

## M2 SPRING DATA JPA

### 1. TUJUAN

CPMK: Mahasiswa mengetahui dan memahami dasar dari web dan dasar microservices.

Sub-CPMK:

- Mahasiswa mampu mengimplementasikan Rest API sederhana dengan menggunakan bahasa Java Spring
- Mahasiswa mampu mengimplementasikan Rest API dengan database sederhana

### 2. DURASI WAKTU

1 pertemuan x 3 jam

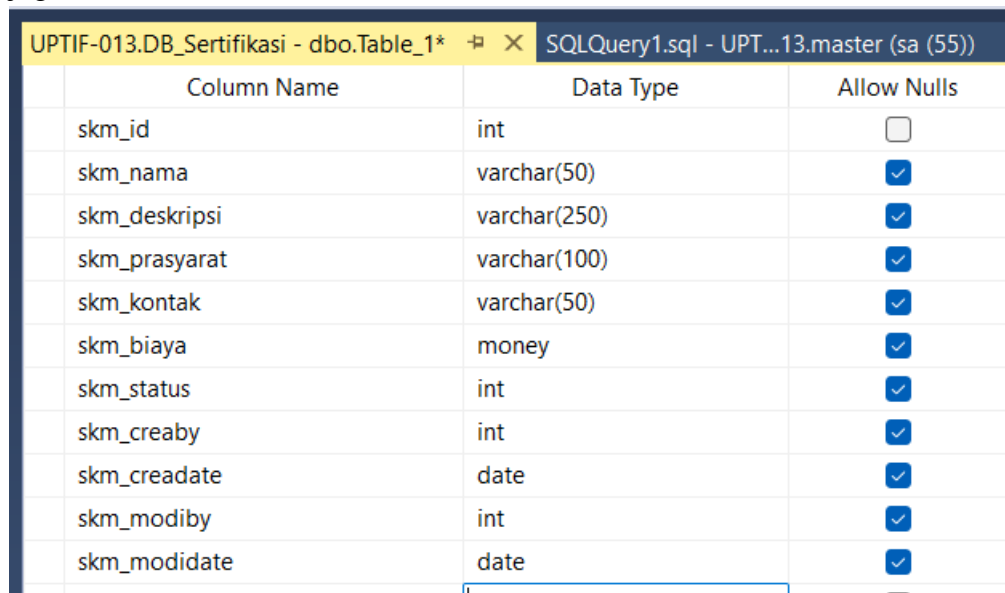
### 3. DASAR TEORI

Java Web

### 4. PERCOBAAN

#### A. Membuat Database

- Bukalah Aplikasi SQL Server Manajemen Studio
- Buatlah database dengan nama DB\_Sertifikasi
- Buatlah tabel dengan nama stf\_msskema dengan struktur seperti dibawah ini, gunakan juga **auto increment** untuk skm\_id.

