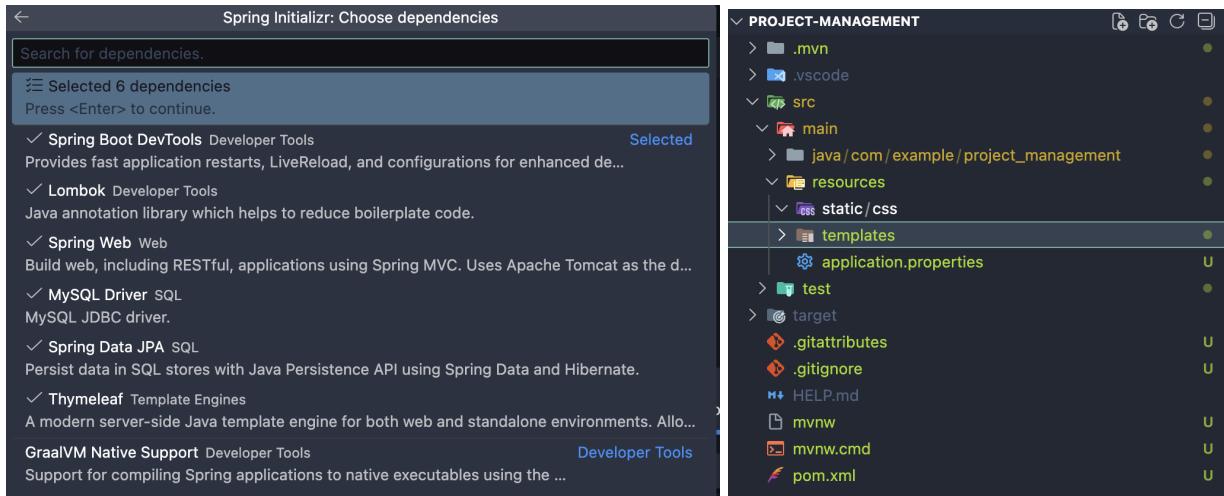


## Lab 7

### Github site & link

<https://github.com/alexis0704/web-dev-practice-lab/tree/main/Lab7/project-management>

### Task 1.1: Create Spring Boot Project



### My selected config:

- Spring Boot: 4.0.0
- Language: Java
- Group Id: com.example
- Artifact Id: product-management
- Packaging: Jar
- Java version: 17

### Project run:



This application has no explicit mapping for /error, so you are seeing this as a fallback.

Mon Dec 01 09:32:37 GMT+07:00 2023

There was an unexpected error (type=NotFound, status=404).

No static resources found.

```
org.springframework.web.servlet.resource.NoResourceNotFoundException: No static resource for request ''.
        at org.springframework.web.servlet.resource.ResourceHttpRequestHandler.handleRequest(ResourceHttpRequestHandler.java:527)
        at org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter.handle(HttpRequestHandlerAdapter.java:50)
        at org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:935)
        at org.springframework.web.servlet.DispatcherServlet.doGet(DispatcherServlet.java:893)
        at org.springframework.web.servlet.FrameworkServlet doGet(FrameworkServlet.java:1003)
        at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:622)
        at org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:874)
        at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:710)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:128)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:107)
        at org.springframework.web.filter.RequestContextFilter.doFilterInternal(RequestContextFilter.java:100)
        at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:116)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:107)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:93)
        at org.springframework.web.filter.CharacterEncodingFilter.doFilterInternal(CharacterEncodingFilter.java:199)
        at org.springframework.web.filter.CharacterEncodingFilter.doFilter(OncePerRequestFilter.java:116)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:107)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:107)
        at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:165)
        at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:77)
        at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:482)
        at org.apache.catalina.core.ErrorReportValve.invoke(ErrorReportValve.java:102)
        at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:72)
        at org.apache.catalina.connector.CoyoteAdaptor.service(CoyoteAdaptor.java:341)
        at org.apache.coyote.ajp13.AjpProcessor.process(AjpProcessor.java:207)
        at org.apache.coyote.AbstractProtocol$ConnectionHandler.process(AbstractProtocol.java:903)
        at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1778)
        at org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:52)
```

## Task 1.2: Database Setup

```
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
);

-- 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'),
    ('P002', 'iPhone 15 Pro', 999.99, 25, 'Electronics', 'Latest iPhone model'),
    ('P003', 'Office Chair', 199.99, 50, 'Furniture', 'Ergonomic office chair');
```

