**Capstone 3 | Part 1**

**Case Study: A customer can make a payment either by Card or by Wallet or by Cash or by Net banking**

**Answer 1 Use Case Diagram**



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

1. **Entity Classes: The Bedrock of Your System's Data**

Entity classes are the foundation for your system's data. They encapsulate real-world objects or concepts, serving as blueprints that define the data structure and potentially some behaviour. Each entity class instance represents a unique entity within the system.

* **Attributes:** These are the properties that define an entity. For instance, a customer entity class might have attributes like name, email, address, and phone number.
* **Relationships:** Entities can have relationships with each other. These relationships can be modelled using associations in the class diagram. For example, a customer entity might have a "one-to-many" relationship with an Order entity class, indicating a single customer can have multiple orders.
* **Behaviour:** While primarily focused on data, entity classes can encapsulate some simple behaviour related to their data. A Product entity class might have a method to calculate its discounted price based on a discount rate.

1. **Boundary Classes: The Interface to the Outer World**

Boundary classes act as the system's gateway, handling interactions with external actors like users and other systems. They play a crucial role in translating these external interactions into a format the system understands.

* **User Interaction:** Boundary classes handle user actions such as clicks, form submissions, or API requests. They interpret these actions and extract relevant data.
* **Technology Independence:** Boundary classes should be agnostic to the specific UI technology or communication protocol being used. This allows for easier integration with different UI frameworks or external systems without affecting the core functionalities.
* **Examples:** a **UserLoginController class** might be a boundary class responsible for receiving username and password from a login form and converting them into a user authentication request for the system.
* Similarly, a **ProductAPIService class** could handle incoming product data feeds from a supplier system and translate them into a format compatible with your internal product data structure.

1. **Controller Classes (or Use Case Classes): The Orchestrators of System Logic**

Controller Classes, also known as Use Case classes, are the workhorses behind user interactions and system functionalities. They take center stage in coordinating the flow of a specific use case within the system.

* **Use Case Focus:** Each controller class typically maps to a specific use case in the system. A use case is a sequence of actions that delivers a specific value to an actor (user or external system). For instance, an **OrderProcessingController class** would handle the "Place Order" use case.
* **Interaction with Other Classes:** Controller classes receive requests from boundary classes. They then interact with entity classes to perform operations on data (e.g., creating, updating, deleting entities). Additionally, they might call upon other controller classes for complex workflows that involve multiple use cases.
* **Business Logic:** The core business logic of the system resides within controller classes. They implement the rules and processes that govern how data is manipulated and use cases are fulfilled.

**Advantage of Categorization:**

* **Loose Coupling:** Changes in UI technology or communication protocols only affect boundary classes, while the core functionalities remain protected within entity and control classes. This promotes easier maintenance and future modifications.
* **Reusability:** Entity classes can be reused across different use cases, and controller classes can be designed with modularity in mind, allowing for code reuse in various functionalities.
* **Testability:** Isolating functionalities makes it easier to write unit tests for each layer, ensuring the reliability of your system.

