# **MODUL VI : RELASI ENTITY DI DALAM JAVA SPRING BOOT**



**Kompetensi Pengetahuan**

1. Peserta dapat memahami tipe relation entity
2. Peserta dapat memahami cara implementasi relasi antar entity

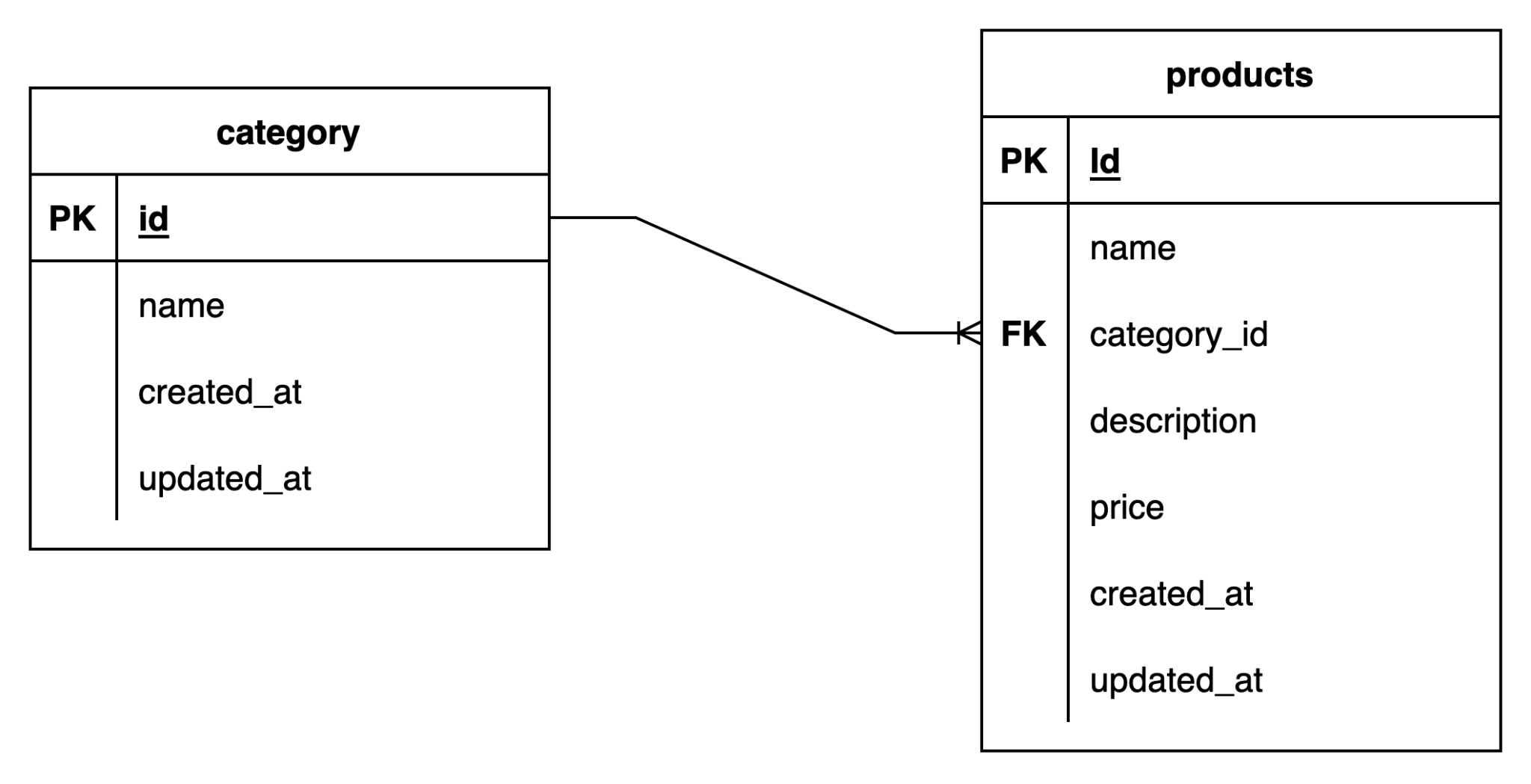
**Kompetensi Keterampilan**

1. Peserta dapat mengimplementasi untuk melakukan relation antar 2 entity di dalam Spring Boot

Pada sub modul sebelumnya telah berhasil implementasi Create dan Read pada entity product. Selanjutnya dalam sebuah service Rest API sering kali dalam setiap entity mempunyai relation ke entity lain. Adapun relation yang umumnya terjadi dalam sebuah Rest API adalah

* One to One
* One to Many
* Many to Many

Selanjutnya pada modul kali ini akan diimplementasikan One to Many dengan case entity Product akan mempunyai relation One to Many terhadap entity Category. Sehingga 1 Category nantinya akan bisa dipakai oleh banyak product seperti gambar berikut



Sedangkan untuk respons JSON dari sisi product nantinya akan mempunyai bentuk respond sebagai berikut

{

"id": 1,

"name": "Koko P21",

"description": "Baju Muslim",

"price": 50000,

"category": {

"id": "1",

"name": "baju"

},

"created\_at": "2022-02-20 11:00:00",

"updated\_at": "2022-02-21 11:00:00"

}

Langkah-langkah implementasi relation tersebut ke dalam sebuah Rest API adalah sebagai berikut

## **Other Modul Structure Approach**

Supaya lebih mempermudah management folder, pendekatan lain yang bisa digunakan adalah menggabungkan modul api dan impl sehingga modul product cukup dijadikan 1 modul

1. Pindahkan package dto yang ada di dalam product-api kedalam package controller dalam modul product-impl dengan cara cukup drag package dto dan geser ke package controller.



