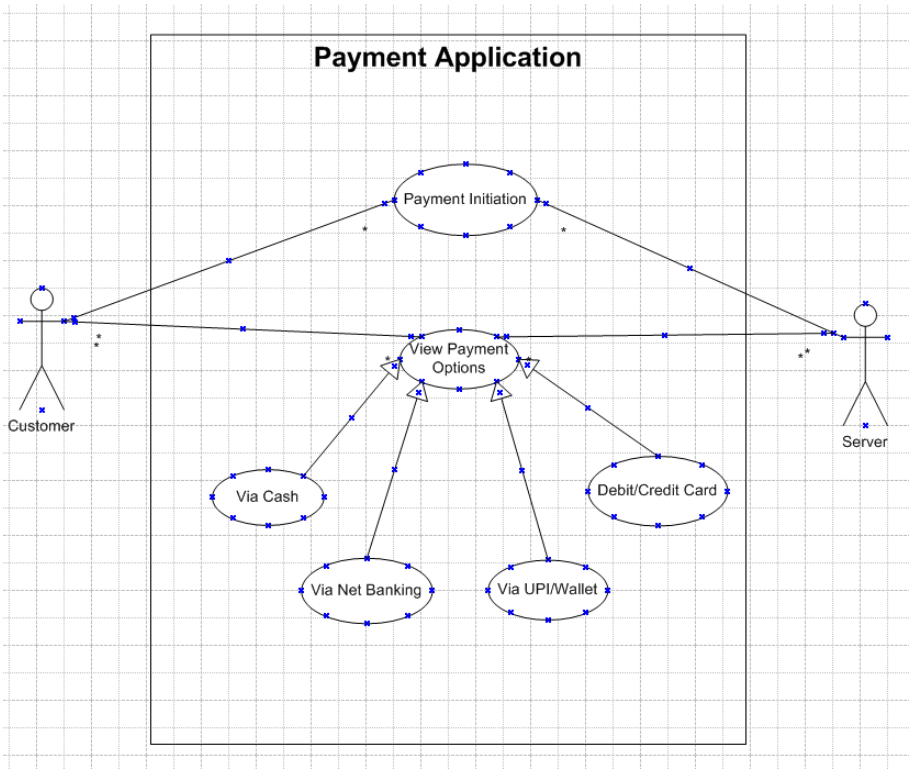
Question: 1 – Create use case diagram



Question: 2 - Boundary Classes, Controller classes, Entity Classes.

Boundary Classes handle **interaction between actors and the system**, such as user interfaces or forms.
Controller Classes manage **requests and commands**, coordinating between boundaries and entities.
Entity Classes represent **business data and logic**, typically mapping to database tables.

Boundary Classes

- Represent system interfaces (UI forms, queries).
- Mediate communication between users/actors and the system.

Ex: Product Listing Screen

ATM Network Interface

Controller Classes

- Handle user commands and requests.
- Orchestrate flow between boundary and entity classes.

Ex: Product Listing Controller

Withdrawal Controller

Entity Classes

- Represent stored data and business rules.
- Persist data, often used by backend designers.

Ex: Product

Account

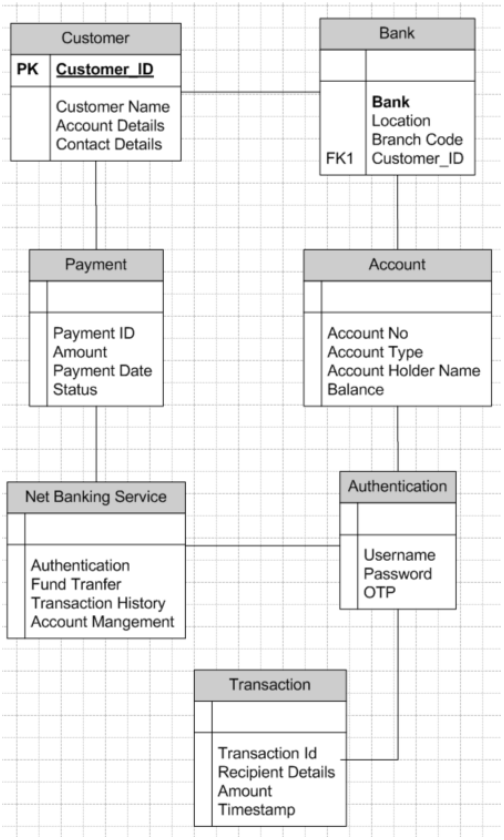
Question:3 Place these classes on Three tire Architecture

