Product Creation Form & API Usage - Developer Instructions

This document provides **clear backend-agnostic guidance** for frontend developers on how to implement a product creation form and interact with APIs. The goal is to **avoid sharing code** and instead define a step-by-step integration protocol.

📋 Product Form Requirements

1. Basic Fields (Mandatory)

Developers must create input fields for the following:

- Product Name (text)
- SKU (text)
- Description (multiline/textarea)
- Purchase Price (number)
- Selling Price (number)
- Stock Quantity (number)
- Low Stock Alert (number)

2. Dropdown Selections (Dynamic)

Populate dropdowns using API data:

- Category (required)
- Brand (required)
- Unit (required)
- Tax Group (optional)

Use the respective GET APIs to populate options.

3. Price Groups Section (Dynamic)

- For each price group (loaded from API), show:
- Price input (number)
- Discount Percent input (number)
- Placeholder for price input: default selling price
- Send only filled-in price groups when submitting to backend

4. Inventory by Store (Multiple Entries)

- Enable dynamic entry of store-level inventory
- Each row should include:

- Store (dropdown from API)
- Quantity
- Min Stock
- Cost Price
- Allow adding/removing store rows

5. Status Toggle

- Boolean switch (Active/Inactive)

🕐API Usage Instructions

1. Load Master Data (on page load)

Call all these APIs to populate dropdowns and dynamic sections:

Data Type	Endpoint
Categories	`GET /api/category?limit=100`
Brands	`GET /api/brand?limit=100`
Units	`GET /api/unit?limit=100`
Tax Groups	`GET /api/tax-group/active`
Stores	`GET /api/store?limit=100`
Price Groups	`GET /api/product/creation-data`

Use the `creation-data` API to fetch price group definitions and defaults.

2. Create Product (on form submission)

Submit form data using:

POST `/api/product`

Body structure:

```
{
  "name": "Product A",
  "sku": "SKU123",
  "description": "Product details...",
  "purchasePrice": 50,
  "sellingPrice": 80,
  "stock": 100,
```

```
  "lowStockAlert": 5,
  "categoryId": "cat123",
  "brandId": "brand456",
  "unitId": "unit789",
  "taxGroupId": "tax123",  // Optional
  "status": true,
  "priceGroups": [
    { "priceGroupId": "pg1", "price": 75, "discountPercent": 5 },
    { "priceGroupId": "pg2", "price": 70, "discountPercent": 10 }
  ],
  "stores": [
    { "storeId": "store1", "quantity": 50, "minStock": 10, "costPrice": 48 },
    { "storeId": "store2", "quantity": 50, "minStock": 5, "costPrice": 49 }
  ]
}
```

Notes:

- Send only **price groups with values**
- Stores can be empty (optional)

🔗Developer Checklist

Before Submission:

-

After Submission:

- Show success message
- Reset form or redirect to product list

Error Handling:

- Catch and display API errors
- Handle 401/403 for auth failures

🔧Backend Support Expectations

- API returns all necessary master data
- Accepts and validates full product payload
- Auto-applies default prices where missing
- Sends useful error responses for bad input

Let me know if you need a similar instruction set for **editing products** or **billing/invoicing features**.