**Analogy: Restaurant in Action**

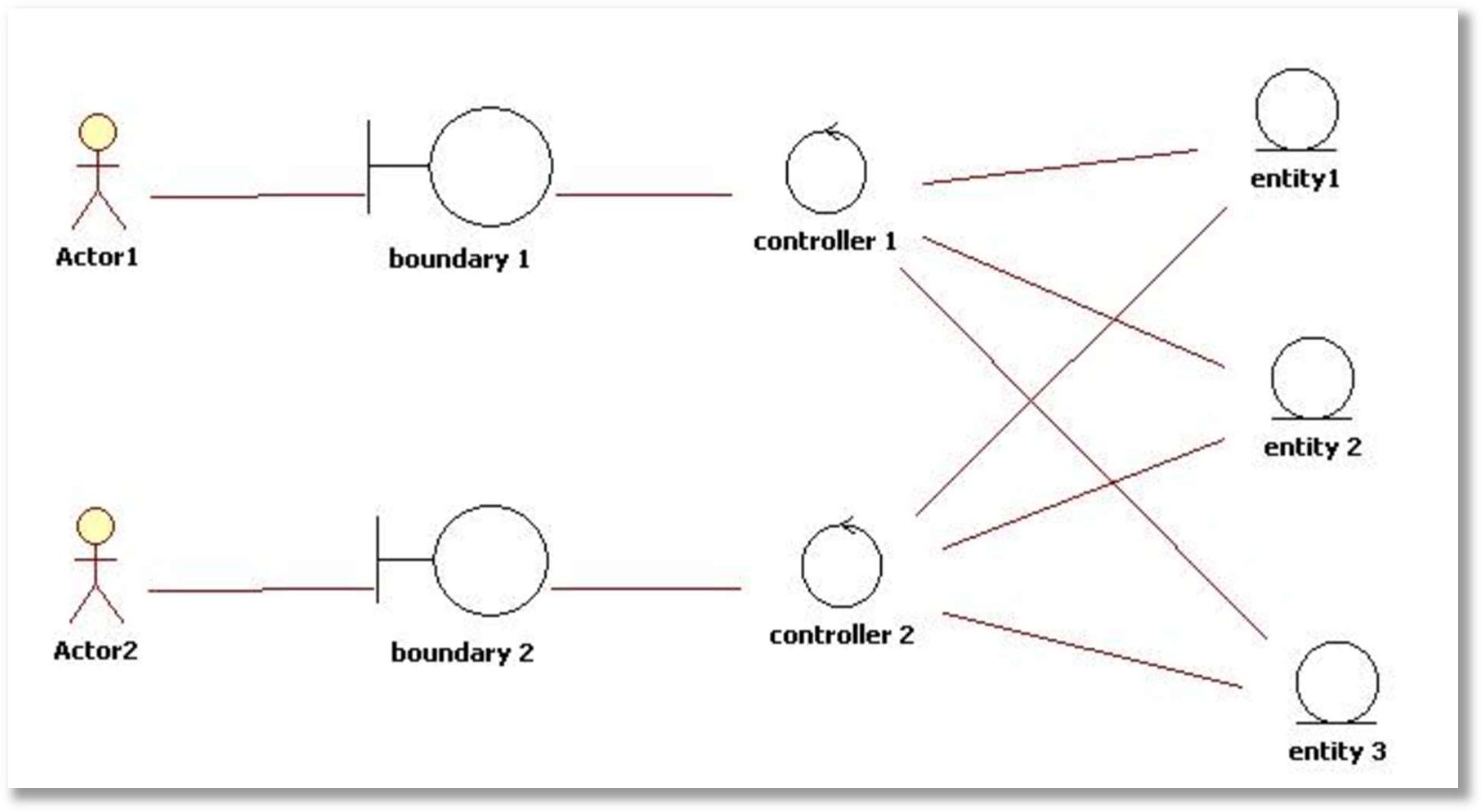
**Imagine a restaurant:**

1. **Entity Classes:** Menu items (food and drinks) with attributes like name, price, and ingredients represent entities.
2. **Boundary Classes:** The waiters who take orders and present the menu act as boundary classes. They translate between customer requests and the kitchen.
3. **Controller Classes (Chefs):** The chefs function as controller classes. They receive orders from waiters (boundary classes), interact with ingredients (entity classes) to prepare the food (perform operations), and might collaborate with other chefs for complex dishes (calling other controller classes).

