VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Web Application Development

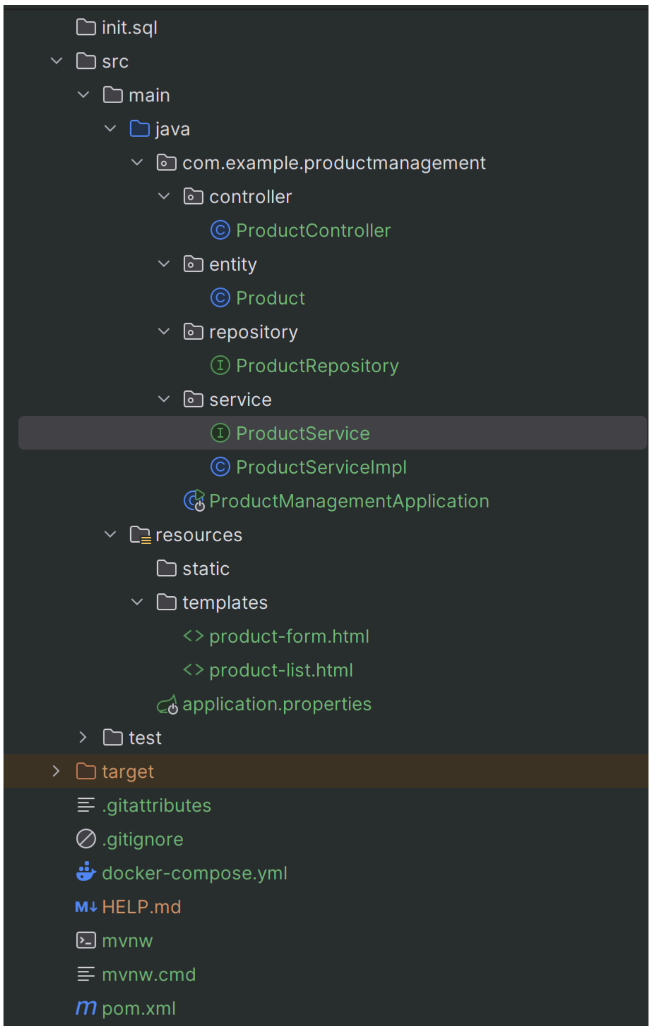
LAB REPORT

LAB 7

Đặng Ngọc Thái Sơn - ITCSIU23033

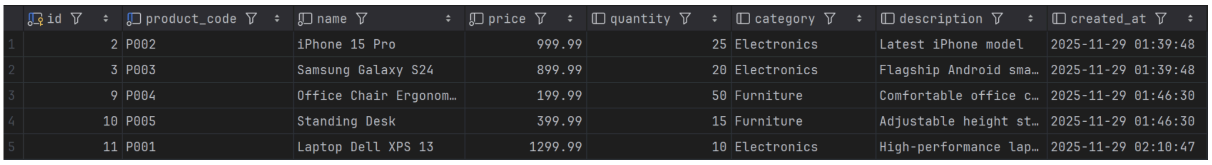
EXERCISE 1: PROJECT SETUP & CONFIGURATION

Task 1.1: Create Spring Boot Project (5 points)



```
    init.sql
  src
    main
      java
        com.example.productmanagement
          controller
            ProductController
          entity
            Product
          repository
            ProductRepository
          service
            ProductService
            ProductServiceImpl
          ProductManagementApplication
      resources
        static
        templates
          product-form.html
          product-list.html
        application.properties
    test
  target
  .gitattributes
  .gitignore
  docker-compose.yml
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Task 1.2: Database Setup (5 points)



	id	product_code	name	price	quantity	category	description	created_at
1	2	P002	iPhone 15 Pro	999.99	25	Electronics	Latest iPhone model	2025-11-29 01:39:48
2	3	P003	Samsung Galaxy S24	899.99	20	Electronics	Flagship Android sma…	2025-11-29 01:39:48
3	9	P004	Office Chair Ergonom…	199.99	50	Furniture	Comfortable office c…	2025-11-29 01:46:30
4	10	P005	Standing Desk	399.99	15	Furniture	Adjustable height st…	2025-11-29 01:46:30
5	11	P001	Laptop Dell XPS 13	1299.99	10	Electronics	High-performance lap…	2025-11-29 02:10:47

Task 1.3: Configure application.properties (5 points)

