# **MODUL V : IMPLEMENTASI DATABASE MYSQL DENGAN SERVICE**



**Kompetensi Pengetahuan**

1. Peserta dapat memahami cara menambah, membaca, menghapus, dan mengubah data di database dari project spring boot

**Kompetensi Keterampilan**

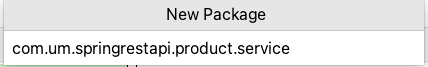
1. Peserta dapat mengimplementasi untuk menambah, membaca, menghapus, dan mengubah data di database dari project spring boot

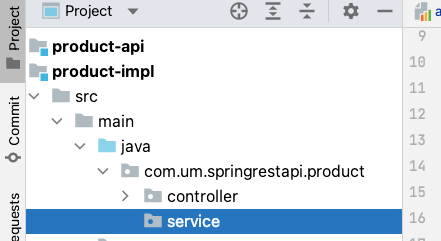
Pada modul ini akan dibahas mengenai implementasi CRUD antara project Spring Boot Rest API yang telah dibuat dengan database MySQL melalui service dan repository.

## **Service dan Repository**

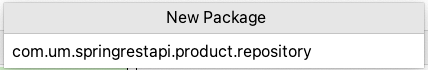
Pada sub modul ini akan dijelaskan mengenai langkah dalam membuat code pada layer service untuk business logic dan layer repository yang akan berhubungan dengan database seperti Create, Update, Delete, Read data.

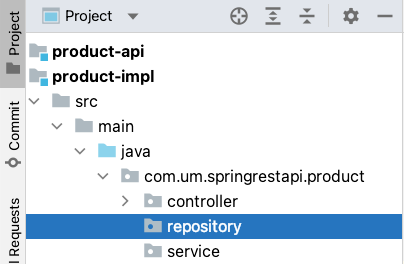
1. Buat sebuah package baru di dalam modul product-impl dengan nama service



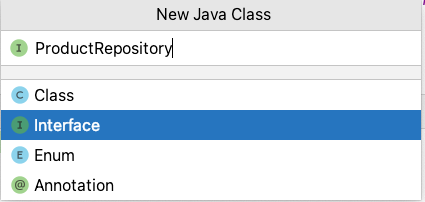


1. Buat sebuah package baru di dalam modul product-impl dengan nama repository

****

****

1. Buat sebuah interface baru pada package repository dengan nama ProductRepository



**ProductRepository.java**

package com.um.springrestapi.product.repository;

import com.um.springrestapi.product.model.Product;

import org.springframework.data.repository.CrudRepository;

import org.springframework.stereotype.Repository;

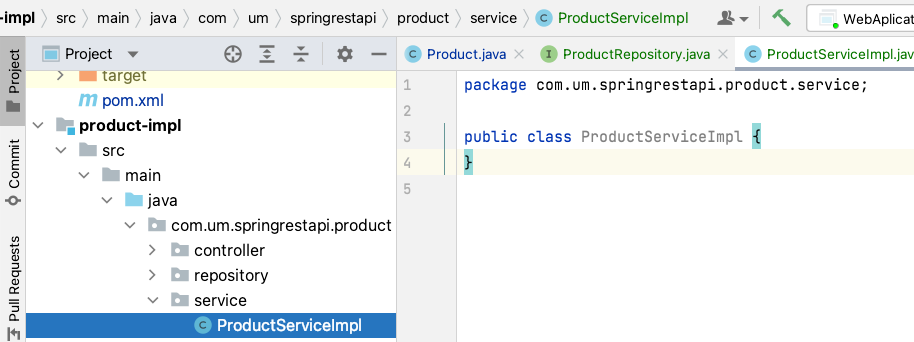
@Repository

public interface ProductRepository extends CrudRepository<Product, Long> {

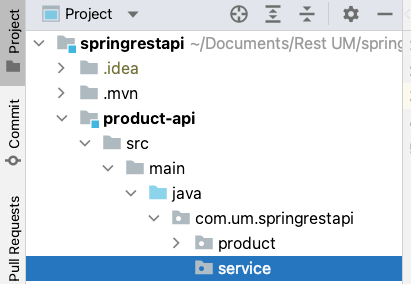
}

Dalam Java Spring Boot untuk menggunakan CRUD repository cukup tambahkan annotation @Repository dan extends CrudRepository dengan Model Product dan Id Long yang nantinya akan otomatis oleh Java Spring Boot secara Magic dapat melakukan proses Create, Read, Update, dan Delete ke dalam database.

