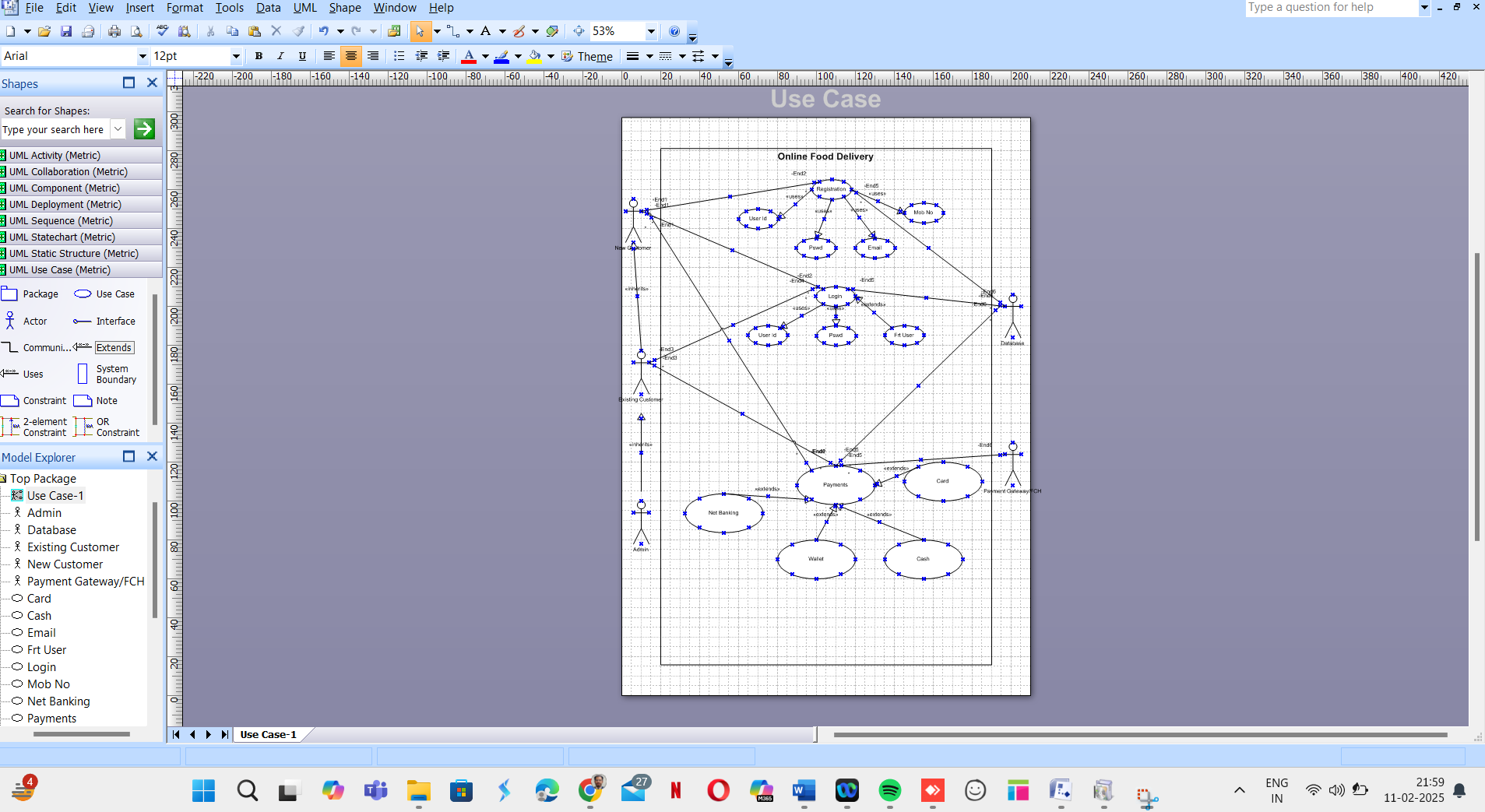
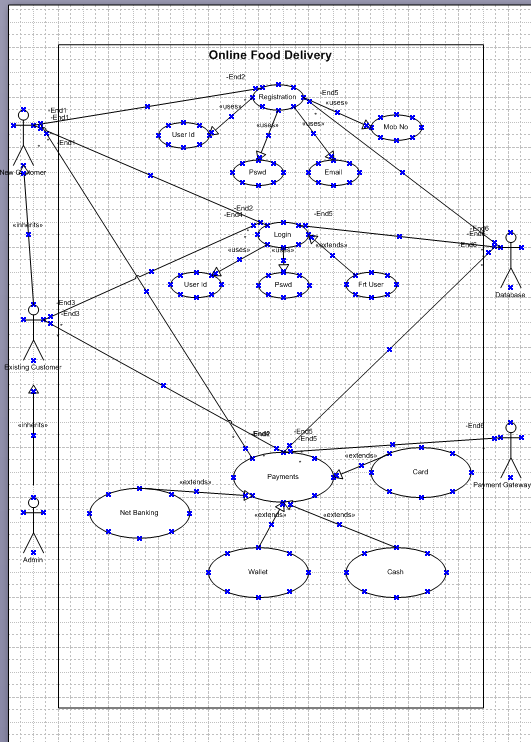
**Capstone Project 3**

