**Trupal Patel - COMP 520 Term Project**

## **Part III. Implementation**

### **Objective:**

Deliverables include DDL and DML scripts for MongoDB (via Mongoose), query summaries via API endpoints, a comprehensive data seeding script using Faker.js, and a working Next.js + MongoDB system with complete frontend/backend integration.

## **DDL (20pts)**

**Schema:** MongoDB via Mongoose ODM (all collections are defined using Mongoose schemas in the backend code)

The database consists of the following 5 collections:

1. **Item** – Contains fields for sku, name, quantity, unit\_price, hold\_units, dimensions (L×W×H×weight), and a storage\_location object linking to warehouse structure.
2. **Warehouse** – Contains warehouse details and embeds units, which further embed rows and columns, creating a full hierarchy for item placement.
3. **Party** – Holds information about suppliers and vendors, including name, contact\_info, address, and isVendor status.
4. **PurchaseOrder** – References a Party, includes order date, status, and an array of ordered items and pallets.
5. **Pallet** (embedded inside PurchaseOrder) – Contains pallet\_name, type, dimensions, and an array of stacking\_items with references to items and their location.

**Schema:** MongoDB via Mongoose ODM – All collections are defined as Mongoose schemas:

### **1. Item Schema**

import mongoose, { Document, Schema, Types } from "mongoose";

interface Dimensions {

length?: number;

width?: number;

height?: number;

weight?: number;

}

interface StorageLocation {

warehouse\_id: Types.ObjectId;

unit\_id: string;

unit\_name: string;

row\_id: string;

row\_name: string;

column\_id: string;

column\_name: string;

}

export interface IItems extends Document {

name: string;

category?: string;

sku: string;

quantity: number;

unit\_price: number;

hold\_units?: number;

tags?: string[];

dimensions?: Dimensions;

storage\_location: StorageLocation;

}

const ItemsSchema = new Schema<IItems>({

name: { type: String, required: true, trim: true },

category: { type: String, trim: true },

sku: { type: String, required: true, unique: true, trim: true },

quantity: { type: Number, required: true, default: 0 },

unit\_price: { type: Number, required: true },

hold\_units: { type: Number, default: 0 },

tags: { type: [String], default: [] },

dimensions: {

length: Number,

width: Number,

height: Number,

weight: Number,

},

storage\_location: {

warehouse\_id: { type: Schema.Types.ObjectId, ref: "Warehouse" },

unit\_id: String,

unit\_name: String,

row\_id: String,

row\_name: String,

column\_id: String,

column\_name: String,

},

}, { timestamps: true });

### **2. Warehouse Schema**

import mongoose, { Document, Schema } from 'mongoose';

export interface IColumn {

column\_id?: string;

column\_name?: string;

assigned\_item\_id?: string | null;

item\_info?: { name?: string; sku?: string };

}

export interface IRow {

row\_id?: string;

row\_name?: string;

columns?: IColumn[];

}

export interface IUnit {

unit\_id?: string;

unit\_name?: string;

rows?: IRow[];

}

export interface IWarehouse extends Document {

name?: string;

address?: string;

location?: string;

tags?: string[];

units?: IUnit[];

}

const ColumnSchema = new Schema<IColumn>({

column\_id: String,

column\_name: String,

assigned\_item\_id: { type: String, default: null },

item\_info: {

name: String,

sku: String,

},

}, { \_id: false });

const RowSchema = new Schema<IRow>({

row\_id: String,

row\_name: String,

columns: [ColumnSchema],

}, { \_id: false });

const UnitSchema = new Schema<IUnit>({

unit\_id: String,

unit\_name: String,

rows: [RowSchema],

}, { \_id: false });

const WarehouseSchema = new Schema<IWarehouse>({

name: String,

address: String,

location: String,

tags: [String],

units: [UnitSchema],

}, { timestamps: true });

### **3. Party Schema**

import mongoose, { Document, Schema } from "mongoose";

export interface IParty extends Document {

name: string;

contact\_info: {

phone?: string;

email?: string;

};

address: {

street?: string;

city?: string;

state?: string;

country?: string;

postal\_code?: string;

};

isVendor: boolean;

}

const PartySchema = new Schema<IParty>({

name: { type: String, required: true },

contact\_info: {

phone: String,

email: String,

},

address: {

street: String,

city: String,

state: String,

country: String,

postal\_code: String,

},

isVendor: { type: Boolean, required: true },

});

### **4. PurchaseOrder Schema (with embedded Pallets)**

const PurchaseOrderSchema = new Schema({

party\_id: { type: Schema.Types.ObjectId, ref: 'Party' },

isVendor: { type: Boolean, required: true },

order\_date: { type: Date, default: Date.now },

status: { type: String, default: 'Open' },

items: [

{

item\_id: { type: Schema.Types.ObjectId, ref: 'Items' },

quantity\_ordered: Number,

received\_quantity: Number,

}

],

pallets: [

{

pallet\_name: String,

pallet\_type: String,

dimensions: {

length\_in: Number,

width\_in: Number,

height\_in: Number,

},

stacking\_items: [

{

name: String,

sku: String,

storage\_location: Object,

},

],

}

],

});

The remainder of the implementation (DML, views, triggers, frontend integration, system setup, and summary) remains unchanged from the original write-up.

