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

Q1 . Draw a Use Case Diagram- 4 Marks

Answer: A **use case diagram** is a type of **Unified Modeling Language (UML)** diagram.

It is used in software and system design to show how different types of users (called **actors**) interact with a system to achieve their goals.

It visually represents the **functional requirements** of a system, showing the system's use cases, actors, and the relationships among them.

Component of use case:

1.Actor: It can be primary or secondary actor who interact with the system.

2.System boundry: Defines the scope of the system and what's included within it, usually shown as a rectangle that contains all use cases.

3.Use case: It is the function or activity perform by the actor.

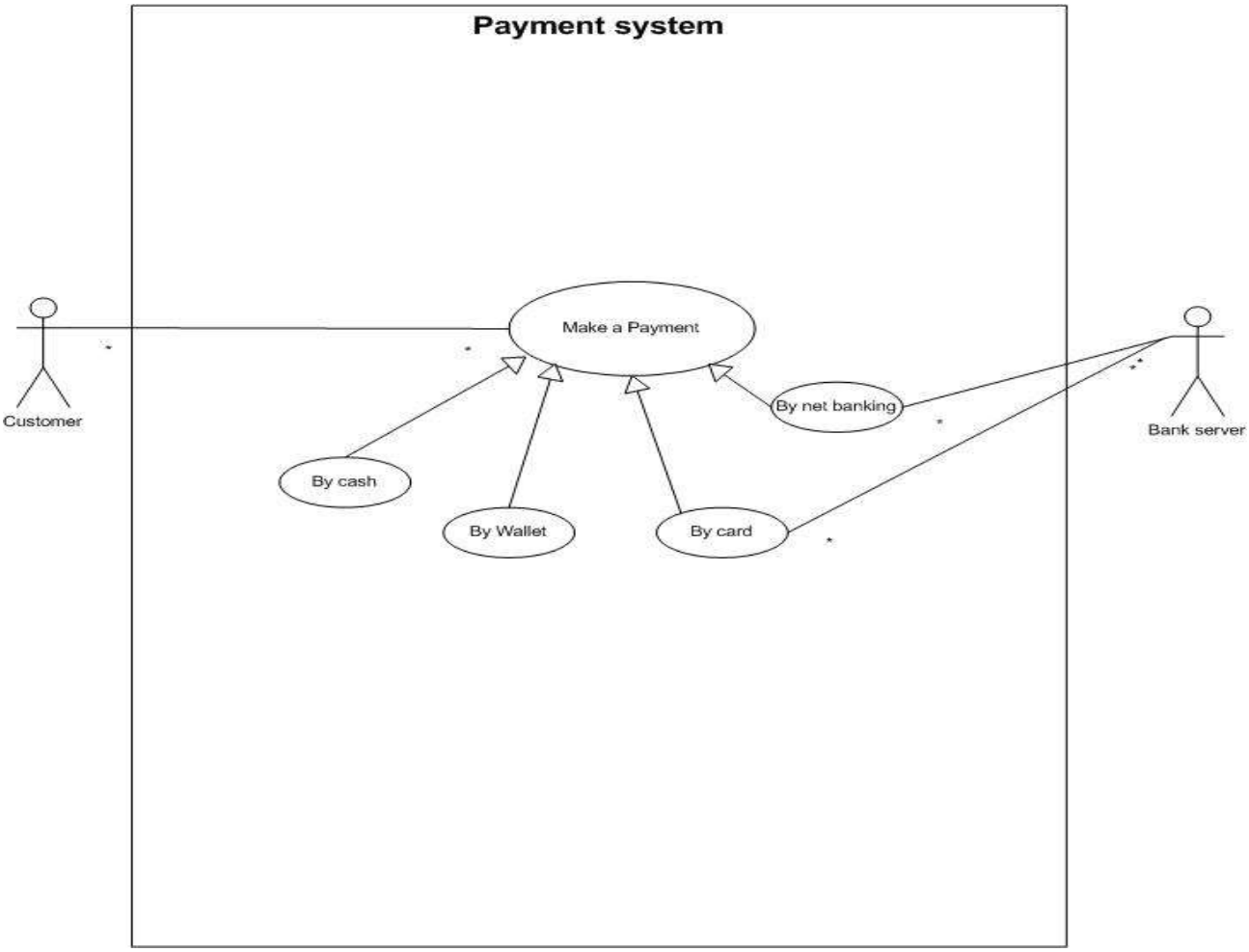
4.Relationship:

1. Association: It is relationship between actor and use case.

2.Include: Shows that one use case is always included with another usecase.

3.Extend: Indicates an optional or conditional relationship.

4.**Generalization**: Generalization is used when two or more actors share common characteristics, and you want to show that one actor is a specialized version of another.



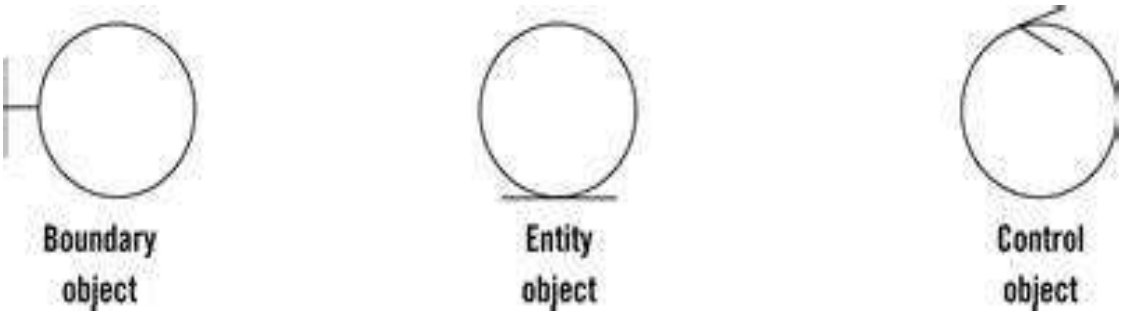
Q2. Derive Boundary Classes, Controller classes, Entity Classes.- 4 Marks

Answer:

1. **Boundary Classes**: Boundary classes manage the interactions between the system and external actors like customers, banks, or wallet providers.
2. **Controller Classes**: Controller classes acts as intermediaries between boundary and entity class.
3. **Entity Classes**: Entity classes represent the core data structures and business logic of the application.

