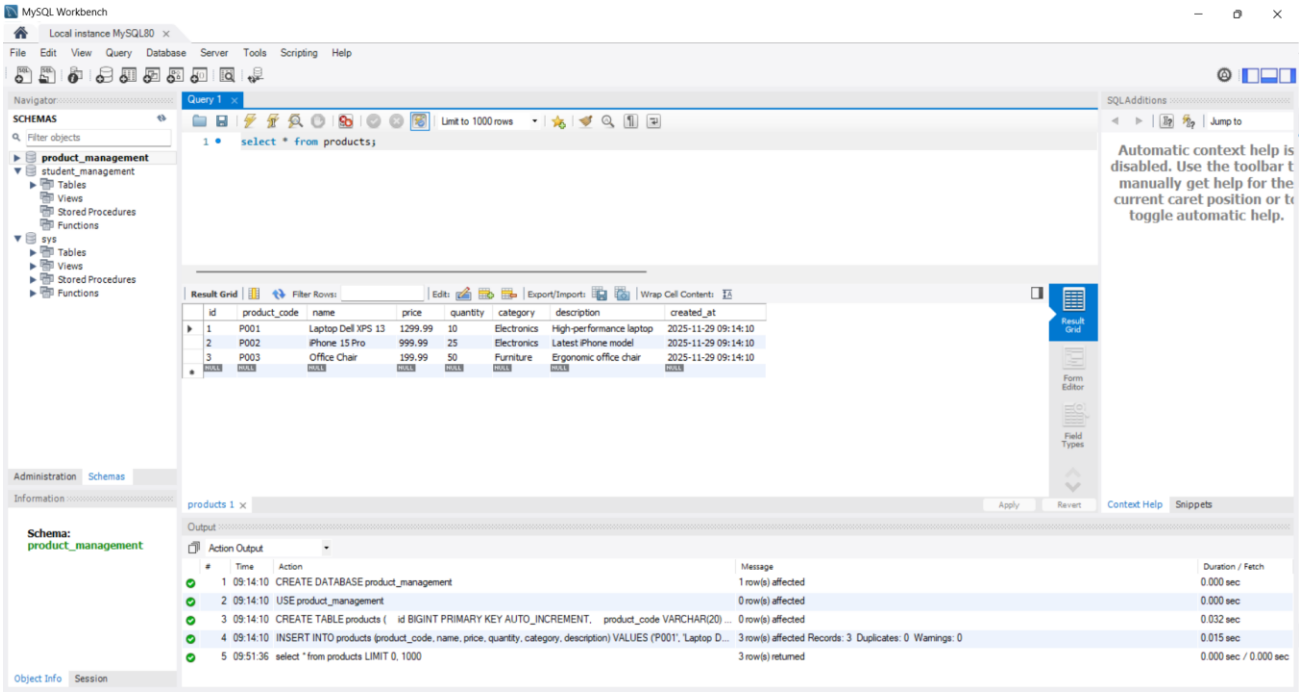
Web Application Development Lab 07

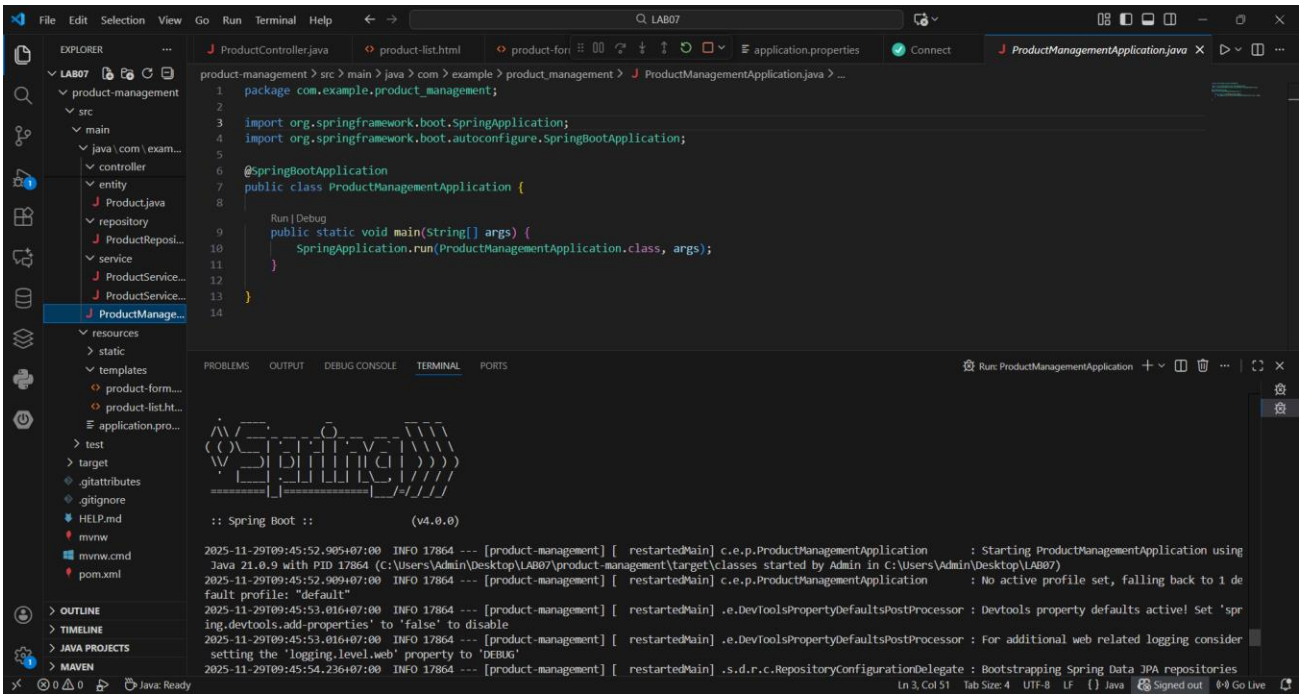
SPRING BOOT & JPA CRUD

Part B - HOMEWORK

I. PROJECT SETUP & CONFIGURATION



Database set up



Spring Boot setup

II. ADDING AND UPDATING A PRODUCT

```
<!-- Actions -->
<div class="actions">
    <a th:href="@{/products/new}" class="btn btn-primary">➕ Add New Product</a>

    <form th:action="@{/products/search}" method="get" class="search-form">
        <input type="text" name="keyword" th:value="${keyword}" placeholder="Search products..." />
        <button type="submit" class="btn btn-primary">🔍 Search</button>
    </form>
</div>
```

The “Add new product” button takes us to /products/new.

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

This creates a new Product object, and gives us the “product-form”

```
<form th:action="@{/products/save}" th:object="${product}" method="post">
    <!-- Hidden ID field for updates -->
    <input type="hidden" th:field="*{id}" />

    <div class="form-group">
        <label for="productCode">Product Code *</label>
        <input type="text"
               id="productCode"
               th:field="*{productCode}"
               placeholder="Enter product code (e.g., P001)"
               required />
    </div>
```