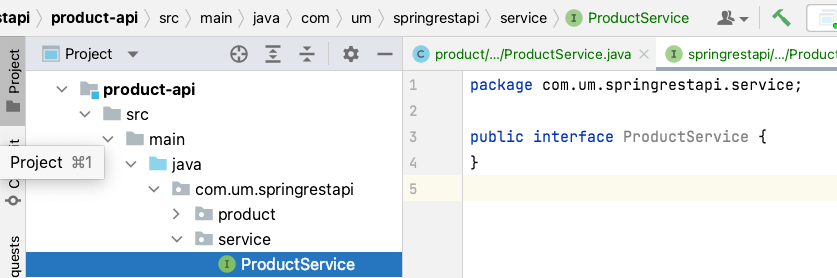
1. Buat sebuah class ProductServiceImpl pada package service di dalam modul product-impl



1. Buat sebuah package baru dengan nama service di dalam modul product-api



1. Buat sebuah interface dengan nama ProductService pada package service di dalam modul product-api



1. Isikan pada ProductService di dalam modul product-api berupa list nama function, type return and input yang akan digunakan dalam proses CRUD nantinya



**ProductService.java**

import com.um.springrestapi.product.dto.output.ProductOutput;

import java.util.List;

public interface ProductService {

ProductOutput getOne(Long id);

List<ProductOutput> getAll();

ProductOutput addOne(ProductInput input);

Boolean deleteOne(Long id);

}

1. Refresh Gradle Terlebih dahulu



1. Lalu kembali ke modul product-impl pada class ProductServiceImpl di dalam package service tambahkan implement interface dari ProductService yang ada di dalam modul product-api

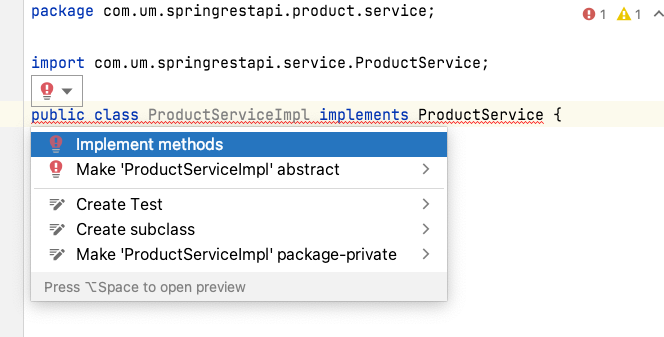
package com.um.springrestapi.product.service;

import com.um.springrestapi.service.ProductService;

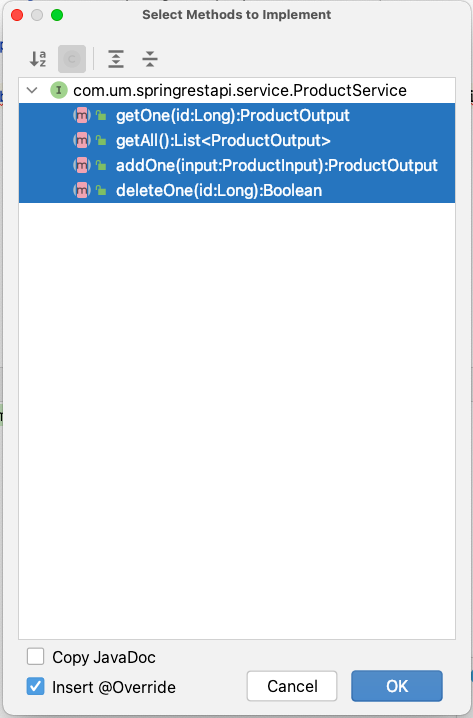
public class ProductServiceImpl implements ProductService {

}

1. Agar pada class ProductServiceImpl otomatis mengimplementasi semua method yang ada di dalam ProductService caranya adalah dengan klik tombol Lampu Merah lalu pilih menu Implements Methods



Lalu pilih tombol OK



Maka nanti akan otomatis terbuat method-method yang dibutuhkan pada interface Product Service



**ProductServiceImpl.java**

package com.um.springrestapi.product.service;

import com.um.springrestapi.product.dto.input.ProductInput;

import com.um.springrestapi.product.dto.output.ProductOutput;

import com.um.springrestapi.service.ProductService;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class ProductServiceImpl implements ProductService {

@Override

public ProductOutput getOne(Long id) {

return null;

}

@Override

public List<ProductOutput> getAll() {

return null;

}

@Override

public ProductOutput addOne(ProductInput input) {

return null;

}

@Override

public Boolean deleteOne(Long id) {

return null;

}

}

1. Tambahkan variable ProductRepository di dalam ProductServiceImpl serta untuk inisialisasi buat sebuah constructor untuk memberikan value pada ProductRepository tersebut.