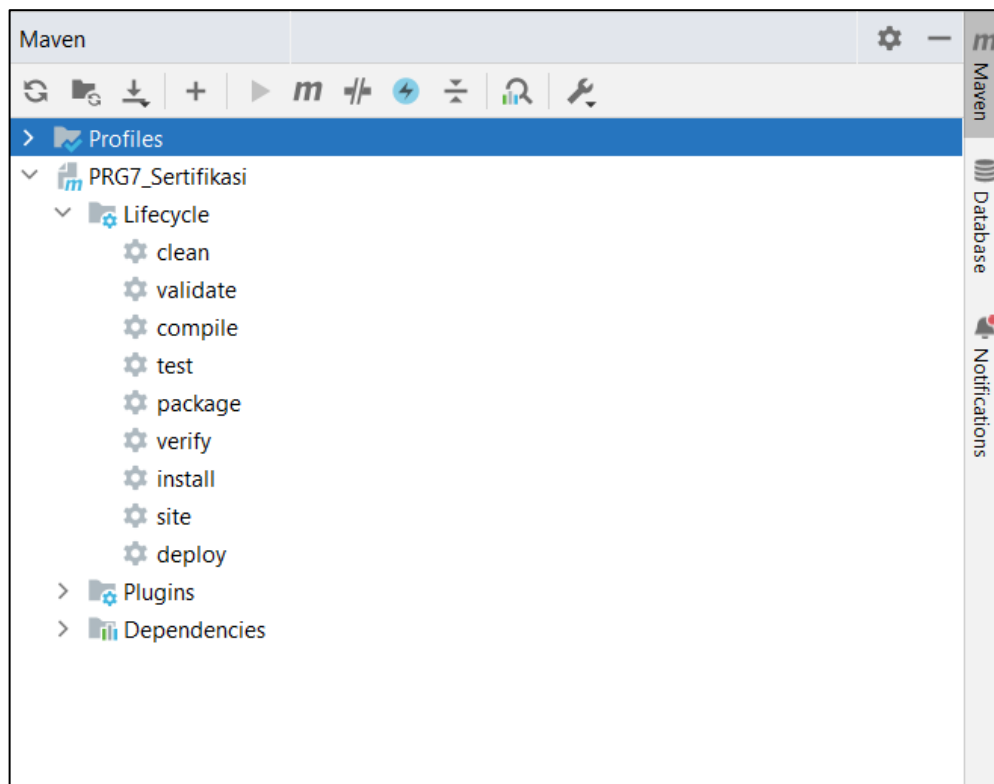


Column Name	Data Type	Allow Nulls
skm_id	int	<input type="checkbox"/>
skm_nama	varchar(50)	<input checked="" type="checkbox"/>
skm_deskripsi	varchar(250)	<input checked="" type="checkbox"/>
skm_prasyarat	varchar(100)	<input checked="" type="checkbox"/>
skm_kontak	varchar(50)	<input checked="" type="checkbox"/>
skm_biaya	money	<input checked="" type="checkbox"/>
skm_status	int	<input checked="" type="checkbox"/>
skm_creaby	int	<input checked="" type="checkbox"/>
skm_creadate	date	<input checked="" type="checkbox"/>
skm_modiby	int	<input checked="" type="checkbox"/>
skm_modidate	date	<input checked="" type="checkbox"/>

- Bukalah projek yang sebelumnya (PRG7\_Sertifikasi), tambahkan beberapa dependencies tambahan, yaitu:
  - Spring Data JPA
  - Spring Data JDBC
  - MS SQL Server Driver

```
35 <dependency>
36     <groupId>org.springframework.boot</groupId>
37     <artifactId>spring-boot-starter-test</artifactId>
38     <scope>test</scope>
39 </dependency>
40 <dependency>
41     <groupId>org.springframework.boot</groupId>
42     <artifactId>spring-boot-starter-data-jpa</artifactId>
43 </dependency>
44 <dependency>
45     <groupId>org.springframework.boot</groupId>
46     <artifactId>spring-boot-starter-jdbc</artifactId>
47 </dependency>
48 <dependency>
49     <groupId>com.microsoft.sqlserver</groupId>
50     <artifactId>mssql-jdbc</artifactId>
51     <version>10.2.2.jre17</version>
52 </dependency>
53 <dependency>
54     <groupId>org.springframework.boot</groupId>
55     <artifactId>spring-boot-starter-data-jpa</artifactId>
56 </dependency>
57 </dependencies>
```

5. Selanjutnya, coba refresh pom.xml dengan klik maven yang berada di samping kanan, lalu klik tombol refresh



6. Dalam Spring Data JPA, kita mengatur koneksi melalui application.properties

```

1 spring.datasource.url=jdbc:sqlserver://localhost:1433;databaseName=DB_Sertifikasi;encrypt=false
2 spring.datasource.username=sa
3 spring.datasource.password=polman
4 spring.jpa.show-sql=true
5 spring.jpa.hibernate.naming.physical.strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

```

7. Dalam hal ini kita mengatur beberapa hal, mulai dari url yang berisi konfigurasi datasource yang akan kita ambil, dan juga terkait spring jpa.
8. Selain itu, kita juga dapat membuat/mengupdate database secara otomatis dengan cara menggunakan fungsi ddl-auto
  - a. none – tidak menghasilkan perintah DDL apa pun
  - b. create – hanya menghasilkan database perintah create
  - c. drop – hanya menghasilkan perintah drop database
  - d. drop-and-create – menghasilkan perintah drop database diikuti dengan perintah create
9. Setelah itu, kita akan membuat DtoResponse sebagai kelas yang akan menjadi return dari setiap Rest API yang kita buat. Buatlah package response, lalu buatlah kelas DtoResponse.java dengan isi sebagai berikut

```

1 package id.co.prg7_sertifikasi.response;
2
3 public class DtoResponse {
4     4 usages
5     private Integer status;
6     4 usages
7     private Object data;
8     3 usages
9     private String message;
10
11     public DtoResponse() {
12     }
13
14     2 usages
15     public DtoResponse(Integer status, Object data) {
16         this.status = status;
17         this.data = data;
18     }
19
20     10 usages
21     public DtoResponse(Integer status, Object data, String message) {
22         this.status = status;
23         this.data = data;
24         this.message = message;
25     }
26 }

