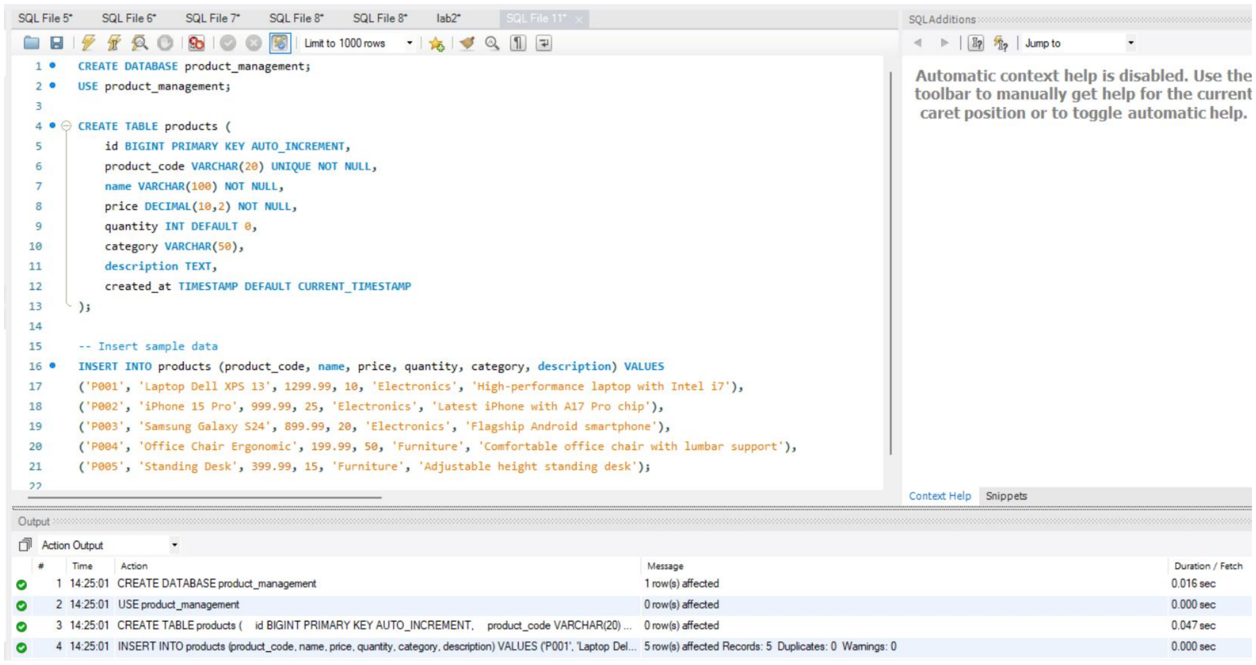
Task 1.2: Database Setup

Create Database and Table:

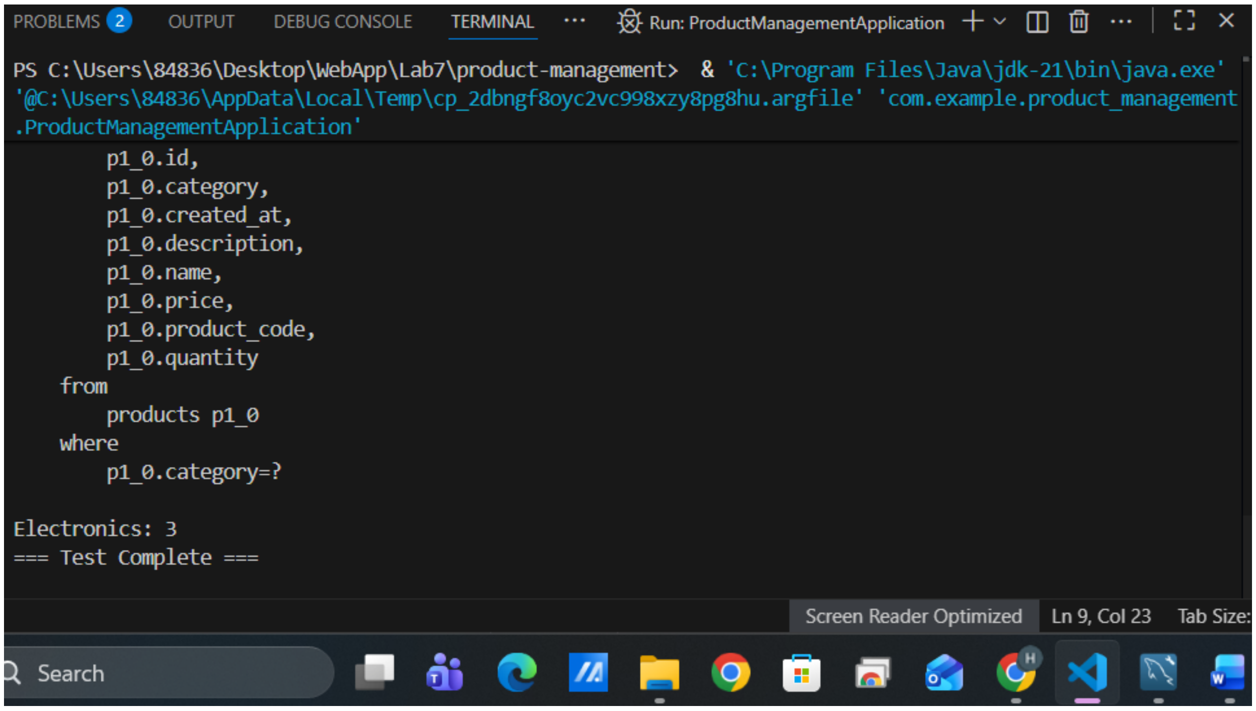


```sql
CREATE DATABASE product_management;
USE product_management;

CREATE TABLE products (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    product_code VARCHAR(20) UNIQUE NOT NULL,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    quantity INT DEFAULT 0,
    category VARCHAR(50),
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Insert sample data
INSERT INTO products (product_code, name, price, quantity, category, description) VALUES
('P001', 'Laptop Dell XPS 13', 1299.99, 10, 'Electronics', 'High-performance laptop with Intel i7'),
