Spring boot

1. Annotation:

- Giúp cấu hình và điều khiển hành vi của ứng dụng
- **`@SpringBootApplication`**

- **Ý nghĩa**: Đây là annotation chính để đánh dấu lớp chính của ứng dụng Spring Boot. Nó kết hợp ba annotation quan trọng: `@Configuration`, `@EnableAutoConfiguration`, và `@ComponentScan`.
- **Sử dụng**: Được sử dụng trên lớp chính của ứng dụng, nơi có phương thức `main` để khởi chạy ứng dụng Spring Boot.

- **`@RestController`**

- **Ý nghĩa**: Được sử dụng để đánh dấu lớp là một controller trong Spring MVC và cho phép trả về các dữ liệu (thay vì view) từ các phương thức của controller.
- **Sử dụng**: Thường được kết hợp với các phương thức xử lý HTTP requests

  - **Annotation Trong Spring Data JPA**

  - **@Entity**: Đánh dấu một lớp là thực thể của cơ sở dữ liệu, tương ứng với một bảng.
  - **@Table**: Xác định tên bảng trong cơ sở dữ liệu mà thực thể sẽ ánh xạ đến.
  - **@Id**: Đánh dấu trường là khóa chính của thực thể.
  - **@GeneratedValue**: Chỉ định chiến lược tự động sinh giá trị cho khóa chính.

- **@Column**: Chỉ định các thuộc tính của cột trong bảng.

- **Annotation Trong Spring Security**
  - **@EnableWebSecurity**: Kích hoạt bảo mật cho ứng dụng web.
  - **@Secured**: Đảm bảo rằng chỉ các người dùng với vai trò được chỉ định mới có thể truy cập phương thức.
  - **@PreAuthorize**: Phân quyền truy cập vào các phương thức dựa trên biểu thức SpEL.

- **Annotation Trong Spring MVC**

- **@RestController**: Đánh dấu lớp là một controller trong ứng dụng RESTful. Tương đương với `@Controller` và `@ResponseBody`.
- **@RequestMapping**: Chỉ định các URL mà phương thức controller sẽ xử lý. Có thể sử dụng các biến thể như `@GetMapping`, `@PostMapping`, `@PutMapping`, và `@DeleteMapping` để chỉ định phương thức HTTP cụ thể.
- **@PathVariable**: Lấy giá trị từ phần của URL được chỉ định.
- **@RequestParam**: Lấy giá trị từ tham số của query string trong URL.
- **@RequestBody**: Chuyển đổi dữ liệu từ request body thành đối tượng Java.
- **@ResponseBody**: Chỉ định rằng giá trị trả về của phương thức sẽ được viết vào response body thay vì được render vào một view.

Các annotation bổ sung

1. `RequiredArgsConstructor`

+ là một annotation của Lombok được sử dụng để tự động tạo một constructor cho các trường dữ liệu (`fields`) của lớp mà không được khởi tạo trong lớp và được đánh dấu là `final` hoặc `@NonNull`.
+ Khi bạn sử dụng `@RequiredArgsConstructor`, Lombok sẽ tự động tạo một constructor mà tất cả các trường `final` (hoặc được đánh dấu `@NonNull`) đều được khởi tạo thông qua constructor đó.
+ Điều này giúp giảm bớt việc viết mã boilerplate cho các constructor.
+ Sử dụng từ khóa `final` cho các biến phụ thuộc giúp đảm bảo rằng các phụ thuộc không thể bị thay đổi sau khi được khởi tạo, tăng cường tính an toàn và ổn định của ứng dụng.

```
@RequiredArgsConstructor

public class CategoryController {

   private final CategoryService categoryService;
```

2. `@PrePersist` là một annotation JPA được sử dụng để thực hiện các hành động trước khi thực thể được lưu vào cơ sở dữ liệu.

+ `@PrePersist` đảm bảo rằng phương thức `onCreate()` sẽ được gọi ngay trước khi thực thể được chèn vào cơ sở dữ liệu. Đây là một phần trong vòng đời của thực thể JPA, và nó cho phép bạn thực hiện các hành động cần thiết trước khi dữ liệu được lưu.

3. Annotation `@MappedSuperclass` trong JPA (Java Persistence API)
    + Được sử dụng để đánh dấu một lớp mà không phải là một thực thể JPA (entity) nhưng chứa các thuộc tính và phương thức chung mà bạn muốn các thực thể khác kế thừa.
    + Đây là một phần trong chiến lược kế thừa của JPA và giúp chia sẻ các thuộc tính giữa các thực thể mà không cần phải tạo ra các bảng riêng cho lớp cha.

Kiến trúc mô hình spring boot

1. Kiến trúc 3 layer, hay còn gọi là mô hình 3 lớp, là một kiến trúc phát triển phần mềm phổ biến trong lập trình ứng dụng. Kiến trúc này bao gồm 3 tầng (*layer*) logic được phân chia rõ ràng và độc lập với nhau:
    - **Presentation layer (*tầng trình diễn*)**: Là tầng giao diện người dùng, được sử dụng để hiển thị dữ liệu và tương tác với người dùng. Tầng này sử dụng các công nghệ như HTML, CSS, JavaScript, JSP, thư viện front-end như Angular hoặc React, và cung cấp các chức năng như xác thực người dùng, kiểm tra đầu vào và hiển thị dữ liệu.
    - **Business Logic layer (*tầng xử lý logic*)**: Là tầng xử lý logic của ứng dụng, bao gồm các quy trình kinh doanh và tính năng của ứng dụng. Tầng này thực hiện các tác vụ xử lý dữ liệu, xử lý nghiệp vụ, xử lý lỗi, và tạo ra các kết quả phù hợp. Tầng này sử dụng các công nghệ Java, như Spring Boot, để triển khai logic của ứng dụng.
    - **Data Access layer (*tầng truy cập dữ liệu*)**: Là tầng truy cập cơ sở dữ liệu, được sử dụng để lưu trữ và truy xuất dữ liệu. Tầng

này sử dụng các công nghệ như JDBC, Hibernate, Spring Data JPA để tương tác với cơ sở dữ liệu.

2. Luồng đi

- **Nhận yêu cầu HTTP** từ client.
- **Controller** xử lý yêu cầu và có thể gọi các services
- **Service Layer** xử lý logic nghiệp vụ và tương tác với repository nếu cần.
- **Repository Layer** thực hiện thao tác với cơ sở dữ liệu.
- **Trả về phản hồi** cho client.

3. DTO

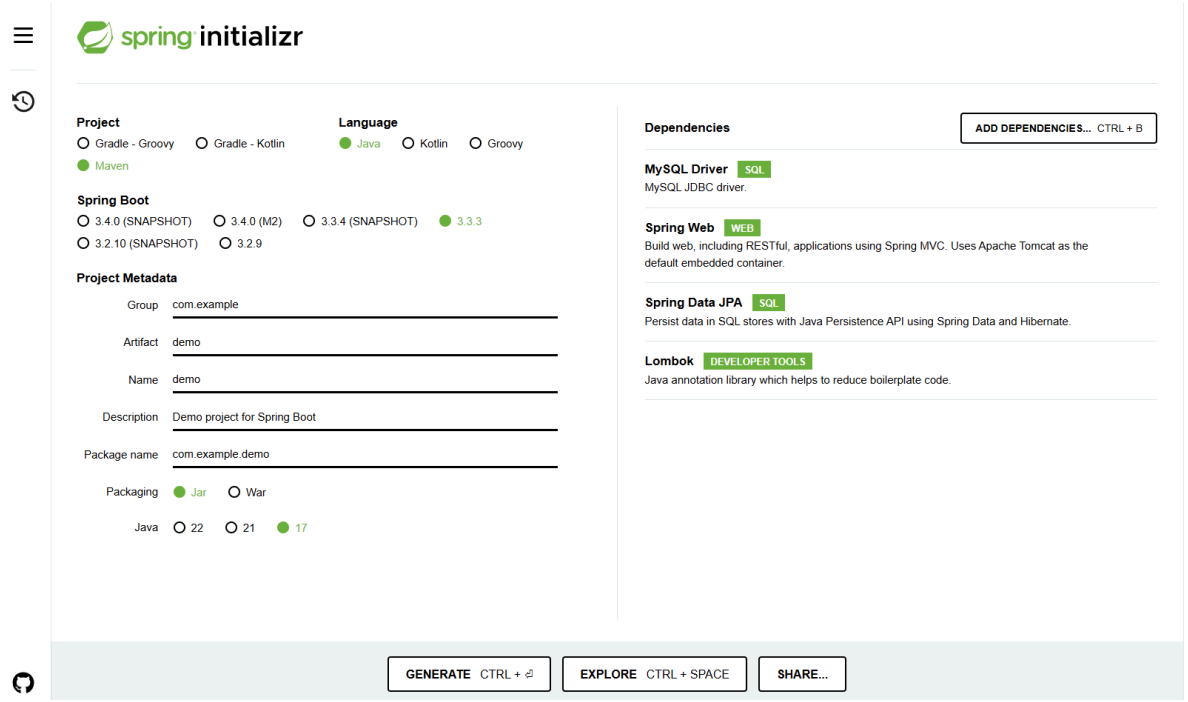
- DTO (Data Transfer Object) là một mẫu thiết kế được sử dụng để truyền dữ liệu giữa các lớp hoặc qua các lớp của ứng dụng, đặc biệt là giữa tầng controller và tầng service hoặc tầng dữ liệu. DTO thường được sử dụng để:

