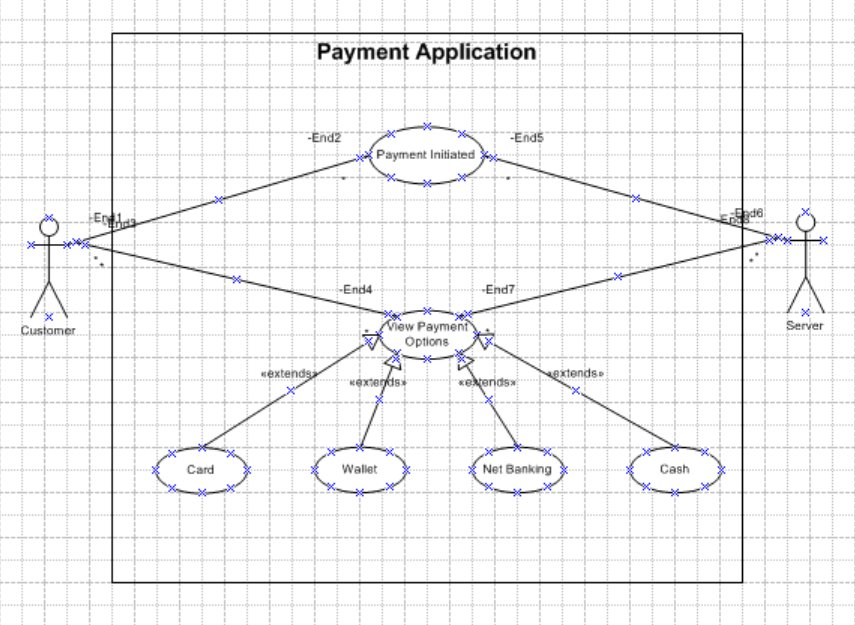
**CAPSTONE PROJECT 3**

Case Study 1 (Questions from Q1 to Q6)

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

**Question 1:** Draw a Use Case Diagram **4 marks**

**Answer:**

****

**Question 2:** Derive Boundary Classes, Controller classes, Entity Classes. **4 marks**

**Answer:** In Object-Oriented Design (OOD), **Boundary, Controller, and Entity classes** are essential components of the **Model-View-Controller (MVC) architecture**. These classes help structure a system for maintainability, scalability, and efficiency.

* ***Boundary Classes (View Layer / UI Layer):***
* **Definition:** Boundary classes represent the **interaction between the system and external actors** (users, systems, or devices). They handle **input and output** operations, such as **GUI screens, web pages, forms, and APIs**.
* **Example:**
* A **Login Form** where users enter credentials.
* An **API endpoint** that receives user data.
* **Key Responsibilities:**
* Interacts with **users or external systems**.
* Passes **data to controllers** for processing.
* Displays **results from controllers or entities**.
* ***Controller Classes (Control Layer / Business Logic Layer):***
* **Definition:** Controller classes manage the **flow of data** between **Boundary** and **Entity** classes. They **process user requests, apply business logic, and control system behavior**.
* **Example:**
* A **LoginController** that verifies user credentials.
* A **ShoppingCartController** that manages cart operations.
* **Key Responsibilities:**
* **Receives input** from boundary classes.
* **Processes data** using entity classes.
* **Implements business rules** and system logic.
* **Sends responses** to the boundary class
* ***Entity Classes (Model Layer / Data Layer):***
* **Definition:** Entity classes represent **real-world objects and store business data**. These classes are typically **mapped to a database** and contain **attributes and relationships**.
* **Example:**
* A **User** entity storing login credentials.
* A **Product** entity representing an item in an inventory system.
* **Key Responsibilities:**
* A **User** entity storing login credentials.
* A **Product** entity representing an item in an inventory system.

|  |  |
| --- | --- |
| **Boundary Classes** | PaymentOptionBoundary  CardPaymentBoudary  WalletPaymentBoundary  CashPaymentBoundary  NetBankingPaymentBoundary |
| **Controller Classes** | PaymentInitiatedController  CardPaymentController  WalletPaymentController  CashPaymentController  NetBankingPaymentController |
| **Entity Classes** | Customer  Payment  Card  Wallet  Server |

**Conclusion:**

* **Boundary Classes** handle user interactions.
* **Controller Classes** manage business logic.
* **Entity Classes** store and manage data.

By properly organizing these classes, systems become modular, scalable, and easy to maintain.

**Question 3:** Place these classes on a three tier Architecture. **4 marks**

**Answer:** The 3-tier architecture is a software architecture design pattern that divides an application into three logical and physical layers. This approach separates concerns, enhances scalability, simplifies maintenance, and improves application performance.

***Layers in a 3-tier Architecture***

* **Presentation Layer (User Interface Layer):** This layer interacts directly with the user. It provides the graphical user interface (GUI) or command-line interface (CLI) for the user to interact with the application.
* **Business Logic Layer (Application Layer):** This layer contains the application's core logic and rules. It processes user requests and determines how data should be managed and transferred between the layers.
* **Data Layer (Database Layer):** This layer manages the storage, retrieval, and manipulation of data. It interacts with the database to perform operations such as querying, updating, or deleting records.

