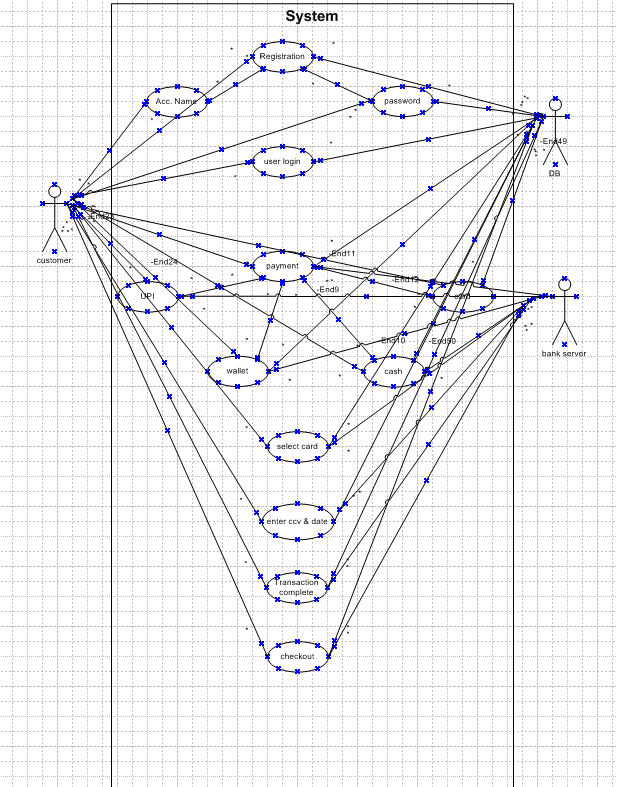
**Capstone Project3– Part -1/2**

**Q1. Draw a Use Case Diagram**

**ANS:**



**Q2**

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

**ANS:**

Boundary classes:

Boundary classes represent the system's interaction with external elements like users, devices, or other systems. They are responsible for managing user input/output and are often related to the **User Interface (UI)** or the **API layer** of a system. These classes facilitate the flow of information into and out of the system.

|  |  |
| --- | --- |
| Primary actors- Who initiate the use cases and interact with the system  Boundary class – (Use Cases) Actors speak to the system (Authentication information)  All the actors should be – Primary Actors | Customer registration  Customer login  Bank server logs in  Customer logout  Bank Server Logs out |

Controller classes:

Controller classes act as intermediaries between the boundary classes (UI/External Systems) and the entity classes (business logic). They handle user input, coordinate the system’s behaviour, and update the model accordingly. Controller classes are responsible for the orchestration of different components.

|  |  |
| --- | --- |
| controller class – (Handles Users (Primary actors) input and processes the data  Use case will consider as Controller class system | Registration Controller  Login Controller  Payment Controller  Credential controller |

Entity Classes:

Entity classes represent the core business objects of the system. They model the real-world concepts or entities that are central to the application. These classes are typically persistent (can be stored in a database) and contain the main business logic and attributes related to the core objects of the system.

|  |  |
| --- | --- |
| Each actor will be considered as an  entity | Customer  Bank server  Cash  Card  Net Banking |

**Q3**

**Place these classes on a three tier Architecture**

**ANS:**

**1. Presentation Layer (Boundary Classes)**

The **Boundary Classes** represent the interaction points between the system and its primary actors. They capture inputs from users, display outputs, and pass control to the controller classes for processing.

|  |  |
| --- | --- |
| **Registration UI** (Customer Registration) | * Input: Customer details (e.g., name, email, password). * Output: Success or error message |
| **Login UI** (Customer Login) | * Input: Customer login credentials (username, password). * Output: Login success or failure message. |
| **Payment UI** (Payment Process) | * Input: Payment details (amount, payment method). * Output: Payment success or failure. |
| **Credential UI** (Bank Server Login) | * Input: Bank server credentials. * Output: Bank server authentication result. |

2. **Business Logic Layer (Controller Classes):**

The **Controller Classes** process the data and business logic. They are responsible for interpreting user input from the boundary classes, invoking appropriate operations on the entities, and returning results.

|  |  |
| --- | --- |
| Registration Controller | * Receives customer details from Registration UI. * Validates and registers the customer (creates Customer entity). * Returns success or failure to Registration UI. |
| Login Controller | * Receives login credentials from Login UI. * Authenticates the credentials by checking the stored customer data. * Returns login success or failure to Login UI. |
| Payment Controller | * Receives payment details from Payment UI. * Validates the payment method and processes the payment. * Returns payment status (success or failure) to Payment UI |
| Credential Controller | * Receives bank server credentials from Credential UI. * Authenticates the bank server. * Returns success or failure to Credential UI |

**3. Data Layer (Entity Classes)**

The **Entity Classes** represent the core business objects. They store data and encapsulate the main business logic related to the domain. These entities are responsible for holding customer information, payment details, and transaction data.