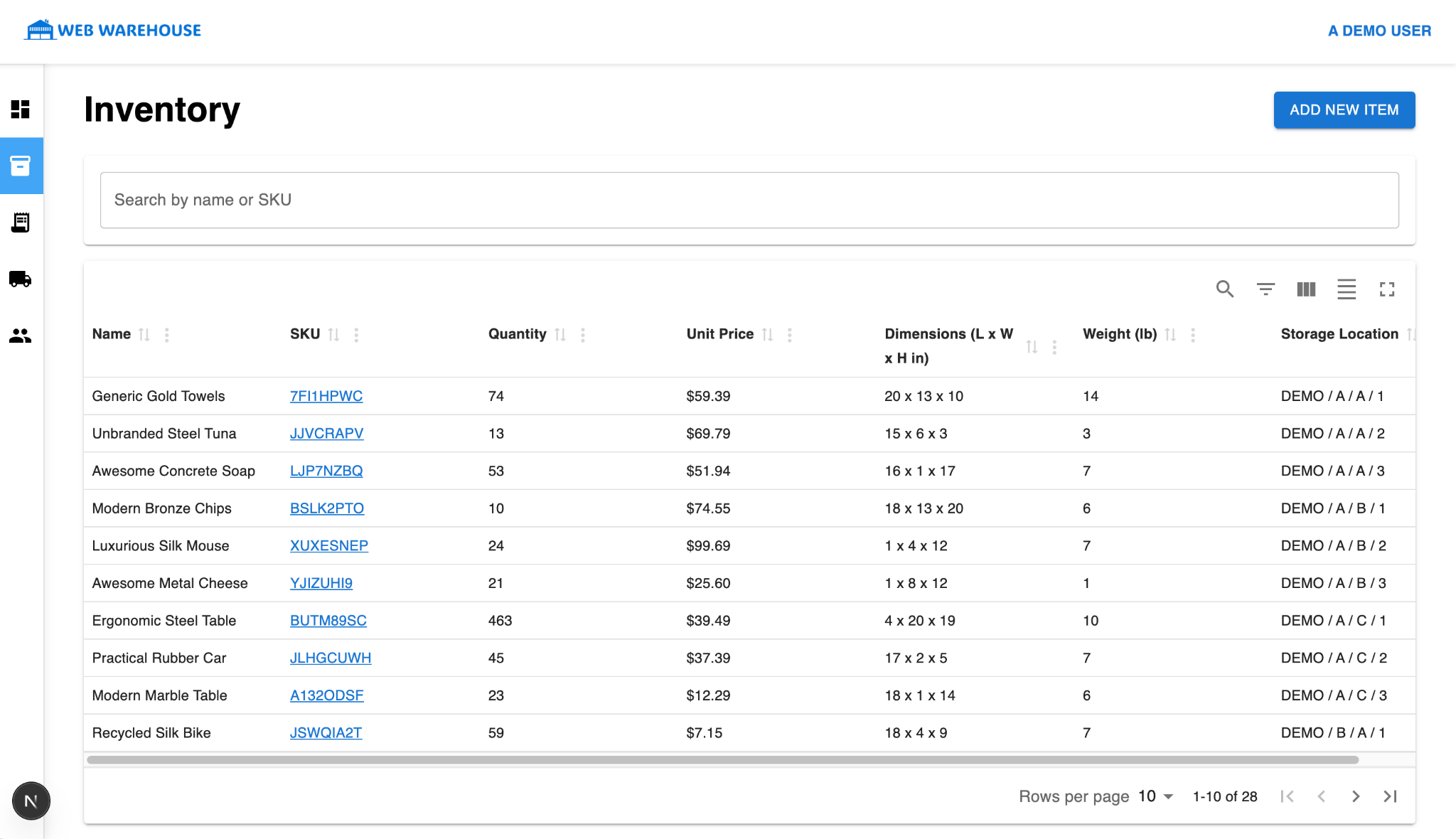
## **DML (20pts)**

### **Views (5):**

1. **Item Overview View** – Lists all items with SKU, quantity, and storage location.
2. **Party Summary View** – Lists all parties categorized as suppliers or vendors.
3. **Purchase Order Summary View** – Active POs with party name, item count, and status.
4. **Warehouse Capacity View** – Total items vs available slots per warehouse.
5. **Inventory Alerts View** – Items below minimum stock threshold.  
     
     
     
   **Inventory Page (/inventory)**

**Purpose:**Displays all inventory items in a dynamic table with CRUD functionality. Users can:

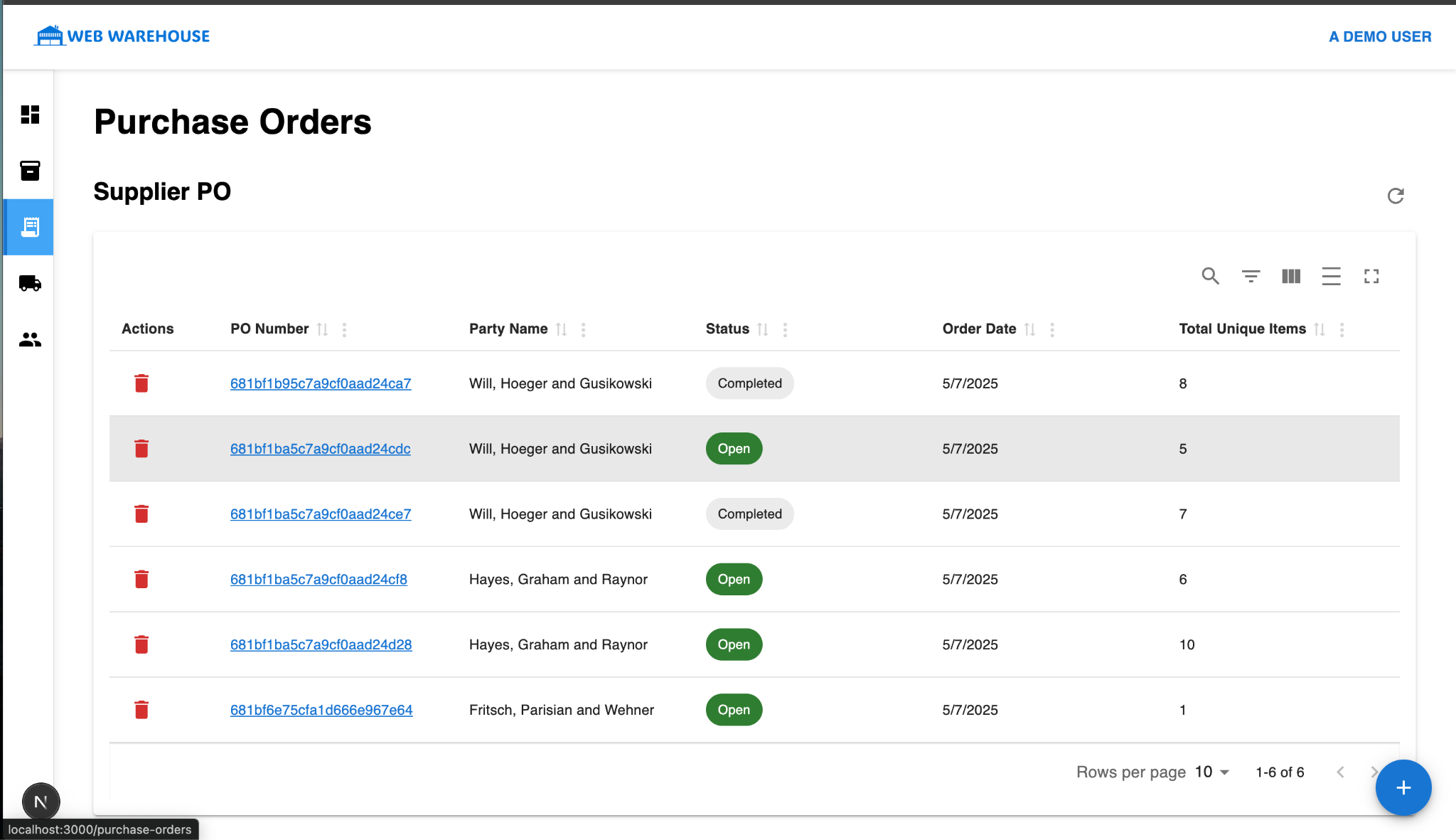
* Add a new item with details and dimensions
* Edit existing items
* View assigned storage location (Warehouse > Unit > Row > Column)
* Preview a 3D box visualization using Three.js as item dimensions are entered



