A customer can make a payment either by Card or by Wallet or by Cash or by Net banking.

Q1. Draw a Use Case Diagram - 4 Marks

A Use Case Diagram is a visual representation in UML that shows how different users (actors) interact with a system and what functionalities (use cases) the system provides.

Key Components of a Use Case Diagram:

❖ **System**: Represented by a rectangle, it defines the boundaries of the system being modeled.

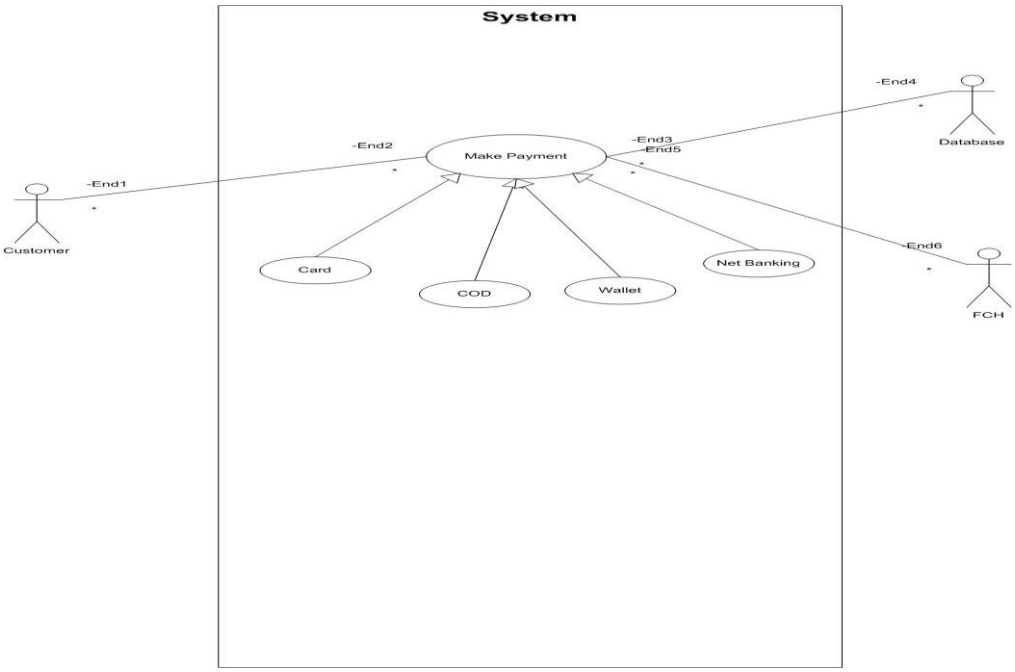
❖ **Actors**: External entities (users, other systems, hardware) that interact with the system. They are depicted as stick figures.

❖ **Use Cases:** Actions or tasks that actors can perform within the system. They are represented by ovals.

❖ **Relationships:** Show how actors and use cases interact, including

1. associations (simple interaction)
2. includes (one use case uses another),
3. extends (one use case extends another)
4. generalizations (inheritance)

It is mainly used during **requirement gathering** to understand **what the system should do** from the user's perspective.



Q2. Derive Boundary Classes, Controller classes, Entity Classes

Boundary Classes

Definition:

Boundary Classes are UML classes that represent the interaction between the system and external actors (users, other systems).

They are responsible for taking input from the actor and showing output back to the actor.

In other words, they work like the UI screens or APIs that act as the "boundary" between the system and the outside world.

Example:

- Payment Screen (Payment UI)
- Login Form (Login UI)

Controller Classes

Definition:

Controller Classes are UML classes that control the flow of logic between the boundary classes (UI) and the entity classes (data).

They coordinate actions, apply business rules, validate data, and manage workflows.

Example:

- Payment Controller (decides which payment mode to execute)
- Order Controller (manages order creation and validation)

Entity Classes

Definition:

Entity Classes are **UML classes that represent real-world data and business objects** that the system must store, retrieve, or manipulate.

They usually map to **database tables or persistent objects** and hold attributes and relationships.

Example:

- Customer (customer ID, name, email)
- Payment (payment ID, amount, status)

Q3. Place these classes on a three tier Architecture

3-Tier Architecture is a client-server architecture divided into Presentation, Business Logic, and Data Access layers. It enhances scalability, maintainability, and separation of concerns.

