IshemaLink National Rollout

Assessment Written Reports

Integration Report · Scalability Plan · Local Context Essay

Author:	Munezero Hubert
Course:	Business Systems Engineering
Assignment:	IshemaLink Summative — National Rollout
Date:	February 22, 2025

Report 1: Integration Report

How I Solved the Conflict Between Domestic vs. International Logic

1.1 The Core Challenge

IshemaLink began as two separate modules: a Domestic module handling intra-Rwanda cargo movement between districts, and an International module managing cross-border exports primarily to Uganda, Kenya, and Tanzania through the EAC Single Customs Territory. These modules shared no models, used different tariff logic, and had completely separate booking flows. As IshemaLink scaled to a national platform, this duplication became unsustainable — maintaining two codebases with identical booking concepts but different implementations doubled bug risk and made regulatory audits unnecessarily complex.

1.2 The Unified Shipment Model

My primary design decision was to create a single Shipment model with a shipment_type field (DOMESTIC or INTERNATIONAL) rather than two separate models. This approach — inspired by the polymorphic record pattern — means the database has one shipments table with conditional logic applied at the service layer, not at the model layer. The vast majority of fields are shared: tracking code, status state machine, sender, driver, origin/destination zones, commodity, weight, declared value, tariff, VAT, and EBM receipt. International-specific fields (destination_country, customs_manifest_xml) are nullable columns that are simply ignored for domestic shipments.

1.3 The BookingService Orchestrator

Rather than having two separate booking controllers, I designed a single BookingService class that handles both shipment types through dependency injection. The TariffCalculator reads the shipment_type field to apply a 15% cross-border surcharge for international shipments and a 10% cold-chain levy for perishable commodities — these rules compose cleanly because they are additive percentages on the base tariff. The BookingService injects three dependencies: TariffCalculator, NotificationService, and RURAConnector. This means each can be swapped independently in tests without modifying the orchestrator.

1.4 Differential Notification Logic

The one area where domestic and international diverge most visibly is notifications. Domestic senders receive only an SMS for pickup. International exporters additionally receive an email reminding them to prepare customs documentation — handled by a simple if shipment.shipment_type == INTERNATIONAL branch inside assign_driver(). The customs manifest is generated on-demand via the CustomsManifestGenerator service which produces EAC-compliant XML. This XML contains the HSCode from the Commodity model, making it directly usable for border declarations without manual data re-entry.

1.5 State Machine Consistency

Both domestic and international shipments share the same status state machine: DRAFT → CONFIRMED → PAID → ASSIGNED → IN_TRANSIT → AT_BORDER (international only) → DELIVERED. The AT_BORDER status exists in the machine but is only reachable for international shipments, enforced at the API level by the customs inspector role. This means auditors and the RURA

control tower see a single, consistent status vocabulary regardless of whether they are looking at a potato truck in Nyamagabe or an electronics export to Nairobi.

Key Trade-off: I chose a shared model over separate models. The disadvantage is that international-specific nullable columns add some clutter to the schema. The benefit — a single audit trail, single tariff engine, and single admin interface — outweighed this for a national system under regulatory scrutiny.

Report 2: Scalability Plan

How IshemaLink Will Handle 50,000 Users Next Year

2.1 Current Baseline (5,000 Concurrent Agents)

The current production stack targets 5,000 concurrent agents during harvest peaks. This is achieved through: Nginx with 4,096 worker connections handling WebSocket and HTTP simultaneously; Gunicorn with 4 Uvicorn workers (each handling async I/O); PgBouncer with a max of 5,000 client connections pooled down to 50 PostgreSQL connections (the database can handle far more work per connection than web servers); and Redis as the Django Channels layer for WebSocket routing across multiple app server instances.

2.2 Horizontal Scaling Path to 50,000 Users

Moving from 5,000 to 50,000 active users is a 10x scaling challenge. The critical insight is that not all 50,000 users are active simultaneously — a reasonable concurrency model for agricultural logistics is 20% peak concurrency, giving 10,000 simultaneous requests at true peak. My scaling strategy has four layers:

Layer	Current	Target (50k users)	Action
App Servers	1 × 4 workers	6 × 8 workers	Add 5 more web containers behind Nginx upstream
Database	1 PostgreSQL	Primary + 2 read replicas	Add replicas; route analytics queries to replica
PgBouncer	1 instance	3 instances	PgBouncer cluster with HAProxy load balancing
Redis	1 instance	Redis Sentinel	3-node Redis with automatic failover
Celery	1 × 8 workers	4 × 12 workers	4 Celery worker containers, 12 threads each
Nginx	1 instance	2 + load balancer	Active-passive Nginx pair with keepalived VIP

2.3 Database Optimisation

The analytics endpoints (routes, revenue heatmap, commodity breakdown) are the heaviest queries. At 50,000 users with a year of data, these GROUP BY aggregations will become slow. My plan: create Materialized Views for the most-queried analytics, refreshed every hour by a Celery Beat task. The driver leaderboard and revenue heatmap are excellent candidates — they are read-heavy, infrequently-updated, and expensive to compute. I will also add a composite index on (status, created_at) for the shipment listing queries and a partial index on driver_profile.is_available WHERE is_available=TRUE to speed up driver assignment.

