Capstone Project 3 (Part 1)

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

**Q1. Draw a Use Case diagram**



**Q2. Derive Boundary Classes, Controller Classes, Entity Classes.**

Answer:

**1. Boundary Classes (User Interface Components)**

Boundary classes manage the interaction between the **system** and the **actor (user)**. These typically represent screens, forms, or UIs.

* **PaymentPage** – Interface where the customer selects the payment method.
* **CardPaymentUI** – Interface for inputting card details like card number, expiry date, CVV, etc.
* **WalletPaymentUI** – Interface that allows payment through digital wallets like Paytm or PhonePe.
* **CashPaymentUI** – Interface to confirm offline cash payment entry.
* **NetBankingUI** – Interface where the user logs into their bank account and authorizes the payment.

**2. Controller Classes (Business Logic Handlers)**

Controller classes handle the business rules and the flow of logic between boundary and entity classes.

* **PaymentController** – Manages overall payment logic and directs request based on selected method.
* **CardPaymentController** – Validates and processes card payment transactions securely.
* **WalletPaymentController** – Authenticates and processes wallet-based payments.
* **CashPaymentController** – Registers confirmation for cash transactions.
* **NetBankingController** – Handles net banking credentials and transaction flow securely.

**3. Entity Classes (Data & Business Rules)**

Entity classes represent **business objects**, store data, and contain business logic.

* **Payment** – A generic class capturing attributes like PaymentID, amount, date, and status.
* **CardPayment** – Stores card-specific data like cardholder name, card number, CVV, and expiry date.
* **Wallet** – Represents a digital wallet with balance tracking and transaction history.
* **CashTransaction** – Represents a cash-based payment, usually entered manually.
* **NetBankingTransaction** – Stores information related to bank account used, IFSC code, transaction ID, etc.

**Q3. Place these classes on a three-tier architecture**

Answer:

A **Three-Tier Architecture** separates the application into three logical layers:

1. **Presentation Layer**
2. **Business Logic Layer**
3. **Data Access Layer (Data Layer)**

Each class derived from the use case diagram is placed in one of these layers based on its responsibility.

**1. User Layer -**

This layer is responsible for interacting with the user. It contains **Boundary Classes**.

* PaymentPage
* CardPaymentUI
* WalletPaymentUI
* CashPaymentUI
* NetBankingUI

These classes collect input from the user and display output or error messages.

**2. Business Logic Layer (Application Layer) -**

This layer processes business logic and workflows. It contains **Controller Classes**.

* PaymentController
* CardPaymentController
* WalletPaymentController
* CashPaymentController
* NetBankingController

They receive input from the Presentation Layer, apply the rules, and call the appropriate Entity classes.

**3. Data Layer -**

This layer handles data storage, retrieval, and database communication. It contains **Entity Classes**.

* Payment
* CardPayment
* Wallet
* CashTransaction
* NetBankingTransaction

These classes represent data objects that are mapped to the database and handle data operations like insert, update, delete, and fetch.

**Q4. Explain Domain model for customer making payment through net banking**

Answer:



**Q5. Draw a sequence diagram for payment done by Customer Net banking**

Answer:

A Sequence Diagram is one of the most important types of interaction diagrams in the Unified Modeling Language (UML). It is used to model the dynamic behaviour of a system by showing how objects and components interact with each other in a sequentialorder over time to complete a specific task or functionality.



**Q6 Explain Conceptual Model for this case**

Answer:

A **Conceptual Model** is a high-level representation of the business objects (concepts) involved in a system. It is independent of implementation and focuses on understanding the structure and relationships of the entities from a business perspective.

In the given case, a customer can make a Payment using different Payment Methods such as Card, Wallet, Cash, or Net Banking. The conceptual model captures the entities involved and how they are related.

**Key Concepts (Entities):**

1. **Customer** – The user who initiates the payment.
2. **Payment** – The transaction made by the customer.
3. **Payment Method** – The method used to make the payment.
* Card
* Wallet
* Cash
* Net Banking
1. **Bank** – Involved in Net Banking payment method.