|  |  |
| --- | --- |
| Customer | Represents a customer in the system. |
| Bank Server | Represents the bank server in the system that handles login/logout operations. |
| Payment | Represents a payment made by the customer. |
| Payment Method | Abstract base class for different payment methods (Card, Cash, Net Banking). |
| Card | Represents a payment made via a credit/debit card. |
| Cash | Represents a payment made in cash. |
| Net Banking | Represents a payment made through net banking. |

**Q4.**

**Explain Domain Model for Customer making payment through Net Banking**

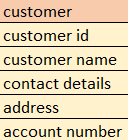
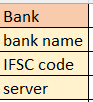
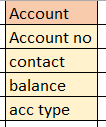
**ANS**:

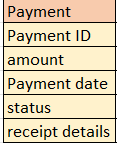
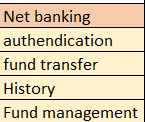
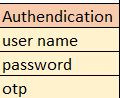
A **Domain Model** is a conceptual representation of the entities, their attributes, and the relationships between them within a particular domain of interest. It is used in software engineering to represent the knowledge and behaviour of a domain. This model helps to identify the core concepts and their interrelationships before developing a system. It serves as a foundation for developing object-oriented systems, guiding how the system will function. Domain model is similar to the entity relationship model. The tables are connected to each other. In the below diagram, the customer table is connected to bank table, which is why the customer is able to make payment.

Customer table is also connected to payment table, because he should make the payment. Now the payment is done by net banking, so payment table is connected to net banking table.

The account is in the bank, so the account table is connected to the bank table. The authentication table is connected to both net banking table and bank table, because authentication is to be performed there.

Also, the authentication table is connected to transaction table, because authentication will be done while transaction.

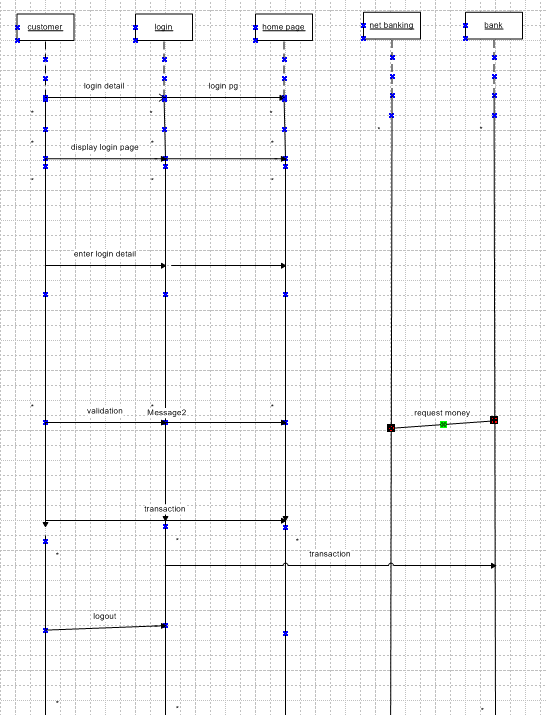
  



**Q5**

**Draw a sequence diagram for payment done by Customer Net Banking**

**ANS:**

****

**Q6**

**Explain Conceptual Model for this Case**

**ANS:**

A **Conceptual Model** is an abstract representation of the key components and relationships in a system, often serving as a high-level map of how different elements interact. It focuses on the core entities, their attributes, and how they are linked together without getting into technical implementation details.

**Entities and Their Relationships:**

**1. Customer:**

* **Attributes**:
  + Customer Id: Unique identifier for the customer.
  + name: Name of the customer.
  + email: Email address of the customer.
  + address: Address for the customer (for invoicing or delivery).
  + Payment History: List of past payments made by the customer.
* **Relationship**:
  + A **Customer** can make multiple **Payments** (One-to-Many relationship).

**2. Payment:**

* **Attributes**:
  + Payment Id: Unique identifier for the payment.
  + amount: The total amount for the payment.
  + Payment Date: The date and time when the payment was made.
  + status: The current status of the payment (e.g., Pending, Completed, Failed).
  + Payment Method: Type of payment used (e.g., Card, Wallet, Cash, Net Banking).
* **Relationship**:
  + A **Payment** is linked to a single **Payment Method** (One-to-One relationship).
  + A **Payment** is associated with one **Customer** (Many-to-One relationship).

**3. Payment Method (Abstract Class):**

* **Attributes**:
  + Method Type: Type of payment method (e.g., Card, Wallet, Cash, Net Banking).
* **Relationship**:
  + A **Payment Method** is used in a **Payment** (One-to-One relationship).
  + This is an **abstract class** for the different types of payment methods (Card, Wallet, etc.), so it's extended by other specific payment methods.

