**Capstone Project 3 Part 1**

Submitted by: Bhavana Padol

Batch: 10th Aug 2024

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**

**Answer:**



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

**Answer:**

**Boundary Classes (User Interface):** These represent interaction points between users and the system. They handle inputs from users and outputs to users, such as the UI screens and forms.

* **Login Screen**: Handles user login.
* **Registration Screen**: Manages the sign-up process.
* **Order Page**: Displays available food items for customers.
* **Tracking Page**: Displays the real-time tracking information for an ongoing delivery.

**Controller Classes**: These classes are responsible for coordinating activities between boundary and entity classes. They process user input, invoke actions on entity classes, and manage the flow of data between the user interface and the business logic.

* **Use Account Controller**: Manages user registration, login, and profile updates.
* **Order Controller**: Manages placing orders, payment processing, and status updates.
* **Deliver Controller**: Coordinates between the order system and the tracking module.
* **Notification Controller**: Manages sending real-time notifications to users.

**Entity Classes (Business Logic)**: These represent the core data and business logic. They encapsulate the system's state and business rules.

* **User**: Represents customer details, such as name, email, address, and preferences.
* **Order**: Captures details of food orders, such as items, quantities, prices, and order status.
* **Delivery**: Represents the delivery process, including the delivery person’s details and tracking information.
* **Restaurant**: Represents a food provider with details of menu items, ratings, and operating hours.

**Q3. Place these classes on a three tier Architecture.**

**Answer:**

1. **Presentation Layer:**

**Boundary Classes** **(User Interface)**

* **Login Screen**: Handles user login.
* **Registration Screen**: Manages the sign-up process.
* **Order Page**: Displays available food items for customers.
* **Tracking Page**: Displays the real-time tracking information for an ongoing delivery.

1. **Business Logic Layer:**

**Controller Classes**

* **Use Account Controller**: Manages user registration, login, and profile updates.
* **Order Controller**: Manages placing orders, payment processing, and status updates.
* **Deliver Controller**: Coordinates between the order system and the tracking module.
* **Notification Controller**: Manages sending real-time notifications to users.

1. **Data Layer (Database Layer):**

**Entity Classes (Business Logic)**:

* **User**: Represents customer details, such as name, email, address, and preferences.
* **Order**: Captures details of food orders, such as items, quantities, prices, and order status.
* **Delivery**: Represents the delivery process, including the delivery person’s details and tracking information.
* **Restaurant**: Represents a food provider with details of menu items, ratings, and operating hours.

**Q4. Explain Domain Model for Customer making payment through Net Banking**

**Answer:**

Domain model is similar to the entity relationship model.

The tables are connected to each other.

In the below diagram,

1. The customer table is connected to bank table, which is why the customer is able to make payment.
2. Customer table is also connected to payment table, because he should make the payment.
3. Now the payment is done by net banking, so payment table is connected to net banking table.
4. The account is in the bank, so the account table is connected to the bank table.
5. The authentication table is connected to both net banking table and bank table, because authentication is to performed there.
6. 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**

**Answer:**



**Q6. Explain Conceptual Model for this Case**

**Answer:**

A conceptual model is a high-level representation of a system that helps in understanding, visualizing, and communicating the essential aspects of a domain. It provides a clear and simplified view of the domain, making it easier to understand.

**Key Elements of the Conceptual Model:**

1. **Entities**:
   * Customer
   * Payment
   * Order
   * Payment Methods (Card, Wallet, Cash, Net Banking)
2. **Attributes**:
   * Customer Id: Unique identifier for the customer.
   * Payment Method: Chosen payment type (Card, Wallet, Cash, Net Banking).
   * Amount: Total amount to be paid.
   * Order Id: Unique identifier for the order.
3. **Relationships**:
   * Customer makes a Payment using one of the available Payment Methods.
   * Customer places an Order and completes it by making a Payment.

