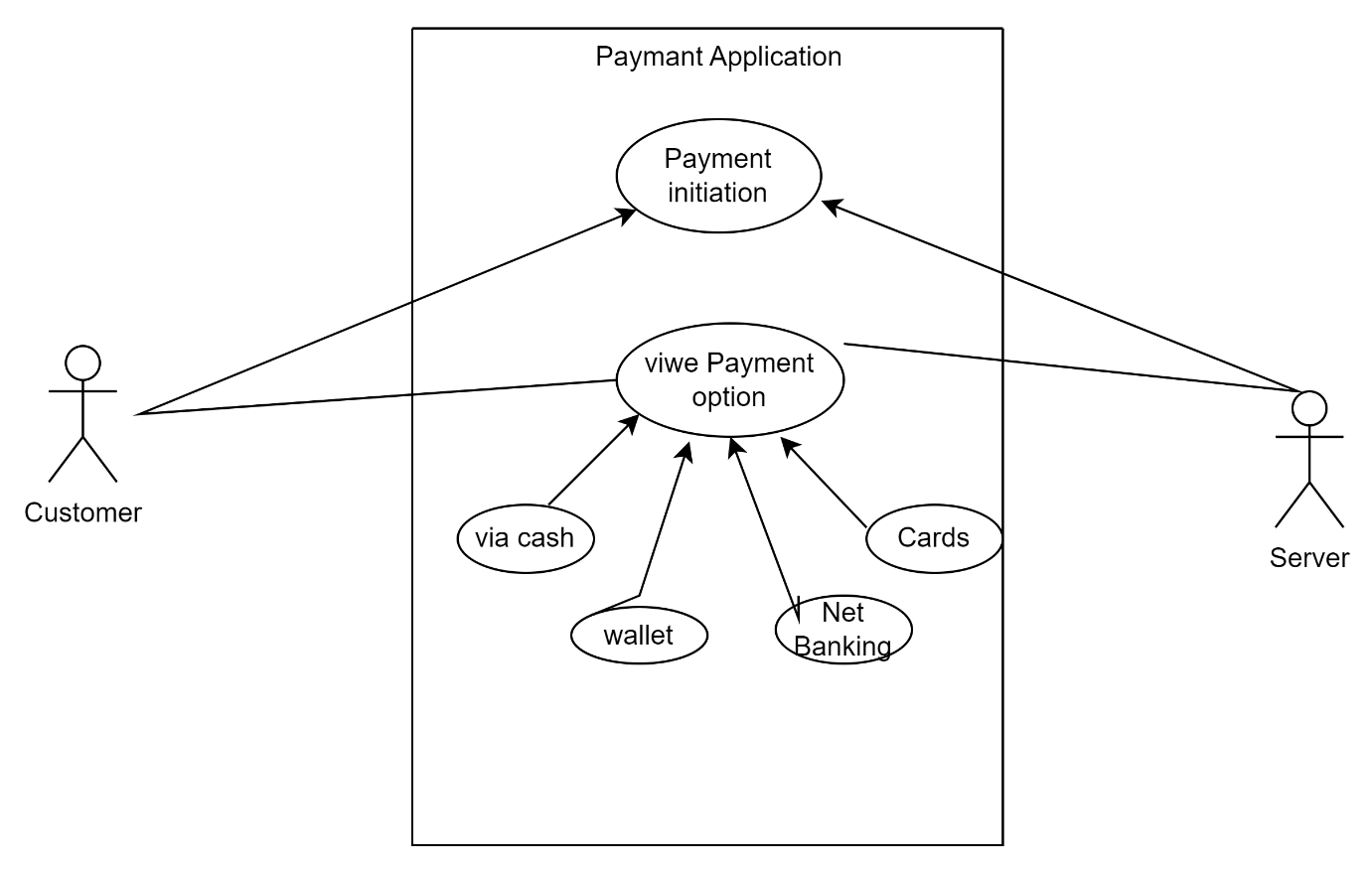
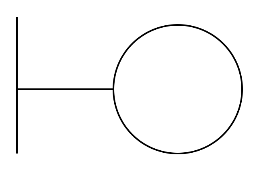
**Q1. Draw a Use Case Diagram : A Use Case Diagram is a visual representation of how users (actors) interact with a system to achieve specific goals. It is part of Unified Modeling Language (UML) and is commonly used in business analysis and software development to define system functionality from a user perspective.**



