School of Computer Science and Engineering

Winter Semester 2023-24

Continuous Assessment Test – II

SLOT: C2 + TC2

Programme Name & Branch: B. Tech. Computer Science & Engineering (Core and Specializations)

Course Name & Code: Software Engineering & BCSE 301L

Class Number (s):
VL2023240500717, VL2023240500591, VL2023240500732, VL2023240500677, VL2023240505793, VL2023240500704, VL2023240500605, VL2023240500657, VL2023240500738, VL2023240500669, VL2023240500726, VL2023240500616, VL2023240500653, VL2023240500625, VL2023240500645, VL2023240500711, VL2023240500596, VL2023240500599, VL2023240500612, VL2023240500722

Faculty Name (s):
Dr.ANBARASI M, Dr.KUMAR K, Dr.SWATHI J.N, Dr.DEEPA.K, Dr.AKILA VICTOR, Dr.SHASHANK MOULI SATAPATHY, Dr.CHELLATAMILAN T, Dr.PARTHIBAN K, Dr.PEREPI RAJARAJESWARI, Dr.BAIJU B V, Dr.VETRISELVI T, Dr.POORNIMA N, Dr.KRISHNARAJ N, Dr.JAFAR ALI IBRAHIM S, VASANTHI P, Dr.MADHAN E S, Dr.BASKARAN P, Dr.SAYAN SIKDER, Dr.TAMIZHSELVI SP, Dr.DEEPIKA J, Dr.SAURABH AGRAWAL, Dr.RUP KUMAR DEKA, Dr.SAMRIDDHI SARKAR

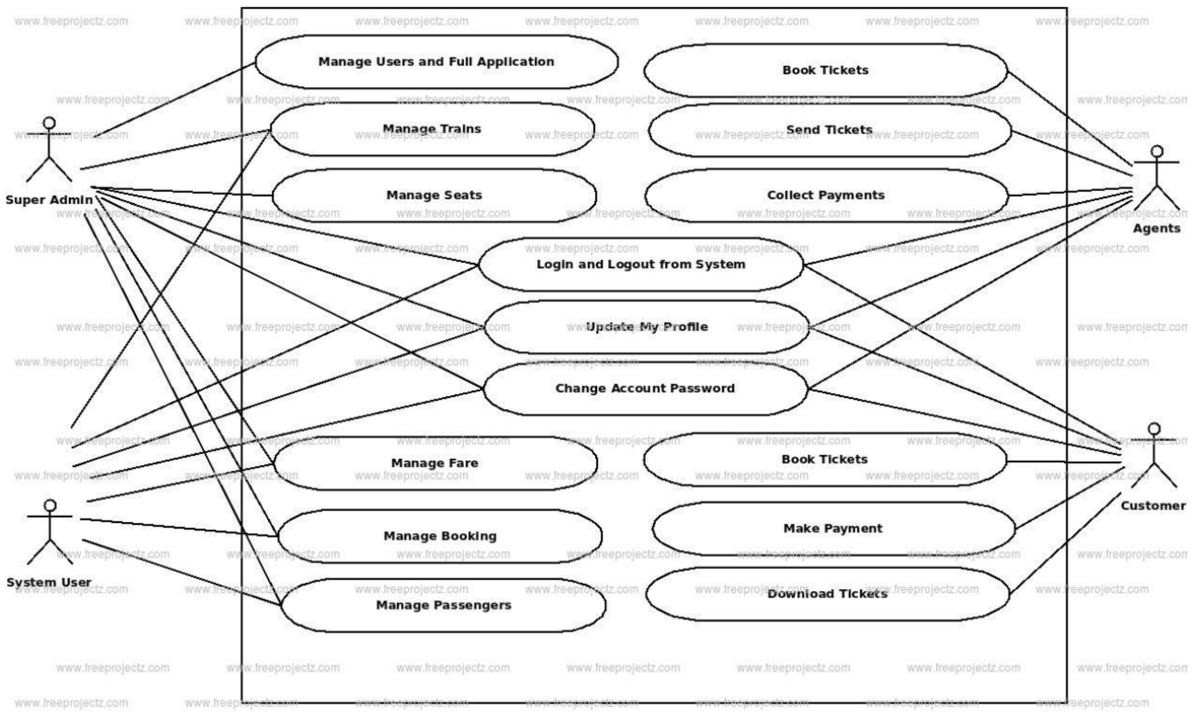
Exam Duration: 90 Min. **Maximum Marks: 50**

