**Q1. Draw a Use Case Diagram**

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

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

**3-Tier Architecture:**

3-Tier Architecture there will be Business Logic written in Business logic layer, having constraint to proceed input depend upon logic and gives output. From each and every Use case, going to extract classes. For ex: In ATM, user can only some amount of cash, this logic has been written in the business logic layer. After UCD+UCDD, team will start building Software and For that MVC Architecture will going to use

MVC Architecture:

1. **Boundary class**:

Actor Who has involved in this Use case, He may perform some **action** or support some action. The Kind of Action user will perform or initiate some action will be represented by **Presentation layer or Application Layer** of an application and known as **Boundary class**.

Symbol:

Boundary Class

1. **Controller Class:**

The specific **Actor** will perform some Action Based on some functionality like Login, Registration or Withdraw cash are the functionalities will be represented in **Controller layer** or in **Business Logic Layer** in middle of the 3 tier architecture or known as **Controller Class.**

**Controller classes are a type of class in Object-Oriented**

Analysis and Design (OOAD) that handle the flow of control and manage the

interactions between Boundary classes (responsible for system interfaces) and

Entity classes (responsible for data and business logic). Their primary role is to

process incoming requests, coordinate activities, and direct data between other

components of the system.

Symbol**:**

****

1. **Entity Class**:

In the Use Case perspective, **As an Info of an** **Actor** for person or user his name, profession, person\_id, Mobiile\_num this information has to be recorded by means of Modelling. The Model object knows all about the data that need to be displayed and represented by **Entity Class or Form**

**Class**. **Details of the particular actor who has involved.** It will be in **Data Layer.**

Symbol:

1. **Boundary Classes**

**PayByCashBC()**

**PayByNBBC ()**

**PayByCardBC()**

**AgntMkPayBC()**

1. **Controller Class:**

****

**PayCC()**

1. **Entity Class**:

**CashEC ()**

**NBEC()**

**CardEC()**

**AgentEC()**

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

There will be the guideline to put all classes in 3 tier architecture

1. Place all entity classes in DB Layer
2. Place Primary Actor associated Boundary Class in Application Layer
3. Place controller class in Business Logic Layer
4. If Governing Body influence or reusability is there with any of remaining Boundary classes, place them in Business logic Layer else place them in Application layer

**PayByNBBC ()**

**PayByCardBC()**

**PayByCashBC()**

**AgntMkPayBC()**

Application Layer

****

**PayCC()**

Business Logic Layer

**CashEC ()**

**NBEC()**

**CardEC()**

**AgentEC()**

Data Layer

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

A Domain Model is a conceptual representation of the real-world entities and business logic in a specific domain (area of interest) — like online banking, e-commerce, healthcare, etc.

It shows:

* The main objects/entities
* Their attributes (properties/data)
* Relationships between those entities

**Purpose of a Domain Model:**

* To help developers and stakeholders understand how the system should work.
* It's used during software design and analysis to map out how things interact in a system.
* Often the foundation for object-oriented programming (like classes in Java, C#, etc.)

**What it is:** A **more detailed, technical model** derived from the conceptual model.

**Purpose:** To serve as the **foundation for software design and implementation**.

**Focus:** Includes **classes, attributes, behaviors**, and sometimes **relationships**.

Often includes enough detail for **developers to start coding**.

**Components of a Domain Model**

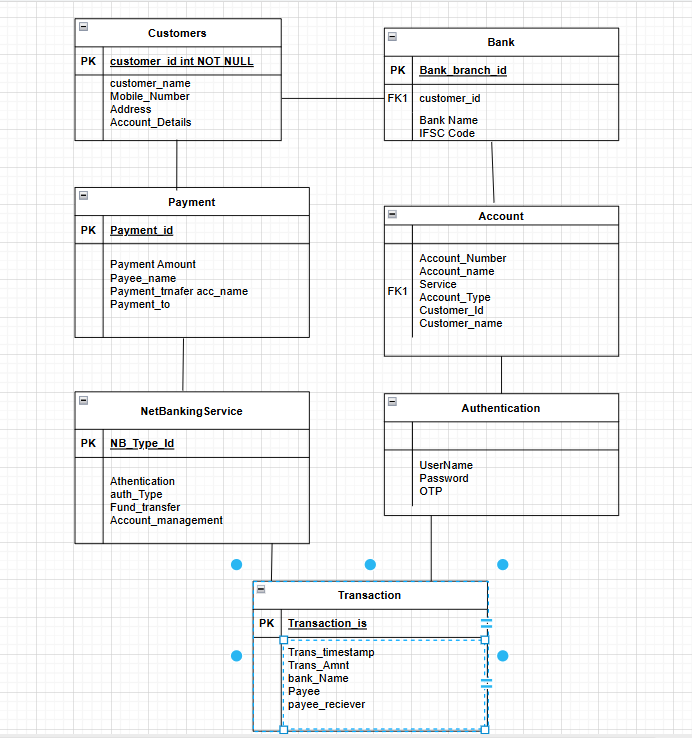
1. **Entities (Classes or Objects):**

These are the "things" in the domain, like: Customer, BankAccount, Transaction, etc.

1. **Attributes:** These are the properties or data each object holds: For example: Customer has name, email, customerId
2. **Relationships:**

Defines how entities are connected:

* A Customer has a BankAccount
* A Transaction involves a Customer and a Beneficiary



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

To understand what a sequence diagram is, it's important to know the role of the [Unified Modeling Language](https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language?usecase=uml), better known as UML. UML is a modelling toolkit that guides the creation and notation of many types of diagrams, including behaviour diagrams, interaction diagrams, and structure diagrams.

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios. Note that there are two types of sequence diagrams: UML diagrams and code-based diagrams.

**Benefit of Sequence diagram:**

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

* Represent the details of a UML use case.
* Model the logic of a sophisticated procedure, function, or operation.
* See how objects and components interact with each other to complete a process.
* Plan and understand the detailed functionality of an existing or future scenario.

**There will be 3 Step Process to draw Sequence diagram:**

1. Identify the classes from Use Case Diagram
2. Place the identified classes on 3 tier Architecture
3. Finding out each sequence of step

**Step 1: Identify the classes from Use Case Diagram**

**Boundary Classes(View/Presentation layer):**

**PayByCashBC()**

**PayByNBBC ()**

**PayByCardBC()**

**AgntMkPayBC()**

**Controller Class(Controller/Business Logic Layer):**

****

**PayCC()**

**Entity Class**(Model/Controller layer):

**CashEC ()**

**NBEC()**

**CardEC()**

**AgentEC()**

**Step 2: Place the identified classes on 3 tier Architecture**

There will be the guideline to put all classes in 3 tier architecture

1. Place all entity classes in DB Layer
2. Place Primary Actor associated Boundary Class in Application Layer
3. Place controller class in Application Layer
4. If Governing Body influence or reusability is there with any of remaining Boundary classes, place them in Business logic Layer else place them in Application layer

