TASK - 1	Design of problem statement, perform a detailed feasibility study and finalize the process model to be used.
DATE	

Aim: The aim is to design a problem statement, perform a detailed feasibility study and finalize the process model to be used.

PROBLEM STATEMENT:

In an Electronic Cash Counter system, the user enters the secret code being verified with the database of the customers. If a match is found, then the transaction can be performed. If the secret code does not match, the user is requested to re-enter the code and try again. The users must be displayed transaction options such as withdrawal, balance enquiry, mini statement, and pin change. The users can withdraw the amount from their account if the amount is available in their account. The users can also deposit the amount in their account either by cash or cheque. The users can check their balance in their account through balance enquiry and also they can get the mini statement for recent transaction. The pin change if needed by the user can also be done. After the completion of transaction process, the system should display the balance amount to the user and provided the facility to print the receipt of the same.

SCOPE:

The Scope of Electronic Cash Counter system is to allow users to securely withdraw, deposit, check balances, view mini statements, and change PINs. It ensures user-friendly navigation, data security, and compliance with financial regulations.

FEASIBILITY STUDY:

Feasibility Study is used to evaluate a project's potential, including the technical, financial, and economic aspects, and to determine whether it should proceed.

- **Technical Feasibility:** The technical feasibility study always focuses on the existing computer hardware, software and personal. This also includes need for more hardware, software or personal and possibility of procuring or installing such facilities. Electronic Cash Counter is a system that can work on single stand alone Pentium machine with 128 MB RAM, Hard disk drive size of 80 GB, mouse, monitor and keyboard & it also require internet connection to corresponding computer.The equipments are easily available in the market, so technically the system is very much feasible

- **Economical Feasibility:** This feasibility is useful to find the system development cost and checks whether it is justifiable. The cost overheads include software and hardware maintenance cost, training costs that includes cost required for manpower, electricity, stationary etc. The Electronic Cash Counter system will provide the right type of information at right time, and in the required format. This will save time required for decision-making and routine operations. Considering all these advantages, the cost overheads of the system are negligible. So the system is economically feasible.

- **Operational Feasibility :** It is also known as resource feasibility. The operation users of the system are expected to have minimum knowledge of computer. The developed system is simple to use, so that the user will be ready to operate the system. The Electronic Cash Counter system can be developed using JAVA programming language & Mysql database which is platform independent and user friendly. So the system is operationally feasible.

PROCESS MODEL

Agile Model

The Agile process model is preferred for developing an Electronic Cash Counter due to several reasons,

1. **Iterative and Incremental Development:**

Agile promotes an iterative and incremental development approach. This allows for continuous refinement and improvement of the system based on user feedback.

2. **Flexibility to Changing Requirements:**

Agile allows for flexibility in adapting to changes during the development process, ensuring the system aligns with the evolving requirements and market conditions.

3. **Frequent User Involvement:**

Agile emphasizes regular collaboration with end-users. In the context of an Electronic Cash Counter system, involving users throughout the development process ensures that the system meets their needs, enhances usability, and aligns with regulatory requirements**.**

4. **Continuous Integration and Testing:**

Agile encourages continuous integration and testing, allowing developers to identify and address issues early in the development cycle. This helps in ensuring the reliability and security of the software.

5. **Rapid Delivery of Value:**

Agile focuses on delivering a minimum viable product (MVP) quickly. In the case of an Electronic Cash Counter, this means that essential functionalities can be delivered and deployed rapidly, allowing users to start benefiting from the system sooner.

6. **Adaptability to Risks:**

The financial sector comes with inherent risks and uncertainties. Agile provides a framework for adapting to changes and managing risks effectively during the development process, ensuring that the system can respond to unforeseen challenges.

7. **Close Collaboration Between Teams:**

Agile encourages close collaboration among cross-functional teams. For a complex system like an Electronic Cash Counter, this collaboration is essential as it involves integrating hardware, software, security measures, and compliance considerations.

8. **Customer Satisfaction:**

Agile focus on customer satisfaction aligns with the goal of delivering a reliable and user-friendly Electronic Cash Counter system. Regular feedback and involvement of end-users contribute to a product that better meets their expectations.

The classification of UML (Unified Modeling Language) diagrams into Behavior Diagrams and Structure Diagrams.

Behavior Diagrams
These diagrams focus on the dynamic aspects of a system, describing how objects interact and change over time.
Activity Diagram: Represents workflows and processes within a system.

State Machine Diagram: Describes the different states an object goes through in response to events.

Use Case Diagram: Shows system functionality from an external user's perspective.

Interaction Diagram: A subset of behavior diagrams that focuses on object interactions. It includes:

Communication Diagram: Emphasizes message flow between objects.

Sequence Diagram: Shows the chronological order of message exchanges.

Interaction Overview Diagram: Provides a high-level view of interactions within a system.

Timing Diagram: Captures time-based behavior of objects in a system.

Structure Diagrams
These diagrams define the static aspects of a system, including relationships between different components.

- Object Diagram: Displays instances of classes and their relationships.
- Class Diagram: Represents the system's classes, attributes, and methods.
- Component Diagram: Illustrates software components and their dependencies.
- Composite Structure Diagram: Shows the internal structure of a class or component.

Result:

Thus the problem statement has been defined, feasibility study has been explained in detail and the process model is finalized.

TASK - 2	Software Requirement Specification Document
DATE	

Aim:

To prepare Software Requirement Specification Document for Book Bank system.

1 Introduction

The **Railway Reservation System version 1.0** is a sophisticated solution, aiming to revolutionize the way we book and manage train tickets, fostering a culture of efficient travel and seamless connectivity. The Railway Reservation System is a comprehensive platform designed to streamline the ticketing process, ensuring that seats are readily available to all passengers. This system goes beyond traditional manual reservation models, incorporating modern technology to enhance accessibility and user experience. The Railway Reservation System serves as a guiding light, opening the world of convenient travel to everyone.

1.1 Purpose.

The purpose of this document is to illustrate the requirements of the project **Railway Reservation System**. The document gives the detailed description of both functional and non-functional requirements proposed by the client. The document is developed after a number of consultations with the client and considering the complete requirements specifications of the given project. The final product of the team will be meeting the requirements of this document

1.2 Scope

The **Railway Reservation System** will include features such as an online train schedule catalog, digital ticket booking, automated seat allocation/cancellation, personalized travel suggestions, and passenger management. It will be accessible to both passengers and administrators through a user-friendly interface.

1.3 Definitions, Acronyms, and Abbreviations

Member	The one who registers himself and purchase books from the bank.
Database	Database is used to store the details of members and books
Administrator	The one who verifies the availability of book and issue the

1.4 References

The references for the above software are as follows:-

i. www.google.co.in

ii. www.wikipedia.com

iii. IEEE. Software Requirements Specification Std. 30-1993.

1.5 Overview

- Chapter 1.0 discusses the purpose and scope of the software.
- Chapter 2.0 describes the overall functionalities and constraints of the software and user characteristics.
- Chapter 3.0 details all the requirements needed to design the software.

2. System description

2.1 Product Perspective

This **Railway Reservation System** replaces the traditional, manual ticket booking process by which a lot of paperwork will be reduced. This system will provide a search functionality to facilitate train schedules and ticket availability. This search will be based on various categories viz train name, train number, source, and destination. Also, advanced search features are provided in order to filter various categories. In this system, the Admin should have a computer to view the details of the passengers. The Admin should be able to see the number of tickets a particular passenger has booked, details of the journey, whether the ticket is confirmed, cancelled, or on the waiting list. This is the primary feature. Another feature is that the railway admin is able to edit the passenger or booking details. The admin also looks after long-pending reservations and cancellations.

2.2.Product Function

There are two different users who will be using this product.

1)User(Member)

2)Admin

The **Railway Reservation System** serves as a comprehensive platform, encompassing key functions to revolutionize the ticket booking and travel management process. With robust train schedule and seat management, it enables administrators to efficiently organize and maintain accurate information on train timings, routes, classes, and seat availability. The system facilitates user interaction through seamless account creation, authentication, and personalized passenger profiles. Online reservation functionality streamlines the process, allowing passengers to book tickets conveniently with timely notifications. Personalized travel suggestions based on user preferences and passenger management features, including booking and cancellation reports for administrators, contribute to an optimized travel experience. The system further allows administrators to configure settings, ensuring flexibility and customization.

2.3 User Function

a) Admin:

The **admin can view the different categories of trains available in the reservation system**. The admin can also view the list of trains available in each category such as express, passenger, superfast, and AC/Non-AC classes. The admin can manage ticket cancellations and refunds initiated by passengers. They can add new train schedules and their details to the database. The admin can also edit the information of the existing trains in the database. The admin can check reports of ticket bookings, cancellations, and seat availability. Furthermore, the admin can access all the accounts of the registered passengers..

b) User (Member):

The **passenger can view the list of trains available in each category**. Passengers can view the different categories of trains such as express, passenger, superfast, and AC/Non-AC classes.A passenger can view the tickets they have booked. They can place a request for ticket booking on a particular train and can search for a specific train based on train name, number, source, or destination. Passengers can also view the history of their previous bookings and cancellations.

2.4 System constraints

System constraints outline limitations or restrictions that may impact the development or operation of the Railway Reservation System.

- **Compatibility Constraints**: The system must be compatible with modern web browsers such as Google Chrome, Mozilla Firefox, and Safari.
- **Budgetary Constraints**: The development and maintenance budget for the Railway Reservation System is limited, requiring cost-effective solutions and efficient resource utilization.

- **Infrastructure Requirements**: The system is dependent on a reliable internet connection for users to access online features and services.
- **User Access Constraints:** Users must have internet access to benefit from the online reservation and personalized recommendation features.
- **Hardware Limitations:** The system should be designed to run on standard hardware configurations, taking into account potential limitations in processing power and memory.

- **Data Privacy and Security Compliance:** The system must adhere to data privacy and security regulations, imposing constraints on the collection, storage, and processing of user data.
- **Software Dependency Constraints:** The system may rely on third-party software or libraries, and any dependencies should be clearly defined.
- **Accessibility Standards:** The Book Bank System must comply with accessibility standards to ensure usability for individuals with disabilities.
- **Backup and Recovery Constraints:** The system must implement regular backup procedures, and recovery mechanisms should be in place in case of data loss or system failures.

2.5 System dependencies

The **Railway Reservation System** depends on a designated database system for the storage and management of train schedules, booking details, and passenger records. Network infrastructure is crucial for online ticket booking, availability checks, and communication between system components, while seat allocation modules are essential for automated booking and cancellation processes.

The system may use third-party libraries or APIs for functionalities such as payment processing or travel suggestions. Web technologies, including HTML, CSS, and JavaScript, are integral to the user interface and online features. Integration with an existing user authentication system, stable server infrastructure, and compliance with geographical data hosting and accessibility standards are additional dependencies. Backup and recovery mechanisms safeguard against data loss, and the choice of programming language and framework, along with secure integration with a payment gateway, further contribute to the system's functionality.

3. Specific system requirement

3.1 Functional Requirement

3.1.1 User Management- Users should be able to create accounts with unique identifiers. The system must authenticate users securely for access to personalized features.

3.1.2 Catalog Management - Administrators should have the ability to add, modify, or remove books from the digital catalog. The system must store and display comprehensive information about each book, including titles, authors, genres, and availability status.

3.1.3 Search and Navigation - Users should easily search for books based on titles, authors, genres, or keywords. Users should be able to filter search results based on availability, genre, or other relevant criteria.

3.1.4 Online Reservation System - Users must log in to reserve books. Users should be able to reserve books online, and the system should notify them when reserved books are available for pickup.

3.1.5 Automated Check-in/Check-out - Automated check-in and check-out processes should be enabled using barcode scanning technology.

3.1.6 Personalized Recommendations - Users should be able to create and manage profiles with reading preferences. The system should suggest relevant book titles based on user profiles and preferences.

3.1.7 Resource Management- Administrators should generate reports on the usage patterns of books. Admin should be able to manage the book collection based on demand and usage reports.

3.2 Non Functional requirement

3.2.1 Performance-This system should work concurrently in multiple processors between the working hours in book bank. The system should support at least 500 users.

3.2.2 Safety- The database may get crashed at any certain time due to virus or operating system failure. Therefore, it is required to take the database backup.

3.2.3 Usability- The user interface shall be intuitive and user-friendly. The system should be accessible to users with disabilities.

3.2.4 Security- User login and access to sensitive data should be secure. All communication between the user's browser and the server should be encrypted.

3.3 External Interface

3.3.1 User Interface

The homepage should provide quick access to key functionalities such as book search, reservation, and user login. The search interface must allow users to search for books using various criteria, and the results should be presented in a clear and organized manner. Each book detail page

should display comprehensive information, including the book cover, author, and availability status. Users should be able to easily navigate to their profile, where they can view their borrowing history and personalized book recommendations.

3.3.2 Hardware Interface

- **User Devices-**The system should be compatible with standard personal computers, laptops, and mobile devices for user access.
- **Barcode Scanners**- Self-service kiosks require compatible barcode scanners for automated check-in and check-out processes.
- **Network Compatibility-** A stable network infrastructure is essential to facilitate communication between the server and user devices.

2.1.2 Software interface

The software interfaces are specific to the target book bank software systems.

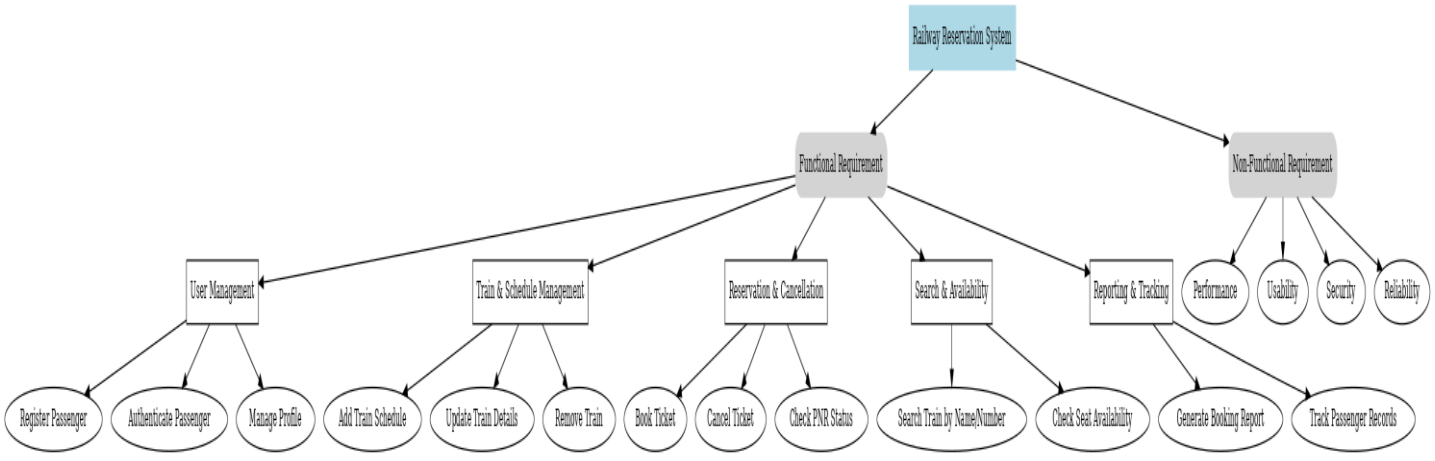
- Languages supported: java (Front end)
- Database: SQL Server (Back end)
- MS-Office
- ArgoUml /StarUml

2.1.3 Communication Interface

These are protocols that are needed to directly interact with the customers. Apart from these protocols, to maintain a healthy relationship with the customer, both formal and informal meetings, group discussions and technical meetings will be conducted frequently

3. Appendices

Appendix A: Definition
Appendix B: Database Schema



The diagram represents the **Functional and Non-Functional Requirements** of a **Railway Reservation System**.

1. Functional Requirements

These are essential operations that the system must perform:

- **User Management**: Includes user registration and authentication for secure access.
- **Ticket Booking**: Allows users to search for trains, book tickets, and receive confirmation.
- **Payment Management**: Supports multiple payment options and ensures successful payment confirmation.
- **PNR Status and Ticket Inquiry**: Enables users to check their **PNR (Passenger Name Record)** status for booking details.
- **Cancellation and Refunds**: Facilitates ticket cancellation based on **PNR number** and processes refunds accordingly.

2. Non-Functional Requirements

These define system quality and performance aspects:

- **Performance**: The system should efficiently handle multiple users and transactions.
- **Usability**: The interface should be user-friendly for seamless booking and inquiry processes.

Result:
Thus, the Software Requirement Specification Document for Book Bank system has been Prepared.

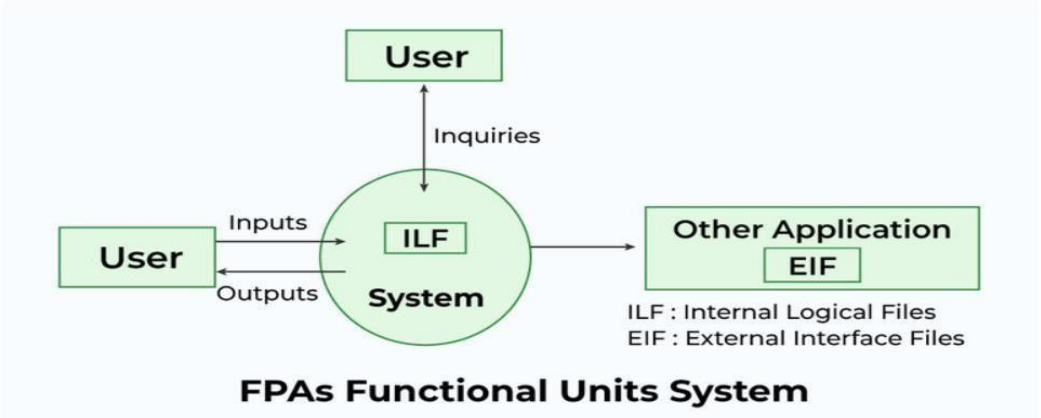
TASK – 3	**To Prepare a detailed estimate using the FP and COCOMO Model. Also prepare a detailed schedule of the project.**
DATE	

Aim:

To perform the estimation (FP and COCOMO) and to develop a project schedule.

FP Model:

The functional size of the product is measured in terms of the function point, which is a standard of measurement to measure the software application.



