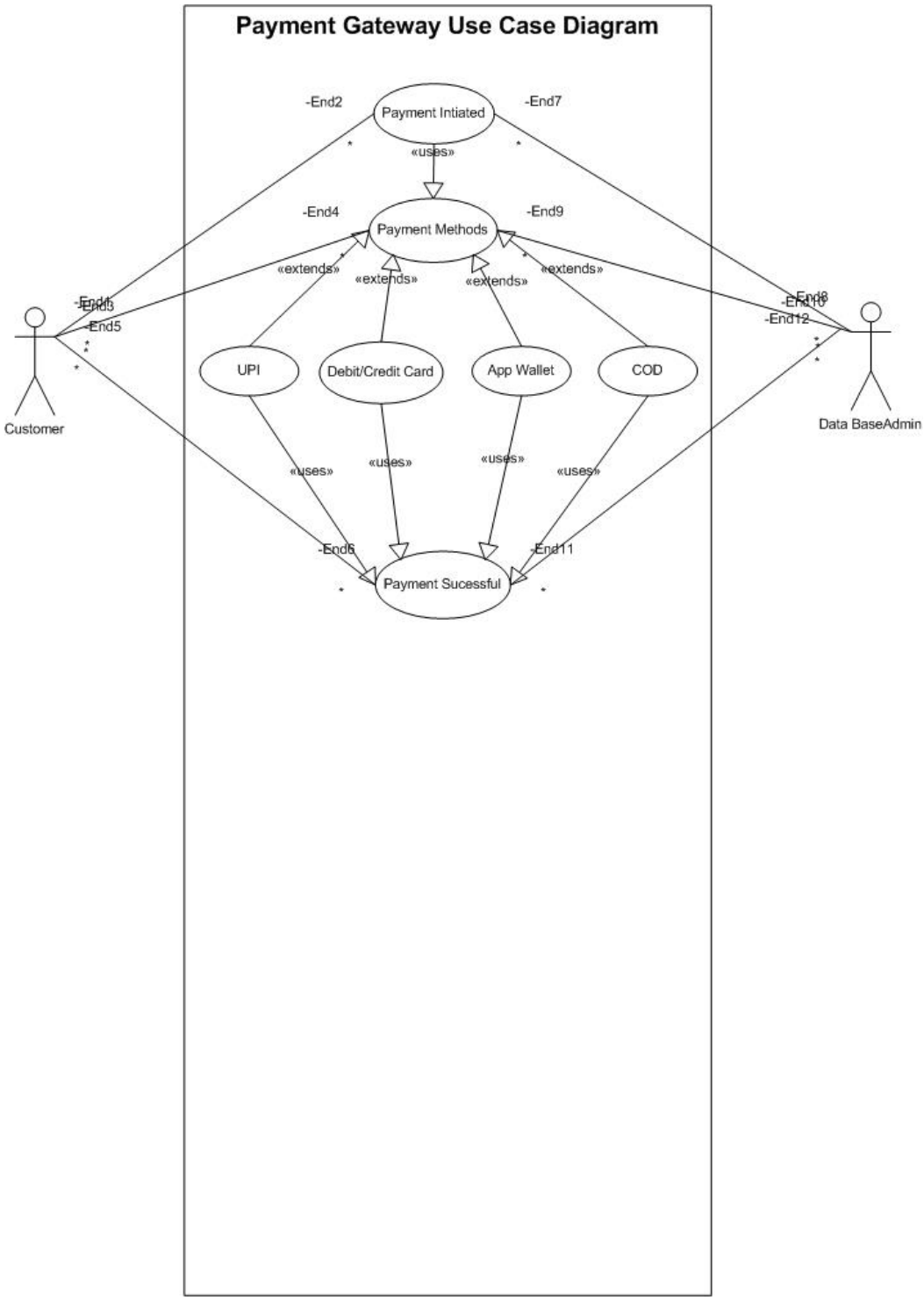
Question 1 - Use Case Diagram

A Use Case Diagram is a simple flow picture that shows how different users (like customers, admins, or staff) interact with a system. It helps us understand **who does what** in the system and **what features they use**. This diagram is very important in projects because it gives a clear view of the system's functions from the user's side.



