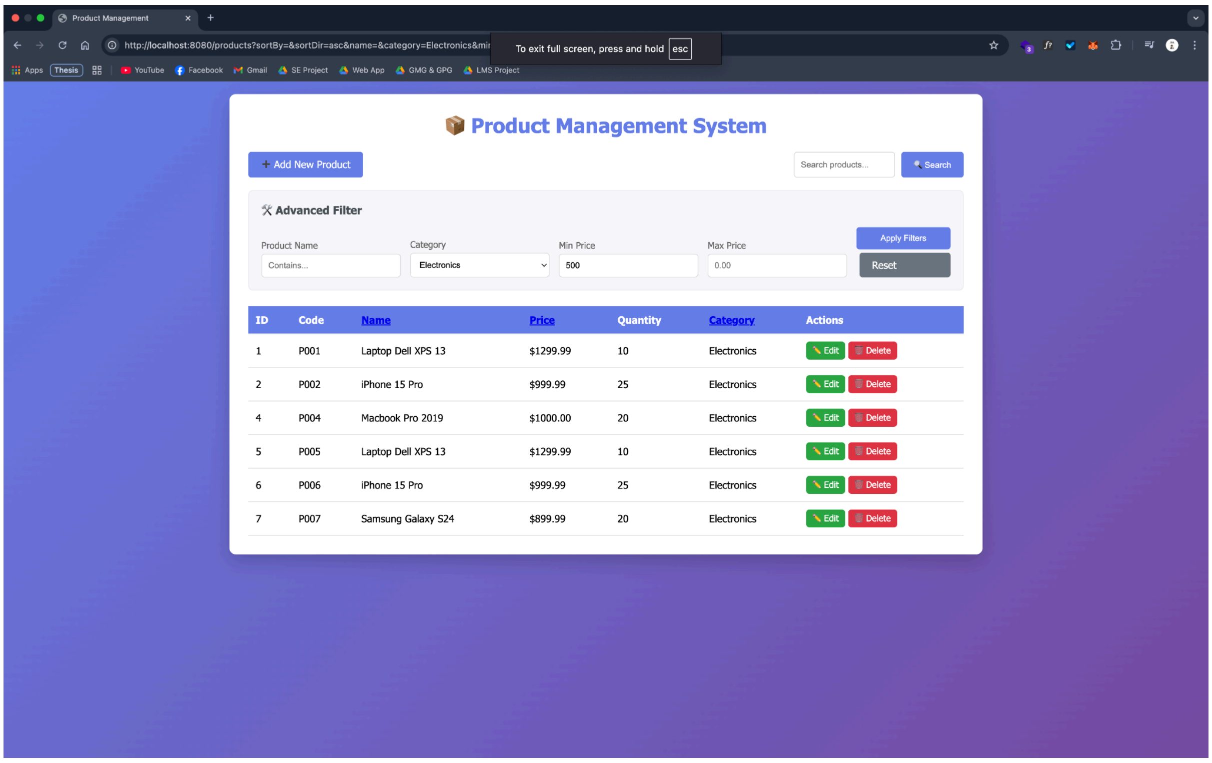
**Name:** Phạm Gia Phúc

**ID:** ITCSIU22178

# Web App Development Laboratory - Lab 7 Assignment

1. **GitHub link:** https://github.com/phamgiaphuc/product-management
2. **Exercise 5: Advanced Search**


Product Management
http://localhost:8080/products?sortBy=&sortDir=asc&name=&category=Electronics&mi
To exit full screen, press and hold esc
Apps
Thesis
YouTube
Facebook
Gmail
SE Project
Web App
GMG & GPG
LMS Project

📦 Product Management System

+ Add New Product | Search products... | 🔍 Search

🛠 Advanced Filter

| Product Name | Category | Min Price | Max Price |
|---|---|---|---|
| Contains... | Electronics | 500 | 0.00 |

