

LAB 07 PRACTICE (INCLASS)

Output:

ID	Code	Name	Price	Quantity	Category	Actions
1	P001	Laptop Dell XPS 13	\$1299.99	11	Electronics	<button>Edit</button> <button>Delete</button>
2	P002	iPhone 15 Pro	\$999.99	25	Electronics	<button>Edit</button> <button>Delete</button>
3	P003	Office Chair	\$199.99	50	Furniture	<button>Edit</button> <button>Delete</button>

Workflow:

Default: Client sends requests mapping via localhost (currently 8079) to the controller ProductController where @RequestMapping("/products") and operations reside. It then fetches operations from the service layer, serving as the business logic between the controller and the repository.

```
@Controller
@RequestMapping("/products")
public class ProductController {

    private final ProductService productService;

    @Autowired  Unnecessary `@Autowired` annotation
    public ProductController(ProductService productService) {
        this.productService = productService;
    }
}
```

```

// List all products
@GetMapping
public String listProducts(Model model) {
    List<Product> products = productService.getAllProducts();
    model.addAttribute("products", products);
    return "product-list"; // Returns product-list.html
}

// Show form for new product
@GetMapping("/new")
public String showNewForm(Model model) {
    Product product = new Product();
    model.addAttribute("product", product);
    return "product-form";
}

// Show form for editing product
@GetMapping("/edit/{id}")
public String showEditForm(@PathVariable Long id, Model model, RedirectAttributes redirectAttributes) {
    return productService.getProductById(id)
        .map(product -> {
            model.addAttribute("product", product);
            return "product-form";
        })
        .orElseGet(() -> {
            redirectAttributes.addFlashAttribute("error", "Product not found");
            return "redirect:/products";
        });
}

```

In previous labs, the query execution must be done manually (for each operation towards students) including `PreparedStatement` and `ResultSet`. However, Repository Layer (`ProductRepository`) extends `JpaRepository` helps holding CRUD operations, additionally custom query methods that Spring Data JPA automatically implements based on method names.

```

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    // Spring Data JPA generates implementation automatically!

    // Custom query methods (derived from method names)
    List<Product> findByCategory(String category);

    List<Product> findByNameContaining(String keyword);

    List<Product> findByPriceBetween(BigDecimal minPrice, BigDecimal maxPrice);

    List<Product> findByCategoryOrderByPriceAsc(String category);

    boolean existsByProductCode(String productCode);

    // All basic CRUD methods inherited from JpaRepository:
    // - findAll()
    // - findById(Long id)
    // - save(Product product)
    // - deleteById(Long id)
    // - count()
    // - existsById(Long id)
}

```

Operations:

1. List

```
// List all products
@GetMapping
public String listProducts(Model model) {
    List<Product> products = productService.getAllProducts();
    model.addAttribute("products", products);
    return "product-list"; // Returns product-list.html
}
```

- URL: /products
- HTTP Method: GET
- This method fetches all products from the ProductService and adds them to the model for rendering in the product-list.html template.

2. Create

- URL: /products/new
- HTTP Method: GET
- This method prepares a new, empty Product object and passes it to the product-form.html template for creating a new product.

```
// Show form for new product
@GetMapping("/new")
public String showNewForm(Model model) {
    Product product = new Product();
    model.addAttribute("product", product);
    return "product-form";
}
```

+ Add New Product

Product Code *

Product Name *

Price (\$) *

Quantity *

Category *

Description

Save Product Cancel

3. Update (Edit)

- URL: /products/edit/{id}
- HTTP Method: GET
- This method retrieves a specific product by ID from the ProductService. If the product is found, it is sent to the form template for editing. If not, an error message is flashed, and the user is redirected to the product list.

```
// Show form for editing product
@GetMapping("/edit/{id}")
public String showEditForm(@PathVariable Long id, Model model, RedirectAttributes redirectAttributes) {
    return productService.getProductById(id)
        .map(product -> {
            model.addAttribute("product", product);
            return "product-form";
        })
        .orElseGet(() -> {
            redirectAttributes.addFlashAttribute("error", "Product not found");
            return "redirect:/products";
        });
}
```

Apply changes to both operations (create / update)

It uses the `@ModelAttribute` annotation to bind the submitted product data to a `Product` object, then calls the service to save it. After saving, a success or error message is flashed, and the user is redirected to the list of products.

```
// Save product (create or update)
@PostMapping("/save")
public String saveProduct(@ModelAttribute("product") Product product, RedirectAttributes redirectAttributes) {
    try {
        productService.saveProduct(product);
        redirectAttributes.addFlashAttribute("message",
            product.getId() == null ? "Product added successfully!" : "Product updated successfully!");
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", "Error saving product: " + e.getMessage());
    }
    return "redirect:/products";
}
```

4. Delete

- URL: `/products/delete/{id}`
- HTTP Method: GET
- This method deletes a product based on the provided ID and then flashes a success or error message, redirecting the user to the product list.

```
// Delete product
@GetMapping("/delete/{id}")
public String deleteProduct(@PathVariable Long id, RedirectAttributes redirectAttributes) {
    try {
        productService.deleteProduct(id);
        redirectAttributes.addFlashAttribute("message", "Product deleted successfully!");
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", "Error deleting product: " + e.getMessage());
    }
    return "redirect:/products";
}
```

Product Management System

Add New Product

Search products...

Search

ID	Code	Name	Price	Quantity	Category	Actions
1	P001	Laptop Dell XPS 13	\$1299.99	11	Electronics	<button>Edit</button> <button>Delete</button>
2	P002	iPhone 15 Pro	\$999.99	25	Electronics	<button>Edit</button> <button>Delete</button>
3	P003	Office Chair				<button>Edit</button> <button>Delete</button>

localhost:8079

Are you sure you want to delete this product?

Don't allow localhost:8079 to prompt you again

Cancel OK

A screenshot of a web application titled "Product Management System". The main page displays a table of products with columns for ID, Code, Name, Price, Quantity, Category, and Actions (Edit and Delete buttons). A modal dialog box is overlaid on the page, centered over the second product row. The dialog has a title "localhost:8079" and asks "Are you sure you want to delete this product?". It includes a checkbox option "Don't allow localhost:8079 to prompt you again" and two buttons at the bottom: "Cancel" and "OK".