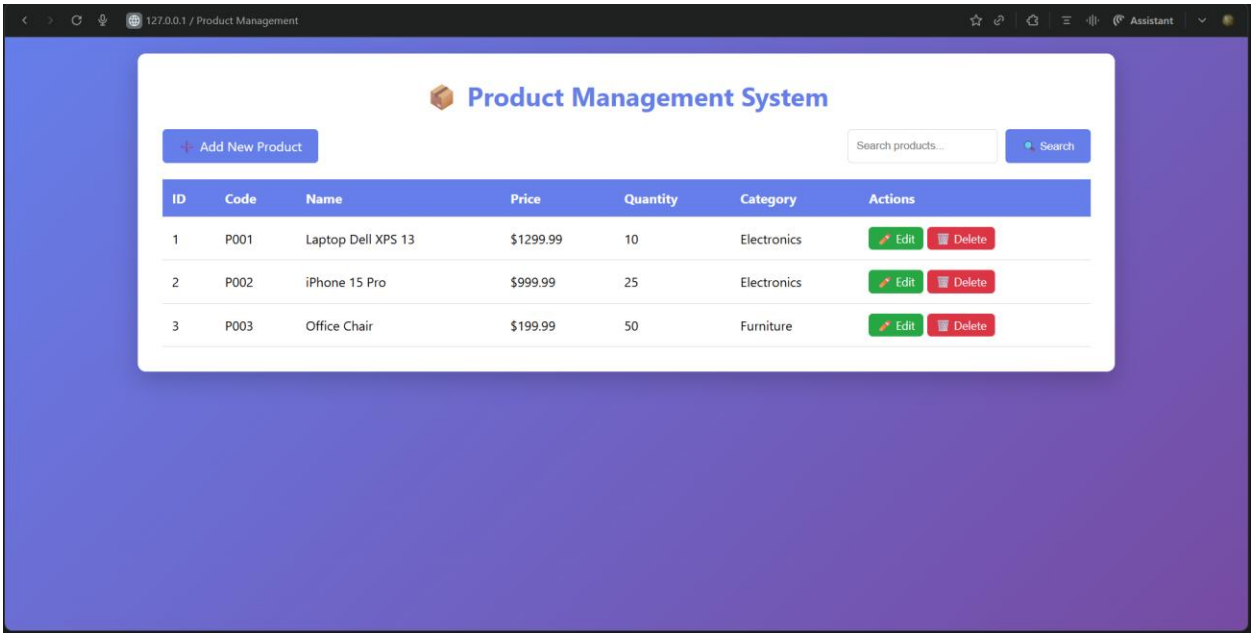
On Minh Quan

ITITWE23009

Web App Dev Lab 7 practice report

**List all products: http:// 127.0.0.1:8081/products**




Browser sends GET /products.

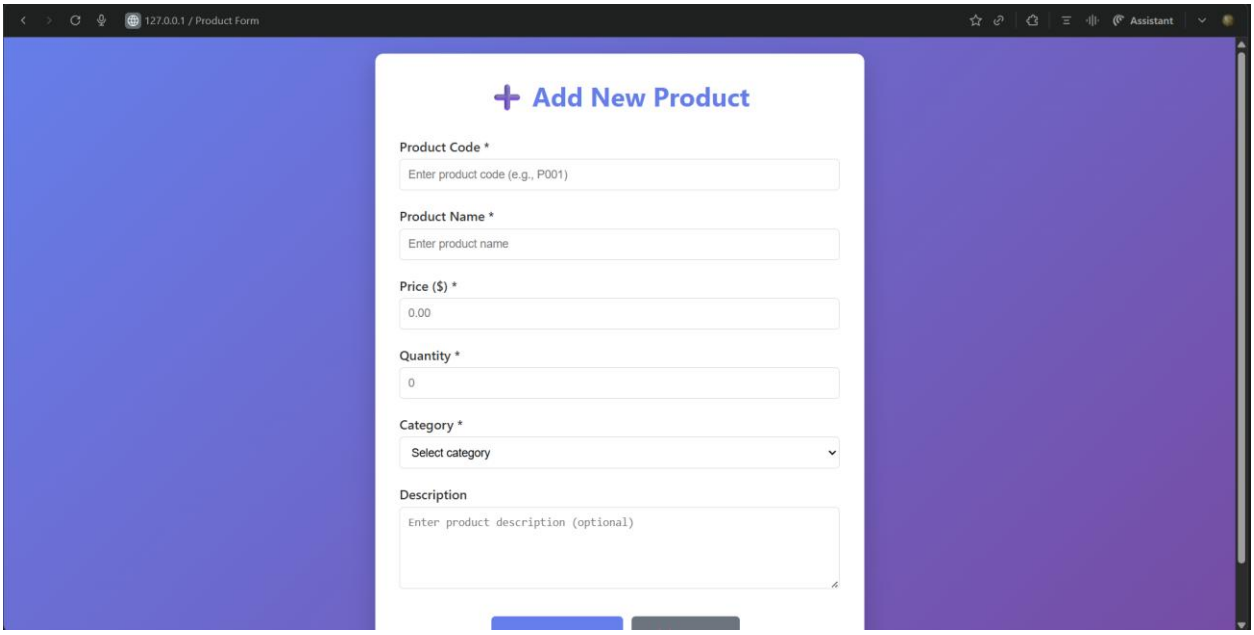
@GetMapping without a path on a controller annotated with @RequestMapping("/products") handles this in listProducts(Model model).

