**Domain Driven Design Document**

**[Digital Payment System]**

**I. Introduction**

**A. Purpose/Objective**

The objective of this DDD documentation is to define the architecture and design of a Digital Payment System that enables users to transfer money between wallets and bank accounts, top up wallets through bank transfers or cards, process secure merchant payments, and detect fraudulent transactions based on user behavior. The system needs to be secure, scalable, and maintainable, ensuring seamless transactions across different services.

**B. Domain Scope**

The Digital Payment System targets both individual consumers and merchants. For consumers, it allows:

* Transferring funds between wallets and bank accounts.
* Top-up of wallets via bank transfers or card payments.
* Fraud detection and prevention based on behavior.

**For merchants, it allows:**

* Receiving secure payments from users.
* Integration with their e-commerce or point-of-sale systems.

The system will integrate with external financial institutions, payment gateways, fraud detection systems, and merchant services.

**C. Domain Description**

This system must handle financial transactions reliably and securely, supporting various types of transactions such as:

**Money Transfer:** Between wallets and bank accounts.

**Wallet Top-Up:** Via bank transfer or card payments.

**Merchant Payments:** Secure payments to merchants.

**Fraud Detection:** Monitoring and detecting unusual user behaviors to prevent fraud.

**D. Context**

The analysis of the Digital Payment System will focus on the interactions between different bounded contexts:

**Wallet Management:** Handling digital wallets, balances, and transaction histories.

**Bank Account Management:** Managing bank account details and transfers.

**Merchant Payment Processing:** Facilitating payments between consumers and merchants.

**Fraud Detection:** Monitoring transactions and user behaviors to flag potentially fraudulent activity.

**Security & Compliance:** Ensuring adherence to regulations and implementing security measures for financial transactions.

**II. Strategic Design**

**A. Bounded Contexts**

The following bounded contexts are identified within the Digital Payment System:

**1.Wallet Context:**

**Responsibility:** Manage user wallets, balances, and wallet-based transactions (e.g., transfer funds, top-up).

**Ubiquitous Language:** Wallet, Wallet Balance, Wallet Transaction, Top-Up.

**Key Domain Concepts:** Wallet ID, Transaction History, Balance, Payment Method.

**2.Bank Account Context:**

**Responsibility:** Manage user bank account details, facilitate transfers between bank accounts and wallets.

**Ubiquitous Language:** Bank Account, Bank Transfer, External Account.

**Key Domain Concepts:** Bank Account ID, IBAN, Routing Number, Bank Transfer.

**3.Merchant Payment Context:**

**Responsibility:** Facilitate secure payments to merchants and manage merchant accounts.

**Ubiquitous Language:** Merchant Account, Payment, Payment Gateway, Invoice.

**Key Domain Concepts:** Merchant ID, Payment Method, Payment Status, Invoice.

**4.Fraud Detection Context:**

**Responsibility:** Monitor user behavior and detect fraudulent activity based on transaction patterns.

**Ubiquitous Language:** Fraud Detection, Anomaly, Suspicious Transaction, Fraudulent Activity.

**Key Domain Concepts:** Transaction ID, Risk Score, Behavior Pattern, Alert.

**5.Security & Compliance Context:**

**Responsibility:** Handle encryption, secure data storage, compliance with regulations like PCI DSS, and ensure secure transaction processes.

**Ubiquitous Language:** Encryption, Tokenization, Compliance, Audit Log.

**Key Domain Concepts:** Encryption Key, Tokenized Data, Compliance Report, Audit Trail.

**B. Context Map**

The context map illustrates how the above contexts interact:

**Wallet Context ↔ Bank Account Context:** Customer-Supplier Relationship — The Wallet context consumes bank account information for initiating transfers.

**Wallet Context ↔ Merchant Payment Context:** Shared Kernel — Both contexts share transaction data to process merchant payments.

**Fraud Detection Context ↔ Wallet Context:** Customer-Supplier Relationship — Fraud detection monitors transactions and behaviors in the Wallet context.

**Security & Compliance Context ↔ All Contexts:** Shared Kernel — Security and compliance practices must be uniformly applied across all contexts to ensure data protection and regulatory adherence.

**C. Sub-Domains**

The subdomains are categorized as follows:

**1.Core Subdomain:** Payment Processing — Encompasses transaction handling, wallet management, and merchant payment processing.

**2.Supporting Subdomain:** Fraud Detection — Monitors user activities and analyzes transaction patterns to detect fraud.

**3.Generic Subdomain:** Security and Compliance — Generic support services like encryption, data security, and compliance with regulations.

