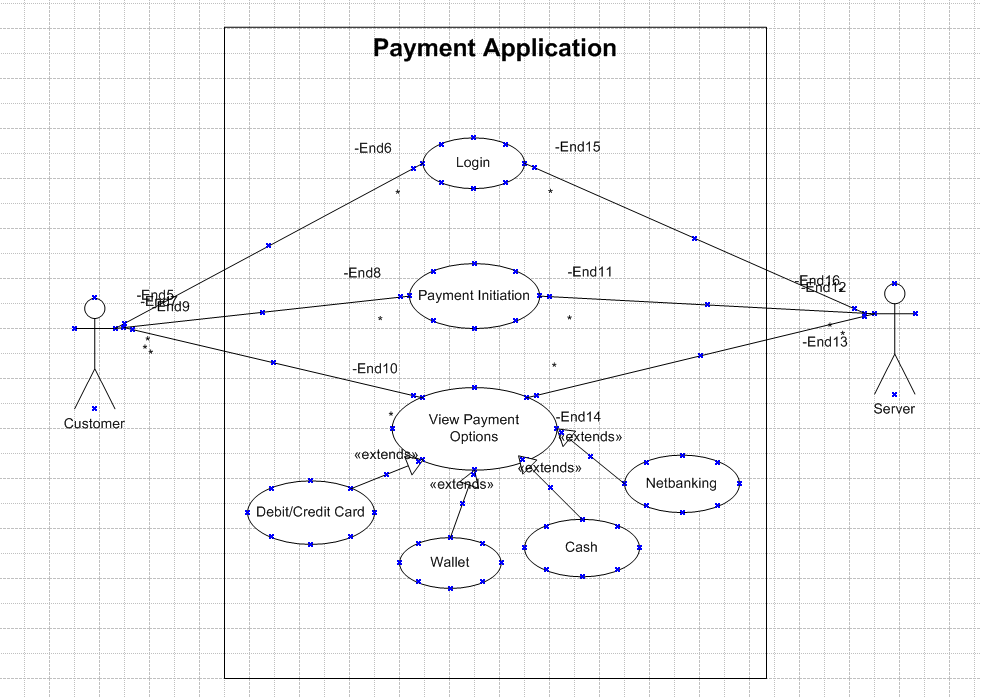
Q1. Draw a Use Case Diagram- 4 Marks

**Answer:**



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

**Answer:**

1. Boundary Classes

These classes handle the interaction between the system and its external entities (users, devices, or other systems).

Examples:

PaymentInterface

Represents the user interface for payment processing.

Responsibilities: Accept user input for payment type, amount, and details (e.g., card number, wallet ID).

ReceiptGenerator

Handles the display and delivery of receipts after a payment.

Responsibilities: Generate and display confirmation messages or email receipts.

NotificationService

Sends payment-related notifications (e.g., email or SMS).

Responsibilities: Inform users about successful payments or errors.

2. Controller Classes

Controller classes manage the flow of control between boundary classes and entity classes. They coordinate the application logic.

Examples:

PaymentController

Manages the payment process flow.

Responsibilities:

Validate payment details from the PaymentInterface.

Route payment requests to the appropriate entity class (e.g., CardPayment, WalletPayment).

Handle success or failure responses and notify the ReceiptGenerator or NotificationService.

TransactionController

Tracks the payment transactions.

Responsibilities:

Log transaction details.

Update the database with payment status.

3. Entity Classes

Entity classes represent the data and core business logic of the system. They persist and manipulate the data.

Examples:

Payment

Represents the generic payment data (e.g., amount, date, status).

Responsibilities: Store and retrieve payment details.

CardPayment

Handles card-specific payment processing.

Responsibilities: Validate card details, initiate card transactions.

WalletPayment

Handles wallet-specific payment processing.

Responsibilities: Check wallet balance, deduct the amount.

CashPayment

Handles cash-based transactions (e.g., recording cash payments).

Responsibilities: Record the payment receipt and mark as paid.