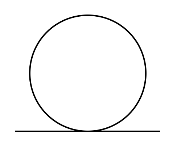
**Q2. Derive Boundary Classes, Controller classes, Entity Classes.**

Boundary Classes: Act as an interface between the system and external users (UI, APIs). Handle user input, output formatting, and interaction with external services. Ensure data validation and request handling.



Controller Classes: Manage the flow of data between boundary and entity classes. Contain application logic and coordinate system operations. Call appropriate entity methods to process business logic.

Entity Classes: Represent real-world objects and store business data. Contain attributes and methods related to business logic. Are typically mapped to a database for persistence.



**Q3. Place these classes on a three tier Architecture.**

1. Presentation Layer (UI Layer) – Boundary Classes

This layer handles user interactions. The boundary classes act as the interface between the user and the system.

Boundary Classes:

PaymentPage (Collects payment details from the customer)

ConfirmationPage (Displays transaction status)

2. Business Logic Layer (Application Layer) – Controller Classes

This layer processes the payment, applies rules, and interacts with external systems like a payment gateway.

Controller Classes:

Payment Controller (Validates payment details, selects payment method)

Transaction Processor (Handles actual payment processing)

3. Data Layer (Database Layer) – Entity Classes

This layer manages the storage of customer and transaction data.

Entity Classes:

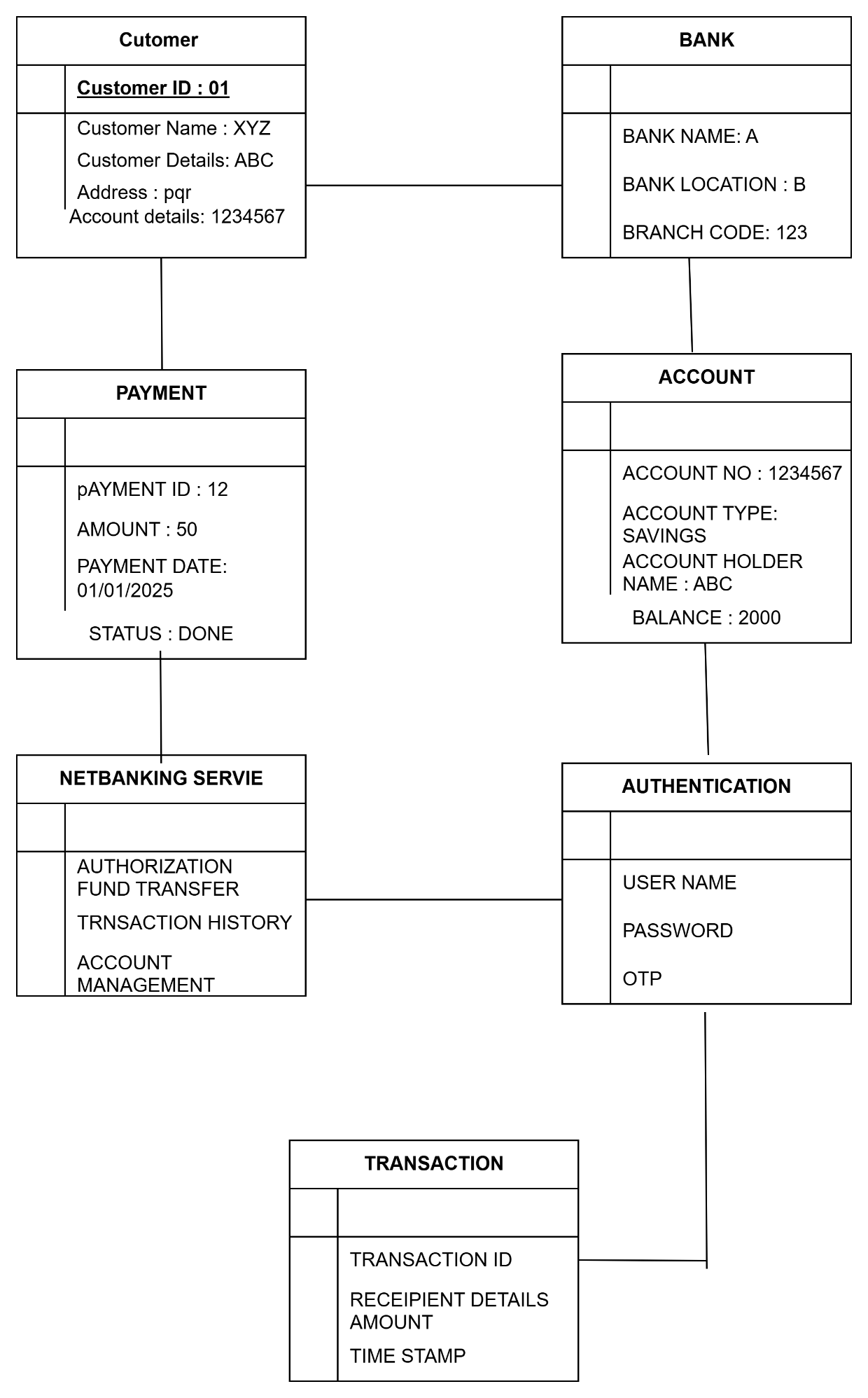
Customer (Stores customer details)

