International University

School of Computer Science and Engineering

Web Application Development

Laboratory

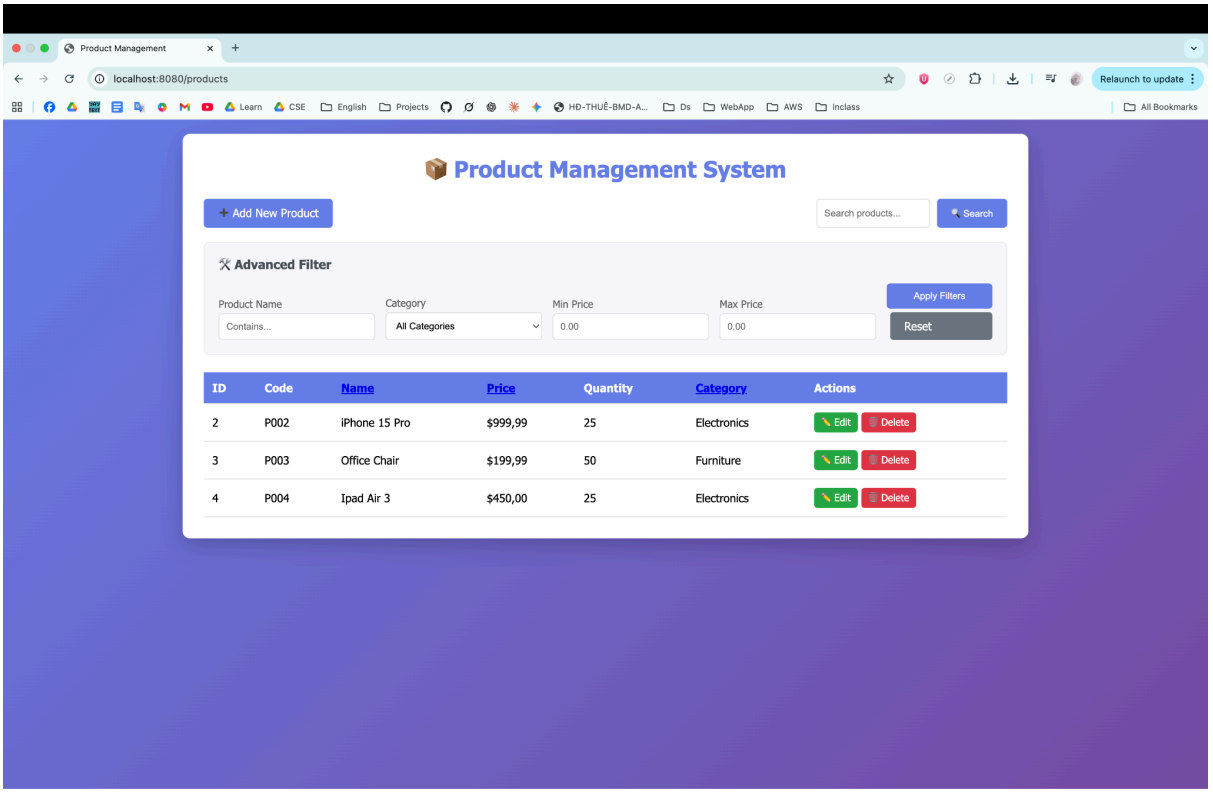
IT093IU

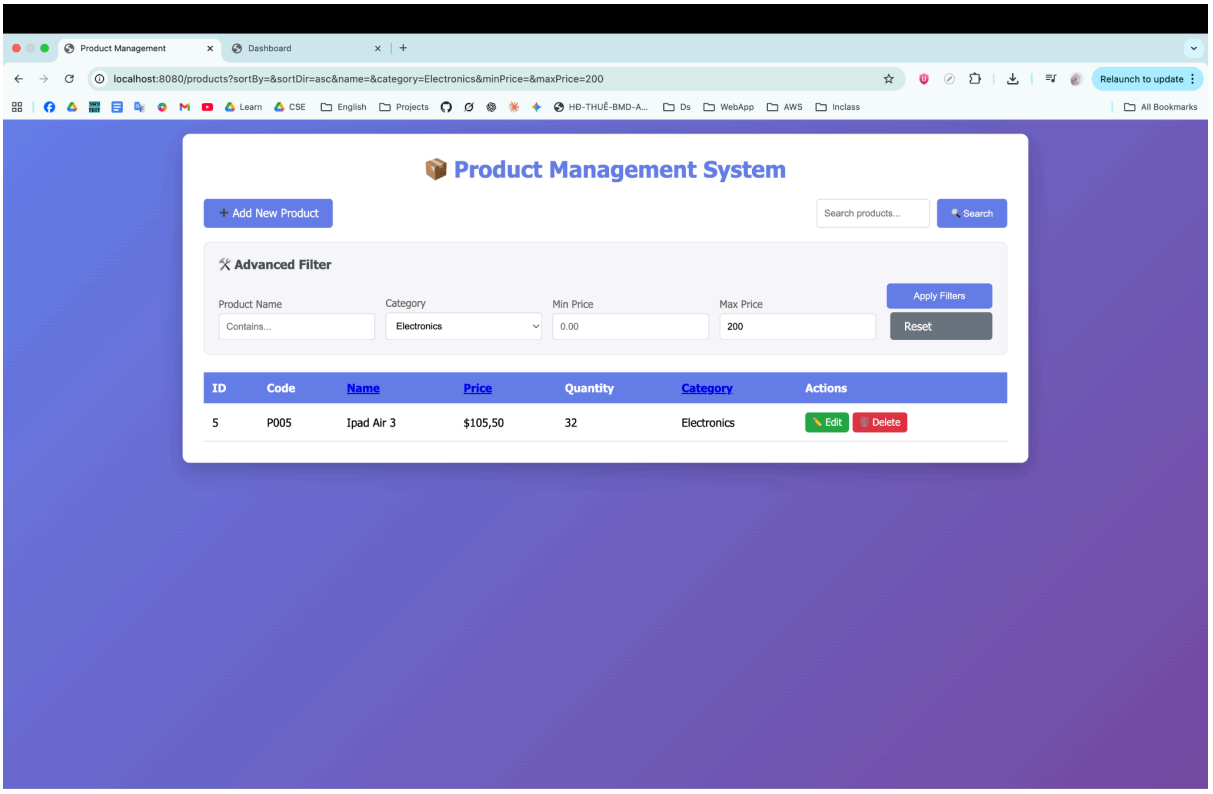
Lab #7

Submitted by

Nguyễn Hồng Ngọc Hân - ITCSIU22229

EXERCISE 5: ADVANCED SEARCH
Task 5.1: Multi-Criteria Search
Task 5.2: Category Filter
Task 5.3: Search with Pagination





