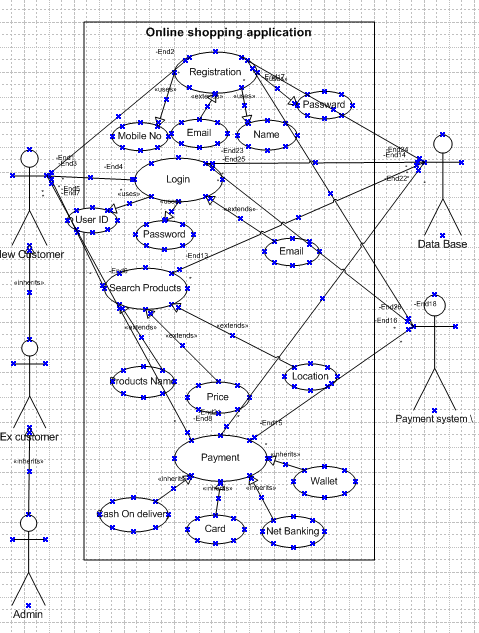
**Prep 3 part 1**

**Q 1. Draw a Use Case Diagram**

**Answer:**

****

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

**Answer:**

**Boundary Class**

|  |  |
| --- | --- |
| **Boundary Class – (use case) Actor speak to the system (Authentication information)**  1. Combination of 1 actor and a use case 1 boundary class  2. combination of 2 actor and a use case 2 boundary class  3. combination of 3 actor and a use case 3 boundary class  All the actors should be (Primary Actor)  Primary Actors who initiate the use cases and interact with the system | **Customer registration**   * 1 customer login * 2 Bank server login * 3 customer logouts * 4 bank server logouts |

**Controller Class**

|  |  |
| --- | --- |
| **Controller Class (Handal users (Primary Actor)**  **Input and process the data**   * User case will consider class system | **Registration controller**   * **Login controller** * **Payment controller** * **Credential Controller** * **Net Banking Controller** * **Logout Controller** |

**Entity Class**

|  |  |
| --- | --- |
| **Controller class (Handles Users (Primary actors) Input and process the data)**   * Use case will consider as controller class system   **1 Entity Class -All Actor**   * Each actor will be considered as an entity | **Registration controller**   * Login controller * Payment controller * Credential controller * Net banking controller * Logout controller   Customer  Bank server  Cash  Card  Net Banking |

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

**Answer:**

|  |  |
| --- | --- |
| **Application Layer** | Customer Registration  Customer Login  Bank server Login |
| **Business Logic Layer**  **(Primary Actors associated with boundary class)** | Customer  Bank server |
| **Data base layer (All the entity classes)- All actor** | * **Customer** * **Bank server** * **Cash** * **Card** * **Net banking** |

In this three-tier architecture, the application tier handles the user interface,