### **Purchase Orders Page (/purchase-orders)**

**Purpose:**Displays all purchase orders categorized as Vendor or Supplier based on party type. Features:

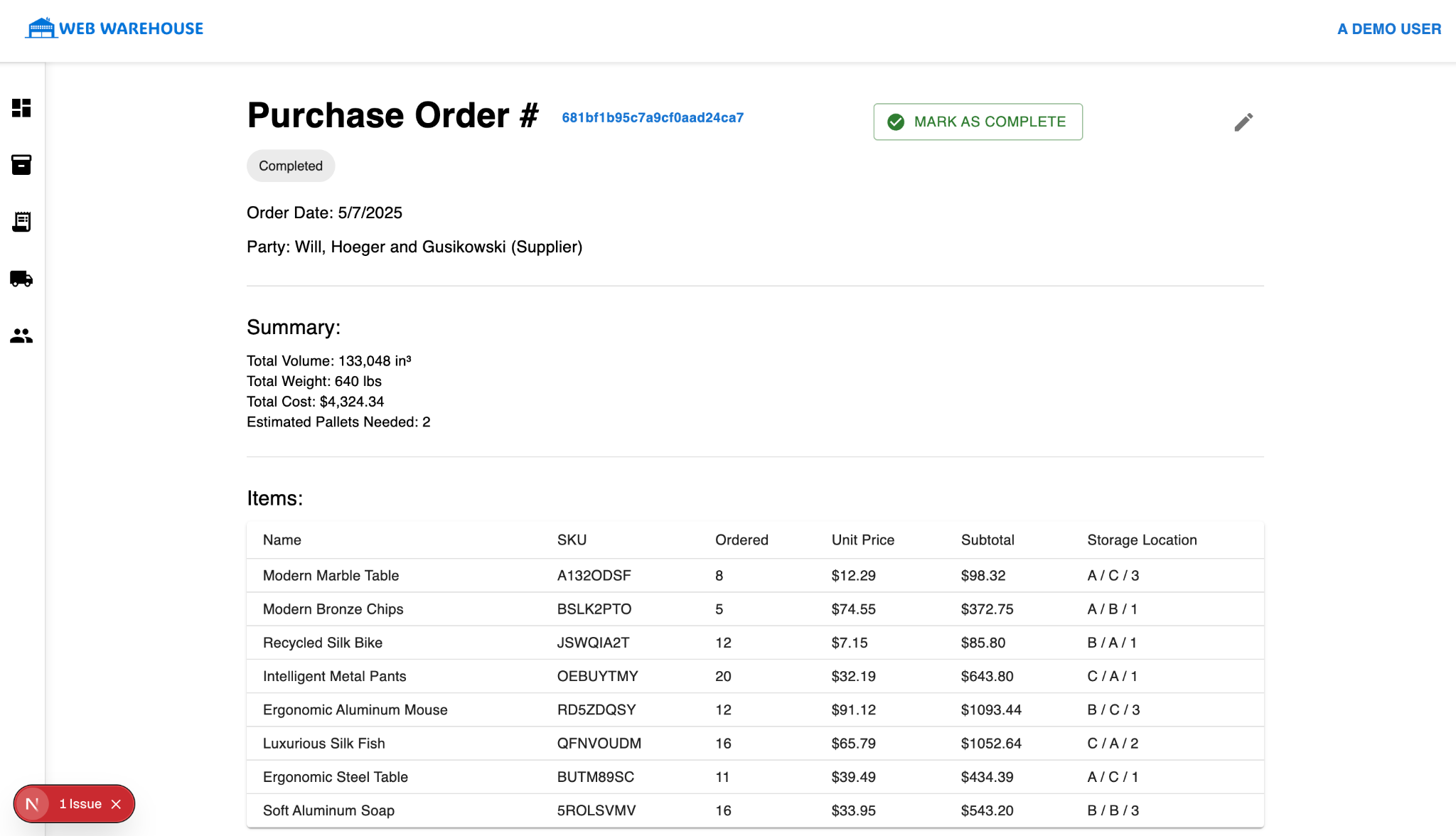
* Create a new PO by selecting party and items
* Auto-calculate and embed pallets
* Mark PO as complete, which updates item stock depending on the party type