Question 2 - Derive Boundary Classes, Controller classes, Entity Classes.

In any software system, we divide the work into three types of classes to keep things clean and easy to manage. These are Boundary Class, Controller Class, and Entity Class.

Boundary Class: This class connects users to the system. It takes user inputs and shows results back to them. It acts like the front desk person in a shop, talking to customers but not making business decisions.

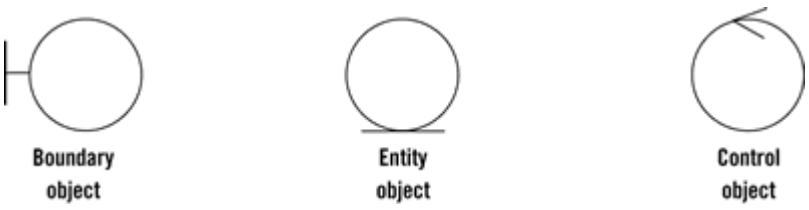
Example: In Gold & Silver shopping app, the Login Page, Product Page, or Payment Screen are boundary classes. They take input from the customer and show information like item price or payment status.

Controller Class: This class works like a manager. It takes the input from the boundary class, understands what to do, and then talks to the entity class to perform the real work.

Example: When the user clicks "Buy Now", the controller checks if the user is logged in, if the item is in stock, and then asks the entity class to process the order.

Entity Class: This class handles the main business logic and data. It saves, updates, and keeps the records safe, like a cashier or accountant who manages bills and balances.

Example: Entity classes manage details like user profile, product info, order history, and payment records in the database.



Each class plays an important role to keep the system clean and working properly. This way, the system becomes easy to maintain and change later.

Question 3 - Place These Classes on 3-Tier Architecture

In software development, we organize the system into three main layers called 3-Tier Architecture. Each layer has its own role, and the classes we discussed earlier (Boundary, Controller, Entity) fit into these layers.

1. Presentation Tier (User Interface Layer): This layer interacts directly with the user. It collects inputs and shows outputs using screens like webpages or mobile apps. It includes Boundary Classes because they handle what the user sees and does.
Example: When a customer browses the gold items on the app or enters login details, it's all handled in the presentation tier.

2. Business Logic Tier (Middle Layer): This layer handles all the logic and decisions. It connects the presentation tier with the database. Controller Classes are part of this layer, because they decide what to do based on user actions and business rules.
Example: When a user clicks "Place Order", this layer checks stock, calculates the price, verifies payment, etc., before proceeding.
3. Data Tier (Storage Layer): This layer is where all the data is saved and managed, like product details, user profiles, order history, etc. Entity Classes belong here because they deal with storing and retrieving real data.
Example: If a user orders a silver bracelet, this layer stores the order details and updates the stock in the database.

By placing each class in its right tier, the system becomes well-organized, easier to update, and more secure.

Question 3 - Explain Domain Model for Customer making payment through Net Banking

A Domain Model shows how different parts of a system are connected in a simple visual format. It's like showing the major tables or objects, such as customer, bank, account, payment, and transaction, along with their basic details. This model helps everyone understand how the information flows when a customer makes an online payment.

Customer

ID	Name	Details	Address	Account No
EMP1234	M.Eswar	BA	Hyderabad	0001234

Bank

Bank Name	Bank Location	Branch Code
Equitas Bank	Ameerpet	EQT5678

Payment

Payment ID	Payment Amount	Payment Date	Status
CHP18906	₹10,000	24/05/2025	Completed

Account

Account Number	Account Type	Balance	Account Holder Name
EQT180654	Saving	₹5,00,000	Malladi Eswar

Net Banking Service

Authentication	Fund Transfer	Transaction History	Account Management
Done	Done	₹10,000	Done

