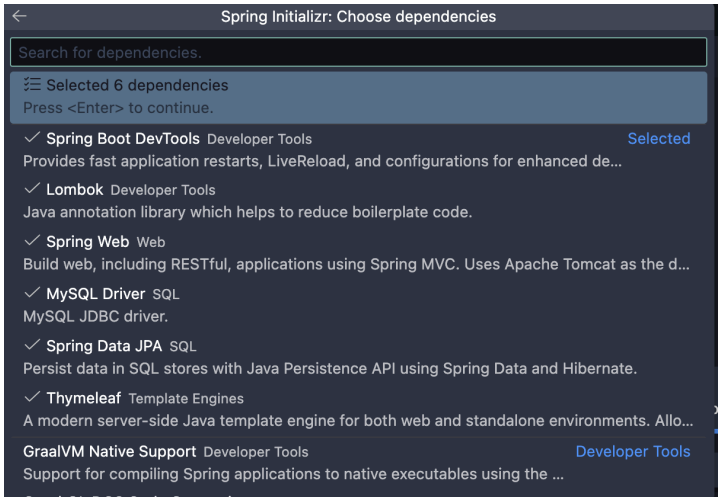
ITITWE23039 - Phan Kim Vĩnh Phúc

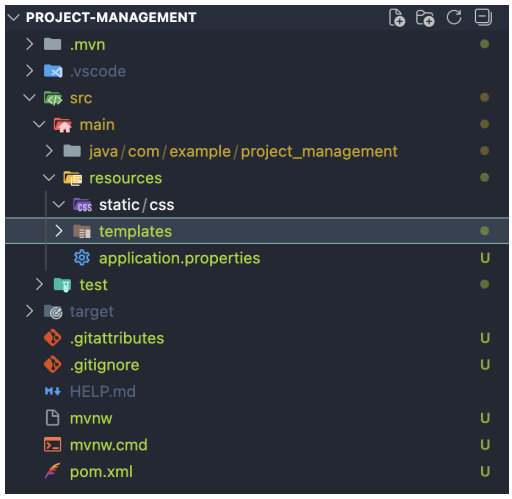
Lab 7

Github site & link

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

Task 1.1: Create Spring Boot Project

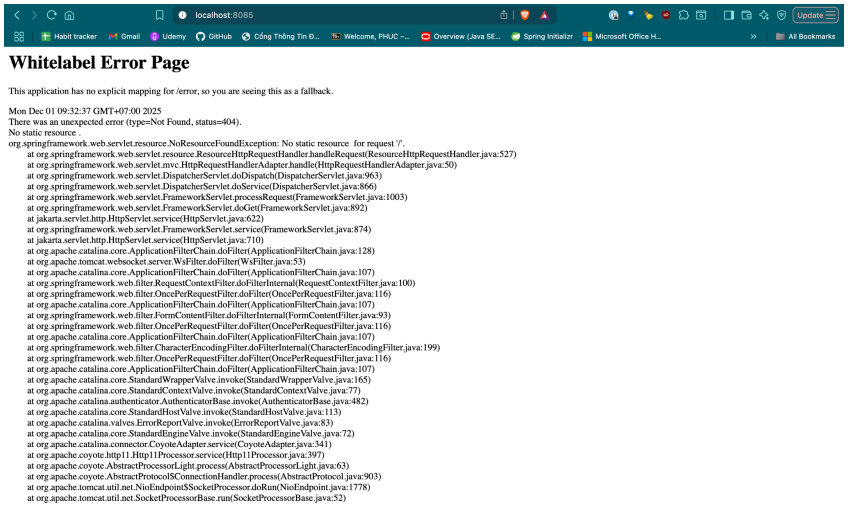




My selected config:

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

Project run:



Task 1.2: Database Setup

```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');
```

Task 1.3: Configure application.properties

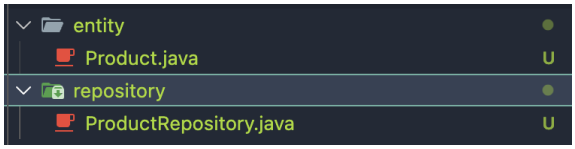
src > main > resources > application.properties

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

# Server Port
server.port=8085

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

Project Run

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.

```java
package com.example.project_management.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.math.BigDecimal;
import java.time.LocalDateTime;

Windsurf: Refactor | Explain
@Entity
@NoArgsConstructor
@Getter
@Setter
@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;
```

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

    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
    Windsurf: Refactor | Explain | ×
    @PrePersist
    protected void onCreate() {
        this.createdAt = LocalDateTime.now();
    }

    Windsurf: Refactor | Explain | Generate Javadoc | ×
    @Override
    public String toString() {
        return "Product{" +
                "id=" + id +
                ", productCode='" + productCode + '\'' +
                ", name='" + name + '\'' +
                ", price=" + price +
                ", quantity=" + quantity +
                ", category='" + category + '\'' +
                '}';
    }
}
```

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

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

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.example.project_management.entity.Product;

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

Windsurf: Refactor | Explain
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    // Spring Data JPA generates implementation automatically!

    // Custom query methods (derived from method names)
    Windsurf: Refactor | Explain | ×
    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)
}

