**1.Draw a Use Case Diagram: A customer can make a payment either by Card or by Wallet or by Cash or by Net banking.**



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

In UML (Unified Modeling Language) and MVC (Model-View-Controller)-based systems, classes are often categorized into boundary, entity, and controller classes to separate concerns and maintain a clean architecture.
**Entity Class:** Represents the core business logic and data.

These classes model real-world concepts and business rules.
Typically corresponds to database tables or persistent data.
They are the most stable classes in the system — they rarely change unless the business logic changes.

**Boundary Class (Interface Class or View Class):** Acts as the interface between the system and external actors (users, other systems).
 It manages the interaction between the user and the system.
 Responsible for input/output, user interface, or API endpoints.
 It does not contain business logic.

**Controller Class**: Acts as a **mediator** between boundary and entity classes.Coordinates the flow of information.
 Handles requests, invokes appropriate entity methods, and returns responses to the boundary.
 May contain application-specific logic, like validation or workflows.

****

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

 Application layer: Payment method boundary,Card payment

 Business layer: Payment controller, Wallet controller

 Data layer: Customer, Payment

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

A domain model for a customer making a payment through net banking would include entities like Customer, Bank, Payment, and Transaction. It would also define the relationships between these entities, such as a Customer making Payments to a Bank, and each Payment resulting in a Transaction.


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

**6.Explain Conceptual Model for this Case**The Conceptual Model is a high-level representation of the key concepts (objects), their attributes, and the relationships between them — without getting into technical implementation or software-specific details. It's used to understand the domain logic clearly from a business perspective.To represent what the system must handle conceptually — including the entities involved in the net banking payment process and how they relate.Key elements of model:
**1. Entity:** Customer, Product, order, payment
**2. Attributes:** CustID, name, address, email
**3. Relationship:** A customer place a order, **7. 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**

It’s a software architectural pattern used for separating concerns in application design — commonly used in web and desktop applications.
To organize code in such a way that business logic, UI, and input handling are kept separate, making the system:

* Easier to maintain, More scalable, Better for testing and collaboration

 **Components of MVC**

| Component | Role | Deals With |
| --- | --- | --- |
| Model | Manages business logic and data | Data layer |
| View | Displays data to the user | UI layer |
| Controller | Handles user input, interacts with model and view | Input logic |

**Rules:
1.** Each actor becomes a Boundary Class
 Example: A Customer actor → CustomerForm, CustomerPortal
 Handles UI interaction for that actor.

2.Each use case becomes a Controller Class
 Example: Use case Make Payment → PaymentController
 Coordinates the logic flow and interacts with models.

3.Each business object becomes an Entity Class
 Example: Customer, Payment, Account
 These are derived from nouns in the use case descriptions and business logic.

4.One controller per use case or group of related use cases
 Avoid creating too many controllers; group related actions.

**Guidelines to Place Classes in 3-Tier Architecture**3-Tier = Presentation, Business Logic, Data Access

| Tier | Includes | Class Types |
| --- | --- | --- |
| Presentation Layer | Handles UI or external interface | Boundary Classes (Views) |
| Business Logic Layer | Core application logic | Controller Classes + business rules |
| Data Layer | Manages persistent data | Entity Classes (Models) |

**Class Placement Guidelines:**

1. Boundary Classes → Presentation Tier, Example: LoginForm, PaymentPage,Deal with user input/output.
2. Controller Classes → Business Logic Tier, Example: PaymentController, CustomerController, Manage workflows and system logic.
3. Entity Classes → Data Tier, Example: Customer, Transaction, Payment, these interact with the database or persistent storage.

**8. Explain BA contributions in project (Waterfall Model – all Stages)**A Business Analyst (BA) plays a key role throughout the Waterfall Model of software development, which is a sequential and linear approach. Here’s how a BA contributes at each stage:

1. Requirements Gathering & Analysis

* Elicit Requirements: Engage with stakeholders (users, customers, SMEs) through interviews, surveys, and workshops.
* Document Requirements: Create detailed Business Requirement Documents (BRDs), Functional Requirements Specifications (FRS), and Use Cases.
* Clarify Ambiguities: Resolve conflicting requirements and ensure alignment between business needs and technical capabilities.
* Validate Requirements: Ensure all requirements are signed off and traceable.

*.* 2. System Design

* Support Designers: Clarify requirements for architects and designers.
* Ensure Alignment: Verify that the proposed system design aligns with business needs.
* Participate in Reviews: Attend design walkthroughs to ensure business requirements are interpreted correctly.

3. Implementation (Coding)

* Assist Developers: Answer questions regarding requirements.
* Clarify Business Logic: Provide deeper business context for complex rules.
* Change Management: Help assess the impact of any proposed requirement changes (though Waterfall typically resists change after this phase).

 4. Testing

* Support QA Team: Help develop test scenarios based on requirements.
* Review Test Cases: Ensure test coverage maps to all requirements (Requirement Traceability Matrix - RTM).
* User Acceptance Testing (UAT): Coordinate or assist with UAT, involving end-users to validate the system meets their expectations.