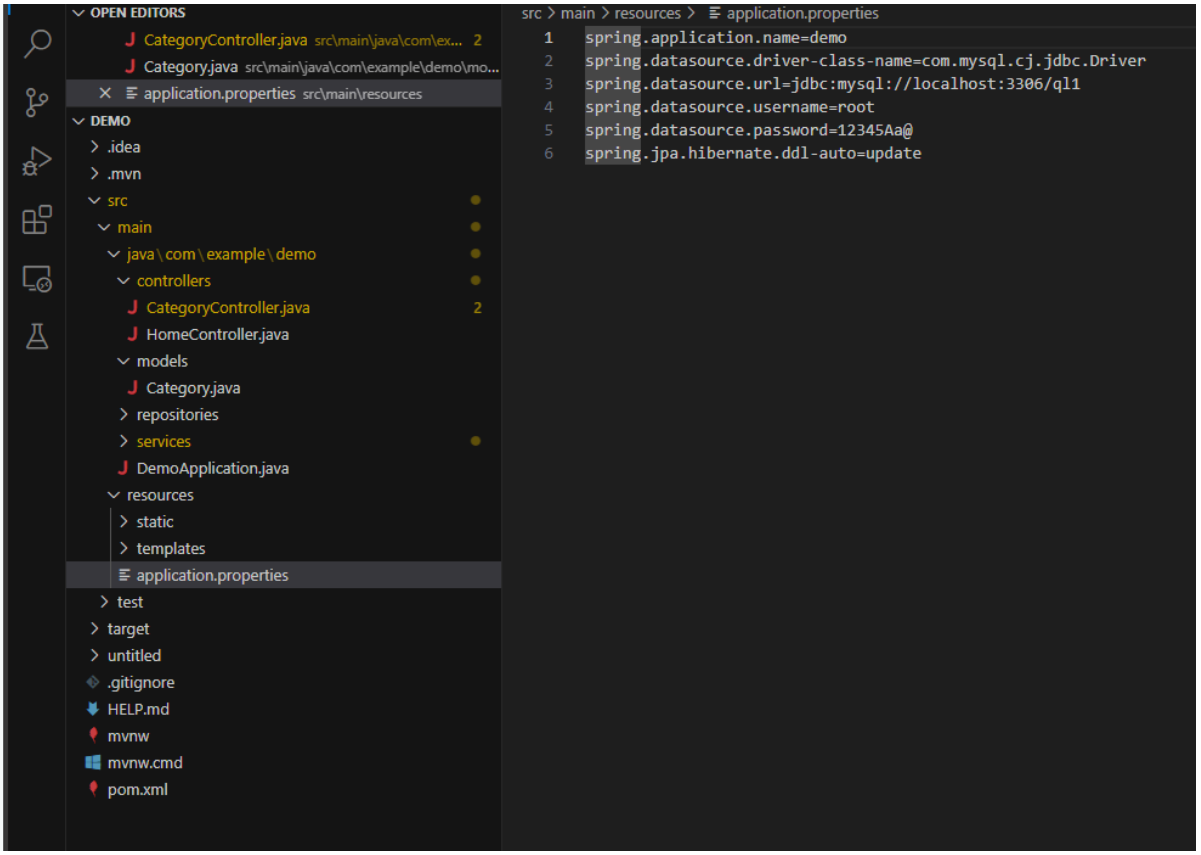
1. **Tách Biệt Dữ Liệu:** Tách biệt dữ liệu mà bạn gửi qua mạng hoặc từ client, với cách dữ liệu được lưu trữ trong cơ sở dữ liệu. Điều này cho phép bạn tạo ra các lớp dữ liệu tùy chỉnh phù hợp với nhu cầu của giao diện người dùng hoặc API mà không cần phải phụ thuộc vào cấu trúc của entity.
2. **DTO (Data Transfer Object) là một mẫu thiết kế dùng để chuyển dữ liệu giữa các lớp hoặc các tầng trong ứng dụng. DTO thường được sử dụng để giao tiếp giữa lớp dịch vụ và lớp điều khiển (controller), hoặc giữa lớp điều khiển và lớp dữ liệu.**
3. **giúp làm sạch và phân tách các lớp trong ứng dụng của bạn, giữ cho mã của bạn dễ bảo trì và mở rộng hơn.**

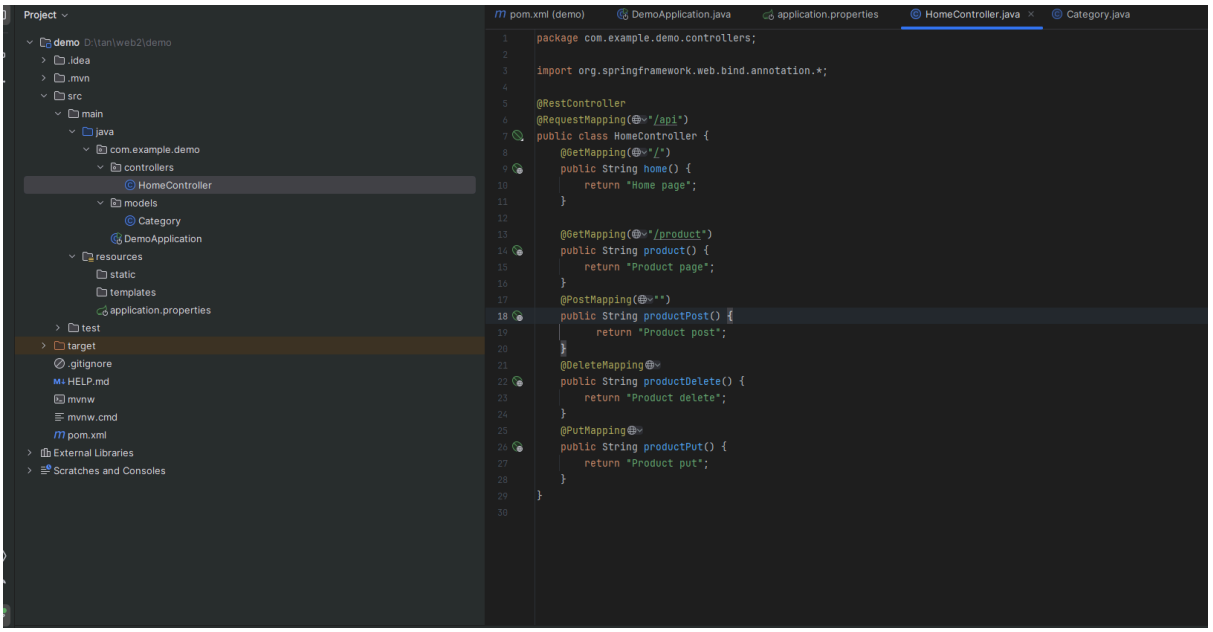
Phần cuối: ResponseEntity

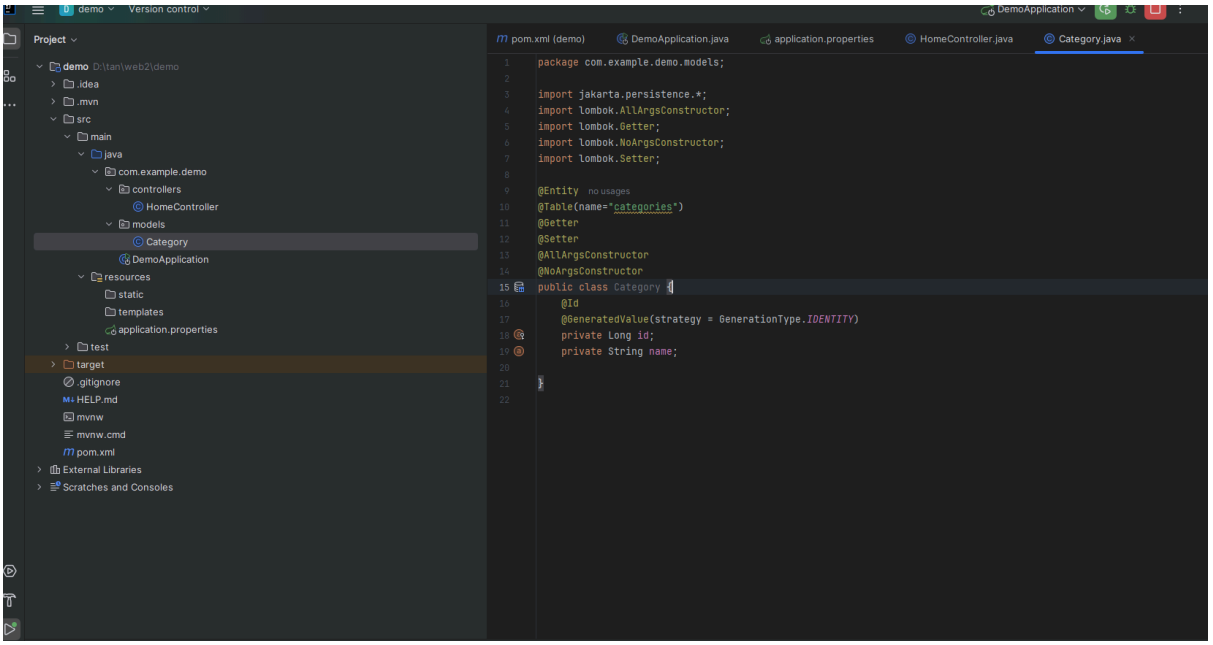
- `ResponseEntity` là một lớp trong Spring Framework được sử dụng để xây dựng phản hồi HTTP trong các ứng dụng Spring Boot. Nó cho phép bạn kiểm soát chi tiết hơn về cách phản hồi từ máy chủ được gửi đến client. `ResponseEntity` là một phần quan trọng trong việc xử lý các yêu cầu HTTP và có thể được sử dụng để:
- **Thiết lập mã trạng thái HTTP**: Bạn có thể chỉ định mã trạng thái HTTP cụ thể cho phản hồi, như 200 OK, 404 Not Found, 500 Internal Server Error, v.v.