**ProductServiceImpl.java**

public class ProductServiceImpl implements ProductService {

private final ProductRepository repository;

public ProductServiceImpl(ProductRepository repository) {

this.repository = repository;

}

…

1. Tambahkan code untuk mendapatkan satu product berdasarkan id pada method getOne dalam class ProductServiceImpl



**ProductServiceImpl.java**

…

@Override

public ProductOutput getOne(Long id) {

Optional<Product> product = repository.findById(id);

if (product.isEmpty()) {

return null;

}

Product temp = product.get();

return ProductOutput.*builder*()

.id(temp.getId())

.name(temp.getName())

.price(temp.getPrice())

.description(temp.getDescription())

.build();

}

…

1. Tambahkan code untuk mendapatkan satu product berdasarkan id pada method getOne dalam class ProductServiceImpl



**ProductServiceImpl.java**

…

@Override

public ProductOutput addOne(ProductInput input) {

Product product = Product.*builder*()

.name(input.getName())

.description(input.getDescription())

.price(input.getPrice())

.build();

this.repository.save(product);

return ProductOutput.*builder*()

.id(product.getId())

.name(product.getName())

.price(product.getPrice())

.description(product.getDescription())

.build();

}

…

## **Controller**

Pada sub bab ini akan dibuat sebuah code pada layer controller yang berhubungan dengan request dan response dengan client.

1. Pada class ProductController di dalam package controller pada module product-impl tambahkan ProductService serta constructor untuk melakukan default assign value.



**ProductController.java**

@RestController

@RequestMapping("/products")

public class ProductController {

private final ProductService service;

public ProductController(ProductService service){

this.service = service;

}

. . .

1. Selanjutnya ubah method pada getOne untuk mendapatkan data product dari database melalui ProductService.



**ProductController.java**

. . .

@GetMapping("/{id}")

public ResponseEntity<ProductOutput> getOne(@PathVariable Long id) {

ProductOutput productOutput = service.getOne(id);

return ResponseEntity.*ok*(productOutput);

}

. . .

1. Selanjutnya ubah method pada insertOne untuk memasukkan data product dari request client ke dalam database melalui ProductService.



**ProductController.java**

. . .

@PostMapping

public ResponseEntity<ProductOutput> insertOne(@RequestBody ProductInput productInput) {

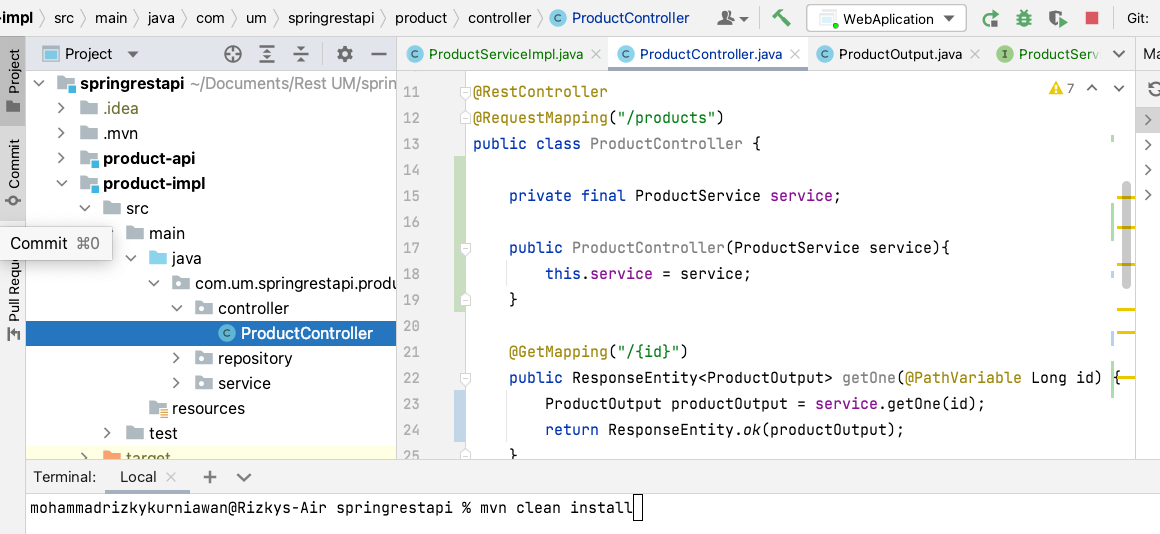
ProductOutput productOutput = service.addOne(productInput);