**Q1 Draw a Use Case Diagram - 4 Marks**

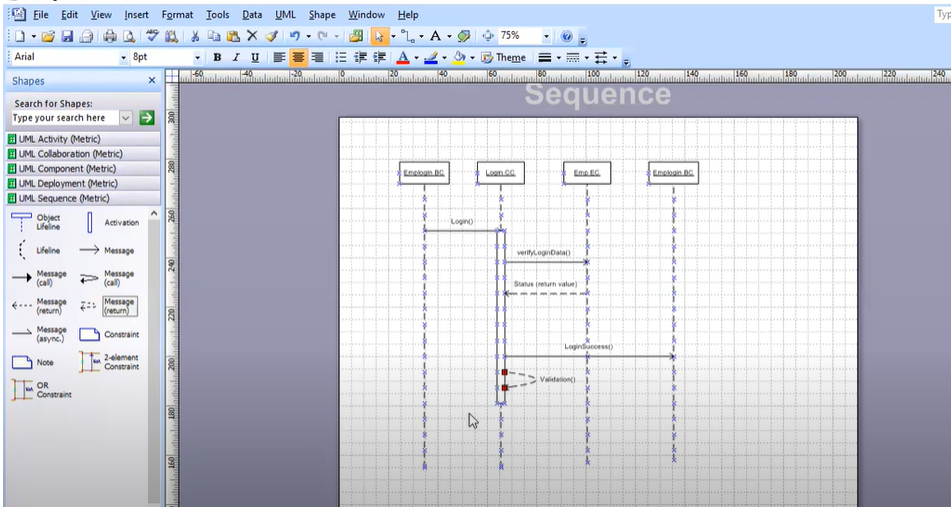
A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

****

****

**Q2. Derive Boundary Classes, Controller classes, Entity Classes. - 4 Marks**

* Boundary Classes: Payment Interface, Card Interface, Wallet Interface, Net Banking Interface
* Controller Classes: Payment Controller, Transaction Controller
* Entity Classes: Customer, Payment, Transaction, Bank Details



**Q3. Place these classes on a three tier Architecture. - 4 Marks**

**1. Presentation Tier:**

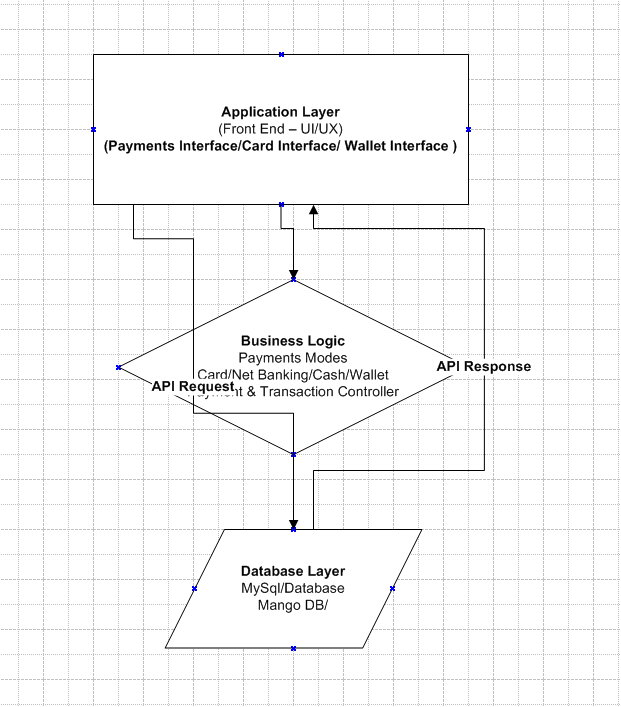
The presentation tier is the user interface or client layer of the application. It is responsible for presenting data to the user and receiving input from the user. This tier can be a web browser, mobile app, or desktop application. Here Payment Interface, Card Interface, Wallet Interface, Net Banking Interface

