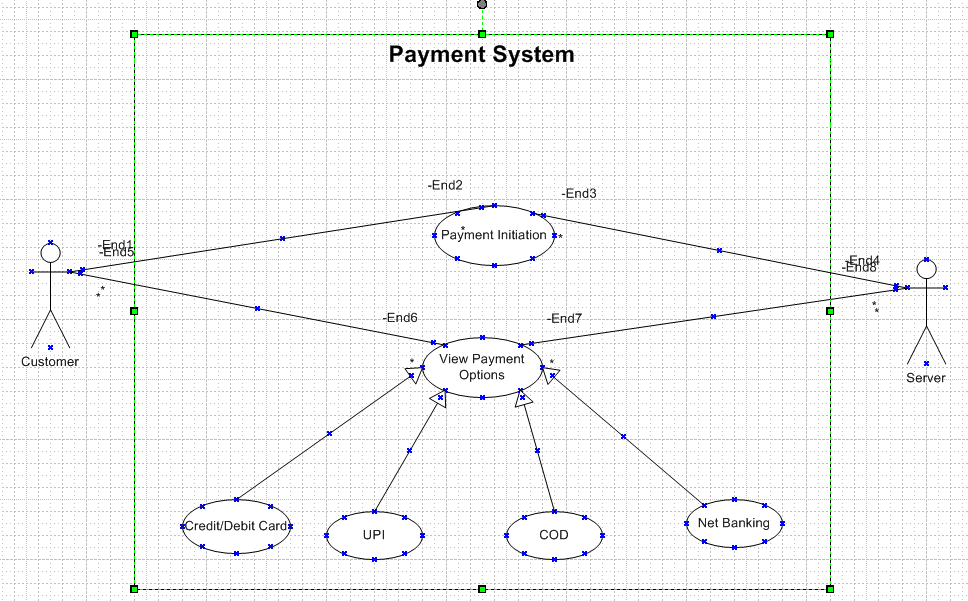
**CAPSTONE PREP – 3**

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

**Draw a Use Case Diagram**

****

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

Boundary Classes (UI/Interface Layer)

Boundary classes handle user interactions and act as interfaces between the external world and the system.

* Payment - Handles customer interactions related to payments.
* CardPayment - Handles card-specific interactions (like entering card details).
* Wallet Payment - Handles wallet-related transactions.
* Cash Payment - Handles cash payments.
* Net Banking Payment - Manages net banking payment interactions.

Controller Classes (Business Logic layer)

Controller classes process user requests, coordinate business logic, and interact with entity classes.

* Payment - Controls the payment flow.
* Card Payment - Handles payment logic for cards.
* Wallet Payment - Manages wallet payments.
* Cash Payment - Validates and confirms cash payments.
* Net Banking - Manages net banking transactions.

Entity Classes (Data Layer)

Entity classes represent the core business objects that store data.

* Payment - Stores common payment details (amount, status, method).
* Card Payment - Extends Payment, includes card number, expiry, CVV.
* Wallet Payment - Extends Payment, includes wallet ID, balance.
* Cash Payment - Extends Payment, includes cash receipt details.
* Net Banking - Extends Payment, includes bank details and transaction ID.
* Customer - Represents the user making the payment.
* Transaction - Logs payment transactions.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Class Type** | **Class Name** | **Description** |
| **Database layer** | Entity Class | Payment | Represents payment entity in database |
|  | Entity Class | Card Payment | Represents payments made via Card. |
|  | Entity Class | Wallet Payment | |  | | --- | |  |  |  | | --- | | Represents payments made via Wallet. | |
|  | Entity Class | CashPayment | Represents payments made via Cash. |
|  | Entity Class | NetBankingPayment | Represents payments made via Net Banking. |
| **Application Layer** | Boundary Class | PaymentService | Provides services for processing payments. |
|  | Boundary Class | PaymentRepository | Handles database operations for payments. |
| **Business Logic Layer** | Controller Class | PaymentController | Handles API requests related to payments. |
|  | Controller Class | CardPayment | Handles Card Payment processing. |
|  | Controller Class | WalletPayment | |  | | --- | |  |  |  | | --- | | Handles Wallet Payment processing. | |
|  | Controller Class | CashPayment | Handles Cash Payment processing. |
|  | Controller Class | NetBankingPayment | Handles Net Banking Payment processing. |

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

A domain model is a conceptual representation that defines the structure, relationships and behaviours of entities within a specific problem domain