MVC rules to derive classes from Use case diagram :

1. Combination of one actor and a use case result in one boundary class.
2. Combination of two actor a use case results in two boundary classes.
3. Combination of three actors and a use case results in three boundary classes and so on.
4. Use case will result in a controller class.
5. Each actor will result in one entity class.



Boundary Classes:

1. **CustomerPaymentBoundary**
    - Responsibilities:
        - Collect payment details (amount, payment method) from the customer.
        - Send the collected details to the controller.
        - Display payment success or failure to the customer.
2. **BankPaymentBoundary**
    - Responsibilities:
        - Transmit payment authorization requests to the bank server for Card or Net Banking payments.
        - Receive authorization status and send it to the controller.

Controller Class:

1. **PaymentController**
    - Responsibilities:
        - Accept payment details from the boundary.
        - Identify the payment method (Card, Wallet, Cash, or Net Banking).
        - Process the payment by interacting with the relevant payment entity.
        - Notify the boundary of the result.

Entity Classes:

1. **Customer**
    - Attributes: name, email, paymentDetails.
    - Responsibilities:
        - Represent the customer's information and preferences.
2. **BankServer**
    - Attributes: bankName, authorizationStatus.
    - Responsibilities:
        - Handle the authorization process for card or net banking payments.

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

Answer

1. Application Layer (Boundary Layer):

This layer is responsible for interacting with the user and external systems. It contains the **Boundary Classes**, which act as interfaces between the system and the external actors.
Classes in this layer:

CustomerPaymentBoundary

- Handles input from the customer (e.g., payment details) and displays the payment status.

BankPaymentBoundary

- Acts as an interface to send and receive payment authorization requests from the bank server.

2. Business Logic Layer (Controller Layer):
This layer contains the **Controller Classes**, which handle the application's logic, workflows, and processing. This layer manages the core functionality of the application and it acts as the intermediary between the Application Layer and the Data Layer.
Classes in this layer:

PaymentController

- Processes the Make a Payment use case.
- Coordinates the flow between the boundary classes and entity classes.
- Determines the payment method (Card, Wallet, Cash, or Net Banking) and delegates processing to the appropriate logic.

3. Data Layer (Entity Layer):
This layer contains the **Entity Classes**, which represent the application's core data structures and are responsible for data storage, retrieval, and management.
Classes in this layer:

Customer

- Represents customer details and preferences, possibly stored in a database.

BankServer

- Represents the bank server actor and handles communication with external banking systems for payment authorization.

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

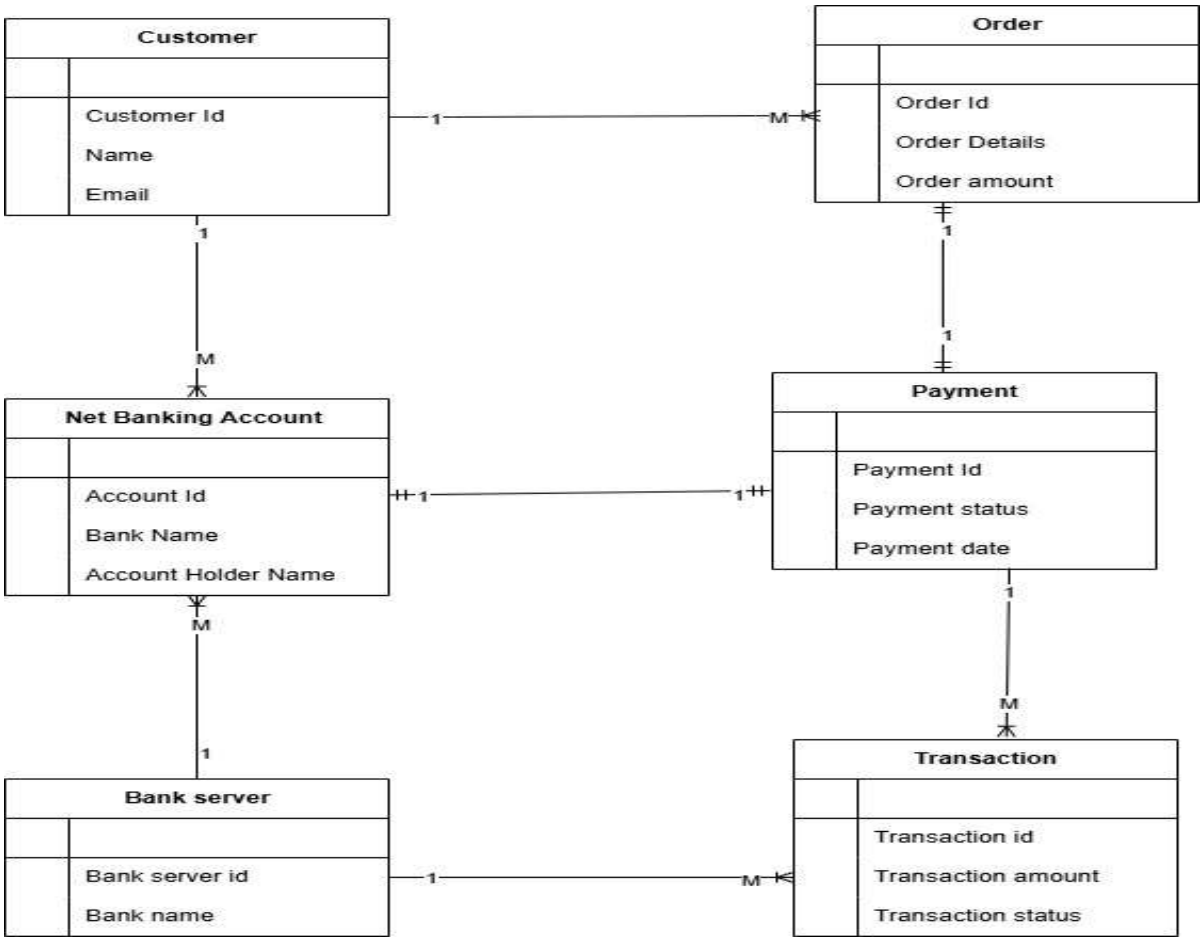
Answer:

A **Domain Model** is a conceptual representation of the objects (entities), their attributes, behaviors, and relationships in a specific problem domain.

It acts as a bridge between real-world concepts and software design, helping to define how the system and its components interact at a high level.