1. Presentation Layer – User Interface (e.g., website or mobile app)

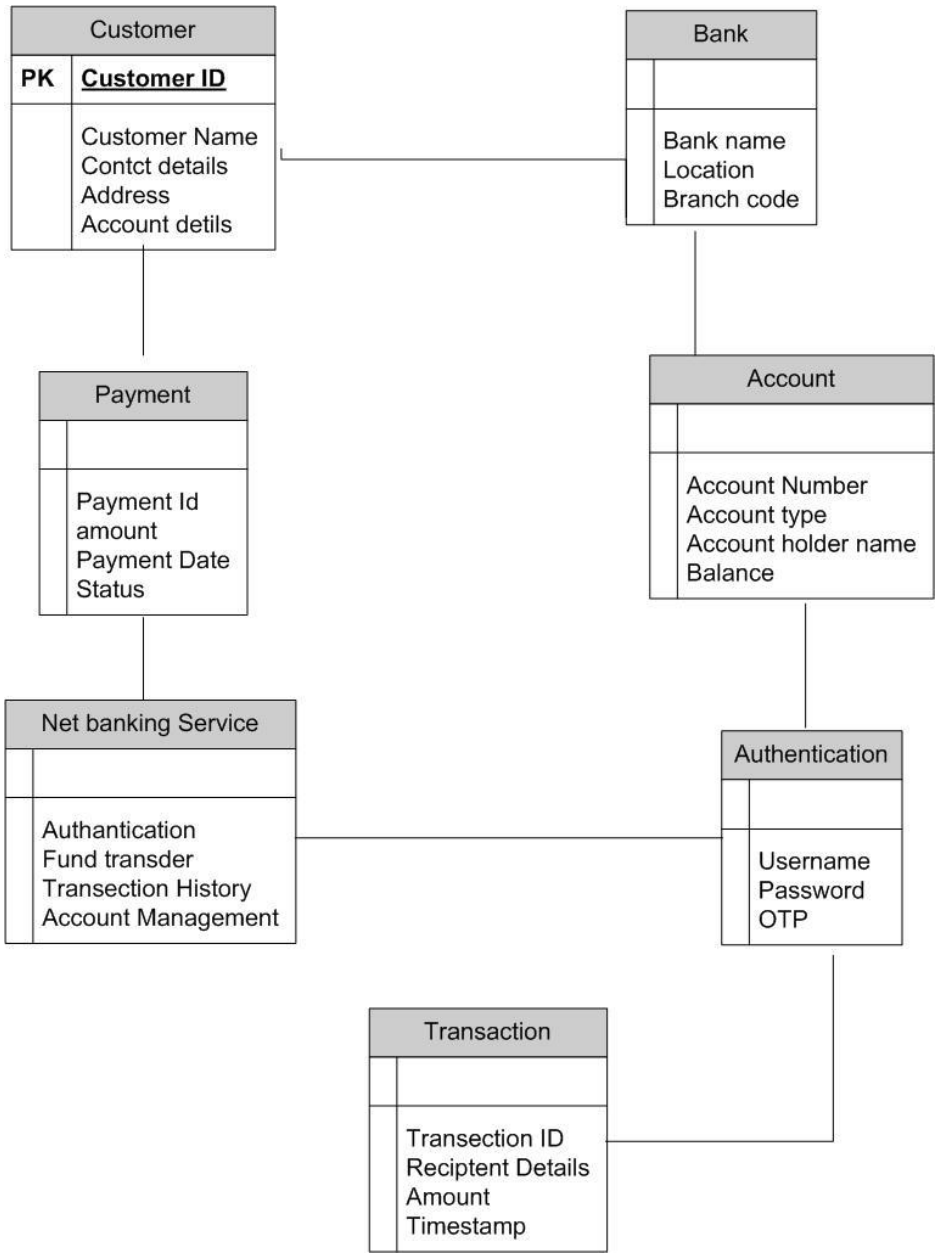
2. Business Logic Layer – Application logic (e.g., Java code validating transactions)

3. Data Layer – Database (e.g., MySQL storing product details) This structure allows each layer to be developed, updated, and scaled independently.