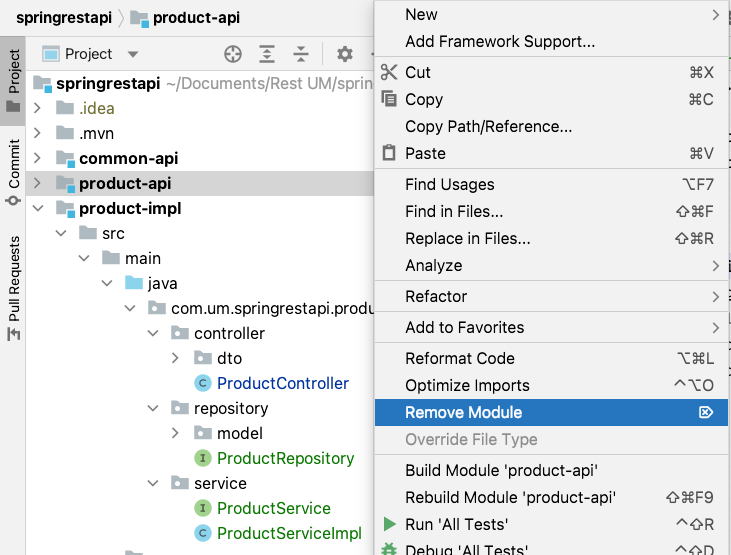
1. Pindahkan package model yang ada di dalam product-api kedalam package controller dalam modul product-impl



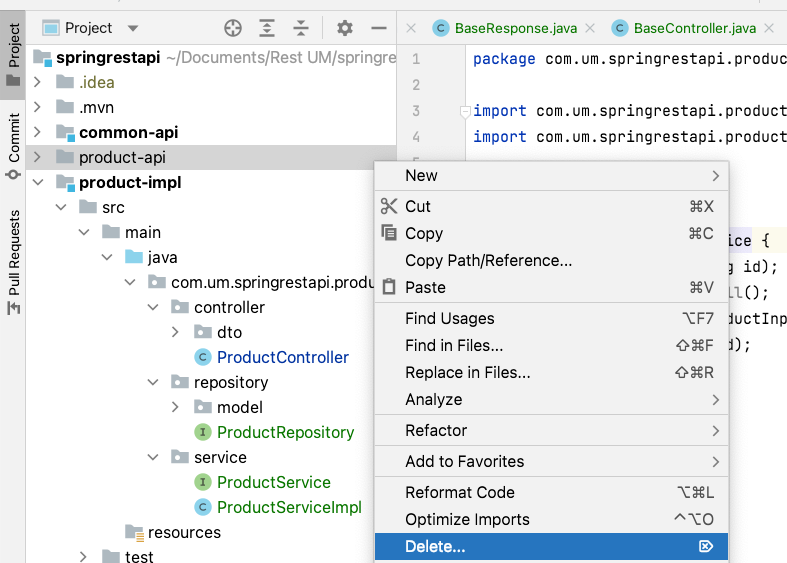
1. Pindahkan class ProductService yang ada di dalam package service module product-api kedalam package service dalam modul product-impl



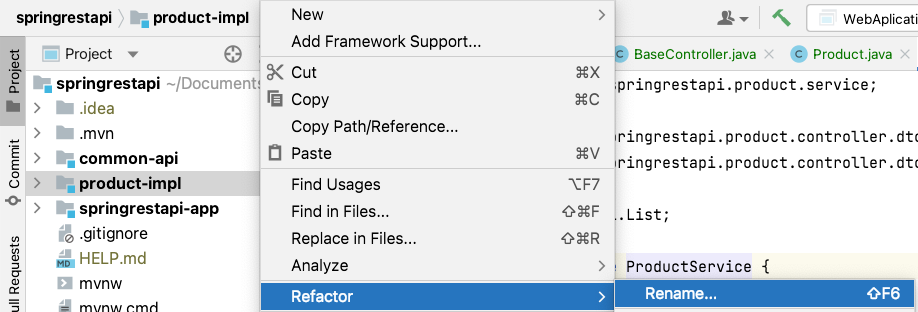
1. Selanjutnya remove modul product-api dengan cara klik kanan pada modul lalu pilih menu remove



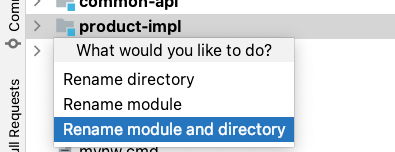
Selanjut klik kanan kembali pada product-api lalu pilih menu delete untuk menghapus modul tersebut dari project



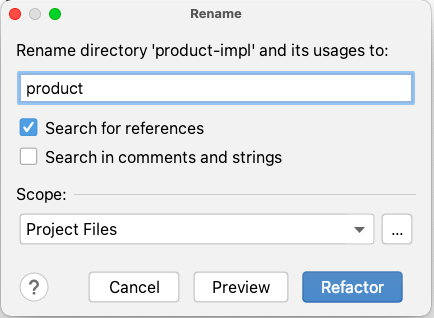
1. Ganti nama pada product-impl menjadi product dengan cara klik kanan pada modul lalu pilih menu Refactor lalu pilih menu Rename



Lalu pilih menu Rename module dan directory



Lalu ganti nama menjadi product dan klik tombol Refactor



1. Selanjutnya ubah nama modul common-api menjadi modul common



1. Ubah artifactId pada file pom.xml di dalam modul common ubah artifact id tersebut menjadi common dan tambahkan dependency object mapper, validation, jpa, dan MySQL di dalam pom.xml modul common



**Pom.xml (module common)**

*<?*xml version="1.0" encoding="UTF-8"*?>*

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>springrestapi</artifactId>

<groupId>com.um</groupId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>common</artifactId>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>org.modelmapper</groupId>

<artifactId>modelmapper</artifactId>

<version>2.4.4</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

</dependencies>

</project>

1. Ubah artifactId pada file pom.xml di dalam modul product ubah artifact id tersebut menjadi product dan dependency yang digunakan hanya dependency modul common



**Pom.xml (module product)**

*<?*xml version="1.0" encoding="UTF-8"*?>*

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>springrestapi</artifactId>

<groupId>com.um</groupId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>product</artifactId>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>com.um</groupId>

<artifactId>common</artifactId>

<version>${project.version}</version>

</dependency>

</dependencies>

</project>

1. Ubah pada file pom.xml di modul springrestapi-app dependency yang sebelumnya product-impl ubah ke dalam product



**Pom.xml (module springrestapi-app)**

. . .