***Benefits of 3-Tier Architecture***

* **Scalability:** Each layer can be scaled independently.
* **Modularity:** Code is more organized, making it easier to maintain and update.
* **Security:** Sensitive data is stored in the data layer, isolated from direct user access.
* **Reusability:** The business logic layer can be reused across multiple interfaces (e.g., web and mobile apps).
* **Flexibility:** Changes in one layer do not significantly impact others.

|  |
| --- |
| ***User Layer*** |
| PaymentMethodSelectionBoundary |
| CardPaymentBoundary |
| WalletPaymentBoundary |
| CashPaymentBoundary |
| NetBankingPaymentBoundary |
| ***Business Logic*** |
| PaymentController |
| CardPaymentController |
| WalletPaymentController |
| CashPaymentController |
| NetBankingPaymentController |
| ***Data Tier*** |
| Customer (Entity Class) |
| Payment (Entity Class) |
| Card (Entity Class) |
| Wallet (Entity Class) |
| BankAccount (Entity Class) |

**Question 4:** Explain Domain Model for Customer making payment through Net Banking **4 marks**

Answer: A **Domain Model** represents the real-world concepts involved in a system and their relationships. It consists of **entities, attributes, and associations** that define how different components interact.

In the case of **Customer making a payment through Net Banking**, the key entities are:

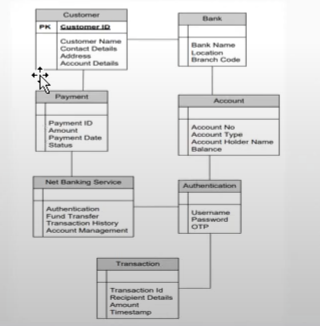
* **Customer** – User making the payment.
* **Bank Account** – Customer’s registered bank details.
* **Payment** – Transaction details.
* **Merchant** – Seller receiving the payment.
* **Bank Gateway** – A secure channel to process payments.

***Domain Model – Key Entities & Their*** Attributes

|  |  |  |
| --- | --- | --- |
| **Entity** | **Attributes** | **Relationships** |
| **Customer** | Customer ID, Name, Email, Phone No. | Has a **Bank Account**, Initiates **Payment** |
| **Bank Account** | Account No, Bank Name, IFSC Code, Balance | Linked to a **Customer**, Used for **Payments** |
| **Payment** | Payment ID, Amount, Status, Date & Time | Initiated by **Customer**, Sent via **Bank Gateway** |
| **Merchant** | Merchant ID, Business Name, Bank Account | Receives **Payment** from **Customer** |
| **Bank Gateway** | Gateway ID, Security Token, Processing Status | Facilitates **Payment** securely |

***Process Flow for Net Banking Payment***

* **Customer selects net banking** as a payment method on the merchant's website.
* **Payment request is sent** to the **Bank Gateway**.
* **Bank Gateway verifies details** and redirects the customer to their bank’s portal.
* **Customer enters credentials** and approves the payment.
* **Bank deducts the amount** from the **Bank Account** and confirms success.
* **Merchant receives confirmation** and updates the order status.

****

**Question 5:** Draw a sequence diagram for payment done by Customer Net Banking **4 marks**

**Answer:** A Sequence Diagram represents the step-by-step flow of a system, showing how different objects interact over time. In the case of Customer making a payment through Net Banking, the key participants are:

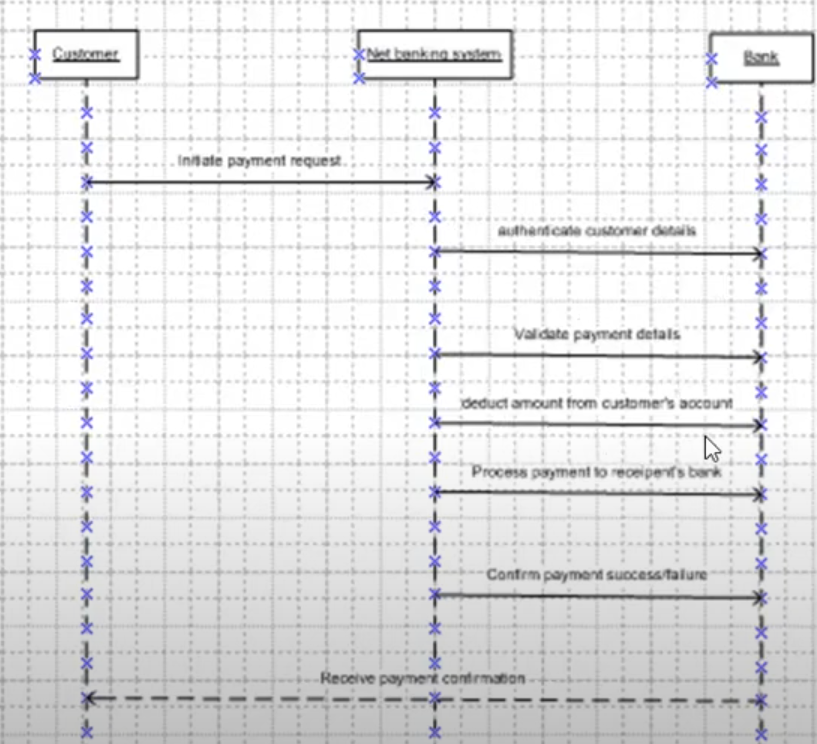
* ***Actors (Objects in the System)***
* **Customer** – Initiates the payment.
* **Merchant Website** – Sends the payment request.
* **Bank Gateway** – Processes the transaction securely.
* **Bank Server** – Verifies credentials and transfers funds.
* **Merchant Account** – Receives the payment confirmation.

***Sequence Diagram Steps***

* **Customer selects Net Banking** as the payment method on the merchant’s website.
* **Merchant Website redirects** to the **Bank Gateway**.
* **Bank Gateway requests authentication** from the **Bank Server**.
* **Customer enters login credentials** for authentication.
* **Bank Server verifies details** and deducts the amount from the **Bank Account**.
* **Transaction success message** is sent to the **Bank Gateway**.
* **Bank Gateway confirms payment** to the **Merchant Website**.
* **Merchant updates order status** and displays a confirmation message to the customer.