**4. Card (Subclass of Payment Method):**

* **Attributes**:
  + Card Number: The number on the card.
  + Expiry Date: The expiry date of the card.
  + cvv: The CVV number on the card.
* **Relationship**:
  + A **Card** is a type of **Payment Method** used in a **Payment** (One-to-One relationship).

**5. Wallet (Subclass of Payment Method):**

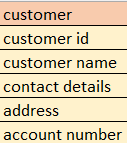
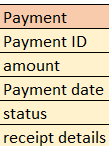
* **Attributes**:
  + Wallet Id: Unique identifier for the wallet.
  + balance: Current balance available in the wallet.
* **Relationship**:
  + A **Wallet** is a type of **Payment Method** used in a **Payment** (One-to-One relationship).

**6. Cash (Subclass of Payment Method):**

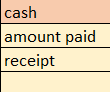
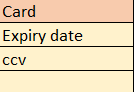
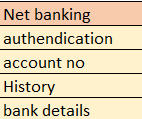
* **Attributes**:
  + Amount Paid: The cash amount used for the payment.
* **Relationship**:
  + **Cash** is a type of **Payment Method** used in a **Payment** (One-to-One relationship).

**7. Net Banking (Subclass of Payment Method):**

* **Attributes**:
  + Account Number: The account number from which the payment is made.
  + Bank Name: The name of the bank.
* **Relationship**:
  + **Net Banking** is a type of **Payment Method** used in a **Payment** (One-to-One relationship).



**Q7**

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

**ANS:**

The **MVC Architecture** (Model-View-Controller) is a software design pattern commonly used in the development of web and desktop applications. It is a way to structure an application by separating concerns into three distinct components:

1. **Model**:
   * The **Model** represents the core data or business logic of the application.
   * It is responsible for retrieving, storing, and manipulating the data.
   * The **Model** can be thought of as the “data” or “domain layer,” which includes classes representing the main objects or entities of the system, such as Customer, Order, Product, etc.
   * It does not handle presentation or interaction; it is only concerned with the data and rules of the business logic.
2. **View**:
   * The **View** represents the **user interface (UI)** that displays data from the **Model**.
   * The **View** is responsible for presenting the information to the user, and it typically interacts with the **Model** to display the relevant data.
   * It does not directly handle business logic; its purpose is purely presentation.
   * Examples of **Views** include HTML pages, forms, and visual components (e.g., buttons, tables, etc.).
3. **Controller**:
   * The **Controller** serves as an intermediary between the **Model** and the **View**.
   * It takes input from the **View**, processes it (often through the **Model**), and updates the **View** accordingly.
   * The **Controller** is responsible for controlling the flow of the application and updating the **View** based on changes in the **Model** or input from the user.

**Basic Flow of MVC:**

1. The **View** interacts with the **Controller** by sending user input.
2. The **Controller** processes the input, possibly updating the **Model**.
3. The **Model** updates the data and informs the **Controller**.
4. The **Controller** updates the **View**, which then reflects the changes in the UI.
5. ination of three actor and one use case results in three boundary class.
6. 2. Use case will result in controller class.
7. 3. Each actor will result in one entity class.
8. ination of three actor and one use case results in three boundary class.
9. 2. Use case will result in controller class.
10. 3. Each actor will result in one entity class.
11. ination of three actor and one use case results in three boundary class.
12. 2. Use case will result in controller class.
13. 3. Each actor will result in one entity class.

**Rules to derive the classes from use case diagram:**

**1.**

* Combination of one actor and one use case results in one boundary class.
* Combination of two actor and one use case results in two boundary class.
* Combination of three actor and one use case results in three boundary class

2. Use case will result in controller class

3. Each actor will result in one entity class

**Guidelines to place classes in 3-tier architecture-**

Presentation Layer-This layer is nothing but a user interface. View is inside this layer Tier architecture: - has 3 layers

* Application layer
* Business logic layer
* Data base layer

Data flowing from the 3 layers explains the 3-tier architecture where the information is passed. Here in elicitation techniques 3 tier architecture works as a medium where as a BA we get to know the information shared by the requester will it fit in application layer, business logic layer or data base layer

**Q.8**

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

**ANS:**

**1. Requirements Gathering and Analysis (Requirement Phase)**

* **Stakeholder Identification**: The BA identifies all key stakeholders (e.g., clients, end users, project managers, IT team, etc.) and engages them to understand their needs and expectations.
* **Business Needs Assessment**: The BA conducts meetings, interviews, workshops, and surveys to understand the business problem that needs solving. They gather high-level business objectives and constraints.
* **Documentation of Requirements**: The BA documents both functional (what the system should do) and non-functional (performance, security, etc.) requirements. This is usually captured in a Software Requirement Specification (SRS) document.
* **Use Cases and User Stories**: The BA creates use cases or user stories that describe the interactions between the system and its users, which will later be used for design and development.
* **Feasibility Study**: The BA may conduct a feasibility analysis (technical, operational, and financial) to assess if the solution is viable within the project constraints.
* **Sign-off from Stakeholders**: Once requirements are documented, the BA ensures that they are reviewed and approved by stakeholders, ensuring alignment with business objectives.