****

**PayByCashBC()**

**AgntMkPayBC()**

**PayCC()**

Application Layer

**PayByCardBC()**

**PayByNBBC ()**

Business Logic Layer

**CashEC ()**

**NBEC()**

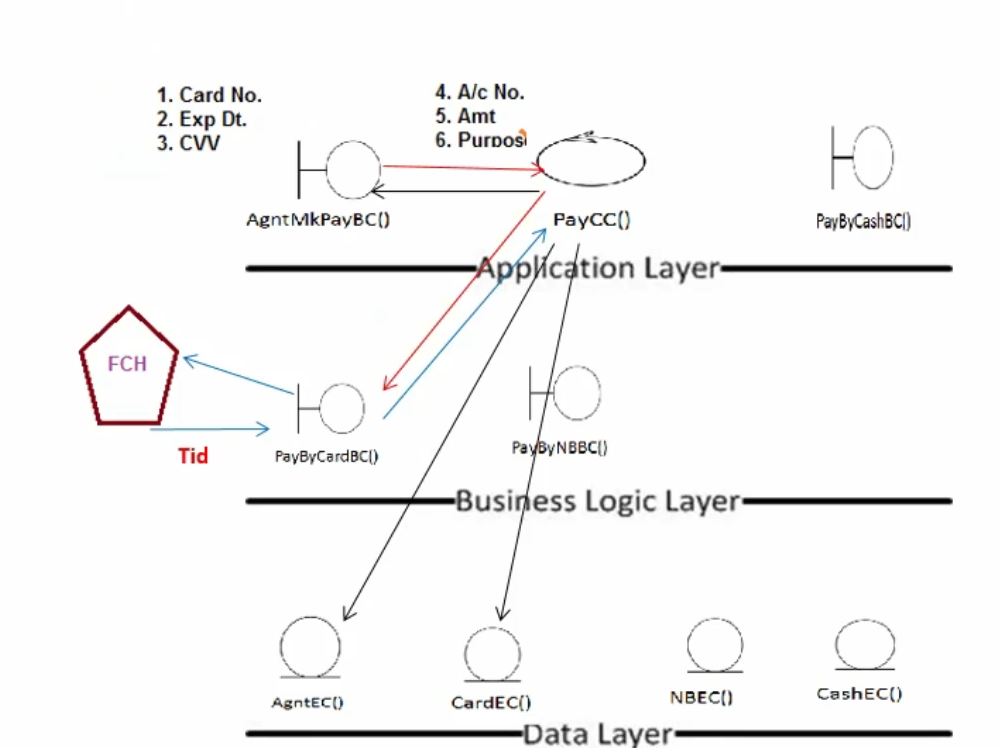
**CardEC()**

**AgentEC()**

Data Layer

**Step 3: Finding out each sequence of step**

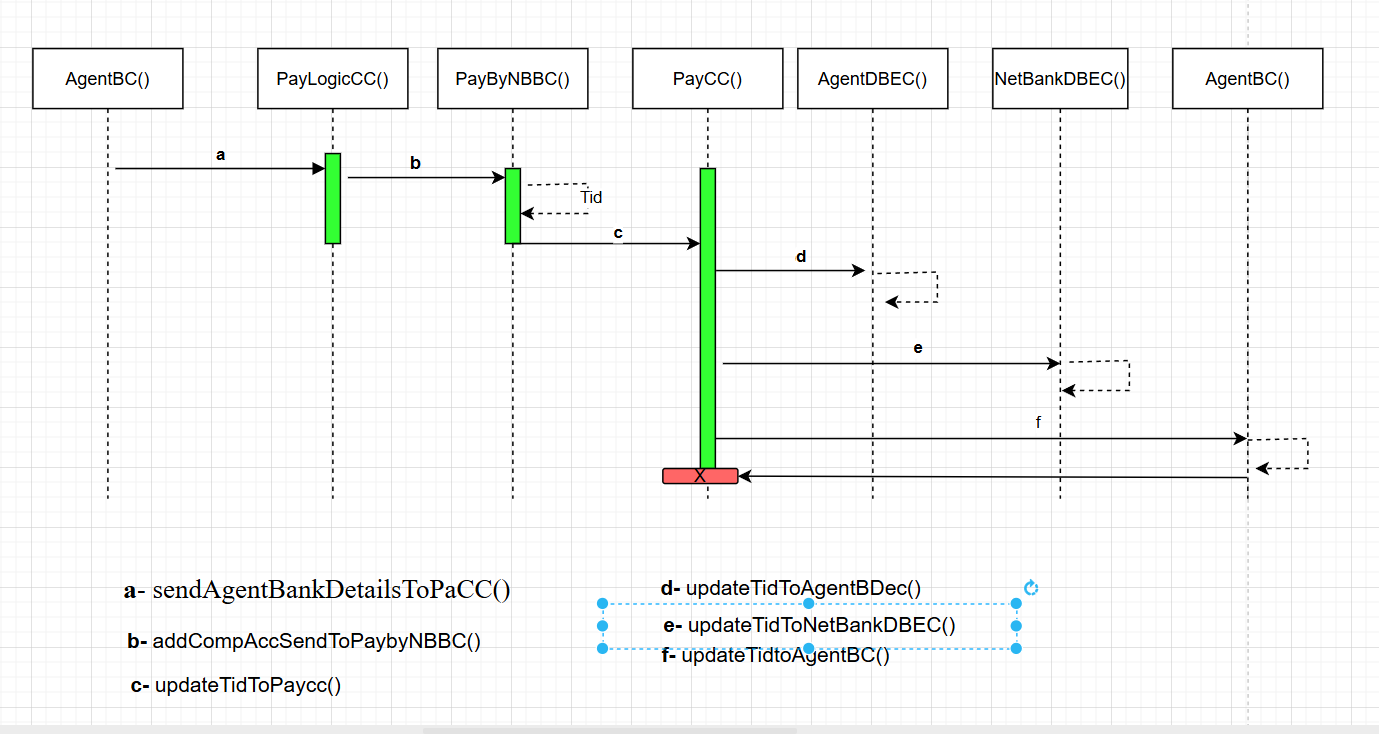
To find each sequence step need to write down all steps



**Process of the above diagram**

1. Customer wants to make Payment via Card so it needs Card no., Exp dt. & CVV which will be used by AgntMkPayBC ().
2. AgntMkPayBC () will go to PayCC() and ask internally for the Account no.,Amount and Purpose.
3. Now it goes with input request to the PayByCardBC() in Application layer, So here User will get Message from third party FCH like “Do not Refresh” and after some time we get Tid with “Transaction successful”
4. The same Tid will update in the AgntEC() Database in Data Layer
5. The same Tid will update in CardEC() Database in Data Layer
6. And then same Tid will get to the Agent in application layer

Note : SO here total 6 steps are there and Third party interaction like FCH will not be consider in step



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

A Conceptual Model is a high-level representation of a system that helps in understanding the key components and their relationships, without going into technical implementation details. For a payment done by a customer using Net Banking, the conceptual model outlines how various entities (like customer, bank, merchant) interact during the payment process.

**Conceptual Model** is an **abstract, high-level representation of a system**. It defines the *structure*, *entities*, *relationships*, and *business rules* in a way that is **technology-agnostic** — meaning it's not tied to any specific programming language or platform.