**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:** MVC (Model-View-Controller) is a software architectural pattern that separates an application into three main components:

1. **Model**: Represents the data and business logic. It is responsible for managing the application's data, logic, and rules.
2. **View**: Displays the data (UI/Presentation layer). It is responsible for rendering the user interface and presenting the data from the model to the user.
3. **Controller**: Acts as an intermediary between the Model and the View. It processes the user input, interacts with the model, and updates the view accordingly.

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

1. **Model**:

* Extract classes that manage the core logic and data from the **use case diagram**.
* These classes represent real-world entities or concepts that need to be maintained in the system (e.g., Customer, Payment, Order).
* Derive business logic entities and ensure their behaviour is encapsulated in the Model layer.

1. **View**:

* Identify the **UI elements** that need to interact with the user based on the use case interactions.
* Use the **boundary classes** or UI-related components from the use case diagram to determine the elements needed in the View layer.
* The View will receive data from the Model and present it to the user.

1. **Controller**:

* The **controller classes** are derived from the processes triggered by user actions in the use case diagram.
* These classes respond to user input, communicate with the Model to update data, and control the flow of data to the View.

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

1. **Presentation Layer (View):**

* This layer is nothing but a user interface.
* View is inside this layer.
* These classes interact directly with the user.
* Presentation layer is responsible for displaying information and also receiving the input from the user.

1. **Business Logic Layer (Controller):**

* This layer is nothing but business logic.
* Model and controller are inside this layer.
* Controller handles the user input, process the request and co-ordinates interaction between the model and view.

1. **Data Layer (Model):**

* Classes which are responsible for data access and storage are in this layer.
* It contains the classes which interacts with the database, files and other data sources.

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

**Answer:**

|  |  |  |
| --- | --- | --- |
| **Stage** | **Activities** | **Artifacts & Resources** |
| Pre-project | Initial meetings with stakeholders, identify high-level business goals, and gather input for a potential project. | Meeting notes, Initial business case |
| Planning | BA helps define project scope, objectives, resources, and deliverables. | Project Charter, High-level scope document, Risk assessment |
| Project Initiation | Detailed requirement scoping and analysis. The BA coordinates with stakeholders to ensure all expectations are captured. | Project Charter (detailed), Requirements Management Plan, Stakeholder Register |
| Requirements Gathering | BA collects functional and non-functional requirements through workshops, interviews, and surveys. | Business Requirements Document (BRD), Functional Requirements Document (FRD), Use Cases |
| Requirements Analysis | BA analyzes the gathered requirements, refines them, and ensures clarity and feasibility. | Refined BRD/FRD, Use Case Diagrams, Traceability Matrix |
| Design | Work with system architects to ensure the solution aligns with the documented business needs. Review technical designs. | System Design Document, Review Notes, Requirements Traceability Matrix |
| Development | Provide clarifications to the development team about business requirements. Review progress to ensure requirements are being met. | Clarification logs, Progress Reports, Change Request Forms |
| Testing | Collaborate with QA to ensure test cases match business requirements. Participate in User Acceptance Testing (UAT) to validate the system. | Test Cases, UAT Scripts, Defect Logs, RTM |
| UAT | Conduct user acceptance testing, gather feedback from stakeholders, and ensure the system meets all business expectations before final delivery. | UAT Results, Sign-off Document, Training Materials |

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

**Answer:**

**Conflict Management** is the process of handling disagreements effectively to minimize negative impacts while fostering collaboration. In a business or project setting, conflicts can arise from differences in opinions, priorities, or working styles.

**Thomas–Kilmann Conflict Management Technique**

The Thomas–Kilmann model identifies five conflict resolution styles based on two dimensions: assertiveness (the extent to which a person seeks to satisfy their own concerns) and cooperativeness (the extent to which a person seeks to satisfy others' concerns). Here's an explanation of the five conflict management styles using this technique:

1. **Competing (High Assertiveness, Low Cooperativeness)**:

* Focuses on winning the argument by pushing for one's own goals.
* Used when quick decisions are necessary or in situations where standing firm is vital.
* Example: A customer service manager insists on enforcing a strict policy despite opposition.