NetBankingPayment

Handles net banking-specific payment processing.

Responsibilities: Communicate with the bank API for authentication and fund transfer.

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

**Answer:**

1. Presentation Layer (Boundary Classes)

This layer handles user interaction and presentation.

Classes:

PaymentInterface

Handles user input for payment methods and displays results.

Examples: GUI for payment selection, web forms, or mobile app screens.

ReceiptGenerator

Displays the receipt on the screen or sends it to the user via email/SMS.

NotificationService

Sends notifications about payment status to users.

2. Business Logic Layer (Controller Classes)

This layer implements the business rules and processes. It acts as a mediator between the presentation layer and the data layer.

Classes:

PaymentController

Handles the overall payment workflow, such as selecting the appropriate payment method, validating inputs, and processing transactions.

TransactionController

Manages the transaction log and updates transaction status in the system.

Validators (optional sub-controllers)

These can validate data, such as card details or wallet credentials.

3. Data Layer (Entity Classes)

This layer manages data storage and retrieval. It interacts with the database or external systems.

Classes:

Payment

A generic entity to represent payment data (amount, status, date).

CardPayment

Contains logic for card-based transactions, such as communicating with a card payment gateway.

WalletPayment

Handles wallet-related operations like checking balances and deducting amounts.

CashPayment

Records cash payments and updates the status in the database.

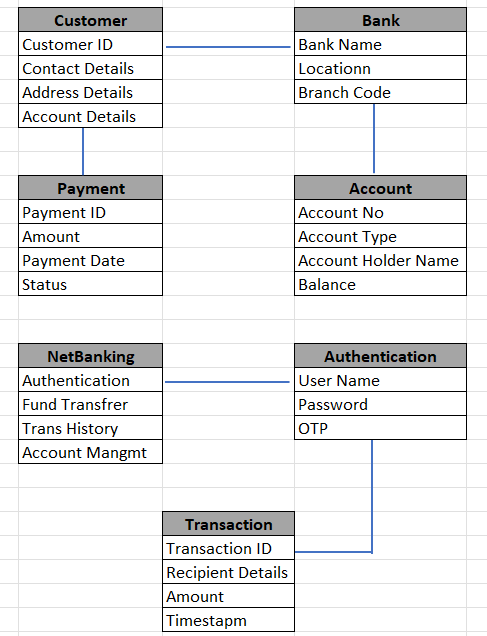
NetBankingPayment

Integrates with the bank's API for net banking transactions.

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

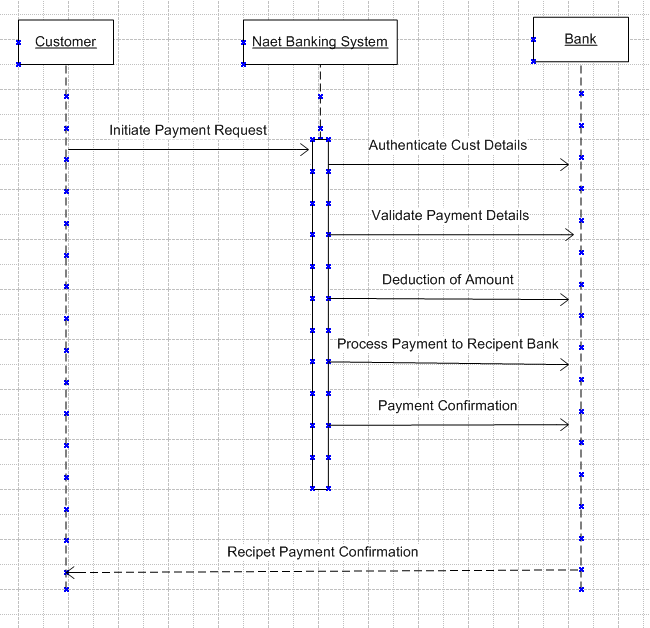
**Answer:**

A Domain Model for a customer making a payment through Net Banking represents the key entities, relationships, and attributes involved in the process. This model is part of the conceptual design phase, focusing on the business logic and interactions rather than technical implementation.



