LAB 7 EXERCISES: SPRING BOOT & JPA CRUD

Name: Nguyễn Nhân Khang

ID: ITITWE22128

EXERCISE 1: PROJECT SETUP & CONFIGURATION

Query 1

Limit to 1000 rows

```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
);

-- 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');
```

Output

Action Output

#	Time	Action	Message
1	08:56:53	CREATE DATABASE product_management	1 row(s) affected
2	08:56:53	USE product_management	0 row(s) affected
3	08:56:53	CREATE TABLE products (id BIGINT PRIMARY KEY AUTO_INCREMENT, product_code VARCHAR(20) ...	0 row(s) affected
4	08:56:53	INSERT INTO products (product_code, name, price, quantity, category, description) VALUES ('P001', 'Laptop Del...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0

Task 1.3: Configure application.properties

application.properties | pom.xml | Product.java 1 | ProductRepository.java 1 | DemoApplication.java 1 | product-list.html | product-f

demo > src > main > resources > application.properties

```properties
# Application Name
spring.application.name=demo

# Server Port
server.port=8082

# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/product_management?useSSL=false&serverTimezone=UTC&allowPublicKeyRetrieval=true
spring.datasource.username=root
spring.datasource.password=1234567
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.demo=DEBUG
```

EXERCISE 2: ENTITY & REPOSITORY LAYERS

```java
package com.example.demo.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;
```

```
    }

    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 + '\'' +
                '}';
    }
}
```

Explain:

Key Components:

- **Identity:**
  - `@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`: The primary key (`id`) is auto-incremented and managed directly by the Database.
- **Data Constraints:**
  - `productCode`: Must be unique (`unique=true`), mandatory (non-nullable), with a maximum length of 20 characters.
  - `price`: Utilizes `BigDecimal` to ensure monetary precision (critical for financial applications), defined with a total precision of 10 digits and a scale of 2 decimal places.
  - `createdAt`: Marked as `updatable = false` to ensure the creation timestamp remains immutable (cannot be changed) after it has been persisted.
- **Lifecycle Callback:**
  - `@PrePersist`: The `onCreate()` method executes automatically immediately before the data is persisted to the Database for the first time. This guarantees that `createdAt` is always populated with the current real-time value without requiring manual setting from the Controller.

Task 2.2: Create Product Repository

Code:

demo > src > main > java > com > example > demo > repository > ProductRepository.java > Language Support for Java(TM) by Red Hat > ProductRepository

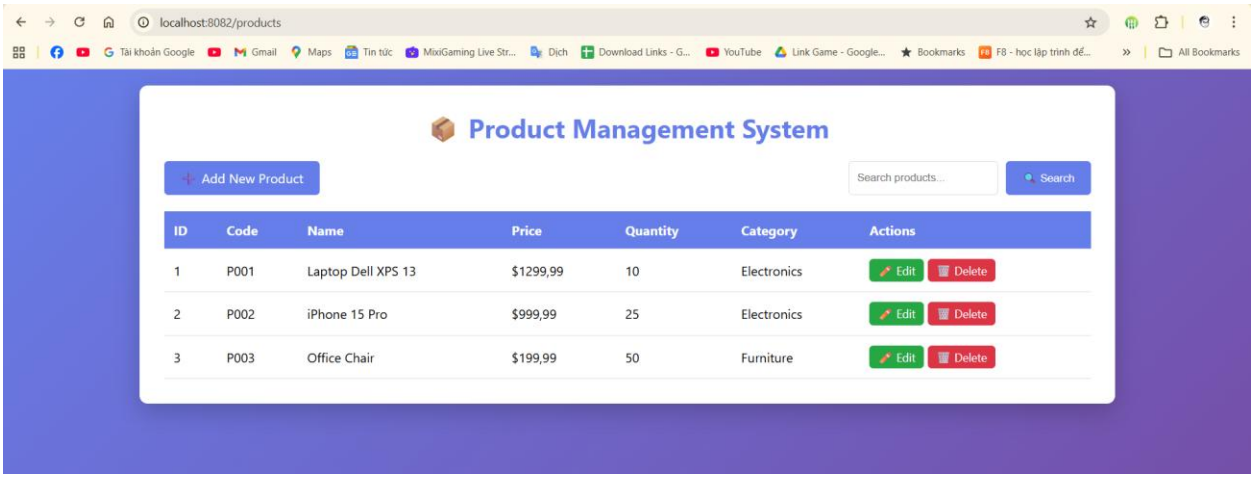
```java
package com.example.demo.repository;

import com.example.demo.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)
```

Explain:

Key Components:

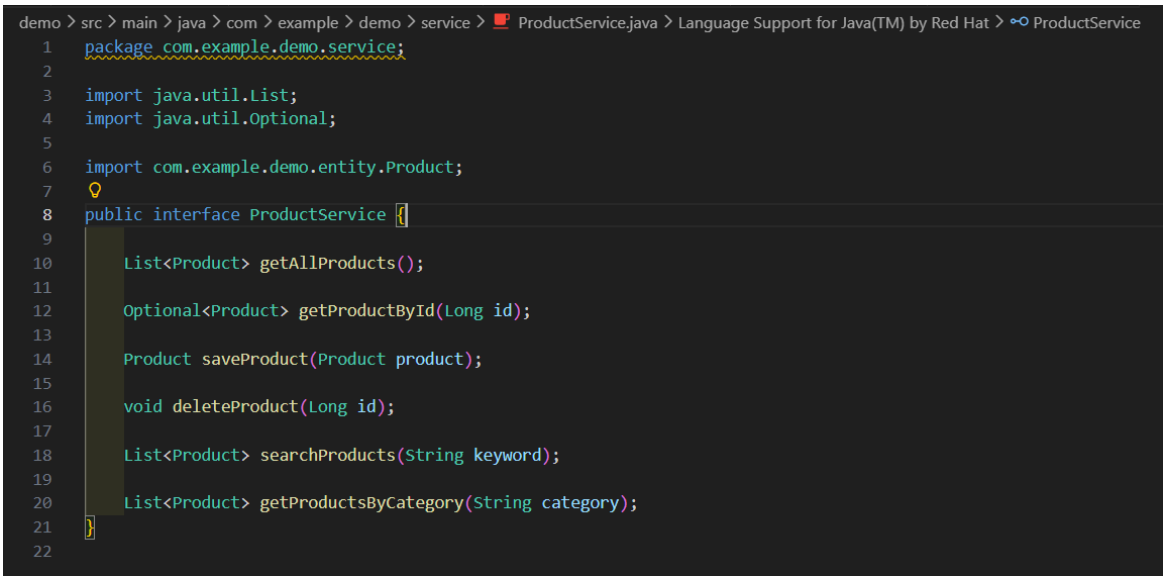
- **@Repository:**
  - Marks this interface as a **Spring Bean** belonging to the Repository layer (Data Access Layer).
  - Enables **Exception Translation**: Converts raw SQL errors (`SQLException`) into Spring's consistent unchecked exceptions (`DataAccessException`), making error handling logic significantly easier.
- **extends JpaRepository<Product, Long>:**
  - **Product:** Specifies the **Entity** type that this repository manages.
  - **Long:** Specifies the data type of the **Primary Key** (`@Id`) within the `Product` Entity.
  - **Benefit:** Automatically provides standard **CRUD** methods (Create, Read, Update, Delete) such as `save()`, `findAll()`, and `deleteById()` without requiring a single line of implementation code.

Task 2.3: Test Repository



EXERCISE 3: SERVICE LAYER

Task 3.1: Create Service Interface



Task 3.2: Implement Service

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

import com.example.demo.entity.Product;
import com.example.demo.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);
    }

    @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);
    }
}
```

EXERCISE 4: CONTROLLER & VIEWS

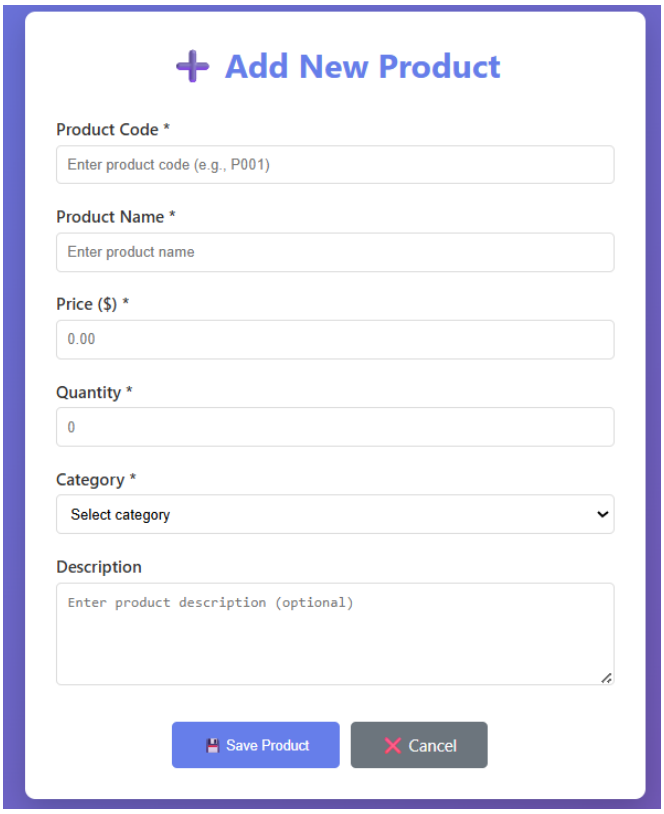
Task 4.1: Create Product Controller

```
package com.example.demo.controller;

import com.example.demo.entity.Product;
import com.example.demo.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";
    }
}
```

Task 4.2: Create Product List View

Task 4.3: Create Product Form View



Expalin the code flow with MVC:

Step 1: User Requests Edit Form (GET Request)

- **Browser:** The user clicks the link `http://localhost:8080/products/edit/5`.
- **Controller:** Spring maps this request to the `showEditForm(@PathVariable Long id, ...)` method. The `id` variable receives the value `5`.
- **Service:** The Controller calls `productService.getProductById(5)`.
- **Repository:** The Service calls `productRepository.findById(5)`.
- **Database:** Returns the record where ID=5.
- **Controller:**
    - *If found:* Adds the `product` object to the Model and returns the `"product-form"` view.
    - *If not found:* Adds an error message to `RedirectAttributes` and redirects back to the product list.
- **View (Thymeleaf):** Renders the `product-form.html` file, populating the input fields (`th:field`) with data from Product ID 5.

Step 2: User Modifies and Clicks Save (POST Request)

- **Browser:** Sends form data (code, name, price, etc.) to `POST /products/save`.
- **Controller:** Binds the incoming data to a `Product` object using `@ModelAttribute`. **Note:** This object contains `id=5` (retrieved from the hidden field).
- **Service:** The Controller calls `productService.saveProduct(product)`.
- **Repository:** The Service calls `productRepository.save(product)`.
- **Hibernate Logic:** Since the `id` (5) already exists in the Database, Hibernate executes an **UPDATE** statement (instead of INSERT).
- **Controller:** Adds a "Saved successfully" message to flash attributes and redirects to `/products`.
- **Browser:** Reloads the product list page, displaying the updated data and the success message.