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.

Time Allocation

- **Take-Home Portion**: 90 minutes maximum
- **Live Discussion**: 30-45 minutes (scheduled separately)

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.

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

    # Create new product
    product = Product(
        name=data['name'],
        sku=data['sku'],
        price=data['price'],
        warehouse_id=data['warehouse_id']
    )

    db.session.add(product)
    db.session.commit()

    # Update inventory count
    inventory = Inventory(
        product_id=product.id,
        warehouse_id=data['warehouse_id'],
        quantity=data['initial_quantity']
    )

    db.session.add(inventory)
    db.session.commit()

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

Your Tasks:

1. **Identify Issues**: List all problems you see with this code (technical and business logic)
2. **Explain Impact**: For each issue, explain what could go wrong in production
3. **Provide Fixes**: Write the corrected version with explanations

Additional Context (you may need to ask for more):

- Products can exist in multiple warehouses
- SKUs must be unique across the platform
- Price can be decimal values
- Some fields might be optional

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.

Given Requirements:

- Companies can have multiple warehouses
- Products can be stored in multiple warehouses with different quantities
- Track when inventory levels change
- Suppliers provide products to companies
- Some products might be "bundles" containing other products

Your Tasks:

1. **Design Schema**: Create tables with columns, data types, and relationships
2. **Identify Gaps**: List questions you'd ask the product team about missing requirements
3. **Explain Decisions**: Justify your design choices (indexes, constraints, etc.)

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

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.

Submission Instructions

1. **Create a document** with your responses to all three parts

2. **Include reasoning** for each decision you made
3. **List assumptions** you had to make due to incomplete requirements
4. **Submit within 90 minutes** of receiving this case study
5. **Be prepared** to walk through your solutions in the live session

---

Live Session Topics (Preview)

During our video call, we'll discuss:

- Your debugging approach and thought process
- Database design trade-offs and scalability considerations
- How you'd handle edge cases in your API implementation
- Questions about missing requirements and how you'd gather more info
- Alternative approaches you considered

Evaluation Criteria

Technical Skills:

- Code quality and best practices
- Database design principles
- Understanding of API design
- Problem-solving approach

Communication:

- Ability to identify and ask about ambiguities
- Clear explanation of technical decisions
- Professional collaboration style

Business Understanding:

- Recognition of real-world constraints
- Consideration of user experience
- Scalability and maintenance thinking

Note: This is designed to assess your current skill level and learning potential. We don't expect perfect solutions - we're more interested in your thought process and ability to work with incomplete information.