Ex No: 1	IDENTIFY A SOFTWARE SYSTEM THAT NEEDS TO BE DEVELOPED-EBOOK MANAGEMENT SYSTEM

Aim:

To identify a software system that needs to be developed - eBook management system.

Introduction:

In the digital age, where electronic books (eBooks) have become increasingly popular, managing one's eBook collection efficiently is paramount. An eBook Management System Website offers a comprehensive solution for users to organize, access, and enjoy their eBooks seamlessly. This introduction outlines the significance of such a website and its key features.

User:

Represents users of the system who interact with the EBook Management System. Users can register, login, search for an eBook, download a eBook, add reviews, and rate the books. They can also interact with the administrator for the registration purposes.

Administrator:

Represents the administrator of the system who manages the system by interacting with the database and the users. An administrator can login to the system with their own credentials, add or remove an eBook, and various other functions that manages the database and resolves issues within the system.

EBook:

Represents electronic books stored in the database of the system. Each eBook has attributes such as title, author, genre, etc. Users can read or download an eBook as they see fit.

Database:

Represents the storage mechanism for storing and managing eBooks and user data. Responsible for operations such as adding, removing, updating, searching eBooks, and managing user accounts.

Interface for downloads:

The books can be downloaded into the user's devices and the user can decide which file format the book needs to be in.

Ratings and Reviews:

The users can and rate and review a book. This feature helps other users determine the quality of the book and recommend it to others or warn other users of its short comings.

Cross Platform Support:

Develop the website to be responsive and accessible across various devices.

Architecture of eBook Management System:

Client-Server Architecture:

- Implement a client-server architecture where the web browser acts as the client, and the Server hosts the application logic, data, and services.
- Use RESTful APIs for communication between the client and server to ensure scalability and maintainability.

Database Design:

NoSQL Database Schema:

Design a noSql database schema to store user profiles, eBook metadata (title, author, description), user preferences, reading history, and relationships (e.g. user-book, book- genre).

Normalization:

Normalize the database schema to reduce redundancy and maintain data integrity, ensuring each piece of data is stored in only one place.

Indexes and Query Optimization:

- Create indexes on frequently queried fields to improve database performance.
- Optimize database queries by avoiding unnecessary joins, using appropriate data types, and caching frequently accessed data.

Testing:

Unit Testing:

Write unit tests for individual components, such as controllers, services, and models, to ensure they behave as expected.

Integration Testing:

Perform integration tests to verify interactions between different modules and components of the system.

End-to-End Testing:

Conduct end-to-end tests to simulate user interactions and validate the system's functionality from the user's perspective.

System Testing:

Validate eBook management system functionalities comprehensively to ensure reliability, usability, and performance in real-world scenarios.

Result:

Thus a software system that needs to be developed – the eBook management system- was identified successfully.

Ex No: 2	DOCUMENT THE SOFTWARE REQUIREMENT SPECIFICATION (SRS) FOR THE EBOOK MANAGEMENT SYSTEM

Aim:

To document the Software Requirements Specification (SRS) for the eBook Management system.

Introduction:

The eBook Management System is a software solution designed to streamline the organization, storage, and access of eBooks for users. It provides a centralized platform, where users can catalogue, categorize, and download their eBooks effectively, enhancing their reading experience and productivity. This document outlines the functional and non-functional requirements of the eBook Management System.

Purpose:

The purpose of this SRS document is to define the requirements of the eBook Management System. It serves as a guideline for the development team to understand the scope of the project, implement the desired functionalities, and ensure the system meets the needs of its users.

Problem Statement:

Design and implement a simple eBook management system that allows users to search and download eBooks of various formats. The system also has an administrator role who manages the database that contains the eBooks and the system itself .The system should utilize design patterns to enhance modularity, flexibility, and maintainability.

Scope:

The eBook Management System will allow users to:

- Browse and search for eBooks based on various criteria.
- Read eBooks directly within the application.
- Download the eBooks from the platform into their system.
- Add ratings and reviews to the books.

Product Functions:

- User Account Management: Readers can create accounts to manage their e-book libraries, track the books they have downloaded, and to keep track of their interactions with the books.
- Search and Discovery: Users can easily search for e-books based on genre, author, or keywords, with advanced filtering options for efficient browsing.
- Content Distribution: The system facilitates the distribution of e-books to users through secure downloads.
- Notification System: Users receive notifications via email or SMS regarding new e-book releases, promotions, or account activities.
- Reporting and Analytics: Administrators can generate reports on user engagement, popular genres, and sales trends to inform marketing strategies and content curation.
- Content Security: Robust encryption and Digital Rights Management (DRM) mechanisms are implemented to protect e-book content from unauthorized access and piracy.

User Classes and Characteristics:

User:

Represents users of the system who interact with the EBook Management System. Users can register, login, search for an eBook, download a eBook, add reviews, and rate the books. They can also interact with the administrator for the registration purposes.

Administrator:

Represents the administrator of the system who manages the system by interacting with the database and the users. An administrator can login to the system with their own credentials, add or remove an eBook, and various other functions that manages the database and resolves issues within the system.

EBook:

Represents electronic books stored in the database of the system. Each eBook has attributes such as title, author, genre, etc. Users can read or download an eBook as they see fit.

Database:

Represents the storage mechanism for storing and managing eBooks and user data. Responsible for operations such as adding, removing, updating, searching eBooks, and managing user accounts.

Functional Requirements:

Assumptions and Dependencies:

- Users and administrators are assumed to possess basic computer literacy and proficiency in the English language.
- Applicants may be required to scan and upload documents for verification purposes.
- Each user must have a unique user ID and password for accessing the platform.
- An administrator role is essential for overseeing system operations and managing user accounts.
- Continuous internet connectivity is necessary for seamless access to the e-book management system.
- Users must have compatible web browsers installed on their devices to access the platform effectively.

Operating Environment:

Particulars: Client System, Server System

Operating System: Windows/Linux/Android/iOS, Linux

Processor: Intel or AMD

Hard Disk: 1 GB for client system, 1 TB for server system

RAM: 256 MB for client system, 8 GB for server system

User Interfaces

• Reader/User: Individuals seeking access to digital books use the online interface to browse, search, and download e-books. This interface is designed to be intuitive and user-friendly, accessible via web browsers on desktops, laptops, tablets, and smartphones.

• Administrator: System administrators have access to a comprehensive dashboard for managing user accounts, moderating content, and generating reports. This interface is designed for efficient data management and decision-making.

Hardware Interfaces

The client systems (devices used by readers, authors, and administrators) interact with the server hardware to access and manage e-books. These client systems require standard hardware components such as processors, memory, and storage devices to support web browsing and application usage.

Software Interfaces

• Frontend Client: The user interface for readers and authors is developed using HTML, CSS, and JavaScript, with dynamic elements powered by JSP (JavaServer Pages). Additionally, the administrator interface utilizes Java for local application development.

• Web Server: The e-book management system is hosted on a apache tomcat server, ensuring robust performance and scalability for handling user requests and data processing.

• Backend Database: The system utilizes an Mongo db database for storing e-book metadata, user information, transaction records, and other relevant data. This backend infrastructure provides reliability, security, and efficient data management capabilities.

Communications Interfaces

The e-book management system relies on internet connectivity for user interactions and data transmission. Communication between client devices and the server occurs over the HTTP (Hypertext Transfer Protocol), enabling seamless exchange of information and interactions between users and the system. Additionally, email notifications and SMS

alerts are sent to users and administrators using appropriate communication protocols for timely updates and notifications.

Non-functional Requirements

Performance Requirements

Response Time: The system should respond to user actions, such as search queries or page loading, within 2 seconds under normal load conditions to ensure a seamless user experience.

Scalability: The system should be able to handle a growing number of users and e-books without significant degradation in performance. It should be able to support at least 10,000 concurrent users without experiencing slowdowns or crashes.

Data Retrieval Speed: E-books should be retrieved from the database and delivered to users for viewing or download within 5 seconds on average, even during peak usage periods.

Safety Requirements

Data Security: All user data, including personal information and payment details, should be encrypted and stored securely to prevent unauthorized access or data breaches.

Backup and Recovery: Regular backups of the system data should be performed to ensure data integrity and facilitate quick recovery in the event of system failures or disasters.

Security Requirements

Authentication: Each user must be provided with a unique ID and password for accessing their account. Authentication should be required before accessing any account information or e-books to prevent unauthorized usage.

Data Encryption: Communication between client devices and the server should be encrypted using SSL/TLS protocols to protect data during transmission.

Access Control: The system should implement role-based access control mechanisms to restrict access to sensitive features or data based on user roles and permissions.

Software Quality Attributes

Usability: The system should be intuitive and easy to use, with clear navigation and minimal learning curve for users.

Reliability: The system should be reliable and available 24/7, with minimal downtime for maintenance or upgrades.