```

10. Terdapat 3 konstruktor yang dibuat, dan jangan lupa untuk mengenerate getter dan setter dari setiap atribut
11. Selanjutnya, buatlah model baru dengan nama Skema.java di package model, lalu isikan dengan kode dibawah ini

```

1  package id.co.prg7_sertifikasi.model;
2
3  import jakarta.persistence.*;
4
5  import java.math.BigDecimal;
6  import java.util.Date;
7
8  24 usages
9  @Entity
10 @Table(name = "stf_msskema")
11 public class Skema {
12     3 usages
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     @Column(name = "skm_id")
16     private Integer skm_id;
17     3 usages
18     @Column(name = "skm_nama")
19     private String skm_nama;
20     3 usages
21     @Column(name = "skm_deskripsi")
22     private String skm_deskripsi;
23     3 usages
24     @Column(name = "skm_prasyarat")
25     private String skm_prasyarat;
26     3 usages
27     @Column(name = "skm_kontak")
28     private String skm_kontak;
29
30     @Column(name = "skm_biaya")
31     private BigDecimal skm_biaya;
32     3 usages
33     @Column(name = "skm_status")
34     private Integer skm_status;
35     3 usages
36     @Column(name = "skm_creaby")
37     private Integer skm_creaby;
38     3 usages
39     @Column(name = "skm_creadate")
40     private Date skm_creadate;
41     3 usages
42     @Column(name = "skm_modiby")
43     private Integer skm_modiby;
44     3 usages
45     @Column(name = "skm_modidate")
46     private Date skm_modidate;
47
48     // Constructors, getters, and setters

```

12. Buatlah konstruktor, getter, dan juga setter dari setiap atributnya.

13. Disini kita menggunakan beberapa anotasi, diantaranya yaitu:

- a. **@Entity**: Anotasi ini digunakan untuk menandai bahwa kelas tersebut adalah entitas yang dapat dipetakan ke dalam basis data. Setiap entitas biasanya mewakili sebuah tabel dalam basis data.
- b. **@Table**: Anotasi ini digunakan untuk menentukan nama tabel yang akan digunakan untuk menyimpan entitas. Dapat digunakan untuk menyesuaikan nama

tabel jika nama kelas tidak cocok dengan nama tabel yang diinginkan dalam basis data.

- c. **@Id**: Anotasi ini digunakan untuk menandai bahwa suatu atribut adalah primary key dari tabel yang sesuai dengan entitas. Setiap entitas harus memiliki setidaknya satu atribut yang dianotasi dengan @Id.
  - d. **@GeneratedValue**: Anotasi ini digunakan untuk menandai bahwa nilai dari primary key akan di-generate secara otomatis oleh sistem. Ada beberapa jenis strategi yang dapat digunakan untuk men-generate nilai primary key, seperti IDENTITY, SEQUENCE, TABLE, dan lain-lain.
  - e. **@Column**: Anotasi ini digunakan untuk menyesuaikan properti dari kolom dalam tabel basis data. Misalnya, Anda dapat menggunakan anotasi ini untuk menentukan nama kolom, tipe data, panjang maksimum, dan sifat-sifat lainnya dari kolom tersebut.
14. Selanjutnya, kita akan membuat Vo dari Skema, buatlah SkemaVo.java di package vo, lalu isikan dengan kode dibawah ini, tambahkan juga konstruktor, getter, dan setter

<pre> 1 package id.co.prg7_sertifikasi.vo; 2 3 import id.co.prg7_sertifikasi.model.Skema; 4 5 import java.math.BigDecimal; 6 7 public class SkemaVo { 8     private Integer id; 9     private String nama; 10    private String deskripsi; 11    private String prasyarat; 12    private String kontak; 13    private BigDecimal biaya; 14    private Integer status; 15 16    public SkemaVo() { 17 </pre>	<pre> 19 @ 20 public SkemaVo(Skema skema) { 21     this.id = skema.getSkem_id(); 22     this.nama = skema.getSkem_nama(); 23     this.deskripsi = skema.getSkem_deskripsi(); 24     this.prasyarat = skema.getSkem_prasyarat(); 25     this.kontak = skema.getSkem_kontak(); 26     this.biaya = skema.getSkem_biaya(); 27     this.status = skema.getSkem_status(); 28 } 29 30 public Integer getId() { 31     return id; 32 } 33 34 public void setId(Integer id) { 35     this.id = id; 36 } 37 38 public String getNama() { 39     return nama; 40 } 41 42 public void setNama(String nama) { 43     this.nama = nama; 44 } </pre>
---	--

15. Selanjutnya, buat juga SkemaConstant di package constant dengan beberapa variabel yang akan kita gunakan nantinya

```

1 package id.co.prg7_sertifikasi.constant;
2
3 public class SkemaConstant {
4     2 usages
5     public static final String mNotFound = "Scema not found";
6     2 usages
7     public static final String mEmptyData = "No data available";
8     1 usage
9     public static final String mCreateSuccess = "Scema created successfully";
10    1 usage
11    public static final String mCreateFailed = "Failed to create Scema";
12    1 usage
13    public static final String mUpdateSuccess = "Scema updated successfully";
14    1 usage
15    public static final String mUpdateFailed = "Failed to update Scema";
16    1 usage
17    public static final String mDeleteSuccess = "Scema deleted successfully";
18    1 usage
19    public static final String mDeleteFailed = "Failed to delete Scema";
20 }