Our form posts the information to /products/save

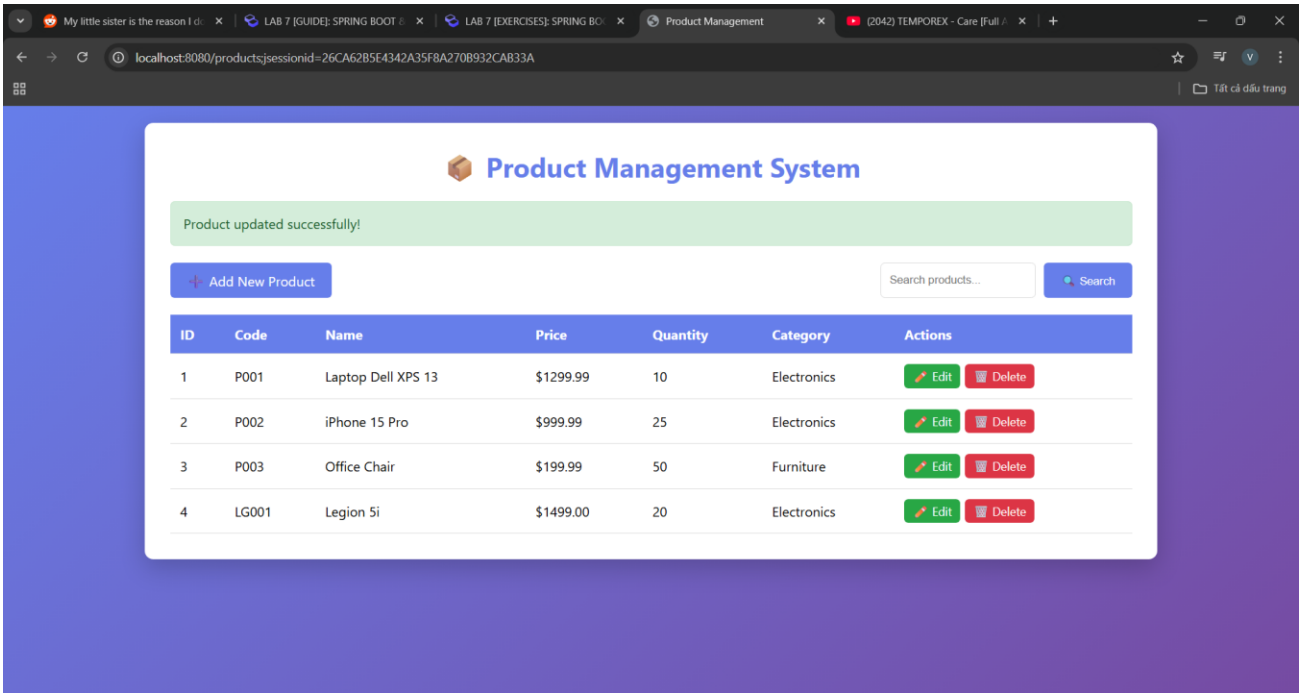
```java
// Save product (create or update)
@PostMapping("/save")
public String saveProduct(@ModelAttribute("product") Product product, RedirectAttributes redirectAttributes) {
    try {
        productService.saveProduct(product);
        redirectAttributes.addFlashAttribute("message",
             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

Next, we will try the add the new product to the database, we use productService for this.