Key Component of a Domain Model:

1. **Entities:**
    - Represents real-world objects or concepts (e.g., Customer, Payment).
    - Typically corresponds to tables in a database or classes in object-oriented programming.
2. **Attributes:**
    - Properties or characteristics of an entity (e.g., Name, PaymentID, Amount).
3. **Relationships:**
    - Defines how entities are related (e.g., One-to-Many, Many-to-One).
    - Includes multiplicity (e.g., 1, *, 0..1) to specify cardinality.



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

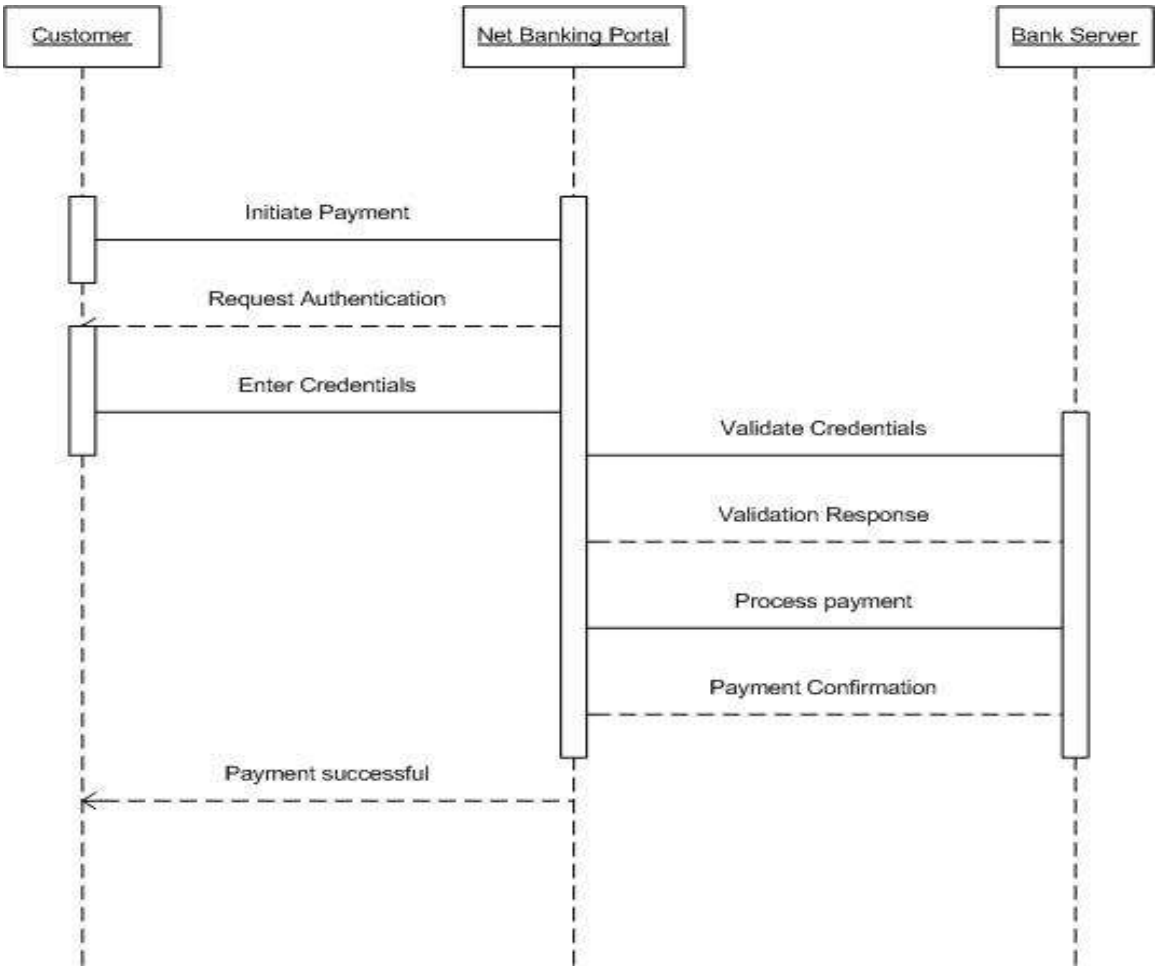
Answer: A **sequence diagram** is a type of **UML (Unified Modeling Language)** diagram that models the interaction between objects or components in a specific order over time. It visually represents the flow of messages, events, and actions between different participants (actors or objects) involved in a system.

Sequence diagrams are especially useful for understanding the behavior of a system or a process, such as the flow of transactions in an application.

Key Components of a Sequence Diagram:

1. **Actors and Objects:**
   - Represent external users or systems (actors) and internal components (objects) involved in the interaction.
2. **Lifelines:**
   - Depict the lifespan of an actor or object during the interaction.

3. **Messages:**
    - Represent the flow of communication between actors and objects in the system (e.g., method calls, responses).
4. **Activation Bars:**
    - Show the duration of an object performing a task or being active.
5. **Time Sequence:**
    - The top-to-bottom layout shows events happening in chronological order.



Q6. Explain Conceptual Model for this Case- 4 Marks

Answer: A **Conceptual Model** is a high-level representation of a system or process that focuses on **key entities**, **their relationships**, and **interactions**.

It is designed to provide a simplified and abstract understanding of how the system works without diving into implementation details.

Conceptual Model for Customer Making Payment through Net Banking

Key Features of the Conceptual Model

1. **Entities**: Represent real-world objects or concepts involved in the system.
2. **Attributes**: Define the characteristics or properties of each entity.
3. **Relationships**: Describe how entities interact with one another, including cardinality.
4. **Abstraction**: It is technology-independent and serves as the foundation for the system's logical design.

Entities in the Case

1. **Customer**:
    - Attributes: Customer ID, Name, Email
    - Represents the user making the payment.
2. **Order**:
    - Attributes: Order ID, Order Details, Order Amount
    - Represents the product/service purchased by the customer.
3. **Payment**:
    - Attributes: Payment ID, Payment Status, Payment Date
    - Represents the payment process for an order.
4. **Net Banking Account**:
    - Attributes: Account ID, Bank Name, Account Holder Name
    - Represents the customer's bank account used for payment.
5. **Bank Server**:
    - Attributes: Bank Server ID, Bank Name
    - Represents the bank's system that processes the transaction.
6. **Transaction**:
    - Attributes: Transaction ID, Transaction Amount, Transaction Status
    - Represents the actual monetary transaction processed between the customer’s bank and the system.