<u>General instruction(s):</u>

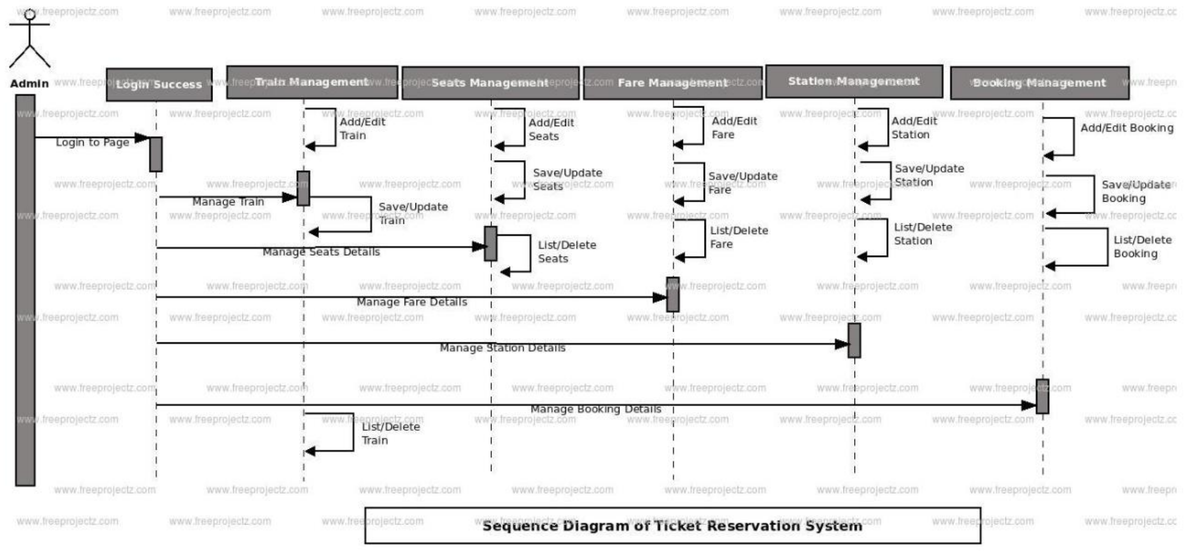
Printed material is not permitted.

Q. No.	Question	Marks	CO	BL
1	You are part of a project team developing an enhanced ticket reservation system for a public transportation network. This system must integrate real-time seat availability, fare calculation, passenger details, and payment processing. Focusing on the system parameters for fare calculation, which involves factors like distance, ticket type (e.g., adult, child, senior), peak hours pricing etc. a) Model the interaction between users and this ticket reservation system using suitable UML approach. (4 Marks) b) Describe suitable UML approach to model the ticket booking scenario (6 Marks)	10	CO3	BL3
Ans	**Schema**: a) Identification of suitable UML diagram – 1 Mark, Diagram representation - 3 Marks b) Identification of suitable UML diagram – 1 Mark, Diagram representation along with all system parameters mentioned in the question - 5 Marks			