Authentication

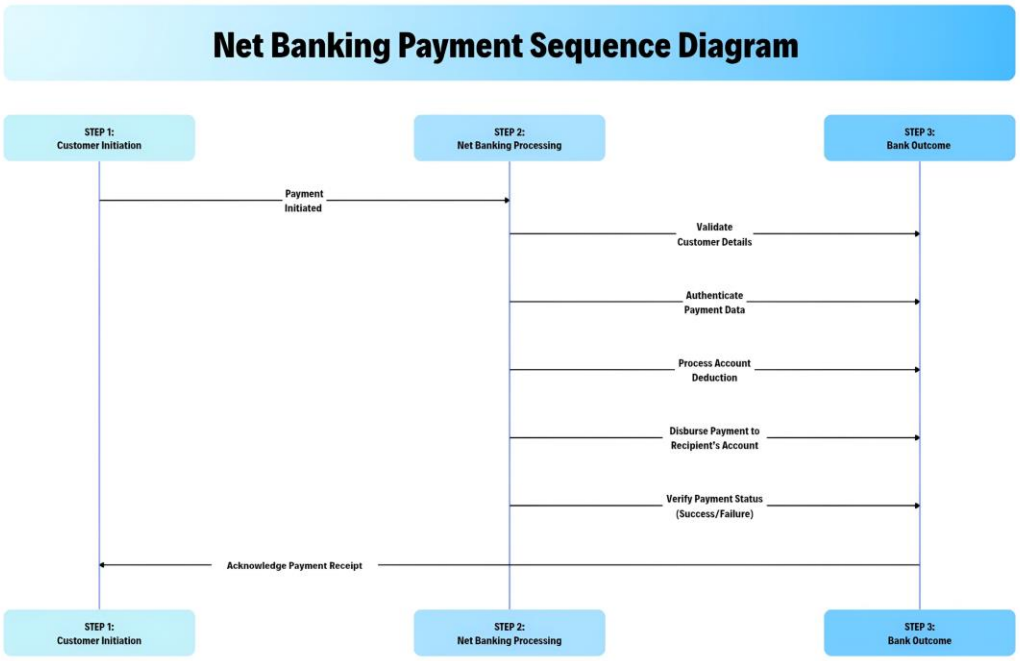
User Name	Password	OTP
Eswar	EswarNet1234	5782

Transaction

Transaction ID	Reception Details	Amount	TimeStamp
CHP18906	BHR6742	₹10,000	24/04/2025

Question 5 - Draw a sequence diagram for payment done by Customer Net Banking

A sequence diagram is used in software development to show how different parts of a system talk to each other step by step over time. It helps us understand the flow of events that happen when a user performs a particular task - *like making an online payment*.



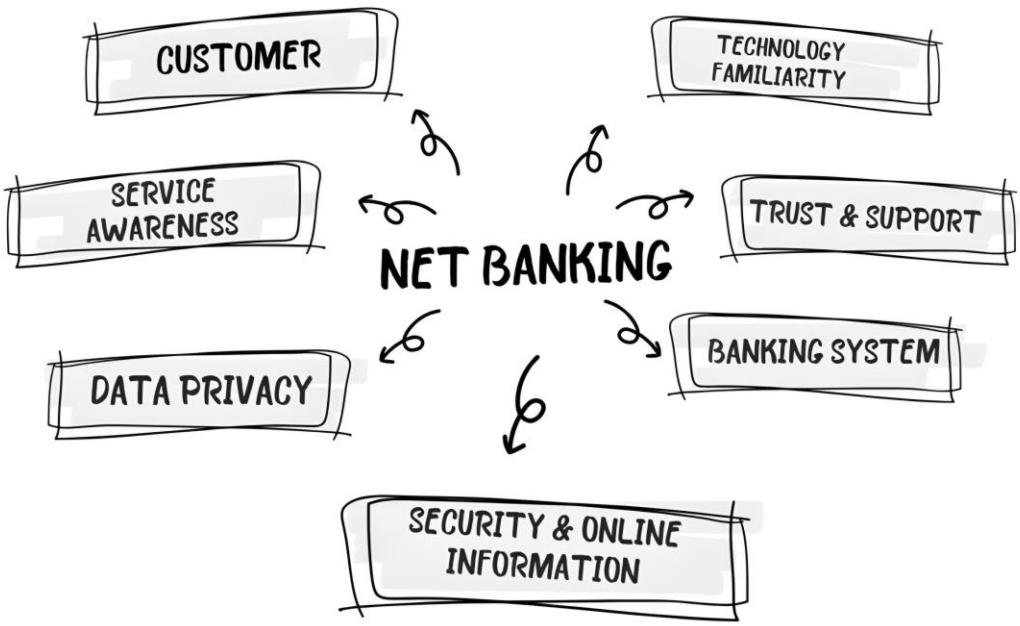
Question 6 - Explain Conceptual Model for this Case