FPAs Functional Units System

- **Step-1:**

  F = 14 * scale

  Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:

  0 - No Influence

  1 - Incidental

  2 - Moderate

  3 - Average

  4 - Significant

  5 - Essential

- **Step-2:** Calculate Complexity Adjustment Factor (CAF).

  CAF = 0.65 + (0.01 * F)

- **Step-3:** Calculate Unadjusted Function Point (UFP).

  TABLE (Required)

Function Units	Low	Avg	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Multiply each individual function point to corresponding values in TABLE.

- **Step-4:** Calculate Function Point.

  FP = UFP * CAF

Computing the function point when all complexity adjustment factor (CAF) and weighting factors are average, Also computing the productivity, documentation, cost per function for the following data:

1. Number of user inputs = 24
2. Number of user outputs = 46
3. Number of inquiries = 8
4. Number of files = 4
5. Number of external interfaces = 2
6. Effort = 36.9 p-m
7. Technical documents = 265 pages
8. User documents = 122 pages
9. Cost = $7744/ month

COCOMO Model

A project was estimated to be 400 KLOC. The effort and development time for each of the three model i.e., organic, semi-detached & embedded is calculated by basic COCOMO.

The basic COCOMO equation takes the form:
Effort=a*(KLOC)^b PM
Tdev=c*(efforts)^d Months

Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi detected	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Estimated Size of project= 400 KLOC

(i) Organic Mode

E = 2.4 * (400)1.05 = 1295.31 PM
D = 2.5 * (1295.31)0.38=38.07 PM

(ii) Semidetached Mode

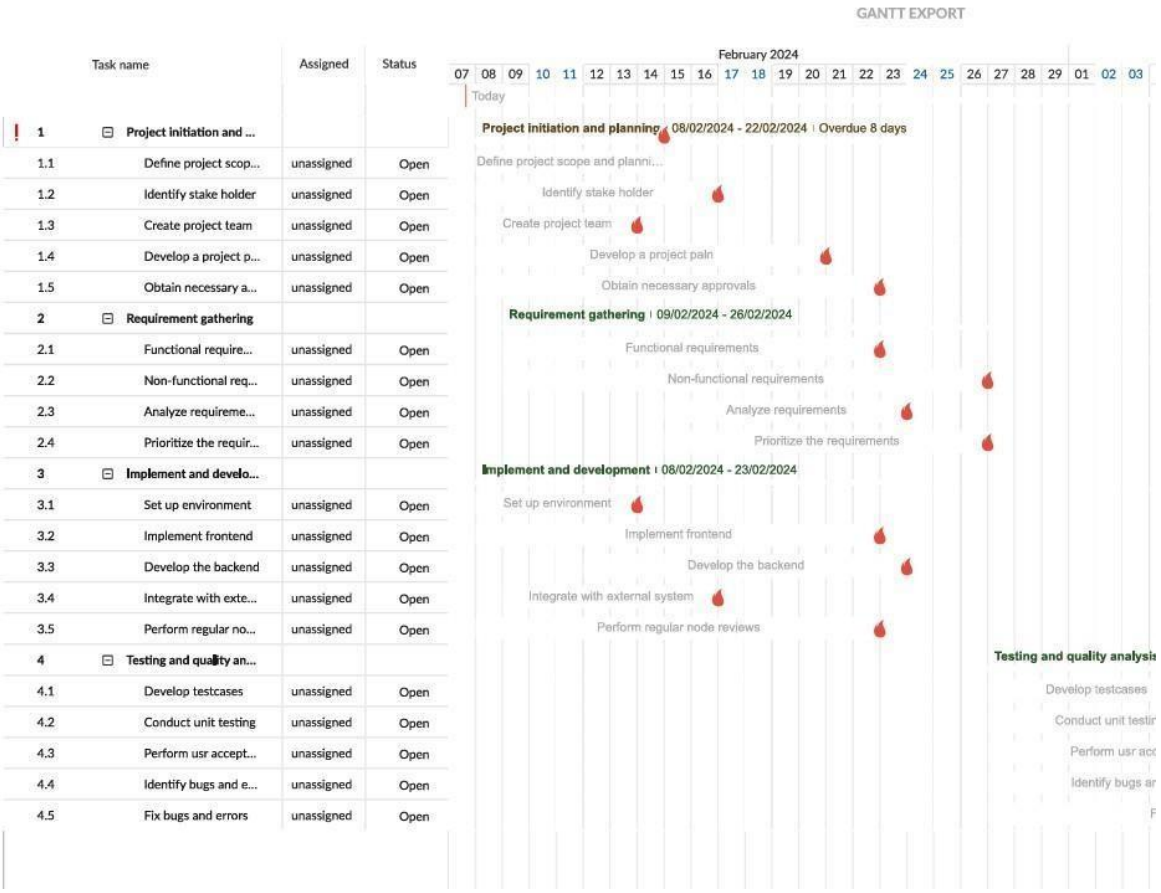
E = 3.0 * (400)1.12=2462.79 PM
D = 2.5 * (2462.79)0.35=38.45 PM

(iii) Embedded Mode

E = 3.6 * (400)1.20 = 4772.81 PM
D = 2.5 * (4772.8)0.32 = 38 PM

Project scheduling:

A project schedule is a timetable that organizes tasks, resources and due dates in an ideal sequence so that a project can be completed on time.



Result:

Thus, the estimation (FP and COCOMO) has been performed project schedule is prepared.

TASK - 4	# USECASE DIAGRAM
DATE	

Aim:

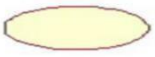
The aim is to Identify Use Cases and develop the Use Case diagram,Entity Relationship Diagram and class diagram for the given scenario.

Use case diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

Use case

A use case describes the sequence of actions a system performs yielding visible results



Actor

An actor represents the roles that the users of the use cases play. An actor may be a person (e.g. student, customer), a device (e.g workstation), or another system (e.g. bank, institution).

E-R Diagram

- The Entity-Relation model represents real-world entities and the relationship between them. An ER diagram is used to design database data models.
- Components of a ER Diagram
  - ✓ Entities
  - ✓ Attributes
  - ✓ Relationship

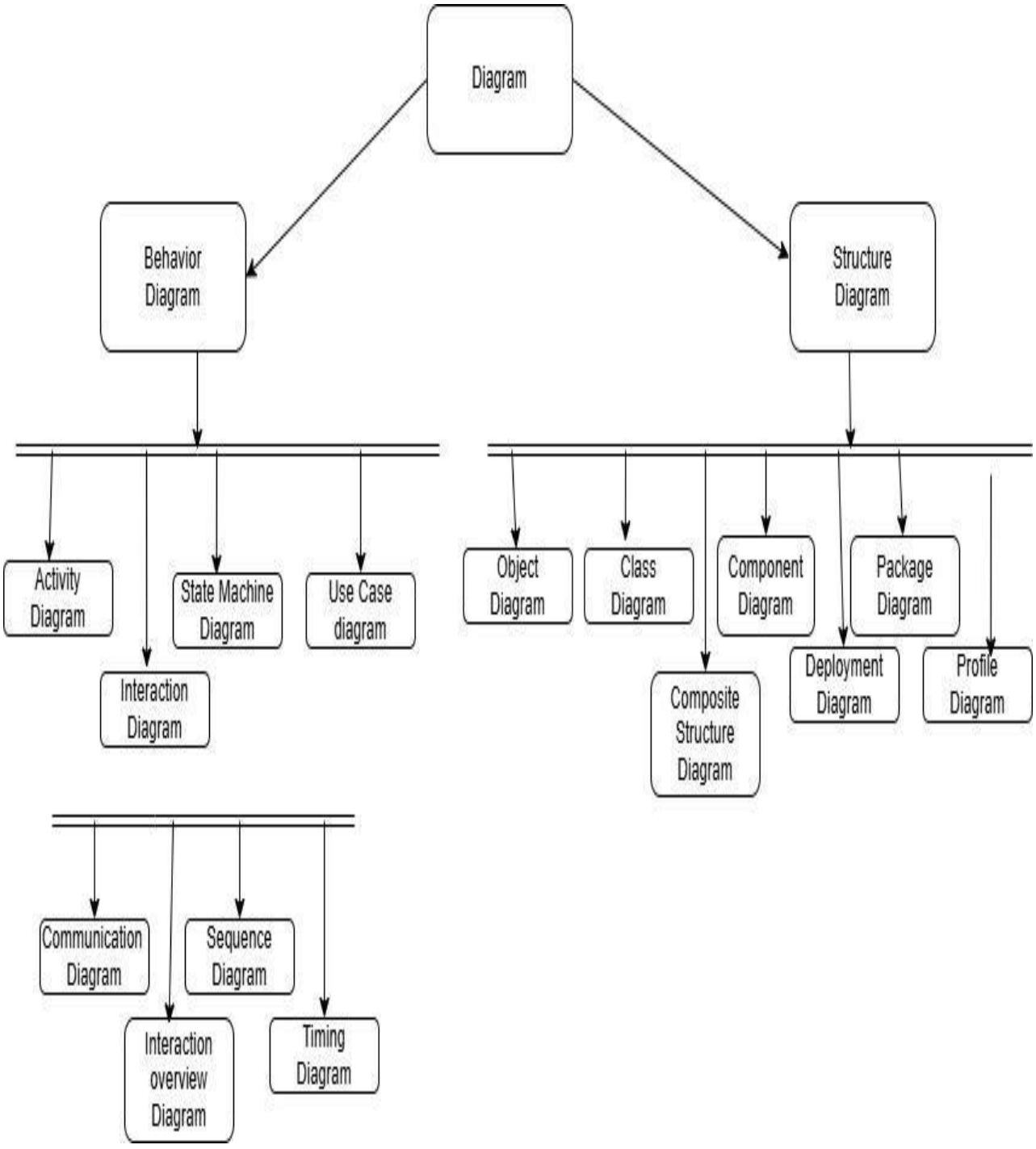
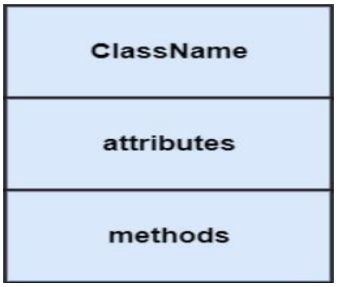


Diagram
Behavior Diagram
Structure Diagram
Activity Diagram
State Machine Diagram
Use Case diagram
Interaction Diagram
Object Diagram
Class Diagram
Component Diagram
Package Diagram
Deployment Diagram
Profile Diagram
Composite Structure Diagram
Communication Diagram
Sequence Diagram
Interaction overview Diagram
Timing Diagram

Class Diagram:

Class Diagram a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects



The purpose of the class diagram can be summarized as:

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

Class diagrams commonly contain the following things

- Classes
- Interfaces
- Collaborations
- Dependency, generalization and association relationships

The class diagram includes three main classes: Book, Member, and Librarian. The Book class represents individual books with attributes such as bookID, title, author, ISBN, and availability. It also includes methods for checking out, returning, updating availability status, and displaying book details. The Member class represents library members with attributes like memberID, name, contact, and a list of borrowed books. It includes methods for borrowing, returning, and displaying borrowed books. The Librarian class represents the librarian with attributes such as librarianID and name. It includes methods for adding, removing, and updating book details, as well as issuing and returning books for members

The given diagram represents the Use Case Model for a Railway Reservation System, illustrating the interactions between different users and system functionalities.

Actors Involved:

User: The primary actor who interacts with the system to perform actions like:

Login

Checking train availability

Booking tickets

Making payments

Checking PNR status

Cancelling tickets

Application Admin: Responsible for managing application-level configurations.

Database Admin: Manages and maintains the system database, ensuring secure and efficient data storage.

Manager: Oversees system operations and may have access to administrative functions.

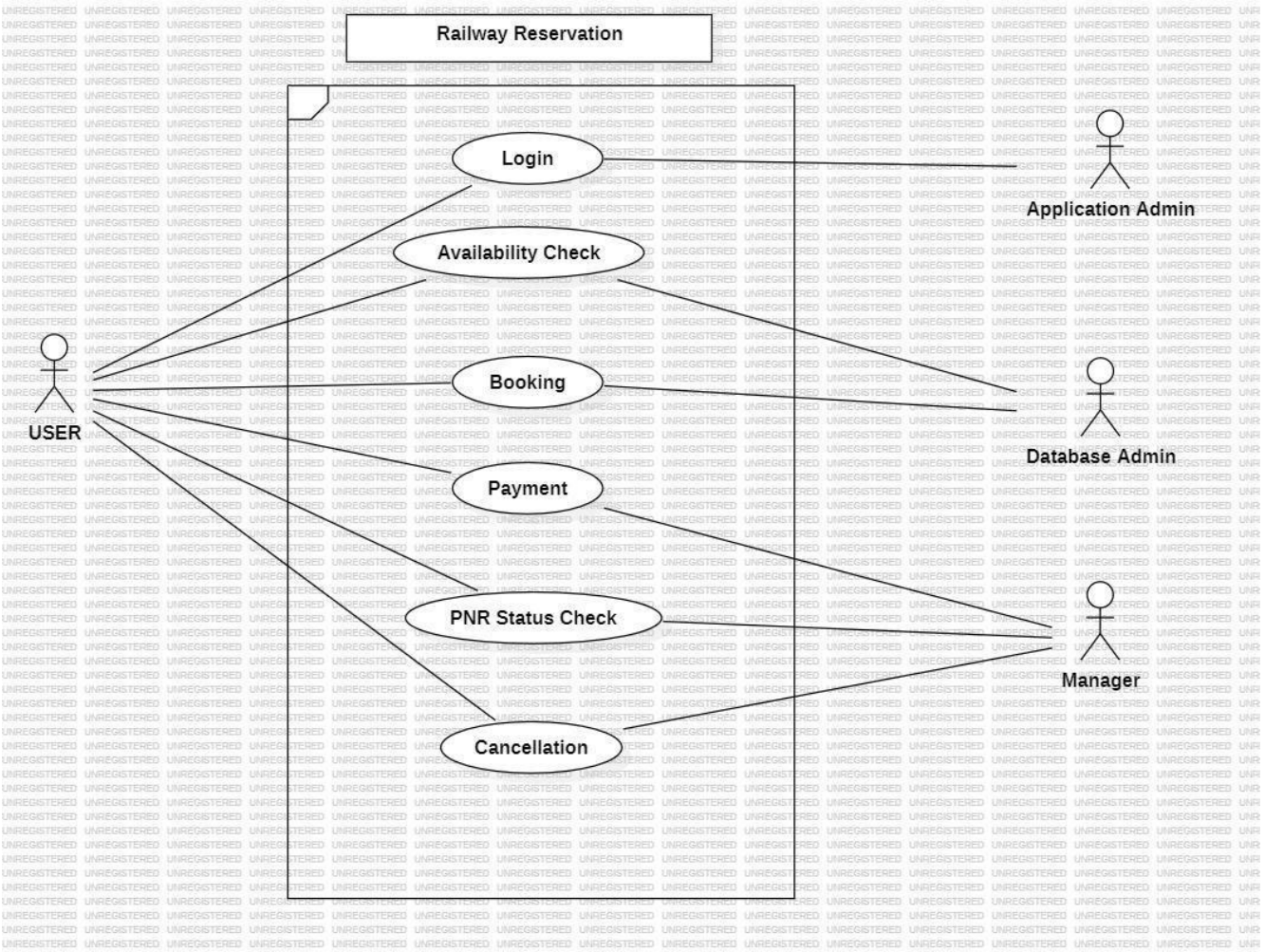
System Functions (Use Cases):

Login: Users must log in to access services.

Availability Check: Users can check the availability of trains before booking.

Booking: Enables ticket reservations based on user preferences.

Payment: Supports payment processing for successful bookings



PNR Status Check: Allows users to track their ticket status.

Cancellation: Provides the option to cancel booked ticket

Result

Thus the Use Cases has been identified and the Use Case diagram, Entity Relationship Diagram and class diagram for the given scenario has been designed

TASK - 5	CLASS DIAGRAM
DATE	

Aim:

The aim is to find the interaction between objects and represent them using UML Sequence & Communication diagrams.

Class diagrams

Class diagram depict interactions of objects and their relationships. They also include the messages passed between them. There are two types of interaction diagrams −

- Sequence Diagram
- Collaboration Diagram

Interaction diagrams are used for modeling

- the control flow by time ordering using sequence diagrams.
- the control flow of organization using collaboration diagrams.
-

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur.

