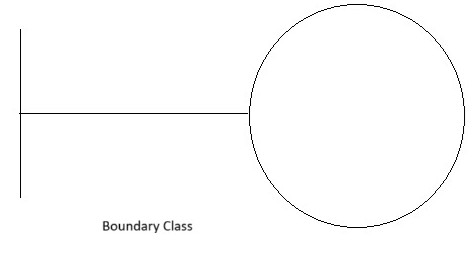
Case Study 1(Q1-Q6 24Marks)

A customer can pay through Card, Wallet, Cash, or Net banking.

Q1. Draw a Use Case Diagram- 4 Marks



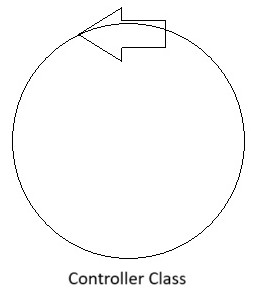
Q2. Derive Boundary Classes, Controller classes, Entity Classes.- 4 Marks



A **Boundary Class** acts as an interface between the external environment (e.g., user interface, APIs) and the core system, managing the input and output of data. It is responsible for validating, formatting, and transferring data to and from the system without containing business logic.

**Boundary Classes:**

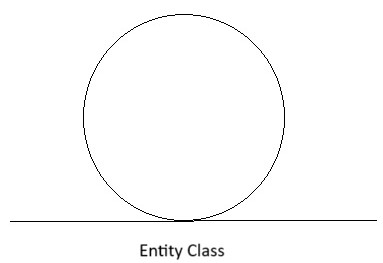
* **Payment Controller** handles payment initiation and status checking for all payment methods.
* **Card Payment Controller**, **Wallet Payment Controller**, **Cash Payment Controller**, and **Net Banking Payment Controller** manage respective payment type interactions.



A **Controller class** in software development (especially in MVC architecture) handles user inputs, processes them, and interacts with models and views to manage application flow. It acts as an intermediary between the user interface and the data models.

Base Payment Controller

Card Payment Controller



An **Entity class** represents a real-world object or concept in the system, typically corresponding to a table in a database, encapsulating its attributes and behaviors. It is used to store and manipulate data related to that entity.

Payment

Payment Method

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

**Presentation Layer (UI Layer)**: Handles user interaction (e.g., selecting payment method, inputting details).

* Classes: Payment View

**Business Logic Layer (BLL)**: Processes the payment logic and manages the entities.

* Classes: Payment, Card Payment, Wallet Payment, Cash Payment, Net Banking Payment, Payment Controller, Card Payment Controller, Wallet Payment Controller, Cash Payment Controller, Net Banking Payment Controller

**Data Access Layer (DAL)**: Responsible for persisting payment data.

* Classes: Payment Repository

Q4. Explain the Domain Model for Customers making payments through Net Banking-4 Marks

**Domain Modeling** is creating an abstract representation of the concepts, relationships, and rules within a specific problem domain. It serves as a blueprint to understand and communicate the structure and behavior of the system being modeled.



**How the Domain Model Works**

1. **Customer Initiates Payment**
   * A customer logs into the net banking system and selects the payment option. The system authenticates the customer using their credentials.
2. **Transaction Creation**
   * The customer enters details such as the amount, merchant details, and payment purpose. A transaction record is created in the system, marked as "Pending."
3. **Payment Gateway Processing**
   * The transaction is forwarded to the payment gateway, which acts as an intermediary to ensure secure communication between the customer's bank and the merchant's bank.
4. **Bank Account Validation**
   * The payment gateway validates the customer's bank account details, checks available balance, and authorizes the transaction.
5. **Transaction Update**
   * If successful, the system deducts the specified amount from the customer's bank account and credits it to the merchant's account. The transaction status is updated to "Completed." If there’s an issue, the status changes to "Failed."
6. **Notification**
   * The system sends a notification (email/SMS) to the customer with details about the transaction (amount, status, timestamp, etc.).
7. **Reconciliation and Reporting**
   * The system maintains logs for all transactions for reconciliation and reporting purposes. Both customers and merchants can view their respective transaction histories.

