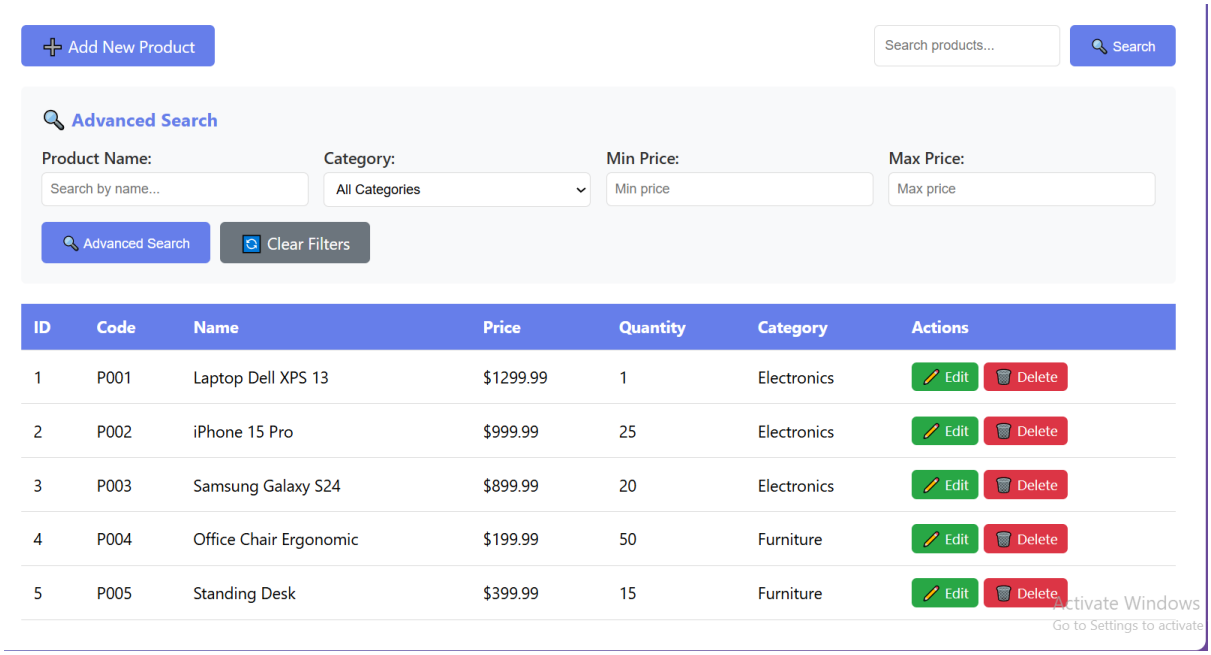
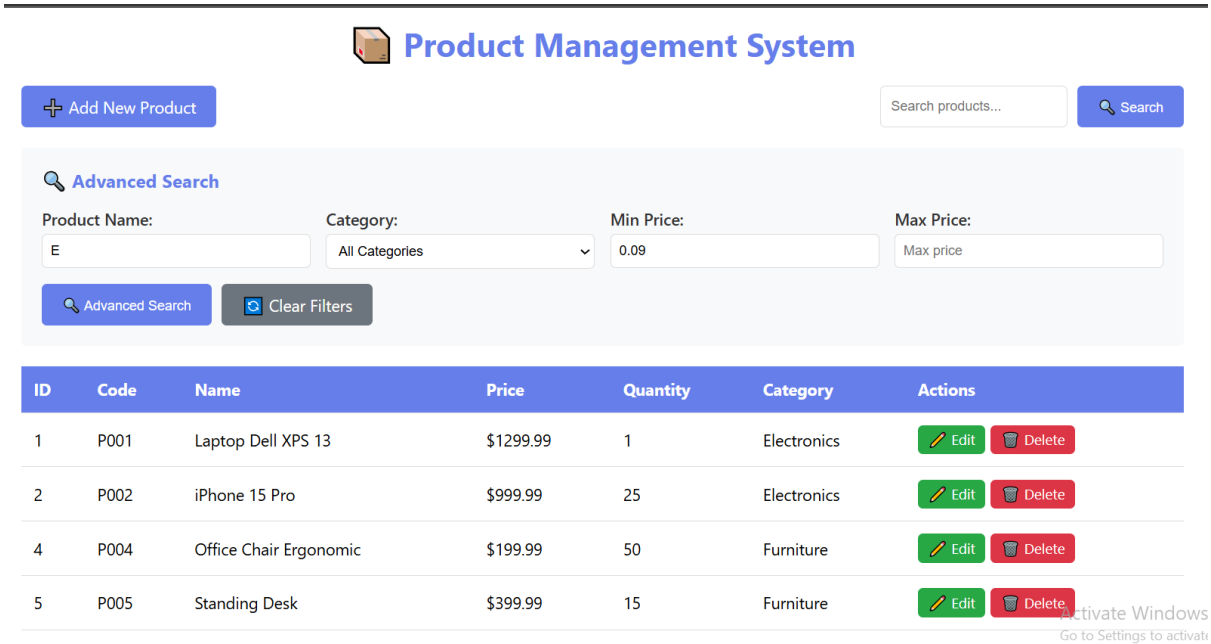
