

Brief Walkthrough of the Logic

1. System Purpose and Functionality

The Payment Management System facilitates secure management of financial transactions within an organization. It supports user authentication, role-based access control, payment processing, status updates, and financial reporting with audit trail functionality.

2. Overall Architecture

Your system follows a Layered Architecture Pattern divided into:

- Presentation Layer (Controllers): Handles user input, menus, and UI navigation.
- Service Layer (Business Logic): Enforces business rules, manages transactions, validation, and audit logging.
- Data Access Layer (DAO): Abstracts direct database interactions including CRUD operations.
- Utility Layer: Handles database connections and configuration management.

3. User Authentication and Roles

- Users register or login with credentials.
- Roles assigned: ADMIN, FINANCE_MANAGER, and VIEWER.
- Role-based access control restricts features like adding/updating payments (only ADMIN and FINANCE_MANAGER allowed) and viewing reports (all roles).

4. Payment Management Workflow

- Add Payment:
 - Login user inputs payment details: amount, type (INCOMING/OUTGOING), category (SALARY, VENDOR_PAYMENT, CLIENT_INVOICE), and optional remarks.
 - System validates input (e.g., positive amount).
 - New payment is created with status PENDING and timestamps.
 - Payment is saved via DAO to the PostgreSQL database.
 - An audit log is recorded capturing who added payment and when.

- Update Payment Status:
 - Authorized users can update payment status through valid transitions (e.g., PENDING → PROCESSING → COMPLETED).
 - Payment status update is persisted in the database.
 - Audit trail logs the status changes.
- Retrieve and List Payments:
 - Payments can be retrieved by ID or user.
 - Listings display payment details including amount, status, type, and category.

5. Reporting

- The system supports generating financial reports by categories based on completed payments.
- Reports include:
 - Monthly Report – aggregates sums by category for a specified month and year.
 - Quarterly Report – aggregates sums by category for a quarter and year.
 - Custom Report – details payments and associated audit log entries within a custom date range.
- Reports can be displayed in the console and exported as CSV files for further analysis.

6. Database Interaction

- Uses PostgreSQL with tables for users, payments, and audit trails.
- DAO classes use JDBC with PreparedStatements to safely execute SQL commands, preventing SQL injection.
- Entities like `User`, `Payment`, and `AuditTrail` map closely to database tables.
- Database connection details are externalized in a configuration (`db.properties`).

7. Design Patterns Employed

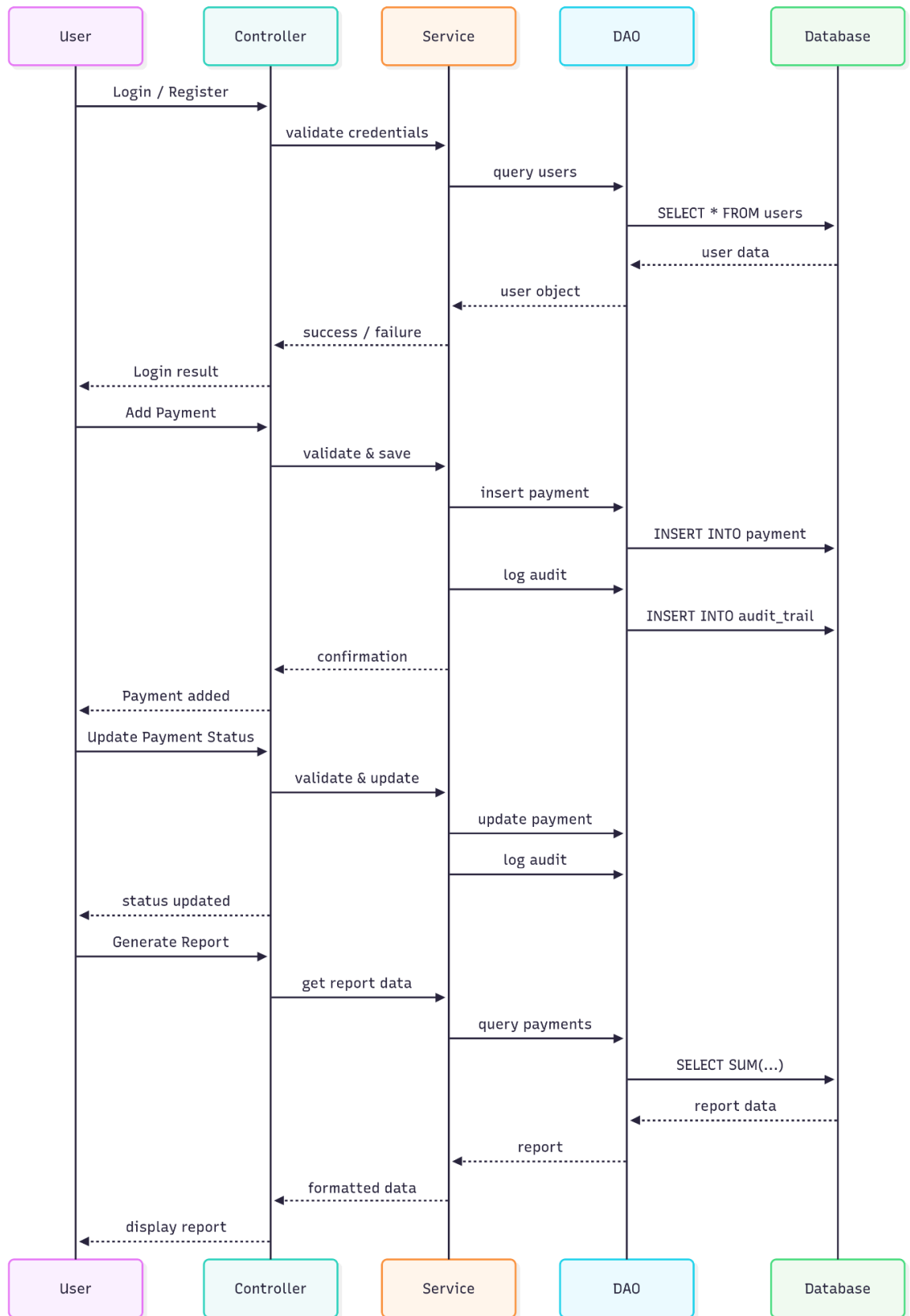
- DAO Pattern: Separates database logic through interfaces like `PaymentDao`, `UserDao`, and `AuditTrailDao`.
- Service Layer: Encapsulates business rules, validating inputs and coordinating DAO operations and audit logging.
- Singleton / Factory: Implicit in service instantiation to promote modularity and easy maintenance.

8. Security and Validation

- User credentials are currently stored in plain text
- Input validations enforce business rules like positive amount and valid status transitions.
- Role-based authorization checks protect sensitive operations.
- SQL queries use PreparedStatements for security.
- Audit trails maintain logs of user actions on payments.

9. Exception Handling

- Uses custom exceptions (`BusinessException`, `NotFoundException`) to handle domain-specific errors gracefully.
- Errors are caught and user-friendly messages displayed in the console UI.



10. How It Runs

- Setup Java 17+, Maven, and PostgreSQL 14+.
- Clone the project from
https://github.com/vikhileshs_Zeta/Payment-Management-System.git
- Configure `db.properties` with correct database credentials.
- Build the project via Maven.
- Run the `Main` class, which provides a CLI for:
- Use menus to navigate the system according to user roles.