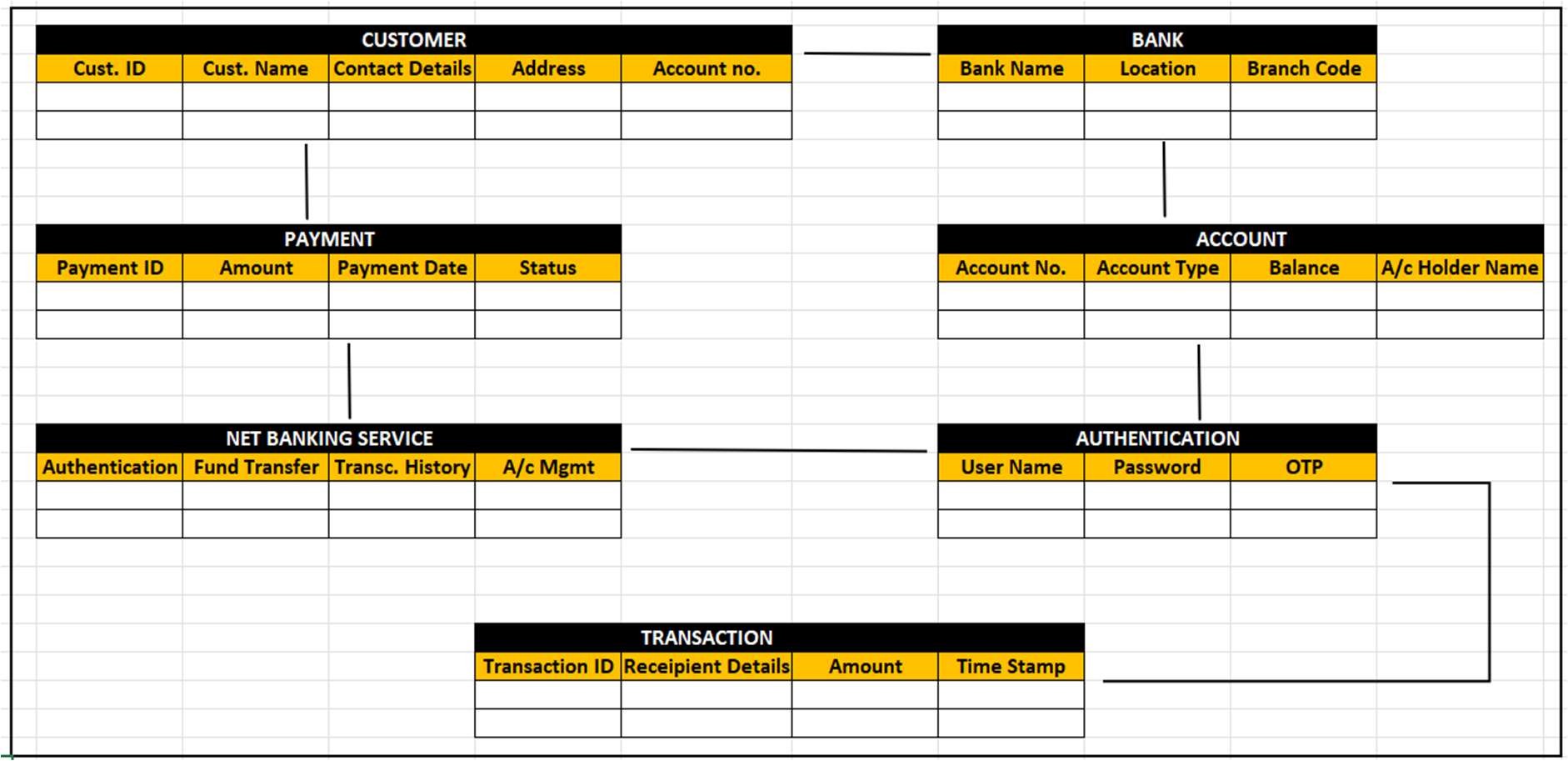
**Question 3- Place these classes on a three tier Architecture.**

1. **Presentation Layer:**
   * **Boundary Classes:** Reside in the Presentation Layer. They directly interact with the user interface (UI) or external systems. They receive user input, format data for the system, and present system responses back to the user interface.
2. **Business Logic Layer:**
   * **Controller Classes (Use Case Classes):** These classes live in the Business Logic Layer. They orchestrate the system's core functionalities, implementing business rules and logic. They interact with boundary classes to receive requests and interact with entity classes to perform operations on data.
3. **Data Access Layer:**
   * **Entity Classes:** Entity classes reside in the Data Access Layer. They represent the data model of the system and encapsulate data structures and relevant behaviour. They interact with the database or persistence mechanism through Data Access Objects (DAOs) to store, retrieve, and update data.

