2. Questions for Product Team (Identify Gaps)

Area		Questions
Inventory Changes	→	Should we track who made the change (user/system)?
Bundles	→	Can they contain other bundles (nested)?
Warehouses	→	Can a warehouse belong to more than one company (e.g. shared facilities)
Suppliers	→	Do suppliers provide to specific warehouses or to the entire company?
Products	→	Are there categories or variants (like sizes, colors)?
Stocking Rules	→	Do we need reorder thresholds, min/max stock levels per warehouse?
Pricing	→	Can price vary by supplier or warehouse, or is it global per SKU?
Auditing	→	Should we track changes to product metadata (e.g., name, price edits)?

Part 3. Design Decisions & Justification

Decision		Justification
sku as UNIQUE	→	Ensures SKU is global and enforceable at the DB level
inventory as composite PK	→	Prevents duplicate entries for a product-warehouse pair
inventory_changes as history log	→	Enables tracking and auditability of stock movement
bundles with self-reference check	→	Prevents infinite loops in nested bundles
is_bundle flag	→	Allows quick querying without a join to bundles
product_suppliers table	→	Models real-world many-to-many supplier relationships

- **Code:**

```
from flask import Flask, jsonify, request
from sqlalchemy import func
from datetime import datetime, timedelta

from models import db, Company, Warehouse, Product, Inventory, ProductThreshold, Supplier, ProductSupplier, Sale
```

```
app = Flask(__name__)

@app.route('/api/companies/<int:company_id>/alerts/low-stock', methods=['GET'])
def get_low_stock_alerts(company_id):
    alerts = []
    try:
        # Step 1: Get all warehouses for the company
        warehouses = Warehouse.query.filter_by(company_id=company_id).all()

        if not warehouses:
            return jsonify({"alerts": [], "total_alerts": 0})

        warehouse_ids = [w.id for w in warehouses]
        warehouse_map = {w.id: w.name for w in warehouses}

        # Step 2: Get products with recent sales (last 30 days)
        thirty_days_ago = datetime.utcnow() - timedelta(days=30)
        active_product_ids = (
            db.session.query(Sale.product_id)
            .filter(Sale.warehouse_id.in_(warehouse_ids))
            .filter(Sale.sold_at >= thirty_days_ago)
            .distinct()
            .all()
        )
        active_product_ids = [p[0] for p in active_product_ids]

        if not active_product_ids:
            return jsonify({"alerts": [], "total_alerts": 0})

        # Step 3: Get thresholds for those products
        thresholds = {
```

```
        row.product_id: row.threshold
        for row in
ProductThreshold.query.filter(ProductThreshold.product_id.in_(active_product_ids)).all()
    }

    # Step 4: Check inventory for these products in all company warehouses
    inventory_records = (
        db.session.query(Inventory)
        .filter(Inventory.product_id.in_(active_product_ids))
        .filter(Inventory.warehouse_id.in_(warehouse_ids))
        .all()
    )

    # Step 5: Calculate sales velocity (daily average over last 30 days)
    sales_data = (
        db.session.query(
            Sale.product_id,
            Sale.warehouse_id,
            func.sum(Sale.quantity).label('total_sales')
        )
        .filter(Sale.product_id.in_(active_product_ids))
        .filter(Sale.warehouse_id.in_(warehouse_ids))
        .filter(Sale.sold_at >= thirty_days_ago)
        .group_by(Sale.product_id, Sale.warehouse_id)
        .all()
    )

    sales_map = {
        (s.product_id, s.warehouse_id): s.total_sales / 30 for s in sales_data  # avg per day
    }
```

```
# Step 6: Build alerts
for inv in inventory_records:
    threshold = thresholds.get(inv.product_id)
    if threshold is None:
        continue  # skip if no threshold set

    if inv.quantity >= threshold:
        continue  # not low-stock

    product = Product.query.get(inv.product_id)

    # Estimate stockout days
    avg_daily_sales = sales_map.get((inv.product_id, inv.warehouse_id), 0)
    if avg_daily_sales > 0:
        days_until_stockout = int(inv.quantity / avg_daily_sales)
    else:
        days_until_stockout = None  # can't estimate

    # Get one supplier (for simplicity)
    supplier_info = (
        db.session.query(Supplier)
        .join(ProductSupplier, Supplier.id == ProductSupplier.supplier_id)
        .filter(ProductSupplier.product_id == inv.product_id)
        .first()
    )

    alerts.append({
        "product_id": product.id,
        "product_name": product.name,
        "sku": product.sku,
        "warehouse_id": inv.warehouse_id,
```

```
            "warehouse_name": warehouse_map[inv.warehouse_id],
            "current_stock": inv.quantity,
            "threshold": threshold,
            "days_until_stockout": days_until_stockout,
            "supplier": {
                "id": supplier_info.id,
                "name": supplier_info.name,
                "contact_email": supplier_info.contact_info
            } if supplier_info else None
        })

    return jsonify({
        "alerts": alerts,
        "total_alerts": len(alerts)
    })

except Exception as e:
    return jsonify({"error": "Internal server error", "details": str(e)}), 500
```

Supporting Table Models

```
class ProductThreshold(db.Model):
    __tablename__ = 'product_thresholds'
    product_id = db.Column(db.Integer, db.ForeignKey('products.id'), primary_key=True)
    threshold = db.Column(db.Integer, nullable=False)

class Sale(db.Model):
    __tablename__ = 'sales'
    id = db.Column(db.Integer, primary_key=True)
    product_id = db.Column(db.Integer, db.ForeignKey('products.id'))
    warehouse_id = db.Column(db.Integer, db.ForeignKey('warehouses.id'))
```

```
quantity = db.Column(db.Integer, nullable=False)

sold_at = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
```