Apply Filters | Reset

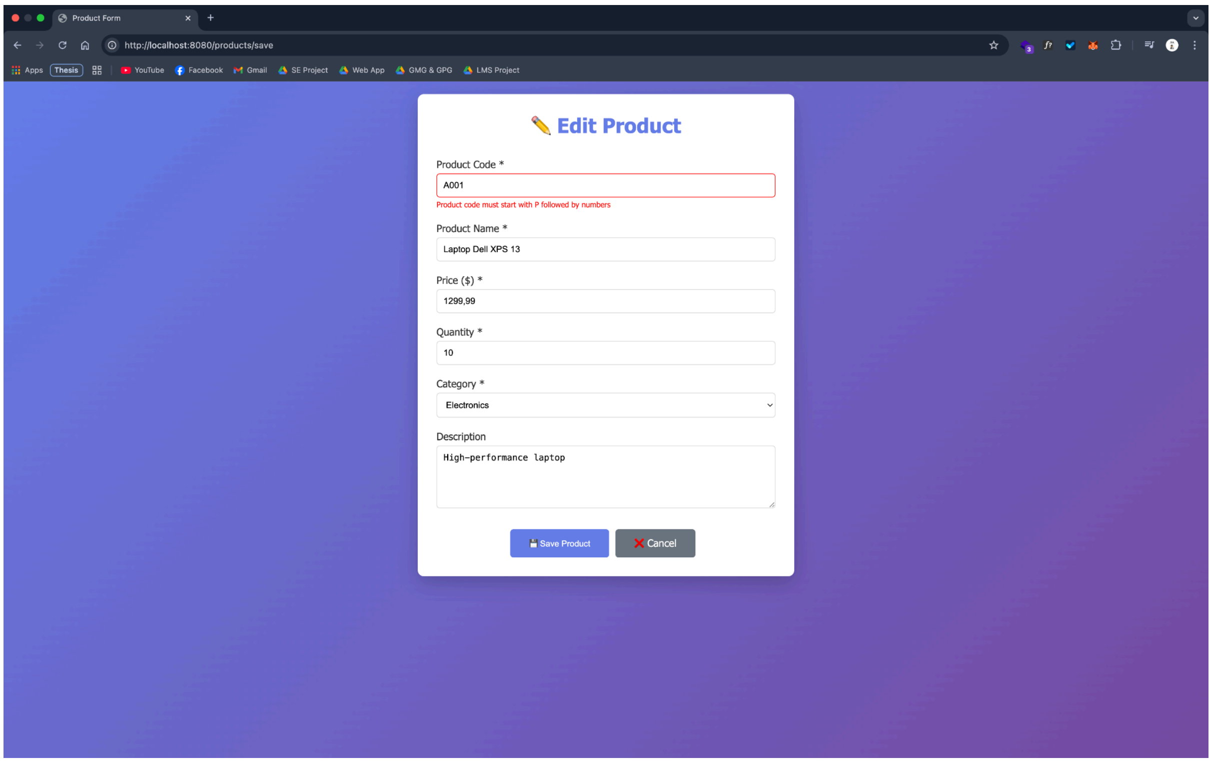
| ID | Code | Name | Price | Quantity | Category | Actions |
|---|---|---|---|---|---|---|
| 1 | P001 | Laptop Dell XPS 13 | $1299.99 | 10 | Electronics | ✏️ Edit 🗑 Delete |
| 2 | P002 | iPhone 15 Pro | $999.99 | 25 | Electronics | ✏️ Edit 🗑 Delete |
| 4 | P004 | Macbook Pro 2019 | $1000.00 | 20 | Electronics | ✏️ Edit 🗑 Delete |
| 5 | P005 | Laptop Dell XPS 13 | $1299.99 | 10 | Electronics | ✏️ Edit 🗑 Delete |
| 6 | P006 | iPhone 15 Pro | $999.99 | 25 | Electronics | ✏️ Edit 🗑 Delete |
| 7 | P007 | Samsung Galaxy S24 | $899.99 | 20 | Electronics | ✏️ Edit 🗑 Delete |