**Relationships:**

* A Customer can make one or many Payments.
* A Payment is made using one Payment Method.
* If Net Banking is used, the Payment is associated with a Bank.
* Each Payment Method is a specialization (subtype) of the general Payment Method class

**Q7. What is MVC architecture? Explain MVC rules to derive classes from use case diagram and guidelines to place classes in 3-tier architecture**

Answer:

**1. What is MVC Architecture?**

**MVC** stands for **Model-View-Controller**. It is an architectural pattern used to separate concerns in software applications, especially in web and desktop development. The primary goal of MVC is to divide an application into three interconnected components:

* **Model** – Represents the business logic and data (e.g., database operations, calculations).
* **View** – Represents the user interface (UI) that displays data to the user.
* **Controller** – Manages input from the user and updates the model and view accordingly.

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

The **Model-View-Controller (MVC)** pattern helps in systematically identifying classes from a **Use Case Diagram** by mapping each element to a specific type of class:

1. **Actors → Boundary Classes (View Layer)**
	* Actors in use case diagrams represent users or external systems.
	* These translate into **Boundary Classes**, which handle interaction between users and the system.

 Example: LoginPage, PaymentScreen, UserDashboard.

1. **Use Cases → Controller Classes (Logic Layer)**
	* Each use case describes a system function or process.
	* These map to **Controller Classes**, which manage workflows and coordinate between view and model.

 Example: LoginController, PaymentController, OrderManager.

1. **Business Concepts / Nouns → Entity Classes (Data Layer)**
	* Nouns and data elements in use case narratives point to **Entity Classes**, which hold business data.

Example: Customer, Payment, BankAccount, Order.

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

Once classes are derived, we place them in the respective tiers of a **3-Tier Architecture**, which separates concerns for better design and maintenance.

**1. Presentation Tier (UI Layer)**

* Contains **Boundary Classes**.
* Handles user input and output (UI screens, forms).

Classes: LoginPage, PaymentForm, DashboardView.

**2. Application Tier (Business Logic Layer)**

* Contains **Controller Classes**.
* Coordinates business logic, validates input, and connects UI to data.

 Classes: PaymentController, UserManager, OrderProcessor.

**3. Data Tier (Persistence Layer)**

* Contains **Entity Classes**.
* Manages data storage, retrieval, and mapping to database.

Classes: Customer, Payment, Transaction, Bank.

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

Answer:

|  |  |  |
| --- | --- | --- |
| **Stage** | **Activities** | **Artifacts & Resources** |
| **Pre-project** | - Understand business need | - Business Case |
| - Identify stakeholders | - Feasibility Report |
| - Conduct feasibility study | - Stakeholder Register |
| **Planning** | - Assist in project scoping | - BA Plan |
| - Define BA approach | - Risk Register |
| - Identify risks and constraints | - Communication Plan |
| **Project Initiation** | - Conduct stakeholder meetings | - Project Charter |
| - Understand project goals | - Stakeholder Analysis |
| - Support project charter creation | - High-Level Requirements |
| **Requirements Gathering** | - Conduct interviews, workshops | - Requirements Document |
| - Capture functional & non-functional requirements | - Use Case Diagram |
|   | - Process Flows |
| **Requirements Analysis** | - Analyse and validate requirements | - BRD, FRD, SRS |
| - Resolve conflicts | - Requirement Traceability Matrix (RTM) |
| - Prioritize requirements |   |
| **Design** | - Support design team | - Wireframes |
| - Review design documents for requirement coverage | - UI Mockups |
|   | - Data Flow Diagrams |
| **Development** | - Clarify requirements to developers | - Updated RTM |
| - Support functional queries | - Change Requests (if any) |
| **Testing** | - Support test case preparation | - Test Plan |
| - Validate test coverage | - Test Scenarios |
| - Participate in defect triaging | - Defect Logs |
| **UAT (User Acceptance)** | - Facilitate UAT planning | - UAT Scripts |
| - Support users during UAT | - Sign-off Document |
| - Collect feedback and sign-off | - UAT Feedback Report |

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

Answer:

* **What is Conflict Management?**

Conflict management is the process of identifying and resolving disagreements or disputes between individuals or groups in a constructive and effective manner. It involves communication, problem-solving, and negotiation to ensure the conflict does not negatively impact the project's goals, timeline, or team collaboration.