Layer	Class Type	Example	Justification (Why it belongs here)
Presentation	**Boundary Classes**	`Payment UI`, `CardUI`	These classes handle **user interaction**, take input, and display output but do not contain business logic.
Business Logic Layer	**Controller Classes**	`Payment Controller`	This class **controls the payment process**, validates data, and coordinates between UI and data.
Data Layer	**Entity Classes**	`Customer`, `Payment`, `Card Details`,	These classes represent **business data** and are stored in the database for processing and retrieval.

Q4. Explain Domain Model for Customer making payment through Net Banking

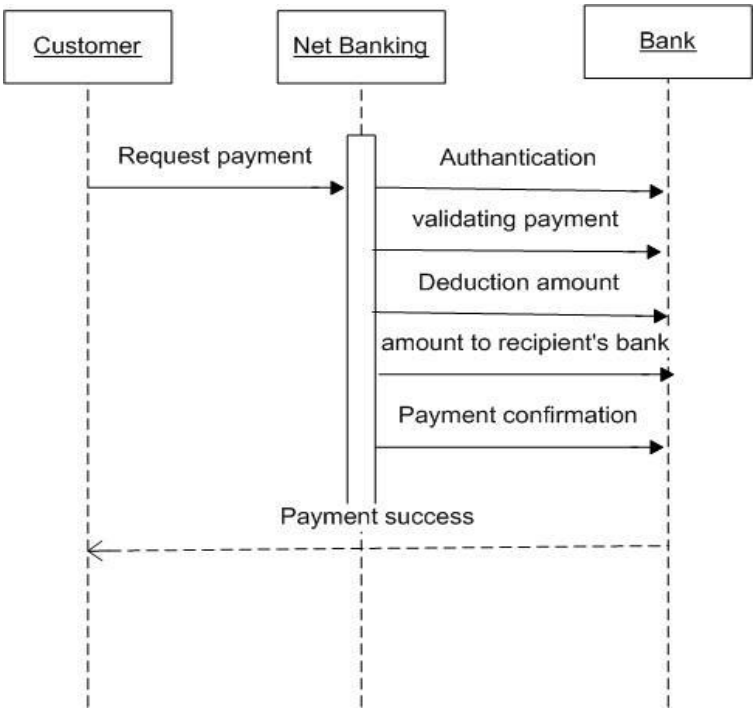
A Domain Model is a conceptual representation of the key entities, their attributes, ehaviors, and relationships within a specific problem domain or business area. It is typically used in software development and business analysis to capture and communicate the structure and rules of the domain that the system will address.



Q5. Draw a sequence diagram for payment done by Customer Net Banking

A Sequence Diagram is a UML behavioral diagram that shows how objects interact in a particular scenario, in the order they occur.
It focuses on:

- Objects/Actors involved
- Messages passed
- Order of events (top-to-bottom flow)



Q6. Explain Conceptual Model for this Case

A Conceptual Model is a high-level representation of the system that describes what the system should do rather than how it is implemented.
It identifies:

- Key concepts (entities/objects) in the problem domain
- Relationships between those concepts
- Overall business rules or behavior

It is usually shown as a class diagram (without technical details) and helps business stakeholders and developers have a shared understanding of the system.

Q7. What is MVC architecture? Explain MVC rules to derive classes from use case diagram and guidelines to place classes in 3-tier architecture

MVC (Model–View–Controller) is a software architectural pattern that divides an application into three interconnected components:

1. Model – Represents data and business logic
2. View – Represents UI (presentation layer)
3. Controller – Represents application logic and flow control

Its main goal is to separate concerns, making the system easier to maintain, test, and scale.

MVC Rules to Derive Classes from Use Case Diagram

When deriving classes from a Use Case Diagram, we follow these rules:

1. Identify Actors & Use Cases → These help derive Boundary Classes (View).
    a. Each actor–system interaction leads to a Boundary Class (e.g., PaymentUI for Make Payment use case).
2. Identify System Operations → These become Controller Classes.
    a. Each use case generally maps to one Controller Class that coordinates the process.
3. Identify Entities (Business Concepts) → These form Model (Entity Classes).
    a. Extract nouns from use case description to identify real-world data objects like Customer, Payment, BankAccount.