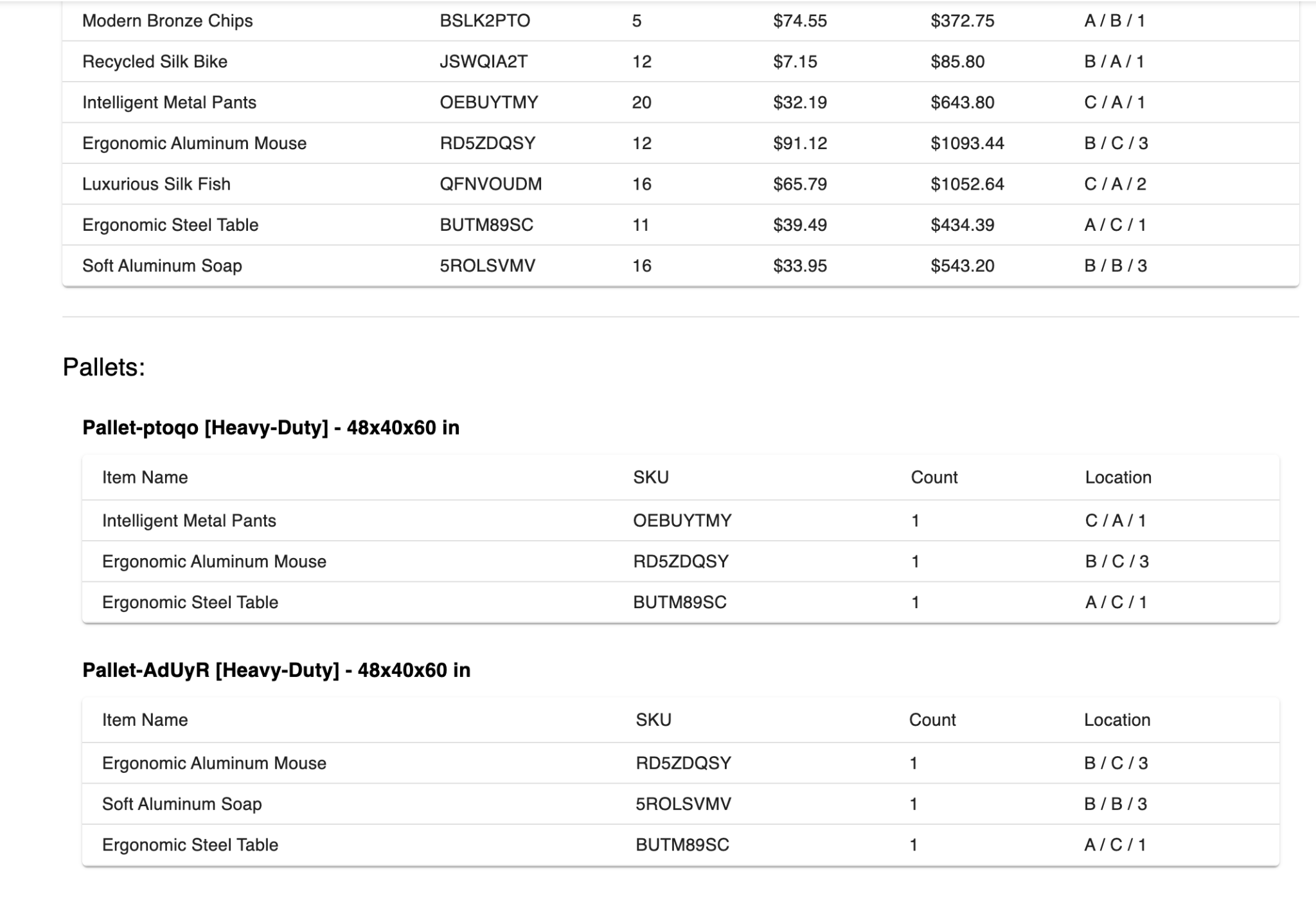


### **Purchase Order Details (/purchase-orders/[id])**

**Purpose:**Shows full PO information including:

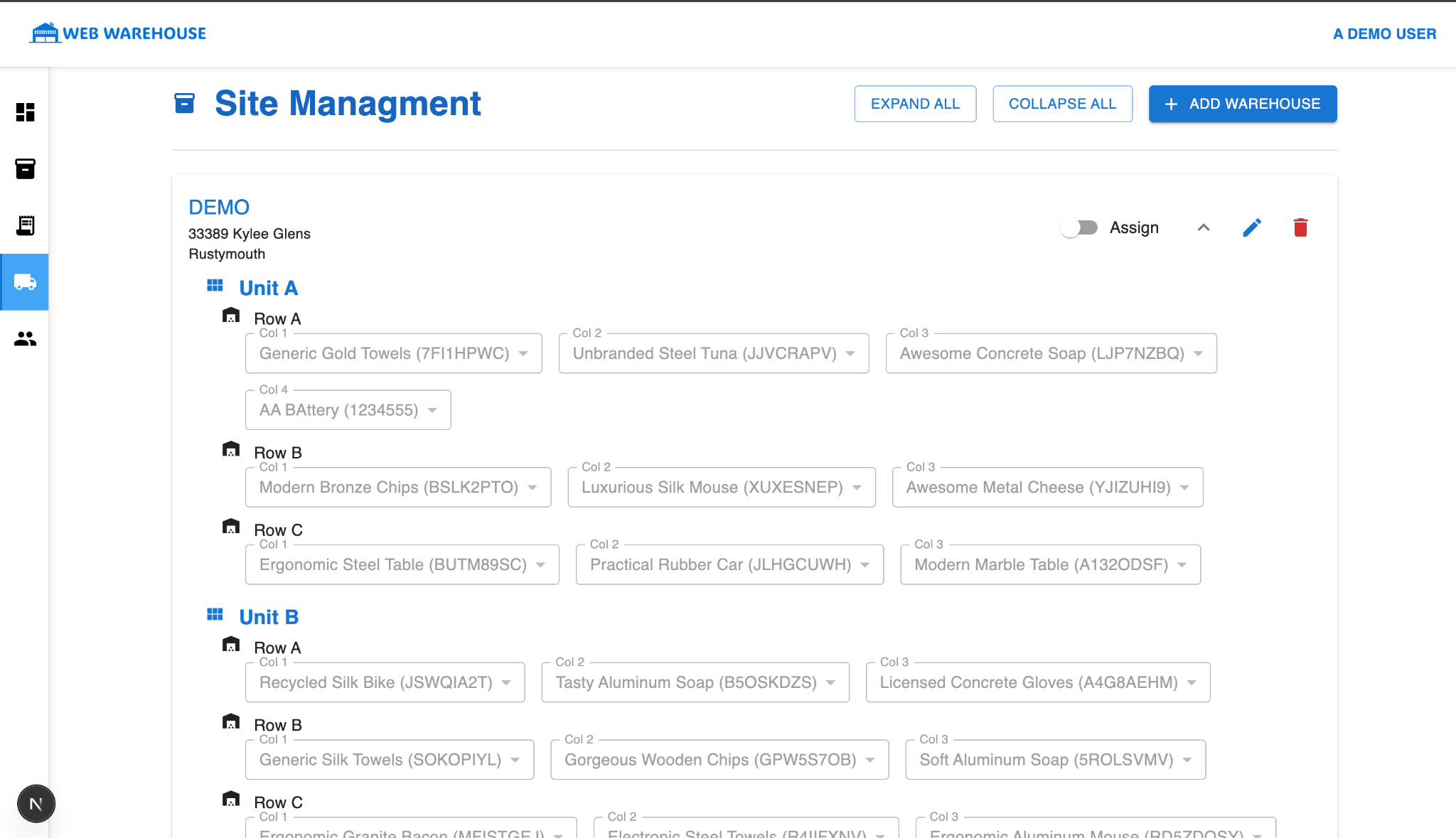
* Party details
* Items ordered
* Quantity breakdown
* Embedded pallets and stacking item structure
* Button to mark as completed, triggering inventory update