- User Layer
  Product Listing Screen
  ATM Network Interface

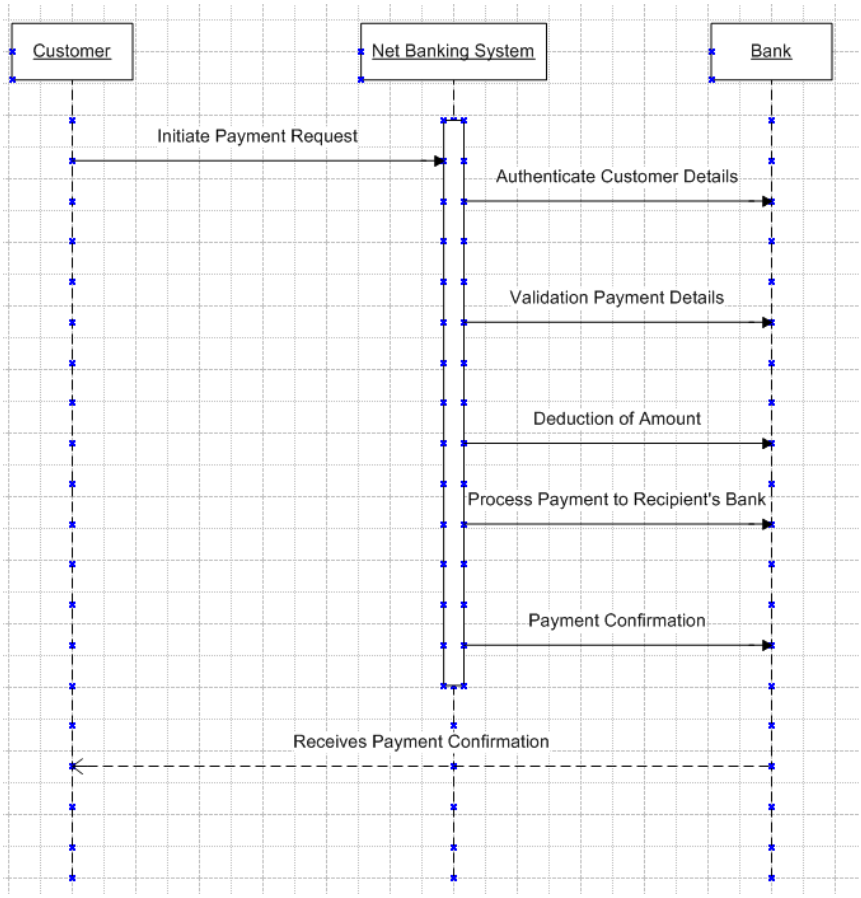
- Business Logic
  Product Listing Controller
  Withdrawal Controller

- Data Tier
  Product (Entity Class)
  Account (Entity Class)

Question: 4 – Domin Model for Customer making payment through Net Banking



Question: 5 – Sequence Diagram for payment done by customer Net Banking



Question: 6 – Conceptual Model for this Case

A conceptual model offers a simplified, high-level overview of a system to support understanding, visualization, and effective communication of key domain aspects.

It presents essential information in a clear and accessible way, making the domain easier to grasp.

Key components of a conceptual model include:

1. **Entities:** Examples are Customer, Product, Order, and Payment.
2. **Attributes:** Such as customerId, name, email, and phoneNumber.
3. **Relationships:** For instance, a Customer places an Order.