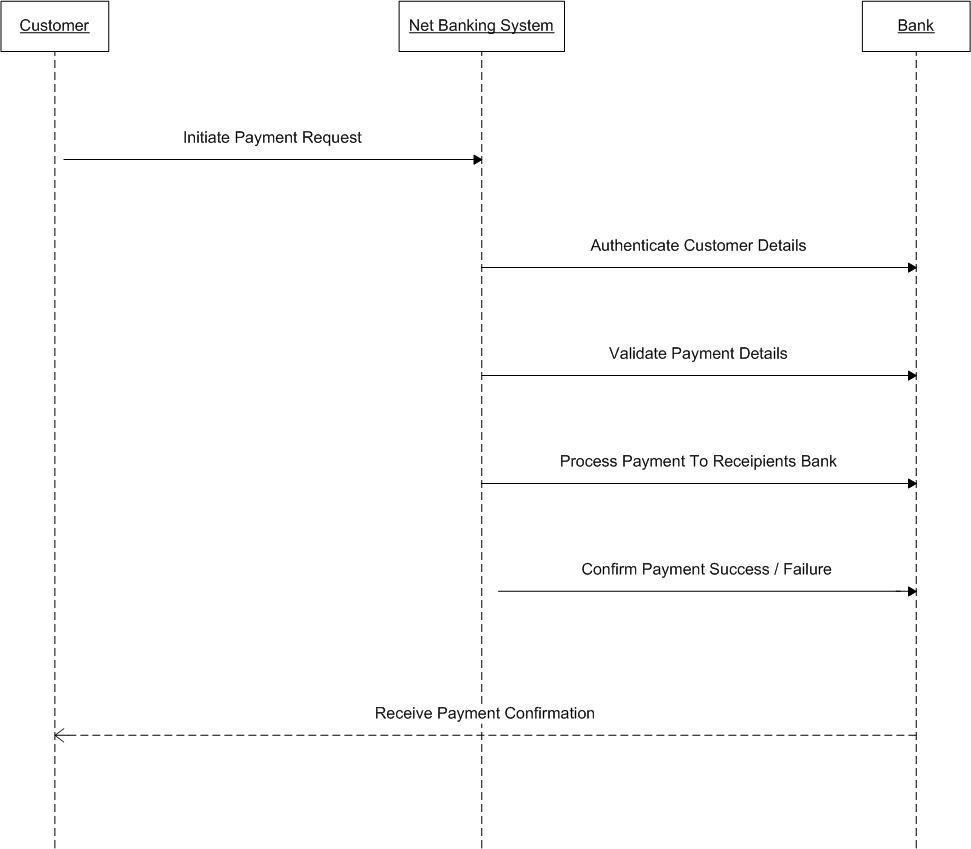
|  |  |  |
| --- | --- | --- |
| **Layer** | **Classes** | **Responsibilities** |
| **Presentation**  **Layer** | Boundary Classes | User interaction, data formatting, UI  presentation |
| **Business Logic** | Controller Classes (Use Case Classes) | Business logic implementation, use case  orchestration, interaction with other layers |
| **Data Access** | Entity Classes | Data modelling, data access through DAOs |



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



**Question 5- Sequence Diagram for Payment done by Customer**



**Question 6- Explain Conceptual Model for this**

The Conceptual Model outlines the key steps involved in a customer making a payment through net banking. It focuses on the flow of information and actions between the customer, their bank, and the merchant (the business receiving the payment).

**Components:**

* **Customer:** The person initiating the online payment.
* **Customer's Bank:** The financial institution where the customer holds an account.
* **Merchant's Bank:** The financial institution where the merchant holds an account (may differ from the customer's bank).

**Process Flow:**

1. **Initiate Payment:** The customer visits the merchant's website or app and chooses net banking as the payment method. They enter the order details and payment amount.
2. **Customer Login:** The customer is redirected to their bank's secure net banking login page. They enter their login credentials (username and password).
3. **Payment Details and Verification:**
   * The customer verifies the payment details, including the recipient (merchant), amount, and possibly the order details.
   * Some banks might require an additional security step like a one-time password (OTP) sent via SMS or a security question.
4. **Authorization:**
   * The customer confirms the payment by clicking a submit button or similar action.
   * This transmits the payment request and customer information to their bank for authorization.
5. **Funds Transfer:**
   * The customer's bank verifies the account details and available balance.
   * If sufficient funds are available and all security checks pass, the bank initiates the transfer of funds.
   * In some cases, the payment gateway may be involved, facilitating the transfer between banks.
6. **Payment Confirmation:**
   * The customer's bank sends a confirmation message to the customer, indicating successful payment.
   * The merchant may also receive a notification of the successful payment.

**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.**

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

* **Model:** Represents the application's data and business logic. It encapsulates the data, data access logic, and business rules. The model interacts with databases or other data sources to manage and retrieve data.
* **View:** Handles the presentation layer, responsible for displaying information to the user. It uses data from the model to generate the user interface (UI) elements and receives user input.
* **Controller:** Acts as the intermediary between the model and the view. It receives user input from the view, interacts with the model to process data or perform actions, and updates the view based on the model's state.

**MVC Benefits:**

* **Separation of Concerns:** Makes code more modular, easier to understand, maintain, and test. Developers can focus on specific aspects without worrying about the intricate details of other parts.
* **Reusability:** The model can be reused across different views, and the view can be modified without affecting the model's logic.
* **Testability:** Each component is more isolated, making unit testing more efficient.
* **Improved Maintainability:** Changes to one component are less likely to break other parts of the application.

**Deriving Classes from Use Case Diagrams**

Use case diagrams depict interactions between actors (users or external systems) and the system being developed. Here's how to use them to identify classes for an MVC application:

1. **Identify Actors:** Each actor in the diagram maps to a class in your application that represents the user or external system.
2. **Identify Use Cases:** Each use case translates to a set of interactions between the actors and the system.
3. **Model Classes:** Use cases that involve manipulating data point to model classes. These classes encapsulate data structures and business logic related to the use case.
4. **View Classes:** Use cases that involve displaying information lead to view classes. These classes are responsible for presenting data in a human-readable format on the UI.
5. **Controller Classes:** Use cases that require user interaction and data processing result in controller classes. These classes receive user input, interact with model objects, and update the view based on the model's state.