**Entities & Attributes**

1. Customer

* CustomerID
* Name
* Email
* PhoneNumber
* Address

1. Payment

* PaymentID
* CustomerID
* Amount
* Currency
* PaymentStatus
* Timestamp

1. PaymentMethod

* MethodID (Unique Identifier)
* MethodType
* PaymentID

1. Bank Account

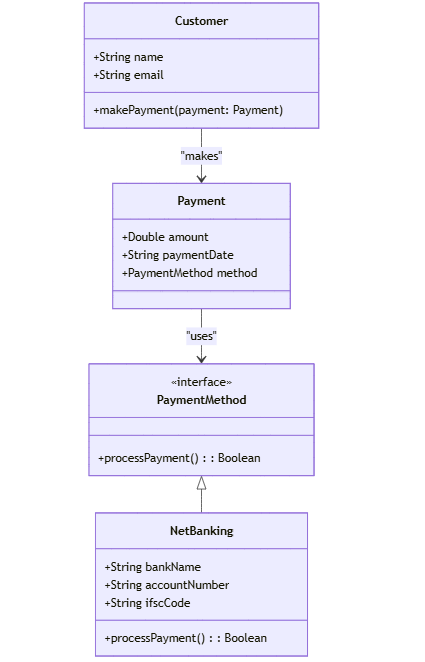
* BankAccountID
* CustomerID
* BankName
* AccountNumber

1. NetBankingTransaction

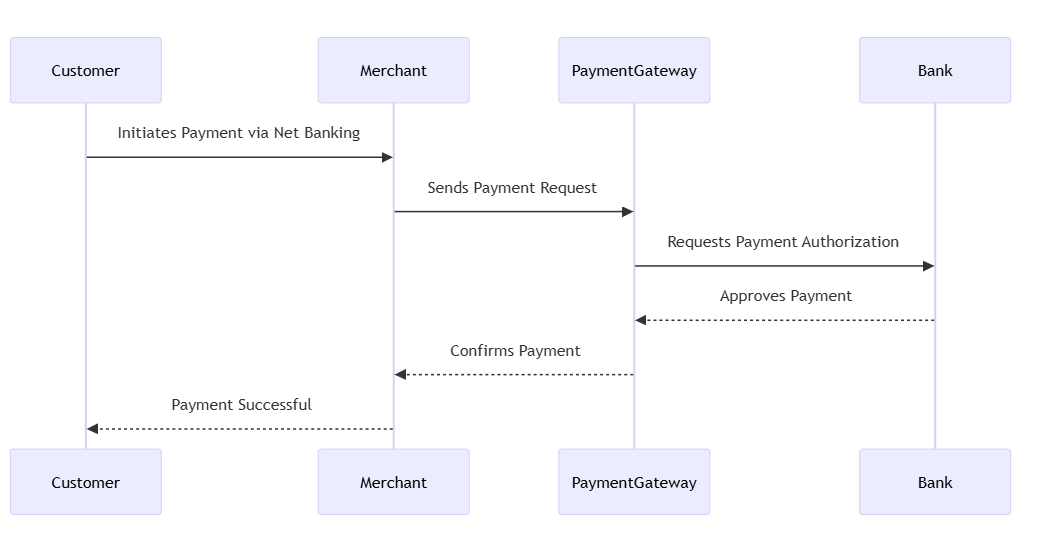
* TransactionID
* BankAccountID
* PaymentID
* TransactionStatus

**Relationships**

* A Customer can make multiple Payments.
* Each Payment is associated with one Payment Method.
* Net Banking Payment Method requires aBank Account.
* A Bank Account belongs to a Customer.
* A Net Banking Transaction links a Bank Account to a Payment.

****

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

****

1. **Explain Conceptual Model for this Case**

A conceptual model represents the high-level structure of the system, focusing on key entities and their relationships without delving into implementation details.

Entities and Relationships

* Customer
* A customer initiates a payment.
* Can have multiple payment options available.
* Payment
* Represents a transaction initiated by the customer.
* Linked to exactly one payment method.
* Payment Method
* A payment can be made using one of the following methods:

Card (Credit/Debit)

Wallet (Digital wallets like Paytm, Google Pay, etc.)

Cash

Net Banking

Each payment method has unique attributes (e.g., card number for Card, wallet provider for Wallet).

Example Conceptual Schema

* Customer (CustomerID, Name, Email, Phone)
* Payment (PaymentID, CustomerID, Amount, Date, Status, Payment MethodID)
* PaymentMethod (Payment MethodID, Type)
* Card (Payment MethodID, Card Number, Expiry Date, CVV, Card Holder Name)
* Wallet (Payment MethodID, Wallet Provider, Wallet ID)
* Cash (Payment MethodID)
* Net Banking (Payment MethodID, BankName, AccountNumber)
* A Customer makes a Payment.
* A Payment is processed using one Payment Method.
* The Payment Method is a generalization that includes Card, Wallet, Cash,or Net Banking.
* Each payment method has specific attributes relevant to its type.

