**Sales Chatbot**

**Objective:**

The objective of this project is to create a robust application that allows users to search, filter, and browse a list of products based on specific criteria such as name, price range, rating, and stock availability. The application is developed with Django as the backend and ReactJS for the frontend, providing a seamless and user-friendly interface for querying and filtering products.

**Technology Stack:**

**Backend:**

* **Framework:** Django (with Django REST Framework)
* **Database:** SQLite
* **Programming Language:** Python
* **Key Libraries:**
  + Django REST Framework: To build API endpoints.
  + Django ORM: For querying and managing data in the database.

**Frontend:**

* **Framework:** ReactJS
* **State Management:** React Hooks (useState, useEffect)
* **Key Libraries:**
  + Axios: For making API requests.
  + CSS: For styling components and providing a clean user interface.

**Features:**

**Backend Features:**

1. **API Endpoint:**  
   A single API endpoint (/api/products) handles the product search and filtering.
2. **Search Functionality:**
   * Users can search for products by name or description using a query string.
   * Implements case-insensitive search using icontains.
3. **Filtering Options:**
   * **Category Filter:** Filter products by category.
   * **Price Range Filter:** Supports minimum and maximum price filtering.
   * **Rating Filter:** Filter products by a minimum rating.
   * **Stock Availability:** Filter based on stock status (in-stock or out-of-stock).
4. **Error Handling:**
   * Graceful handling of invalid or missing query parameters.
5. **Serialization:**
   * The Response object returns product data as a JSON payload with fields such as id, name, price, category, rating, and stock.

**Frontend Features:**

1. **Search and Filter Interface:**
   * Users can search for products using an input box.
   * Dropdowns for filtering by price, rating, and stock availability.
2. **Dynamic Filtering:**
   * The filterProducts function dynamically filters the product list based on selected criteria.
3. **API Integration:**
   * Uses Axios to fetch data from the backend API endpoint.
4. **Real-Time Updates:**
   * The product list updates dynamically as users modify search or filter inputs.
5. **Error and Loading States:**
   * Displays error messages if the API request fails.
   * Shows a loading message while fetching data.
6. **Responsive Design:**
   * The interface is styled with CSS for responsiveness and an intuitive user experience.

**Workflow:**

1. **Backend Workflow:**
   * The user sends a GET request to the /api/products endpoint with query parameters such as q (query), category, min\_price, max\_price, min\_rating, and in\_stock.
   * The API processes the request and applies filters using Django ORM.
   * A JSON response is returned with a list of filtered products.
2. **Frontend Workflow:**
   * The application fetches the product list from the backend API using Axios.
   * Filters are applied locally in the React component for real-time updates.
   * Filtered products are displayed in a styled and user-friendly list.

**Advantages:**

* **Efficient Searching and Filtering:** Combines backend and frontend filtering to optimize performance and improve user experience.
* **Scalability:** The backend is designed to handle a growing database of products.
* **User-Friendly Interface:** Simple and intuitive design makes it easy for users to interact with the application.
* **Dynamic and Interactive:** Real-time updates ensure users see the most relevant results.

**Challenges:**

1. **Handling Large Data:** Managing a large database of products requires efficient querying and optimization of Django ORM filters.
2. **Dynamic Filtering:** Implementing frontend and backend filtering together while ensuring consistent results.
3. **Error Handling:** Ensuring all possible edge cases, such as invalid inputs, are gracefully handled.