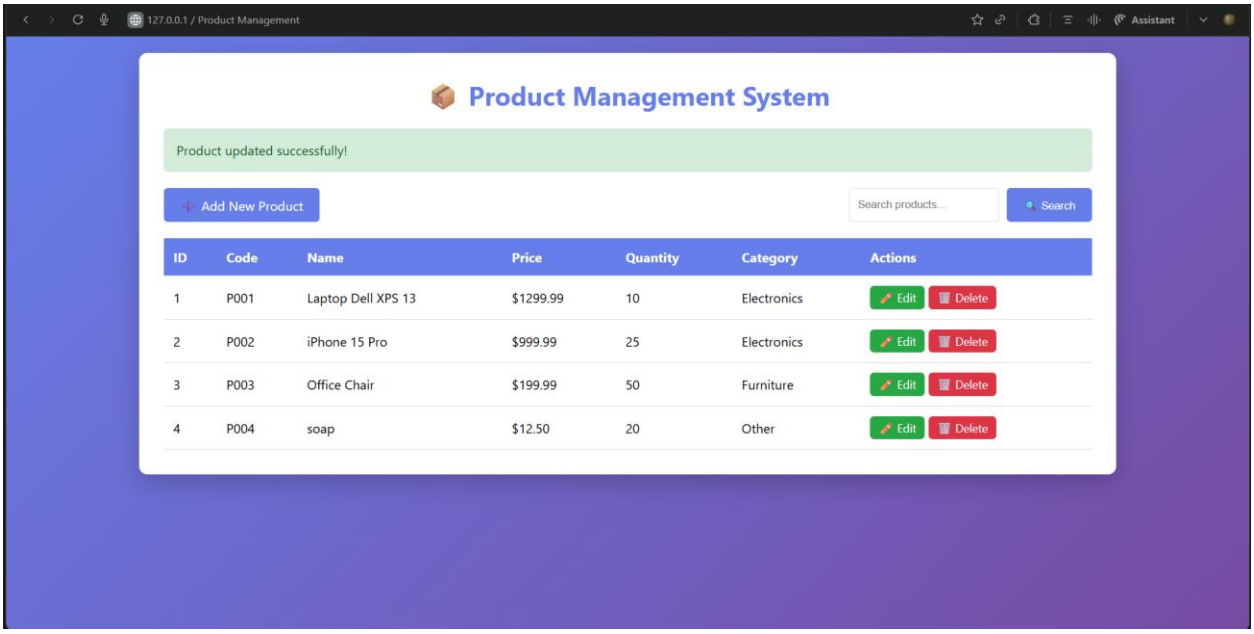
Controller calls productService.getAllProducts(), which calls productRepository.findAll() to load all Product entities from the products table.

The result List<Product> is put into the model as attribute "products".

Method returns "product-list", so Thymeleaf renders templates/product-list.html, which loops over ${products} and displays them