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

MVC (Model-View-Controller) is a software design pattern used to separate concerns in an application, making it more maintainable and scalable. It divides the application into three main components:

* Model: Represents the data and business logic.
* View: Handles the user interface and presentation.
* Controller: Manages user input, interacts with the model, and updates the view.

MVC Rules to Derive Classes from a Use Case Diagram

To derive classes from a use case diagram, follow these rules:

* Identify the Actors and Use Cases
* Actors represent external entities (e.g., Customer, Payment System).
* Use cases represent functionalities (e.g., Make Payment).
* Extract the Key Entities
* Identify nouns from the use case description (e.g., Payment, Customer, Wallet, Card, Cash, Net Banking).
* These entities become model classes.
* Determine the Controller Classes
* Identify verbs that represent actions (e.g., process payment, validate transaction).
* These actions become methods in controller classes.
* Define View Components
* Identify screens/UI elements required for user interaction (e.g., Payment Page, Confirmation Page).

Guidelines to Place Classes in a 3-Tier Architecture

A 3-tier architecture consists of:

* Presentation Layer (UI Layer)
* Contains views that interact with the user (e.g., HTML pages, Mobile UI).
* Example: Payment Page, Transaction Confirmation Page.
* Business Logic Layer (Service Layer)
* Contains business rules, validations, and service classes.
* Example: Payment Service (handles processing logic), Transaction Validator.
* Data Access Layer (Persistence Layer)
* Manages database operations.
* Example: Payment DAO, Transaction Repository.

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

|  |  |  |
| --- | --- | --- |
| **Stage** | **Activities** | **Artifact and Resources** |
| Pre-Project | - Identify business need for payment options - Conduct feasibility study - Prepare business case | - Business Case Document - Feasibility Study Report - Stakeholder Identification |
| Planning | - Define project scope, objectives, and deliverables - Identify stakeholders and dependencies - Create high-level project plan | - Project Charter - Stakeholder Register - High-Level Project Plan |
| Project Initiation | - Conduct stakeholder meetings to gather initial expectations - Define roles and responsibilities of team | - Project Scope Statement - RACI Matrix - Initial Risk Assessment |
| Requirement Gathering | - Conduct workshops, interviews, and surveys to gather payment requirements - Define functional and non-functional requirements - Identify regulatory compliance requirements | - Business Requirement Document (BRD) - Stakeholder Requirements Document (SRD) - Meeting Notes |
| Requirement Analysis | - Analyze payment options (Card, Wallet, Cash, Net Banking) - Prioritize requirements based on business needs and feasibility - Identify integration needs with banking/payment gateways | - Requirement Traceability Matrix (RTM) - Process Flow Diagrams - Gap Analysis Report |
| Design | - Collaborate with technical teams to design payment system architecture - Define user workflows for each payment method - Document UI/UX wireframes and API specifications | - Functional Specification Document (FSD) - System Design Document (SDD) - Wireframes & Prototypes |
| Development | - Clarify business logic and handle requirement changes - Assist developers in understanding payment flows - Ensure compliance requirements are implemented | - Updated RTM - Change Request Document (if any) - Payment Processing Logic |
| Testing | - Validate test cases for payment transactions - Support SIT (System Integration Testing) and verify compliance - Perform defect analysis and requirement validation | - Test Cases & Test Scenarios - Defect Logs - Test Execution Reports |
| UAT | - Conduct UAT sessions with business users - Ensure all payment methods function as expected | - UAT Test Cases & Results - UAT Sign-off Document - User Feedback Report |

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

Conflict management is the process of identifying, addressing, and resolving conflicts in a constructive manner to maintain a healthy work or personal environment. It involves strategies that help minimize the negative impact of conflicts while fostering collaboration and understanding.

The Thomas-Kilmann technique is a widely used framework for conflict resolution, developed by Kenneth Thomas and Ralph Kilmann. It categorizes conflict-handling behavior based on two 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 concerns of others.

Based on these dimensions, the model identifies five conflict-handling styles:

* Identify the Conflict
* Observe the situation and recognize the signs of conflict.
* Understand the emotions and concerns involved.
* Identify the people affected by the issue.
* Discuss the details
* Allow both parties to express their viewpoints.
* Encourage open communication and active listening.
* Avoid blame and focus on facts.
* Agree on the root problem
* Analyze the underlying cause of the conflict.
* Ensure both parties agree on what the real issue is.
* Differentiate between personal issues and professional disagreements.
* Check for every possible solution
* Brainstorm multiple resolutions.
* Evaluate the pros and cons of each option.
* Consider compromises or new alternatives.
* Negotiate the solution to avoid Future conflicts
* Agree on a mutually beneficial solution.
* Set clear guidelines for future similar issues.
* Document the decision to prevent recurrence.