Payment Method (Stores payment options: Card, Wallet, Cash, Net Banking)

Transaction (Stores payment history and transaction details)

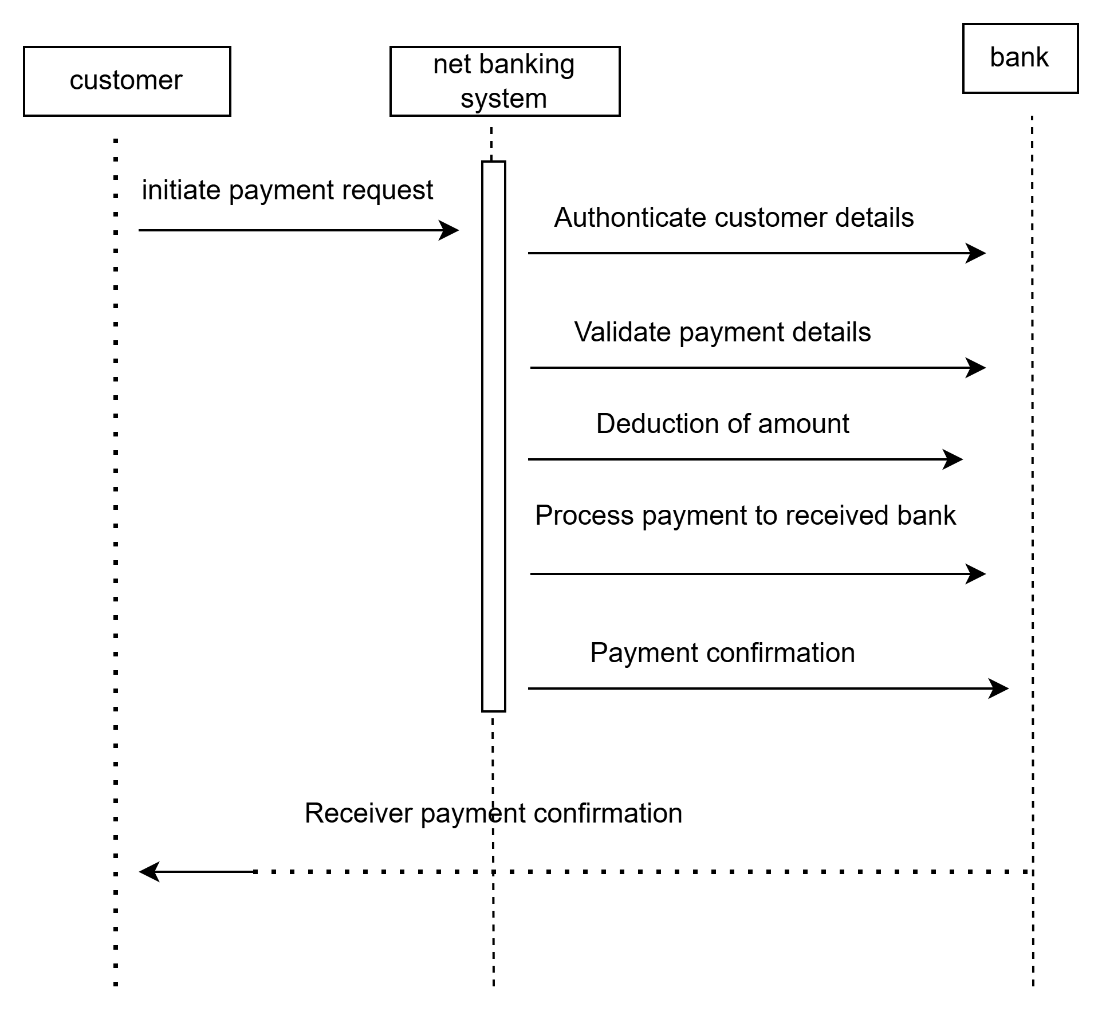
**Q4. Explain Domain Model for Customer making payment through Net Banking**

A Domain Model represents the real-world entities, their attributes, and the relationships between them in a business process. In the context of a customer making a payment through Net Banking, the domain model will identify the key entities and their interactions.



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

Sequence diagram is a type of interaction diagram used in software engineering and systems design to illustrate how processes operate with one another and in what order.

****

**Q6. Explain Conceptual Model for this Case :**

A Conceptual Model represents high-level entities and their relationships without detailing attributes or technical aspects. Here’s how we can structure a conceptual model for the payment system:

Entities:

1. Customer – The individual making a payment.

2. Payment – The transaction made by the customer.

3. Payment Method – The mode through which payment is made.

Card

Wallet

Cash

Net Banking

Relationships:

A Customer can make one or multiple Payments.

Each Payment is associated with exactly one Payment Method.

A Payment Method can be of type Card, Wallet, Cash, or Net Banking.

**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 is a software design pattern used to separate concerns in application development, improving maintainability and scalability. It divides an application into three interconnected components:

1. Model

Represents the business logic and data layer.

Handles database interactions, validation, and processing.

Example: Payment class storing transaction details.

2. View

Represents the user interface.

Displays data received from the Model.

Example: A webpage showing payment options.

3. Controller

Manages user input, processes requests, and updates the Model/View accordingly.

Acts as a mediator between Model and View.

Example: PaymentController handling payment processing.

MVC Rules to Derive Classes from Use Case Diagram

1. Identify the Nouns in Use Cases: Convert them into Model classes (e.g., Customer, Payment).

2. Identify User Interactions: Define Controllers to handle them (e.g., PaymentController).

3. Derive Views from UI-related Use Cases: Create UI classes (e.g., PaymentPage).

4. Establish Relationships: Link Controllers with Models and Views for proper data flow.

Guidelines to Place Classes in 3-Tier Architecture

1. Presentation Layer (View)

Includes UI components like webpages or forms.

Example: PaymentPage.aspx, Payment.jsp.