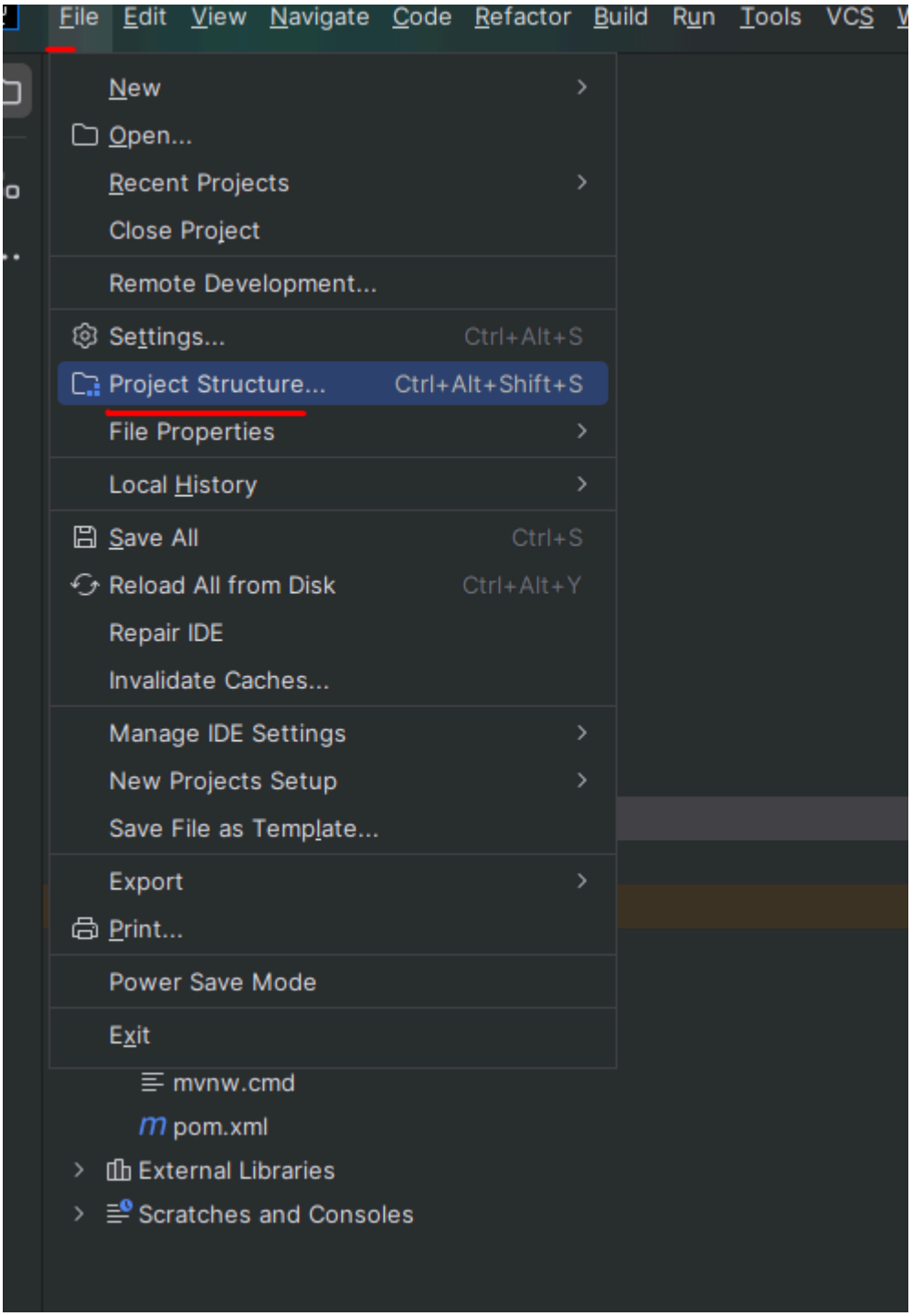
- **Thêm các tiêu đề HTTP**: Bạn có thể thêm hoặc sửa đổi các tiêu đề HTTP trong phản hồi, như `Content-Type`, `Authorization`, `Cache-Control`, v.v.
- **Chứa dữ liệu phản hồi**: Bạn có thể đóng gói dữ liệu mà bạn muốn gửi đến client trong phản hồi HTTP.
    - **Cấu trúc của `ResponseEntity`**
- `ResponseEntity` có thể chứa ba thành phần chính:
- **Mã trạng thái HTTP**: Ví dụ: `200 OK`, `404 Not Found`, `500 Internal Server Error`.
- **Tiêu đề HTTP**: Ví dụ: `Content-Type`, `Location`, `Cache-Control`.
- **Thân phản hồi (Body)**: Dữ liệu thực tế mà bạn muốn gửi đến client.
- **Các phương thức chính của `ResponseEntity`**
    - **`ResponseEntity.ok()`**: Trả về mã trạng thái 200 OK với thân phản hồi.
    - **`ResponseEntity.status(HttpStatus status)`**: Chỉ định mã trạng thái HTTP cụ thể cho phản hồi.
    - **`ResponseEntity.header(String headerName, String headerValue)`**: Thêm hoặc thay đổi tiêu đề HTTP.
    - **`ResponseEntity.body(T body)`**: Chỉ định dữ liệu thân phản hồi.

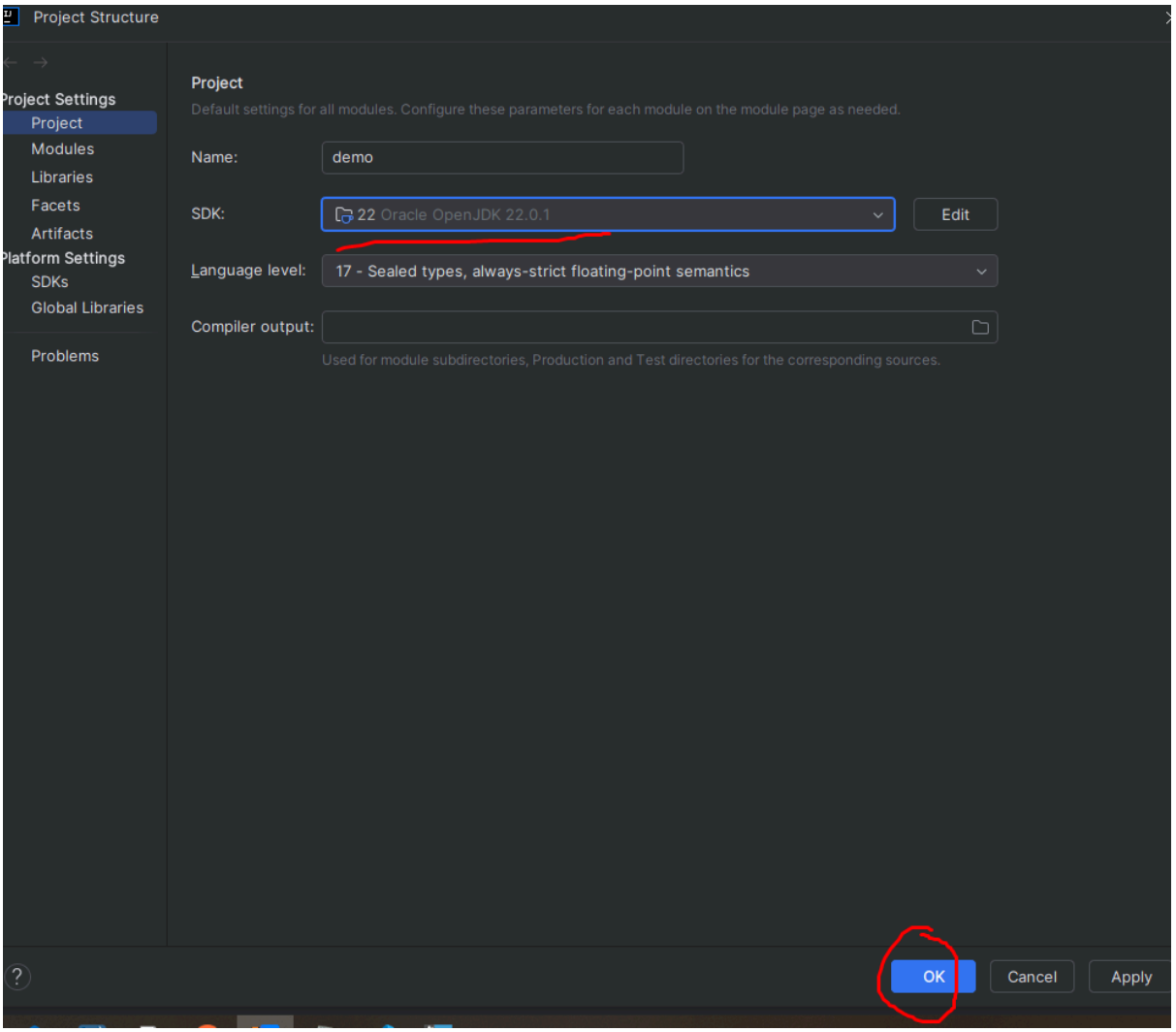

spring initializr
Project
Gradle - Groovy
Gradle - Kotlin
Maven
Language
Java
Kotlin
Groovy
Spring Boot
3.4.0 (SNAPSHOT)
3.4.0 (M2)
3.3.4 (SNAPSHOT)
3.3.3
3.2.10 (SNAPSHOT)
3.2.9
Project Metadata
Group com.example
Artifact demo
Name demo
Description Demo project for Spring Boot
Package name com.example.demo
Packaging Jar War
Java 22 21 17
Dependencies
ADD DEPENDENCIES... CTRL + B
MySQL Driver SQL
MySQL JDBC driver.
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.
GENERATE CTRL + ⏎
EXPLORE CTRL + SPACE
SHARE...

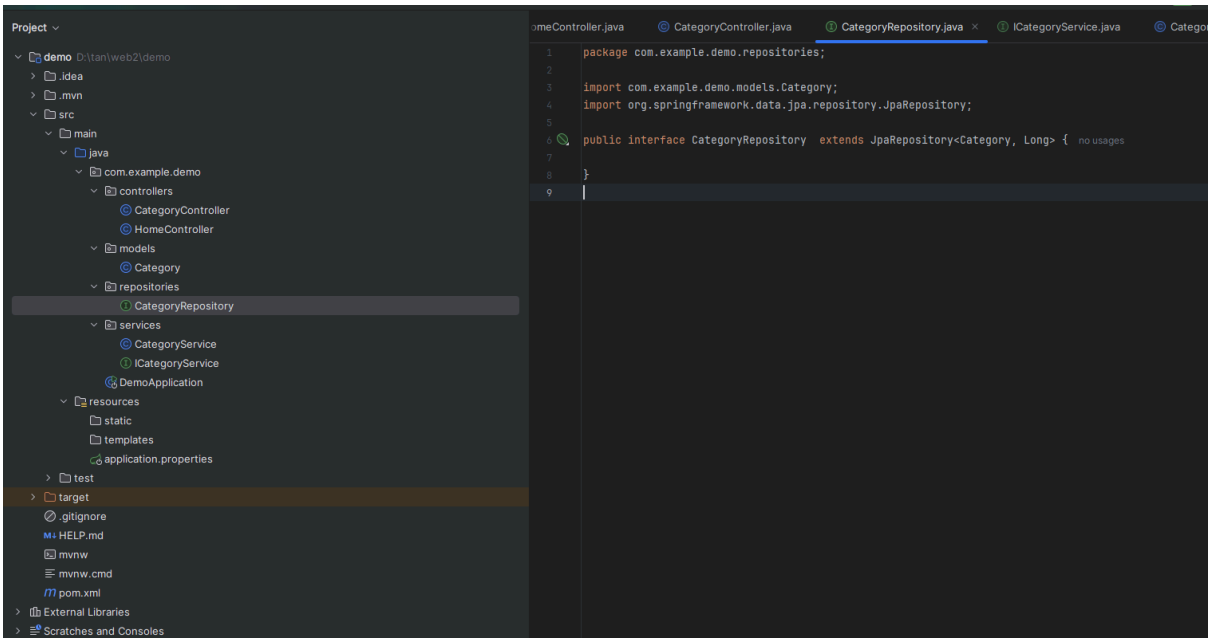

OPEN EDITORS
CategoryController.java src\main\java\com\ex... 2
Category.java src\main\java\com\example\demo\mo...
application.properties src\main\resources
DEMO
.idea
.mvn
src
main
java \ com \ example \ demo
controllers
CategoryController.java 2
HomeController.java
models
Category.java
repositories
services
DemoApplication.java
resources
static
templates
application.properties
test
target
untitled
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
src > main > resources > application.properties
1 spring.application.name=demo
2 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
3 spring.datasource.url=jdbc:mysql://localhost:3306/ql1
4 spring.datasource.username=root
5 spring.datasource.password=12345Aa@
6 spring.jpa.hibernate.ddl-auto=update