A Conceptual Model is a simple way to explain the system at a high level. It helps everyone understand the system, what it does, who uses it, and how things are connected. For this Net Banking Payment project, this model shows how users, banks, and supporting features work together.

Key Concepts in the Conceptual Model:

1. **Customer:** This represents the person using Net Banking, like Eswar who logs in to pay for a product using his online banking account.

2. **Service Awareness:** The customer should know what services are available in the Net Banking system like Fund Transfer, Transaction History, or Bill Payments.
3. **Privacy of Data:** The system must make sure that the customer's personal details and banking information are protected. Only authorized users should access the data.
4. **Technology Awareness:** The customer should be familiar with basic online usage like entering OTPs, passwords, and using secure login methods to use Net Banking without confusion.
5. **Trust and Support:** The customer must feel that the bank is secure and responsive. A strong customer service and smooth payment experience builds trust in the system.
6. **Bank:** The bank acts as a service provider that manages user accounts, handles payments, verifies login, and maintains transaction records securely.
7. **Online Information:** Customers should get timely and accurate updates through messages or dashboards about their payment status, account balance, or transaction receipts.
8. **Security and Privacy:** The bank must use proper security features like firewalls, encryption, and authentication to ensure no unauthorized access or data leaks during transactions.



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 widely used architectural pattern in software design. It helps in organizing code neatly by dividing the application into three major components Model, View, and Controller.

Model (Entity Classes): These are the classes that represent the real-world data. They are responsible for storing and processing the data.

Example: *Customer, Order, Product, Payment* are all Model classes because they contain the actual business data like gold weight, jewellery price, order date, customer address, etc. These classes are placed in the Database Layer.

View (Boundary Classes): This part is responsible for interacting with the user. It collects inputs from the customer and displays outputs.

Example: Screens like *Login Form, Product Display Page, Order Confirmation Page* are boundary classes. If a customer logs in to buy a gold necklace, the interface he sees and interacts with is part of the View. These are placed in the Application Layer.

Controller (Controller Classes): These classes act like a bridge between the View and Model. They receive inputs from the user (via View), process the logic, and update the data (via Model).

Example: *Login Controller, Payment Controller, Order Controller* manage what happens after the user clicks "Place Order" or "Pay Now". These controllers handle the flow of data and decision-making. Placed in the Application Layer or Business Logic Layer based on reuse and complexity.

Rules to Derive Classes from Use Case Diagram:

1. **One Actor with One Use Case → One Boundary Class**
   Example: Customer using "Add to Cart"

2. **Two Actors with One Use Case → Two Boundary Classes**
   Example: Admin and Manufacturer using “Manage Products”

3. **Each Use Case → One Controller Class**
   Example: “Checkout” use case

4. **Each Actor → One Entity Class**
   Example: Customer, Manufacturer, Admin become Entity classes

Guidelines to Place Classes in 3-Tier Architecture:

Layer	Class Type	Example in Jewellery E-Commerce
Database Layer	Entity Classes	Customer, Order, Product, Payment
Application Layer	Primary Boundary Classes & Controllers	Login Form, Product Search Page, Checkout Controller
Business Logic Layer	Reusable Controllers / Governing Forms	Inventory Controller, Report Controller (used by multiple actors)

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

In a Jewellery E-Commerce project (where customers buy gold and silver jewellery online), a Business Analyst (BA) plays a key role from start to end. Below is how a BA contributes in each stage of the Waterfall Model:

1. Pre-Project Phase

- I first do a **SWOT Analysis** to understand if an online jewellery store is feasible (e.g., strength: high gold demand, weakness: trust in online quality).
- I perform a **GAP Analysis** to check what's missing in the current jewellery market (e.g., most stores don't offer home delivery).
- I also identify the **Root Cause** of problems (e.g., why customers avoid online buying – maybe due to lack of real images or certification).

2. Planning Phase

- I do a **Stakeholder Analysis** (e.g., owners, gold vendors, delivery team).
- I go through the **project plan** shared by the PM to understand the timelines and scope.
- I prepare a **BA strategy** – like how I will gather requirements, how often I'll interact with the client, etc.

3. Requirement Gathering Phase

- I interact with the client (say, a jewellery brand owner) to understand what features they want: product catalogue, virtual try-on, online payments.
- I use **Brainstorming** sessions to get inputs from marketing and delivery teams too.
- I then organize the requirements and use **MoSCoW** to prioritize them (e.g., "Must have" – online payments, "Could have" – 3D jewellery viewer).
- I validate all requirements using **SMART** and **FURPS** techniques to ensure they are clear and achievable.

4. Requirement Analysis Phase

- I create **UML diagrams** (like Use Case and Activity) to visually represent how customers will use the website.
- I draft the **SRS (Software Requirement Specification)** document with clear business and technical needs.
- Once the client confirms, I take a **sign-off**.
- I prepare the **RTM (Requirement Traceability Matrix)** to map each requirement with its test cases later.

5. Design Phase

- I help the QA team in writing **Test Cases** (e.g., check if the invoice is generated after jewellery payment).

- I explain the **solution design** to the client, like how the cart, wish list, and payment gateway will work.
- I start working on the **End-User Manual**, which will guide users (like shop managers or customer care teams) on how to use the system.

6. Coding Phase

- I organize **JAD (Joint Application Development)** sessions between developers and business users to avoid confusion.
- I clarify any doubts the development team has (e.g., "Should COD be available for gold orders?").
- I prepare the client for **UAT (User Acceptance Testing)** by briefing them about the new features.

7. Testing Phase

- I request **test data** from the client (like sample product listings, dummy customer accounts).
- I support the QA team in testing the workflows (like checkout, order tracking).
- Once the client is happy, I take a **sign-off** on the **Client Project Acceptance Form**.

8. Deployment & Implementation Phase

- I share the final **RTM** with the PM or Client to track that all requirements are delivered.
- I plan **training sessions** for users like jewellery sales agents, warehouse staff, etc.
- I document **lessons learned** (e.g., "Next time, involve jewellery designers early to avoid last-minute changes in product info").

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

Conflict Management is the process of identifying, addressing, and resolving disputes that arise between stakeholders during the project lifecycle. For example In a Jewellery E-Commerce project, conflicts may occur between the marketing team and developers regarding feature release dates or between the client and UI designers over the design layout of a product page.

To manage such situations, the **Thomas–Kilmann Conflict Mode Instrument (TKI)** offers five practical techniques based on two factors: assertiveness and cooperativeness. These help team members handle conflicts based on the situation.

Five Conflict Handling Modes in Thomas–Kilmann Technique:

1. **Competing**: High assertiveness, Low cooperativeness
   Example: When the Product Owner insists on launching a "Diwali Jewellery Sale" feature within a strict deadline, even if the development team needs more time.

2. **Accommodating**: Low assertiveness, High cooperativeness
   Example: A BA agrees to a UI design change requested by the client, even if it's minor and not part of the original scope, just to maintain good relations.

