StockMaster MVP – Complete Step-by-Step Build Guide for Cursor

0. MVP Scope (Defined from Hackathon Problem Statement)

This MVP focuses on implementing only the core must-have features from the hackathon problem statement while keeping the system small, efficient, and quickly buildable using Cursor AI + Node.js/React stack.

Included Features:

- Authentication (Signup, Login, OTP-based Reset)
- Dashboard KPIs (Total stock, low stock, pending operations)
- Product & Warehouse Management
- Inventory Operations: Receipts, Deliveries, Internal Transfers, Stock Adjustments
- Stock Movement Ledger (History)
- Dynamic filtering for operations

1. Project Setup Instructions (Cursor-Oriented)

1.1 Folder Structure

Use a clean 2-folder architecture:

```
backend/
frontend/
```

1.2 Global Cursor System Prompt

Paste the following into Cursor's **Architect/System Prompt** so all code generated follows your architecture:

You are helping build an MVP web app called "StockMaster" for a hackathon.
Tech stack: Node.js + Express backend, MongoDB (Mongoose ORM), React frontend.
Core modules:
- Auth (Signup/Login/OTP Reset)
- Dashboard KPIs
- Products, Warehouses
- Operations: Receipts, Deliveries, Transfers, Adjustments
- Stock Movement Ledger

Rules:
- Clean modular folder structure
- Use controllers, services, routes

- Use JWT auth
- OTP should be mocked (code logged in console)
- On validation of operations, update StockLevel and create StockMovement records
- Avoid unnecessary complexity (like microservices)
- Generate TypeScript code where possible

2. Data Model Specification (Use in Cursor to Generate Schemas)

These models are optimized for fast development, low complexity, and easy aggregation for dashboard KPIs.

2.1 User Model

- name
- email
- passwordHash
- role (manager/staff)
- otpCode
- otpExpiry
- createdAt / updatedAt

2.2 Warehouse Model

- name
- code
- type (warehouse/rack)
- address (optional)

2.3 Product Model

- name
- sku
- category
- unitOfMeasure
- reorderLevel
- active

2.4 StockLevel Model

- productId
- warehouseId
- quantity

2.5 Operation Model

Fields:
- type: RECEIPT / DELIVERY / TRANSFER / ADJUSTMENT
- status: DRAFT / WAITING / READY / DONE / CANCELLED

- sourceWarehouse
- destWarehouse
- party (supplier/customer)
- scheduledDate (for transfers)
- validatedAt
- createdBy
- createdAt

2.6 OperationLine Model

- operationId
- productId
- quantity
- unitPrice (optional)

2.7 StockMovement Model

- productId
- operationId
- quantityChange (+/-)
- fromWarehouseId
- toWarehouseId
- movementType
- createdAt

3. Backend API Plan (Cursor-Ready Implementation Steps)

3.1 Authentication APIs

Endpoints:
POST /api/auth/signup
POST /api/auth/login
POST /api/auth/request-otp
POST /api/auth/reset-password
GET /api/auth/me

Implementation Rules:
- Use JWT
- OTP: generate 6-digit code, store in user table, log to console
- Reset password using otpCode + otpExpiry

3.2 Products & Warehouses APIs

Products:
POST /api/products
GET /api/products

PUT /api/products/:id

Warehouses:
POST /api/warehouses
GET /api/warehouses

3.3 Operations API (Core Inventory Logic)

POST /api/operations
GET /api/operations
GET /api/operations/:id
PUT /api/operations/:id
POST /api/operations/:id/validate

Validation Logic:

RECEIPT → Increase stock
DELIVERY → Decrease stock
TRANSFER → Decrease source + increase destination
ADJUSTMENT → Set counted quantity, log difference

3.4 Dashboard Summary Endpoint

GET /api/dashboard/summary
Returns:
- totalProductsInStock
- lowStockCount
- outOfStockCount
- pendingReceiptsCount
- pendingDeliveriesCount
- scheduledTransfersCount

3.5 Stock Movement History API

GET /api/stock-movements

4. Frontend Pages (React)

4.1 Auth Pages

- Login
- Signup
- Forgot Password
- Reset Password

4.2 Dashboard Page

Displays KPIs + Recent Operations.

4.3 Products Page

Table with: name, sku, total stock, reorder status.

4.4 Operations List Page

Filters: type, status.
Shows all inventory documents.

4.5 Operation Form Page

Dynamic UI based on operation type (receipt/delivery/transfer/etc.)

4.6 Stock Ledger Page

Table showing all StockMovement entries.

5. Demo Scenario for Hackathon Judges

1. Create Product 'Steel Rods'
2. Create Warehouses
3. Receive 100 units → Validate
4. Transfer 60 units to another warehouse
5. Deliver 20 units to customer
6. Adjust 3 units damaged

6. Stretch Goals (If Time Allows)

- Low stock alerts
- Role-based permissions
- CSV Export
- Mobile Responsive UI