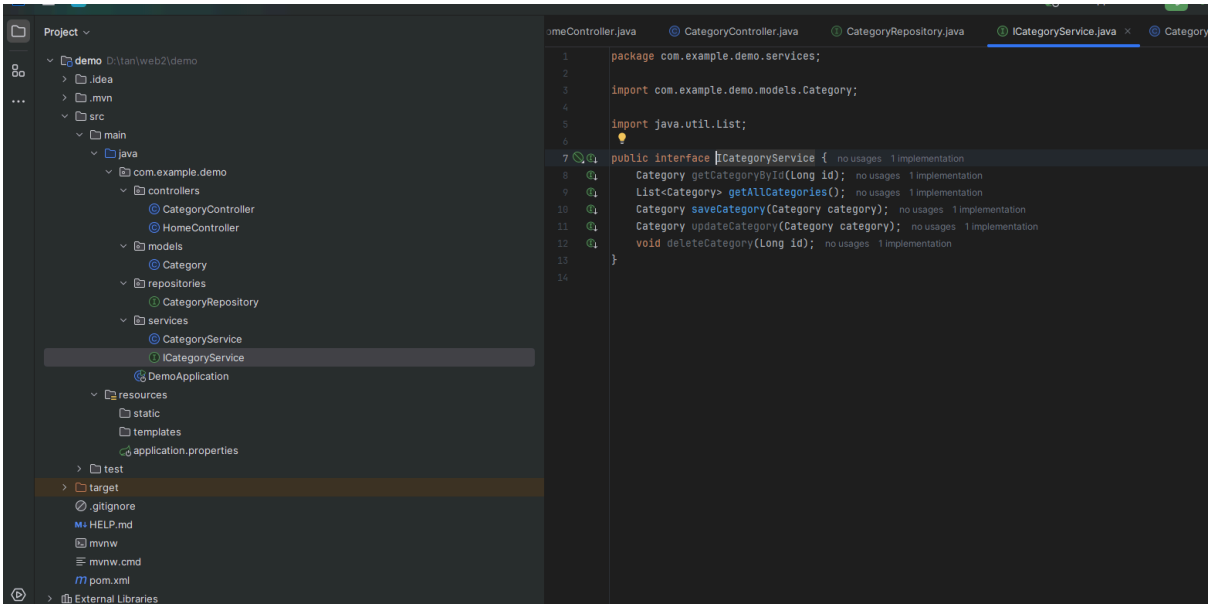

Project
demo D:\tan\web2\demo
.idea
.mvn
src
main
java
com.example.demo
controllers
HomeController
models
Category
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
pom.xml (demo)
DemoApplication.java
application.properties
HomeController.java
Category.java
package com.example.demo.controllers;
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api")
public class HomeController {
@GetMapping("/")
public String home() {
return "Home page";
}
@GetMapping("/product")
public String product() {
return "Product page";
}
@PostMapping("")
public String productPost() {
return "Product post";
}
@DeleteMapping
public String productDelete() {
return "Product delete";
}
@PutMapping
public String productPut() {
return "Product put";
}
}

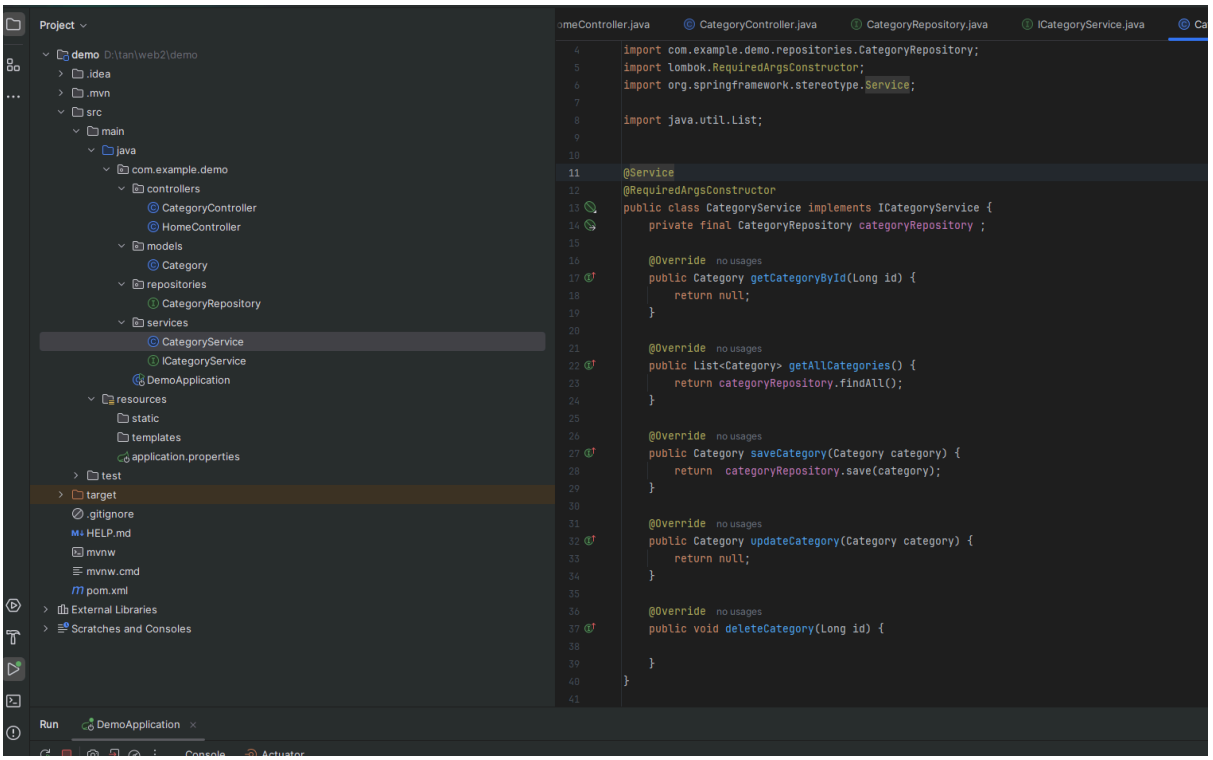

demo
Version control
DemoApplication
Project
demo D:\tan\web2\demo
.idea
.mvn
src
main
java
com.example.demo
controllers
HomeController
models
Category
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
pom.xml (demo)
DemoApplication.java
application.properties
HomeController.java
Category.java
package com.example.demo.models;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
@Entity no usages
@Table(name="categories")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Category {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String name;
}

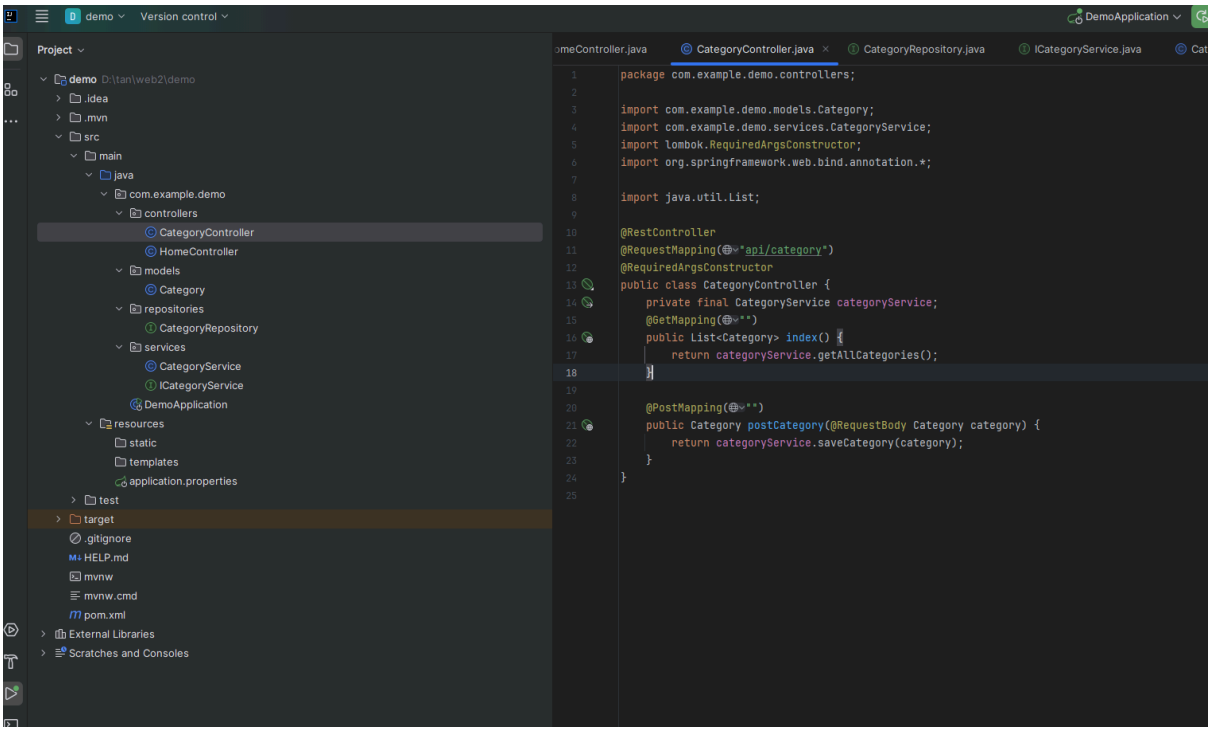

File Edit View Navigate Code Refactor Build Run Tools VCS
New
Open...
Recent Projects
Close Project
Remote Development...
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
File Properties
Local History
Save All Ctrl+S
Reload All from Disk Ctrl+Alt+Y
Repair IDE
Invalidate Caches...
Manage IDE Settings
New Projects Setup
Save File as Template...
Export
Print...
Power Save Mode
Exit
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles

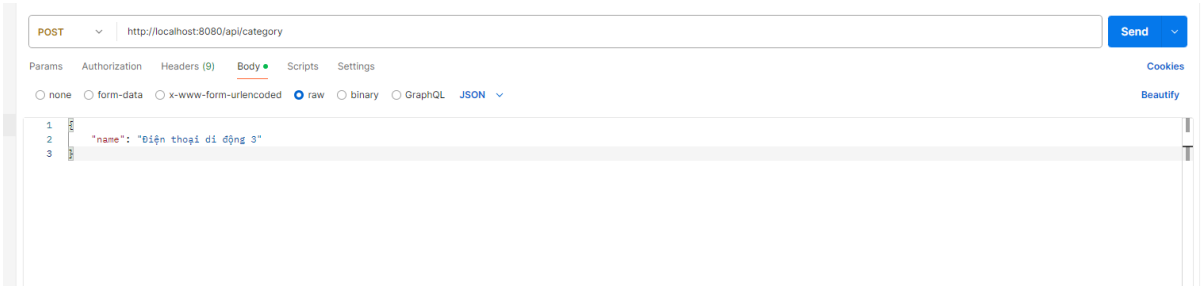

Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
Project
Default settings for all modules. Configure these parameters for each module on the module page as needed.
Name:
demo
SDK:
22 Oracle OpenJDK 22.0.1
Edit
Language level:
17 - Sealed types, always-strict floating-point semantics
Compiler output:
Used for module subdirectories, Production and Test directories for the corresponding sources.
OK
Cancel
Apply