Maintainability: The system should be designed with modular architecture and well-commented code to facilitate future updates and modifications.

Portability: The system should be compatible with a wide range of devices and operating systems, allowing users to access e-books from desktops, laptops, tablets, and smartphones.

Interoperability: The system should be able to integrate with external systems and services, such as payment gateways and content delivery networks, to provide seamless functionality.

Result:

Thus the Software Requirement Specifications (SRS) for the eBook management system is documented successfully.

Ex.No:3	IDENTIFY THE USE CASES AND DEVELOP THE USE CASE DIAGRAM FOR E-BOOK MANAGEMENT SYSTEM

Aim:

To identify the Use Cases and develop the Use Case diagram for the E-book management System using the StarUML tool.

Introduction of Use Case Diagram:

Use case diagrams are graphical representations that depict the interactions network users (actors) and a system to achieve specific goals or tasks. They are an essential tool in software development for capturing and communicating system requirements in a clear and Visual manner.

Purpose:

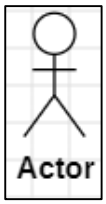
. Requirement Analysis: Use case diagrams to help stakeholders, including developers, designers, and clients, understand the system's functionality and behaviour from a user's perspective. They provide a high-level overview of the system's features and interactions.

. Communication: Use case diagrams to facilitate communication between stakeholders by providing a common visual language to discuss system requirements and functionalities. They help ensure that everyone involved in the project has a shared understanding of the system's scope and objectives.

. Design Validation: Use case diagram to assist in validating the system design by identifying potential gaps, inconsistencies, or missing functionalities. They allow stakeholders to review and refine the system requirements before proceeding with the implementation phase.

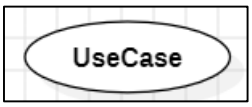
Notation:

1. Actor:



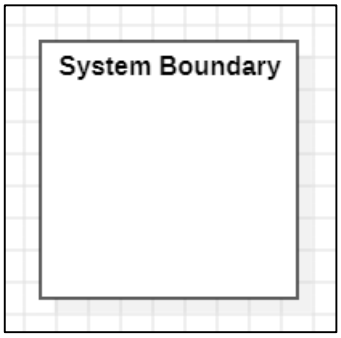
Actors are external entities that interact with the system. These can include users, other systems, or hardware devices. In a Use Case Diagram context, actors initiate use cases and receive the outcomes.

2. Use Case:

Use cases are like 

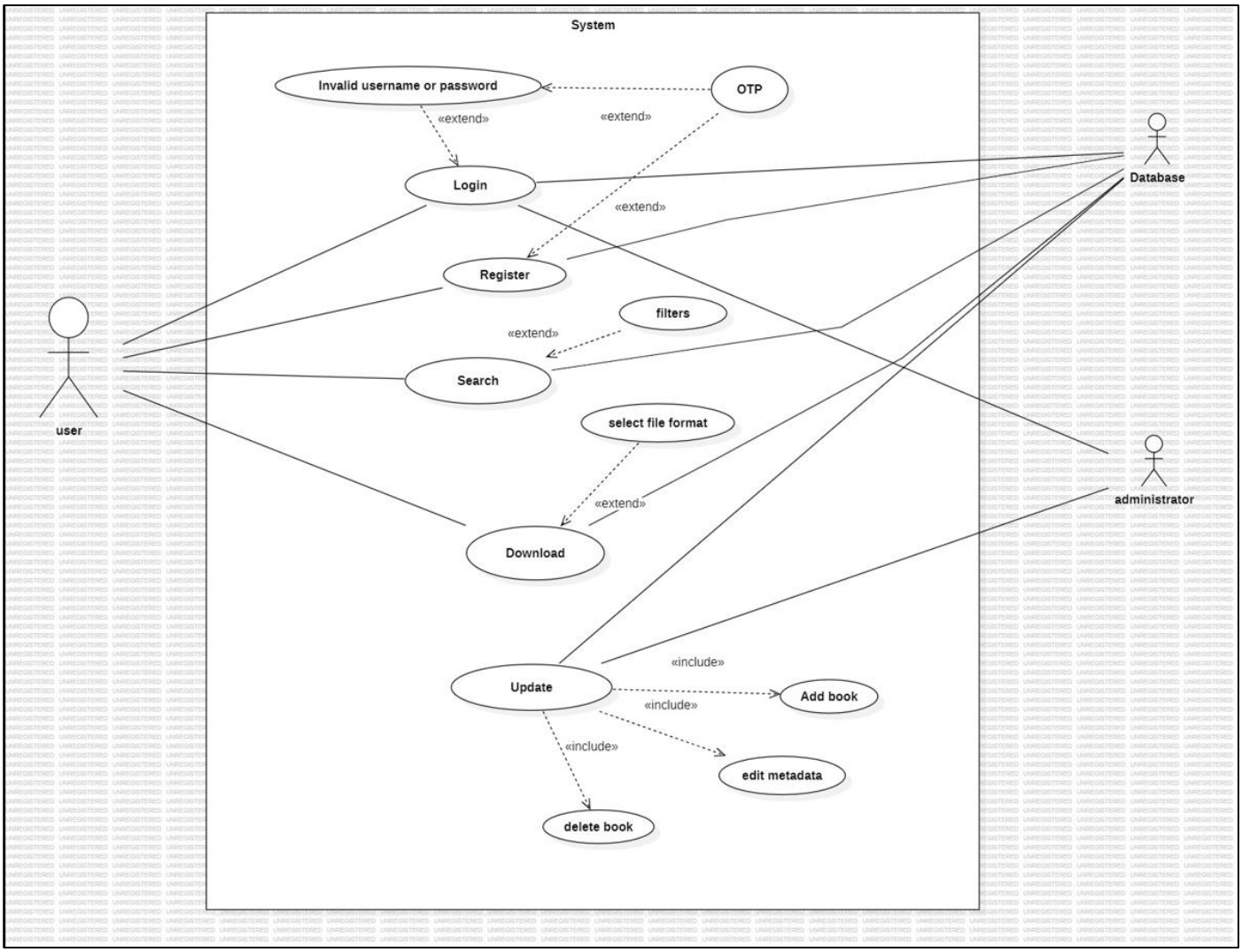
scenes in the play. They represent specific things your system can do. In the online shopping system, examples of use cases could be “Place Order,” “Track Delivery,” or “Update Product Information”. Ovals represent use cases.

3. System Boundary:



The system boundary is a visual representation of the scope or limits of the system you are modelling. It defines what is inside the system and what is outside. The boundary helps to establish a clear distinction between the elements that are part of the system and those that are external to it.

Use Case of E-book Management System



1. Register

- **Description:** Allows users to register to the system.
- **Actors:** User, Administrator
- **Primary Actor:** User
- **Preconditions:** The user has access to a device with internet connectivity.

Scenario:

- The user navigates to the eBook Management System website.
- The user selects the "Register" option.
- The user enters their name, username and password.
- The system verifies user credentials.
- If credentials are valid, the user is registered; otherwise, an error message is displayed.

2. Login

- **Description:** Allows users to login to the system.
- **Actors:** User
- **Primary Actor:** User
- **Preconditions:** The user has access to a device with internet connectivity.

Scenario:

- The user navigates to the eBook Management System website.
- The user selects the "Login" option.
- User enter their username and password.
- The system verifies user credentials.
- If credentials are valid, the user is logged in; otherwise, an error message is displayed.

3. Update

- **Description:** Allows admin to upload, organize, and manage their eBook collection.
- **Actors:** Admin, Database
- **Primary Actor:** Admin
- **Preconditions:** Admin is logged in and has access to their personal website.
- **Postconditions:** Admin's eBook collection is successfully updated and organized.

Scenario:

- Admin accesses their personal website.
- Admin uploads a new eBook to their website.
- Admin organizes eBooks into categories or collections.
- Admin edits metadata (title, author, description) of an eBook.

4. Search

- **Description:** Allows User to search the books based on their interest.
- **Actors:** User, Database
- **Primary Actor:** User
- **Precondition:** The user has access to a device with internet connectivity.

Scenario:

- The user navigates to the eBook Management System website.
- The user selects the "Seach Book" option.
- User enters the book id or book name or author name to search for the book they want.
- If the searched book appears the user reads it or downloads it.

5. Download:

- **Description:** Allows Users to download the book they want.
- **Actors:** User, Database.
- **Primary Actor: User.**
- **Precondition:** The user has access to a device with internet connectivity.

Scenario:

- The user navigates to the eBook Management System website.
- The user selects the "Download" option.
- The user searches for the book they want to download, after seeing the book details the user clicks the download option.

Result:

Thus to identify the use cases and develop the use case diagram for e-book management system was executed successfully.