Relationships in the Conceptual Model

1. **Customer ↔ Order**:
    - A customer can place multiple orders (1:M), but each order belongs to one customer.
2. **Order ↔ Payment**:
    - Each order corresponds to one payment (1:1).
3. **Payment ↔ Net Banking Account**:

    - A payment is made through one net banking account, and each account can process multiple payments (1:M).
4. **Net Banking Account ↔ Bank Server**:
    - A net banking account is associated with one bank server, but a bank server handles multiple accounts (1:M).
5. **Payment ↔ Transaction**:
    - A payment generates multiple transactions (e.g., for service charges), but each transaction is linked to one payment (1:M).

How the Conceptual Model Works

1. **Customer Initiates Payment**: The customer selects a product/service (Order) and opts for payment via Net Banking.
2. **Order Validation**: The system verifies the order details and amount before proceeding to the payment process.
3. **Payment Processing**: The payment information is sent to the customer's Net Banking Account.
4. **Bank Server Interaction**: The bank server validates the account details and processes the transaction.
5. **Transaction Record**: A transaction record is created for every payment attempt, whether successful or failed.
6. **Payment Confirmation**: Once the payment is processed successfully, the order is confirmed, and the status of the payment and transaction is updated.

Q7. What is MVC architecture? Explain MVC rules to derive classes from use case diagram and guidelines to place classes in 3-tier architecture- 8 Marks

Answer: MVC (Model-View-Controller) is a **design pattern** used to organize and structure software applications by separating them into three interconnected components:

- **Model**: Manages the application's data and business logic.
- **View**: Handles the user interface and presentation of data.
- **Controller**: Acts as an intermediary, processing user input and managing interactions between the Model and View.

Key Components of MVC Architecture

1. **Model**:
    - Represents the **data structure** and encapsulates the logic for data processing.
    - Handles database interaction, calculations, and state management.
    - Example: Classes like Order, Product, and Customer.
2. **View**:
    - Displays data to the user and handles the presentation layer.
    - Does not contain business logic.

- Example: Web pages, forms, or graphical components like ProductView or OrderSummary.

3. **Controller**:
   - Processes user input and updates the Model or View accordingly.
   - Contains business logic to coordinate data flow between Model and View.
   - Example: OrderController processes customer orders and updates the system.

MVC rules to derive classes from Use case diagram :

6. Combination of one actor and a use case result in one boundary class.
7. Combination of two actor a use case results in two boundary classes.
8. Combination of three actors and a use case results in three boundary classes and so on.
9. Use case will result in a controller class.
10. Each actor will result in one entity class.

Guidelines to Place 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 0classes,place them in Business logic layer else place them in Application layer.

In **3-Tier Architecture**, classes are placed into three layers to organize application functionality:

1. **Presentation Layer (UI)**
   - Contains **View classes** that deal with the user interface and capture user inputs.
   - Only communicates with the Business Logic Layer.
   - Example: LoginView, ProductView.
   - Tools: HTML, CSS, JavaScript (web), or GUI frameworks.
2. **Business Logic Layer (BLL)**
   - Contains **Controller classes** and **business logic** that process data and coordinate between the Presentation and Data layers.
   - Implements workflows and rules derived from the use cases.
   - Example: OrderController, PaymentProcessor.
   - Tools: Programming languages like Java, Python, C#, etc.
3. **Data Layer**
   - Contains **Model classes** that manage the data structure and interact with the database.
   - Handles CRUD operations (Create, Read, Update, Delete).
   - Example: Order, Customer, Product.
   - Tools: SQL, NoSQL, or ORM frameworks like Hibernate or Django ORM.

Q8.Explain BA contributions in project (Waterfall Model– all Stages)– 8 Marks

Answer: The **Waterfall Model** is a traditional and sequential software development methodology, where each phase must be completed before the next one begins. It's often compared to a waterfall because progress flows downward through clearly defined stages. It's a linear and rigid approach, making it easier to manage in projects with well-understood requirements that are unlikely to change.

Basic Phases of the Waterfall Model:

1. Requirements Gathering

- **BA's Contribution**:
  - Meet with stakeholders to gather requirements.
  - Document business needs and functional requirements.
  - Prioritize requirements based on importance.
- **Key Artifacts**:
  - Business Requirements Document (BRD).
  - Use Cases/Use Case Diagrams.
  - Requirement Traceability Matrix (RTM).

2. Requirement Analysis

- **BA's Contribution**:
  - Analyze the gathered requirements for clarity, feasibility, and alignment with business goals.
  - Break down high-level requirements into detailed and actionable tasks.
  - Identify any conflicts, gaps, or dependencies in the requirements.
  - Collaborate with stakeholders to validate and finalize the requirements.
  - Ensure that requirements are testable and measurable.
- **Key Artifacts**:
  - Detailed Requirement Specifications (DRS)
  - Requirement Feasibility Report
  - Refined RTM
  - Stakeholder Approval Documents

3. **System Design**

- **BA's Contribution**:
  - Work with design teams to ensure that requirements are understood and reflected in the design.
  - Review system architecture and design documents.
  - Validate design against the documented requirements.
- **Key Artifacts**:
  - System Design Document (SDD).
  - UI/UX wireframes or mockups.

- Data Flow Diagrams (DFD).

4. Development

- **BA's Contribution**:
  - Support the development team by clarifying requirements.
  - Address questions and provide additional details if needed.
  - Ensure alignment with requirements during the build phase.
- **Key Artifacts**:
  - Clarification documents for development teams.
  - Updated RTM as development progresses.

5. Testing

- **BA's Contribution**:
  - Work with QA to ensure test cases are created based on requirements.
  - Conduct User Acceptance Testing (UAT) to verify that the solution meets business needs.
  - Identify and log any defects or gaps.
- **Key Artifacts**:
  - Test Case Documents.
  - UAT Plan and Results.
  - Defect Logs.

6. Deployment

- **BA's Contribution**:
  - Support training and documentation for end-users.
  - Ensure that the solution is fully aligned with business needs.
  - Collect feedback after deployment and address any issues.
- **Key Artifacts**:
  - User Manuals.
  - Training Documents.
  - Feedback Reports.

7. Maintenance

- **BA's Contribution**:
  - Assist with change requests and system updates.
  - Analyze the impact of changes and ensure they align with business goals.
- **Key Artifacts**:
  - Change Requests.
  - Updated RTM for any new or revised requirements..