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



Q6. Explain the Conceptual Model for this Case- 4 Marks

**Conceptual Model** for a given case, we would consider the high-level representation of the problem domain, focusing on the key entities, their attributes, and relationships. Here's how to explain it:

1. **Definition of Conceptual Model**: A **conceptual model** outlines the foundational structure of a system or problem domain, showcasing the major elements (entities) and their interactions without delving into technical implementation details. It helps stakeholders understand the system's overall behavior and scope.
2. **Application to the Case**:
   * Identify the **entities** involved in the case (e.g., users, products, processes, or services).
   * Define the **attributes** that describe these entities (e.g., user name, product type, or order date).
   * Highlight the **relationships** between entities (e.g., "A user places an order" or "A product belongs to a category").
   * Incorporate any **business rules or constraints** relevant to the system's behavior.

A diagram (e.g., an Entity-Relationship Diagram or UML diagram) often complements this explanation to convey the connections visually. If you share the specifics of the case, I can refine the conceptual model further.

Conceptual Model for Customers making payments through Net Banking

1. **Customer Initiates Payment**
   * The customer logs into the net banking system using secure credentials and selects the option to make a payment.
2. **Authentication**
   * The system verifies the customer’s credentials and performs additional security checks (e.g., OTP or two-factor authentication) to confirm their identity.
3. **Transaction Details Entry**
   * The customer provides payment details, including:
     + Merchant information (e.g., name, account number, IFSC code).
     + Payment amount.
     + Purpose or description.
4. **Authorization via Payment Gateway**
   * The system forwards the transaction details to a payment gateway, which:
     + Validates the details.
     + Communicates with the customer’s bank to ensure sufficient funds are available.
     + Secures authorization from the customer.
5. **Transaction Processing by Banking System**
   * Once authorized:
     + The customer’s bank debits the specified amount from the bank account.
     + The merchant’s bank account is credited with the payment.
6. **Transaction Status Update**
   * The system updates the status of the transaction (e.g., "Completed" or "Failed") based on the outcome of the processing.
7. **Notification to Customer**
   * The customer receives a notification (email, SMS, or app alert) with details of the transaction, including:
     + Amount.
     + Timestamp.
     + Status.
8. **Reconciliation and Records**
   * Both the customer and merchant can view their respective transaction histories for record-keeping.

**High-Level Representation**

In a **Conceptual Model**, the focus is on understanding the relationships and interactions between these core entities:

* **Customer** is linked to a **Bank Account**.
* **Bank Account** is associated with one or more **Payment Transactions**.
* **Payment Transactions** involve **Merchants** and are facilitated by **Payment Gateways**.
* The **Banking System** ensures secure processing of all operations.

**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

**Ans 7.)** MVC (Model-View-Controller) is a software design pattern that separates an application into three interconnected components. This separation helps organize code, improve maintainability, and enable scalability.

1. **Model**:
   * Represents the data and business logic of the application.
   * Handles data operations, rules, and state management.
   * Does not interact directly with the user interface.
2. **View**:
   * Displays data from the model to the user.
   * Handles user interface and presentation logic.
   * Does not contain

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

**Ans 8.)**

The Business Analyst (BA) plays a critical role in all stages of the Waterfall Model, ensuring that the project aligns with business objectives and stakeholder expectations. Below is an explanation of the BA's contributions at each stage:

**1. Requirement Analysis**

* Key Responsibilities:
  + Elicit, analyze, and document detailed functional and non-functional requirements.
  + Collaborate with stakeholders to understand business needs and objectives.
  + Create artifacts like Business Requirement Documents (BRD), Functional Specification Documents (FSD), and Use Case Diagrams.
* Outcome:
  + Well-documented and signed-off requirements, serving as the foundation for the project.

**2. System Design**

* Key Responsibilities:
  + Provide inputs for system architecture based on business requirements.
  + Validate that the proposed design aligns with business goals.
  + Assist in creating wireframes, prototypes, and data flow diagrams.
* Outcome:
  + A design blueprint that meets both technical and business needs.

**3. Implementation (Development)**

* Key Responsibilities:
  + Act as a liaison between the business stakeholders and the development team.
  + Clarify requirements and address questions or ambiguities during development.
  + Ensure the development aligns with the documented requirements.
* Outcome:
  + Accurate implementation of requirements in the system.

**4. Testing**

* Key Responsibilities:
  + Support the creation of test cases and test scenarios to validate requirements.
  + Perform User Acceptance Testing (UAT) to ensure the system meets business needs.
  + Verify that all identified defects or issues are addressed.
* Outcome:
  + A defect-free system that meets user expectations.