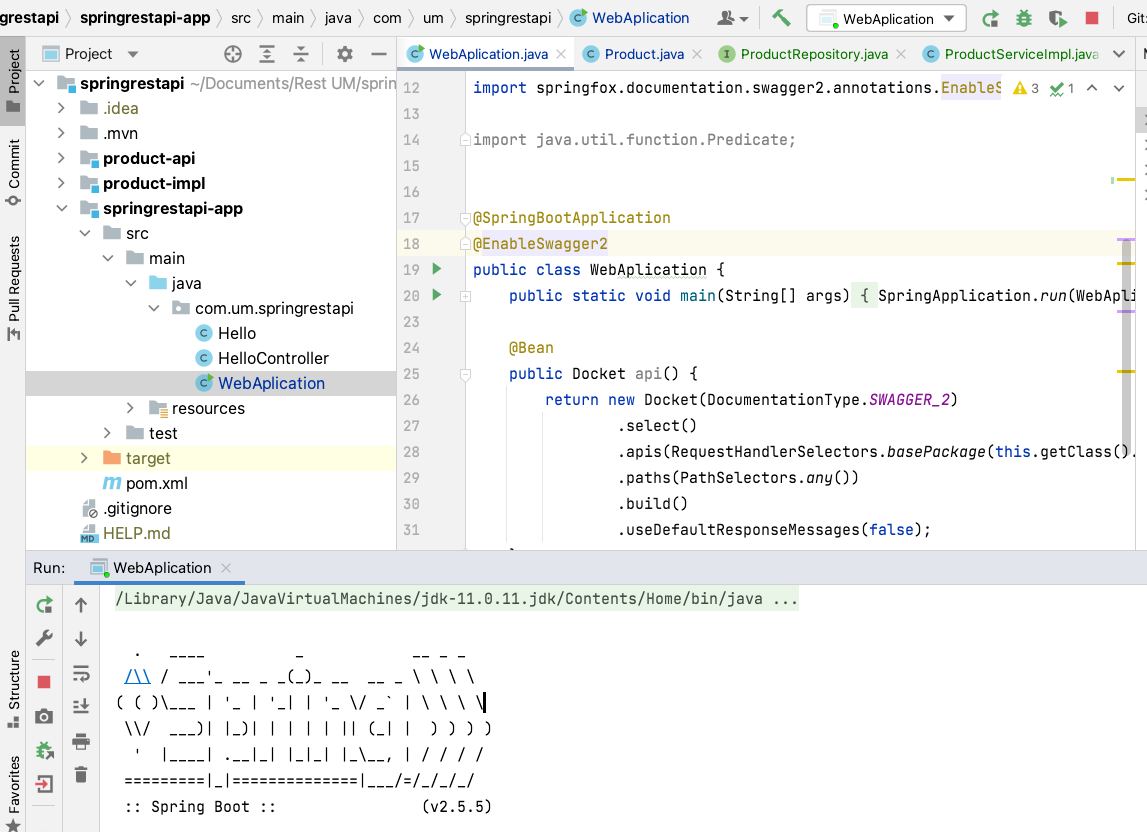
return ResponseEntity.*ok*(productOutput);

}

. . .

1. Setelah selesai lakukan mvn clean install dan running project tersebut

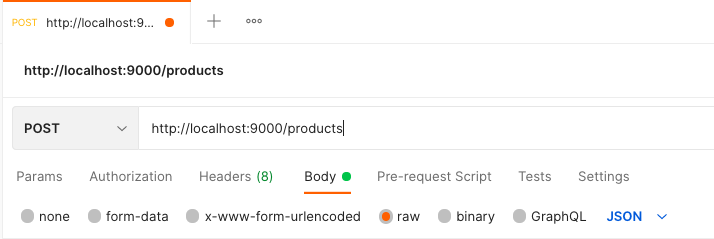




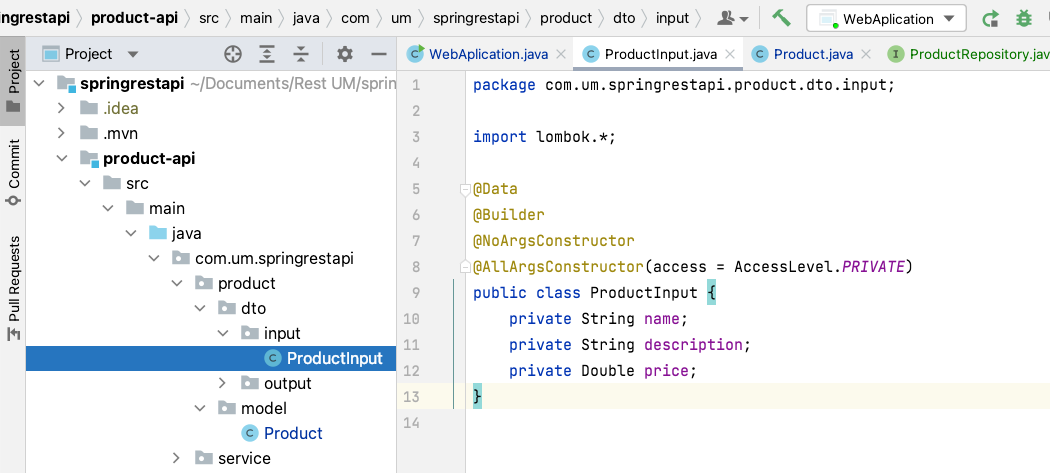
## **Testing ke Postman**

Setelah berhasil melakukan running project selanjutnya dilakukan uji coba Rest API menggunakan Postman sebagai berikut