```

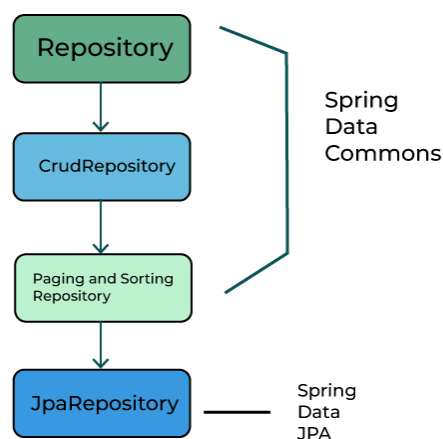
16. Selanjutnya, kita akan membuat package baru dengan nama repository, disini kita akan membuat SkemaRepository, yang dimana akan menghubungkan secara otomatis dengan database

```

1 package id.co.prg7_sertifikasi.repository;
2
3 import id.co.prg7_sertifikasi.model.Skema;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 4 usages
8 @Repository
9 public interface SkemaRepository extends JpaRepository<Skema, Integer> {
10 }

```

17. Dalam repository ini, kita meng-extends kelas JpaRepository. Biasanya, terdapat 2 yang sering kita gunakan, yaitu:



- a. **CrudRepository**: Antarmuka yang menyediakan metode CRUD dasar seperti save, findById, delete, dan lainnya. Cocok untuk aplikasi dengan kebutuhan akses data yang sederhana.
  - b. **PagingAndSortingRepository**: Turunan dari CrudRepository yang menambahkan metode untuk operasi paging dan sorting pada data. Cocok untuk mengelola daftar data yang besar dan memerlukan paginasi dan pengurutan.
  - c. **JpaRepository**: adalah antarmuka yang merupakan turunan dari PagingAndSortingRepository di dalam framework Spring Data JPA. Ini menyediakan metode CRUD dasar seperti yang disediakan oleh CrudRepository, namun juga menambahkan kemampuan untuk melakukan operasi paging dan sorting pada data seperti yang disediakan oleh PagingAndSortingRepository.
18. Selanjutnya, buatlah SkemaDao.java, letakkan di package dao dengan kode seperti dibawah ini

```
1 package id.co.prg7_sertifikasi.dao;
2
3 import id.co.prg7_sertifikasi.vo.SkemaVo;
4
5 import java.util.List;
6
7 public interface SkemaDao {
8     List<SkemaVo> getAllSkemas();
9 }
10
```

19. Selanjutnya, buatlah SkemaDaoImpl didalam package dao/impl

```
13 @Repository
14 public class SkemaDaoImpl implements SkemaDao {
15     @Autowired
16     private SkemaRepository skemaRepository;
17
18     @Override
19     public List<SkemaVo> getAllSkemas() {
20         Iterable<Skema> skemas = skemaRepository.findAll();
21         List<SkemaVo> skemaVos = new ArrayList<>();
22         for (Skema item: skemas) {
23             SkemaVo skemaVo = new SkemaVo(item);
24             skemaVos.add(skemaVo);
25         }
26         return skemaVos;
27     }
28 }
29
```

20. Disini kita menggunakan ORM Spring Data JPA dimana disini menggunakan fungsi `findAll()` yang akan dikembalikan dalam bentuk `vo`
21. Selanjutnya, kita akan memanggilnya didalam service. Buatlah kelas `SkemaService.java`, tambahkan kode seperti dibawah ini

```
1 package id.co.prg7_sertifikasi.service;
2
3 import id.co.prg7_sertifikasi.response.DtoResponse;
4
5 5 usages 1 implementation
6 public interface SkemaService {
7     1 usage 1 implementation
8     DtoResponse getAllSkemas();
9 }
```

22. Buat juga `SkemaServiceImpl` dengan kode seperti dibawah ini

```
14 @Service
15 @Transactional
16 public class SkemaServiceImpl implements SkemaService {
17     2 usages
18     @Autowired
19     private SkemaDao skemaDao;
20
21     @Autowired
22     private SkemaRepository skemaRepository;
23
24     1 usage
25     @Override
26     public DtoResponse getAllSkemas() {
27         if(skemaDao.getAllSkemas() != null) {
28             return new DtoResponse( status: 200, skemaDao.getAllSkemas());
29         }
30         return new DtoResponse( status: 200, data: null, mEmptyData);
31     }
32 }
```