**5. Deployment**

* Key Responsibilities:
  + Coordinate with stakeholders for go-live readiness.
  + Ensure that all requirements are met in the deployed system.
  + Facilitate user training sessions and create user manuals if required.
* Outcome:
  + Smooth transition from development to production with stakeholder satisfaction.

**6. Maintenance**

* Key Responsibilities:
  + Gather feedback from users and prioritize enhancements or fixes.
  + Collaborate with technical teams to implement changes.
  + Continuously align the system with evolving business needs.
* Outcome:
  + Sustained system performance and adaptability to changing requirements.

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

**Ans 9.** Conflict management involves identifying and addressing disputes effectively to minimize negative impacts and foster positive outcomes. It aims to resolve disagreements in a way that strengthens relationships and aligns with organizational or team goals.

**Thomas–Kilmann Conflict Management Model**

The Thomas–Kilmann model outlines five conflict resolution styles based on two dimensions:

* **Assertiveness:** The extent to which a person tries to satisfy their concerns.
* **Cooperativeness:** The extent to which a person tries to satisfy the concerns of others.

**Five Conflict Management Styles**

1. **Competing (High Assertiveness, Low Cooperativeness)**

* Focuses on winning the conflict at the expense of others.
* Best used in situations requiring quick decisions or where a firm stance is essential (e.g., emergencies).

1. **Collaborating (High Assertiveness, High Cooperativeness)**

* Involves working together to find a win-win solution that satisfies all parties.
* Effective for resolving complex conflicts where all viewpoints are important.

1. **Compromising (Moderate Assertiveness, Moderate Cooperativeness)**

* Seeks a middle ground where both parties give up something to reach an agreement.
* Suitable for conflicts requiring quick resolution with equal power dynamics.

1. **Avoiding (Low Assertiveness, Low Cooperativeness)**

* Involves sidestepping the conflict or postponing discussion.
* Useful when the issue is trivial or when emotions need to cool down before resolution.

1. **Accommodating (Low Assertiveness, High Cooperativeness)**

* Prioritizes the needs of others over one's concerns.
* Best for maintaining harmony or when the relationship is more important than the issue.

**Conclusion**

* The Thomas–Kilmann technique provides a structured way to analyze and resolve conflicts.
* Choosing the right style depends on the context, the importance of the issue, and the relationships involved.
* Using this framework can improve interpersonal dynamics and organizational effectiveness.

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

**Ans 10.**

**Reasons for Project Failure**

Project failure can occur due to various factors. Below are common reasons categorized into key areas:

**1. Poor Planning**

* Inadequate or unrealistic project goals and timelines.
* Lack of a clear project scope or objectives.
* Insufficient resource allocation (time, budget, or personnel).

**2. Communication Gaps**

* Miscommunication between stakeholders, team members, or clients.
* Lack of transparency in project progress or issues.
* Inadequate documentation and reporting.

**3. Requirement Issues**

* Incomplete or poorly defined requirements.
* Scope creep due to uncontrolled changes or additions.
* Misalignment between stakeholder expectations and deliverables.

**4. Lack of Stakeholder Engagement**

* Minimal involvement or commitment from key stakeholders.
* Conflicts or disagreements among stakeholders.
* Lack of user feedback during development.

**5. Team-related Challenges**

* Insufficient skills or training of team members.
* High staff turnover or lack of team cohesion.
* Ineffective leadership or project management.

**6. Technical Failures**

* Poor system design or architecture.
* Failure to adopt the right tools or technologies.
* Inefficient testing leading to undetected defects.

**7. External Factors**

* Market changes or shifts in organizational priorities.
* Regulatory changes or unforeseen risks.
* Economic or environmental disruptions.