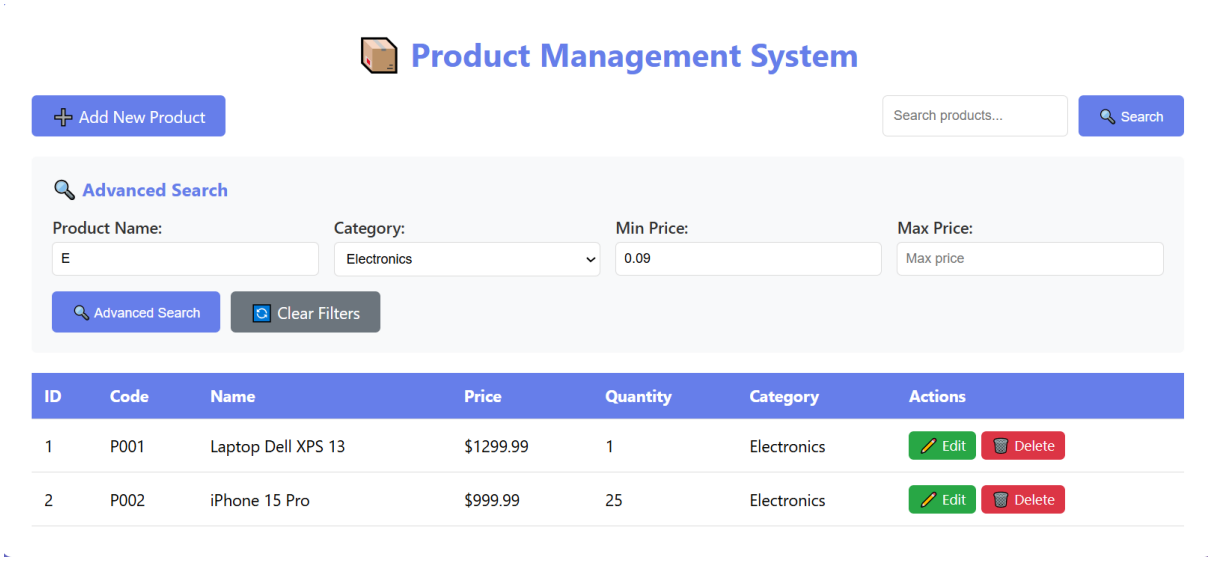
Lab 7 Homework
Name: Nguyễn Hữu Hoàng Nam