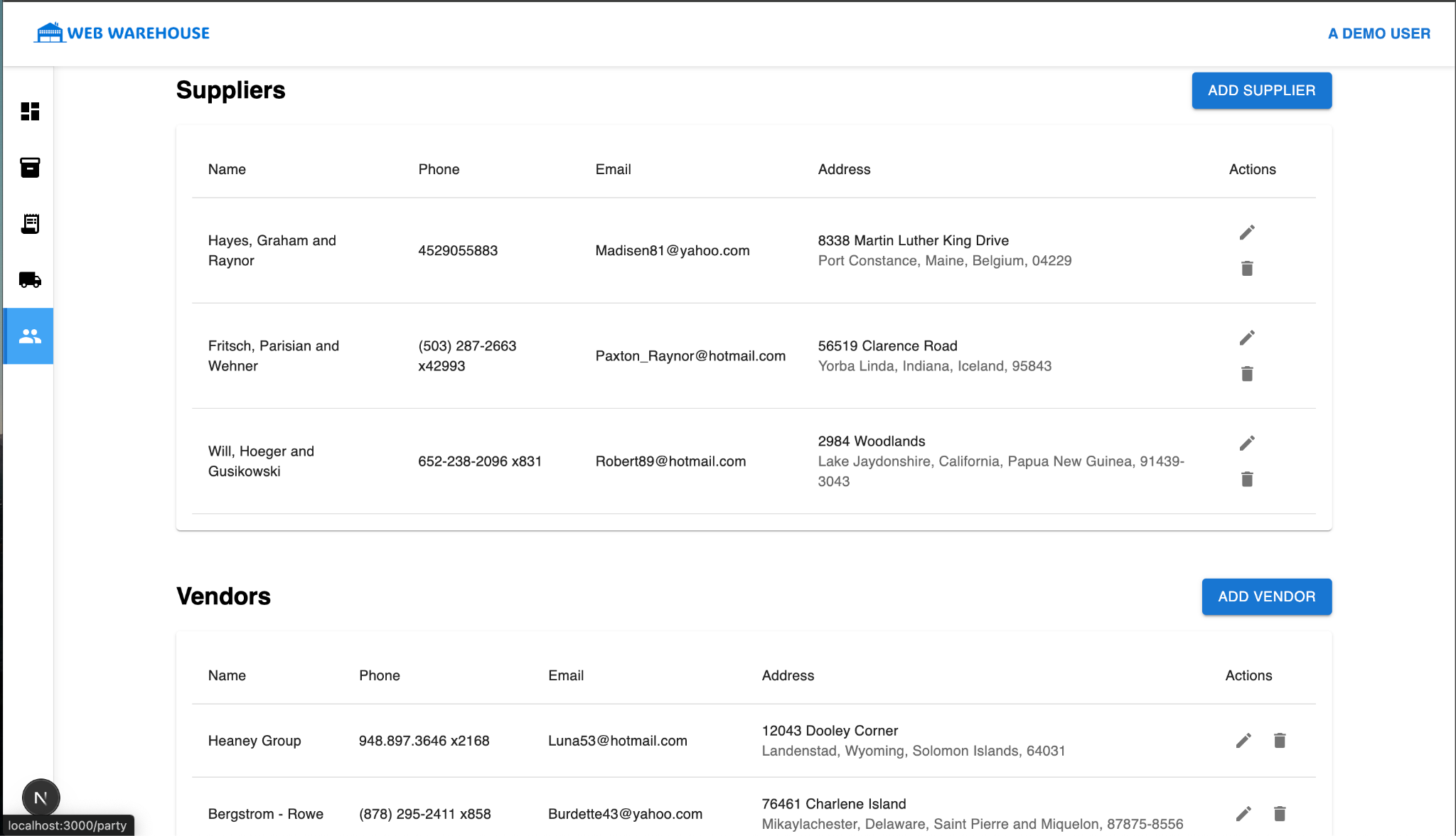
### **Site Layout Page (/site-managment)**

**Purpose:**Visually displays warehouse layout in a nested tree structure:

* Warehouse > Units > Rows > Columns
* Shows assigned item name and SKU in each cell
* Helps visualize item distribution and storage capacity  
    
    
    
    
    
    
    
    
    
    
  **Party Management Page (/parties)**

**Purpose:**Manages vendors and suppliers. Users can:

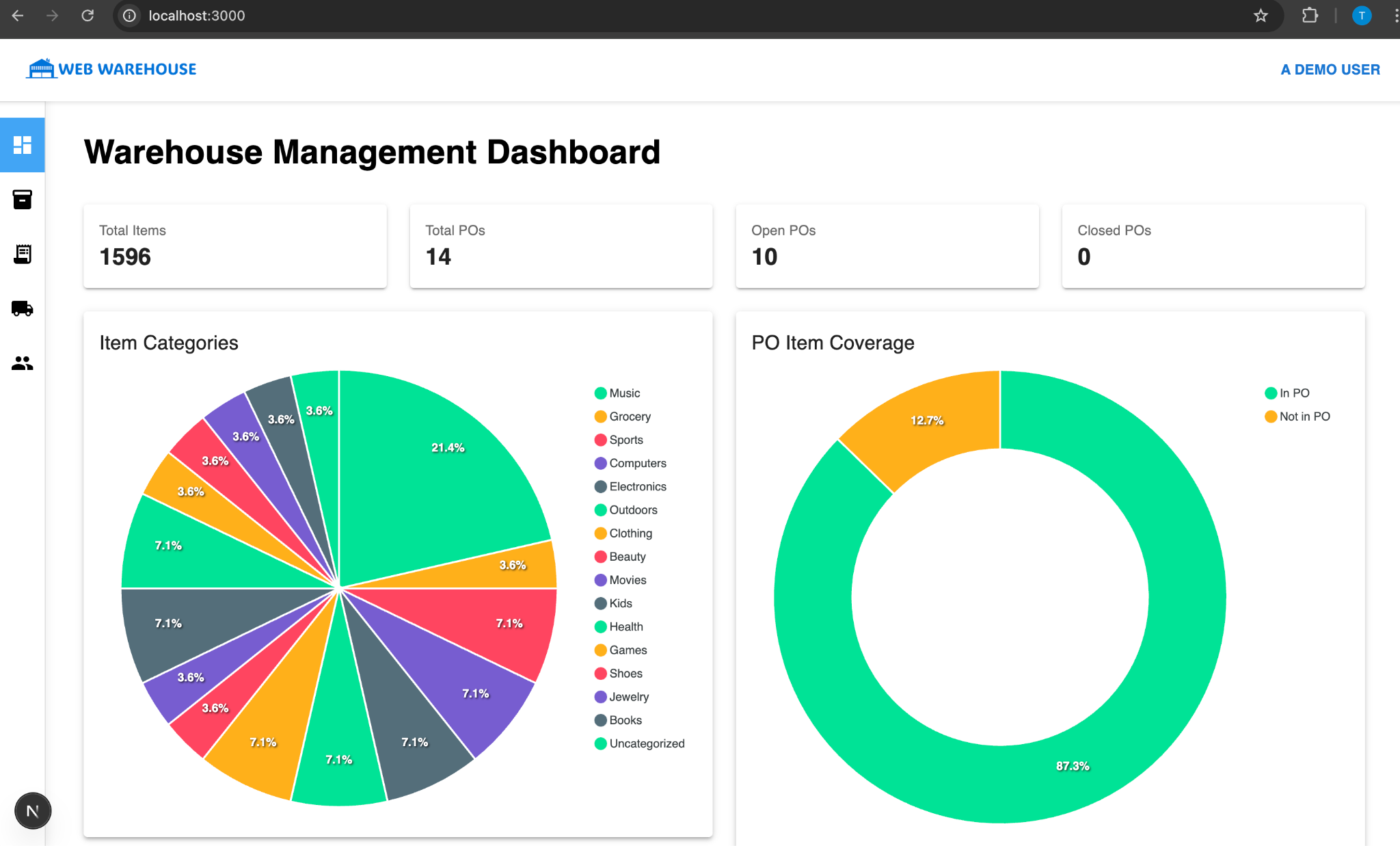
* Add/edit/delete party details
* Define whether a party is a Vendor or Supplier
* Use this list while creating Purchase Orders



