOnUserInput - Searches for books based on user input.
3. **Notification:** do/sendNextAvailabilityDate - Notifies the user about the next availability date if the book is not available.
4. **BookReturning:** This is a composite state with substates for handling returned and overdue books.
    - **Returned:** do/markBookAvailable - Marks the book as available after it's returned.

- **Overdue:** do/calculateFine - Calculates the fine for an overdue book.

5. **BookBorrowing:** do/updateRecords - Updates the records after a book is borrowed.
6. **Payment:** do/generateBill - Generates a bill for overdue books.

Brief Description: This state diagram models user interactions with a library system. It starts with user login. If the login is successful, the user can search for books. If a book is available, the user can borrow it. If not, they are notified about the next availability date. The diagram also handles book returns, including the calculation of fines for overdue books. The composite state BookReturning manages different scenarios based on the book's return status. The diagram shows a clear flow of actions from login to book search, borrowing, returning, and payment (if applicable).

3.5 Use case diagram

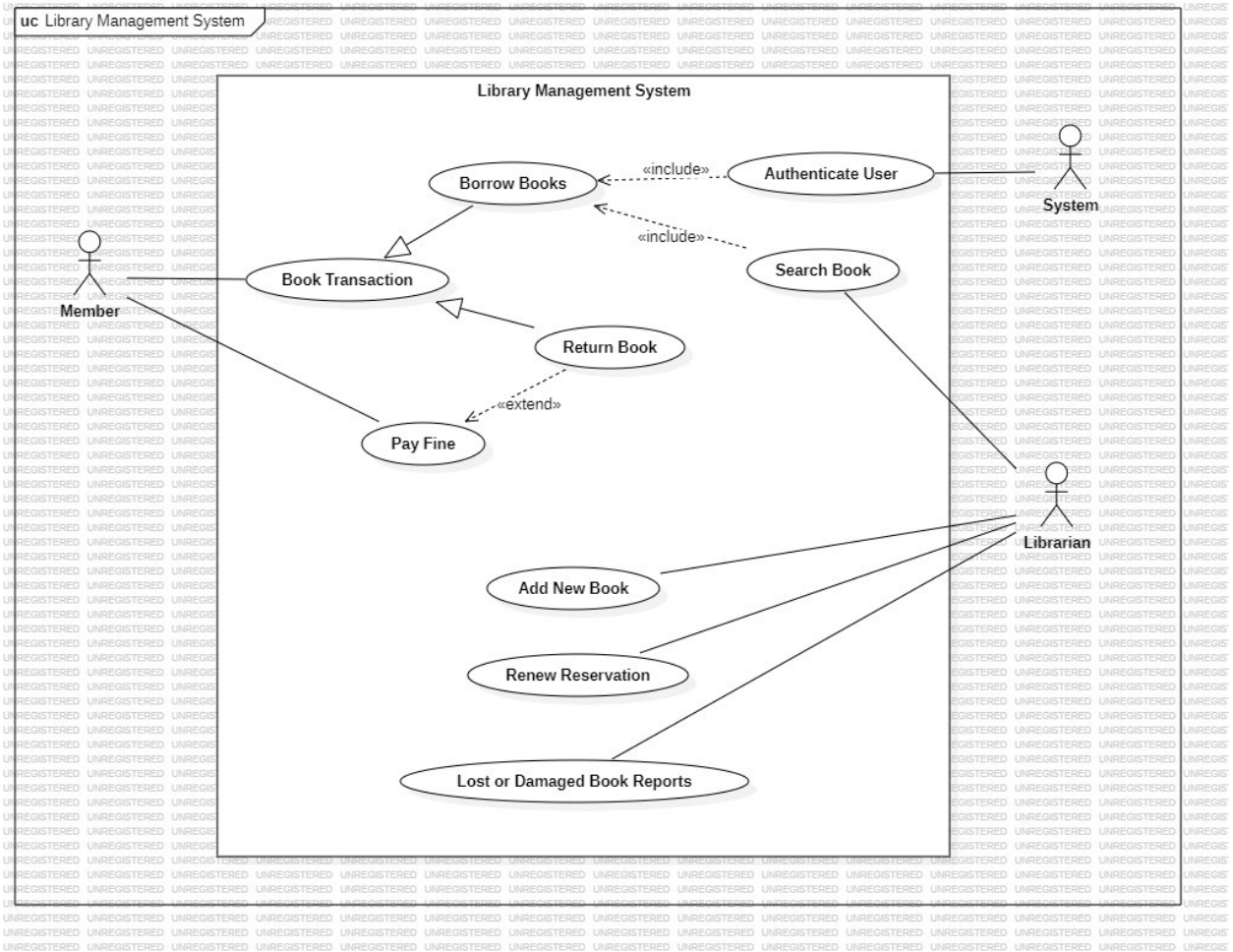


Fig 3.6

Description:

- **Use Cases (Minimum 5):**

1. **Authenticate User:** Verifies the identity of a member or librarian.
2. **Borrow Books:** Allows a member to borrow books.
3. **Search Book:** Enables searching for books in the library catalog.
4. **Return Book:** Allows a member to return borrowed books.
5. **Pay Fine:** Allows a member to pay fines for overdue books.
6. **Add New Book:** Enables the librarian to add new books to the catalog.
7. **Renew Reservation:** Allows members to renew their book reservations.
8. **Lost or Damaged Book Reports:** Allows members to report lost or damaged books.

- **Actors:**

- **Member:** A registered user of the library.
- **Librarian:** A staff member managing the library.
- **System:** Represents the library's automated system

Explanation:

- A Member can borrow books (which requires authentication and book searching), return books (which also requires authentication and can optionally involve paying a fine), and potentially renew reservations or report lost/damaged books.
- A Librarian can add new books to the system, renew reservations, and handle lost or damaged book reports.
- The System is involved in user authentication and book searching, supporting the "Borrow Books" and "Return Book" use cases.

3.6 Sequence Diagram

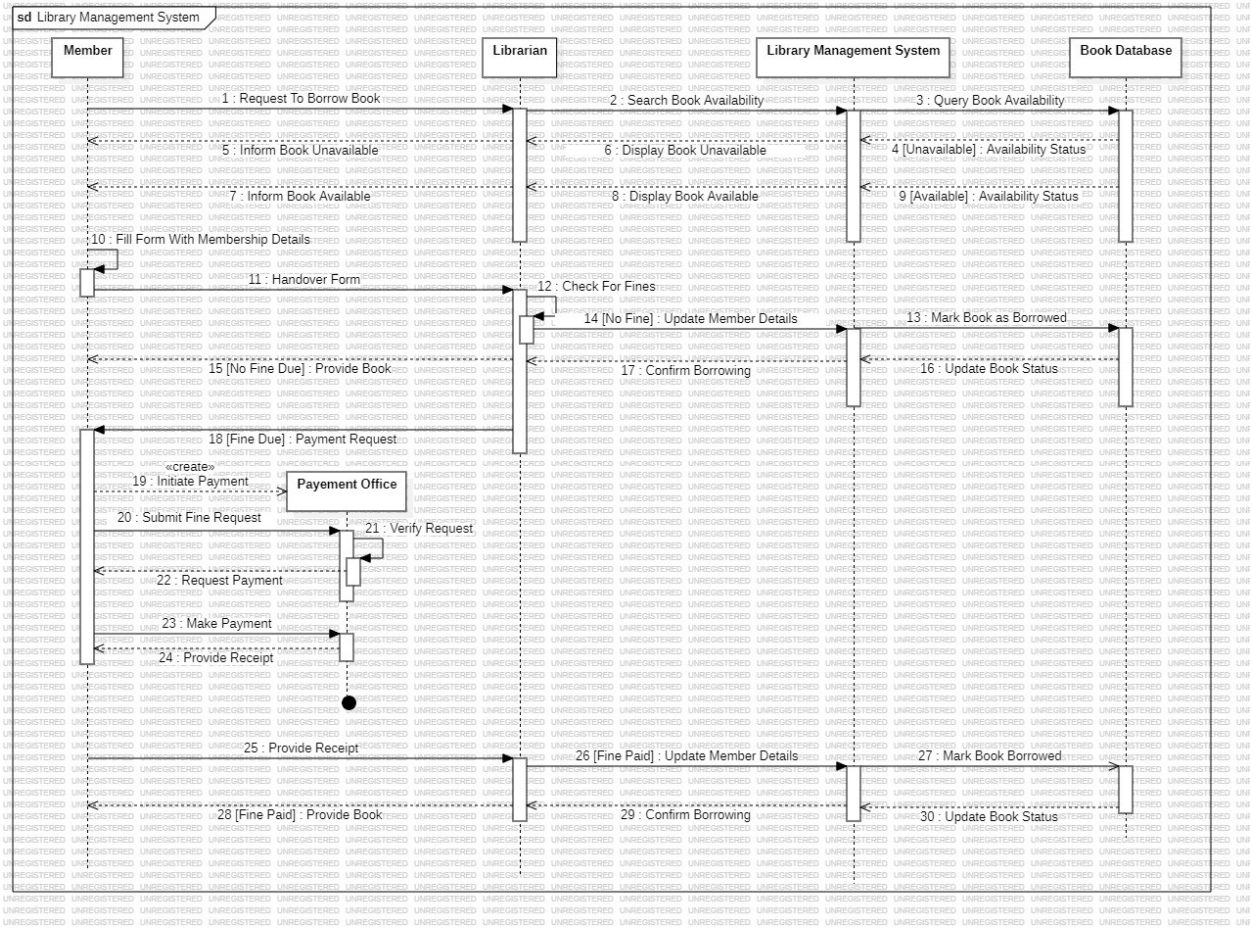


Fig 3.7

Description:

1. Use Case: Borrow Books

Scenario: A library member, Alice, wants to borrow a book titled "The Secret Garden." She approaches the librarian, Bob, and requests the book. Bob checks the library system to confirm the book's availability. The system confirms that the book is available. Bob then checks Alice's membership details and confirms that she has no outstanding fines. Bob then lends the book to Alice, and the system updates the book's status to "Loaned" and records the loan in Alice's borrowing history.