**III. Tactical Design**

**A. Wallet Context**

**a. Entities**

**Wallet**: Represents a digital wallet where users store funds.

**Attributes**: Wallet ID, User ID, Balance, Payment Methods.

**Wallet Transaction**: Represents a single transaction involving the wallet.

**Attributes**: Transaction ID, Amount, Source Wallet, Destination Wallet, Date, Status.

**b. Value Objects**

**Money:** Represents an amount of currency in a specified unit (e.g., USD, EUR).

**Attributes:** Value, Currency.

**c. Aggregates / Aggregate Root**

**Wallet (Aggregate Root):** The Wallet entity serves as the root, aggregating all transactions and managing the balance.

**Aggregate Root:** Wallet\_ID.

**Associated Entities:** Wallet Transaction, Payment Method.

**d. Domain Services**

**WalletService:** Handles business logic for wallet-related actions such as transfer funds, check balance, and top-up wallet.

**PaymentService:** Manages payment operations like processing wallet-to-wallet or wallet-to-bank transfers.

**e. Domain Events**

**FundsTransferred:** Event triggered when money is transferred between wallets or from wallet to bank account.

**WalletTopUp:** Event triggered when a wallet is topped up successfully.

**f. Repositories**

**WalletRepository:** Interface for storing and retrieving wallet data.

**WalletTransactionRepository:** Interface for managing wallet transactions.

**g. Factories**

**WalletFactory:** Factory for creating new wallet instances.

**WalletTransactionFactory:** Factory for creating wallet transactions.

**h. Application Services**

**TransferFundsService:** Handles the transfer of funds between wallets and to external accounts.

**TopUpWalletService:** Manages wallet top-ups from cards or bank transfers.

**B. Bank Account Context**

**a. Entities**

**BankAccount:** Represents a user’s bank account linked to the payment system.

**Attributes:** Account ID, Bank Name, Routing Number, Account Number, User ID.

**b. Value Objects**

**IBAN:** International Bank Account Number, used to uniquely identify a bank account internationally.

**c. Aggregates / Aggregate Root**

**BankAccount (Aggregate Root):** Manages bank account details and coordinates bank transfers.

**Aggregate Root:** BankAccount\_ID.

**Associated Entities:** Bank Account Information.

**d. Domain Services**

**BankTransferService:** Handles the logic for initiating and processing bank transfers.

**e. Domain Events**

**BankTransferInitiated:** Event triggered when a bank transfer is initiated.

**BankTransferCompleted:** Event triggered when a bank transfer is successfully completed.

**f. Repositories**

**BankAccountRepository:** Repository for managing bank account data.

**g. Application Services**

**TransferToBankService:** Manages the process of transferring funds from a wallet to a bank account.

**C. Fraud Detection Context**

**a. Entities**

**SuspiciousActivity:** Represents an event or pattern identified as suspicious.

**Attributes:** Transaction ID, Suspicion Reason, Risk Score.

**b. Domain Services**

**FraudDetectionService:** Analyzes transaction patterns to detect and prevent fraud based on machine learning algorithms or rule-based systems.

**c. Domain Events**

**FraudulentTransactionDetected:** Event triggered when a suspicious or fraudulent transaction is detected.

**IV. Implementation Considerations**

**a. Technology Considerations**

**Backend Framework:** Spring Boot (Java) for building RESTful APIs and managing business logic.

**Frontend Framework:** React/Angular for web-based user interfaces.

**Messaging System:** Kafka or RabbitMQ for event-driven communication between services.

**Database:** SQL (PostgreSQL/MySQL) for transactional data and NoSQL (MongoDB) for event sourcing and logs.

**Fraud Detection:** AI/ML algorithms (TensorFlow, Scikit-learn) for detecting fraudulent patterns.

**b. Architectural Pattern**

**Event-Driven Architecture:** Events (e.g., transaction completed, fraud detected) trigger responses and workflows across the system.

**CQRS:** Separate commands (write) and queries (read) for scalability and performance.

**Microservices:** Decompose into independent services (e.g., Wallet, Bank Account, Merchant Payment) that communicate via APIs.

**c. Security & Compliance**

**Tokenization:** Sensitive payment data (e.g., card details) should be tokenized for storage.

**Encryption:** End-to-end encryption for transactions and data at rest.

**PCI DSS Compliance:** Adhere to payment card security standards for handling card details.

**d. Testing**

**Unit Testing:** Junit5 for core business logic testing.

**Integration Testing:** Mock external dependencies (e.g., payment gateways) using Mockito.

**Domain Testing:** Test domain events and behaviors using Cucumber.

**V. Glossary**

**Transaction:** A single operation involving the movement of funds between accounts.

**Wallet:** A digital representation of a user’s funds.

**Top-Up:** The process of adding funds to a wallet.

**Fraud Detection:** The process of identifying unusual or suspicious activity in transactions.

**Tokenization:** The process of converting sensitive data into a secure, non-sensitive token.

**Compliance:** Adhering to financial regulations like PCI DSS.