Ex No: 4	IDENTIFY THE CONCEPTUAL CLASSES AND DEVELOP A DOMAIN MODEL AND ALSO DERIVE A CLASS DIAGRAM FROM THAT

Aim:

To identify the conceptual classes and develop a Domain Model and derive a Class Diagram from that EBook Management System.

Conceptual Classes:

In the EBook Management System, several conceptual classes can be identified to represent the various entities and functionalities of the system. Here are some key conceptual classes for the EBook Management System:

User:

Represents users of the system who interact with the EBook Management System. Users can register, login, search for an eBook, download a eBook, add reviews, and rate the books. They can also interact with the administrator for the registration purposes.

Administrator:

Represents the administrator of the system who manages the system by interacting with the database and the users. An administrator can login to the system with their own credentials, add or remove an eBook, and various other functions that manages the database and resolves issues within the system.

EBook:

Represents electronic books stored in the database of the system. Each eBook has attributes such as title, author, genre, etc. Users can read or download an eBook as they see fit.

Database:

Represents the storage mechanism for storing and managing eBooks and user data. Responsible for operations such as adding, removing, updating, searching eBooks, and managing user accounts.

Domain Model for EBook Management System:

User:

- **Attributes:**
  - Username
  - password
  - email
- **Operations:**
  - login(username, password)
  - register(username, password, email)
  - searchEbooks(query)
  - downloadEbooks(ebook)
  - RateEbook(ebook, rating)
  - addReview(ebook, review)
  - interactsWith(admin)

Administrator:

- **Attributes:**
  - username
  - password
  - email
- **Operations:**
  - login(username, password)
  - addEbook(ebook)
  - removeEbook(ebookId)
  - updateEbook(ebookId, updatedEbook)
  - searchEbooks(query)
  - getAllEbooks()
  - managesUsers()
  - resolvesIssues()

Ebook:

- **Attributes:**
  - title
  - author
  - genre
  - publicationDate
  - ISBN
  - ebookId
  - filePath
  - fileSize
  - description
  - rating
  - reviews

- **Operations:**
  - open()
  - close()
  - download()
  - rate(rating)
  - addReview(review)
  - getAverageRating()
  - getReviews()
  - getFilePath()
  - getSize()
  - getEbookId()
  - setEbookId(ebookId: String): void

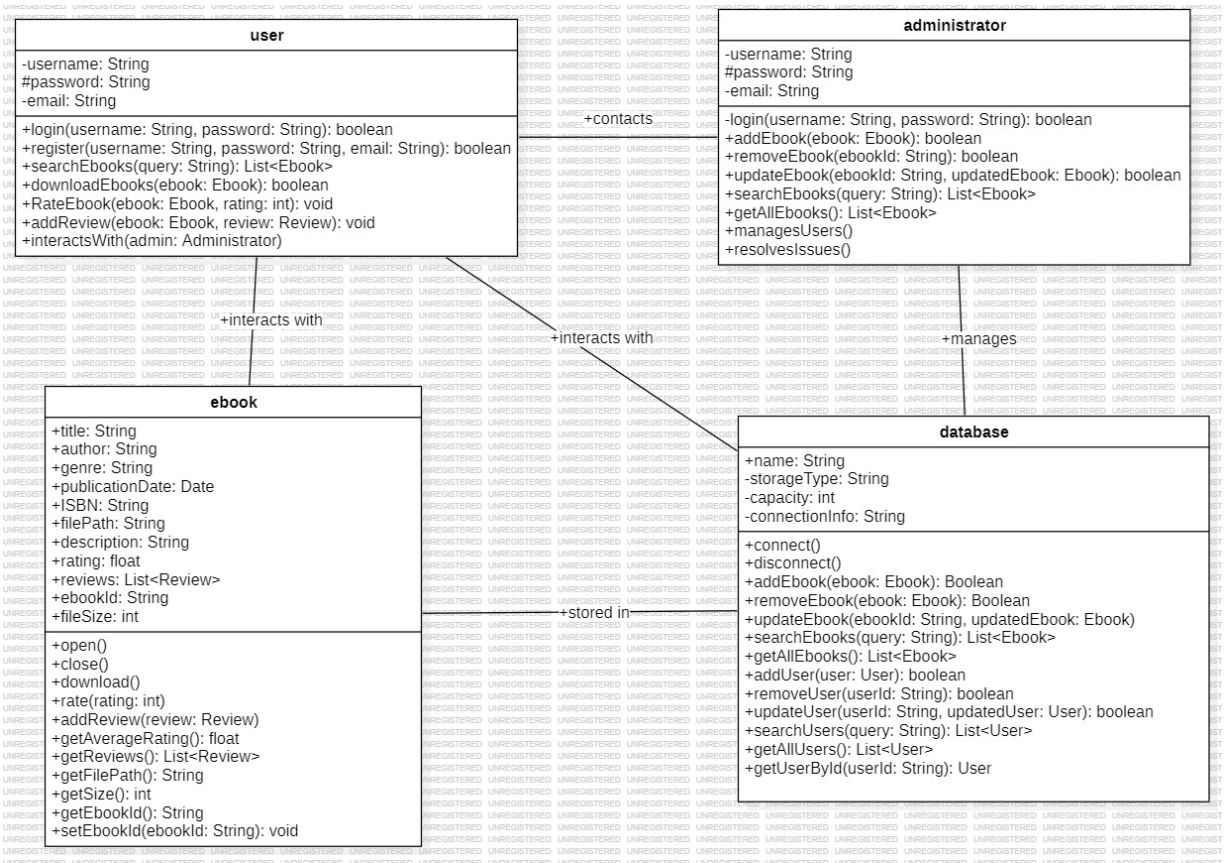
Database:

- **Attributes:**
  - name
  - storage

- capacity
- connectionInfo

- **Operations:**
  - connect()
  - disconnect()
  - addEbook(ebook)
  - removeEbook(ebook)
  - updateEbook(ebookId, updatedEbook)
  - searchEbooks(query)
  - getAllEbooks()
  - addUser(user)
  - removeUser(userId)
  - updateUser(userId, updatedUser)
  - searchUsers(query)
  - getUserById(userId)
  - getAllUsers()

Class Diagram for EBook Management System:



Result:

Thus the conceptual classes, Domain Model and Class Diagram for eBook Management System was identified and developed successfully.

Ex No:5	USING THE IDENTIFIED SCENARIOS, FIND THE INTERACTION BETWEEN OBJECTS AND REPRESENT THEM USING UML SEQUENCE AND COLLABORATION DIAGRAMS

Aim:

Using the identified scenarios, find the interaction between objects and represent them using UML Sequence and Collaboration Diagrams for E-book Management Systems.

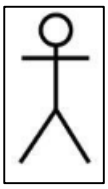
Sequence Diagram Definition:

Sequence diagrams, commonly used by developers, model the interactions between objects in a single use case. They illustrate how the different parts of a system interact with each other to carry out a function, and the order in which the interactions occur when a particular use case is executed.

A sequence diagram shows how different parts of a system work in a 'sequence' to get something done.

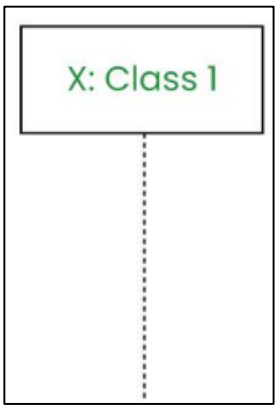
Sequence Diagram Notation:

1. Actors:



An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

2. Lifelines:

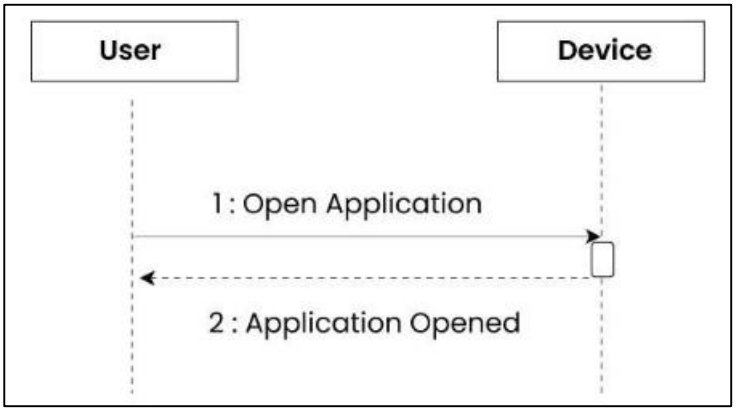


A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram.