Question: 7 – MVC Architecture

The Model-View-Controller (MVC) is an architectural framework that structures an application into three main logical parts: the Model, the View, and the Controller.

Each component has a distinct responsibility:

- **View**: This component manages the application's presentation layer, which is what the user sees and interacts with.
- **Model**: It is responsible for handling the application's data and the underlying business logic that governs it.
- **Controller**: This part serves as an intermediary, facilitating communication and data flow between the Model and the View.

MVC Architecture Rules

1. Combination of One Actor and a use case results in one Boundary class

2. Combination of Two Actors and a use case results in two Boundary classes

3. Combination of Three Actors and a use case results in Three Boundary classes and so on....

Note: only one primary actor is to be considered with a use case.

4. Use case will result in a controller class

5. Each Actor will result in one entity class

Question: 8 – BA Contributions in waterfall model

Stage	Activities	Artifacts & Resources
Pre-project	- Conduct feasibility studies - Prepare the business case - Identify high-level business problems/opportunities	- Business Case - Feasibility Study Report - Problem Statement
Planning	- Define project scope and objectives - Identify and analyze stakeholders - Plan requirements management approach	- Scope Statement - Stakeholder Register - Requirements Management Plan
Project Initiation	- Assist in developing the project charter - Elicit high-level requirements	- Project Charter - High-level Solution Document
Requirements Gathering	- Elicit detailed requirements (interviews, workshops, surveys) - Document user stories and use cases	- Interview Notes - Workshop Minutes - Use Case Diagrams - User Stories
Requirements Analysis	- Analyze, prioritize, and model requirements - Create detailed requirements documentation - Validate requirements with stakeholders	- Business Requirements Document (BRD) - Functional Requirements Document (FRD) - Requirements Traceability Matrix (RTM) - Prototypes/Wireframes
Design	- Review design documents to ensure they meet requirements - Clarify requirements for the design/technical team	- Reviewed Design Documents - Updated Traceability Matrix
Development	- Provide clarifications to the development team - Manage change requests and conduct impact analysis	- Change Request Logs - Clarification Log
Testing	- Review test plans and test cases - Assist in defect triage and prioritization	- Reviewed Test Plans & Cases - Defect Reports
UAT	- Plan and coordinate User Acceptance Testing (UAT) - Support business users during testing - Obtain final sign-off from stakeholders	- UAT Plan & Scenarios - UAT Sign-off Document

Question:9 – Conflict Management and Thomes Kilmann Theory

Conflict management is the practice of productively resolving disagreements between individuals or groups. A well-known method for this is the Thomas-Kilmann technique, which helps assess a person's conflict resolution style and guides them in choosing suitable strategies to handle such situations.

5 options of Conflict Management

- Competing
- Avoiding
- Accommodating
- Collaborating
- Comprising

5 Steps to Conflict Management

- Identify the conflict
- Discuss the details
- Agree with the root problem
- Check for every possible Solution for the conflict.
- Negotiate The Solution to avoid the future Conflicts

Question: 10 - List down the reasons for project failure

- Unclear Objectives and Requirements
- Inadequate Risk Management
- Poor Communication
- Poor Planning
- Scope Creep
- Lack of Stakeholder Engagement
- Resource Constraints
- Technical Challenges

Question: 11 – Challenges faced in project for BA

- Scope Creep and Scope Management
- Managing Stakeholder Expectations
- Unclear or Changing Requirements
- Time and Resource Constraints
- Quality Assurance and Testing
- Documentation and Knowledge Management
- Technology Constraints and Complexity

Question:12 - Document Naming Standards

A document numbering standard provides a systematic method for assigning a unique identifier to every document created during a project's development cycle.

For instance, imagine a project with the ID **PROJ123** needs a Requirements Specification Document. A unique identifier for this document can be created by combining several key details:

- **Project ID:** PROJ123
- **Document Type:** REQ (short for Requirements)
- **Version:** 1.0
- **Date:** 2025-09-05

When these elements are combined, they form a complete and unique document identifier such as: PROJ123-REQ-1.0-2025-09-05.