Q9. What is conflict management? Explain using Thomas– Kilmann technique– 6Marks

Answer: Conflict management is the process of identifying, addressing, and resolving disagreements or disputes effectively and constructively. It aims to reduce the negative impacts of conflict, such as stress or reduced productivity, while maximizing its potential benefits, like improved understanding or creative problem-solving.

In any organization or group, conflicts may arise due to differences in goals, values, opinions, or needs. Effective conflict management ensures that disagreements are resolved in a way that fosters cooperation, trust, and mutual respect.

The **Thomas–Kilmann Conflict Mode Instrument (TKI)** is a model designed to understand and handle conflicts based on individual behavior in conflict situations. It categorizes conflict-handling strategies based on two primary dimensions:

1. **Assertiveness**: The degree to which a person tries to satisfy their own concerns.
2. **Cooperativeness**: The degree to which a person tries to satisfy the concerns of others.

Using these dimensions, the model identifies **five conflict management styles**:

1. Competing (High Assertiveness, Low Cooperativeness)

- **Description**:
    - This style prioritizes one's own needs or goals over those of others. It often involves using authority, power, or assertiveness to resolve the conflict in one's favor.
- A manager insists on implementing a new company policy despite opposition because it aligns with strategic objectives.

2. Accommodating (Low Assertiveness, High Cooperativeness)

- **Description**:
    - This style focuses on satisfying the needs of others, often at one's own expense. It prioritizes harmony and relationships over personal goals.
- **Example**:
  A team member agrees to a colleague's approach to avoid tension, even though they had a different idea.

3. Avoiding (Low Assertiveness, Low Cooperativeness)

- **Description**:
    - This style involves sidestepping or withdrawing from the conflict without trying to resolve it.

- It is often used when the conflict is trivial or when emotions are high, and more time is needed.

- **Example**:
  A project leader decides to delay a discussion about a minor disagreement until after a major deadline.

4. Collaborating (High Assertiveness, High Cooperativeness)

- **Description**:
  - This style seeks a win-win solution by addressing the concerns of all parties. It involves open communication, cooperation, and creative problem-solving.
- **Example**:
  Two departments work together to create a shared budget plan that benefits both their goals.

5. Compromising (Moderate Assertiveness, Moderate Cooperativeness)

- **Description**:
  - This style involves finding a middle ground where both parties make concessions. It balances assertiveness and cooperativeness but may not fully satisfy either side.
- **Example**:
  Two colleagues agree to split a resource 50/50 after failing to agree on a single approach.

Q10. List down the reasons for project failure– 6 Marks

Answer: Project failure can occur due to a variety of factors, ranging from poor planning and communication to technical issues and lack of stakeholder involvement. Below are key reasons why projects fail:

1. **Poor Project Planning**
2. **Lack of Stakeholder Engagement**
3. **Inadequate Resource Allocation**
4. **Poor Communication**
5. **Scope Creep**
6. **Unrealistic Expectations or Goals**
7. **External Factors:**

1. Poor Project Planning

- **Description**: Inadequate planning is one of the most common causes of project failure. Without a clear roadmap, objectives, and strategy, projects are prone to delays, scope creep, and resource mismanagement.

- **Impact**:
    - Missed deadlines.
    - Increased costs.
    - Poor-quality output.

2. Lack of Stakeholder Engagement

- **Description**: When stakeholders (e.g., clients, end-users, or team members) are not actively involved in the project, it leads to a disconnect between what is expected and what is delivered. This can cause dissatisfaction, scope misalignment, and even abandonment of the project.
- **Impact**:
    - Misalignment between deliverables and stakeholder expectations.
    - Project scope becomes unclear.
    - Decreased morale and trust in the team.

3. Inadequate Resource Allocation

- **Description**: Insufficient or misallocated resources (e.g., time, money, personnel) can severely hinder a project's progress. Without the right resources at the right time, critical tasks may be delayed or incomplete.
- **Impact**:
    - Delayed project timelines.
    - Increased costs.
    - Overworked team members.

4. Poor Communication

- **Description**: Effective communication is critical for the success of any project. Lack of communication or unclear instructions can lead to misunderstandings, missed deadlines, and mistakes.
- **Impact**:
    - Confusion about roles and responsibilities.
    - Misunderstood objectives.
    - Reduced collaboration and teamwork.

5. Scope Creep

- **Description**: Scope creep occurs when the scope of the project expands without proper approval, usually due to undefined or poorly defined project boundaries. This can lead to overwork, missed deadlines, and failure to meet original objectives.
- **Impact**:
    - Delays and increased costs.
    - Decreased focus on core deliverables.

- Overburdened project team.

6. Unrealistic Expectations or Goals

- **Description**: Setting unrealistic or overly ambitious goals can lead to failure if the team does not have the necessary tools, skills, or time to meet them. This can cause burnout, missed milestones, and project abandonment.
- **Impact**:
  - Unrealistic deadlines or goals.
  - Decreased team morale and motivation.
  - Increased pressure and stress.

7. External Factors:

1. **Changing Market Conditions**: Sudden shifts in market trends, regulations, or competition affecting the project's relevance or viability.
2. **Budget Cuts or Funding Issues**: Insufficient funding or abrupt budget reductions that stall progress.
3. **Technological Challenges**: Adoption of immature or incompatible technology that hinders project implementation.
4. **Dependence on Third Parties**: Delays or failures from vendors, suppliers, or other external parties critical to the project.
5. **Legal or Regulatory Issues**: Non-compliance with laws, regulations, or contractual obligations leading to penalties or project halt.

Q11.List the Challenges faced in projects for BA– 6 Marks

Answer: Business Analysts (BAs) play a key role in ensuring that a project aligns with the business goals and requirements. However, they face a variety of challenges during the course of a project. Below are the major challenges faced by BAs:

1. **Unclear or Changing Requirements**
2. **Managing Stakeholder Expectations**
3. **Communication Barriers**
4. **Scope Creep**
5. **Balancing Technical and Business Needs**
6. **Time and Resource Constraints**
7. **Additional Challenges:**

1. Unclear or Changing Requirements

- **Description**: One of the most significant challenges for BAs is dealing with vague, incomplete, or constantly changing requirements. If the stakeholders have not clearly defined what they want or if the requirements evolve throughout the project, it

becomes difficult for BAs to document and communicate these requirements effectively.

- **Impact**:
  - Misalignment between business needs and deliverables.
  - Increased rework and scope creep.
  - Confusion among team members and stakeholders.