Project
demo D:\tan\web2\demo
.idea
.mvn
src
main
java
com.example.demo
controllers
CategoryController
HomeController
models
Category
repositories
CategoryRepository
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
CategoryController.java
CategoryRepository.java
ICategoryService.java
package com.example.demo.repositories;
import com.example.demo.models.Category;
import org.springframework.data.jpa.repository.JpaRepository;
public interface CategoryRepository extends JpaRepository<Category, Long> { no usages
}

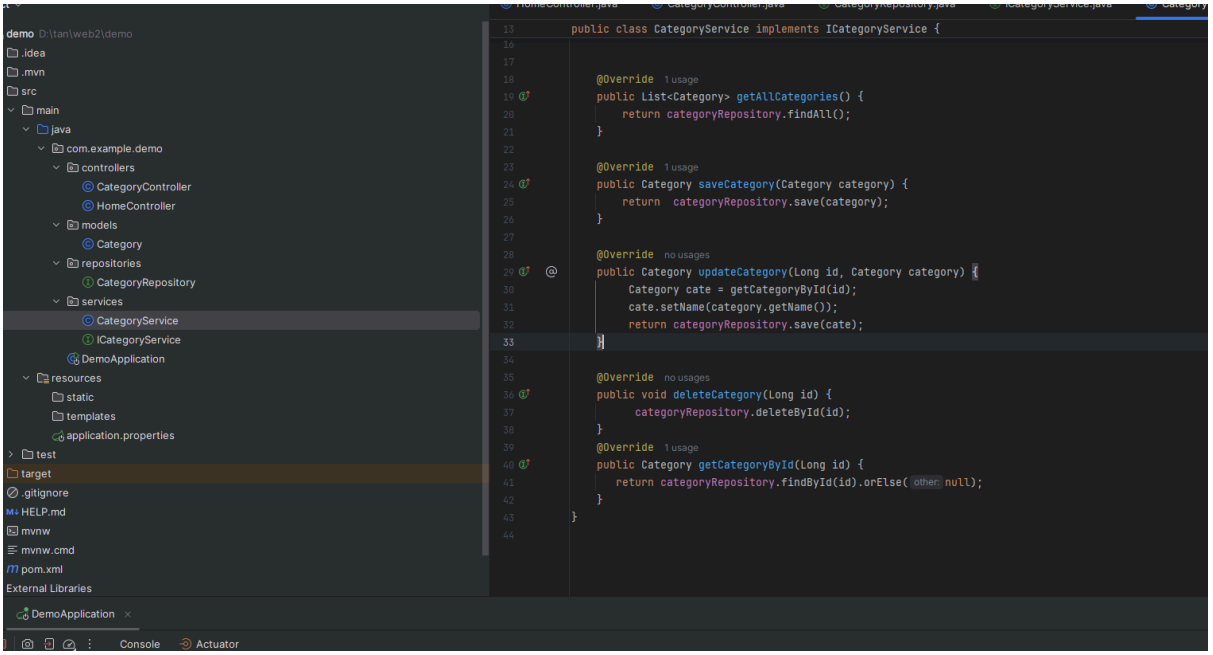

Project
demo D:\tan\web2\demo
.idea
.mvn
src
main
java
com.example.demo
controllers
CategoryController
HomeController
models
Category
repositories
CategoryRepository
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
CategoryController.java
CategoryRepository.java
ICategoryService.java
package com.example.demo.services;
import com.example.demo.models.Category;
import java.util.List;
public interface ICategoryService { no usages 1 implementation
Category getCategoryById(Long id); no usages 1 implementation
List<Category> getAllCategories(); no usages 1 implementation
Category saveCategory(Category category); no usages 1 implementation
Category updateCategory(Category category); no usages 1 implementation
void deleteCategory(Long id); no usages 1 implementation
}

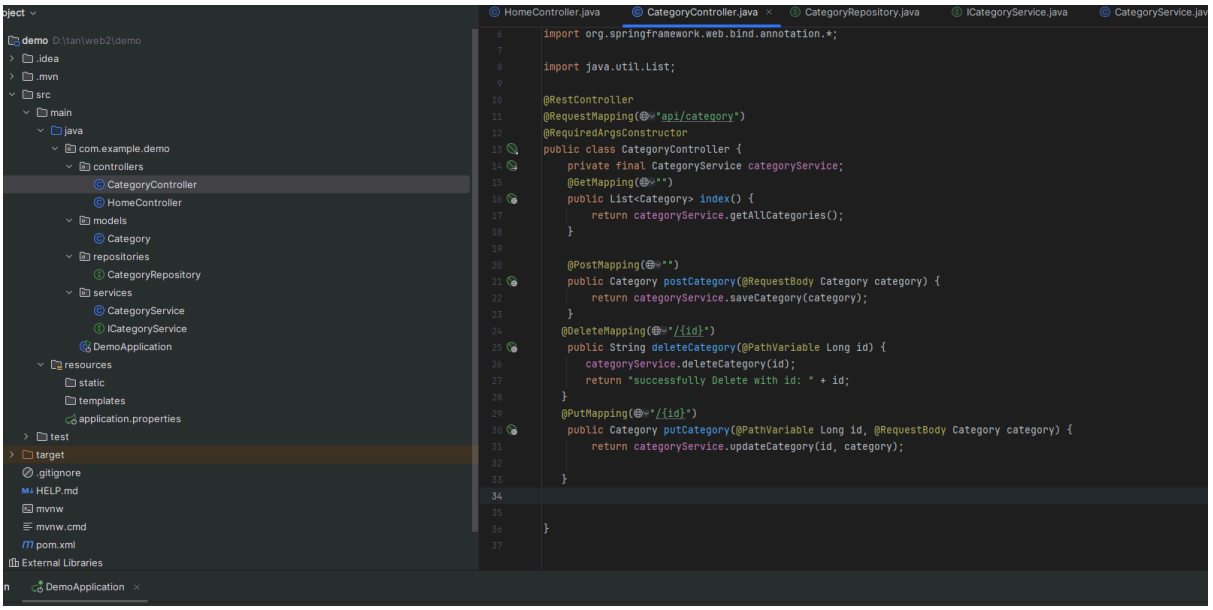

Project
demo D:\tan\web2\demo
.idea
.mvn
src
main
java
com.example.demo
controllers
CategoryController
HomeController
models
Category
repositories
CategoryRepository
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
Run
DemoApplication
CategoryController.java
CategoryRepository.java
ICategoryService.java
import com.example.demo.repositories.CategoryRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
@RequiredArgsConstructor
public class CategoryService implements ICategoryService {
private final CategoryRepository categoryRepository ;
@Override no usages
public Category getCategoryById(Long id) {
return null;
}
@Override no usages
public List<Category> getAllCategories() {
return categoryRepository.findAll();
}
@Override no usages
public Category saveCategory(Category category) {
return categoryRepository.save(category);
}
@Override no usages
public Category updateCategory(Category category) {
return null;
}
@Override no usages
public void deleteCategory(Long id) {
}
}

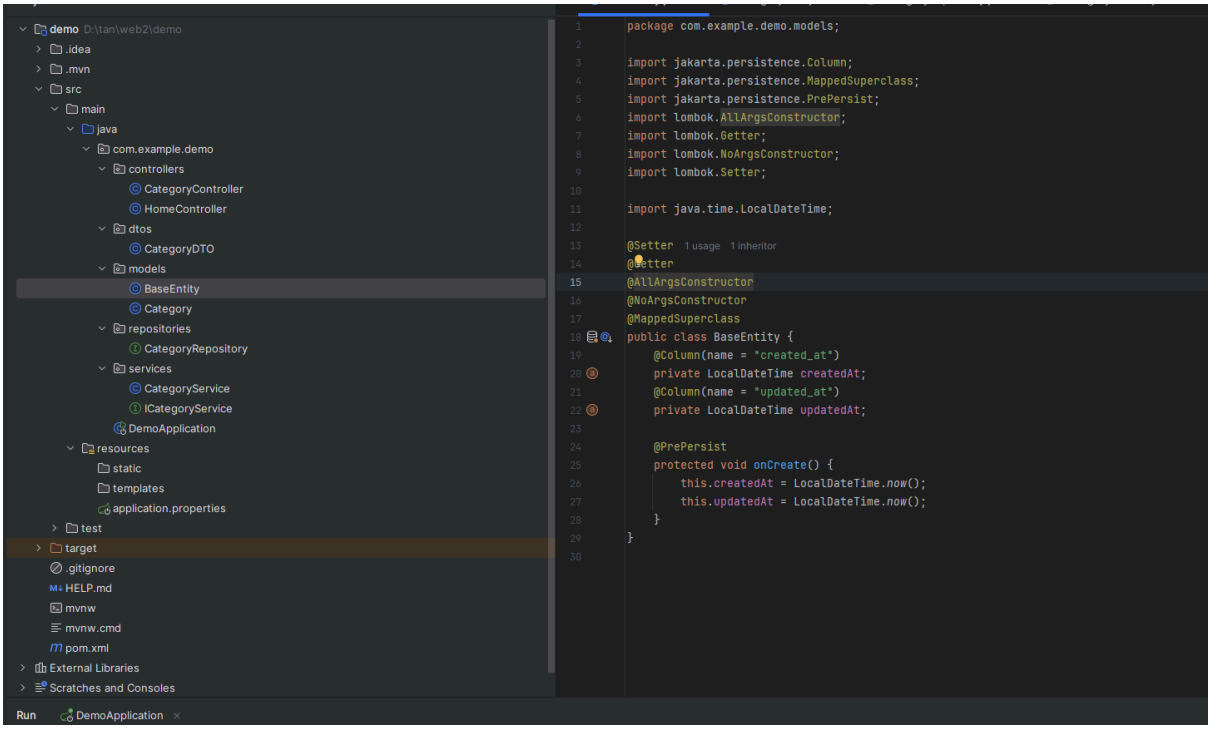

demo
Version control
DemoApplication
Project
demo D:\tan\web2\demo
.idea
.mvn
src
main
java
com.example.demo
controllers
CategoryController
HomeController
models
Category
repositories
CategoryRepository
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
CategoryController.java
CategoryRepository.java
ICategoryService.java
package com.example.demo.controllers;
import com.example.demo.models.Category;
import com.example.demo.services.CategoryService;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("api/category")
@RequiredArgsConstructor
public class CategoryController {
private final CategoryService categoryService;
@GetMapping("")
public List<Category> index() {
return categoryService.getAllCategories();
}
@PostMapping("")
public Category postCategory(@RequestBody Category category) {
return categoryService.saveCategory(category);
}
}