2. Business Logic Layer (Controller + Model Logic)

Contains business rules and controllers.

Example: Payment Controller, Payment Service.

3. Data Access Layer (Model + Database Interaction)

Manages database operations.

Example: Payment, Database Helper.

This approach ensures modularity, maintainability, and scalability in software development.

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

|  |  |  |  |
| --- | --- | --- | --- |
| Stages | Activities | Artifacts | Resources |
| Pre-Project | Understand business objectives.  Identify stakeholders and key decision-makers.  Conduct feasibility study and market research.  Define high-level business problems and opportunities. | Feasibility Report – Evaluates project viability.  Business Case Document – Justifies the need for the project. | Business Analyst (BA), Business Sponsor, Subject Matter Experts (SMEs), Project Manager. |
| Planning | Define project scope.  Identify risks and constraints.  Plan requirement activities.  Establish timelines and dependencies. | Project Scope Document – Defines project objectives and deliverables.  Risk Assessment Plan – Identifies potential risks and mitigation strategies. | BA, Project Manager (PM), Stakeholders, Technical Leads. |
| Project Initiation | Conduct stakeholder meetings.  Define high-level requirements. | Business Requirement Document (BRD) – Captures high-level business needs. | BA, Project Sponsor, SMEs, Project Manager. |
| Requirement Gathering | Collect detailed requirements via workshops, interviews, and surveys.  Identify functional and non-functional requirements. | Requirement Traceability Matrix (RTM) – Maps requirements to business objectives.  Use Case Diagrams – Represents system interactions. | BA, End Users, SMEs, Development Team. |
| Requirement Analysis | Analyze, prioritize, and validate requirements.  Ensure clarity, completeness, and feasibility. | Functional Requirement Specification (FRS) – Defines system functionality.  Data Flow Diagrams (DFD) – Represents system processes. | BA, Architects, Developers, Testers. |
| Design | Support UI/UX design and database structuring.  Ensure business requirements align with technical design. | UI Wireframes – Visual representation of the interface.  System Design Documents – Defines architecture and technical solutions. | BA, Designers, Architects, Developers. |
| Development | Clarify requirements for developers.  Handle change requests and update documentation. | Updated BRD/FRS – Reflects any changes.  Change Log – Tracks modifications. | BA, Developers, Technical Leads. |
| Testing | Assist in creating test cases.  Validate test scenarios and support defect triaging. | Test Cases – Defines test scenarios.  Defect Logs – Records issues found during testing.  Test Summary Reports – Documents test results. | BA, Testers, QA Team. |
| User Acceptance Testing (UAT) | Facilitate UAT process.  Gather user feedback and ensure business objectives are met. | UAT Test Plan – Defines user testing strategy.  UAT Sign-Off Document – Confirms system readiness. | BA, End Users, UAT Team, Stakeholders. |
| Deployment | Deployment system to production configure servers and security perform the final testing and train end users | Deployment plan Release notes training manuals | BA, developers ,support team. |
| Maintenance | Monitor system performance , Handle user Queries apply updates and new features | Isuue logs change request logs Updated BRD | BA, developers ,support team. |

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

Conflict Management & Thomas-Kilmann Technique

Conflict management is the process of resolving disagreements between individuals or groups in a constructive manner to maintain a healthy work or personal environment. Effective conflict resolution ensures smooth communication, better teamwork, and productivity.

The Thomas-Kilmann technique is a widely used model for assessing conflict resolution styles. It helps individuals understand their natural approach to conflicts and guides them in choosing the most suitable strategy to handle different situations.

5 Steps of Conflict Management

1. Identify the Conflict – Recognize that a conflict exists and define the issue clearly. Understanding the nature of the conflict is the first step in resolving it.

2. Discuss the Details – Gather relevant information about the conflict, listen to all parties involved, and understand different perspectives to get a complete picture.