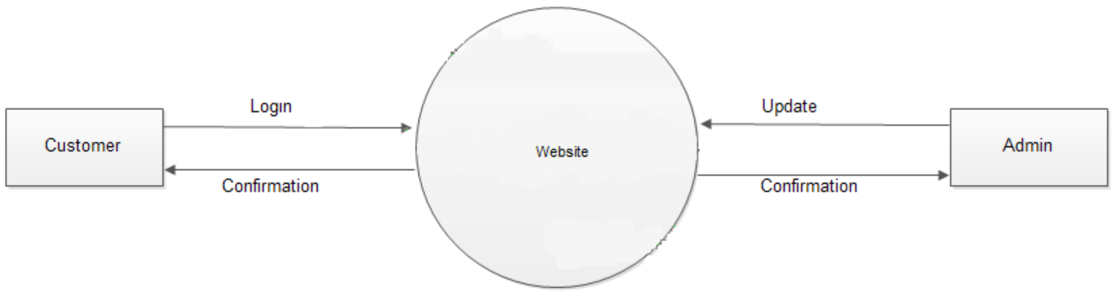
Usecase diagram:

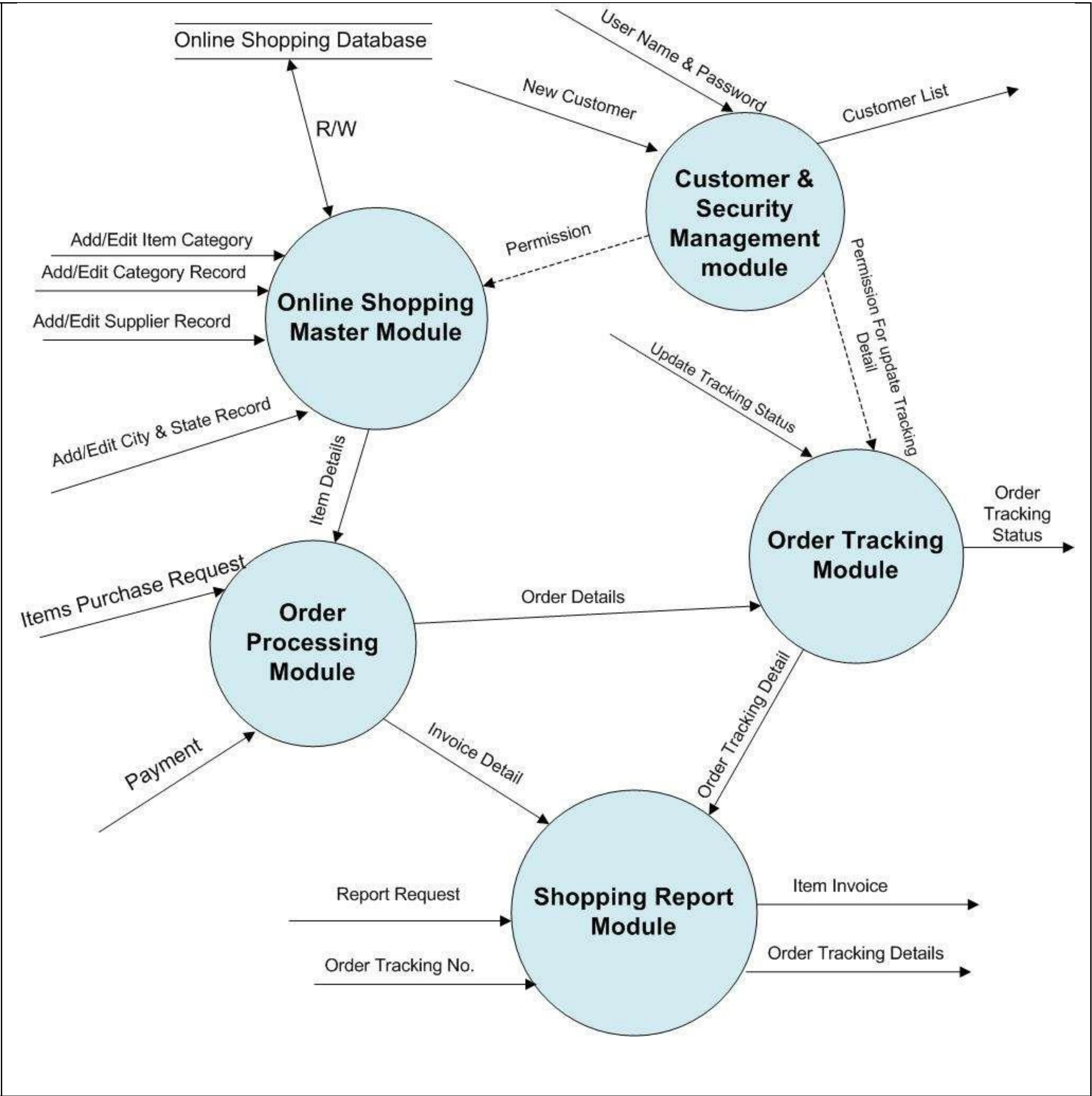


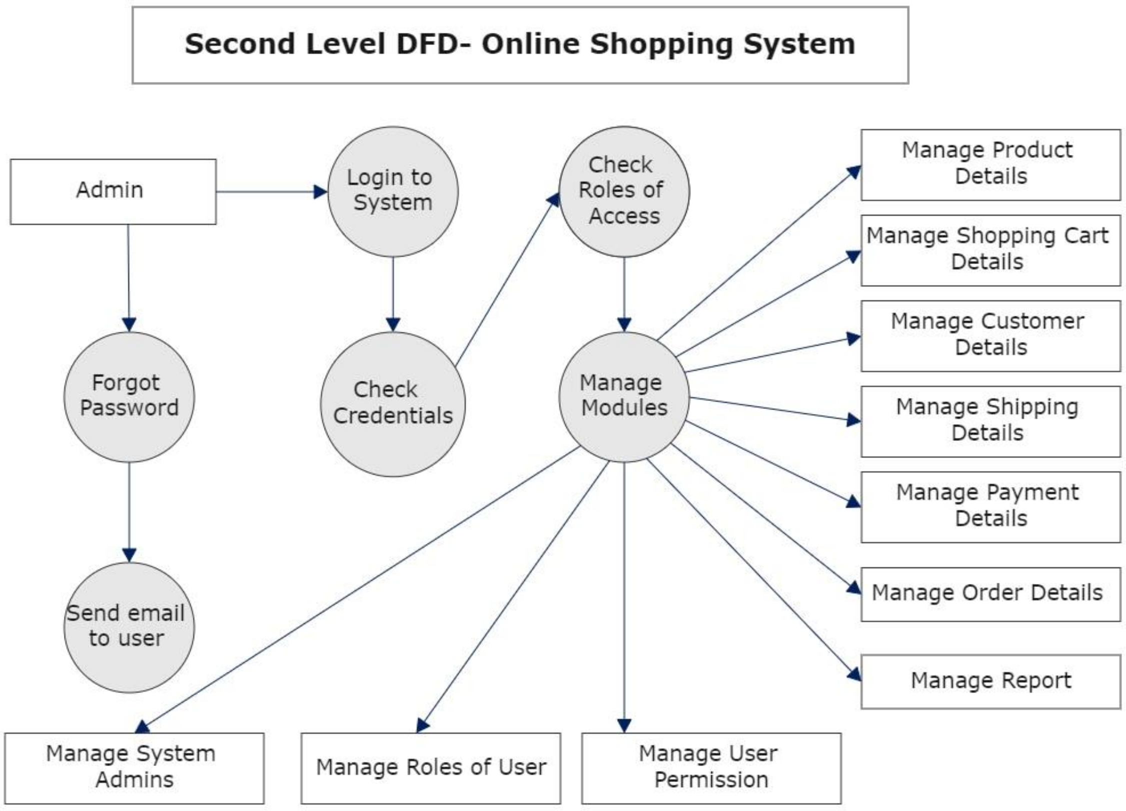
Sequence diagram

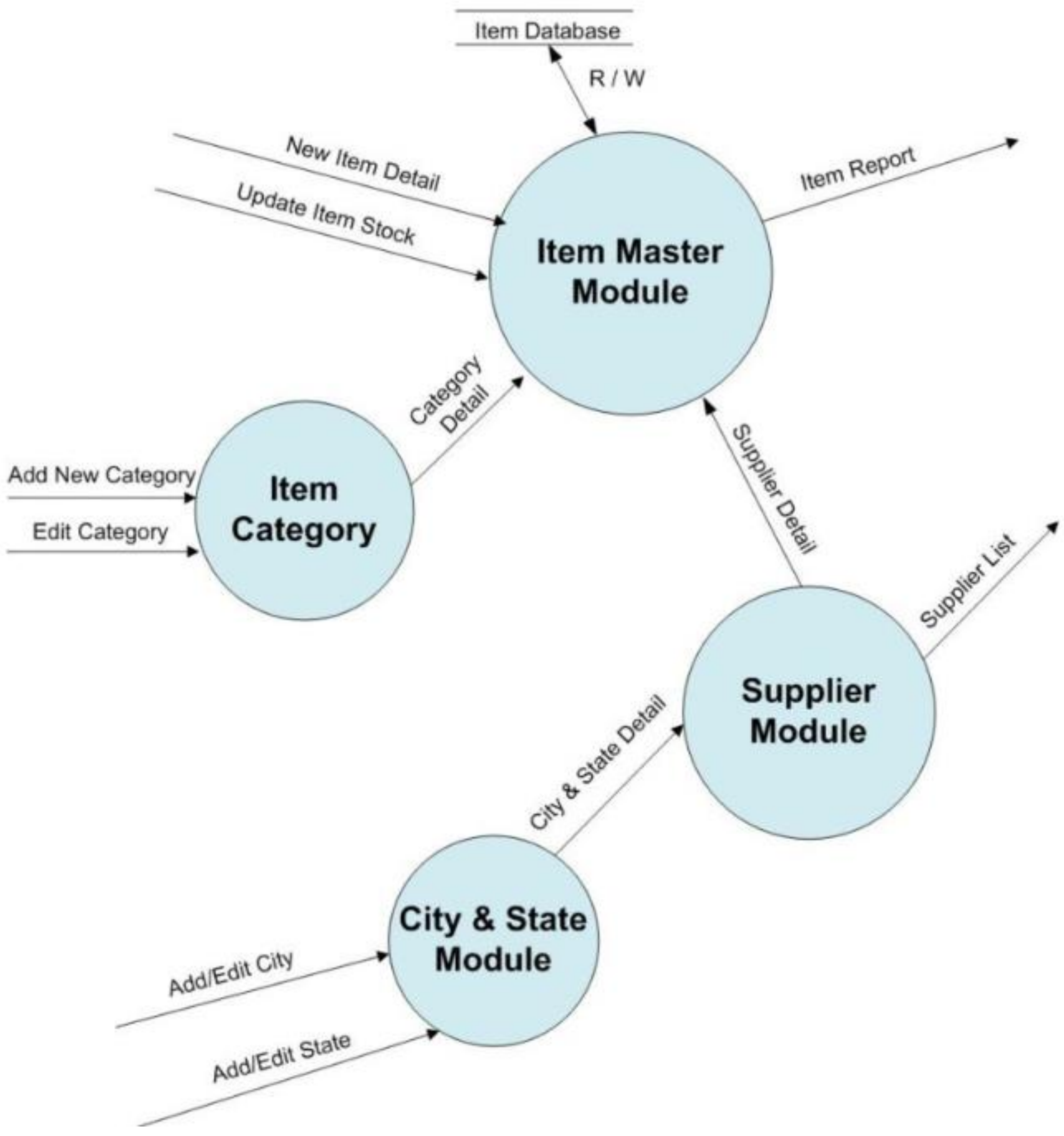


2	The online shopping platform facilitates a seamless shopping experience for users by offering a wide range of products across various categories. Users can browse through the extensive product catalogue, add items to their carts, and effortlessly place orders. To enhance user experience, the platform allows customers to manage their accounts, providing features such as updating personal information and viewing order history. Additionally, the system ensures secure transactions by enabling users to process payments using trusted payment gateways. Behind the scenes, administrators oversee the platform's operations, managing product listings, user accounts, and order processing. Data regarding products, users, and orders are stored in dedicated databases, ensuring efficient retrieval and management of information.	10	CO3	BL2