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

**Answer:**

****

Q6. Explain Conceptual Model for this Case- 4 Marks

**Answer:**

Conceptual model is high-level representation of a system that helps in understanding, visualising, and communicating the essential aspects of a domain.

It provides a clear and simplified view of the domain, making it easier to understand.

Key Elements of the Conceptual Model-

Entities

These represent real-world objects or concepts involved in the system.

Customer: The person making the payment.

Payment: The action or process of transferring money.

Payment Method: The mode of payment chosen by the customer (e.g., Card, Wallet, Cash, or Net Banking).

Transaction: The record of the payment, including its status, amount, and date.

Relationships-

A Customer initiates a Payment.

A Payment is processed using a specific Payment Method.

A Payment Method can be Card, Wallet, Cash, or Net Banking.

A Transaction is created as a result of the Payment.

Attributes-

Customer: Name, Email, Contact Number.

Payment: Payment ID, Amount, Status, Timestamp.

Payment Method: Type (Card, Wallet, Cash, Net Banking).

Transaction: Transaction ID, Payment Reference, Status, Timestamp.

Diagram Representation:

The conceptual model can be visualized as an Entity-Relationship Diagram (ERD):

Entities:

Customer

Payment

Payment Method

Transaction

Relationships:

A Customer makes a Payment.

A Payment uses a Payment Method.

A Payment generates a Transaction.

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:**

What is MVC Architecture?

Model-View-Controller (MVC) is a software design pattern that separates an application into three interconnected components:

Model: Represents the data and business logic of the application.

View: Represents the presentation layer that displays data to the user.

Controller: Acts as an intermediary, managing user interactions, updating the model, and selecting the appropriate view to display.

MVC Architecture Rules:

1. Combination of One Actor and an use case results in one Boundary class

2. Combination of Two Actors and an use case results in two Boundary classes

3. Combination of Three Actors and an use case results in Three Boundary classes and so on.

Note: only one primary actor is to be considered with a use case.

