Smart‑Inventory & Dispatch Manager – Technical Documentation  
  
(By : Voonna Gowri Ganesh )

# 1. Project Overview

Smart‑Inventory & Dispatch Manager is an end‑to‑end, async FastAPI + MongoDB (Motor) solution for managing inventory, hubs, dispatch flows, drivers, and vehicles. It focuses on batch‑wise stock tracking, FIFO‑based dispatch, auditable stock transactions, and clean separation between service layer and data layer. The system intentionally avoids paid map APIs at this stage; mapping/route‑planning is positioned as a future enhancement using open‑source options (OpenStreetMap, OSRM, Leaflet).

# 2. Technology Stack & State

* Language: Python 3.x (async/await)
* Web Framework: FastAPI (ASGI)
* App Server: Uvicorn
* Database: MongoDB (Motor async driver)
* Models/Validation: Pydantic
* Logging: Python logging via custom get\_logger
* Packaging/Runtime: Docker‑ready (optional)
* Time: All timestamps in UTC
* No paid/third‑party map APIs used in current build (future: OSM/OSRM/Leaflet)

# 3. Objectives

* Enable robust, auditable stock management at batch level.
* Support FIFO consumption during dispatch to minimize expiries.
* Provide clean, predictable APIs with strong validation and error semantics.
* Maintain hub lifecycle (active/closed) and associated governance.
* Lay foundation for operational dashboards, notifications, and analytics.

# 4. Features Implemented

## 4.1 Hub Management

* Register, update, delete (archive into ClosedHubs), search, list by status, view closed hubs.

## 4.2 Inventory Management

* Register product + initial batch, update product/batches, FIFO dispatch between hubs.
* Batch‑level quantities, purchase value, and unit cost tracking.
* StockTransactions audit log for every IN/OUT action.
* Helpers for expiry warnings (< 30 days).

## 4.3 Queries & Reporting (Initial)

* Product summary per hub (total quantity, nearest expiry, batches count).
* List batches for product in hub; list products with search/pagination.

## 4.4 Vehicle & Driver (per endpoints plan)

* Vehicle/Driver registration, status updates, soft‑deletes, retirement logic (drivers > 50).
* Dispatch allocation logic (driver+vehicle → dispatch in‑progress).

# 5. System Architecture

Layered, async architecture:

* API Layer: FastAPI routers (not shown here) expose endpoints.
* Service Layer: HubService, InventoryService encapsulate business logic and validations.
* Data Layer: Motor client to MongoDB; strong indexes recommended for hot paths.
* Models: Pydantic request/response schemas.
* Utilities: helpers for normalization, time, IDs, and logging.

High‑Level Flow (textual):

Client → FastAPI Router → Service Layer  
 → MongoDB (Collections: Hubs, ClosedHubs, InventoryProducts, InventoryBatches, StockTransactions, Dispatches, ...)  
 → Responses (JSON) with consistent status + messages  
 → Logging at INFO/ERROR with contextual identifiers

# 6. Data Models (Collections)

## 6.1 Hubs

Hubs:  
 hub\_id: str (unique, uppercase)  
 hub\_name: str (unique)  
 hub\_manager: str (unique among active hubs)  
 hub\_phone\_number: str  
 hub\_address: str  
 status: "Active" | "Deactive"  
 hub\_opening\_date: datetime(UTC)  
 created\_at: datetime(UTC)  
 updated\_at: datetime(UTC|null)  
  
ClosedHubs:  
 (copy of Hub fields) +  
 hub\_closed\_date: datetime(UTC)  
 no\_of\_days\_active: int  
 deleted\_by: str|null  
 reason: str|null  
 status: "Closed"

## 6.2 Inventory

InventoryProducts (Product Master):  
 Product\_ID: str (unique, normalized)  
 Product\_Name: str  
 Category: str  
 Brand: str|null  
 Selling\_Price: float  
 Product\_Description: str|null  
 created\_at: datetime(UTC)  
 updated\_at: datetime(UTC)  
  
InventoryBatches (Per Hub + Product + Batch):  
 Product\_ID: str  
 Hub\_ID: str  
 Batch\_No: str (unique per Product\_ID+Hub\_ID)  
 Quantity: int  
 Expiry\_Date: datetime(UTC)  
 Purchase\_Value: float  
 Purchase\_Unit\_Price: float  
 status: "active" | "depleted" | "archived"  
 created\_at: datetime(UTC)  
 last\_updated: datetime(UTC)  
  
StockTransactions (Immutable audit):  
 transaction\_id: str  
 type: "IN" | "OUT" | "ADJUSTMENT" | "ARCHIVE"  
 Product\_ID: str  
 Hub\_ID: str  
 Batch\_No: str|null  
 Quantity: int  
 Unit\_Price: float|null  
 Total\_Value: float|null  
 reference: str|null  
 timestamp: datetime(UTC)  
 remarks: str  
  