* **What is Thomas–Kilmann Technique?**

The **Thomas–Kilmann Conflict Mode Instrument (TKI)** is a widely used tool to understand how different people handle conflict. It identifies **five conflict-handling styles** based on two behavioral dimensions:

* **Assertiveness** – The extent to which a person tries to satisfy their own concerns.
* **Cooperativeness** – The extent to which a person tries to satisfy the other person’s concerns.

Based on these, the five conflict-handling styles are:

| **Style** | **Description** |
| --- | --- |
| **1. Competing** | High assertiveness, low cooperativeness. One party seeks to win. |
| **2. Collaborating** | High assertiveness, high cooperativeness. Find a win-win solution. |
| **3. Compromising** | Moderate assertiveness and cooperativeness. Both parties give up something. |
| **4. Avoiding** | Low assertiveness and cooperativeness. Delays or avoids the conflict. |
| **5. Accommodating** | Low assertiveness, high cooperativeness. One party gives in to the other. |

* **5 Steps of Conflict Management Process**
1. **Identify the Conflict**
Understand the source and parties involved in the disagreement.
2. **Understand Everyone’s Interests**
Listen actively to all stakeholders and understand their concerns and expectations.
3. **Evaluate Possible Solutions**
Brainstorm and analyze potential solutions that meet everyone’s interests.
4. **Select the Best Solution**
Choose a resolution strategy (e.g., compromise or collaboration) based on the situation.
5. **Implement and Follow-up**
Ensure the chosen solution is executed and review the outcome to avoid future conflicts.

**Q10. List down the reasons for project failure**

Answer:

Project failure can occur due to multiple reasons, ranging from poor planning to lack of communication. Below are the key reasons:

 **1. Poor Requirements Gathering**

* Incomplete, unclear, or misunderstood requirements lead to building the wrong product.
* Lack of stakeholder involvement during requirement elicitation is a major cause.

 **2. Scope Creep (Uncontrolled Changes in Scope)**

* Frequent additions or changes to requirements without proper evaluation and approval.
* Leads to delays, increased costs, and reduced quality.

 **3. Ineffective Project Planning and Scheduling**

* No clear roadmap or unrealistic timelines can derail progress.
* Lack of risk planning, buffer time, or milestones causes chaos.

**4. Communication Gaps**

* Miscommunication between stakeholders, project managers, developers, and testers.
* Leads to misunderstandings, duplicated efforts, and missed expectations.

 **5. Inadequate Risk Management**

* Failing to identify and mitigate risks in the early phase.
* Unexpected issues can impact time, cost, and performance.

 **6. Lack of Stakeholder Engagement**

* Stakeholders not being involved or responsive leads to incorrect direction and missed requirements.
* Their buy-in is crucial for project acceptance.

 **7. Weak Leadership or Inexperienced Team**

* Project manager without leadership skills or team lacking technical competence.
* Affects decision-making, coordination, and execution.

 **8. Budget Overruns**

* Incorrect cost estimation or poor resource utilization.
* Can cause project delays, quality compromise, or complete halt.

 **9. Poor Quality Control**

* Inadequate testing or ignoring quality assurance practices.
* Results in bugs, rework, and reduced customer satisfaction.

 **Q11. List the Challenges faced in projects for BA**

Answer:

1. **Unclear or Changing Requirements**
* Frequent changes or poorly defined requirements make it difficult to maintain clarity and consistency throughout the project.
1. **Managing Stakeholder Expectations**
* Stakeholders may have different or unrealistic expectations, making it challenging to ensure alignment and satisfaction.
1. **Scope Creep and Scope Management**
* Unauthorized or unplanned changes in project scope can affect timelines, cost, and overall project success.
1. **Time and Resource Constraints**
* Limited time and inadequate resources often hinder thorough analysis, documentation, and validation efforts.
1. **Quality Assurance and Testing**
* Ensuring that the developed solution meets business needs and is thoroughly tested as per requirements can be a major challenge.
1. **Documentation and Knowledge Management**
* Maintaining up-to-date, accurate documentation and sharing knowledge across teams is often overlooked, leading to gaps.
1. **Technology Constraints and Complexity**
* Adapting to complex or unfamiliar technologies may slow down requirement analysis and communication with technical teams.