**Future Enhancements:**

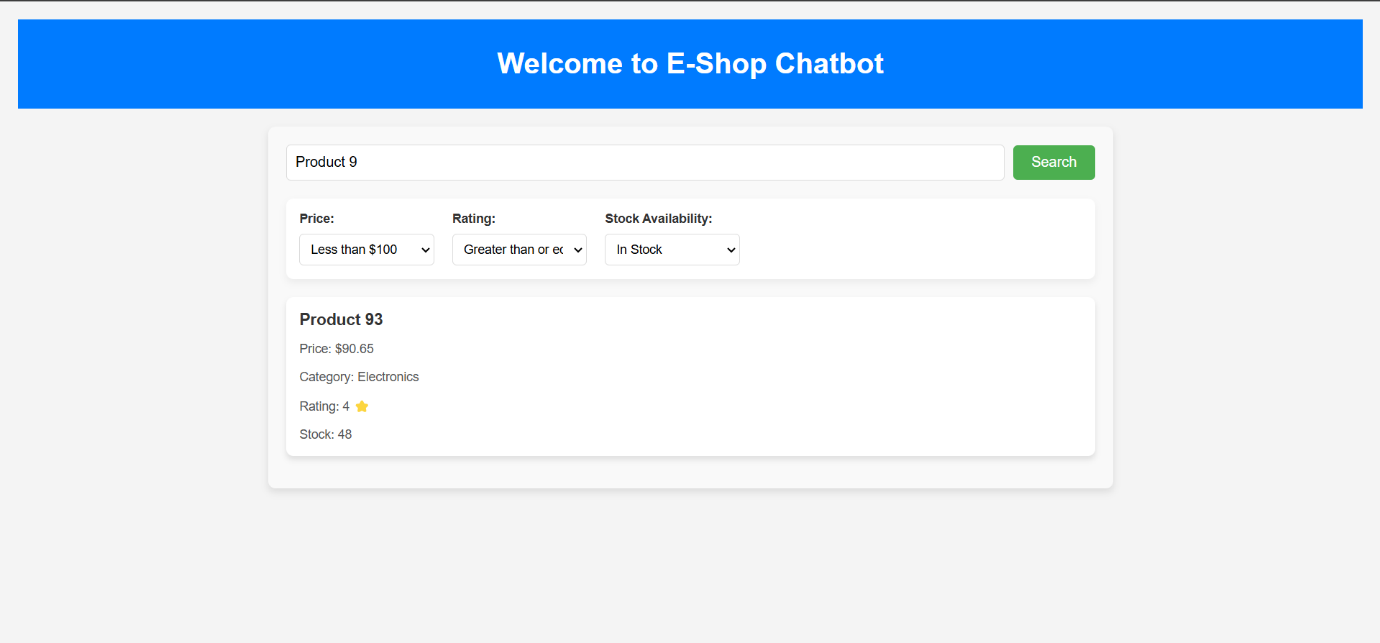
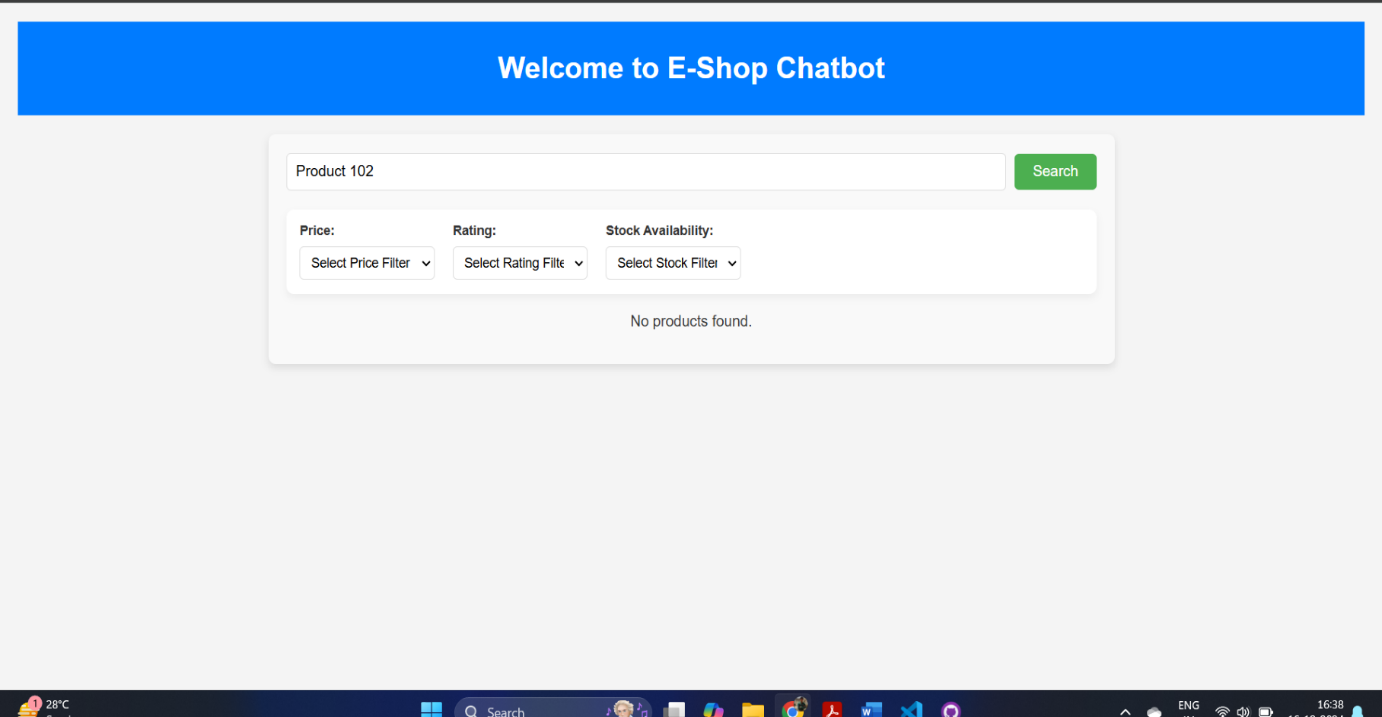
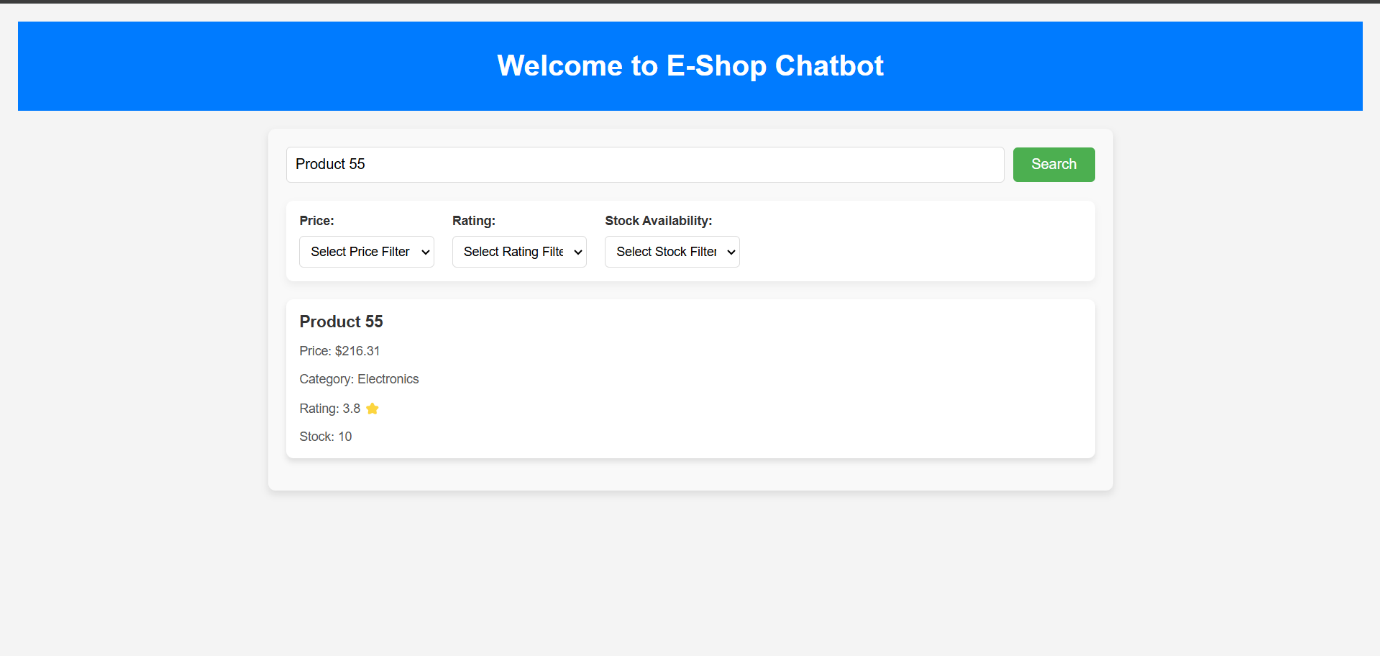
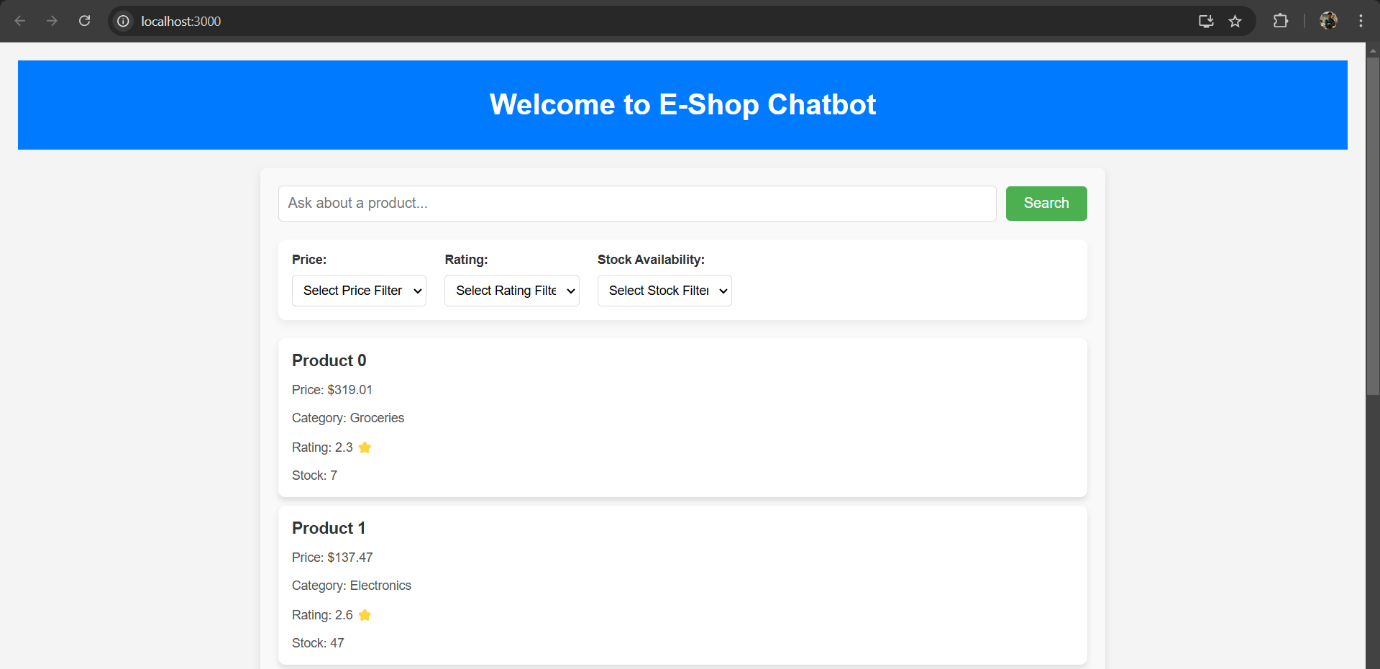
1. **Advanced Filters:**
   * Add filters for more attributes like brand, date added, and discount.
   * Allow combining multiple filters (e.g., price + rating + category).
2. **Pagination:**
   * Implement pagination in the API and frontend to handle large datasets efficiently.
3. **Sorting Options:**
   * Add sorting options for price, rating, and popularity.
4. **Enhanced UI/UX:**
   * Use Material-UI or Bootstrap for a more polished design.
5. **Authentication:**
   * Implement user authentication to allow personalized recommendations and saved filters.
6. **Recommendation System:**
   * Suggest related products based on user search history or preferences.

**Conclusion:**

This project demonstrates a comprehensive approach to building a product search and filtering application using Django and ReactJS. With robust backend APIs and an interactive frontend, the application offers an efficient solution for querying and filtering product data. Future enhancements will further improve functionality, scalability, and user experience.

**NOTE:** Screenshots are attached on next page.

**Screenshotss**

****