1. Buka aplikasi postman dan buat request baru dengan menguji insert data terlebih dahulu dengan method POST dan alamat url http://localhost:9000/products



1. Selanjutnya kirim data product melalui body dengan format raw JSON, adapun data yang dikirimkan sesuai dengan format model class ProductInput

****

****

1. Selanjutnya untuk menguji EndPoint insert product klik tombol Send pada Postman dan nanti akan terdapat Respond dari Rest API yang telah dibuat dan dijalankan



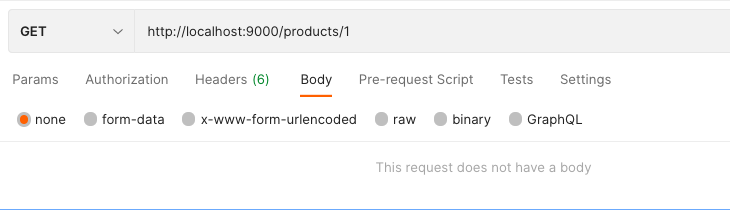


Berdasarkan hasil diatas dapat dilihat bahwa response code dari end point yang telah diuji adalah 200, hal ini berdasarkan dari konfigurasi di class controller dimana menggunakan method **ok,** sedangkan response body yang tampil adalah sebuah file JSON dari hasil input product ke database melalui ProductRepository dengan kembalian object Product yang di mapping ke model Product Output

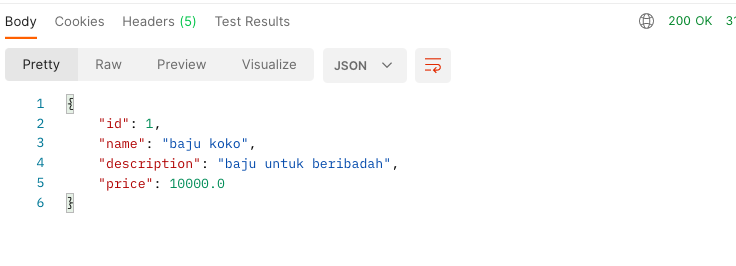
1. Untuk melihat apakah data yang dikirimkan melalui postman sudah masuk kedalam database, selanjutnya bisa di lihat melalui MySQLWorkbench pada database rest\_api table products maka akan terlihat 1 data masuk sesuai yang diinputkan melalui postman



1. Selanjutnya dilakukan test pada endpoint yang kedua yakni mendapatkan data product dengan metode get serta mengirimkan id product yakni 1 melalui URL Param



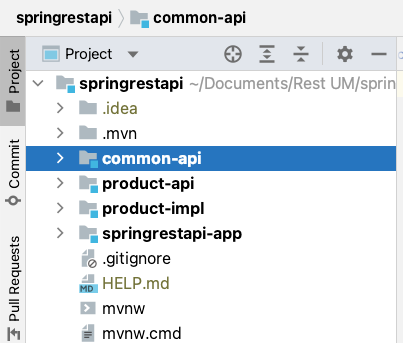
1. Hasil dari get product diatas jika berhasil akan terlihat sesuai berikut



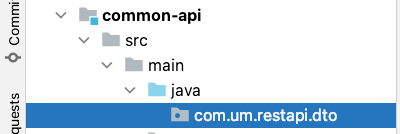
## **Response Rest API**

Response pada Rest API yang telah dibuat sebelumnya masih perlu dilakukan perbaikan. Hal ini dikarenakan standard untuk respond sebuah Rest API tidak langsung berbentuk object data dari sebuah model. Namun ada standard yang umumnya digunakan oleh industri yakni mempunya format berikut. Adapun Langkah-langkah untuk meningkatkan bentuk kualitas response Rest API yang telah dibuat adalah sebagai berikut

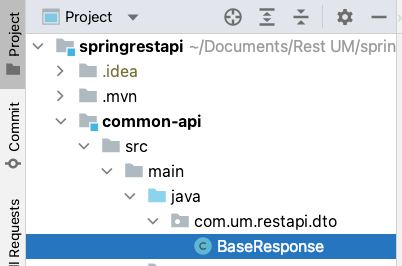
1. Buat sebuah modul baru dengan nama common-api