**Example:**

Imagine a use case where a user searches for products in an online store. This would translate to the following classes:

* + **Model:** Product class representing product data (name, description, price, etc.)
  + **View: SearchResultsView** class showing product search results
  + **Controller: ProductSearchController** class handling user search queries and updating the **SearchResultsView** with results from the Product model

**3-Tier Architecture (Presentation, Business Logic, Data Access)**

A 3-tier architecture separates an application into three logical tiers:

* **Presentation Tier:** The UI layer that handles user interaction and displays information. This aligns with the MVC view component.
* **Business Logic Tier:** Encapsulates the application's core logic and business rules. This corresponds to the MVC model component and some controller logic.
* **Data Access Tier:** Manages interaction with data sources (databases, files, etc.). This is often implemented outside the MVC architecture directly within the model component or in a dedicated data access layer.

**Placing Classes in a 3-Tier Architecture:**

* The **presentation tier** primarily consists of view classes from MVC.
* The **business logic tier** includes model classes from MVC and any additional classes containing business rules.
* The **data access tier** may involve separate classes responsible for interacting with data sources, or this functionality may be embedded within the model classes.

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

|  |  |  |
| --- | --- | --- |
| **Waterfall Stages** | **BA Activities** | **Artifacts / Resources** |
| Pre Project |  | Business Case |
| Enterprise Analysis: SWOT, GAP, Market Research, Feasibility Study, Root Cause Analysis, Decision Analysis, Strategy Analysis, Project Scope & Business Case writing, Risk Analysis | Statement of Work Purchase Order (PO) Team- Sr. BA, Business |
| Architects, Presales |
|  | Consultants |
| Planning, Estimations and Assessments | i.  Understanding assumptions and constraints along with business rules and business goals |  |
| ii.  Plan packages for big projects |  |
| iii.  Understand the project plan from PM | Team- PM & Sr. BA |
| iv.  BA conducts Stakeholder Analysis |  |
| v.  Plan BA Approach Strategy (req. gathering techniques, communication, req. mgmt. documents to follow, tools to |  |
| use, change request methodology) for the project |  |
| Requirement Gathering | i.  Stakeholders identify and document |  |
| ii.  Client gives BRD or BA prepares BRD by interacting with Client - Brainstorming, Document Analysis, Reverse engineering, Interviews, workshops, Focus Groups, Observation, Questionnaires. | BRD |
| iii.  Prototyping can be used by BA to make the Client to give more specific requirements | Team- BA, PM |
| iv.  Sort the gathered Requirements (avoiding duplicate Reqs, grouping into similar functionality or into modules) |  |
| v.  Prioritize requirements - MoSCoW |  |
| vi.  Validate Requirements - FURPS |  |
| Requirement Analysis | i.  Draws UML Diagrams (Use case and Activity Diagrams) | Functional Requirements Specification |
| ii.  Prepares Functional Requirements from Business Requirements | SSD (Supplementary Support Document) |
| iii.  All Architects comes up with Technical Requirements (SSD) | SRS (Software Requirements Specification) |
| iv.  SRS will have Functional Requirements and Technical Requirements | RTM (Requirements Traceability |
| v.  Takes Signoff on SRS from Client. SRS is the first legal binding Doc between the Business and the technical Team | Matrix) |
| vi.  BA prepared RTM from SRS before Design phase starts. (BA is the owner of RTM). |  |
| vii.  BA traces how requirements are dealt in each phase of development life cycle from Design till UAT | Team- BA, PM, Solution Architect, DB- Architect, NW- |
|  | Architect |
| Design | i.  From Use case Diagram, Test Manager or BA will prepare Test Cases |  |
| ii.  Communicates with Client on the design and Solution documents (updates Status to Client and make them understand how the solution would look like to prepare them to drive UAT) |  |
| iii.  BA will initiate the preparation of End user manuals | Solution Document |
| iv.  updates RTM | Design Document - HDD - ADD |
| v.  From Use case Diagram Solution-Architect recommends Architecture of the IT solution | Team- BA, PM, Solution Architect, DB- Architect, NW- Architect, GUI Designer, Test Manager |
| vi.  DB Architect uses Persistence Classes (Entity Classes) and comes up with ER Diagrams or DB Schema. |  |
| vii.  GUI Designer will look into Transient Classes (Boundary Classes) and designs all possible Screens for the IT Solution’s |  |
|  |  |
| Coding | i. BA organizes JAD Sessions |  |
| ii.   BA clarifies queries of Technical Team during Coding |  |
| iii.   Developers refer Diagrams and Transient (Controller Classes) of BA and code their unit | LDD - CDD |
| iv.   Update End user manuals | Application |
| v.   Update RTM |  |
| vi.   Conducts regular Status meetings with technical team and the Client and tuning Client for participation in UAT | Team- BA, Dev. Team, PM |
| Testing | i. BA- Prepares Test Cases from Use Cases or assists Test Manager to do so |  |
| ii.   BA performs high level testing |  |
| iii.   BA prepares Client for UAT | Test Concerning Documents Application with less errors |
| iv.   Test Data is requested by BA from Client |  |
| v.   Updates End User Manuals | Team- Testing Team, BA, PM, Client |
| vi.   Updates RTM |  |
| vii.    Take signoff from Client on Client Project Acceptance form |  |
| Deployment and Implementation | i.  Forwards RTM to Client or the PM which should be attached to the Project Closure Document |  |
| ii.  Coordinates to complete and share End User Manuals |
| iii.  Plans and Organizes Training Sessions for End Users |
| iv.  Prepares Lessons learned from this project (to take precautions for coming projects) |