**2. System Design Phase**

* **Design Validation**: The BA ensures that the design meets the business needs and aligns with the **requirements** gathered in the previous phase. They help the technical team translate business requirements into technical specifications.
* **Review Design Documents**: The BA reviews design documents such as System Architecture Design, Data Flow Diagrams, Wireframes, and User Interface (UI) designs to ensure that the design adheres to the business objectives and requirements.
* **Clarification of Requirements**: During this phase, the BA often helps clarify ambiguities in the requirements and ensures that they are being addressed in the design.
* **Risk Identification**: The BA helps identify potential risks or gaps in the design that may affect business goals or timelines, and works to mitigate them early in the process.
* **Stakeholder Communication**: The BA acts as a liaison between the technical team and the business stakeholders, ensuring that stakeholders are updated on the design and its alignment with the requirements.

**3. Implementation or Coding Phase**

* **Monitor Development Progress**: While the development team handles coding, the BA monitors progress to ensure that the implementation remains aligned with the business requirements. The BA may attend sprint reviews or regular development meetings (if agile practices are used).
* **Provide Clarifications**: During coding, developers might encounter ambiguities in the requirements. The BA is available to provide clarifications, making sure the development process stays on track.
* **Change Management**: If any change requests or requirements changes occur during the coding phase, the BA ensures they are properly documented and assessed for impact on the timeline and budget.
* **Integration with Business Processes**: The BA ensures that the system's integration with existing business processes is considered during the implementation phase, providing guidance on how the solution will fit into the operational environment.

**4. Testing Phase**

* **Test Planning**: The BA assists in creating test plans by ensuring that business requirements are translated into test cases and that the test plan covers all business scenarios.
* **Review Test Cases**: The BA reviews the test cases to ensure that they accurately validate the system against the business requirements and objectives.
* **Functional Testing**: The BA often participates in User Acceptance Testing (UAT), where they validate the functionality of the system against the initial business requirements. The BA ensures that the solution meets the needs of the business users.
* **Defect Management**: If any defects or issues arise during testing, the BA helps prioritize them based on business impact. They also work with the testing and development teams to ensure issues are resolved promptly.
* **Sign-off**: The BA participates in the final sign-off of the system, ensuring that it meets the requirements before moving to the next phase.

**5. Deployment Phase**

* **Deployment Planning**: The BA works with the technical team to ensure the deployment is aligned with business needs. They help plan for user training, documentation, and data migration if necessary.
* **Change Management**: The BA helps communicate any changes in business processes that will result from the new system. They ensure that users are prepared for the changes and that the transition is smooth.
* **Stakeholder Coordination**: The BA coordinates with stakeholders to ensure that the system meets the business needs and that users are informed about the deployment schedule, changes in operations, and training requirements.

**6. Maintenance and Support Phase**

* **Monitoring System Performance**: The BA monitors the system's performance and ensures that it continues to meet business needs and objectives.
* **Post-Deployment Support**: The BA provides post-deployment support by gathering feedback from users and business stakeholders. They may identify any issues or areas for improvement.
* **Change Requests**: Based on feedback or evolving business requirements, the BA helps prioritize and manage **change requests** or **enhancements** that are necessary for system improvement.
* **Ensure Continuous Alignment**: The BA ensures that any new business needs or changes in business processes are accounted for and reflected in the system over time.
* **User Training and Support**: The BA may assist in additional training or user support to ensure that users are fully able to leverage the system's capabilities in their daily tasks.

**Q9**

**What is conflict management? Explain using Thomas – Kilmann technique**

**ANS:**

**Conflict management** refers to the process of identifying, addressing, and resolving disagreements or disputes in a way that minimizes negative outcomes and promotes healthy relationships. It involves dealing with conflicts in an organized manner, with the goal of finding a constructive solution that is acceptable to all parties involved. Conflict management is important in various environments, including workplaces, teams, and personal relationships, as it helps prevent escalation and fosters collaboration, productivity, and trust.

Conflict can arise due to differences in opinions, values, goals, or perceptions, and it is essential to manage conflicts effectively to ensure that they do not negatively impact the work environment or relationships.

**Thomas-Kilmann Conflict Mode Instrument (TKI)**

One of the widely recognized methods of conflict management is the **Thomas-Kilmann Conflict Mode Instrument (TKI)**, which is based on two fundamental dimensions:

1. **Assertiveness**: The extent to which an individual seeks to satisfy their own concerns (i.e., their level of self-interest).
2. **Cooperativeness**: The extent to which an individual seeks to satisfy the concerns of others (i.e., their willingness to accommodate others' needs).