3. **Avoiding**: Low assertiveness, Low cooperativeness
   Example: The development team postpones discussions on less critical wish list module issues to focus on core payment integration during a sprint.

4. **Collaborating**: High assertiveness, High cooperativeness
   Example: The QA (Quality Assurance) and developer teams work together to solve a critical bug in the "Add to Cart" function during peak festival sales season, ensuring both speed and quality.

5. **Compromising**: Moderate assertiveness and cooperativeness
   Example: The client wants a “Try Jewellery Virtually” feature, but due to time limits, the team agrees to launch a basic version now and enhance it in the next sprint.

5 Steps for Conflict Management:

1. **Identify the Conflict:**
   Ex: A delay in feature delivery raises concerns among the client.
2. **Discuss the Details:**
   BA arranges a call between the technical team and the client to understand both sides.
3. **Agree on the Root Problem:**
   They find that unclear requirement details caused repeated rework.
4. **Explore Possible Solutions:**
   Propose additional grooming sessions before every sprint.
5. **Negotiate and Finalize:**
   All agree to involve the client early in each sprint to reduce miscommunication.

Question 10 - List down the reasons for project failure

Project failures can happen due to the following reasons Example - In Jewellery E-Commerce project:

Improper Requirement Gathering:
If the BA doesn’t collect detailed requirements like payment gateway options for high-value jewellery or image zoom for product view, the system might miss critical features, leading to customer dissatisfaction.

Frequent Requirement Changes:
When the client keeps changing the design or flow of the product catalogue or checkout process after development has started, it delays timelines and increases rework.

Lack of User Involvement:
If actual users like jewellery buyers or customer care staff aren’t involved during UAT, important practical issues (like price fluctuations or live gold rate integration) may go unnoticed.

Poor Executive Support:
If senior management does not approve budgets on time for secure hosting or does not provide feedback on product releases, it affects planning and delivery.

Unrealistic Expectations:
Assuming the app will be ready within 2 weeks without understanding the effort needed for features like product recommendation, secure transactions, and real-time inventory sync leads to failure.

Improper Planning:
Not scheduling tasks like festive season load testing (e.g., for Diwali sales) or failing to assign roles clearly among the QA, developer, and content upload teams results in confusion and missed deadlines.

Question 11 - List the Challenges faced in projects for BA

Several challenges arise during the project lifecycle Example - In Jewellery E-Commerce project:

Lack of Proper Training:
Sometimes, team members may not fully understand how jewellery-specific platforms work (e.g., purity filters, gold rate integration). The BA has to train the internal team and sometimes even the client on the basics of the system.

Difficulty in Getting Requirement Sign-Off:
Clients may hesitate to approve requirements for modules like online gold loan eligibility or custom jewellery order tracking. Delays in sign-offs slow down development.

Frequent Requirement Changes:
The client may initially request a simple "Gold Purchase" module, then later add EMI or dynamic pricing based on weight. Managing these changes while sticking to the timeline becomes challenging.

Coordination Between Developers and Testers:
A BA often needs to clarify confusion between developers and testers. For instance, the developer may build a gold cart with GST inclusive, but the tester checks it against outdated requirements. BAs must keep both teams aligned.

Client Meetings and Communication Gaps:
Scheduling meetings with jewellery brand owners, who may not be tech-savvy, and explaining system flows like secured payment or BIS Hallmark certificate uploads is often time-consuming and requires patience.

Status Reporting and Tracking Progress:
The BA is responsible for preparing weekly updates, but when issues like inventory mismatch or vendor portal delay occur, it becomes tough to justify pending tasks to stakeholders.

Driving UAT Completion:
Getting business users to test the site, especially busy jewellery store owners, is tough. The BA must coordinate closely, ensure test cases are shared, and follow up for feedback and final sign-off.

Handling People and Conflict:
Conflicts may arise when developers complain about unclear logic (e.g., real-time gold rate updates), or when clients demand last-minute UI changes. A BA needs strong communication and people skills to manage expectations.