**2.Business Logic Tier:**

The Logic tier is the middle layer of the 3-tier architecture. It is responsible for processing and managing the business logic of the application. This tier communicates with the presentation tier to receive user input and communicates with the data management tier to retrieve or store data. This tier may include application servers, web servers, or APIs. Here Payment Controller, Transaction Controller will be in business logic layer.

**3.Data Management Tier:** The data management tier is the bottom layer of the 3-tier architecture. It is responsible for managing and storing data. This tier can include databases, data warehouses, or data lakes. The data management tier communicates with the application tier to receive or store data. Here the transaction details along with Customer, Payment, Transaction, Bank Details

Example: MYSQL, Oracle database.

****

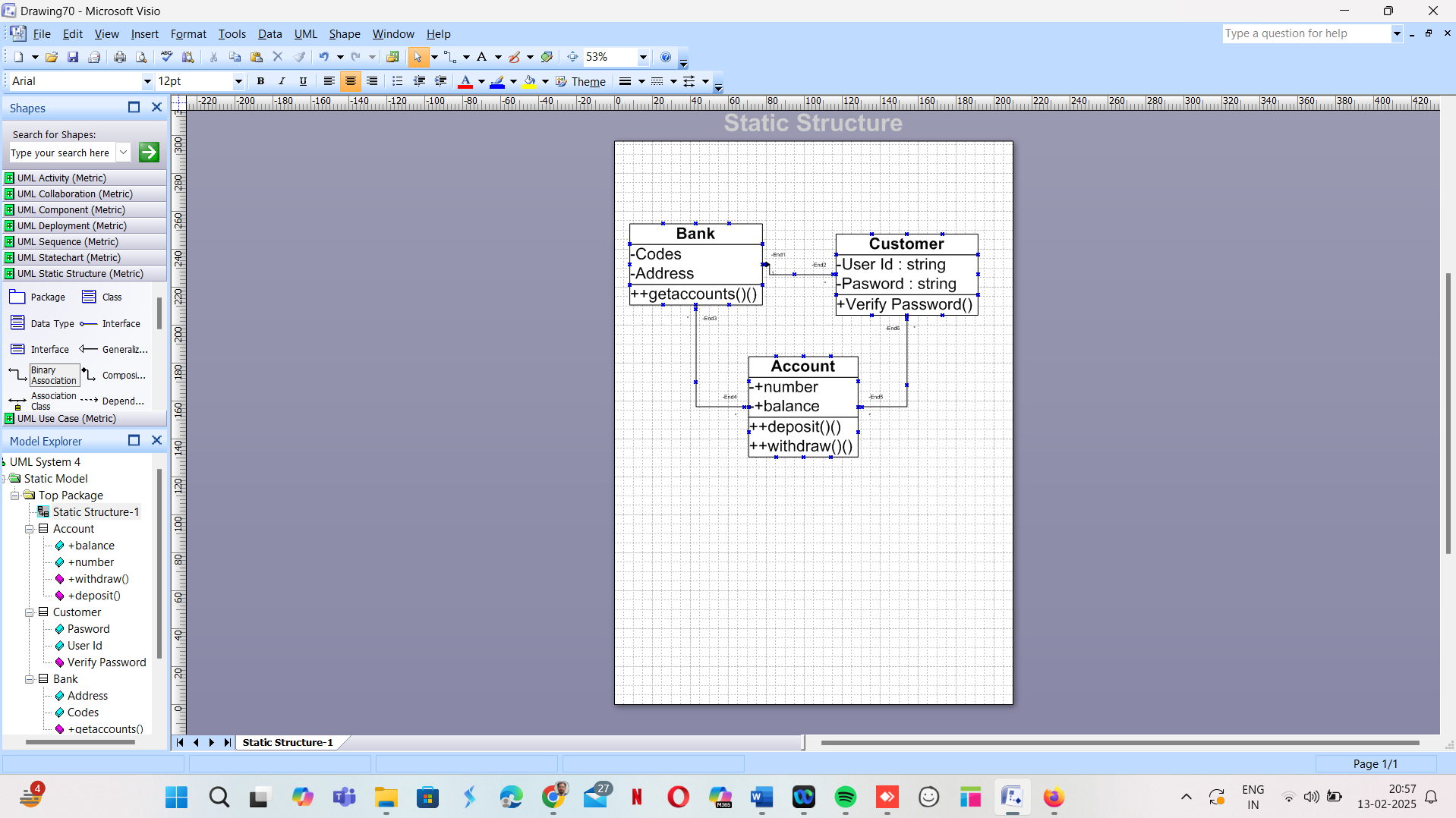
**The Three tier Architecture could be broken into below**