('P002', 'iPhone 15 Pro', 999.99, 25, 'Electronics', 'Latest iPhone with A17 Pro chip'),
('P003', 'Samsung Galaxy S24', 899.99, 20, 'Electronics', 'Flagship Android smartphone'),
('P004', 'Office Chair Ergonomic', 199.99, 50, 'Furniture', 'Comfortable office chair with lumbar support'),
('P005', 'Standing Desk', 399.99, 15, 'Furniture', 'Adjustable height standing desk');
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

#	Time	Action	Message	Duration / Fetch
1	14:25:01	CREATE DATABASE product_management	1 row(s) affected	0.016 sec
2	14:25:01	USE product_management	0 row(s) affected	0.000 sec
3	14:25:01	CREATE TABLE products (id BIGINT PRIMARY KEY AUTO_INCREMENT, product_code VARCHAR(20) ...	0 row(s) affected	0.047 sec
4	14:25:01	INSERT INTO products (product_code, name, price, quantity, category, description) VALUES ('P001', 'Laptop Del...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec

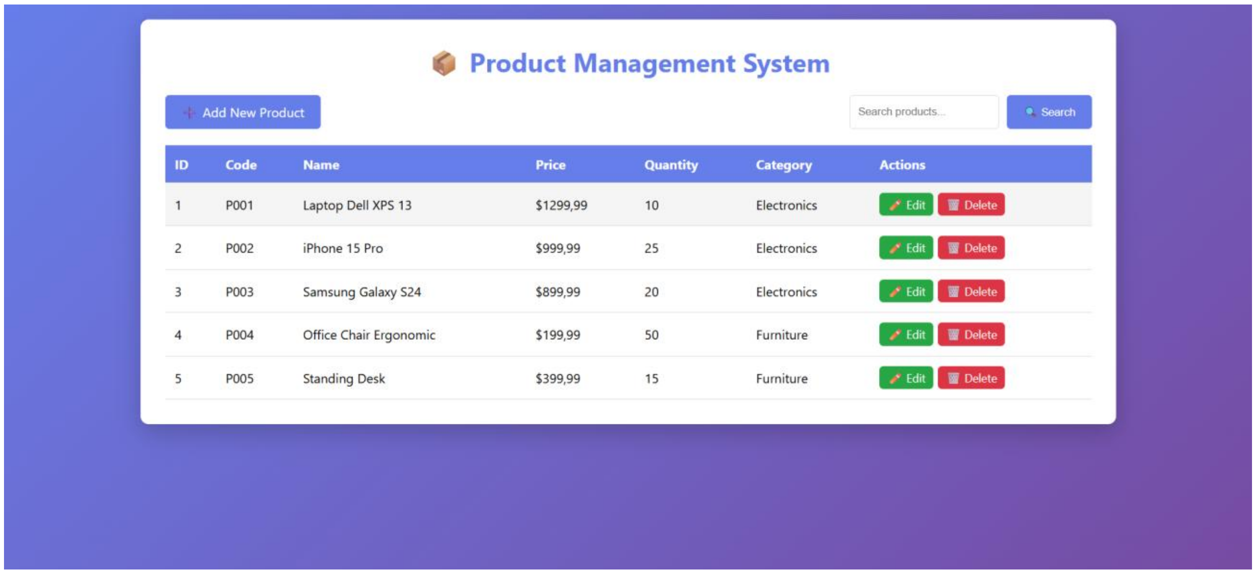
Task 2.3: Test Repository



```
PS C:\Users\84836\Desktop\WebApp\Lab7\product-management>  & 'C:\Program Files\Java\jdk-21\bin\java.exe'
'@C:\Users\84836\AppData\Local\Temp\cp_2dbngf8oyc2vc998xzy8pg8hu.argfile' 'com.example.product_management
.ProductManagementApplication'
        p1_0.id,
        p1_0.category,
        p1_0.created_at,
        p1_0.description,
        p1_0.name,
        p1_0.price,
        p1_0.product_code,
        p1_0.quantity
    from
        products p1_0
    where
        p1_0.category=?

Electronics: 3
=== Test Complete ===