```
# Application Name
spring.application.name=product-management

# Server Port
```

```
server.port=8080

# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/product_management?useSSL
=false&serverTimezone=UTC&allowPublicKeyRetrieval=true
spring.datasource.username=root
spring.datasource.password=123456
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# JPA/Hibernate Configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

# Thymeleaf Configuration
spring.thymeleaf.cache=false
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html

# Logging
logging.level.org.springframework=INFO
logging.level.com.example.productmanagement=DEBUG
```

- Configurates a Spring Boot application named product-management.
- Set the application to run on server port 8080
- Establish a connection to MySQL database with specified URL, username, password, driver
- Configures **Hibernate/JPA** to:
    - Automatically update the database schema based on entity classes.
    - Show and format SQL statements in the logs for easier debugging.
    - Use the MySQL dialect for SQL compatibility.
- Sets up **Thymeleaf** template engine to:
    - Disable caching for real-time template updates during development.
    - Load HTML templates from the `/templates/` folder in the classpath.
- Defines **logging levels** to:
    - Show informational logs for Spring framework internals.
    - Enable detailed debug logging for the application's own code to facilitate troubleshooting.

EXERCISE 2: ENTITY & REPOSITORY LAYERS

Task 2.1: Create Product Entity

```
package com.example.productmanagement.entity;

import jakarta.persistence.*;
import java.math.BigDecimal;
import java.time.LocalDateTime;

@Entity
```

```
@Table(name = "products")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "product_code", unique = true, nullable = false, length
= 20)
    private String productCode;

    @Column(nullable = false, length = 100)
    private String name;

    @Column(nullable = false, precision = 10, scale = 2)
    private BigDecimal price;

    @Column(nullable = false)
    private Integer quantity;

    @Column(length = 50)
    private String category;

    @Column(columnDefinition = "TEXT")
    private String description;

    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;

    // Constructors
    public Product() {
    }

    public Product(String productCode, String name, BigDecimal price,
Integer quantity, String category, String description) {
        this.productCode = productCode;
        this.name = name;
        this.price = price;
        this.quantity = quantity;
        this.category = category;
        this.description = description;
    }

    // Lifecycle callback
    @PrePersist
    protected void onCreate() {
        this.createdAt = LocalDateTime.now();
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getProductCode() {
        return productCode;
```

```java
    }

    public void setProductCode(String productCode) {
        this.productCode = productCode;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public BigDecimal getPrice() {
        return price;
    }

    public void setPrice(BigDecimal price) {
        this.price = price;
    }

    public Integer getQuantity() {
        return quantity;
    }

    public void setQuantity(Integer quantity) {
        this.quantity = quantity;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }

    @Override
    public String toString() {
        return "Product{" +
                "id=" + id +
                ", productCode='" + productCode + '\'' +
                ", name='" + name + '\'' +
                ", price=" + price +
                ", quantity=" + quantity +
                ", category='" + category + '\'' +
                '}';
    }
}
```

Explanation:

1. **Database Mapping (@Entity, @Table)**
    - **@Entity:** Tells the application (Spring Data JPA) that this class represents a database object.
    - **@Table(name = "products"):** Explicitly states that instances of this class will be stored in a database table named products.
2. **Primary Key (@Id, @GeneratedValue)**
    - **id:** This is the unique identifier for the row.
    - **GenerationType.IDENTITY: T**his tells the database to automatically handle the ID generation
3. **Automatic Timestamping (@PrePersist)**
    - **createdAt:** Stores when the product was added.
    - **onCreate():**
        - The @PrePersist annotation makes this method run automatically right before the data is saved to the database.
        - It sets the createdAt field to the current time (LocalDateTime.now())
        - The updatable = false ensures this timestamp never changes after the initial insert.
4. **Data Fields & Constraints**
    - **productCode**: A unique string to identify the product (like a SKU). The annotation enforces that it cannot be null and must be unique in the database.
    - **price**: Uses **BigDecimal** instead of double or float. This is the industry standard for financial data to prevent rounding errors. The **precision=10, scale=2** allows numbers like **12345678.99.**
    - **quantity**: Stores the stock count.
    - **description**: Marked with columnDefinition = "TEXT", allowing it to store a large amount of text compared to a standard string (VARCHAR).

5. **Boilerplate Code**
    - **Constructors:** Includes a no-argument constructor (required by JPA) and a parameterized constructor for easy object creation.
    - **Getters/Setters**: Allow other parts of the application to read and modify the private fields.
    - **toString**: Provides a string representation of the object, useful for logging and debugging.

Task 2.2: Create Product Repository