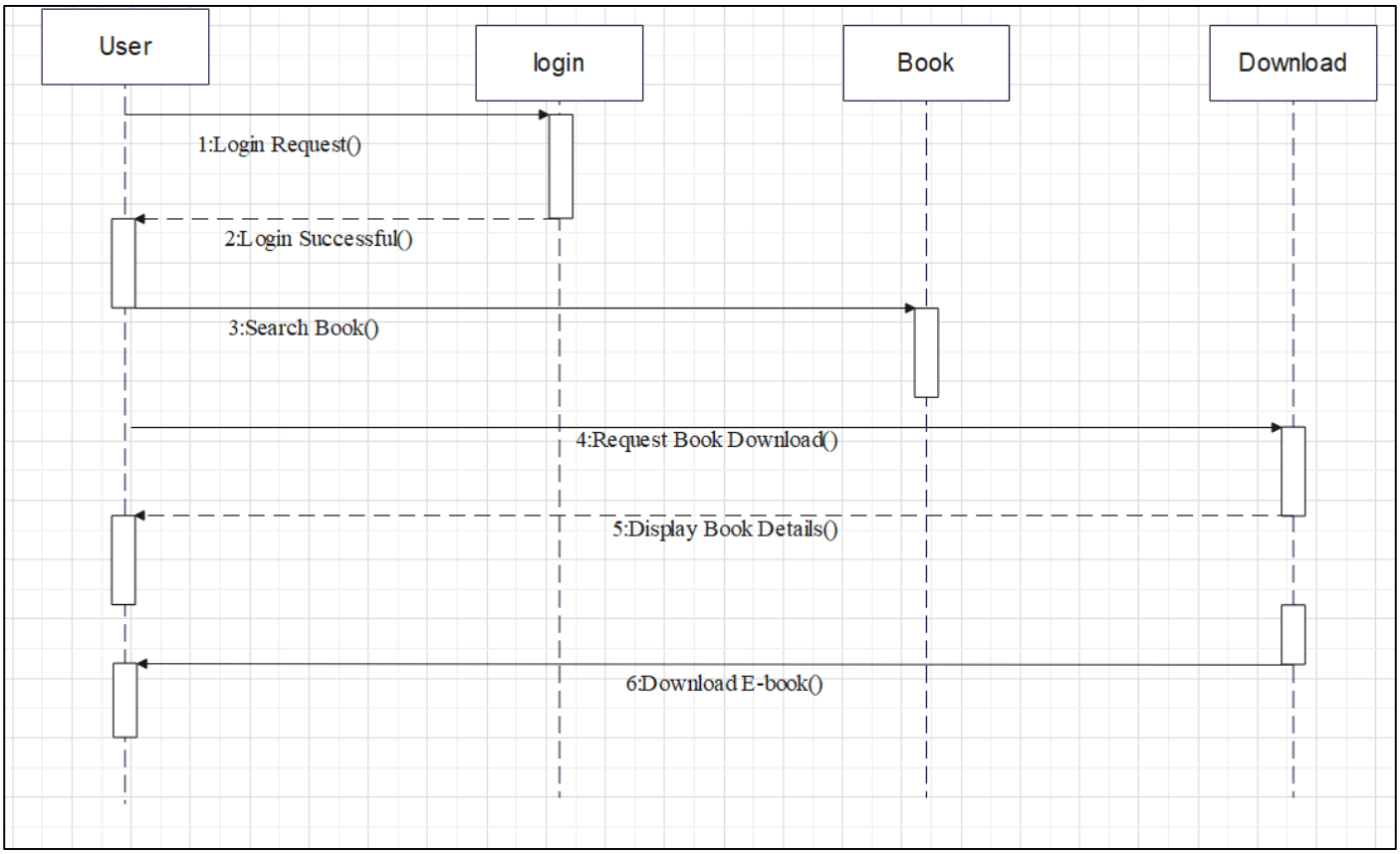
3. Messages:

Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline.

- We represent messages using arrows.
- Lifelines and messages form the core of a sequence diagram.



Sequence Diagram for E-book Management System:

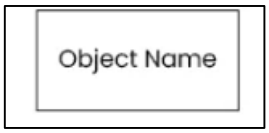


Collaboration Diagram Definition:

A Collaboration Diagram is a type of Interaction Diagram that visualises the interactions and relationships between objects in a system. It shows how objects collaborate to achieve a specific task or behaviour. Collaboration diagrams are used to model the dynamic behaviour of a system and illustrate the flow of messages between objects during a particular scenario or use case.

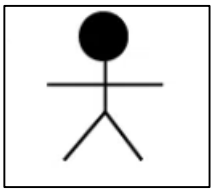
Collaboration Diagram Notation:

1. Objects/Participants:



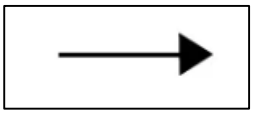
Objects are represented by rectangles with the object's name at the top. The diagram shows each object participating in the interaction as a separate rectangle. Objects are connected by lines to indicate messages being passed between them.

2. Actors:



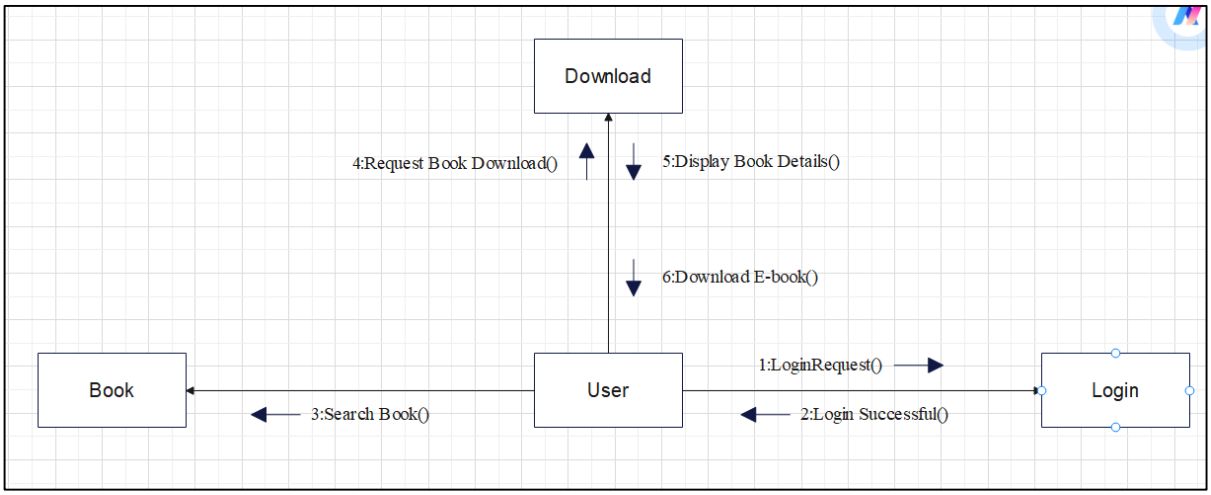
They are usually depicted at the top or side of the diagram, indicating their involvement in the interactions with the system's objects or components. They are connected to objects through messages, showing the communication with the system.

3. Messages:



Messages represent communication between objects. Messages are shown as arrows between objects, indicating the flow of communication. Each message may include a label indicating the type of message (e.g., method call, signal). Messages can be asynchronous (indicated by a dashed arrow) or synchronous (solid arrow).

Collaboration Diagram for E-book Management:



Result:

Thus, using the identified scenarios, it was successfully executed to find the interaction between objects and represent them using UML Sequence and Collaboration Diagrams for E-book Management Systems.

Ex. No:6	DRAW RELEVANT STATE CHART AND ACTIVITY DIAGRAMS FOR THE SAME SYSTEM

Aim:

To draw relevant state charts and activity diagrams for the E-book management system.

State Chart Diagram Definition:

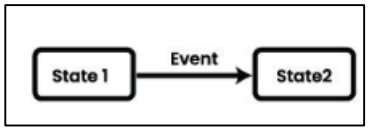
A State Machine Diagram represents the system's condition or part of the system at finite instances of time. It's a behavioural diagram and it represents the behaviour using finite state transitions.

State Chart Diagram Notation:

1. Initial State:

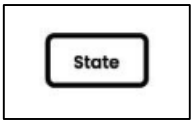
We use a black-filled circle to represent the initial state of a System or a Class.

2. Transition:



We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.