1. Pada modul common-api buat sebuah package baru dengan nama dto



1. Lalu buat sebuah class baru dengan nama BaseResponse yang akan digunakan sebagai base model response pada Rest API





**BaseResponse.java**

package com.um.restapi.dto;

import lombok.Data;

@Data

public class BaseResponse<T> {

private Boolean success = Boolean.*TRUE*;

private String message = "Success";

private T data;

public BaseResponse(T data) {

this.data = data;

}

public BaseResponse(Boolean success, String message) {

this.success = success;

this.message = message;

}

}

1. Implementasi base response diatas ke class Product Controller yang terdapat pada modul product-impl dengan tambahkan modul pada pom.xml di modul product-impl



1. Selanjutnya implementasi BaseResponse ke dalam class Product Controller dengan mengubah type return pada setiap method dengan sebagai berikut



**ProductController.java**

package com.um.springrestapi.product.controller;

import com.um.restapi.dto.BaseResponse;

import com.um.springrestapi.product.dto.input.ProductInput;

import com.um.springrestapi.product.dto.output.ProductOutput;

import com.um.springrestapi.service.ProductService;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/products")

public class ProductController {

private final ProductService service;

public ProductController(ProductService service){

this.service = service;

}

@GetMapping("/{id}")

public ResponseEntity<BaseResponse<ProductOutput>> getOne(@PathVariable Long id) {

ProductOutput productOutput = service.getOne(id);

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

}

@GetMapping

public ResponseEntity<BaseResponse<List<ProductOutput>>> getOne() {

return ResponseEntity.*ok*(null);

}

@PostMapping

public ResponseEntity<BaseResponse<ProductOutput>> insertOne(@RequestBody ProductInput productInput) {

ProductOutput productOutput = service.addOne(productInput);

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

}

@DeleteMapping

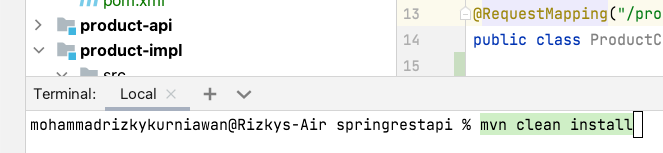
public ResponseEntity<BaseResponse<Boolean>> deleteOne(@PathVariable Long id) {

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

}

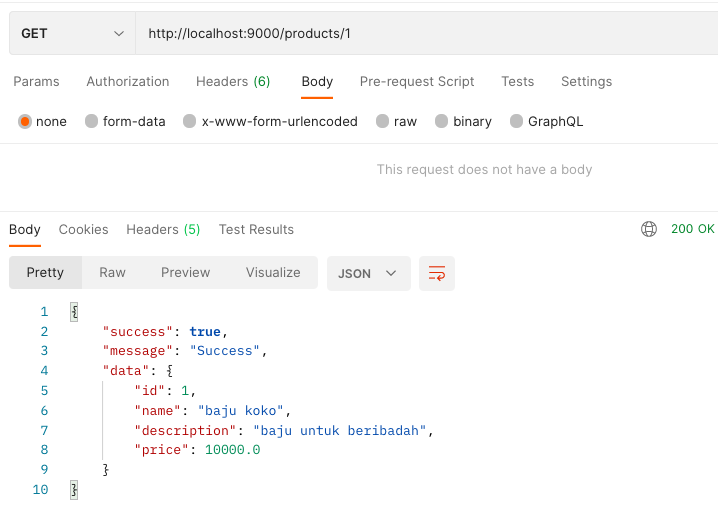
}

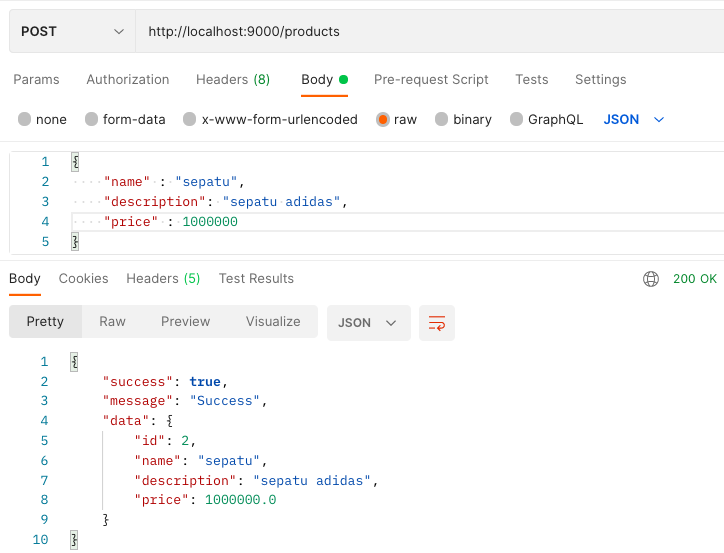
1. Setelah selesai lakukan mvn clean install dan running project tersebut