**Question 9- What is conflict management? Explain using Thomas – Kilmann Technique**

**Conflict management** is the process of effectively handling disagreements and disputes to minimize negative outcomes and reach mutually beneficial solutions. It involves a set of skills and strategies that can be applied in various settings, from personal relationships to professional environments.

**The Thomas-Kilmann Conflict Mode Instrument (TKI)** is a popular tool used to understand and improve conflict management styles. Developed by Kenneth Thomas and Kenneth Kilmann, the TKI identifies five primary conflict handling modes:

1. **Competing (Competitive):** This style focuses on winning the conflict and getting your way. You assert your position strongly and may use forceful tactics to achieve your goals. While effective in urgent situations, it can damage relationships and lead to resentment.
2. **Accommodating (Accommodating):** This approach prioritizes maintaining harmony and avoiding conflict. You readily concede to the other party's wishes to preserve peace. This can be helpful in fostering cooperation but might leave you feeling unheard or disrespected in the long run.
3. **Avoiding (Avoiding):** This style involves withdrawing from the conflict or postponing dealing with it. You might ignore the issue or change the subject. While it can provide temporary relief, it doesn't resolve the underlying problem and can create further tension.
4. **Collaborating (Collaborative):** This approach seeks a win-win solution that satisfies everyone's needs. You work with the other party to find a mutually beneficial outcome through open communication and problem- solving. This requires effort and time but leads to the most sustainable solutions.
5. **Compromising (Compromising):** This style involves finding a middle ground by giving up some things on both sides. It's faster than collaboration but might not fully address everyone's concerns. It can be a good option when time is limited or complete agreement isn't feasible.

# **Using the TKI:**

The TKI helps you identify your natural conflict handling style and understand how others approach conflict. This method allows you to:

* + Choose the most appropriate style for a specific situation.
  + Adapt your approach to better interact with someone with a different style.
  + Develop a more collaborative and assertive communication style for effective conflict resolution.

By understanding different conflict modes, you can navigate disagreements more constructively, build stronger relationships, and achieve more positive outcomes.

**Question 10- List down the reasons for project failure**

1. Improper requirement gathering
2. Continuous change in requirements
3. Lack of user involvement
4. Lack of executive support
5. Unrealistic expectations
6. Improper planning

**Question 11- List the Challenges faced in projects for BA**

1. Lack of training
2. Obtaining sign-off on requirements
3. Change Management- wrt. Cost and timelines
4. Coordination between developers and testers
5. Conducting meetings
6. Making sure status reporting is effective
7. Driving clients for UAT completion
8. People management (coordination with different people and different teams)
9. Overall making sure project health is in good shape and delivered as per the timelines without any issues.

**Question 12- Write about document naming standards**

Effective document naming standards are crucial in IT environments for efficient file organization, retrieval, and collaboration. Clear and consistent naming practices save time, prevent confusion, and ensure everyone can easily find the information they need.