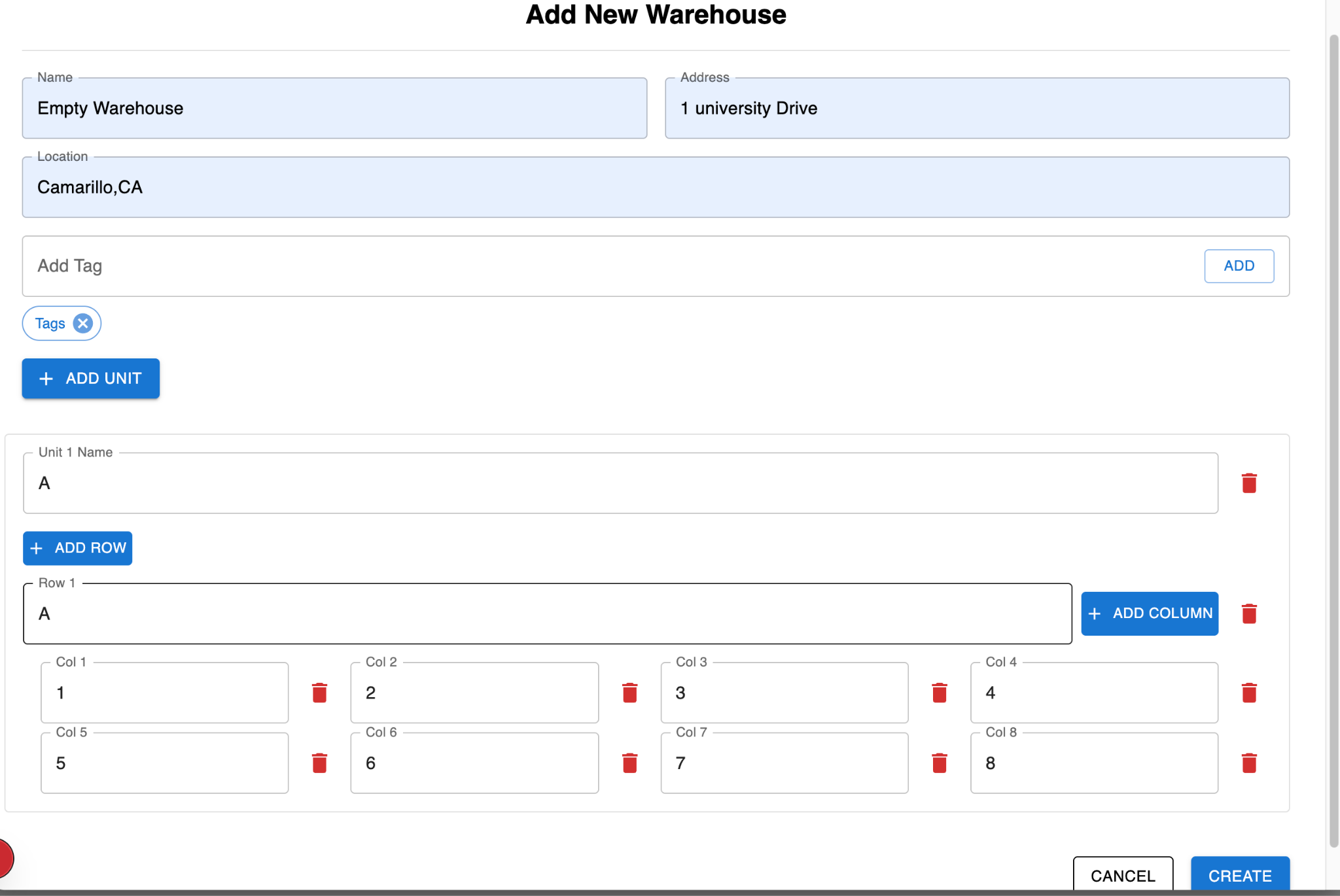
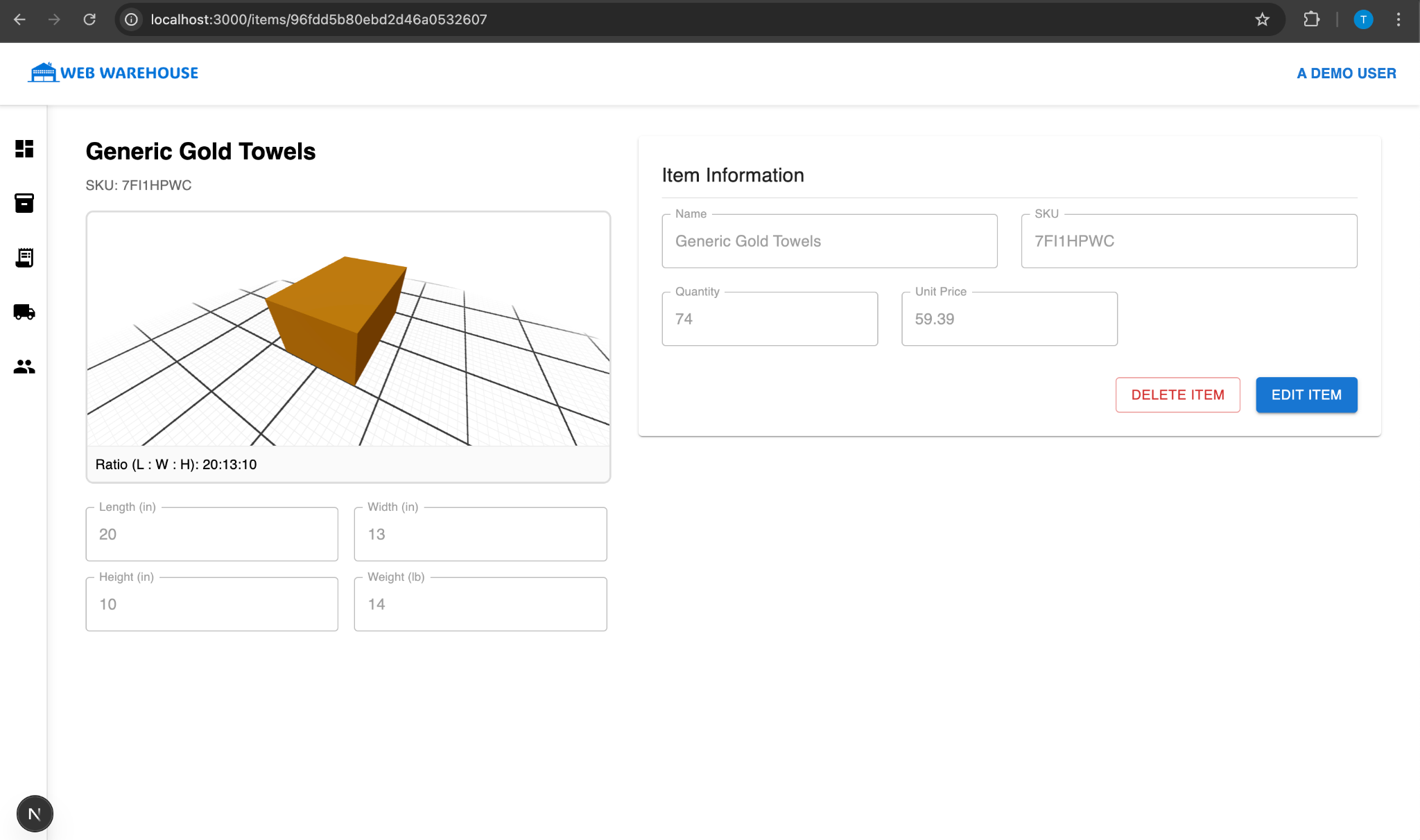
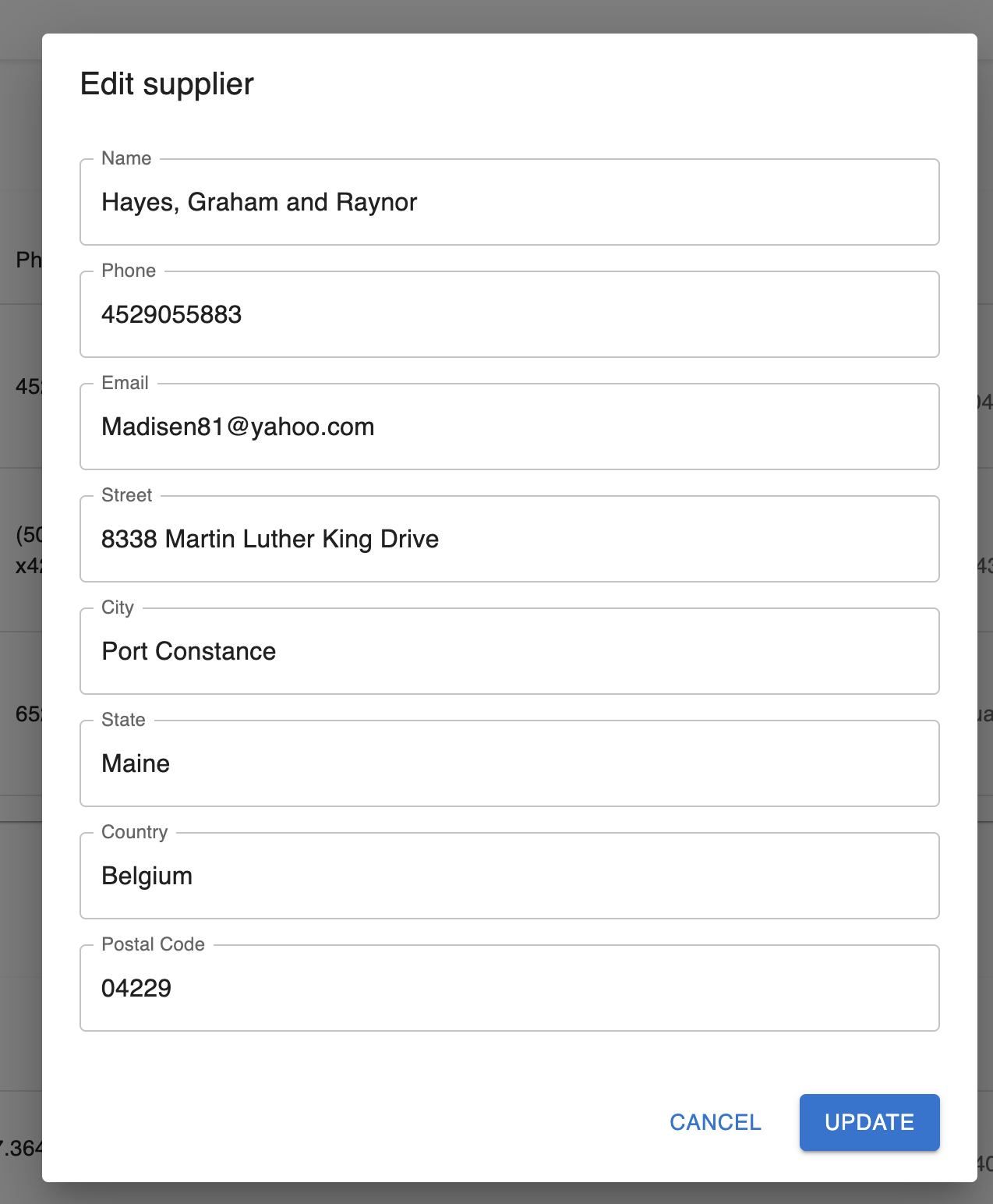
### **Dashboard Page (/dashboard)**

