

Sum Up yang dipelajari pada Tutorial 4: Hal yang saya pelajari pada Tutorial 4 adalah beberapa hal sebagai berikut – yang pertama yaitu penggunaan database dan melakukan debugging dalam project spring boot. Saya juga belajar mengenai penggunaan Slf4j untuk melakukan debugging program menggunakan variable log. Selanjutnya saya belajar untuk mengimplementasikan suatu method dengan menggunakan database, saya juga belajar untuk menggunakan method POST untuk melakukan masukan, dan yang terakhir yaitu belajar menggunakan object sebagai parameter.

Latihan Menambahkan Delete

Pada viewall.html tambahkan

```
<a th:href="/student/delete/' +${student.npm}">Delete Data</a><br/>
```

Tambahkan method deleteStudent yang ada di class StudentMapper

- Tambahkan method delete student yang menerima parameter NPM.
- Tambahkan annotation delete di atas dan SQL untuk menghapus

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

Lengkapi method deleteStudent yang ada di class StudentServiceDatabase

- Tambahkan log untuk method tersebut dengan cara menambahkan log.info ("student " + npm + " deleted");
- Panggil method delete student yang ada di Student Mapper

```
void deleteStudent (String npm);
```

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

Lengkapi method delete pada class StudentController

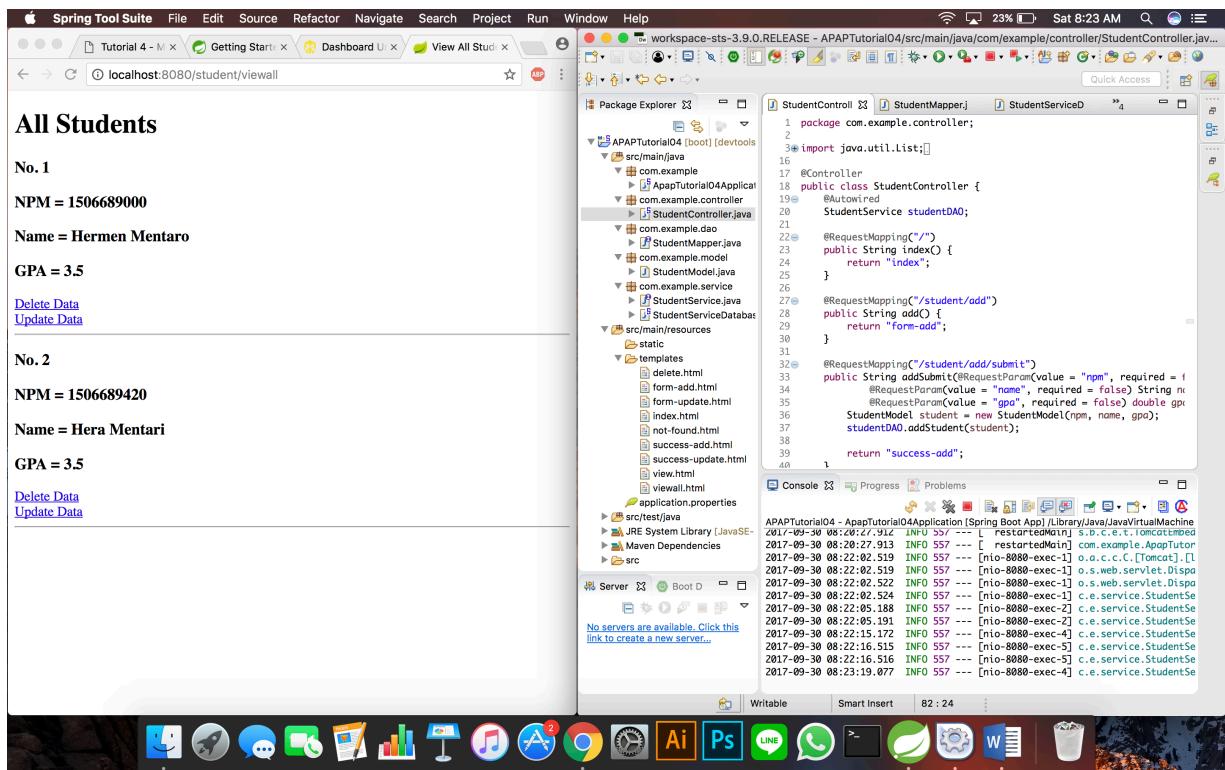
- Tambahkan validasi agar jika mahasiswa tidak ditemukan tampilkan view not-found
- Jika berhasil delete student dan tampilkan view delete

```
@RequestMapping("/student/delete/{npm}")
public String delete(Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent(npm);

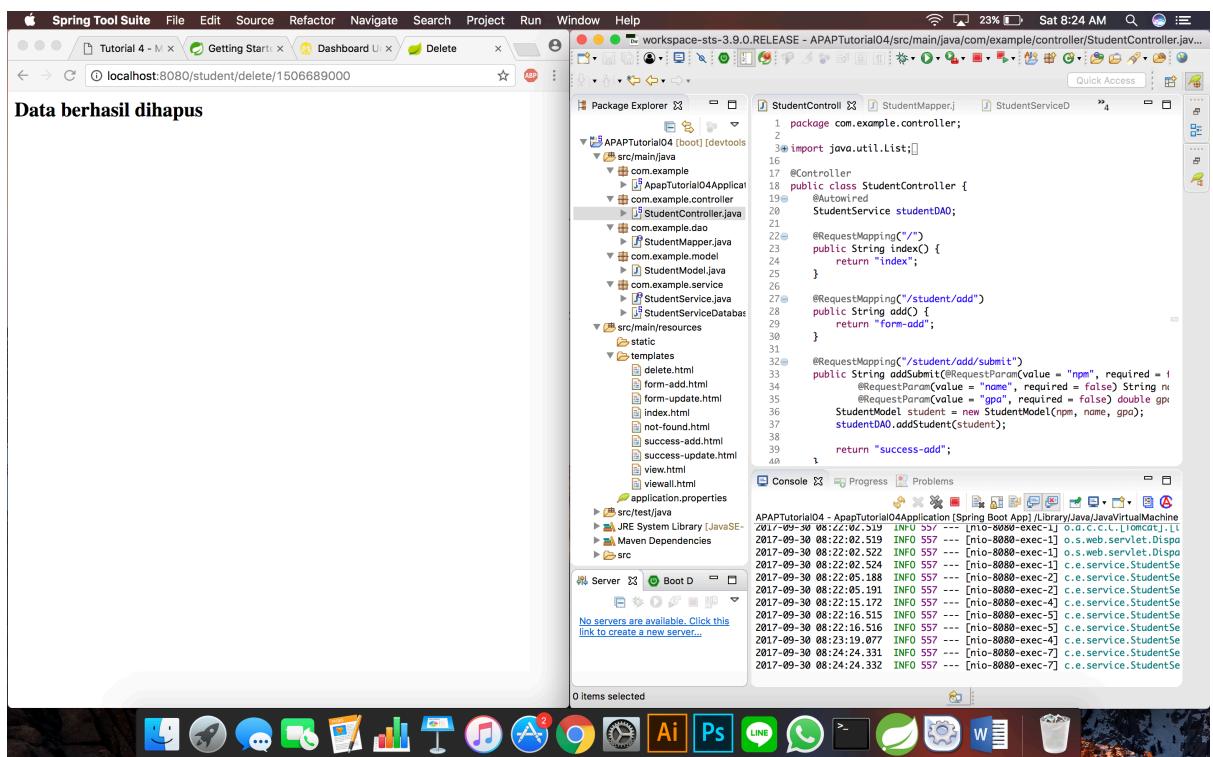
    if (student != null) {
        studentDAO.deleteStudent(npm);
        return "delete";
    } else {
        model.addAttribute("npm", npm);
        return "not-found";
    }
}
```