→ **Explanation:**

- **Backend Dynamic Query Construction:** The core filtering mechanism is implemented within the searchProductsAdvanced service method, which utilizes the Spring Data JPA Specification interface to construct a dynamic, type-safe SQL query at runtime via the Criteria API. Instead of relying on multiple hardcoded repository methods, this approach initializes an empty list of Predicate objects and conditionally populates it based on the user's input. The logic checks each parameter - name, category, and price range—for null values; if a parameter is present, a corresponding SQL clause is generated (e.g., a case-insensitive LIKE for names, an equality check for categories, or greaterThanOrEqualTo/lessThanOrEqualTo for price boundaries). Finally, the CriteriaBuilder combines these distinct predicates using an AND operator, ensuring that the returned results satisfy all active criteria simultaneously.

- **Request Handling and Input Sanitization:** The listProducts controller method acts as the orchestration layer, handling HTTP GET requests that may contain any combination of optional filter parameters. A critical component of this logic is input sanitization: the method explicitly checks for empty or whitespace-only strings and converts them to null before passing them to the service layer, preventing the generation of empty SQL queries. Furthermore, the controller handles sorting logic by dynamically constructing a Sort object based on the user's selection (sortBy) and direction (sortDir). To ensure a seamless user experience, the controller reinjects all search parameters back into the Model. This state persistence allows the frontend to retain the user's search criteria in the input fields after the page reloads.
- **Frontend Integration and State Preservation:** The presentation layer, implemented using Thymeleaf, integrates the search logic by creating a "sticky" form interface and context-aware navigation. The search form inputs bind to the model attributes returned by the controller, ensuring that the user's filter criteria remain visible after a search is executed. Additionally, the data table headers implement complex URL construction logic; each column header generates a link that toggles the sort direction while simultaneously embedding the current filter parameters (name, category, minPrice, maxPrice). This design ensures that sorting the table does not inadvertently reset the active search filters, allowing users to refine and organize large datasets effectively without losing their context.

## 3. Exercise 6: Validation



✏️ Edit Product

| | |
|---|---|
| Product Code * | A001 |
| | Product code must start with P followed by numbers |
| Product Name * | Laptop Dell XPS 13 |
| Price ($) * | 1299,99 |
| Quantity * | 10 |
| Category * | Electronics |
| Description | High-performance laptop |

→ **Explanation:**

- **Entity Validation Strategy:** The Product entity employs a robust validation strategy that combines Jakarta Bean Validation annotations with JPA schema constraints to ensure data integrity before persistence. String-based fields, such as name and category, utilize @NotBlank and @Size annotations to prevent the storage of empty or malformed text. The productCode field enforces strict business formatting via a Regular Expression (@Pattern), mandating that all codes begin with the prefix "P" followed by at least three integers, while simultaneously enforcing database-level uniqueness. Numerical fields are governed by logical boundaries; price relies on BigDecimal precision with @DecimalMin and @DecimalMax annotations to prevent negative or excessive values, and quantity is constrained to non-negative integers using @Min. Finally, the entity utilizes the @PrePersist lifecycle hook to automatically manage the createdAt timestamp, removing the responsibility of audit logging from the service layer.

4. **Exercise 7: Sorting and Filtering**

- **Server-Side Sorting Mechanism and State Toggling:** The sorting functionality is implemented using server-side rendering, where the application triggers a new HTTP GET request rather than relying on client-side JavaScript. This logic is embedded directly within the table header (<th>) anchor tags using Thymeleaf’s URL expression syntax (@{...}). For every sortable column, the link dynamically constructs a URL that includes a sortBy parameter targeting the specific field (e.g., 'price' or 'name'). Simultaneously, the logic determines the sort direction (sortDir) using a ternary operation: it evaluates the current sort direction and toggles it—if the current state is 'asc', the link is generated with 'desc', and vice versa. This ensures that repeated clicks on the same header cycle the list between ascending and descending order.