**Add new product: http:// 127.0.0.1:8081/products/new**




Browser sends GET /products/new

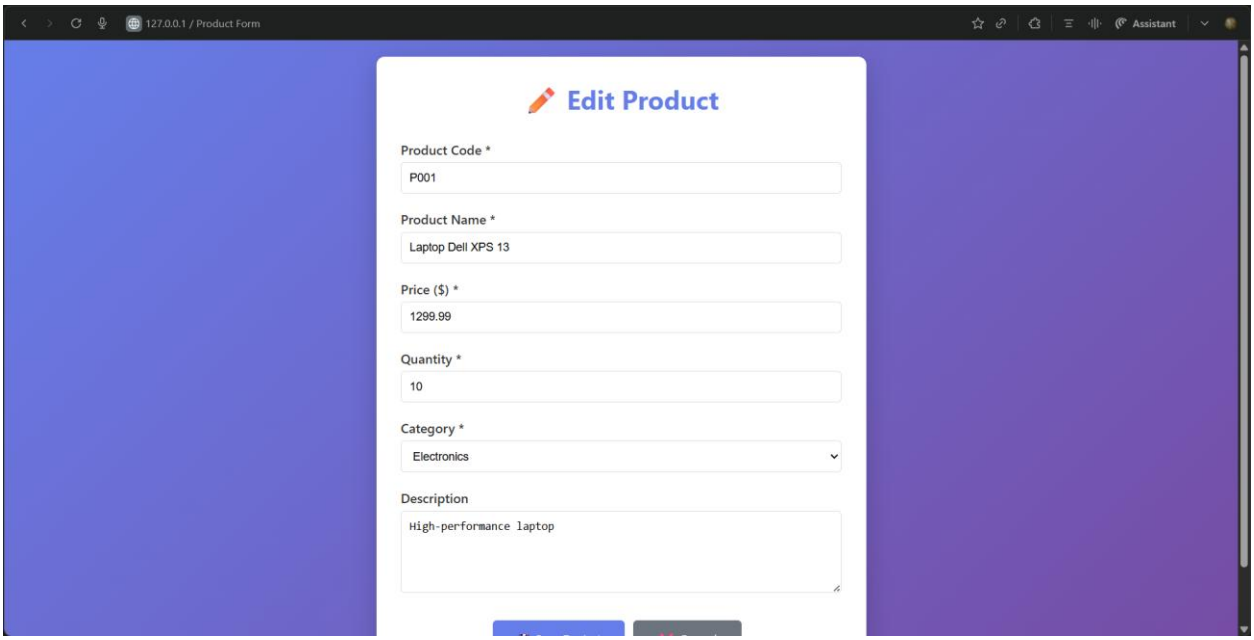
@GetMapping("/new") in ProductController handles this with showNewForm(Model model)

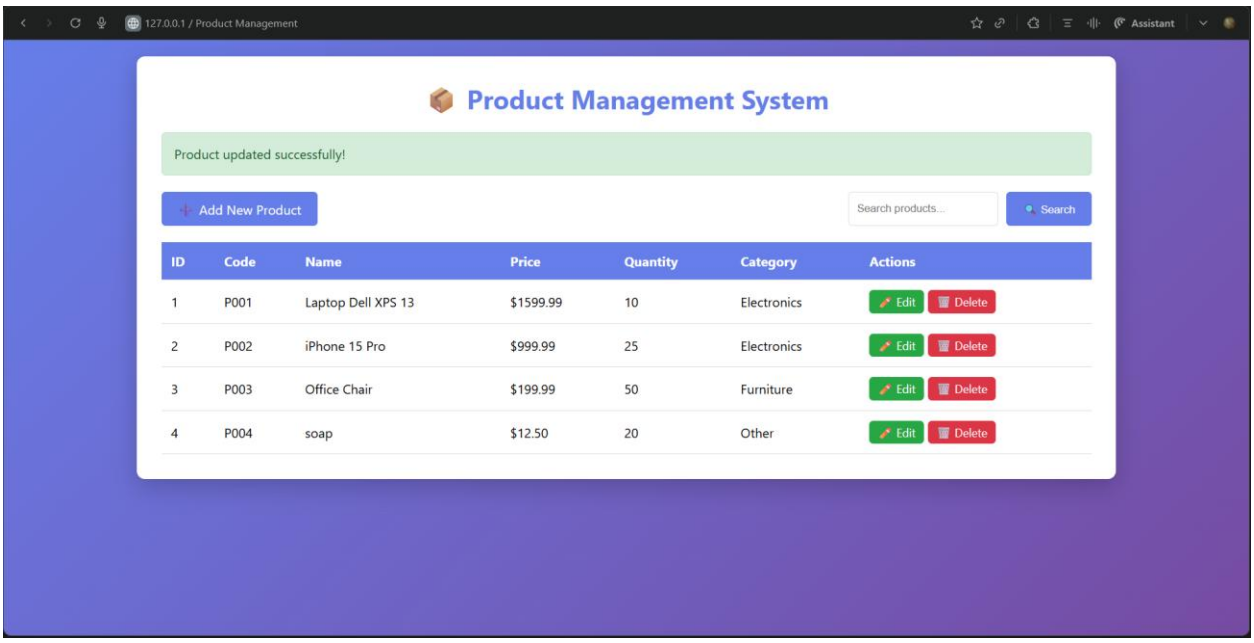
Controller creates new Product() and adds it to the model as "product"

