High-Level Design Document

System: Payment Gateway Platform
Version: 1.0
Prepared by: Architecture & Engineering Team

---

1. Introduction

This document describes the **High-Level Design (HLD)** of the **Payment Gateway Platform**, which enables secure, scalable, and fast online payment services for e-commerce platforms, subscription services, and mobile applications. The platform is designed to support millions of daily transactions with a focus on fault tolerance, compliance, and low latency.

The primary objectives of this document are:

- To describe the system architecture at a high level
- To define how different modules interact with each other
- To identify technologies and tools used
- To specify key workflows, APIs, and security protocols

The document is intended for architects, developers, testers, and stakeholders.

---

2. Business Context

The Payment Gateway Platform solves the problem of **unified payment processing** for global merchants. Businesses often need to accept payments from multiple channels: web, mobile, in-store kiosks, and partner systems.

The system abstracts away:

- Complex integration with multiple banks
- Security and compliance requirements
- Managing transaction life cycles
- Fraud prevention and user notifications

---

3. Goals and Objectives

Goals:

- Provide a **single, unified payment API** for merchants.
- Enable merchants to **track transactions in real-time**.

- Support **multi-region deployments** with active-active clustering.
- Ensure **high availability and low latency**.

Objectives:

1. Support 5000+ TPS (transactions per second).
2. Support multi-currency payments (USD, EUR, INR, GBP).
3. Reduce payment failures due to technical issues.

---

4. Architecture Overview

Microservices Architecture

The platform is implemented using **microservices**, ensuring loose coupling and scalability.

Key Modules:

1. API Gateway
2. Authentication Service
3. Payment Processor
4. Bank Connector
5. Fraud Detection Service
6. Notification Service
7. Database Layer
8. Cache Layer
9. Admin Dashboard
10. Reporting and Analytics Module

---

5. Detailed Module Descriptions

5.1 API Gateway

- Serves as the **entry point** for all client requests.
- Performs **rate limiting** and basic request validation.
- Deployed using **AWS API Gateway** with CloudFront.

5.2 Authentication & Authorization Service

- Manages **OAuth 2.0** and **JWT-based** authentication.
- Integrates with third-party identity providers.

5.3 Payment Processor Service

- Central service that:

  - Validates transactions.
  - Routes requests to **Bank Connectors**.
  - Updates payment status.

5.4 Bank Connector

- Provides **adapters for multiple banks** and payment networks.
- Handles protocol translation between our system and external APIs.

5.5 Fraud Detection Service

- Uses **machine learning models** for real-time risk scoring.
- Runs in **parallel** to the payment flow.

Tech:

- Models trained using Python (TensorFlow, Scikit-Learn).
- Kafka is used for real-time streaming.

---

6. Extended Architecture Diagram

(Textual representation)

```
csharp
CopyEdit
[Clients: Web/Mobile]
     |
     v
[API Gateway] --> [Auth Service]
     |
     v
[Payment Processor] ---> [Bank Connectors]
     |
     v
[Fraud Detection] (parallel processing)
     |
     v
[DB Cluster] <--> [Cache]
     |
     v
[Notification Service]
```

---

7. Data Flow

Step-by-step Process (Detailed):

1. **Payment Initiation:** Customer initiates payment via merchant application.
2. **API Gateway:** Request is validated for basic schema correctness.
3. **Authentication:** JWT token is verified; unauthorized requests are rejected.

4. **Payment Processor:**
    - Extracts and validates card/bank details.
    - Sends request to **Bank Connector**.
5. **Bank Connector:**
    - Converts request to bank-specific format.
    - Sends to the bank securely.
6. **Bank Response:** Response (success/failure) returned.
7. **Fraud Check:** Runs asynchronously.
8. **Transaction Update:** Payment status stored in PostgreSQL.
9. **Notification:** Email/SMS sent.
10. **Merchant Webhook:** Merchant notified via webhook.

---

8. Technology Stack

Frontend:

- Angular
- ReactJS (for merchant widgets)

Backend:

- Java Spring Boot
- Node.js (for webhook processing)

Databases:

- PostgreSQL: Transactional data
- MongoDB: JSON logs

Cache:

- Redis

Queue:

- Kafka

Cloud Provider:

- AWS (EKS, Lambda, S3, API Gateway, RDS)

Monitoring:

- Prometheus
- Grafana
- AWS CloudWatch

---

9. APIs

Sample APIs

- **POST** `/api/v1/payment/initiate`
- **GET** `/api/v1/payment/status/{transactionId}`
- **GET** `/api/v1/user/history`

Sample Response:

```
json
CopyEdit
{
  "transactionId": "TXN987654321",
  "status": "SUCCESS",
  "amount": 2000,
  "currency": "USD"
}
```

---

10. Security

1. **Encryption:**
   - TLS 1.3 for data in transit
   - AES-256 for data at rest
2. **Compliance:**
   - PCI DSS Level 1
3. **Authentication:**
   - OAuth 2.0
   - JWT
4. **Additional:**
   - HSM for secure key management

---

11. Monitoring and Logging

- Centralized logging with ELK Stack.
- Metrics collected:
  - TPS
  - Latency
  - Error rate

---

12. High Availability

- **Active-active deployment** across multiple availability zones.
- **Auto-scaling** based on CPU and TPS metrics.

---

13. Scalability Strategy

- **Horizontal scaling** for microservices.
- **Database read replicas**.
- **Caching of frequently accessed data**.

---

14. Disaster Recovery

- **RPO:** 5 minutes
- **RTO:** 15 minutes

Strategy:

- Multi-region backups.
- Failover DNS.

---

15. Non-Functional Requirements (Detailed)

- **Performance:**
  - < 200 ms average latency per transaction.
  - 95th percentile latency < 400 ms.
- **Reliability:**
  - 99.9% uptime SLA.
- **Capacity Planning:**
  - Must handle seasonal spikes.

---

16. Admin Dashboard Features

- Transaction search by ID, date, or user.
- Reporting on TPS, success rate, revenue.
- Fraudulent activity visualization.

---

17. Future Enhancements

1. AI-driven risk scoring using deep learning.
2. Integration with blockchain for transaction transparency.
3. Support for biometric authentication.

---

18. Glossary

- **TPS:** Transactions Per Second
- **HLD:** High-Level Design
- **PCI DSS:** Payment Card Industry Data Security Standard

---

19. References

- PCI DSS Compliance Guidelines
- AWS Well-Architected Framework
- ISO 27001 Standards

---

20. Conclusion

This document provides a detailed view of the Payment Gateway Platform, including all major modules, workflows, APIs, security measures, and future plans.
By following a microservices-based design and cloud-native approach, the platform ensures scalability, reliability, and high performance.

End of Document