}

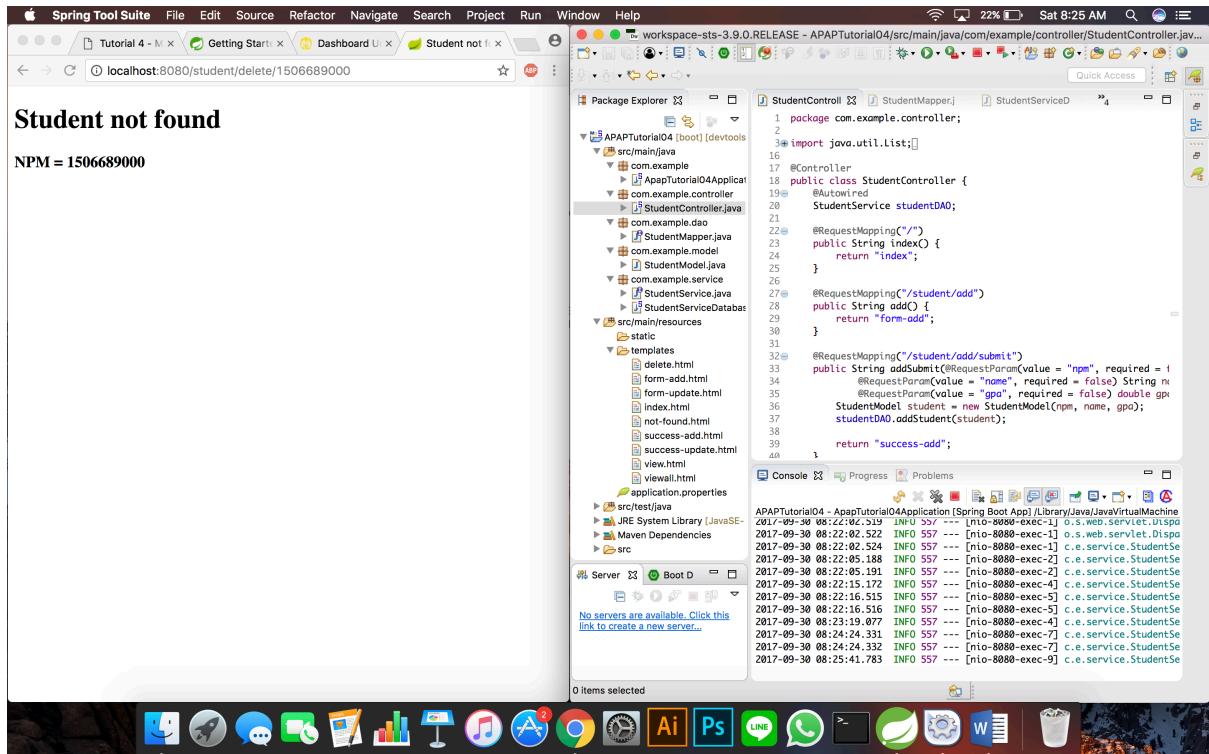
[http://localhost:8080/student/viewall:](http://localhost:8080/student/viewall)



