Specscart.

Title: Advanced Product Catalog with Faceted Search & Filtering

1.Introduction

A robust and intuitive product catalog is the backbone of any e-commerce platform. For a senior-level candidate, merely displaying products isn't enough; we expect sophisticated features like advanced search, dynamic filtering (faceted navigation), and performance optimisation for growing datasets. This assignment challenges you to build a comprehensive product browsing experience that can handle a moderate number of products efficiently, providing a rich user experience.

We are looking for a demonstration of your ability to design flexible data models, implement efficient search and filtering algorithms, optimise for performance, and craft a highly interactive user interface.

2. Core Problem Statement

Design and implement a complete e-commerce application focused on a **feature-rich product catalog with advanced faceted search and filtering capabilities**. The system should allow users to easily discover products by combining text search with various aflribute filters.

3. Functional Requirements

Your solution should cover the following core functionalities:

3.1. Frontend (User Interface):

- **Product Listing Page:**
  - Display products in a visually appealing grid or list view. Each product card should show at least product image, name, and price.
  - Implement **pagination** to handle a large number of products efficiently. For example, display 12-24 products per page, with clear navigation for "Next Page," "Previous Page," or page numbers.
  - Implement **sorting options** that can be applied to the current filtered/searched results (e.g., by price ascending/descending, by name A-Z/Z-A, by "relevance" if search is active).
- **Faceted Navigation (Filters):**
  - A prominent sidebar or section for dynamic filters based on various product aflributes (facets). Implement at least **4-5 diverse facet types**, including:

- **Categories:** (e.g., "Electronics," "Apparel," "Books," "Home & Kitchen"). Should support multi-selection (users can select "Electronics" AND "Home & Kitchen").
- **Brands:** (e.g., "Sony," "Samsung," "Nike," "HP"). Should support multi-selection.
- **Price Range:** Implement using a slider or predefined ranges (e.g., "$0-50", "$50-100", "Above $100").
- **Other Attribute:** Choose at least one product-specific aflribute (e.g., "Color," "Size," "Material," "Memory (GB)") that supports multi-selection for discrete values.

- **Dynamic Filter Options:** The available filter options for *each* facet, and the *count* of products matching that option, should reflect the **current search/filter context**. For example, if "Electronics" is selected, the "Brand" filter should only show brands of electronic products and their respective counts within that filtered set.
- Filters should dynamically update the product list *without full page reloads*. Use AJAX/ Fetch API calls for a smooth user experience.

- **Search Bar:**
  - A prominent search bar allowing users to enter keywords to search across product names and descriptions.
  - Search results should dynamically update as the user types (implement **debouncing** to prevent excessive API calls) or after submifling the search query.
- **Active Filters Display:**
  - Clearly display the currently applied filters (e.g., "Filters: Category: Electronics, Brand: Sony, Price: $100-$500").
  - Provide an easy way to **remove individual filters** or **clear all applied filters** with a single click.
- **URL Management (Deep Linking):**
  - The current search terms, filters, and pagination/sorting parameters **must be reflected in the URL query string** (e.g., /?search=laptop&category=electronics,computers&brand=dell&price_min=500&sort=price_asc). This enables users to share specific search results and allows for browser history navigation.
- **Product Detail Page (Simplified):** A basic page showing more details of a single product when clicked from the listing page (can be a simplified static page or fetch additional details from backend).

3.2. Backend (APIs & Data):

- **Product Data Model:**
  - Design a flexible and extensible data model for products in your chosen database. Each product should have at least:
    - Unique ID
    - Name (string)
    - Description (string)
    - Price (number)
    - Image URL (string)
    - **Multiple Categories:** A product can belong to one or more categories (e.g., a laptop can be in "Electronics" and "Computers").
    - **Brand:** (string)
    - **Arbitrary/Dynamic Attributes:** A way to store diverse aflributes specific to product types without rigid schema changes (e.g., "Screen Size" for electronics, "Material" for apparel, "Weight" for groceries). This could be implemented using:
      - A JSONB field (PostgreSQL, MongoDB)
      - Separate key-value pairs table (EAV model) if relational database without JSON support.
      - A flexible document structure (NoSQL).
    - (Optional): Stock level, average rating.
  - **Mock Data Generation:** You are expected to generate a **mock dataset of 500-1000 products** to adequately test filtering and search performance. This data should be diverse and include various categories, brands, price ranges, and values for other aflributes you choose.
- **Product Retrieval API:**
  - A **single primary API endpoint** (e.g., GET /products) that supports:
    - **Full-text search:** Querying product names and descriptions by keyword. Case-insensitivity should be supported.
    - **Multi-faceted filtering:** Filtering by multiple selected categories, brands, price ranges (min/max), and other aflributes simultaneously.
    - **Pagination:** limit and offset (or page and pageSize) parameters to retrieve subsets of results.
    - **Sorting:** sort_by (e.g., price, name, relevance) and sort_order (asc, desc) parameters.
  - The API response should include:
    - The list of products matching the applied criteria.
    - The **total count of products** matching the current filters/search (before pagination).
    - **Metadata for dynamic facets:** For each facet (category, brand, etc.), return the available options and the **count of products for each option within the *current filtered result set***. This is crucial for the frontend to display accurate counts next to filter checkboxes. Example: If 100 electronics products are shown, and 50 are "Sony" and 30 are "Dell", the "Brand" facet should reflect this, even if other filters are applied.