## Task 1.3: Configure application.properties

```
src > main > resources > application.properties
1 # Application Name
2 spring.application.name=product-management
3
4 # Server Port
5 server.port=8085
6
7 # Database Configuration
8 spring.datasource.url=jdbc:mysql://localhost:3306/product_management?useSSL=false&serverTimezone=UTC&
allowPublicKeyRetrieval=true
9 spring.datasource.username=root
10 spring.datasource.password=MyPassword!
11 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
12
13 # JPA/Hibernate Configuration
14 spring.jpa.hibernate.ddl-auto=update
15 spring.jpa.show-sql=true
16 spring.jpa.properties.hibernate.format_sql=true
17 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
18
19 # Thymeleaf Configuration
20 spring.thymeleaf.cache=false
21 spring.thymeleaf.prefix=classpath:/templates/
22 spring.thymeleaf.suffix=.html
23
24 # Logging
25 logging.level.org.springframework=INFO
26 logging.level.com.example.productmanagement=DEBUG
```

## Project Run

```

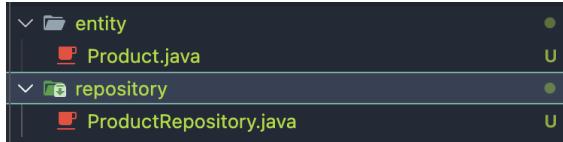
:: Spring Boot ::          (v4.0.0)

2025-12-01T09:12:18.489+07:00  INFO 90474 --- [product-management] | restartedMain c.e.p.ProjectManagementApplication
on : Starting ProjectManagementApplication using Java 25 with PID 90474 (/Users/alexis/Documents/-study/Web Dev /Lab/Practices/Lab/Lab7/project-management/target/classes started by alexis in /Users/alexis/Documents/-study/Web Dev /Lab/Practices/Lab/Lab7/project-management)
2025-12-01T09:12:18.491+07:00  INFO 90474 --- [product-management] | restartedMain c.e.p.ProjectManagementApplication
on : active profile set, falling back to 1 default profile: "default"
2025-12-01T09:12:18.529+07:00  INFO 90474 --- [product-management] | restartedMain c.e.p.DevToolsPropertyDefaultsPostProcessor : Default profile 'default' is active!
2025-12-01T09:12:18.529+07:00  INFO 90474 --- [product-management] | restartedMain c.e.p.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2025-12-01T09:12:18.546+07:00  INFO 90474 --- [product-management] | restartedMain c.e.p.DevToolsPropertyDefaultsPostProcessor : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-12-01T09:12:18.552+07:00  INFO 90474 --- [product-management] | restartedMain s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 31 ms. Found 1 JPA repository interface.
2025-12-01T09:12:18.553+07:00  INFO 90474 --- [product-management] | restartedMain o.s.boot.tomcat.TomcatWebServer : Tomcat initialized with port 8085 (http)
2025-12-01T09:12:19.252+07:00  INFO 90474 --- [product-management] | restartedMain o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-12-01T09:12:19.253+07:00  INFO 90474 --- [product-management] | restartedMain o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/11.0.14]
2025-12-01T09:12:19.275+07:00  INFO 90474 --- [product-management] | restartedMain b.w.c.s.WebApplicationContextInitializer : Root WebApplicationContext: initialization completed in 145 ms
2025-12-01T09:12:19.403+07:00  INFO 90474 --- [product-management] | restartedMain o.hibernate.jpa.internal.util.LogHelper : HHHH000204: Processing PersistenceUnitInfo [name: default]
2025-12-01T09:12:19.403+07:00  INFO 90474 --- [product-management] | restartedMain org.hibernate.Version : HHH000412: Hibernate ORM core version 5.1.8.Final
2025-12-01T09:12:19.403+07:00  INFO 90474 --- [product-management] | restartedMain o.s.o.j.p.SpringPersistenceUnitInitInfo : LoadTimeWeaver setup: ignoring JPA class transformer
2025-12-01T09:12:19.666+07:00  INFO 90474 --- [product-management] | restartedMain com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2025-12-01T09:12:19.778+07:00  INFO 90474 --- [product-management] | restartedMain com.zaxxer.hikari.HikariPool : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@6334642
2025-12-01T09:12:19.778+07:00  INFO 90474 --- [product-management] | restartedMain com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.

```

## Task 2.1, 2.2: Create Product Repository and Create Product Entity

I only pasted in the code from the lab guide.