3. State:

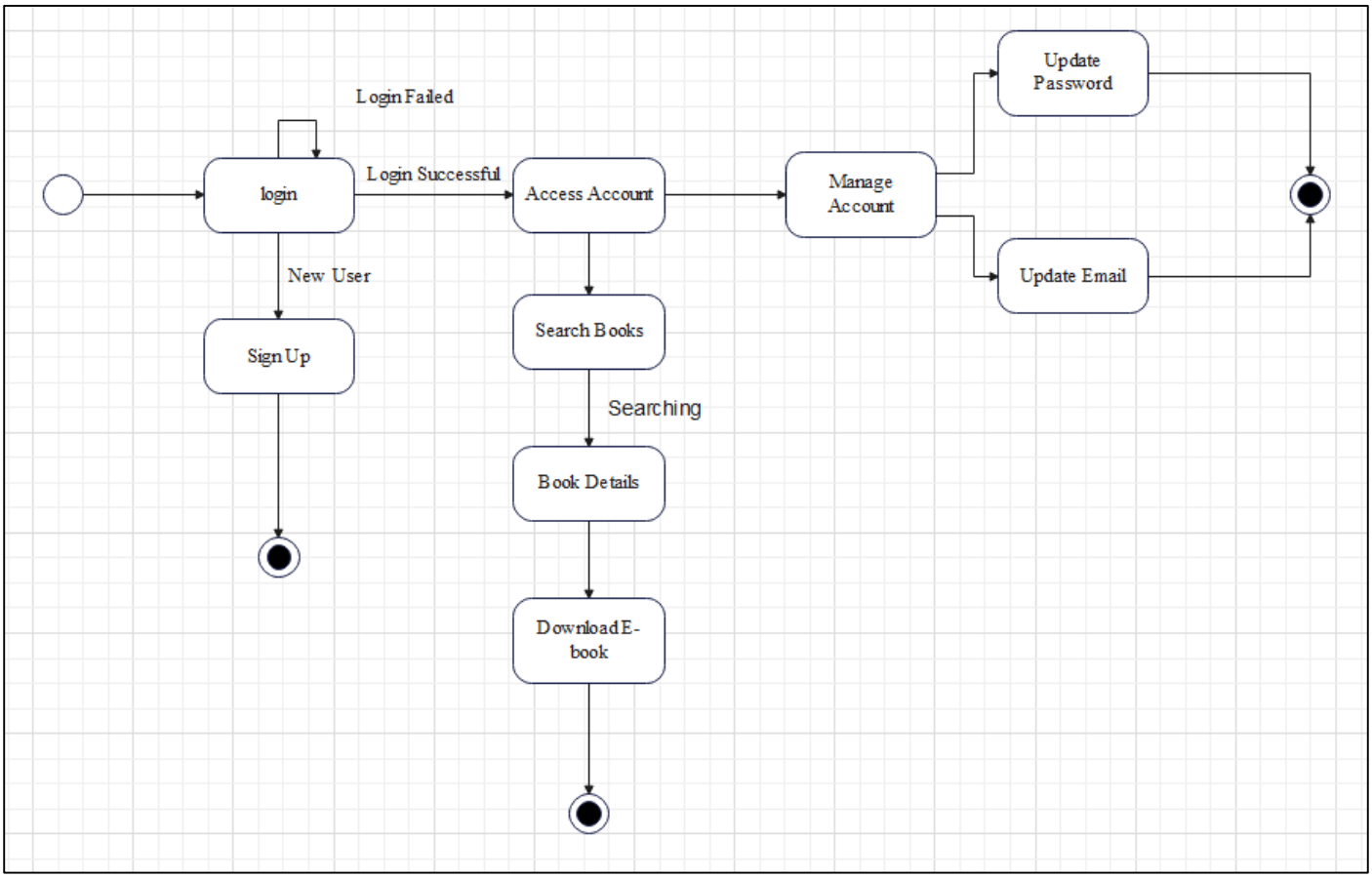


We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.

4. Final State:

We use a filled circle within a circle notation to represent the final state in a state machine diagram.

State Chart Diagram for E-book Management System:



Activity Diagram Definition:

Activity Diagrams are used to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We can depict both sequential processing and concurrent processing of activities using an activity diagram i.e., an activity diagram focuses on the condition of flow and the sequence in which it happens.

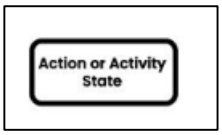
- We describe what causes a particular event using an activity diagram.
- An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.
- They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system.

Activity Diagram Notation:

1. Initial State:

A process can have only one initial state unless we are depicting nested activities.We use a black-filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.

2. Action or Activity State:

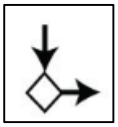


An activity represents the execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically, any action or event that takes place is represented using an activity.

3. Action Flow or Control Flow:

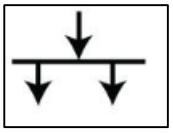
Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another activity state.

4. Decision Node or Branching:

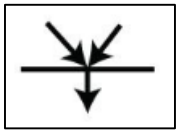


When we need to make a decision before deciding the flow of control, we use the decision node. The outgoing arrows from the decision node can be labelled with conditions or guard expressions. It always includes two or more output arrows.

5. Fork:



Fork nodes are used to support concurrent activities. When we use a fork node both the activities get executed concurrently i.e. no decision is made before splitting the



activity into two parts.

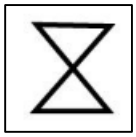
6. Join:

Join nodes are used to support concurrent activities converging into one. We have two or more incoming edges and one outgoing edge for join notations.

7. Final State or End State:

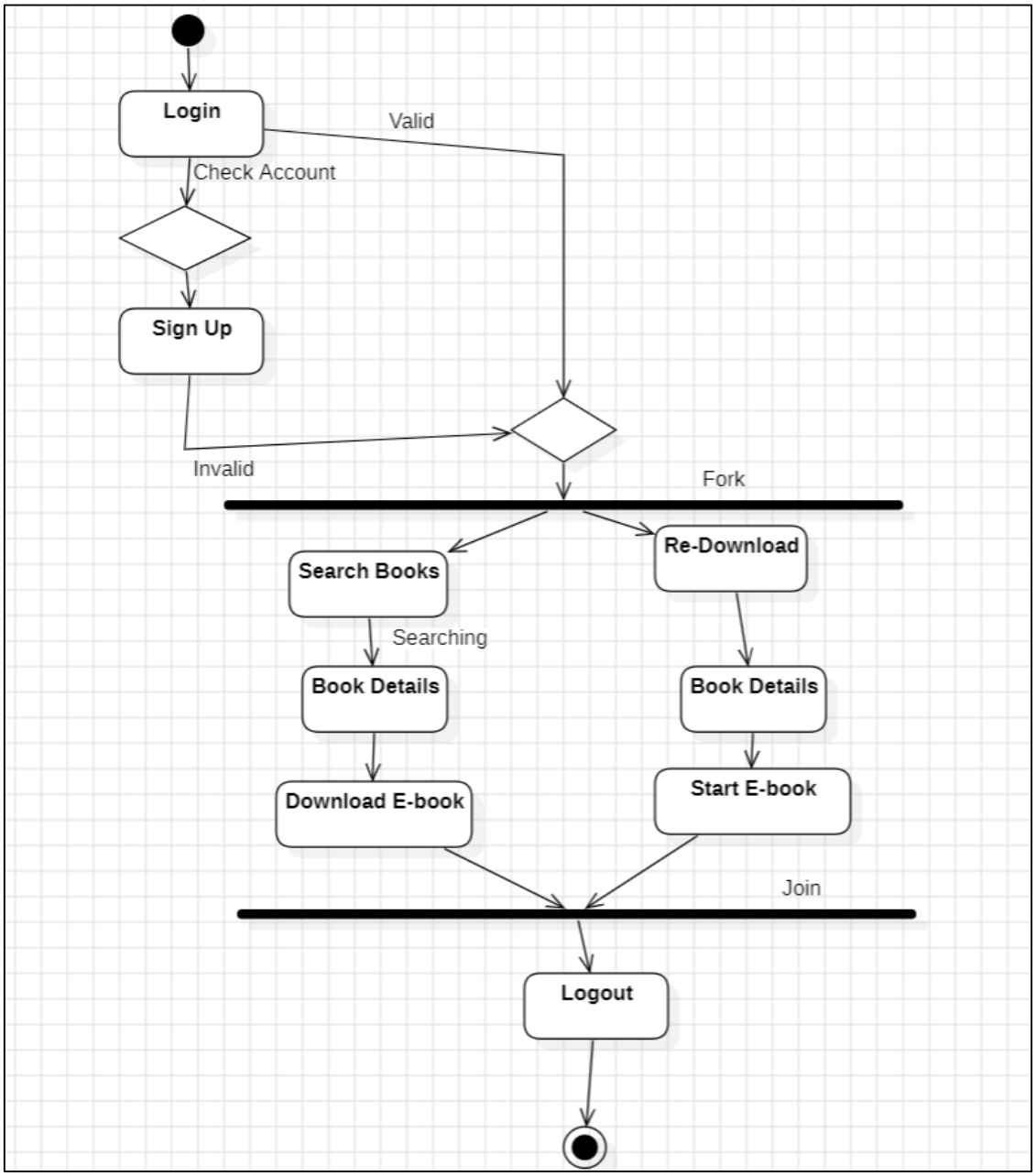
The state that the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.

8. Time Event:



This refers to an event that stops the flow for a time; an hourglass depicts it. We can have a scenario where an event takes some time to complete.