Guidelines to Place Classes in 3-Tier Architecture

Tier	What it Contains	Class Type (MVC Component)
Presentation Layer (UI Layer)	Screens, forms, input/output handling	View (Boundary Classes)
Business Logic Layer	Workflow control, validation, decision-making	Controller Classes
Data Layer (Persistence Layer)	Database entities, data access objects	Model (Entity Classes)

Q8. Explain BA contributions in project (Waterfall Model – all Stages

Stage	Activities	Artifacts (Deliverables)	Resources (People/Tools)
Pre-Project	- Identify business need - Conduct feasibility study - Prepare business case	Business Case Document, Feasibility Report	Business Analyst, Business Sponsor, SME
Planning	- Define scope - Prepare project plan, schedule, budget - Identify risks	Project Charter, Project Plan, Risk Register	Project Manager, BA, Stakeholders
Project Initiation	- Stakeholder identification - Kick-off meeting - Assign roles and responsibilities	Stakeholder Register, RACI Matrix, Communication Plan	Project Manager, BA, Team Leads

Requirements Gathering	- Conduct interviews, workshops, surveys - Elicit functional & non-functional requirements	BRD (Business Requirement Document), User Stories, Use Case Diagrams	Business Analyst, SMEs, Stakeholders
Requirements Analysis	- Analyze & prioritize requirements - Resolve conflicts - Validate with stakeholders	SRS (System Requirement Specification), RTM (Requirement Traceability Matrix)	BA, Product Owner, Solution Architect
Design	- Create solution design, data models, mockups - Review with stakeholders	Solution Design Document, Wireframes, Data Model	Solution Architect, BA, UI/UX Designer
Development	- Coding and unit testing - Build integrations & APIs	Source Code, Technical Documentation	Developers, Tech Lead, DevOps
Testing	- Prepare test cases - Execute SIT (System Integration Testing) - Log defects	Test Plan, Test Cases, Defect Logs	QA/Testers, BA (for validation), Test Tools (JIRA, Selenium)
UAT (User Acceptance Testing)	- Coordinate with end users - Execute UAT scripts - Get sign-off	UAT Plan, UAT Test Cases, UAT Sign-off Document	End Users, BA, QA, Business Sponsor

Q9. What is conflict management? Explain using Thomas – Kilmann technique

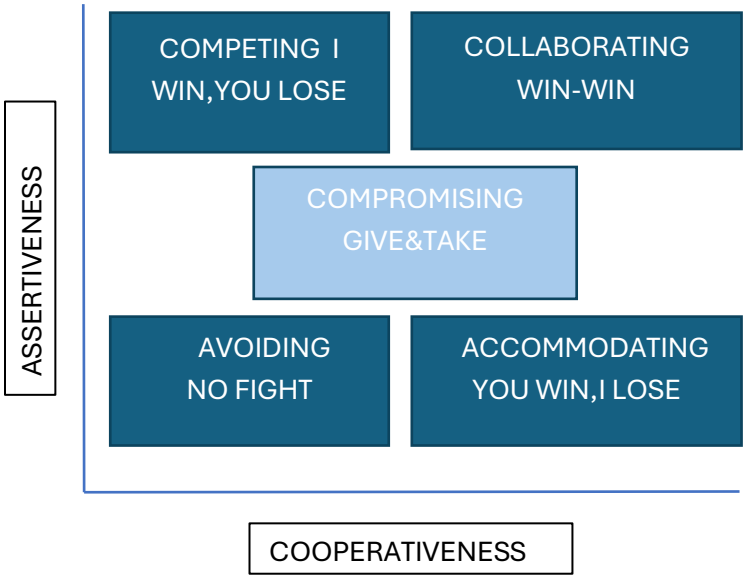
Conflict management is the process of resolving conflicts or disagreements between individuals or groups in a constructive manner.
Thomas Kilmann technique is a widely used tool for assessing conflict resolution styles & guiding individuals in selecting appropriate strategies to manage conflicts.

It identifies **five conflict-handling styles** based on two dimensions:

- **Assertiveness:** The extent to which a person tries to satisfy their own concerns.
- **Cooperativeness:** The extent to which a person tries to satisfy the concerns of others.

