Domain-Driven Design Document

Digital Payment Management System

I. Introduction

A. Purpose/Objective

This document defines the domain model, architecture, and design patterns for the Digital Payment Management System.

B.Domain Scope

The application serves e-commerce platforms, financial institutions, and merchants to manage orders, payments, order tracking, and customer interactions effectively.

C.Domain Description

The system facilitates customer management, order placement, payment processing, and transaction history tracking.

D.Context

This analysis is limited to the backend service layers responsible for managing the core domain logic of the Digital Payment Management System, excluding UI or external 3rd-party integrations (e.g., payment gateways).

II. Strategic Design

A. Bounded Contexts

a. Define core bounded contexts within our shopping cart system domain

1.Customer Context(Handle customer profiles, personal data, and contact info.).
2.Order Management Context(Handle order placement, items, and tracking status).
3.Payment Context(Handle all payment processes and statuses).
4.Product Catalog Context(Store and retrieve product details and categories).

b.Every Contexts should be explained with these as follows

1. In Customer Context the Ubiquitous Language: Customer, Full Name, Email, Phone.
2. In Order Management Context the Ubiquitous Language: Order, OrderStatus, OrderHistory, Items, Quantity.
3. In Payment Context the Ubiquitous Language: Payment, PaymentMethod, PaymentStatus.
4. In Product Catalog Context the Ubiquitous Language: Product, Category, Price, Inventory.

B. Context Map

a. In Source Context of Order Management and the Target Context is Customer Context, In Source Context of Order Management and the Target Context is Product Catalog, In Source Context of Payment Context and the Target Context is Order Management are mapping.

b. In Relationship Type of Customer-Supplier Customer-Supplier and in Conformist relation define Order fetches static product details (no feedback loop) and in Customer-Supplier define Payment depends on Order status before payment initiation.

c. Provide proper brief document for interactions using the relationships.

Example: The Payment Context acts as a Customer to the Order Management Context, depending on order status updates before processing a transaction.

C. Sub-Domains

a. In Payment Processing the type is core ,In Order Management the type is Core and in Customer Management the type is Supporting.

III. Tactical Design

A. Order Context

Entities:

1. Order
2. OrderItem
3. OrderHistory

Value Objects:

1. OrderStatus
2. OrderNote

Aggregates:

1. Order

Domain Services:

1. OrderService

Domain Events:

1. OrderCreated
2. OrderStatusUpdated

Repositories:

1. OrderRepository

Factories:

1. OrderFactory

Application Services:

1. OrderManagementService

B. Payment Context

Entities:

1. Payment

Value Objects:

1. PaymentStatus
2. PaymentMethod
3. Amount

Aggregates:

1. Payment

Domain Services:

1. PaymentProcessorService

Domain Events:

1. PaymentCompleted
2. PaymentFailed

Repositories:

1. PaymentRepository

Factories:

1. PaymentFactory

Application Services:

1. PaymentManagementService

C. Customer Context

Entities:

1. Customer

Value Objects:

1. ContactInfo
2. FullName

Aggregates:

1. Customer

Domain Services:

1. CustomerLookupService

Domain Events:

1. CustomerCreated
2. CustomerUpdated

Repositories:

1. CustomerRepository

Factories:

1. CustomerFactory

Application Services:

1. CustomerManagementService

D. Product Context

Entities:

1. Product
2. Category

Value Objects:

1. Price
2. Description

Aggregates:

1. Product

Domain Services:

1. ProductSearchService

Domain Events:

1. ProductAdded
2. ProductUpdated

Repositories:

1. ProductRepository

Factories:

1. ProductFactory

Application Services:

1. ProductCatalogService

IV. Implementation Considerations

a. Technology Considerations:

1. Spring Boot for backend domain modeling.
2. Kafka for domain events messaging.
3. Hibernate/JPA for ORM-based persistence.

b. Architectural Pattern:

1. CQRS for separating command vs query models in order/payment.
2. Event Sourcing for tracking historical state of orders/payments.
3. Resilience: CircuitBreaker using Resilience4j, Retry for payment retries.
4. SAGA Pattern for distributed transaction management.

c. Testing

1. Unit Testing with JUnit5.
2. Integration Testing using TestContainers or Mockito.
3. Security Testing with Spring Security test utils.

d. Security

1. OAuth2.0 / JWT for API authentication/authorization.
2. Spring Security method-level and endpoint-level protection.

V. Glossary

Order:	Transaction initiated by customer including product selections.
Payment:	Financial transaction linked to an order.
OrderStatus:	Lifecycle stage of order (e.g., PENDING, SHIPPED, DELIVERED).
Customer:	Person making purchases in the system.
Aggregate Root:	Entry point to access domain object cluster.
Domain Event:	An important event that occurs in the domain (e.g., OrderCreated).

Example Illustrative Snippets:

Product Context – Entity

Category Context – Entity:

Entity: Category

Description: Represents a classification or group to which products belong. It helps organize products for easier navigation and management.

Attributes:

1. id
2. category_name
3. description

Customer Context – Entity:

Entity: Customer

Description: Represents a user of the digital payment system who places orders and completes transactions.

Attributes:

1. id
2. first_name
3. last_name
4. email
5. phone

Order Context – Entity:

Entity: Order

Description: Represents a purchase placed by a customer. Maintains total amount and order status.

Attributes:

1. id
2. customer_id
3. order_date
4. status
5. total_amount

Payments Context – Aggregate:

Aggregate: Payment

Description: Represents payment information related to an order. Manages payment method, status, and amount.

Entities:

1. payments

Entity: Payment
Attributes:

1. id
2. order_id
3. payment_date
4. payment_method
5. amount
6. payment_status

Order Context – OrderItems Entity:

Entity: OrderItem
Description: Represents individual product line-items that are part of a larger order.
Attributes:

1. id
2. order_id
3. product_id
4. quantity
5. price_at_order

Order Context – OrderHistory Entity:

Entity: OrderHistory
Description: Tracks the lifecycle and status updates of an order for audit and visibility purposes.
Attributes:

1. id
2. order_id
3. status
4. updated_at
5. note

Context Map: Order & Payment Contexts:

Relationships:

Order → Payment: Customer-Supplier (Conformist)
Order → Product: Customer-Supplier (Conformist)

Description:

The Order Context depends on Product Context to retrieve product information.
The Payment Context depends on Order Context for payment triggers and validation.

Integration:

API calls from Order Context to:

1. Fetch product info by product_id.
2. Initiate payment processing.
3. Notify on order status change.