**Conceptual Model**

* **What it is:** A **high-level, abstract representation** of the system.
* **Purpose:** To help **understand** the problem domain and communicate with **non-technical stakeholders**.
* **Focus:** Focuses on **real-world concepts**, their **relationships**, and **terminology** used in the domain.
* **No technical details** (like classes, methods, databases, etc.).

**Purpose:**

* Understand and analyse the problem domain.
* Communicate clearly with stakeholders (business & technical).
* Form the foundation for detailed system design (logical & physical models).

**Components of Conceptual Modelling:**

**1. Entities**

* These are the main objects/things in the system.
* In our example: Customer, Merchant, Bank, Payment Gateway, Transaction.

| **Entity** | **Attributes** |
| --- | --- |
| Customer | customer\_id, name, email, account\_no |
| Merchant | merchant\_id, name, business\_type |
| Bank | bank\_id, name, IFSC\_code |
| Transaction | transaction\_id, date, amount, status |
| Gateway | gateway\_id, name, security\_protocol |

**2. Attributes**

* Properties or characteristics of entities.
* Example:
  + **Customer**: Customer\_ID (Primary Key) Name Email Phone\_Number, Address
  + **Transaction**: amount, date, status
  + **Bank**: Bank\_ID (Primary Key) Bank\_Name, IFSC\_Code Branch
  + **Net\_Banking**: Account\_ID (Primary Key), Username Password (hashed/encrypted) Account\_Type (e.g., Savings, Current), Balance
  + **Payment**: Payment\_ID (Primary Key), Date\_Time Amount Status (e.g., Success, Failed, Pending), Mode (Net Banking)
  + **Merchant**: Merchant\_ID, (Primary Key), Merchant\_Name, Merchant\_Account\_Number, Merchant\_Bank\_ID (Foreign Key)

**3. Relationships**

Describe how entities are connected or interact.

* **Customer –– owns ––> Net\_Banking\_Account**
* One customer can have one or more Net Banking accounts.
* **Cardinality**: 1-to-many
* **Net\_Banking\_Account –– is with ––> Bank**
* Each account is associated with a specific bank.
* **Cardinality**: many-to-1
* **Customer –– makes ––> Payment**
* A customer initiates one or more payments.
* **Cardinality**: 1-to-many
* **Payment –– is processed via ––> Net\_Banking\_Account**
* A payment is made using one Net Banking account.
* **Cardinality**: many-to-1
* **Payment –– goes to ––> Merchant**
* Each payment is sent to a merchant.
* **Cardinality**: many-to-1
* **Merchant –– has bank ––> Bank**
* A merchant is associated with a bank (where payment is deposited).
* **Cardinality**: many-to-1

**4. Processes / Functions**

* Activities that occur in the system.
* Example:
  + Login authentication
  + Payment authorization
  + Fund transfer
  + Confirmation notification

**5. Constraints / Business Rules**

* Conditions that must be satisfied.
* Example:
  + A transaction must not exceed the account balance.
  + Only verified users can initiate transactions.

**Types of Conceptual Models:**

1. **ER Diagrams (Entity-Relationship Diagrams)**
   * Focuses on entities, attributes, and relationships.
   * Very common in database and information system modeling.
2. **Use Case Diagrams (UML)**
   * Shows interactions between users and the system.
   * Useful in understanding system requirements.
3. **Data Flow Diagrams (DFD)**
   * Focuses on how data flows through the system.
   * Identifies processes, data stores, and external entities.
4. **Class Diagrams (UML)**
   * Describe the classes (like entities), their attributes, and methods.
   * Often used for object-oriented modeling.

**Benefits of Conceptual Modelling:**

* **Clarity:** Everyone from business users to developers can understand it.
* **Foundation:** Guides development, testing, and documentation.
* **Validation:** Ensures all user requirements are captured.
* **No Tech Jargon:** Avoids premature technical decisions.

**In Summary:**

Conceptual modelling is like **sketching the blueprint of a house** before building it. You're figuring out what the system is, what it must do, and how parts relate—without touching code or databases yet.

Top of Form

Bottom of Form

**Customer (Customer\_ID, Name, Email, Phone\_Number, Address)**

**|**

**| owns**

**↓**

**Net\_Banking\_Account (Account\_ID, Username, Password, Account\_Type, Balance, Bank\_ID)**

**|**

**| is with**

**↓**

**Bank (Bank\_ID, Bank\_Name, IFSC\_Code, Branch)**

**Customer**

**|**

**| makes**

**↓**

**Payment (Payment\_ID, Date\_Time, Amount, Status, Mode, Account\_ID, Merchant\_ID)**

**|**

**| goes to**

**↓**

**Merchant (Merchant\_ID, Merchant\_Name, Merchant\_Account\_Number, Merchant\_Bank\_ID)**

**|**

**| has bank**

**↓**

**Bank**

**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, or Model-View-Controller, is a software design pattern that separates an application's logic into three interconnected parts: the Model, the View, and the Controller. This separation of concerns enhances code organization, maintainability, and testability. MVC is widely used in web application development and other software projects.

Here's a breakdown of each component:

* **Model**:

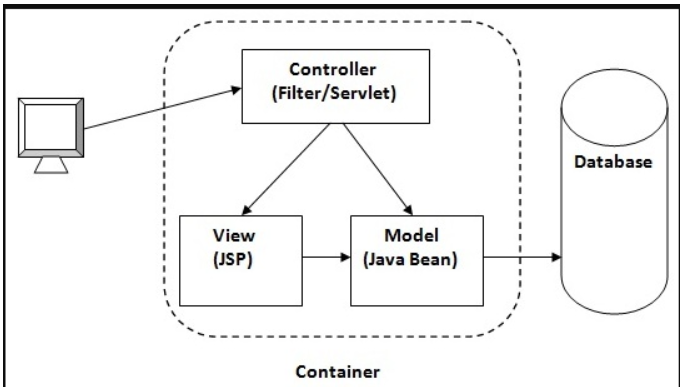
The Model handles the data and logic related to the application's functionality. It manages data storage, retrieval, and manipulation.

* **View**:

The View is responsible for presenting the data to the user. It displays the information from the Model in a user-friendly format.

* **Controller**:

The Controller acts as an intermediary between the Model and the View. It receives user input, interacts with the Model to perform actions, and then updates the View to reflect the changes.



**Benefits of MVC:**

* Improved Code Organization: MVC helps break down large applications into smaller, more manageable components.
* Enhanced Maintainability: The separation of concerns makes it easier to modify or update specific parts of the application without affecting other parts.
* Increased Testability: Each component can be tested independently, making it easier to identify and fix errors.
* Reusable Components: MVC promotes the creation of reusable components that can be used in different parts of the application or even in other projects.
* Scalability: MVC allows for easier scaling of the application as it grows in complexity.

**MVC Architecture Rules:**

1. One Actor and One use case results in one Boundary class
2. Two Actors and **one use** case results in two boundary Classes
3. Three Actor and One Use Case results in Three Boundary Class

Note: Only one Primary actor is to be considered with use case

1. Every use case will have a controller class
2. Every Actor will become an Entity Class

**There will be the guideline to put all classes in 3 tier architecture**

1. Place all entity classes in DB Layer
2. Place Primary Actor associated Boundary Class in Application Layer
3. Place controller class in Application Layer
4. If Governing Body influence or reusability is there with any of remaining Boundary classes, place them in Business logic Layer else place them in Application layer

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

The Waterfall Model is a linear and sequential approach to software development. Each phase must be completed before moving to the next.

