Lab 7: Detailed Code Explanation

Here is the detailed line-by-line explanation of the key files involved in Exercises 1-4, matching your specific codebase.

Exercise 1: Project Configuration

```
pom.xml
```

This file defines the project dependencies.

```xml
<dependencies>
    <!-- 1. Spring Web: For building RESTful APIs -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- 2. Spring Data JPA: For database access -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- ... other dependencies ... -->
</dependencies>
```

```
src/main/resources/application.properties
```

This file configures the database connection.

```properties
# Database Connection details
spring.datasource.url=jdbc:mysql://localhost:3306/product_management
spring.datasource.username=root
spring.jpa.hibernate.ddl-auto=update
```

Exercise 2: Entity Class

```
src/main/java/com/example/product_management/entity/Product.java
```

Mapping the Java class to the 'products' table.

```java
package com.example.product_management.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)
    private String productCode;

    @Column(nullable = false)
    private String name;

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

    // ... (other fields omitted for brevity) ...

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

Exercise 3: Repository Interface

`src/main/java/com/example/product_management/repository/ProductRepository.java`

This is where your specific custom search methods are defined.

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

import com.example.product_management.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> {

    // 1. Find by Category
    // Spring automatically creates SQL: SELECT * FROM products WHERE
category = ?
    List<Product> findByCategory(String category);

    // 2. Find by Name containing a keyword
    // SQL: SELECT * FROM products WHERE name LIKE %keyword%
    List<Product> findByNameContaining(String keyword);

    // 3. Find by Price Range
    // SQL: SELECT * FROM products WHERE price BETWEEN min AND max
    List<Product> findByPriceBetween(BigDecimal minPrice, BigDecimal
maxPrice);

    // 4. Find by Category and Sort by Price
    // SQL: SELECT * FROM products WHERE category = ? ORDER BY price ASC
    List<Product> findByCategoryOrderByPriceAsc(String category);

    // 5. Check if Product Code Exists
    // SQL: SELECT count(*) > 0 FROM products WHERE product_code = ?
    boolean existsByProductCode(String productCode);
}
```

Exercise 4: Service Layer

```
src/main/java/com/example/product_management/service/ProductSer
vice.java
```

This interface defines the business operations available.

```
public interface ProductService {
    List<Product> getAllProducts();           // Get list of all products
    Optional<Product> getProductById(Long id); // Get one product
    Product saveProduct(Product product);     // Create or Update
    void deleteProduct(Long id);              // Delete
    List<Product> searchProducts(String keyword); // Search
    List<Product> getProductsByCategory(String category); // Filter
}
```

```
src/main/java/com/example/product_management/service/ProductSer
viceImpl.java
```

The implementation class contains the actual business logic and transaction management.

```
@Service
@Transactional
public class ProductServiceImpl implements ProductService {

    private final ProductRepository productRepository;

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

    @Override
    public List<Product> searchProducts(String keyword) {
        // Uses the custom method from the repository provided above
        return productRepository.findByNameContaining(keyword);
    }

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

    // ... other standard CRUD methods (findAll, save, delete) ...
}
```