flow

📦 Full Inventory Management System Explained (Beginner-Friendly)

🧱 1. Folder Structure Overview (Backend + Frontend)

🖥️ Backend (`/backend`)

- **controllers/** → Handles logic (e.g., creating PO, reducing stock).
- **models/** → Defines data structure (MongoDB collections).
- **routes/** → Connects URLs (like `/api/items`) to controller functions.
- **middleware/** → Manages authentication and authorization.
- **utils/** → Shared helper functions (like token handling, stock update).
- **server.js** → Main entry point to run Express app.
- **config/db.js** → MongoDB connection file.

🌐 Frontend (`/frontend/src`)

- **components/layout/** → Layout, Sidebar, Dropdowns for navigation.
- **pages/** → All forms and lists (like AddItemForm, ClientList).
- **main.jsx, App.jsx** → Main React components bootstrapping the app.
- **index.css, tailwind.config.js** → Styles using Tailwind CSS.

🔁 2. Full Workflow Summary: Real-World Flow

```plaintext
CopyEdit
Purchase Order → Goods Receipt → Stock IN
Delivery Challan → Sales Invoice → Stock OUT
Sales Return / Purchase Return → Stock Adjusted
Manual Stock Adjustment → Manual Corrections
```

📌 3. How Components Link Together

🔐 Authentication & Roles

- **Login Page (`Login.jsx`)**: Allows login via email/password.

- JWT token stored in **localStorage** after login.
- Token sent in headers to access **protected APIs** on backend.

📦 Item & Stock Management

`Item` Model

```
js
CopyEdit
{
  name, sku, unit, category, price, lowStockAlert, quantity
}
```

🔗 Frontend

- `AddItemForm.jsx`: Add or edit items.
- `StockList.jsx`: Displays item table with filter options.
- `Stock.js`: Backend model to track real-time quantity.
- **Stock updated** when:
  - Goods received (via PO)
  - Challan dispatched
  - Returns happen

📄 Purchase Flow

Step 1: Create Purchase Order

- `AddPOForm.jsx` → `purchaseOrderRoutes.js` → `purchaseOrderController.js`
- Vendor, Items, Rates, Tax added.
- **Does NOT increase stock yet.**

Step 2: Goods Receipt

- `AddGRNForm.jsx` → `goodsReceiptController.js`
- Linked to PO.
- **Increases stock** using `increaseStock(itemId, qty)` in utils.

🚚 Sales Flow

Step 1: Create Delivery Challan

- `AddDCForm.jsx` → `deliveryChallanRoutes.js` → `deliveryChallanController.js`

- Client, items, transporter data entered.
- **Decreases stock** using `decreaseStock()`.

Step 2: Generate Invoice

- `AddInvoiceForm.jsx` → linked to Delivery Challan
- Shows actual bill.
- **No stock update**, as it's already reduced in DC.

🔁 Returns (Both Sides)

Sales Return

- `AddSalesReturn.jsx` → `salesReturnRoutes.js`
- Refers to Sales Invoice
- **Increases stock back**

Purchase Return

- `AddPurchaseReturn.jsx` → `purchaseReturnRoutes.js`
- Refers to GRN or PO
- **Decreases stock back**

⚙️ Manual Stock Adjustment

- `AddStockAdjustment.jsx` → `stockAdjustments`
- For expired/lost/damaged items.
- Choose: +ve or -ve quantity
- **Directly modifies stock**

🧩 4. Shared References and Data Linking

✅ Shared Data Fields

- **Item ID** is the backbone:
  - Used in PO, GRN, DC, Invoice, Returns, Stock
- **Vendor ID** used in PO, GRN
- **Client ID** used in DC, Invoice, Sales Return

✅ How Forms Use References

- In `AddGRNForm.jsx`: select PO → fetch vendor/items
- In `AddInvoiceForm.jsx`: select DC → fetch client/items

📊 5. Reports & Dashboard (UI Logic)

Reports:

- `StockReport.jsx`, `SalesReport.jsx`, etc.
- Pull backend data using query params (like date, client, item).
- Show tables and allow CSV/PDF download.

Dashboard:

- Graphs using bar chart & pie chart libraries (e.g., Recharts)
- Shows:
  - Total Items
  - Total Stock
  - Low Stock Count
  - Charts: Stock by Item & Category
- Powered by summary API routes (`/dashboard/summary`) showing counts & trends

---

🔒 6. Role & Access Management

Users can be:

- **Admin**: Full Access
- **Manager**: Can manage inventory but not users
- **Viewer**: Read-only access

Protection:

- Middleware `authMiddleware.js` and `roleMiddleware.js`
- Each route in backend is protected with:

```
js
CopyEdit
router.post('/add-po', authMiddleware, roleMiddleware(['admin', 'manager']))
```

🔧 7. Helpful Utilities in `utils/`

- `increaseStock()` / `decreaseStock()` – Reusable for all modules
- `generateToken()` / `decodeToken()` – JWT handling
- `generateDocumentNumber()` – For PO/GRN/Invoice auto-numbering