2. Managing Stakeholder Expectations

- **Description**: BAs often deal with multiple stakeholders with varying interests, priorities, and expectations. Aligning these expectations and ensuring that all stakeholders are on the same page can be difficult.
- **Impact**:
  - Conflicting priorities and demands.
  - Delays in decision-making.
  - Dissatisfaction and mistrust from stakeholders.

3. Communication Barriers

- **Description**: Effective communication is crucial for a BA, but it can be hindered by language differences, cultural barriers, technical jargon, or misunderstandings. Miscommunication between stakeholders, developers, or even within the BA team can lead to project delays and failures.
- **Impact**:
  - Misunderstanding of requirements.
  - Incorrect documentation and specifications.
  - Inefficient collaboration between teams.

4. Scope Creep

- **Description**: Scope creep occurs when the project's objectives expand beyond the original scope without proper control. BAs are responsible for managing and controlling scope and preventing unnecessary changes that could lead to delays or additional costs.
- **Impact**:
  - Increased project costs and timeline.
  - Disrupted focus on critical deliverables.
  - Resource strain and overburdened teams.

5. Balancing Technical and Business Needs

- **Description**: BAs must bridge the gap between business requirements and technical solutions. Sometimes, there may be a misalignment between what the business wants and what is technically feasible, or the business may not fully understand the technical limitations.

- **Impact**:
    - Delays or roadblocks due to unrealistic business expectations.
    - Technical teams may be unable to deliver on business requests.
    - Tension between business and IT departments.

6. Time and Resource Constraints

- **Description**: BAs are often faced with tight timelines and limited resources, which can make it difficult to conduct thorough analysis, gather detailed requirements, and perform proper validation. This constraint can lead to rushed decisions and incomplete documentation.
- **Impact**:
    - Low-quality deliverables.
    - Inadequate time for stakeholder engagement and requirement validation.
    - Increased project risk and errors.

Additional Challenges:

- **Lack of Stakeholder Commitment**: If stakeholders do not participate or engage actively, it becomes difficult for BAs to gather the necessary information or get approvals.
- **Data and System Integration Issues**: Many projects require integration with existing systems or data sources, and managing these integrations can be complex.
- **Managing Conflicting Priorities**: Multiple stakeholders often have different views on what is the most important requirement, making it difficult to prioritize.
- **Limited Business Domain Knowledge**: In some projects, BAs may not be familiar with the specific business domain, which can hinder their ability to gather accurate requirements and provide appropriate solutions.

Q12. Write about Document Naming Standards– 4 Marks

Answer: Document Naming Standards refer to a set of guidelines or conventions established to ensure uniformity, consistency, and clarity in naming documents across an organization or project. Adhering to these standards facilitates efficient document management, easy retrieval, and proper version control.

Key Elements of Document Naming Standards:

1. **Structure and Format:**
    - Follow a predefined structure that typically includes essential components such as project name, document type, version, and date.
    - Example: [ProjectName]_[DocumentType]_[Version]_[Date].
2. **Components of Naming Standards:**
    - **Project Name/Code:** A unique identifier for the project. For example, OnlineAgriStore or OAS.

- **Document Type:** Specify the purpose of the document (e.g., BRD for Business Requirement Document, SRS for Software Requirement Specification, TestPlan for Test Plan).
- **Version:** Use version numbers to track changes (e.g., v1.0, v2.1).
- **Date:** Include the creation or modification date in YYYYMMDD format to maintain chronological clarity.
- **Author/Team (Optional):** Indicate who created or owns the document, e.g., JohnDoe.

3. **General Guidelines:**
   - Use clear, descriptive terms instead of ambiguous or abbreviated names.
   - Avoid special characters (!@#$%^&*) and spaces; use underscores (_) or hyphens (-) instead.
   - Ensure names are concise but meaningful.
   - Maintain a consistent naming format across all documents.
4. **Examples of Document Names:**
   - OnlineAgriStore_BRD_v1.0_20241220.docx
   - OAS_SRS_v2.0_20241222.xlsx
   - OnlineAgriStore_TestPlan_v1.1_20241218.pdf
5. **Version Control:**
   - Use versioning to track changes and iterations. Increment the number for major changes (e.g., v1.0 to v2.0) and use sub-versions for minor updates (e.g., v1.1 to v1.2).
6. **Folder Structure Alignment:**
   - Integrate document naming conventions with the folder structure for better organization. Example folder hierarchy:

     markdown
     Copy code
     - OnlineAgriStore
       - BRD
         - OnlineAgriStore_BRD_v1.0_20241220.docx
       - SRS
         - OnlineAgriStore_SRS_v2.0_20241222.xlsx
       - Test Plans
         - OnlineAgriStore_TestPlan_v1.1_20241218.pdf

7. **Common Abbreviations:**
   Standardize abbreviations for frequently used terms, such as:
   - BRD: Business Requirement Document
   - SRS: Software Requirement Specification
   - UAT: User Acceptance Testing
   - MoM: Minutes of Meeting

Benefits of Document Naming Standards:

- Enhances collaboration by reducing ambiguity.
- Improves document search ability and retrieval.
- Facilitates proper version tracking and audit trails.
- Ensures consistency across teams and projects.

Q13.What are the Do and Don'ts of a Business analyst–6 Marks

Answer: A Business Analyst (BA) plays a pivotal role in bridging the gap between business needs and technical solutions. To be effective, there are key practices to follow (Do's) and behaviors to avoid (Don'ts). Below is a list of essential Do's and Don'ts for a Business Analyst.

Do's of a Business Analyst

1. **Do Understand the Business Needs and Stakeholder Expectations**:
    - A BA must have a deep understanding of the business domain, the specific project goals, and the stakeholders' expectations. This is crucial for ensuring that the project aligns with the business strategy.
2. **Do Communicate Clearly and Effectively**:
    - A BA needs to facilitate communication between various teams, such as business users, technical teams, and stakeholders. Clear communication helps in avoiding misunderstandings and scope creep.
3. **Do Document Requirements Thoroughly**:
    - Proper documentation is essential for creating a solid foundation for the project. It helps in tracking progress and ensuring alignment between business and technical teams.
4. **Do Involve Stakeholders Regularly**:
    - Involving stakeholders throughout the project lifecycle helps in maintaining alignment and ensures the delivered solution meets business expectations.
5. **Do Prioritize Requirements**:
    - Not all requirements are equally important. It is crucial to prioritize them based on business value, risk, or urgency to ensure that critical features are implemented first.
6. **Do Adapt to Changes and Stay Flexible**:
    - Projects often face changes in scope, priorities, or technologies. A BA must be adaptable and flexible to manage these changes without derailing the project.