* Presentation Layer: Payment Interface, Card Interface, Wallet Interface
* Business Logic Layer: Payment Controller, Transaction Controller
* Data Layer: Customer, Payment, Transaction, Bank Details

**Q4. Explain Domain Model for Customer making payment through Net Banking - 4 Marks**

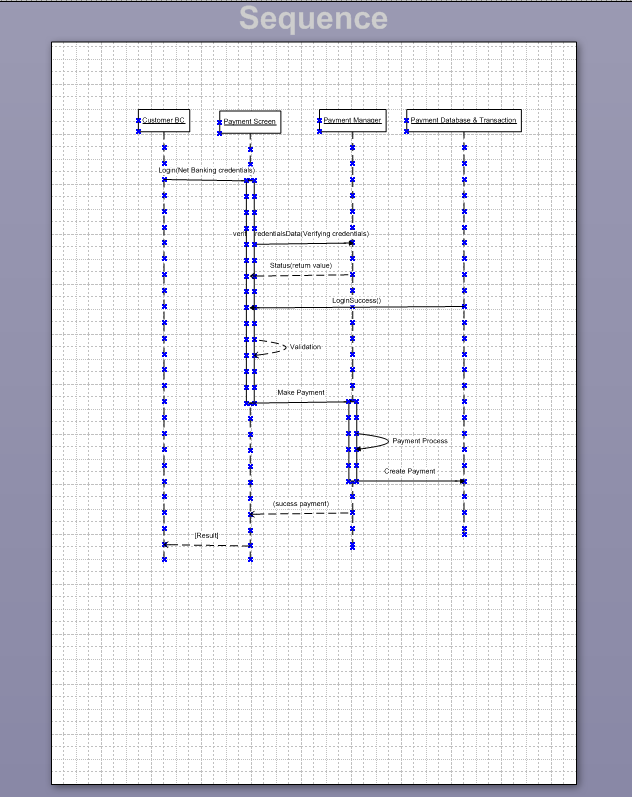
The domain model represents objects involved in net banking payments, such as:

* Customer initiates payment
* Net Banking Interface facilitates transactions
* Transaction Record maintains details
* Bank Server verifies and processes the payment



**Q5. Draw a sequence diagram for payment done by Customer Net Banking - 4 Marks**

* Customer selects Net Banking payment method
* Payment system requests authentication from the Bank
* Bank authenticates and processes payment
* Confirmation sent to the payment system and customer



**Q6. Explain Conceptual Model for this Case - 4 Marks**

A conceptual model abstracts key entities and their relationships. Entities include Customer, Payment, Transaction, and Bank. Relationships:

* Customer makes a Payment
* Payment is processed through Net Banking
* Transaction is recorded

**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 Architecture:**

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

* **Model:** Manages data and business logic.
* **View:** Handles the UI and presentation layer.
* **Controller:** Acts as an intermediary between Model and View, processing user input and updating the Model or View accordingly.

