Technical Challenge: Salvit Advisors SKU Management Platform

Context

Salvit Advisors is transitioning from fragmented, spreadsheet-based workflows to a centralized, web-based SKU Management Platform. Because our clients manage thousands of SKUs across multiple brands, we need a "pro-level" interface that solves the problem of **scale**. The goal is to move away from editing items one-by-one and toward a powerful, rule-based configuration system.

In the base repository https://github.com/SalvitCamiloZC/frontend-senior-challenge, you are given some pre-defined styles under the `styles` folder and `mock_data.json`. Every component you build needs to have the look and feel of the given styles, and use `mock_data.json` (see data structure section below) as the base data for the main table.

The Goal

Build a **Dynamic SKU Management & Rule Engine** that allows users to manage product catalogs across different e-commerce companies. You must demonstrate how a user can efficiently manage thousands of SKUs using **Bulk Operations** and a **Centralized Rule Builder**.

1. Requirements & Business Logic

Based on `mock_data.json`, you will create an interface with the following requirements:

A. Multi-Tenant

- **Company Switcher:** Toggle between different e-commerce companies. Changing the company must refresh the entire data context.
- **Data Table:** An interactive table capable of handling many SKUs.
  - **Filtering:** The ability to filter SKUs by Category or Title.
  - **Multi-Select:** Implement checkboxes to select multiple SKUs for bulk editing.

B. The Rule Builder & Hierarchical Resolution

Instead of just editing a single SKU, the user needs to define logic that applies at scale. The primary goal of the interface is to set the monetary cost of SKUs.

To achieve this, cost rules must be defined via a rule builder.

- **The Rule Builder:** Build an interface to create a cost rule with different levels:
    - **Category-Level:** The cost applies to all SKUs within a specific category.
    - **Title-Level:** The cost applies to all SKUs sharing a Title.
    - **Individual SKU:** The cost applies specifically to one SKU.

  Each rule can have an optional `startDate` and `endDate`. A rule with dates always overrides a "General" rule (no dates) at the same level.

  The SKU is the unique identifier and is strictly non-editable. It serves as the base ID for all logic.

Hierarchical Resolution: Having different levels of cost rules implies that a given SKU can have overlapping costs, conflicting. You must solve the rules conflicts based on this priority:

- *SKU Rule > Title Rule > Category Rule.*

The main table must display '**Today's effective cost,**' representing the SKU cost for the current date based on the winning rule.

C. Bulk Actions & UX

- **Bulk Edit Bar:** When multiple SKUs are selected, a "Bulk Action" menu should appear (e.g., "Edit Category," "Edit Title," or "Create Cost Rule"). Creating a cost rule for multiple SKUs at the same time should create one SKU-level rule for each.
- **Status Badges:** Clearly indicate for each SKU if its cost is "Inherited" (from Category/Title rule), "Resolved" (Specific SKU rule exists), or “Missing Cost” if there’s no cost defined for the SKU.
- **Detail Drawer:** Clicking a SKU opens a drawer to view its metadata, add and edit rules, and see its specific **Resolution Trace** (showing exactly which rule is currently defining its cost and why).
- **Persistence:** For the purpose of this test, all changes should be **saved locally (State/LocalStorage) or simulated**. No external database connection is required.

2. Tech Stack & Tools

- **Framework:** React.js
- **State Management:** Redux.
- **Styling:** Tailwind CSS (Following the provided Salvit Brand System).

- **Leverage AI:** You are **expected** to use AI tools (Antigravity, Cursor, Copilot, Codex, etc.) to accelerate your workflow. We are evaluating your ability to prompt, architect, and validate AI-generated code.

3. Styling Guidelines

Please use the following brand constants in your Tailwind implementation:

- **Primary Blue:** `#002D72` (`brand-blue`) — Use for headers and authority.
- **CTA Orange:** `#F47C1F` (`brand-orange`) — Use for primary actions.
- **Accent Indigo:** `#4f46e5` (`brand-indigo`) — Use for data highlights.
- **Fonts:** `Barlow` for headings, `Inter` for UI/Body.

4. Data Structure & Relationships

The application relies on mock_data.json to simulate a relational database. Understanding these relationships is key to building the "Company Switcher" and the "Rule Builder" correctly.

Entities Breakdown:

- **clients**: Represents the tenants (e-commerce companies).
  - Acts as the root filter. All other entities (products, titles, categories, rules) are scoped by client_id.
- **categories**: The highest level of product grouping.
- **titles**: A mid-level grouping (e.g., "Smartphones Series A").
  - **Relation:** Belongs to a Category (categoryId).
- **products**: The individual SKUs (Stock Keeping Units).
  - **Relation:** Belongs to a Title (titleId) and a Category (categoryId).
- **cogs_rules**: The core logic entity. Contains the cost rules already active in the system.
  - **Polymorphic Relation:** The rule applies to a target based on targetType ("CATEGORY", "TITLE", or "SKU").
  - targetId: Corresponds to the ID of the category, title, or SKU depending on the type.
  - value: The monetary cost.
  - startDate / endDate: Defines the validity period. If null, it is a "General" rule (lower priority than a dated rule).

Entity Hierarchy for Resolution:
Client -> Category -> Title -> SKU