2. Use Case: Return Book

Scenario: A library member, Alice, returns a book titled "The Secret Garden" to the librarian, Bob. Bob receives the book and checks it for any damage. He then uses the library system to record the book's return. The system updates the book's status to "Available" and removes the loan record

from Alice's borrowing history. If the book is overdue, the system calculates the fine, which Alice then pays.

Brief Description of the Sequence Diagram:

The sequence diagram depicts the process of a member borrowing a book from the library, handling both cases where fines are due and where they are not. The Member interacts with the Librarian, who uses the Library Management System to check book availability in the Book Database. If the book is unavailable, the Member is informed. If available, the Librarian checks for any outstanding fines for the member.

3.7 Activity Diagram

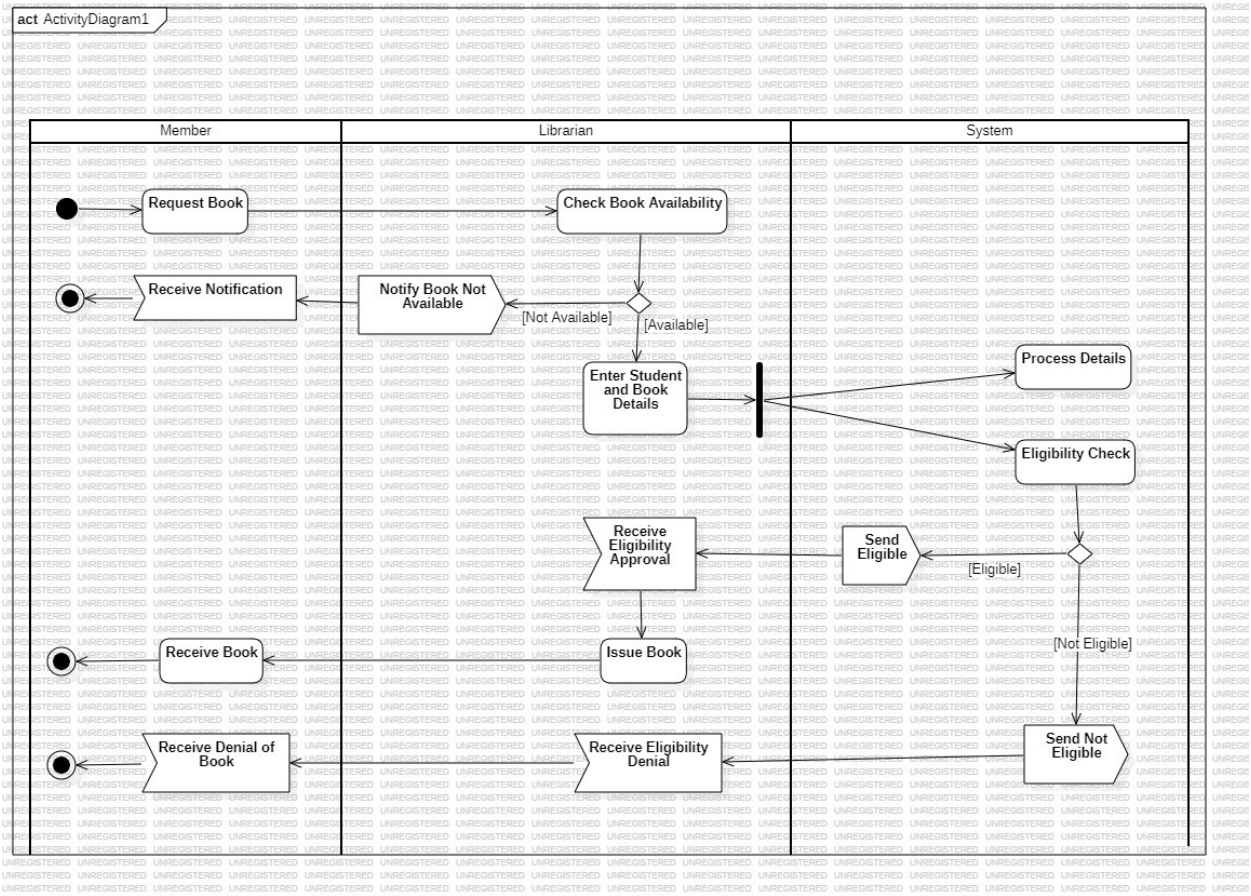


Fig 3.8

Description:

1. Swimlanes with responsible persons:

- **Member:** Responsible for requesting the book, receiving notifications (availability or denial), and receiving the book (if available).
- **Librarian:** Responsible for checking book availability, notifying the member if the book is not available, entering student and book details, receiving eligibility approval/denial, and issuing the book (if eligible).
- **System:** Responsible for processing details, performing the eligibility check, and sending eligibility approval/denial.

2. Brief explanation:

This activity diagram depicts the process of a member requesting a book from a library. The process begins with the Member making a "Request Book." The Librarian then checks the "Book Availability" in the system. There's a decision point:

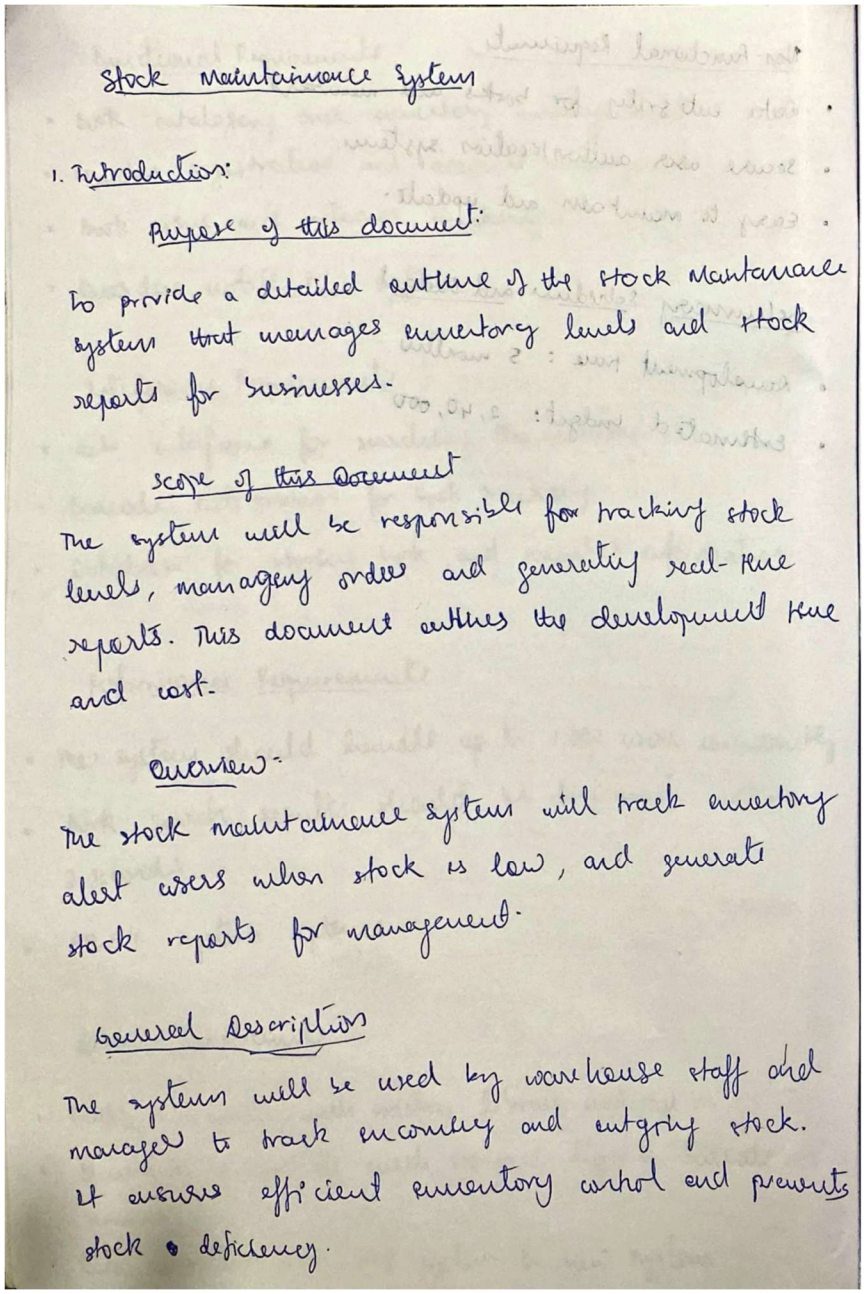
- **Not Available:** If the book is not available, the Librarian sends a "Notify Book Not Available" notification to the Member, and the process ends for this request.
- **Available:** If the book is available, the Librarian "Enter Student and Book Details" into the system. The System then performs an "Eligibility Check." Another decision point follows:
    - **Eligible:** If the member is eligible to borrow the book, the System sends an "Send Eligible" message to the Librarian, who then "Issue Book" to the Member. The Member then "Receive Book," completing the successful borrowing process.
    - **Not Eligible:** If the member is not eligible, the System sends a "Send Not Eligible" message to the Librarian, who then "Receive Eligibility Denial" and notifies the Member with "Receive Denial of Book," completing the failed borrowing process.

Stock Maintenance System

4.1 Problem Statement

A system to maintain inventory for a business, allowing for realtime stock updates, supplier management, order tracking, and low-stock alerts. The system should also generate detailed reports on stock movements and provide analytics for inventory optimization.

4.2 Software Requirements Specification (SRS) Document



Stock maintainance system

1. Introduction:

Purpose of this document:

To provide a detailed outline of the stock mantainance system that manages inventory levels and stock reports for businesses.

Scope of this Document

The system will be responsible for tracking stock levels, managing order and generating real-time reports. This document outlines the development time and cost.

Overview:

The stock maintainance system will track inventory alert users when stock is low, and generate stock reports for management.

General Description

The system will be used by warehouse staff and managers to track incoming and outgoing stock. It ensures efficient inventory control and prevents stock deficiency.

