Simple Payment Processing Service (PPS) - Spring Boot 3.x

1. Project Overview and Value Proposition

The **Simple Payment Processing Service (PPS)** is a mission-critical backend application built with **Java 17 and Spring Boot 3.x**. It serves as a secure, insulating middleware layer, allowing internal merchant applications to initiate and manage financial transactions without tightly coupling to specific Nigerian Payment Gateways (PGs) like Paystack or Flutterwave.

🎯 Key Engineering Goals

- **Transactional Integrity:** Guaranteed status tracking and reconciliation.
- **Decoupling:** Abstraction of external PG APIs into a single, unified interface.
- **Resilience:** Handling asynchronous status updates via secure webhooks.

2. Core Design Principles

2.1 Idempotency Guarantee (Critical)

To prevent severe errors like double-charging customers, the service implements strong **idempotency**.

- **Mechanism:** Every inbound initiate request must provide a unique Idempotency-Key header.
- **Process:** The Service layer checks the key against the database **before** executing any external payment logic. If the key is found and the transaction is already marked as PENDING or SUCCESS, the original status is immediately returned, preventing re-execution.

2.2 Layered Monolith Architecture

The application adheres to a clean layered architecture, facilitating separation of concerns and future scalability:

1. **Controller:** Handles API routing, input validation, and security.
2. **Service:** Contains all core business logic (Idempotency, Status Transition).
3. **Gateway:** Contains vendor-specific clients (e.g., PaystackClient) and translates internal requests to external API calls.
4. **Repository:** Persistence layer using Spring Data JPA for PostgreSQL.

3. Technology Stack and Dependencies

Component	**Technology**	**Role**
Backend	Java 17 / Spring Boot 3.x	Core API Framework

Database	PostgreSQL	Primary persistence for Transaction and Merchant data.
ORM	Spring Data JPA / Hibernate	Data access and entity mapping.
Security	Spring Security	API key authentication and endpoint protection.
Messaging	**Apache Kafka** / Spring Kafka	**Reliable asynchronous event processing and notification system.**
Build Tool	Maven	Project management and compilation.

4. Local Development and Setup

4.1 Prerequisites

- Java Development Kit (JDK) 17+
- Maven 3.6+
- Docker (Recommended for running PostgreSQL and Kafka)

4.2 Database Setup (PostgreSQL)

For running in the prod profile, use Docker to spin up a local instance:

```
# Start the dedicated PostgreSQL container
docker run --name pps-postgres -e POSTGRES_USER=ppsuser -e
POSTGRES_PASSWORD=ppspass -e POSTGRES_DB=ppsdb -p 5432:5432 -d postgres
```

4.3 Environment Variables (Security)

The application **must** be configured with the following variables. **Crucially, sensitive keys are not hardcoded.**

Variable Name	**Description**	**Example Value**
DB_USERNAME	PostgreSQL database user.	ppsuser
DB_PASSWORD	PostgreSQL database password.	ppspass
PAYSTACK_SECRET_KEY	Secret API key for Paystack Gateway.	sk_live_xyz123abc
MERCHANT_API_KEY	Internal key used by merchant apps for Basic Auth.	merchant123_api_secret
KAFKA_BROKERS	Connection string for Kafka cluster.	localhost:9092

4.4 Build and Run

1. **Clone the Repository:**
   ```
   git clone [your-repository-url]
   cd simple-payment-processing-service
   ```

2. **Build the Project:**
   ```
   mvn clean install
   ```

3. **Run with Production Profile (Requires Docker DB running):**
   ```
   java -jar target/pps-0.0.1-SNAPSHOT.jar --spring.profiles.active=prod
   ```

   The application will start on http://localhost:8080.

5. API Specification

All synchronous endpoints require **API Key Authentication** (configured via Spring Security, typically using the Authorization: Basic header derived from the MERCHANT_API_KEY).

5.1 POST /api/v1/transactions/initiate

Initiates a payment request, logs it, and returns a redirect URL for customer authorization.

Header	Example	Description
Authorization	Basic base64key	Merchant API Key for authentication.
Idempotency-Key	order-A92B3C-202501	Unique key to prevent duplicate calls.

Request Body (JSON):

```
{
  "amount": 15500.00,
  "currency": "NGN",
  "merchantRef": "ORDER-99342",
  "customerEmail": "customer.name@example.com",
  "paymentMethod": "CARD"
}
```

Successful Response (200 OK):

```
{
```

```
  "transactionId": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
  "status": "PENDING",
  "authorizationUrl":
"[https://paystack.co/pay/a1b2c3d4e5f6](https://paystack.co/pay/a1b2c3d4e5f6)"
}
```

5.2 GET /api/v1/transactions/{id}

Retrieves the current status of a transaction using the internal transactionId.
Successful Response (200 OK):

```
{
  "transactionId": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
  "status": "SUCCESS",
  "amount": 15500.00,
  "pgReference": "T67890FGHIJ"
}
```

5.3 POST /api/webhooks/pg/paystack (External)

This endpoint is dedicated to receiving inbound real-time status updates from the Payment Gateway.

- **Security:** This endpoint performs **Signature Verification** (using the PG Secret Key) to ensure the payload originated from the trusted Gateway.
- **Processing:** It immediately writes the raw payload to the WebhookEvent table and triggers an asynchronous listener for status update and merchant notification.

6. Testing and Quality Assurance

- **Unit Tests:** JUnit 5 is used for testing Service layer logic, ensuring methods like checkIdempotency() and updateStatus() work reliably.
- **Integration Tests:** Utilizes Testcontainers to spin up a PostgreSQL instance for true integration testing against the database schema and JPA repositories.
- **Mocking:** Mockito is used extensively to mock external dependencies (like PaystackClient), isolating our service logic from third-party API availability.

7. Future Road Map

The current architecture is designed to evolve gracefully:

- **Multi-Gateway Support:** Introduce a new GatewayProvider interface to easily integrate additional PGs (e.g., Flutterwave) by simply implementing a new client class.
- **Admin Dashboard:** Build a simple Spring Boot Admin module for transaction monitoring, search, and manual reconciliation tools.