📑 8. Suggested Database Structure

```bash
CopyEdit
collections/
├── users
├── roles
├── items
├── categories
├── units
├── vendors
├── clients
├── purchaseOrders
├── goodsReceipts
├── deliveryChallans
├── salesInvoices
├── returns
├── stockAdjustments
├── stock
├── stockLedger
```

Stock Ledger:

- New model to track each transaction:

```js
CopyEdit
{
  itemId,
  transactionType: 'IN' | 'OUT' | 'RETURN' | 'ADJUST',
  source: 'GRN' | 'DC' | 'Manual',
  qty, timestamp
}
```

🧠 Beginner Takeaways

Concept	What to Understand
Model	A structure for data (like "Item" or "Vendor")
Controller	Contains code to handle request logic
Route	URL that triggers a controller method
Component	A part of the frontend UI (form, table, dropdown)
Page	A full-screen view (like Add PO, Dashboard)

API Call	A request sent from frontend to backend
Token	Secret proof that a user is logged in
Role	What the user can or cannot do

🛠️ What You Can Improve Next

- ✅ Add PDF/Print buttons (using `react-to-print` or `jspdf`)
- ✅ Add document number generator
- ✅ Add filtering in reports (by client/date/item)
- ✅ Add `StockLedger.js` for transaction logs
- ✅ Improve mobile responsiveness
- ✅ Add toast notifications and spinners for form submissions
- ✅ Protect routes fully with role-based middleware

🧱 BACKEND STRUCTURE — SIMPLIFIED OVERVIEW

```
sql
CopyEdit
backend/
│
├── config/              --> DB connection (MongoDB URI setup)
│   └── db.js
│
├── controllers/         --> Logic: "what happens" when API is called
│   └── deliveryChallanController.js, etc.
│
├── middleware/          --> JWT Auth + Role checking
│   └── authMiddleware.js
│
├── models/              --> MongoDB Schema definitions (tables)
│   └── Client.js, Item.js, SalesInvoice.js, ...
│
├── routes/              --> URL routes like /api/items, /api/invoices
│   └── itemRoutes.js, salesInvoiceRoutes.js, ...
│
├── utils/               --> Shared helper logic
│   └── increaseStock(), decodeToken(), etc.
│
└── server.js            --> Main Express app
```

🔁 DATA FLOW IN REAL-LIFE OPERATIONS

Let's follow an example:

🔸 Scenario: You place a **Purchase Order** to a vendor

1. **Frontend Form**: `AddPOForm.jsx`
2. **Route Triggered**: `POST /api/purchase-orders`
3. **Route File**: `purchaseOrderRoutes.js`
4. **Controller Logic**: `purchaseOrderController.createPurchaseOrder()`
5. **Data Stored** in: `PurchaseOrder` collection (Mongoose model)

📦 At this stage: **Stock is NOT increased**

🔸 Scenario: Vendor sends goods → You create a **Goods Receipt**

1. Form: `AddGRNForm.jsx`
2. Backend Route: `POST /api/goods-receipts`
3. Controller: `goodsReceiptController.createGRN()`
4. Data:
   - **Link to PO** using `purchaseOrderId`
   - Items list is taken from PO
5. **Stock Updated**:
   - Calls `increaseStock(itemId, qty)`
   - Adds/updates quantity in `Stock.js` model
   - Logs into `StockLedger.js`

🔸 Scenario: You dispatch goods to customer using a **Delivery Challan**

1. Form: `AddDCForm.jsx`
2. Route: `POST /api/delivery-challans`
3. Controller: `deliveryChallanController.createDC()`
4. Stock Reduced:
   - Calls `decreaseStock(itemId, qty)`
   - Again logs in `StockLedger.js`

🔸 Scenario: You issue a **Sales Invoice**

1. Form: `AddInvoiceForm.jsx`
2. Route: `POST /api/sales-invoices`
3. Data saved with reference to `deliveryChallanId`

4. ✅ **No Stock Update here**, because DC already handled that

---

🔗 DATA RELATIONSHIPS (REFERENCES)

Each module **connects using MongoDB ObjectIDs** as reference.

Data Module	References	Description
PurchaseOrder	`vendorId`, `items[]`	Vendor + Item links
GoodsReceipt	`purchaseOrderId`, `items[]`	Links to PO and items
DeliveryChallan	`clientId`, `items[]`	Links to customer and stock
SalesInvoice	`deliveryChallanId`, `clientId`	Uses DC for item info
SalesReturn	`salesInvoiceId`, `items[]`	Returns from invoice
PurchaseReturn	`goodsReceiptId`, `items[]`	Returns against goods received
StockAdjustment	`itemId`, quantity, reason	Manual update
StockLedger	`itemId`, `sourceId`, type	Audit log of stock movement

🔍 Reference Example in Mongoose

```
js
CopyEdit
client: { type: mongoose.Schema.Types.ObjectId, ref: 'Client' },
```

⚙️ STOCK UPDATE FLOW (Shared via Utils)

Every time stock needs to be updated, you use:

- `increaseStock(itemId, qty)` → Called during:
  - Goods Receipt
  - Sales Return
- `decreaseStock(itemId, qty)` → Called during:
  - Delivery Challan
  - Purchase Return