**Rules to Derive Classes from Use Case Diagram:**

* **Identify Entity Classes:** These represent business objects derived from nouns in the use case (e.g., Customer, Payment).
* **Identify Boundary Classes:** These handle interactions between the system and users, typically screens, forms, or APIs.
* **Identify Controller Classes:** These manage workflows and logic between the boundary and entity classes.

**Guidelines to Place Classes in a 3-Tier Architecture:**

* **Presentation Layer (View):** UI elements like forms, webpages, or mobile app screens (e.g., Payment Screen).
* **Business Layer (Controller):** Business rules, logic, and workflows (e.g., Payment Processor).
* **Data Layer (Model):** Database interactions and business entities (e.g., Payment Transaction).

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

**1. Requirement Gathering & Analysis**

* Elicits requirements through stakeholder meetings.
* Documents Business Requirements Document (BRD).
* Creates use cases and process flows.

**2. System Design**

* Defines functional and non-functional requirements.
* Prepares wireframes and UI mockups.
* Maintains a Requirement Traceability Matrix (RTM).

**3. Implementation (Development)**

* Works closely with developers to clarify business logic.
* Reviews system prototypes and provides feedback.
* Supports change management.

**4. Testing**

* Assists in User Acceptance Testing (UAT).
* Validates test cases to ensure alignment with requirements.
* Helps in defect analysis and resolution.

**5. Deployment**

* Ensures a smooth go-live process with stakeholders.
* Provides training sessions for end-users.
* Documents system workflows.

**6. Maintenance & Support**

* Manages post-deployment change requests.
* Tracks system performance and improvement areas.
* Acts as a liaison between users and IT teams.

**Q9. What is conflict management? Explain using Thomas-Kilmann technique**  
Conflict management is the process of resolving disagreements effectively. The **Thomas-Kilmann Conflict Model** has **five strategies**:

1. **Competing:** High assertiveness, low cooperation (used in high-stakes decisions).
2. **Collaborating:** High assertiveness, high cooperation (win-win approach).
3. **Compromising:** Medium assertiveness, medium cooperation (mutual concessions).
4. **Avoiding:** Low assertiveness, low cooperation (used when issues are minor).
5. **Accommodating:** Low assertiveness, high cooperation (prioritizing relationships over winning).

Example: If two teams argue over a project's priority, the BA might use **collaborating** to find a balanced solution.

**Q10. List down the reasons for project failure**

1. **Unclear Requirements** – Poorly defined scope leads to misalignment.
2. **Lack of Stakeholder Engagement** – Delays in decision-making.
3. **Scope Creep** – Frequent changes without proper impact analysis.
4. **Inadequate Risk Management** – Failing to anticipate risks.
5. **Poor Communication** – Gaps between teams and management.
6. **Budget or Time Overruns** – Inadequate planning and resource allocation.

**Q11. List the Challenges faced in projects for BA**

**Requirement Ambiguity:** Stakeholders may provide vague or conflicting inputs.

**Scope Creep:** Frequent changes in business requirements.

**Technology Limitations:** Constraints in existing systems affecting feasibility.

**Stakeholder Resistance:** Users may resist adopting new processes.

**Managing Expectations:** Aligning business needs with technical feasibility.

**Limited Documentation:** Lack of historical records for reference

**Q12. Write about Document Naming Standards**

* Follow a clear, structured format: **[Project Name] *[Document Type]*[Version]**
* Example: **PaymentSystem\_BRD\_V1.0.docx**
* Versioning should be incremental (V1.0, V1.1, V2.0).
* Use standard abbreviations (BRD = Business Requirements Document).

**Q13. What are the Do’s and Don’ts of a Business Analyst**  
**Do’s:**

* Maintain clear communication with stakeholders.
* Keep thorough documentation.  
  Validate requirements with users.  
  Be proactive in risk identification**.**

**Don’ts:**

* Assume requirements without stakeholder confirmation.  
  Ignore feasibility analysis.  
  Overlook non-functional requirements (security, scalability).  
  Delay issue resolution.
* Ignore stakeholder feedback
* Skip documentation

**Q14. Write the difference between packages and sub-systems**

* **Packages**: Logical grouping of classes
* **Sub-systems**: Independent components with defined functionalities

**Packages:** Logical grouping of related classes for organization.

* Example: A "Payment Processing" package contains classes like Payment, Transaction, and Wallet.