Fig 4.1

Functional Requirements

- stock level monitoring
- Automatic low-stock alerts
- stock report generation.
- Supplier and order management.

Interface Requirements:

- Web based interface for stock management
- Integration with point-of-scale systems
- API for supplier communication

Performance Requirements:

- Handle upto 10,000 stock items.
- Generate reports within 5 seconds
- Less than 1% system downtime.

Design Constraints:

- Must support multiple warehouse locations.
- Ensure compatibility with existing point-of-sale software
- Restriction to standard inventory formats.

Fig 4.2

Non- Functional Attributes

- Reliable with 99.5% uptime
- Data security for stock levels
- Scalable for large inventory

Preliminary Schedule and Budget:

- Development time: 4 months
- estimated budget: ₹3, 20, 000

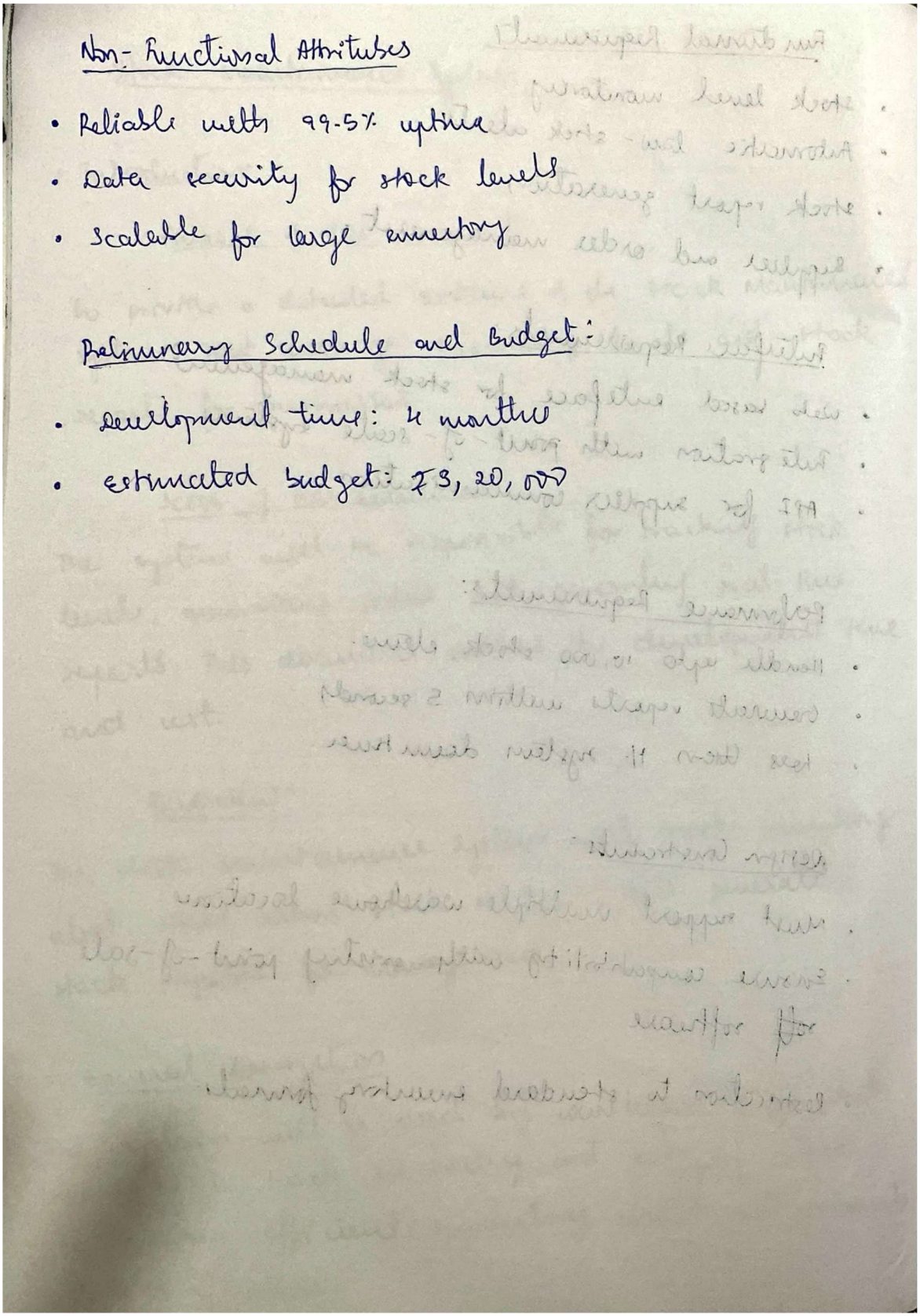


Fig 4.3

4.3 Class Diagram

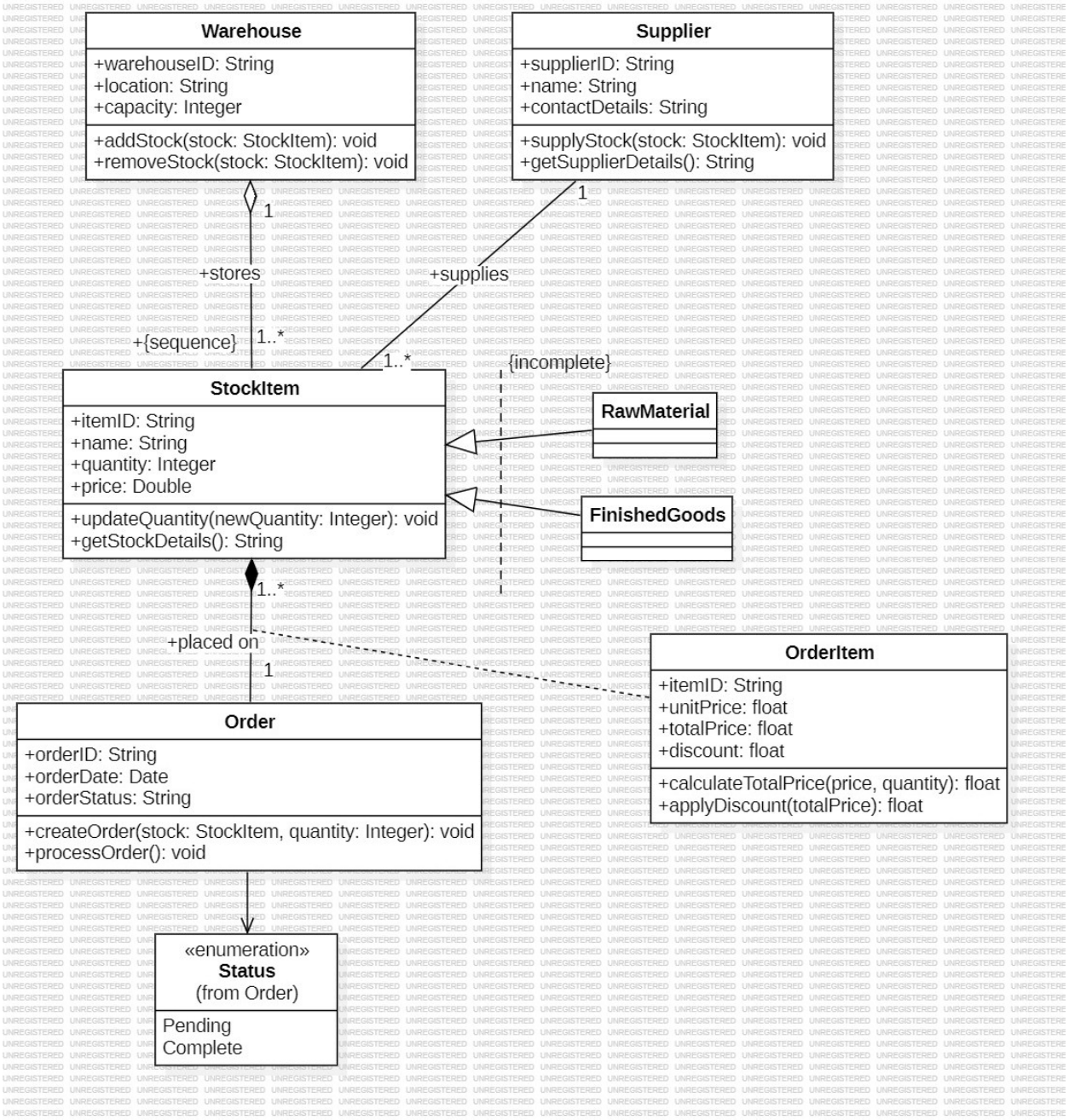


Fig 4.4

Description:

Classes:

1. **Warehouse:**
    - Attributes: warehouseID (String), location (String), capacity (Integer)
    - Operations: addStock(stock: Stockitem), removeStock(stock: Stockitem)
2. **Supplier:**

- Attributes: supplierID (String), name (String), contactDetails (String)
- Operations: supplyStock(stock: Stockitem), getSupplierDetails()

3. **Stockitem:**
    - Attributes: itemID (String), name (String), quantity (Integer), price (Double)
    - Operations: updateQuantity(newQuantity: Integer), getStockDetails()
4. **RawMaterial:** (Specialization of Stockitem)
5. **FinishedGoods:** (Specialization of Stockitem)
6. **Order:**
    - Attributes: orderID (String), orderDate (Date), orderStatus (String) (Status enum: Pending, Complete)
    - Operations: createOrder(stock: Stockitem, quantity: Integer), processOrder()
7. **OrderItem:**
    - Attributes: itemID (String), unitPrice (float), totalPrice (float), discount (float)
    - Operations: calculateTotalPrice(price, quantity), applyDiscount(totalPrice)

Brief Description:

This class diagram models an inventory and order management system. It defines the key entities: warehouses, suppliers, stock items (including raw materials and finished goods), orders, and order items. It shows how warehouses store stock items, how suppliers provide stock, and how orders are composed of order items. The diagram uses generalization to represent different types of stock items and an enumeration to define order statuses. The ordering of stock items within a warehouse is also specified.

