Smart Pharmaceutical Inventory System – Development Plan

Step 0: Set Up Dev Environment

- Install Node.js + npm (for inventory backend & streaming backend)
- Install MongoDB locally
- Install Python 3.11+ + pip (for Raspberry Pi backend)
- Install OpenCV + FastAPI in Python virtualenv
- Set up React + TypeScript project (or Angular if preferred)
- Install Docker if you want containerized services

Step 1: Inventory & POS Backend (Node.js/TS)

Goal: MVP backend that can manage medicines, batches, and POS logic

Tasks: 1. Initialize Node.js + TS project 2. Connect to MongoDB 3. Implement Medicine CRUD API (name, batch, qty, expiry, fridgeID, temp) 4. Implement POS API (scan barcode → select batch using LIFO → decrement stock) 5. Implement Alerts API (dummy for now, will connect to sensors later) 6. Test endpoints with Postman

Iteration: Fully working backend without sensors. Test with dummy data.

Step 2: Frontend Dashboard MVP

Goal: Display inventory and POS interface

Tasks: 1. Set up React project + Tailwind / Material UI 2. Create Inventory Table page (list medicines, batches, qty) 3. Create POS page (scan barcode input, decrement stock) 4. Hook frontend to backend APIs 5. Test basic CRUD & POS flows

Iteration: Working standalone inventory & POS system.

Step 3: Raspberry Pi Edge Backend (Python/FastAPI)

Goal: Read sensors and send data to backend

Tasks: 1. Connect DS18B20 temperature sensor to Pi, read temp in Python 2. Write FastAPI app exposing POST /sensor-data → sends temp to backend 3. Simulate sending to Node.js backend first 4. Optional: connect Pi camera and capture frames

Iteration: Pi backend can read sensors and POST dummy data to cloud backend.

Step 4: Device Onboarding / Pairing

Goal: Associate Pi with a user and fridge

Tasks: 1. Define device ID + token system 2. Implement backend API to pair device with user 3. Test Python Pi script to authenticate with backend and store user/fridge config

---

Step 5: Integrate Sensor Data into Dashboard

Goal: Show live fridge temp and alerts

Tasks: 1. Backend receives Pi sensor POSTs and stores in DB 2. Frontend fetches fridge temp via API 3. Implement threshold check → trigger alert if temp > lowest temp requirement 4. Display alerts on dashboard

Iteration: MVP working fridge monitoring.

Step 6: Live Streaming

Goal: Show camera feed from Pi on dashboard

Tasks: 1. Python backend exposes MJPEG or WebSocket video stream 2. Node.js streaming backend fetches frames from Pi 3. Frontend displays video feed 4. Optional: small snapshots every X seconds for backup

---

Step 7: Optional Computer Vision

Goal: Count vials and track stock visually

Tasks: 1. Start with OpenCV grid-based detection (12x12 slots) 2. Count occupied vs empty slots 3. Send counts to backend 4. Compare with DB stock → alert if mismatch

---

Step 8: Optional Motion Detection

Goal: Trigger alert if unauthorized movement

Tasks: 1. Use frame-difference algorithm on Pi camera 2. If movement detected → send alert to backend 3. Optional: save snapshot or video snippet

---

Step 9: Connect Everything

- Inventory & POS backend + Pi edge backend + streaming backend + frontend dashboard
- Test full workflow:
- New stock added
- Fridge temp monitored
- Alerts triggered
- Camera feed displayed
- CV counts updated
- POS works as expected

Step 10: Iterative Improvements

- Add CV model for more accurate vial detection (TensorFlow Lite / YOLOv5 Nano)
- Add motion detection history
- Improve frontend charts (Recharts or Chart.js)
- Fine-tune threshold alerts
- Optionally add predictive analytics for stock depletion

Key Advice

- Build iteratively: Inventory → Sensors → Streaming → CV → Motion
- Test each module independently first
- Focus on MVP first (inventory + temp monitoring + alerts) before adding CV/motion
- Keep Pi and backend lightweight — avoid overloading the Pi with heavy ML models initially
- Use version control (Git) to track progress and maintain backups
- Document each step for easier debugging and future reference