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

* Poor Planning and Requirements Management
* Unclear project scope (scope creep)
* Vague or incomplete requirements
* Lack of proper project documentation
* Inadequate Leadership and Stakeholder Engagement
* Weak project management
* Lack of executive support
* Poor communication among stakeholders
* Poor Resource Management
* Insufficient budget allocation
* Inadequate human resources or skill gaps
* Poor time management and unrealistic deadlines
* Ineffective Risk Management
* Failure to identify potential risks
* No contingency plans
* Ignoring early warning signs
* Technical Challenges
* Use of outdated or incompatible technology
* Poor system integration
* Insufficient testing and quality assurance
* Unrealistic Expectations
* Overpromising results to stakeholders
* Underestimating complexity and effort required
* Lack of alignment between business goals and project objectives
* Lack of User or Customer Involvement
* Not gathering feedback from end-users
* Delivering a solution that does not meet business needs
* Ignoring user experience (UX) considerations

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

* Requirement-Related Challenges
* Unclear or Changing Requirements: Stakeholders often struggle to define their needs, leading to scope creep.
* Conflicting Stakeholder Expectations: Different departments may have opposing priorities.
* Incomplete Requirements Gathering: Missing critical details can cause issues later.
* Stakeholder Management Challenges
* Unavailability of Stakeholders: Key stakeholders may not be accessible for input.
* Resistance to Change: Users may resist new processes or technologies.
* Miscommunication: Business and technical teams often have different terminologies.
* Technical and Process Challenges
* Lack of Technical Knowledge: A BA may struggle to bridge the gap between business and IT.
* Poor Documentation Standards: Unclear documentation can lead to misunderstandings.
* Integration with Legacy Systems: Compatibility issues can arise with old software.
* Project Management Challenges
* Tight Deadlines & Resource Constraints**:** Limited time and budget can impact quality.
* Scope Creep: Constantly changing requirements can affect timelines and costs.
* Poor Risk Management: Lack of anticipation for risks can lead to project failure.
* Communication and Collaboration Issues
* Lack of Stakeholder Engagement: Some stakeholders may not actively participate.
* Difficulty in Translating Business Needs to Technical Teams: Ensuring alignment is challenging.
* Cultural and Geographical Barriers: Distributed teams may face collaboration challenges.

1. **Write about Document Naming Standards**

Document Naming Standards are a set of guidelines that help organizations maintain consistency, clarity, and efficiency in naming files and documents. A well-defined naming convention ensures that documents are easy to locate, understand, and manage, reducing confusion and errors in collaborative environments.

Example:

Project ID: PRJ123

Document Type: REQ

Version: v1.0

Date: 13-02-2025

Document Identifier: PRJ123-REQ-v1.0-20250210

This format ensures that all stakeholders can quickly identify the document's purpose, version, and relevance.

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

Do’s for a Business Analyst:

* Understand Business Needs Clearly
* Conduct thorough stakeholder interviews.
* Analyze business processes to identify pain points and areas for improvement.
* Communicate Effectively
* Use clear, concise, and structured communication.
* Actively listen to stakeholders and clarify requirements.
* Document Everything
* Maintain detailed documentation (BRD, FRD, Use Cases, User Stories, etc.).
* Ensure all requirements are traceable and well-organized.
* Collaborate with Stakeholder
* Engage with business users, developers, testers, and project managers.
* Facilitate workshops and brainstorming sessions.
* Think Critically and Analytically
* Break down complex problems into smaller, manageable parts.
* Use data-driven decision-making.

Don’ts for a Business Analyst

* Don’t Assume Requirements Without Validation
* Don’t Neglect End Users
* Don’t Overcomplicate Solutions
* Don’t Ignore Documentation
* Don’t Work in Isolation
* Don’t Delay Stakeholder Engagement
* Don’t Disregard Risks
* Don’t Ignore Feedback

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

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Package** | **Subsystem** |
| Definition | |  | | --- | |  |  |  | | --- | | A logical grouping of related classes, interfaces, and sub-packages within a software system. | | A larger, self-contained module within a system that provides a distinct functionality, often comprising multiple packages. |
| Scope | Typically smaller in scope, used to organize code efficiently within a system. | Broader in scope, representing a major functional component of a system. |
| Encapsulation | Helps in organizing classes to avoid name conflicts and improve code maintainability. | Encapsulates a set of related functionalities, services, or components. |
| Implementation | Found in programming languages like Java | Implemented as a part of system architecture, often involving multiple packages |
| Example | A networking package in a software application. | A "Payment Processing" sub-system in an e-commerce application |