BA (Business Analyst) Contributions by Stage:

* 1. **Requirement Gathering & Analysis**
* Primary Stage for BA Involvement
* **Key Contributions:**
  + Meet with stakeholders to elicit requirements
  + Conduct interviews, surveys, workshops
  + Create Business Requirement Documents (BRD), Use Case Diagrams, Process Flows
  + Define scope and project objectives
  + Ensure requirements are clear, complete, and feasible
  1. **System Design**
* BA provides functional input for the design team.
* **Key Contributions:**
  + Translate business needs into Functional Specification Documents (FSD)
  + Collaborate with solution architects/UI designers
  + Validate whether proposed design meets business requirements
  + Define data flow diagrams, wireframes, mockups (if needed)
  1. **Implementation (Coding)**
* Developers begin coding based on designs and specs.
* **Key Contributions:**
  + Act as a bridge between developers and stakeholders
  + Clarify any ambiguities in requirements
  + Ensure alignment of development with business goals
  + Handle change requests and assess their impact
  1. **Testing**
* BA supports the QA team to ensure software quality.
* **Key Contributions:**
  + Create or review test cases and scenarios
  + Validate that each requirement is covered in testing (traceability matrix)
  + Participate in User Acceptance Testing (UAT)
  + Help detect and prioritize defects based on business impact
  1. **Deployment**
* Software is released to production.
* **Key Contributions:**
  + Assist in rollout planning
  + Provide support documents for users and stakeholders
  + Coordinate with stakeholders for go-live readiness
  + Facilitate training or user onboarding
  1. **Maintenance**
* Post-deployment support and changes.
* **Key Contributions:**
  + Monitor system performance from a business perspective
  + Collect feedback from users
  + Document enhancement requests
  + Help in analysing new requirements for future releases

**What Documents to Write?**

Business analyst is the mediator between the client and technical team. His prime responsibility to get the requirement, prioritize and convey to the team. In this phase BA prepares documentation in following style.

There will be three phases for BA with respect to the Documentation

* + 1. **Preproject**:
* In Preproject Phase, first client will come up with the RFP (Request For Proposal) or RFI (Request For Information) or SOW (Statement of Work)
* Depend upon RFP, RFI or SOW Senior BA or Senior Consultant will do analysis first like what Enterprise analysis, gap analysis, feasibility study, Decision analysis to understand what client is all about, what is his organization do, status of an organization, whether there is any risk involve in the requirement or opportunity rendering all these things he summarizes into the PBA.
* Then it will be going to share with the Business Sponsor and Business Consultant.
* In PBA will have number of solution and for each solution there will be ranking to the solution and also it involves for each solution what will be the cost, time and budget and resources will require.
* The solution should be quantitative not only qualitative
* After going through BCD, client will choose one solution then that solution will get freeze.
  + 1. **Implementation:**

Planning:

* Once solution got freeze, then it will go for another phase of the requirements
* In Between Implementation & preproject phase, There will be planning phase.
* In planning phase, PM prepares Gantt chart which shows Mapping of the resources to the task, time estimation of each with that task. Also PM will decide which model has to follow for the project
* Senior BA involve in this phase will decide which strategy has to follow, how many documents has to prepare, what should be the standard of the document, document are fully compliant or partially done. Strategy also involves how to prioritize the requirement? Validate the requirement? How to trace the requirement?

Implementation 1.0:

* Once PBA done and requirement got freeze then PM prepare project planning and senior BA will strategy planning. After this Implementation phase will start and from here only BA will start work because now Senior BA will give his PBA and strategy to follow
* The respective document for development stages has shown in below Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Preproject | | Implementation | | | | Support | |
|  | | **Planning** | | **Implementation** | |  | |
| EA | RFP, RFI, SOW,  Business Case Document | PM & Sr. BA will prepare PBA (Planning Business Analysis) | | **Req. Gathering** | BRD | Enhancement/  Change Management | CR,  CRT |
| SWOT | **Req. Analysis** | FRS,  SRS,  RTM,  SSD |
| RCA | **Design** | HDD, ADD |
| Strategy | **Coding** | LDD, CDD |
| GAP | **Testing** | TCD |
| FS | **Deployment** | User Manual, Training Manual |
| DA |  |  |  |  |

* + 1. **Support phase:**
* In support type of project need to understand the CR (Change Request) and Change management from
* client and has to prepare the doc.BA will also prepare the CRT( Change Request Tracker)
* In case of new version or release based on existing BRD new version of BRD will prepare
* BA will prepare User Manuals for Client

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

Conflict Management - Conflict Management is the process of identifying,

addressing, and resolving disagreements or disputes in a constructive manner to

prevent escalation and maintain productive relationships. It involves techniques and

In summary, the Thomas-Kilmann Conflict Mode Instrument (TKI) provides a valuable framework for understanding and managing conflict. By becoming more aware of different conflict handling styles and their potential outcomes, individuals can learn to navigate disagreements in a more constructive and effective manner, leading to better relationships and outcomes.

Using the Thomas-Kilmann Model:

**Key Objectives of Conflict Management are** –

* + 1. Resolve Disputes Constructively: Focus on solutions that satisfy all parties

involved.

* + 1. Maintain Relationships: Preserve trust and respect between individuals or teams.
    2. Improve Collaboration: Use conflict as an opportunity to generate new ideas and better understanding.

4. Enhance Productivity: Minimize disruptions caused by unresolved conflicts.

5. Promote Healthy Communication: Encourage open dialogue and mutual respect

**Thomas - Kilmann approach** - The Thomas-Kilmann Conflict Management Model is a

framework developed by Kenneth W. Thomas and Ralph H. Kilmann that identifies

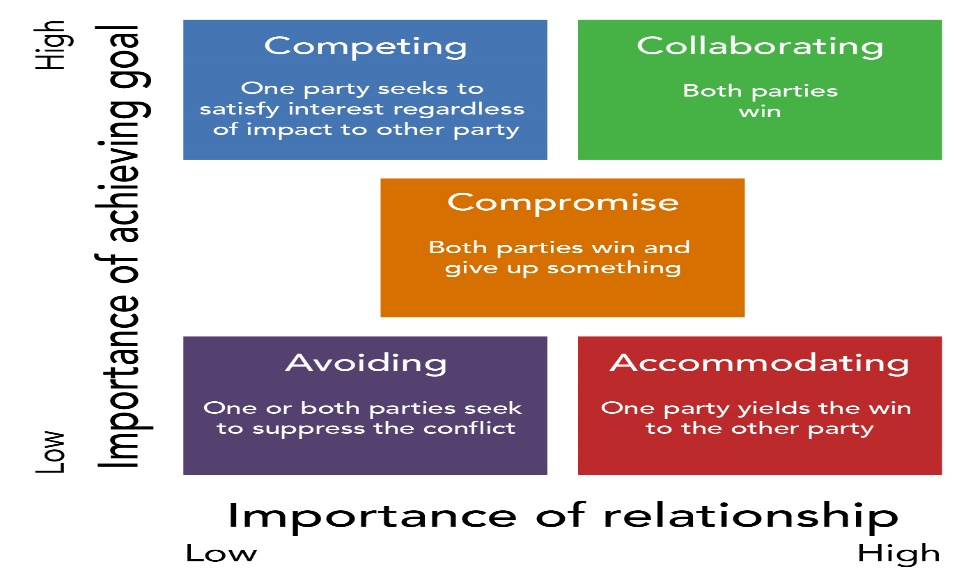
five primary styles of handling conflict based on two dimensions: assertiveness (the

degree to which a person seeks to satisfy their own concerns) and cooperativeness