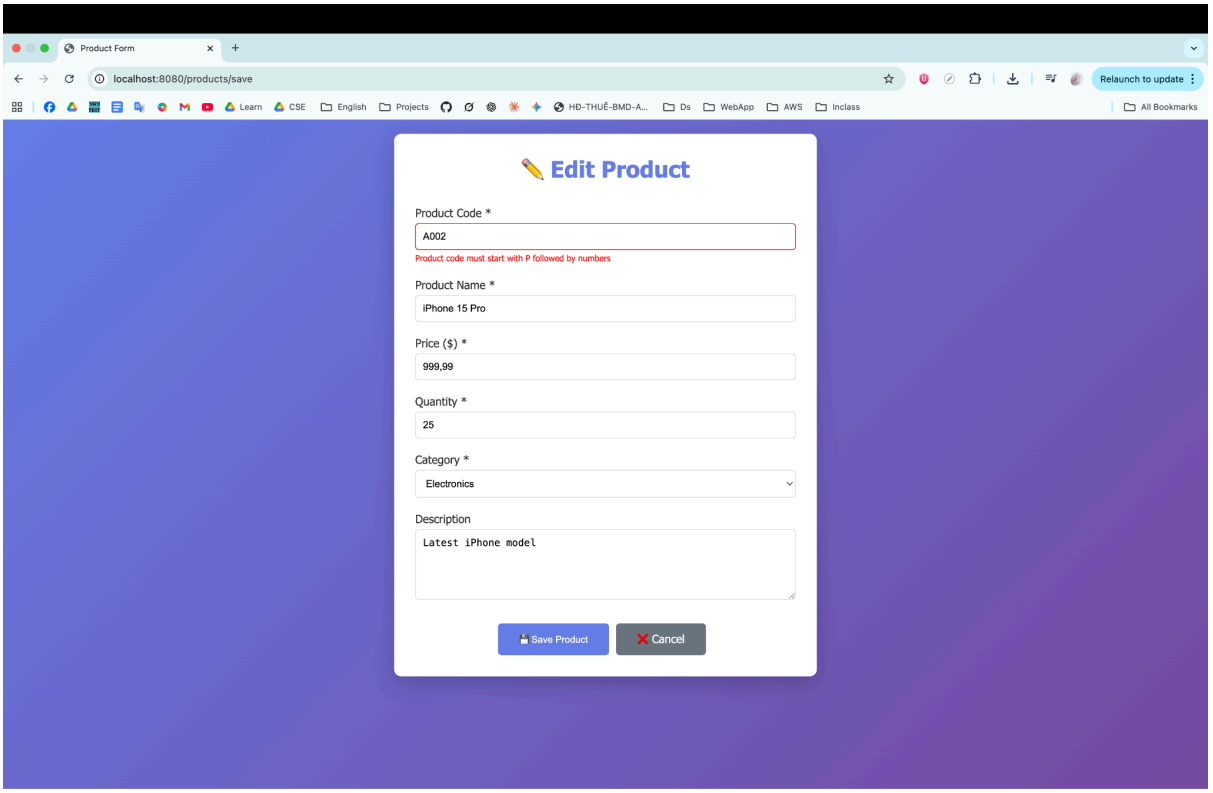
1. Browser sends the GET request to /products.
2. Controller Receives Request: ProductController.listProducts method is invoked. It extracts parameters and constructs the Sort object (Sort.by("price"). ascending()).
3. Controller Calls Service (Search & Sort): Controller calls ProductService.searchProductsAdvanced(name, category, minPrice, maxPrice, sort).
4. Service Builds Query: ProductServiceImpl.searchProductsAdvanced uses the Specification interface to build dynamic Predicate objects for filtering (e.g., cb.like(root.get("name"), "%phone%"), cb.equal(root.get("category"), "Electronics")).
5. Service Calls Repository: Service calls productRepository.findAll(Specification, Sort).
6. JPA Executes Query: Spring Data JPA executes a single query combining the WHERE clause (from Specification) and the ORDER BY clause (from Sort).
7. Controller Adds Data: Controller receives the filtered and sorted List<Product>. It adds the results (products) and all search/sort parameters to the Model.
8. Returns view name "product-list".
9. Thymeleaf Renders HTML: Thymeleaf renders the page, showing the results and maintaining form/sorting state.
10. Response sent to browser.