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

**Challenges Faced by a Business Analyst (BA) in Projects**

1. **Eliciting Clear Requirements**
   * Stakeholders may have unclear, incomplete, or conflicting requirements.
   * Difficulty in extracting implicit needs or identifying hidden constraints.
2. **Managing Stakeholder Expectations**
   * Aligning diverse and sometimes conflicting interests of multiple stakeholders.
   * Unrealistic expectations regarding project timelines, scope, or outcomes.
3. **Scope Creep**
   * Frequent changes or additions to the project scope.
   * Balancing the need for flexibility with adherence to timelines and budgets.
4. **Communication Barriers**
   * Misunderstandings due to lack of clear communication or technical jargon.
   * Difficulty in bridging the gap between technical teams and non-technical stakeholders.
5. **Time and Resource Constraints**
   * Limited time to gather and document requirements thoroughly.
   * Insufficient resources, such as tools, personnel, or data, to support analysis.
6. **Dealing with Ambiguity**
   * Uncertain or incomplete information during project initiation.
   * Making decisions with limited or evolving data.
7. **Resistance to Change**
   * Resistance from stakeholders or end-users when introducing new processes or technologies.
   * Overcoming organizational inertia or cultural barriers to change.
8. **Ensuring Stakeholder Engagement**
   * Difficulty in getting timely input or approvals from busy stakeholders.
   * Lack of stakeholder involvement during critical project stages.
9. **Cross-Functional Collaboration Challenges**
   * Working with teams from different domains, geographies, or organizational hierarchies.
   * Managing conflicts or misaligned priorities across teams.
10. **Technology or Domain Knowledge Gaps**
    * Keeping up with rapidly changing technologies or industry trends.
    * Gaining domain expertise to effectively understand and solve business problems.

**Q12.** Write about Document Naming Standards– 4 Marks

**Ans 12.**

Document naming standards are predefined guidelines for naming files and documents in a structured, consistent, and meaningful manner. These standards help improve document organization, retrieval, and collaboration.

**Key Components of Document Naming Standards**

1. **Consistency**
   * Follow a uniform document format to ensure clarity and prevent confusion.
2. **Descriptive Naming**
   * Use meaningful names that describe the document's content, purpose, or context.
   * Example: ProjectName\_RequirementSpec\_v1.0.docx
3. **Version Control**
   * Include version numbers or dates to track updates and revisions.
   * Example: Proposal\_v2.1\_2023-12-01.docx
4. **Avoid Special Characters**
   * Use underscores (\_) or hyphens (-) instead of spaces or special characters.
   * Example: CustomerFeedback\_Q4-2023.xlsx
5. **Use of Metadata**
   * Incorporate key metadata such as project name, team/department, document type, or creation date.
   * Example: HR\_Policy\_Guide\_2024.pdf
6. **Standardized Abbreviations**
   * Use standardized abbreviations for terms to ensure readability.
   * Example: Req for Requirements, Spec for Specifications.

**Benefits of Document Naming Standards**

* Efficiency: Speeds up document retrieval and reduces search time.
* Collaboration: Ensures team members can quickly identify and use documents.
* Version Management: Simplifies tracking of updates and eliminates confusion over the latest version.
* Scalability: Supports large-scale project documentation.

**Q13.**What are the Do’s and Don’ts of a Business Analyst–6 Marks

|  |  |  |
| --- | --- | --- |
| Category | Do’s | Don’ts |
| Requirement Gathering | |  | | --- | | - Elicit clear, concise, and complete requirements. |  |  |  |  | | --- | --- | --- | |  | - Use effective techniques like interviews, workshops, and surveys. |  | | - Assume requirements without validation from stakeholders.  - Ignore stakeholders' implicit or unstated needs. |
| Communication | - Communicate effectively and adapt to the audience. | - Use excessive technical jargon with non-technical stakeholders. |
|  | - Listen actively and ensure alignment across all parties. | - Overlook stakeholder feedback or concerns. |
| Documentation | - Maintain detailed and structured documentation. | - Produce ambiguous or incomplete documents. |
|  | - Ensure proper version control of documents. | - Skip documentation updates after changes. |
| Stakeholder Engagement | - Build strong relationships with stakeholders. | - Exclude key stakeholders from critical discussions. |
|  | - Facilitate collaboration between teams. | - Let conflicts escalate without resolution efforts. |
| Problem-Solving | - Focus on finding practical, value-driven solutions. | - Propose solutions without understanding the problem fully. |
|  | - Analyze risks and impacts before suggesting changes. | - Ignore potential risks or downstream effects of decisions. |
| Professional Behavior | |  | | --- | | - Stay objective and impartial in conflicts. |  |  | | --- | |  | | - Take sides or show bias towards certain stakeholders or teams. |
|  | - Continuously enhance skills and domain knowledge. | |  | | --- | | - Resist learning new tools, methodologies, or industry updates. | |