(the degree to which a person seeks to satisfy the concerns of others). The model

emphasizes that there is no one-size-fits-all approach, and the choice of style

depends on the situation and goals



**There are 5 Conflict Management Styles as per the Thomas Kilmann technique**

* + - 1. **Competing (High Assertiveness, Low Cooperativeness)** :
* Description: The individual pursues their own interests at the expense of others.
* Use Case: Best for quick, decisive action or when the outcome is critical, such as during emergencies.
* Example: Enforcing a strict deadline despite objections.
  + - 1. **Accommodating (Low Assertiveness, High Cooperativeness)** :
* Description: The individual puts the other party's needs above their Own
* Use Case: Best when maintaining relationships is more important than the issue at hand or when the issue is trivial to you.
* Example: Agreeing to a colleague's suggestion to avoid conflict.
  + - 1. **Avoiding (Low Assertiveness, Low Cooperativeness)** :
* Description: The individual sidesteps or postpones the conflict without addressing it directly.
* Use Case: Best when the issue is minor, emotions are high, or there’s a need to gather more information.
* Example: Delaying a heated discussion until emotions cool down.
  + - 1. **Compromising (Moderate Assertiveness, Moderate Cooperativeness):**
* Description: Both parties make concessions to reach a mutually acceptable solution.
* Use Case: Best when a quick, temporary solution is needed or when both parties have equally strong positions.
* Example: Splitting resources evenly between two departments
  + - 1. **Collaborating (High Assertiveness, High Cooperativeness)**
* Description: The individual works with the other party to find a win-win solution that fully satisfies both sides.
* Use Case: Best when the issue is important to both parties and requires a creative, long-term solution.
* Example: Jointly designing a new project strategy that incorporates everyone's ideas

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

Projects can fail for a variety of reasons, often due to poor planning,

mismanagement, or unforeseen circumstances. Below is a list of common factors

that contribute to project failure:

1. **Poor Project Planning:**

* Lack of clear goals and objectives.
* Insufficient detail in the project plan.
* Unrealistic timelines and resource estimates.
* Failure to identify and address potential risks.
* 2. Inadequate Requirements Management
* •Misunderstanding or incomplete requirements.
* •Frequent changes in requirements without proper change control.
* •Lack of stakeholder involvement in defining requirements.
* 3. Weak Leadership and Governance
* •Inexperienced or unengaged project managers.
* •Lack of executive sponsorship or support.
* •Poor decision-making or lack of accountability.
* 4. Ineffective Communication
* •Breakdown in communication between stakeholders and team members.
* •Unclear or inconsistent instructions and updates.
* •Failure to share critical information in a timely manner.
* 2. Inadequate Requirements Management
* •Misunderstanding or incomplete requirements.
* •Frequent changes in requirements without proper change control.
* •Lack of stakeholder involvement in defining requirements.
* 3. Weak Leadership and Governance
* •Inexperienced or unengaged project managers.
* •Lack of executive sponsorship or support.
* •Poor decision-making or lack of accountability.
* 4. Ineffective Communication
* •Breakdown in communication between stakeholders and team members.
* •Unclear or inconsistent instructions and updates.
* •Failure to share critical information in a timely manner.
* 2. Inadequate Requirements Management
* •Misunderstanding or incomplete requirements.
* •Frequent changes in requirements without proper change control.
* •Lack of stakeholder involvement in defining requirements.
* 3. Weak Leadership and Governance
* •Inexperienced or unengaged project managers.
* •Lack of executive sponsorship or support.
* •Poor decision-making or lack of accountability.
* 4. Ineffective Communication
* •Breakdown in communication between stakeholders and team members.
* •Unclear or inconsistent instructions and updates.
* •Failure to share critical information in a timely manner.
* 2. Inadequate Requirements Management
* •Misunderstanding or incomplete requirements.
* •Frequent changes in requirements without proper change control.
* •Lack of stakeholder involvement in defining requirements.
* 3. Weak Leadership and Governance
* •Inexperienced or unengaged project managers.
* •Lack of executive sponsorship or support.
* •Poor decision-making or lack of accountability.
* 4. Ineffective Communication
* •Breakdown in communication between stakeholders and team members.
* •Unclear or inconsistent instructions and updates.
* •Failure to share critical information in a timely manner.

1. **Inadequate Requirements Management:**

* Misunderstanding or incomplete requirements.
* Frequent changes in requirements without proper change control.
* Lack of stakeholder involvement in defining requirements.

1. **Weak Leadership and Governance**

* Inexperienced or unengaged project managers.
* Lack of executive sponsorship or support.
* Poor decision-making or lack of accountability.

1. **Ineffective Communication**

* Breakdown in communication between stakeholders and team members.
* Unclear or inconsistent instructions and updates.
* Failure to share critical information in a timely manner.
* Resource Constraints
* •Insufficient funding or budget overruns.
* •Shortage of skilled personnel or key resources.
* •Overloading team members with unrealistic workloads.
* 6. Scope Creep
* •Uncontrolled expansion of project scope without adjusting resources, time, or
* budget.
* •Lack of a formal process to manage changes in scope.
* 7. Poor Risk Management
* •Failure to identify, assess, or mitigate risks.
* •Ignoring potential threats to the project’s success.
* •Underestimating the impact of external factors like market changes or
* economic conditions.
* Resource Constraints
* •Insufficient funding or budget overruns.
* •Shortage of skilled personnel or key resources.
* •Overloading team members with unrealistic workloads.
* 6. Scope Creep
* •Uncontrolled expansion of project scope without adjusting resources, time, or
* budget.
* •Lack of a formal process to manage changes in scope.
* 7. Poor Risk Management
* •Failure to identify, assess, or mitigate risks.
* •Ignoring potential threats to the project’s success.
* •Underestimating the impact of external factors like market changes or
* economic conditions.

1. **Resource Constraints**

* Insufficient funding or budget overruns.
* Shortage of skilled personnel or key resources.
* Overloading team members with unrealistic workloads.

1. **Scope Creep**

* Uncontrolled expansion of project scope without adjusting resources, time, or budget.
* Lack of a formal process to manage changes in scope.

1. **Poor Risk Management**

* Failure to identify, assess, or mitigate risks.
* Ignoring potential threats to the project’s success.
* Underestimating the impact of external factors like market changes or economic conditions.

1. **Unrealistic Expectations**

* Overpromising to stakeholders or customers.
* Setting unachievable goals without proper analysis.

1. **Lack of Stakeholder Engagement**

* Failure to involve key stakeholders in decision-making.
* Misalignment between stakeholder expectations and project deliverables.
* Resistance to change from stakeholders or team members.