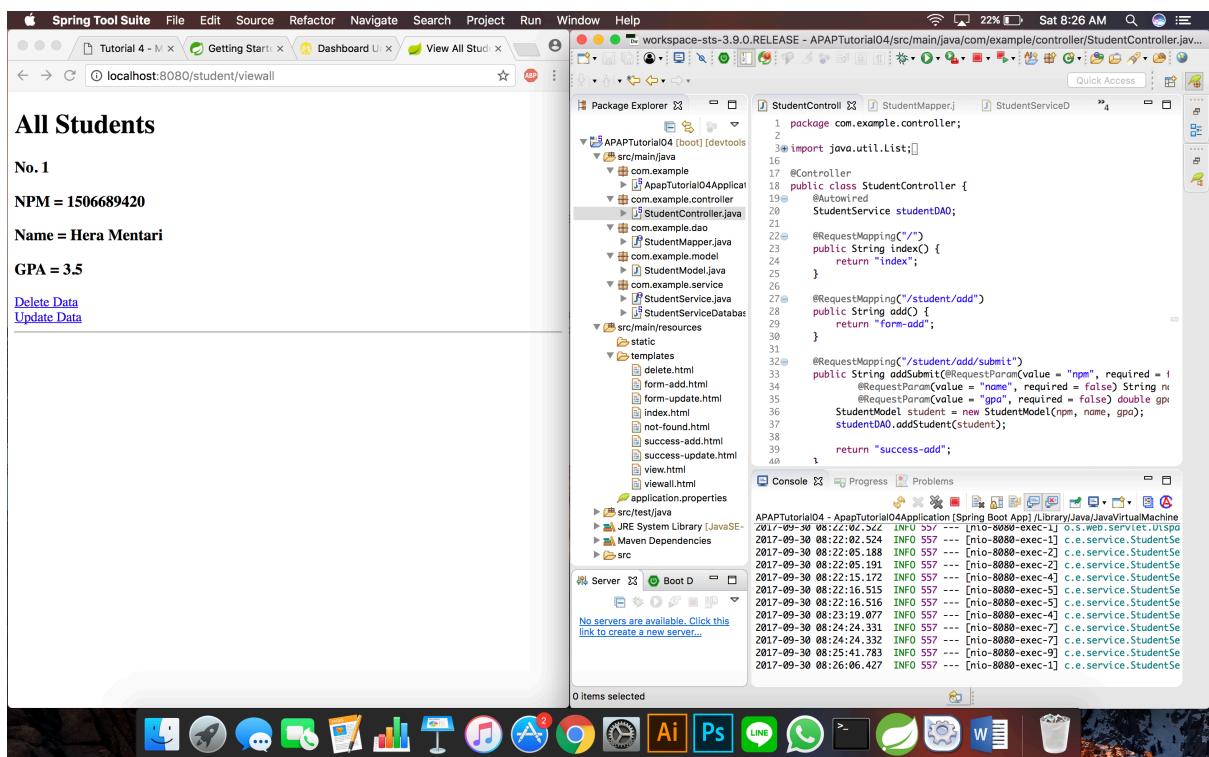
[http://localhost:8080/student/delete/1506689000:](http://localhost:8080/student/delete/1506689000)



<http://localhost:8080/student/delete/1506689000>:



<http://localhost:8080/student/viewall>:



Latihan Menambahkan Update

Tambahkan method updateStudent pada class StudentMapper

- a. Parameternya adalah StudentModel student
- b. Annotationnya adalah @Update
- c. Lengkapi SQL update-nya Hint: Query SQL untuk update

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

Tambahkan method updateStudent pada interface StudentService

```
void updateStudent (StudentModel student);
```

Tambahkan implementasi method updateStudent pada class StudentServiceDatabase. Jangan lupa tambahkan logging pada method ini.

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

Tambahkan link Update Data pada viewall.html

```
<a th:href="/student/update/' +${student.npm}" >Update Data</a><br/>
```

Copy view form-add.html menjadi form-update.html.

- a. Ubah info-info yang diperlukan seperti title, page-header, tombol menjadi update dll.
- b. Ubah action form menjadi /student/update/submit
- c. Ubah method menjadi post
- d. Untuk input npm ubah menjadi sebagai berikut. readonly agar npm tidak dapat diubah, th:value digunakan untuk mengisi input dengan npm student yang sudah ada. Input name ubah menjadi sebagai berikut. Lakukan hal yang sama untuk GPA.

```
<!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">Problem Editor</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">Save</button>
    </div>
</form>
</body>
</html>
```

Copy view success-add.html menjadi success-update.html.

