**Lab 7**

**Name:** Trần Gia Khôi

**ID:** ITITIU21231

**Exercise 1: Project Setup & Configuration**

**Task 1.1: Create Spring Boot Project**

Project created with correct structure

**A screenshot of a computer program

AI-generated content may be incorrect.**

The project has been created with correct structure. The “Product” is a **Model** so it will locate at *entity* package. “ProductController” is a **Controller** so it will locate at *controller* package. The same go with “ProductRepository”, “ProductService” and “ProductServiceImpl”.

All dependencies added

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

All required dependencies have been added such as “spring-boot-starter-data-jpa”, which is Spring Data JPA and “spring-boot-devtools”, which is Spring Boot DevTools.

**Task 1.2: Database Setup**

Database created

A screenshot of a computer

AI-generated content may be incorrect.

The message in the above picture indicates that the database “product\_management” has been created successfully.

Table structure correct

A screenshot of a computer

AI-generated content may be incorrect.

The table has been structured correctly with total 8 columns.

Sample data inserted

A screenshot of a computer

AI-generated content may be incorrect.

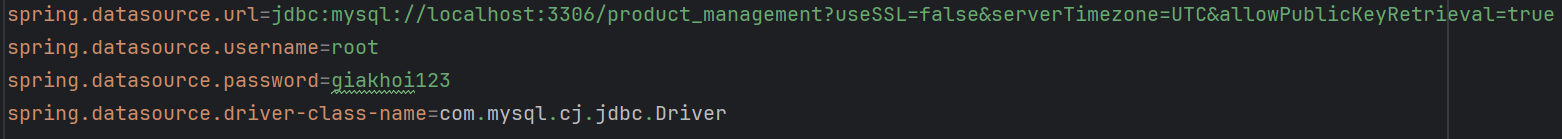
I insert one data which has its name is “Xiaomi 17T Pro”.

**Task 1.3: Configure application.properties**

A screenshot of a computer program

AI-generated content may be incorrect.

Database connection configured



The application.properties file has been configured correctly. For example in the database configuration, the username and password has been provide correctly as well as datasource url.

JPA properties set correctly

A screen shot of a computer code

AI-generated content may be incorrect.

**Check point #1**

A screen shot of a computer code

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

As we can see, the project run perfectly without errors. Since Tomcat started on port 8080 successfully. And the message “Started ProductManagementApplication in 2.617 seconds” in the console indicates that the project has been built perfectly.

**Exercise 2: Entity & Repository Layers**

**Task 2.1: Create Product Entity**

All JPA annotations correct

A black background with text

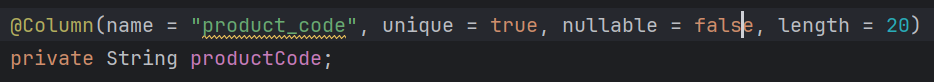
AI-generated content may be incorrect.

An “@Entity” annotation is used to defined a JPA entity that represents data that can be persisted in a database. A “@Table” annotation is used to specify the details of the table that will be used to persist an entity in the database.

A black background with white text

AI-generated content may be incorrect.

An “@Id” annotation is a key component of JPA and is used to designate the primary key of an entity. A “@GeneratedValue” annotation is used in conjunction with the “@Id” annotation to automatically generate unique values for primary key columns in a database tables.



A “@Column” annotation is used to customize the mapping of Java class fields to database columns.

Proper data types used

A computer screen shot of a program

AI-generated content may be incorrect.

For example, *id* is set with a **Long** type. A *name, category, description* is set with **String** type.

Constructors implemented

A screen shot of a computer

AI-generated content may be incorrect.

Getters/setters complete

A screen shot of a computer program

AI-generated content may be incorrect.

A black screen with text and colorful text

AI-generated content may be incorrect.

For the setters, the **Lombok** annotation “Setters” has generated for us.