1. **Inadequate Quality Management**

* Insufficient testing or quality assurance.
* Deliverables that do not meet the agreed-upon standard
* Neglecting customer or end-user feedback.

1. **Technology Issues**

* Reliance on outdated or incompatible technologies.
* Technical failures or bugs that disrupt progress.
* Underestimating the complexity of technical requirements.

1. **Organizational Challenges**

* Lack of alignment between project goals and organizational strategy.
* Internal politics or power struggles affecting decision-making.
* Resistance to new processes or systems within the organization.

1. **External Factors**

* Changes in regulatory requirements or compliance issues.
* Economic downturns, market shifts, or supply chain disruptions.
* Natural disasters or other unforeseen events
* 14.Failure to Monitor and Control
* •Inadequate tracking of project progress against the plan.
* •Ineffective use of tools and metrics for project management.
* 15.Cultural and Team Issues
* •Lack of collaboration or teamwork.
* •Conflicts within the team that are not resolved effectively.
* •Cultural differences leading to miscommunication or misunderstandings
* 14.Failure to Monitor and Control
* •Inadequate tracking of project progress against the plan.
* •Ineffective use of tools and metrics for project management.
* 15.Cultural and Team Issues
* •Lack of collaboration or teamwork.
* •Conflicts within the team that are not resolved effectively.
* •Cultural differences leading to miscommunication or misunderstandings

1. **Failure to Monitor and Control**

* Inadequate tracking of project progress against the plan.
* Ineffective use of tools and metrics for project management.

1. **Cultural and Team Issues**

* Lack of collaboration or teamwork.
* Conflicts within the team that are not resolved effectively.
* Cultural differences leading to miscommunication or misunderstandings

**Question – 11 List the Challenges faced in projects for BA.**

Business Analysts (BAs) play a critical role in bridging the gap between stakeholders and technical teams. However, they often face challenges that can impact project success.

Here’s a list of common challenges faced by BAs in projects:

1. **Unclear or Evolving Requirements** - Stakeholders may not fully understand or articulate their need, Frequent changes in requirements disrupt planning and development, Lack of clarity in requirements documentation.

2. **Managing Stakeholders** - Identifying and engaging all relevant stakeholders, handling conflicting stakeholder priorities or expectations, gaining stakeholder buy-in for decisions or deliverables.

3. **Communication Barriers** - Bridging the gap between technical teams and non- technical stakeholders, Overcoming misunderstandings due to jargon or differing perspectives, Lack of timely feedback from stakeholders or team members.

4. **Time Constraints** - Tight deadlines to gather, document and validate requirements, Pressure to complete tasks quickly, compromising quality, balancing multiple projects or competing priorities

5. **Lack of Stakeholder Involvement** - Stakeholders not dedicating enough time to the project, Delayed decisions or inputs from stakeholders, Resistance from stakeholders to change existing processes or systems.

6. **Handling Scope Creep** - Managing unplanned changes or additions to project scope, Lack of proper change control mechanisms, Difficulty maintaining focus on core project objectives.

7. **Insufficient Domain Knowledge** - Difficulty understanding complex or unfamiliar business domains, Limited access to Subject Matter Experts (SMEs) for clarification, need to quickly learn and adapt to industry-specific terminologies or practices.

8. **Conflict Resolution** - Mediating conflicts between stakeholders with differing priorities, resolving disputes within the project team, such as developers vs. testers, Balancing the interests of multiple departments or business units.

9. **Technical Constraints** - Aligning business needs with existing technology limitations, understanding technical jargon and constraints imposed by development teams, Ensuring compatibility with legacy systems or third-party integrations.

10.**Unrealistic Expectations** - Stakeholders expecting quick solutions for complex problems, Pressure to deliver beyond the project’s scope or capabilities. Unrealistic assumptions about technology capabilities or timelines

**Q12. Write about Document Naming Standards**

File Naming Standards are used to save the file with particular name or format. This is important in sharing and keeping track of data files.

The following are the best standards in Naming Convention –

1. It should be Named Consistently.

2. File names should be short (<25 characters)

3. Avoid special characters or spaces in a file name.

4. Use Capital and Underscores instead of spaces or slashes.

5. Use date format as per ISO 8601: YYMMDD

6. Include a version number.

7. Write down naming convention.

We must consider following naming conventions –

* Date of Creation
* Short Description
* Work
* Location
* Project name or number
* Sample
* Analysis
* •Version Number
* For example – We have a project with ID “PROJ456BANK” and we are working with
* Requirement Specification Document then –
* Project ID - PROJ456BANK
* Document Type – REQ
* Version – 1.0
* Date – 2024-12-18
* Then the naming convention of the document will be “PROJ456BANK-REQ-1.0-
* 2024-12-18”
* Analysis
* •Version Number
* For example – We have a project with ID “PROJ456BANK” and we are working with
* Requirement Specification Document then –
* Project ID - PROJ456BANK
* Document Type – REQ
* Version – 1.0
* Date – 2024-12-18
* Then the naming convention of the document will be “PROJ456BANK-REQ-1.0-
* 2024-12-18”
* Analysis
* •Version Number
* For example – We have a project with ID “PROJ456BANK” and we are working with
* Requirement Specification Document then –
* Project ID - PROJ456BANK
* Document Type – REQ
* Version – 1.0
* Date – 2024-12-18
* Then the naming convention of the document will be “PROJ456BANK-REQ-1.0-
* 2024-12-18”
* Analysis
* Version Number

For example – We have a project with ID “PROJ456BANK” and we are working with

Requirement Specification Document then

* Project ID - PROJ456BANK
* Document Type – REQ
* Version – 1.0
* Date – 2024-12-18

Then the naming convention of the document will be “PROJ456BANK-REQ-1.0- 2024-12-18”

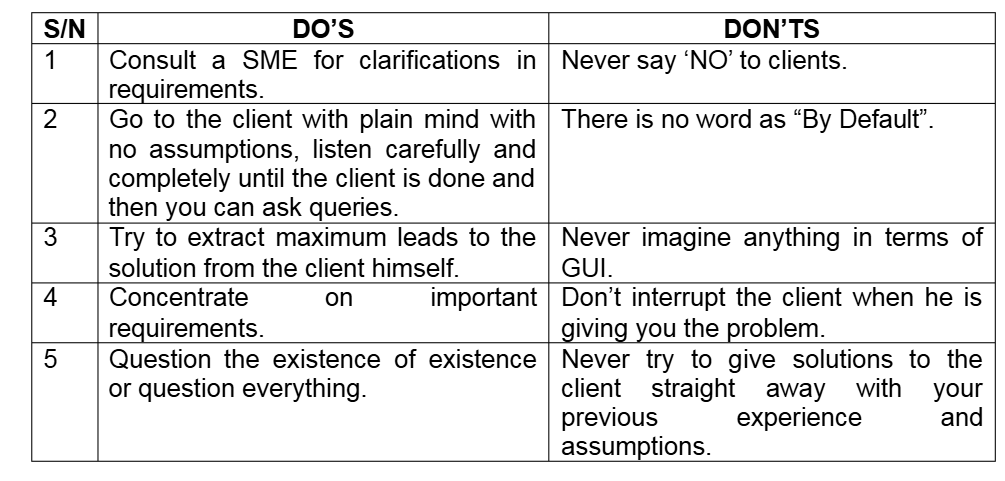
**Q13. What are the Do’s and Don’ts of a Business analyst**