- **Database/Persistence:**
  - Use a relational database (e.g., SQLite, PostgreSQL, MySQL) that can support efficient querying for search and filtering.
  - **Indexing:** Clearly discuss or implement appropriate database indexes to optimise search and filter performance. Explain your indexing strategy.
  - **Search Engine (Highly Valued - Optional):** For a truly advanced solution, consider integrating a dedicated search engine like Elasticsearch or Apache Solr for the full-text search and faceted navigation aspects. If you choose this, justify the decision and explain the simplified setup (e.g., using a client library). A strong, optimised database query-based solution is also perfectly acceptable if you can demonstrate performance.

4. Architectural & Design Considerations

- **Data Modeling:** Focus on a flexible schema that can adapt to new product aflributes and new filter types in the future without constant database migrations.
  Discuss your approach to storing product categories and arbitrary aflributes.
- **Query Optimisation:** How will your backend efficiently handle complex queries involving multiple filters (AND logic between different facets, OR logic within a single multi-selected facet) and full-text search on a growing dataset? Consider:
  - Appropriate database indexing for frequently queried fields.
  - Efficient join operations if using relational schema.
  - Strategies for full-text search (e.g., LIKE, full-text search extensions, or a dedicated search engine).
  - How to efficiently calculate facet counts dynamically based on the current filtered set.
- **API Design:** Design a well-structured RESTful API that is intuitive, versioned (even if conceptually), and efficiently supports all required parameters (search, filter, pagination, sort).
- **Frontend State Management:** How will the frontend manage the application's state (current filters, search terms, pagination, sorting)? Consider using a state management library (e.g., Redux, Zustand, Vuex, React Context) or demonstrating clear paflerns for managing local component state.
- **User Experience (UX) & Responsiveness:** Ensure a smooth and responsive filtering and search experience. Avoid unnecessary loading states or flickering.Consider basic responsive design principles for different screen sizes.
- **Performance:** Aim for quick response times for search and filter operations, especially with the mock dataset.
- **Technology Stack:** You have full freedom to choose your preferred languages and frameworks (e.g., Node.js/Express, Python/Flask/Django, Java/Spring Boot, Go, .NET Core for backend; React, Angular, Vue, plain JS for frontend). Briefly justify your choices if they are non-standard.

5. Deliverables

Please submit the following:

1. **Source Code:**
   - A well-organised and commented codebase for both frontend and backend components.
   - Ensure all necessary configuration files (e.g., package.json, requirements.txt, pom.xml) are present.
   - Include scripts or instructions for seeding your mock product data into the database.

2. **README.md File:** This file is crucial for evaluation. It must include:
   - **Project Title and Overview:** Briefly describe the project and its purpose.
   - **Setup Instructions:** Clear, step-by-step instructions on how to set up, install dependencies, and run your application (including database setup, data seeding, and how to start frontend/backend).
   - **Technology Stack:** List all languages, frameworks, libraries, and tools used for both frontend and backend.
   - **Data Model Design:**
     - Provide your product data model schema (e.g., SQL DDL, JSON schema description).
     - Explain your approach to storing and querying product categories and arbitrary aflributes, especially how it supports the dynamic filtering requirements.
   - **Search & Filtering Logic:**
     - A detailed explanation of how your backend implements full-text search, multi-faceted filtering (including logic for combining filters), pagination, and sorting.
     - Discuss your approach to calculating dynamic facet counts for the frontend.
     - Crucially, detail your **query optimization strategies** (e.g., specific database indexing, full-text search features, query paflerns) and why you chose them. If you used an external search engine, explain its integration.
   - **Design Choices & Trade-offs:**
     - Explain any significant architectural or design decisions made (e.g., choice of database, frontend state management, server-side vs. client-side filtering).
     - Discuss the reasons behind these choices and any known limitations of your current implementation or areas for future improvement (e.g., advanced relevance scoring, caching strategies).
   - **Demonstration Steps:** Clear steps on how we can easily test the search, filtering, pagination, and sorting functionalities. Provide example search terms and filter combinations that showcase the dynamic nature of the filters.
3. **(Optional but Recommended):** A short video (2-5 minutes) demonstrating the application's functionality, showcasing the search, filtering (with dynamic facet updates), pagination, and URL reflection.

6. Evaluation Criteria

Your submission will be evaluated on the following aspects:

- **Logical Reasoning & Problem Solving:**
  - How effectively you design and implement the complex search and multi-faceted filtering logic.
  - Your approach to handling dynamic facet counts based on the current filtered set.
- **Data Modeling:**
  - Flexibility, scalability, and efficiency of your product data model for a growing catalog and diverse aflributes.
- **Performance Optimisation:**
  - Evidence of strong consideration for query performance and efficient data retrieval, especially with a larger dataset.
  - Appropriate use of indexing or external search technologies.
- **Code Quality:**
  - Readability, maintainability, and organisation of your codebase.
  - Adherence to best practices for the chosen language/framework.
  - Proper use of comments and consistent coding style.
- **Technical Proficiency:**
  - Effective use and understanding of chosen technologies (e.g., database features, frontend reactive paflerns, search libraries).
- **Documentation:**
  - Clarity, completeness, and accuracy of your README.md file.
  - Ease of sefling up and running the application based on your instructions.

Good luck! We look forward to seeing your comprehensive solution.