


<i>Session Date</i>	:	13 Desember 2019
<i>Semester</i>	:	Gasal
<i>Subject</i>	:	1132102 – Pemrograman Berorientasi Objek
<i>Week/Session</i>	:	3/3
<i>Key Topics</i>	:	<i>Creating CRUD Application using Spring Framework</i>
<i>Activity</i>	:	<i>Exercise</i>
<i>Duration</i>	:	100 minutes
<i>Objective</i>	:	Mahasiswa mampu membuat web aplikasi yang dapat melakukan operasi CRUD dengan memanfaatkan fitur: Spring Boot, Spring MVC, Spring JPA, Hibernate, dan MySQL.
<i>Lecturer</i>	:	TMP
<i>Instructor</i>	:	SEP

Persiapan:

Pada praktikum ini, Anda akan mengembangkan Web Application CRUD dengan menggunakan Spring Boot, Spring MVC, Spring JPA, Hibernate, dan MySQL sebagai *database*. Sebelum memulai praktikum, Anda dapat membuat *database* terlebih dahulu dengan rincian yang terdapat pada Gambar 1.

Field	Type
 id	int(10) unsigned NOT NULL
name	varchar(100) NOT NULL
nim	varchar(100) NOT NULL

Gambar 1. Rincian table

Langkah-langkah Praktikum:

1. Langkah awal untuk memulai proyek ini masih sama dengan praktikum sebelumnya, namun *dependencies* yang dibutuhkan tidak hanya Web saja. *Dependencies* yang akan digunakan adalah **JPA**, **MySQL**, dan **Web**.

JPA (*Java Persistence API*) merupakan sebuah spesifikasi yang terdiri atas API untuk menyimpan object (yang disebut sebagai entity) ke dalam *database relational*, biasanya menggunakan *framework* Hibernate, Java Bean. *Dependencies* yang sudah dipilih pada tahap ini akan disimpan dalam *file* konfigurasi.

2. Selanjutnya Anda perlu menuliskan konfigurasi seperti Kode Program 1 pada pom.xml Anda. POM merupakan singkatan dari *Project Object Model* yang berfungsi untuk menyimpan konfigurasi proyek Maven.

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>bootstrap</artifactId>
  <version>4.0.0</version>
</dependency>
```

Kode Program 1

3. Anda dapat membuat *package* dan *class* baru untuk menampung model yang akan Anda definisikan. Anda dapat mendefinisikan *class* Biodata seperti Kode Program 2.

```
package com.example.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="biodata")
public class Biodata {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(name="name")
    private String name;

    @Column(name="nim")
    private String nim;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getNim() {
        return nim;
    }

    public void setNim(String nim) {
        this.nim = nim;
    }
}
```

Kode Program 2

Anotasi yang biasa digunakan untuk membuat suatu table sederhana (yang paling sering digunakan) adalah:

- a) `@Entity`: digunakan sebagai inisialisasi dalam pembuatan suatu table.
- b) `@Table`: digunakan untuk inisialisasi nama table yang akan di-*custom*.
- c) `@Id`: digunakan sebagai penanda sebuah *Primary Key*
- d) `@GeneratedValue`: digunakan sebagai parameter *strategy* pada kolom yang diberi label *Primary Key*.
- e) `@Column`: digunakan untuk melakukan kustomisasi pada nama kolom.

Anotasi lainnya dapat Anda *explore* melalui *hibernate documentation*:

<http://hibernate.orh/search/documentation>.

- 4. Kemudian Anda dapat membuat *package* baru sebagai *repository* Anda. Anda dapat menambahkan interface *repository* dengan nama `BiodataRepository.java` yang akan meng-*inherit* `CrudRepository`. `BiodataRepository` pada Kode Program 3 akan digunakan untuk mengakses data yang ada di dalam *database* Anda.

```
package com.example.repository;

import org.springframework.data.repository.CrudRepository;
import com.example.model.Biodata;

public interface BiodataRepository extends CrudRepository<Biodata, Long>{

}
```

Kode Program 3

- 5. Tambahkan *package* baru yang akan digunakan sebagai *service layer* oleh proyek Anda. Kemudian tambahkan *interface* dengan nama `BiodataService.java` yang berisi seperti Kode Program 4.

```
package com.example.service;

import com.example.model.Biodata;
import java.util.List;

public interface BiodataService {
    public List<Biodata> getAllBiodata();

    public Biodata getBiodataById(long id);

    public void saveOrUpdate(Biodata biodata);

    public void deleteBiodata(long id);
}
```

Kode Program 4

6. Buatlah *class* BiodataServiceImpl.java yang akan mengimplementasikan fungsi yang terdapat pada *interface* BiodataService.java. Anda dapat mendefinisikan *interface* tersebut seperti pada Kode Program 5.