Activity Diagram for E-book Management System:



Result:

Thus the relevant state chart and activity diagrams for the E-book management system was executed successfully.

Ex No:7	IMPLEMENT THE SYSTEM AS PER THE DETAILED DESIGN

Aim:

To implement the E-book Management System as per the detailed design.

Component Diagram Definition:

Component diagrams are used to visualize the organization of system components and the dependency relationships between them. They provide a high-level view of the components within a system.

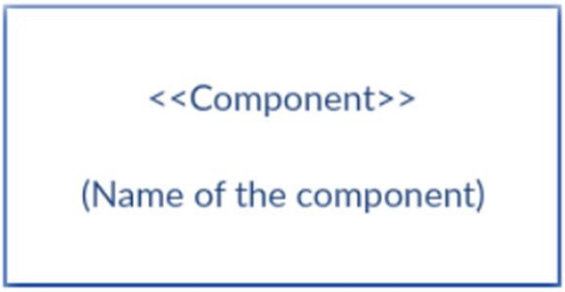
The components can be a software component such as a database or user interface; or a hardware component such as a circuit, microchip or device; or a business unit such as supplier, payroll or shipping.

Component Diagram Notation:

1. Component:

There are three ways the component symbol can be used.

a.



Rectangle with the component stereotype (the text <<component>>). The component stereotype is usually used above component name to avoid confusing the shape with a class icon.

b.

Rectangle with the component icon in the top right corner and the name of

the component.

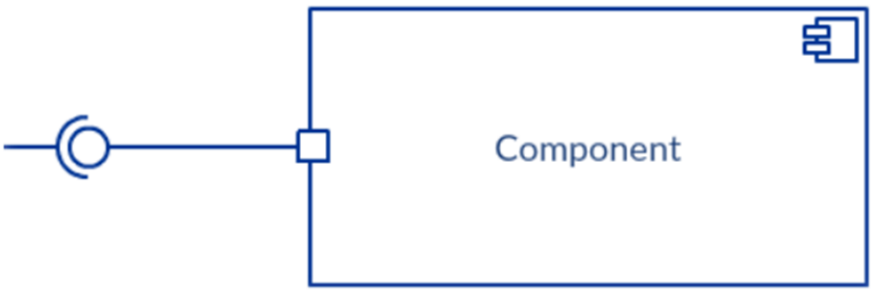
c.

Rectangle with the component icon and the component stereotype.

2. Provided Interface and Required Interface:

Interfaces in component diagrams show how components are wired together and interact with each other. The assembly connector allows linking the component's required interface (represented by a semi-circle and a solid line) with the provided interface (represented by a circle and solid line) of another component. This shows that one component is providing the service that the other is requiring.

3. Port:

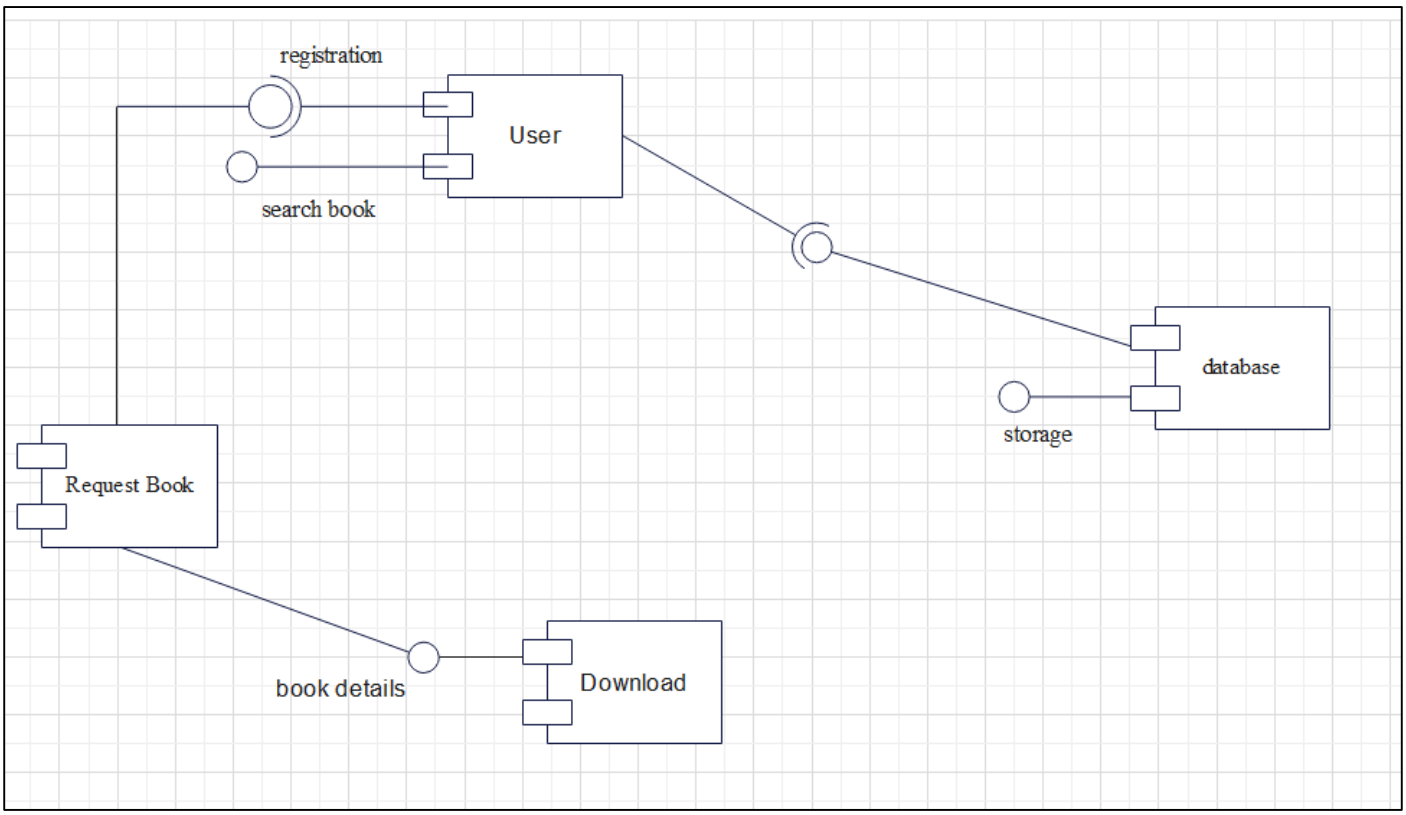


Port (represented by the small square at the end of a required interface or provided interface) is used when the component delegates the interfaces to an internal class.

4. Dependencies:

Although you can show more detail about the relationship between two components using the ball-and-socket notation (provided interface and required interface), you can just as well use a dependency arrow to show the relationship between two components.

Component Diagram for E-book Management System:



Deployment Diagram Definition:

A Deployment Diagram in software engineering is a type of Structural UML Diagram that shows the physical deployment of software components on hardware nodes. It illustrates the mapping of software components onto the physical resources of a system, such as servers, processors, storage devices, and network infrastructure.

Deployment Diagram Notation:

1. Component:

A component represents a modular and reusable part of a system, typically implemented as a software module, class, or package. It encapsulates its behaviour and data and can be deployed independently.

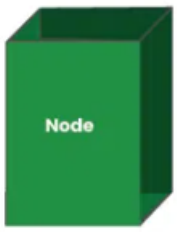
2. Artifact:

An artifact represents a physical piece of information or data that is used or produced in the software development process. It can include source code files, executables, documents, libraries, configuration files, or any other tangible item.