**Key Principles**

1. **Descriptiveness:** File names should clearly describe the document's content. Include keywords that accurately reflect the information within.
2. **Consistency:** Use a consistent format throughout your organization. This could involve a combination of elements like:
   * **Project Name or Identifier:** Identify the project or system the document relates to (e.g., "VPN\_Configuration").
   * **Document Type:** Indicate the type of document (e.g., "Report," "User Guide," "Procedure").
   * **Version Number:** Use version numbers (e.g., "v1.2") to track changes and identify the latest version.
   * **Date:** Include dates for time-sensitive documents or if versioning isn't the primary focus (e.g., "2024-06-17").
3. **Conciseness:** While descriptive, keep names concise to avoid overly long file paths. Use abbreviations judiciously, ensuring they are commonly understood within your team.
4. **Clarity:** Avoid special characters, symbols, or spaces (use underscores instead). This ensures compatibility across different operating systems and avoids potential file path issues.
5. **Standardized Abbreviations:** If using abbreviations, define and document them for everyone's reference.

# **Benefits of Effective Document Naming:**

* + **Improved Search and Retrieval:** Easy-to-understand file names allow for faster searching and locating of documents.
  + **Enhanced Collaboration:** Consistent naming facilitates teamwork and knowledge sharing, as everyone knows where to find and store documents.
  + **Reduced Confusion:** Clear file names minimize ambiguity and misidentification of documents.
  + **Increased Efficiency:** Time saved searching for files translates to improved productivity.
  + **Better Compliance:** Consistent naming can support compliance with data retention and information governance policies.

**Document name std. reference**

**[Project ID][Document Type]V[x]D[y].ext**

**Question 13- What are the Do’s and Don’ts of a Business Analyst**

* + Never say NO to Client
  + There is NO word called as "BY DEFAULT"
  + Never imagine anything in terms of GUI
  + Question the existence of existence / question everything in the world ex: what client gives is not always correct
  + Consult an SME for Clarifications in Requirements
  + Every Problem of Client is unique. No two problems of different Client are same. May be the approach, technology, place of use, local laws may be varied to make them (Problems) to be different.
  + Go to Client with a plain mind with no assumptions. Listen carefully and completely until Client is done and then you can ask your Queries. Please do not interrupt the Client, when he/ She is giving you the problem. Maximum Try to extract the leads to Solution from the Client itself. Never try to give Solutions to Client straight away with your previous experience and assumptions. Try to concentrate on the important and truly required Requirements. Don't be washed away by add on functionalities or don’t imagine solutions of screen basis.

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

|  |  |  |
| --- | --- | --- |
| **Feature** | **Packages** | **Subsystems** |
| **Purpose** | Organize related code elements for reusability and namespace management | Decompose a complex system into functional units that deliver specific  features or capabilities |
| **Focus** | Code reusability and modularity,  promoting maintainability and collaboration | Delivering well-defined functionality within a larger system context |
| **Granularity** | Smaller unit, often encompassing a single  concept or task | Larger unit, composed of multiple packages  and potentially other subsystems |
| **Instantiation** | Individual elements (classes, functions)  within a package can be instantiated and used independently | Subsystems are typically instantiated as a  whole unit to provide their complete functionality |
| **Example** | A package containing database access functions or utility functions for string manipulation | A subsystem for user authentication, handling login, registration, and authorization logic, or a payment processing subsystem that interacts with  payment gateways |
| **Relationship** | Packages can be nested within other packages to create a hierarchical structure, promoting organization and code reuse  across different parts of the system | Subsystems can have dependencies on other subsystems, forming a network of interconnected functionalities that work  together to achieve the overall system goals |
| **Development Stage** | Packages are often established during the initial coding phase to group related functionalities | Subsystems are typically defined during the system design phase to break down the system into manageable functional  components |

**Question 15- What is Camel Casing**

Camel case, also sometimes stylized as camelCase or CamelCase, is a naming convention for writing phrases without spaces or punctuation, where each word starts with a capital letter (except for the first word, which can be lowercase or uppercase depending on the specific variation).

# **Key Points**

* **Format:** Phrases are written together, with the first letter of each word (except potentially the first) capitalized.
* **Examples:** numberOfDonuts, emailAddress, fileInputStream, userName (lowercase first word), HTTPRequest (uppercase first word)

# **Variations:**

* **Lower camel case (also known as dromedary case):** The first word starts with a lowercase letter (e.g., userName). This is the most common variation in programming languages.
* **Upper camel case (also known as Pascal case):** The first letter of the entire phrase is capitalized (e.g., HTTPRequest). This is typically used for class names in some programming languages.

# **Uses:**

* + **Programming Languages:** Camel case is widely used for naming variables, functions, methods, and classes in programming languages like Java, JavaScript, Python, C++, and many others. It improves readability by clearly separating words without relying on spaces or underscores.
  + **File Names:** Camel case can be used for file names, particularly in web development, to enhance readability and consistency.