```
package com.example.service;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.example.model.Biodata;
import com.example.repository.BiodataRepository;

@Service
@Transactional
public class BiodataServiceImpl implements BiodataService{

    @Autowired
    BiodataRepository biodataRepository;

    @Override
    public List<Biodata> getAllBiodata(){
        return (List<Biodata>) biodataRepository.findAll();
    }

    @Override
    public Biodata getBiodataById(long id) {
        return biodataRepository.findById(id).get();
    }

    @Override
    public void saveOrUpdate(Biodata biodata) {
        biodataRepository.save(biodata);
    }

    @Override
    public void deleteBiodata(long id) {
        biodataRepository.deleteById(id);
    }

}
```

Kode Program 5

Bean merupakan *instance* pada suatu *class* dan diatur oleh Spring IoC Container. Anda perlu menambahkan setidaknya anotasi berikut:

- a) `@Autowired`

Anotasi yang digunakan untuk melakukan *inject instance* dari suatu bean ke objek yang memiliki *dependency*.

- b) `@Configuration`

Anotasi yang digunakan agar Spring *container* dapat membuat definisi bean dari suatu kelas dan menunjukkan bahwa suatu kelas mendeklarasikan satu atau lebih *method* `@bean` (kelas konfigurasi).

Cara lain yang dapat digunakan adalah dengan menggunakan `@ComponentScan` untuk mencari secara otomatis kelas `@Configuration`.

c) `@Component`

Anotasi yang digunakan agar Spring *container* dapat membuat definisi bean dari suatu kelas.

d) `@ComponentScan`

Anotasi yang digunakan untuk melakukan *scan* `@Component`, `@Service`, `@Controller`, `@RestController`, `@Repository` pada saat aplikasi mulai dijalankan, ketika `@Component`, `@Service`, `@Controller`, `@RestController`, `@Repository` ditemukan disinilah proses pembentukan bean terjadi yang nantinya akan digunakan untuk proses *dependency injection*.

e) `@Controller`

Anotasi yang digunakan agar Spring container dapat membuat definisi bean dari suatu kelas dan menunjukkan kelas tersebut adalah kelas *controller*.

f) `@EnableAutoConfiguration`

Anotasi yang digunakan agar Spring dapat melakukan konfigurasi secara otomatis berdasarkan *dependency* jar yang ditambahkan. (Best practice) Dalam satu proyek Spring, anotasi `@EnableAutoConfiguration` hanya boleh digunakan satu kali dan biasanya ditambahkan hanya di kelas `@Configuration` utama.

g) `@Repository`

Anotasi yang digunakan agar Spring *container* dapat membuat definisi bean dari suatu kelas dan menunjukkan kelas tersebut adalah kelas *repository*. Anotasi ini juga menandakan bahwa kelas yang menggunakan anotasi `@Repository` merupakan Java *class* yang dapat mengakses *database* secara langsung.

h) `@RestController`

Anotasi yang digunakan agar Spring *container* dapat membuat definisi bean dari suatu kelas dan menunjukkan kelas tersebut adalah kelas *rest controller*.

i) `@RequestMapping`

Anotasi yang digunakan untuk melakukan *mapping* url dengan *method*.

j) `@Service`

Anotasi yang digunakan agar Spring *container* dapat membuat definisi bean dari suatu kelas dan menunjukkan kelas tersebut adalah kelas *service*.

k) `@SpringBootApplication`

Anotasi yang digunakan untuk menggantikan deklarasi `@Configuration`, `@EnableAutoConfiguration` dan `@ComponentScan` secara bersamaan dengan menggunakan atribut default dari masing-masing anotasi yang digantikan tadi. Untuk `@ComponentScan` atribut default untuk *package* yang discan adalah package yang ada pada kelas yang memiliki anotasi ini.

7. Anda dapat membuat *package* baru yang berfungsi untuk menampung *file controller*, dan kemudian tambahkan *class* baru dengan nama `BiodataController.java` yang sudah didefinisikan pada Kode Program 5. *Controller* merupakan bagian yang berfungsi sebagai penghubung antara Model dan View. *Controller* berfungsi untuk memproses suatu data.