1. **Collaborating (High Assertiveness, High Cooperativeness)**:
   * Both parties work together to find a mutually beneficial solution.
   * It’s ideal when the issue is significant and both parties want to satisfy their own concerns.
   * Example: A customer and a development team collaborate on improving a payment system to make it user-friendly while maintaining technical integrity.
2. **Compromising (Moderate Assertiveness, Moderate Cooperativeness)**:

* Both parties give up something to reach a middle ground.
* Suitable when time is limited or when finding an immediate solution is more important than achieving a perfect one.
* Example: Negotiating with a client on a feature delivery timeline by scaling down some aspects to meet deadlines.

1. **Avoiding (Low Assertiveness, Low Cooperativeness)**:

* Avoids the conflict entirely, postponing or ignoring it.
* Useful when the conflict is trivial or when other issues take priority.
* Example: Ignoring a minor disagreement about the design of a payment interface when there are more pressing matters.

1. **Accommodating (Low Assertiveness, High Cooperativeness)**:

* One party gives in to the other’s concerns, often at the expense of their own.
* Applied when maintaining the relationship is more important than the outcome.
* Example: A company accepts a client's demand for a specific payment method even though it's less efficient for the business.

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

**Answer:**

1. **Poor Planning:** Inadequate project planning leads to unclear direction and lack of coordination among team members. It results in missed deadlines, cost overruns, and failure to deliver expected outcomes. Planning should include scope, resources, timelines, and risk management to be effective.
2. **Unclear Objectives and Requirements:** When the project's goals and requirements are not clearly defined or frequently change, it creates confusion among stakeholders and the team. This can result in delivering something that doesn’t meet the needs or expectations, or the inability to finish the project successfully.
3. **Inadequate Risk Management**: Not identifying potential risks and failing to prepare mitigation strategies lead to significant issues during the project lifecycle. Risk management helps foresee problems like budget cuts, resource unavailability, or scope changes that could derail the project.
4. **Poor Communication**: Lack of proper communication among stakeholders, team members, and management can lead to misunderstandings, missed tasks, and incorrect assumptions. Effective communication is crucial for ensuring that everyone is on the same page regarding objectives, responsibilities, and expectations.
5. **Scope Creep**: Scope creep occurs when there are uncontrolled changes or continuous expansion of the project scope without adjusting time or budget. It overwhelms the project team and results in the inability to deliver within the agreed-upon scope and schedule.
6. **Lack of Stakeholder Engagement**: When stakeholders, including customers, sponsors, or key decision-makers, are not actively involved in the project, it can result in misalignment with their needs, missed opportunities for feedback, and lack of support, which are essential for project success.
7. **Resource Constraints**: Inadequate resources, whether financial, human, or material, lead to an inability to complete the project as planned. Insufficient staffing, unavailability of skilled professionals, or budget cuts can severely impact project progress.
8. **Technical Challenges**: Sometimes, projects face unexpected technical difficulties, whether due to software limitations, infrastructure problems, or lack of expertise. These issues can prevent the project from achieving its goals or functioning as expected, leading to failure.

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

**Answer:**

1. **Unclear or Changing Requirements:** Difficulty in defining or maintaining clear, stable project requirements due to evolving client needs or stakeholder input.
2. **Managing Stakeholder Expectations:** Ensuring all stakeholders have a unified understanding of project goals and managing differing expectations throughout the project lifecycle.
3. **Scope Creep and Scope Management:** Managing changes that expand the project's scope beyond the initial agreement without affecting the project timeline or budget.
4. **Time and Resource Constraints:** Dealing with limited timeframes and insufficient resources, which can hinder the ability to meet project deadlines.
5. **Quality Assurance and Testing:** Ensuring that the delivered solution meets quality standards and properly addressing defects or gaps found during the testing phase.
6. **Documentation and Knowledge Management:** Maintaining accurate and up-to-date documentation while ensuring knowledge is effectively shared across the team.
7. **Technology Constraints and Complexity:** Navigating limitations in technology and dealing with the complexities of integrating different systems or adapting to new technologies.