# **Benefits:**

* + **Readability:** Camel case makes code and file names easier to read and understand compared to using underscores or other separators.
  + **Consistency:** Using a consistent naming convention like camel case promotes code maintainability and collaboration within a team.
  + **Automatic Tools:** Many programming languages and development tools are designed to work well with camel-cased names, providing auto-completion and other features.

# **Choosing Between Lower and Upper Camel Case:**

The specific choice between lower camel case and upper camel case often depends on the programming language or coding style guide being followed. However, here's a general guideline:

* + **Lower camel case:** Commonly used for variables, functions, and methods.
  + **Upper camel case:** Typically used for class names.

By understanding camel case and its variations, you can effectively name your code elements, improving readability, maintainability, and consistency in your software development projects.

Question 16- Illustrate Development server and what are the accesses does business analyst has?

A development server refers to a dedicated environment or server that is used during the software development process. It provides a platform for developers and testers to build, test, and debug applications before they are deployed to a production environment. The development server typically replicates the target production environment to ensure compatibility and accurate testing.

# **Access Types**

* + Read-Only Access
  + Collaborative Access
  + Limited Configuration Access

**Question 17- What is Data Mapping**

Data mapping is the process of defining how data elements from one source (system, database, etc.) correspond to data elements in another source. It's like creating a translation guide to ensure information flows smoothly between different systems.

# Key points about data mapping:

* + **Purpose:** Ensures data accuracy, consistency, and usability when transferring or integrating data from different sources.

# **Benefits:**

* **Improved Data Quality:** Minimizes errors and inconsistencies that can arise when data formats or structures differ.
* **Enhanced Data Integration:** Creates seamless data flow between systems, enabling better data analysis and decision-making.
* **Increased Efficiency:** Saves time and effort by avoiding manual data transformation and cleaning.
  + **Process:** Typically involves identifying data elements in both sources, matching them based on content or structure, and defining any necessary transformations (e.g., changing data formats, units of measurement).

# Common Data Mapping Techniques:

* + **Manual Mapping:** Done directly by data analysts or developers, often for smaller datasets or simple mappings.
  + **Data Mapping Tools:** Specialized software helps automate the process, especially for complex mappings or large datasets. These tools can identify potential matches, suggest transformations, and streamline the overall mapping process.

# Use Cases for Data Mapping:

* + **Data Migration:** Moving data from one system to another (e.g., migrating customer data from a legacy system to a new CRM).
  + **Data Warehousing:** Combining data from various sources into a central repository for analysis.
  + **Data Integration:** Enabling different systems to communicate and share data seamlessly.
  + **ETL (Extract, Transform, Load):** Extracting data from various sources, transforming it to a consistent format, and loading it into a target system (e.g., data warehouse).

**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.**

An API is a set of definitions and protocols that allow applications to communicate with each other. It acts as an intermediary, enabling them to exchange data and functionality in a standardized way. Here's an analogy:

Imagine you have two different restaurants (applications) that want to share customer orders (data). An API would be like a waiter (mediator) who takes orders from one restaurant (source), translates them into a format the other restaurant (target) understands, and delivers them efficiently.

# **API Integration**

API integration is the process of connecting your application with another application's API to share data or functionality. This allows you to leverage the capabilities of the other application within your own.

# Your Scenario: Date Format Conversion with API Integration

In your specific case, your application (with dd-mm-yyyy format) needs to accept data from another application (US- based with mm-dd-yyyy format). Here's how you can use API integration to handle the date format conversion:

# Identify the Other Application's API:

* + Check if the US application provides an API for data exchange.
  + If so, consult their API documentation to understand the data format they expect and provide instructions for data transfer.

# Date Format Conversion Logic:

* + Implement logic within your application to convert the received mm-dd-yyyy dates to your desired dd-mm-yyyy format. Libraries or built-in functions in your programming language can often be used for this task (e.g., datetime library in Python).

# Integration Approach (Choose one based on your application's architecture):

* + **Direct API Integration:**
    - If your application can directly connect to the other application's API, you can make API calls to retrieve data and perform the date conversion before storing it in your system.
    - This approach requires coding within your application to handle API calls, data parsing, and date conversion.

# Middleware Integration:

* + - If your application doesn't directly interact with external APIs, you might use a middleware server. This server would act as an intermediary, receiving data from the other application, performing date conversion, and then delivering the converted data to your application.
    - This approach requires setting up and managing the middleware server, but it can decouple your application from the specifics of the other application's API.