Backend Engineering Intern Case Study - Theoretical Q&A

Part 1: Code Review & Debugging

Q1. What issues are in the original code?
- No validation of input fields (e.g., missing keys raise KeyError).
- SKU uniqueness is not enforced.
- Product is tied to a single warehouse, but should support multiple.
- No transaction handling (partial writes possible).
- No error handling (e.g., for IntegrityError).
- Assumes all fields are always present and valid.
- Price may require Decimal, but float is used directly.
- No check for existing warehouse_id.

Q2. What is the impact?
- API may crash with missing or bad data.
- Duplicate SKUs can corrupt product catalog.
- Incomplete data if second commit (inventory) fails.
- Violates business rule: product can exist in multiple warehouses.

Q3. Fixes Made:
- Added validation of required fields.
- Checked for existing SKU before inserting.
- Used SQLAlchemy transaction block.
- Converted price safely to Decimal.
- Created inventory only after product commit.
- Handled exceptions and returned proper HTTP status codes.

Part 2: Database Design

Tables Designed:
1. Companies: id, name
2. Warehouses: id, name, location, company_id (FK)
3. Products: id, name, sku (unique), price, threshold
4. Inventory: id, product_id (FK), warehouse_id (FK), quantity, last_updated

5. Suppliers: id, name, contact_email
6. SupplierProduct: supplier_id, product_id (Composite PK)
7. Sales: id, product_id, warehouse_id, quantity, sale_date
8. ProductBundles: id, bundle_name
9. BundleItems: bundle_id, product_id, quantity_in_bundle

Missing Requirement Questions:
- What defines recent sales? (time window?)
- Can bundles include other bundles?
- Should we track inventory movements as a log?
- How often should alerts be generated?

Design Decisions:
- Used foreign keys for relational integrity.
- Indexed product_id, sku, warehouse_id for performance.
- Normalized structure to support many-to-many (suppliers/products).

Part 3: Low Stock Alert API

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

Business Rules Implemented:
- Only products with recent sales (last 30 days) are considered.
- Inventory quantity across warehouses is checked against thresholds.
- Average daily sales is used to estimate stock-out duration.
- Supplier info is embedded for quick reorder.

Edge Cases Handled:
- No inventory: returns null safely.
- No sales: skips product (no recent activity).
- No supplier: supplier object is set to null.
- Division by zero in stock-out calculation is handled.

Assumptions Made:

Backend Engineering Intern Case Study - Theoretical Q&A

- Recent sales = last 30 days.
- One supplier per product (for simplicity).
- Inventory managed per warehouse.
- Threshold is stored per product, not per warehouse.