System Architecture & Design Logic

1. Three-Tier Architecture Implementation

Presentation Layer → Service Layer → Repository Layer

- **Service Layer Logic:**
  Business Logic Validation: All operations validate user permissions and data integrity. Asynchronous Processing: Uses CompletableFuture<T> for non-blocking operations. Role-Based Access Control: Different service methods enforce specific user roles Repository

- **Layer Logic:**
  Database Abstraction: JDBC implementation with connection pooling
  Query Constants: Centralized SQL query management for maintainability
  Result Set Mapping: Type-safe conversion from database records to domain objects

2. Security & Authentication Logic

- **Password Security:**
  Used Bcrypt For Password Security

- **Role-Based Authorization:**
  ADMIN: Can create payments and update user roles
  FINANCE_MANAGER: Can update payment status and access audit trails
  VIEWER: Read-only access to payments

- **Validation Chain:**
  User existence validation
  Role permission validation
  Password verification
  Business rule validation

3. Payment Lifecycle Management Logic

- **Payment Status Flow:**
  PENDING → PROCESSING → COMPLETED

- **Create Payment Logic:**
  Validate user is ADMIN
  Validate payment data (amount > 0, valid currency format)
  Set initial status to PENDING
  Create payment record with timestamp

Auto-generate audit trail entry

- **Update Payment Logic:**
  Validate user is FINANCE_MANAGER
  Update payment status
  Record updater username and timestamp
  Maintain audit trail for compliance

4. Audit Trail & Compliance Logic

- **Immutable Audit Records:**
  Every payment operation creates an audit trail entry
  Captures: payment ID, user, amount, status changes, timestamps
  Provides complete transaction history for compliance

- **Query Logic:**
  Audit trails searchable by payment ID or date range
  Only FINANCE_MANAGER role can access audit data
  Date filtering using epoch millisecond conversion

5. Reporting & Analytics Logic

- **Monthly/Quarterly Report Generation and Financial Calculation Logic:**
  Incoming Payments: Sum of all INCOMING type payments
  Outgoing Payments: Sum of all OUTGOING type payments
  Net Balance: Incoming - Outgoing
  Balance Type: CREDIT (net positive) or DEBIT (net negative)

6. Database Connection & Performance Logic

- **Used HikariCP Connection Pooling and Query Optimization:**
  Prepared statements prevent SQL injection
  Static query constants for reusability
  Date range queries use epoch milliseconds for performance
  Enum mapping for type safety

7. Error Handling & Resilience Logic

- **Layered Error Handling:**
  Service Layer: Business logic exceptions with meaningful messages
  Repository Layer: Database operation exceptions

Validation Layer: Input validation and authorization failures.Data Consistency & Integrity Logic

- **Timestamp Management:**
  All records use epoch milliseconds for consistent date handling
  Created/Updated timestamps for audit purposes
  Date conversion utilities for reporting

- **Enum Type Safety:**
  PaymentStatus, PaymentType, PaymentCategory, UserRole enums
  Database enum casting: ?::payment_status
  Prevents invalid state transitions

GitHub Repo Link : https://github.com/Pratham27-12/Payment-management-system