# Forique Marketplace Platform: Technical Architecture & Product Strategy Report

## 1. Executive Summary and Strategic Alignment

### 1.1 Document Purpose and Scope

This document serves as the comprehensive Technical and Product Requirement Specification for the "Forique" multi-vendor marketplace. It translates the high-level functional objectives identified in the initial functional requirement documents into a concrete, execution-ready architectural blueprint and granular product workflow. The primary objective is to define the engineering standards—specifically utilizing **Quarkus** and **Kotlin** within a reactive microservices architecture—to build a platform capable of high-frequency trading, robust inventory management through "latch-on" logic, and seamless seller onboarding tailored to the rigorous Indian regulatory landscape.1

The scope encompasses the end-to-end design of the **Seller Web Portal** and the **Super Admin Web Portal**. It addresses the critical need for a centralized backend that supports high concurrency for inventory updates, complex state management for order fulfillment, and automated compliance checks (KYC) involving GST and PAN verification.1 Furthermore, this document establishes the mathematical models for Key Performance Indicators (KPIs) and the algorithmic logic for the "Buy Box"—the primary driver of sales in a multi-seller environment.4

### 1.2 The Strategic Imperative of Reactive Systems

In the highly competitive domain of e-commerce, milliseconds translate directly to revenue. Traditional blocking architectures, often characterized by the Thread-per-Request model, frequently falter under the specific pressures of marketplace dynamics. Events such as flash sales, inventory synchronization across thousands of sellers, and real-time computation of "Buy Box" winners generate burst traffic and I/O-heavy operations that create unacceptable bottlenecks in legacy systems. Consequently, Forique will adopt a **Reactive Systems** approach.5

By leveraging **Quarkus** (the "Supersonic Subatomic Java") with **Kotlin**, the platform achieves a paradigm shift in performance and resource efficiency:

* **Responsiveness:** Non-blocking I/O ensures the system remains responsive even under heavy load, utilizing the underlying Vert.x engine to handle thousands of concurrent connections with a minimal thread footprint.6
* **Resiliency:** Failures in auxiliary services—such as a logistics partner API or a payment gateway—are isolated through reactive patterns, preventing cascading system-wide outages.5
* **Elasticity:** The lightweight nature of Quarkus allows for rapid scaling of individual microservices in a Kubernetes environment, reducing "cold start" times and optimizing cloud infrastructure costs.8
* **Message-Driven Communication:** Asynchronous message passing via the Vert.x Event Bus ensures loose coupling between critical domains like the Order Service, Inventory Service, and Notification Service, facilitating independent evolution and deployment.9

### 1.3 Core Product Differentiators

The Forique platform distinguishes itself through three specific architectural and product decisions:

1. **Latch-On Inventory Model:** Unlike single-seller platforms, Forique utilizes a catalog-centric model where multiple sellers map offers to a single "Golden Record" product. This prevents catalog duplication, improves search relevance, and enables price transparency for the consumer.10
2. **Automated India-Stack Onboarding:** Integration with India's digital infrastructure—specifically Penny Drop, PAN, and GSTIN APIs—automates trust verification, reducing vendor onboarding time from days to minutes while ensuring strict regulatory compliance.12
3. **Algorithmic Buy Box:** A transparent, meritocratic algorithm awards the "Add to Cart" button to sellers based on a weighted score of Price, Speed, and Reliability, ensuring that the platform naturally incentivizes high-quality seller behavior.13

## 2. Technical Architecture: The Reactive Microservices Ecosystem

### 2.1 Architectural Pattern: Hexagonal Microservices

The Forique platform is architected as a distributed system using the **Hexagonal Architecture (Ports and Adapters)** pattern. This design principle is critical for isolating the core business logic (the Domain) from the external interfaces (API adapters, Database adapters, Messaging adapters). This decoupling allows for robust testing, easier technology migration, and distinct separation of concerns.15

In the context of Forique, the "Core" contains the pure Kotlin business rules (e.g., "A seller cannot list a product without active KYC"). The "Ports" define the interfaces for these rules, while the "Adapters" are the Quarkus extensions (Hibernate Reactive Panache, RESTEasy Reactive) that connect the core to the outside world.

#### 2.1.1 Domain Decomposition

The backend is decomposed into specific, domain-driven microservices to ensure modularity and scalability:

| **Microservice** | **Responsibility** | **Key Interactions** |
| --- | --- | --- |
| **Identity & Access (IAM)** | Authentication (OTP/Email), RBAC, Session Management. | Interacts with Keycloak/OIDC providers. |
| **Catalog Service** | Manages "Product Master," categories, attributes, and media. | Uses Elasticsearch for full-text search.17 |
| **Offer & Inventory** | Manages seller-specific data (Price, Stock, SKU) mapped to the Catalog. | High-concurrency inventory decrement logic. |
| **Order Management (OMS)** | Orchestrates the order lifecycle (Placed, Confirmed, Shipped, RTO). | Implements Saga Pattern for distributed transactions.18 |
| **Seller Lifecycle** | Manages Onboarding, KYC verification flows, and document storage. | Integrates with 3rd-party Verification APIs. |
| **Settlement & Commission** | Calculates platform fees, taxes (GST/TCS), and manages payouts. | Generates financial ledgers and invoices. |

### 2.2 The Technology Stack: Quarkus & Kotlin

The selection of Quarkus and Kotlin is a strategic decision to combine the maturity of the Java ecosystem with modern reactive paradigms.