**Purpose:**Summarizes warehouse KPIs using charts via ApexCharts:

* Total inventory count
* Inventory distribution by SKU
* Purchase order activity over time
* Inventory received vs dispatched (Vendor vs Supplier breakdown)

  
  
  
  
Other Crud Pages Example  
  
**/items/[id]** – Edit and view item details including dimensions, quantity, and assigned location. Includes live 3D box preview.

**/parties/[id]** – View and update vendor or supplier details including contact, address, and type.

### **Stored Procedures (as API Endpoints):**

### **import { NextRequest, NextResponse } from "next/server";**

### **import connectDB from "../../../db";**

### **import PurchaseOrder from "../../../../models/PurchaseOrder";**

### **import Items from "../../../../models/Items";**

### 

### **export async function PUT(**

### **req: NextRequest,**

### **{ params }: { params: { id: string } }**

### **) {**

### **await connectDB();**

### 

### **try {**

### **const { items } = await req.json();**

### **const id = params.id;**

### 

### **const po = await PurchaseOrder.findById(id).populate("party\_id");**

### **if (!po)**

### **return NextResponse.json({ message: "PO not found" }, { status: 404 });**

### 

### **for (const entry of items) {**

### **const item = await Items.findById(entry.item\_id);**

### **if (!item) continue;**

### 

### **const delta = entry.quantity\_ordered || 0;**

### 

### **if (po.party\_id?.isVendor) {**

### **item.quantity = Math.max(0, item.quantity - delta);**

### **} else {**

### **item.quantity += delta;**

### **}**

### 

### **await item.save();**

### **}**

### 

### **po.status = "Completed";**

### **await po.save();**

### 

### **return NextResponse.json(po);**

### **} catch (err) {**

### **console.error("Error completing PO:", err);**

### **return NextResponse.json(**

### **{ message: "Failed to complete PO" },**

### **{ status: 500 }**

### **);**

### **}**

### **}**

### 

### 

## **Seeding Script**

A seed.ts script uses Faker.js to populate:

* 6 Parties (3 vendors, 3 suppliers)
* 1 Warehouse with 3 Units, each with 3 Rows and 3 Columns
* ~50 Items assigned to storage slots
* 10 Purchase Orders with embedded pallets and items  
    