<dependency>

<groupId>com.um</groupId>

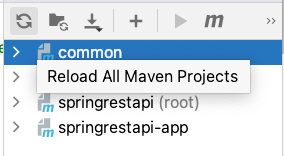
<artifactId>product</artifactId>

<version>${project.version}</version>

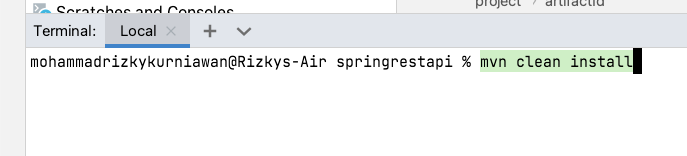
</dependency>

. . .

1. Lakukan refresh maven dan running di terminal



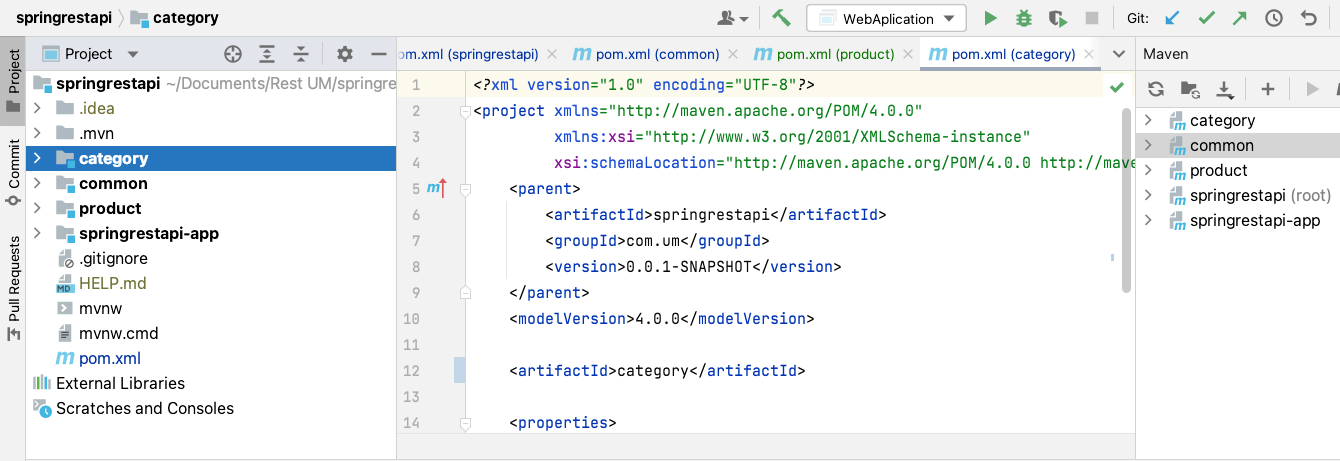
1. Lakukan perintah mvn clean install di terminal



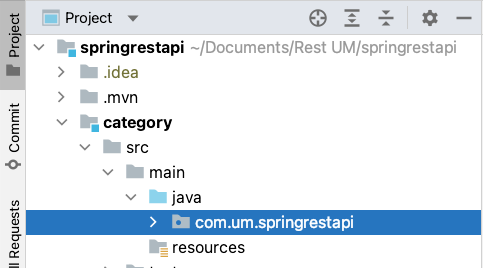
## **Category Module**

Selanjutnya pada bagian ini akan dibuat sebuah modul yang bernama category dan mempunyai endpoint Create dan Get All Category. Adapun langkah-langkah nya adalah sebagai berikut

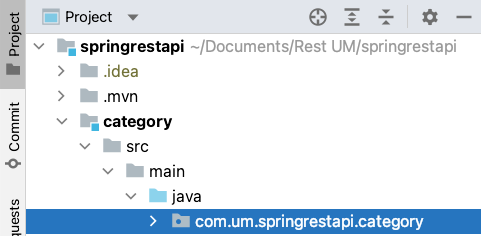
1. Buat sebuah modul baru dengan nama category



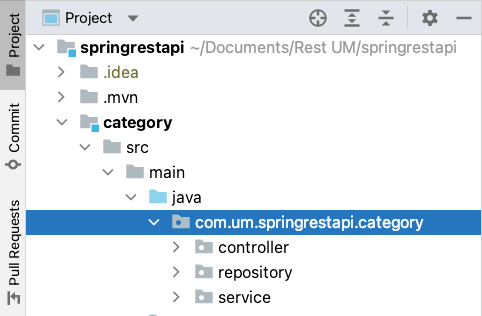
1. Buat sebuah package com.um.springrestapi di dalam folder java



