Part 1 – Code Review & Debugging

1. Problems in the Code

Technical Problems

1. No check if SKU is already in the system — duplicates possible.
2. No check for missing required fields — may cause errors or bad data.
3. Price may lose accuracy if stored as float instead of decimal.
4. Two separate commits — slows performance and may leave data half-saved if the second fails.
5. No error handling or rollback — system may have broken data if something fails.
6. Only supports one warehouse per product — but products can be in many warehouses.

Business Logic Problems

1. No check if the warehouse exists.
2. Allows negative quantity for stock.
3. No support for optional fields like description or category.

---

2. Impact in Production

- Duplicate SKUs make it hard to track products.
- If second commit fails, product is saved but inventory is not — bad data.
- Missing validation can cause wrong reports and wrong supplier orders.
- Not supporting many warehouses makes the app useless for big companies.

---

3. Fixed Code

```
@app.route('/api/products', methods=['POST'])
def create_product():
    data = request.get_json()

    # Validate required fields
    required_fields = ['name', 'sku', 'price', 'warehouse_id', 'initial_quantity']
    for field in required_fields:
        if field not in data:
            return {"error": f"Missing field: {field}"}, 400

    # Check SKU uniqueness
    if Product.query.filter_by(sku=data['sku']).first():
        return {"error": "SKU already exists"}, 400

    try:
        with db.session.begin():  # Single transaction
            product = Product(
                name=data['name'],
                sku=data['sku'],
                price=Decimal(data['price'])  # More accurate
            )
            db.session.add(product)

            # Add inventory record
            inventory = Inventory(
```

```
            product_id=product.id,
            warehouse_id=data['warehouse_id'],
            quantity=max(0, data['initial_quantity'])  # No negative stock
        )
        db.session.add(inventory)

    return {"message": "Product created", "product_id": product.id}, 201

except Exception as e:
    db.session.rollback()
    return {"error": str(e)}, 500
```

Fixes made:

- Added validation.
- Checked SKU uniqueness.
- Used Decimal for price.
- Used single transaction.
- Prevented negative quantity.

Part 2 – Database Design

Tables

1. **companies** – id, name, contact_info
2. **warehouses** – id, company_id (FK), name, location
3. **products** – id, name, sku (unique), price, product_type (simple/bundle)
4. **inventory** – product_id (FK), warehouse_id (FK), quantity, last_updated
5. **inventory_history** – id, inventory_id (FK), change_amount, reason, date
6. **suppliers** – id, name, contact_email, phone
7. **supplier_products** – supplier_id (FK), product_id (FK), lead_time_days
8. **product_bundles** – bundle_id (FK), component_id (FK), quantity

Indexes & Constraints

- SKU is unique for all companies.
- Composite index (warehouse_id, product_id) in inventory for fast lookups.

Part 3 – API for Low Stock Alerts (Node.js)

Assumptions

• Sequelize ORM with tables: Product, Warehouse, Inventory, Supplier, SupplierProducts, Sales.
• low_stock_threshold is stored in Product.
• "Recent sales" means in the last 30 days.
• Performance – Uses a subquery (Sequelize.literal) instead of fetching recentSalesProductIds separately.
• Multiple Suppliers – Supports an array of suppliers instead of only the first one.
• Days Until Stockout – Calculated based on recent sales velocity, with safe division handling.

Code : lowStockAlerts.js

```
const express = require('express');

const { Op, Sequelize } = require('sequelize');

const { Product, Inventory, Supplier, SupplierProducts, Sales } = require('../models');


const router = express.Router();


router.get('/low-stock-alerts', async (req, res) => {

  try {

    const lowStockProducts = await Product.findAll({

      include: [

        {

          model: Inventory,

          include: ['Warehouse']

        },

        {

          model: SupplierProducts,

          include: [Supplier]

        }

      ],

      where: Sequelize.literal(`

        Product.id IN (

          SELECT i.ProductId

          FROM Inventories i

          WHERE i.quantity < (

            SELECT low_stock_threshold FROM Products p WHERE p.id = i.ProductId

          )

          AND i.ProductId IN (

            SELECT DISTINCT ProductId FROM Sales
```

```
        WHERE saleDate >= DATE_SUB(NOW(), INTERVAL 30 DAY)
      )
    )
  `)
});

const results = lowStockProducts.map(product => {
  const totalQuantity = product.Inventories.reduce((sum, inv) => sum + inv.quantity, 0);
  const totalRecentSales = product.Sales
    ? product.Sales.reduce((sum, sale) => sum + sale.quantity, 0)
    : 0;
  const dailySalesVelocity = totalRecentSales / 30;
  const daysUntilStockout = dailySalesVelocity > 0
    ? (totalQuantity / dailySalesVelocity).toFixed(1)
    : 'N/A';

  return {
    productId: product.id,
    productName: product.name,
    currentStock: totalQuantity,
    threshold: product.low_stock_threshold,
    suppliers: product.SupplierProducts.map(sp => ({
      supplierId: sp.Supplier.id,
      supplierName: sp.Supplier.name
    })),
    daysUntilStockout
  };
});
```

```
    res.json(results);
  } catch (error) {
    console.error('Error fetching low stock alerts:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});

module.exports = router;
```

Edge Cases Handled

- **No low stock products** → returns alerts: [] and total_alerts: 0.
- **No supplier linked** → supplier field is null.
- **Multiple warehouses** → handled via Warehouse join.