Returns "product-form", so product-form.html is rendered; the form fields are bound with th:object="${product}" and th:field="*{...}"

When the user submits, the form posts to /products/save (handled by @PostMapping("/save") to actually insert/update)

**Edit product (ID=1): http://127.0.0.1:8081/products/edit/1**




Browser sends GET /products/edit/1

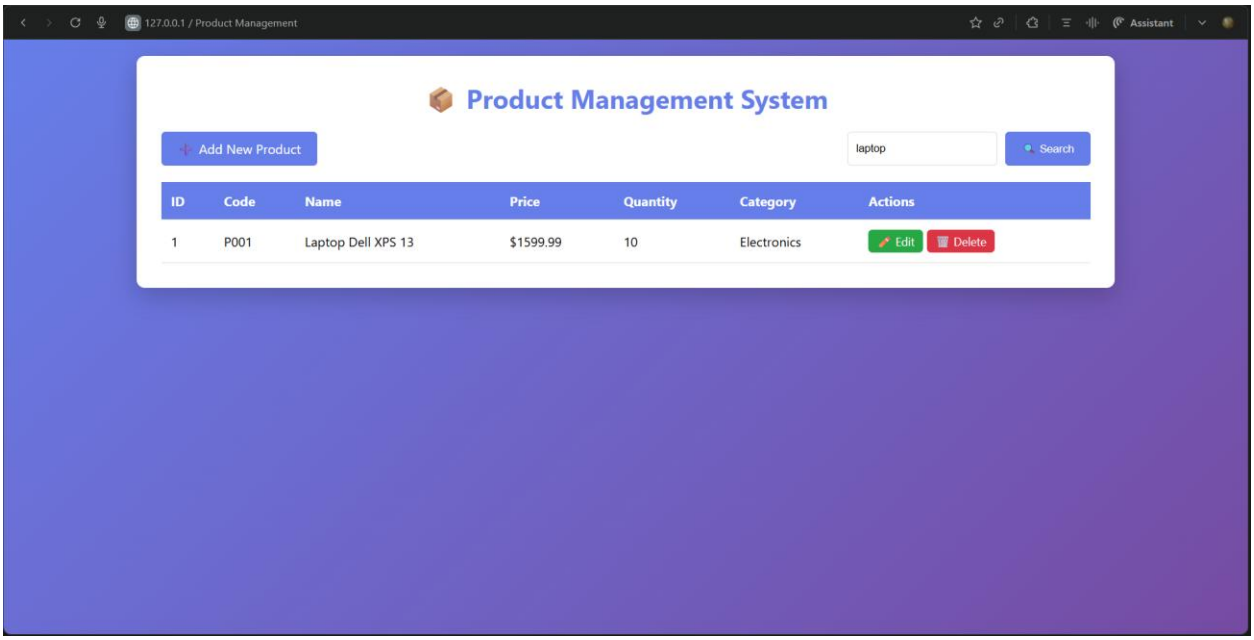
@GetMapping("/edit/{id}") handles this, binding 1 into @PathVariable Long id

Controller calls productService.getProductById(id), which calls productRepository.findById(id) and returns an Optional<Product>

If product exists:

- Put that Product into the model as "product" and return "product-form"
- The form shows existing values

**Search products: http:// 127.0.0.1:8081/products/search?keyword=laptop**




Browser sends GET /products/search?keyword=laptop, usually from the search form on the list page.

