StockFlow Inventory Management System - Case Study Submission

This document summarizes my approach and implementation for the StockFlow Inventory Management System case study. The challenge was broken into three parts: debugging an existing endpoint, designing a database schema, and implementing a new API for low-stock alerts.

Part 1: Code Review & Debugging - POST /api/products

Identified Issues in the Original Flask Code

- No SKU Uniqueness Check: The code assumes SKUs are unique but doesn't enforce or validate this before saving.
- Missing Input Validation: It doesn't check whether required fields are missing.
- Unsafe Database Commit Logic: Two separate commits mean inventory could be saved without a valid product.
- No Error Handling: If something goes wrong, the system crashes without a message.

Fixed Implementation (in Django REST)

- Added a serializer with field validation and foreign key checks.
- Wrapped the logic in a database transaction to prevent partial saves.
- Returned clear, user-friendly error messages for invalid input.

Part 2: Database Schema Design

Entity	Description
Company	Represents a customer using the platform
Warehouse	Belongs to a company; holds inventory
Product	Includes SKU, price, and optional supplier
Inventory	Tracks product quantity per warehouse
InventoryChangeLog	Records inventory movement (sales, restock, etc.)
Supplier	External vendor providing products
Bundle	Composite product made of other products

Questions I'd Ask the Product Team

StockFlow Inventory Management System - Case Study Submission

- Can a product exist without any stock?
- Do all inventory changes require a reason (audit trail)?
- Will thresholds be managed by product, warehouse, or both?

Design Decisions

- Used unique_together to prevent duplicate inventory entries for the same product/warehouse.
- Tracked inventory changes in a dedicated log table (InventoryChangeLog) for flexibility and analytics.
- Used decimal fields for prices to ensure currency accuracy.
- Included low_stock_threshold on the Product model to support alerts.

Part 3: Low-Stock Alert API - GET /api/companies/{company_id}/alerts/low-stock

Business Rules Implemented

- Only returns products below their threshold.
- Only includes products with recent sales activity.

Assumptions

- Average daily sales is calculated over a fixed 30-day window.
- Products with zero recent sales are skipped (to reduce noise).

Final Thoughts

This case study was a great opportunity to think beyond just "making it work" and focus on:

- Designing for data integrity and validation.
- Structuring APIs around real business logic.
- Thinking through missing requirements and assumptions.

I'm happy to walk through any part of this in more detail during the live session.