- Ubah keterangan seperlunya

```

<html>
    <head>
        <title>Update</title>
    </head>
    <body>
        <h2>Data berhasil diubah</h2>
    </body>
</html>
```

Tambahkan method update pada class StudentController

- Request mapping ke /student/update/{npm}
- Sama halnya seperti delete, lakukan validasi.
- Jika student dengan npm tidak ada tampilkan view not-found, jika ada tampilkan view form-update

```

@RequestMapping("/student/update/{npm}")
public String update(Model model, @PathVariable(value = "npm") String npm) {
    StudentModel student = studentDAO.selectStudent(npm);

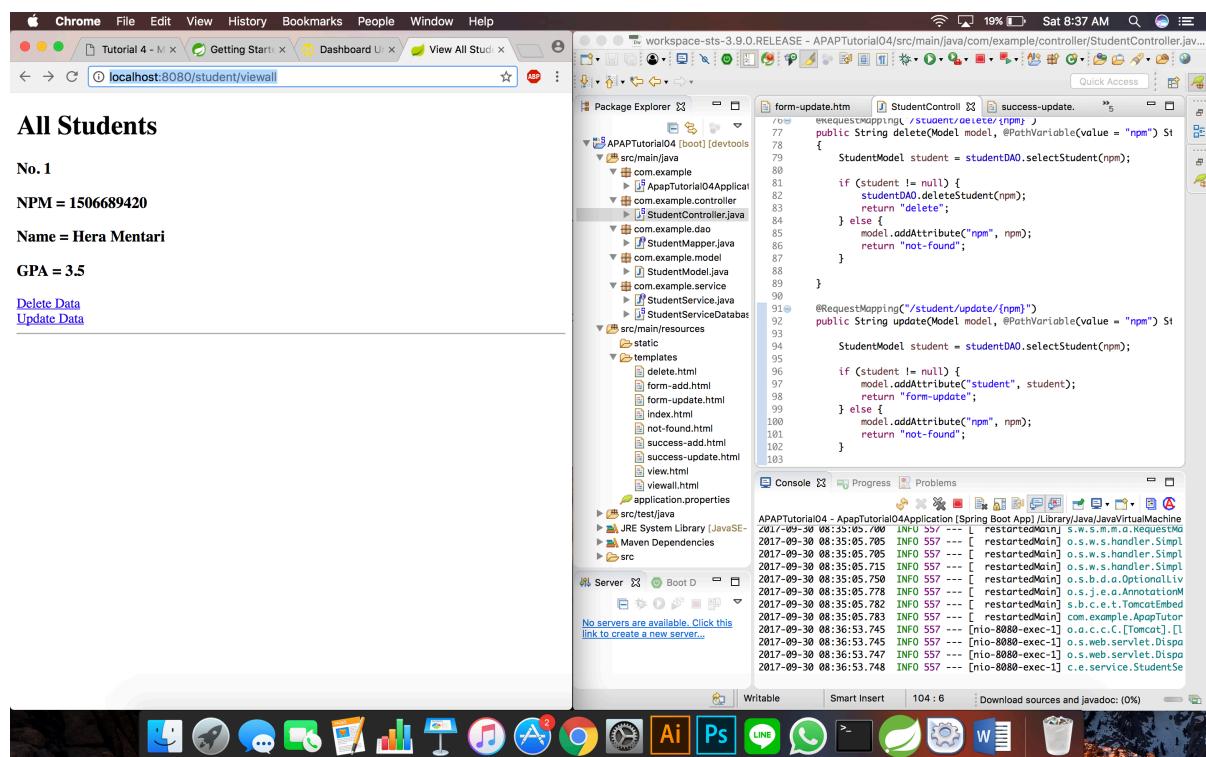
    if (student != null) {
        model.addAttribute("student", student);
        return "form-update";
    } else {
        model.addAttribute("npm", npm);
        return "not-found";
    }
}
```