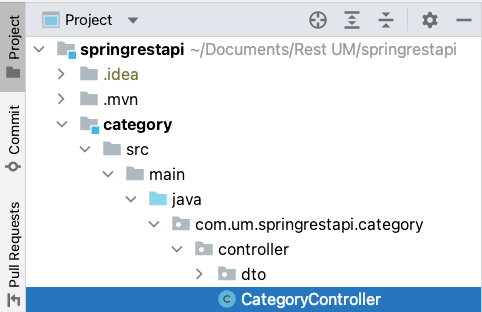
1. Selanjutnya buat package category



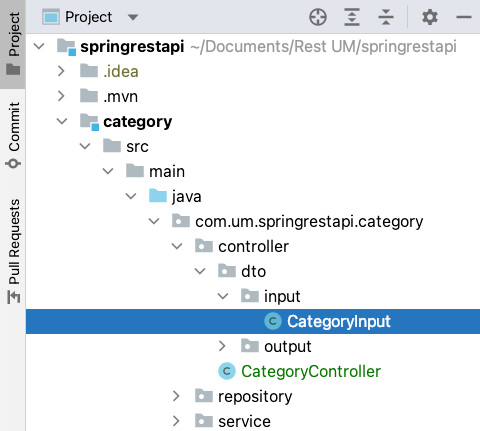
1. Selanjutnya buat 3 package yakni controller, repository, dan service



1. Pada package controller buat sebuah class CategoryController serta sebuah package dto dan di dalam dto buat package input output



1. Pada package input buat sebuah class dengan nama ProductInput dan isikan model class hanya menerima input name category





**CategoryInput.java**

package com.um.springrestapi.category.controller.dto.input;

import lombok.AccessLevel;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.NoArgsConstructor;

import javax.validation.constraints.NotBlank;

import javax.validation.constraints.Pattern;

@Data

@Builder

@NoArgsConstructor

@AllArgsConstructor(access = AccessLevel.*PRIVATE*)

public class CategoryInput {

*//regex: alphabet and space*

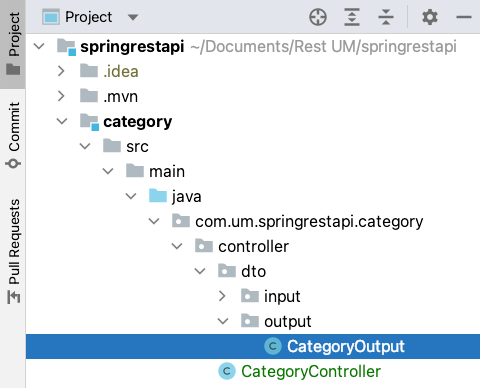
@Pattern(regexp = "[^A-Za-z0-9]+$", message = "Name only alphabet and space")

@NotBlank(message = "name category cannot be blank")

private String name;

}

1. Pada package output buat sebuah class dengan nama ProductOutput dan isikan model class hanya akan memberikan respond id dan name category





**CategoryOutput.java**

package com.um.springrestapi.category.controller.dto.output;

import lombok.\*;

@Data

@Builder

@NoArgsConstructor

@AllArgsConstructor(access = AccessLevel.*PRIVATE*)

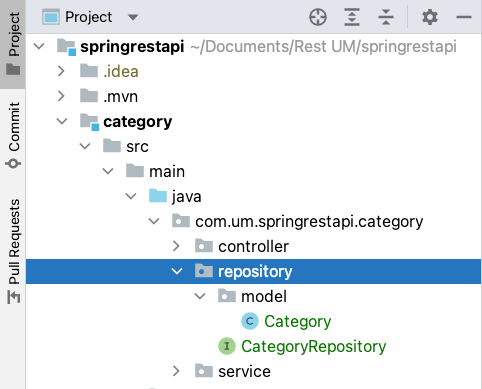
public class CategoryOutput {

private Long id;

private String name;

}

1. Selanjutnya dalam package repository buat class baru dengan CategoryRepository dan buat package model dan didalamnya buat class dengan nama Category



1. Pada model class Category buat model class untuk dijadikan sebuah model entity table category yang berisi atribut Id dan name



**Category.java**

package com.um.springrestapi.category.repository.model;

import lombok.\*;

import org.hibernate.annotations.CreationTimestamp;

import org.hibernate.annotations.UpdateTimestamp;

import javax.persistence.\*;

import java.io.Serializable;

import java.util.Date;

@Entity

@Data

@Table(name = "categories")

@Builder

@NoArgsConstructor

@AllArgsConstructor(access = AccessLevel.*PRIVATE*)

public class Category implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private Long id;

@Column(name = "name", unique = true, nullable = false, length = 255)

private String name;

@Column(name = "created\_at")

@CreationTimestamp

private Date createdAt;

@Column(name = "updated\_at")

@UpdateTimestamp

private Date updatedAt;

}

1. Isikan implementation CRUD Repository pada class CategoryRepository yang digunakan sebagai interacation Create, Read, Update, dan Delete ke dalam database



**CategoryRepository.java**

package com.um.springrestapi.category.repository;

import com.um.springrestapi.category.repository.model.Category;

import org.springframework.data.repository.CrudRepository;

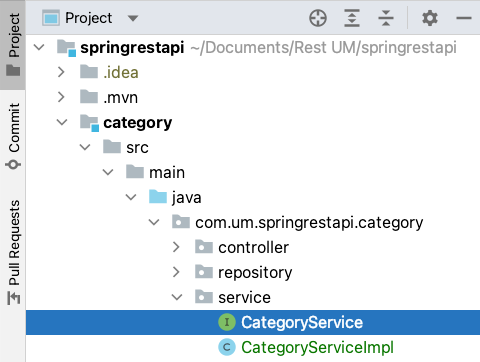
import org.springframework.stereotype.Repository;

@Repository

public interface CategoryRepository extends CrudRepository<Category, Long> {

}

1. Di dalam package service di module category buat sebuah interface dengan nama CategoryService dan class bernama CategoryServiceImpl



1. Pada interface CategoryService buat list method yang akan digunakan controller untuk memanggil logic bisnis di dalam CategoryServiceImpl