Drawing a sequence diagram helps to

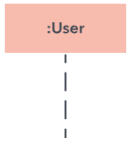
- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

Notations:

1. Object

2. Actor

3. Lifeline



Collaboration Diagram

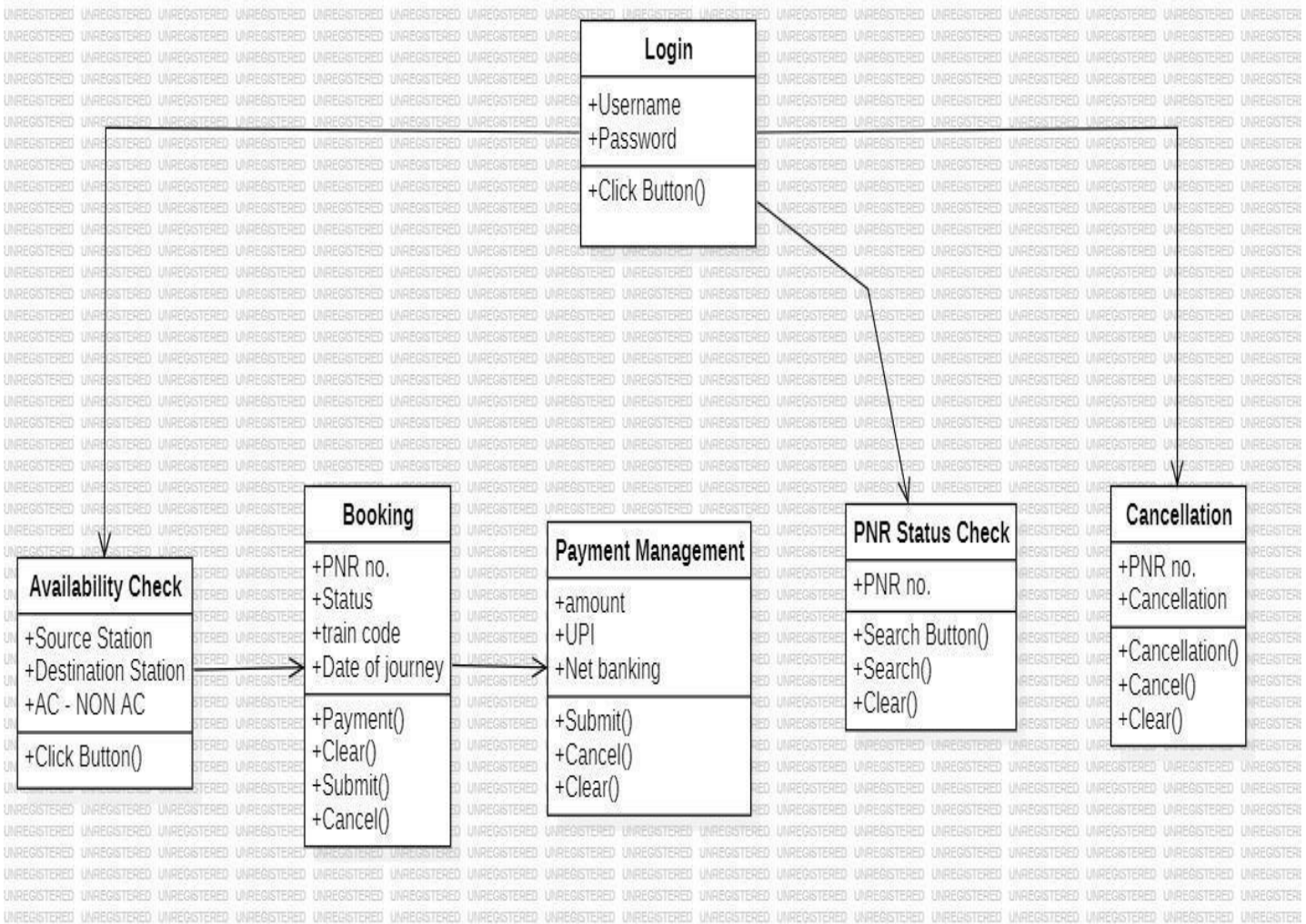
The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming.

An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

Notations

Notations used in communication diagrams are the same notations for sequence diagrams.

- Rectangles represent objects that make up the application.
- Lines between class instances represent the relationships between different parts of the application.
- Arrows represent the messages that are sent between objects.
- Numbering lets you know in what order the messages are sent and how many messages are required to finish a process.



The Class Diagram for the Railway Reservation System visually represents the key components and their interactions within the system. It consists of multiple classes, each with specific attributes and operations that define their roles. The process begins with Login, where users enter their credentials to access the system. Once authenticated, they can proceed to Availability Check, where they input details such as the source and destination stations and ticket preferences (AC or Non-AC) before

Key Components of the System:

1. **Login**
   - Attributes: Username, Password
   - Method: Click Button()
2. **Availability Check**
   - Attributes: Source Station, Destination Station, AC - NON AC
   - Method: Click Button()
3. **Booking**
   - Attributes: PNR no., Status, Train Code, Date of Journey
   - Methods: Payment(), Clear(), Submit(), Cancel()
4. **Payment Management**
   - Attributes: Amount, UPI, Net Banking
   - Methods: Submit(), Cancel(), Clear()
5. **PNR Status Check**
   - Attributes: PNR no.
   - Methods: Search Button(), Search(), Clear()
6. **Cancellation**
   - Attributes: PNR no., Cancellation
   - Methods: Cancellation(), Cancel(), Clear()

Result

Thus, interaction between objects and UML Sequence & Communication diagrams for the given scenario has been design

TASK - 6	SEQUENCE DIAGRAM
DATE	

Aim:

The aim is to find the interaction between objects and represent them using UML Sequence & Communication diagrams.

Interaction diagrams

Interaction diagram depict interactions of objects and their relationships. They also include the messages passed between them. There are two types of interaction diagrams −

- Sequence Diagram
- Collaboration Diagram

Interaction diagrams are used for modeling

- the control flow by time ordering using sequence diagrams.
- the control flow of organization using collaboration diagrams.
-

Sequence diagram

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur.

Drawing a sequence diagram helps to

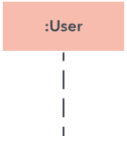
- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

Notations:

1. Object

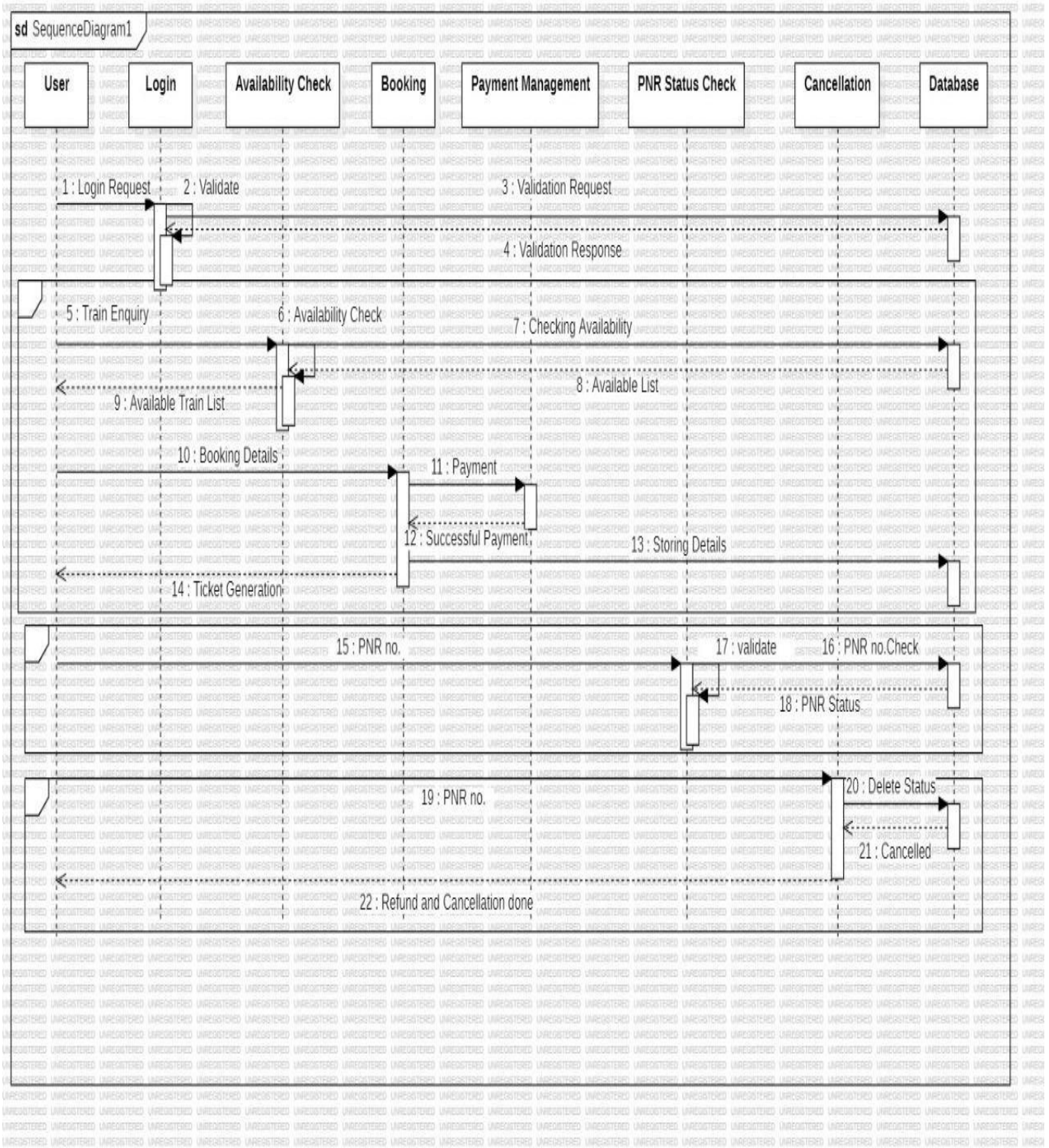
2. Actor

3. Lifeline



Notations used in communication diagrams are the same notations for sequence diagrams.

- Rectangles represent objects that make up the application.
- Lines between class instances represent the relationships between different parts of the application.
- Arrows represent the messages that are sent between objects.
- Numbering lets you know in what order the messages are sent and how many messages are required to finish a process.


sd SequenceDiagram1
User
Login
Availability Check
Booking
Payment Management
PNR Status Check
Cancellation
Database
1 : Login Request
2 : Validate
3 : Validation Request
4 : Validation Response
5 : Train Enquiry
6 : Availability Check
7 : Checking Availability
8 : Available List
9 : Available Train List
10 : Booking Details
11 : Payment
12 : Successful Payment
13 : Storing Details
14 : Ticket Generation
15 : PNR no.
17 : validate
16 : PNR no.Check
18 : PNR Status
19 : PNR no.
20 : Delete Status
21 : Cancelled
22 : Refund and Cancellation done

The Sequence Diagram for the Railway Reservation System visually represents the key components and their interactions within the system. It consists of multiple classes, each with specific attributes and operations that define their roles. The process begins with Login, where users enter their credentials to access the system. Once authenticated, they can proceed to Availability Check, where they input details such as the source and destination stations and ticket preferences (AC or Non-AC) before checking train availability.

Key Components of the System:

1. **Login**
   - Attributes: Username, Password
   - Method: Click Button()
2. **Availability Check**
   - Attributes: Source Station, Destination Station, AC - NON AC
   - Method: Click Button()
3. **Booking**
   - Attributes: PNR no., Status, Train Code, Date of Journey
   - Methods: Payment(), Clear(), Submit(), Cancel()
4. **Payment Management**
   - Attributes: Amount, UPI, Net Banking
   - Methods: Submit(), Cancel(), Clear()
5. **PNR Status Check**
   - Attributes: PNR no.
   - Methods: Search Button(), Search(), Clear()
6. **Cancellation**
   - Attributes: PNR no., Cancellation

Result

Thus,the Sequence & Communication diagrams for the given scenario has been designed and verified successfully

TASK - 7	# INTERACTION DIAGRAM
DATE	

Aim:

The objective is to analyze and represent the interaction between objects using UML Sequence and Interaction Overview Diagrams.

Interaction diagrams

Interaction diagrams illustrate how objects interact with each other and depict the messages exchanged among them. There are two main types:

- **Sequence Diagram** – Represents interactions based on time ordering.
- **Collaboration Diagram** – Shows interactions based on organizational control flow.

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur.

Drawing a sequence diagram helps to

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

Collaboration Diagram

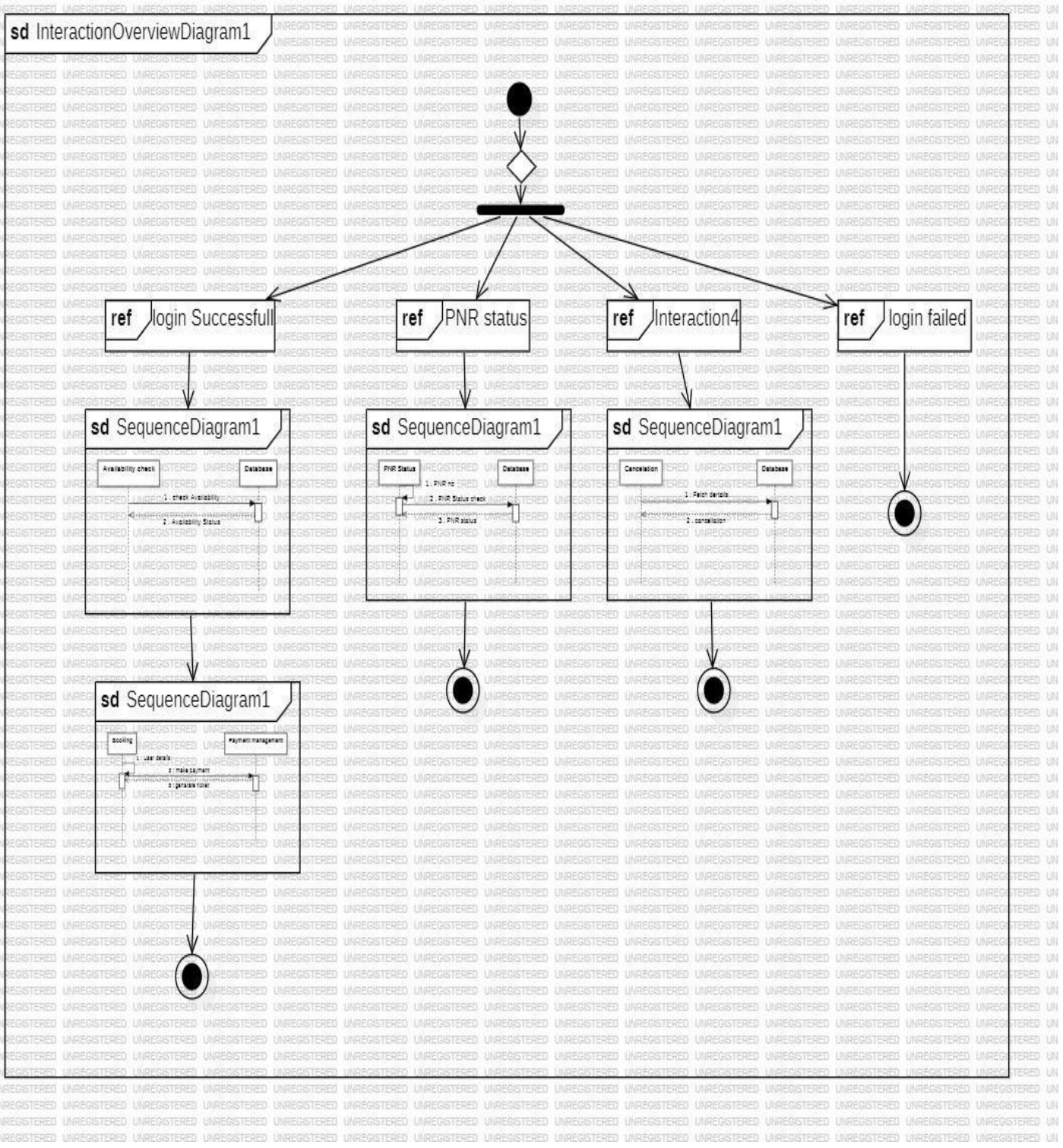
The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming.

An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

Notations

Notations used in communication diagrams are the same notations for sequence diagrams.

- Rectangles represent objects that make up the application.
- Lines between class instances represent the relationships between different parts of the application.
- Arrows represent the messages that are sent between objects.



The Class Diagram for the Railway Reservation System visually represents the key components and their interactions within the system. It consists of multiple classes, each with specific attributes and operations that define their roles. The process begins with Login, where users enter their credentials to access the system. Once authenticated, they can proceed to Availability Check, where they input details such as the source and destination stations and ticket preferences (AC or Non-AC) before checking train availability.

Components of Interaction Diagrams

- **Objects (Participants):** These are the entities that interact within the system. Each object is represented as a lifeline.
- **Lifelines:** A lifeline represents the duration of an object's existence within an interaction. It is depicted as a vertical dashed line.
- **Messages:** Messages are the interactions or communications exchanged between objects, shown as arrows. These can be synchronous, asynchronous, or return messages.
- **Activation Bars:** These indicate the duration for which an object is active and performing an operation.
- **Decision Nodes:** Used in interaction overview diagrams, these control the flow based on conditions.

Result

Thus, interaction between objects and UML Sequence & Communication diagrams for the given scenario has been designed.

TASK - 8	# ACTIVITY DIAGRAM
DATE	

Aim:

The aim is to model the User Interface diagrams, State Transition diagram and Activity diagram for the given scenario.

User Interface Diagram

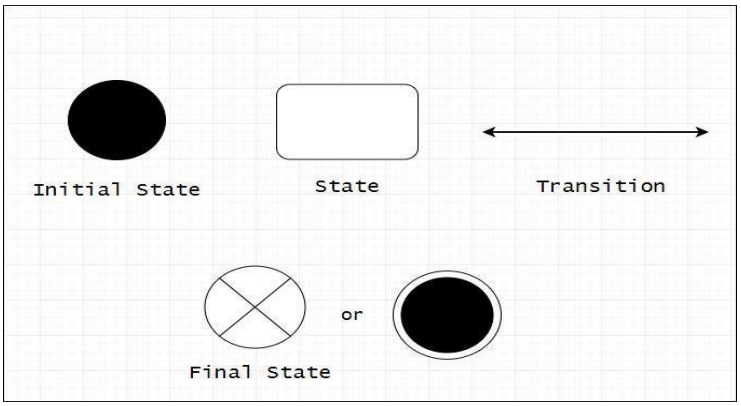
User interface (UI) design is the process designers use to build interfaces in software or computerized devices, focusing on looks or style.

User interface design for a book bank system aims to make it easy and straightforward for users to find, borrow, and return books. It includes clear menus, search bars, and buttons for navigation, ensuring users can quickly locate books they need. The design prioritizes simplicity and ease of use, ensuring a pleasant experience for all users interacting with the system. Visual elements like buttons, icons, and color schemes are chosen to enhance usability and guide users through the process seamlessly. Additionally, the design ensures accessibility for users of all abilities, making it simple for anyone to navigate and utilize the book bank system effectively.

State Transition diagram

The state machine diagram is also called the Statechart or State Transition diagram, which shows the order of states underwent by an object within the system.