Tambahkan method updateSubmit pada class StudentController

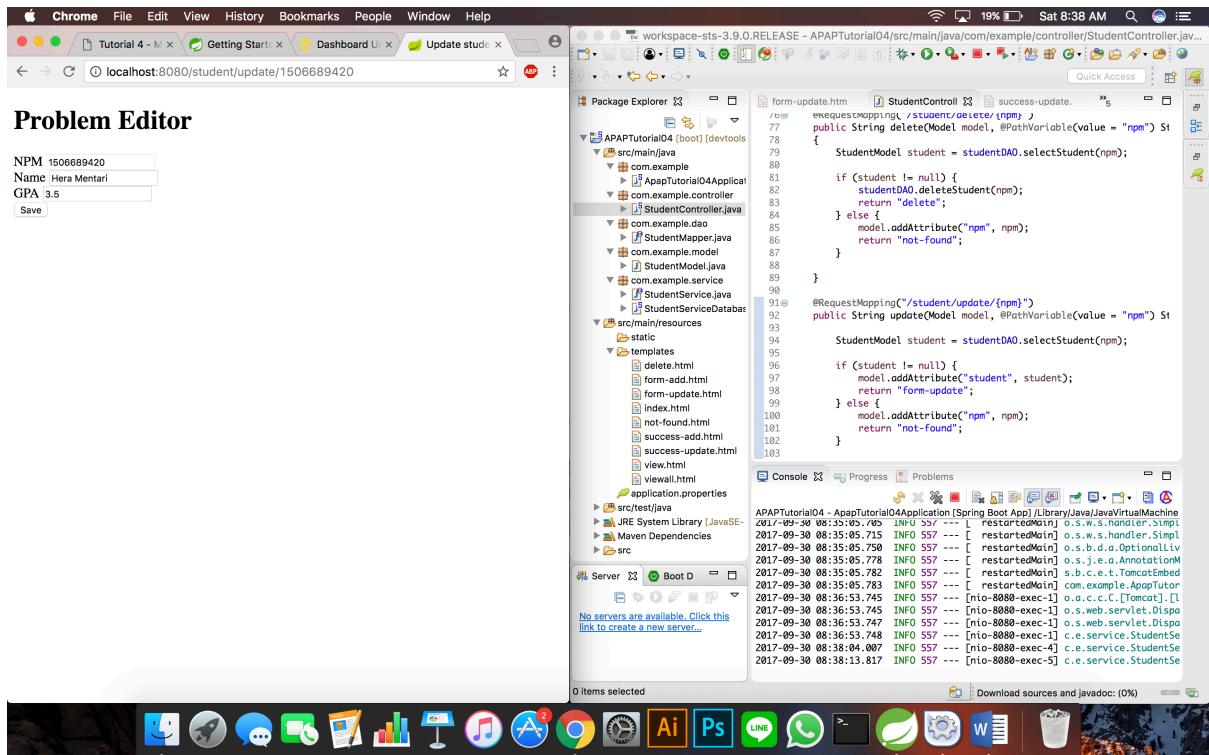
- a. Karena menggunakan post method maka request mappingnya adalah sebagai berikut:
Tutorial 04 – Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot | 5 @RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
- b. Header methodnya adalah sebagai berikut: public String updateSubmit (@RequestParam(value = "npm", required = false) String npm, @RequestParam(value = "name", required = false) String name, @RequestParam(value = "gpa", required = false) double gpa)
- c. Lengkapi method dengan memanggil method update Student dan kembalikan view success-update

```
@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 = studentDAO.selectStudent(npm);
    student.setName(name);
    student.setGpa(gpa);
    studentDAO.updateStudent(student);
    return "success-update";
}
```

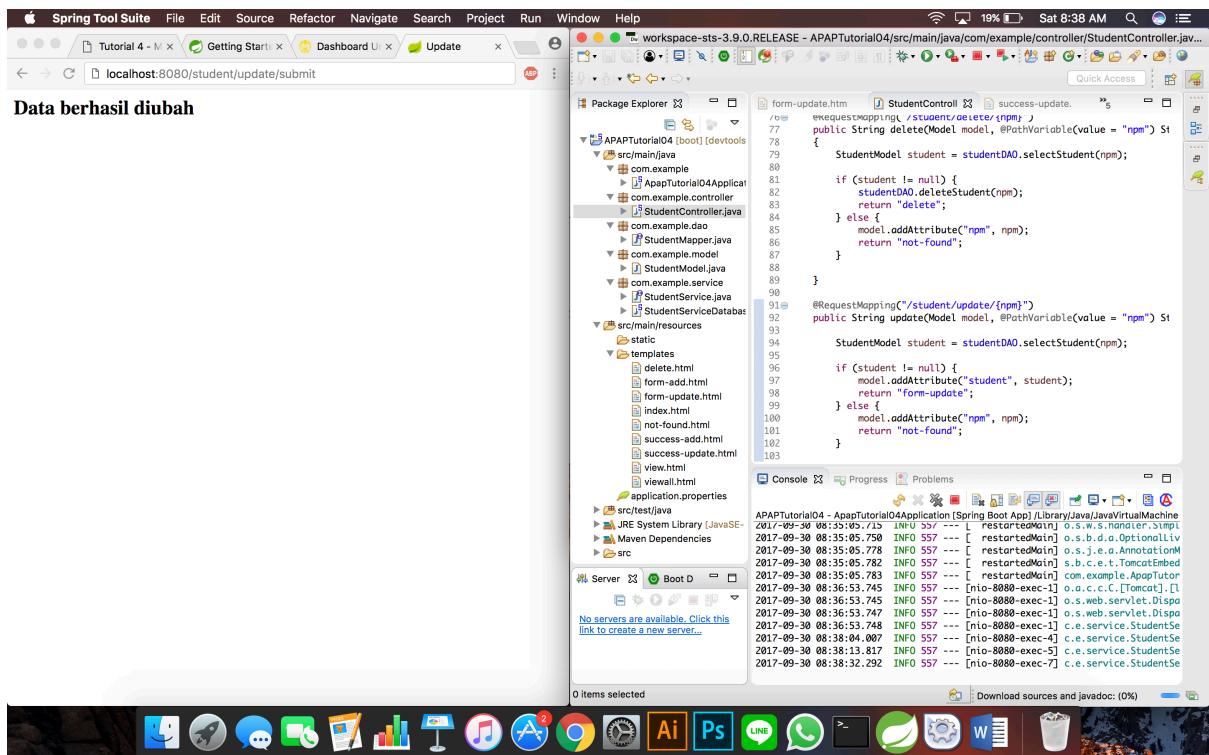
[http://localhost:8080/student/viewall:](http://localhost:8080/student/viewall)