3. Agree on the Root Cause – Instead of focusing on surface-level issues, identify the actual underlying problem that led to the conflict. This ensures that the resolution addresses the core issue.

4. Explore Possible Solutions – Brainstorm different ways to resolve the conflict. Evaluate the advantages and disadvantages of each solution to find the most effective one.

5. Negotiate and Implement the Solution – Choose the best solution that satisfies all parties as much as possible. Implement the resolution while ensuring that future conflicts are minimized by setting clear expectations and communication guidelines.

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

1. Unclear Requirements – Lack of well-defined project scope and objectives.

2. Poor Planning and Management – Inadequate project scheduling, resource allocation, and risk management.

3. Lack of Stakeholder Engagement – Poor communication and coordination among stakeholders.

4. Budget and Time Overruns – Exceeding cost and timeline estimates.

5. Technical Challenges – Incompatible technology, poor system design, or lack of expertise.

6. Scope Creep – Uncontrolled expansion of project scope without proper evaluation.

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

1. Unclear or Changing Requirements – Frequent modifications and lack of clarity in business needs.

2. Stakeholder Conflicts – Managing different expectations and priorities among stakeholders.

3. Limited Domain Knowledge – Understanding complex industry-specific processes and terminologies.

4. Communication Gaps – Misinterpretation of requirements due to poor communication.

5. Technical Constraints – Aligning business needs with system limitations and feasibility.

6. Time and Budget Constraints – Delivering quality analysis within tight deadlines and budgets.

7. Managing Scope Creep – Controlling unplanned changes that impact project timelines.

8. Integration Challenges – Ensuring smooth integration with existing systems and processes.

9. Resistance to Change – Overcoming reluctance from users to adopt new systems or processes.

10. Ensuring Quality Deliverables – Validating and verifying requirements to meet business expectations.

Q12. Write about Document Naming Standards

Document naming standards are a set of guidelines used to create consistent and meaningful file names. These standards help in easy identification, retrieval, and version control of documents within an organization.

Key Elements of Document Naming Standards:

1. Descriptive Name – The document name should clearly indicate its content or purpose (e.g., Project\_Requirements\_Document).

2. Date Format – Use a standard date format (YYYY-MM-DD) to track versions (e.g., Report\_2025-02-09).

3. Version Control – Include version numbers to track updates (e.g., Proposal\_V1.0 or Proposal\_V2.1).

4. Consistent Abbreviations – Use predefined abbreviations to maintain uniformity (e.g., ReqDoc for Requirement Document).

5. Avoid Special Characters – Use underscores (\_) or hyphens (-) instead of spaces and avoid symbols like & or %.

6. Project or Department Identifier – Include project name or department for easy classification (e.g., HR\_Policy\_Guide).

7. Author or Creator Information (if needed) – Add initials or names for identification (e.g., BudgetReport\_JSmith).

8. Document Type – Mention the type of document such as Invoice, Report, Proposal (e.g., ClientProposal\_V3.0).

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

**Do’s:**

1.Understand business needs before proposing solutions.

2. Communicate clearly with stakeholders and teams.

3. Ask relevant questions to uncover hidden needs.

4. Document all requirements thoroughly.

5. Use data for informed decision-making.

6. Collaborate with teams for smooth execution.

7. Prioritize requirements effectively.

8. Stay updated on industry trends.

9. Validate and verify all requirements.

10. Manage stakeholder expectations realistically.

**Don’ts**

1.Don’t assume requirements—always confirm.