**Do’s of a Business Analyst**

1. **Understand Business Needs Clearly**
   * Always begin by gathering and analysing business requirements thoroughly.
   * Engage stakeholders to clarify objectives.
2. **Communicate Effectively**
   * Use clear, concise, and non-technical language when needed.
   * Be an active listener and ensure two-way communication.
3. **Document Requirements Properly**
   * Use standard formats like BRD, FRD, Use Cases, and User Stories.
   * Keep documentation updated and version-controlled.
4. **Collaborate with Stakeholders**
   * Maintain strong relationships with both business and technical teams.
   * Facilitate meetings, workshops, and reviews.
5. **Use Analytical Thinking**
   * Break down complex problems into manageable parts.
   * Identify gaps, risks, and opportunities proactively.
6. **Validate Requirements**
   * Ensure requirements meet business goals.
   * Perform validation through prototyping, walkthroughs, or test cases.
7. **Stay Solution-Focused**
   * Recommend solutions that are feasible, scalable, and aligned with goals.
   * Support project goals, not just the initial ideas.
8. **Stay Updated**
   * Continuously improve domain knowledge, tools, and methodologies.
   * Learn trends in Agile, Scrum, Business Process Modeling, etc.

**Don’ts of a Business Analyst**

1. **Don’t Assume or Guess Requirements**
   * Never work on assumptions—validate with stakeholders.
2. **Don’t Use Technical Jargon Unnecessarily**
   * Avoid confusing non-technical stakeholders with complex terms.
3. **Don’t Ignore Stakeholders**
   * Even less vocal stakeholders may have critical input—engage everyone.
4. **Don’t Skip Documentation**
   * Poor documentation can lead to miscommunication and scope creep.
5. **Don’t Resist Feedback**
   * Be open to critique and willing to revisit your analysis.
6. **Don’t Focus Only on “What” – Understand “Why”**
   * Understanding the purpose behind a requirement adds strategic value.
7. **Don’t Overpromise**
   * Be realistic about timelines, feasibility, and impact.
8. **Don’t Work in Isolation**
   * Business analysis is a collaborative role—teamwork is key.



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

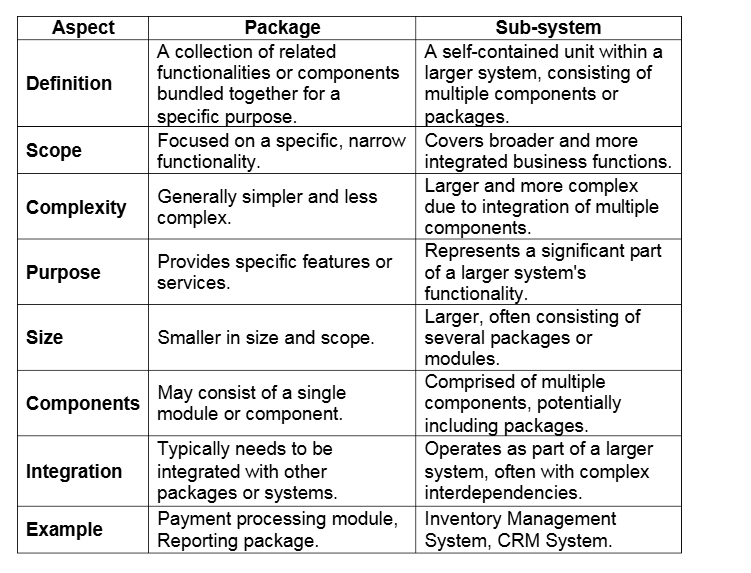
**Packages** – Packages are the collection of components which are not reusable in nature.

Example – Application development companies work on Packages.

**Sub Systems** – Sub Systems are the collection of components which are reusable in

nature.

Example – Product development companies work on Sub Systems



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

Camel-casing is a naming convention in programming where the first word starts in lowercase and each subsequent word starts with an uppercase letter, with no spaces or underscores between words.

Example:

* customerName
* orderTotalAmount
* isLoggedIn

**Where is Camel-Casing Used?**

Camel-casing is commonly used in:

1. Variable names  
   E.g., totalPrice, userEmail
2. Function or method names  
   E.g., calculateDiscount(), getCustomerDetails()
3. Property names in classes/objects  
   E.g., productId, orderStatus

**Types of Camel Case:**

* Lower Camel Case (starts with lowercase) → userName
* Upper Camel Case / Pascal Case (starts with uppercase) → UserName  
  (Used for class or interface names)

**Why Do We Use Camel Casing? – Advantages**

Camel casing is widely used in programming because it offers several benefits:

**1. Improves Readability**

* Combines multiple words into one without using spaces or underscores.
* Capital letters help visually separate words.
* **Example**:  
  useraccountstatus → hard to read  
  userAccountStatus → easy to understand

**2. Follows Coding Standards**

* Many programming languages and frameworks recommend or enforce camel casing for consistency.
* Helps developers write cleaner, more maintainable code.

**3. Avoids Special Characters**

* Unlike snake\_case, camelCase does not use underscores, making it ideal for languages or tools that restrict special characters.

**4. Helps in Auto-completion**

* IDEs and code editors can better suggest camelCase names during typing, improving developer efficiency.

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

Development server - A Development Server is a computer or environment where developers build, test and deploy software applications or services. It is a dedicated machine or instance used for development purposes, often running on a local or remote server. A development server allows developers to code and test their applications before moving them to production.

A Business Analyst can have below types of access based on the needs

1. **ReadOnly Access** – BA may be granted with the ReadOnly access to the development server. This will allow them to view the user interface of the application, navigate through the features and also they will be able to observe the behaviour of the application.

2. **Limited Access** – Depending upon the project needs, the BA’s will be granted limited access to the specific modules in the application.

3. **Limited Configuration Access** - Means BA have the authority to make changes in certain areas of application where they have the access

**Key Features:**

* Contains code under development.
* May have test data instead of real data.
* Frequent code changes occur here.
* Not accessible to end-users or clients.