```

List all products:

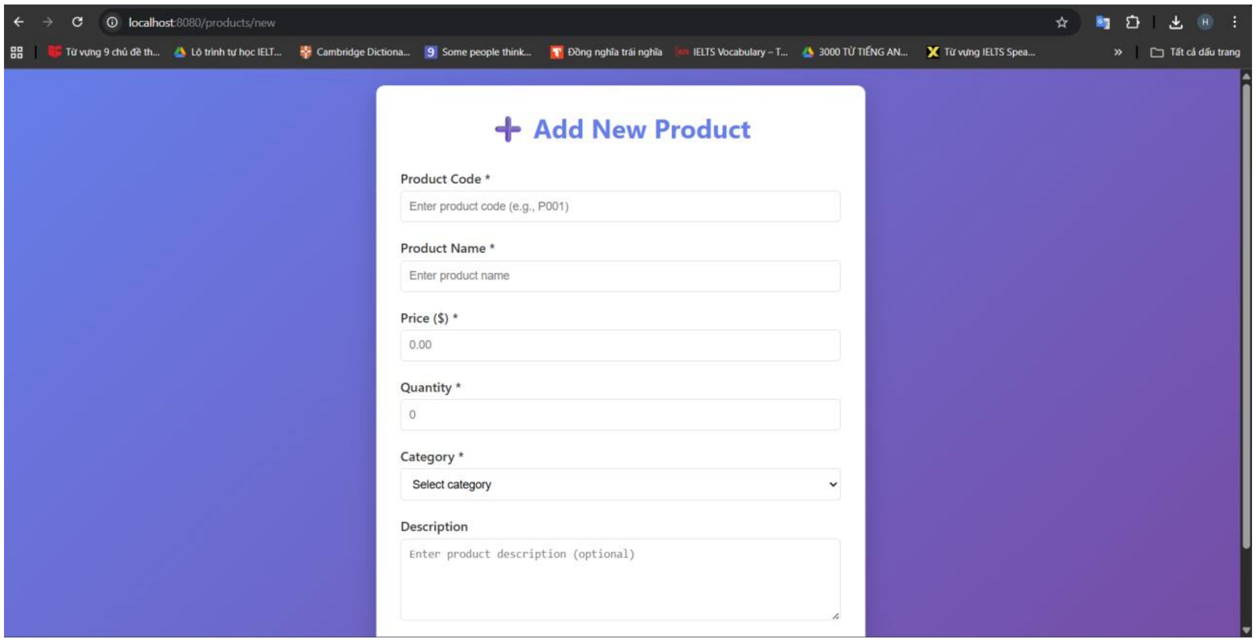
http://localhost:8080/products



📦 Product Management System

ID	Code	Name	Price	Quantity	Category	Actions
1	P001	Laptop Dell XPS 13	$1299,99	10	Electronics	Edit Delete
2	P002	iPhone 15 Pro	$999,99	25	Electronics	Edit Delete
3	P003	Samsung Galaxy S24	$899,99	20	Electronics	Edit Delete
4	P004	Office Chair Ergonomic	$199,99	50	Furniture	Edit Delete
5	P005	Standing Desk	$399,99	15	Furniture	Edit Delete

Add new product: http://localhost:8080/products/new



localhost:8080/products/new

➕ Add New Product

Product Code *
Enter product code (e.g., P001)

Product Name *
Enter product name

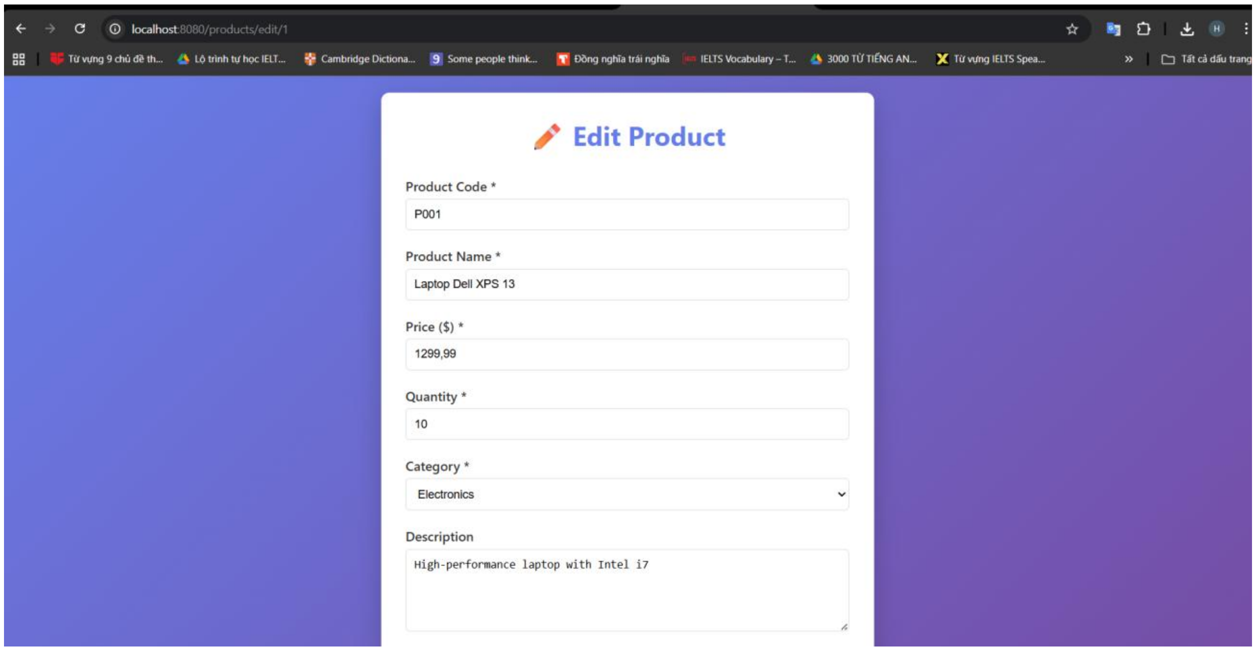
Price ($) *
0.00

Quantity *
0

Category *
