- Delete

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

    List<StudentModel> selectAllStudents ();

    void addStudent (StudentModel student);

    void deleteStudent(String npm);
```

Pertama-tama buatlah method deleteStudent pada StudentService

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

Selanjutnya buatlah methpd deleteStudent dan lengkapi query sql delete pada StudentMapper

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

Kemudian impelementasikan method deleteStudent di kelas StudentServiceDatabase

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

Kemudian buatlah method controller untuk melakukan delete. Dimana nantinya method akan menerima parameter npm yang selanjutnya akan dilakukan pengecekan apakah terdapat siswa dengan npm tersebut. Jika ada maka akan dilakukan penghapusan

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

- 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( attributeName: "student", student);
        return "form-update";
    }
    return "not-found";
}

@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(Model model, @Valid StudentModel student, BindingResult bindingResult)
{
    if (bindingResult.hasErrors())
    {
        model.addAttribute( attributeName: "student", studentDAO.selectStudent(student.getNpm()));
        return "form-update";
    }
    studentDAO.updateStudent(student);
    return "success-update";
}
```

Method ini nantinya akan melakukan return kepada form untuk melakukan update dimana jika sudah di submit maka akan menjalankan method updateSubmit untuk melakukan pembaruan data student.

- Object

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

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

Pertama tama pada html form-update kita menambahkan th:object dan th:field sehingga nantinya jika form di submit akan dibuat dalam bentuk object untuk mempermudah pada controller

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(Model model, @Valid StudentModel student, BindingResult bindingResult)
{
    if (bindingResult.hasErrors())
    {
        model.addAttribute( attributeName: "student", studentDAO.selectStudent(student.getNpm()));
        return "form-update";
    }
    studentDAO.updateStudent(student);
    return "success-update";
}
```

Selanjutnya pada method controller updateSubmit, method akan melakukan pembaruan *student* tanpa perlu membuat object student terlebih dahulu

Question

1.

Perlu dilakukan handle di dalam controller. Dimana nantinya akan di check apakah semua atribut yg required pada suatu object memiliki nilai atau tidak ketika form di submit. Jika terdapat atribut yg tidak memiliki nilai, maka method di dalam controller akan mengembalikan ke halam pengisian form.

2.
Karena GET method memilili tingkat keaman yang rendah sehingga rawan dibobol oleh seorang hacker. Sehingga method GET biasanya hanya digunakan untuk Read atau View. Sedangkan submit form sendiri merupakan proses Input/Update sehingga untuk menjaga keamanan data maka digunakan method POST.
Serta perlu dilakukan penyesuaian pada header atau body method controller sesuai dengan method yg digunakan.

3.
Tidak memungkinkan. Kita hanya dapat menggunakan satu request method pada setiap methodnya.