❖ **Hal yang dipelajari dalam tutorial kali ini**

Pada tutorial kali ini mempelejari beberapa hal, antara lain cara menghubungkan project pada SpringBoot dengan database MySQL. Untuk menghubungkannya bisa menggunakan MyBatis yang berguna untuk melakukan koneksi dan *generate query* dengan *helper annotation*. Dalam tutorial ini juga dipelajari mengenai *aplication.properties*. *Aplication properties* merupakan sebuah file yang berada dalam folder *src/main/resource*, file ini berisi konfigurasi dari aplikasi yang sedang dikerjakan, antara lain port, database config, username dan password phpMyAdmin. Dalam tutorial ini juga dipelajari cara memberika argumen log pada *project* yang dibuat. Log ini berguna untuk melakukan *debugging* pada Spring Boot. Untuk memberikan argumen log ini biasanya menggunakan Slf4j yang ada pada library eksternal Lombok. Untuk menggunakannya bisa dengan cara menambahkan anotasi @Slf4j diatas kelas yang ingin diberikan log.

❖ **Penjelasan *method delete***

Menambahkan beberapa kode pada viewall.html seperti pada gambar dibawah ini

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>View All Students</title>
    </head>
    <body>
        <h1>All Students</h1>

        <div th:each="student,iterationStatus: ${students}">
            <a th:href="'/student/delete/' + ${student.npm}">Delete Data</a><br/>
            <a th:href="'/student/update/' + ${student.npm}">Update Data</a><br/>
            <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
            <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
            <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
            <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
            <hr/>
        </div>
    </body>
</html>
```

Kode diatas berguna untuk menambahkan link Delete Data untuk menghapus data yang ada di database.

Menambahkan method deleteStudent pada *class* StudentMapper seperti gambar dibawah ini.

```
package com.example.dao;

import java.util.List;

@Mapper
public interface StudentMapper
{
    @Select("select npm, name, gpa from student where npm =
    StudentModel selectStudent (@Param("npm") String npm);

    @Select("select npm, name, gpa from student")
    List<StudentModel> selectAllStudents ();

    @Insert("INSERT INTO student (npm, name, gpa) VALUES (#{
    void addStudent (StudentModel student);

    @Delete("DELETE FROM student where npm = #{npm}")
    void deleteStudent (@Param("npm") String npm);

    @Update("UPDATE student SET name = #{name}, gpa = #{gpa}
    void updateStudent (StudentModel student);
}
```

Kode diatas merupakan *script* SQL yang berguna untuk menghapus data *student* di database dengan inputan berupa NPM pada url atau bisa juga dengan langsung mengklik link Delete Data di browser.

Method deleteStudent pada *class* StudentServiceDatabase seperti pada gambar dibawah ini.

```
@Override
public void deleteStudent (String npm)
{
    log.info("student" + npm + " deleted");
    studentMapper.deleteStudent(npm);
}
```

Method diatas merupakan log untuk *delete student* serta memanggil methodStudent yang ada pada *class* studentMapper.

Method deleteStudent pada *class* StudentController seperti pada gambar dibawah ini.

```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent (npm);
    if (student != null) {
        studentDAO.deleteStudent (npm);
    }
    else {
        model.addAttribute("npm", npm);
        return "not-found";
    }
    return "delete";
}
```

Method diatas merupakan *controller* untuk melakukan *delete* data *student*. Pada method diatas dilakukan validasi auntuk mengecek apakah NPM yang diinputkan di url ada didatabase atau tidak. Jika ada maka akan dipanggil method deleteStudent pada class StudentService dan mengembalikan halaman delete.html. Jika data NPM tidak ditemukan maka akan mengembalikan halaman not-found.html.

- **Penjelasan *method update***

Menambahkan method updateStudent pada *class* StudentMapper

```
import java.util.List;

@Mapper
public interface StudentMapper
{
    @Select("select npm, name, gpa from student where npm = #{npm}")
    StudentModel selectStudent (@Param("npm") String npm);

