Software Requirements Specification (SRS)
Headless E-Commerce Service Platform

Motia Hackathon Project

Version 1.4 - Multi-Tenant SaaS Scalability

1 Introduction

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

1.1 SaaS Scalability Objectives

To scale as a service provider, the platform implements:

- **Multi-Tenant Isolation:** Strict data and event separation between different merchant accounts.
- **Extensible Logic (Webhooks):** Allowing merchants to inject custom logic into the order flow.
- **Elastic Resource Allocation:** Scaling compute power based on individual store traffic.

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. Motia's `context.auth` 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 be able to pause and wait for a `fraud.verified` event before proceeding to inventory deduction.

3.2 FR-2.1: Headless Fulfillment Orchestration

Individual merchants can toggle these sub-steps in their dashboard:

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

4 Scaling as a Service Provider (SaaS Strategy)

4.1 SR-3.0: Multi-Tenant State Partitioning

Requirement: Merchant A must never be able to access Merchant B's data, even at the database level.

Implementation: Use Motia's `State Groups` with path-based isolation: `/artifacts/{appId}/public/dat`
This ensures physical data separation and simplifies compliance with data privacy laws (GDPR/CCPA).

4.2 SR-3.1: Merchant Webhook Engine

Requirement: Merchants need to sync their orders with external ERPs or CRMs.
Implementation: A dedicated Event Step `webhook.dispatcher` that subscribes to all `order.*` events and fires signed `POST` requests to merchant-defined endpoints.

4.3 SR-3.2: Analytics & Aggregation Stream

Requirement: Provide merchants with real-time sales dashboards.
Implementation: Use a segmented stream `context.streams.merchant_stats[storeId]` to aggregate live sales data without cross-pollinating data between tenants.

5 Advanced Observability

5.1 FE-4.0: The Platform "Pulse" Dashboard

A global view for the service provider (you) to monitor the health of all stores, identifying if a specific merchant's integration is failing or if a specific region is experiencing high latency.

6 Technical Requirements

6.1 TR-5.1: High-Availability Idempotency

Each merchant request includes an `Idempotency-Key`. If a merchant's server retries an API call, Motia's state check ensures the order isn't duplicated across the multi-tenant event bus.

6.2 TR-5.2: API Rate Limiting per Tenant

Prevent one viral store from crashing the entire platform. Implement "Fair Use" throttling using Motia's global state to track requests per `Store-ID`.