**Sub-systems:** Independent functional components within a system.

* Example: "Payment Module" and "User Management" are subsystems in an e-commerce platform.

**Q15. What is camel-casing and explain where it will be used**  
Camel-casing is a **naming convention** where multiple words are joined together, with each word (except the first) capitalized.

**Types of Camel-Casing**

* **Lower Camel Case (camelCase)** → First letter is lowercase, subsequent words start with uppercase.
  + Example: customerName, validateTransaction().
  + **Used in:** Variable names, method names in Java, JavaScript, Python.
* **Upper Camel Case (PascalCase)** → Every word starts with an uppercase letter.
  + Example: CustomerDetails, TransactionManager.
  + **Used in:** Class names, UI components, API models

**Where is it Used?**

* **Programming Languages:** Java, C#, JavaScript, Python.
* **Database Naming Conventions:** In NoSQL databases (e.g., MongoDB).
* **API Naming:** REST API endpoints use camelCase (e.g., /getCustomerDetails)

**Q16. Illustrate Development server and what are the accesses a BA has?**  
Development servers host test environments. BAs may have:

* Read access to test data
* Access to logs and reports
* Limited debugging permissions

A **Development Server** is an environment where code is written, tested, and modified before deployment. It is separate from the production server to avoid impacting live users.

**BA Access in a Development Server**

A Business Analyst (BA) **typically does not have direct code access** but has **limited permissions** to:

| **Access Type** | **BA Privileges** |
| --- | --- |
| **Requirement Verification** | Ensuring business logic aligns with the application |
| **UI Testing** | Reviewing UI/UX based on wireframes |
| **Test Data Review** | Checking if test cases align with requirements |
| **UAT Coordination** | Assisting testers in validating system functionality |

* **BA does not have** database access or permissions to modify server configurations.

**Q17. What is Data Mapping?**  
Data mapping links data fields between source and destination. Used in migration and integration.

Data Mapping is the **process of matching data fields** from one system to another to enable seamless data integration.

**2. Why is Data Mapping Important?**

* Ensures **data consistency** between systems.
* Helps in **data migration** and **API integrations**.
* Prevents **data loss or corruption** when transferring information.

**3. Data Mapping Types**

1. **Direct Mapping:** When source and target fields match exactly.
   * Example: CustomerID → CustomerID.
2. **Transformation Mapping:** When data needs conversion.
   * Example: Date Format (dd-mm-yyyy → mm-dd-yyyy).
3. **Multi-Field Mapping:** When multiple source fields map into a single target field.
   * Example: FirstName + LastName → FullName.

**Q18. What is API? Explain how you would use API integration in the case of your application**  
An API (Application Programming Interface) allows different systems to communicate. In this case, the system must handle date format conversion between **dd-mm-yyyy (local)** and **mm-dd-yyyy (US)** by mapping and transforming data formats correctly.

**1. API (Application Programming Interface)**

An **API** is a set of rules that allows different software applications to communicate.

* **Example:** A Payment API processes transactions between a website and a bank.
* **Types of APIs:**
  + **REST API** (Most commonly used)
  + **SOAP API**
  + **GraphQL API**

**Handling Date Format Differences in API Integration**

**Scenario:**

* Your system uses **dd-mm-yyyy** format.
* The US-based application sends dates in **mm-dd-yyyy** format.

**3. Solutions**

**(A) Middleware Conversion**

* Introduce an **intermediate layer** that reformats the date before processing.
* Example: Convert 02-13-2025 (US) to 13-02-2025 (India).

**(B) Code-Based Date Formatting**

* Convert incoming API date format using programming logic.

**C) API Standardization Using ISO 8601**

* Use the **ISO 8601 format (YYYY-MM-DD)** to avoid confusion.
* Example: "2025-02-13" instead of "02-13-2025" or "13-02-2025".

**(D) API Documentation Update**

* Clearly define the expected date format in the API contract.
* Ensure the consuming application is aware of the required format