4.4 State Diagram

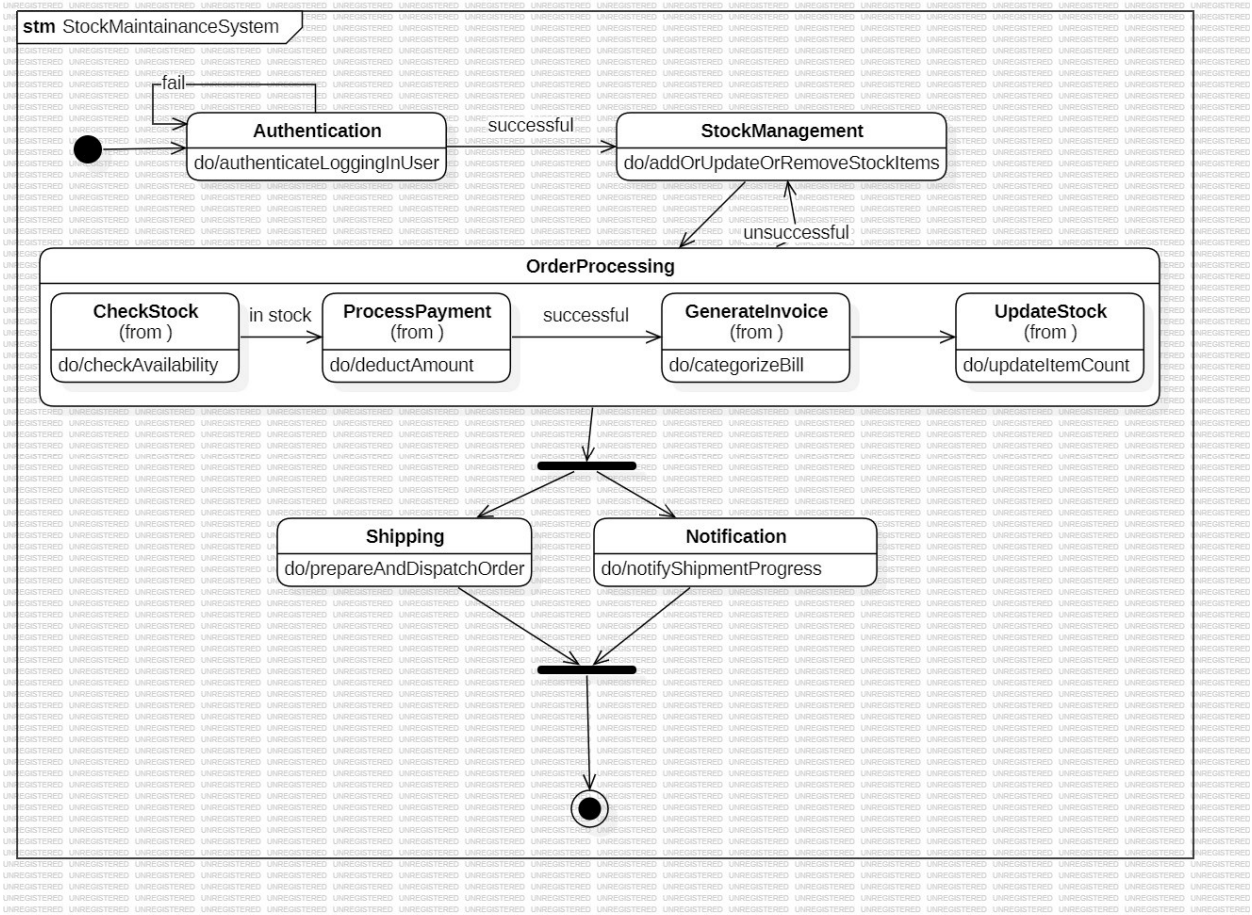


Fig 4.5

Description:

States:

1. **Authentication:** do/authenticateLoggingInUser - Handles user login.
2. **StockManagement:** do/addOrUpdateOrRemoveStockitems - Manages stock items (adding, updating, or removing).
3. **OrderProcessing:** This is a composite state containing substates for order processing activities.
    - **CheckStock:** do/checkAvailability - Checks the availability of stock.
    - **ProcessPayment:** do/deductAmount - Processes payment for the order.
    - **GenerateInvoice:** do/categorizeBill - Generates the invoice.
    - **UpdateStock:** do/updateItemCount - Updates the stock count after the order.
4. **Shipping:** do/prepareAndDispatchOrder - Prepares and dispatches the order.

5. **Notification:** do/notifyShipmentProgress - Notifies the customer about shipment progress

Brief Description:

This state diagram models the stock maintenance process, including user authentication, stock management, order processing, shipping, and customer notification. The process begins with user authentication. Upon successful login, the user can either manage stock items or process orders. The order processing involves checking stock availability, processing payment, generating an invoice, and updating stock. After updating the stock, the shipping and notification processes occur concurrently. The use of a composite state for OrderProcessing and the fork/join pseudo-states for shipping and notification effectively demonstrate the concurrent nature of these activities.

4.5 Use case diagram

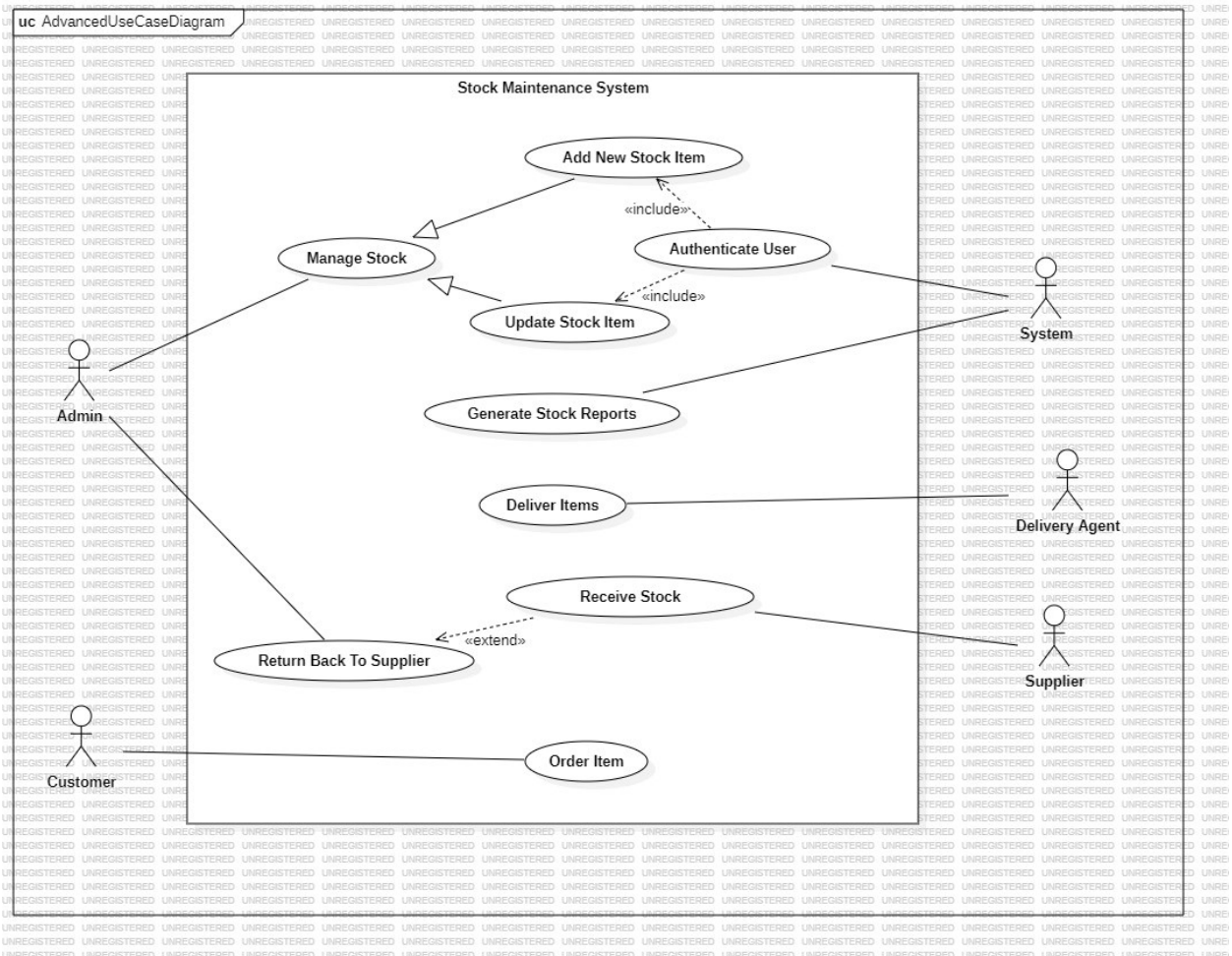


Fig 4.6

Description:

- **Use Cases:**
  1. **Authenticate User:** Verifies the identity of an Admin.
  2. **Add New Stock Item:** Allows an Admin to add new stock items to the system.
  3. **Update Stock Item:** Allows an Admin to update existing stock items.
  4. **Generate Stock Reports:** Allows an Admin to generate reports about the stock.
  5. **Deliver Items:** Allows a Delivery Agent to deliver items to customers.
  6. **Receive Stock:** Allows a Supplier to deliver stock to the warehouse.
  7. **Return Back To Supplier:** Allows the system to return items back to a Supplier.
  8. **Order Item:** Allows a Customer to order an item.
- **Actors:**
  - **Admin:** Manages the stock within the system.
  - **System:** The automated stock maintenance system.
  - **Delivery Agent:** Delivers items to customers.
  - **Supplier:** Supplies stock to the system.
  - **Customer:** Orders items from the system.

Explanation:

- An Admin can manage stock (which includes authentication, adding new items, and updating existing ones) and generate stock reports.
- A Delivery Agent is responsible for delivering items.
- A Supplier is responsible for delivering (receiving stock) to the system, and in some cases, the system will return items back to the supplier.
- A Customer can order items.
- The System is involved in user authentication and stock management, supporting the "Manage Stock" use case.