	• Design a DFD level – 0 and Level -1 diagram that shows how data moves through the online shopping system, illustrating interactions between users, administrators, and the underlying data stores and processes**.** (6 marks) • Convert the DFD into structure chart to define the various module structure of the software and show the interaction among different modules. (4 marks)			
Ans	**Schema**: With a proper dataflow as per the question DFD level- 0 – 2 marks DFD level – 1 – 4 marks DFD into Structure Chart (or Architectural mapping) with suitable transform flow and transaction flow between modules - 4 marks  Customer Login Confirmation Website Update Confirmation Admin			


Online Shopping Database
R/W
User Name & Password
New Customer
Customer List
Customer & Security Management module
Permission
Permission For update Tracking Detail
Add/Edit Item Category
Add/Edit Category Record
Add/Edit Supplier Record
Add/Edit City & State Record
Online Shopping Master Module
Update Tracking Status
Item Details
Order Tracking Module
Order Tracking Status
Items Purchase Request
Order Processing Module
Order Details
Payment
Invoice Detail
Order Tracking Detail
Report Request
Order Tracking No.
Shopping Report Module
Item Invoice
Order Tracking Details


Second Level DFD- Online Shopping System
Admin
Login to System
Check Roles of Access
Forgot Password
Check Credentials
Manage Modules
Send email to user
Manage Product Details
Manage Shopping Cart Details
Manage Customer Details
Manage Shipping Details
Manage Payment Details
Manage Order Details
Manage Report
Manage System Admins
Manage Roles of User
Manage User Permission

	 Item Database R / W New Item Detail Update Item Stock Item Master Module Item Report Category Detail Add New Category Edit Category Item Category Supplier Detail Supplier List Supplier Module City & State Detail City & State Module Add/Edit City Add/Edit State			
3.	You are designing a cloud-based document editing and collaboration tool that allows multiple users to edit documents simultaneously, track changes, and communicate via integrated chat. The system must be scalable, allowing future expansions such as AI-based writing assistance and advanced document versioning. Given the need for modularity, scalability, and future integration, your design approach must be carefully considered. • Identify the appropriate architecture style for the scenario and justify the same. (5 Marks) • Based on the above scenario, give your design assumptions for, o Any three types of relevant coupling to have enhanced modularity o Any two types of suitable cohesion for well-defined functionality (5 marks)	10	CO3	BL3
Ans	**Schema**: Architectural style Identification and Justification – 5 Marks Three relevant coupling - 3 marks Two relevant cohesion - 2 marks			

For the scenario described of designing a cloud-based document editing and collaboration tool, the most suitable architecture is Layered Architecture.

In a layered architecture:

Presentation Layer: This layer can handle functionalities like document editing interfaces, real-time collaboration interfaces, and integrated chat communication interfaces.

Business Logic Layer: This layer manages core functionalities such as tracking changes, document versioning, user authentication, and potentially integrating AI-based writing assistance in the future.

Data Access Layer: This layer deals with accessing and managing the document data stored in the cloud, including storage and retrieval of documents and associated metadata.

This architecture aligns well with the requirements of the scenario:

Modularity: It promotes modularity by dividing the system into distinct layers, each responsible for specific tasks.
Scalability: It allows for scalability by enabling each layer to scale independently based on demand.
Future Expansions: It supports future expansions such as integrating AI-based writing assistance and advanced document versioning by allowing for additional layers or modifications within existing layers without affecting the entire system.
Overall, a layered architecture provides a structured approach to designing a cloud-based document editing and collaboration tool that meets the requirements of simultaneous editing, change tracking, communication, scalability, and future expansions.