1. Lakukan test Rest API di postman untuk input product maupun get product





## **Model Mapper**

Pada sub modul sebelumnya pada class ProductService untuk melakukan mapping data model dari ProductInput ke model Product (Database), atau dari Product (Database) ke ProductOutput mempunyai cukup code yang panjang. Sebagai contoh sebagai berikut

Product product = Product.*builder*()

.name(input.getName())

.description(input.getDescription())

.price(input.getPrice())

.build();

ProductOutput.*builder*()

.id(temp.getId())

.name(temp.getName())

.price(temp.getPrice())

.description(temp.getDescription())

.build();

Supaya code yang kita buat lebih ringkas dalam melakukan mapping atau converter dari model satu ke model lainnya maka kita dapat menggunakan dependency **ModelMapper**. Adapun langkah-langkahnya adalah sebagai berikut

1. Masukkan dependency ModelMapper dengan version terbaru ke file pom.xml pada modul product-impl



**pom.xml**

<dependency>

<groupId>org.modelmapper</groupId>

<artifactId>modelmapper</artifactId>

<version>2.4.4</version>

</dependency>

1. Lakukan refresh maven



1. Buat sebuah tipe variable object ModelMapper dengan nama mapper dibawah variable repository serta tambahkan annotation **@Autowired** yang mempunyai fungsi agar otomatis akan dicari instance ModelMapper tersebut



**ProductServiceImpl.java**

...

@Autowired

private ModelMapper mapper;

...

1. Pada method getOne ubah mapping yang sebelumnya masih manual gunakan dependency ModelMapper dengan cara sebagai berikut

mapper.map([data input], [Class Model sebagai Hasil Mapping]);

Sebelum implementasi object mapper



**ProductServiceImpl.java**

. . .

@Override

public ProductOutput getOne(Long id) {

Optional<Product> product = repository.findById(id);

if (product.isEmpty()) {

return null;

}

Product temp = product.get();

return Product product = Product.*builder*()

.name(input.getName())

.description(input.getDescription())

.price(input.getPrice())

.build();

}

. . .

Sesudah implementasi object mapper



**ProductServiceImpl.java**

. . .

@Override

public ProductOutput getOne(Long id) {

Optional<Product> product = repository.findById(id);

if (product.isEmpty()) {

return null;

}

Product temp = product.get();

return mapper.map(temp, ProductOutput.class);

}

. . .

1. Pada method addOne ubah mapping yang sebelumnya masih manual gunakan dependency ModelMapper sehingga menjadi sebagai berikut

@Override

public ProductOutput addOne(ProductInput input) {

Product product = mapper.map(input, Product.class);

this.repository.save(product);

return mapper.map(product, ProductOutput.class);

}

1. Lakukan maven clean install dan running project Rest API dan testing ke dalam postman kembali untuk get data maupun insert data

## **Validation**

Pada proses input data product selanjutnya perlu dilakukan proses validasi untuk data yang diinput agar sesuai dengan ketentuan yang diinginkan sebelum dimasukkan kedalam database. Adapun langkah-langkah untuk integrasi validation adalah sebagai berikut

1. Tambahkan dependency spring boot validation pada file pom.xml di modul product-api



**pom.xml**

. . .

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>mysql</groupId>

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

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

. . .

1. Refresh maven pada project



1. Tambahkan validation pada class model ProductInput pada package input di dalam package dto product dengan menggunakan annotation berikut



**ProductInput.java**

import lombok.\*;

import javax.validation.constraints.NotBlank;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Pattern;

@Data

@Builder

@NoArgsConstructor

@AllArgsConstructor(access = AccessLevel.*PRIVATE*)

public class ProductInput {

*//regex only alphabet and space*

@Pattern(regexp = "[^A-Za-z0-9]+$")

@NotBlank(message = "Description cannot be blank")

private String name;

@NotEmpty(message = "Description cannot be empty")

private String description;

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

private Double price;

}



* **@Pattern** merupakan validasi berupa dari pola teks atau regex sesuai dengan format yang diinginkan
* **@NotBlank** merupakan validasi berupa data tidak boleh kosong dan panjang karakter lebih dari 1
* **@NotNull** merupakan validasi berupa data tidak boleh null
* **@NotEmpty** merupakan validasi berupa data tidak boleh null maupun kosong

1. Tambahkan annotation @Valid sebelum annotation @RequestBody pada method insertOne pada class ProductController didalam package controller dan modul product-impl



**ProductController.java**

. . .