Question: 13 - the Do’s and Don’ts of a Business analyst

Do’s Don’ts and Challenges

• Never say NO to client

• Never imagine anything in terms of GUI

• There is NO word called as “By Default”

• Consult an SME for clarifications in Requirements

Challenges

• Obtaining sign-off on requirement

• Change Management- with respect to cost and timelines

• Coordination between developers & testers

• Conducting meetings

• Driving client for UAT completion

• People Management (coordinating with different people and different teams)

Question: 14 - The difference between packages and sub-systems

Based on the definitions provided in the image, here are different examples for packages and sub-systems:

- **Packages**: A collection of components that are not reusable in nature.

Example: A custom-built reporting feature for a specific client's financial software. The components are tailored to that client's unique business processes and are not intended for use in other applications.

- **Sub-systems**: A collection of components that are reusable in nature.

Example: A user authentication module that can handle login, registration, and password recovery. This module is designed to be integrated into multiple different applications (e.g., a web portal, a mobile app, and an internal tool) with minimal changes.

Question:15 - camel-casing and explain where it will be used

Camel-casing is a naming convention in computer programming used for naming elements like variables, functions, and identifiers.

In this style, the first word begins with a lowercase letter, and each subsequent word starts with an uppercase letter.

Different Examples:

- **Variable:** userName
- **Function:** calculateFinalScore
- **Identifier:** customerAddressDetails

Question:16 - Illustrate Development server and what are the accesses does business analyst has

A **Development Server** is an isolated, non-public environment where developers actively write, build, and test new software and code changes. It is the first stop in the development lifecycle, allowing developers to experiment and debug their work without affecting the live application that end-users interact with.

Business Analyst Access

A Business Analyst typically has **limited, often read-only, access** to the development server. Their primary role on this server is to:

- **Review and Verify**: Check early versions of features to ensure they align with business requirements.
- **Provide Quick Feedback**: Offer immediate input to developers during the building phase.

BAs do not perform formal testing or make changes on the development server, as it is often unstable. They have more extensive, hands-on access to later environments like the **UAT (User Acceptance Testing) or Staging server**, which are specifically designed for validation and formal testing before a feature goes live.

Question: 17 - Data Mapping

Data mapping is the process of matching data fields from one source system to their corresponding fields in a target system. It creates a "map" that guides how data is transferred and transformed, ensuring accuracy and consistency when integrating or migrating data between different databases or applications.

For example, a data map would specify that a source field named First_Name should be matched to a target field called FirstName and could include a rule to transform a state value like "California" into its abbreviation "CA" for the target system.

Question:18 – API

An **API (Application Programming Interface)** is a set of rules and protocols that allows different software applications to communicate with each other. It acts as an intermediary, enabling one system to access and use the data or functionality of another without needing to know the specifics of its internal implementation. This allows for seamless **API integration**, which is the process of connecting two or more applications to automate tasks and share data in real-time.

API Integration for Date Format Conversion

In the scenario where your application uses the date format **dd-mm-yyyy** and needs to accept data from a US application using **mm-dd-yyyy**, an API integration is the ideal solution to handle this discrepancy automatically.

Here's how you would use it:

1. **Receive the Data**: The US application sends a date, for example, 09-02-2025 (September 2, 2025), to your application through an API endpoint.
2. **Transform in the API Layer**: Before the data is saved into your system, the API integration layer intercepts it. This layer contains logic specifically designed to handle data transformation.
3. **Parse and Reformat**: The API logic will parse the incoming string, recognizing it as the mm-dd-yyyy format. It will then use a function (like formatDate() or a similar utility in the programming language) to reformat this date into your application's required dd-mm-yyyy format. In this case, 09-02-2025 is converted to 02-09-2025.
4. **Use the Corrected Data**: Your application then receives the date in the correct format (02-09-2025) and can process or store it without any errors.

This process ensures that the data is consistent and accurate within your system, without requiring any changes to the underlying code of either the source or target application. The API acts as the "translator" for the date formats, making the integration seamless.