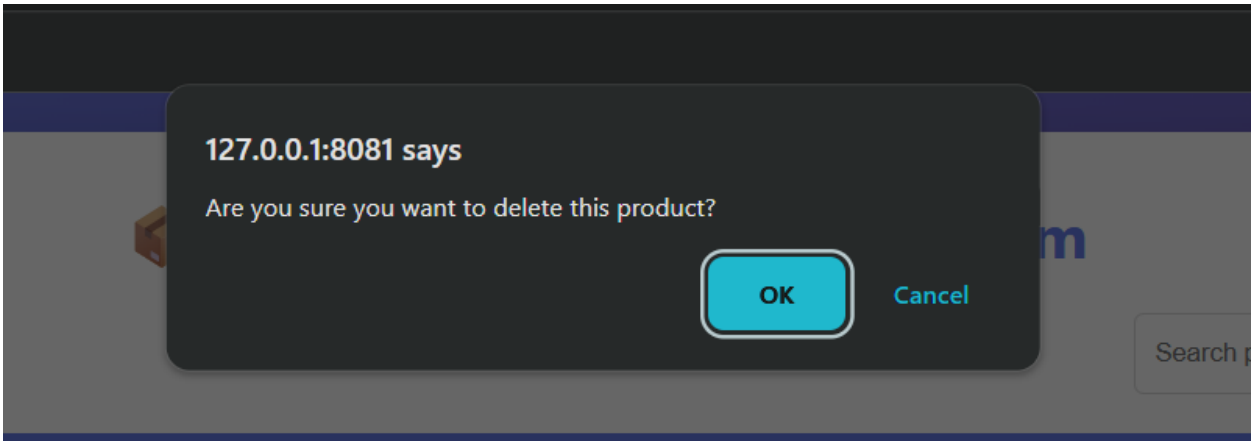
@GetMapping("/search") handles this, binding keyword from the query string via @RequestParam("keyword") String keyword.

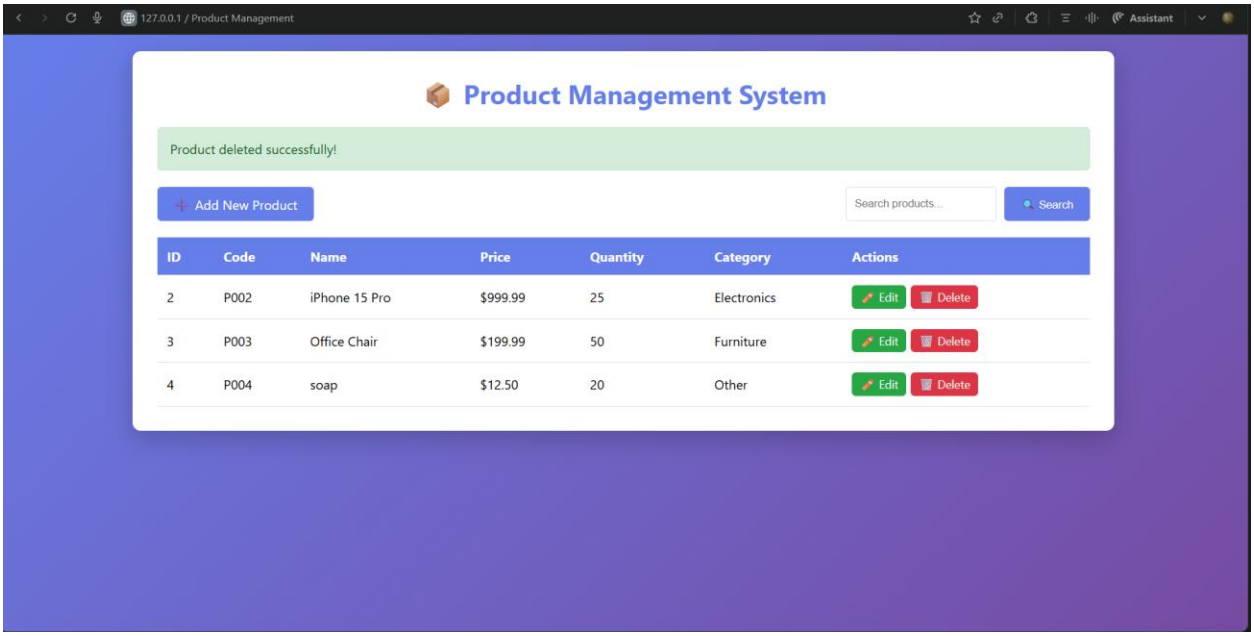
Controller calls productService.searchProducts(keyword), which calls productRepository.findByNameContaining(keyword) (a Spring Data JPA derived query) to get matching products

Add the resulting List<Product> to model as "products" and the keyword back to the model to refill the search box

Return "product-list" so the same list page is reused to display only matching products

**Delete product (ID=1): http:// 127.0.0.1:8081/products/delete/1**




Browser sends GET /products/delete/1

@GetMapping("/delete/{id}") handles this, binding 1 into @PathVariable Long id

Controller calls productService.deleteProduct(id), which calls productRepository.deleteById(id) to remove the row from the products table

On success, add a flash "message" like “Product deleted successfully!”; on error, add an "error" message

Return "redirect:/products" so the browser is redirected to the list page and sees the updated list plus the flash message