Business logic layer manages the business logic and coordinate between the tiers

Data layer tier handles data storage and retrieval

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

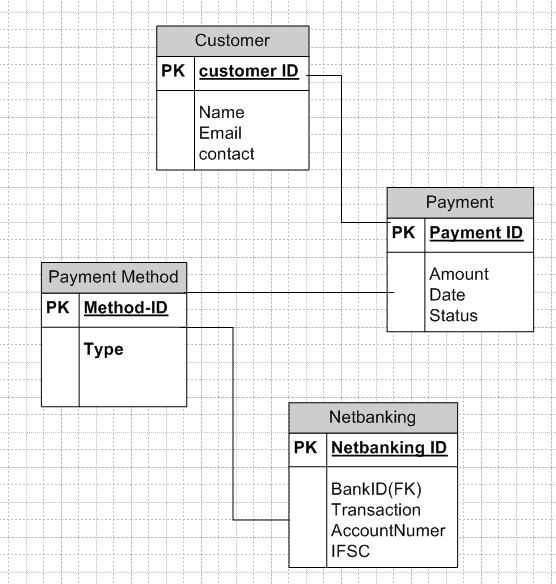
**Answer:**

|  |  |  |
| --- | --- | --- |
| **Entity** | **Attributes** | **Relationships** |
| **Customer** | Customer ID, name, email, phone | Has multiple Payments |
| **Payment** | Payment ID, amount, date, status | Associated with one PaymentMethod (NetbankingPayment in this case) |
| **Payment Method** | Method ID, type | Generalization with NetbankingPayment |
| **Net banking Payment** | bankID, bankName, transaction ID, accountNumber, IFSC | Subclass of PaymentMethod |
| **Bank** | bankID, bankName, branchCode, IFSC, supportContact | Associated with NetbankingPayment |

**Process Flow in the Domain Model**

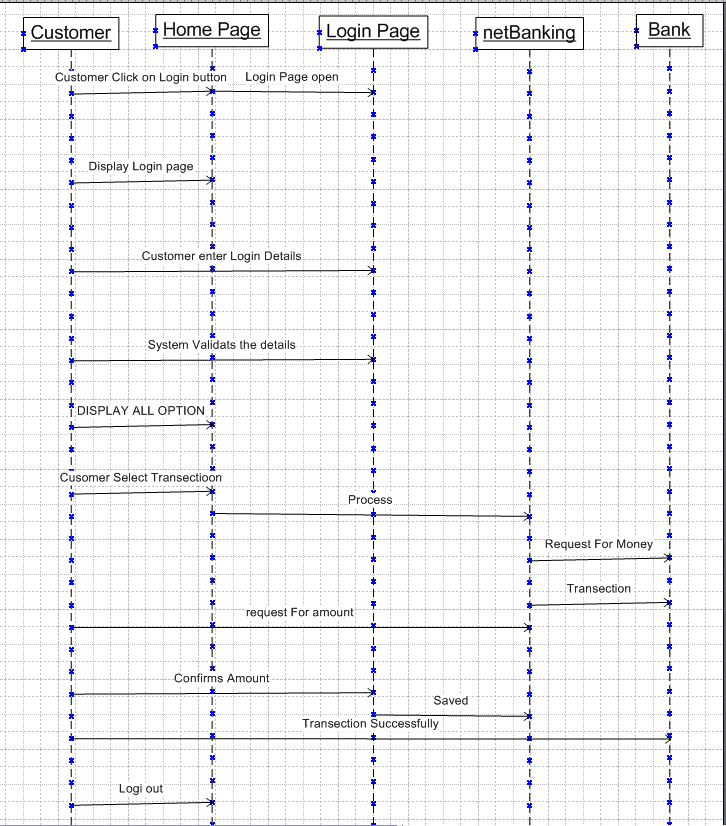
1. Customer initiates a Payment.
2. Payment selects NetbankingPayment as the PaymentMethod.
3. NetbankingPayment contains details like bankID, transactionID, accountNumber, and IFSC, which are necessary for the bank transaction.
4. Bank details provide branch-specific data and contact information if needed

This Domain Model captures the essential relationships and attributes involved in a net banking payment process, showing how a **customer** interacts with a **Payment** through **NetbankingPayment** with the involvement of a **Bank**.



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

**Answer:**



**Q 6. Explain conceptual Model for this Case**

**Anwer:**

1.Customer:

* Represents the person who initiates a payment.
* Attributes like CustomerID, Name, and Contact Information (e.g., email, phone) uniquely identify and describe the customer.
* Business Rule: Each customer may have multiple payment transactions in their history.

2. Payment:

* Represents a transaction initiated by the customer to complete a purchase.
* Attributes include PaymentID, Amount, Date, and Status.
* Linked to a Customer (one-to-many): Each payment is associated with one customer, but a customer can initiate multiple payments.
* Linked to a Payment Method (many-to-one): Each payment can use only one payment method.