Based on these two dimensions, the **Thomas-Kilmann Conflict Mode Instrument (TKI)** identifies **five conflict management styles**:

**The Five Conflict Management Styles:**

1. **Competing (High Assertiveness, Low Cooperativeness)**:
   * **Description**: In this style, one person pursues their own concerns at the expense of others. The goal is to win, and the other party’s concerns are not considered important.
   * **When to Use**: When quick, decisive action is needed (e.g., in emergencies or situations requiring firm, authoritative decisions).
   * **Risks**: Can damage relationships if overused, as it can create a win-lose situation.
   * **Example**: A manager making a quick decision without consulting the team to meet a tight deadline.
2. **Accommodating (Low Assertiveness, High Cooperativeness)**:
   * **Description**: This style involves giving in to the wishes of others. The person prioritizes the concerns of others over their own.
   * **When to Use**: When the issue is not important to you, or when maintaining the relationship is more important than the outcome.
   * **Risks**: The person may feel resentful if they constantly accommodate others and neglect their own needs. It can lead to unbalanced outcomes.
   * **Example**: A team member agreeing to a decision they don’t fully support, just to avoid conflict or keep the peace.
3. **Avoiding (Low Assertiveness, Low Cooperativeness)**:
   * **Description**: In this style, individuals avoid the conflict altogether. They neither assert their own needs nor cooperate with others. It involves sidestepping the issue or withdrawing from the situation.
   * **When to Use**: When the issue is trivial or when the potential damage from the conflict outweighs any benefits from addressing it.
   * **Risks**: Problems may fester and grow worse if not addressed. It can also lead to feelings of frustration and disengagement.
   * **Example**: An employee ignoring a disagreement with a colleague and hoping it will resolve itself on its own.
4. **Collaborating (High Assertiveness, High Cooperativeness)**:
   * **Description**: This is often considered the most effective conflict management style. It involves working with others to find a solution that satisfies everyone’s needs. It requires open communication and the willingness to explore multiple perspectives.
   * **When to Use**: When both parties' concerns are important, and a long-term solution is needed. This style fosters creativity and problem-solving.
   * **Risks**: It can be time-consuming and may not always be practical when there is a lack of trust or cooperation.
   * **Example**: Two colleagues brainstorming together to solve a problem in a way that meets both of their needs.
5. **Compromising (Moderate Assertiveness, Moderate Cooperativeness)**:
   * **Description**: In this style, both parties give up something to reach a mutually acceptable solution. It is a middle-ground approach that tries to meet halfway.
   * **When to Use**: When both parties have equally important but conflicting goals, and a quick, practical solution is needed.
   * **Risks**: The solution may not fully satisfy either party, leading to feelings of dissatisfaction or unmet needs.
   * **Example**: Two people negotiating a budget and agreeing to cut a little from both of their proposed amounts to reach a middle-ground figure.

**Q.10**

**List down the reasons for project failure**

**ANS:**

1.**Improper requirement gathering:**

If the requirements of the project are not gathered correctly, then this can lead to project failure.

2.**Lack of stakeholder involvement:**

A project can fail if the stakeholders are not participating in the process. The stakeholder input and feedback plays very important role to meet the goals

3.**Ineffective or less communication:**

If there are communication issues between stakeholders, team members then this can lead to misunderstandings or delays in project or even can lead to project failure.

4.**Continuous change in the requirement:**

If the requirements keep on changing frequently, this can also lead to project failure. Because the scope of the project will also keep on changing which will lead to project failure

5.**Poor risk management:**

Poor risk management can also lead to project failure. The team fails to identify the risks and do the risk mitigation, which can lead to unexpected challenges or delays in project. Lack of user involvement. Lack of executive support.

6**.Unrealistic expectations:**

This means the goals that cannot be achieved or the goals that are out of scope

**7.Improper planning:**

The project can fail if the planning is not done properly. The milestones, goals should be discussed. If there is no proper planning, then team may face difficulties in addressing the issues or to track the progress.

8.**Insufficient resources:**

Insufficient resources can also lead to project failure. The project may fail due to lack of technology knowledge or lack of finances.

**Q11**

**List the Challenges faced in projects for BA**

**ANS:**

* Obtaining sign-off on the requirement
* Change management
* Lack of training
* Co-ordination between developers and testers
* Conducting meetings.
* Making sure status report is effective
* Driving clients for UAT completion
* Making sure that the project is going on right track and delivered as per the timelines without any issues
* Gathering clear and unambiguous requirements can be challenging
* Unable to understand what stakeholder is trying to convey
* Scope creep- change in requirement or scope of the project during the project lifecycle can lead to scope creep
* Managing the stakeholder with conflicting interest can be a difficult task for BA
* BA may face difficulties in understanding the requirements if the domain is not familiar to him
* Poor communication between stakeholder and BA can affect the process of gathering the information
* Technical complexity

**Q12**

**Write about Document Naming Standards**

**ANS:**

Document Naming Standards refer to a set of established rules or guidelines for naming documents in a consistent, organized, and meaningful way. These standards are crucial in both personal and professional settings, especially for organizations that manage large volumes of documents. The main goal of document naming standards is to ensure clarity, ease of retrieval, and maintain order within a document management system.