***Benefits of Sequence Diagram***

* **Clear Transaction Flow** – Shows step-by-step payment processing.
* **Identifies Key Actors** – Customer, Bank, and Merchant interactions.
* **Ensures Security & Validation** – Authentication steps for secure transactions.
* **Easy Debugging** – Helps in identifying payment failures and bottlenecks.



This sequence diagram ensures a **structured and secure** Net Banking payment process.

**Question 6:** Explain Conceptual Model for this Case **4 marks**

**Answer:** A **Conceptual Model** represents high-level business entities and their relationships, focusing on **what** the system must include rather than **how** it will be implemented. It is useful for understanding the **domain structure** before designing the actual system.

**Case Overview**

A **Customer** can make a payment through **four different modes**:

* **Card Payment** – Using debit/credit cards.
* **Wallet Payment** – Using digital wallets like Paytm, Google Pay, etc.
* **Cash Payment** – Direct cash transactions.
* **Net Banking** – Online banking transactions.

**Key Entities in the Conceptual Model**

|  |  |  |
| --- | --- | --- |
| **Entity** | **Attributes** | **Relationships** |
| **Customer** | Customer ID, Name, Email, Phone No. | Can make multiple **Payments** |
| **Payment** | Payment ID, Amount, Date, Status | Linked to a **Payment Method** |
| **Payment Method** | Method ID, Type (Card, Wallet, Cash, Net Banking) | Used for making **Payments** |
| **Bank Account** | Account No, Bank Name, IFSC Code | Required for **Net Banking** |
| **Card** | Card No, Expiry Date, CVV, Bank Name | Used for **Card Payments** |
| **Wallet** | Wallet ID, Wallet Name, Balance | Used for **Wallet Payments** |

**Explanation of Model Components**

* **Customer:**
* A **Customer** can make multiple **Payments** using different methods.
* **Payment:**
* Represents a **single transaction** made by a customer.
* Linked to a **Payment Method** that defines the mode of payment.
* **Payment Method:**
* Defines whether the payment is made by **Card, Wallet, Cash, or Net Banking**.
* Associated with **Bank Account, Card, or Wallet**, depending on the method chosen.
* **Bank Account:**
* Required **only for Net Banking payments**.
* **Card:**
* Stores card-related details when a **Card Payment** is made.
* **Wallet:**
* Represents digital wallets like **Paytm, Google Pay, or PhonePe**.
* **Cash:**
* Simple entity representing direct **cash payments**.

***Payment Process Flow Based on Conceptual Model***

* **Customer selects a payment method (Card/Wallet/Cash/Net Banking).**
* **System verifies** the selected method details (e.g., Card expiry, Wallet balance).
* **Payment is processed** through the respective entity (Bank, Wallet, or Cash counter).
* **Transaction status is updated** (Success/Failure).

***Benefits of This Conceptual Model***

* **Flexible Payment System** – Supports multiple modes.
* **Clear Entity Relationships** – Well-defined dependencies.
* **Scalability** – Easy to add new payment methods.
* **Security** – Bank & Card details remain separate.

**Conclusion**

This **Conceptual Model** provides a **high-level structure** for handling customer payments through **multiple payment methods**, ensuring a **secure and flexible** transaction process

**Question 7:** 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:** Model-View-Controller (MVC) is a software architectural pattern used to separate an application into three interconnected components:

* **Model** – Represents the data and business logic of the application. It is responsible for retrieving, storing, and managing data.
* **View** – Displays the data to the user and manages UI components. It updates based on the Model’s data.
* **Controller** – Acts as an intermediary between Model and View. It processes user inputs and updates the Model accordingly.

This separation enhances modularity, maintainability, and scalability in software development.

***Key Benefits of MVC Architecture:***

* **Separation of Concerns** – Improves maintainability and scalability.
* **Reusability** – Components can be modified or replaced independently.
* **Better Collaboration** – Developers can work on different components simultaneously.

***How MVC Works?***

* The **user interacts** with the View (e.g., clicking a button).
* The **Controller processes** the request and updates the Model.
* The **Model updates the data** and notifies the View.
* The **View refreshes** to display updated information.

This structured approach is widely used in **web applications, desktop software, and mobile apps**, making development more efficient and organized.

***Rules to Derive Classes from Use Case Diagram:***

* **Identify Actors and Use Cases –** Determine key functionalities and entities interacting with the system.
* **Extract Nouns from Use Case Descriptions –** Convert nouns into potential classes (e.g., "Customer," "Order," "Product").
* **Define Responsibilities for Each Class –** Assign appropriate attributes and methods based on use case requirements.
* **Determine Relationships Between Classes –** Establish associations, generalizations, and dependencies among classes.
* **Identify Boundaries Between Model, View, and Controller –** Classify each class into MVC components based on its function.

***Guidelines to Place Classes in 3-Tier Architecture:***

The **3-tier architecture** consists of:

* **Presentation Layer (UI Layer)** – Contains UI-related classes and components.
* Place **View** classes here.
* Handles user interactions and displays data.
* **Business Logic Layer (Application Layer)** – Contains the core functionality and processing logic.
* Place **Model** classes here.
* Implements business rules, validations, and calculations.
* **Data Access Layer (Database Layer)** – Handles interactions with the database.
* Place data handling and database interaction classes here.
* Responsible for CRUD (Create, Read, Update, Delete) operations.

By following these guidelines, the system becomes **modular, scalable, and easier to maintain**.