Notations



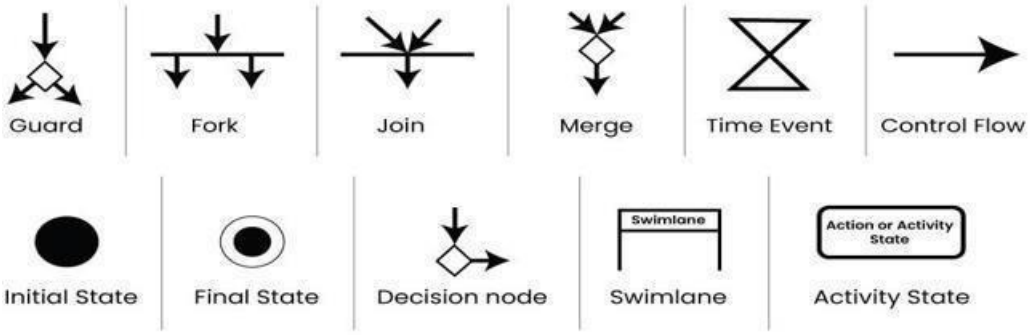
A state chart diagram for a book bank system illustrates the various states and transitions that the system and books can undergo. It typically starts with an "Idle" state where the system is ready to handle requests. When a user logs in, the system transitions to a "Logged In" state. From there, it may move to states like "Searching" when the user is looking for books, "Borrowing" when a book is selected for loan, and "Renewing" if the user extends the loan period. Once a book is returned, it reverts to the

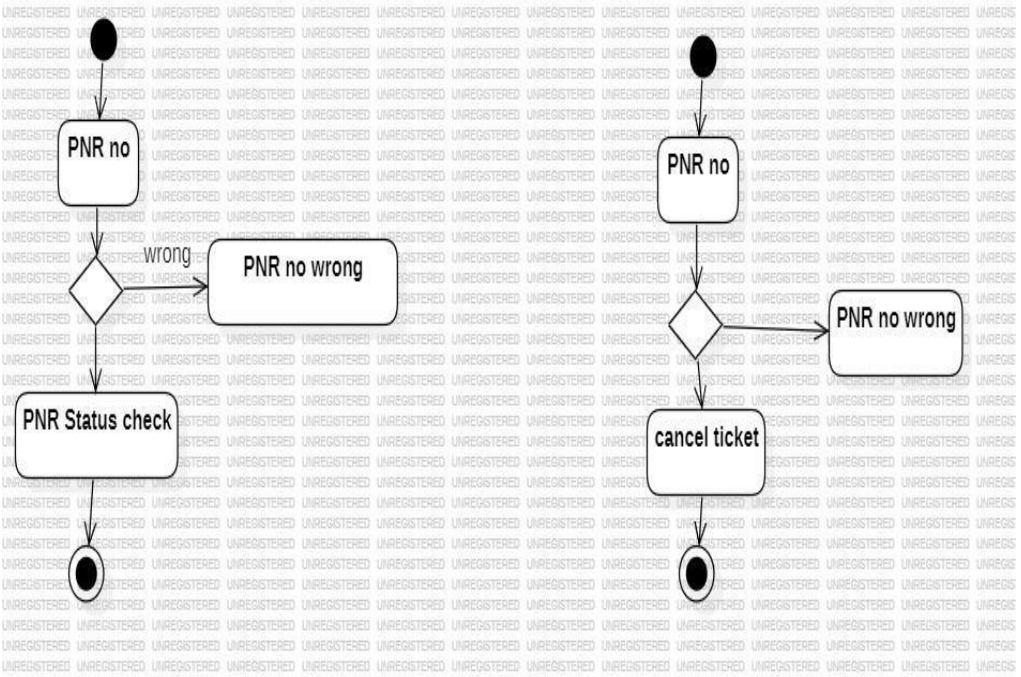
"Available" state. The state chart simplifies understanding the system's behaviour and possible states during user interactions, ensuring smooth management of book borrowing and returning processes.

Activity diagram

Activity diagrams show the flow of one activity to another within a system or process

Notations





The first diagram illustrates the process of checking a Passenger Name Record (PNR) status. The process begins when the user enters their PNR number. The system then verifies whether the PNR is valid. If the PNR number is incorrect, an error message is displayed, and the process stops. However, if the PNR is correct, the system retrieves and displays the status of the booked ticket. This ensures that users can only access valid ticket details, improving the efficiency and reliability

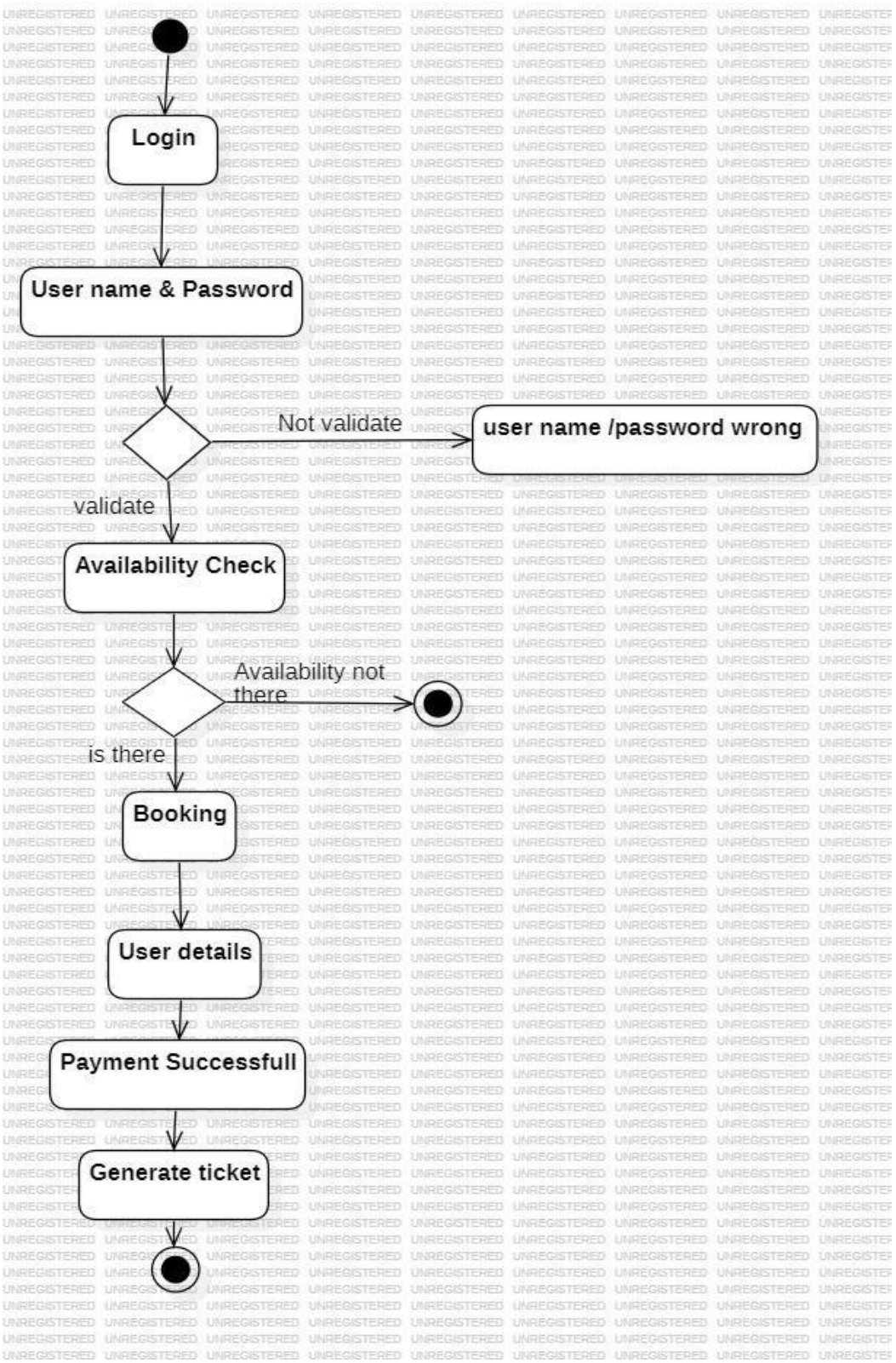


Figure: Activity Diagram

The third diagram (larger image) represents the process of booking a ticket:

1. Start: The user begins the booking process by logging in.
2. User Authentication:
    - The user enters a username and password.
    - If the credentials are incorrect, an error message is displayed: "Username/Password Wrong".
    - If correct, the system proceeds to check ticket availability.
3. Ticket Availability Check:
    - If tickets are not available, the process stops.
    - If available, the system proceeds to booking.
4. Booking and User Details:
    - The user enters necessary details.
    - Payment is processed.
5. Payment Confirmation:
    - If the payment is successful, the system generates the ticket.
6. End: The process is completed.

This diagram ensures that the ticket booking process follows proper validation and checks, preventing unauthorized access or errors in booking.

Result

Thus the User Interface diagrams, State Transition diagram and Activity diagram for the given scenario has been designed.

TASK – 9	# PACKAGE DIAGRAM
DATE	

Aim:

The aim is to model the package diagrams, State Transition diagram and Activity diagram for the given scenario.

Package Diagram

A Package Diagram is a structural UML (Unified Modeling Language) diagram used to depict the organization of a system into packages, modules, or components. It shows dependencies between different parts of a system, making it useful for understanding large and complex software architectures.

The package diagram for the Online Railway Reservation System illustrates how different modules interact to provide functionalities such as:

Train information management

User account handling

Ticket booking

Payment processing

Each package represents a specific responsibility within the system, ensuring modularity and separation

In large software systems, where multiple services, databases, and interfaces interact with each other, package diagrams are particularly beneficial. They help developers, system architects, and stakeholders understand how different parts of the system communicate, which dependencies exist, and how modifications in one component may affect the rest of the system.

The Online Railway Reservation System Package Diagram follows this modular approach, dividing the system into various interconnected packages. Each package serves a specific purpose, ensuring that different functionalities such as user management, train data retrieval, booking operations, payment processing, and notification handling are logically separated.

"Available" state. The state chart simplifies understanding the system's behaviour and possible states during user interactions, ensuring smooth management of book borrowing and returning processes.

The Online Railway Reservation System is a complex system that enables users to search for trains, book tickets, process payments, and receive notifications regarding their bookings. This system relies on multiple components working together, each handling a specific function.

To better understand this system, let’s analyze the different packages included in the package diagram and their relationships.

The Root Package: OnlineRailwayReservation

At the top of the hierarchy is the OnlineRailwayReservation package, which acts as the central module that ties all sub-packages together. This package encapsulates the core services of the system and ensures seamless interaction between various functionalities.

Within this main package, multiple sub-packages are defined, each responsible for a specific aspect of the railway reservation process. These sub-packages are:

Train Service (Manages train schedules and availability)

User Account Management (Handles user authentication and profiles)

Booking System (Manages ticket reservations and storage)

Payment Processing (Handles online transactions)

Notification System (Sends alerts and updates to users)

Each of these sub-packages follows a layered architecture, separating user interfaces (UI),

Train Service Module

Components

TrainService: Handles all train-related operations, such as retrieving schedules, routes, and seat availability.

TrainDB: A database that stores information about trains, including routes, timings, and seat availability.

Functionality

Whenever a user searches for available trains, the request is forwarded to TrainService, which processes the request and fetches data from TrainDB. The TrainService module ensures that users receive up-to-date train schedules and availability status.

For example, if a user searches for a train from Chennai to Bangalore, the TrainService queries the TrainDB, retrieves the relevant data, and displays the results to the user.

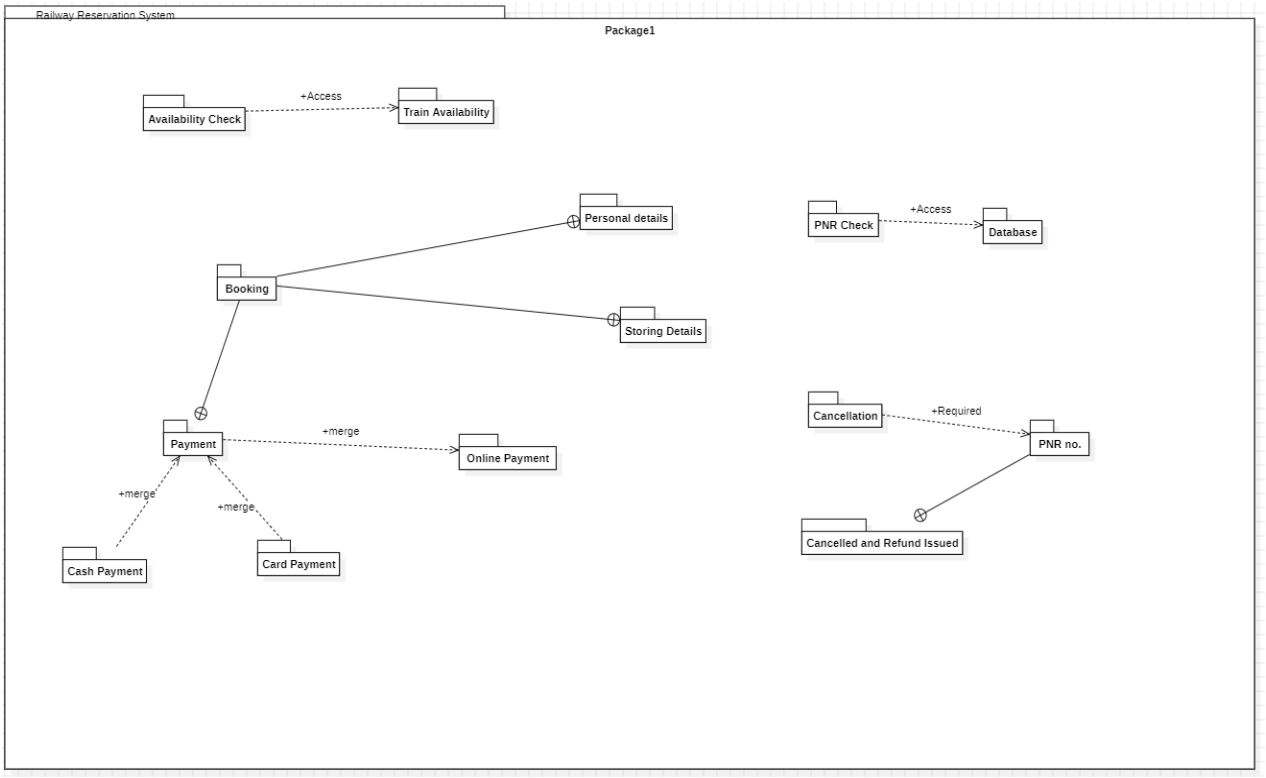
Relationships

TrainService accesses TrainDB to fetch and update train-related information.

The BookingService module also interacts with TrainService to verify seat availability before proceeding with a booking.

The TrainService package manages all train-related operations, including fetching schedules, routes, and availability.

The Online Railway Reservation System package diagram represents a structured and modular architecture where different components interact to facilitate user authentication, train booking, payment processing, and notification services. At the core of this system lies the OnlineRailwayReservation package, which acts as the central hub, linking various modules such as the Train Service, User Account Management, Booking System, Payment Gateway, and Notification Service. This structure ensures seamless communication between the different components, providing a well-organized and efficient railway reservation system.



The Train Service Module plays a crucial role in retrieving train schedules, routes, and seat availability. It comprises the TrainService component, which interacts with databases to fetch real-time information. This module also works alongside the AccountService, which handles user authentication and profile management. The UserDB stores essential user credentials, ensuring a secure and reliable login system. When a user searches for available trains, the TrainService queries the TrainDB for relevant details, and this information is then used by the BookingService to facilitate reservations.

The User Account Management Module allows users to register, log in, and manage their profiles through the UserAccountUI. This UI interacts with the AccountService, which verifies login credentials and retrieves account-related data from the UserDB. A successful authentication process allows users to proceed with train bookings. This module ensures that only registered and authenticated users can access the booking services, thereby enhancing security and user experience.

The Booking System is one of the most significant modules in the architecture, as it handles train reservations. Users interact with the BookingUI to search for trains, select travel details, and initiate bookings. The BookingService plays a pivotal role in this process, as it manages all booking-related operations. It accesses the TrainDB to verify seat availability and updates the BookingDB once a reservation is confirmed. By structuring the booking process efficiently, this module ensures a smooth and hassle-free experience for passengers.

An integral part of the railway reservation system is the Payment Processing Module, which ensures that users can securely pay for their tickets. The PaymentGateway interacts with the BookingService to facilitate transactions. When a user proceeds to payment after booking confirmation, the system verifies and processes the transaction through this gateway.

Result

Thus the Package diagram for the given scenario has been designed. Successfully and tested

scenario has been designed.

TASK - 10	# COMPONENT DIAGRAM
DATE	

Aim:

The aim is to model the component diagram for the given scenario.

Component Diagram

A Component Diagram is a structural UML (Unified Modeling Language) diagram used to visualize the high-level structure of a system. It depicts the various software components, their relationships, and how they interact within the system. This type of diagram is particularly useful for understanding system architecture, dependencies, and interactions between different modules.

The component diagram for the Online Railway Reservation System illustrates how different components work together to provide functionalities such as:

Each component represents a specific service within the system, ensuring modularity and separation of concerns. This helps in better maintainability, scalability, and reusability of the system.

Component Overview

The Online Railway Reservation System consists of multiple components categorized into:

1. User Interface Layer - Handles interactions with the users.
2. Business Logic Layer - Contains core functionalities and service components.
3. Data and External Services Layer - Manages database storage and third-party services.

User Interface Layer:

- User Account UI: Allows users to register, log in, and manage their accounts.
- Booking UI: Enables users to search for trains, book tickets, and manage reservations.

Business Logic Layer:

- Account Service: Handles user authentication and profile management by interacting with the User Database.
- Booking Service: Manages the booking process, including verifying seat availability, processing reservations, and updating booking records.
- Train Service: Retrieves train schedules, routes, and seat availability.

Data and External Services Layer:

- Databases:
  - User Database: Stores user account details and authentication data.
  - Booking Database: Maintains records of all bookings and reservations.
  - Train Database: Holds train-related information, such as schedules, routes, and available seats.
- External Services:
  - Payment Gateway: Processes online payments securely for ticket bookings.
  - Notification Service: Sends booking confirmations, alerts, and updates to users.

Component Interactions

1. User Account Management:
    - The User Account UI communicates with the Account Service, which verifies user credentials using the User Database.
    - Once authenticated, users can access booking services.
2. Train Search and Booking:
    - The Booking UI interacts with the Booking Service.
    - The Booking Service checks seat availability using the Train Service, which queries the Train Database.
    - If seats are available, the booking is confirmed and stored in the Booking Database.
3. Payment Processing:
    - After selecting a train, users proceed with payment through the Payment Gateway.
    - The Payment Gateway communicates with the Booking Service to finalize the transaction.
4. Notification System:
    - Once a booking is successful, the Notification Service sends confirmation messages via email or SMS.