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Aspect** | | |  | | --- | | **Packages** | | |  | | --- | | **Sub-Systems** | |
| Definition | |  | | --- | | Logical grouping of related classes, interfaces, or components within a system. |  |  | | --- | |  | | A distinct, standalone module or functional unit within a larger system. |
| |  | | --- | | Purpose |  |  | | --- | |  |  |  | | --- | |  | | Used for organizing and structuring code for better maintainability and reusability. | Represents a major functional component or division of a system's architecture. |
| Dependency | Packages are smaller and often depend on other packages for functionality. | Sub-systems are larger, self-contained, and can function independently or with minimal dependencies. |
| Level of Abstraction | Operates at a lower level, focusing on code organization. | Operates at a higher level, focusing on system architecture and functionality. |
| Examples | Java util package (e.g., java.util.ArrayList). | Payment processing sub-system in an e-commerce platform. |

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

**Ans 15.**

Camel-casing is a naming convention in programming where multiple words are combined into a single identifier, with the first word starting in lowercase and subsequent words beginning with an uppercase letter. It resembles the humps of a camel, hence the name.

**Types of Camel-Casing**

1. **Lower Camel Case (camelCase):**
   * Starts with a lowercase letter.
   * Commonly used for variable names, method names, and function names.
   * Example: order Details, fetch Data
2. **Upper Camel Case (Pascal Case):**
   * Starts with an uppercase letter.
   * Typically used for class names, namespaces, and enums.
   * Example: Employee Details, Customer Info

**Where Camel-Casing is Used**

1. **Programming Languages:**
   * Widely used in Java, JavaScript, C#, Python, and other programming languages for naming conventions.
2. **Variables and Functions:**
   * Helps improve readability and clarity in code.

**3.Class and Object Names:**

* Distinguishes classes or objects in object-oriented programming.

**4.APIs and Libraries:**

* + Used for consistent naming in public-facing APIs and libraries.
  + Example: getUserInfo, process Payment.

**5.Database Field Naming (sometimes):**

* + Applied to improve consistency in column naming.
  + Example: first Name, last Name.

**Benefits of Camel-Casing**

* Improves code readability and maintainability.
* Ensures consistency across codebases and teams.
* Helps avoid naming conflicts in programming.

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

Ans 16.

A **Development Server** is an environment used during the software development phase where code is written, tested, and debugged before moving to staging or production. It is a controlled space that mimics the production environment to a limited extent but is specifically tailored for development activities.

**Illustration of Development Server**

1. **Environment Setup:**
   * Includes the necessary tools, frameworks, and libraries.
   * A shared server where developers upload code for integration testing.
2. **Components:**
   * **Code Repository**: Version control system like Git or SVN.
   * **Database**: Test database mirroring production with dummy data.
   * **Application Server**: Hosts the application for testing purposes.
3. **Purpose:**
   * Testing new features and functionality.
   * Identifying and fixing bugs early.
   * Running integration tests for components.

**Accesses a Business Analyst (BA) Typically Has**

A BA’s access to the development server is often limited to ensure security and maintain focus on their responsibilities. However, they may have access to the following:

1. **Read-Only Access to the System:**
   * Allows the BA to view the application under development for testing or validation.
   * Example: Verifying if the user interface matches the requirements.
2. **Test Data and Configuration:**
   * Access to dummy data for testing workflows and use cases.
   * Viewing system logs to understand errors or issues related to requirements.
3. **Bug Tracking Systems:**
   * Tools like Jira or Bugzilla integrated with the development server for logging and tracking defects.
4. **User Acceptance Testing (UAT) Preparation:**
   * Verifying the implementation of requirements before transitioning to the staging or production server.
5. **Collaborative Tools:**
   * Access to collaboration tools or environments like Confluence for reviewing documentation or progress.

**Restricted Areas for a BA**

* **Codebase Access:**  
  BAs generally do not have permission to modify or upload code.
* **Database Writing Permissions:**  
  Direct modifications to databases are typically restricted to avoid accidental errors.

Q17. What is Data Mapping 6Marks ?

**Data Mapping** is the process of connecting or linking data elements from one format or structure to another. It involves establishing relationships between two data models to facilitate the transformation, migration, or integration of data between systems. Data mapping is essential in various processes, including data migration, system integration, and data warehousing.