4. Use case will result in a controller class

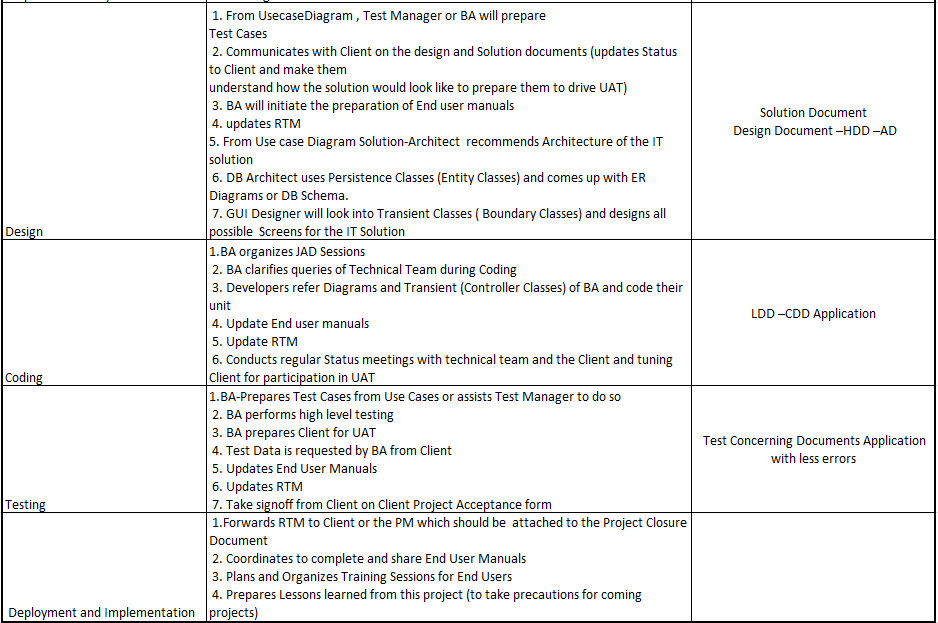
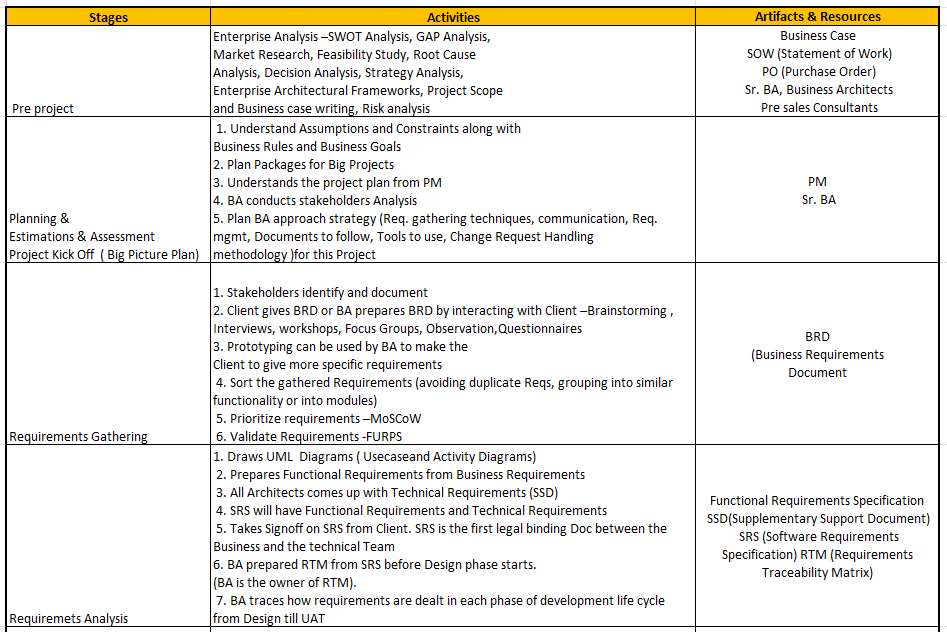
5. Each Actor will result in one entity class

Guidelines to place identified MVC Classes in a 3 Tier Architecture

* + Place all Entity Classes in DB Layer
  + Place Primary Actor associated Boundary Class in Application Layer
  + Place Controller Class in Application Layer
  + If governing Body influence or Reusability is there with any of remaining Boundary Classes, place them in Business Logic Layer else place them in Application Layer

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

**Answer:**



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

**Answer:**

Conflict Management refers to the process of identifying, addressing, and resolving disagreements or disputes constructively in a way that minimizes negative outcomes and enhances collaboration. It plays a crucial role in maintaining harmony within teams, improving relationships, and ensuring project success.

5 Steps of Conflict Management

Identify the conflict

Discuss the details

Agree with the Root problem

Check for every possible solution for the conflict

Negotiate the solution to avoid future conflicts

Thomas–Kilmann Conflict Management Model

The Thomas–Kilmann Conflict Mode Instrument (TKI) is a widely used framework for understanding and addressing conflicts. It identifies five conflict management styles based on two dimensions:

Assertiveness: The extent to which an individual attempts to satisfy their own concerns.

Cooperativeness: The extent to which an individual attempts to satisfy the concerns of others.

The Five Conflict Management Styles

Competing (High Assertiveness, Low Cooperativeness):

Focuses on achieving one's own goals at the expense of others.

Often used in situations where quick decision-making is critical or when a firm stance is necessary.

Example: A manager decides to implement a strict deadline without considering the team's concerns due to urgent client demands.

Collaborating (High Assertiveness, High Cooperativeness):

Involves finding a win-win solution that satisfies all parties.

Requires open communication, creativity, and mutual respect.

Best used when long-term relationships and innovative solutions are important.

Example: Two team members work together to combine their ideas into a comprehensive solution that benefits the project.

Compromising (Moderate Assertiveness, Moderate Cooperativeness):

Involves finding a middle ground where both parties give up something to reach an agreement.