 **Q12. Write about Document Naming Standards**

Answer:

Document naming standards are structured guidelines used to consistently name project-related documents. These standards help in easy identification, organization, version control, and retrieval of documents throughout the project lifecycle.

**Purpose of Using Naming Standards:**

* To maintain clarity and consistency across all documents.
* To easily track document versions and updates.
* To ensure smooth collaboration among stakeholders.
* To support effective documentation management.

**Typical Elements in a Naming Convention:**

1. **Project Identifier** – Unique code or ID for the project.
2. **Document Type** – Short form of the document category (e.g., REQ for Requirements).
3. **Version Number** – Indicates the version of the document (e.g., 1.0).
4. **Date** – The date of document creation or revision.

**Example:**
Suppose we have a project with the ID **"PROJ123"**, and we are working with a **Requirements Specification Document**, with:

* Project ID: PROJ123
* Document Type: REQ
* Version: 1.0
* Date: 2025-04-10

Then the document identifier would be:
 **PROJ123-REQ-1.0-2025-04-10**

 **Q13. What are the Do’s and Don’ts of a Business analyst**

Answer:

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **DO'S** | **DON'TS** |
| 1 | Consult an SME for clarifications in requirements. | Never say NO to the client. |
| 2 | Go to the client with a plain mind with no assumptions. Listen carefully and completely until the client is done, and then you can ask queries. | There is no word as “By default”. |
| 3 | Try to extract maximum leads to the solution from the client himself. | Never imagine anything in terms of GUI. |
| 4 | Concentrate on the important requirements. | Do not interrupt the client when he is giving you the problem. |
| 5 | Question the existence of existence/ Question everything. | Never try to give solutions to the client straight away with your previous experience and assumptions. |

 **Q14. Write the difference between packages and sub-systems**

Answer:

**Packages** are collections of components that are **not reusable**, meaning they are built for a specific system or purpose and cannot be easily used in other systems. These are often tightly coupled with the application they are developed for. Companies that provide **custom software solutions**, such as small IT service firms or freelance developers, usually work with packages. Since every client has unique requirements, the focus is on quick delivery rather than reusability.

**Sub-Systems** are **reusable collections of components**. They are designed in a way that allows them to be integrated into various systems with minimal changes. Sub-systems follow standardized interfaces and can operate independently. **Product-based companies** or **large-scale enterprises** that build software platforms prefer using sub-systems. For example, a company like SAP or Salesforce may develop a sub-system for user management or data analytics that can be reused across multiple products or client solutions.

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

Answer:

**Camel-casing** is a style of writing compound words or phrases where each word is joined without spaces, and every word **after the first** starts with a **capital letter**. The first word remains in **lowercase**. This format makes long variable or method names easier to read.

 **Example:**

* employeeId
* loginStatus
* getCustomerInfo

This style is called **camel-casing** because the capital letters in the middle resemble the humps of a camel.

 **Where is it used?**

Camel-casing is widely used in **software development** for:

* Naming **variables** in code (e.g., userEmail, orderNumber)
* Naming **functions or methods** (e.g., fetchData(), sendNotification())
* Creating **object property names** in many programming languages like **Java, JavaScript, C++, and Python**.

 **Purpose of Using Camel-Case:**

* Increases **clarity and readability** in code.
* Makes it easier to understand long names without using separators like underscores.
* Follows the **conventions** recommended by many programming languages and frameworks.

 **Q16. Illustrate development server and what are the accesses does business analyst has?**

Answer:

**Development Server –**

A **Development Server** is a server environment used by the development team to **build, develop, and test** the application or software. It is an internal environment where developers write code, debug, and integrate different modules of the project. It is **not accessed by the client or end users**.

This environment is usually the **first stage** in the software development lifecycle before the code moves to staging or production servers.

**Access for Business Analyst (BA):**

A **Business Analyst has limited access** to the development server because the BA’s primary role is not technical but functional.