3. Payment Method:

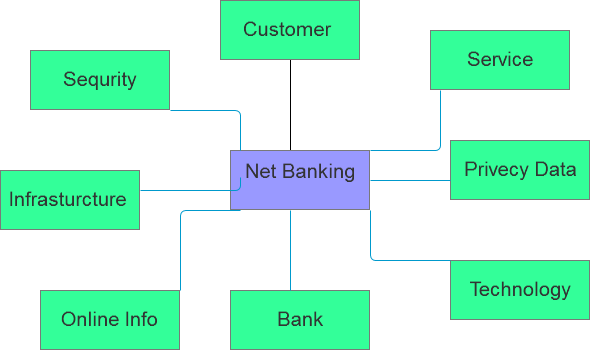
* Represents the various ways in which a customer can make a payment, including net banking.
* Attributes include MethodID and Type (e.g., Card, Wallet, Net Banking).
* NetbankingPayment is a specialized type of Payment Method with specific attributes and requirements for Net Banking.

4 . NetbankingPayment:

* A specialized form of Payment Method used when a customer chooses Net Banking as their payment method.
* Attributes include NetbankingID, TransactionID, AccountNumber, and IFSC.
* Connected to Bank (many-to-one): Each net banking transaction is processed by a single bank, but a bank may process multiple transactions.
* Business Rule: A Net Banking payment requires bank information to validate and process the transaction securely.

5 . Bank:

* Represents the financial institution that facilitates the net banking transaction.
* Attributes include BankID, BankName, BranchCode, IFSC, and SupportContact.
* Business Rule: A bank provides branch-specific information and support contacts to help with issues related to net banking transactions.



**Q 7. WhatisMVCarchitecture? Explain MVCrules to derive classes from use case diagram and guidelines to place classes in 3-tier architecture?**

**Answer:**

MVC (Model-View-Controller) architecture is a universal pattern of a structure in which an application is divided into three parts which are all dedicated to certain parts of the whole application. This pattern is normally used in software development to create organized and easy-to-maintain code.

* **Model:** It is worth stating that the Model stands as the data layer for the application. It is directly involved in managing the data as well as the control of the application’s logic and rules.
* **View:**The View is in the presentation tier. It plays a role of presenting the information given by the Model to the user and transferring the user commands to the Controller. The View is used to display the data to the user in a readable and manageable way using the interface created by the Controller.
* **Controller:**The Controller CE works in the middle between the Model and the View. It takes the input from the View, sometimes modifies it with the help of the Model, and sends it back to the View. the results back to the View.

Class Diagram are a key component of object-oriented programming, helping developers visualize the structure of their software and its interactions. In software development, it’s important to identify classes based on a use case to create an accurate and effective class diagram. In this article, we’ll explore the steps involved in identifying classes based on a use case and then developing a class diagram. We’ll also discuss how to refine a class diagram into an MVC detailed class diagram.

**Step 1: Identify Use Cases**

Before creating a class diagram, it’s essential to understand the problem domain and the use cases that the software will address. A use case represents a particular functionality that the system must provide to its users. Identifying use cases can be done through discussions with stakeholders, reviewing requirements documents, and brainstorming sessions.

**Step 2: Identify Classes**

Once you’ve identified the use cases, you can start identifying the classes that will be required to support those use cases. A class is a blueprint for creating objects, which contain properties and methods. Classes can represent real-world entities or abstract concepts.

For example, if the use case involves managing customer information, you might identify a Customer class, which would contain properties such as name, address, and email. If the use case involves processing payments, you might identify a Payment class, which would contain properties such as payment amount, payment method, and payment date.

**Step 3: Identify Relationships**

After identifying the classes, you need to determine how they are related to one another. Relationships between classes can be represented using association, aggregation, or composition.

Association represents a relationship between two classes, where one class uses the services of another class. Aggregation represents a relationship between two classes, where one class contains another class as a part, but the part can exist independently of the whole. Composition represents a relationship between two classes, where one class contains another class as a part, and the part cannot exist independently of the whole.