#### 2.2.1 Kotlin and Mutiny: The Reactive Glue

Business logic is written in **Kotlin**, leveraging its conciseness, null-safety, and interoperability with Java.19 However, the defining characteristic of the codebase is the use of **Mutiny**, the reactive programming library native to Quarkus.

Unlike imperative programming where threads block while waiting for database responses, Mutiny utilizes an event-driven model.

* **Uni<T>:** This type is used for operations returning a single result (or failure). For example, findProductById(id) returns a Uni<Product>. The thread does not wait; it subscribes to the result.20
* **Multi<T>:** This type is used for data streams. For example, streamOrderUpdates() returns a Multi<Order>, allowing the frontend to receive real-time updates via Server-Sent Events (SSE) or WebSockets.21

**Architectural Insight:** The unification of imperative and reactive programming in Quarkus allows developers to write code that looks sequential (using Kotlin Coroutines or Mutiny's fluent API) but executes non-blocking I/O under the hood. This significantly lowers the cognitive load compared to traditional callback-based reactive frameworks.5

#### 2.2.2 Inter-Service Communication: The Vert.x Event Bus

In a microservices architecture, synchronous HTTP calls (REST) between services create tight coupling and latency chains. If the Order Service calls the Inventory Service, which calls the Database, a failure in the database cascades back to the Order Service.

Forique mitigates this by using the **Vert.x Event Bus** for asynchronous, intra-cluster communication.7

* **Mechanism:** Services publish messages to specific "addresses" on the Event Bus. Consumers (other services) subscribe to these addresses.
* **Pattern:** Fire-and-Forget or Request-Reply (Non-blocking).
* **Use Case:** When a seller updates inventory, the Inventory Service processes the update and immediately acknowledges the request. It then publishes an inventory.updated event. The Catalog Service consumes this to update the search index, and the Notification Service consumes it to alert users. This ensures the Seller Portal remains snappy while consistency is handled in the background.9

### 2.3 Data Persistence Strategy: Polyglot & Reactive

The platform utilizes a persistence strategy that selects the best data store for the specific domain problem, all accessed via reactive drivers.

#### 2.3.1 Operational Data (PostgreSQL + Hibernate Reactive)

Core transactional data (Orders, Sellers, Financials) resides in **PostgreSQL**. The interaction layer uses **Hibernate Reactive with Panache**, which supports non-blocking database drivers.

* **Why not standard Hibernate?** Standard JDBC drivers block the thread. In a high-traffic marketplace, this would quickly exhaust the thread pool. Hibernate Reactive uses the Vert.x PostgreSQL client to perform non-blocking queries.23
* **Schema Design:** The schema is highly normalized for transactional integrity but utilizes **JSONB** columns for flexible product attributes (see Section 3).

#### 2.3.2 Search & Discovery (Elasticsearch)

The Catalog Service replicates data to **Elasticsearch** (or OpenSearch) to support complex queries (e.g., "Gold earrings under ₹5000"). **Hibernate Search** automatically synchronizes database changes to the search index, abstracting the complexity of keeping the two data stores in sync.17

#### 2.3.3 Bulk Data Processing

A critical requirement for sellers is the ability to upload bulk inventory files (CSV/Excel).

* **Challenge:** Parsing a 10,000-row Excel file is CPU and memory intensive. Doing this on a blocking thread would freeze the application.
* **Solution:** Forique utilizes **Mutiny** to process file streams. The system reads the file chunk-by-chunk, parsing rows reactively. Valid rows are batched and inserted using PostgreSQL COPY commands or batched reactive inserts, maximizing throughput while keeping memory footprint low.26

## 3. The "Latch-On" Inventory Logic

### 3.1 Conceptual Framework: The Golden Record

The "Latch-On" model is the defining feature of modern multi-vendor marketplaces (e.g., Amazon, Flipkart). In this model, the platform acts as the custodian of the catalog. A specific physical product (e.g., "Samsung Galaxy S24, 256GB, Black") exists as a single unique entity in the database—the **Product Master** or "Golden Record."

Sellers do not create new product pages for existing items. Instead, they create an **Offer** that "latches on" to the Product Master. This offer contains only the seller-specific data: Price, Quantity, Shipping SLA, and Condition. This structure prevents search results from being cluttered with duplicates and allows the system to present a price comparison table to the buyer.10

### 3.2 Database Schema Implementation

To support this, the database schema must strictly decouple the product definition from the sales proposition.