**Q12. Write about Document Naming Standards**

**Answer:**

Document Naming Standards refer to a structured and consistent method for naming files and documents within an organization or project. These standards are essential for ensuring that documents are easily identifiable, searchable, and manageable throughout the document lifecycle.

Here are key aspects of document naming standards:

* **Consistency**: Establishing a uniform format across all documents, such as the use of underscores (\_), dashes (-), or spaces. For example, using "Project\_Report\_2024" ensures uniformity and readability.
* **Version Control**: Including a version number or date stamp in the document name helps track revisions and updates. For instance, "Project\_Plan\_v1.0" or "Requirements\_20231015".
* **Descriptive Titles**: The name should clearly reflect the content or purpose of the document. For example, "Marketing\_Strategy\_Q4\_2023" rather than a vague "Strategy\_2023".
* **Standardized Abbreviations**: Define and consistently use abbreviations for common terms to maintain clarity and conciseness, such as "PRD" for Product Requirement Document or "SRS" for Software Requirements Specification.

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

**Answer:**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **DO’S** | **DON’T’S** |
| 1. | Engage stakeholders regularly. | Assume you know all the requirements. |
| 2. | Document requirements clearly. | Overlook the importance of validation. |
| 3. | Use data to drive decisions. | Rely solely on intuition. |
| 4. | Prioritize tasks effectively. | Ignore deadlines. |
| 5. | Communicate findings succinctly. | Use jargon that confuses stakeholders. |
| 6. | Stay adaptable to changes. | Resist feedback or new ideas. |
| 7. | Collaborate with cross-functional teams. | Work in isolation. |
| 8. | Keep learning and updating skills. | Settle for outdated methods or tools. |
| 9. | Focus on delivering value. | Lose sight of project goals. |
| 10. | Test and validate solutions thoroughly. | Skip the testing phase. |

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

**Answer:**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Packages** | **Sub-system** |
| **Definition** | A logical grouping of related classes or components. | A larger, complex unit composed of multiple packages or components. |
| **Purpose** | Organizes code for better structure and management. | Performs a specific function within a larger system. |
| **Scope** | Focuses on organizing similar classes within a single layer. | Encapsulates multiple packages and may span multiple architectural layers. |
| **Example** | Grouping classes like Card Payment, Wallet Payment. | A "Payment Processing" system integrating all payment methods. |

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

**Answer:**

Camel casing is a naming convention in which multiple words are combined into a single word with the first word in lowercase and each subsequent word starting with an uppercase letter. The name "camel case" comes from the way the capital letters in the middle of the word resemble the humps of a camel.

**Example:**

* **Single Word**: payment
* **Camel Cased**: paymentMethod, cardPayment, walletBalance, cashPayment, netPayment

**Where Camel Casing is Used**

1. Variable Naming:

Camel casing is frequently used for naming variables in programming languages. For instance, when defining a variable for the payment method in a payment processing system, you might use payment Type or customer Payment.

1. Function and Method Names:

When creating functions or methods, camel casing is often used to improve readability. For example, a method to process a card payment could be named processCardPayment, while one for verifying wallet balances might be check Wallet Balance.

1. Class Names:

While camel casing is common for method and variable names, many programming languages (especially in object-oriented programming) use Pascal casing (a variant of camel casing where the first letter is also capitalized) for class names. For example, a class representing payment methods could be named Payment Processor.

1. API Endpoints and URL Parameters:

In web development, camel casing can also be used for naming API endpoints and URL parameters to maintain consistency. For example, an endpoint for handling payments might be /api/processPayment.

1. Database Field Names:

In database design, field names can be defined using camel casing to improve clarity. For instance, a field representing the customer's email might be named customer Email.

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

**Answer:**

A Development Server is an environment where developers work on the application before it goes live to production. It simulates the production environment and allows for safe coding, integration, testing, and debugging. In this case, the development server will host the code and components needed for handling the various payment methods.