2. Don’t use excessive jargon that confuses stakeholders.

3. Don’t ignore user feedback and needs.

4. Don’t skip documentation or leave gaps.

5. Don’t be rigid; adapt to changing requirements.

6. Don’t rely only on IT solutions—consider processes too.

7. Don’t neglect risk analysis and dependencies.

8. Don’t work in isolation—engage with all teams.

9. Don’t ignore non-functional requirements.

10. Don’t delay feedback loops to avoid rework.

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

|  |  |
| --- | --- |
| **packages** | **sub-systems** |
| A package is a logical grouping of related classes, interfaces, or components, | subsystem is a self-contained module with a specific functionality in a system. |
| A package helps organize code for better reusability and maintainability, | a subsystem represents a functional unit that interacts with other parts of the system. |
| A package does not function independently and is mainly for code organization | a subsystem can operate independently and may have its own dependencies. |
| A package is commonly used in programming languages like Java (java.util), | a subsystem is used in system design, such as a payment processing subsystem in an e-commerce application |
| A package contains only code elements | a subsystem can include multiple packages, components, and external interfaces. |

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

**Camel Casing** is a naming convention in which multiple words are combined into a single string, and each word starts with a capital letter (except for the first word in lowerCamelCase). It is commonly used in programming, file naming, and documentation for better readability and consistency.

**Types of Camel Casing**

1. **lowerCamelCase** – The first word starts with a lowercase letter, and subsequent words start with uppercase.
   * Example: customerDetails, orderNumber, fileNameFormat
   * Used in variable names, function names, and API methods.
2. **UpperCamelCase (PascalCase)** – Every word starts with an uppercase letter.
   * Example: CustomerDetails, OrderNumber, FileNameFormat
   * Used in class names, document naming conventions, and database table names.

**Where is Camel Casing Used?**

**1. Programming & Software Development**

* **Variable Names:** totalAmount, customerAddress
* **Function Names:** calculateTax(), validateUserInput()
* **API Methods:** getUserDetails(), updateOrderStatus()
* **Class Names:** EmployeeRecord, ProductDetails

**2. Database & Data Models**

* **Table Names:** CustomerDetails, OrderHistory (UpperCamelCase)
* **Column Names:** userName, emailAddress (lowerCamelCase)

**3. Documentation & Business Analysis**

* **Requirement Documents:** BusinessRequirementsDocument.docx
* **File Naming Standards:** TestPlan\_Sprint1\_20250311.docx
* **Process Flow & UML Diagrams:**
  + Class names: UserProfile, OrderManagement
  + Attributes: firstName, accountBalance

**4. Software Testing & Automation**

* **Test Cases:** verifyLoginFunctionality()
* **Automation Scripts:** testUserAuthentication()

**5. Business Rules & Pseudocode**

* **Logical Conditions:**

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

A Development Server is an environment where software is built, tested, and modified before deployment. It includes frontend, backend, database, and testing components.

Access for a Business Analyst (BA):

1. Read-Only Access – Can view logs and system behavior.

2. Application Testing Access – Can log in to test environments to validate requirements.

3. Database Access (Limited) – Can run SQL queries to check data but cannot modify it.

4. User Acceptance Testing (UAT) – Ensures the developed features meet business needs.

5. Bug Tracking & Reporting – Uses Jira/Trello to log and track issues.

6. Version Control (Restricted Access) – Can view changes in Git/SVN but not

**Q17. What is Data Mapping :**

Data mapping is the process of linking data fields from one system to another to ensure consistency and accuracy. It is commonly used in data migration, integration, and transformation processes. Data mapping helps in standardizing data formats when transferring information between different databases or applications. It ensures that data is correctly interpreted and used by the target system. This process is essential for maintaining data integrity and preventing loss or corruption during transfers.

**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**

An API (Application Programming Interface) is a set of rules and protocols that allow different software applications to communicate with each other. APIs define how requests are made, what responses are expected, and how data is exchanged between systems.

API Integration in My Application

In the case of my Online Agriculture Product Store application, API integration can be used to receive product details, prices, and orders from different manufacturers or vendors and update them in the system.

For instance, if the application accepts product information, order details, or user data from a US-based system, we need to handle date format differences because:

My application follows the "dd-mm-yyyy" format

The US application follows the "mm-dd-yyyy" format