4.6 Sequence Diagram

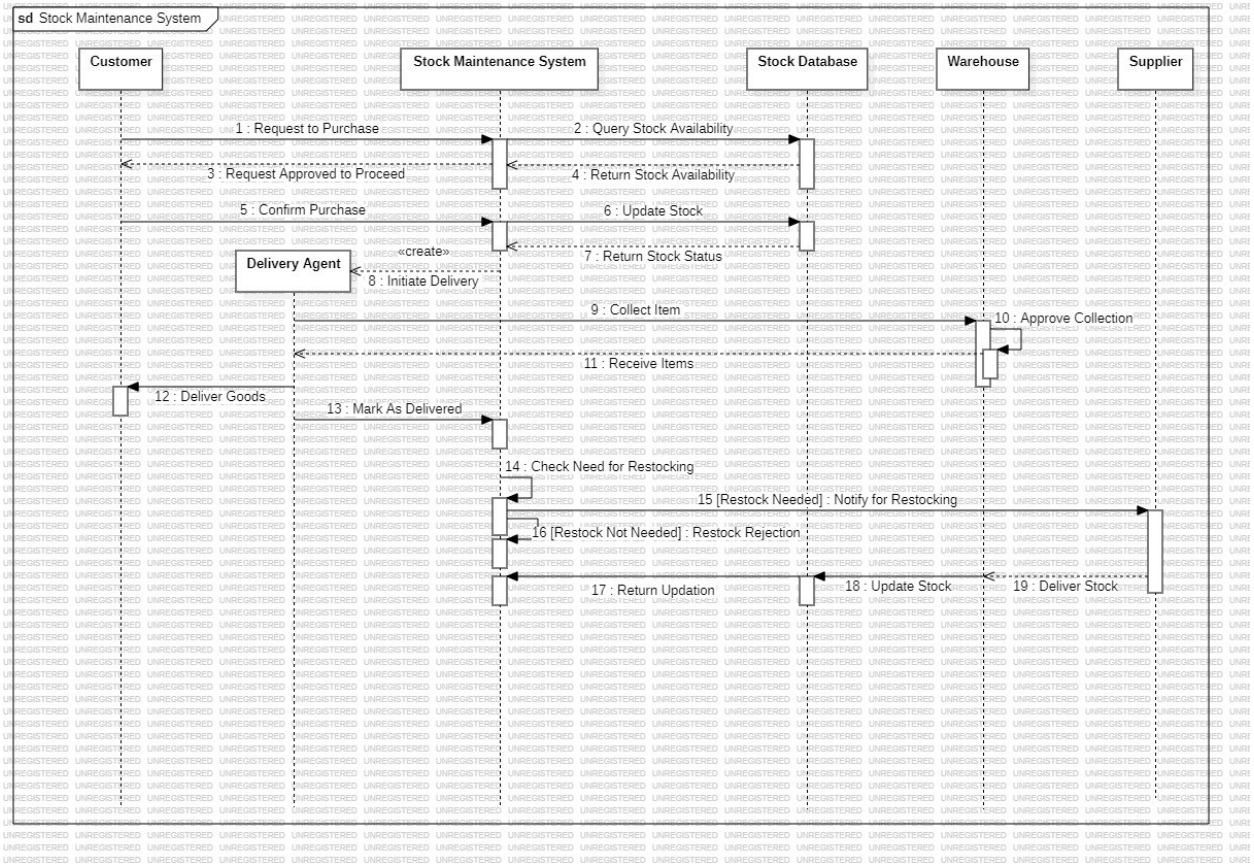


Fig 4.7

Description:

1. Use Case: Order Item (Customer perspective leading to delivery)

Scenario: A customer wants to purchase an item. They make a purchase request through the system. The system checks stock availability. If the stock is available, the system confirms the purchase and creates a delivery request. A delivery agent is notified, collects the item from the warehouse, delivers it to the customer, and marks the delivery as complete in the system.

2. Use Case: Receive Stock (Supplier perspective leading to warehouse update)

Scenario: After items have been sold, the system checks if restocking is needed. If the stock level is low, the system notifies the supplier to deliver more stock. The supplier delivers the stock to the warehouse. The warehouse approves the collection of the stock and notifies the system, which then updates the stock levels in the database.

Brief Explanation:

The diagram shows the interactions required for a customer to order an item and the subsequent delivery process, as well as the process of restocking from the supplier. It uses conditional logic (the branches after message 14) to depict the different paths based on the need for restocking. The Stock Database and Warehouse are passive objects. The Delivery Agent is a transient object, created for the delivery process and presumably destroyed after the delivery is complete.

4.7 Activity Diagram

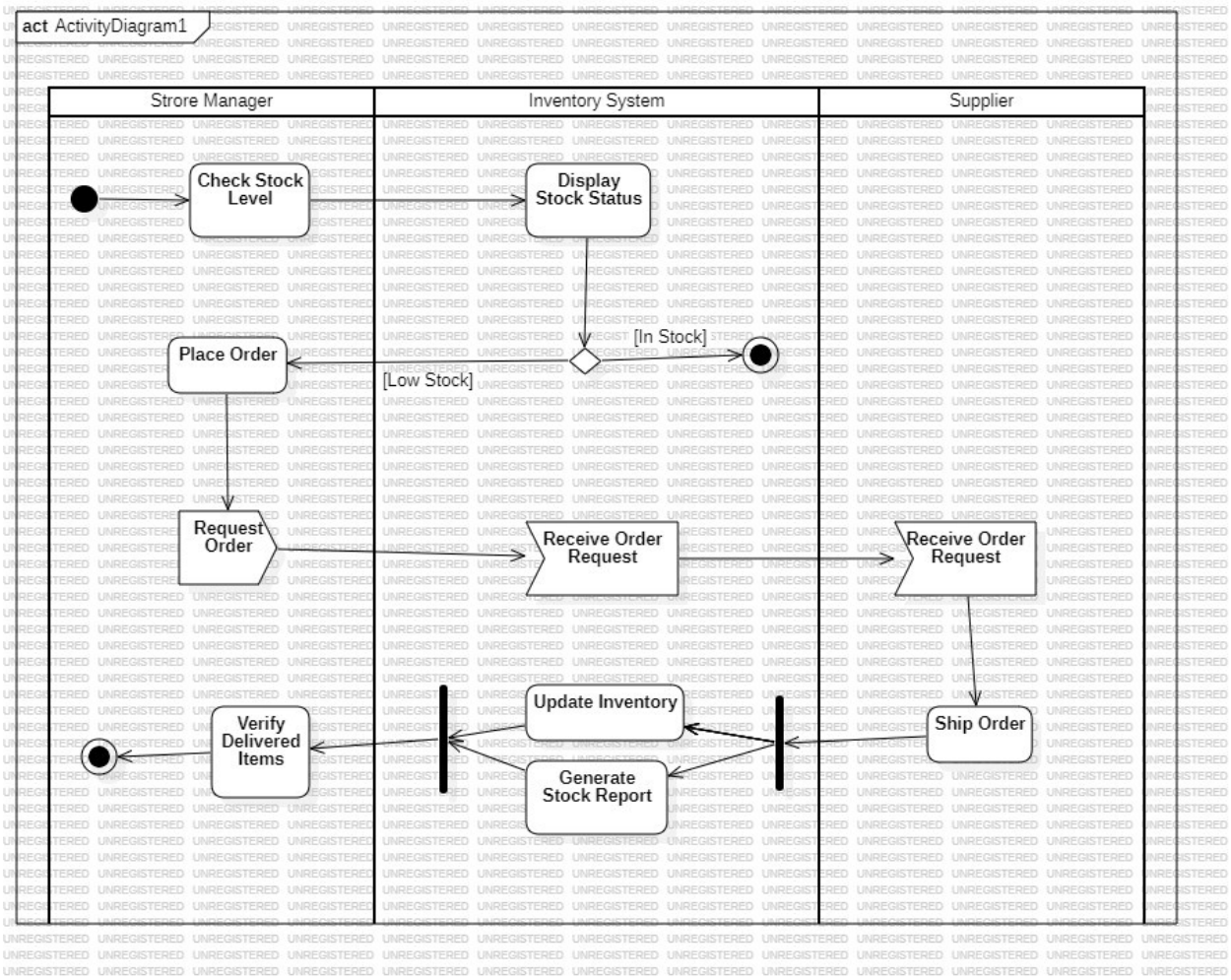


Fig 4.8

Description:

1. Swimlanes with responsible persons:

- **Store Manager:** Responsible for checking stock level, placing orders, requesting orders, and verifying delivered items.
- **Inventory System:** Responsible for displaying stock status, receiving order requests, updating inventory, and generating stock reports.
- **Supplier:** Responsible for receiving order requests and shipping orders.