#### 3.2.1 The Product Master Table

Managed by the Admin or the first seller who creates the listing (the "Brand Owner"). This table stores immutable facts.

| **Column Name** | **Data Type** | **Constraint** | **Description** |
| --- | --- | --- | --- |
| product\_id | UUID | PK | Unique system identifier. |
| asin\_sku | VARCHAR | UNIQUE, NOT NULL | Global identifier (e.g., ASIN/EAN). |
| title | VARCHAR | NOT NULL | Canonical product name. |
| brand\_id | UUID | FK | Reference to Brand entity. |
| category\_id | UUID | FK | Reference to Category hierarchy. |
| attributes | JSONB | - | Dynamic attributes (Color, Material, etc.). |
| status | ENUM | - | DRAFT, ACTIVE, SUPPRESSED. |

Why JSONB for Attributes?

Jewelry products have highly variable attributes. A ring requires ring\_size, gold\_purity, and diamond\_weight. A necklace needs chain\_length and clasp\_type. An Entity-Attribute-Value (EAV) model, while flexible, suffers from poor query performance due to excessive joins. A flat table with hundreds of nullable columns is unmaintainable.

PostgreSQL JSONB offers the perfect middle ground: flexible schema validation within the application and high-performance indexing (GIN indexes) for filtering queries (e.g., SELECT \* FROM products WHERE attributes @> '{"material": "Gold"}').28

#### 3.2.2 The Seller Offer Table

Managed by individual sellers. This table stores mutable, competitive data.

| **Column Name** | **Data Type** | **Constraint** | **Description** |
| --- | --- | --- | --- |
| offer\_id | UUID | PK | Unique offer identifier. |
| product\_id | UUID | FK | Link to the Product Master. |
| seller\_id | UUID | FK | Link to the Seller. |
| seller\_sku | VARCHAR | - | Seller's internal inventory code. |
| selling\_price | DECIMAL | NOT NULL | The price offered to the customer. |
| mrp | DECIMAL | NOT NULL | Maximum Retail Price (for discount calc). |
| stock\_qty | INT | DEFAULT 0 | Real-time inventory count. |
| fulfillment\_channel | ENUM | - | FBF (Forique), SELLER\_FLEX, DROPSHIP. |
| is\_buy\_box\_winner | BOOLEAN | - | Cached flag for quick lookup. |

### 3.3 The Latching Workflow

The Seller Portal must guide the user through a specific decision tree to ensure catalog hygiene.

1. **Product Search:** The seller enters a unique identifier (UPC, EAN, ISBN) or a product name.
2. **Existence Check:**
   * **Scenario A: Match Found.** The system displays the Product Master details. The seller validates it matches their item. The seller clicks "Sell Yours".31 The system then prompts *only* for Offer details (Price, Qty, Tax Code).
   * **Scenario B: No Match.** The system allows the seller to "Create New Product." This initiates a "Product Creation Request" which enters a moderation queue.
3. **Gating & Authorization:** Before a latch is permitted, the system checks **Brand Gating** rules.
   * *Rule:* Is this Brand restricted?
   * *Check:* Does seller\_permissions table contain an entry for this brand\_id?
   * *Outcome:* If restricted, the seller is blocked and prompted to upload a "Brand Authorization Letter".32

### 3.4 Catalog Conflict Resolution

A major challenge in latch-on models is **Content Conflict**. If Seller A lists a "Gold Ring" and Seller B latches on but tries to change the description to "Silver Ring," the integrity of the catalog is compromised. Forique implements a **Contribution Score Logic** to resolve these conflicts.33

* **Logic:** Every attribute in the product\_master has a "source" and a "confidence score."
* **Hierarchy of Truth:**
  1. **Brand Owner / Super Admin:** Score 100 (Immutable overwrite).
  2. **High-Volume Sellers:** Score 80 (Trusted contributors).
  3. **New Sellers:** Score 20 (Suggestions only).
* **Resolution Mechanism:** When a seller submits an update to a Master attribute, the system compares their score against the existing data's source score. If the new score is higher, the update is applied. If lower, the update is stored in a suggested\_edits table for manual Admin review.35

## 4. Seller Onboarding & KYC Flows (India Context)

### 4.1 The Onboarding State Machine

Seller onboarding is not a single step but a workflow managed by the Seller Lifecycle Service. The account moves through a defined state machine:

REGISTERED $\rightarrow$ EMAIL\_VERIFIED $\rightarrow$ KYC\_SUBMITTED $\rightarrow$ VERIFICATION\_PENDING $\rightarrow$ ACTIVE (or REJECTED).

### 4.2 Comprehensive KYC Verification Architecture

To operate legally in India, the marketplace must comply with GST (Goods and Services Tax) regulations and PMLA (Prevention of Money Laundering Act) guidelines. This requires a robust, API-driven verification stack.1

#### 4.2.1 Step 1: Business Identity (GSTIN)

The Goods and Services Tax Identification Number (GSTIN) is the primary key for business identity.

* **Process:** Seller enters their 15-digit GSTIN.
* **API Interaction:** The backend triggers a call to a third-party compliance API (e.g., Perfios, Signzy, Karza).
* **Data Extraction:** The API returns the Legal Name, Trade Name, Taxpayer Status, and Registered Address directly from the GSTN database.
* **Validation:** The system automatically checks if taxpayer\_status == "Active". If "Suspended" or "Cancelled," the onboarding is halted immediately.36

#### 4.2.2 Step 2: Legal Entity (PAN)

The Permanent Account Number (PAN) is required for income tax purposes.