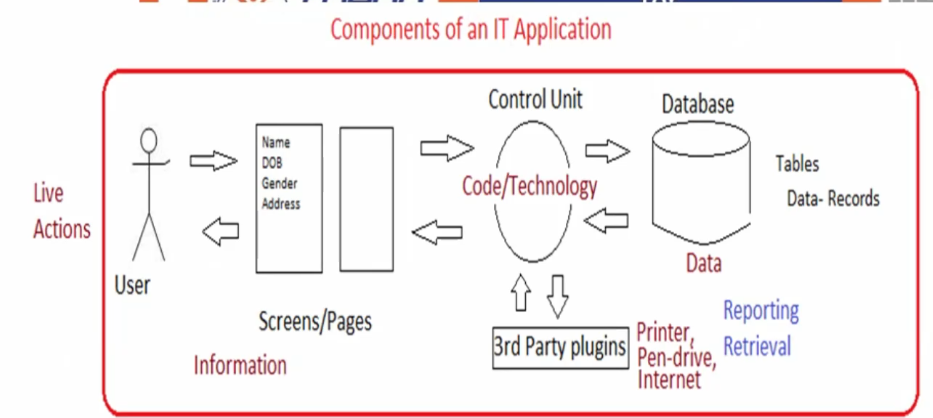
**Accesses a Business Analyst Has on Development Server:**

A Business Analyst (BA) may have limited or read-only access, depending on the organization and project needs. Typical access includes:

1. **Access to View UI Prototypes or Builds**
   * To validate if development aligns with requirements.
   * To perform basic walkthroughs for demo purposes.
2. **Access to Logs or Test Data (Optional)**
   * For analyzing defects, workflows, or logic flow.
   * Sometimes used to support testers or developers.
3. **No Access to Modify Code**
   * BAs do not write or modify code.
   * Their role is more focused on observation, validation, and requirement clarification.
4. **Access to Development Tools or Issue Trackers**
   * Tools like JIRA, Confluence, or Trello may be linked to the Dev Server.
   * Used to track progress, raise queries, or log requirements.

**Q17. What is Data Mapping**

Data Mapping - Data Mapping is the process of establishing relationships between two distinct data models. It is the technique used to link data from one system, database, or format to another, ensuring that the data is correctly interpreted and transferred. This process is crucial when integrating different systems, migrating data, or transforming data between formats for analysis or reporting.



Object-CAR

|  |
| --- |
| Identity Regd Number |
| State-Attribute-Values    Colour - Red  Make - Maruti  Fuel – Petrol |

**Table Name Entities Record**

|  |  |  |  |
| --- | --- | --- | --- |
| **CAR** | | | |
| **Identity** | **Colour** | **Make** | **Fuel** |
| **Regd Number** | Red | Maruti | Petrol |

The database engineers will design DB based on given input by the engineer like an Object he received as shown in above diagram Object as Car.

**Table Name**: Object name will be considered as table name

**Entities:** Identity, field names like colour, Make and fuel or Attribute known as Entities

**Record:** Values of each all attributes in column is known as Record

**Data mapping is essential for:**

* Data Migration
* System Integration
* Data Transformation
* ETL (Extract, Transform, Load) processes

It ensures data moves **accurately and consistently** between systems or formats.

**Types of Data Mapping**

**1.** **One-to-One Mapping:** Each field in the source corresponds to a single field in the target. Example: A field "FirstName" in the source database maps directly to the "First\_Name" field in the target database.

**2.** **One-to-Many Mapping:** One field in the source maps to multiple fields in the target. Example: A "FullAddress" field in the source could map to multiple fields such as "Street", "City", "State", and "ZipCode" in the target.

**3. Many-to-One Mapping:** Multiple fields in the source are combined or consolidated into a single field in the target. Example: First name and lastname fields from the source system could be combined into a single "FullName" field in the target system.

**4. Many-to-Many Mapping**: Multiple fields in the source are mapped to multiple fields in the target. This is often used when integrating systems with complex data relationships. Example: Multiple product categories in the source system could map to various sub-categories in the target system.

**Benefits of Data Mapping:**

* Ensures data consistency across systems.
* Enables accurate data migration and integration.
* Supports reporting, analytics, and decision-making.
* Helps in data validation and error tracking**.**

**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**

Application Programming Interface or API - Application Programming Interface (API) is a set of rules, protocols, and tools that allow different software applications to communicate and interact with each other. APIs define how different software components should interact, enabling one system or service to access features or data from another without needing to understand the internal workings of the other system.

**Key components of API are described below**

1. **API Endpoint:** A specific URL or URI (Uniform Resource Identifier) that represents a specific function or resource available via the API. Example: https://api.example.com/users (Endpoint for fetching user data)

2. **HTTP Methods (Verbs):** These methods define the type of operation to be performed on the resource:

GET: Retrieve data from the API.POST: Send data to the API, usually to create a new resource.

PUT: Update an existing resource.

DELETE: Remove a resource.

3. **Request Headers:** Metadata sent along with the request, such as authentication tokens, content types, or session IDs.

4. **Request Body:** Data sent along with the request, usually in JSON or XML format, that contains the necessary information for the API to process.

5. **Response Body**: The data returned by the API after processing the request, typically in a structured format like JSON or XML.

6. **Authentication and Authorization:** Many APIs require security mechanisms to verify the identity of the user or system making the request (e.g., via API keys, OAuth, or JWT tokens)

**Benefits of APIs –**

**1. Efficiency:** APIs allow businesses and developers to reuse existing software, components, or services, speeding up development time.

**2. Scalability:** APIs enable systems to scale by allowing new services or components to be added easily without disrupting existing systems.

**3. Integration:** APIs allow for the seamless integration of external services, platforms, or data sources, enabling systems to communicate and share information across different environments.

**4. Security:** APIs can offer controlled access to services or data, enabling security features like authentication, rate limiting, and encryption to ensure safe data transmission

5. **Modularity**: APIs promote modular design by allowing applications to be broken down into smaller, independent services, which can be updated or replaced without affecting the whole system.

**For the above scenario, we can follow below procedure**

**1. Establish API communication** - set up API communication between your application and other application to exchange data.

**2. Do Data formatting**- while sending the data from one application to another,

convert the date format from dd-mm-yyyy to mm-dd-yyyy.

3. While receiving the data from another application, parse the data and extract

the date, month and year and re-arrange them accordingly.

4. Perform Data Validation and ensure that the converted date remains in a valid

format.

**Date Format Scenario:**

**Use Case:**

* Your application expects date input in **dd-mm-yyyy** format.
* You are receiving data from a **US-based system** through an API, where the date format is **mm-dd-yyyy**.

**How to Handle This Using API Integration:**

**Step 1: Receive the Data via API**

* Accept incoming data through the API (usually in JSON or XML format).

**Json format**

{

"userName": "John Doe",

"birthDate": "04-17-2025" // US format: mm-dd-yyyy

}

**Step 2: Parse the Incoming Date Format**

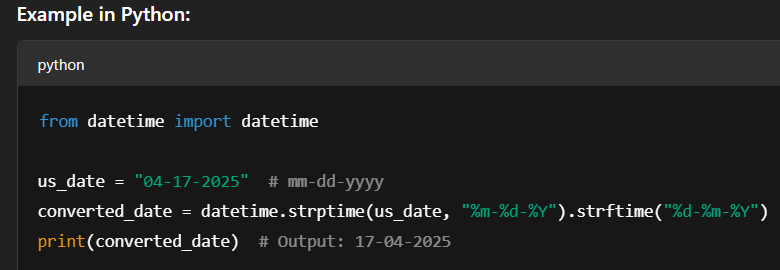
* Identify the format (in this case, mm-dd-yyyy).

**Step 3: Convert the Date Format**

* Use a date conversion function in your application code to **transform mm-dd-yyyy to dd-mm-yyyy**.

**Example in Python:**

Python



**Step 4: Store or Process Data in Your Format**

* Save the converted date (dd-mm-yyyy) in your system/database.

**Benefits of Handling it This Way:**

* Ensures data consistency in your system.
* Prevents errors in reporting, filtering, or data validation.
* Makes your application **robust and globally compatible**.