****

Communicational Coupling: Modules communicate with each other by passing messages, but they are not tightly coupled to each other's internal implementations. In this scenario, when a user makes changes to a document, the editing module can send a message to the collaboration module to update the document's status. Similarly, the chat module can send messages to notify users about new chat messages or updates in the document.

Data Coupling: Modules share data through well-defined interfaces or data structures, but they do not rely on each other's internal data representations. For instance, the document editing module can pass the edited content to the version control module using a standardized data format. Similarly, the chat module can exchange messages with other modules using predefined data structures.

Interaction Coupling: Modules are coupled through events or notifications, allowing them to react to specific events without direct dependencies on each other's implementations. when a user starts editing a document, an event is triggered and broadcasted to other users collaborating on the same document, enabling real-time updates. Similarly, the chat module can generate events for new messages, which are then distributed to users subscribed to the chat channel.

Functional Cohesion: Functional cohesion focuses on organizing modules based on related functionality or tasks. In the document editing and collaboration tool, functional cohesion can be achieved by grouping modules based on the specific functionalities they provide. For example:
Document Editing Module: Responsible for handling user interactions related to editing documents, such as adding, deleting, and formatting text.
Collaboration Module: Manages real-time collaboration features, including simultaneous editing, change tracking, and version control.
Chat Module: Handles integrated chat functionality, allowing users to communicate with each other while collaborating on documents.
AI Integration Module (Future Expansion): Provides AI-based writing assistance features, such as grammar checking, style suggestions, and language translation.
Each module focuses on a specific aspect of the overall system functionality, promoting clarity, maintainability, and scalability.

Communicational Cohesion: Communicational cohesion involves grouping modules based on the data they share and the interactions they have with each other. In the document editing and collaboration tool, communicational cohesion can be achieved by organizing modules that share common data or collaborate closely with each other. For example:

Document Editing Module & Collaboration Module: These modules may share data related to document content, changes, and user interactions. They closely collaborate to ensure real-time updates and synchronization of document changes across multiple users.
Collaboration Module & Chat Module: These modules may exchange messages and notifications to facilitate communication between users while collaborating on documents. They share common data such as user identities and chat messages, enabling seamless integration of chat functionality with the collaboration features.

4 For the give leap year pseudocode

```
function isLeapYear(year):
    if (year is not divisible by 4) then
        return False (not a leap year)
    else if (year is not divisible by 100) then
        return True (leap year)
    else if (year is divisible by 400) then
        return True (leap year)
    else
        return False (not a leap year)
```

- Design a test case for basis path testing. (6 Marks)
- Apply equivalence class partitioning for testing leap year checking functionality. (4 Marks)

Marks	CO	BL
10	CO4	BL4

Ans **Schema:**

- Identifying basis set of paths – 2 marks
  - Test case for all identified paths – 4 marks.
- Two valid ECP testcases – 2 marks
- Two invalid ECP Testcases – 2 marks

Test case using basis path Testing

Now, let's break down the basis paths for this pseudocode:

Path 1: year is divisible by 4, not divisible by 100, and divisible by 400. (True)
Path 2: year is divisible by 4, not divisible by 100, and not divisible by 400. (True)
Path 3: year is divisible by 4, divisible by 100, and not divisible by 400. (False)
Path 4: year is not divisible by 4. (False)
Now, let's design test cases for each basis path:

Test Case 1 (Path 1):

Input: year = 2000
Expected Output: True
Test Case 2 (Path 2):

Input: year = 2020
Expected Output: True
Test Case 3 (Path 3):

Input: year = 1900
Expected Output: False
Test Case 4 (Path 4):