    @Select("select npm, name, gpa from student")
    List<StudentModel> selectAllStudents ();

    @Insert("INSERT INTO student (npm, name, gpa) VALUES (#{npm}, #{name}, #{gpa})")
    void addStudent (StudentModel student);

    @Delete("DELETE FROM student where npm = #{npm}")
    void deleteStudent (@Param("npm") String npm);

    @Update("UPDATE student SET name = #{name}, gpa = #{gpa} WHERE npm = #{npm}")
    void updateStudent (StudentModel student);
}
```

Method diatas berisi *script* SQL untuk melakukan *update* data mahasiswa yang ada diatabase dengan menerima inputan berupa objek *student*.
Method updateStudent pada interface StudentService

```
public interface StudentService
{
    StudentModel selectStudent (String npm);

    List<StudentModel> selectAllStudents ();

    void addStudent (StudentModel student);

    void deleteStudent (String npm);

    void updateStudent (StudentModel student);
}
```

Method updateStudent pada *class* StudentServiceDatabase.

```
    @Override
    public void updateStudent (StudentModel student) {
        log.info("Student {} updated to {}", student.getNpm(),student.getName());
        studentMapper.updateStudent(student);
    }
```

Method ditasa berisi log untuk melakukan *update student* serta pada method diatas juga diapanggil method updateStudent yang ada pada class studentMapper.

Link Update Data pada halaman viewall.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>View All Students</title>
    </head>
    <body>
        <h1>All Students</h1>

        <div th:each="student,iterationStatus: ${students}">
            <a th:href="'/student/delete/' + ${student.npm}">Delete Data</a><br/>
            <a th:href="'/student/update/' + ${student.npm}">Update Data</a><br/>
            <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
            <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
            <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
            <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
            <hr/>
        </div>
    </body>
</html>
```

Link Update Data berguna untuk melakukan update data student langsung pada halaman browser.
Halaman form-update.html

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>

<title>Update student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

    <h1 class="page-header">Update Editorr</h1>

    <form action="/student/update/submit" method="post">
        <div>
            <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${student.npm}"/>
        </div>
        <div>
            <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" />
        </div>
        <div>
            <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}"/>
        </div>

        <div>
            <button type="submit" name="action" value="save">Update</button>
        </div>
    </form>

</body>

</html>
```

Pada halaman diatas ada beberapa poin yang perlu di-*highlight*. Pertama, terdapat anotasi readonly dan th:value seperti yang ditampilkan pada gambar dibawah ini.

```
<h1 class="page-header">Update Editorr</h1>

<form action="/student/update/submit" method="post">
    <div>
        <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${student.npm}"/>
    </div>
    <div>
        <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" />
    </div>
    <div>
        <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}"/>
    </div>

    <div>
        <button type="submit" name="action" value="save">Update</button>
    </div>
</form>
```

Readonly berguna agar nilai NPM tidak dapat diubah dan th:value berguna untuk mengisi input dengan NPM *student* yang sudah ada, th:value juga ditambahkan pada name dan gpa. Sedangkan readonly tidak perlu ditambahkan.

Halaman success-update.html

```
<html>
    <head>
        <title>Add</title>
    </head>
    <body>
        <h2>Data berhasil diupdate</h2>
    </body>
</html>
```

Method update pada *class* StudentController

```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent (npm);
    if (student != null) {
        model.addAttribute("student", student);
        return "form-update";
    }
    else {
        model.addAttribute("npm", npm);
        return "not-found";
    }

}
```

Sama seperti method delete, pada mehod update juga diperlukan validasi untuk mengecek apakah NPM yang diinputkan ada atau tidak didatabase. Jika ada maka akan mengembalikan halaman form-update.html, jika tidak ada maka akan mengembalikan halaman not-found.html.

Method updateSubmit pada class StudentController

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(@RequestParam(value = "npm", required = false) String npm,
                           @RequestParam(value = "name", required = false) String name,
                           @RequestParam(value = "gpa", required = false) double gpa) {
StudentModel student = new StudentModel(npm, name, gpa);
studentDAO.updateStudent(student);

return "success-update";
}
```