Don'ts of a Business Analyst:

1. **Don't Make Assumptions**:
    - A BA must avoid making assumptions about what the business or stakeholders want, as it can lead to incorrect solutions and misunderstandings.
2. **Don't Focus Only on Technical Details**:

- While technical knowledge is important, the BA's primary role is to understand and represent business needs, not to get lost in technical details.

3. **Don't Ignore the Bigger Picture**:
   - A BA should not focus solely on individual requirements or features without understanding how they contribute to the overall business goal.
4. **Don't Neglect Stakeholder Alignment**:
   - If stakeholders are not aligned, it can lead to miscommunication, conflicting priorities, and project delays.
5. **Don't Overlook the User Perspective**:
   - Business Analysts often focus on business requirements but forget to consider the end-users. This can lead to a solution that may not be user-friendly or meet the users' needs.
6. **Don't Delay Documentation**:
   - Putting off documentation or doing it hastily leads to incomplete or unclear requirements that can cause confusion and delays.
7. **Never say No to the client.**
   - It is a critical guideline for Business Analysts (BAs) when working with stakeholders. While it doesn't mean agreeing to everything without consideration, it reflects a mindset of openness, collaboration, and solution-/.oriented thinking.

Q14. Write the difference between packages and sub-systems– 4 Marks

Answer:

Both **packages** and **sub-systems** are used in software design to organize and modularize a system. However, they serve different purposes and have different characteristics. Here's a comparison between the two:

Aspect	**Package**	**Subsystem**
Definition	A group of related classes or components for organizing code.	A larger, semi-independent module handling a major functionality of the system.
Scope	Operates within a single system or application.	Can operate independently.
Purpose	Organizes and manages code for better modularity and readability.	Provides significant functionality
Dependency	Can depend on other packages.	Collaborates with other sub-systems to ensure the system works as a whole.
Size	Smaller, focusing on specific functionalities or logic.	Larger, encompassing multiple packages or modules.

Implementation	Defined during coding to structure and organize components.	Designed during architectural planning to handle specific system functionalities.
Focus	Code-level organization for developers.	High-level functionality and system operation.
Example	user-authentication, payment-processing.	Order Management Sub-System**,** User Management Sub-System**.**

Q15. What is camel-casing and explain where it will be used- 6 Marks

Answer: Camel Casing is a method of writing compound words or phrases without spaces, where each word after the first one starts with a capital letter. The first word is written in lowercase. This naming convention is called "camel case" because the capital letters in the middle resemble the humps of a camel.

Example of Camel Casing:

- firstName
- userProfile
- orderDetails
- totalAmount

Where Camel Casing is Used for

As a Business Analyst (BA), you might use cam`1el casing in various parts of documentation, especially in contexts where you're dealing with **code, system components**, or **naming conventions**. Here's how camel casing can be useful:

1. **Naming Variables, Parameters, and Fields**
2. **Naming system Functions and Methods**
3. **Describing API Endpoints and Query Parameters**
4. **Naming Business Objects or Entities**
5. **Documenting Business Rules or Attributes**
6. **Naming System Components and Modules**
7. **Software Requirements and Use Cases**

1. Naming Variables, Parameters, and Fields

In documentation that involves system design or technical specifications, camel casing is used for variables, fields, function parameters, and attributes that need to be represented in the documentation.

Example:

- If you're documenting a **user input form**, you might refer to fields as userName, emailAddress, or contactNumber.

2. Naming Functions and Methods

When you're documenting **business processes** or workflows that map to **system functions or methods**, camel casing is used for function/method names.

Example:

- "The **createNewUser** function will allow users to register on the platform."
- "The **generateInvoice** method should automatically calculate and issue the invoice.

3. Describing API Endpoints and Query Parameters

For **API documentation**, camel casing is often used to describe endpoint paths or query parameters.

Example:

- **API Endpoints**: /getUserDetails, /updateOrderStatus
- **Query Parameters**: orderId, customerName, paymentMetho

4. Naming Business Objects or Entities

In **use case diagrams**, **data flow diagrams**, or when referring to business objects or entities, camel casing helps in naming **objects, attributes**, or **objects in classes** to maintain consistency with coding practices.

Example:

- A **customer** object might have attributes like customerName, customerEmail, orderHistory.

5. **Documenting Business Rules or Attributes**

In a **business rules document** or **requirements document**, camel casing may be used to maintain consistency, especially when mapping business rules to system functionality.

Example:

- "If the **orderAmount** exceeds $500, the **discountRate** is 10%."

- "The **userAccount** must be verified before proceeding with the order."

6. **Naming System Components and Modules**

When documenting **system architecture** or **components** in terms of sub-systems, modules, or services, camel casing is often used to name components in a readable and structured format.

Example:

- "The **inventoryManagementSystem** component interacts with the **orderProcessingSystem**."
- "The **paymentGatewayService** will handle payment transactions."

7**. Software Requirements and Use Cases**

In **software requirements documents** or **use case specifications**, camel casing helps to represent **user actions** or **system responses** concisely.

Example:

- "The user must be able to **submitOrder** and receive an order confirmation."
- "The system will **sendEmailNotification** to the user upon successful payment."

Q16. Illustrate Development server and what are the accesses does business analyst has?-6 Marks

Answer: A **Development Server** is an environment where software is developed, tested, and debugged during the early stages of a project. It serves as a controlled workspace for developers and other team members to build and test code without impacting the live system (Production Server). This is part of the Software Development Life Cycle (SDLC) and ensures that features and functionality are stable before deployment to users.

Components of a Development Server:

1. **Application Server:**
    - Hosts the application code being developed.
    - Examples: Apache Tomcat, Node.js, or .NET Core.
2. **Database Server:**
    - A database (e.g., MySQL, PostgreSQL, or MongoDB) to store and test application data.
    - Allows testing of CRUD (Create, Read, Update, Delete) operations.
3. **Version Control System:**
    - Tracks changes to the codebase and integrates them into the development environment.
    - Examples: Git, GitHub, Bitbucket.

4. **Testing Tools and Debugging Tools:**
    - Tools like Selenium or Postman to validate features and debug errors.
5. **Network Configuration:**
    - Often hosted on an internal network accessible only to the development team.