```java
package com.example.productmanagement.repository;

import com.example.productmanagement.entity.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.math.BigDecimal;
import java.util.List;

@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)

}
```

Explanation:

- By extending **JpaRepository,** it automatically provides all standard database operations (Save, Delete, Find All, Get by ID) without writing any implementation code.
- The custom methods (like findByCategory or findByPriceBetween) use **Derived Query methods**. Spring Data JPA reads the name of the method and automatically generates the corresponding SQL query.
- The **@Repository** annotation registers this interface so Spring can inject it into the services.

Task 2.3: Test Repository

```
=== Testing Repository ===
Hibernate:
    select
        count(*)
    from
        products p1_0
Total products: 5
Hibernate:
    select
        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
Product{id=2, productCode='P002', name='iPhone 15 Pro', price=999.99, quantity=25,
category='Electronics'}
Product{id=3, productCode='P003', name='Samsung Galaxy S24', price=899.99,
quantity=20, category='Electronics'}
Product{id=9, productCode='P004', name='Office Chair Ergonomic', price=199.99,
quantity=50, category='Furniture'}
Product{id=10, productCode='P005', name='Standing Desk', price=399.99, quantity=15,
category='Furniture'}
Product{id=11, productCode='P001', name='Laptop Dell XPS 13', price=1299.99,
quantity=10, category='Electronics'}
Hibernate:
    select
        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 ===
```

EXERCISE 3: SERVICE LAYER (10 points)

Task 3.1: Create Service Interface (3 points)

```
package com.example.productmanagement.service;
```

```java
import com.example.productmanagement.entity.Product;

import java.util.List;
import java.util.Optional;

public interface ProductService {

   List<Product> getAllProducts();

   Optional<Product> getProductById(Long id);

   Product saveProduct(Product product);

   void deleteProduct(Long id);

   List<Product> searchProducts(String keyword);

   List<Product> getProductsByCategory(String category);
}
```

Define the “contract”—listing what operations are available

Task 3.2: Implement Service (7 points)

```java
package com.example.productmanagement.service;

import com.example.productmanagement.entity.Product;
import com.example.productmanagement.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

@Service
@Transactional
public class ProductServiceImpl implements ProductService {

   private final ProductRepository productRepository;

   @Autowired
   public ProductServiceImpl(ProductRepository productRepository) {
       this.productRepository = productRepository;
   }

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

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

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

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

    @Override
    public List<Product> searchProducts(String keyword) {
        return productRepository.findByNameContaining(keyword);
    }

    @Override
    public List<Product> getProductsByCategory(String category) {
        return productRepository.findByCategory(category);
    }
}
```

- The actual code that fulfills the contract.
- **Transaction Management:** The **@Transactional** annotation ensures data integrity. If an error occurs during a complex operation (like saving multiple items), it rolls back everything so the database isn't left in a corrupt state.

EXERCISE 4: CONTROLLER & VIEWS (15 points)

Task 4.1: Create Product Controller

```java
package com.example.productmanagement.controller;

import com.example.productmanagement.entity.Product;
import com.example.productmanagement.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import java.util.List;

@Controller
@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
   }

   // 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";
               });
   }