2. Brief explanation:

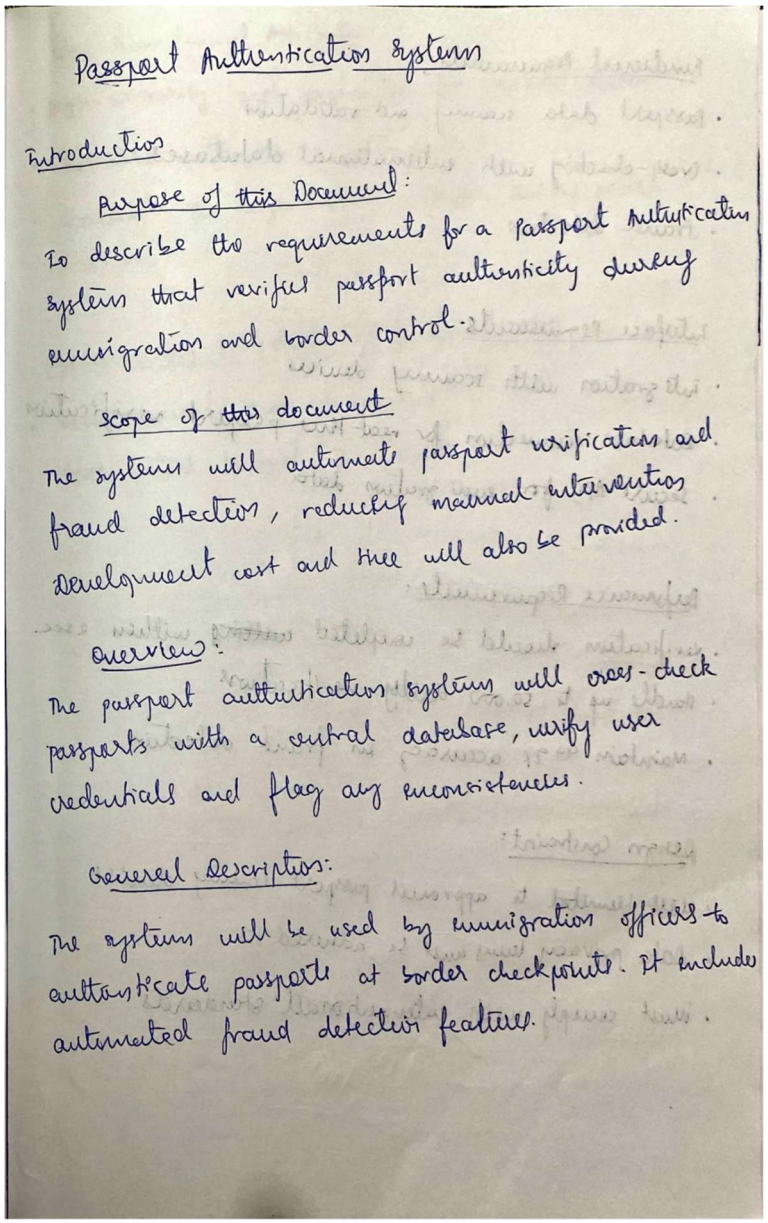
This activity diagram describes the process of managing inventory and ordering new stock. The Store Manager begins by checking the Stock Level. The Inventory System then displays the Stock Status. If the stock is "In Stock," nothing further happens. If the stock is "Low Stock," the Store Manager "Place Order." The Inventory System then "Receive Order Request", and this information is sent to the Supplier in order to "Receive Order Request". The Supplier then "Ship Order" to the Store manager who "Verify Delivered Items". The Inventory System then "Update Inventory" and "Generate Stock Report" concurrently. The process ends after verification and inventory update/report generation. The diagram uses a decision point to determine if an order needs to be placed and a fork/join to show concurrent processes.

Passport Automation System

5.1 Problem Statement

A system to automate the passport application and processing workflow, including applicant registration, document verification, appointment scheduling, status tracking, and delivery of issued passports. The system should ensure secure data handling and integrate with national security databases for background checks.

5.2 Software Requirements Specification(SRS) Document



Passport Authentication System

Introduction

Purpose of this Document:

To describe the requirements for a passport authentication systems that verifies passport authenticity during immigration and border control.

Scope of this document

The systems will automate passport verification and fraud detection, reducing manual intervention Development cost and time will also be provided.

Overview:

The passport authentication systems will cross-check passports with a central database, verify user credentials and flag any inconsistencies.

General Descriptions:

The systems will be used by immigration officers to authenticate passports at border checkpoints. It include automated fraud detection features.

Fig 5.1

Functional Requirements:

- Passport data scanning and validation
- Cross-checking with international databases
- Fraud detection

Interface Requirements

- Integration with scanning devices
- Database connection for real-time passport verification
- Secure API for immigration data

Performance Requirements:

- Verification should be completed ~~within~~ within 2 sec
- Handle up to 50,000 daily verifications
- Maintain 99.9% accuracy in fraud detection

Design Constraint:

- ~~Must~~ Limited to approved passport scanning device
- Data privacy laws must be adhered to
- Must comply with international standards

Fig 5.2

Non-Functional Attributes

- High security and encryption for passport data.
- Reliable with 99.9% uptime
- Scalable for high-volume traffic during peack seasons.

Preliminary Schedule and Budget.

- Development time: 6 months
- Estimated budget: ₹5,60,000

Fig 5.3

5.3 Class Diagram

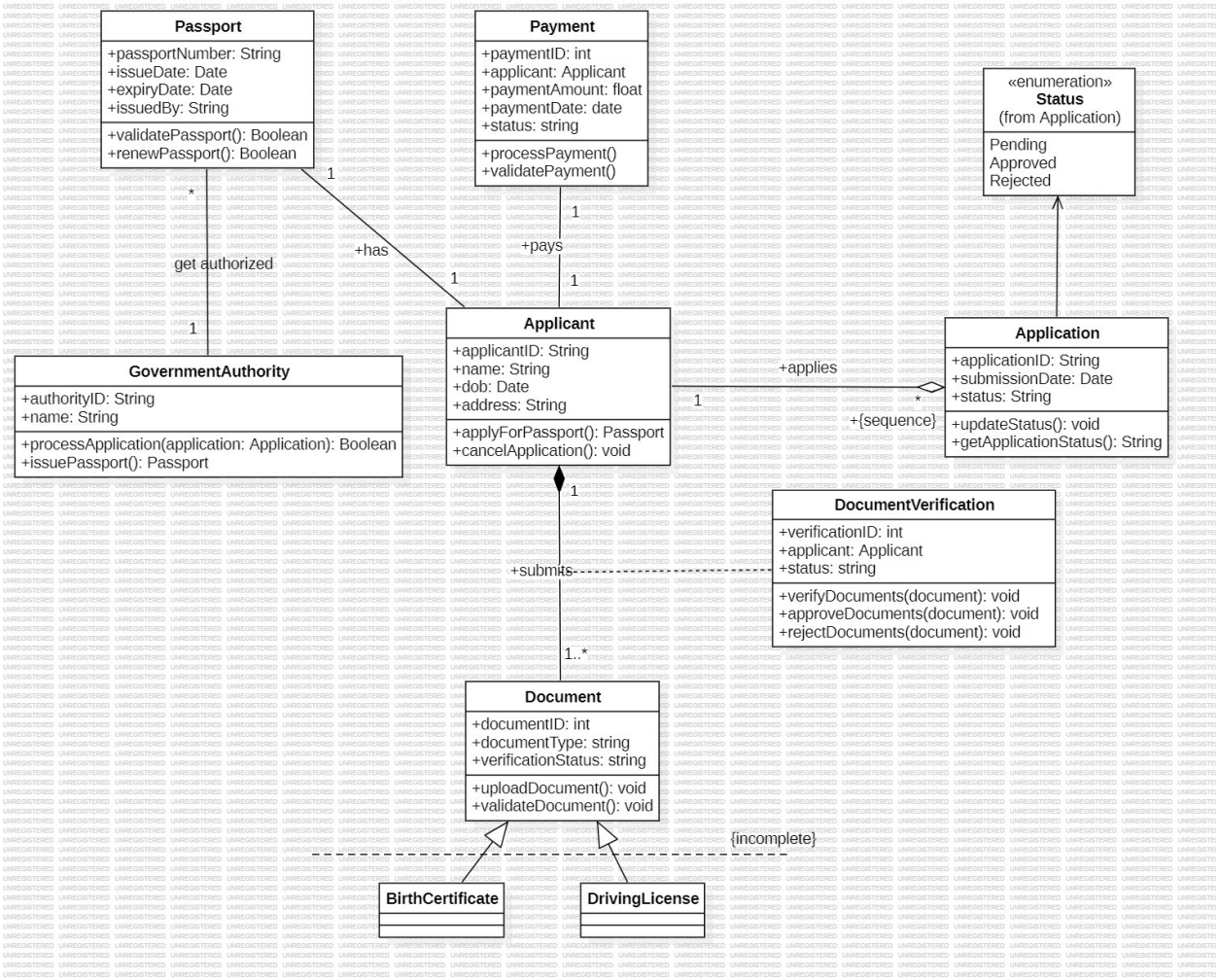


Fig 5.4

Description:

Classes:

1. **Passport:**
    - Attributes: passportNumber (String), issueDate (Date), expiryDate (Date), issuedBy (String) (Likely "Regional Passport Office, Bengaluru")
    - Operations: validatePassport(), renewPassport()
2. **Payment:**
    - Attributes: paymentID (int), applicant (Applicant), paymentAmount (float), paymentDate (date), status (string)
    - Operations: processPayment(), validatePayment()

3. **Applicant:**
    - Attributes: applicantID (String), name (String), dob (Date), address (String) (Should include state, likely Karnataka, and potentially PIN code for Bengaluru)
    - Operations: applyForPassport(), cancelApplication()
4. **Application:**
    - Attributes: applicationID (String), submissionDate (Date), status (String) (Status enum: Pending, Approved, Rejected)
    - Operations: updateStatus(), getApplicationStatus()
5. **GovernmentAuthority:**
    - Attributes: authorityID (String), name (String) (Likely "Passport Seva Kendra, Bengaluru" or similar)
    - Operations: processApplication(), issuePassport()
6. **DocumentVerification:**
    - Attributes: verificationID (int), applicant (Applicant), status (string)
    - Operations: verifyDocuments(), approveDocuments(), rejectDocuments()
7. **Document:**
    - Attributes: documentID (int), documentType (string), verificationStatus (string)
    - Operations: uploadDocument(), validateDocument()
8. **BirthCertificate:** (Specialization of Document)
9. **DrivingLicense:** (Specialization of Document)

Brief Description:

This class diagram models a passport application system in Bengaluru, Karnataka. It represents the key entities involved in the process: applicants, their applications, passports, payments, the government authority (Passport Seva Kendra), document verification, and various supporting documents. The relationships between these entities are clearly defined, showing how applicants apply for passports, how applications are processed and authorized, how payments are made, and how documents are verified. The diagram also captures the different statuses of applications and the types of documents that can be submitted. The context of Bengaluru is reflected in the potential values of attributes like issuedBy (for Passport) and name (for GovernmentAuthority). The ordering of Applications is also captured.