Question 12 - Write about Document Naming Standards

It helps all team members, including clients, easily identify, track, and refer to the correct documents across different phases of the project. A proper document naming format looks like this:

[Project ID] [Document Type] V[Version]D[Year].ext

Breaking it down:

- **Project ID**: A unique code for the project (example: JEW001 for Jewellery Project 1)
- **Document Type**: Specifies what the document is about (e.g., BRD, SRS, RTM)
- **V[Version]**: Version number (e.g., V1.0 for the first version)
- **D[Year]**: Year of creation (e.g., D2025)
- **ext**: File extension (e.g., .docx, .xlsx)

Example: If the Business Requirement Document for a gold jewellery platform is created in 2025 with version 1.0, the file name would be: **JEW001_BRD_V1.0D2025.docx**

This system ensures that everyone on the team, including developers, QA, testers, and stakeholders, is on the same page and uses the latest version of the correct document without confusion.

Question 13 - What are the Do's and Don'ts of a Business Analyst?

Below are the key do's and don'ts based on Example - In Jewellery E-Commerce project:

Do's:

- **Always listen completely before responding:** When a client, like a jewellery store owner, explains their issues with the online ordering system, let them finish completely before asking questions.
- **Clarify, don't assume:** Never make assumptions about how features like "gold purity filter" or "engraving options" work. Ask detailed questions and validate with the client.
- **Stay neutral and gather facts:** Approach every meeting without bias or prior conclusions. For example, if one feature worked for a silver vendor, it doesn't mean it will work the same for a diamond vendor.
- **Involve SMEs (Subject Matter Experts) when needed:** If the client mentions complex tax rules for gold or gemstone pricing, consult with domain experts rather than assuming.
- **Capture unique problems:** Understand that each vendor's needs are different. A shop in Hyderabad might want multi-language support, while one in Delhi may need dynamic pricing, respect these variations.
- **Encourage client participation:** Push for active input from stakeholders during feature demos, especially for modules like payment gateways or product upload flows.
- **Prioritize requirements:** Focus on business-critical features like "secure payment" or "trust indicators for certification" over cosmetic UI feedback during early phases.

Don’ts:

- **Never interrupt the client:** If the jeweller is describing an issue with return policies for customized rings, don’t jump in midway with assumptions or quick solutions.
- **Avoid GUI (Graphical User Interface) imagination:** Don’t promise the layout or screen design upfront unless validated by the UX/UI team. Instead, focus on what the client wants the system to do.
- **Don’t rely on ‘By Default’ logic:** For instance, don’t assume that all products should display “offers” by default. Confirm it with the client.
- **Don’t offer pre-built solutions too early:** Even if a past project had a similar return module, wait to hear the client's full problem before suggesting it as a solution.
- **Don’t generalize or ignore details:** Two similar shops might have different packaging needs, respect and capture those details.
- **Never say NO to the client:** Even if a feature seems out of scope like “AI-based jewellery suggestions,” acknowledge the request and discuss its feasibility politely.

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

We organize the system into smaller and larger units to keep it manageable and scalable. Two key ways of doing this are using Packages and Sub-systems. Both help in organizing the code, but they serve different purposes.

Package: A package is like a small folder in the system that focuses on one specific feature. It groups together related classes or functions.

Example: We may have a “Payment Package” that includes all classes related to payment processing (like Net Banking, Credit Card, UPI, Wallet). It is small and focused, so it handles only payment-related logic.

Sub-system: A sub-system is a bigger block that includes multiple packages. It represents a major part of the system with its own functionality and boundaries.

Example: An “Order Management Sub-system” could include packages for Cart, Checkout, Payment, and Order Tracking. It is larger and covers an entire business function. It manages higher-level dependencies, like how Cart talks to Payment and then to Order.

Question 15 - What is camel-casing and explain where it will be used

Camel-casing is a naming style used in programming where each word in a compound name starts with a capital letter, except for the first word. There are no spaces or special characters used. This method makes long variable names easier to read and understand.