Method diatas berfungsi untuk menerima data NPM, name, dan gpa yang akan diinputkan pada halaman form-update. Data tersebut nantinya akan dimasukan kedalam objek student yang baru dibuat. Selanjutnya, akan dipanggil method updateStudent yang berisi objek student pada class StudentService. Setelah itu akan mengembalikan halaman success-update.html yang menandakan proses *update* data mahasiswa telah sukses dilakukan.

❖ **Object sebagai parameter**

Method updateSubmit pada *class* StudentController

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(@ModelAttribute StudentModel student) {
    studentDAO.updateStudent(student);

    return "success-update";
}
```

Pada method sama seperi method updateSubmit yang menggunakan @RequestParam, bedanya pada inputan yang diterima oleh method ini dalam bentuk objek student yang dikirim dari halaman form-update.htlm. Hal berikutnya yang dilakukan menambahkan ekspresi th:object=”${student}” dan th:field=”*{nama_fileld}” pada halaman form-update.html, lebih jelasnya seperti pada gambar dibawah ini.

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>

<title>Update student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

    <h1 class="page-header">Update Editorr</h1>

    <form action="/student/update/submit" th:object="${student}" method="post">
        <div>
            <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:field="*{npm}" th:value="${student.npm}"/>
        </div>
        <div>
            <label for="name">Name</label> <input type="text" name="name" th:field="*{name}" th:value="${student.name}" />
        </div>
        <div>
            <label for="gpa">GPA</label> <input type="text" name="gpa" th:field="*{gpa}" th:value="${student.gpa}"/>
        </div>

        <div>
            <button type="submit" name="action" value="save">Update</button>
        </div>
    </form>

</body>

</html>
```

Ekspresi th:object=”${student}” pada form diatas berguna untuk mendeklarasikan sebuah objek yang akan mengumpulkan data mahasiswa yang akan diupdate. Sedangkan ekspresi th:field=”*{nama_field}” digunakan untuk mengekspresikan *field* nama, npm, dan gpa yang sesuai dengan objek Student.

❖ **Pertanyaan**

1. Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required seperti jika menggunakan RequestParam? Apakah validasi diperlukan?

   Jawaban: Cara melakukan validasi input dengan menambahkan anotasi @Valid dan objek BindingResult seperti pada gambar dibawah ini

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(@Valid StudentModel student, BindingResult bindingresult) {
    if (bindingresult.hasErrors()) {
        return "gagal";
    }
    else {
    studentDAO.updateStudent(student);

    return "success-update";
    }
}
```

Pada method updateSubmit juga dibuat kondisi jika kondisi null terjadi maka akan memanggil halaman gagal.html, jika kondisi null tidak terjadi maka akan data *student* akan diupdate. Selain itu, juga diperlukan notasi @NotNull pada variable nama dan gpa pada *class* StudentMapper pada gambar dibawah ini.

```
{
    private String npm;

    @NotNull
    private String name;

    @NotNull
    private double gpa;

}
```

Validasi diperlukan untuk menghindari apabila *users* tidak menginputkan nama dan gpa pada saat melakukan *update* data mahasiswa di browser.

2. Menurut Anda, mengapa form submit biasanya menggunakan POST method dibanding GET method? Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?
   Jawaban: Karena method POST lebih aman dibandingkan dengan method GET. Pada saat *user* menginputkan data dan menekan tombol update, maka di url tidak menampilkan data apa saja yang baru diupdate. Berbeda dengan menthod GET, pada method GET data yang barusan diubah akan ditampilkan pada URL.
   Iya, perlu caranya dengan menambahkan method = RequestMethod.POST pada anotasi @RequestMapping.

3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?
   Jawaban: Bisa, tetapi saat menjalankannya harus satu per satu.