5.4 State Diagram

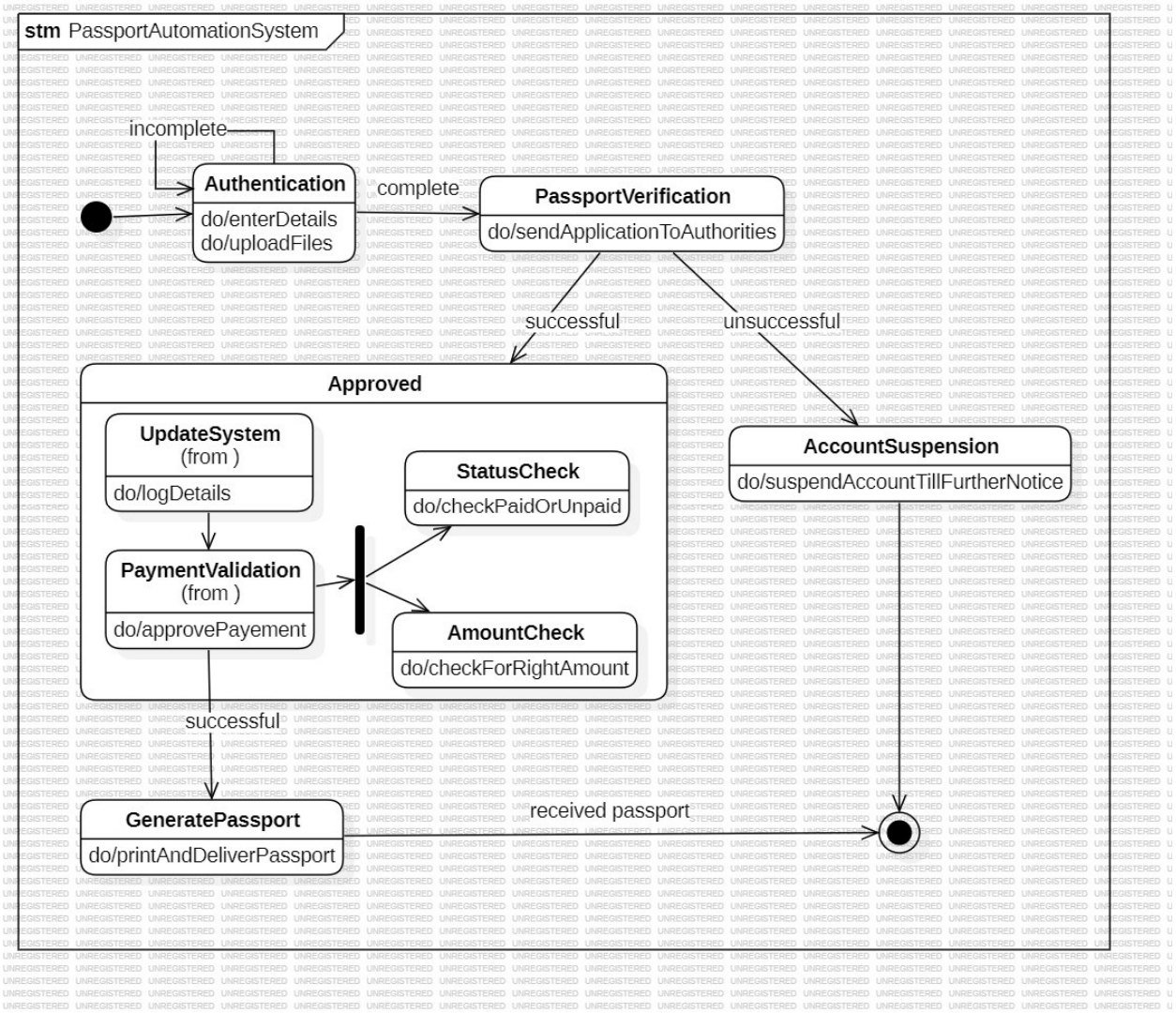


Fig 5.5

Description:

States (Minimum 6):

1. **Authentication:** do/enterDetails, do/uploadFiles - The applicant enters personal details and uploads required documents.
2. **PassportVerification:** do/sendApplicationToAuthorities - The application is sent to the relevant authorities (Passport Seva Kendra in Bengaluru).
3. **Approved:** This is a composite state handling the post-approval process.
    - **UpdateSystem:** do/logDetails - System logs the application details.
    - **PaymentValidation:** do/approvePayment - Payment is validated.

- **StatusCheck:** do/checkPaidOrUnpaid - Checks if payment has been made.
- **AmountCheck:** do/checkForRightAmount - Checks if correct amount has been paid.

4. **AccountSuspension:** do/suspendAccountTillFurtherNotice - The applicant's account is suspended.
5. **GeneratePassport:** do/printAndDeliverPassport - The passport is printed and delivered.

Brief Description:

This state diagram models the process of passport automation in Bengaluru. It begins with authentication and document upload. The application is then sent for verification to the relevant authorities. If approved, the system updates the details, validates the payment, checks the payment status and amount, and then generates and delivers the passport. If rejected, the applicant's account is suspended. The composite state Approved manages the post-approval steps, including concurrent activities. The diagram provides a clear view of the application lifecycle, from initial authentication to final passport delivery or account suspension.

5.5 Use case diagram

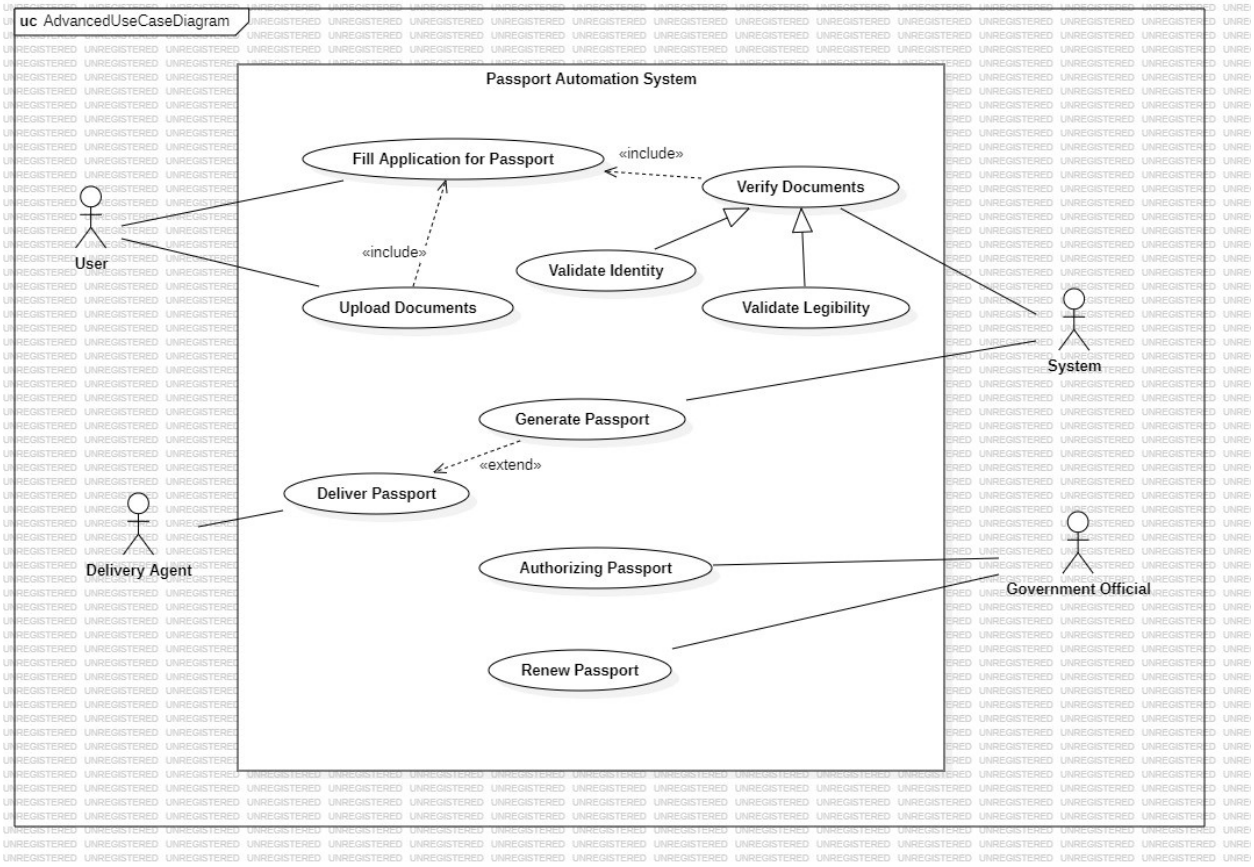


Fig 5.6

Description:

- **Use Cases (Minimum 5):**

1. **Fill Application for Passport:** The user fills out the online passport application form.
2. **Upload Documents:** The user uploads the required documents (e.g., birth certificate, address proof).
3. **Verify Documents:** The system verifies the uploaded documents.
4. **Validate Identity:** The system validates the user's identity.
5. **Validate Legibility:** The system checks the legibility of the uploaded documents.
6. **Generate Passport:** The system generates the physical passport.
7. **Deliver Passport:** The passport is delivered to the user.
8. **Authorizing Passport:** A Government Official authorizes the passport.
9. **Renew Passport:** The user renews their existing passport.

 Actors:

- **User:** The applicant applying for or renewing a passport.
- **System:** The automated passport application system.
- **Delivery Agent:** The person responsible for delivering the passport.
- **Government Official:** The person responsible for authorizing the passport.

Explanation:

- A User fills out the application, uploads documents, and may also renew their passport.
- The System verifies documents (including checking legibility) and validates the user's identity. The System also generates the passport.
- A Delivery Agent delivers the generated passport.
- A Government Official authorizes the passport.