23. Buatlah `SkemaRest` di package `rest`, lalu ketikkan kode berikut



```

1 package id.co.prg7_sertifikasi.rest;
2
3 import id.co.prg7_sertifikasi.response.DtoResponse;
4 import id.co.prg7_sertifikasi.service.SkemaService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.*;
7
8 @RestController
9 @RequestMapping("/skemas")
10 public class SkemaRest {
11     2 usages
12     @Autowired
13     private SkemaService skemaService;
14
15     public SkemaRest(SkemaService skemaService) {
16         this.skemaService = skemaService;
17     }
18
19     @GetMapping("/getSkemas")
20     public DtoResponse getSkemas() {
21         return skemaService.getAllSkemas();
22     }
23 }

```

24. Cobalah untuk menjalankan aplikasi ini, lalu cobalah untuk mengakses Rest API ini

The screenshot shows a REST client interface with the following details:

- URL:** http://localhost:8080/skemas/getSkemas
- Method:** GET
- Status:** 200 OK
- Time:** 1860 ms
- Size:** 897 B
- Response Body (JSON):**

```

{
  "status": 200,
  "data": [
    {
      "id": 1,
      "nama": "Programmer",
      "deskripsi": "Programmer",
      "piasyat": "web",
      "kontak": "0812575855",
      "biaya": 48000.0000
    }
  ]
}

```

25. Dapat dilihat bahwa kita telah berhasil membuat RestAPI sederhana dari database

26. Kita akan coba membuat sebuah RestAPI dengan menggunakan query native yang kita ketikkan sendiri, bukalah SkemaConstant, disini kita akan menaruh query tersebut

```

11 public static final String mDeleteFailed = "Failed to delete Scema";
12 2 usages
13 public static final String qAllDataActive = "SELECT * FROM stf_msskema WHERE skm_status = ?1";
14 }

```

27. Selanjutnya, bukalah SkemaRepository, tambahkan kode seperti dibawah ini

```

12  @Repository
13  public interface SkemaRepository extends JpaRepository<Skema, Integer> {
14      1 usage
15      @Query(value = qAllDataActive, nativeQuery = true)
16      List<Skema> findByStatus(int status);

```

28. Disini kita menggunakan anotasi @Query, @Query digunakan bersama dengan parameter value yang berisi query yang ingin dieksekusi, dan nativeQuery yang disetel sebagai true yang menunjukkan bahwa query tersebut adalah query SQL native.

29. Bukalah SkemaDao, tambahkan kode seperti dibawah ini

```

7  public interface SkemaDao {
8      2 usages 1 implementation
9      List<SkemaVo> getAllSkemas();
10     2 usages 1 implementation
11     List<SkemaVo> getSkemaActive();

```

30. Selanjutnya, implementasikan fungsi getSkemaActive didalam SkemaDaoImpl seperti dibawah ini

```

30  public List<SkemaVo> getSkemaActive() {
31      Iterable<Skema> skemas = skemaRepository.findByStatus(1);
32      List<SkemaVo> skemaVos = new ArrayList<>();
33      for (Skema item: skemas) {
34          SkemaVo skemaVo = new SkemaVo(item);
35          skemaVos.add(skemaVo);
36      }
37      return skemaVos;
38  }

```

31. Fungsi diatas akan memanggil findByStatus yang telah kita buat di repository

32. Selanjutnya, bukalah SkemaService, tambahkan kode seperti dibawah ini

```

6  public interface SkemaService {
7      1 usage 1 implementation
8      DtoResponse getAllSkemas();
9      1 usage 1 implementation
10     DtoResponse getSkemaActive();

```

33. Implementasikan dalam SkemaServiceImpl dengan kode seperti dibawah ini

```

31         @Override
32         public DtoResponse getSkemaActive() {
33             if(skemaDao.getSkemaActive() != null) {
34                 return new DtoResponse( status: 200, skemaDao.getSkemaActive());
35             }
36             return new DtoResponse( status: 200, data: null, mEmptyData);
37         }