Both helpers:

- Update the `Stock.js` model (track current quantity)
- Write logs into `StockLedger.js` model (for audit trail)

🔐 AUTH + ACCESS FLOW

- All backend routes are protected by `authMiddleware.js`
- Routes with sensitive operations (like adding users) also use role checks.

```
js
CopyEdit
router.post('/add', authMiddleware, roleMiddleware(['admin']));
```

The **JWT Token** is:

- Issued on login
- Stored in localStorage (frontend)
- Sent in headers as:

  ```
  http
  CopyEdit
  Authorization: Bearer <token>
  ```

🧠 Beginner-Level Tips to Remember

Concept	Meaning
Model	Blueprint of your data (like table schema)
Controller	Where logic lives for handling API calls
Route	Exposes an endpoint like `/api/items`
Reference	A connection between two models using ObjectId
Utils	Shared code, like updating stock
Middleware	Filters or checks before hitting controllers

✅ Summary

Your backend is designed like a real-world ERP system:

- Each real-world action → has a clear API
- All forms are linked → using object references
- Stock updates are **centralized** and **logged**
- You've separated concerns cleanly (routes, controllers, models)

📄 RELATIONAL STRUCTURE & MODULE DEPENDENCIES

🔹 1. Purchase Module

A. Purchase Order (PO)

- Created with:

- **Vendor** (ref: `Vendor._id`)
- **Items** with quantities, rates, and GST

- Saved as `PurchaseOrder` model.
- Key relations:
  - Vendor → PO (one-to-many)
  - PO → Items (many-to-many via embedded array)

B. Goods Receipt Note (GRN)

- Created **only after** a PO exists.
- Based on selected PO:
  - Select items & received quantities.
- Updates:
  - **Stock** (`increaseStock` method)
  - **PO status** (`Pending`, `Partially Received`, `Received`)
- Key relations:
  - GRN → PurchaseOrder (ref)
  - GRN → Items (ref)
  - GRN → Stock (update)
  - GRN → StockLedger (can be extended)

C. Purchase Return

- Works **after GRN** is generated.
- Allows returning GRN items.
- Affects:
  - **Stock** (`decreaseStock`)
  - **StockLedger** (via `transactionType: 'OUT'`)
- Key relations:
  - PurchaseReturn → PurchaseOrder (ref as `referenceId`)
  - PurchaseReturn → Items (ref)
  - PurchaseReturn → StockLedger

🔹 2. Sales Module

A. Sales Invoice

- Independent of PO or GRN.

- Based on:
  - **Client**
  - **Items with quantity and price**
- Affects:
  - **Stock** (`decreaseStock`)
  - **StockLedger** (can be improved to log OUT)
- Key relations:
  - SalesInvoice → Client
  - SalesInvoice → Items
  - SalesInvoice → Stock (decrease)
  - SalesInvoice → StockLedger (optional but recommended)

B. **Sales Return**

- Created **only after Sales Invoice**.
- Validates returned quantity doesn't exceed invoiced qty.
- Affects:
  - **Stock** (`increaseStock`)
  - **StockLedger** (type `RETURN`)
- Key relations:
  - SalesReturn → SalesInvoice
  - SalesReturn → Items
  - SalesReturn → Stock
  - SalesReturn → StockLedger

C. **Delivery Challan**

- Separate document for dispatch (non-billing dispatch).
- Affects:
  - **Stock** (`decreaseStock`)
  - Can be linked to Sales Invoice or work independently.
- Key relations:
  - DeliveryChallan → Items
  - DeliveryChallan → Stock

🔹 3. Inventory Module

A. Item Master

- The heart of the system.
- All transactions are based on items from this master.
- Fields: name, SKU, unit, price, stockThreshold, GST, etc.

B. Stock

- Tracks current quantity for each item.
- Updated via:
  - GRN (IN)
  - PO (IN)
  - Delivery Challan / Sales Invoice (OUT)
  - Stock Adjustment (IN/OUT)
  - Sales Return (IN)
  - Purchase Return (OUT)
- Relation:
  - One stock entry per item.

C. Stock Ledger

- Audit trail for all item-level stock movements.
- Every transaction should log here.
- Contains:
  - `transactionType`: IN / OUT / ADJUST / RETURN
  - `source`: e.g., `GoodsReceipt`, `SalesInvoice`, etc.
  - `sourceId`: ObjectId of the source doc

D. Stock Adjustment

- Manual increase or decrease in stock.
- Also logs into `StockLedger`.

📌 Final Flow Summary

```
[VENDOR] → [PO] → [GRN] → [STOCK IN] → [LEDGER]
                                ↓
                            [PURCHASE RETURN] → [STOCK OUT] → [LEDGER]

[CLIENT] → [SALES INVOICE] → [STOCK OUT] → [LEDGER]
                            ↓
```

```
                    [SALES RETURN] → [STOCK IN] → [LEDGER]

[DELIVERY CHALLAN] → [STOCK OUT] → [LEDGER]

[STOCK ADJUSTMENTS] → IN / OUT → [LEDGER]
```