3. Interface:

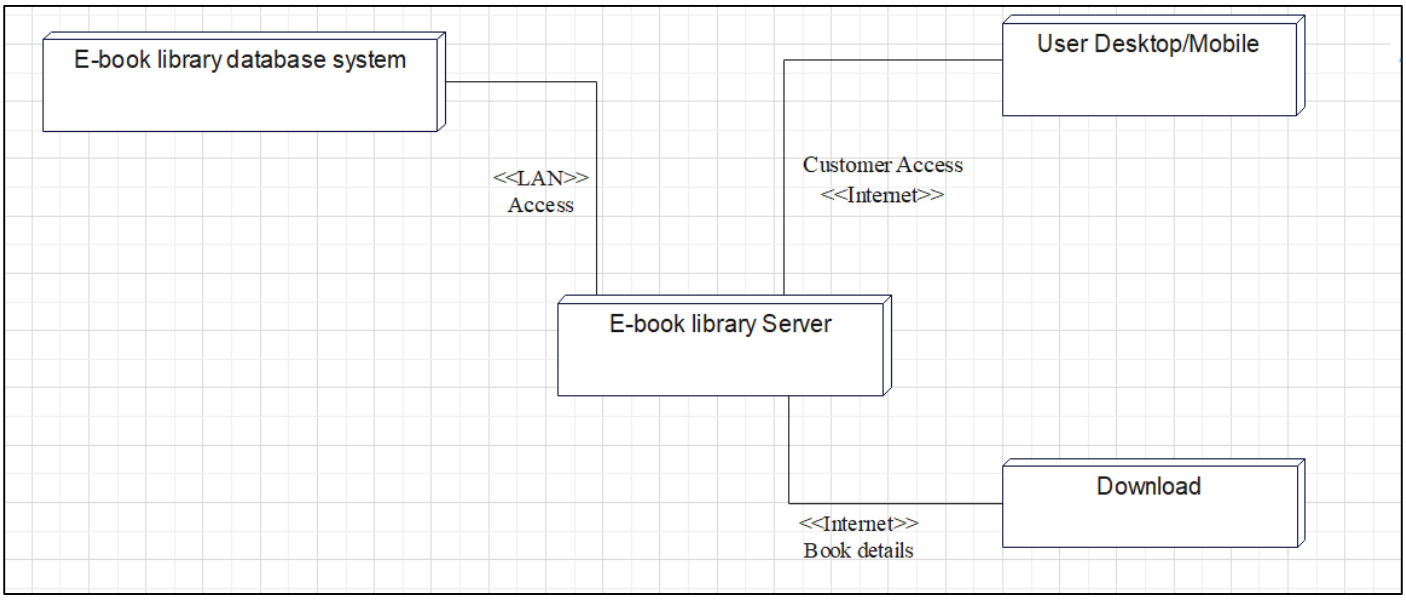
An interface defines a contract specifying the methods or operations that a component must implement. It represents a point of interaction between different components or subsystems.

4. Node:



A node represents a physical or computational resource, such as a hardware device, server, workstation, or computing resource, on which software components can be deployed or executed.

Deployment Diagram for E-book Management:



Code Generation from Class Diagram:

ebook.java:

```
import java.util.*;
public class user {
    public user() {
    }
    private String username;
    protected String password;
    private String email;
    public boolean login(String username, String password) {
        return false;
    }
    public boolean register(String username, String password, String email) {
        return false;
    }
    public List<Ebook> searchEbooks(String query) {
        return null;
    }
    public boolean downloadEbooks(Ebook ebook) {
```

```
        return false;
    }
    public void RateEbook(Ebook ebook, int rating) {
        return null;
    }
    public void addReview(Ebook ebook, Review review) {
        return null;
    }
    public void interactsWith(Administrator admin) {
    }
}
```

user.java

```
import java.util.*;
public class user {
    public user() {
    }
    private String username;
    protected String password;
    private String email;
    public boolean login(String username, String password) {
        return false;
    }
    public boolean register(String username, String password, String email) {
        return false;
    }
    public List<Ebook> searchEbooks(String query) {
        return null;
    }
    public boolean downloadEbooks(Ebook ebook) {
        return false;
    }
    public void RateEbook(Ebook ebook, int rating) {
```

```
        return null;
    }
    public void addReview(Ebook ebook, Review review) {
        return null;
    }
    public void interactsWith(Administrator admin) {
    }
}
```

database.java:

```
import java.util.*;
public class database {
    public database() {
    }
    public String name;
    private String storageType;
    private int capacity;
    private void Attribute1;
    private String connectionInfo;
    public void connect() {
    }
    public void disconnect() {
    }
    public Boolean addEbook(Ebook ebook) {
        return null;
    }
    public Boolean removeEbook(Ebook ebook) {
        return null;
    }
    public void updateEbook(String ebookId, Ebook updatedEbook) {
    }
    public List<Ebook> searchEbooks(String query) {
        return null;
```

```
    }
    public List<Ebook> getAllEbooks() {
        return null;
    }
    public boolean addUser(User user) {
        return false;
    }
    public boolean removeUser(String userId) {
        return false;
    }
    public boolean updateUser(String userId, User updatedUser) {
        return false;
    }
    public List<User> searchUsers(String query) {
        return null;
    }
    public List<User> getAllUsers() {
        return null;
    }
    public User getUserById(String userId) {
        return null;
    }
}
```

administrator.java:

```
import java.util.*;
public class administrator {
    public administrator() {
    }
    private String username;
    protected String password;
    private String email;
    private boolean login(String username, String password) {
```

```
        return false;
    }
    public boolean addEbook(Ebook ebook) {
        return false;
    }
    public boolean removeEbook(String ebookId) {
        return false;
    }
    public boolean updateEbook(String ebookId, Ebook updatedEbook) {
        return false;
    }
    public List<Ebook> searchEbooks(String query) {
        return null;
    }
    public List<Ebook> getAllEbooks() {
        return null;
    }
    public void managesUsers() {
    }
    public void resolvesIssues() {
    }
}
```

Result:

Thus, the E-book Management System implementation as per the detailed design was executed successfully.

Ex No: 8	TEST THE SOFTWARE SYSTEM FOR ALL THE SCENARIOS IDENTIFIED AS PER THE USECASE DIAGRAM

Aim:

To test the software system for all the scenarios identified as per the usecase diagram.

eBook Management System:

In today's digital age, the consumption of literature has evolved significantly. With the rise of electronic books (eBooks), readers are increasingly turning to digital platforms for accessing and managing their reading materials. To cater to this growing demand and to streamline the process of managing eBooks, an eBook management system offers a comprehensive solution.

An eBook management system is a software application designed to facilitate the organization, storage, retrieval, and consumption of electronic books (eBooks). It serves as a centralized platform for users to discover, acquire, organize, and download eBooks across various devices.

I can provide you with a general approach to testing the software system for the scenarios identified in the use case diagram for an eBook management system.

Review Use Case Diagram:

- Ensure that you thoroughly understand the interactions and functionalities depicted in the use case diagram.
- Identify primary actors (users) and their goals within the system.

Identify Test Scenarios:

- Consider scenarios from different perspectives, including those of different user roles (e.g., admin, regular user).
- Cover a wide range of functionalities, including basic operations and edge cases.

Create Test Cases:

- Develop detailed test cases with clear steps, inputs, and expected outcomes.
- Cover positive scenarios (where the system behaves as expected) and negative scenarios (where errors or exceptions are expected).

Prepare Test Environment:

- Set up a separate environment for testing to avoid interference with the production system.
- Ensure that the test environment mirrors the production environment as closely as possible in terms of configuration and data.

Execute Test Cases:

- Follow the test cases meticulously, recording both the steps taken and the actual results observed.
- Use test data that covers various scenarios, including boundary values and invalid inputs.

Report Bugs:

- Document any discrepancies or defects encountered during testing in a bug tracking system.
- Provide sufficient information to help developers reproduce and diagnose the issues effectively.

Unit Testing:

- Unit testing is highly suitable for a simple eBook management system. It involves testing individual components or units of the system (e.g., classes, methods) in isolation to verify their functionality.
- It ensures that the core functionalities such as eBook creation, metadata handling, and user authentication work correctly at a granular level.

Example:

Component: eBook class

Functionality to Test: getEbookId() method

Test Case: Verify that the getEbookId() method of the eBook class returns the correct Id of an eBook.

Regression Testing:

- Regression testing is crucial for a simple eBook management system to detect and prevent the introduction of new defects or issues when modifications or enhancements are made.
- It includes re-running previously executed tests to ensure that existing functionalities still work after changes are implemented (e.g., bug fixes, updates), thereby maintaining system stability and reliability.

Example:

Scenario Change: Update the search algorithm to improve performance

Test Case: After implementing the search algorithm update, re-run previous test cases (e.g., search functionality, eBook upload) to ensure that existing functionalities are not affected and continue to work correctly.

Black Box testing:

- Black box testing focuses on testing the functionality of the eBook management system without knowing its internal structure or implementation details.
- It validates user-facing features such as eBook upload, search functionality, and user interface interactions to ensure they meet specified requirements.

Example:

Functionality to Test: Search functionality

Test Case: Enter a known eBook title into the search field and verify that the system returns the expected eBook details.

White Box Testing:

- White box testing involves examining the internal structure and logic of the eBook management system, including code paths, branches, and data flows.

- It verifies the correctness of algorithms, error handling mechanisms, and data validations implemented within the system.

Example:

Component: eBook validation logic

Test Case: Validate that the system correctly handles validation of eBook metadata (e.g., title, author) against defined rules (e.g., length constraints, required fields).

Integration Testing:

- Integration testing is essential for verifying interactions between different components and modules of the eBook management system.
- It ensures that integrated functionalities such as database access, user authentication, and eBook processing work together seamlessly and handle data exchanges correctly.

Example:

Components to Integrate: User authentication module, eBook management module

Test Case: Authenticate a user and verify that authorized users can perform eBook management operations (e.g., upload, delete) seamlessly.