Dispatches:  
 dispatch\_id: str  
 Product\_ID: str  
 From\_Hub\_ID: str  
 To\_Hub\_ID: str  
 Quantity: int  
 Batch\_Consumption: [ { Batch\_No, Qty, Unit\_Cost } ]  
 Vehicle\_Assigned: str|null  
 Driver\_Assigned: str|null  
 Timestamp: datetime(UTC)  
 Status: "In-Progress" | "In-Transit" | "Completed" | "Cancelled"  
 Request\_Ref: str|null  
 Notes: str|null

## 6.3 Vehicles & Drivers

Vehicles:  
 Vehicle\_ID: str (unique)  
 Vehicle\_Number: str (unique)  
 Capacity: int  
 Status: "Available" | "Unavailable" | "In-Transit" | "Under-Maintenance"  
 created\_at, updated\_at: datetime(UTC)  
  
ClosedVehicles (archived):  
 (copy of fields) + Closed\_Date: datetime(UTC), reason: str|null  
  
Drivers:  
 Driver\_ID: str (unique)  
 Driver\_Name: str  
 Age: int (<= 50 on registration)  
 License\_No: str (unique)  
 Contact\_Number: str|null  
 Address: str|null  
 Salary: float|null  
 Status: "Available" | "Assigned" | "On-Leave" | "Inactive"  
 Hub\_ID: str|null  
 Registered\_At: datetime(UTC)  
  
RetiredDrivers (archived/retired):  
 (copy of fields) + Retired\_At: datetime(UTC), retirement\_reason: str

# 7. Error Handling & Logging

* Validation errors → raise ValueError or custom exceptions; map to HTTP 400.
* Resource missing (e.g., hub not found) → raise LookupError/404.
* Conflict (duplicates, insufficient stock) → raise 409.
* Database errors (DuplicateKeyError) are caught and converted to user‑friendly messages.
* Consistent logging via get\_logger: INFO for normal ops, ERROR for failures, with contextual IDs (hub\_id, product\_id, batch\_no, dispatch\_id).

Examples:  
- On hub create, check for manager conflicts and duplicate keys; log at INFO/ERROR.  
- On inventory register/update, log master/batch creation or merge; log transaction IDs.  
- On dispatch, log quantities, from/to hubs, batch consumption details; log remaining stock.

# 8. Helper Methods (Utilities)

* \_now\_utc(): timezone‑aware current UTC datetime
* \_to\_utc\_datetime\_from\_date(d): converts YYYY‑MM‑DD to UTC datetime
* \_normalize\_id(s): trims and uppercases identifiers
* \_gen\_transaction\_id(): unique stock transaction IDs
* \_gen\_dispatch\_id(): unique dispatch IDs
* Hub helpers: \_normalize\_hub\_id(), \_normalize\_hub\_name()
* DB guard: \_ensure\_db(), \_ensure\_hub\_exists(), \_get\_product\_master(), \_get\_total\_available()

# 9. API Endpoints (as implemented in endpoints.txt)

## 9.1 Hub Management

### 9.1.1 Register Hub

**Method:** POST

**Path:** /hubs/register

Request Body (JSON):

{  
 "hub\_id": "string", // unique  
 "hub\_name": "string",  
 "hub\_manager": "string",  
 "hub\_phone\_number": "string",  
 "hub\_address": "string"  
}

Conditions:

* hub\_id and hub\_name unique; one active hub per hub\_manager.
* Server sets status='Active', hub\_opening\_date=now(UTC).

Response (201):

{  
 "message": "Hub created successfully",  
 "hub\_id": "string",  
 "hub\_name": "string"  
}

### 9.1.2 Update Hub

**Method:** PUT

**Path:** /hubs/update

Request Body (JSON):

{  
 "hub\_id": "string", // not editable, used to locate record  
 "hub\_name": "string (optional)",  
 "hub\_manager": "string (optional)",  
 "hub\_phone\_number": "string (optional)",  
 "hub\_address": "string (optional)",  
 "status": "Active/Deactive (optional)"  
}

Conditions:

* At least one updatable field must be provided.
* status ∈ {Active, Deactive}.

Response (200):

{  
 "message": "Hub details updated successfully",  
 "hub\_id": "string"  
}

### 9.1.3 Delete (Archive) Hub

**Method:** DELETE

**Path:** /hubs/delete

Request Body (JSON):

{  
 "hub\_id": "string",  
 "hub\_name": "string",  
 "hub\_manager": "string (optional)",  
 "deleted\_by": "string (optional)",  
 "reason": "string (optional)"  
}

Behavior:

* Move to ClosedHubs with hub\_closed\_date and no\_of\_days\_active; remove from Hubs.

Response (200):

{  
 "message": "Hub deleted successfully",  
 "hub\_id": "string",  
 "hub\_name": "string"  
}

### 9.1.4 Search Hubs

**Method:** GET

**Path:** /hubs/search

Query Params: hub\_id (optional), hub\_name (optional), skip, limit

Response: paginated list of matching hubs.