**Question 8:** Explain BA contributions in project (Waterfall Model – all Stages) **8 marks**

**Answer:** A **Business Analyst (BA)** plays a critical role in each stage of the **Waterfall Model**, ensuring that business requirements are well understood, documented, and implemented correctly. Below is a stage-wise breakdown of BA contributions:

1. **Requirement Gathering & Analysis**

* Key Responsibilities:
* Interacts with stakeholders to understand business needs.
* Gathers, documents, and analyses requirements.
* Prepares **Business Requirement Document (BRD)** & **Functional Requirement Specification (FRS/FSD)**.
* Creates **Use Case Diagrams, Flowcharts, and Data Models**.
* Validates requirements through stakeholder meetings.
* BA Deliverables:
* Requirement documents (BRD, SRS, FSD).
* Use case diagrams, wireframes, and mock-ups.
* Gap analysis reports.

1. **System Design**

* Key Responsibilities:
* Collaborates with system architects and designers to ensure requirements are correctly translated into the system design.
* Provides business logic and workflow details for system components.
* Reviews **System Design Document (SDD)** to validate alignment with business needs.
* BA Deliverables:
* Requirement Traceability Matrix (RTM) to track requirement fulfilment.
* Business process models and system flow diagrams.

1. **Implementation (Development)**

* Key Responsibilities:
* Assists developers by clarifying requirements.
* Ensures the development team understands functional and business logic.
* Participates in sprint reviews or development checkpoints.
* BA Deliverables:
* Updated requirement documents if any change requests arise.
* Clarification notes and business logic explanations.

1. **Testing (Verification & Validation)**

* Key Responsibilities:
* Assists in preparing **User Acceptance Test (UAT) cases**.
* Validates test cases to ensure they align with business requirements.
* Supports QA and testing teams in identifying defects related to requirements.
* Conducts UAT with end users and gathers feedback.
* BA Deliverables:
* UAT test cases and scenarios.
* Requirement validation reports and defect tracking.

1. **Deployment (Implementation & Go-Live)**

* Key Responsibilities:
* Supports business users in transitioning to the new system.
* Provides end-user training and prepares training materials.
* Ensures all business requirements are successfully implemented.
* Assists in handling post-go-live issues.
* BA Deliverables:
* Training manuals and user guides.
* Change management support.

1. **Maintenance & Support**

* Key Responsibilities:
* Monitors system performance based on business expectations.
* Gathers feedback from end-users for enhancements.
* Assists in managing system updates or patches.
* BA Deliverables:
* Enhancement request documents.
* Lessons learned reports for future improvements.

A Business Analyst (BA) ensures that the project aligns with business objectives at every stage of the Waterfall Model. Their contributions help minimize risks, improve communication between stakeholders and developers, and enhance the overall quality of the final product.

**Question 9:** What is conflict management? Explain using Thomas – Kilmann technique **6 marks**

**Answer:** Conflict management refers to the **process of handling disputes and disagreements** effectively between individuals or groups to achieve a constructive resolution. It aims to **minimize negative impacts** while maximizing **learning and collaboration**. Effective conflict management ensures better teamwork, productivity, and workplace harmony.

Different conflict management models are used based on the **situation and environment** (e.g., workplace, business, personal relationships). While **Thomas-Kilmann** is one of the most popular models, others like **Rahim’s Model, IBR Approach, Dual Concern Model, and Pondy’s Model** provide additional perspectives.

***Thomas-Kilmann Conflict Management Model***

The **Thomas-Kilmann Conflict Mode Instrument (TKI)** identifies **five conflict resolution styles** based on two dimensions:

* **Assertiveness** (how much a person tries to satisfy their own concerns).
* **Cooperativeness** (how much a person tries to satisfy the other person’s concerns).

The five conflict-handling styles are:

* **Competing (High Assertiveness, Low Cooperativeness):**
* A win-lose approach where one party pursues its goals at the expense of the other.
* Used when quick, decisive action is needed (e.g., emergencies).
* **Collaborating (High Assertiveness, High Cooperativeness):**
* A win-win approach where both parties work together to find a mutually beneficial solution.
* Best for complex issues requiring innovative solutions.
* **Compromising (Moderate Assertiveness, Moderate Cooperativeness):**
* A balanced approach where both parties make concessions to reach a middle ground.
* Used when a quick resolution is needed, and both sides have equal power.
* **Avoiding (Low Assertiveness, Low Cooperativeness):**
* A lose-lose approach where the conflict is ignored or postponed.
* Used when the issue is trivial or when emotions are too high to resolve the conflict constructively.
* **Accommodating (Low Assertiveness, High Cooperativeness):**
* A lose-win approach where one party gives in to the other’s demands.
* Used when maintaining relationships is more important than winning the argument.

The **Thomas-Kilmann model** helps individuals and organizations select the best conflict resolution style based on the situation. **Balancing assertiveness and cooperativeness** are key to effective conflict management, leading to better teamwork and decision-making.