* **Process:** Seller enters PAN.
* **Validation:** The system validates the PAN structure (e.g., 4th character 'P' for Individual, 'C' for Company, 'F' for Firm).
* **Cross-Verification:** The system calls the NSDL PAN Verification API. It checks if the name returned matches the Legal Name retrieved from the GSTIN step. A fuzzy match logic (e.g., Levenshtein distance > 90% similarity) is used to handle minor spelling variances.37

#### 4.2.3 Step 3: Bank Account Verification (Penny Drop)

To prevent settlement failures and fraud, the platform verifies the bank account before enabling payouts.

* **Process:** The seller provides Account Number and IFSC.
* **The "Penny Drop" Mechanism:** The system initiates a specialized API transaction that deposits a nominal amount (₹1) into the target account.38
* **Success Criteria:** The bank returns the **Registered Beneficiary Name** upon successful deposit.
* **Automated Decision:** The system compares this Beneficiary Name against the Trade Name or Legal Name from the GST step.
  + *Match:* Account is marked VERIFIED.
  + *Mismatch:* Account is flagged RISK\_REVIEW.
  + *Reasoning:* This prevents a scenario where a seller registers a legitimate business but diverts funds to an unrelated personal account.40

### 4.3 Secure Document Management

Sellers must upload digital copies of documents (Cancelled Cheque, GST Certificate, PAN Card).

* **Reactive Upload Pattern:** Handling large file uploads on the application server consumes memory and threads. Forique uses **Quarkus Reactive Multipart** to orchestrate the upload but offloads the data transfer.
* **Presigned URL Flow:**
  1. Frontend requests an upload slot for gst\_cert.pdf.
  2. Backend authenticates the request and calls AWS S3 (or compatible store) to generate a **Presigned URL** with a short expiry (e.g., 5 minutes).
  3. Backend returns this URL to the frontend.
  4. Frontend uploads the file directly to S3.
  5. S3 triggers a webhook (via Lambda/EventBridge) to notify the backend that the upload is complete.
* **Security:** All documents are stored with Server-Side Encryption (SSE-S3). Access is restricted via strict IAM policies, ensuring only authorized Admin roles can view sensitive KYC documents.41

## 5. Forique Seller Web Portal Features

The Seller Portal is the command center for vendors. It is designed as a Single Page Application (SPA) that consumes the microservices via a secure API Gateway.

### 5.1 Dashboard & Business Intelligence

* **Real-time Ticker:** Provides immediate visibility into "Today's Sales," "Pending Orders" (specifically those nearing SLA breach), and "Low Inventory" alerts.
* **Business Health:** Visualizes trends using charting libraries (e.g., Recharts/Chart.js). Critical metrics include Sales vs. Returns graph and the **Buy Box Win Percentage** over time, helping sellers understand their competitive standing.4

### 5.2 Advanced Catalog Management

* **Add Single Product:** Implements the "Latch-on" workflow described in Section 3.3.
* **Bulk Listing Tool:** A robust interface for high-volume sellers to upload CSV/Excel files. The system provides a downloadable "Error Report" for rows that failed validation (e.g., "Row 4: Invalid SKU", "Row 10: Brand Restricted"). This utilizes the reactive file parsing logic to handle large files without timeout.44
* **Image Management:** Integrated tools for basic image manipulation (crop, rotate). The system implements an automated computer vision check (via a lightweight model or 3rd party API) to enforce marketplace standards, such as requiring a pure white background for the main image.

### 5.3 Order Processing (OMS)

* **Order Panel:** A Kanban or List view separating orders by state: New, Packed, Dispatched, Delivered, RTO.
* **Label Generation:** Integration with Logistics Aggregators (e.g., Shiprocket, Pickrr, ClickPost). The seller can click "Ship," and the system automatically generates the shipping label and a GST-compliant tax invoice.1
* **SLA Management:** Visual countdown timers indicate the "Dispatch By" deadline. Orders approaching the breach time are highlighted in red to prevent auto-cancellation and penalty fees.

### 5.4 Payments & Settlements

* **Ledger View:** A transparent financial breakdown for every order ID.
  + Formula: $\text{Net Payout} = \text{Selling Price} - (\text{Commission} + \text{Shipping Fee} + \text{Fixed Closing Fee} + \text{TCS} + \text{TDS})$.
* **Payout Status:** Sellers can track the status of their payouts (e.g., "Processing", "Settled") along with the NEFT/RTGS UTR (Unique Transaction Reference) number for bank reconciliation.45

## 6. Forique Super Admin Web Portal Features

### 6.1 Master Governance Dashboard

* **Platform KPIs:** High-level metrics including Total GMV, Active Sellers, Active Listings, and Daily Active Users (DAU).
* **Risk Radar:** A specialized widget that alerts Admins to anomalous behavior, such as a seller with a sudden spike in sales (potential brushing scam) or a high volume of returns (quality issues).46

### 6.2 Brand & Catalog Governance

* **Brand Approval Workflow:** A dedicated interface for reviewing "Brand Authorization Letters." Admins can Approve or Reject a seller's request to list under a protected brand.1
* **Golden Record Management:** Tools for catalog curation. Admins can merge duplicate products (mapping multiple ASINs to one), split incorrect variations, and "lock" specific attributes (e.g., fixing the "Material" field to "Silver" so no seller can erroneously change it to "Steel").