   // 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";
   }

   // 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";
   }
```

```
    // Search products
    @GetMapping("/search")
    public String searchProducts(@RequestParam("keyword") String keyword,
Model model) {
         List<Product> products = productService.searchProducts(keyword);
         model.addAttribute("products", products);
         model.addAttribute("keyword", keyword);
         return "product-list";
    }
}
```

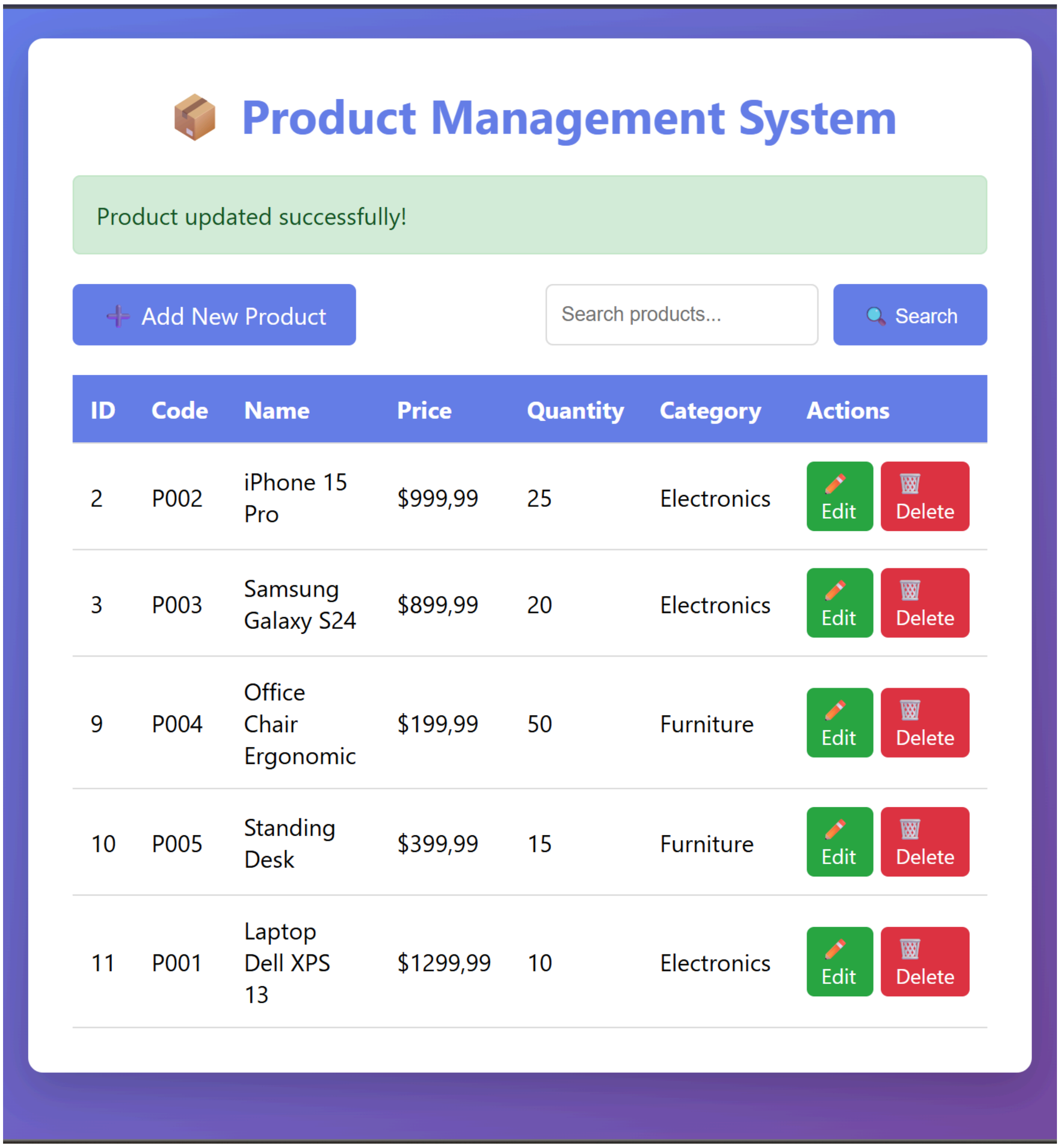
Explanation for visualize the form:

- Use @GetMapping to catch the URL request.
- Call productService to fetch data (or create an empty object).
- Put that data into the Model so the HTML can use it.