Functionality

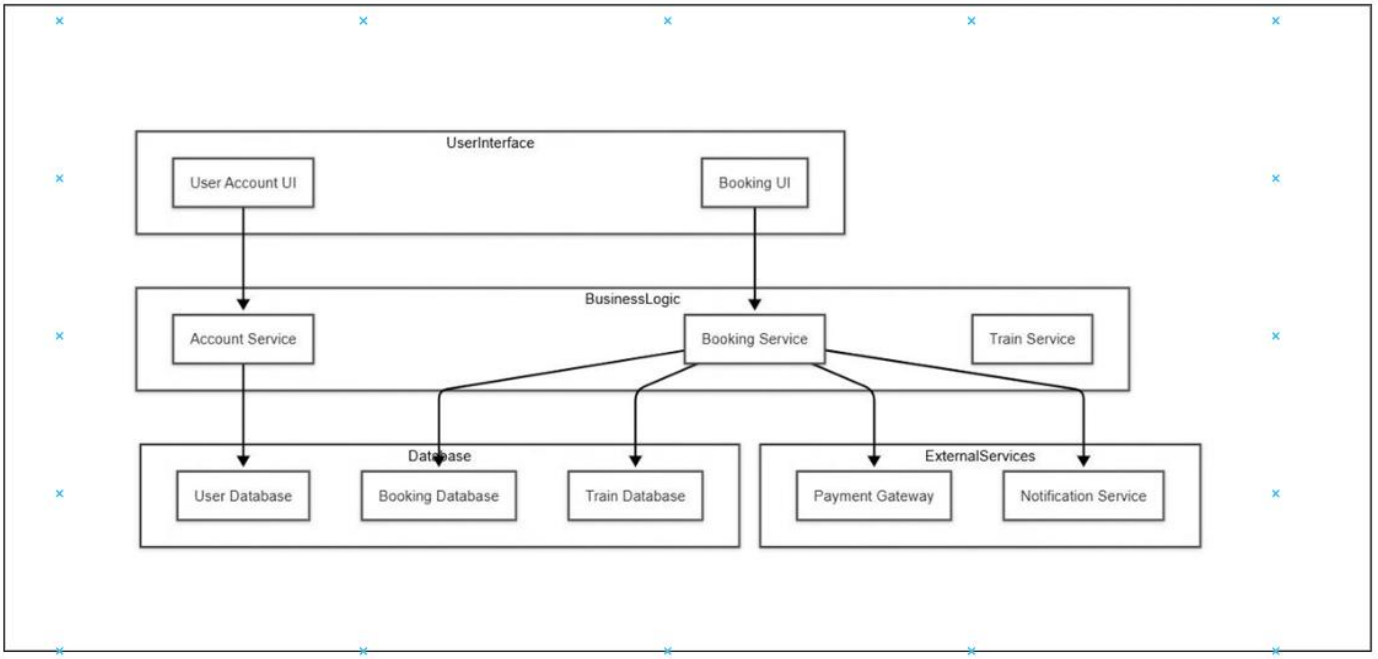
Component diagrams are essential for understanding system architecture, ensuring modularity, and defining clear dependencies between different parts of a software application. Each component in the diagram represents a self-contained unit with well-defined interfaces that communicate with other components. This modular approach enhances reusability, maintainability, and scalability of the system.

In software development, component diagrams are particularly useful for large and complex applications, such as enterprise systems, web applications, and distributed systems. They help developers, architects, and stakeholders understand the relationships between different components, ensuring that changes in one part of the system do not adversely affect others.

A typical component diagram consists of three main elements:

1. Components – These are the core building blocks of the system, representing different functionalities such as authentication, data processing, or external service integration.
2. Interfaces – They define how components interact with each other, ensuring proper communication through clearly defined inputs and outputs.
3. Dependencies – These show the relationships between components, indicating how one component relies on another for its operations.

By using component diagrams, software engineers can identify potential issues early in the development process, optimize system design, and improve overall performance. Whether designing a desktop application, cloud-based service, or enterprise solution, component diagrams serve as a valuable tool in ensuring a well-structured and efficient software architecture.



The given Component Diagram represents the architecture of an Online Railway Reservation System, showing how various components interact to enable user authentication, train search, booking management, payment processing, and notifications. The system is divided into three main layers: User Interface Layer, Business Logic Layer, and Data & External Services Layer.

The User Interface Layer consists of two components: User Account UI and Booking UI. The User Account UI allows users to log in, register, and manage their profiles, while the Booking UI enables users to search for trains, book tickets, and check their reservations. These UIs interact with the Business Logic Layer, which contains the core services of the system.

In the Business Logic Layer, the Account Service is responsible for handling user authentication and profile management by communicating with the User Database. The Booking Service manages ticket reservations by verifying seat availability and updating the Booking Database. Additionally, the Train Service fetches train schedules, routes, and seat availability from the Train Database.

The Data & External Services Layer includes various databases and third-party services that support the system. The User Database stores user login credentials and personal details, the Booking Database maintains ticket reservations, and the Train Database contains details about train schedules and seat availability. The system also interacts with external services such as the Payment Gateway, which processes online transactions securely, and the Notification Service, which sends booking confirmations and updates to users.

When a user logs in via the User Account UI, the Account Service verifies the credentials using the User Database. Upon successful authentication, the user can search for available trains through the Booking UI, which interacts with the Booking Service. The Booking Service then queries the Train Service to check seat availability in the Train Database. If seats are available, the booking is confirmed, and the details are stored in the Booking Database. The user then proceeds to make a payment via the Payment Gateway, which processes the transaction and notifies the Booking Service upon successful payment. Finally, the Notification Service sends an email or SMS confirmation to the user regarding their booking.

Result

Thus the Component diagram for the given scenario has been designed. Successfully and tested

TASK - 11	# DEPLOYMENT DIAGRAM
DATE	

Aim:

The aim is to model the Deployment diagram for the given scenario.

Deployment Diagram

A Deployment Diagram is a type of UML (Unified Modeling Language) diagram that models the physical deployment of software components on hardware nodes. It illustrates the architecture of the system from a hardware perspective, showing how software is distributed across machines and devices in a network. Deployment diagrams are mainly used to visualize the hardware topology of a system, the software artifacts (like executables, libraries, databases, etc.) deployed on each hardware node, and the communication between nodes. They are especially useful during the system design and deployment phases to understand where and how the system components will run in the real-world environment.

- To represent the physical view of the system architecture.
- To show the configuration of runtime processing nodes and the software components deployed on them.
- To illustrate how system components are distributed across hardware environments (e.g., client-server systems, cloud services, embedded systems).
- To help in analyzing system scalability, performance, and security aspects.

Key Elements in Deployment Diagrams

1. **Nodes** – Represent physical devices or execution environments (e.g., server, client machine, cloud instance, mobile device).
2. **Artifacts** – Executable files, libraries, databases, or other deployable software units.
3. **Communication Paths** – Lines that show the communication links between different nodes. These can represent network connections, protocols, or data exchanges.
4. **Dependencies** – Show how artifacts depend on each other across nodes.

Example Scenario: Online Railway Reservation System

In the context of the **Online Railway Reservation System**, the deployment diagram might include:

- **Client Node**: Represents the end user's device (like a mobile phone, laptop, or desktop). The User Interface (UI) of the system is deployed here, allowing users to interact with the system.
- **Web Server Node**: Hosts the application layer, which includes components like the Booking Service, Account Service, and Train Service. These components handle business logic and process user requests.
- **Database Server Node**: Stores system data, such as user information, booking details, and train schedules. This includes the **UserDB**, **BookingDB**, and **TrainDB**.
- **Payment Gateway Node (External Node)**: Represents a third-party payment processing service, ensuring secure transactions.
- **Notification Server Node**: Handles sending emails or SMS alerts to users after a booking or transaction.

These nodes are connected via **network communication paths**, indicating how data flows between the client, web server, database, and external services.

A **Deployment Diagram** provides a clear picture of how software components are physically distributed and interact across various hardware environments. For systems like the Online Railway Reservation System, it plays a critical role in

ensuring smooth deployment, optimizing system performance, and managing resource allocation effectively. It also helps developers and network engineers understand the system's runtime behavior and how different components collaborate to deliver seamless functionality to users.

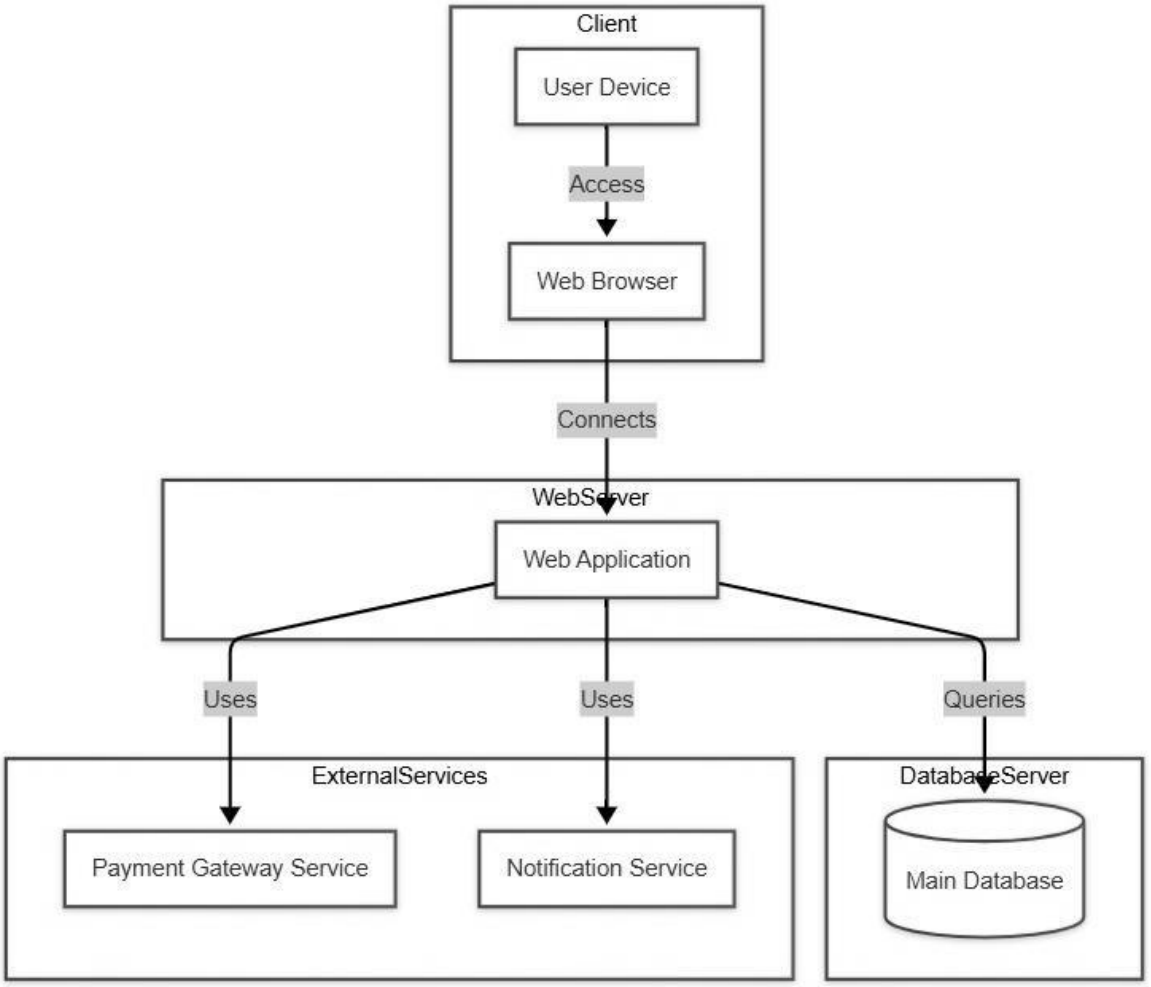
Component diagrams are essential for understanding system architecture, ensuring modularity, and defining clear dependencies between different parts of a software application. Each component in the diagram represents a self-contained unit with well-defined interfaces that communicate with other components. This modular approach enhances reusability, maintainability, and scalability of the system.

In software development, component diagrams are particularly useful for large and complex applications, such as enterprise systems, web applications, and distributed systems. They help developers, architects, and stakeholders understand the relationships between different components, ensuring that changes in one part of the system do not adversely affect others.

A typical component diagram consists of three main elements:

4. Components – These are the core building blocks of the system, representing different functionalities such as authentication, data processing, or external service integration.
5. Interfaces – They define how components interact with each other, ensuring proper communication through clearly defined inputs and outputs.
6. Dependencies – These show the relationships between components, indicating how one component relies on another for its operations.

By using component diagrams, software engineers can identify potential issues early in the development process, optimize system design, and improve overall performance. Whether designing a desktop application, cloud-based service, or enterprise solution, component diagrams serve as a valuable tool in ensuring a well-structured and efficient software architecture.



The givenDeployment Diagram represents the architecture of an Online Railway Reservation System, showing how various components interact to enable user authentication, train search, booking management, payment processing, and notifications. The system is divided into three main layers: User Interface Layer, Business Logic Layer, and Data & External Services Layer.

Explanation of the Deployment Diagram

1. Client Layer (User Interaction)

- User Device: Represents the device used by the end user, such as a laptop, mobile phone, or tablet.
- Web Browser: The user accesses the system through a web browser. This acts as an interface to interact with the application.
- Connection to Web Server: The browser sends requests to the Web Server to access the application.

2. Web Server Layer (Application Processing)

- Web Application: The core application that handles all business logic and user requests.
- Uses External Services:
  - Payment Gateway Service: Handles online payments for ticket bookings.
  - Notification Service: Sends booking confirmations and updates via email or SMS.
- Queries Database Server:
  - Fetches and updates booking details, train schedules, user information, and other data.

3. Database Server Layer (Data Storage)

- Main Database: Stores all necessary information, including:
  - User details
  - Train schedules
  - Ticket bookings
  - Transaction records

Functionality of the Deployment Diagram

1. User Access:
   - The user interacts with the system via a web browser on their device.
   - The browser sends HTTP requests to the Web Server to fetch pages or perform operations.
2. Web Application Processing:
   - The Web Server processes the user's request, validates inputs, and executes necessary operations.
   - It interacts with external services (e.g., Payment Gateway, Notification Service) as required.
3. Database Communication:
   - The Web Application queries the Main Database to retrieve or update information.
   - Data is stored securely, ensuring availability for future requests.
4. External Services Integration:
   - If a user makes a payment, the Web Server interacts with the Payment Gateway Service for transaction processing.
   - Once a booking is confirmed, the Notification Service sends alerts to the user.

This Deployment Diagram effectively represents how different components interact in a web-based system. It ensures scalability, security, and efficiency, making it a well-structured approach for an Online Railway Reservation System or any similar web application. The layered architecture allows smooth communication between users, applications, external services, and databases, ensuring an optimized and seamless user experience.

Result

Thus the deployment diagram for the given scenario has been designed. Successfully and tested

TASK - 1	Design of Problem Statement, Feasibility Study, and Process Model for Stock Management System
DATE	

Aim:
The aim is to design a problem statement, perform a detailed feasibility study, and finalize the process model to be used for developing a Stock Management System.

Problem Statement:
In a Stock Management System, administrators and staff can efficiently manage the inventory of a business by adding, updating, deleting, and tracking products. Each product record includes details such as product ID, name, category, quantity, supplier, purchase date, and selling price.
The system automatically updates stock levels after sales or restocking activities. If the stock quantity of any product falls below the defined threshold, the system alerts the user to reorder that item.
The users can also perform actions such as viewing available stock, checking transaction history, generating reports, and managing supplier information. The system ensures data accuracy, reduces human errors, and provides up-to-date information for decision-making and inventory control. It also allows for easy search and filtering of products, improving operational efficiency.

Scope:
The Stock Management System aims to automate and simplify the process of tracking, managing, and organizing stock items within a business.
The scope includes:

- Managing product and supplier records
- Monitoring stock levels in real time
- Generating stock and sales reports
- Sending restock alerts when inventory is low
- Providing user authentication and secure access
- Enhancing decision-making with accurate, timely information

This system is suitable for retail shops, warehouses, and wholesale businesses looking to reduce manual errors, prevent stockouts, and improve profitability through efficient stock control.

Feasibility Study:
Feasibility study evaluates the practicality of the project in terms of technical, economic, and operational aspects.

1. Technical Feasibility:
The Stock Management System can be developed and executed on standard computing infrastructure with the following minimum requirements:

- Processor: Intel Core i3 or higher
- RAM: 4 GB or above
- Hard Disk: 80 GB or above
- Operating System: Windows or Linux
- Software Requirements:
    - Front-end: Java, Python (Django), or PHP
    - Back-end: MySQL or PostgreSQL
    - Web Server: Apache or XAMPP

Since all required resources are easily available and compatible with modern hardware, the project is technically feasible.

2. Economic Feasibility:
The cost of developing the system is minimal compared to the benefits it offers. The system will reduce

manual efforts in maintaining records, prevent stock mismatches, and improve operational efficiency.
The expenses mainly include:

- Software setup and development costs
- System maintenance and periodic upgrades
- Minimal training for staff

As the system saves time and increases accuracy, it proves to be economically feasible.

3. Operational Feasibility:
The system is designed for easy operation, even by users with limited technical knowledge. It provides a simple graphical interface for stock entry, sales updates, and report generation.
Features like search filters, alert messages, and reports make it convenient for users to operate.
The development using Java and MySQL (or Python Django framework) ensures the system is platform-independent, scalable, and user-friendly. Therefore, it is operationally feasible.

Process Model:
Agile Model
The Agile Software Development Model is chosen for the Stock Management System because it supports iterative, incremental, and flexible development.

Reasons for Choosing Agile:

1. Iterative and Incremental Development:
   Enables continuous enhancement through regular user feedback.
2. Flexibility to Changing Requirements:
   Allows the inclusion of new features like barcode scanning or supplier dashboards as requirements evolve.
3. Frequent User Involvement:
   Ensures the developed system meets the expectations of store managers and staff.
4. Continuous Integration and Testing:
   Helps identify and fix errors early, improving system reliability and performance.
5. Rapid Delivery of Value:
   Core modules (stock entry, reports) can be released early for immediate use.
6. Adaptability to Risks:
   Agile helps in managing risks like data inconsistency and system downtime effectively.
7. Cross-Team Collaboration:
   Developers, testers, and business analysts work closely to ensure seamless development.
8. Customer Satisfaction:
   Regular feedback cycles ensure the system is user-friendly and meets business goals.