Suitable for resolving issues quickly when both parties' stakes are moderate.

Example: A project manager agrees to extend a deadline slightly while the team agrees to work extra hours.

Avoiding (Low Assertiveness, Low Cooperativeness):

Involves sidestepping or ignoring the conflict, either temporarily or permanently.

Effective when the conflict is trivial or when more time is needed to gather information.

Example: A team member chooses not to engage in a disagreement during a meeting, preferring to address it privately later.

Accommodating (Low Assertiveness, High Cooperativeness):

Involves prioritizing the concerns of others over one's own.

Useful for maintaining relationships or when the other party's needs are more critical.

Example: A team member agrees to use another colleague's preferred method for completing a task to preserve teamwork.

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

**Answer**:

Project failures can occur for various reasons, often due to poor planning, execution, or communication. Below is a list of common reasons for project failure:

1. Poor Planning and Estimation

Inadequate scoping of project requirements.

Unrealistic timelines or budget estimations.

Lack of a clear roadmap or project plan.

2. Unclear or Changing Requirements

Ambiguous or incomplete requirements during the planning phase.

Frequent and uncontrolled scope changes (scope creep).

Misalignment between stakeholders on project goals.

3. Ineffective Communication

Lack of clear communication among team members or stakeholders.

Failure to convey updates, risks, or changes in scope.

Misunderstandings due to unclear or jargon-filled messages.

4. Inadequate Resource Allocation

Insufficient availability of skilled team members.

Over-reliance on specific individuals (bottlenecks).

Lack of tools, infrastructure, or technology needed for project execution.

5. Poor Risk Management

Failure to identify, analyze, and mitigate risks early.

Ignoring potential risks or assuming they won’t impact the project.

Lack of contingency plans for unexpected challenges.

6. Weak Leadership and Governance

Inadequate project sponsorship or executive support.

Ineffective project management practices.

Lack of decision-making authority for project managers.

7. Technical Issues

Use of outdated or inappropriate technology.

Technical failures during implementation.

Underestimation of system complexities or integration challenges.

8. Stakeholder Mismanagement

Lack of stakeholder involvement in critical phases.

Conflicting interests among stakeholders.

Disengagement of key stakeholders over time.

9. Budget Overruns

Overspending due to poor cost control or unforeseen expenses.

Failure to allocate budget for contingency plans.

Mismanagement of funds or misaligned priorities.

10. Unrealistic Expectations

Overpromising on deliverables or project outcomes.

Pressure to deliver within impossible deadlines or constraints.

Misalignment between project goals and stakeholder expectations.

11. Lack of Team Collaboration

Conflicts or lack of cooperation among team members.

Poor team dynamics or low morale.

Insufficient coordination in distributed or remote teams.

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

**Answer**: A Business Analyst (BA) often encounters several challenges while working on projects, as they play a pivotal role in bridging the gap between stakeholders and the development team. Below is a list of common challenges faced by BAs in projects:

1. Unclear or Incomplete Requirements

Stakeholders are unable to articulate their needs clearly.

Requirements are vague, ambiguous, or constantly changing.

Difficulty in identifying implicit requirements.

2. Conflicting Stakeholder Interests

Different stakeholders have conflicting priorities or objectives.

Difficulty in managing expectations and reaching a consensus.

Political dynamics among stakeholders impacting decisions.

3. Communication Barriers

Miscommunication between stakeholders and the technical team.

Use of technical jargon that stakeholders may not understand.

Challenges in conveying complex requirements to non-technical stakeholders.

4. Scope Creep

Continuous addition of new features or changes without proper analysis.

Stakeholders introducing last-minute requirements.

Lack of strict change management processes.

5. Resistance to Change

Stakeholders or end-users reluctant to adopt new systems or processes.

Teams clinging to old methods rather than embracing improvements.

Difficulty in demonstrating the value of changes.

6. Limited Stakeholder Availability

Stakeholders are too busy or unavailable for elicitation and validation.

