🛍️ Problem Statement: Shop-Site Application

📖 Application Overview

You are tasked with building the **logic** for a simple **Shop-Site Application**.

The application allows users to:

1. Select a **product category** (Fruits, Electronics, Clothing) from the homepage.
2. View products belonging to that category.
3. **Sort products** (by name or price).
4. **Filter products** by brand.
5. Add products to a **Shopping Cart**.
6. View their cart in a table at the bottom of the page.

📂 Provided Files

- `index.html` → Displays 3 categories: Fruits, Electronics, Clothing.
- `products.html` → Displays the selected category's products and the Cart.
- `styles.css` → Provides all styling.
- `products.js` → Contains the list of products (with name, category, brand, price, image).
- `app.js` → 🚩 *This is the only file you will edit.*

⚠️ Rules

- **Do not** modify `index.html`, `products.html`, `styles.css`, or `products.js`.
- Implement all logic in `app.js`.
- Use the given **IDs** and **CSS class names** to target elements.
- Keep the DOM structure consistent so automated tests and the UI function correctly.

📝 Requirements (To-Do in `app.js`)

1. Navigation

- Function: `goToProducts(category)`
- Called when a category (`.category`) is clicked on the homepage.
- Redirect to:

```
products.html?category=<CategoryName>
```

2. Rendering Products

- Function: `renderProducts(list)`

- Input: A list of product objects.
- Output: Render **product cards** inside:
  - **Container ID**: `#product-list`
- Each product card (`.item`) must contain:
  - Product image (from `/images/<product.image>`)
  - Product name
  - Product brand
  - Product price
  - A button labeled **"Add to Cart"** which calls `addToCart(index)`

3. Sorting

- Function: `sortProducts()`
- Dropdown: `#sort-select`
- Options:
  - `"default"` → Reset to original dataset order.
  - `"name"` → Sort alphabetically (A–Z).
  - `"price-asc"` → Sort by price (Low → High).
  - `"price-desc"` → Sort by price (High → Low).
- After sorting, call `renderProducts(filteredByCategory)` .

4. Filtering

- Function: `filterProducts()`
- Dropdown: `#filter-select`
- Options:
  - `"all"` → Show all products in the selected category.
  - Any brand name → Show only products of that brand.
- After filtering, call `renderProducts(filteredByCategory)` .

5. Cart Management

- Function: `addToCart(index)`
  - Add product from `filteredByCategory[index]` to a **cart array**.
  - If the product already exists, increase its quantity.
  - Otherwise, add it with `quantity = 1` .
  - Call `renderCart()` .
- Function: `renderCart()`
  - Cart table body: `#cart-items`
  - Each row (`<tr>`) must include:
    - Product image (thumbnail ~60px width)
    - Product name
    - Product brand
    - Price per item
    - Quantity

  - Total (price × quantity)

6. Initialization

- On `products.html` page load:
  - Read `category` from query string:

    ```
    const params = new URLSearchParams(window.location.search); const category =
    params.get("category");
    ```

  - Update page title: `#category-title` → `<CategoryName> Products`
  - Filter `products[]` (from `products.js`) by category → assign to `filteredByCategory`.
  - Call `renderProducts(filteredByCategory)`.

✅ Summary of Required Selectors

- Category cards: `.category` (index.html)
- Product list container: `#product-list` (products.html)
- Each product card: `.item` (rendered dynamically)
- Sort dropdown: `#sort-select` (products.html)
- Filter dropdown: `#filter-select` (products.html)
- Cart table body: `#cart-items` (products.html)
- Page title: `#category-title` (products.html)

🎯 Goal

After implementing `app.js`, the Shop-Site Application should behave like a mini shopping platform where:

- Categories navigate correctly.
- Products display properly with image, brand, and price.
- Sorting works (Name, Price Low → High, Price High → Low).
- Filtering works by brand.
- Users can add products to a Cart table, with correct quantities and totals.