As a Business Analyst, the following accesses are typically granted:

1. **Read-Only Access to Requirements Documentation**: The BA may have access to documentation repositories like Confluence, where user stories, business requirements, and functional requirements are stored. This helps the BA verify that development aligns with business needs.
2. **Read-Only Access to Development Builds**: The BA may be granted access to the development server to observe or verify the latest builds of the application to see the progress in real-time and ensure the functionality (e.g., payment options) matches the business requirements.
3. **Test Data Access**: BAs can sometimes access test data or create mock scenarios in collaboration with the QA team to validate that customer payment flows (Card, Wallet, Cash, Net Banking) are working correctly.
4. **Access to Jira or Project Management Tools**: The BA will typically have full access to Jira (or equivalent tools) to manage user stories, track the status of development tasks, and communicate with the development team regarding issues or clarifications needed for payment method implementations.
5. **Collaboration Tools**: The BA will use tools like Slack, Zoom, or Teams to coordinate with developers, testers, and other stakeholders to clarify requirements, report issues, and track progress.
6. **Access to Test Reports**: The BA may review test reports or logs generated from automated tests to ensure that the payment gateways (Card, Wallet, Net Banking) have been integrated correctly and are functioning as expected.

**Q17. What is Data Mapping**

**Answer:**

Data Mapping is the process of establishing relationships between data fields in different data models or databases. It involves defining how data from one source (such as a database, application, or file) corresponds to data in another source. This is crucial for data integration, transformation, and migration processes, ensuring that data remains consistent, accurate, and meaningful across various systems.

**Key Aspects of Data Mapping**

1. Purpose: Data mapping serves as a bridge between different data sources, helping to ensure that data is correctly translated and utilized in applications or systems. It is essential in scenarios like data migration, system integration, and data warehousing.
2. Types of Data Mapping:
   * One-to-One Mapping: Each field in the source corresponds to one field in the target.
   * One-to-Many Mapping: A single field in the source corresponds to multiple fields in the target.
   * Many-to-One Mapping: Multiple fields in the source correspond to a single field in the target.
   * Many-to-Many Mapping: Multiple fields in the source correspond to multiple fields in the target.
3. Data Mapping Techniques:
   * Manual Mapping: Involves human intervention to define the mapping rules.
   * Automated Mapping: Utilizes software tools to automatically generate mapping based on predefined rules or algorithms.
4. Mapping Tools: Various tools and software solutions facilitate data mapping, providing graphical interfaces to create and manage mapping rules. Examples include ETL (Extract, Transform, Load) tools, data integration platforms, and database management systems.

**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 different software applications to communicate with each other. APIs define the methods and data formats that applications can use to request and exchange information. They enable the integration of different systems, allowing them to work together seamlessly.

API Integration in Your Application.

In the case of your application that accepts dates in the format dd-mm-yyyy but needs to integrate with another application from the US that uses the format mm-dd-yyyy, API integration would be essential for ensuring proper data exchange. Here’s how you would approach this:

1. API Setup

* Identify the Source API: Determine the API from the US application that provides the date data. This could involve reviewing their API documentation to understand the endpoints and authentication methods.
* Authentication: If the source API requires authentication (e.g., API keys, OAuth), set up the necessary credentials to access the API securely.

1. Data Request

* Make API Calls: Use your application to send requests to the source API. For example, if you need to retrieve date data, construct a GET request to the appropriate endpoint.

1. Data Conversion

* Format Conversion: Once you receive the date data in mm-dd-yyyy format, convert it to dd-mm-yyyy format before processing or storing it in your application. You can use Python’s datetime module for this.

1. Data Processing

* Use Converted Data: After conversion, use the date in your application as needed, whether for displaying to users, storing in a database, or any other business logic.

1. Error Handling

* Implement Error Handling: Make sure to include error handling for scenarios such as API downtime, invalid data formats, or connection issues. This ensures your application remains robust and user-friendly.

API integration is a powerful way to connect different applications and systems. By correctly handling data formats during integration—like converting dates from mm-dd-yyyy to dd-mm-yyyy—you can ensure seamless data flow and enhance the functionality of your application.