```

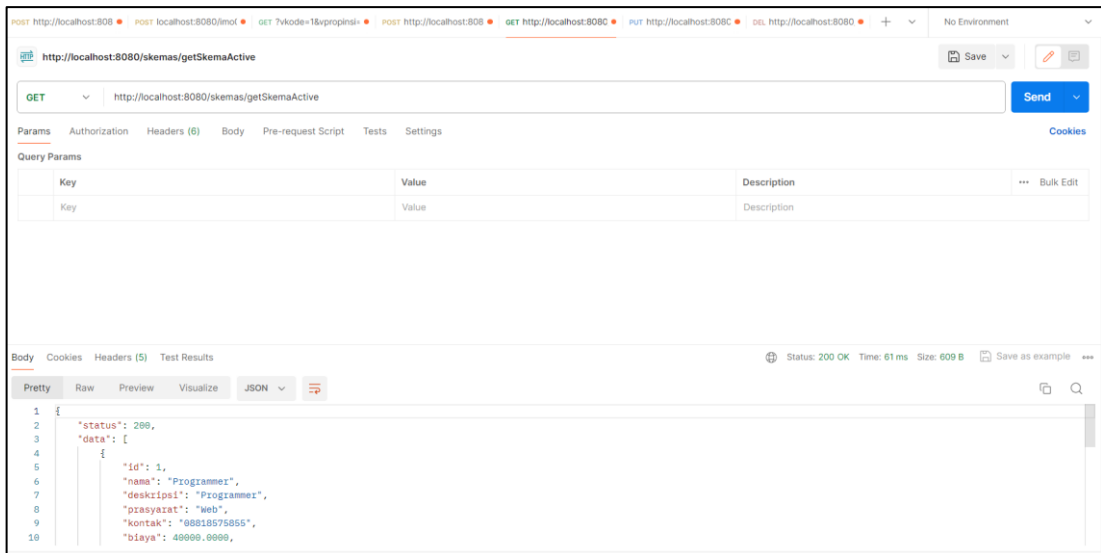
34. Didalam SkemaRest, tambahkan lagi RestAPI dengan nama fungsi getSkemaActive dengan kode seperti dibawah ini

```

24         @GetMapping("/getSkemaActive")
25         public DtoResponse getSkemaActive() {
26             return skemaService.getSkemaActive();
27         }

```

35. Coba jalankan kembali aplikasi diatas, cobalah untuk mengakses di Postman



36. Disini hanya akan menampilkan data dengan status 1  
 37. Selanjutnya, kita akan bekerja dari service langsung ke repository, tambahkan kode didalam SkemaService seperti dibawah ini

```

6         public interface SkemaService {
7             1 usage 1 implementation
8             DtoResponse getAllSkemas();
9             1 usage 1 implementation
10            DtoResponse getSkemaActive();
11            1 usage 1 implementation
12            DtoResponse saveSkema(Skema skema);
13            1 usage 1 implementation
14            DtoResponse updateSkema(Skema skema);
15            1 usage 1 implementation
16            DtoResponse deleteSkema(Skema skema);
17        }

```

38. Didalam SkemaServiceImpl tambahkan kode seperti dibawah ini

```

39      @Override
40      public DtoResponse saveSkema(Skema skema) {
41          try {
42              skemaRepository.save(skema);
43              return new DtoResponse( status: 200, skema, mCreateSuccess);
44          } catch (Exception e) {
45              return new DtoResponse( status: 500, skema, mCreateFailed);
46          }
47      }

```

39. Kode diatas akan memanggil skemaRepository dengan method save, lalu apabila berhasil akan mengembalikan DtoResponse dengan status 200, dan message mCreateSuccess.

40. Tambahkan kode untuk update dan juga delete, seperti dibawah ini

```

49      @Override
50      public DtoResponse updateSkema(Skema skema) {
51          try {
52              Skema updatedSkema = skemaRepository.save(skema);
53              if (updatedSkema != null) {
54                  return new DtoResponse( status: 200, updatedSkema, mUpdateSuccess);
55              } else {
56                  return new DtoResponse( status: 404, data: null, mNotFound);
57              }
58          } catch (Exception e) {
59              return new DtoResponse( status: 500, data: null, mUpdateFailed);
60          }
61      }
62
63      1 usage
64      @Override
65      public DtoResponse deleteSkema(Skema skema) {
66          Skema skemaData = skemaRepository.findById(skema.getSkem_id()).orElseThrow();
67          if(skemaData != null) {
68              try {
69                  skemaRepository.delete(skema);
70                  return new DtoResponse( status: 200, skemaData, mDeleteSuccess);
71              } catch (Exception e) {
72                  return new DtoResponse( status: 500, skemaData, mDeleteFailed);
73              }
74          }
75          return new DtoResponse( status: 404, data: null, mNotFound);
76      }