**Step 4: Develop Class Diagram**

Using the identified classes and relationships, you can create a class diagram. A class diagram is a visual representation of the classes and their relationships. It shows the properties and methods of each class and the relationships between the classes.

In a class diagram, each class is represented as a box, with the class name at the top of the box. The properties of the class are listed in the middle of the box, and the methods are listed at the bottom of the box. The relationships between classes are represented using lines, with different line types representing different types of relationships.

**Step 5: Refine the Class Diagram into an MVC Detailed Class Diagram**

After creating a class diagram, you can refine it into an MVC detailed class diagram. The (MVC) pattern is a software design pattern that separates an application into three interconnected components: the model, the view, and the controller.

**Q 8 . Explain BAcontributions in project (Waterfall Model– all Stages)**

**Answer:**

**Requirements Gathering and Analysis**:

* **Primary Role**: This is the most critical phase for the BA in the Waterfall Model. The BA works with stakeholders to understand, gather, and document requirements thoroughly, as changes to requirements in later stages are challenging.
* **Key Contributions:**
  + Conducts stakeholder interviews, workshops, and surveys to gather detailed functional and non-functional requirements.
  + Creates detailed **Requirements Documents**, such as the Business Requirements Document (BRD) and System Requirements Specification (SRS).
  + Defines scope, prioritizes requirements, and ensures alignment with business objectives.
  + Validates requirements with stakeholders through reviews and sign-offs to mitigate misunderstandings or scope changes later.

**System Design**:

* **Primary Role**: While the BA may not be directly involved in designing the system, they contribute by clarifying requirements and providing input on functional specifications.
* **Key Contributions**:
  + Acts as a bridge between stakeholders and the technical team to ensure that design decisions align with business goals.
  + Reviews design documents to ensure they meet specified requirements.
  + Assists in creating **Functional Specifications** and may collaborate with architects and developers to address questions on requirements.

**Development:**

* **Primary Role**: During this phase, the BA’s role is more supportive, but they remain available to clarify requirements and address any ambiguities that arise.
* **Key Contributions**:
  + Provides ongoing support to the development team, answering questions, and clarifying requirement details as needed.
  + Assists in managing **change requests** if minor adjustments to the requirements are necessary.
  + Ensures traceability by maintaining alignment between requirements and code implementation.

**Testing:**

* **Primary Role**: The BA’s role in the testing phase is to ensure that the final product aligns with documented requirements and meets business objectives.
* **Key Contributions**:
  + Develops or reviews **User Acceptance Testing (UAT)** plans and test cases to confirm that all functional and non-functional requirements are met.
  + Supports **testers** in understanding the business requirements, especially when there are complex use cases.
  + Coordinates UAT, facilitates testing sessions, and collects feedback from end users to identify any gaps.

**Deployment**:

* **Primary Role**: During deployment, the BA assists in ensuring a smooth transition from development to the live environment and helps with documentation and training.
* **Key Contributions**:
  + Prepares **training materials** and conducts training sessions for end users, if needed, to ensure they are comfortable with the system.
  + Documents **user manuals** or support guides to assist with knowledge transfer and ensure operational readiness.
  + Collaborates with stakeholders to verify and formally sign off on the system after deployment.

**Maintenance**:

* **Primary Role**: Post-deployment, the BA supports stakeholders by gathering feedback, tracking enhancements, and managing any change requests.
* **Key Contributions**:
  + Collects feedback to identify areas for improvement and may help prioritize changes or feature enhancements.
  + Updates requirements documentation based on enhancements or changes and coordinates with development teams to address post-launch issues.

**Q 9. What is conflict management? Explain using Thomas– Kilmann technique**

**Answer:**

Disagreements can arise in any organization. Having people in the workplace who know how to manage conflict is key to keeping turnover low, productivity high and customers satisfied.In this article, we explain what conflict management is and why it’s important, tips on choosing one of five common conflict management strategies (with examples) and the skills you need to successfully manage conflict in the workplace.