 5. Deployment

* Facilitate Communication: Bridge the gap between business and technical teams during rollout.
* Training & Documentation: Help create user manuals and training sessions.
* Post-Deployment Support: Address any business-related issues or clarifications.

 6. Maintenance

* Issue Analysis: Assist in identifying root causes for reported issues.
* Enhancement Requirements: Gather new requirements for future releases or updates.
* Monitor Performance: Provide feedback for improvements based on user input.

Summary Table:

| Waterfall Stage | BA Contribution |
| --- | --- |
| Requirements | Elicit, document, validate Act as Lead |
| Design | Clarify and review Act as collaborator |
| Implementation | Support and clarify Act as support |
| Testing | Validate requirements vs. outcomes Act as validator |
| Deployment | Coordinate training & rollout Act as coordinator |
| Maintenance | Suggest improvements, analyze issues Act as adviser |

 **9. What is conflict management? Explain using Thomas – Kilmann technique**Conflict Management is the process of identifying and handling conflicts in a sensible, fair, and efficient manner. It's a key soft skill in project management, team leadership, and business analysis, as conflicts are inevitable when people with different perspectives work together.

 **Thomas-Kilmann Technique (TKI Model)**

The Thomas-Kilmann model is a widely-used framework that identifies five main approaches to conflict management based on two dimensions:
Assertiveness – The extent to which you try to satisfy your own needs.
Cooperativeness – The extent to which you try to satisfy the other person's needs.

These dimensions form a 2x2 matrix with a fifth style in the middle:

| **Style** | **Assertiveness** | **Cooperativeness** | **Description** |
| --- | --- | --- | --- |
| **1. Competing** (Win-Lose) | High | Low | You pursue your own goals at the expense of others. Used when quick, decisive action is needed (e.g. emergencies). |
| **2. Collaborating** (Win-Win) | High | High | You work together to find a solution that fully satisfies both parties. Ideal for complex issues and long-term relationships. |
| **3. Compromising** (Lose-Lose to some extent) | Medium | Medium | Both parties give up something to reach a mutually acceptable solution. Useful for temporary solutions or when time is limited. |
| **4. Avoiding** (No Deal) | Low | Low | You don’t address the conflict, hoping it resolves itself or isn't worth engaging. Useful when the issue is trivial or emotions are high. |
| **5. Accommodating** (Lose-Win) | Low | High | You put others' needs before your own. Good for maintaining harmony but can lead to resentment if overused. |

 **10. List down the reasons for project failure
1. Unclear or Incomplete Requirements**
 Lack of proper requirement gathering
 Vague or conflicting expectations
 Changing requirements not managed well

**2. Poor Planning** Inadequate project scope definitionUnrealistic timelines and budgetsLack of contingency planning

**3. Lack of Stakeholder Involvement** Poor communication with stakeholdersNo proper buy-in or feedback from end-usersIgnoring key stakeholder needs

**4. Weak Project Management** Ineffective leadership or governanceNo clear roles and responsibilitiesFailure to monitor progress and adapt

**5. Inadequate Risk Management**

* Risks not identified or mitigated early
No backup plans for critical path issues

**6. Poor Communication**

* Miscommunication between team members or departments
Lack of regular updates or status reports
Language or cultural barriers in diverse teams

**7. Scope Creep**

* Uncontrolled changes or additions without proper evaluation
No change control mechanism in place

**8. Lack of Resources**

* Insufficient staff, budget, or tools
Over-reliance on unavailable or underqualified personnel

**11. List the Challenges faced in projects for BA**1. Unclear or Evolving Requirements

* Stakeholders don’t know what they want.
Frequent requirement changes mid-project (scope creep).
Difficulty in eliciting requirements from non-technical stakeholders.

2. Communication Gaps

* Misunderstandings between business and technical teams.
Poor documentation leading to confusion.
Language or cultural barriers in global teams.

3. Lack of Stakeholder Involvement

* Stakeholders unavailable for interviews or sign-offs.
Delayed feedback or indecisiveness.
Conflicting opinions among stakeholders.

4. Time Constraints

* Tight deadlines for requirement gathering or analysis.
Pressure to deliver documentation quickly, compromising quality.

5. Managing Expectations

* Unrealistic expectations from clients or sponsors.
Difficulty in explaining limitations of technology or processes.

6. Changing Scope

* Frequent additions to the scope without proper impact analysis. Lack of formal change control process.

7. Technical Constraints

* BA not familiar with the technical environment.
Difficulty translating business needs into technical requirements.

8. Inadequate Tools or Resources

* Outdated requirement management tools.
Limited access to stakeholders or subject matter experts (SMEs).

 **12. Write about Document Naming Standards**Document Naming Standards refer to a predefined set of rules used to name files and documents in a consistent, logical, and easily searchable way. These standards help in maintaining version control, improving collaboration, and simplifying file retrieval across teams and projects.