Select category

Description
Enter product description (optional)

Edit product (ID=1): http://localhost:8080/products/edit/1



Edit Product

Product Code *
P001

Product Name *
Laptop Dell XPS 13

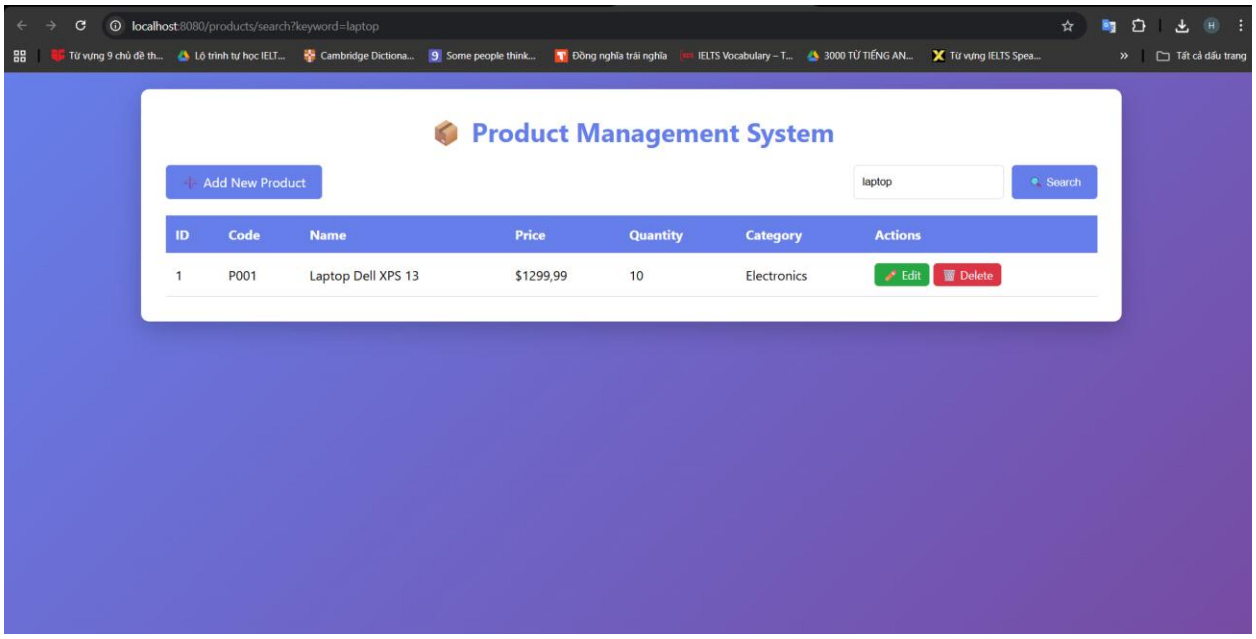
Price ($) *
1299,99

Quantity *
10

Category *
Electronics

Description
High-performance laptop with Intel i7

Search products: http://localhost:8080/products/search?keyword=laptop

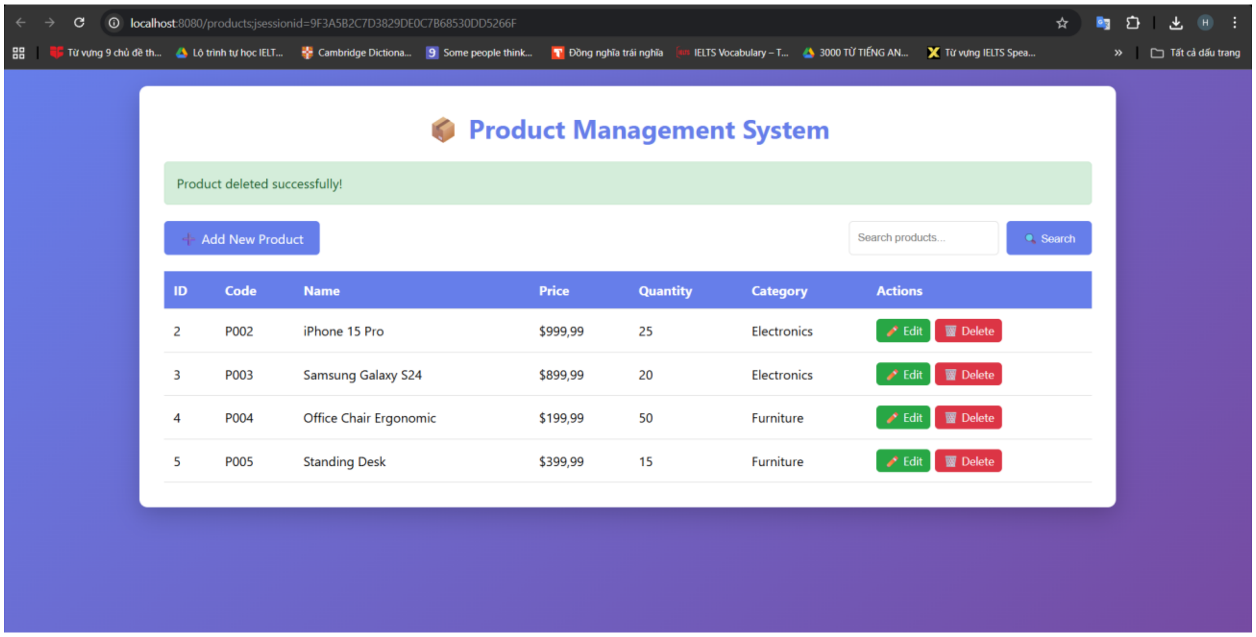


Product Management System

laptop

ID	Code	Name	Price	Quantity	Category	Actions
1	P001	Laptop Dell XPS 13	$1299,99	10	Electronics	Edit Delete

Delete product (ID=1): http://localhost:8080/products/delete/1



Product Management System

Product deleted successfully!