But in Product.java, I did use Lombok annotations to reduce some of the boilerplate code.

```

1 package com.example.project_management.entity;
2
3 import jakarta.persistence.*;
4 import lombok.Getter;
5 import lombok.NoArgsConstructor;
6 import lombok.Setter;
7
8 import java.math.BigDecimal;
9 import java.time.LocalDateTime;
10
11 @Windsurf: Refactor | Explain
12 @Entity
13 @NoArgsConstructor
14 @Getter
15 @Setter
16 @Table(name = "products")
17 public class Product {
18
19     @Id
20     @GeneratedValue(strategy = GenerationType.IDENTITY)
21     private Long id;
22
23     @Column(name = "product_code", unique = true, nullable = false, length = 20)
24     private String productCode;
25
26     @Column(nullable = false, length = 100)
27     private String name;
28
29     @Column(nullable = false, precision = 10, scale = 2)
30     private BigDecimal price;
31
32     @Column(nullable = false)
33     private Integer quantity;
34
35     @Column(length = 50)
36     private String category;
37
38     @Column(columnDefinition = "TEXT")
39     private String description;
40
41     @Column(name = "created_at", updatable = false)
42     private LocalDateTime createdAt;
43
44     public Product(String productCode, String name, BigDecimal price, Integer quantity, String category,
45                   String description) {
46         this.productCode = productCode;
47         this.name = name;
48         this.price = price;
49         this.quantity = quantity;
50         this.category = category;
51         this.description = description;
52     }
53
54     // Lifecycle callback
55     @PrePersist
56     protected void onCreate() {
57         this.createdAt = LocalDateTime.now();
58     }
59
60     @Override
61     public String toString() {
62         return "Product{" +
63             "id=" + id +
64             ", productCodes='" + productCode + '\'' +
65             ", name='" + name + '\'' +
66             ", price=" + price +
67             ", quantity=" + quantity +
68             ", category='" + category + '\'' +
69             '}';
70     }
71

```

ProductRepository.java interface extends the JpaRepository. The added methods are named according to the convention, so we can have the desired services from JpaRepository.

```

1 package com.example.project_management.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import com.example.project_management.entity.Product;
7
8 import java.math.BigDecimal;
9 import java.util.List;
10
11 Windsurf: Refactor | Explain
12 @Repository
13 public interface ProductRepository extends JpaRepository<Product, Long> {
14     // Spring Data JPA generates implementation automatically!
15
16     // Custom query methods (derived from method names)
17     // Windsurf: Refactor | Explain | X
18     List<Product> findByCategory(String category);
19
20     List<Product> findByNameContaining(String keyword);
21
22     List<Product> findByPriceBetween(BigDecimal minPrice, BigDecimal maxPrice);
23
24     List<Product> findByCategoryOrderByPriceAsc(String category);
25
26     boolean existsByProductCode(String productCode);
27
28     // All basic CRUD methods inherited from JpaRepository:
29     // - findAll()
30     // - findById(Long id)
31     // - save(Product product)
32     // - deleteById(Long id)
33     // - count()
34     // - existsById(Long id)
35 }

```

## Task 2.3: Test Repository

Pasting in the temporary test, I get the following output.

```

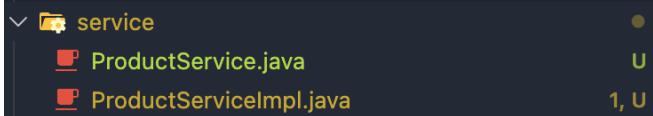
on  : Started ProjectManagementApplication in 2.552 seconds (process running for 2.852)
== Testing Repository ==
Hibernate:
    select
        count(*)
    from
        products p1_0
Total products: 3
Hibernate:
    select
        p1_0.id,
        p1_0.category,
        p1_0.created_at,
        p1_0.description,
        p1_0.name,
        p1_0.price,
        p1_0.product_code,
        p1_0.quantity
    from
        products p1_0
Product{id=1, productCode='P001', name='Laptop Dell XPS 13', price=1299.99, quantity=10, category='Electronics'}
Product{id=2, productCode='P002', name='iPhone 15 Pro', price=999.99, quantity=25, category='Electronics'}
Product{id=3, productCode='P003', name='Office Chair', price=199.99, quantity=50, category='Furniture'}
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: 2
== Test Complete ==

```

## Task 3.1, 3.2: Create Service Interface, Implement Service

Again, I simply pasted in the code from the lab guide.



