Product Requirements Document (PRD): Simple Payment Processing Service (PPS)

1. Introduction and Goals

Project Name: Simple Payment Processing Service (PPS)
Target Audience: Internal Merchant Applications (acting as a secure middleware)
Version: 1.0

1.1 Project Objective

To build a resilient, secure, and highly available service layer (abstraction layer) that enables internal merchant applications to initiate and manage customer transactions without integrating directly with external Payment Gateways (PGs) like Paystack or Flutterwave.

1.2 Core Goal: Transactional Integrity

The primary goal is to ensure the integrity of every transaction, preventing double charges and ensuring reliable, asynchronous status updates back to the merchant.

1.3 Key Functional Requirements (What it Does)

ID	Feature	Description	Priority
FR1.1	**Transaction Initiation**	Accept an inbound API request from a Merchant containing payment details (amount, currency, customer reference, idempotencyKey) and forward the request to the configured external Payment Gateway.	P1
FR1.2	**Transaction Status Query**	Allow Merchant applications to query the current status of a transaction (e.g., PENDING, SUCCESS, FAILED) using the internal transactionId.	P1

FR1.3	**Gateway Abstraction**	The PPS must handle the unique API contracts, authentication, and error codes of different PGs, exposing a single, unified transaction API to the Merchant.	P1
FR1.4	**Asynchronous Webhook Handling**	Securely receive and process real-time status updates (webhooks) from external PGs and update the internal transaction status accordingly.	P1
FR1.5	**Merchant Notification**	After receiving a PG webhook, the PPS must send a secondary, reliable webhook notification back to the originating Merchant's application endpoint.	P1

1.4 Non-Functional Requirements (How Well it Works)

Category	Requirement	Target
Security	**Data Masking**	Card numbers/CVVs must *never* be stored. Sensitive data like customer full names and emails must be encrypted at rest (or proxied).
Integrity	**Idempotency**	Implement strong IdempotencyKey handling to guarantee that identical requests are processed only once, preventing double charging.
Reliability	**Uptime**	The service must target 99.9% uptime.
Performance	**Latency**	Transaction initiation latency

		(excluding PG response time) must be less than 50ms.
Scalability	**Architecture**	Built using a scalable Spring Boot monolith (layered) structure that can be split into microservices (e.g., Transaction, Webhook, Settlement) in the future.