### 9.1.5 List Closed Hubs

**Method:** GET

**Path:** /hubs/closed

Response: list of closed hub documents (archived).

### 9.1.6 List Hubs by Status

**Method:** GET

**Path:** /hubs/by-status

Query Param: status=Active|Deactive.

## 9.2 Inventory Management

### 9.2.1 Register Inventory

**Method:** POST

**Path:** /inventory/register

Request Body (JSON):

{  
 "Hub\_ID": "string",  
 "Product\_ID": "string",  
 "Product\_Name": "string",  
 "Quantity": 100,  
 "Value": 5000.0,  
 "Selling\_Price": 60.0,  
 "Product\_Description": "string (optional)",  
 "Expiry\_Date": "YYYY-MM-DD",  
 "Brand": "string",  
 "Batch\_No": "string (optional)"  
}

Behavior: creates/updates product master; creates new batch or merges by exact expiry; logs StockTransactions IN; returns expiry warning if < 30 days.

Response (201): includes hub info, product\_id, batch\_no, quantity added, optional warning.

### 9.2.2 Update Inventory

**Method:** PUT

**Path:** /inventory/update

Behavior: updates master fields and adds stock into an existing batch (by Batch\_No or Expiry match) or creates new batch; logs StockTransactions IN; optional expiry warning.

Response (200): product\_id, batch\_no, merged/new flags, new batch quantity, optional warning.

### 9.2.3 Delete Inventory (Soft Archive)

**Method:** DELETE

**Paths:** /inventory/batch/delete, /inventory/product/delete

Behavior: soft‑archive batch/product; prevent deleting non‑empty batches unless forced; log StockTransactions ARCHIVE.

### 9.2.4 Dispatch Inventory

**Method:** POST

**Path:** /inventory/dispatch

Behavior: validate hubs and availability; consume batches FIFO by Expiry\_Date; decrement quantities; mark depleted when zero; create Dispatches record; write StockTransactions OUT per consumed batch; return remaining quantity.

Response (201): dispatch\_id, batch consumption breakdown, remaining quantity at source.

### 9.2.5 Search Inventory

**Method:** GET

**Path:** /inventory/search

Query: Product\_ID, Product\_Name, Hub\_ID, Category, Brand, skip, limit. Response: aggregated product view with totals and nearest expiry.

### 9.2.6 View Batches for a Product in a Hub

**Method:** GET

**Path:** /inventory/{Hub\_ID}/{Product\_ID}/batches

### 9.2.7 Low Stock Alerts

**Method:** GET

**Path:** /inventory/low-stock

### 9.2.8 Expiry Alerts

**Method:** GET

**Path:** /inventory/expiring

### 9.2.9 Stock Transactions Log

**Method:** GET

**Path:** /inventory/transactions

### 9.2.10 Reports

**Summary:** /inventory/report/summary

**Movement:** /inventory/report/movement

## 9.3 Vehicle Management

### 9.3.1 Add Vehicle (POST)

### 9.3.2 Update Vehicle (PUT)

### 9.3.3 Delete Vehicle (DELETE)

### 9.3.4 Search Vehicle (GET)

### 9.3.5 Dispatch Vehicle Allocation (GET)

## 9.4 Driver Management

### 9.4.1 Add Driver (POST)

### 9.4.2 Update Driver (PUT)

### 9.4.3 Delete Driver (DELETE)

### 9.4.4 Search Driver (GET)

### 9.4.5 Retirement Audit (POST)

# 10. Error Semantics (HTTP)

* 400 Bad Request – validation failures (e.g., Quantity <= 0, invalid date format)
* 404 Not Found – missing Hub/Product/Batch/Driver/Vehicle
* 409 Conflict – duplicate keys, insufficient stock for dispatch
* 500 Internal Server Error – unhandled exceptions

# 11. Future Enhancements

* OpenStreetMap/OSRM/Leaflet integration for routing (no paid APIs).
* Real‑time inventory updates via WebSocket/Kafka; alerting on thresholds.
* AI/ML‑based demand forecasting + auto‑replenishment.
* Barcode/QR for batches; scan‑based IN/OUT.
* IoT sensor (temp/humidity) hooks for perishable goods.
* RBAC and audit trails across modules.
* Email/SMS/WhatsApp notifications for expiries, low stock, dispatch confirmations.
* Centralized multi‑hub dashboard; heatmaps.
* Blockchain‑based immutable dispatch ledger (regulatory industries).
* ERP/SAP integration; mobile apps for field operations.
* Conversational chatbot for quick insights.

# 12. Conclusion

This document consolidates the scope and implementation of Smart‑Inventory & Dispatch Manager: a practical, async, MongoDB‑backed platform for inventory and dispatch operations. It captures the current endpoints, data models, and architectural principles while outlining a clear path for growth with routing, real‑time updates, analytics, and ML.