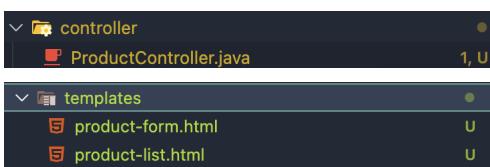
Here, the service class uses the repository's functions to carry out the needed business logic.

```
src > main > java > com > example > project_management > service > ProductService.java > ...
1 package com.example.project_management.service;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import com.example.project_management.entity.Product;
7
8 Windsurf: Refactor | Explain
9 public interface ProductService {
10
11     List<Product> getAllProducts();
12
13     Optional<Product> getProductById(Long id);
14
15     Product saveProduct(Product product);
16
17     void deleteProduct(Long id);
18
19     List<Product> searchProducts(String keyword);
20
21     List<Product> getProductsByCategory(String category);
22 }
```

For the implemented class, there will be @Service annotation and @Transactional annotation (this means that when there's an error, the steps that are carried out will be rolled back)

```
src > main > java > com > example > project_management > service > ProductServiceImpl.java > Java > % ProductServiceImpl > ...
1 package com.example.project_management.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import org.springframework.transaction.annotation.Transactional;
6
7 import com.example.project_management.entity.Product;
8 import com.example.project_management.repository.ProductRepository;
9
10 import java.util.List;
11 import java.util.Optional;
12
13 Windsurf: Refactor | Explain
14 @Service
15 @Transactional
16 public class ProductServiceImpl implements ProductService {
17
18     private final ProductRepository productRepository;
19
20     @Autowired
21     Unnecessary '@Autowired' annotation
22     public ProductServiceImpl(ProductRepository productRepository) {
23         this.productRepository = productRepository;
24     }
25
26     Windsurf: Refactor | Explain | Generate Javadoc | X
27     @Override
28     public List<Product> getAllProducts() {
29         return productRepository.findAll();
30     }
31
32     Windsurf: Refactor | Explain | Generate Javadoc | X
33     @Override
34     public Optional<Product> getProductById(Long id) {
35         return productRepository.findById(id);
36     }
37
38     Windsurf: Refactor | Explain | Generate Javadoc | X
39     @Override
40     public Product saveProduct(Product product) {
41         // Validation logic can go here
42         return productRepository.save(product);
43     }
44
45     Windsurf: Refactor | Explain | Generate Javadoc | X
46     @Override
47     public void deleteProduct(Long id) {
48         productRepository.deleteById(id);
49     }
50
51     Windsurf: Refactor | Explain | Generate Javadoc | X
52     @Override
53     public List<Product> searchProducts(String keyword) {
54         return productRepository.findByNameContaining(keyword);
55     }
56
57     Windsurf: Refactor | Explain | Generate Javadoc | X
58     @Override
59     public List<Product> getProductsByCategory(String category) {
60         return productRepository.findByCategory(category);
61     }
62 }
```