@PrePersist lifecycle callback

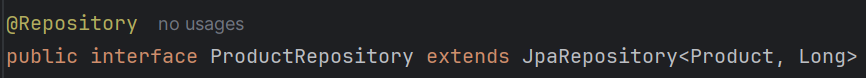
A computer screen shot of text

AI-generated content may be incorrect.

A “@PrePersist” is used to define a method that should be executed before an entity is persisted into the database.

**Task 2.2: Create Product Repository**

Extend JpaRepository correctly



Custom query methods correct

A screen shot of a computer program

AI-generated content may be incorrect.

By extending JpaRepository, we do not need to write a SQL query by hand, the custom query methods from JpaRepository have done it for us automatically.

Method naming conventions followed

The pattern naming conventions is “findBy + PropertyName + Condition”



For example in this custom method, we will *findBy* category (PropertyName) where condition is *category.*

**Task 2.3: Test Repository**

Repository methods work correctly

A screen shot of a computer

AI-generated content may be incorrect.

Data retrieved from database



**Check point #2:**

By defining fields as well as using annotations like “@Id”, “@Table”, “@Column” properly in Product (Entity), the ProductRepository (Repository) is able to query in the table “products” and use its custom methods to find specific product. For example, the method *findByCategory* has successfully find the product with category “Electronics” and return the list of products that have “Electronics” as category.

**Exercise 3: Service Layer**

**Task 3.1: Create Service Interface**

A screen shot of a computer program

AI-generated content may be incorrect.

The ProductService file has been created with necessary methods and ready to be implemented.

**Task 3.2: Implement Service**

@Service and @Transactional annotations

A black screen with white text

AI-generated content may be incorrect.

A “@Service” annotation is used to mark a class as a service provider, indicating that it holds the business logic of the application.

A “@Transactional” is used to manage database transactions automatically.

Constructor injection implemented

A black background with white text

AI-generated content may be incorrect.

Constructor injection is a technique in Spring where dependencies are provided to a class at the time of its instantiation. This method ensures that the class is always in a valid state with all its dependencies properly initialized.

So by using constructor injection, the *productRepository*will be always in a valid state with all its dependencies initialized.

All methods implemented correctly

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

By calling custom methods in ProductRepository, the ProductRepository will connect to database and perform some database behavior then return the result back to Service class, which is ProductServiceImpl.

**Exercise 4: Controller & Views**

**Task 4.1: Create Product Controller**

All request mapping correct

A black background with white text and words

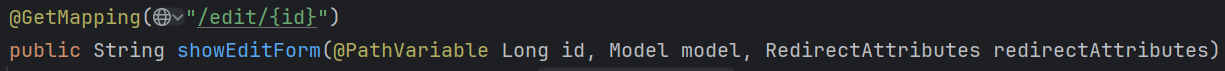
AI-generated content may be incorrect.

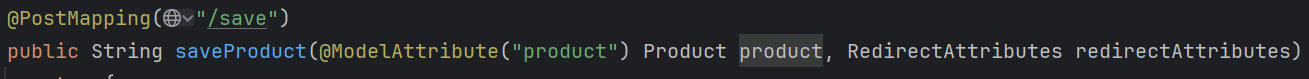
A black background with blue text

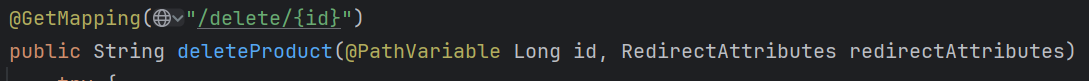
AI-generated content may be incorrect.

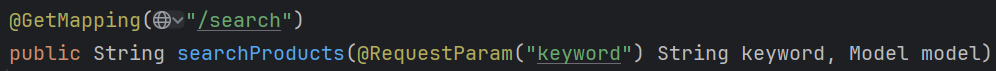
A black background with blue and green text

AI-generated content may be incorrect.









The request mapping is “/products” and its corresponding URL endpoints in a GET and POST method.

Service injection works

A screen shot of a computer

AI-generated content may be incorrect.