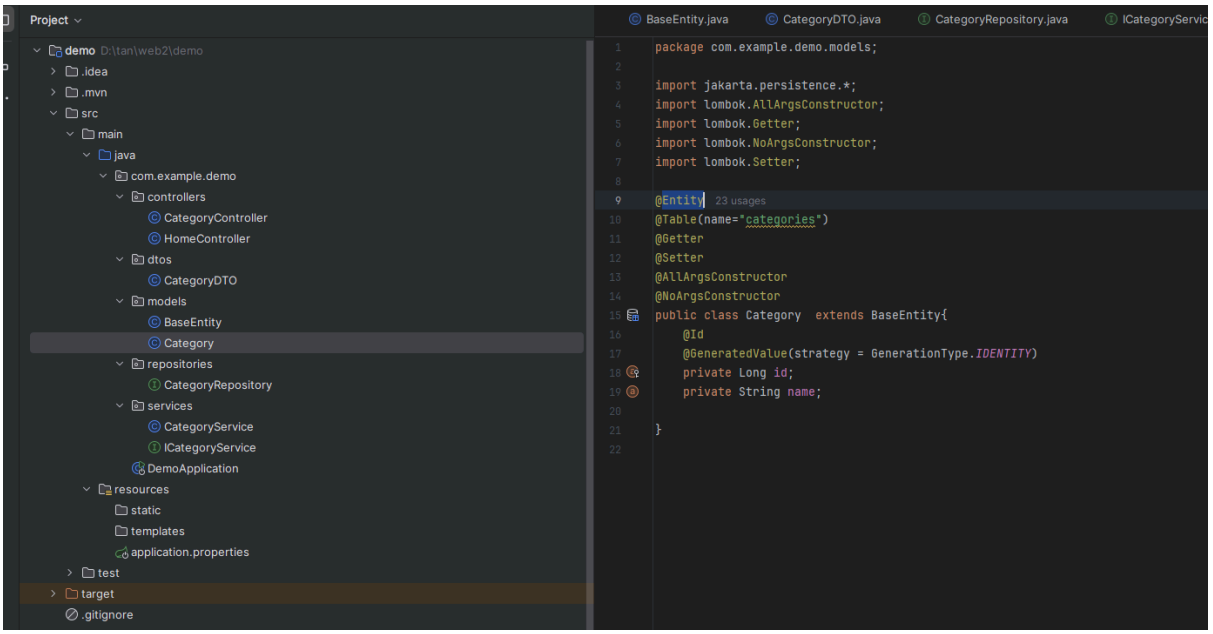


Chức năng delete, edit

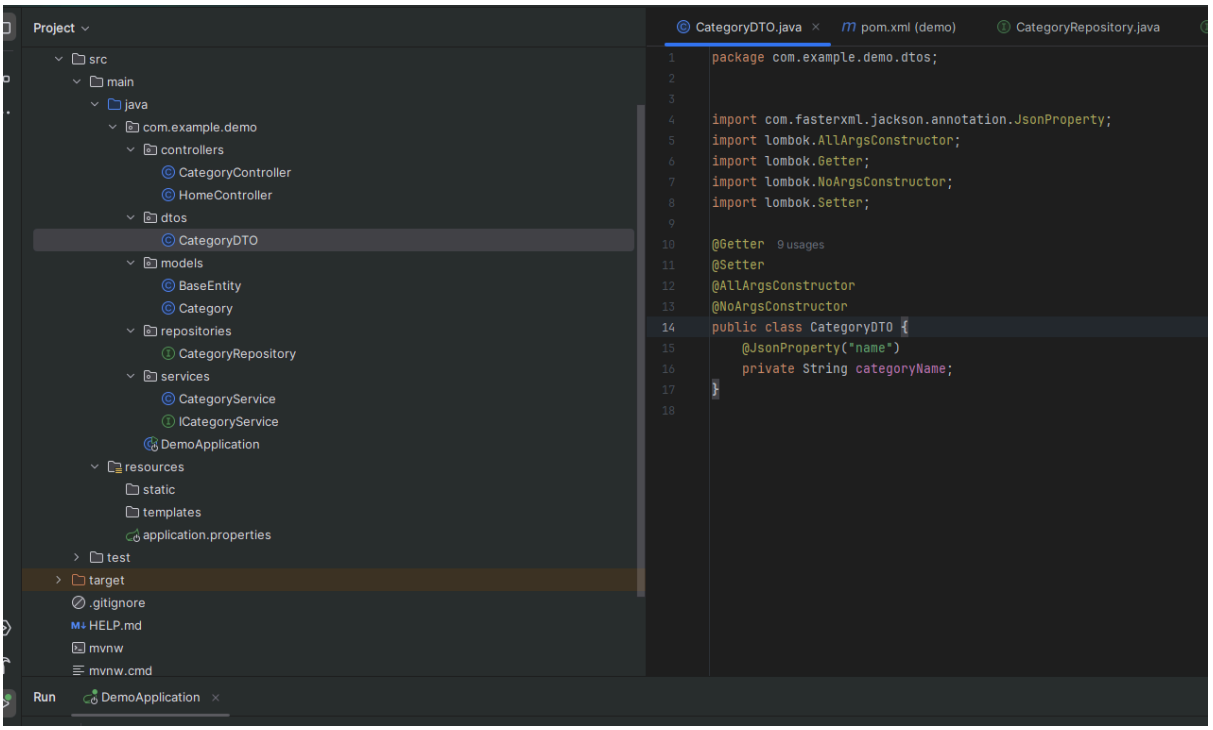


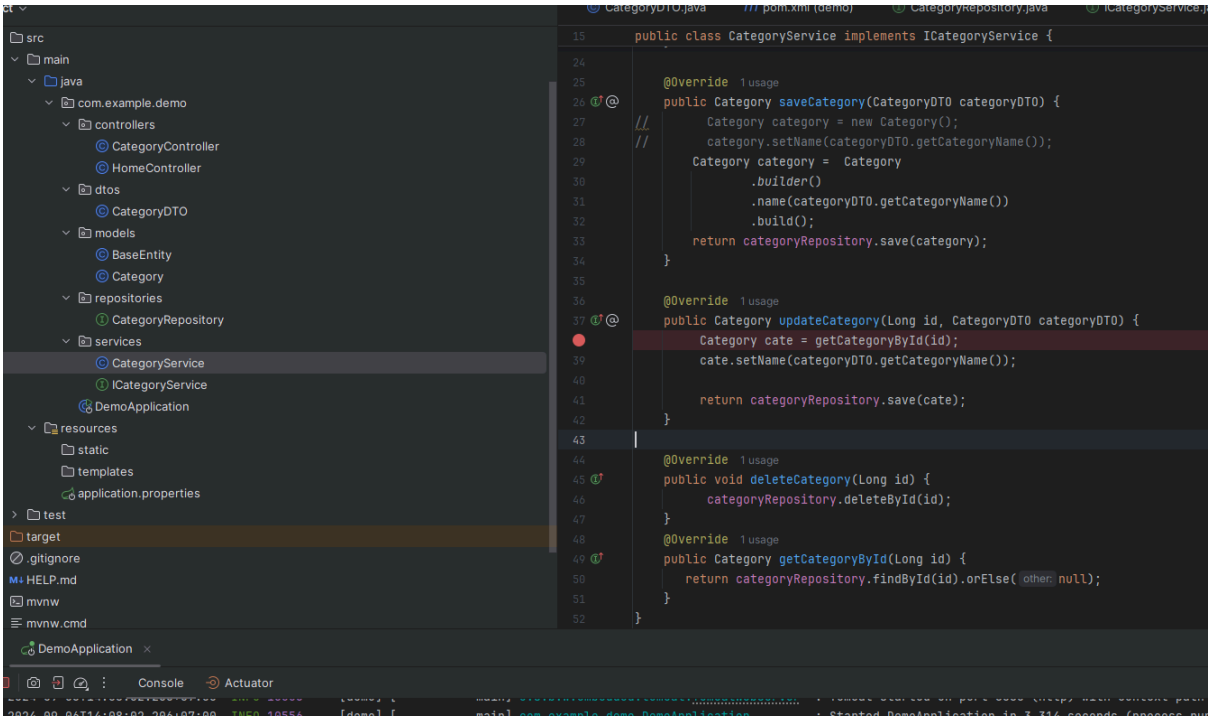

HomeController.java
CategoryController.java
CategoryRepository.java
ICategoryService.java
demo D:\tan\web2\demo
.idea
.mvn
src
main
java
com.example.demo
controllers
CategoryController
HomeController
models
Category
repositories
CategoryRepository
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
DemoApplication
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("api/category")
@RequiredArgsConstructor
public class CategoryController {
private final CategoryService categoryService;
@GetMapping("")
public List<Category> index() {
return categoryService.getAllCategories();
}
@PostMapping("")
public Category postCategory(@RequestBody Category category) {
return categoryService.saveCategory(category);
}
@DeleteMapping("/{id}")
public String deleteCategory(@PathVariable Long id) {
categoryService.deleteCategory(id);
return "successfully Delete with id: " + id;
}
@PutMapping("/{id}")
public Category putCategory(@PathVariable Long id, @RequestBody Category category) {
return categoryService.updateCategory(id, category);
}
}