ID	Code	Name	Price	Quantity	Category	Actions
2	P002	iPhone 15 Pro	$999,99	25	Electronics	Edit Delete
3	P003	Samsung Galaxy S24	$899,99	20	Electronics	Edit Delete
4	P004	Office Chair Ergonomic	$199,99	50	Furniture	Edit Delete
5	P005	Standing Desk	$399,99	15	Furniture	Edit Delete

Explain CRUD flow:

1. **CREATE flow (Adding a product)**

This flow involves two steps: showing the empty form, then processing the submission.

Step 1: Showing the form

1. User action: User clicks the "Add New Product" button in product-list.html, which links to @{/products/new}.
2. Controller: The showNewForm method creates a new, empty Product object and binds it to the model, then returns product-form.
3. View: product-form.html renders. The title displays "Add New Product" because the product ID is null.

Step 2: Saving the Data

1. User action: The user fills out the form and clicks "Save Product". The form submits a POST request to @{/products/save}.
2. Controller: The saveProduct method receives the form data mapped to a Product object.
3. Service: Calls productService.saveProduct(product).
   - Inside ProductServiceImpl, productRepository.save(product) is called.

4. Database: Because the Product ID is null, Hibernate performs an INSERT statement.
    - The @PrePersist method in Product.java automatically sets the createdAt timestamp.
5. Result: The controller redirects the user back to /products with a success message.

2. READ flow (Listing products)

The default view when the application starts or the user navigates to the main page.

1. User action: The user accesses /products.
2. Controller: The ProductController.java receives the GET request at the listProducts method.
3. Service: The controller calls productService.getAllProducts().
4. The ProductServiceImpl calls productRepository.findAll().
5. Repository: ProductRepository executes a SQL SELECT * query via JPA.
6. View: The controller adds the list of products to the Model and returns the product-list view. The product-list.html template uses th:each="product : ${products}" to iterate through the list and generate an HTML table row for every product found.

3. UPDATE Flow (Editing a Product)

This flow reuses the Create logic but includes the existing ID.

Step 1: Pre-filling the Form

1. **User Action:** In the product list, the user clicks the "Edit" button next to a specific product. The link is @{/products/edit/{id}}.
2. **Controller:** The showEditForm method extracts the ID from the URL.
3. **Service:** It calls productService.getProductById(id) to find the existing data.
4. **View:** The controller passes this specific Product object to product-form.html.
    - Thymeleaf pre-fills the input fields (Name, Price, etc.) using th:field="*{name}".
    - Crucially, there is a hidden input field: <input type="hidden" th:field="*{id}" />. This ensures the ID is sent back during submission.

Step 2: Saving Changes

1. **User action:** User modifies data and clicks save. The form POSTs to /products/save.

2. **Logic:** The flow is identical to **Create Step 2**, but because the Product object now has an existing ID (from the hidden field), the Repository performs an UPDATE query instead of an INSERT.

4. DELETE Flow

1. **User action:** User clicks the "Delete" button in product-list.html.
   - A JavaScript confirmation dialog (onclick="return confirm...") asks for verification.
   - If confirmed, the browser requests @{/products/delete/{id}}.
2. **Controller:** The deleteProduct method captures the ID.
3. **Service:** Calls productService.deleteProduct(id).
   - ProductServiceImpl calls productRepository.deleteById(id).
4. **Result:** The product is removed from the database, and the controller redirects the user to /products with a flash message "Product deleted successfully!".