Software Requirements Specification (SRS)

Headless E-Commerce Service Platform

Project: Motia Hackathon | **Version:** 1.5 - Multi-Tenant SaaS Scalability, Integrity & Plugin Architecture

1. Introduction

The Headless E-Commerce Service Platform is a standalone SaaS engine designed to power thousands of independent e-commerce brands. This system provides isolated, white-label order processing logic, state, and streaming capabilities.

1.1 SaaS Scalability & Integrity Objectives

- **Multi-Tenant Isolation:** Strict data and event separation between merchant accounts.
- **Extensible Logic:** Merchants inject custom logic into the order flow via Webhooks and Plugins.
- **System Integrity & Trust:** Ensuring every transaction is unique and protected against technical failure or network instability.

2. Core Functional Requirements

2.1 FR-1.0: Tenant-Aware API Intake

The API Step must identify the merchant via an X-Store-ID header. The system must validate that the API key has permissions for that specific store ID before processing.

3. Asynchronous Workflow & Merchant Customization

3.1 FR-2.0: The Saga Pattern with Merchant Overrides

If a merchant has a custom "Fraud Detection" rule, the Payment Step must pause and wait for a fraud.verified event before proceeding to inventory deduction.

3.2 FR-2.1: Headless Fulfillment Orchestration

Merchants can toggle these sub-steps in their dashboard:

1. **Reserve Shipping:** Interfaces with the merchant's preferred carrier (FedEx, UPS, DHL).
2. **Notify Warehouse:** Supports 3PL integration via outbound events.
3. **Generate Invoice:** White-labeled PDF generation with merchant branding.
4. **Trigger Delivery Pipeline:** Real-time tracking streams branded to the store.

4. Plugin Architecture & Developer Extensibility

4.1 FR-4.0: Standardized Plugin Interface

To allow third-party developers to "plug in" to the system, the platform provides a modular interface:

- **Plugin Manifest (plugin.json):** Every plugin must provide metadata including plugin_id, version, hooks_subscribed, and required_permissions.
- **The "Hook" System:** Developers can register logic to specific lifecycle hooks such as onOrderCreated, beforePayment, or afterFulfillment.

4.2 FR-4.1: Seamless Installation & Hot-Swapping

- **Plugin Registration:** Developers can install a plugin by providing its URL or package name. The system registers this against the X-Store-ID.
- **Sandboxed Execution:** Plugins run in an isolated context to ensure that a bug in a developer's plugin does not crash the core Order Processor.
- **Merchant Toggle:** Merchants can enable or disable installed plugins via their dashboard without restarting the core service.

5. SaaS Strategy & Data Isolation

5.1 SR-3.0: Multi-Tenant State Partitioning

- **Requirement:** Merchant A must never access Merchant B's data.
- **Implementation:** Use State Groups with path-based isolation: /artifacts/{appId}/public/data/{storeId}.

5.2 SR-3.1: Merchant Webhook Engine

A dedicated Event Step webhook.dispatcher subscribes to all order.* events and fires signed requests to merchant-defined endpoints.

6. Technical Requirements & Reliability (The "Golden" Feature)

6.1 TR-5.1: High-Availability Idempotency Engine

Each merchant request must include a unique Idempotency-Key.

- **Prevents Double-Charging and Duplicate Orders:** Identifies the key and prevents the order from being duplicated across the event bus if a connection fails.
- **Maintains System Integrity Under Stress:** Verifies previous requests via state check to prevent corrupted data during high-volume spikes.
- **Enhances Trust:** Guarantees "Exactly-Once" processing logic for merchant financial safety.

6.2 TR-5.2: API Rate Limiting per Tenant

Implement "Fair Use" throttling per Store-ID to prevent one viral store from impacting others.

7. Advanced Observability

7.1 FE-4.0: The Platform "Pulse" Dashboard

A global view for the service provider to monitor the health of all stores and active plugin performance.