@PostMapping

public ResponseEntity<BaseResponse<ProductOutput>> insertOne(@Valid @RequestBody ProductInput productInput) {

ProductOutput productOutput = service.addOne(productInput);

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

}

. . .

1. Masih di class ProductController untuk handle error validation yang sudah dibuat pada model ProductInput selanjutnya tambahkan method untuk handle error tersebut



**ProductController.java**

@ExceptionHandler(MethodArgumentNotValidException.class)

@ResponseStatus(code = HttpStatus.*BAD\_REQUEST*)

@ResponseBody

public BaseResponse<?> handleValidationError(MethodArgumentNotValidException ex) {

BindingResult bindingResult = ex.getBindingResult();

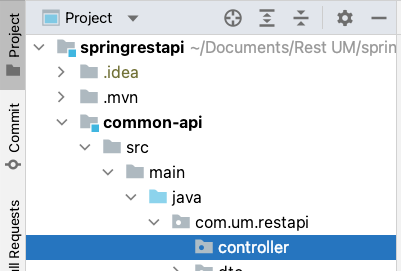
FieldError fieldError = bindingResult.getFieldError();

String defaultMessage = fieldError.getDefaultMessage();

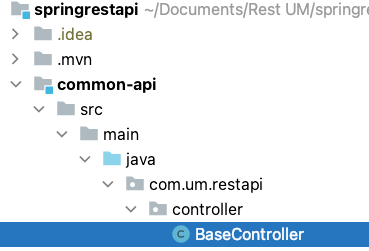
return new BaseResponse<>(false, defaultMessage);

}

1. Supaya validation message lebih rapi selanjutnya buat package controller di dalam modul common-api



1. Buat file class BaseController untuk sebagai parent controller yang ada di dalam product-impl sehingga exception menjadi lebih terstruktur



1. Pindahkan method handleValidationError yang ada dalam ProductController di dalam modul procut-impl ke dalam class BaseController di modul common-api



**ProductController.java**

import com.um.restapi.dto.BaseResponse;

import org.springframework.http.HttpStatus;

import org.springframework.validation.BindingResult;

import org.springframework.validation.FieldError;

import org.springframework.web.bind.MethodArgumentNotValidException;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.bind.annotation.ResponseStatus;

public class BaseController {

@ExceptionHandler(MethodArgumentNotValidException.class)

@ResponseStatus(code = HttpStatus.*BAD\_REQUEST*)

@ResponseBody

public BaseResponse<?> handleValidationError(MethodArgumentNotValidException ex) {

BindingResult bindingResult = ex.getBindingResult();

FieldError fieldError = bindingResult.getFieldError();

String defaultMessage = fieldError.getDefaultMessage();

return new BaseResponse<>(false, defaultMessage);

}

}

1. Lalu implemantasikan BaseController inheritance ke ProductController, sehingga pada class ProductController tambahkan extends ke BaseController



**ProductController.java**

. . .

public class ProductController extends **BaseController** {

. . .

1. Pada class BaseController di modul common-api tambahkan method handle exception yang lebih general dengan respon status code adalah 500 atau Internal Server Error



**BaseController.java**

. . .

@ExceptionHandler(Exception.class)

@ResponseStatus(code = HttpStatus.*INTERNAL\_SERVER\_ERROR*)

@ResponseBody

public BaseResponse<?> handleExceptionError(Exception ex) {

String message = ex.getMessage();

BaseResponse<?> response = new BaseResponse<>(false, message);

return response;

}

. . .

# 

## **Rangkuman Modul V**

1. CrudRepository merupakan interfeca yang ada dalam Spring Data guna melakukan operasi CRUD generik pada repositori tipe tertentu
2. Supaya code yang kita buat lebih ringkas dalam melakukan mapping atau converter dari model satu ke model lainnya maka kita dapat menggunakan dependency **ModelMapper**
3. Pada proses input data product selanjutnya perlu dilakukan proses validasi untuk data yang diinput agar sesuai dengan ketentuan yang diinginkan sebelum dimasukkan kedalam database



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

1. Buat sebuah end point untuk **multiple insert,**  **update, delete, dan getAll** dari sebuah Product
2. Tambahkan validation yang diperlukan untuk ketiga end point diatas
3. Uji coba di dalam postman dari endpoint **getOne, getAll, insert, update, dan delete**.

# 

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

1. Buka Link Berikut

[**https://forms.gle/khQGVH9Tu3wXtBxd6**](https://forms.gle/khQGVH9Tu3wXtBxd6)

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