### 6.3 Dispute Resolution Center

* **Ticket Management:** A unified inbox for disputes raised by Customers (e.g., "Item not received") or Sellers (e.g., "Wrong return received").
* **Evidence Review:** Admins have access to the full order audit trail, chat logs between buyer and seller, and delivery proof images uploaded by the logistics partner.
* **Force Action:** Administrative override capabilities to "Force Refund" (debiting the seller), "Deny Claim," or "Split Liability" (where the platform absorbs the cost).1

### 6.4 Financial Reconciliation & Tax

* **Commission Rule Engine:** A dynamic interface powered by **Kogito/Drools**. Admins can configure complex commission structures without deploying code.48
  + *Example Rule:* IF Category == 'Imitation Jewelry' AND Price < 500 THEN Commission = 5% ELSE Commission = 12%.
* **Tax Reports:** Automated generation of GSTR-8 reports (Tax Collected at Source) which marketplaces in India are required to file monthly.

## 7. Comprehensive KPIs & The Buy Box Algorithm

### 7.1 Marketplace Health KPIs (Admin)

The Admin dashboard relies on a balanced scorecard of "Lagging" (outcome) and "Leading" (predictive) indicators.50

| **KPI Category** | **Metric** | **Definition** | **Target Benchmark** |
| --- | --- | --- | --- |
| **Growth** | **GMV** | Gross Merchandise Value (Total Sales). | > 15% MoM Growth |
| **Growth** | **AOV** | Average Order Value. | Context dependent |
| **Retention** | **NPS** | Net Promoter Score (Customer Satisfaction). | > 50 |
| **Retention** | **Seller Churn** | % of sellers inactive > 30 days. | < 2% |
| **Quality** | **ODR** | Order Defect Rate (Negative feedback + Claims). | < 1% (Strict) |
| **Conversion** | **Cart Abandonment** | % of carts not converted to orders. | < 70% |

### 7.2 Seller Performance Scorecard

These metrics form the "Reputation System" for sellers and directly influence their ability to win the Buy Box.52

| **Metric** | **Calculation Formula** | **Weight in Algorithm** |
| --- | --- | --- |
| **Price Competitiveness** | $\frac{\text{Lowest Marketplace Price}}{\text{Seller Price}}$ | **High (40%)** |
| **Fulfillment Latency** | Avg time to handover vs. SLA | **High (30%)** |
| **Order Defect Rate** | $\frac{\text{Returns + Claims}}{\text{Total Orders}}$ | **Critical** (Threshold < 1%) |
| **In-Stock Rate** | % of time SKU is available (Depth) | **Medium (15%)** |
| **Feedback Score** | Average Rating (1-5 stars) | **Medium (15%)** |

### 7.3 The Buy Box Algorithm Logic

The "Buy Box" is the mechanism that determines which seller's offer is added to the cart when a user clicks the primary "Add to Cart" button. Forique uses a **Weighted Meritocracy Algorithm**, moving away from pure price wars to value total customer experience.

The Algorithm Formula:

$$Score\_s = (W\_p \times P\_{score}) + (W\_f \times F\_{score}) + (W\_r \times R\_{score}) + (W\_i \times I\_{score})$$

**Variable Definitions:**

* $P\_{score}$ (Price Score): Normalized score inversely proportional to price. Lower price yields a higher score, but extremely low prices (potential error/fraud) may be flagged.
* $F\_{score}$ (Fulfillment Score): Assigns points based on the reliability of the shipping method.
  + *Forique Express (FBF):* 100 points (Platform controlled).
  + *Seller Flex (Verified Logistics):* 80 points.
  + *Merchant Fulfilled:* 50 points.
* $R\_{score}$ (Reliability Score): Derived from the Seller's ODR and Feedback metrics. If ODR > 1%, this score drops to 0, effectively removing the seller from the Buy Box.
* $I\_{score}$ (Inventory Score): Penalizes low stock depth. A seller with 1 unit left is risky; a seller with 100+ is reliable.

Execution Strategy:

The Offer Service recalculates this score for every seller associated with a product\_id. The seller with the highest $Score\_s$ has their is\_buy\_box\_winner flag set to TRUE. This calculation is event-driven, triggered by Price Changes, Inventory Updates, or the Daily Performance Refresh job.13

## 8. Technical Implementation Guidelines

### 8.1 Project Structure

To maintain code hygiene and support independent scalability, the project follows a multi-module Maven/Gradle structure 54:

* forique-platform (Root)
  + core-domain (Shared DTOs, Enums, Exceptions, Utility functions)
  + service-catalog (Quarkus Service: Product Master, Search)
  + service-inventory (Quarkus Service: Offers, Stock)
  + service-oms (Quarkus Service: Order Saga, State Machine)
  + infra-gateway (API Gateway configuration)

### 8.2 Handling Concurrency: The Inventory Decrement Problem

One of the hardest problems in e-commerce is handling "Flash Sales" where thousands of users attempt to buy the last remaining units of a product simultaneously. A naive UPDATE will result in overselling (negative stock).

