JAVA OOP

GROUP

1. BRIAN KIPKIRUI – SCT212-0057/2023.
2. AUSTIN KANJA – SCT212-0493/2023.
3. EUGENE KIPKOECH – SCT212-0477/2023.
4. EVANS KIPROTICH – SCT212-0178/2022.

Object-Oriented Analysis and Design (OOA & OOD) for Payment and Financial Management Module

1. Introduction

This document provides the Object-Oriented Analysis (OOA) and Design (OOD) for the Payment and Financial Management Module of the My Dairy Application. It outlines key components including actors, use cases, class responsibilities, design patterns, and system interactions.

2. Problem Domain

The module automates payments between farmers, collectors, groups, and processors, manages deductions for agro-vet loans, integrates with MPesa/bank APIs, and generates financial reports.

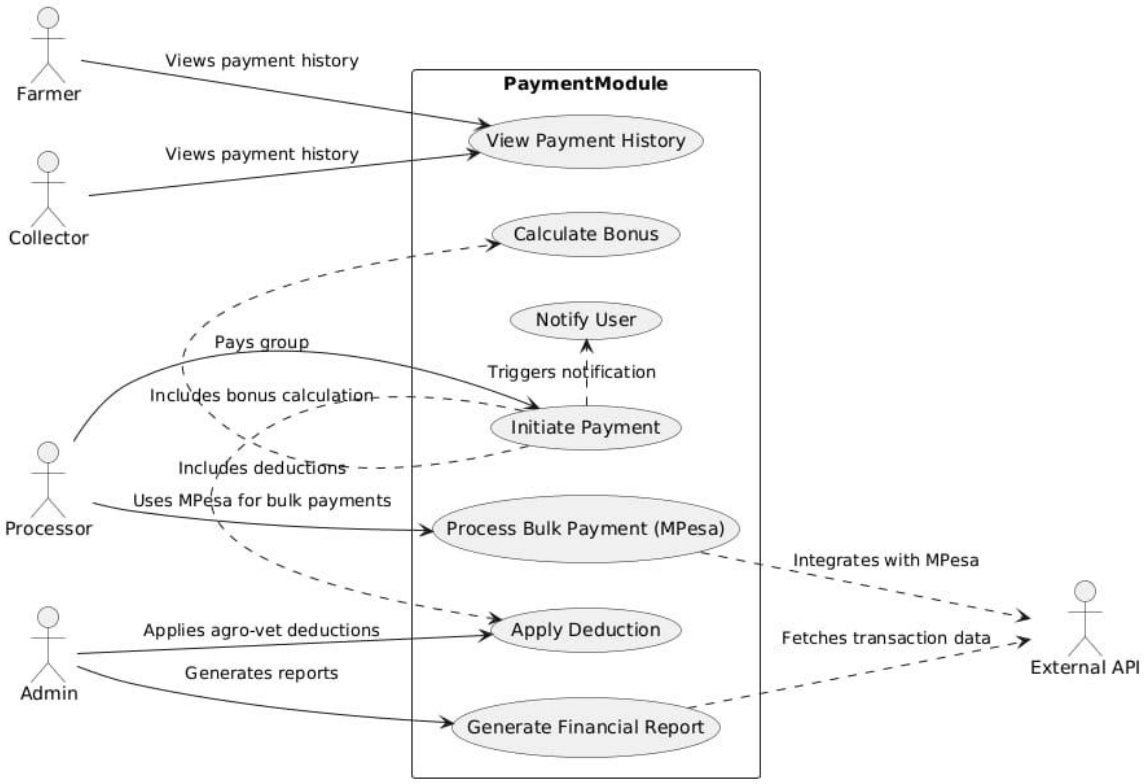
3. Key Objects and Responsibilities

Object	Responsibilities
User	Represents farmers, collectors, processors, and groups with roles and payment history.
Transaction	Manages payments, deductions, and bonuses. Tracks status (success/failed).
MilkCollection	Provides milk quantity and quality data to calculate payments.
AgroVetPurchase	Tracks agro-vet loan purchases for deductions.
PaymentProcessor	Calculates total payments (base amount + quality bonus - deductions).
PaymentGateway	Interface for MPesa/bank integrations.
ReportGenerator	Generates income, expense, and loan reports.