**1. Accommodating**

This conflict management style is when you give in to accommodate the other person's needs. You can use the accommodating style when the issue being argued is not as important to you as it is to the other person. This can be an appropriate style to use if you wish to keep the peace in the workplace or if you know that you are in the wrong. You can also use this type of conflict management style when you use empathy and put yourself in the person's situation.For example, a customer is demanding a refund even though they do not have a warranty. You know that you are in the right, but because you want to retain the customer and the product in question is not very expensive, you decide to give in.

**2. Avoiding**

This conflict management style involves simply avoiding the issue at hand. In this case, you would continually avoid the person or the issue. Use this style when you feel that you do not have time to discuss the issue or it seems trivial to you.You can also benefit from using the avoiding style if you are not sure how to respond or you have not formed a proper opinion yet. An example of this type of conflict management is when your colleagues are arguing for a more comfortable dress code, but you are too busy working on end-of-year financial submissions to get involved.

**3. Compromising**

This conflict management style gives you the opportunity to find a middle-ground solution for everyone involved. Use the compromising style when finding a solution is more important to everyone than having people win the argument. For example, you could distribute duties evenly so that you can finish a project before the deadline.

**4. Collaborating**

This conflict management style goes beyond finding the middle ground to finding a solution that will make everyone happy. Use the collaborating style when the relationship between those involved is more important than the conflict. One example is finding a solution between shareholders to keep the relationship strong.

**5. Competing**

This conflict management style involves sticking to your argument and rejecting that of others until you get your way. Use this style when a decision has to be made quickly, a long-term conflict needs to be resolved or you are standing up for your rights or the rights of others.For example, a customer is harassing one of the employees. In this case, you would reject the person's argument and insensitive comments.

High

A Compecting Collaborating

S

S Compromising

E

R

T

I Avoiding Accommodating

V

Low

Cooperativeness High

**Q 10. List down the reasons for project failure**

**Answer:**

**1 . Poorly Defined Requirements**:

* Lack of clarity or incomplete requirements can lead to misunderstandings, scope creep, and rework, resulting in time and budget overruns.

2 **Unclear Objectives and Scope**:

* Without clear project goals and scope definition, team members may lack direction, and stakeholders may have misaligned expectations, leading to project drift or failure to meet the intended outcomes.

3 **Inadequate Planning and Estimation**:

* Insufficient planning, including inaccurate time and cost estimates, can create delays, financial strain, and resource shortfalls during the project lifecycle.

4 **Weak Leadership and Governance**:

* Lack of strong leadership, decision-making, or governance can lead to poor project oversight, lack of accountability, and inability to handle challenges effectively.

5 **Scope Creep**:

* Uncontrolled changes or additions to the project scope, often without corresponding adjustments in time or budget, can lead to missed deadlines and budget overruns.

6 **Poor Communication**:

* Ineffective communication among stakeholders, team members, and sponsors can lead to misunderstandings, delays, and misaligned expectations.

7 **Inadequate Risk Management**:

* Failure to identify, assess, and mitigate risks can result in unexpected issues that derail the project.

8 **Lack of Stakeholder Engagement**:

* When key stakeholders are not involved or invested in the project, there may be insufficient buy-in, support, and input, affecting project alignment and success.

**Q 11. List the Challenges faced in projects for BA**

**Answer:**

1Ambiguous or Unclear Requirements:

* BAs frequently deal with vague, incomplete, or constantly changing requirements, making it challenging to define a clear project scope and objectives.

2 Stakeholder Conflicts and Misalignment:

* Conflicting stakeholder interests or lack of consensus on project requirements can lead to misalignment and impact decision-making.

3 Scope Creep:

* Continuous additions or changes to project requirements without corresponding adjustments in time, resources, or budget can lead to scope creep, making it difficult to manage project boundaries.