**CategoryService.java**

package com.um.springrestapi.category.service;

import com.um.sprtingrestapi.controller.dto.output.CategoryOutput;

import java.util.List;

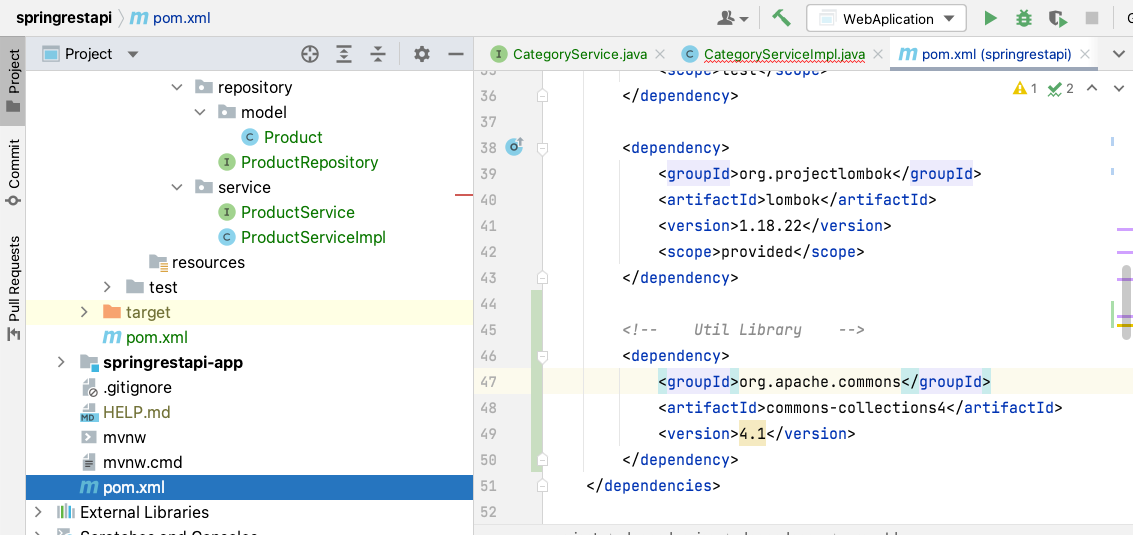
public interface CategoryService {

CategoryOutput addOne(CategoryInput input);

List<CategoryOutput> getAll();

}

1. Untuk mempermudah mengubah struktur data dari iterable ke list maka perlu ditambahkan dependency **Apache Common Collection** ke dalam modul root project





**Pom.xml (root project)**

. . .

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-collections4</artifactId>

<version>4.1</version>

</dependency>

. . .

1. Pada class CategoryServiceImpl implementasikan method untuk melakukan Create dan Read ke dalam CategoryRepository



**CategoryServiceImpl.java**

package com.um.springrestapi.category.service;

import com.um.springrestapi.category.controller.dto.input.CategoryInput;

import com.um.springrestapi.category.controller.dto.output.CategoryOutput;

import com.um.springrestapi.category.repository.CategoryRepository;

import com.um.springrestapi.category.repository.model.Category;

import org.apache.commons.collections4.IterableUtils;

import org.modelmapper.ModelMapper;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.ArrayList;

import java.util.List;

@Service

public class CategoryServiceImpl implements CategoryService{

@Autowired

private CategoryRepository repository;

@Autowired

private ModelMapper mapper;

@Override

public CategoryOutput addOne(CategoryInput input) {

Category category = mapper.map(input, Category.class);

this.repository.save(category);

return mapper.map(category, CategoryOutput.class);

}

@Override

public List<CategoryOutput> getAll() {

Iterable<Category> categories = repository.findAll();

List<Category> categoryList = IterableUtils.*toList*(categories);

List<CategoryOutput> outputs = new ArrayList<>();

for (Category category: categoryList) {

outputs.add(mapper.map(category, CategoryOutput.class));

}

return outputs;

}

}

1. Selanjutnya yang terakhir tambahkan implementasi code pada class CategoryController



**CategoryController.java**

package com.um.springrestapi.category.controller;

import com.um.restapi.controller.BaseController;

import com.um.restapi.dto.BaseResponse;

import com.um.springrestapi.category.controller.dto.input.CategoryInput;

import com.um.springrestapi.category.controller.dto.output.CategoryOutput;

import com.um.springrestapi.category.service.CategoryService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import javax.validation.Valid;

import java.util.List;

@RestController

@RequestMapping("/categories")

public class CategoryController extends BaseController {

@Autowired

private CategoryService service;

@GetMapping

public ResponseEntity<BaseResponse<List<CategoryOutput>>> getAll() {

List<CategoryOutput> categoryOutput = service.getAll();

return ResponseEntity.*ok*(new BaseResponse<>(categoryOutput));

}

@PostMapping

public ResponseEntity<BaseResponse<CategoryOutput>> insertOne(@Valid @RequestBody CategoryInput categoryInput) {

CategoryOutput categoryOutput = service.addOne(categoryInput);

return ResponseEntity.*ok*(new BaseResponse<>(categoryOutput));

}

}

1. Ubah pada file pom.xml di modul springrestapi-app tambahkan dependency category



**Pom.xml (module springrestapi-app)**

. . .

<dependency>

<groupId>com.um</groupId>

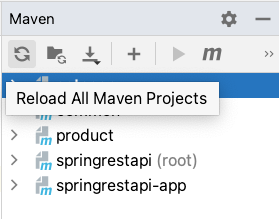
<artifactId>category</artifactId>

<version>${project.version}</version>

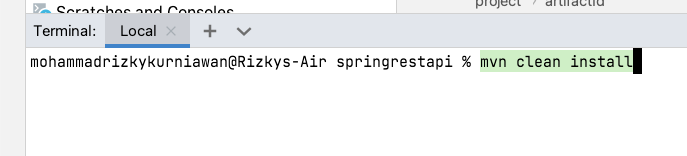
</dependency>

. . .