Strategy: Optimistic Locking with Hibernate Reactive

Forique employs Optimistic Locking using the @Version annotation in JPA/Hibernate.

**Implementation Logic:**

1. **Read:** Transaction reads the Inventory record. Current Qty: 10, Version: 1.
2. **Process:** User requests 1 unit. New Qty = 9.
3. **Write:** The system attempts to update:  
   SQL  
   UPDATE inventory SET quantity = 9, version = 2 WHERE id = {id} AND version = 1;
4. **Race Condition Handling:** If another transaction updated the record milliseconds earlier, the version in the DB would be 2. The WHERE clause (version = 1) will fail to match any row.
5. **Outcome:** The database returns "0 rows affected." Hibernate throws an OptimisticLockException.
6. **Recovery:** The application catches this exception and automatically retries the operation (Read -> Process -> Write) or notifies the user that stock has changed.56

### 8.3 Distributed Transactions: The LRA Saga Pattern

Order processing involves multiple services: Inventory (Reserve Stock), Payment (Charge Customer), and OMS (Create Order). If Payment fails, the Reserved Stock must be released. In a distributed system, ACID transactions across services are impossible.

Forique implements the **Saga Pattern** using **MicroProfile LRA (Long Running Actions)** in Quarkus.

* **Coordinator:** An LRA Coordinator manages the transaction lifecycle.
* **Participants:**
  + InventoryService: Annotated with @LRA(Type.REQUIRED). Provides a @Compensate method (e.g., releaseStock).
  + PaymentService: Annotated with @LRA(Type.REQUIRED). Provides a @Compensate method (e.g., refundPayment).
* **Flow:** If the createOrder method throws an exception (e.g., payment declined), the LRA Coordinator invokes the @Compensate methods of all participating services in reverse order, ensuring eventual consistency.18

### 8.4 Observability and Compliance

* **Metrics:** **Quarkus Micrometer** is used to expose operational metrics to Prometheus. Key metrics include http\_server\_requests\_seconds\_max (latency) and business metrics like orders\_placed\_total.
* **Tracing:** **OpenTelemetry** is implemented to trace requests as they propagate through the API Gateway, over the Event Bus, and into the Database. This is vital for debugging "Where did the order get stuck?" in the distributed architecture.6
* **Audit Logging:** To comply with the **Indian IT Act**, all critical data changes (Price updates, User login, Order status changes) are captured via **Debezium** (Change Data Capture) or application-level interceptors and stored in an immutable audit log for the statutory retention period (180 days to 1 year).1

## 9. Conclusion

The Forique platform strategy leverages the cutting edge of the Java ecosystem—**Quarkus and Kotlin Reactive**—to solve the persistent challenges of e-commerce: concurrency, scale, and catalog complexity. By implementing a rigorous "Latch-on" data model, the platform ensures a clean, unified customer experience comparable to top-tier global marketplaces. Simultaneously, the integration of India-specific automated KYC flows and a transparent, data-driven Buy Box algorithm builds deep trust with the seller community. This architecture is engineered not merely for launch, but to scale elastically and reliably as the marketplace grows from hundreds to millions of daily transactions.