- Tell the application to render the HTML template name (e.g., "product-list").

Explanation for save or delete the product:

- Use @PostMapping (or GetMapping for delete) to receive the command.
- Call productService to change the database (Save, Update, or Delete).
- Put a success/error message into RedirectAttributes (so it survives the refresh).
- Command the browser to "redirect:/" to a new URL (usually the list page) to show the updated data.

Task 4.2: Create Product List View

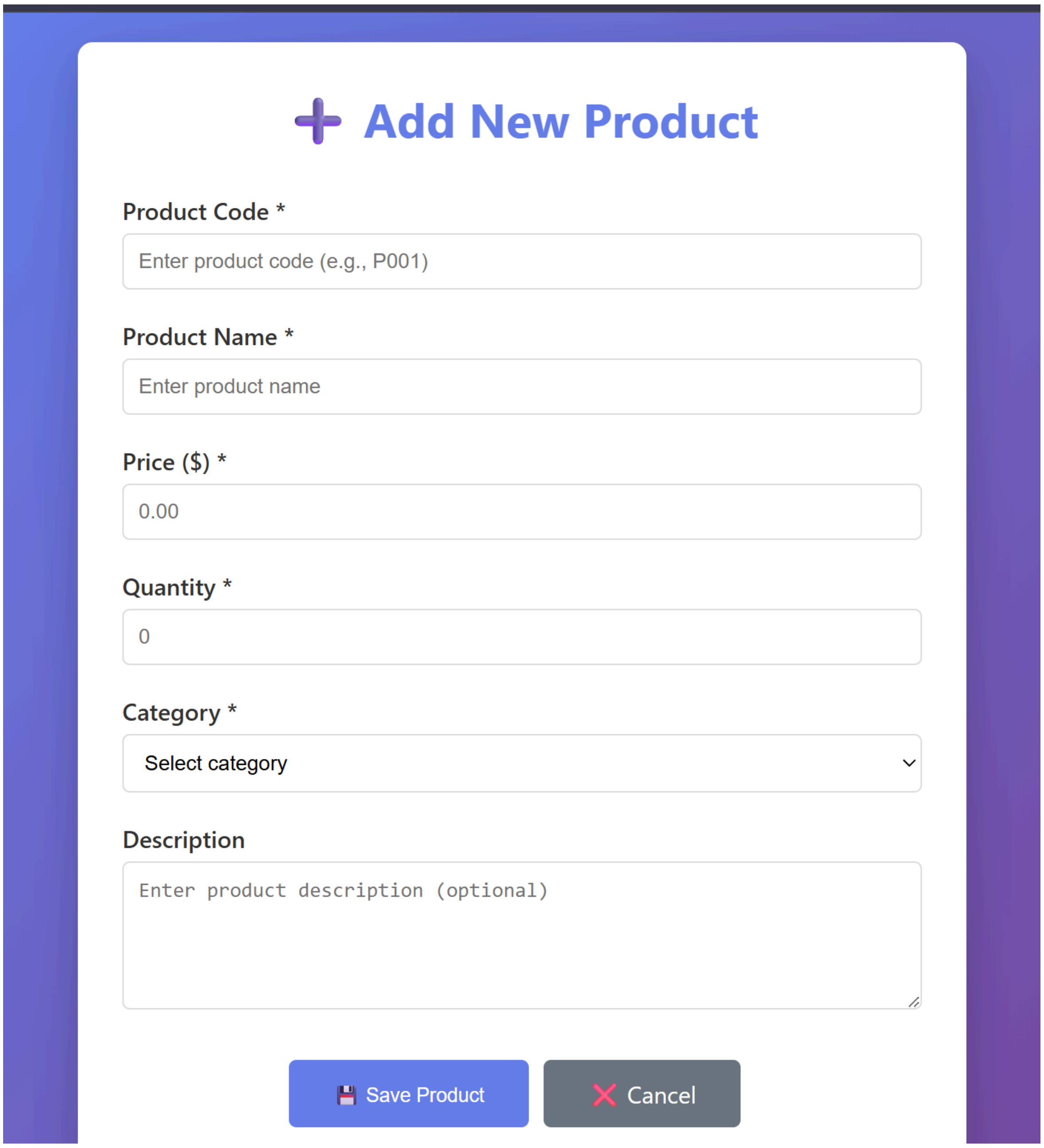


Product Management System

Product updated 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
9	P004	Office Chair Ergonomic	$199,99	50	Furniture	Edit Delete