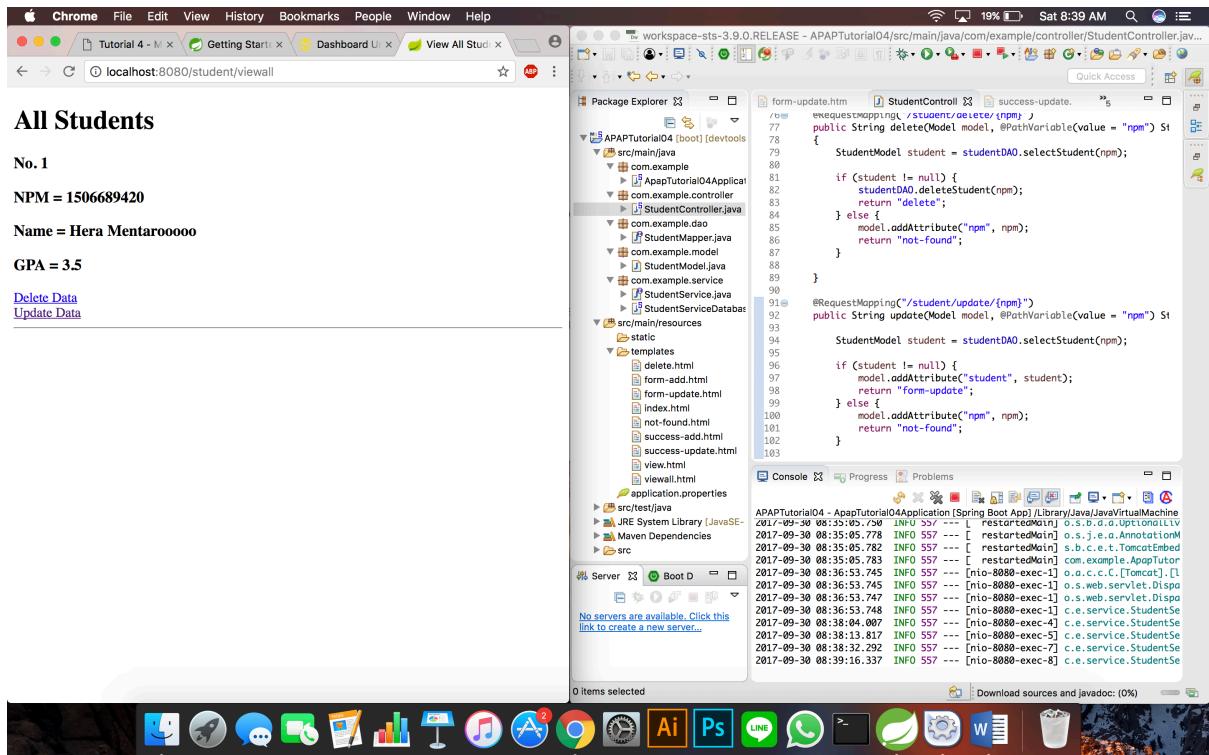
[http://localhost:8080/student/update/1506689420:](http://localhost:8080/student/update/1506689420)



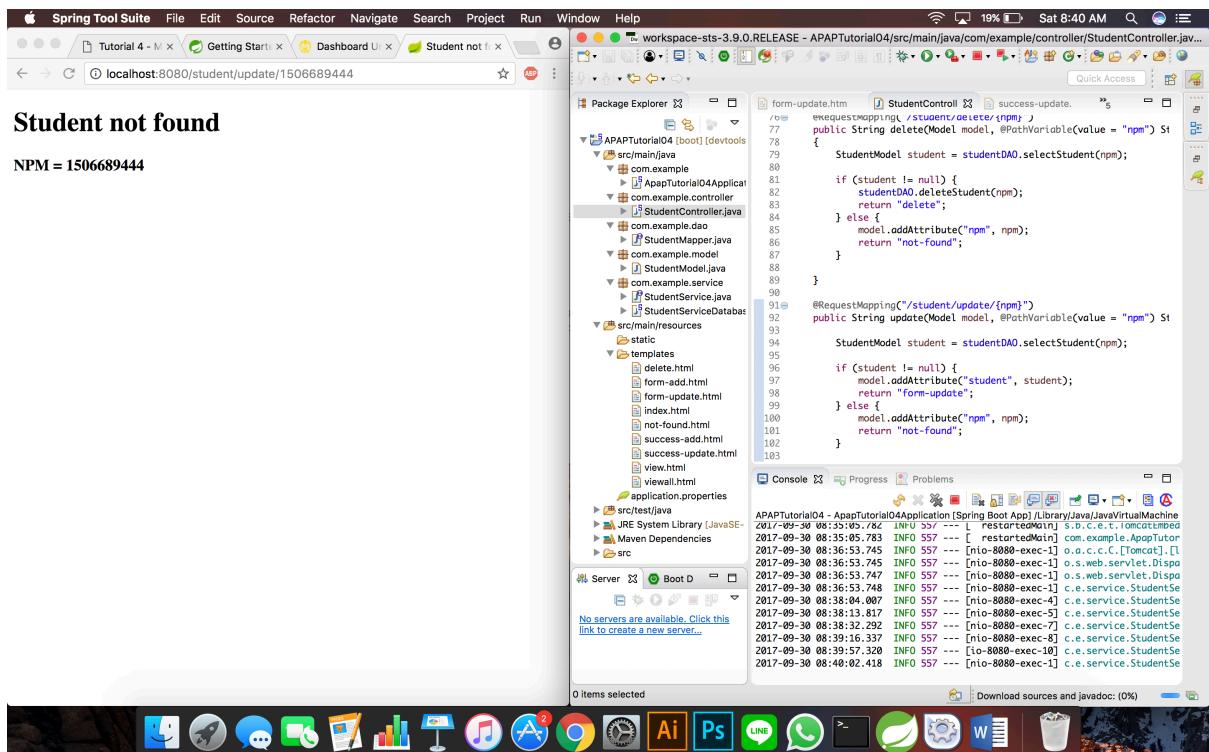
[http://localhost:8080/student/update/submit:](http://localhost:8080/student/update/submit)