	Input: year = 1999 Expected Output: False These test cases cover all the basis paths of the leap year pseudocode and ensure that the logic for determining leap years is thoroughly tested. **<u>Testing with Equivalence class Partitioning</u>** Equivalence class partitioning is a technique used to divide the input data into partitions or classes based on similar characteristics. Each partition represents a set of equivalent inputs that should be treated similarly by the system. Here's how we can apply equivalence class partitioning to the leap year problem: Valid years (Leap years): Class 1: Years divisible by 4 but not by 100 (e.g., 2004, 2008). Class 2: Years divisible by 400 (e.g., 2000, 2400). Invalid years (Non-leap years): Class 3: Years not divisible by 4 (e.g., 2001, 2003). Class 4: Years divisible by 100 but not by 400 (e.g., 1700, 1900). Based on equivalence class partitioning, we need to select representative test cases from each class. Here are the test cases: Test Case 1 (Class 1): Input: year = 2004 Expected Output: True Test Case 2 (Class 2): Input: year = 2000 Expected Output: True Test Case 3 (Class 3): Input: year = 2001 Expected Output: False Test Case 4 (Class 4): Input: year = 1900 Expected Output: False These test cases cover each equivalence class and help ensure that the leap year function behaves correctly for representative inputs within each class.			

5	You are provided with a simplified JavaScript function for a web-based application that calculates and displays the yearly subscription cost for a user based on the number of users and the chosen subscription plan. The application offers two plans: 'basic' at $100 per user per year and 'premium' at $150 per year. A discount is also applied for more than 100 users: a 10% discount for the 'basic' plan and a 15% discount for the 'premium' plan.
	function calculateYearlySubscriptionCost(numberOfUsers, planType)
	{
	   let costPerUser = planType === 'basic' ? 100: 150;
	   let discount = numberOfUsers > 100 ? (planType === 'basic' ? 0.1: 0.15) : 0;
	   return numberOfUsers * costPerUser * (1 - discount);
	}
		10	CO4	BL5

	Write five comprehensive sets of test cases to validate and verify the correct functioning of the calculateYearlySubscriptionCost function. Your test cases should cover all possible inputs and scenarios, including boundary cases. For each test case, specify the input values, the expected output, and a brief rationale for why the test case is necessary.			

Ans

Students should provide a series of test cases that cover:

- Both subscription plans ('basic' and 'premium').
- User counts below, at, and above the discount threshold (100 users).
- Edge cases, such as 0 users or negative numbers.
- Invalid inputs, such as non-numeric values or missing parameters.

Example test cases might include:

Test Case 1: calculateYearlySubscriptionCost(50, 'basic')

- Expected output: 5000
- Rationale: Tests the basic cost calculation without a discount.

Test Case 2: calculateYearlySubscriptionCost(101, 'premium')

- Expected output: 12835 (rounded if necessary)
- Rationale: Verifies the premium discount is applied correctly just above the threshold.

Test Case 3: calculateYearlySubscriptionCost(100, 'basic')

- Expected output: 10000
- Rationale: Edge case for the discount threshold.

Test Case 4: calculateYearlySubscriptionCost(-1, 'premium')

- Expected output: Error or 0 (depending on how error handling is defined)
- Rationale: Tests handling of invalid user counts.

Test Case 5: calculateYearlySubscriptionCost(50, null)

- Expected output: Error (invalid planType)
- Rationale: Verifies function's response to missing or invalid planType.

Marking Rubric (10 Marks):

Coverage of Scenarios (5 Marks):

- 1 mark per test case that correctly identifies and tests a unique scenario, including plan types, user count thresholds, and edge cases.

Correctness of Expected Outputs (3 Marks):

- 0.5 marks for each test case with the correct expected output.

Rationale and Error Handling (2 Marks):

- 1 mark for including test cases that check error handling or invalid inputs.
- 1 mark for providing a clear rationale for each test case, explaining its importance and what it verifies.