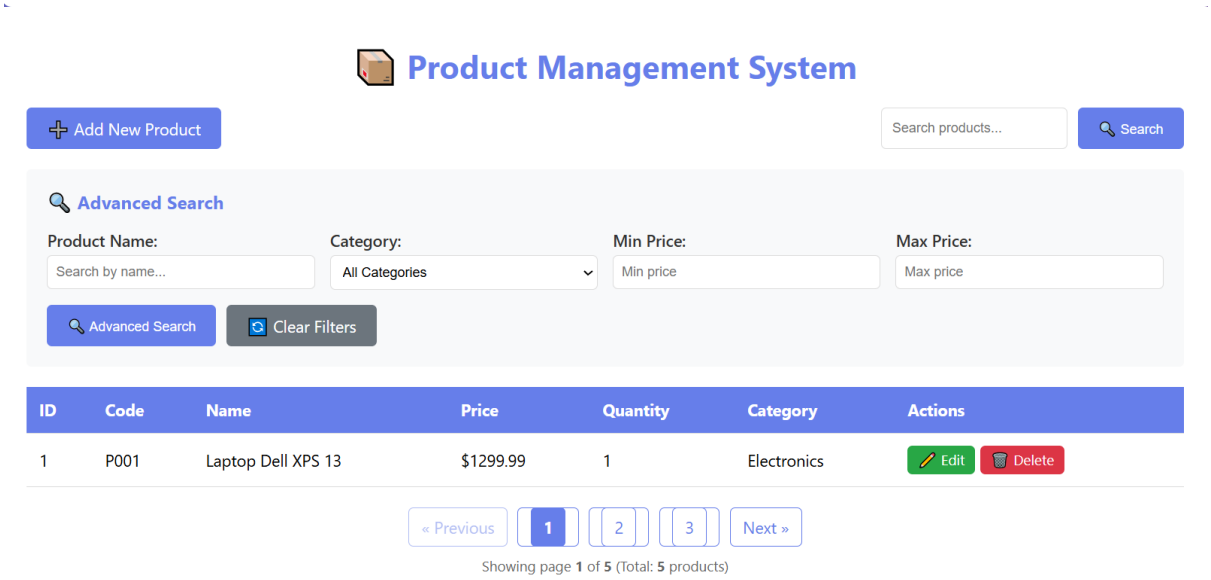
StudentID: ITCSIU23028

Exercise 5:









Flow(Advanced Search):

1. User fills in advanced search form with criteria.
2. User clicks "Advanced Search" button.
3. Browser sends GET request to `/products/advanced-search` with all parameters.
4. Controller receives parameters (name, category, minPrice, maxPrice).
6. Controller calls `productService.advancedSearch.
7. Service calls `productRepository.searchProducts.
8. Repository executes custom JPQL query .
9. Repository returns filtered List<Product>.
10. Controller adds filtered products to Model.
11. Returns "product-list" view.
12. User sees only products matching ALL criteria.

Flow(Search with pagination):