4 Communication Gaps:

* Miscommunication or lack of clear communication between technical and non-technical stakeholders can result in misunderstandings, project delays, and misaligned expectations.

5 Limited Access to Stakeholders:

* Delays in stakeholder feedback or restricted access to key stakeholders can hinder requirement gathering, clarification, and timely decision-making.

6 Resource Constraints:

* BAs may face limitations in terms of resources (time, budget, or personnel), impacting their ability to perform thorough analysis, documentation, or validation.

**Q 12. Write about Document Naming Standards**

**Answer:**

[ProjectID] [Document Type]V[v]D[y].extension

Ex. [PQ777FRDV1.docx] or [PQ666FRD1.1docx]

**1 Project or Department Identifier:**

* Use a unique identifier or abbreviation representing the project or department to distinguish documents and make them quickly identifiable. For example, a project named "Online Agriculture Products Store" might use "OAPS" as its identifier.

**2 Document Type:**

* Include the type of document, such as "REQ" for Requirements, "SRS" for Software Requirements Specification, "BRD" for Business Requirements Document, or "UAT" for User Acceptance Testing. This makes it clear what kind of information is in the document.

**3 Version Number:**

* Adding a version number (e.g., v1.0, v2.1) is essential to track updates and revisions. Use a consistent versioning scheme, such as "major.minor" (e.g., 1.0 for initial release and 1.1 for minor updates).

**4 Date of Creation or Last Revision:**

* Adding the date in a standardized format (e.g., YYYYMMDD) helps users understand when the document was created or last modified. This is useful for tracking document currency.

**5 Author or Team Initials:**

* Including the initials of the author or team responsible for creating the document can make it easier to follow up with the right person or group. This is optional but helpful in larger organizations.

**6 Description or Keywords:**

* Add a brief, meaningful description to indicate the document's specific purpose, such as "System\_Design" or "Project\_Overview." This can help differentiate similar documents.

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

**Answer:**

Never say NO to the clint

There is no word called as "BY DEFAULT"

Never imagine anything in terms of GUI

Question everything in the world

Go to the client with plain mind that is with no assumptions

Listen to the client very carefully and after he is done, then ask questions

Don't interrupt the client.

Never try to give solutions to the client right away.

Try to concentrate only on important and required things.

Be like a lotus in mud- if a client comes with a fancy requirement, then talk to the project manager first.

Requirement hurried-project buried.

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

**Answer:**

|  |  |
| --- | --- |
| **Packages** | **Sub-Systems** |
| Logical grouping of related classes or components | Cohesive, independent functional unit within a larger system |
| Organize and manage code within a system | Divide complex systems into manageable, self-contained units |
| Lower (focused on classes and modules) | Higher (focused on system functionality) |
| Limited to components within the same system or module | May encapsulate multiple packages and span across systems |
| Fine-grained dependencies within the same system | Loose coupling, often with clear interfaces for interaction |
| Represented as folders containing classes or interfaces | Represented as independent modules or components with boundaries and interfaces |
| CustomerManagement package in a banking app | OrderProcessing sub-system in an e-commerce app |

**Q 15. What is camel-casing and explain where it will be used**

**Answer:**

1 **Lower Camel Case** (also called "camelCase"):

* The first letter of the first word is lowercase, and the first letter of each subsequent word is uppercase.
* **Example**: customerDetails, calculateTotal, isActive.

2 **Upper Camel Case** (also called "PascalCase"):

* The first letter of each word is capitalized, including the first word.
* **Example**: CustomerDetails, CalculateTotal, IsActive.

**Where Camel-Casing is Used**

1. **Variable and Property Names** (typically lower camel case)

* Lower camel case is commonly used for naming variables and properties in programming. This allows developers to quickly identify variables in code.
* **Example**: employeeName, totalAmount, isAvailable.