1. Lakukan refresh maven dan running di terminal



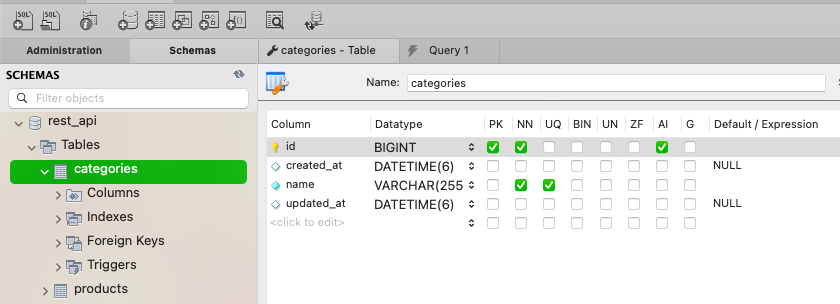
1. Lakukan perintah mvn clean install di terminal



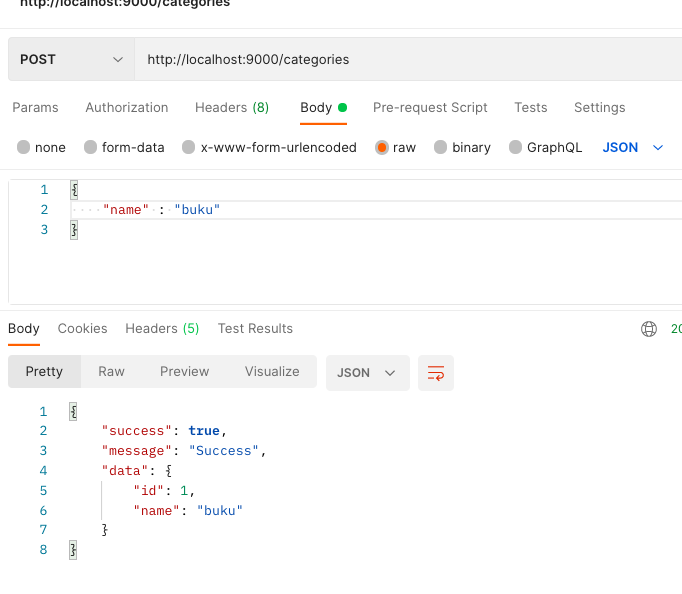
1. Lakukan run project



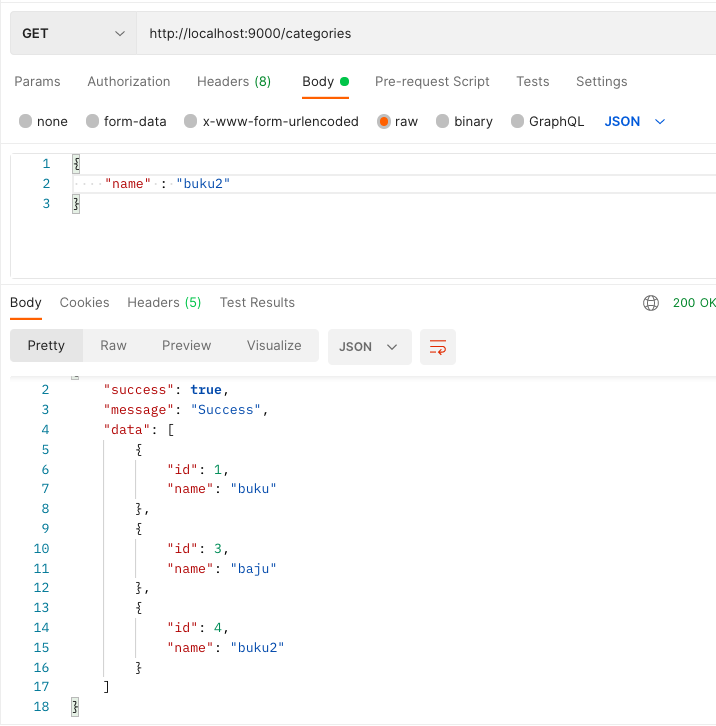
1. Pastikan Rest API telah berjalan dengan baik dengan bisa di cek di dalam database rest api sudah otomatis dibuat table baru dengan nama categories



1. Test api untuk create category dengan end point categories dan method post



1. Test api untuk get category

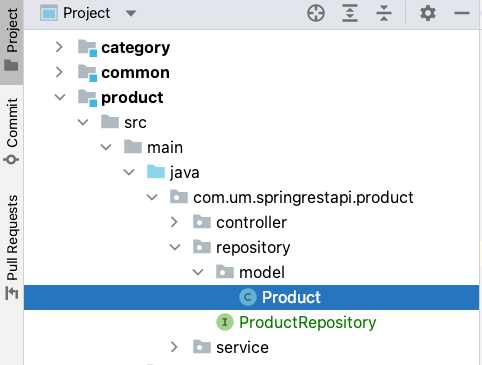


## **Relation Product and Category**

Pada bagian terakhir ini akan diimplementasikan bagaimana melakukan relasi antara entity product dan category yang mempunyai relation One to Many. Adapun langkah-langkah implementasi sebagai berikut

1. Pada class model Product tambahkan Object Category dan tambahkan annotation

* **@ManyToOne** merupakan annotation untuk menunjukkan tipe relation yang digunakan dalam object tersebut
* **@JoinColumn(name = "categoryId", referencedColumnName = "id")** merupakan annotation yang digunakan untuk mendeklarasikan foreign key dan mempunya relation ke atribut entity lain sebagai contoh pada kasus ini nantinya akan dibuat sebuah foreign key di dalam entity product yang bernama categoryId dan mempunyai referensi ke primary key Id di dalam entity category





**Product.java (module product)**

. . .