**Question 10:** List down the reasons for project failure **6 marks**

**Answer:** Project failure can occur due to various factors, including poor planning, mismanagement, and unforeseen risks, often due to a combination of factors. Below are some of the **key reasons** for project failure:

* **Lack of Clear Goals and Objectives:**
* **Ambiguous or poorly defined project goals** can lead to confusion among stakeholders and the project team. Without a clear vision, it is difficult to determine what success looks like and to align efforts towards achieving those objectives.
* **Impact:** Misalignment of resources, delayed milestones, and missed expectations.
* **Poor Planning and Scheduling:**
* **Inadequate planning** or lack of detailed **project schedules** can result in unrealistic timelines and missed deadlines. Not allocating sufficient resources, or failure to plan for risks and contingencies, may derail the project.
* **Impact:** Delays, over-budgeting, and scope creep.
* **Ineffective Communication:**
* **Poor communication** between stakeholders, team members, and clients can lead to misunderstandings, missed requirements, and misalignment on expectations.
* **Impact:** Confusion, mistakes, duplicated efforts, and dissatisfaction.
* **Inadequate Risk Management:**
* **Failure to identify, assess, and manage risks** early in the project lifecycle can lead to unforeseen issues that are difficult to address later on. Risks such as technical challenges, scope changes, or resource shortages can cause disruptions.
* **Impact:** Unpreparedness for potential setbacks, increased costs, or project derailment.
* **Lack of Stakeholder Engagement:**
* **Limited involvement or support from key stakeholders** (e.g., sponsors, clients, or end users) can lead to insufficient resources, poor decision-making, and unaddressed expectations.
* **Impact:** Missed opportunities, delays, and misalignment with business needs.
* **Insufficient Resources and Budget:**
* **Inadequate resources (personnel, technology, equipment, etc.)** or **insufficient budget allocation** can hinder the project’s progress. If the project does not have the necessary support, it will struggle to meet its objectives.
* **Impact:** Project delays, resource shortages, inability to meet requirements.

Successful project management requires **clear goals, proper planning, risk assessment, and effective communication**. Addressing these failure points early can improve project success rates. Project failure is often a result of **poor planning, communication, resource allocation, and risk management**. To minimize the chances of failure, it’s important to focus on clear goal-setting, effective planning, constant communication, and active stakeholder involvement. By addressing these challenges proactively, the likelihood of project success increases significantly.

**Question 11:** List the Challenges faced in projects for BA **6 marks**

**Answer:** A **Business Analyst (BA)** plays a crucial role in project success, but they often face several challenges throughout the project lifecycle. Below are some of the key challenges:

* **Unclear or Changing Requirements:**
* **Stakeholders may provide vague or incomplete requirements**, making it difficult to define clear objectives.
* **Frequent changes in requirements** can disrupt project timelines and impact deliverables.
* **Solution:** Conduct thorough requirement gathering sessions and use techniques like **prototyping or Agile methodology** to handle changes effectively.
* **Lack of Stakeholder Engagement:**
* Some stakeholders may be **unavailable, uninterested, or have conflicting expectations**.
* This can lead to delays in decision-making and misalignment of project goals.
* **Solution:** Regular meetings, stakeholder mapping, and clear communication help in managing expectations.
* **Communication Barriers:**
* **Miscommunication between technical and non-technical teams** can lead to misunderstandings.
* Lack of documentation or poor requirement articulation may cause project delays.
* **Solution:** Use structured documentation methods, visual diagrams (e.g., UML, flowcharts), and conduct frequent requirement validation sessions.
* **Technology and System Constraints:**
* Sometimes the existing **IT infrastructure or system limitations** may not support the new business requirements.
* Integrating with legacy systems can be challenging.
* **Solution:** Conduct feasibility analysis early in the project and collaborate with technical teams to find suitable solutions.
* **Managing Conflicting Interests:**
* Different stakeholders may have **conflicting business priorities**.
* For example, the finance team may prioritize cost-cutting, while the operations team may demand high-end features.
* **Solution:** Facilitate discussions to align priorities and find a balanced solution.
* **Tight Deadlines and Resource Constraints:**
* Business analysts often work under **strict project timelines** with **limited resources**, making it difficult to gather and analyse requirements properly.
* **Solution:** Prioritize requirements, break them into phases, and manage scope efficiently.

A **BA must handle multiple challenges**, including **unclear requirements, communication gaps, stakeholder conflicts, and technical constraints**. Effective communication, stakeholder management, structured documentation, and risk assessment can help **overcome these challenges** and ensure project success.

**Question 12:** Write about Document Naming Standards **4 marks**

**Answer:** Document Naming Standards define a consistent structure for naming files and documents within an organization or project. Proper naming conventions help in **easy identification, retrieval, version control, and collaboration**.

***Key Elements of Document Naming Standards***

* **Descriptive and Meaningful Name:**
* The name should clearly indicate the document’s content and purpose.
* Example: **ProjectPlan\_BankingApp\_2024.docx** instead of **Plan1.docx.**
* **Use of Date and Version Number:**
* **Helps track document updates and ensures version control.**
* Example: **Requirements\_v1.0\_2024-03-05.pdf** (version 1.0 created on March 5, 2024).
* **Standard Abbreviations and Keywords:**
* Use predefined abbreviations to keep names concise.
* Example: **SRS\_Ecommerce\_v2.docx (**Software Requirement Specification for an E-commerce project).
* **Avoid Special Characters and Spaces:**
* Use underscores (\_) or hyphens (-) instead of spaces.
* Example: **Test\_Cases\_Inventory\_Management.xlsx** instead of **Test Cases Inventory Management.xlsx.**
* **Indicate Document Type:**
* Include a prefix or suffix to specify the document category
* Example: **BRD\_CRMSoftware\_v1.2.docx** (Business Requirements Document) and **UATReport\_HRSystem\_v3.pdf** (User Acceptance Testing Report).

***Example Naming Conventions***

|  |  |
| --- | --- |
| **Document Type** | **Naming Convention Example** |
| Business Requirement Document | BRD\_ProjectName\_v1.0.docx |
| Functional Specification Document | FSD\_AppName\_v2.0.pdf |
| Test Case Document | TestCases\_ModuleName\_v1.1.xlsx |
| Meeting Minutes | Minutes\_ClientMeeting\_2024-03-05.docx |

Following proper **document naming standards** improves **organization, collaboration, and version tracking**. It ensures that team members can quickly locate the right files, reducing confusion and errors.