1. User types keyword in search box and clicks Search

2. Browser sends GET request to `/products/search?keyword=laptop&page=0&size=10`

3. Controller receives parameters:

5. Controller calls `productService.searchProducts(keyword, pageable)`

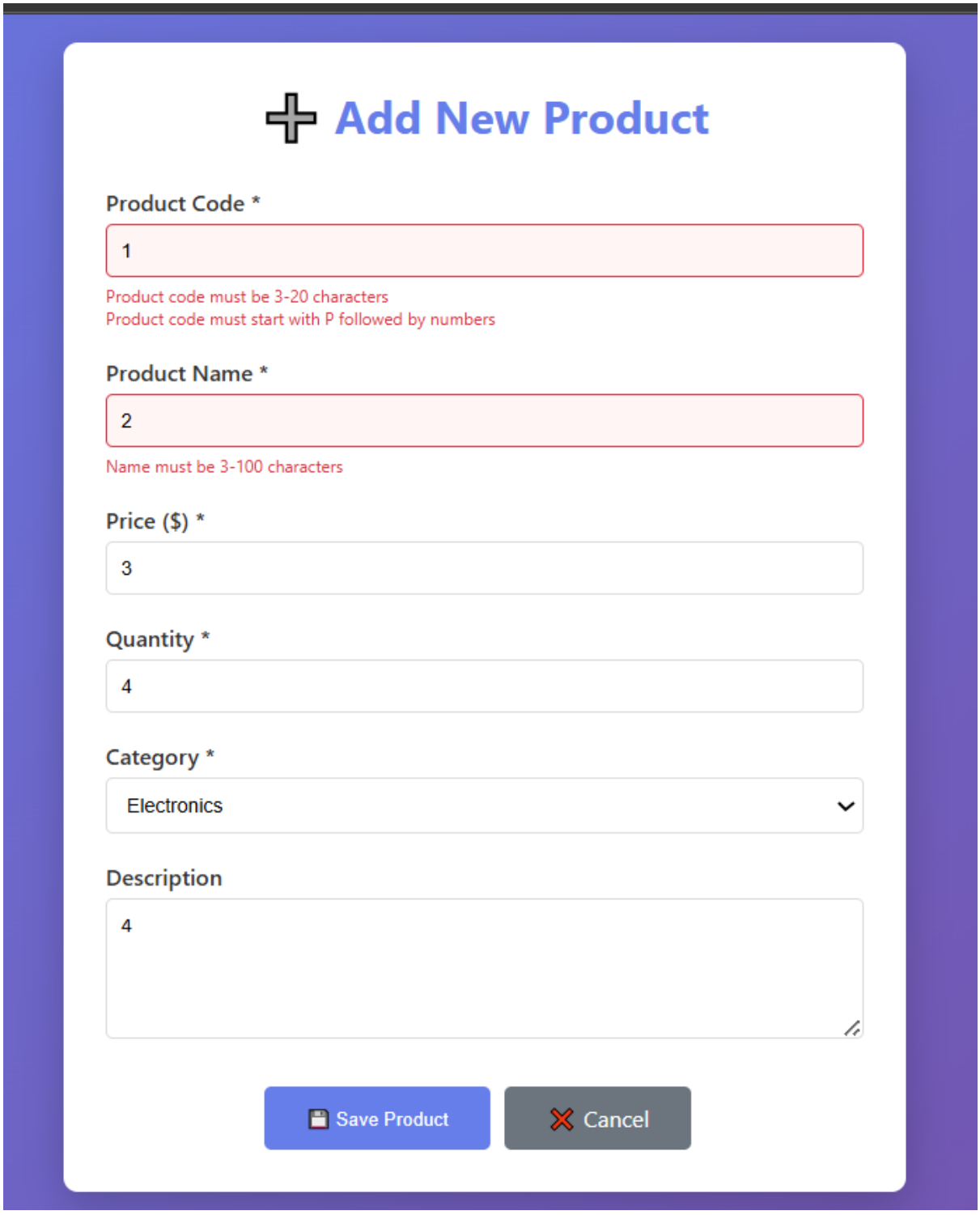
6. Service calls `productRepository.findByNameContainingIgnoreCase(keyword, pageable)`

7. Repository executes paginated query:

9. Controller extracts data from Page object and adds to Model:

10. Returns "product-list" view

Exercise 6:



Flow:

1. User fills in product form with valid data:
2. User clicks Save button
3. Browser sends POST request to `/products/save` with form data
4. Controller receives Product object
5. `@Valid` annotation triggers validation