Here is what the BA typically can do:

* **Read-only access** to the front-end screens or modules to verify if development aligns with business requirements.
* **View UI prototypes or early builds** to confirm screen flow and feature implementation.
* **Access to demo versions** of the application for walkthroughs or internal reviews.
* **Log issues or bugs** found during observation using tools like Jira or Trello.
* **Verify requirement coverage** (i.e., check if user stories or functional points are reflected in the current build).

The BA is **not allowed to modify code**, change configurations, or interfere with backend development activities.

**Q17. What is Data Mapping**

Answer:

**Data Mapping** is the process of **matching data fields from one source to another**. It helps in connecting data from different systems, formats, or databases so that the information can be transferred, integrated, or transformed correctly.

In other words, data mapping defines how data from a **source** (like a user input form, database, or external system) will be **converted and stored** in a **target** system (like another database or software).

**Why is Data Mapping Needed?**

In real-world systems, data is stored in different formats, fields, and database structures. When moving data between systems (like from an old software to a new one), the data cannot simply be copied—it must be **mapped** to ensure that:

* Correct data reaches the right place.
* No data is lost or misinterpreted.
* Data types and formats are properly aligned.
* The business logic is preserved.

**Where is Data Mapping Used?**

* **Data Migration**: Moving data from old systems to new systems.
* **System Integration**: Connecting different applications or databases.
* **ETL Processes** (Extract, Transform, Load): Preparing data for analysis or storage.
* **API Integration**: Sending/receiving data between platforms.
* **Business Intelligence/Reporting**: Standardizing data for reports and dashboards.

 **Benefits of Data Mapping:**

1. Ensures **data accuracy and consistency**.
2. Helps avoid **data duplication or mismatch**.
3. Supports **automation** in data transformation.
4. Enables **smooth system upgrades or transitions**.
5. Helps in maintaining **data quality and compliance**.

**Example:**

If you are transferring customer data from an old system to a new one:

* Source field: cust\_name → Target field: customerFullName
* Source field: dob → Target field: dateOfBirth
* Source field: addr → Target field: address

**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**

Answer:

An **API (Application Programming Interface)** is a **set of rules and protocols** that allows two different software applications to **communicate and exchange data** with each other. It acts as a **bridge** between systems, enabling them to send requests and receive responses, even if they are built on different platforms or technologies.

APIs are commonly used to:

* Access third-party services (like payment gateways, maps, weather data).
* Share data between systems (e.g., web and mobile apps).
* Integrate multiple applications to work together seamlessly.

**API Integration – Date Format Scenario**

Let’s consider a real-life case:

You have built an application where the **date format is dd-mm-yyyy** (for example, **India’s format**).
Now, your system receives **data from a US-based application** via an **API**, and their format is **mm-dd-yyyy**.

This can lead to **confusion or errors**, especially when days and months are less than 12 (e.g., 04-05-2023 could be April 5th or May 4th).

**How to Use API Integration in This Case**

To avoid errors and ensure correct data handling, the API integration must **include a date format conversion process**. Here is how it would work step-by-step:

**Step 1: Connect with the US Application’s API**

Your system sends a request to the US application using their API. The API returns a response that includes user data like name, address, and date of birth in **mm-dd-yyyy** format.

**Step 2: Identify the Format**

The backend (API handler or middleware) knows the **incoming date is in mm-dd-yyyy** format.

**Step 3: Convert the Format**

Use code to **convert mm-dd-yyyy to dd-mm-yyyy** before saving to your system.

In programming languages like Python, JavaScript, or Java, you can use **date libraries** for format conversion. For example:

* From "11-25-1995" (US format)
* Convert to "25-11-1995" (Indian format)

**Step 4: Store or Display Correctly**

After conversion, store the date in the format your system accepts or display it accordingly to your users.

**Step 5: Maintain Consistency in API Documentation**

Mention in your API documentation:

* What date format your system expects (dd-mm-yyyy)
* Whether the integration supports automatic format conversion
* Any validation/error handling if incorrect date format is received

 **Conclusion:**

APIs allow applications to interact and share data effectively. In your application, if there is a mismatch in date formats between your system and the US-based system, API integration should include a **format conversion logic** to ensure accuracy. This approach avoids confusion, ensures correct data storage, and supports global compatibility.