[http://localhost:8080/student/viewall:](http://localhost:8080/student/viewall)



[http://localhost:8080/student/update/1506689444:](http://localhost:8080/student/update/1506689444)



Latihan Menggunakan Object Sebagai Parameter

- a. Pada tutorial di atas Anda masih menggunakan RequestParam untuk menghandle form submit. Sehingga ada banyak parameter pada method Anda. Bayangkan jika Anda memiliki form yang memiliki banyak field, maka parameternya akan sangat banyak dan tidak rapih.
- b. SpringBoot dan Thymeleaf memungkinkan agar method updateSubmit menerima parameter berupa model StudentModel. Metode ini lebih disarankan dibandingkan menggunakan RequestParam.
- c. Cara lengkapnya silakan ikuti pada link Handling Form berikut: (<https://spring.io/guides/gs/handling-form-submission/>)
- d. Tahapannya kurang lebih sebagai berikut: Menambahkan th:object="\${student}" pada tag <form> di view. Menambahkan th:field="*[\${nama_field}]" pada setiap input. Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel. Tes lagi aplikasi Anda.

```
<!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">Problem Editor</h1>

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

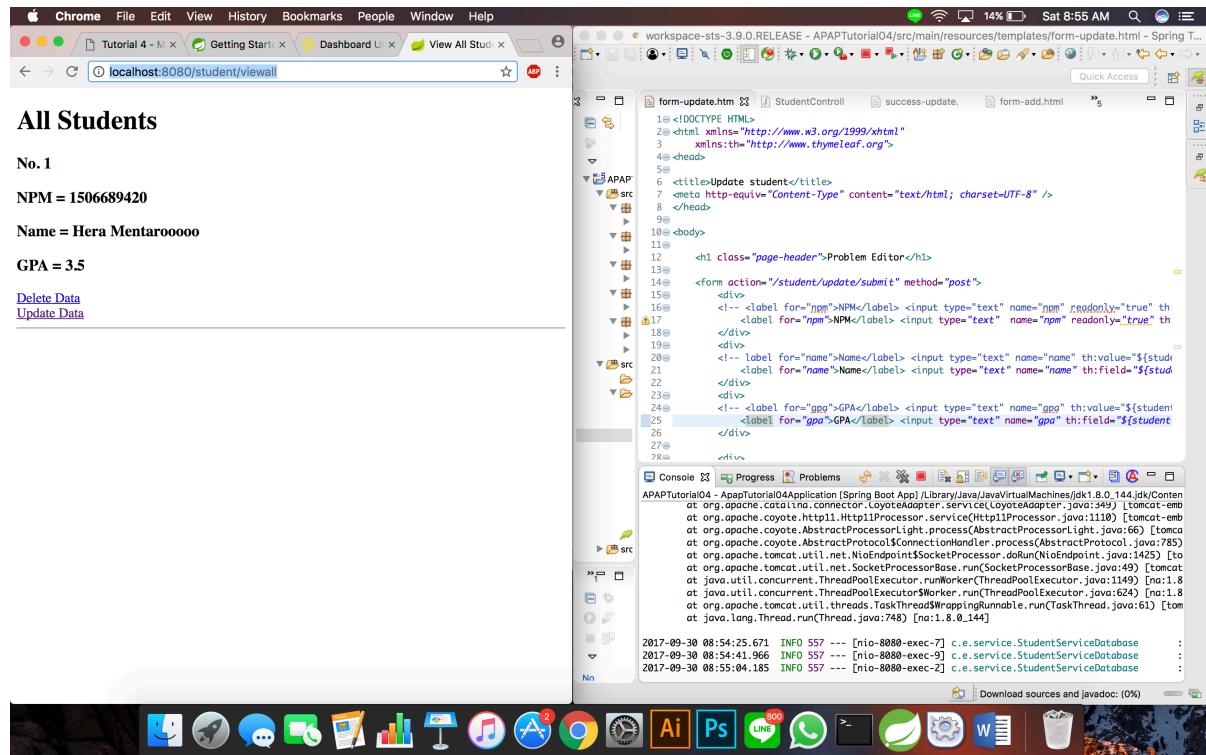
</body>
</html>
```

```

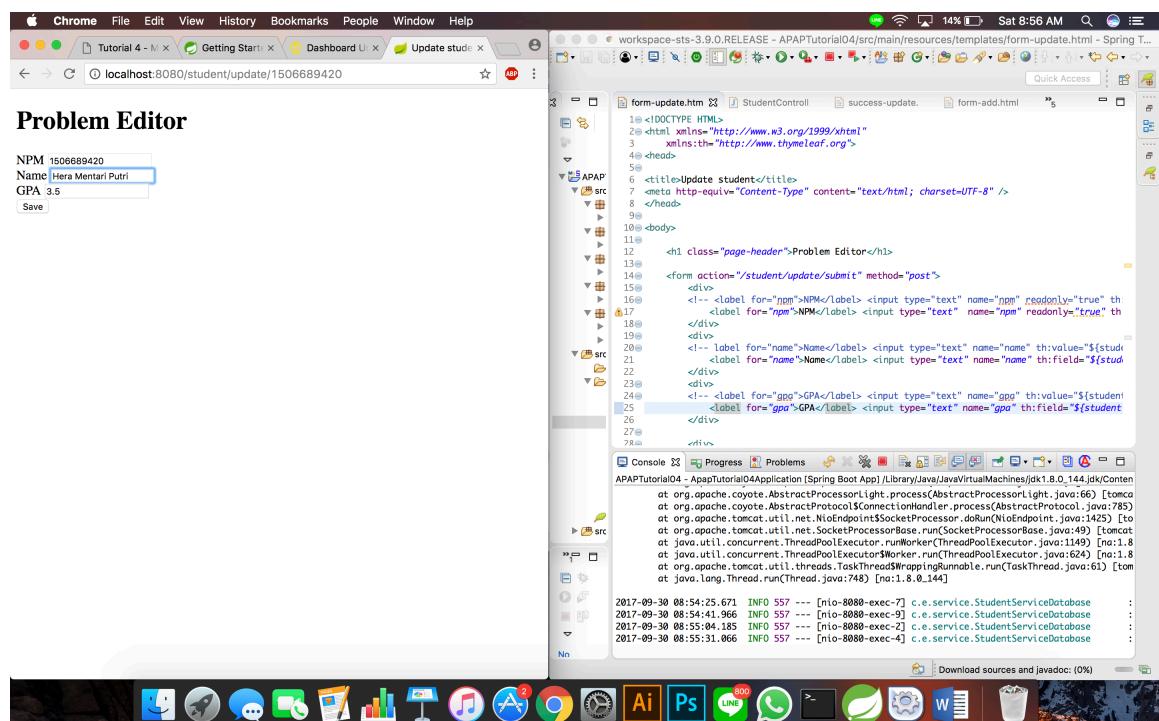
    @RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
    public String updateSubmit(@ModelAttribute StudentModel student) {
        studentDAO.updateStudent(student);
        return "success-update";
    }