**Key Aspects of Data Mapping**

1. **Source and Target Data Structures**  
   Data mapping connects a source data structure (e.g., database, flat file, or application) to a target data structure (e.g., another database, data warehouse, or external system).
2. **Transformation Rules**  
   Data mapping often includes defining transformation rules or business logic to convert data from one format to another. This can include data type conversions, concatenation, splitting, or aggregation.
3. **Manual or Automated Process**
   * **Manual Mapping**: When done manually, a data analyst or developer creates a mapping schema to specify how fields from the source map to those in the target.
   * **Automated Mapping**: Data mapping can also be automated using ETL (Extract, Transform, Load) tools or other data integration platforms.
4. **Types of Data Mapping**
   * **One-to-One Mapping**: One source data element map to one target data element.
   * **One-to-Many Mapping**: A single source data element maps to multiple target data elements.
   * **Many-to-One Mapping**: Multiple source data elements map to a single target data element.
   * **Many-to-Many Mapping**: Multiple source data elements map to multiple target data elements.

**Where Data Mapping is Used**

1. **Data Migration:**
   * Migrating data from one system to another, such as when upgrading databases or moving to a new software platform.
   * Example: Migrating customer information from an old CRM system to a new one.
2. **System Integration:**
   * Connecting different systems, applications, or databases within an organization to ensure data flows smoothly between them.
   * Example: Integrating a payment gateway system with an e-commerce platform.
3. **Data Warehousing:**
   * Consolidating data from multiple sources into a central repository (data warehouse), where it can be analyzed or reported on.
   * Example: Mapping transactional data from operational databases to an analytical model in a data warehouse.
4. **API Data Integration:**
   * Data mapping is used in APIs to ensure that data is transferred correctly between different services or platforms.
   * Example: A payment service provider sending transaction data to an accounting system.

**Steps Involved in Data Mapping**

1. **Identify Source and Target Structures**: Define the source and destination data formats, such as databases, files, or APIs.
2. **Define Mapping Rules**: Establish the rules or transformations required to map data correctly.
3. **Perform Data Transformation**: Convert data according to the established mapping rules.
4. **Validate Data**: Ensure that the mapped data is accurate, complete, and conforms to the required format.
5. **Implement in Target System**: Load the transformed data into the target system or database.

**Challenges in Data Mapping**

* **Data Quality Issues**: Source data may contain inconsistencies, missing values, or incorrect formats.
* **Complex Transformation Logic**: Some data transformations may be complex, requiring advanced business logic.
* **Changing Data Structures**: If source or target structures change over time, mappings need to be updated.
* **Scalability**: Managing large volumes of data during migration or integration.

**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 Applications from the US whose Date Format is mm-dd-yyyy 10 Marks

**Ans 18.** An **API (Application Programming Interface)** is a set of rules, protocols, and tools that allow different software applications to communicate with each other. APIs define the methods and data formats that applications can use to request and exchange information. They allow developers to interact with the functionality of another system, service, or platform without needing to understand its internal workings.

APIs are widely used for integrating different systems, enabling interoperability between diverse applications, and enabling access to third-party services such as payment gateways, social media platforms, or cloud services.

**API Integration for Date Format Conversion:**

In the context of your application, where the **date format is dd-mm-yyyy** and it needs to accept data from another application in the **US with a mm-dd-yyyy format**, API integration can be used to handle this inconsistency in date formatting. Here's how you can approach it:

**Steps for API Integration to Handle Date Format Differences**

**1. Identify the Date Format Issue:**

* Your application expects dates in the format dd-mm-yyyy (day-month-year).
* The external US application sends dates in mm-dd-yyyy (month-day-year) format.

This discrepancy in date formats can cause parsing errors, incorrect data storage, and confusion in displaying dates correctly in your system.

**2. API Design:**

You can design an API integration process where:

* The external application sends data with its date in mm-dd-yyyy format.
* Your application receives this data and uses an **API endpoint** to convert the date format before storing or processing it.

**3. Implementing the Date Format Conversion Logic:**

* When your application receives data from the external source, the API should include a date conversion function.
  + The API receives the date in mm-dd-yyyy format.
  + The API logic parses this date and converts it to the required dd-mm-yyyy format.