EXERCISE 6: VALIDATION

Task 6.1: Add Validation Annotations

Task 6.2: Add Validation in Controller

Task 6.3: Display Validation Errors



1. Browser Sends Request: Browser sends a POST request with form data to /products/save.

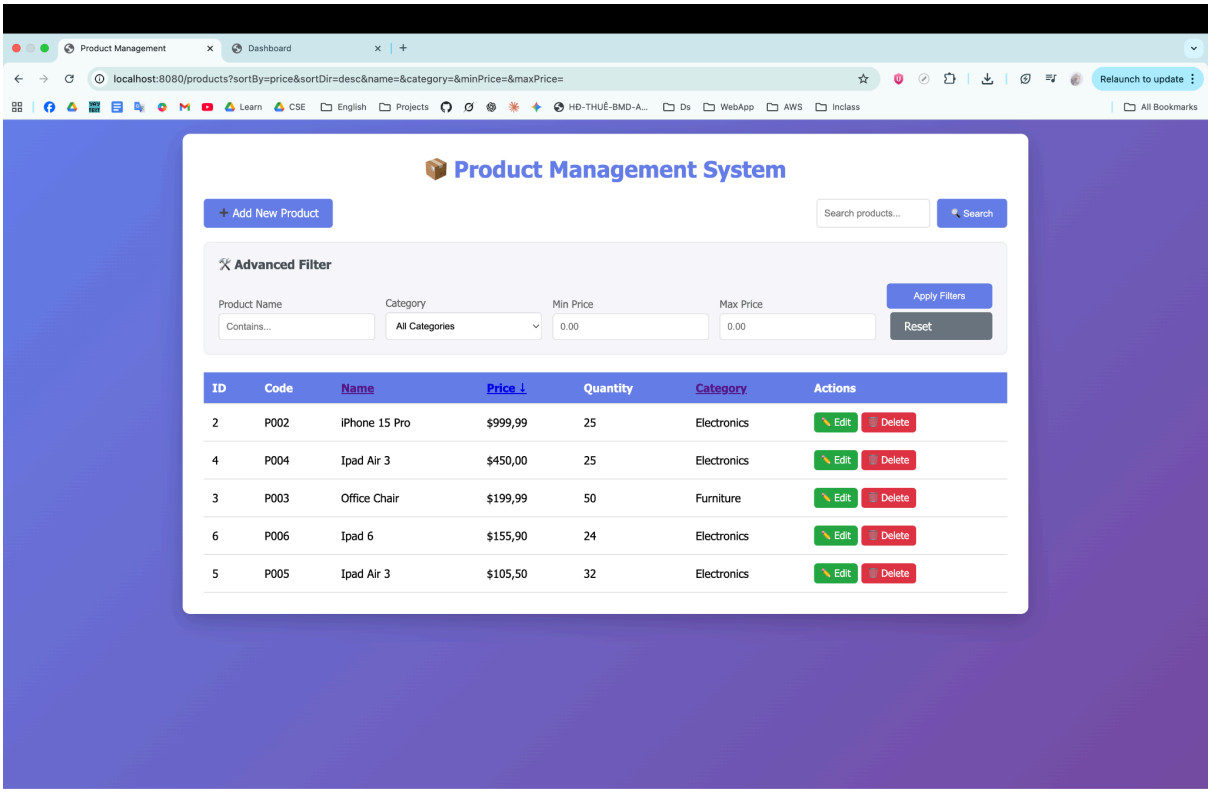
2. Controller Receives & Validates: ProductController.saveProduct method is invoked. Spring attempts to bind form data to the @Valid Product product object.
3. Bean Validation Check: Spring triggers the validation constraints on Product.java (@NotBlank, @DecimalMin, etc.). Errors are recorded in BindingResult.
4. Controller Handles Error: If result.hasErrors() is TRUE. Controller returns the view "product-form".
5. Thymeleaf Renders Errors: product-form.html displays the input data along with the specific error messages using th:errors.
6. Controller Handles Success: If result.hasErrors() is FALSE. Controller calls productService.saveProduct(product).
7. Service & Repository Persist: ProductServiceImpl calls productRepository.save(product), persisting the data to the database.
8. Controller Redirects: Controller adds a success message to RedirectAttributes and issues a redirect to /products.