**Question 13:** What are the Do’s and Don’ts of a Business analyst **6 marks**

**Answer:** A **Business Analyst (BA)** plays a crucial role in bridging the gap between business needs and technical solutions. Following best practices and avoiding common mistakes ensures project success.

***Do’s of a Business Analyst***

* **Understand Business Needs Clearly:**
* Conduct thorough requirement-gathering sessions.
* Use techniques like interviews, workshops, and surveys to gather insights.
* **Maintain Effective Communication:**
* Engage with stakeholders regularly to clarify requirements.
* Use clear, concise, and structured documentation.
* **Use Proper Documentation Standards:**
* Create well-structured **BRD (Business Requirement Document), SRS (Software Requirement Specification), and Use Cases**.
* Maintain version control and traceability of documents.
* **Ensure Collaboration Between Teams:**
* Act as a bridge between **business stakeholders, developers, and testers**.
* Facilitate discussions to ensure all teams are aligned.
* **Adapt to Changes and Be Agile:**
* Be open to requirement changes and manage them efficiently using Agile or Waterfall methodologies.
* Use **Requirement Traceability Matrix (RTM)** to track changes.
* **Perform Risk and Impact Analysis:**
* Identify risks related to scope, budget, and technical feasibility.
* Suggest mitigation strategies to avoid project failure.

***Dont’s of a Business Analyst***

* **Don’t Assume Requirements Without Validation:**
* Never make assumptions about stakeholder needs—always confirm details.
* Conduct proper requirement validation before proceeding.
* **Avoid Using Technical Jargon with Non-Technical Stakeholders:**
* Use **simple, business-friendly language** when communicating with stakeholders.
* Translate complex technical terms into business terms for better understanding.
* **Don’t Ignore Stakeholder Conflicts:**
* Address conflicts between stakeholders early to prevent project delays.
* Facilitate discussions to resolve differences and find a common ground.
* **Don’t Overload Documents with Unnecessary Details:**
* Keep requirement documents **precise, structured, and to the point**.
* Use visual tools like **flowcharts, wireframes, and mock-ups** to simplify complex information.
* **Avoid Poor Time Management:**
* Don’t delay requirement gathering or validation, as it impacts the overall project timeline.
* Prioritize tasks and meet deadlines effectively.
* **Don’t Overlook Testing and UAT:**
* Ensure the system meets business needs by participating in **User Acceptance Testing (UAT)**.
* Validate that all requirements are correctly implemented before deployment.

A Business Analyst must focus on clear communication, proper documentation, collaboration, and adaptability while avoiding assumptions, poor time management, and stakeholder conflicts. Following these Do’s and Don’ts ensures successful project execution.

**Question 14:** Write the difference between packages and sub-systems **4 marks**

Answer:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Package** | **Sub-System** |
| **Definition** | A logical grouping of related **classes, interfaces, and components** within a system. | A **self-contained module or component** of a larger system that performs a specific function. |
| **Purpose** | Used for **organizing** and **structuring** code for better maintainability and reusability. | Represents a **functional unit** of a system, handling a specific part of the overall process. |
| **Dependency** | Packages provide **namespace separation** but do not define system behaviour independently. | A sub-system can function as a **standalone unit** or integrate with other sub-systems. |
| **Implementation** | Found in programming (e.g., Java packages, Python modules) and UML diagrams. | Represents a **higher-level architectural component** in system design. |
| **Example** | In Java: **java.util** (utility package) containing multiple classes like **ArrayList**, **HashMap**. | In a banking system, a **Loan Management** sub-system handles all loan-related operations. |

* **Packages** are used for **code organization**, whereas **sub-systems** represent **functional units** of a system.
* Sub-systems can contain multiple packages, but **packages do not function independently** like sub-systems.

**Question 15:** What is camel-casing and explain where it will be used **6 marks**

**Answer:** **Camel-casing** is a **naming convention** in programming where multiple words are combined into a single term, and each word **after the first starts with an uppercase letter**. It is commonly used in **variable names, function names, and object names** in many programming languages.

The term **"camel casing"** comes from the way the uppercase letters in the middle of a word resemble the **humps of a camel**. When words are joined together without spaces, and each new word **begins with an uppercase letter**, it creates a **bumpy appearance**—similar to the humps of a camel’s back.

**Example of the "Humps" in Camel Casing:**

* **camelCase -** The capital "C" in the middle looks like a hump.
* **myVariableName -** The uppercase "V" and "N" form humps.
* **getUserDetails() -** The uppercase "U" and "D" create humps in the name.

This visual similarity to a camel’s hump led to the name "camel casing" in programming.

***Types of Camel-Casing***

* **Lower Camel Case (camelCase):**
* The first letter is **lowercase**, and subsequent words start with **uppercase**.
* **Example:** customerName, totalAmount, getUserDetails()
* **Usage:** Used for **variables, function names, and method names** in Java, JavaScript, Python, and C#.
* **Upper Camel Case (PascalCase):**
* The first letter of **every word is capitalized**.
* **Example:** EmployeeDetails, OrderProcessing, CalculateTotal()
* **Usage:** Used for **class names, object names, and filenames** in languages like Java, C#, and Python.

***Where is Camel-Casing Used?***

* Programming Languages: Used in Java, C#, JavaScript, Python, and Swift for naming conventions.
* Variable & Function Naming: Used where **variables** follow **lower camel case** (**camelCase**), wherein the **first letter is lowercase** and the **first letter of each subsequent word is uppercase**.
* Database Naming Conventions: Used in **column names** and **stored procedures**.
* File and Folder Naming: Used in software development projects for maintaining consistent naming.