Proper document naming can help improve efficiency, reduce errors, and ensure that anyone accessing or sharing files can easily understand the content of the document without needing to open it. Naming standards are particularly important in collaborative environments, such as businesses, educational institutions, government agencies, and other organizations where multiple people may be working with shared documents.

**Structure of Document Naming Standards**

Document names often follow a **structured format**, with predefined parts that are separated by consistent delimiters (e.g., underscores or hyphens). Below are common elements that can be included in document naming conventions:

1. **Project or Department Code**: A short identifier for the project, department, or team associated with the document.
2. **Document Type**: This could be a descriptor of the document’s content, such as "Report", "Invoice", "Proposal", "Specification", etc.
3. **Title or Subject**: A brief, clear description of the document's content.
4. **Date**: Date information is often essential to specify the version or timeframe of the document. The date is typically written in the format YYYY-MM-DD or YYYYMMDD to enable proper chronological sorting.
5. **Version**: If documents are updated or revised frequently, it is important to include a version number (e.g., v1, v2, final, draft).
6. **Author or Initials**: Including the author’s name or initials can help to quickly identify the person responsible for the document.
7. **Keywords or Tags**: In some cases, adding keywords or tags that categorize the document can help with searching and retrieval.

Example:

[ProjectID][Document Type]V[x]D[y].extension

Example- [PQ777FRDV1D1.docx] or [PQ777FRD1.1.docx]

**Q13**

**What are the Do’s and Don’ts of a Business analyst**

**ANS:**

Never say “NO” to the client.

* There is no word called as “BY DEFAULT”
* Never imagine anything in terms of GUI
* Question everything in the world
* Go to the client with plain mind that is with no assumptions
* Listen to the client very carefully and after he is done, then ask questions
* Don’t interrupt the client.
* Never try to give solutions to the client right away.
* Try to concentrate only on important and required things.
* Be like a lotus in mud- if a client comes with a fancy requirement, then talk to the project manager first.
* Requirement hurried-project buried.
* Never criticize the stakeholder. Always appreciate the stakeholder even for small efforts.

**Do’s for a Business Analyst**

1. **Do Understand the Business Needs Thoroughly**
   * A BA must start by gaining a deep understanding of the business objectives, pain points, and stakeholders' expectations. This will help in translating these needs into actionable requirements.
2. **Do Communicate Effectively**
   * Clear, transparent, and open communication is vital. A BA must act as a bridge between business stakeholders and technical teams, ensuring that everyone is on the same page.
3. **Do Collaborate with Stakeholders**
   * A BA must work closely with stakeholders (e.g., business users, project managers, IT team, etc.) to gather requirements, validate them, and ensure alignment with business objectives.
4. **Do Document Requirements Clearly**
   * Requirements should be clearly documented in a manner that is easily understood by both technical and non-technical stakeholders.
5. **Do Focus on Value Delivery**
   * Always keep the focus on delivering value to the business. The BA should prioritize features or solutions that bring the most value to the organization.
6. **Do Be Adaptable and Flexible**
   * The business environment, project scope, or stakeholder needs may change during the project lifecycle. The BA must adapt to these changes and adjust the requirements or approach accordingly.
7. **Do Facilitate Requirement Elicitation Sessions**
   * A key responsibility of a BA is facilitating workshops, interviews, and brainstorming sessions to capture requirements.
8. **Do Use Proper Tools and Techniques**
   * A BA should be proficient in using the right tools for documentation, modeling, and collaboration.
9. **Do Analyze Data and Trends**
   * Business analysts should use data analysis to understand patterns, identify inefficiencies, and recommend process improvements.
10. **Do Test and Validate Solutions**
    * Ensure that the proposed solutions meet the business requirements by being involved in testing and validation activities like User Acceptance Testing (UAT).