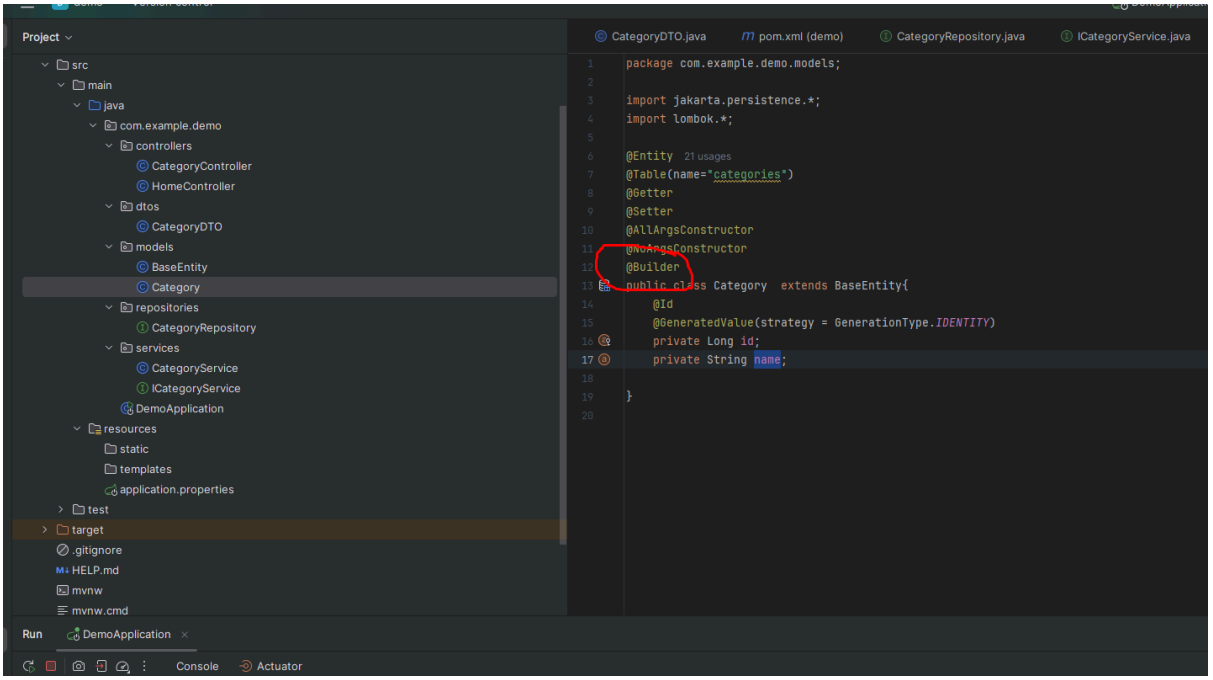

demo D:\tan\web2\demo
.idea
.mvn
src
main
java
com.example.demo
controllers
CategoryController
HomeController
dtos
CategoryDTO
models
BaseEntity
Category
repositories
CategoryRepository
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
Run
DemoApplication
package com.example.demo.models;
import jakarta.persistence.Column;
import jakarta.persistence.MappedSuperclass;
import jakarta.persistence.PrePersist;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import java.time.LocalDateTime;
@Setter 1 usage 1 inheritor
@Getter
@AllArgsConstructor
@NoArgsConstructor
@MappedSuperclass
public class BaseEntity {
@Column(name = "created_at")
private LocalDateTime createdAt;
@Column(name = "updated_at")
private LocalDateTime updatedAt;
@PrePersist
protected void onCreate() {
this.createdAt = LocalDateTime.now();
this.updatedAt = LocalDateTime.now();
}
}



Thực hành DTO

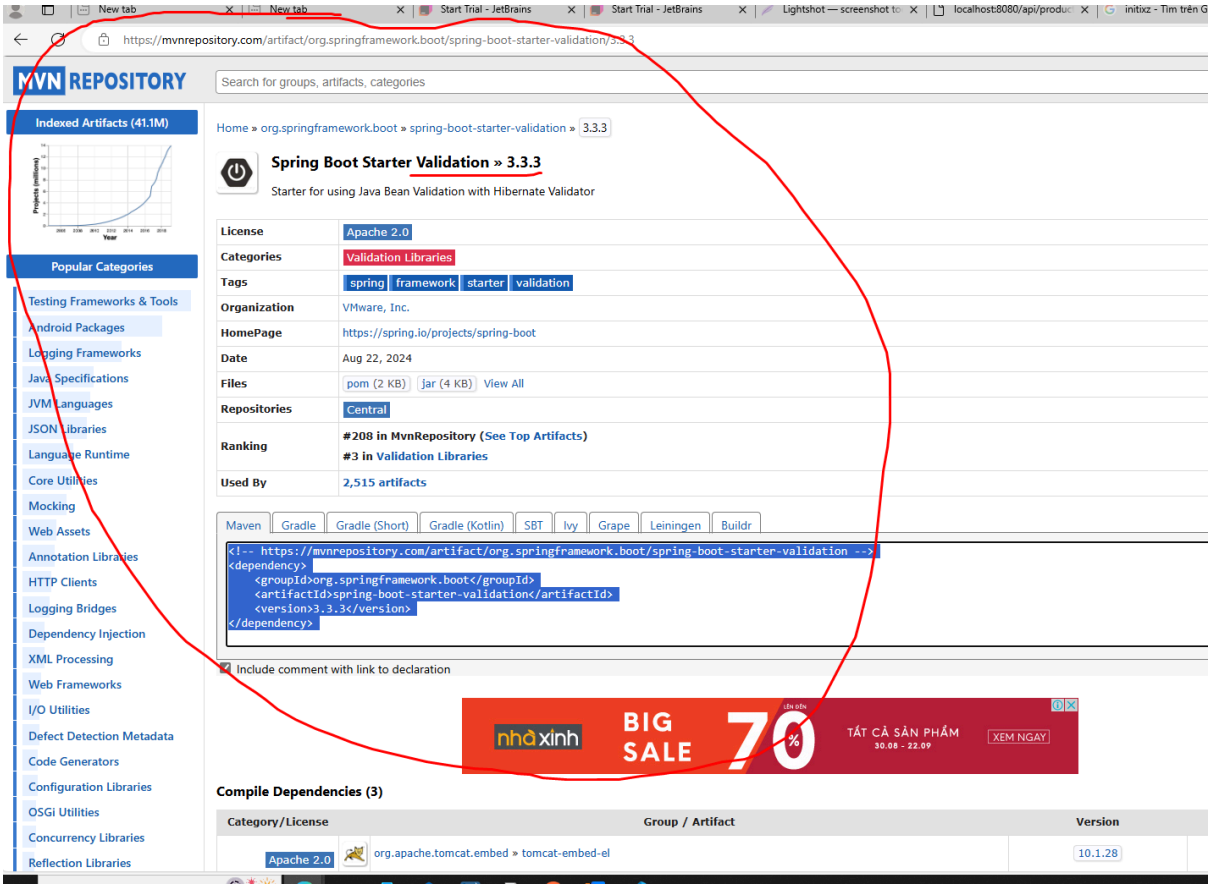


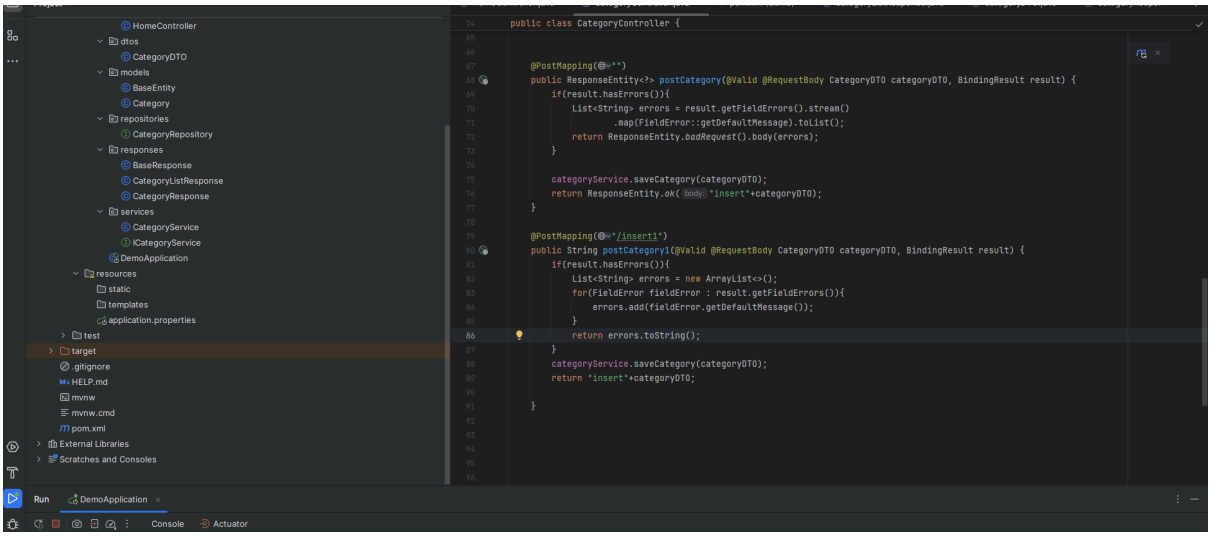

src
main
java
com.example.demo
controllers
CategoryController
HomeController
dtos
CategoryDTO
models
BaseEntity
Category
repositories
CategoryRepository
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
public class CategoryService implements ICategoryService {
@Override 1 usage
public Category saveCategory(CategoryDTO categoryDTO) {
// Category category = new Category();
// category.setName(categoryDTO.getCategoryName());
Category category = Category
.builder()
.name(categoryDTO.getCategoryName())
.build();
return categoryRepository.save(category);
}
@Override 1 usage
public Category updateCategory(Long id, CategoryDTO categoryDTO) {
Category cate = getCategoryById(id);
cate.setName(categoryDTO.getCategoryName());
return categoryRepository.save(cate);
}
@Override 1 usage
public void deleteCategory(Long id) {
categoryRepository.deleteById(id);
}
@Override 1 usage
public Category getCategoryById(Long id) {
return categoryRepository.findById(id).orElse(other: null);
}
}
DemoApplication
Console
Actuator

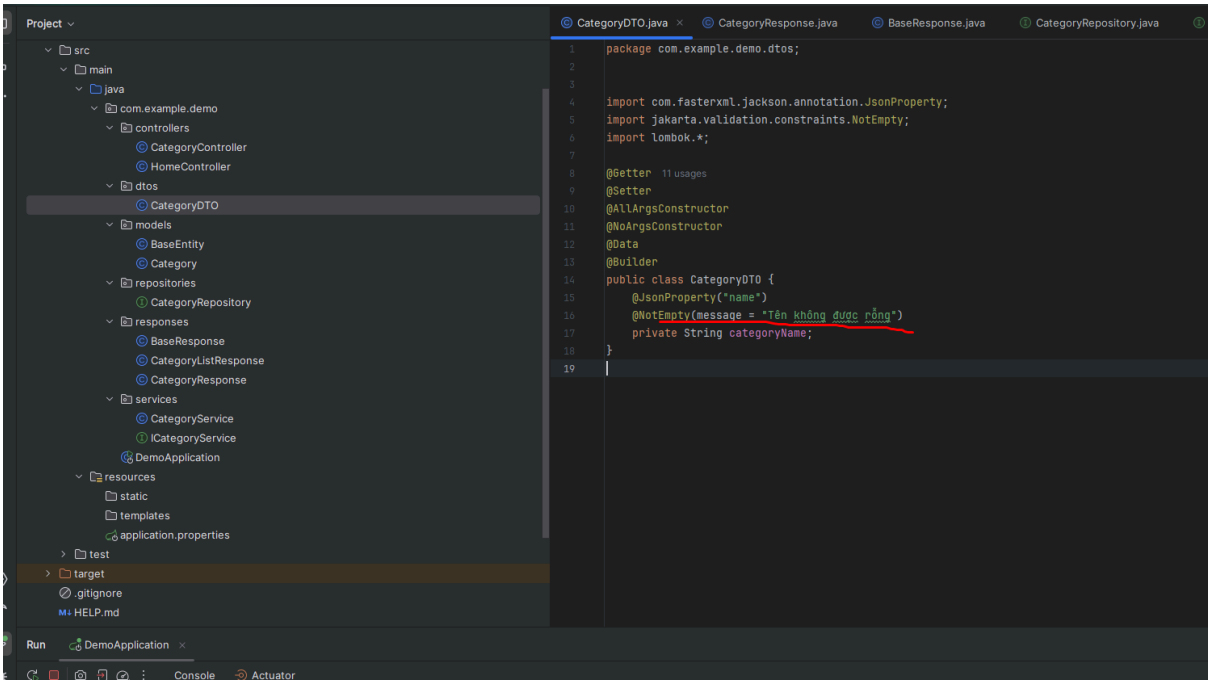

Project
src
main
java
com.example.demo
controllers
CategoryController
HomeController
dtos
CategoryDTO
models
BaseEntity
Category
repositories
CategoryRepository
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
CategoryDTO.java
pom.xml (demo)
CategoryRepository.java
ICategoryService.java
package com.example.demo.models;
import jakarta.persistence.*;
import lombok.*;
@Entity 21 usages
@Table(name="categories")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Category extends BaseEntity{
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String name;
}
Run
DemoApplication
Console
Actuator