#### Works cited

1. Zeera\_Module\_Wise\_FRD.docx
2. Quarkus in Action: Getting started with Quarkus | Red Hat Developer, accessed on January 9, 2026, <https://developers.redhat.com/articles/2025/02/13/quarkus-in-action-getting-started>
3. Instant APIs For KYC, KYB Verification, Onboarding & Compliance in India - Signzy, accessed on January 9, 2026, <https://www.signzy.com/india-api-marketplace>
4. What Are Buy Box Wins? A Seller's Guide to This Critical Amazon Metric - Repricer.com, accessed on January 9, 2026, <https://www.repricer.com/blog/what-are-buy-box-wins/>
5. Quarkus Reactive Architecture, accessed on January 9, 2026, <https://quarkus.io/guides/quarkus-reactive-architecture>
6. Building and understanding reactive microservices using Eclipse Vert.x and distributed tracing | Red Hat Developer, accessed on January 9, 2026, <https://developers.redhat.com/blog/2019/05/13/building-and-understanding-reactive-microservices-using-eclipse-vert-x-and-distributed-tracing>
7. Vert.x Reference Guide - Quarkus, accessed on January 9, 2026, <https://quarkus.io/guides/vertx-reference>
8. Implementing Microservicilities with Quarkus and MicroProfile - InfoQ, accessed on January 9, 2026, <https://www.infoq.com/articles/microservicilities-quarkus/>
9. Using the event bus - Quarkus, accessed on January 9, 2026, <https://quarkus.io/guides/reactive-event-bus>
10. Start Selling Online on Flipkart Seller Hub, accessed on January 9, 2026, <https://seller.flipkart.com/sell-online>
11. Learn how to Latch on Listings on Flipkart Seller Hub - YouTube, accessed on January 9, 2026, <https://www.youtube.com/watch?v=bcfSLJ3wFAY>
12. Enhancing Seller and Customer Onboarding Verification for India's Leading e-Commerce Company - Surepass, accessed on January 9, 2026, <https://surepass.io/case-study/enhancing-seller-and-customer-onboarding-verification-for-indias-leading-e-commerce-company/>
13. The Amazon Buy Box Algorithm: Explained - Brandwoven, Empowering Commerce, accessed on January 9, 2026, <https://gobrandwoven.com/resources/articles/the-amazon-buy-box-algorithm-explained-how-to-win-the-buy-box-on-amazon/>
14. What Is the Amazon Buy Box Algorithm? (And How Do You Win It?) - WebFX, accessed on January 9, 2026, <https://www.webfx.com/blog/marketing/amazon-buy-box-algorithm/>
15. Quarkus hexagonal architecture example - GitHub, accessed on January 9, 2026, <https://github.com/fredofm/quarkus-hexagonal>
16. Ports & Adapters architecture on example | by Wojciech Krzywiec - Medium, accessed on January 9, 2026, <https://wkrzywiec.medium.com/ports-adapters-architecture-on-example-19cab9e93be7>
17. Use Hibernate Search in Standalone mode with Elasticsearch/OpenSearch - Quarkus, accessed on January 9, 2026, <https://quarkus.io/guides/hibernate-search-standalone-elasticsearch>
18. Narayana LRA Participant Support - Quarkus, accessed on January 9, 2026, <https://quarkus.io/guides/lra>
19. Building Microservices with Quarkus and Kotlin | by Kenneth Kogi - Medium, accessed on January 9, 2026, <https://medium.com/@kogii/building-microservices-with-quarkus-and-kotlin-27af0cae2fd5>
20. Hibernate Reactive and Quarkus - Baeldung, accessed on January 9, 2026, <https://www.baeldung.com/java-hibernate-reactive-and-quarkus>
21. Mutiny - Async for mere mortals - Quarkus, accessed on January 9, 2026, <https://quarkus.io/guides/mutiny-primer>
22. Dynamic Routing in Serverless Microservice with Vert.x Event Bus | Blog, accessed on January 9, 2026, <https://vertx.io/blog/dynamic-routing-in-serverless-microservice-with-vert-x-event-bus/>
23. Simplified Hibernate Reactive with Panache - Quarkus, accessed on January 9, 2026, <https://quarkus.io/guides/hibernate-reactive-panache>
24. Using Hibernate Reactive - Quarkus, accessed on January 9, 2026, <https://quarkus.io/guides/hibernate-reactive>
25. Use Hibernate Search with Hibernate ORM and Elasticsearch/OpenSearch - Quarkus, accessed on January 9, 2026, <https://quarkus.io/guides/hibernate-search-orm-elasticsearch>
26. PostgreSQL: Bulk loading huge amounts of data, accessed on January 9, 2026, <https://www.cybertec-postgresql.com/en/postgresql-bulk-loading-huge-amounts-of-data/>
27. A Simple Quarkus App with Kafka and SmallRye Mutiny | by Praveen Ray - Medium, accessed on January 9, 2026, <https://raypvn.medium.com/a-simple-quarkus-app-with-kafka-and-smallrye-mutiny-97d941140a4b>
28. PostgreSQL JSONB vs. EAV: Which is Better for Storing Dynamic Data? - Raz Samuel Blog -, accessed on January 9, 2026, <https://www.razsamuel.com/postgresql-jsonb-vs-eav-dynamic-data/>
29. Replacing EAV with JSONB in PostgreSQL, accessed on January 9, 2026, <https://coussej.github.io/2016/01/14/Replacing-EAV-with-JSONB-in-PostgreSQL/>
30. Storing Dynamic Attributes - Sparse Columns, EAV, and JSONB Explained | Leapcell, accessed on January 9, 2026, <https://leapcell.io/blog/storing-dynamic-attributes-sparse-columns-eav-and-jsonb-explained>
31. Starting an Online Business for Dummies, 4th Edition (2005).pdf - The Swiss Bay, accessed on January 9, 2026, <https://theswissbay.ch/pdf/Gentoomen%20Library/Misc/Starting%20an%20Online%20Business%20for%20Dummies%2C%204th%20Edition%20%282005%29.pdf>