UML Diagram Classification

Behavior Diagrams

These represent the dynamic behavior of the system and interactions between users and processes.

- Use Case Diagram: Shows main user interactions such as Add Stock, Update Stock, View Report, and Generate Invoice.
- Activity Diagram: Illustrates the flow of stock operations such as adding new items or checking stock availability.
- State Machine Diagram: Represents transitions of a product from "In Stock" → "Low Stock" → "Out of Stock".
- Sequence Diagram: Describes message flow between user interface, database, and system modules.
- Communication Diagram: Highlights object interaction for stock updates.
- Interaction Overview Diagram: Provides a high-level overview of process flows.
- Timing Diagram: Captures the timing of updates during sales or restocking operations.

Structure Diagrams

These represent the static structure of the system, including relationships between various system components.

- Class Diagram: Shows classes like Product, Supplier, Stock, User, and Transaction, with their attributes and relationships.
- Object Diagram: Displays instances of these classes at runtime.
- Component Diagram: Illustrates software components and their dependencies (Frontend, Backend, Database).
- Composite Structure Diagram: Shows how components interact internally within the system.

Result:

Thus, the problem statement for the Stock Management System has been designed,
a detailed feasibility study has been carried out, and the Agile Process Model has been finalized for the successful implementation of the project.

TASK - 2	Software Requirement Specification Document
DATE	

1. Introduction

1.1 Purpose
The purpose of this document is to describe the Software Requirements Specification (SRS) for the Stock Management System (SMS). It provides a detailed explanation of the functional and non-functional requirements that define the operation of the system.
This document serves as a reference for developers, testers, and clients, ensuring that all stakeholders share a common understanding of the system's capabilities and limitations.

1.2 Scope
The Stock Management System is designed to automate inventory operations for small and medium-scale businesses. The system will manage products, suppliers, purchase orders, stock updates, and sales records. It allows users to monitor stock levels, track product movements, and generate analytical reports. Administrators can add, update, and delete products, while employees can view and record transactions. The system aims to minimize manual errors, enhance decision-making, and maintain accurate, real-time stock information.

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
Admin	The administrator who manages products, suppliers, and stock records.
User (Staff)	Authorized staff member responsible for daily stock transactions and report viewing.
Database	A centralized data storage system maintaining all stock, product, and supplier details.
SMS	Stock Management System.

1.4 References
1. www.google.com
2. www.wikipedia.org
3. IEEE Software Requirement Specification Standard (IEEE Std 830-1998)

1.5 Overview
- Chapter 1: Introduces the purpose and scope of the Stock Management System.
- Chapter 2: Describes the system's functionalities, user roles, and constraints.
- Chapter 3: Defines functional, non-functional, and interface requirements.
- Chapter 4: Provides appendices including definitions and database schema.

2. System Description
2.1 Product Perspective
The Stock Management System replaces traditional manual inventory tracking with an automated, efficient platform. It reduces paperwork, prevents overstocking and stockouts, and improves operational transparency.
The system provides search and filter features for products and suppliers, and it allows the admin to manage purchases, sales, and stock levels easily. The system requires a stable computer with internet access and a database server for record storage.

2.2 Product Functions
The Stock Management System includes the following major functions:
- User Authentication: Secure login and role-based access for admin and staff.
- Product Management: Add, update, or delete product details such as name, category, quantity,

and price.
- Stock Monitoring: Automatic update of stock levels after every sale or purchase.
- Supplier Management: Maintain supplier contact and purchase history.
- Report Generation: Generate stock, sales, and purchase reports.
- Alert System: Notify admin when product stock falls below a specified threshold.

2.3 User Functions

a) Administrator

- Add, edit, and delete products and suppliers.
- View, update, and verify stock quantities.
- Generate reports such as sales summaries, stock usage, and reorder alerts.
- Manage system users and assign permissions.

b) Staff/User

- View available products and their quantities.
- Record stock-in (purchases) and stock-out (sales) transactions.
- View product details and supplier information.
- Access limited reports related to inventory movements.

2.4 System Constraints

- Hardware Limitations: The system must run on standard hardware (minimum 4GB RAM, 80GB HDD).
- Compatibility: Must work with popular web browsers (Chrome, Firefox, Edge).
- Internet Dependence: Internet connection required for remote database access.
- Security Compliance: Must adhere to data protection and access control standards.
- Budgetary Constraints: The system must be cost-effective with minimal maintenance requirements.
- Backup Requirement: Periodic backups are mandatory to prevent data loss.

2.5 System Dependencies

- Relies on a database system (MySQL or PostgreSQL) for data storage.
- Depends on a web server (Apache or XAMPP) for online functionality.
- Requires network connectivity for real-time updates and remote access.
- May use external APIs for reporting or analytics.
- Integration with existing authentication systems (if any) may be required.

2.6 Requirements Subdomain

This system is intended for businesses, warehouses, and retail stores where inventory tracking, supplier management, and sales reporting are critical.

3. Specific System Requirements

3.1 Functional Requirements

3.1.1 User Management

- Users can register and log in using unique credentials.
- The system validates credentials and grants access based on roles.

3.1.2 Product Management

- Admin can add, modify, or delete products from inventory.
- Each product entry includes ID, name, category, quantity, cost price, and selling price.

3.1.3 Supplier Management

- Admin can add and update supplier information (name, contact, address).
- Track supplier performance and transaction history.

3.1.4 Stock Transactions

- Users can record product purchases (stock-in) and sales (stock-out).
- The system automatically updates product quantity after each transaction.

3.1.5 Reporting
- The system generates daily, weekly, and monthly reports on stock movement and sales.
- Reports can be exported in PDF or Excel format.

3.1.6 Alert Notification
- The system notifies the admin when a product reaches the minimum threshold quantity.

3.2 Non-Functional Requirements

3.2.1 Performance
- The system should support at least 500 concurrent users.
- Database queries should execute within 2 seconds under normal conditions.

3.2.2 Safety
- Regular backups must be performed to prevent data loss.
- The system should have recovery options for accidental deletions.

3.2.3 Usability
- The interface must be simple, intuitive, and responsive.
- The system should be accessible to users with basic computer skills.

3.2.4 Security
- Data encryption must be used for sensitive information (login credentials, stock data).
- Access controls must prevent unauthorized modifications.

3.3 External Interfaces

3.3.1 User Interface
- The homepage should display summary details like total stock, low-stock alerts, and quick access buttons.
- Product pages must show details like category, quantity, and supplier.
- Admin dashboard should allow navigation to reports, transactions, and alerts.

3.3.2 Hardware Interface
- Compatible with computers, laptops, and mobile devices.
- Barcode scanners can be used for quick stock check-ins and check-outs.
- Requires stable network connectivity.

3.3.3 Software Interface
- Frontend: Java, Python (Django), or PHP
- Backend: MySQL or SQL Server
- Tools: StarUML or ArgoUML for modeling
- Office Tools: MS Office for documentation and reports.

3.3.4 Communication Interface
- HTTP/HTTPS protocols for communication between client and server.
- Email or SMS notifications for low-stock alerts.
- Regular technical meetings for updates and feedback collection.

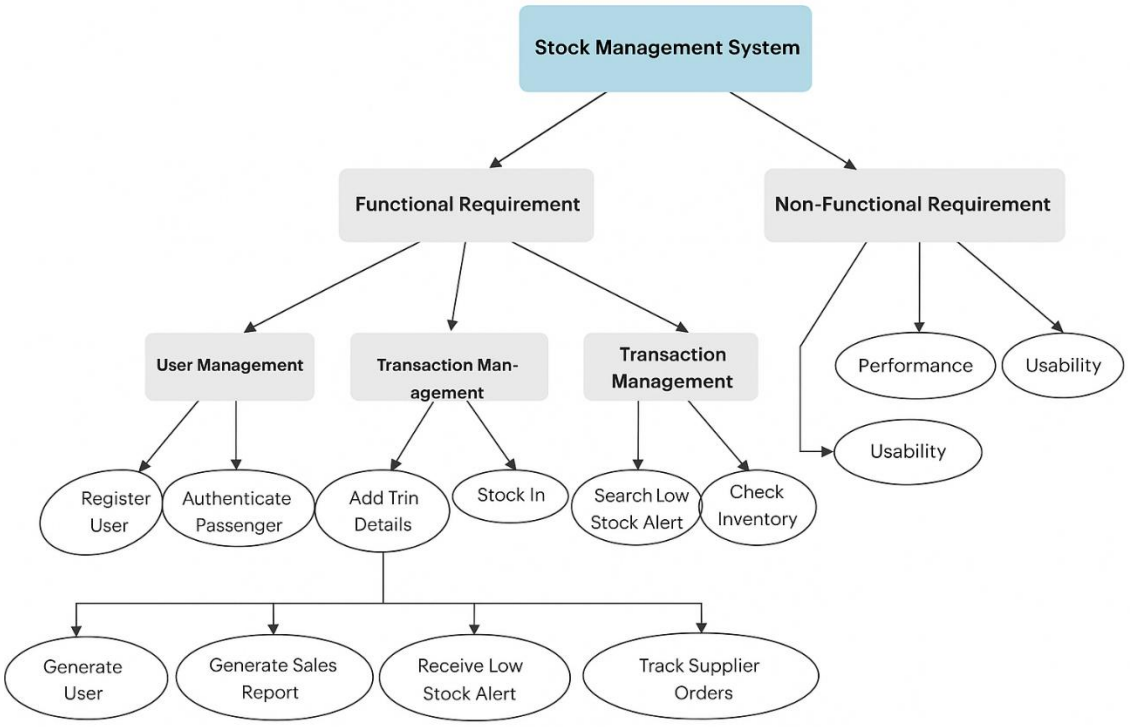
4. Appendices

Appendix A: Definition

Contains definitions of system terms such as Product, Supplier, Stock, Transaction, and User Roles.

Appendix B: Database Schema

Defines database tables such as:
- tbl_product (product_id, name, category, quantity, price)
- tbl_supplier (supplier_id, name, contact, address)
- tbl_transaction (transaction_id, product_id, type, date, quantity)
- tbl_user (user_id, username, role, password)



Result

Thus, the Software Requirement Specification (SRS) document for the Stock Management System has been successfully prepared, detailing all functional, non-functional, and interface requirements, along with dependencies and constraints.

TASK – 3	**To Prepare a detailed estimate using the FP and COCOMO Model. Also prepare a detailed schedule of the project.**
DATE	

Aim:
To perform the estimation (Function Point and COCOMO) and to develop a project schedule for the **Stock Management System**.

FP Model:
The functional size of the Stock Management System is measured in terms of the **Function Point (FP)**, which serves as a standard metric to quantify software functionality.

Step 1:
F = 14 × scale
The **scale** varies from 0 to 5 according to the level of influence of each **Complexity Adjustment Factor (CAF)**:

Scale	Description
0	No Influence
1	Incidental
2	Moderate
3	Average
4	Significant
5	Essential

Step 2:
CAF = 0.65 + (0.01 × F)

Step 3:
Calculate Unadjusted Function Point (UFP) using the following table:

Function Units	Low	Avg	High
EI (External Inputs)	3	4	6
EO (External Outputs)	4	5	7
EQ (External Inquiries)	3	4	6
ILF (Internal Logical Files)	7	10	15
EIF (External Interface Files)	5	7	10

Given Data for Stock Management System:

Parameter	Value
Number of user inputs	24
Number of user outputs	46
Number of inquiries	8
Number of files	4
Number of external interfaces	2
Effort	36.9 person-months
Technical documents	265 pages
User documents	122 pages
Cost	$7744 / month

Step 4: Calculation of UFP
Since all weighting factors are **average**:
UFP = (24 × 4) + (46 × 5) + (8 × 4) + (4 × 10) + (2 × 7)
UFP = 96 + 230 + 32 + 40 + 14 = **412**

Step 5: Calculate Function Point (FP)
Assuming **average complexity**,
F = 14 × 3 = 42
CAF = 0.65 + (0.01 × 42) = 1.07
FP = UFP × CAF = 412 × 1.07 = **440.84 ≈ 441 FP**

Step 6: Productivity, Documentation, and Cost

Metric	Formula	Result
Productivity	Effort / FP	36.9 / 441 = **0.0836 PM/FP**
Documentation	(265 + 122) / FP	387 / 441 = **0.877 pages/FP**
Cost per Function	(Cost × Effort) / FP	(7744 × 36.9) / 441 = **$648.32 per FP**

COCOMO Model
Estimated project size = **400 KLOC**
Basic COCOMO equations:
Effort (E) = a × (KLOC)^b person-months
Development Time (D) = c × (Effort)^d months

(i) Organic Mode
E = 2.4 × (400)^1.05 = **1295.31 PM**
D = 2.5 × (1295.31)^0.38 = **38.07 months**

(ii) Semi-Detached Mode
E = 3.0 × (400)^1.12 = **2462.79 PM**
D = 2.5 × (2462.79)^0.35 = **38.45 months**

(iii) Embedded Mode
E = 3.6 × (400)^1.20 = **4772.81 PM**
D = 2.5 × (4772.8)^0.32 = **38 months**

Project Scheduling
A **project schedule** for the Stock Management System outlines the sequence of development activities, estimated durations, and resource allocation.

Phase	Description	Duration (Weeks)
1. Requirement Analysis	Collect and document functional/non-functional needs	2
2. System Design	Create architecture, ER diagrams, and data flow models	3
3. Implementation	Develop modules for stock entry, monitoring, billing	6
4. Integration and Testing	Integrate modules and test functionalities	3

Phase	Description	Duration (Weeks)
5. Documentation	Prepare technical and user manuals	2
6. Deployment and Maintenance	Install system and provide updates	2

Total Duration: ≈ 18 weeks (≈ 4.5 months)

Result:
Thus, the estimation using **Function Point (FP)** and **COCOMO** models has been successfully performed, and a detailed **project schedule** for the **Stock Management System** has been prepared.

TASK 4:USE CASE DIAGRAM

AIM:
The aim of this system is to streamline product management by enabling the administrator to manage product details, purchases, sales, and stock, while also allowing suppliers to provide products and customers to purchase them.

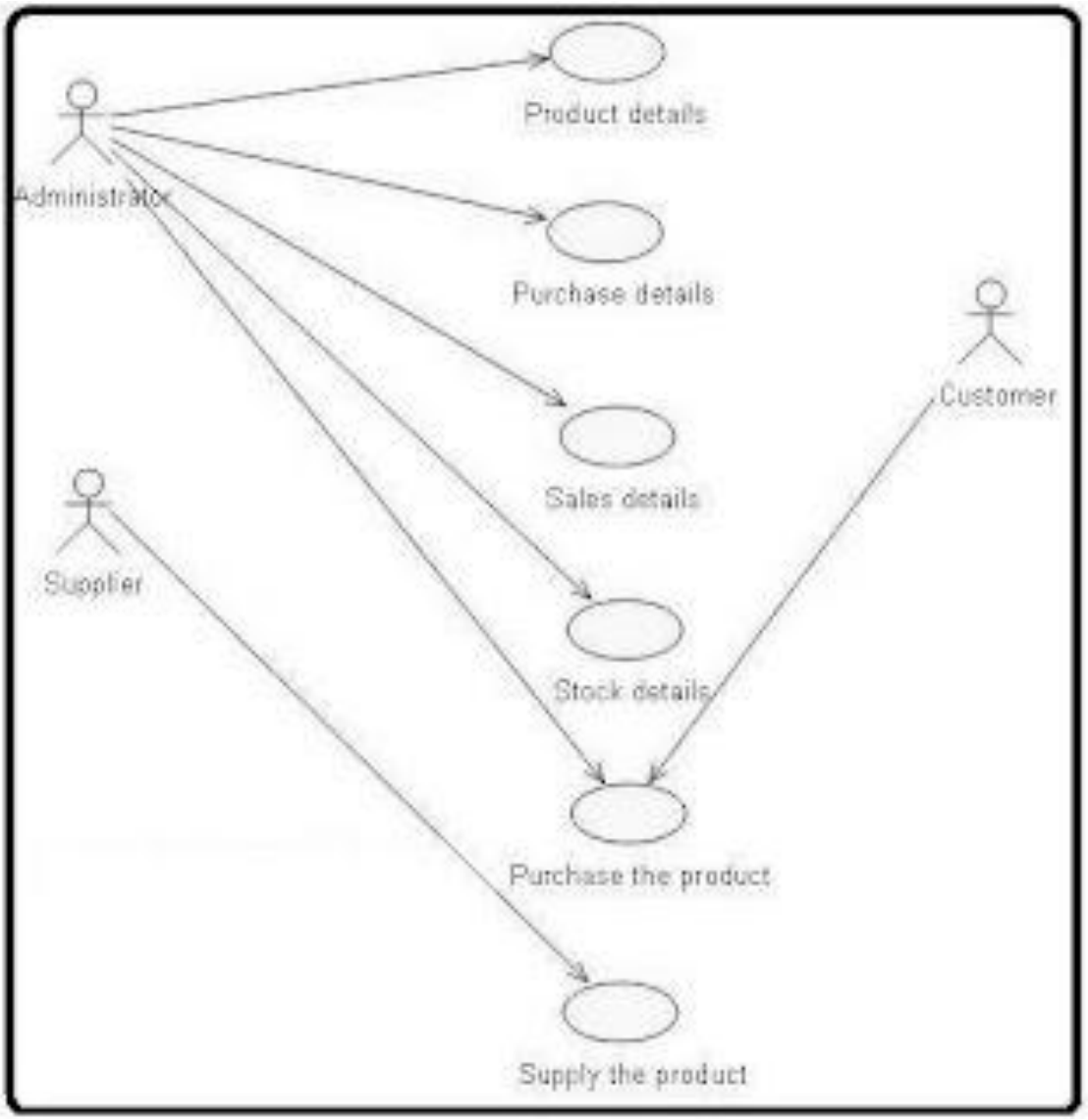
2. Problem Statement
Managing products, purchases, sales, and stock manually often leads to inefficiency, errors, and poor coordination among administrators, suppliers, and customers. This system addresses the problem by creating a centralized and structured way to handle all product-related operations in one place.