10	P005	Standing Desk	$399,99	15	Furniture	Edit Delete
11	P001	Laptop Dell XPS 13	$1299,99	10	Electronics	Edit Delete

Task 4.3: Create Product Form View



➕ Add New Product

Product Code *

Enter product code (e.g., P001)

Product Name *

Enter product name

Price ($) *

0.00

Quantity *

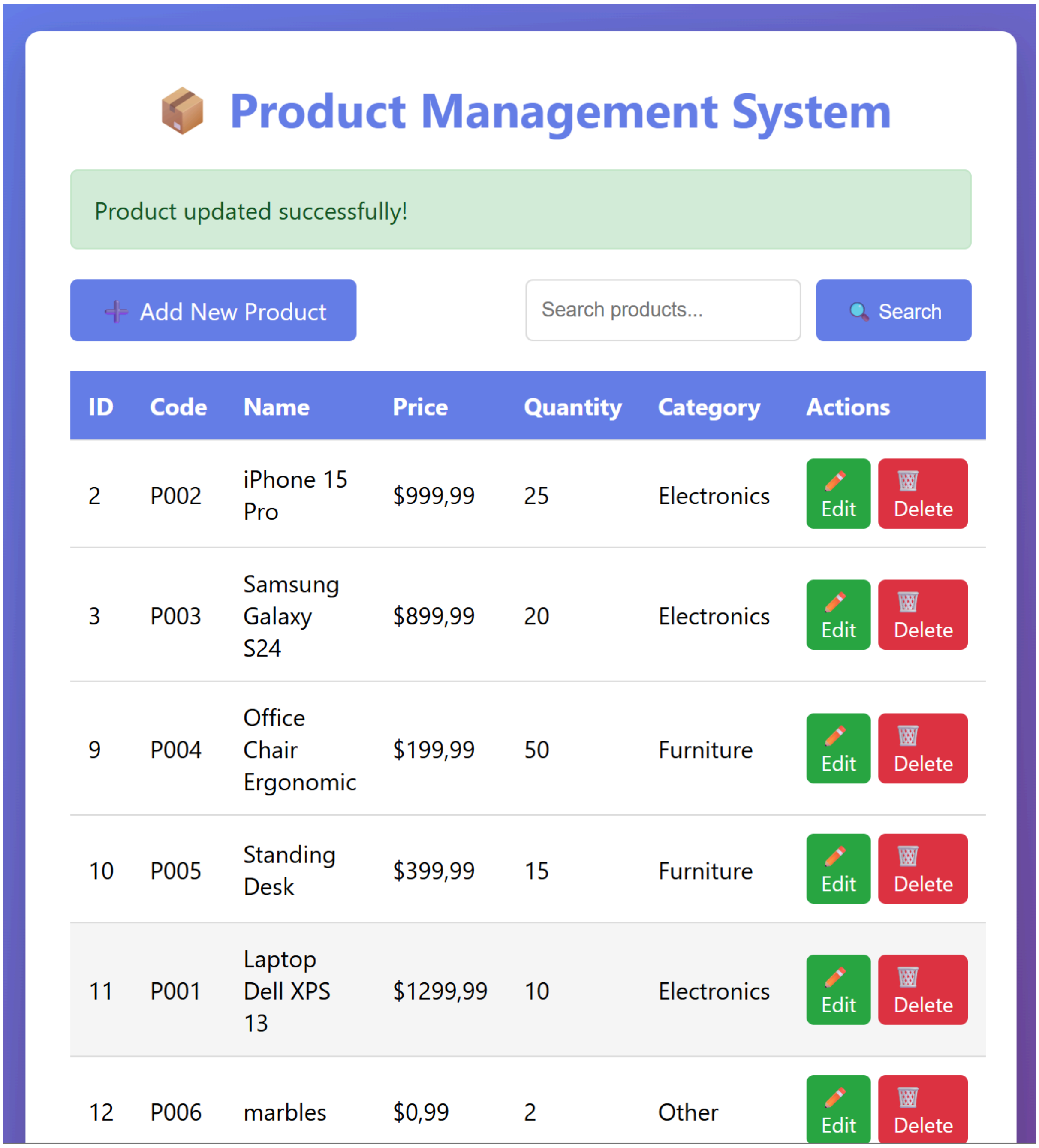
0

Category *

Select category

Description

Enter product description (optional)



Product Management System

Product updated 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
9	P004	Office Chair Ergonomic	$199,99	50	Furniture	Edit Delete
10	P005	Standing Desk	$399,99	15	Furniture	Edit Delete
11	P001	Laptop Dell XPS 13	$1299,99	10	Electronics	Edit Delete
12	P006	marbles	$0,99	2	Other	Edit Delete