NotificationService	Sends SMS/email alerts for payments and deductions.

4. Use Case Diagram

Below is the use case diagram representing key system interactions:



5. Use Case Descriptions

Use Case 1: View Payment History

Actor: Collector, Farmer

Description: Allows users to view their transaction records

Preconditions: User is authenticated and has payment history

Flow:

User selects "Payment History" from dashboard

System retrieves transactions from database

System displays list with:

Date/time stamps

Transaction amounts

Payment/deduction types

Status indicators

Postconditions: User can export or print records

Use Case 2: Calculate Bonus

Trigger: Milk quality verification completes

Business Rules:

Bonus = (QualityScore - 80) × BaseRate × Quantity

Only applies when QualityScore > 80

Data Used:

MilkCollection.quantity

QualityReport.score

PaymentRateTable.bonusMultiplier

Use Case 3: Initiate Payment

Actor: Processor

Steps:

System calculates base payment (Quantity × Rate)

Applies quality bonus if eligible

Deducts agro-vet loans (max 30% of total)

Processor confirms amount

System initiates MPesa transaction

Records transaction with "Pending" status

Exception:

If MPesa fails, retry 3 times before marking "Failed"

Use Case 4: Process Bulk Payment

Actor: Collector

Special Requirements:

MPesa bulk API integration

Minimum 5 payees per batch

Flow:

Collector selects group members

System aggregates individual amounts

Collector confirms total and submits

System sends bulk request to MPesa

Updates all transaction statuses simultaneously

Use Case 5: Apply Deduction

Actor: Processor

Trigger: Agro-vet loan purchase

Rules:

Deductions spread over next 3 payments

SMS notification sent before deduction

Farmer dashboard shows pending deductions

Use Case 6: Generate Financial Report

Actor: Admin

Outputs:

PDF report with:

Income/expense summary

Bonus/deduction breakdown

Payment channel statistics

Excel export for accounting systems

Use Case 7: Notify User

Triggers:

Payment completion

Deduction applied

Report available

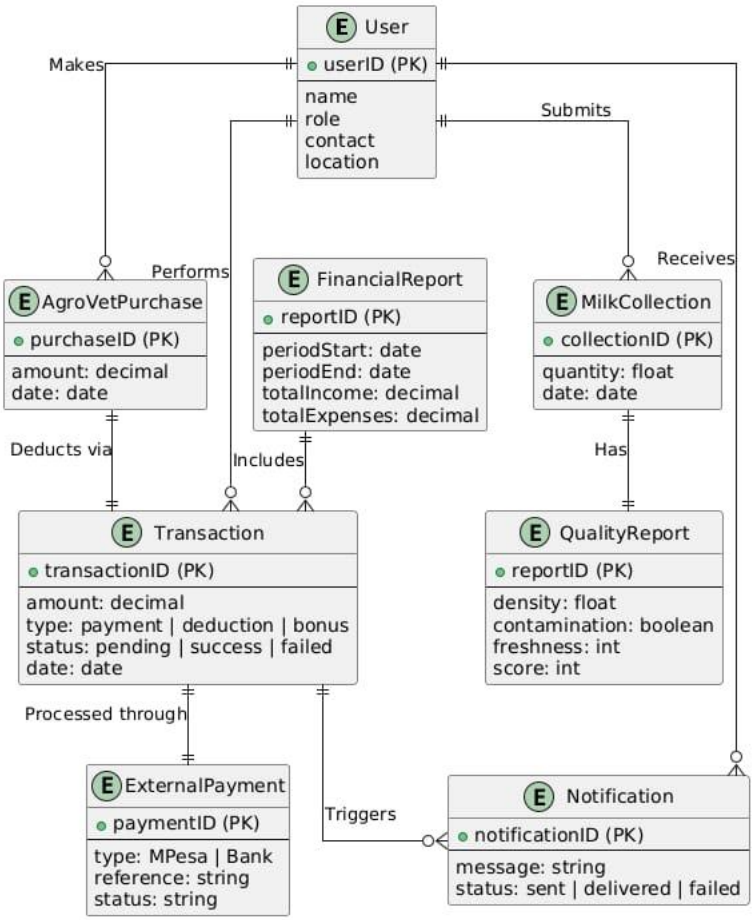
Channels:

SMS for transactions

Email for reports

In-app notifications for deductions

6. Object-Oriented Design (OOD)



Class Descriptions

1.User

Attributes:

- userID (Primary Key): Unique identifier for each user

- name: Full name of the user

- role: User's role in the system (farmer, collector, processor, admin)

- contact: Phone number/email for communication

- location: Geographical coordinates or address

Methods:

- submitMilkCollection(quantity, date): Records milk submission

- viewTransactions(dateRange): Retrieves payment history

- updateProfile(contactInfo): Modifies user details

2.Milk Collection

Attributes:

- collectionID (PK): Unique collection record identifier

- quantity: Milk volume in liters (float)

- date: Collection timestamp

Methods:

- getQualityMetrics(): Retrieves associated quality report

- calculatePayment(): Computes payment amount based on quantity and quality

3.QualityReport Class

Attributes:

- reportID (PK): Unique quality report identifier

- density: Milk density measurement

- contamination: Boolean flag for contamination

- freshness: Score (1-10 scale)

- score: Composite quality score (0-100)

Methods:

- generateReport(): Creates printable quality certificate

- validateQuality(): Checks if meets minimum standards

4.Transaction Class

Attributes:

- transactionID (PK): Unique transaction identifier

- amount: Monetary value (decimal)

- type: ENUM (payment|deduction|bonus)

- status: ENUM (pending|success|failed)

- date: Transaction timestamp

Methods:

- processPayment(): Executes financial transaction

- updateStatus(newStatus): Modifies transaction state

- generateReceipt(): Creates payment confirmation

5.FinancialReport Class

Attributes:

- reportID (PK): Unique report identifier

- periodStart: Report start date

- periodEnd: Report end date

- totalIncome: Sum of all payments

- totalExpenses: Sum of all costs

Methods:

- generatePDF(): Creates printable report

- calculateProfit(): Computes net income

- exportToExcel(): Generates spreadsheet format

Notification Class

Attributes:

- notificationID (PK): Unique message identifier

- message: Content text

- status: ENUM (sent|delivered|failed)

Methods:

- sendAlert(): Dispatches notification

- logDelivery(): Records delivery confirmation

- retryFailed(): Reattempts failed notifications

7. ExternalPayment Class

Attributes:

- paymentID (PK): Unique payment reference

- type: ENUM (MPesa|Bank)

- reference: Gateway transaction ID

- status: Current payment state

Methods:

- processExternal(): Initiates third-party payment

- verifyTransaction(): Confirms payment completion

- reconcilePayment(): Matches with internal records

Key Relationships

1. User to MilkCollection: One-to-many (A user can submit multiple collections)

2. MilkCollection to QualityReport: One-to-one (Each collection has one quality report)

3. MilkCollection to Transaction: One-to-many (Collections may result in multiple transactions - base payment, bonus, etc.)

4. Transaction to ExternalPayment: One-to-one (Each transaction uses one payment gateway)

5. FinancialReport to Transaction: One-to-many (Reports aggregate multiple transactions)

7. Design Principles Applied

- Single Responsibility: Each class handles a specific task.

- Strategy Pattern: PaymentGateway interface supports multiple implementations (MPesa, Bank).

- Observer Pattern: NotificationService listens for updates.

- Dependency Injection: PaymentProcessor relies on MilkCollection and AgroVetPurchase APIs.

8. API Integrations

MilkCollectionService

GET /collections/{id} to retrieve milk data.

AgroVetService

GET /agrovet/purchases/{farmerId} to retrieve deductions.

MpesaGateway

POST /mpesa/payment to process payment.