Payment Gateway Integration Project Report

1. Project Overview

The Payment Gateway Integration project is designed to provide a seamless and secure way to handle payment transactions through providers like PayPal. The system allows users to initiate transactions, monitor their statuses, and handle error scenarios effectively.

2. Technology Stack

- Java 17
- Spring Boot Framework
- REST APIs
- MySQL Database
- JUnit for Testing
- Maven for Dependency Management

3. System Architecture

The system follows a layered architecture:
- Controller Layer: Handles API requests (PaymentController).
- Service Layer: Processes requests, integrates with external providers (HttpServiceEngine).
- Data Layer: Persists and manages transactions in MySQL.
- Exception Handling: Managed globally (GlobalExceptionHandler).

4. Module Descriptions

a) PaymentController: Exposes REST endpoints for initiating and managing transactions.
b) HttpServiceEngine & HttpRequest: Handles communication with external providers like PayPal.
c) PaypalProviderException: Custom exception for PayPal errors.
d) GlobalExceptionHandler: Centralized error handling for the entire system.
e) Constants & Enums: Provides constant values and error codes for consistency.
f) Domain Objects (Amount, ExperienceContext, etc.): Encapsulate payment details.

5. Database Design

The database consists of multiple tables with relationships:
- Payment_Method
- Payment_Type
- Provider
- Transaction_Status
- Transaction
- Transaction_Log

Relationships:
- Transaction references Payment_Method, Payment_Type, Provider, and Transaction_Status.
- Transaction_Log references Transaction.

6. Workflow

1. User initiates a payment request.
2. PaymentController receives the request and forwards it to the Service layer.
3. HttpServiceEngine communicates with the external provider (PayPal).
4. Provider response is validated and stored in the Transaction table.
5. Transaction status is updated accordingly (CREATED, INITIATED, SUCCESS, FAILED).
6. Transaction_Log is updated for tracking status changes.
7. GlobalExceptionHandler ensures all errors are consistently logged and returned.

7. Error Handling & Logging

The system uses GlobalExceptionHandler to catch and manage exceptions.
PaypalProviderException handles provider-specific errors. Meaningful error codes are defined in ErrorCodeEnum for consistency across the system.

8. Testing

JUnit tests (EurekaServiceRegistryApplicationTests) validate the Spring Boot application context.
Additional unit tests can be created for controllers, services, and repositories to ensure reliability.

9. Conclusion & Future Enhancements

The project successfully integrates a payment gateway with a modular architecture. Future improvements may include:
- Adding support for multiple providers (Stripe, Razorpay, etc.).
- Implementing retry mechanisms for failed payments.
- Enhancing security with token-based authentication.
- Adding monitoring and reporting dashboards.