**Don’ts for a Business Analyst**

1. **Don’t Assume Without Clarification**
   * A BA should never make assumptions about what stakeholders want. Assumptions can lead to misunderstandings and wasted resources.
2. **Don’t Ignore Stakeholders**
   * Ignoring or neglecting certain stakeholders can lead to incomplete or incorrect requirements. A BA must involve all key stakeholders early and throughout the project.
3. **Don’t Be Overly Rigid with Requirements**
   * While documentation is important, a BA should be flexible when it comes to changing requirements. Business needs can evolve, and the BA should be ready to pivot.
4. **Don’t Skip the Validation Process**
   * Skipping the validation of requirements with stakeholders can result in a mismatch between the actual system and business needs.
5. **Don’t Overwhelm Stakeholders with Technical Jargon**
   * A BA should avoid overwhelming business stakeholders with technical terms or complex concepts. Keep the conversation focused on business outcomes.
6. **Don’t Work in Isolation**
   * A Business Analyst should not work in isolation. They need to work closely with the project manager, developers, testers, and other stakeholders.
7. **Don’t Focus Only on Requirements Gathering**
   * While gathering requirements is crucial, the BA’s role doesn’t end there. It is also important to monitor the development process, test solutions, and ensure the solution aligns with business needs.
8. **Don’t Neglect Documentation of Assumptions and Constraints**
   * Assumptions and constraints should always be documented. Failing to do so may lead to misunderstandings down the line if they impact requirements or design.
9. **Don’t Disregard the User Experience (UX)**
   * Business Analysts should not only focus on technical or functional requirements but also on user experience. A system may technically meet requirements but still fail if it's difficult to use.
10. **Don’t Forget to Track Changes**
    * Change is inevitable during a project, but failing to track changes can lead to scope creep and confusion.

**Q14**

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

**ANS:**

**Package**:

* A **package** is a way of organizing classes, interfaces, and other code components into a namespace or container to manage the complexity of a software system. It is typically a logical grouping of related classes, interfaces, and modules that share common functionality.
* A package helps in organizing the codebase, reducing name conflicts, and improving maintainability.
* **Example**: In Java, you can have packages like com.companyname.models, com.companyname.controllers, etc.

**Sub-system**:

* A **sub-system** refers to a larger, self-contained unit or a collection of related components (which may include multiple packages) within a system. A sub-system typically represents a distinct part of the overall system that is responsible for specific functionality and can be treated as a module or independent part of the system.
* A sub-system could include not only code (in the form of multiple packages) but also databases, services, interfaces, and even hardware components in some cases.
* **Example**: A "payment processing system" or "inventory management system" in an e-commerce application would be considered a sub-system.

**Q15**

**What is camel-casing and explain where it will be used-**

**ANS:**

**Camel casing** (or **camel case**) is a **naming convention** in which multiple words are combined into one word, and each word after the first starts with an uppercase letter, with no spaces or punctuation. The first letter of the first word is typically lowercase. The resulting "camel case" resembles the humps of a camel (hence the name), where the capital letters are seen as the "humps" and the lowercase letters as the "valley."

**Where Camel-Casing Is Used**

Camel casing is widely used across various programming languages and in different contexts. Here are the most common places where camel casing is applied:

**1. Variable Names**

* **Lower CamelCase** is typically used for naming variables in many programming languages like Java, JavaScript, C#, Python, etc.

**2. Function/Method Names**

* **Lower CamelCase** is often used for naming functions or methods, indicating actions or behaviours.

**3. Class and Type Names**

* **Upper CamelCase (PascalCase)** is commonly used for naming classes, data types, or object-oriented structures.

**4. File and Folder Names (in some contexts)**

* In many modern development environments or frameworks (e.g., JavaScript/TypeScript, React), files and folders might use **camel case** to maintain consistency with variable and function names.

**5. Constant and Enum Values (sometimes)**

* Although **UPPER\_SNAKE\_CASE** is generally preferred for constants in many languages (like C, C++, Java), some languages or frameworks may use **camelCase** or **PascalCase** for constant values in certain cases.

**6. Event Names (in frameworks like JavaScript/React)**

* In JavaScript or React, event handlers are often named using camel case to describe the action being performed.

**7. API Endpoints**

* In RESTful API design, camel case is sometimes used in naming URL endpoints and query parameters.

**8. Configuration Parameters**

* Some configuration files or environment variables (especially in JavaScript-based environments) use camel case to define settings.

**Q16**

**Illustrate Development server and what are the accesses does business analyst has?**

**ANS:**

A **development server** is a server that is set up in a software development environment to allow developers to build, test, and debug their applications before they are deployed to production. It typically runs software that facilitates code testing, debugging, and sometimes staging features that are being developed.

While the BA does not typically engage directly with the server or code, they contribute significantly to **defining the system requirements**, **gathering feedback**, and ensuring that the developed features meet the business needs. In terms of access to a **development server**, the **BA** may have limited, but valuable access.

1. BA does **not** directly interact with the server at the same level as developers, but here’s what kind of access and interaction they may have. A BA may have **view-only** access to the development environment for the purpose of **tracking progress**, **reviewing features**, and **validating functionalities**.
2. BA may not **directly** execute code or deploy software, they may have access to **testing environments** or a **staging server** where features are tested. BAs often have access to test cases and user stories that outline the expected behaviour of the system.
3. Some Business Analysts may have access to tools used for **documentation**, **user stories**, and **requirement management**. While this isn't direct access to the development server, these tools interface with the server and allow BAs to remain in sync with the development team's progress.
4. BAs might have the ability to **monitor deployment** or **release management** to ensure that **features are rolled out** in line with business expectations.
5. A key role of a Business Analyst is to provide feedback based on business requirements. Although BAs may not typically interact directly with the development server's backend, they may provide continuous feedback on the **user interface (UI)** and **user experience (UX)** directly within the development environment.