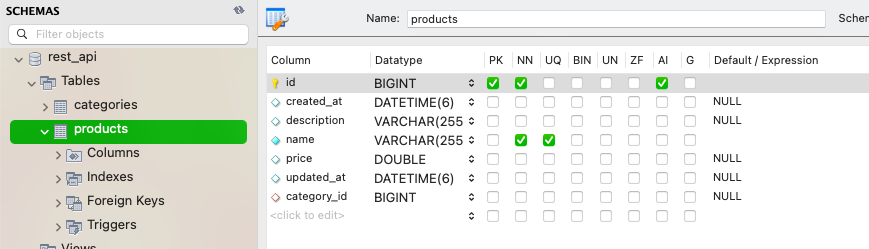
@ManyToOne

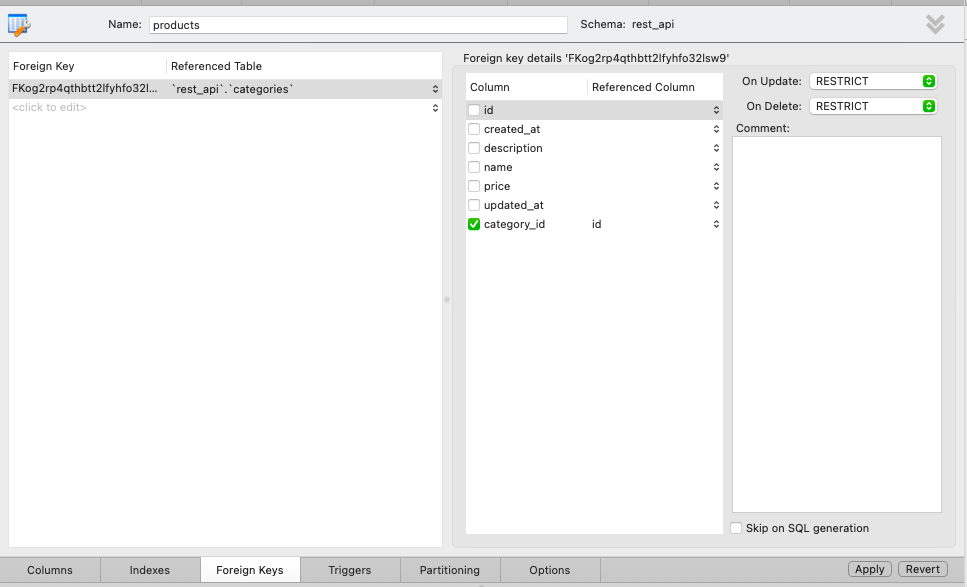
@JoinColumn(name = "categoryId", referencedColumnName = "id")

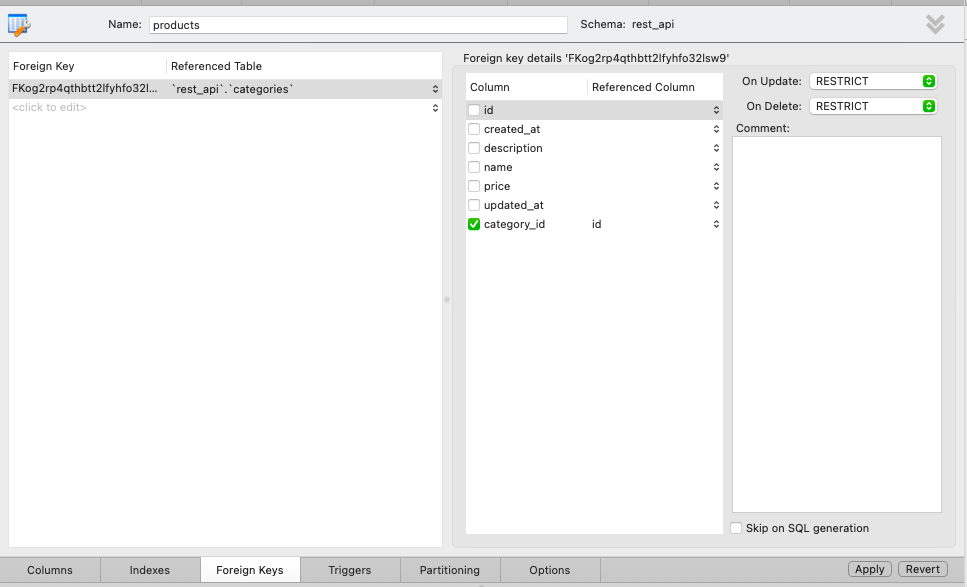
private Category category;

. . .

1. Lakukan mvn clean install pada terminal dan selanjutnya run project, maka nantinya didatabase akan terlihat sebuah atribut baru yakni categoryId yang akan sebagai foreign key dalam entity products dan mepunyai relation ke primary key category yakni id







1. Agar respond yang diberikan terdapat sebuah object category JSON selanjutnya pada class ProductOuput di dalam package dto modul product tambahkan object CategoryOutput dengan nama category



**ProductOutput.java (module product)**

@Data

@Builder

@NoArgsConstructor

@AllArgsConstructor(access = AccessLevel.*PRIVATE*)

public class ProductOutput {

private Long id;

private String name;

private String description;

private Double price;

private CategoryOutput category;

}

1. Selanjutnya tambahkan satu properti lagi untuk class ProductInput pada package dto input. Adapun properti yang ditambahkan adalah categoryId yang bertipe long mempunyai fungsi untuk menerima categoryId dari body product JSON yang akan dikirimkan oleh client sehingga selanjutnya bisa diinputkan ke dalam database



**ProductInput.java (module product)**

. . .