By using service injection, the class ProductService will be always in a valid state with all its dependencies properly initialized.

List products implemented

A screen shot of a computer code

AI-generated content may be incorrect.

The method *listProducts* has been implemented. Its will call the method *getAllProducts* from ProductService and then let ProductService handle the logic business.

New/Edit forms work

A screen shot of a computer program

AI-generated content may be incorrect.

For the new form, it simply returns the “product-form” view to user. For edit form, it will call the method *getProductById()* from ProductService and let it handle the logic business.

Save functionality works

A screenshot of a computer program

AI-generated content may be incorrect.

This method will call a method *saveProduct()* from ProductService and let it handle the logic business. After saving we have two results, which are success and failed. If it succeeds, the successful message will appear. However, if it fails, the error message will be displayed. Both results will redirect user back to the view “product-list” by using “redirect:/products”.

Delete functionality works

A screenshot of a computer screen

AI-generated content may be incorrect.

The method also call the method from ProductService, which is *deleteProduct* and do not care about the logic business behind. Also, there are two results, which are success and fail with the corresponding messages. Both results still direct users back to the view “product-list” by using “redirect:/products”.

**Task 4.2: Create Product List View**

Table displays all products

A screenshot of a computer

AI-generated content may be incorrect.

By browsing to the “/products” endpoint, we can see the table has displayed all products.

Messages displayed correctly

A screenshot of a computer

AI-generated content may be incorrect.

I has updated the first product from “iPhone 15 Pro” to “iPhone 16 Pro” and it send me a message successfully.

**Task 4.3: Create Product Form View**

Form binding works

A computer screen shot of text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

I changed first product from “iPhone 16 Pro” to “iPhone 17 Pro Max” and as you can see in the browser, the name of the first product has been changed by binding “field=”\*{name}””.

All fields present

A screenshot of a computer

AI-generated content may be incorrect.

Save/ Cancel buttons work

A screenshot of a computer

AI-generated content may be incorrect.

Add new product then click “Save Product” button

A screenshot of a computer

AI-generated content may be incorrect.

Product has been updated successfully.

**Checkpoint #4:**

**A diagram of a software system

AI-generated content may be incorrect.**

**Workflow:**

**Scenario 1:** User brows to URL endpoint “/products” (View Products)

1. Browser → GET /products

2. ProductController receives request (@GetMapping)

3. Controller calls productService.getAllProducts()

4. Service calls productRepository.findAll()

5. JPA automatically queries database

6. Repository returns List<Product>

7. Controller adds data to Model

8. Returns view name "product-list"

9. Thymeleaf renders HTML

10. Response sent to browser

**Scenario 2:** User clicks “Add new product”

1. Browser → GET /new

2. ProductController receives request (@GetMapping)

3. Create a Product object

4. Returns view name “product-form”

5. Thymeleaf renders HTML

6. Response sent to browser

**Scenario 3:** User adds new product

1. Browser → GET /products/edit/{id}

2. ProductController receives request (@GetMapping)

3. Controller calls productService.getProductById()

4. Service calls productRepository.findById()

5. JPA automatically queries database

6. Repository returns Product

7. Controller adds data to Model

8. Returns view name "product-form" if success

9. Return view name “product-list” with error message if fail

10. Thymeleaf renders HTML

11. Response sent to browser

**Scenario 4:** User save product

1. Browser → POST /save

2. ProductController receives request (@PostMapping)

3. Controller calls productService.saveProducts()

4. Service calls productRepository.save()

5. JPA automatically queries database

6. Repository returns Product

7. Controller adds data to Model

8. Returns view name "product-list"

9. Thymeleaf renders HTML

10. Response sent to browser

**Scenario 5:** User deletes product

1. Browser → GET /products/delete/{id}

2. ProductController receives request (@GetMapping)

3. Controller calls productService.deleteProduct()

4. Service calls productRepository.deleteById()

5. JPA automatically queries database

6. Returns view name "product-list"

7. Thymeleaf renders HTML

8. Response sent to browser