```

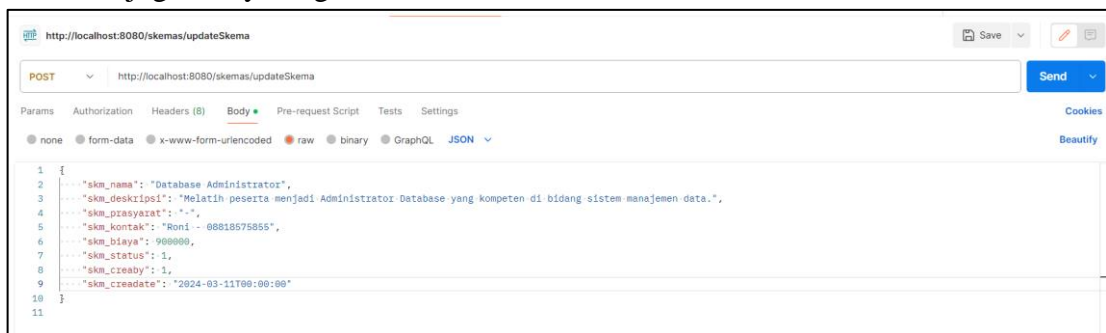
41. Setelah itu, bukalah SkemaRest, tambahkan kode Rest yang memanggil fungsi di service ini

```

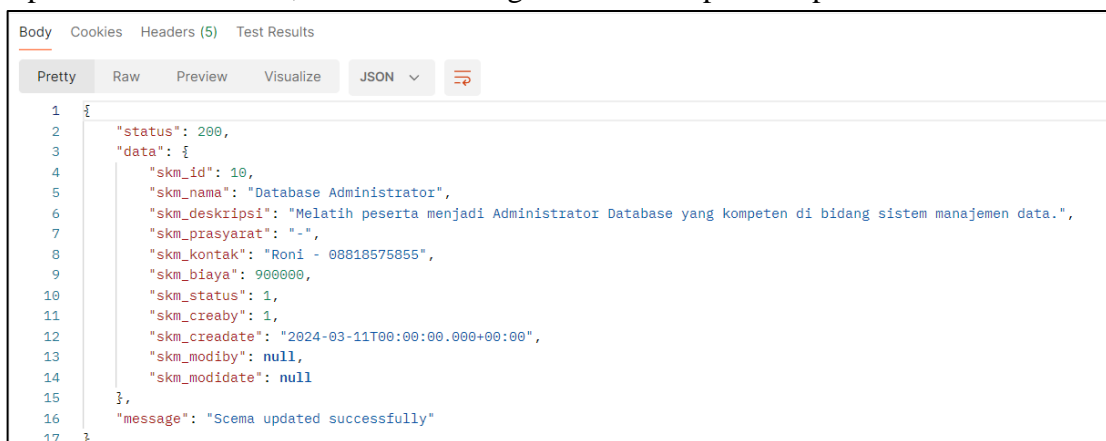
29      @PostMapping("/{saveSkema}")
30      public DtoResponse createSkema(@RequestBody Skema skema) {
31          return skemaService.saveSkema(skema);
32      }
33
34      @PostMapping("/{updateSkema}")
35      public DtoResponse updateSkema(@RequestBody Skema skema) {
36          return skemaService.updateSkema(skema);
37      }
38
39      @PostMapping("/{deleteSkema}")
40      public DtoResponse deleteSkema(@RequestBody Skema skema) {
41          return skemaService.deleteSkema(skema);
42      }

```

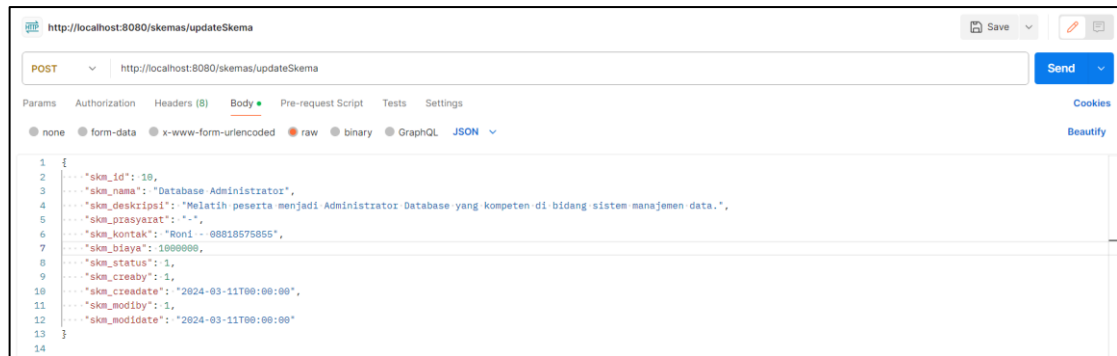
42. Kita akan mencoba satu persatu dalam menjalankan fungsi RestAPI ini. Bukalah Postman lalu gunakan method POST, gunakan link RestAPI seperti ditampilkan. Gunakan juga Body dengan raw data berbentuk JSON