**Q17**

**What is Data Mapping**

**ANS:**

**Data mapping** is the process of transforming and associating data elements from one format or structure to another. It involves establishing relationships between two different sets of data so that data can be properly integrated, transferred, or used in another system. This process is critical in various contexts such as data migration, data integration, and database design, where data from one source needs to be converted into a format that another system can understand or process.

**Types of Data Mapping**

Data mapping can occur in several different scenarios, each requiring different techniques. These include:

1. **Database Mapping**:
   * Involves mapping between tables, columns, and fields of one database to another.
   * Example: When transferring data from a **SQL database** to a **NoSQL database**, or between different relational databases.
2. **XML/JSON Mapping**:
   * Mapping data between formats like XML or JSON and relational databases or other data structures.
   * Example: Converting data in an **XML file** to a relational database (mapping XML tags to database table fields).
3. **ETL (Extract, Transform, Load)**:
   * Data mapping plays a significant role in **ETL processes** for **data warehousing** and **data integration**, where data is extracted from source systems, transformed into the required format, and loaded into a target system.
   * Example: Extracting sales data from multiple systems, transforming it to a common format, and loading it into a **data warehouse**.
4. **Data Migration**:
   * The mapping process is often part of data migration, where data from one system is transferred to another, possibly involving changes in structure or format.
   * Example: Migrating from an **old legacy system** to a **new ERP system**.
5. **API Data Mapping**:
   * When systems communicate via APIs (Application Programming Interfaces), data mapping is used to ensure that data sent by one system matches the format expected by the receiving system.
   * Example: Mapping data from a **payment gateway API** to the format required by an internal accounting system.

**Tools for Data Mapping**

1. **ETL Tools**:
   * **Talend**, **Informatica**, **Microsoft SQL Server Integration Services (SSIS)**: These tools automate the process of extracting data, mapping it to the target system, and loading it.
2. **Data Integration Platforms**:
   * **MuleSoft**, **Dell Boomi**, **Snap Logic**: These platforms provide drag-and-drop interfaces to simplify the mapping of data between various systems.
3. **Database Management Systems (DBMS)**:
   * **MySQL Workbench**, **SQL Server Management Studio (SSMS)**, **Oracle SQL Developer**: These can assist in defining schema mappings between databases.

**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**

**ANS:**

**API (Application Programming Interface)** is a set of rules, protocols, and tools that allow different software applications to communicate with each other. It defines the methods and data formats that applications can use to request and exchange information. APIs enable seamless integration between various systems and allow different software to work together, regardless of their underlying technologies.

**Types of APIs:**

1. **RESTful APIs**: These are the most commonly used APIs and follow REST (Representational State Transfer) principles. They use HTTP methods like GET, POST, PUT, DELETE, etc., to perform operations on resources.
2. **SOAP APIs**: A protocol for exchanging structured information, usually using XML.
3. **GraphQL APIs**: A query language for APIs, where the client can specify exactly what data it needs.

API integration allows different systems to communicate and share data in a structured manner. In the context of your application, which uses **dd-mm-yyyy** as the date format, and is receiving data from another system in the **mm-dd-yyyy** format, API integration becomes critical for transforming the data so that both systems can understand and correctly process the date values.

**How to Handle API Integration for Date Format Transformation**

1. **Receive Data via API**: The US-based application sends data to your application via an API. This data includes a date field, but the date is in the **mm-dd-yyyy** format.
2. **Date Format Conversion in API Integration**:When the data is received, the date must be converted from the **mm-dd-yyyy** format to the **dd-mm-yyyy** format.
3. **Implementation Steps**:Create an endpoint in your application that will accept incoming data from the US system, which includes the date in **mm-dd-yyyy** format. Once the data is received, use a function to convert the date format from **mm-dd-yyyy** to **dd-mm-yyyy**. After converting the date format, you can now store the formatted date in your database or process the data further.
4. After processing the data, your application can send a response back to the US-based application, confirming that the data has been received and stored correctly.

**Advantages of Using API Integration for Date Transformation:**

1. **Seamless Integration**: APIs provide a standard interface for systems to communicate with each other, even when they use different data formats (like different date formats).
2. **Automation**: The date transformation process is automated, ensuring that data from the US system is always properly converted before being used in your application.
3. **Consistency**: By centralizing the date transformation logic within the API layer, you ensure consistent formatting across your system, preventing errors caused by mismatched date formats.
4. **Scalability**: If the application from the US expands or changes its date format in the future, the API integration can be easily updated to handle new formats without requiring changes to the core application logic.