Auto Grocery Infrastructure

Pricing Microservice Specification

Version: 1.2.0
Status: Production Ready
Protocol: gRPC / Protobuf

Architecture Note

This service acts as the **Financial Source of Truth**. It calculates real-time market value for inventory items by correlating warehouse supply metrics with profit margin requirements.

Contents

1. Service Overview & Responsibility

The Pricing Microservice is a high-performance Go-based utility that manages the global product catalog. Unlike traditional static pricing engines, this service implements a **Dynamic Market Feedback Loop**. It does not merely store prices; it actively recalculates them based on the real-time velocity of stock in the warehouse.

Key Responsibilities

- **Price Discovery:** Dynamically adjusting unit prices based on supply metrics.
- **Billing Manifests:** Generating atomic price snapshots for checkout orders.
- **Catalog Persistence:** Maintaining SKU-to-Price mappings in a relational database.

2. Technical Architecture & Network Map

The service is built on the gRPC framework for low-latency communication between the Ordering and Inventory services.

Connectivity Specs

- **Internal Listening Port:** `:50052` (gRPC over HTTP/2).
- **Upstream Dependency:** `InventoryService` (`:50051`).
- **Database Engine:** PostgreSQL 15+.

Authentication & Security

In the current implementation, this service resides within the **Internal Trusted Zone**.

- **gRPC Security:** Currently utilizes insecure transport for local development. For production, the system is designed to implement **mTLS (Mutual TLS)** via certificates.
- **Request Validation:** Every method call performs a "Sanity Check" on SKUs to prevent SQL injection or malformed strings from affecting the pricing logic.

3. Database Schema: The Catalog Engine

The persistence layer is managed via `internal/store/catalog.go`. The schema is optimized for high-read throughput.

The `catalog` Table

Column	Data Type	Constraint
`id`	SERIAL	PRIMARY KEY
`sku`	VARCHAR(255)	UNIQUE, INDEXED
`unit_price`	NUMERIC(12,2)	NOT NULL, DEFAULT 0.00

4. Detailed Method Logic

`GetPrice(sku)`

When this method is called by the **Ordering Service**:

1. The handler initiates a `SELECT` query filtered by the unique SKU.

2. If the SKU is missing, it returns a gRPC `NotFound` error code.
3. If found, it returns the atomic price currently locked in the database.

`CalculateBill(order_items)`

This is the most critical method for the checkout process:

1. **Batch Retrieval:** Instead of querying one by one, the service uses an `IN` clause to fetch all relevant SKU prices in a single DB trip.
2. **Arithmetic Loop:** It iterates through the requested quantities, multiplying them by the retrieved unit prices.
3. **Grand Total:** It calculates the sum and returns a detailed `BillManifest`, which includes per-item line totals and the final payable amount.

5. Background Logic: The Dynamic Re-Pricer

The "Intelligence" of the service lies in `internal/logic/pricer.go`. This background worker ensures that "Smart Pricing" is always active without manual intervention.

Hourly Sync Process

Every 60 minutes, the service executes the following:

1. **gRPC Uplink:** It calls `Inventory.GetStockMetrics()`.
2. **Data Ingestion:** It receives the current `Quantity` and `CostPrice` for every item in the warehouse.
3. **The Margin Algorithm:**

```
if quantity < 10 {
    margin = 1.35 // Scarcity Price (35% profit)
} else if quantity > 500 {
    margin = 1.05 // Liquidation Price (5% profit)
} else {
    margin = 1.20 // Standard Market Price (20% profit)
}
newPrice = costPrice * margin
```

4. **Atomic Update:** The service performs a `Bulk Update` to the catalog so that the new prices are available immediately for the next customer.

6. Error Handling Strategy

The service implements standard gRPC error codes to ensure the **Ordering Service** can handle failures gracefully:

- `Codes.Unavailable`: Returned if the PostgreSQL database is unreachable.
- `Codes.InvalidArgument`: Returned if a billing request contains a negative quantity.
- `Codes.Internal`: Returned if the connection to the *Inventory Service* fails during a re-pricing cycle.