```

Task 2.3: Test Repository

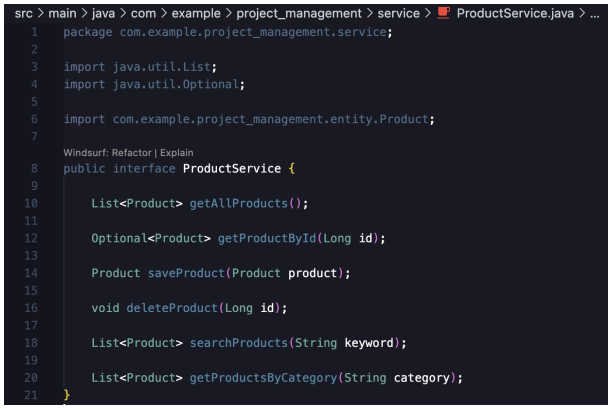
Pasting in the temporary test, I get the following output.

```
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 ===
```

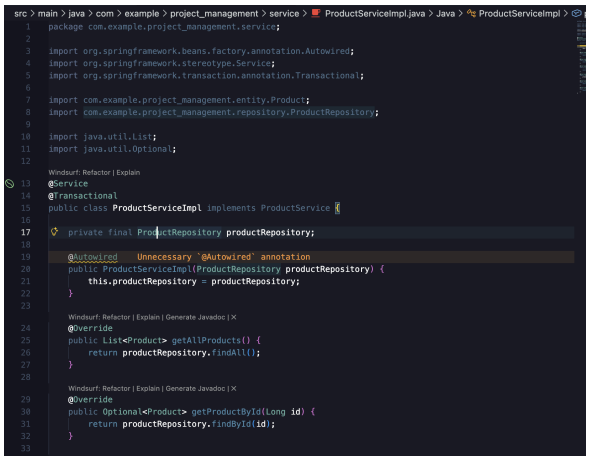
Task 3.1, 3.2: Create Service Interface, Implement Service

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

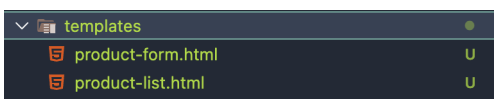
Here, the service class uses the repository's functions to carry out the needed business logic.



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)



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



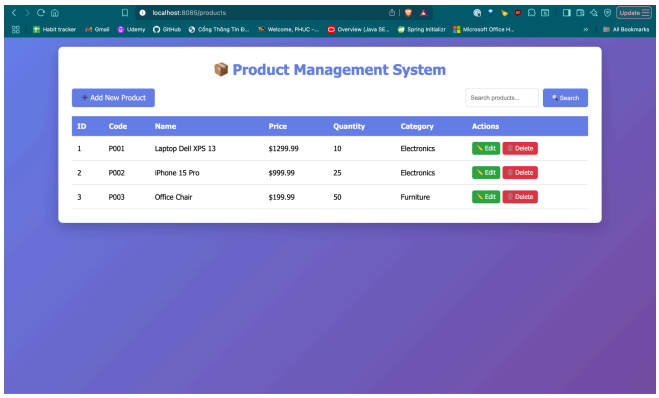
Controller maps the correct mapping to appropriate function calls from the service class and returns the corresponding view, with the results obtained from the service.

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

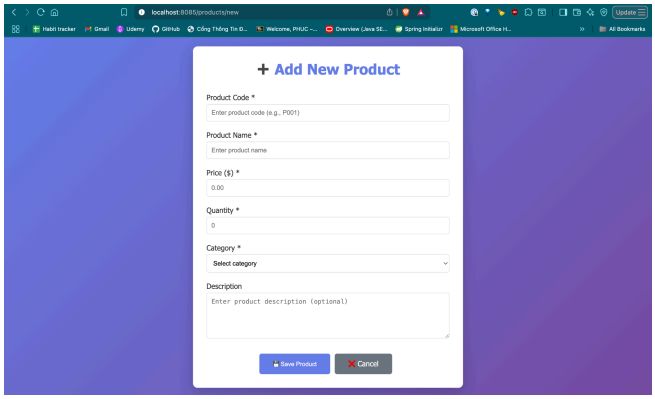
```
>
div class="container">
   <h1 th:text="${product.id != null} ? '✏️ Edit Product' : '➕ Add New Product'">Product Form</h1>

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

Code run

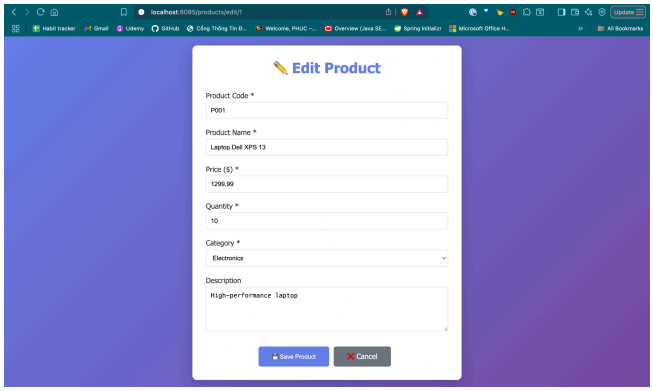
Product list



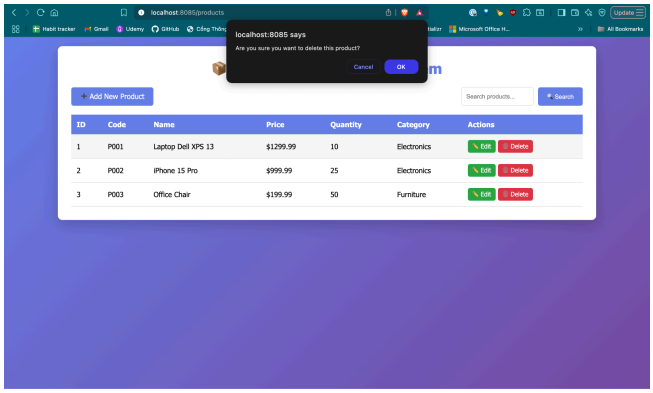
Add new product



Edit product



Delete product



Search product