System Testing:

- System testing evaluates the entire eBook management system as a whole to validate its behaviour and performance against specified requirements.
- It tests end-to-end scenarios, including user workflows (e.g., searching, downloading eBooks), system performance under normal and peak loads, and compatibility with different devices and browsers.

Example:

End-to-End Workflow: User interaction with the eBook management system

Test Case: Simulate a user session where the user logs in, uploads an eBook, performs a search, and downloads the eBook. Verify that each step of the workflow functions as expected.

Result:

Thus the software system was tested successfully for all the scenarios identified as per the use case diagram.

Ex No: 9	IMPROVE THE REUSABILITY AND MAINTAINABILITY OF THE SOFTWARE SYSTEM BY APPLYING APPROPRIATE DESIGN PATTERNS

Aim:

To Improve the reusability and maintainability of the software system by applying appropriate design patterns.

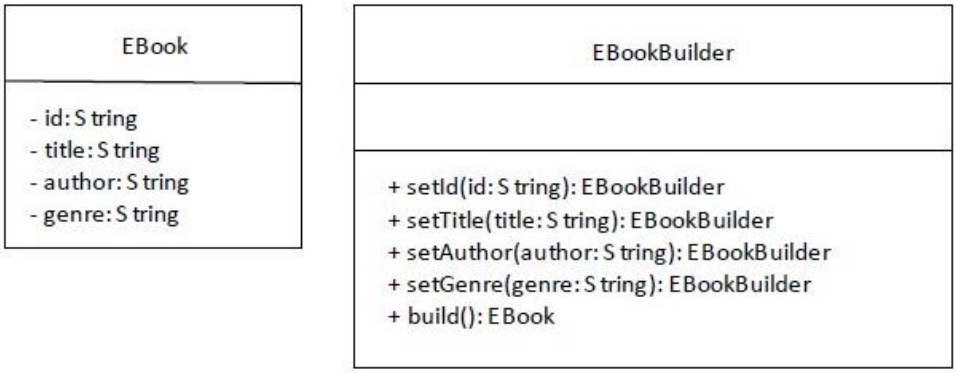
eBook Management System:

An eBook management system is a website designed to help users organize, access, and manage electronic books (eBooks). It serves as a centralized platform where users can store, organize, and download eBooks across various devices such as computers, tablets, and smartphones. The system provides functionalities to manage eBook catalogues, handle user authentication, facilitate eBook acquisition, and offer reading tools for a seamless reading experience.

To enhance the reusability and maintainability of the eBook management system, we can incorporate several design patterns. Here are some design patterns that can be applied:

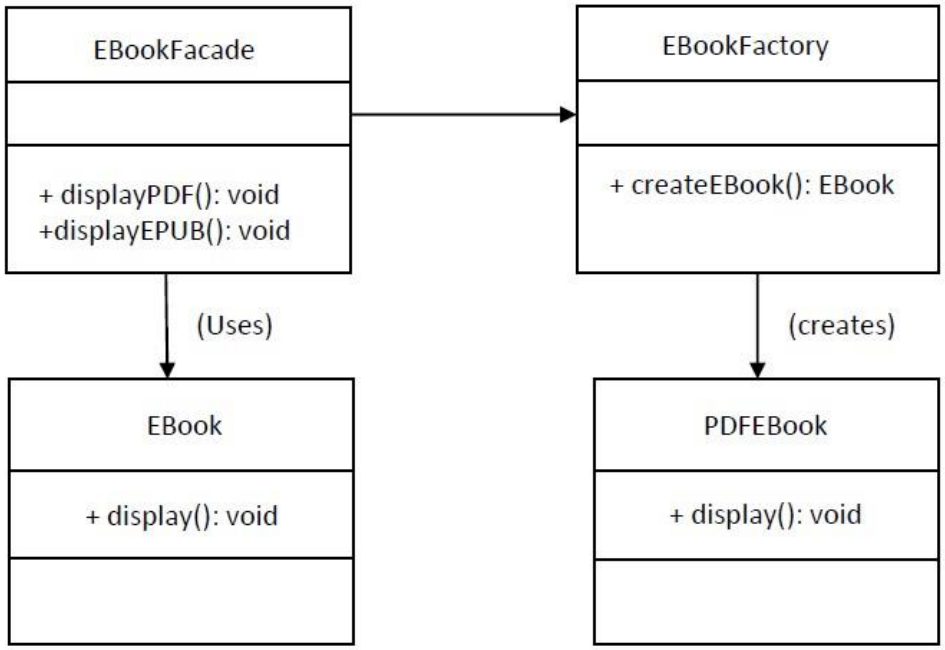
Builder Pattern:

- The Builder pattern separates the construction of complex objects from their representation, allowing for the creation of different configurations or variations of objects using a common construction process. In the eBook management system, we can use a builder to construct eBook objects with different configurations (e.g., title, author, content) without exposing their internal representation.
- This promotes reusability by providing a flexible and composable construction process for creating complex objects.



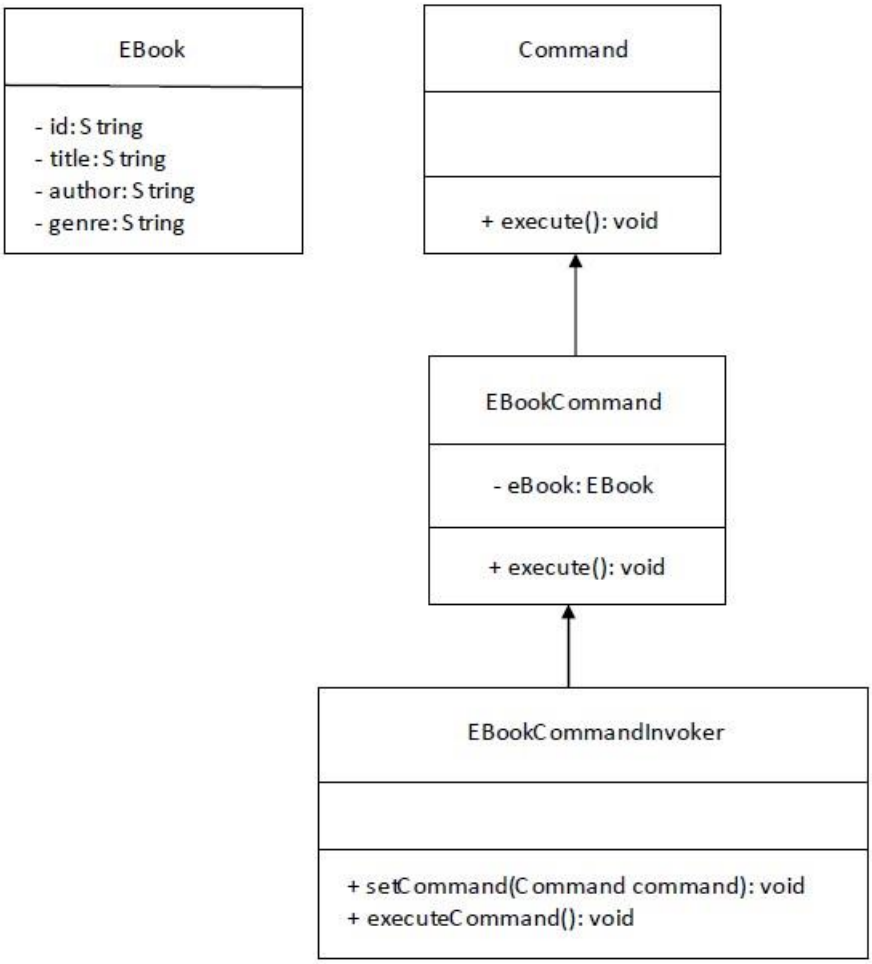
Facade Pattern:

- The Facade Pattern simplifies interactions with an eBook management system by providing a unified interface (EBookFacade) for clients.
- This facade encapsulates the complexity of eBook creation and instantiation using an EBookFactory. Clients can use the facade's methods (displayPDF(), displayEPUB()) to interact with specific eBook formats (PDFEBook, EPUBEBook) without needing to manage the intricacies of the underlying subsystem directly.
- The Facade Pattern promotes simplicity, encapsulation, and abstraction, enhancing the maintainability and usability of the eBook management system.



Command Pattern:

- The Command pattern encapsulates requests as objects, allowing for parameterization of clients with queued or logged requests. For example, we can use the Command pattern to implement undo/redo functionality, batch processing, or asynchronous operations in the eBook management system.
- This promotes reusability by encapsulating request details and allows for the decoupling of requesters and receivers.



By incorporating these design patterns into the eBook management system, we can improve its reusability and maintainability by promoting code reuse, encapsulating complexity, and decoupling components. This leads to a more modular, flexible, and maintainable architecture, making it easier to extend, modify, and maintain the system over time.

Result:

Thus the reusability and maintainability of the software system was improved successfully by applying appropriate design patterns.