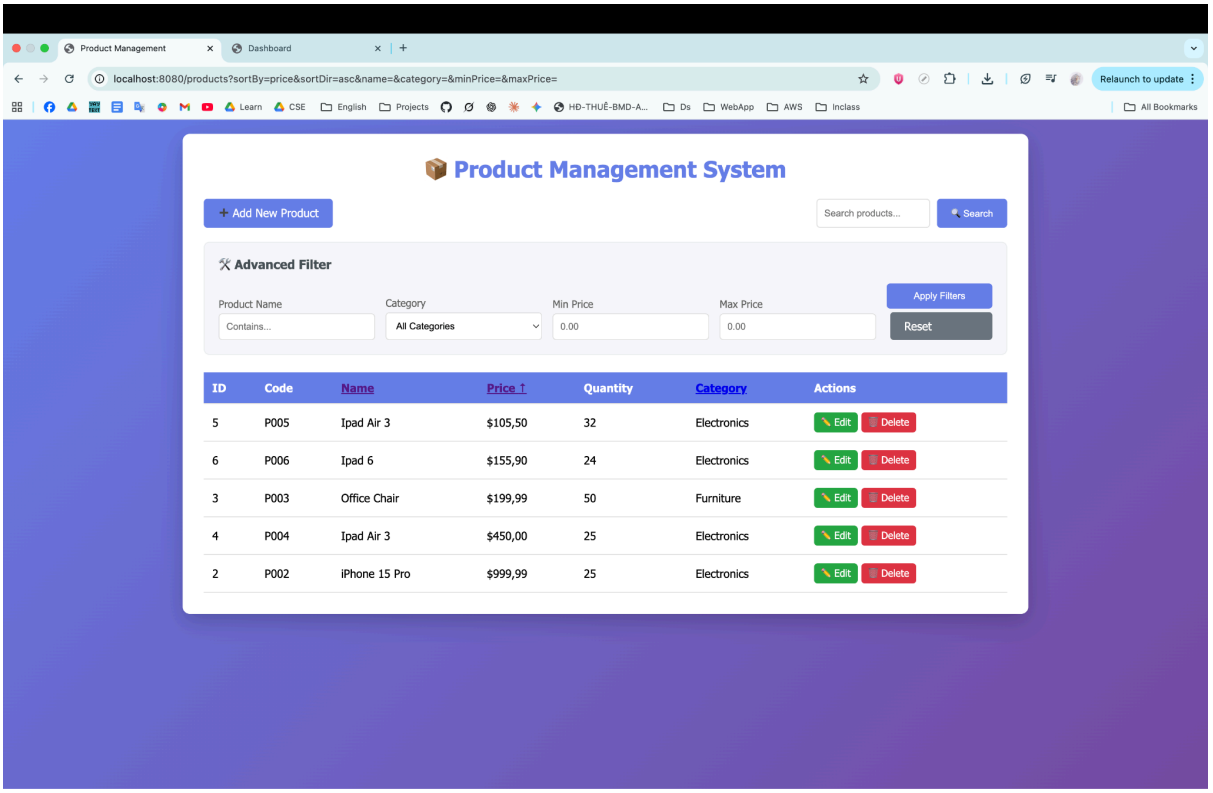
EXERCISE 7: SORTING & FILTERING

Task 7.1: Add Sorting

Task 7.2: Filter by Category

Task 7.3: Combined Sorting and Filtering





Explain:

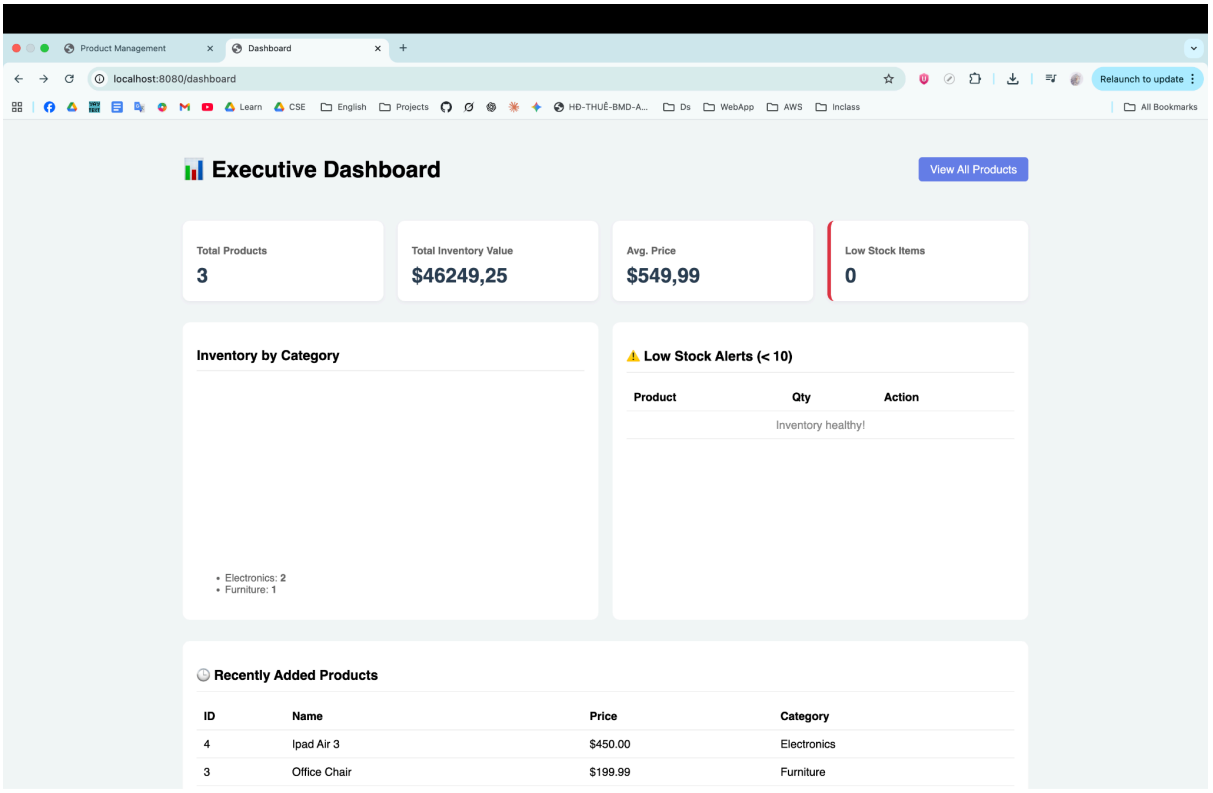
The sorting mechanism in the product-management system is implemented entirely via server-side rendering, ensuring robust and secure sorting logic. This functionality is embedded directly within the anchor tags (<a>) inside the table headers (<th>) in the product-list.html file using Thymeleaf's URL expression syntax (@{...}). When a user clicks a column header (e.g., Name or Price), the system triggers a new HTTP GET request to /products, carrying the crucial parameters: sortBy (the field to sort) and sortDir (the sorting direction). The core logic involves state toggling: it uses a ternary operation to check the current direction and automatically reverse it (from 'asc' to 'desc' and vice versa) for the new URL, allowing users to cycle the sort order with repeated clicks. Furthermore, this mechanism implements combined sorting and filtering by ensuring that all existing filter parameters (name, category, minPrice, maxPrice) are maintained in the new URL, guaranteeing that the re-sorted list remains accurately filtered.

EXERCISE 8: STATISTICS DASHBOARD

Task 8.1: Add Statistics Methods

Task 8.2: Create Dashboard Controller

Task 8.3: Create Dashboard View



1. Browser sends a GET request to /dashboard.
2. Controller Receives Request DashboardController.showDashboard method is invoked.
3. Controller Calls Service (Stats) : Controller makes multiple calls to the service to fetch metrics: getTotalCount(), getTotalInventoryValue(), getAveragePrice(), getAllCategories(), getLowStockProducts(10), and getRecentProducts().
4. Service Calls Repository (Queries): ProductServiceImpl calls the corresponding specialized query methods in ProductRepository (e.g., calculateTotalValue(), findLowStockProducts(), findTop5ByOrderByIdDesc()). It also loops through all categories to call countByCategory(cat) and build the categoryStats Map.
5. Controller Adds Data: Controller receives the computed metrics and the categoryStats Map, and adds all of them to the Model.
6. Returns view name "dashboard".
7. Thymeleaf Renders HTML: Thymeleaf renders the page, displaying the metrics in stat cards and data tables.
8. Response sent to browser.