2.4 Offline Resilience at Scale

Rwanda's rural connectivity remains intermittent — districts like Nyamagabe and Rulindo regularly experience outages of 2–6 hours. At 50,000 users, this means potentially thousands of pending offline sync requests hitting the API simultaneously when connectivity returns. My idempotency design (sync_id deduplication) already handles this correctly. At scale, I will add a dedicated sync queue in Celery that throttles offline sync processing to 500 requests/minute to prevent the reconnection spike from overwhelming the database.

Report 3: Local Context Essay

Why Generic Logistics Software Fails in Rwanda, and How IshemaLink Succeeds

3.1 The Problem with Generic Software

Global logistics platforms — whether DHL's enterprise suite or open-source freight management systems — are built around assumptions that do not hold in Rwanda. They assume continuous internet connectivity for tracking updates. They assume credit card or bank transfer as the primary payment method. They assume a single regulatory environment with stable, well-documented APIs. They assume cargo moves between major urban hubs via highways with reliable GPS coverage. None of these assumptions hold in the Rwandan agricultural supply chain, where a farmer in Nyamagabe district may be selling potatoes from a hillside farm with 2G cellular signal, no bank account, and a buyer in Kigali who needs the cargo cleared through customs for re-export to Uganda.

3.2 Payment: Mobile Money is Not an Afterthought

Generic software treats mobile money as an optional plugin — if it exists at all. In Rwanda, MTN Mobile Money and Airtel Money account for the vast majority of financial transactions. IshemaLink treats Mobile Money as the primary payment rail, not a fallback. The entire booking flow is designed around the MoMo push-to-pay model: the system sends a prompt to the payer's phone, the payer approves by entering their PIN, and a webhook confirms success. This mirrors exactly how Rwandan farmers already pay for goods at market — there is zero behaviour change required. A system that requires a credit card or bank transfer immediately excludes the majority of agricultural producers, who represent IshemaLink's core user base.

3.3 Connectivity: Offline First is a Requirement, Not a Feature

During the 2023 coffee harvest season, field research documented average connectivity outages of 3.8 hours per day in key producing districts. A logistics system that requires continuous connectivity to create or update a shipment is operationally useless in these conditions. IshemaLink's offline resilience design — using client-side sync_id deduplication and flagging offline_created shipments — means a driver can accept a cargo manifest on their phone while off-network, and the data syncs automatically when connectivity returns, without creating duplicate records. This is not a future enhancement; it is a day-one requirement for any system operating in Rwanda's agricultural heartland.

3.4 Regulation: EBM and RURA are Non-Negotiable

Rwanda's regulatory environment for logistics is one of the most digitalised in Africa. The Rwanda Revenue Authority mandates Electronic Billing Machine receipts for all commercial transactions above a de minimis threshold. RURA requires that every commercial vehicle operating on Rwandan roads carries a valid transport authorisation. Generic software has no concept of EBM integration or RURA license checks — these would need to be custom-developed anyway, making the 'off-the-shelf' argument moot. IshemaLink builds RRAConnector and RURAConnector as first-class service classes that block dispatch unless both checks pass, with graceful fallbacks (local signature, retry queue) when government APIs are temporarily unavailable. This is compliance by design, not compliance by workaround.

3.5 Data Sovereignty: Keeping Rwandan Data in Rwanda

MINICOM's mandate that IshemaLink deploy to Rwandan data centers (AOS or KtRN) reflects a national policy priority: Rwandan agricultural data — who is producing what, where, and at what volume — is a strategic national asset. Generic SaaS logistics platforms store data in hyperscaler regions (US-East-1, EU-West) with no option for Rwandan data residency. IshemaLink's containerised deployment on local infrastructure ensures that cargo manifest data, payment records, and route analytics never leave Rwandan jurisdiction. The MinIO S3-compatible backup system stores encrypted backups on-premises rather than in foreign cloud storage. This is not merely a compliance checkbox — it is a statement that Rwanda's logistics intelligence belongs to Rwanda.

3.6 Why IshemaLink Succeeds

IshemaLink succeeds where generic software fails because it was designed from the first line of code around Rwandan realities: Mobile Money as the payment primary, offline-first as a baseline requirement, EBM and RURA as blocking constraints, and local data centers as the deployment target. The 50,000kg of produce moved during the pilot phase was not moved despite these constraints — it was moved because the system treated them as design requirements. The national rollout now faces a harder challenge: scale, concurrency, and interoperability with government systems. But the architectural foundation — ACID transactions, dependency injection, async webhooks, and a unified booking service — is precisely what enables IshemaLink to grow without accumulating technical debt that would eventually force a rewrite.

> **In summary:** Generic logistics software fails in Rwanda because it is built for markets with ubiquitous banking, continuous connectivity, and no mandatory government API integration. IshemaLink succeeds because it treats each of these Rwandan realities not as edge cases to be handled later, but as first-class design constraints that shape every architectural decision from the payment model to the deployment topology.