1. **Method and Function Names** (typically lower camel case)

* Camel case is used to name methods and functions, which makes their purpose clear and differentiates them from classes.
* **Example**: getEmployeeDetails(), processOrder(), calculateInterest().

**Q 16. Illustrate Development server and what are the accesses does business analyst has?**

**Answer:**

**Development Server Architecture**

1. Application Layer: Hosts the application code where developers implement new features and perform testing. This may include front-end (UI/UX) and back-end services (business logic, APIs).
2. Database Layer: Contains a test or dummy database, allowing developers to test data interactions without affecting real data.
3. Middleware & API Services: These services enable communication between different components, simulating integrations with external systems, if applicable.
4. Version Control Integration: Integrates with a version control system (e.g., Git) where code is pushed to the development branch and can be reviewed and tested by others.
5. Testing Tools: Often configured with tools for unit tests, integration tests, and debugging.

**Business Analyst (BA) Access to the Development Server**

The level of access a Business Analyst has to the development server depends on organizational policies, project needs, and the BA’s role. Generally, here’s what a BA can access or perform on the development server:

1. Read-Only Access to Application Interface:
   * BAs may access the development server’s user interface to view the application in its current state, observe new features or changes, and verify requirements alignment.
   * This access allows BAs to understand how the application is evolving and give feedback to developers.
2. Test Data Access (Limited):
   * BAs might be granted limited access to dummy or test data within the development database to validate workflows or test specific requirements without affecting actual data.

**Q 17. What is Data Mapping**

**Answer:**

**Uses of Data Mapping**

1. Data Migration:
   * During migration, data mapping links data fields in the source system to those in the target system to ensure a smooth transfer without data loss or corruption.
2. Data Integration:
   * When combining data from multiple sources, data mapping helps match fields, allowing data from different systems to be merged accurately for analysis or reporting.
3. ETL Processes:
   * In ETL workflows, data mapping defines how data should be extracted, transformed (e.g., format changes, calculations), and loaded into a destination system.
4. Data Transformation:
   * Data mapping applies transformations (e.g., changing date formats, unit conversions) to make data compatible with the target system’s standards.
5. Data Synchronization:
   * For real-time or scheduled synchronization, data mapping keeps fields in sync across systems, ensuring data remains up-to-date across all platforms**.**

**Q 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**

**Answer:**

An API (Application Programming Interface) is a set of rules, protocols, and tools that allows different software applications to communicate and share data or functionalities with each other. APIs enable one system to send or receive data from another system, allowing for integration without directly accessing each other's code or database

**API Integration in the Case of the Application with Different Date Formats**

In your case, the application uses the date format dd-mm-yyyy but is receiving data from a US-based application that provides dates in the mm-dd-yyyy format. API integration can be used here to ensure seamless data exchange and transformation between the two applications.

Steps for API Integration with Date Format Conversion

1. Identify the API Endpoint and Data Structure:
   * Define the API endpoint for receiving the data from the US application.
   * Review the data structure and format for any date fields being sent to ensure they are identified as mm-dd-yyyy.
2. Create an API Wrapper or Middleware:
   * Implement an API wrapper or middleware layer that will handle the incoming data before it’s processed by your application.
   * This layer will check the format of each date field and convert it from mm-dd-yyyy to dd-mm-yyyy.
3. Transform Date Format:
   * In the middleware, apply date transformation logic to reformat dates before inserting them into the application’s database or displaying them on the frontend.
   * This can be done using a programming language function, like Python’s datetime.strptime() and strftime(), or similar methods in JavaScript, Java, etc.
4. Error Handling and Validation:
   * Implement error handling in the API wrapper to check for invalid date entries and ensure all dates adhere to the dd-mm-yyyy format.
   * Include validation rules for dates to ensure they are correctly processed if they’re in the wrong format or missing.
5. Testing and Documentation:
   * Test the API integration to verify that dates are being accurately converted and processed.
   * Document the transformation rules, API endpoints, and any error-handling procedures for clarity and future maintenance.