Camel-casing improves readability and maintainability of code by following a structured naming pattern. It is widely used in programming, database management, and file naming to ensure consistency and clarity.

**Question 16:** Illustrate Development server and what are the accesses does business analyst has? **6 marks**

**Answer:** A **Development Server** is an environment used by software teams to **write, test, and debug** applications before they move to production. It allows developers to work on the codebase without affecting real users.

Business Analysts (Bas) play a crucial role in validating business requirements in this environment, ensuring the application is developed as per the specifications. However, their access to the development server is usually restricted to specific tasks.

***What is a Development Server?***

A **Development Server (DEV Server)** is part of a multi-tier deployment environment that includes:

* **Development Server (DEV)** – Used by developers to write and test code.
* **Testing/Staging Server (UAT)** – Used for quality assurance (QA) and User Acceptance Testing (UAT).
* **Production Server (LIVE)** – The final deployment environment for actual users.

***Key Features of a Development Server:***

* **Sandbox for Developers** – Allows developers to test code in a safe environment.
* **Version Control & Code Repository** – Stores different versions of code to track changes.
* **Integrated Development Tools** – Contains tools like IDEs, debugging software, and APIs.
* **Automated Builds & CI/CD Pipelines** – Helps in continuous integration and deployment.
* **Isolated from Production** – Ensures any failure in development does not affect the live system.

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

A **Business Analyst (BA)** acts as a bridge between business stakeholders and technical teams. While they do not modify code, they may require access to the development server for validation, testing, and ensuring that business needs are met.

**Types of Access a Business Analyst May Have:**

|  |  |
| --- | --- |
| **Access Type** | **Description** |
| Read-Only Access | BA can view application screens, logs, and test data but cannot modify code. |
| Testing & Validation | BA can verify if system features match business requirements. |
| User Acceptance Testing (UAT) | In some cases, BAs are involved in testing before deployment to UAT. |
| Database Access (Limited) | BA may check sample test data but does not modify the database. |
| Reporting Issues | BAs log defects and document discrepancies in requirements. |

***Why is BA Access Limited?***

* **Avoiding Accidental Changes** – Direct modification of code or data can introduce errors.
* **Security & Compliance** – Protects sensitive information from unauthorized changes.
* **Role Separation** – Development is handled by engineers, while BAs focus on business needs.

***How Does a BA Work with the Development Server?***

A Business Analyst typically interacts with the development server in the following ways:

* **Validating Business Requirements:**
* The BA ensures that the development team has implemented all functional and non-functional requirements correctly.
* They check whether user workflows match the documented Business Requirement Document (BRD).
* **Reviewing UI/UX Features:**
* If the system has a graphical interface, the BA may check **screen layouts, forms, and user navigation**.
* They compare the UI with wireframes or mock-ups provided during the design phase.
* **Participating in Internal Testing:**
* BAs often assist in **System Integration Testing (SIT)** to check if different modules work together correctly.
* They work closely with **Quality Analysts (QAs)** to validate features.
* **Supporting Developers with Business Context:**
* Developers may need clarification on **complex business rules or workflow logic**.
* BAs provide detailed explanations and documentation to resolve misunderstandings.
* **Identifying & Reporting Issues:**
* If a feature does not function as expected, the BA logs it as a **defect in an issue tracker (e.g., Jira, Azure DevOps)**.
* They provide feedback and suggest improvements before the feature moves to the **UAT (User Acceptance Testing) phase**.

**Example Scenario: BA Interacting with the Development Server**

* **Project:** Banking Application for Online Loan Processing.
* **BA Responsibility:** Ensuring loan application workflow is correctly implemented.

***BA Access & Activities on the Development Server:***

* **Checking Loan Application Screens** – Ensuring user details and loan eligibility criteria are displayed correctly.
* **Validating Business Rules** – Confirming that interest rates, loan tenure, and credit score logic match business rules.
* **Reporting Issues** – If an incorrect calculation is found in EMI estimation, the BA logs a defect.
* **Providing Clarifications to Developers** – If developers have questions about loan approval logic, the BA explains the process.

**Conclusion:**

* **A Development Server** is an essential part of software development, allowing developers to build and test code safely.
* **A Business Analyst (BA)** interacts with the development server primarily for **requirement validation, UI/UX checks, and testing.**
* BAs have **limited access** to avoid security risks and accidental modifications but play a vital role in ensuring the system meets business needs before moving to UAT and production.

**Question 17:** What is Data Mapping **6 marks**

**Answer:** **Data Mapping** is the process of **matching data fields from one system to another** to ensure seamless data transfer, transformation, and integration. It plays a crucial role in data migration, data warehousing, and API integration by ensuring data consistency and accuracy.

In simple terms, **data mapping acts as a bridge** between different data sources and destinations, ensuring that data is properly structured, formatted, and usable.

***Why is Data Mapping Important?***

* **Data Integrity & Consistency** – Ensures that data remains accurate across systems.
* **Seamless Data Integration** – Helps in connecting different applications or databases.
* **Efficient Data Transformation** – Converts data into the required format for analysis.
* **Prevents Data Loss & Errors** – Avoids missing or mismatched data during migration.
* **Enhances Decision-Making** – Ensures that organizations have **clean, structured, and accurate** data.

