Inventory Management System for B2B SaaS

Overview

You're joining a team building "StockFlow" - a B2B inventory management platform. Small businesses use it to track products across multiple warehouses and manage supplier relationships.

NAME: Piyush Savale

Part 1: Code Review & Debugging (30 minutes)

A previous intern wrote this API endpoint for adding new products. **Something is wrong** - the code compiles but doesn't work as expected in production.

Solution:

1) The handler should be wrapped in a try catch block to prevent crashing and for error handling - If there is any error we can identify the cause and if it fails it might crash the whole code hence we use try catch block
2) Check the body to verify all the required fields are present - if we dont have all the required data then we cant continue to execute the rest of the code
3) Add Null Handling - the values are null then it cant be stored in the db and the api will fail
4) If the price is string convert it to a decimal - sometimes maybe a string maybe passed from the frontend as a number we can convert it to decimal to avoid errors
5) If the SKU is already is present we return an error - if the sku is already present we return to avoid two SKUs having same name

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

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

        # Convert and validate price
        try:
            price = Decimal(data['price'])
        except Exception:
            return jsonify({"error": "Invalid price format"}), 400
```

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

    # Transaction start
    product = Product(
        name=data['name'],
        sku=data['sku'],
        price=price
    )
    db.session.add(product)
    inventory = Inventory(
        product_id=product.id,
        warehouse_id=data['warehouse_id'],
        quantity=data['initial_quantity']
    )
    db.session.commit()

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

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

Part 2: Database Design (25 minutes)

Based on the requirements below, design a database schema. **Note**: These requirements are intentionally incomplete - you should identify what's missing.

Format: Use any notation (SQL DDL, ERD, text description, etc.)

Solution:

Business

ID (PRIMARY KEY,UNIQUE CONSTRAINT) // unique to identify

Buisness_name

Owner_name

Registred_mobileno (UNIQUE,NOT NULL) should be unique

Email

Address

Warehouses[foriegn_key : Warehouse_id] // A business can have multiple warehouses

Suppliers[foreign_key : Suppliers.supplier_id] // can have multiple warehouses

Created_at

Warehouses

Warehouse_id (PRIMARY KEY,UNIQUE CONSTRAINT,Foreign Key)

Warehouse_name

Products[Foreign Key : Products.product_id] // warehouse can have many products

Created_at

Products

Product_id (PRIMARY KEY,UNIQUE CONSTRAINT)

Warehouse_id

Product_name

Price

Unit

Quantity (default 0)

Is_bundle (BOOLEAN DEFAULT FALSE)

Bundle_items [Array of Product_ids]

SKU (Unique)

Supplier_id (Foreign_key : Supplier.supplier_id)

Supplier

supplier_id(PRIMARY , UNIQUE)

Product[]

Supplier_name (NOTNULL)

Supplier_no

Supplier_address

Questions:

Should i decompose the table structure more??

How many units are supported ? (lbs ,kgs , cm)

Can same product exist in multiple warehouses??

Can warehouse ,supplier , product be deleted?

Is there need to track the created_at time?

Can two business have same supplier?

Can a single product have multiple suppliers or only one?

If multiple suppliers are allowed, do we need to track supplier-specific pricing and stock?

Part 3: API Implementation (35 minutes)

Implement an endpoint that returns low-stock alerts for a company.

Business Rules (discovered through previous questions):

- Low stock threshold varies by product type
- Only alert for products with recent sales activity
- Must handle multiple warehouses per company
- Include supplier information for reordering

Endpoint Specification:

GET /api/companies/{company_id}/alerts/low-stock

Expected Response Format:

```
{
  "alerts": [
    {
      "product_id": 123,
      "product_name": "Widget A",
      "sku": "WID-001",
      "warehouse_id": 456,
```

```
      "warehouse_name": "Main Warehouse",
      "current_stock": 5,
      "threshold": 20,
      "days_until_stockout": 12,
      "supplier": {
        "id": 789,
        "name": "Supplier Corp",
        "contact_email": "orders@supplier.com"
      }
    }
  ],
  "total_alerts": 1
}
```

Your Tasks:

1. **Write Implementation**: Use any language/framework (Python/Flask, Node.js/Express, etc.)
2. **Handle Edge Cases**: Consider what could go wrong
3. **Explain Approach**: Add comments explaining your logic

Hints: You'll need to make assumptions about the database schema and business logic. Document these assumptions.

Solution ➕

```
app.get("/api/v1/companies/:company_id/alerts/low_stock",(req,res)=
>{

    const company_id = req.params.company_id;



    try{

        // find the company by id assuming we have a function to do
this

        let company =  findCompanyById(company_id);



        // if the company does not exist return 404

        if(!company){
```

```
            res.status(404).json({
                error: "Company not found"
            })
        }
        // init alerts array
        let alerts = [];

        // find all the users warehouses
        const warehouses = await Warehouse.find({company_id: company_id});

        // loop through all the warehouses
        for(const warehouse of warehouses){
            // find all the products in the warehouse
            const products = await Product.find({warehouse_id: warehouse._id});

            // for each product if the quantity is less than the threshold then add the alert to the alerts array
            for(const product of products){
                if(product.stock < product.threshold){
                    const supplier = await Supplier.findById(product.supplier_id);
                    // simply push it to the array
```

```
                    alerts.push({
                        product_id: product._id,
                        product_name: product.product_name,
                        sku: product.sku,
                        warehouse_id: warehouse._id,
                        warehouse_name: warehouse.warehouse_name,
                        current_stock: product.quantity,
                        threshold: product.threshold,
                        // assuming we have a function to calculate
the days until stockout
                        days_until_stockout:
calculateDaysUntilStockout(product._id, product.quantity),
                        supplier: {
                          id: supplier?._id,
                          name: supplier?.supplier_name,
                          contact_email: supplier?.email || "N/A",
                        },
                      });
            }
        }

        // final response
        res.json({status: "success",
```

```
alerts:alerts,total_alerts:alerts.length});



    }catch(error){

        // error response

        res.status(500).json({

            error: "Internal server error"

        })

    }




})
```