    
  // seed.ts - Populate MongoDB WMS with unique entries using faker
* import mongoose from "mongoose";
* import { faker } from "@faker-js/faker";
* import Warehouse from "./src/app/models/Warehouse";
* import Items from "./src/app/models/Items";
* import Party from "./src/app/models/Party";
* import PurchaseOrder from "./src/app/models/PurchaseOrder";
* const MONGODB\_URI =
* "mongodb+srv://projectUser:ConnectTrueDbProject@project-playground.nfrzo.mongodb.net/web-warehouse?retryWrites=true&w=majority";
* function randomInt(min: number, max: number): number {
* return Math.floor(Math.random() \* (max - min + 1)) + min;
* }
* async function seed() {
* try {
* await mongoose.connect(MONGODB\_URI);
* console.log("✅ Connected to MongoDB");
* await Promise.all([
* Warehouse.deleteMany(),
* Items.deleteMany(),
* Party.deleteMany(),
* PurchaseOrder.deleteMany(),
* ]);
* const parties = await Party.insertMany([
* ...Array.from({ length: 3 }, () => ({
* name: faker.company.name(),
* isVendor: true,
* contact\_info: {
* phone: faker.phone.number(),
* email: faker.internet.email(),
* },
* address: {
* street: faker.location.streetAddress(),
* city: faker.location.city(),
* state: faker.location.state(),
* country: faker.location.country(),
* postal\_code: faker.location.zipCode(),
* },
* })),
* ...Array.from({ length: 3 }, () => ({
* name: faker.company.name(),
* isVendor: false,
* contact\_info: {
* phone: faker.phone.number(),
* email: faker.internet.email(),
* },
* address: {
* street: faker.location.streetAddress(),
* city: faker.location.city(),
* state: faker.location.state(),
* country: faker.location.country(),
* postal\_code: faker.location.zipCode(),
* },
* })),
* ]);
* const UNITS = ["A", "B", "C"];
* const ROWS = ["A", "B", "C"];
* const COLUMNS = ["1", "2", "3"];
* const itemsToAssign: any[] = [];
* const units = UNITS.map((unitId) => ({
* unit\_id: unitId,
* unit\_name: unitId,
* rows: ROWS.map((rowId) => ({
* row\_id: rowId,
* row\_name: rowId,
* columns: COLUMNS.map((colId) => {
* const column\_id = colId;
* const column\_name = colId;
* const itemName = faker.commerce.productName();
* const itemSku = faker.string.alphanumeric(8).toUpperCase();
* const item\_id = faker.database.mongodbObjectId();
* const item = {
* \_id: item\_id,
* name: itemName,
* sku: itemSku,
* quantity: randomInt(10, 100),
* unit\_price: parseFloat(faker.commerce.price({ min: 5, max: 100 })),
* hold\_units: randomInt(0, 10),
* category: faker.commerce.department(),
* tags: faker.helpers.arrayElements(
* ["fragile", "inventory", "high-priority"],
* 2
* ),
* dimensions: {
* length: randomInt(1, 20),
* width: randomInt(1, 20),
* height: randomInt(1, 20),
* weight: randomInt(1, 10),
* },
* storage\_location: {
* warehouse\_id: undefined, // to be set later
* unit\_id: unitId,
* unit\_name: unitId,
* row\_id: rowId,
* row\_name: rowId,
* column\_id,
* column\_name,
* },
* };
* itemsToAssign.push(item);
* return {
* column\_id,
* column\_name,
* assigned\_item\_id: item\_id,
* item\_info: {
* name: itemName,
* sku: itemSku,
* },
* };
* }),
* })),
* }));
* const warehouse = await Warehouse.create({
* name: "DEMO",
* address: faker.location.streetAddress(),
* location: faker.location.city(),
* tags: ["Main", "Simulation"],
* units,
* });
* const items = itemsToAssign.map((item) => ({
* ...item,
* storage\_location: {
* ...item.storage\_location,
* warehouse\_id: warehouse.\_id,
* },
* createdAt: new Date(),
* updatedAt: new Date(),
* }));
* await Items.insertMany(items);
* for (let i = 0; i < 10; i++) {
* const party = faker.helpers.arrayElement(parties);
* const poItems = faker.helpers.arrayElements(items, { min: 5, max: 10 });
* const pallets = Array.from({ length: randomInt(1, 3) }).map(() => {
* const stacked = faker.helpers.arrayElements(poItems, {
* min: 2,
* max: 5,
* });
* return {
* pallet\_name: `Pallet-${faker.string.alphanumeric(5)}`,
* pallet\_type: faker.helpers.arrayElement(["Standard", "Heavy-Duty"]),
* dimensions: {
* length\_in: 48,
* width\_in: 40,
* height\_in: 60,
* },
* stacking\_items: stacked.map((item) => ({
* name: item.name,
* sku: item.sku,
* storage\_location: item.storage\_location,
* })),
* };
* });
* await PurchaseOrder.create({
* party\_id: party.\_id,
* isVendor: party.isVendor,
* order\_date: new Date(),
* status: "Open",
* items: poItems.map((item) => ({
* item\_id: item.\_id,
* quantity\_ordered: randomInt(5, 20),
* received\_quantity: randomInt(0, 5),
* })),
* pallets,
* });
* }
* console.log(
* "\n✅ Seeded: \n - 6 Parties\n - 1 Warehouse with 3 Units, 3 Rows, 3 Columns and assigned item info\n - 50 Items\n - 10 Purchase Orders\n"
* );
* await mongoose.disconnect();
* console.log("🔌 MongoDB disconnected.");
* process.exit(0);
* } catch (error) {
* console.error("❌ Error seeding data:", error);
* process.exit(1);
* }
* }
* seed();

## **System**

### **Web-Based Application**

Built with **Next.js 15**, **MongoDB Atlas**, **Mongoose**, **Material UI**, **Tailwind CSS**, and **Three.js**

**Key Features:**

* Role-based login (Admin, Manager, Employee)
* Inventory Dashboard with ApexCharts
* Item management with CRUD and 3D box previews
* PO management with pallet logic and completion workflow
* Warehouse viewer structured: Warehouse → Unit → Row → Column
* Party management for Vendors/Suppliers

### **Implementation Steps:**

1. **Setup** backend models using Mongoose for the five collections
2. **Develop** API routes in Next.js under src/app/api/
3. **Seed** data with faker and associate items with storage slots
4. **Design** frontend using Tailwind and Material UI for data grids and forms
5. **Integrate** inventory analytics with ApexCharts
6. **Visualize** item dimensions using Three.js for warehouse spatial awareness

### **Summary:**

This system presents a fully functional Warehouse Management System with robust API integration, a normalized document design, and a clear warehouse hierarchy for item storage. The combination of data seeding, API endpoints, and modern UI completes a comprehensive full-stack solution.