***Types of Data Mapping***

* **Manual Data Mapping:**
* Performed by developers **manually**, using scripts or SQL queries.
* Used when data transformation is **simple and well-defined**.
* **Example:** Writing an SQL query to extract customer data from an old system to a new CRM.
* **Semi-Automated Mapping:**
* Uses **data mapping tools** with some manual intervention.
* Allows users to **match fields visually** but requires human input for transformations.
* **Example:** Using **ETL (Extract, Transform, Load) tools** like **Talend, Informatica, or Microsoft SSIS**.
* **Automated Data Mapping:**
* Uses **AI-powered tools** to **automatically detect patterns** and match data fields.
* Common in **Big Data and Machine Learning projects**.
* **Example:** AI-driven mapping tools for large-scale cloud migrations (e.g., AWS Data Migration Service).

***Where is Data Mapping Used?***

* **Data Migration:**
* When moving data from one system to another (e.g., shifting from **an old database to a new CRM system**).
* Ensures that **field names and formats** match between old and new systems.
* Example: Moving data from a **legacy system** (older database) to a modern **cloud-based CRM**.
* **Data Integration (Connecting Different Applications):**
* Ensures **smooth communication between different software applications**.
* Common in **Enterprise Resource Planning (ERP) & Customer Relationship Management (CRM) systems**.
* **Example:** Connecting an **E-commerce platform** with a **Payment Gateway**.
* **Data Warehousing & Analytics:**
* Helps consolidate **data from multiple sources** into a **centralized data warehouse** for reporting and analysis.
* Used in **business intelligence (BI) and data analytics** for decision-making.
* **Example:** Aggregating sales data from multiple regional databases into a **global data warehouse**.
* **API Integration (Web Services):**
* Used when connecting two applications via APIs (e.g., **Google Maps API integration** in an app).
* **Example:** Integrating a **Food Delivery App** with Google Maps API.

***Steps in the Data Mapping Process***

* **Define Data Sources & Destination:**
* Identify **where the data is coming from** and **where it needs to go**.
* Example: Migrating data from **a legacy HR system** to a **modern HRMS (Human Resource Management System)**.
* **Identify Data Fields & Relationships:**
* Compare field names, data types, and relationships between source and target databases.
* Example: Mapping employee\_name to first\_name + last\_name.
* **Apply Transformation Rules:**
* Convert data into the correct format if needed.
* Example: Converting **MM/DD/YYYY to DD-MM-YYYY** for date formats.
* **Validate & Test Mapping:**
* Run test migrations to **check for errors, missing data, or incorrect mappings**.
* **Deploy & Monitor:**
* Once validated, deploy the mapped data to the new system.
* Continuously monitor for any issues and refine mappings if needed.

***Data Mapping Challenges & Solutions***

|  |  |
| --- | --- |
| **Challenge** | **Solutions** |
| Different naming conventions | Use mapping tools to standardize field names. |
| Inconsistent data formats | Apply transformation rules (e.g., date conversion). |
| Missing or incomplete data | Use data validation techniques before mapping. |
| Data duplication | Implement **deduplication** algorithms. |
| Security & Compliance | Ensure **GDPR, HIPAA, and PCI-DSS** compliance. |

**Conclusion:**

* **Data Mapping is essential** for seamless data integration, migration, and transformation.
* It ensures **data consistency, accuracy, and security** across systems.
* Used in **database migration, analytics, API integration, and ETL processes**.
* Mapping can be **manual, semi-automated, or fully automated** based on complexity.

By following structured data mapping techniques, businesses can ensure error-free, efficient, and scalable data transfers across platforms.

**Question 18:** What is API. Explain how you would use API integration in the case of your application Date format is dd-mm-yyyy and it is accepting some data from Other Application from US whose Date Format is mm-dd-yyyy **10 marks**

**Answer:** An **API (Application Programming Interface)** is a set of **rules and protocols** that allows different software applications to communicate with each other. APIs enable seamless data exchange between systems, even if they are built using different technologies.

**Types of APIs**

* **REST API** – Uses HTTP methods (GET, POST, PUT, DELETE) and returns data in JSON/XML.
* **SOAP API** – Uses XML-based messaging for secure data exchange.
* **GraphQL API** – Allows clients to fetch only the required data efficiently.
* **Webhooks** – Event-driven API calls triggered by system actions.

***API Integration in a Real-World Scenario***

**Problem Statement:**

* Your application stores dates in the format **DD-MM-YYYY** (e.g., 25-03-2025).
* Another application in the US is sending data in **MM-DD-YYYY** format (e.g., 03-25-2025).
* You need to **integrate the API and ensure correct date format conversion**.

**Solution:**

When receiving data via API, convert the **MM-DD-YYYY format** into **DD-MM-YYYY format** before storing it in your system.

***API Integration Steps with Date Format Handling***

* **Step 1: API Request from US System**
* The US-based application sends data in **JSON format** with MM-DD-YYYY dates.
* **Step 2: API Endpoint to Receive Data**
* Your application creates an **API endpoint** to accept the data.
* What This Code Does:
* Accepts JSON data via API.
* Converts MM-DD-YYYY to DD-MM-YYYY using Python’s datetime module.
* Returns the corrected data.
* **Step 3: API Response after Date Conversion**
* After processing, the API returns the converted data.
* **Step 4: Storing Data in the Database**
* Once the date is corrected, store it in the database using SQL.

*Database now stores the correct date format (DD-MM-YYYY).*

***Alternative: Middleware for Automatic Date Conversion***

If the integration is complex, use an **API Gateway or Middleware (e.g., AWS API Gateway, MuleSoft, Zapier)** to handle automatic date format conversions before storing data.

**Benefits:**

* Reduces backend development efforts.
* Ensures all API requests conform to the required date format.

**Conclusion:**

**APIs enable seamless communication** between different applications.

* **Date format conversion is crucial** when integrating applications from different regions.
* **Using Python (Flask) or Middleware** ensures smooth data exchange and consistency.
* **Correctly formatted data prevents errors** in storage and reporting.