@NotNull(message = "categoryId cannot be null")

private Long categoryId;

. . .

1. Lakukan mvn clean install pada terminal dan selanjutnya run project. Setelah berhasil selanjutnya testing end point insert product dengan method post dan kirimkan body json yang sudah ditambahkan categoryId dengan contoh sebagai berikut

{

"name" : "Java Spring Boot Rest API",

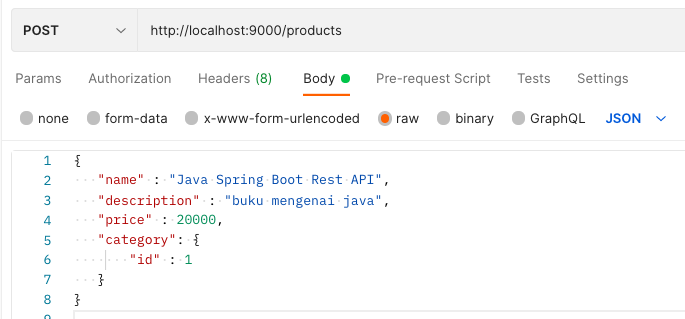
"description" : "buku mengenai java",

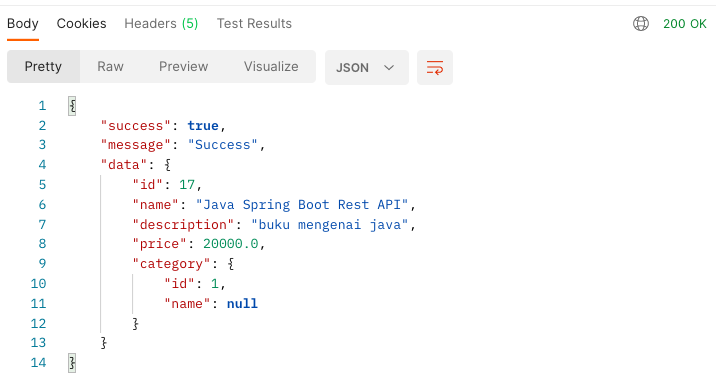
"price" : 20000,

"categoryId" : 1

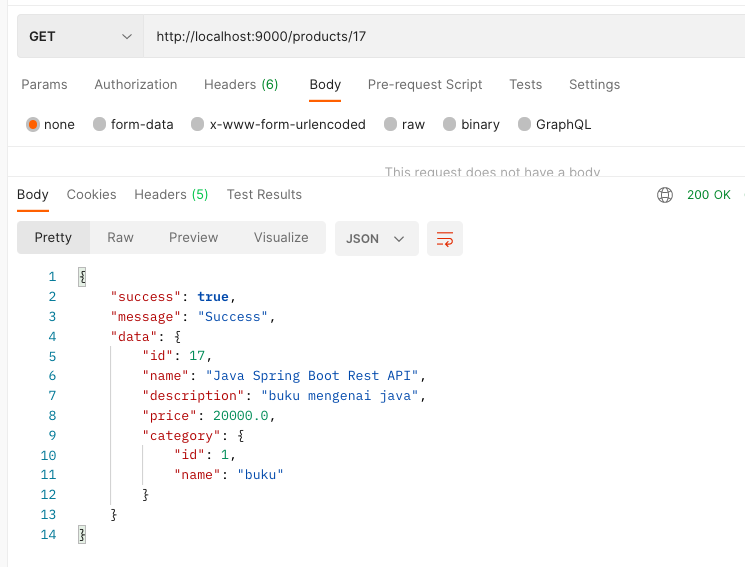
}

1. Test endpoint insert product nantinya respond berhasil akan memberikan respon sebuah data product sekaligus object category





1. Lalu testing endpoint get product by id yang nantinya akan memberikan respond sebuah object product beserta category



# 

## **Rangkuman Modul VI**

1. Dalam SQL antara entity satu dengan entity yang lain dapat ditambahkan sebum relationship
2. Tips relation antar entirety yang umum digunakan adalah One to One, One to Many, dan Many to Many
3. Contoh annotation dari JPA tuna melakukan relation ship type one to many adalah sebagai berikut

* **@ManyToOne** merupakan annotation untuk menunjukkan tipe relation yang digunakan dalam object tersebut
* **@JoinColumn(name = "categoryId", referencedColumnName = "id")** merupakan annotation yang digunakan untuk mendeklarasikan foreign key dan mempunya relation ke atribut entity lain sebagai contoh pada kasus ini nantinya akan dibuat sebuah foreign key di dalam entity product yang bernama categoryId dan mempunyai referensi ke primary key Id di dalam entity category

## 



## **Tugas Praktikum (Ketrampilan) Modul VI**

1. Buat sebuah end point detail category dengan method GET dimana dalam respond end point tersebut terdapat sebuah array of object dari sebuah product sesuai dengan category yang di request seperti contoh berikut

Method GET <http://localhost:8080/categories/1>

Respond

{

"status" : 200,

"message" : "Success",

"data": {

"id" : 3,

"name" : "buku",

"product" : [

{

"id" : 1,

"nama" : "buku anak-anak"

},

{

"id" : 2,

"nama" : "buku pengetahuan"

}

]

}

}

# 

## **Tugas Pilihan Ganda (Pengetahuan) Modul VI**

1. Buka Link Berikut

[**https://forms.gle/NHU1nSKwoE58VUTG8**](https://forms.gle/NHU1nSKwoE58VUTG8)

1. Isikan identitas dan Jawab Soal Pilihan Ganda dalam form tersebut dengan waktu 10 menit
2. Jumlah Soal terdapat 10 Soal