These two dimensions create **five approaches** to handling conflict:

1. **Competing (I Win, You Lose)**
   You focus on your own needs and push your solution.
   - Use when you must make a quick decision or enforce rules.
2. **Collaborating (Win–Win)**
   You and the other person work together to find the best solution for both.
   - Use when both sides' ideas are important and you have time to talk.
3. **Compromising (Give & Take)**
   Both sides give up a little to find a middle solution.
   - Use when you need a quick, fair agreement.
4. **Avoiding (No Fight)**
   You stay away from the conflict or delay talking about it.
   - Use when the issue is small or emotions are too high.
5. **Accommodating (You Win, I Lose)**
   You let the other person have their way to keep peace.
   - Use when the relationship is more important than the issue.



Q10. List down the reasons for project failure

Project failure happens when a project does not meet its goals within the agreed scope, time, cost, or quality constraints, or when it does not deliver the expected value to stakeholders.

Common Reasons for Project Failure

1. **Unclear Objectives and Scope**
    a. Lack of well-defined goals or deliverables.
    b. Frequent scope changes (scope creep) without proper control.
2. **Poor Planning**
    a. Inadequate project schedule, unrealistic deadlines.
    b. No proper risk assessment or resource planning.
3. **Lack of Stakeholder Engagement**
    a. Stakeholders not involved during key stages.
    b. Misunderstanding of stakeholder expectations.
4. **Ineffective Communication**
    a. Lack of clear updates between team members and stakeholders.
    b. Miscommunication leading to delays and errors.
5. **Insufficient Resources**
    a. Lack of skilled people, technology, or budget.
    b. Overloaded team members causing burnout.
6. **Weak Project Management**
    a. No proper monitoring and control.
    b. Poor decision-making or lack of leadership.
7. **Inadequate Risk Management**
    a. Not identifying potential risks early.
    b. No mitigation plan to handle issues when they arise.
8. **Technical Challenges**
    a. Wrong choice of technology or tools.
    b. Integration failures or quality issues in deliverables.
9. **Poor Change Management**
    a. Inability to handle changes in requirements or priorities.
    b. Resistance from end-users.
10. **Lack of User Involvement / Poor Testing**
    a) End-users not testing early.
    b) Deliverables do not meet actual business needs.
11. **Budget Overruns and Delays**
    a) Costs exceed approved budget.
    b) Timeline slips without corrective action.

Q11. List the Challenges faced in projects for BA

1. **Unclear Requirements**
    Stakeholders are unsure of what they want, leading to confusion and rework.

2. **Changing Requirements (Scope Creep)**
   New requirements keep getting added during the project without proper impact analysis.
3. **Multiple Stakeholders with Conflicting Needs**
   Different stakeholders want different solutions, making it hard to prioritize.
4. **Communication Gaps**
   Misunderstandings between business teams, developers, and testers.
5. **Limited Domain Knowledge**
   BA needs time to understand the business process and industry terminology.
6. **Time Constraints**
   Less time given for requirement gathering, documentation, or analysis.
7. **Technical Limitations**
   Requirements may not be feasible due to system or technology constraints.
8. **Data Quality Issues**
   Incomplete, inconsistent, or missing data can affect analysis.
9. **Stakeholder Availability**
   Stakeholders may be busy and not available for discussions or approvals.
10. **Resistance to Change**

- End-users may not accept new systems or processes easily.

11. **Integration with Other Teams**

- Coordinating with development, testing, and operations teams can be challenging.

Q12. Write about Document Naming Standards

A document numbering standard is a systematic approach to assigning unique identifiers to various documents created and used throughout the development process.
Ex. Suppose we have a project with the ID "PROJ123," and we're working with a Requirements
Specification Document.
Project ID: PROJ123
Document Type: REQ
Version: 1.0
Date: 2025-09-24
The document identifier could be: PROJ123-REQ-1.0- 2025-09-24