```

[http://localhost:8080/student/viewall:](http://localhost:8080/student/viewall)



[http://localhost:8080/student/update/1506689420:](http://localhost:8080/student/update/1506689420)



[http://localhost:8080/student/update/submit:](http://localhost:8080/student/update/submit)

The screenshot shows a Java IDE (IntelliJ IDEA) and a web browser. The browser window displays the message "Data berhasil diubah" (Data has been changed). The IDE shows the code for the "form-update.html" template, which contains fields for NPM, Name, and GPA. The code is as follows:

```
1<!DOCTYPE HTML>
2<html xmlns="http://www.w3.org/1999/xhtml"
3    xmlns:th="http://www.thymeleaf.org">
4<head>
5</head>
6<body>
7    <h1 class="page-header">Problem Editor</h1>
8    <form action="/student/update/submit" method="post">
9        <div>
10            <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:>
11                <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:>
12            </div>
13            <div>
14                <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" th:field="${student.name}">
15                <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" th:field="${student.name}">
16            </div>
17            <div>
18                <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}" th:field="${student.gpa}">
19                <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}" th:field="${student.gpa}">
20            </div>
21        </div>
22    </div>
23    </div>
24    <div>
25        <button type="submit" value="Update Student" th:disabled="true">Update Student</button>
26    </div>
27</div>
```

The IDE's terminal window shows log entries from the Spring Boot application, indicating successful database operations. The system tray at the bottom of the screen shows various application icons.

[http://localhost:8080/student/viewall:](http://localhost:8080/student/viewall)

The screenshot shows a Java IDE and a web browser. The browser window displays student details: No. 1, NPM = 1506689420, Name = Hera Mentari Putri, and GPA = 3.5. The IDE shows the code for the "form-update.html" template, which contains fields for NPM, Name, and GPA. The code is identical to the previous screenshot. The IDE's terminal window shows log entries from the Spring Boot application, indicating successful database operations. The system tray at the bottom of the screen shows various application icons.

Pertanyaan

Beberapa pertanyaan yang perlu Anda jawab:

1. Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required? Apakah validasi diperlukan? Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.

Jawab: Melakukan validasi dengan menambahkan anotasi pada model objek. Contohnya seperti anotasi @NotNull digunakan untuk mencegah form yang kosong untuk dikirim. Namun, dalam beberapa kasus tentu tetap membutuhkan validasi.

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?

Jawab: Biasanya menggunakan POST karena method tersebut lebih aman (Safe Method) dapat dikatakan demikian karena method tersebut tidak dapat terlihat pada address bar, sedangkan pada method GET dapat dilihat pada address bar. Untuk menjalankan method POST, dibutuhkan request method POST di header controller.

3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?

Iya, bisa.