```

package com.example.controller;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import com.example.model.Biodata;
import com.example.service.BiodataService;

@Controller
@RequestMapping(value="/biodata")
public class BiodataController {
    @Autowired
    BiodataService biodataService;

    @RequestMapping(value="/list", method=RequestMethod.GET)
    public ModelAndView list() {
        ModelAndView model = new ModelAndView("biodata_list");
        List<Biodata> biodataList = biodataService.getAllBiodata();
        model.addObject("biodataList", biodataList);

        return model;
    }
    @RequestMapping(value="/addBiodata/", method = RequestMethod.GET)
    public ModelAndView addBiodata() {
        ModelAndView model = new ModelAndView();

        Biodata biodata = new Biodata();
        model.addObject("biodataForm", biodata);
        model.setViewName("biodata_form");

        return model;
    }
    @RequestMapping(value="/updateBiodata/{id}", method = RequestMethod.GET)
    public ModelAndView editArticle(@PathVariable long id) {
        ModelAndView model = new ModelAndView();

        Biodata biodata = biodataService.getBiodataById(id);
        model.addObject("biodataForm", biodata);
        model.setViewName("biodata_form");

        return model;
    }
    @RequestMapping(value="/saveBiodata", method=RequestMethod.POST)
    public ModelAndView save(@ModelAttribute("biodataForm") Biodata biodata) {
        biodataService.saveOrUpdate(biodata);

        return new ModelAndView("redirect:/biodata/list");
    }
    @RequestMapping(value="/deleteBiodata/{id}", method=RequestMethod.GET)
    public ModelAndView delete(@PathVariable("id") long id) {
        biodataService.deleteBiodata(id);

        return new ModelAndView("redirect:/biodata/list");
    }
}

```

Kode Program 6

8. Pada *file* `application.properties`, tambahkan *property* sesuai dengan Kode Program 7. Properti tersebut akan digunakan dalam melakukan konfigurasi MySQL, Spring Data JPA dan View Resolvers.

```
#database
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/latihan
spring.datasource.username=root

#spring_jpa
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

spring.mvc.view.prefix=/WEB-INF/
spring.mvc.view.suffix=.jsp
```

Kode Program 7

9. Anda dapat meletakkan *view layer* Anda dalam *package* `src/main/webapp/WEB-INF`. Buatlah *file* `jsp` dengan nama `biodata_list.jsp` (Kode Program 8) dan `biodata_form.jsp` (Kode Program 9) yang akan menggunakan `spring` taglib, `form` taglib dan `JSTL` taglib.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Biodata Form</title>
<link href="../../webjars/bootstrap/4.0.0/css/bootstrap.min.css" rel="stylesheet" />
<script src="../../webjars/bootstrap/4.0.0/js/bootstrap.min.js"></script>
<script src="../../webjars/jquery/3.0.0/js/jquery.min.js"></script>
</head>
<body>
<div class="container">
<spring:url value="/biodata/saveBiodata" var="saveURL" />
<h2>Biodata</h2>
<form:form modelAttribute="biodataForm" method="post" action="${saveURL}" cssClass="form" >
<form:hidden path="id"/>
<div class="form-group">
<label>Name</label>
<form:input path="name" cssClass="form-control" id="name" />
</div>
<div class="form-group">
<label>NIM</label>
<form:input path="nim" cssClass="form-control" id="nim" />
</div>
<button type="submit" class="btn btn-primary">Save</button>
</form:form>
</div>
</body>
</html>
```

Kode Program 8

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Biodata List</title>
<link href="../../webjars/bootstrap/4.0.0/css/bootstrap.min.css" rel="stylesheet" />
<script src="../../webjars/bootstrap/4.0.0/js/bootstrap.min.js"></script>
<script src="../../webjars/jquery/3.0.0/js/jquery.min.js"></script>
</head>
<body>
<div class="container">
<h2>Biodata List</h2>
<table class="table table-striped">
<thead>
<th scope="row">#ID</th>
<th scope="row">Name</th>
<th scope="row">NIM</th>
<th scope="row">Update</th>
<th scope="row">Delete</th>
</thead>
<tbody>
<c:forEach items="${biodataList}" var="biodata">
<tr>
<td>${biodata.id}</td>
<td>${biodata.name}</td>
<td>${biodata.nim}</td>
<td>
<spring:url value="/biodata/updateBiodata/${biodata.id}" var="updateURL" />
<a class="btn btn-primary" href="${updateURL}" role="button">Update</a>
</td>
<td>
<spring:url value="/biodata/deleteBiodata/${biodata.id}" var="deleteURL" />
<a class="btn btn-primary" href="${deleteURL}" role="button">Delete</a>
</td>
</tr>
</c:forEach>
</tbody>
</table>
<spring:url value="/biodata/addBiodata/" var="addURL" />
<a class="btn btn-primary" href="${addURL}" role="button">Add New Biodata</a>
</div>
</body>
</html>

```

Kode Program 9

10. *Build and run Web Application* dengan cara:

Select Run As > Maven Clean

Select Run As > Maven Install

Select Run As > Spring Boot

11. Jika sudah berhasil, maka Anda dapat menjalankan proyek Anda pada *browser* Anda untuk melakukan operasi Create, Update, dan Delete.

Selamat Mengerjakan ☺