Delays in obtaining feedback or approvals.

Lack of engagement during critical project phases.

7. Tight Deadlines

Limited time for thorough requirement analysis and documentation.

Pressure to deliver quickly, leading to overlooked details.

Balancing quality with speed.

8. Managing Expectations

Stakeholders having unrealistic expectations about timelines, budget, or deliverables.

Misalignment between what stakeholders expect and what is feasible.

Challenges in managing disappointment or dissatisfaction.

9. Insufficient Domain Knowledge

Difficulty in understanding the domain-specific terminology and processes.

Challenges in quickly gaining expertise in unfamiliar industries or technologies.

Dependence on Subject Matter Experts (SMEs) for detailed insights.

10. Poor Collaboration with Technical Teams

Misunderstandings between BAs and developers due to lack of clarity.

Resistance from technical teams to adopt the documented requirements.

Difficulty in managing technical constraints while meeting business needs.

Q12. Write about Document Naming Standards– 4 Marks

**Answer:**

Document Naming Standards

Document Naming Standards are guidelines established to create consistent, clear, and easily identifiable names for project-related documents. These standards ensure that documents are organized, traceable, and accessible to all stakeholders involved in a project.

Importance of Document Naming Standards

Consistency: Provides uniformity across all documents, making them easier to locate and understand.

Traceability: Helps track document versions, authors, and purposes.

Collaboration: Simplifies document sharing and reduces confusion among team members.

Time Efficiency: Saves time spent searching for documents.

Compliance: Supports adherence to organizational policies or regulatory requirements.

A well-defined naming standard typically includes the following elements:

1. Project/Program Name or Code

Identifies the project or program the document belongs to.

Example: OAPS for the "Online Agriculture Products Store."

2. Document Type

Specifies the type of document (e.g., BRD, FRS, UAT).

Example: BRD for Business Requirements Document.

3. Version Number

Indicates the revision or iteration of the document.

Example: v1.0 for the initial version and v1.1 for minor revisions.

4. Date

Provides the date the document was created or last updated.

Example: 2024-12-22.

5. Author/Owner Initials

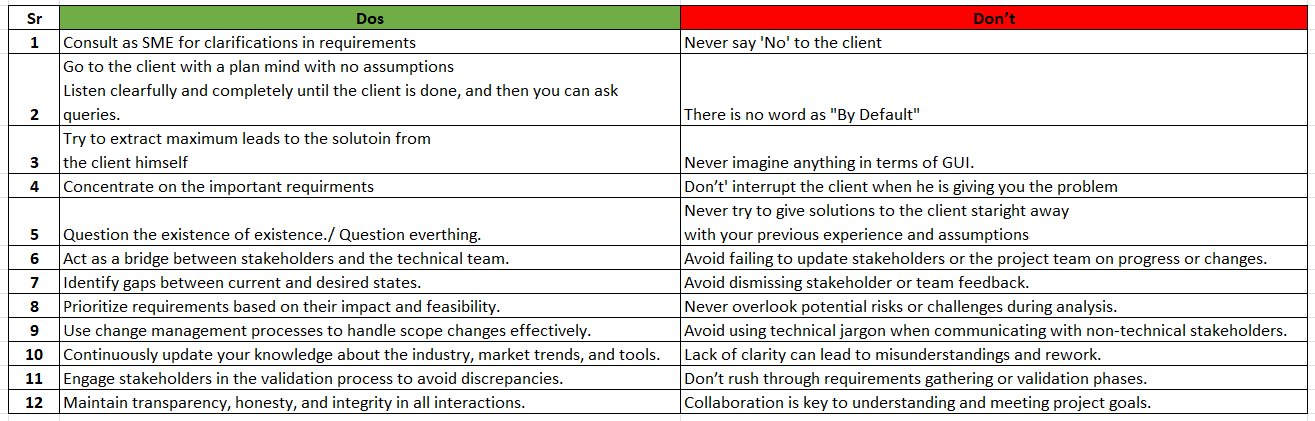
Identifies the person responsible for creating or updating the document.