3. Key Components

- Actors
    - Administrator: Manages product details, purchase details, sales details, and stock details.
    - Supplier: Supplies products to the system.
    - Customer: Purchases products.
- Use Cases
    - Product details
    - Purchase details
    - Sales details
    - Stock details
    - Purchase the product
    - Supply the product

4.Procedure
To create the use case diagram, the first step is to identify the main actors who interact with the system, such as the administrator, supplier, and customer. Next, the system's functional requirements are gathered, which include managing product details, purchase details, sales records, and stock levels. Each requirement is then represented as a use case, showing the

actions the system must perform. The relationships between actors and use cases are drawn by connecting each actor to the functions they are responsible for; for example, the administrator manages product, purchase, sales, and stock details, the supplier provides products, and the customer purchases products. Finally, the diagram is refined to ensure clarity, showing the overall interactions between the actors and the system in a simple, visual form.



4. Result

The system ensures accurate and up-to-date management of products, purchases, sales, and stock while improving coordination between administrators, suppliers, and customers. As a result, operations become more efficient, reliable, and seamless.

TASK 5: CLASS DIAGRAM

1. Aim

The aim of this system is to represent the static structure of product management by defining the main classes—Product, Purchase, Sales, and Stock—along with their attributes, methods, and relationships. This helps in visualizing how different components of the system interact with each other.

2. Problem Statement

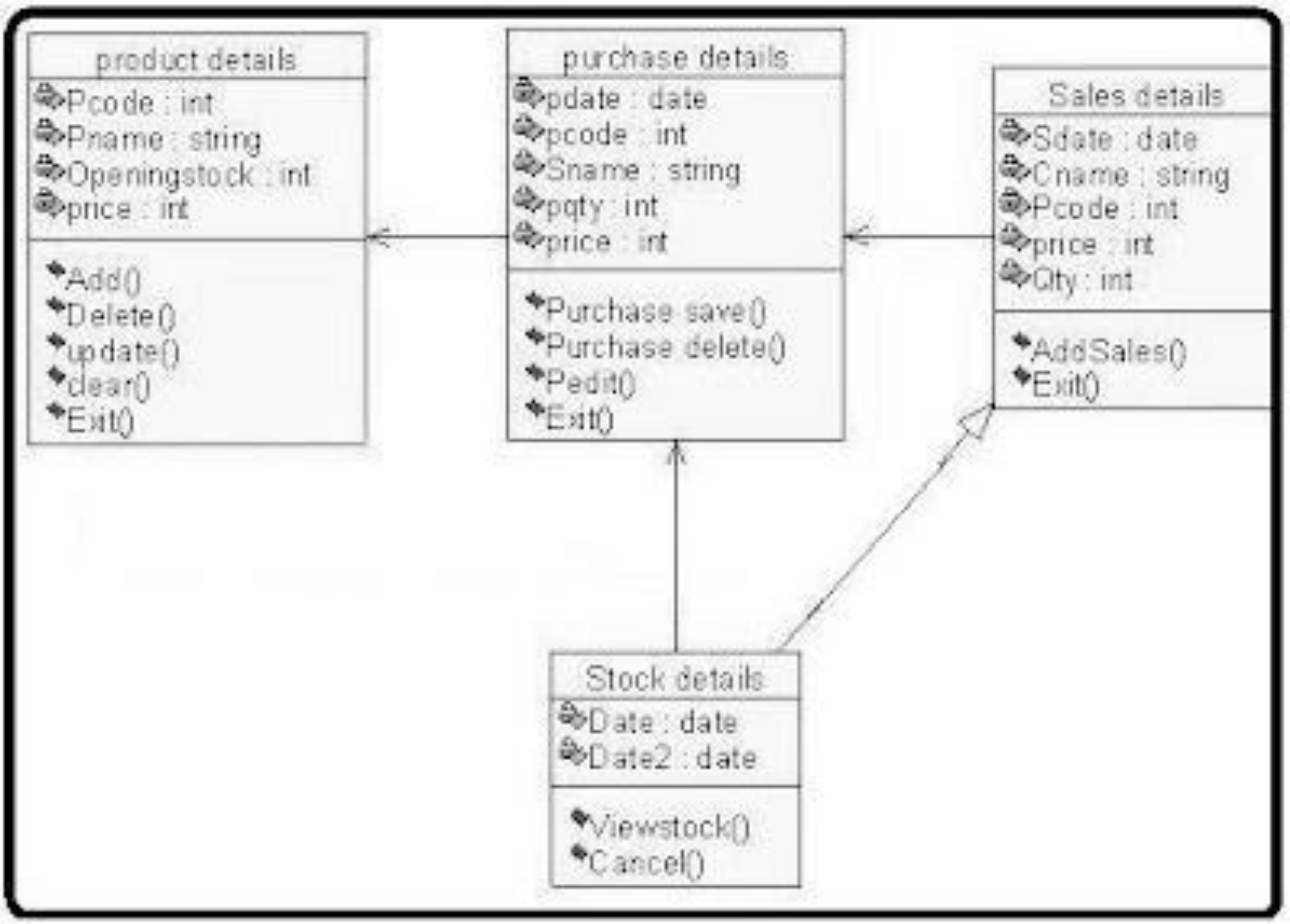
In product management, handling details of products, purchases, sales, and stock without a clear system structure can lead to confusion and inefficiency. A class diagram provides a well-defined blueprint that organizes data and functionality into classes, ensuring consistency in system design and smooth interaction between different entities.

3. Key Components

- Classes and Attributes
    - Product Details: Attributes include Pcode, Pname, OpeningStock, Price. Methods include Add(), Delete(), Update(), Clear(), Exit().
    - Purchase Details: Attributes include Pdate, Pcode, Pname, Pqty, Price. Methods include PurchaseSave(), PurchaseDelete(), Edit(), Exit().
    - Sales Details: Attributes include Sdate, Cname, Pcode, Pqty. Methods include AddSales(), Exit().
    - Stock Details: Attributes include Date, Date2. Methods include ViewStock(), Cancel().
- Relationships
    - Product Details are connected with both Purchase and Sales Details.
    - Purchase Details and Sales Details update Stock Details.

4. Procedure

To create the class diagram, the first step is to identify the main entities of the system, such as Product Details, Purchase Details, Sales Details, and Stock Details. Next, define the attributes (data members) for each class to represent the stored information, and methods (functions) to represent actions performed by the class. Then, establish the relationships between the classes—for example, purchase and sales activities affect stock, and product details are linked to both purchases and sales. Finally, refine the diagram to make sure it clearly conveys the system's structure, ensuring it is easy to understand and implement during development.



5. Result

The class diagram provides a structured view of the product management system, showing how products, purchases, sales, and stock are interrelated. This helps in maintaining clarity in design, ensuring smooth system implementation, and supporting efficient product management operations.

TASK 6: SEQUENCE DIAGRAM

1. Aim

The aim of this sequence diagram is to illustrate the dynamic interaction between the administrator, supplier, customer, and system components (product details, purchase details, sales details, stock details) over time. It shows the order of messages exchanged to complete a process, such as purchasing and supplying a product.

2. Problem Statement

While the use case diagram shows what the system should do and the class diagram shows the static structure, neither explains how the interactions occur step by step in real time. Without this, the flow of activities can remain unclear. The sequence diagram solves this by providing a clear, time-ordered view of how system components and actors collaborate to achieve specific tasks like product purchase, sales, or stock updates.

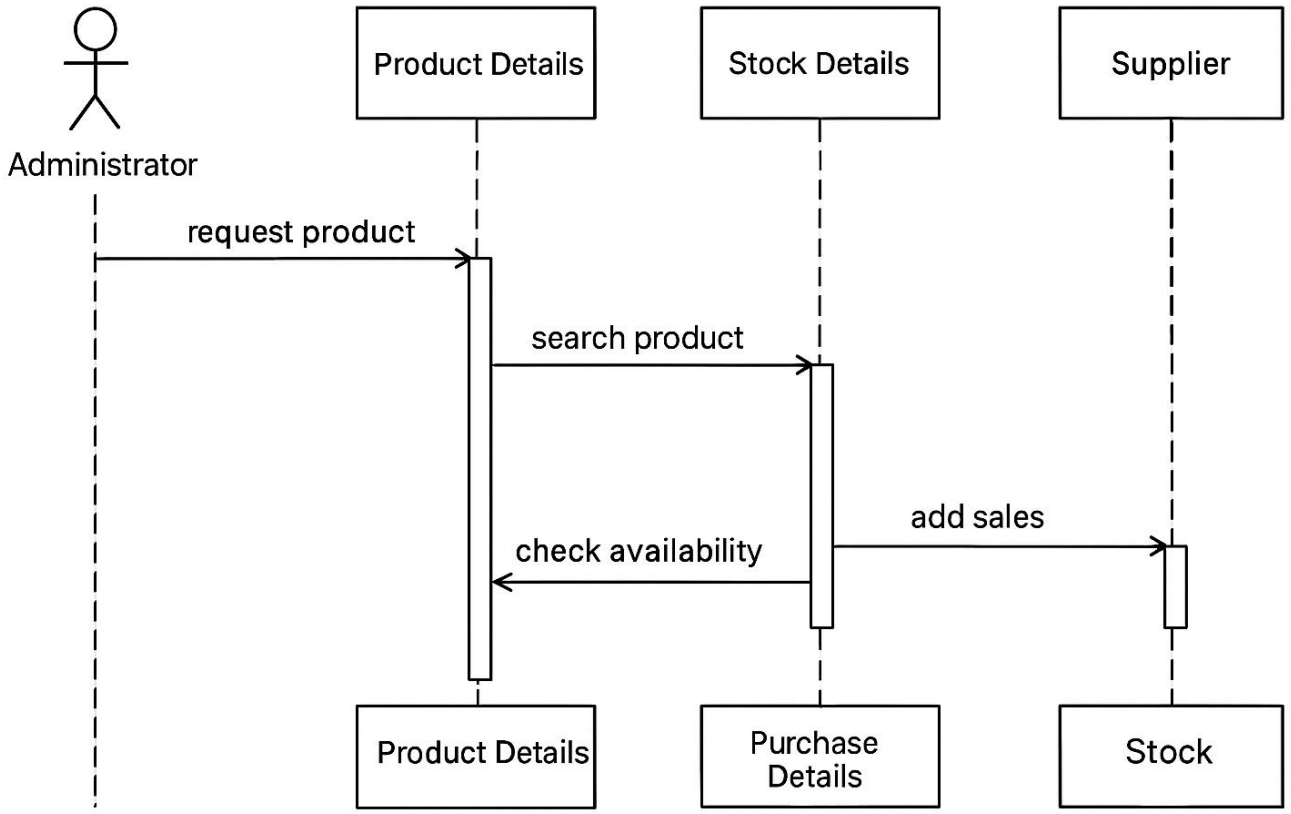
3. Key Components

- Actors
    - Administrator
    - Supplier
    - Customer
- Objects/Entities (from class diagram)
    - Product Details
    - Purchase Details
    - Sales Details
    - Stock Details
- Interactions
    - Administrator adds/manages product details.
    - Supplier supplies products (updates purchase and stock details).
    - Customer purchases product (updates sales and stock details).
    - Stock is updated after every purchase or sale.

4. Procedure

To create the sequence diagram, the first step is to identify a main scenario, such as “Customer purchases a product.” Then, determine the participating actors and system components (customer, administrator, product, purchase, sales, and stock details). Next, draw lifelines for

each actor/object and represent the flow of messages between them, showing the order in which methods are called (e.g., search product → check stock → purchase save → update stock → add sales). Finally, refine the sequence diagram to show a clear, time-ordered process from initiation to completion.



5. Result

The sequence diagram provides a step-by-step visual of how the system works dynamically. It clearly shows the order of interactions between administrator, supplier, customer, and system components, ensuring smooth communication and better understanding of the system's real-time behavior.

TASK 7: INTERACTION DIAGRAM

1. Aim

The aim of this interaction diagram is to represent how objects (such as Product, Purchase, Sales, and Stock details) communicate with each other and with actors (Administrator, Supplier, Customer) to accomplish specific tasks in the product management system. Unlike the sequence diagram which focuses on time order, the interaction diagram highlights the structural organization of messages exchanged between objects.

2. Problem Statement

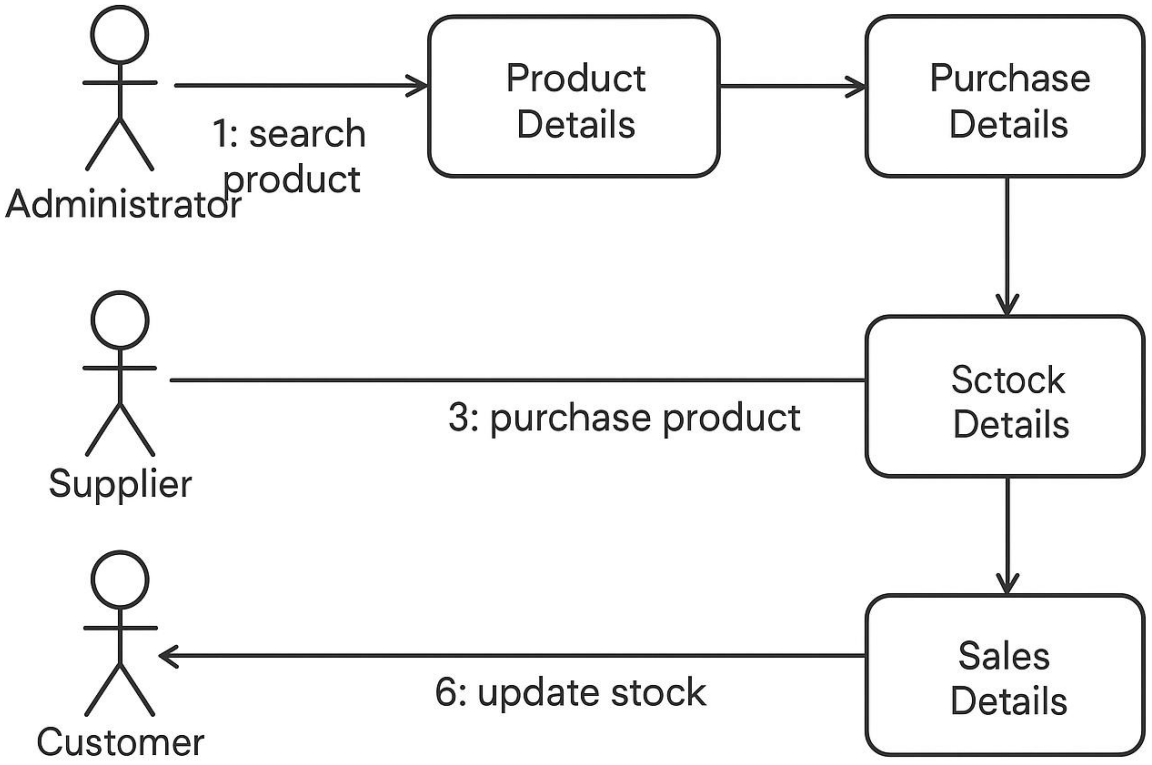
Although the sequence diagram shows the order of events, it does not emphasize the network of relationships among objects. In product management systems, understanding how different entities interact and collaborate is crucial for maintaining clarity in system design. The interaction diagram addresses this by visually representing the communication links and message flows among objects, ensuring the system's operations are fully understood from both behavioral and structural perspectives.

3. Key Components

- Actors
    - Administrator
    - Supplier
    - Customer
- Objects/Entities
    - Product Details
    - Purchase Details
    - Sales Details
    - Stock Details
- Interactions
    - Administrator requests product details and manages sales/purchases.
    - Product Details communicates with Stock Details to verify availability.
    - Supplier interacts with Purchase Details to update stock.
    - Customer interacts with Sales Details to complete a purchase.
    - Stock is updated accordingly after each purchase or supply operation.

4. Procedure

To create the interaction diagram, the first step is to identify a primary use case (e.g., product purchase). Next, determine the participating objects (Administrator, Supplier, Customer, Product, Purchase, Sales, and Stock). Then, map out the communication links between them to show which objects need to send or receive messages. After that, assign message numbers to indicate the order of communication (e.g., 1: search product, 2: check stock, 3: purchase product, 4: update stock). Finally, refine the diagram to clearly represent both the relationships and the flow of communication.



5. Result

The interaction diagram provides a clear visualization of how different system entities and actors collaborate by exchanging messages. It ensures better understanding of system behavior, improves system design, and makes it easier to trace how tasks such as purchasing, supplying, and stock management are carried out within the product management system.

TASK 8: ACTIVITY DIAGRAM

1. Aim

The aim of this activity diagram is to represent the workflow of entering product details into the system. It shows the step-by-step activities and decision points, ensuring that the process is clearly defined and free from ambiguity.