```java
@Override
public Product saveProduct(Product product) {
    // Validation logic can go here
    return productRepository.save(product);
}
```

The productRepository extends JpaRepository, which gives us all of its CRUD operations, including save.

Once the product is added successfully (or unsuccessfully), we redirect the user back to the product-list with the appropriate message.



Added Legion 5i

Similarly, for updating a product

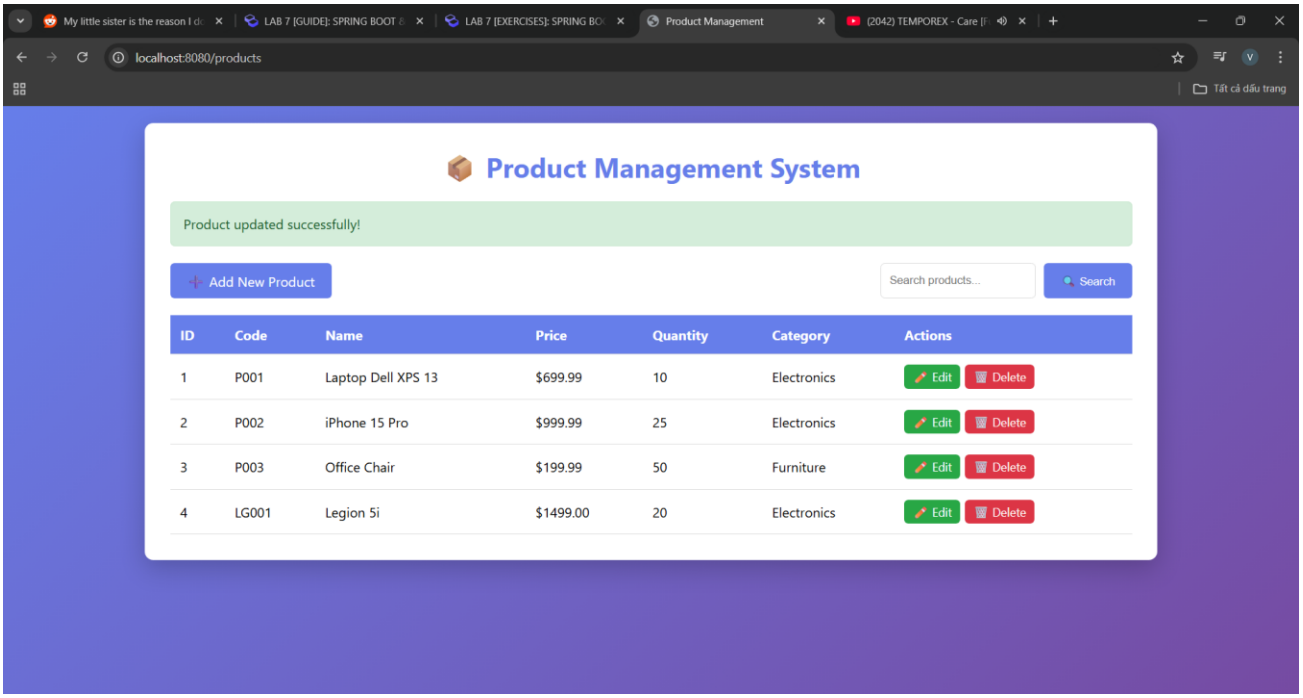
```java
// Show form for editing product
@GetMapping("/edit/{id}")
public String showEditForm(@PathVariable Long id, Model model, RedirectAttributes redirectAttributes) {
    return productService.getProductById(id)
            .map(product -> {
                model.addAttribute("product", product);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

The id used to determine what product is being edited, it maps the from the list to the id of the product in the database.

```java
@Override
public Optional<Product> getProductById(Long id) {
    return productRepository.findById(id);
}
```

Again, we save it to the database and return the user back to the product-list with the appropriate message.

```java
// 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";
}
```



Price update for Dell XPS 13

III. DELETING

```html
<div class="actions-column">
    <a th:href="@{/products/edit/{id}(id=${product.id})}" class="btn btn-success btn-sm">✏️ Edit</a>
    <a th:href="@{/products/delete/{id}(id=${product.id})}"
       class="btn btn-danger btn-sm"
       onclick="return confirm('Are you sure you want to delete this product?')">
        🗑️ Delete
    </a>
</div>
```