5.6 Sequence Diagram

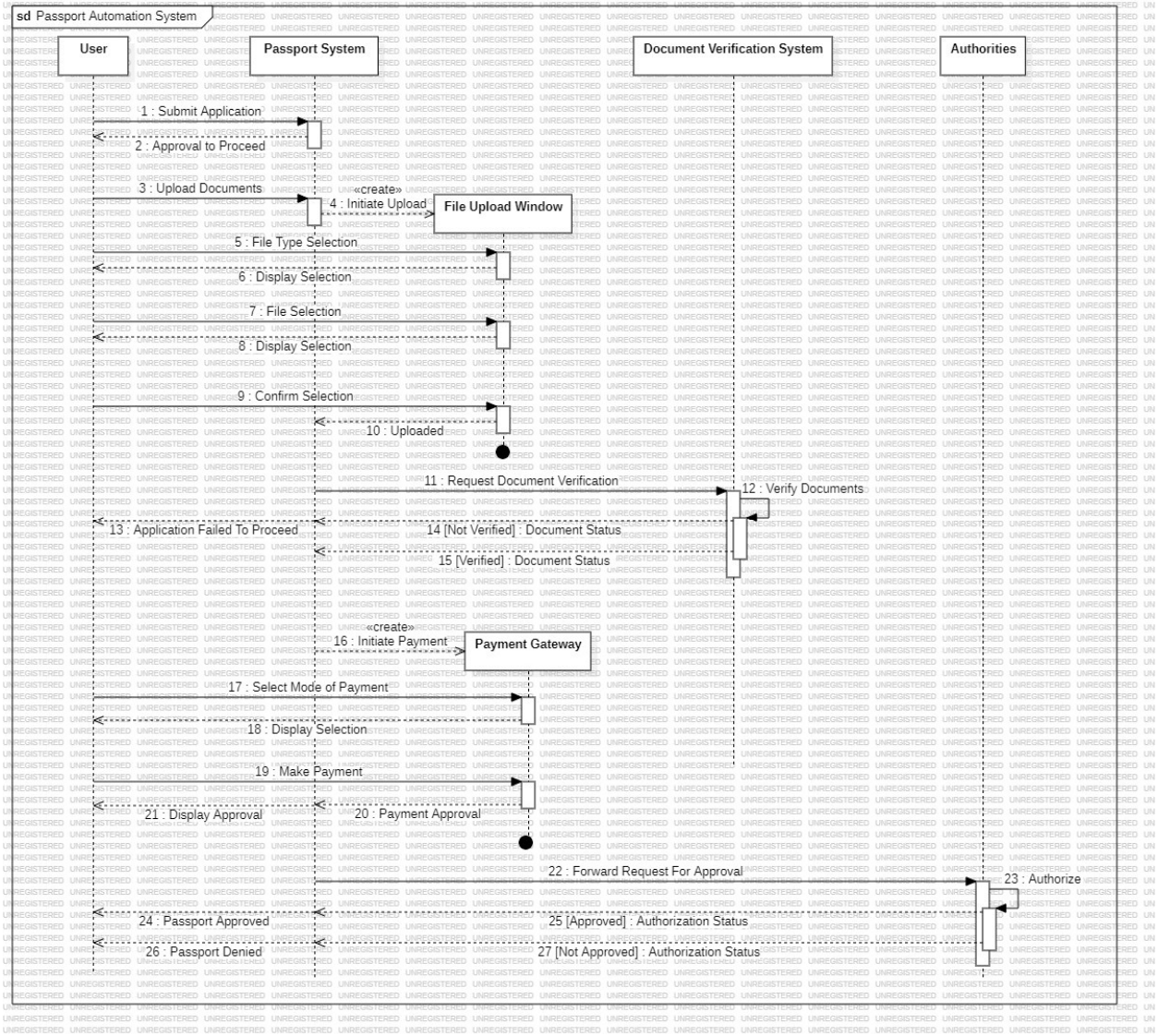


Fig 5.7

Description:

1. Use Case: Apply for Passport (Successful Application)

Scenario: A user in Bengaluru, Karnataka, initiates a passport application online. They enter their personal details and upload the required documents (e.g., birth certificate, address proof) into the system. The system validates the uploaded documents. Upon successful verification, the user is directed to the payment gateway to make the necessary fees. After successfully completing the payment, the application is forwarded to the relevant Passport Seva Kendra in Bengaluru for

authorization. The authorities review the application and approve it. Finally, the system notifies the user that their passport application has been approved and will be processed.

2. Use Case: Apply for Passport (Failed Document Verification)

Scenario: A user in Bengaluru submits a passport application online. They upload the required documents, but the system fails to verify one or more documents due to issues such as incorrect format, illegibility, or missing information. The system displays an error message to the user, informing them that their document verification has failed and indicating the specific issues. The user is then guided to re-upload the affected documents or correct the errors. If the user fails to resolve the issues within a specified timeframe, the application may be rejected.

Brief Description of the Sequence Diagram:

The sequence diagram depicts the online passport application process, focusing on the interactions between the user, the system, and external entities like the Document Verification System, Payment Gateway, and Authorities. It illustrates the flow of events from application submission to document upload, verification, payment processing, and final authorization. The diagram highlights the potential for failed document verification and its impact on the application process.

5.7 Activity Diagram

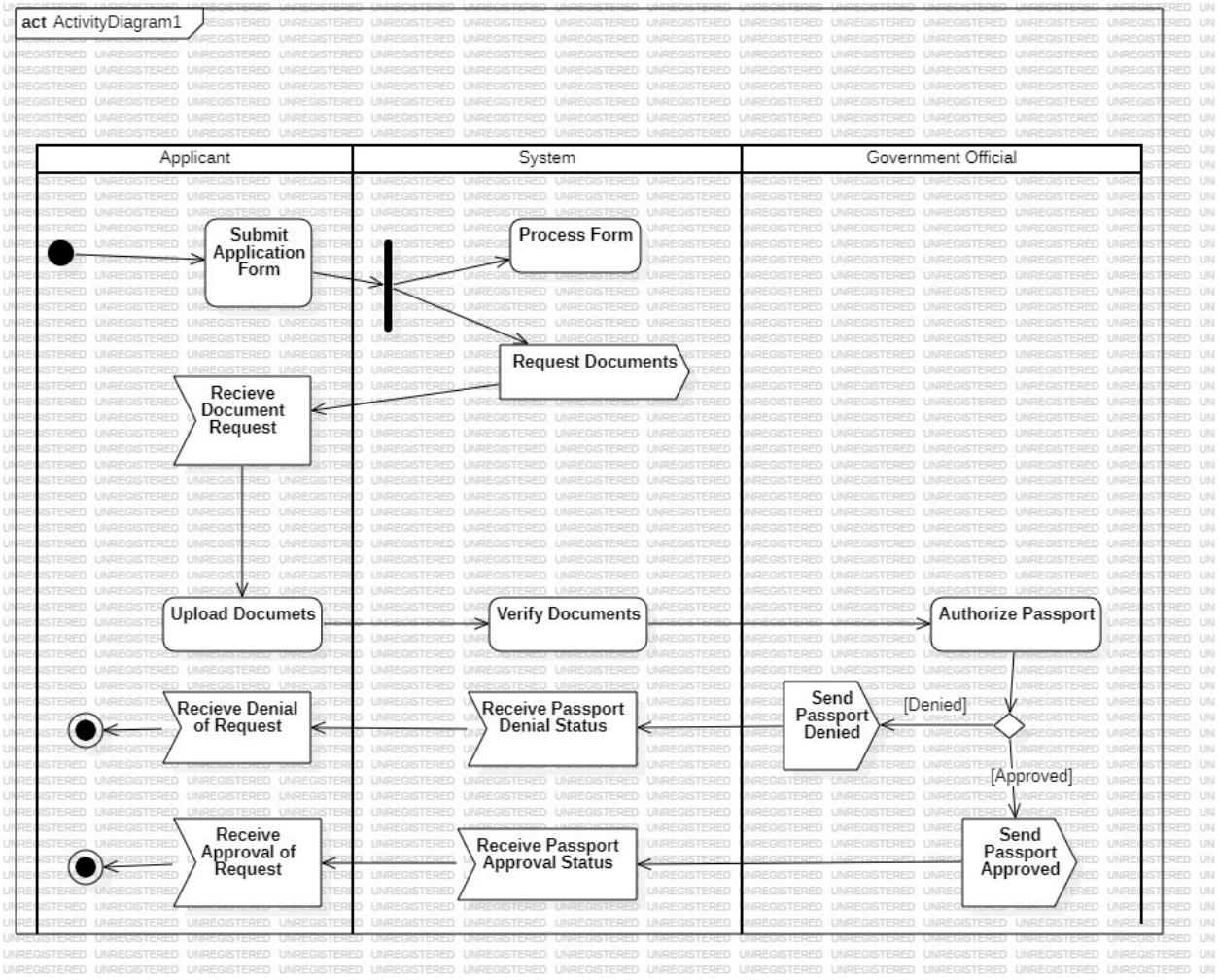


Fig 5.8

Description:

1. Swimlanes with responsible persons:

- **Applicant:** Responsible for submitting the application form, uploading documents, receiving document requests, receiving denial of request, and receiving approval of request.
- **System:** Responsible for processing the form, requesting documents, verifying documents, receiving passport denial status, and receiving passport approval status.

- **Government Official:** Responsible for authorizing the passport, sending passport denied/approved.

2. Brief explanation:

This activity diagram illustrates the passport application process from the Applicant's perspective, involving the System and a Government Official. The Applicant starts by "Submit Application Form." The System then "Process Form" and "Request Documents" from the Applicant. The Applicant receives the document request and "Upload Documents." The System then "Verify Documents." After verification, the application is sent to the Government Official. The Government Official then "Authorize Passport." There's a decision point:

- **Denied:** If the passport is denied, the Government Official sends "Send Passport Denied" to the System, which "Receive Passport Denial Status," and sends "Receive Denial of Request" to the Applicant.
- **Approved:** If the passport is approved, the Government Official sends "Send Passport Approved" to the System, which "Receive Passport Approval Status," and sends "Receive Approval of Request" to the Applicant.

The process ends with the Applicant receiving either a denial or approval notification. The diagram clearly shows the flow of activities and the interaction between the Applicant, the System, and the Government Official, including the decision point for passport authorization.