Q13. What are the Do's and Don'ts of a Business analyst

Do's (Best Practices)	Don'ts (Common Mistakes)
Understand business processes and goals clearly	Don't assume requirements without verification
Engage all key stakeholders and gather input	Don't ignore important stakeholders or end-users
Ask the right questions to uncover hidden needs	Don't use too much technical jargon with business users
Document requirements clearly and completely (BRD, SRS, User Stories)	Don't skip documentation or keep things only verbal
Prioritize requirements based on business value	Don't overpromise on timelines or feasibility
Facilitate collaboration between business, developers, and testers	Don't delay communication or hide risks/issues
Validate and get sign-off on requirements	Don't neglect the impact of changes (scope creep)
Manage changes with proper change control process	Don't be biased towards any stakeholder's opinion
Maintain professionalism and stay solution-focused	Don't resist feedback or avoid constructive criticism
Keep learning domain knowledge, tools, and techniques	Don't forget to ensure requirements are testable and support UAT

Q14. Write the difference between packages and sub-systems

- Packages Collection of components which are not reusable in nature.
  Ex: Application development companies work on Packages.
- Sub systems: Collection of components which are reusable in nature.
  Ex: Product development companies work on Sub systems.

Package	Sub-System
Just a folder to group related elements	A mini-system that performs a function
Used for **organization**	Used for **functionality**
Cannot run by itself	Can run and interact with other sub-systems
Example: Folder containing all payment classes	Example: Payment module handling all transactions

Q15. What is camel-casing and explain where it will be used

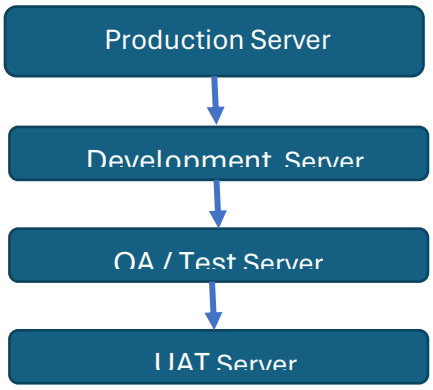
Camel-casing is a way of writing names where the first word starts with a small letter, and every next word starts with a capital letter – with no spaces between words. It is called camel-case because the capital letters look like the humps of a camel.

Camel-casing is mostly used in programming and IT documentation, such as:

- Variable Names → `totalAmount`, `customerName`
- Function Names → `calculateSalary()`, `getUserData()`
- File Naming (sometimes) → `projectPlanDocument`
- Database fields/columns → `orderDate`, `productPrice`

Q16.Illustrate Development server and what are the accesses does business analyst has?

- A development server refers to a dedicated environment or server that is used during the software development process.
- It provides a platform for developers and testers to build, test and debug applications before they are deployed to a production environment.
- As a BA, we have only limited access only.
    - **Read-Only Access** → To see data, check new screens, and validate requirements.
    - **Testing/UAT Access** → To perform sanity checks and confirm requirements are met.
    - **Log Access (sometimes)** → To check error logs or trace issues (not modify code).
    - **No Code Change Access** → BAs usually cannot edit or deploy code.
    - **No Production Access** → Direct production server access is usually restricted



Q17. What is Data Mapping

- Data mapping is the process of connecting data from one source to another.
- It's like creating a guide or map that shows how data in one place corresponds to data in another place.
- This is especially important when you're moving data between different systems or databases.
    - Ensures **data accuracy** and **consistency**
    - Reduces **data loss or mismatch**
    - Helps developers and testers know exactly where data goes

Q18. What is API. Explain how you would use API integration in the case of your application Date format is dd-mm-yyyy and it is accepting some data from Other Application from US whose Date Format is mm-dd-yyyy

- API Integration means connecting two or more applications so they can send and receive data automatically.
- It helps avoid manual data entry and ensures real-time data sharing.

Scenario:

- our application uses dd-mm-yyyy (Indian format).
- Another US-based application sends dates in mm-dd-yyyy format through an API.

Solution Using API Integration:

1. Receive Data via API
    a. API receives data from the US application.
    b. Example: `"date": "05-24-2025"` (US format → 24th May 2025).
2. Convert the Date Format
    a. Use a date conversion logic in your application:
        i. Input: mm-dd-yyyy (05-24-2025)
        ii. Output: dd-mm-yyyy (24-05-2025)
3. Save/Display Data
    a. Store the converted date in your database or display it correctly to users.

Benefits of Using API Integration

- No manual data entry → fewer errors
- Automatic format conversion → consistent data
- Real-time updates → better user experience