6. Spring Validator checks all constraints - ALL PASS

7. BindingResult contains no errors

8. Controller checks: `if (result.hasErrors())` → FALSE

9. Controller proceeds to save:

   - Calls `productService.saveProduct(product)`

   - Service calls `productRepository.save(product)`

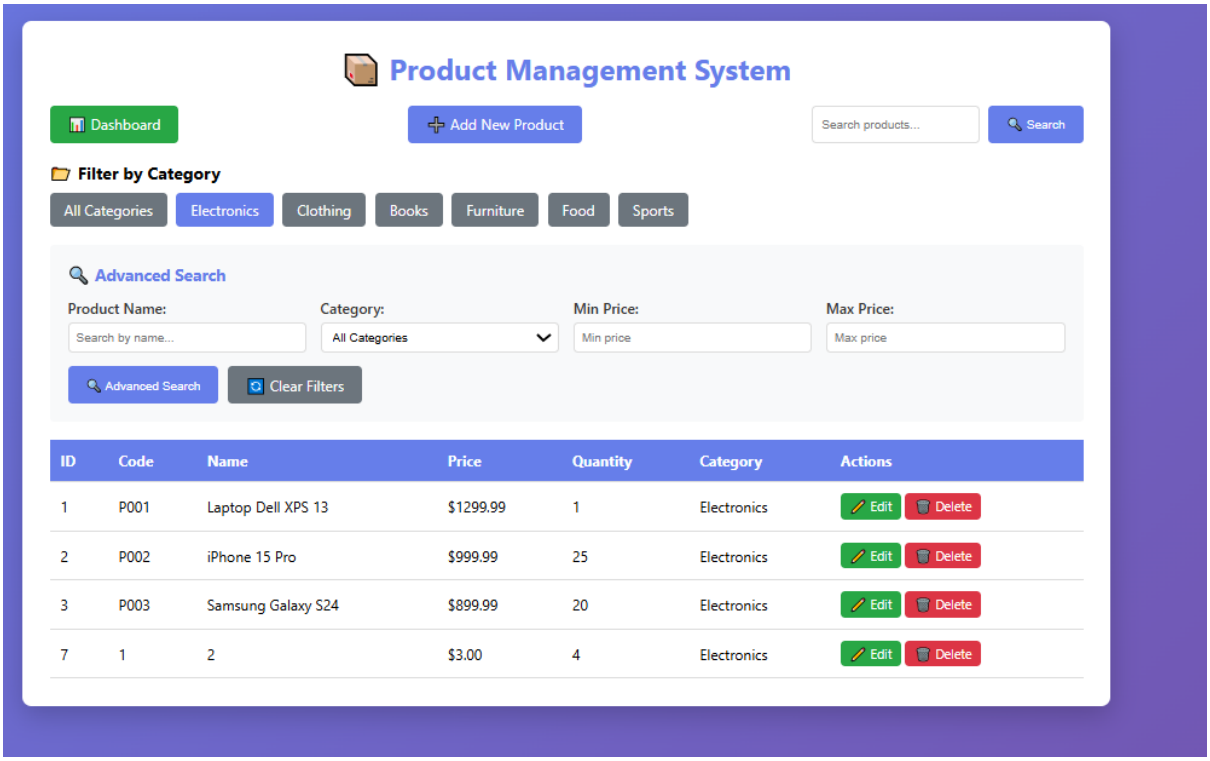
   - Repository saves to database

10. Controller adds success message: `"Product saved successfully!"`

11. Redirects to `/products`

12. User sees product list with success message

Exercise 7:



Flow

1. User selects category "Electronics" AND sorts by "Price" ascending

2. Browser sends GET request with both parameters

3. Controller receives category, sortBy, sortDir

4. If category provided:

   - Calls `productService.getProductsByCategory(category)`

   - Applies manual sorting using Stream API and Comparator

5. If no category (all products):

- Creates Sort object

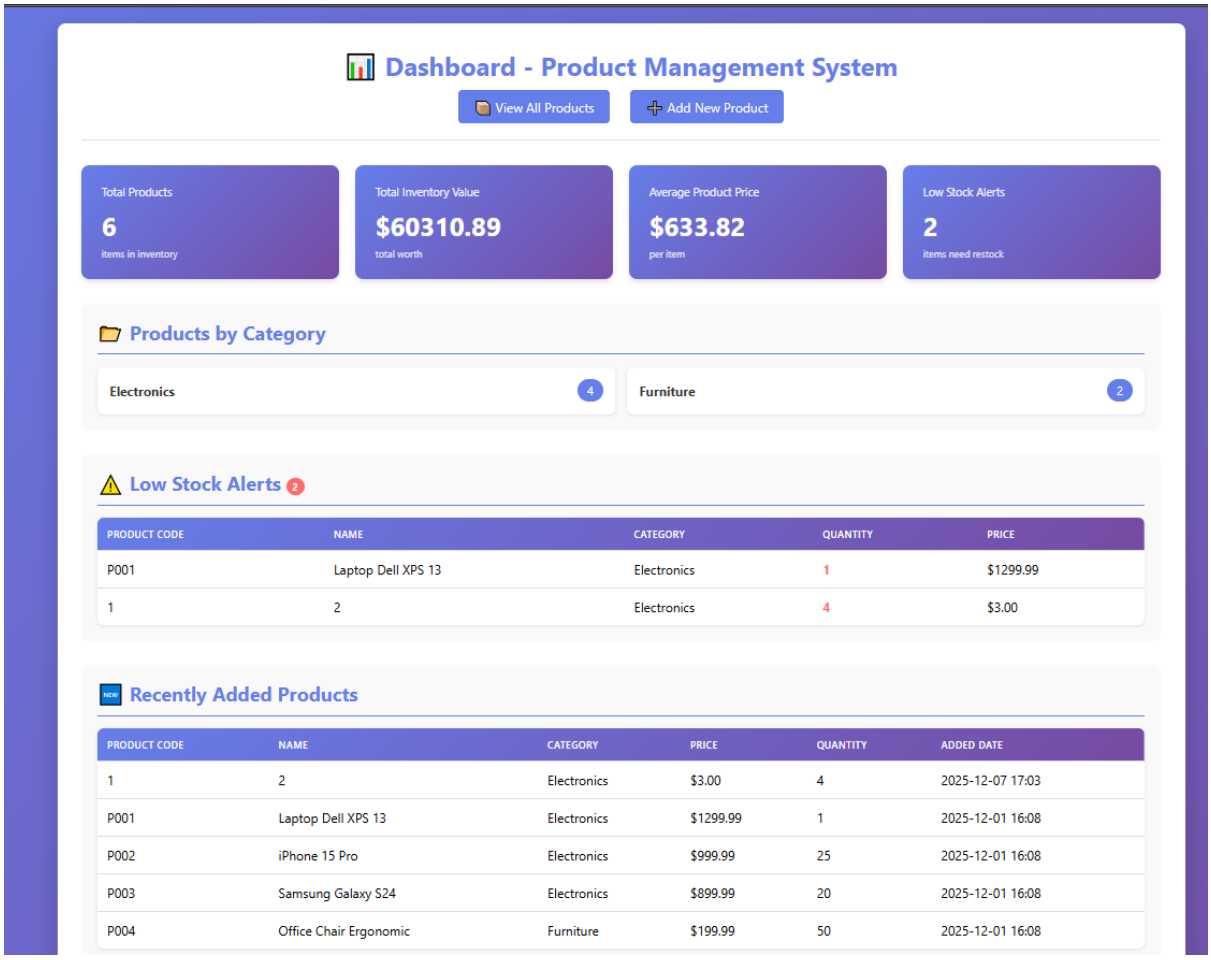
- Calls `productService.getAllProducts(sort)`

6. Returns sorted and filtered List<Product>

7. Controller adds products, category, sortBy, sortDir to Model

8. Returns "product-list" view

Exercise 8:



Flow:

1. User clicks "Dashboard" button from product list.

2. Browser sends GET request to `/dashboard`.

3. Controller gathers 6 types of statistics from ProductService.

4. Service calls repository methods with @Query annotations

5. Repository executes SQL queries .

6. Controller adds all statistics to Model.

7. Returns "dashboard" view.