Example: JD for John Doe.

Example: OAPS\_BRD\_v1.0\_2024-12-22\_JD

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

**Answer**:



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

**Answer:**

Packages and sub-systems are essential concepts in software engineering and system design, but they serve different purposes and operate at different levels of abstraction.

Packages:

Collection of components which are not reusable in nature.

Example: Application development companies work on Packages. Like, Infosys, TCS and HCL etc.

Sub Systems:

Collection of components which are reusable in nature.

Example: Product development companies work on Sub systems. Like, Wipro, IBM and Microsoft, etc.

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

**Answer**:

Camel-Casing

Camel-casing is a naming convention used in programming where each word in a compound phrase starts with a capital letter, except the first word. It creates a clear and readable structure for multi-word identifiers without using spaces or special characters. The name "camel-casing" comes from the uppercase "humps" that resemble a camel's back.

Types of Camel-Casing

Lower Camel Case (camelCase):

The first word starts with a lowercase letter, and each subsequent word starts with an uppercase letter.

Example: customerName, orderTotal, calculateDiscount.

Upper Camel Case (PascalCase):

Each word, including the first one, starts with an uppercase letter.

Example: CustomerName, OrderTotal, CalculateDiscount.

Where Camel-Casing is Used

Camel-casing is commonly used in various programming languages and scenarios to improve code readability and adhere to naming conventions. Below are some key areas where camel-casing is applied:

1. Variable Names

Lower camel case is often used to name variables.

Example:

java

Copy code

int customerAge = 30;

double orderAmount = 150.75;

2. Function/Method Names

Lower camel case is the standard for naming functions or methods.

Example:

python

Copy code

def calculateTotalPrice(items):

return sum(items)

3. Class Names

Upper camel case (PascalCase) is typically used for class names.

Example:

java

Copy code

public class CustomerDetails {

// Class definition

}

4. Object Properties

Lower camel case is used for properties in objects.

Example:

javascript

Copy code

const customer = {

firstName: "John",

lastName: "Doe",

age: 25

};

5. File Names

PascalCase is sometimes used for naming files, especially in object-oriented programming contexts.

Example: CustomerDetails.java, OrderProcessing.cs.

6. JSON Keys

Lower camel case is a common practice for naming keys in JSON objects.

Example:

json

Copy code

{

"firstName": "John",

"lastName": "Doe",

"age": 25

}

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

**Answer:**

Development Server and Business Analyst Access

Development Server

A development server is a server environment where developers create, test, and debug software applications. It acts as a sandbox environment, separate from the production server, to ensure that changes can be made safely without affecting live operations.

Key Features of a Development Server

Isolated Environment: Allows testing of new features and fixes without impacting production.

Version Control: Integrated with version control systems (e.g., Git) to manage code changes.

Debugging Tools: Equipped with tools to diagnose and resolve issues in the codebase.

Testing Support: Enables unit testing, integration testing, and sometimes user acceptance testing (UAT).

Access Control: Access is restricted to authorized personnel such as developers, testers, and sometimes business analysts.

Role of a Business Analyst (BA) in Development Server-

While a BA is not typically involved in coding or direct server management, they may require access to specific areas of the development environment to fulfil their responsibilities.

Accesses for a Business Analyst:

Below are the types of access and actions a BA might have in a development server environment:

1. Requirement Validation

Access to test or staging environments where the development server mirrors the application for verification.

Tasks:

Validate implemented features against documented requirements.

Perform initial user acceptance testing (UAT).

2. Test Data Setup

Access to create or upload sample data in the development environment for testing scenarios.

Tasks:

Populate fields with test cases.

Verify edge cases or specific user workflows.

3. Application Logs and Reports

Read-only access to application logs to identify system behavior during specific scenarios.

Tasks:

Monitor feature functionality.

Collaborate with developers to debug issues during testing.