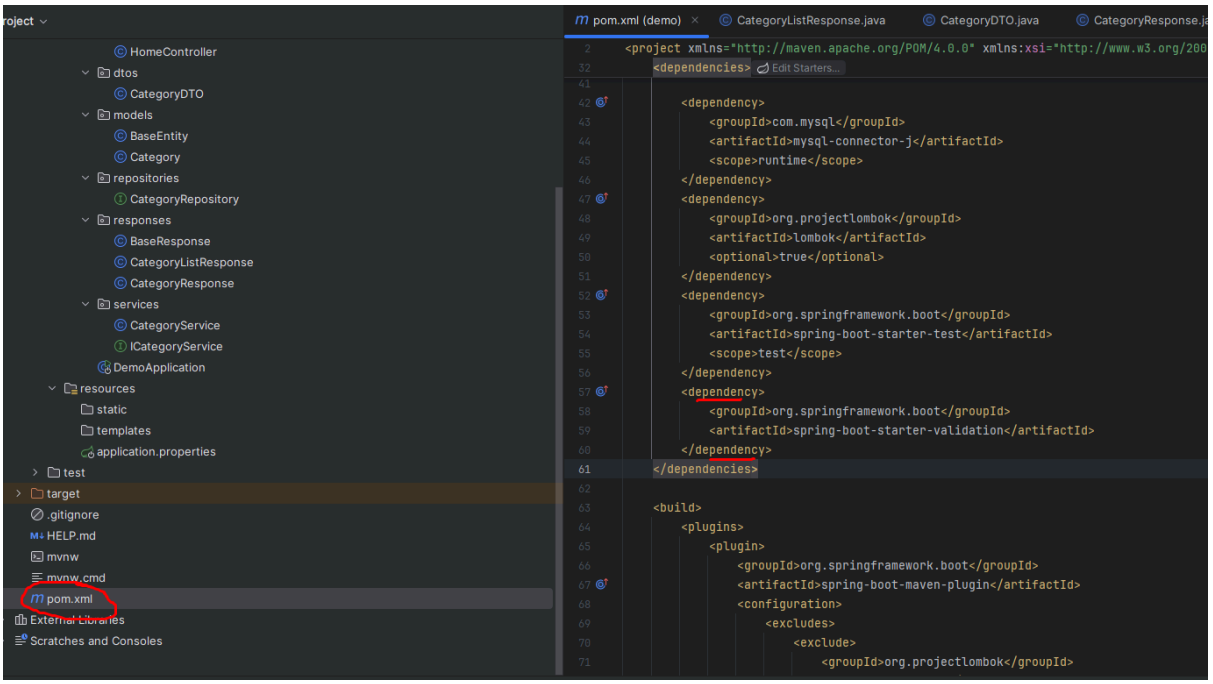
```
    }

    @PutMapping("/{id}")
     public Category putCategory(@PathVariable Long id, @RequestBody CategoryDTO categoryDTO) {
         return categoryService.updateCategory(id, categoryDTO);
    }


}
```

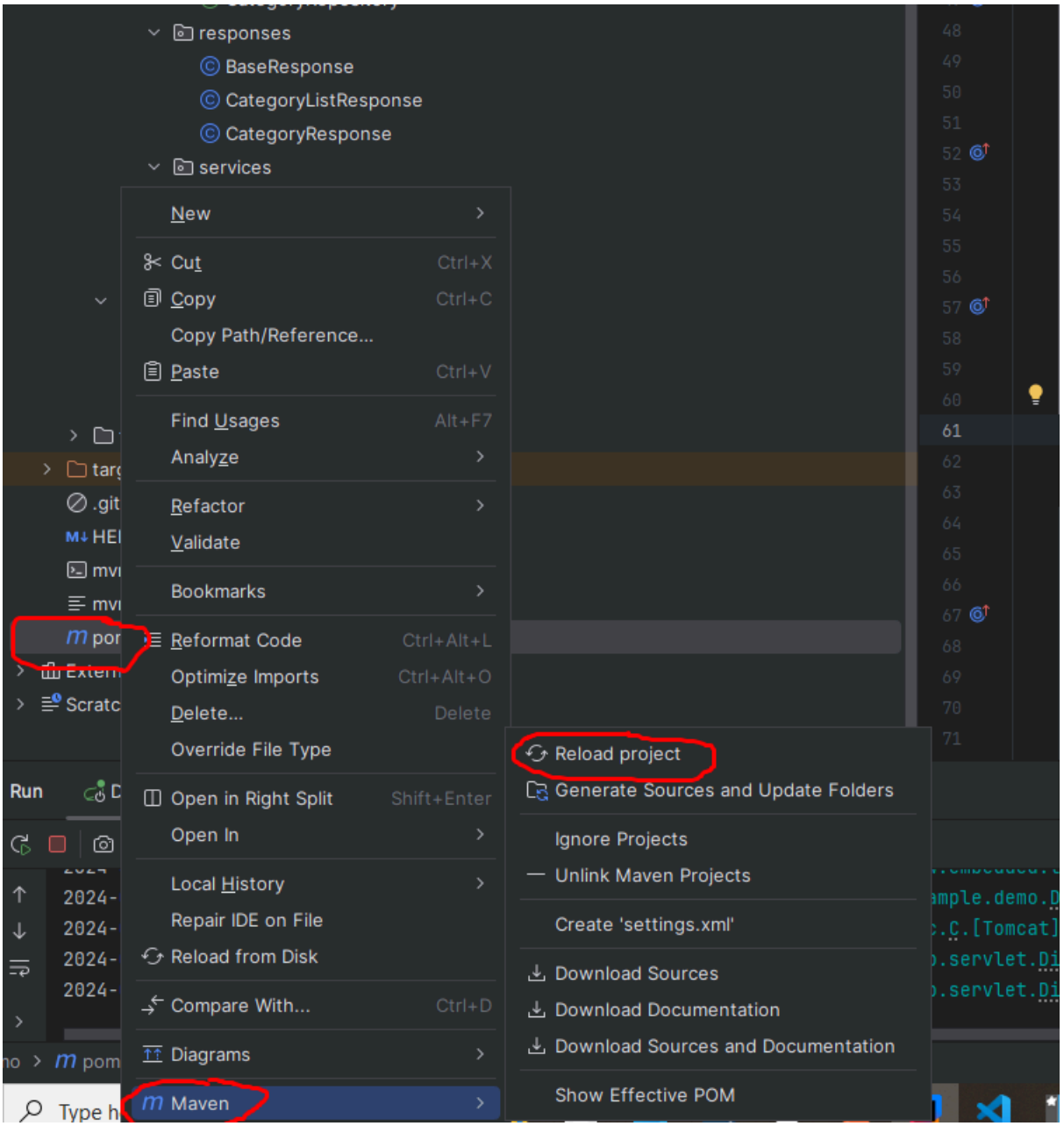
Validation




HomeController
dtos
CategoryDTO
models
BaseEntity
Category
repositories
CategoryRepository
responses
BaseResponse
CategoryListResponse
CategoryResponse
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
public class CategoryController {
@PostMapping("")
public ResponseEntity<?> postCategory(@Valid @RequestBody CategoryDTO categoryDTO, BindingResult result) {
if(result.hasErrors()){
List<String> errors = result.getFieldErrors().stream()
.map(FieldError::getDefaultMessage).toList();
return ResponseEntity.badRequest().body(errors);
}
categoryService.saveCategory(categoryDTO);
return ResponseEntity.ok(body: "insert"+categoryDTO);
}
@PostMapping("/insert1")
public String postCategory1(@Valid @RequestBody CategoryDTO categoryDTO, BindingResult result) {
if(result.hasErrors()){
List<String> errors = new ArrayList<>();
for(FieldError fieldError : result.getFieldErrors()){
errors.add(fieldError.getDefaultMessage());
}
return errors.toString();
}
categoryService.saveCategory(categoryDTO);
return "insert"+categoryDTO;
}
Run
DemoApplication
Console
Actuator


Project
src
main
java
com.example.demo
controllers
CategoryController
HomeController
dtos
CategoryDTO
models
BaseEntity
Category
repositories
CategoryRepository
responses
BaseResponse
CategoryListResponse
CategoryResponse
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
CategoryDTO.java
CategoryResponse.java
BaseResponse.java
CategoryRepository.java
package com.example.demo.dtos;
import com.fasterxml.jackson.annotation.JsonProperty;
import jakarta.validation.constraints.NotEmpty;
import lombok.*;
@Getter 11 usages
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Data
@Builder
public class CategoryDTO {
@JsonProperty("name")
@NotEmpty(message = "Tên không được rỗng")
private String categoryName;
}
Run
DemoApplication
Console
Actuator


project
HomeController
dtos
CategoryDTO
models
BaseEntity
Category
repositories
CategoryRepository
responses
BaseResponse
CategoryListResponse
CategoryResponse
services
CategoryService
ICategoryService
DemoApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
pom.xml (demo)
CategoryListResponse.java
CategoryDTO.java
CategoryResponse.ja
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/200
<dependencies> Edit Starters...
<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.projectlombok</groupId>
<artifactId>lombok</artifactId>
<optional>true</optional>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-validation</artifactId>
</dependency>
</dependencies>
<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
<configuration>
<excludes>
<exclude>
<groupId>org.projectlombok</groupId>


responses
BaseResponse
CategoryListResponse
CategoryResponse
services
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path/Reference...
Paste Ctrl+V
Find Usages Alt+F7
Analyze
Refactor
Validate
Bookmarks
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Override File Type
Open in Right Split Shift+Enter
Open In
Local History
Repair IDE on File
Reload from Disk
Compare With... Ctrl+D
Diagrams
Maven
Reload project
Generate Sources and Update Folders
Ignore Projects
Unlink Maven Projects
Create 'settings.xml'
Download Sources
Download Documentation
Download Sources and Documentation
Show Effective POM
Run