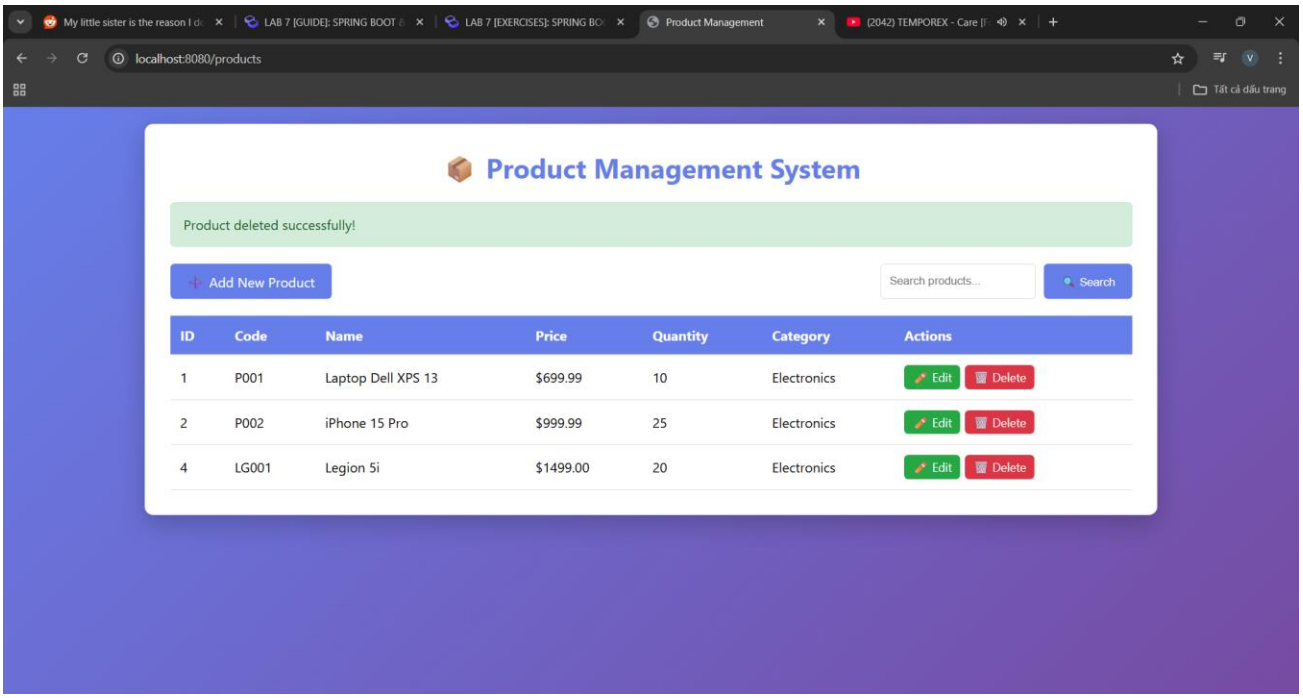
The delete button sends us to /products/delete/ according to the product id.

```java
// 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";
}
```

Similar to adding a product, we invoke productService to do our bidding, and ask it the delete the product by id. This maps to the deletProductById of the productRepository

```java
@Override
public void deleteProduct(Long id) {
    productRepository.deleteById(id);
}
```

Lastly, we send the user back to the list with the appropriate message.



Deleting the office chair

IV. **LISTING**

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

    private final ProductService productService;

    @Autowired
    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
    }
```

In the ProductController, going to /products automatically lists out the product, which invoke productService to retrieve all the products for listing and returns product-list.html.

```java
@Override
public List<Product> getAllProducts() {
    return productRepository.findAll();
}
```

The productService provides mapping for getAllProducts() to productRepository.findAll(), which is a JPARepository CRUD operation.

```
<tr th:each="product : ${products}">
    <td th:text="${product.id}">1</td>
    <td th:text="${product.productCode}">P001</td>
    <td th:text="${product.name}">Product Name</td>
    <td th:text="'$' + ${#numbers.formatDecimal(product.price, 1, 2)}">$99.99</td>
    <td th:text="${product.quantity}">10</td>
    <td th:text="${product.category}">Electronics</td>
    <td>
        <div class="actions-column">
            <a th:href="@{/products/edit/{id}(id=${product.id})}" class="btn btn-success btn-sm">🖍️ Edit</a>
            <a th:href="@{/products/delete/{id}(id=${product.id})}"
               class="btn btn-danger btn-sm"
               onclick="return confirm('Are you sure you want to delete this product?')">
                🗑️ Delete
            </a>
        </div>
    </td>
</tr>
```

With the list of products, product-list lists each of them out in a table.