Development Server Workflow:

1. **Code Development:** Developers write code on their local machines and push it to the version control system.
2. **Build and Deploy:** The code is built and deployed to the development server for testing.
3. **Testing and Debugging:** QA teams and developers test the features, fix bugs, and ensure the application meets requirements.
4. **Approval:** Once validated, the code moves to staging for further testing before production deployment.

Accesses for Business Analyst (BA)

While the Business Analyst does not require full access to the Development Server, they have limited and specific permissions to ensure they can perform their role effectively.

1. **Accesses Typically Granted to a BA:**

- **Requirement Validation:**
    - View the deployed builds or features for validation against requirements.
    - Access to a demo environment or staging server derived from the development server.
- **Testing and Feedback:**
    - User interface testing or walkthroughs to ensure the application aligns with user stories and business needs.
- **Project Documents:**
    - Access to configuration documentation, test results, or logs for requirement traceability.
- **Issue Tracking Tools:**
    - Tools like Jira or Azure DevOps to raise, review, and track issues or changes.

2. **Tools/Systems Access for a BA:**

- **Read-only Access to Version Control:**
    - To review code comments, configurations, or repository structure if necessary.
- **Bug Tracking Tools:**
    - Permission to log and track bugs or enhancements.
- **Testing Environment:**
    - Limited access to execute UAT or observe test scenarios.

3. **Examples of Access Scenarios:**

- Reviewing a deployed feature to confirm search functionality meets requirements.
- Checking logs for errors related to specific modules to inform developers.
- Collaborating with testers to verify test cases align with requirements.

Q17. What is Data Mapping 6Marks

Answer: Data mapping is the process of linking or associating data from one format or structure to another, typically during data migration, integration, or transformation tasks.

It involves defining relationships between data elements in different databases, applications, or data systems to ensure that the data is transferred accurately and consistently.

Key Purposes of Data Mapping:

1. **Data Transformation**: When data is moved from one system or database to another, data mapping helps transform data from its original format to a new format that the destination system can understand. For example, transforming dates from MM/DD/YYYY to YYYY-MM-DD format.
2. **Data Integration**: In cases where multiple systems need to work together (e.g., integrating customer data from a CRM system into an ERP system), data mapping ensures that data from each system is correctly interpreted and merged.
3. **Data Migration**: When transferring data from an old system to a new one, data mapping ensures the data fits into the new system's structure. For example, mapping columns from one database to fields in another.
4. **Data Consistency**: It ensures that data is consistent across systems by defining how data elements in one system correlate with those in another. For example, ensuring the "first_name" field in one system matches the "given_name" field in another system.
5. **Data Cleansing**: Sometimes, data needs to be cleaned up (e.g., removing duplicates or correcting errors). Data mapping helps in setting the rules for how the data should be cleaned and transformed.

Types of Data Mapping:

1. **Manual Mapping**: Data elements are manually matched between the source and target systems. This is often used when the volume of data is small or when complex business rules need to be applied.
2. **Automated Mapping**: Tools or software are used to automatically map data fields from one system to another. This is typically used for large-scale data integration tasks, reducing manual effort and errors.
3. **One-to-One Mapping**: Each source data element maps to exactly one target data element. For example, a "customer_name" in one database might directly map to the "full_name" field in the other.

4. **One-to-Many or Many-to-One Mapping**: A single source data element might map to multiple target elements (one-to-many), or multiple source elements might map to one .target element (many-to-one). This is used when data needs to be split or combined during the mapping procsess.

Benefits of Data Mapping:

- **Consistency**: It ensures that data remains consistent across multiple systems, reducing the risk of errors.
- **Efficiency**: Streamlines the process of transferring or integrating data, saving time and effort.
- **Accuracy**: By defining clear rules for data mapping, the risk of misinterpretation or corruption of data is reduced.

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 10 Marks

Answer: An **API (Application Programming Interface)** is a set of rules, protocols, and tools that allow two or more software applications to communicate and exchange data. APIs define the way requests and responses are formatted and processed, enabling seamless integration between systems.

Step-by-Step Explanation**:**

1. **Understand the Problem**:

- **Your application format**: dd-mm-yyyy (e.g., 23-02-2024 for 23rd February 2024).
- **US application format**: mm-dd-yyyy (e.g., 02-23-2024 for 23rd February 2024).
- Problem arises because your application expects a different format than the one received from the US application.

2. **Importance of Correct Date Handling**:

- Misinterpretation of dates could lead to errors (e.g., 03-05-2024 could be March 5th or May 3rd).
- Ensures accurate processing, storage, and communication between systems.

3. **Steps to Solve the Problem**:

Step 1: Parse the Incoming Date

- Extract the date from the API response.

- Assume the US app sends a date string like 02-23-2024.
- Break the string into its components:
    - 02 = Month
    - 23 = Day
    - 2024 = Year.

Step 2: Convert to Standard Format

- Rearrange the extracted components into the required format dd-mm-yyyy:
    - Day becomes the first part.
    - Month becomes the second part.
    - Year remains the last part.
- For example, 02-23-2024 (US format) becomes 23-02-2024.

Step 3: Use the Converted Date in Your Application

- Use the converted date for storage, processing, or displaying to users in your application.

Step 4: Handle Errors Gracefully

- If the incoming date format is invalid or missing, display a proper error message or default value.
- Example: Handle cases like 13-32-2024, which are invalid.

Step 5: Test Thoroughly

- Test the conversion logic with multiple dates to ensure it handles all scenarios, including:
    - Valid dates (e.g., 12-25-2024 to 25-12-2024).
    - Invalid dates (e.g., 02-30-2024).
    - Edge cases like leap years.

4. **Key Considerations**:

- **Validation**:
    - Ensure the received date matches the mm-dd-yyyy format before conversion.
- **Error Handling**:
    - Use proper error messages for invalid or malformed dates.
- **Automation**:
    - Automate this conversion in your API integration layer.
- **Interoperability**:
    - If sending data back to the US app, convert the date back to their format (mm-dd-yyyy).

5. **Benefits of This Approach**:

- Prevents errors caused by mismatched formats.
- Ensures seamless integration between the two applications.
- Improves user experience by displaying dates in the expected format.

6. **Conclusion**:

To handle date format differences in API integration:

- Parse the incoming date in the mm-dd-yyyy format.
- Convert it to the required dd-mm-yyyy format.
- Handle errors and test thoroughly. This ensures that your application correctly processes dates, avoiding confusion or errors.