Example: Let's say we are naming variables or functions in our application. Instead of writing *Add new product to cart*, we write it as *addNewProductToCart*. Similarly, *View Customer Orders* becomes *viewCustomerOrders.*

This style is useful when we write code for functions like:

- *checkGoldPriceToday()*
- *applyDiscountOnCheckout()*
- *generateInvoiceForOrder()*

Why We Use Camel-Casing:

- Helps developers easily read and identify parts of a function or variable.
- Brings uniformity in naming across the code base.
- Makes it easier to collaborate in teams because everyone follows the same standard.
- Reduces confusion between words in a name (like *productnamequantity* vs *productNameQuantity*).

Camel-casing is mostly used in coding, especially while writing:

- Variable names
- Function names
- Class names

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

A Development Server is used for testing the system in real time before it reaches the end-user. It mirrors the live system but is safe for trial runs, debugging, and corrections.

As a Business Analyst, my role in the development server is mostly observer and reviewer:

- I usually have **Read-Only Access**, which means I can see how the system behaves but I cannot make changes.
- Sometimes, I have **Collaborative Access**, where I can test flows like jewellery product listings, order tracking, or invoice generation with the QA team.
- I may also be given **Limited Configuration Access**, like updating test data for customer names, prices, or order scenarios, but without changing the core code.

This limited access ensures that while I'm closely monitoring how the features work, I don't accidentally make any change that could affect the developers' work. It helps in raising bugs and validating features against the requirements in a controlled and safe environment.

Question 17 - What is Data Mapping?

Data mapping is the process of matching data from one system or format to another. It is mainly used when two systems need to talk to each other, especially during system integration, data migration, or report generation.

For example: When we are connecting the online shopping application with the payment gateway, we must ensure that data like Customer Name, Order ID, Payment Amount, Payment Status, etc. Are correctly matched and transferred from one system to another without any mismatch.

Here's how it works:

- First, we identify what fields are present in the source system (e.g., the shopping cart).
- Then we check what fields are expected in the destination system (e.g., payment processor or invoice generator).
- We then map them so that the data flows correctly. For Example, the "Order ID" in the shopping cart matches with the "Transaction ID" in the payment gateway.
- If some formats are different (like date format or currency), we also define how to convert them properly.

This process ensures that customers' payment data, invoices, shipping addresses, and other order-related information are smoothly transferred between systems with zero confusion.

Data mapping helps in maintaining accuracy, avoids errors, and ensures a smooth flow of data between modules.

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 (Application Programming Interface) is like a bridge that allows two different software systems to talk to each other. It helps in exchanging data between systems in a structured and secure way, even if those systems are built using different technologies or formats.

Suppose Jewellery E-Commerce application stores order and delivery dates in **dd-mm-yyyy** format (like 25-07-2025), but we're receiving some data from an international logistics partner based in the **US**, where dates are formatted as **mm-dd-yyyy** (like 07-25-2025).

To handle this using API integration, here's how we proceed:

Step 1 – Establish API Communication

We integrate our jewellery app with the external system (e.g., courier service) using APIs so that both systems can exchange information like order status and delivery dates.

Step 2 – Convert Outgoing Dates

When sending data (like shipment requests or invoices) from our system to theirs, we take our local date format (dd-mm-yyyy) and convert it to their format (mm-dd-yyyy). **Example**: "25-07-2025" becomes "07-25-2025".

Step 3 – Parse Incoming Dates

When we receive order updates or delivery confirmations from their system in mm-dd-yyyy, we convert it back to our format before saving it into our database or showing it on the user screen.
Example: “07-30-2025” is parsed to “30-07-2025”.

Step 4 – Validate the Dates

We also put checks to make sure the converted dates are real and correct, like not accepting 31-07-2025 or wrong leap year data.

This helps avoid confusion in delivery schedules and ensures our users, vendors, and system all work on the same timeline.