2. Problem Statement

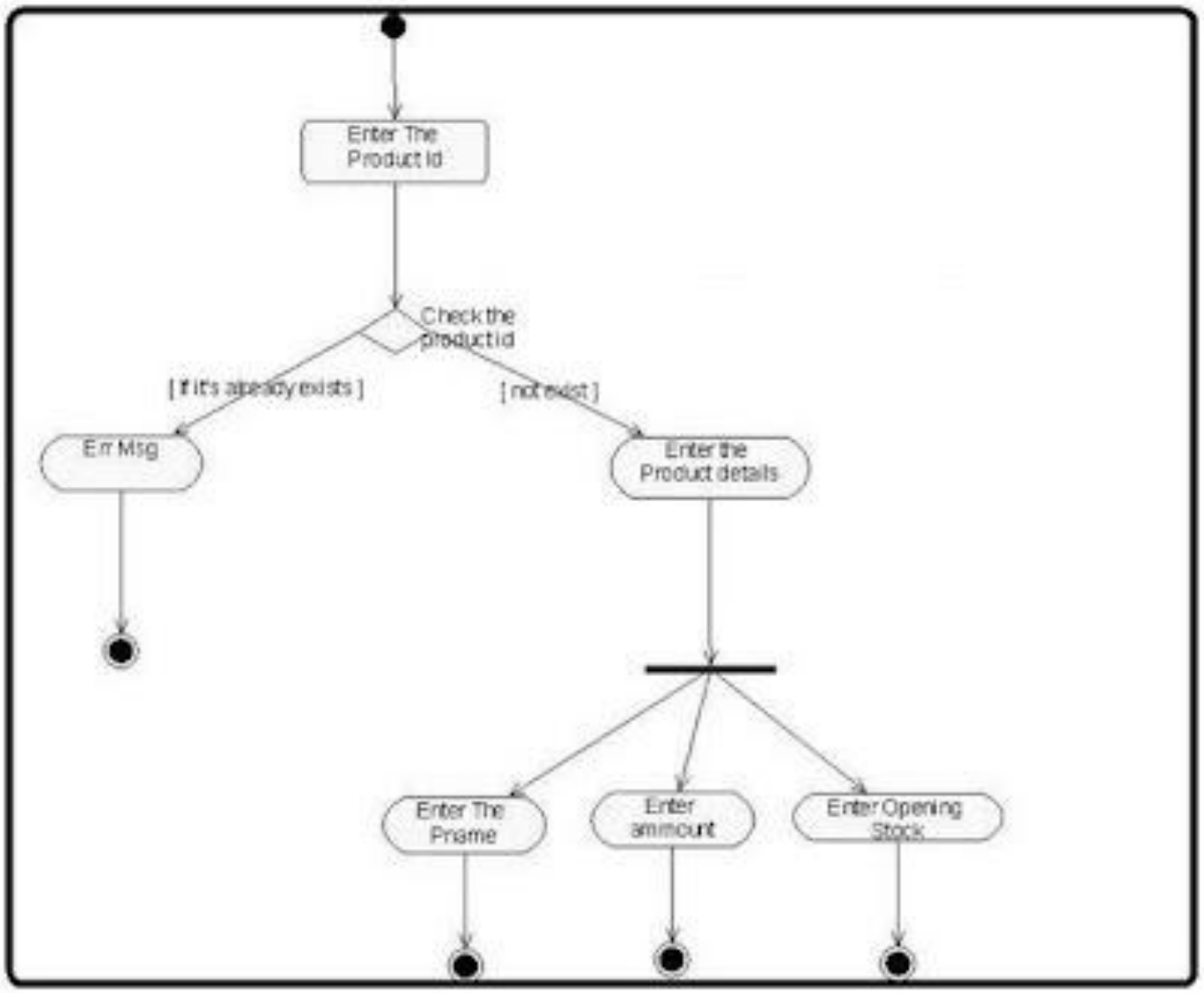
In product management systems, entering product details without a structured process can lead to duplicate entries, missing data, or errors in stock management. An activity diagram provides a visual model of the process, including decision points and actions, ensuring accuracy and consistency while adding product details.

3. Key Components

- Activities
    - Enter the Product ID
    - Check the Product ID
    - Enter the Product Details (name, amount, opening stock)
    - Display Error Message (if duplicate ID)
- Decisions
    - If the product ID already exists → Show error message.
    - If the product ID does not exist → Proceed to enter product details.
- Start and End Points
    - Black filled circle → Start of the process.
    - Black concentric circles → End of the process.

4. Procedure

To create the activity diagram, the first step is to identify the workflow for adding a product to the system. Begin with the initial activity, “Enter the Product ID.” Then, add a decision node to check whether the product ID already exists. If it exists, the process ends with an error message. If it does not exist, the workflow proceeds to entering product details, which includes entering the product name, amount, and opening stock. These parallel activities are represented as concurrent actions. Finally, the process ends after successfully adding all details.



5. Result

The activity diagram provides a clear, visual representation of the process for entering product details, highlighting decision points and possible outcomes. It ensures that the system prevents duplicate entries and requires all necessary information to be entered, making product management more reliable and efficient.

TASK 9: PACKAGE DIAGRAM

1. Aim

The aim of this package diagram is to organize the system into meaningful groups (packages), making it easier to manage large and complex structures in product management. It shows dependencies and relationships between different parts of the system.

2. Problem Statement

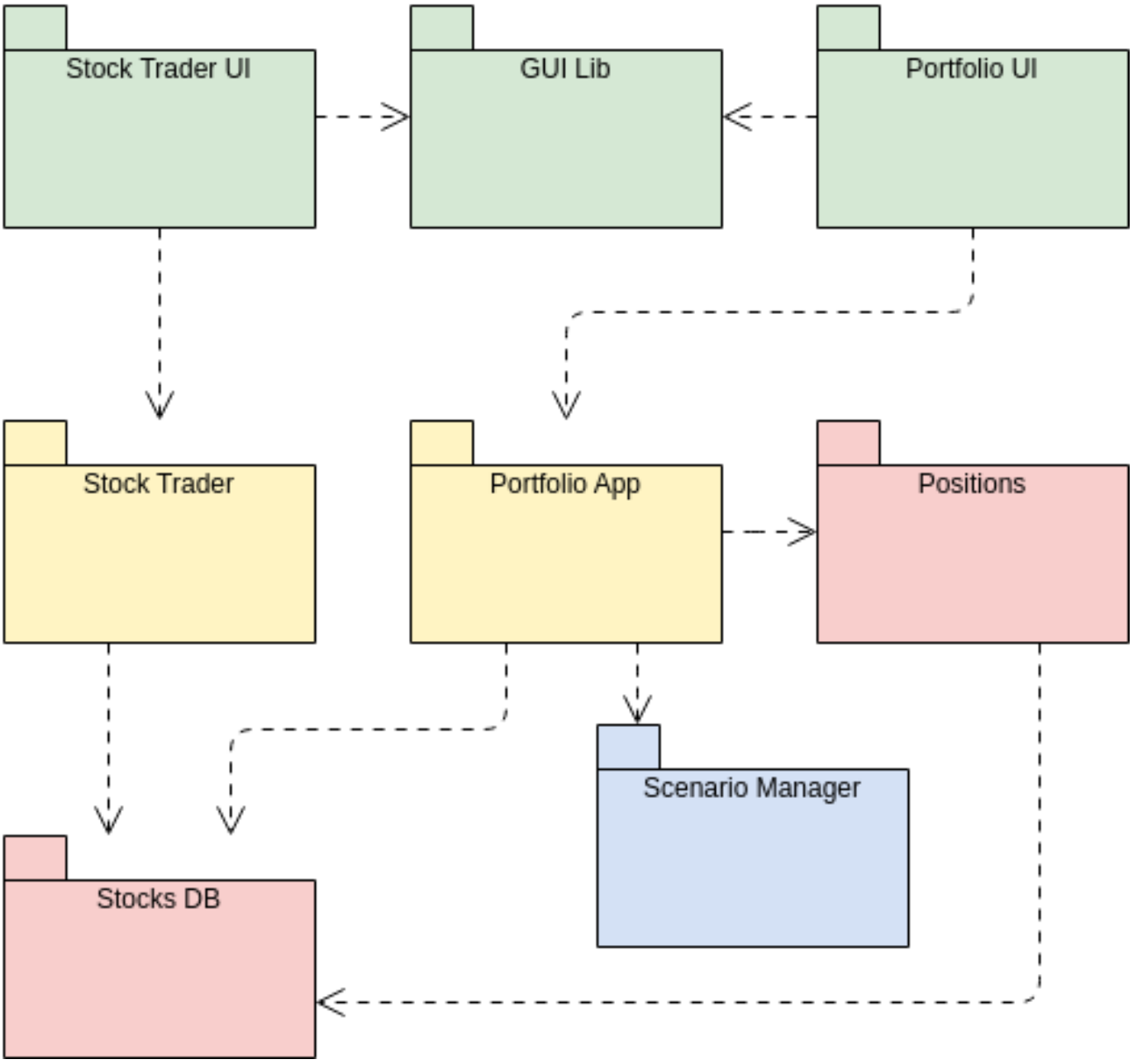
When a system grows, managing all classes and components in a single view becomes difficult. Without proper grouping, the design can become confusing and hard to maintain. A package diagram solves this problem by logically grouping related classes into packages, reducing complexity and improving modularity.

3. Key Components

- Packages
    - Product Management Package: Contains Product Details.
    - Purchase Management Package: Contains Purchase Details.
    - Sales Management Package: Contains Sales Details.
    - Stock Management Package: Contains Stock Details.
- Dependencies
    - Purchase Management depends on Product Management (to link product codes).
    - Sales Management depends on both Product and Stock Management.
    - Stock Management depends on updates from Purchase and Sales Management.

4. Procedure

To create the package diagram, first identify logical groupings of classes from the class diagram (such as product, purchase, sales, and stock). Next, encapsulate these classes inside packages with meaningful names. Then, establish dependency relationships among packages — for example, Sales depends on Product and Stock, while Purchase updates Stock. Finally, refine the diagram for clarity, ensuring it reflects the modular structure of the system.



5. Result

The package diagram provides a high-level view of the system's modular structure. It improves understanding, reduces complexity, and makes system maintenance easier by showing how the system is divided into logical groups and how those groups depend on one another

TASK 10: COMPONENT DIAGRAM

1. Aim

The aim of this component diagram is to show the physical components of the system and how they interact to provide functionality, such as login, stock management, and database access.

2. Problem Statement

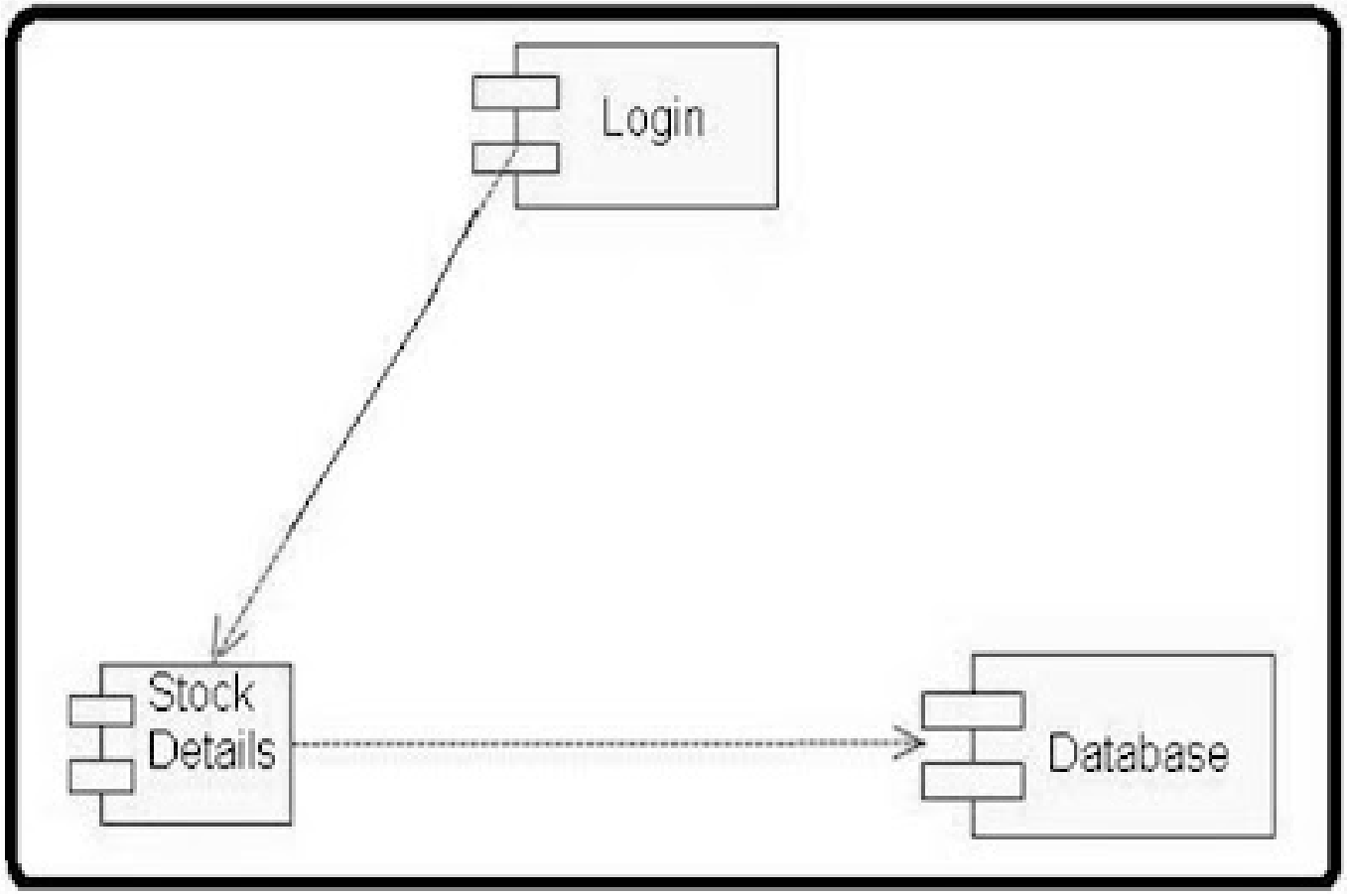
In complex systems, it can be difficult to visualize the actual software components and how they connect. A component diagram solves this by showing the system's high-level building blocks (like login module, stock management, and database) and the relationships between them.

3. Key Components

- Components
    - Login: Manages user authentication.
    - Stock Details: Manages stock operations.
    - Database: Stores and retrieves product, purchase, sales, and stock information.
- Dependencies
    - Login component interacts with Stock Details.
    - Stock Details component communicates with Database to fetch or update records.

4. Procedure

To create the component diagram, first identify the system's main software components (login, stock details, database). Next, define the interfaces each component exposes or requires. Then, draw dependency relationships between them, for example, login depends on stock details, and stock details depends on the database. Finally, refine the diagram for clarity.



5. Result

The component diagram provides a clear understanding of the high-level architecture of the system, showing how modules like login and stock details interact with the database. This improves system maintainability and modular design.

TASK 11: DEPLOYMENT DIAGRAM

1. Aim

The aim of this deployment diagram is to show how the system's software components (reports, purchase, sales, stock) are deployed on hardware nodes, such as a printer, to generate reports.

2. Problem Statement

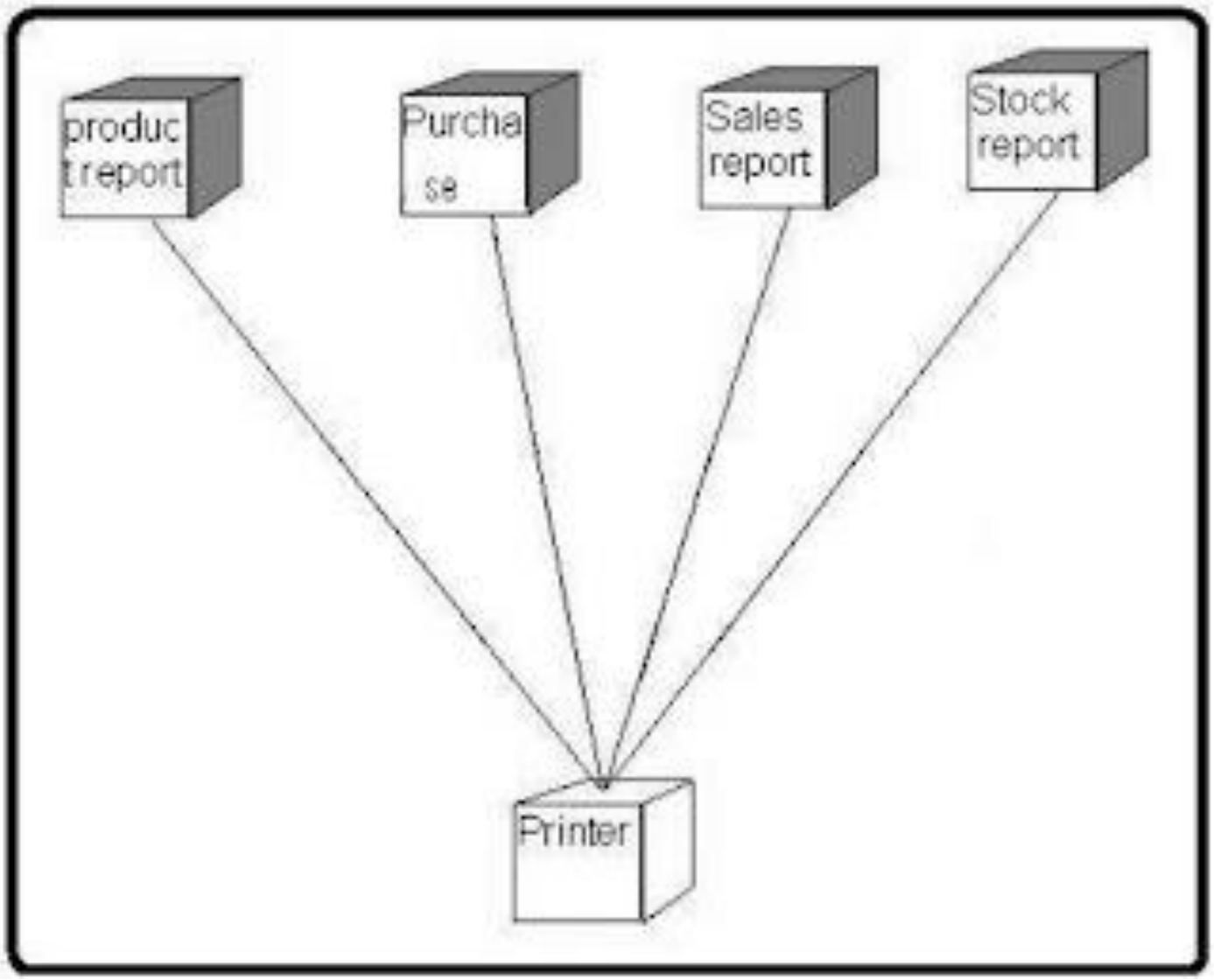
In product management, generating reports (product, purchase, sales, stock) requires mapping software elements to physical hardware. Without a deployment diagram, the actual hardware-software relationship remains unclear.

3. Key Components

- Nodes
    - Printer: Hardware device for report generation.
- Artifacts (Software Components Deployed)
    - Product Report
    - Purchase Report
    - Sales Report
    - Stock Report

4. Procedure

To create the deployment diagram, first identify the physical hardware node involved (printer). Next, determine the software artifacts that will be deployed onto this node, such as product, purchase, sales, and stock reports. Then, draw deployment relationships from the artifacts to the printer node.



5. Result

The deployment diagram clearly shows how software artifacts (reports) are mapped to the hardware (printer). This helps in understanding the physical deployment of the system and ensures smooth integration of reporting features with the system's hardware.