32. Difficulty Adding a Product to an Existing Listing - Amazon Seller Central, accessed on January 9, 2026, <https://sellercentral.amazon.com/seller-forums/discussions/t/80a08a23234a5b4542b7493d0675c975>
33. From Chaos to Conversion: Resolving Retail Contribution with Multiple Sellers on Amazon, accessed on January 9, 2026, <https://www.bellavix.com/from-chaos-to-conversion-resolving-retail-contribution-with-multiple-sellers-on-amazon/>
34. Contribution Score and Detail Page Control for Brand Owners Using a Reseller Model, accessed on January 9, 2026, <https://sellercentral.amazon.com/seller-forums/discussions/t/5db6cc8d-029e-420c-91c6-78d724db3aa5>
35. CONFLICT THE PRODUCT - Amazon Seller Central, accessed on January 9, 2026, <https://sellercentral.amazon.com/seller-forums/discussions/t/cfb8e487-dee1-4291-a06b-bf58d0604779>
36. GST Verification API: Use Cases and Benefits - Ekychub, accessed on January 9, 2026, <https://www.ekychub.in/gst-verification-api>
37. Top 10 PAN Card Verification API Providers In India [2024] - Perfios, accessed on January 9, 2026, <https://perfios.ai/resources/blogs/top-10-pan-verification-api-providers-in-india/>
38. Bank Account Verification API - Penny Drop - HyperVerge, accessed on January 9, 2026, <https://hyperverge.co/in/integrations-marketplace/bank-account-verification-api/>
39. Necessity of Bank Account Verification: Stop Fraud & Simplify KYC Processes - Zoop.one, accessed on January 9, 2026, <https://www.zoop.one/blog/methods-bank-account-verification>
40. Simple Ways to Stop Drop-offs During Bank Account Verification - Protean, accessed on January 9, 2026, <https://www.proteantech.in/articles/bank-account-verification-02-06-2025/>
41. Presign requests - AWS SDK for Kotlin, accessed on January 9, 2026, <https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/presign-requests.html>
42. Amazon S3 Client - Quarkiverse Documentation, accessed on January 9, 2026, <https://docs.quarkiverse.io/quarkus-amazon-services/dev/amazon-s3.html>
43. Upload to S3 Asynchronously using Reactive Programming | by Keshav Rathinavel, accessed on January 9, 2026, <https://medium.com/@keshavrathinavel/upload-to-s3-asynchronously-using-reactive-programming-d170bcd477f7>
44. Uploading Files With Quarkus (Update) - YouTube, accessed on January 9, 2026, <https://www.youtube.com/watch?v=mU74XKmUMbw>
45. Has changed the way Vertx eventBus multithreading is handled between Quarkus 1.13.7 and Quarkus 2.0.0? - Stack Overflow, accessed on January 9, 2026, <https://stackoverflow.com/questions/68244809/has-changed-the-way-vertx-eventbus-multithreading-is-handled-between-quarkus-1-1>
46. Multi-Vendor Marketplace Analytics: KPIs, Reporting & Performance Optimization - Flxpoint, accessed on January 9, 2026, <https://flxpoint.com/blog/multi-vendor-marketplace-analytics>
47. Flipkart Dispute Resolution: How to Handle Disputes and Protect Your Seller Account, accessed on January 9, 2026, <https://infobeamsolution.com/flipkart-dispute-resolution-how-to-handle-disputes-and-protect-your-seller-account/>
48. Bringing Drools rules into the cloud with Kogito: a step by step path - KIE Community, accessed on January 9, 2026, <https://blog.kie.org/2021/09/bringing-drools-rules-into-the-cloud-with-kogito-a-step-by-step-path.html>
49. Defining and executing business rules with Drools - Quarkus, accessed on January 9, 2026, <https://quarkus.io/guides/drools>
50. Top 10 ecommerce KPIs for online merchants - Checkout.com, accessed on January 9, 2026, <https://www.checkout.com/blog/top-10-kpis-for-ecommerces>
51. Marketplace KPIs you need to follow for your platform, accessed on January 9, 2026, <https://origami-marketplace.com/en-gb/multi-vendor-marketplace-kpis/>
52. The Amazon BuyBox: The 14 Most Important Criteria [Guide 2025] - SELLERLOGIC, accessed on January 9, 2026, <https://www.sellerlogic.com/en/blog/the-14-key-criteria-for-winning-the-buy-box-on-amazon-and-how-to-keep-your-metrics-in-check/>
53. Amazon Buy Box 2025: Proven Strategies to Win and Maximize Sales - Seller Labs, accessed on January 9, 2026, <https://www.sellerlabs.com/blog/amazon-buy-box-2025-strategies-to-win/>
54. Structuring Multi-Project Builds - Gradle User Manual, accessed on January 9, 2026, <https://docs.gradle.org/current/userguide/multi_project_builds_intermediate.html>
55. Effective Project Structuring for Microservices with Quarkus - DEV Community, accessed on January 9, 2026, <https://dev.to/yanev/effective-project-structuring-for-microservices-with-quarkus-1lf0>
56. Simplified Hibernate ORM with Panache - Quarkus, accessed on January 9, 2026, <https://quarkus.io/guides/hibernate-orm-panache>
57. Hibernate Concurrent Update and Quarkus Panache - GitHub, accessed on January 9, 2026, <https://github.com/manoelcampos/hibernate-concurrency>
58. Optimistic vs. Pessimistic locking - Stack Overflow, accessed on January 9, 2026, <https://stackoverflow.com/questions/129329/optimistic-vs-pessimistic-locking>
59. A SAGA demo with Quarkus and Long Running Action - GitHub, accessed on January 9, 2026, <https://github.com/jbcodeforce/saga-lra-quarkus>