 Why Naming Standards Are Important:

 Ensure clarity and consistency
Avoid confusion or duplication
Facilitate quick search and retrieval
Support version control
Help with audit and compliance tracking

 **Key Elements of a Naming Standard**

**A good document name typically includes:**

| Element | Example | Description |
| --- | --- | --- |
| Project Code | PRJ001 | Unique ID or acronym for the project |
| Document Type | BRD, FRS, UAT | Indicates the document content (e.g., Business Requirement Document) |
| Subject or Title | LoginModule | Brief description of the document topic |
| Version Number | v1.0, v2.1 | Shows the version or revision |
| Date (Optional) | 2025-04-23 | Date in YYYY-MM-DD format |
| Author Initials | JS | Useful for tracking authorship |
|  |  |  |

 **PRJ001\_BRD\_LoginModule\_v1.0\_2025-04-23\_JS.docx**

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

1. Do Understand the Business

 2. Do Communicate Clearly

3. Do Document Thoroughly

4. Do Collaborate Actively

 5. Do Stay Objective

 6. Do Embrace Change Management

 7. Do Use Visual Aids

 8. Do Validate and Verify Requirements

**Don’ts of a Business Analyst**

 1. Don’t Make Assumptions

2. Don’t Work in Isolation

 3. Don’t Use Too Much Jargon
 4. Don’t Overlook Non-Functional Requirements

 5. Don’t Skip Reviews or Sign-Offs

6. Don’t Ignore Conflicts

 7. Don’t Rush Requirement Gathering

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

| **Feature** | **Packages** | **Sub-systems** |
| --- | --- | --- |
| Definition | A logical grouping of related classes or elements. | A semi-independent unit within a system that performs a specific function. |
| Purpose | Organize and manage model complexity. | Structure large systems into functional modules. |
| Focus | Organizational/logical structure | Functional/system-level behavior |
| Granularity | Finer level (e.g., classes, interfaces) | Coarser level (e.g., modules, components) |
| Encapsulation | May or may not encapsulate behavior | Typically encapsulates a complete function |
| Usage in UML | Represented as folders (rectangles with tabs) | Can be modeled using components or composite structures |
| Dependencies | Used to manage dependencies between classes | May have interfaces for communication with other sub-systems |
| Examples | A UserModule package containing Login, Profile, and Register classes | A Billing System sub-system handling all billing-related operations |
| Visibility | Generally used within the development environment | May be visible to end-users (via features) |

 **15. What is camel-casing and explain where it will be used**Camel Casing (or camelCase) is a naming convention in programming and writing identifiers (like variable names, functions, etc.), where: The first word starts in lowercase, And each subsequent word starts with an uppercase letter,No spaces or punctuation are used between words.
 **1. Programming Languages**
 **2. Variable and Function Naming
3. JSON Keys
4. Front-End Development** - Improves readability.
 - Avoids spaces and special characters (which aren’t allowed in identifiers).
 - Follows standard coding conventions in many languages and frameworks.
 - Keeps code consistent and maintainable.

**16. Illustrate Development server and what are the accesses does business analyst has?**A Development Server is an environment where developers build, test, and debug applications before moving them to staging or production. It acts as a sandbox that mimics the real system but is isolated for development purposes.

 Key Features of a Development Server:
- Used by developers and testers
- Contains the latest (sometimes unstable) code
- Allows frequent code updates and testing
- May be integrated with version control (like Git)
- Typically has limited security/restrictions compared to production

**Common Access for BAs:**

* View functional flows and UI changes
Participate in UAT or exploratory testing
Validate requirements by reviewing working modules
Log bugs or improvement points using a bug-tracking tool (e.g., Jira)
Compare actual vs expected results based on BRD/FRS

Restricted Activities:

* No direct code changes
No database modification (unless read-only access is given)
No deployment or server configuration access

**17. What is Data Mapping**Data Mapping is the process of matching fields from one data source to another. It defines how data elements from a source system relate to those in a destination system — often as part of a data migration, integration, or transformation process.

| Use Case | Purpose |
| --- | --- |
| Data Migration | Moving data from old system to new one |
| System Integration | Connecting different systems (e.g., CRM → ERP) |
| ETL Processes | In Data Warehousing to Extract, Transform, Load data |
| APIs and Web Services | Mapping fields from request to response |
| Reporting / BI | Standardizing data from multiple sources |

**18. 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 stands for Application Programming Interface.
It is a set of rules that allows two software applications to communicate with each other and exchange data securely and efficiently.
API Integration means connecting two or more applications through their APIs so they can share data and trigger actions automatically.

| Task | Description |
| --- | --- |
| Identify data formats | Ensure formats are known for all systems |
| Coordinate with tech teams | Work with developers to define transformation logic |
| Update interface specs or API contracts | Clearly document format expectations |
| Test and validate | Ensure the integration handles edge cases correctly (e.g., 01-02-2025 = Jan 2 or Feb 1?) |