## Task 4.1, 4.2, 4.3: Create Product Controller, Create Product List View, Create Product Form View



Controller maps the correct mapping to appropriate function calls from the service class and returns the corresponding view, with the results obtained from the service.

```
src > main > java > com > example > project_management > controller > ProductController.java
  1 package com.example.project_management.controller;
  2
  3 import org.springframework.beans.factory.annotation.Autowired;
  4 import org.springframework.stereotype.Controller;
  5 import org.springframework.ui.Model;
  6 import org.springframework.web.bind.annotation.*;
  7 import org.springframework.web.servlet.mvc.support.RedirectAttributes;
  8
  9 import com.example.project_management.entity.Product;
 10 import com.example.project_management.service.ProductService;
 11
 12 import java.util.List;
 13
 14 Windsurf: Refactor | Explain
 15 @Controller
 16 @RequestMapping("/products")
 17 public class ProductController {
 18
 19     private final ProductService productService;
 20
 21     @Autowired Unnecessary '@Autowired' annotation
 22     public ProductController(ProductService productService) {
 23         this.productService = productService;
 24     }
 25
 26     // List all products
 27     @GetMapping
 28     public String listProducts(Model model) {
 29         List<Product> products = productService.getAllProducts();
 30         model.addAttribute("products", products);
 31         return "product-list"; // Returns product-list.html
 32     }
 33
 34     // Show form for new product
 35     @GetMapping("new")
 36     public String showNewForm(Model model) {
 37         Product product = new Product();
 38         model.addAttribute("product", product);
 39         return "product-form";
 40     }
 41
 42     // Show form for editing product
 43     @GetMapping("edit/{id}")
 44     public String showEditForm(@PathVariable Long id, Model model, RedirectAttributes redirectAttributes) {
 45         return productService.getProductById(id)
 46             .map(product -> {
 47                 model.addAttribute("product", product);
 48                 return "product-form";
 49             })
 50             .orElseGet(() -> {
 51                 redirectAttributes.addFlashAttribute("error", "Product not found");
 52                 return "redirect:/products";
 53             });
 54     }
 55
 56     // Save product (create or update)
 57     @PostMapping("/save")
 58     public String saveProduct(@ModelAttribute("product") Product product, RedirectAttributes redirectAttributes) {
 59         try {
 60             productService.saveProduct(product);
 61             redirectAttributes.addFlashAttribute("message",
 62                 product.getId() == null ? "Product added successfully!" : "Product updated successfully!");
 63         } catch (Exception e) {
 64             redirectAttributes.addFlashAttribute("error", "Error saving product: " + e.getMessage());
 65         }
 66         return "redirect:/products";
 67     }
 68
 69     // Delete product
 70     @GetMapping("delete/{id}")
 71     public String deleteProduct(@PathVariable Long id, RedirectAttributes redirectAttributes) {
 72         try {
 73             productService.deleteProduct(id);
 74             redirectAttributes.addFlashAttribute("message", "Product deleted successfully!");
 75         } catch (Exception e) {
 76             redirectAttributes.addFlashAttribute("error", "Error deleting product: " + e.getMessage());
 77         }
 78         return "redirect:/products";
 79     }
 80
 81     // Search products
 82     @GetMapping("search")
 83     public String searchProducts(@RequestParam("keyword") String keyword, Model model) {
 84         List<Product> products = productService.searchProducts(keyword);
 85         model.addAttribute("products", products);
 86         model.addAttribute("keyword", keyword);
 87         return "product-list";
 88     }
 89 }
```

The views use the results and show them properly with the help of Thymeleaf  
Example

```
div class="container">
    <h1 th:text="${product.id != null} ? '📝 Edit Product' : '+ Add New Product'">Product Form</h1>

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

## Code run

### Product list

A screenshot of a web browser displaying the 'Product Management System'. The title bar shows the URL 'localhost:8088/products'. The main content area has a purple header with the text 'Product Management System'. Below it is a white card with a table titled 'Product Management System'. The table has columns: ID, Code, Name, Price, Quantity, Category, and Actions. There are three rows of data:

ID	Code	Name	Price	Quantity	Category	Actions
1	P001	Laptop Dell XPS 13	\$1299.99	10	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>

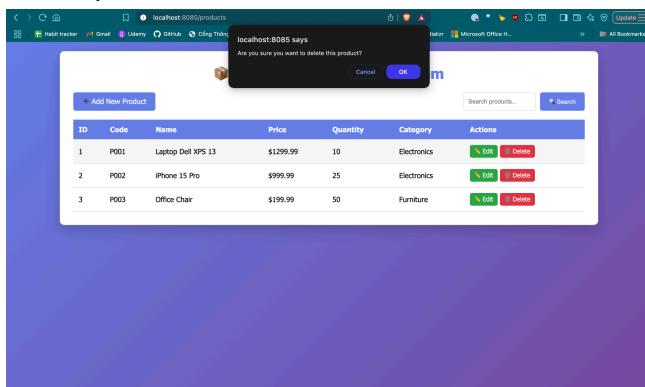
### Add new product

A screenshot of a web browser displaying the 'Add New Product' form. The title bar shows the URL 'localhost:8088/products/new'. The main content area has a white card with the title '+ Add New Product'. It contains several input fields: 'Product Code \*' (with placeholder 'Enter product code (e.g., P001)'), 'Product Name \*' (with placeholder 'Enter product name'), 'Price (\$)' (with placeholder '0.00'), 'Quantity \*' (with placeholder '0'), 'Category \*' (a dropdown menu with 'Select category'), and 'Description' (a text area with placeholder 'Enter product description (optional)'). At the bottom are two buttons: a blue 'Save Product' button and a grey 'Cancel' button.

### Edit product

A screenshot of a web browser displaying the 'Edit Product' form. The title bar shows the URL 'localhost:8088/products/edit/1'. The main content area has a white card with the title 'Edit Product'. It contains the same set of input fields as the 'Add New Product' form: 'Product Code \*' (P001), 'Product Name \*' (Laptop Dell XPS 13), 'Price (\$)' (1299.99), 'Quantity \*' (10), 'Category \*' (Electronics), and 'Description' (High-performance laptop). The 'Save Product' and 'Cancel' buttons are at the bottom.

## Delete product



## Search product