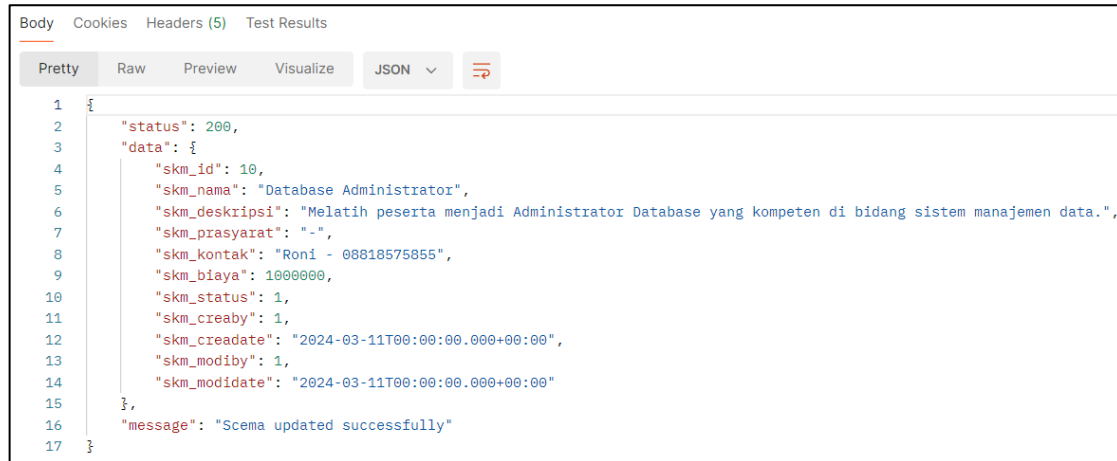
43. Apabila kita klik Send, maka akan mengembalikan response seperti dibawah ini



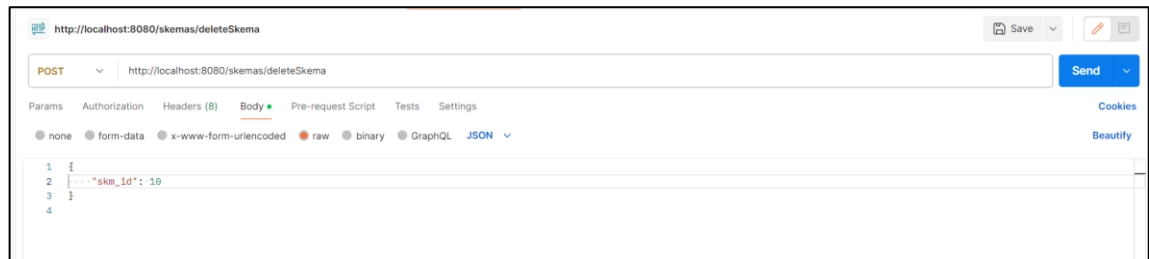
44. Untuk mengupdate, tambahkan skm\_id didalam raw datanya. Lalu coba ubah beberapa datanya



45. Selanjutnya, coba klik Send maka akan muncul response seperti dibawah ini





46. Selanjutnya, cobalah untuk menghapus salah satu data, dengan mengetikkan skm\_id nya sebagai raw datanya



## 5. LATIHAN

- Ubahlah seluruh fungsi yang telah didefinisikan di Percobaan pertemuan 2 dengan menggunakan database. Struktur database yaitu seperti dibawah ini (pro\_id dan usr\_id bersifat identity(1,1))

UPTIF-290.DB_Sertif...i - dbo.stf_msprodi SQLQuery1.sql - UPTI...ertifikasi (sa (58))*			
	Column Name	Data Type	Allow Nulls
	pro_id	int	<input type="checkbox"/>
	pro_nama	varchar(100)	<input checked="" type="checkbox"/>
	pro_singkatan	varchar(50)	<input checked="" type="checkbox"/>
	pro_jurusan	varchar(50)	<input checked="" type="checkbox"/>
	pro_status	int	<input checked="" type="checkbox"/>
	pro_creaby	int	<input checked="" type="checkbox"/>
	pro_creadate	date	<input checked="" type="checkbox"/>
	pro_modiby	int	<input checked="" type="checkbox"/>
	pro_modidate	date	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

UPTIF-013.DB_Sertif...si - dbo.stf_msuser SQLQuery5.sql - UPTI...ertifikasi (sa (63))			
	Column Name	Data Type	Allow Nulls
	usr_id	int	<input type="checkbox"/>
	usr_nama	varchar(100)	<input checked="" type="checkbox"/>
	usr_email	varchar(100)	<input checked="" type="checkbox"/>
	usr_username	varchar(100)	<input checked="" type="checkbox"/>
	usr_password	varchar(MAX)	<input checked="" type="checkbox"/>
	usr_role	varchar(100)	<input checked="" type="checkbox"/>
	usr_status	int	<input checked="" type="checkbox"/>
	pro_id	int	<input checked="" type="checkbox"/>
	usr_creaby	int	<input checked="" type="checkbox"/>
	usr_creadate	date	<input checked="" type="checkbox"/>
	usr_modiby	int	<input checked="" type="checkbox"/>
	usr_modidate	date	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

2. Buatlah CRUD dari setiap tabel beserta dengan validasi seperti yang ada di Percobaan sebelumnya
3. Password disimpan dengan menggunakan enkripsi