4. Access to Prototypes/Demos

Access to a web-based or local demo of the application hosted on the development server.

Tasks:

Review UI/UX.

Provide feedback based on business rules.

5. Collaboration Tools

Integration with tools hosted on the development server, such as Jira or Confluence.

Tasks:

Update requirement traceability.

Monitor progress on features or issues logged by the development team.

Q17. What is Data Mapping 6 Marks.

**Answer:**

Data mapping is the process of connecting or "mapping" data fields from one data source (such as a database, file, or application) to corresponding fields in another source or destination. It ensures that data is correctly transferred, transformed, or consolidated between systems, applications, or processes.

Purpose of Data Mapping

The primary goal of data mapping is to maintain data integrity and consistency when data is moved, transformed, or integrated between systems. It is an essential step in processes like data migration, data integration, ETL (Extract, Transform, Load), and system interconnectivity.

How Data Mapping Works

Identify Source Data: Define the structure and fields of the data from the source system.

Identify Target Data: Define the structure and fields of the data in the target system.

Define Mappings: Establish relationships between the source fields and the target fields.

Apply Transformations (if needed): Specify rules or formulas to transform the data during mapping (e.g., changing formats, units, or types).

Validate Mapping: Ensure that mappings are correct, complete, and meet business requirements.

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:**

What is an API?

An API (Application Programming Interface) is a set of rules and protocols that allow different software applications to communicate and exchange data. It acts as a bridge between systems, enabling them to share information or functionalities without directly exposing the underlying code.

Key Characteristics of an API

Interoperability: Allows diverse systems to interact, even if they are built on different technologies.

Standardization: Follows specific formats (e.g., REST, SOAP) to ensure consistent communication.

Data Exchange: Facilitates sending and receiving data in structured formats like JSON or XML.

Security: Often secured using authentication methods like API keys, OAuth, or tokens.

Using API Integration in Your Application

In the context of your Online Agriculture Products Store, API integration could facilitate data exchange with external systems, such as payment gateways, logistics partners, or third-party databases. For this example, let's address the scenario of handling date formats during API integration.

Scenario: Handling Date Format Differences

Problem:

Your application: Expects date inputs in the format DD-MM-YYYY.

US Application: Sends date inputs in the format MM-DD-YYYY.

Directly using the US application's date without transformation will cause errors or incorrect data interpretation.

Solution: Transforming Date Formats Using API Integration

Step 1: Receive the API Request

The US application sends data to your application's API in the MM-DD-YYYY format.

Example Request:

json

Copy code

{

"order\_id": "12345",

"order\_date": "12-25-2024",

"customer\_name": "John Doe"

}

Step 2: Analyse the Incoming Data

Your application parses the API request payload to extract the date field.

Step 3: Transform the Date Format

Convert the MM-DD-YYYY format into your application's required format DD-MM-YYYY.

Example Transformation Code (Python):

python

Copy code

from datetime import datetime

def transform\_date(date\_str):

# Parse the date in US format

us\_format = datetime.strptime(date\_str, "%m-%d-%Y")

# Convert to required format

return us\_format.strftime("%d-%m-%Y")

# Example usage

incoming\_date = "12-25-2024"

transformed\_date = transform\_date(incoming\_date)

print(transformed\_date) # Output: "25-12-2024"

Step 4: Store/Process the Data

Use the transformed date in your application to store or process the order.

Step 5: Respond to the API Request

Send a success response back to the US application.

Example Response:

json

Copy code

{

"status": "success",

"message": "Order processed successfully"

}

Implementation Guidelines

Validation:

Validate the incoming date to ensure it adheres to the expected format before transformation.

Error Handling:

Handle scenarios where the date format is incorrect or missing, and return an appropriate error message.

Example Error Response:

json

Copy code

{

"status": "error",

"message": "Invalid date format. Expected MM-DD-YYYY."

}

Testing:

Test the API integration thoroughly using tools like Postman or Swagger to ensure data consistency.

API Documentation:

Document the API to clarify the expected date format and response structure.