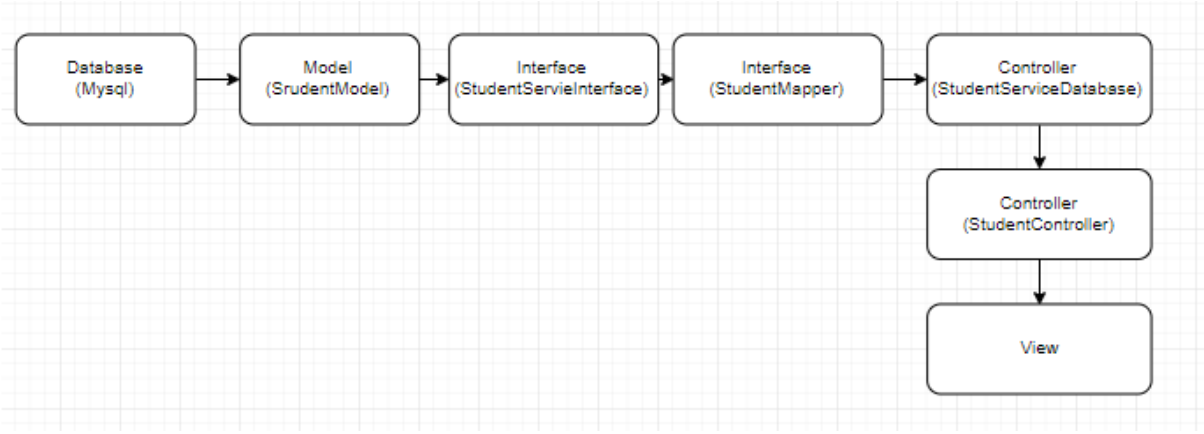
Ringkasan: Pada tutorial kali ini kita belajar mengenai program sederhana untuk menampilkan, menambahkan, memodifikasi dan menghapus data yang disimpan di basis data.



Gambar diatas menerangkan gambaran umum dari projek yang dikerjakan pada tutorial 4 ini. Untuk bertransaksi dengan database mysql kita menggunakan file **resources/application.properties.**

```
# Data Source
spring.datasource.platform=mysql
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/eeap
spring.datasource.username=root
spring.datasource.password=mysql
spring.datasource.initialize=false
server.port=8090
```

Secara umum file ini berfungsi sebagai informasi yang dibutuhkan untuk dapat terkoneksi dengan database. Informasi yang diperlukan seperti apa platform dari database, serta informasi dari nama, username dan password dari database.

```
package com.example.model;

import ...

@Data
@AllArgsConstructor
@NoArgsConstructor
public class StudentModel
{
    private String npm;
    private String name;
    private double gpa;

}
```

Untuk dapat berkomunikasi dengan database diperlukan model. Model com.example.model/**StudentModel** berfungsi sebagai kelas yang mengatur data.

```java
package com.example.service;

import ...

public interface StudentService
{
    StudentModel selectStudent (String npm);


    List<StudentModel> selectAllStudents ();


    void addStudent (StudentModel student);


    void deleteStudent (String npm);

    void updateStudent(String npm, String name, double gpa);
}
```

Interface berfungsi sebagai template bagi controller. Interface merupakan “aturan” yang harus dipatuhi oleh controller yang mengimplementasikannya.

```java
package com.example.dao;

import ...

@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(@Param("npm") String npm, @Param("name") String name, @Param("gpa") double gpa);
```

Sama dengan interface sebelumnya, interface ini menambahkan fungsi ddl dan dml dari database. Fungsi nya adalah melakukan operasi terhadap data.

```java
package com.example.service;

import ...

@Slf4j
@Service
public class StudentServiceDatabase implements StudentService
{
    @Autowired
    private StudentMapper studentMapper;


    @Override
    public StudentModel selectStudent (String npm)
    {
        log.info ("select student with npm {}", npm);
        return studentMapper.selectStudent (npm);
    }
```

Controller ini lebih berfungsi eksekutor dari fungsi yang telah dibuat sebelumnya.

```java
@RequestMapping("/student/add")
public String add () { return "form-add"; }


@RequestMapping("/student/add/submit")
public String addSubmit (
        @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.addStudent (student);

    return "success-add";
}
```

StudentController selain berperan sebagai controller yang mengarahkan request yang masuk. Juga berperan untuk melakukan validasi dari data yang masuk ke request.

Jawaban Pertanyaan:

1. Method apa saja yang Anda tambahkan pada latihan menambahkan delete, jelaskan
   a. Langkah pertama menambahkan baris kode pada StudentService

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

   b. Lalu menambahkan pada StudentServiceDatabase yang meng-override fungsi delete pada interface StudentService.

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

   Kenapa harus di-override?, karena keterbatasan interface. Interface tidak dapat mengimplementasikan fungsi tertentu. Maka kita membuat Controller yang mengimplementasikan interface dari StudentService.

   c. Menambahkan baris kode pada Interface StudentMapper untuk menghapus data di database.

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

   d. Lalu tambahkan baris kode di controller

```
@Autowired
StudentService studentDAO;

@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( s: "npm", npm);
        return "not-found";

    }
}
```

   Setelah sebelumnya kita membuat service untuk menghapus data. Pada tahapan ini kita menggunakan service tersebut. Pada method delete ini kita melakukan pengecekan terhadap data dari student. Apabila data tersedia di database maka kita akan langsung melakukan penghapusan data tersebut. Jika tidak ada maka akan dikembalikan pesan error.

2. Method apa saja yang Anda tambahkan pada latihan menambahkan update, jelaskan
   a. Langkah pertama menambahkan baris kode pada StudentService.

```
void updateStudent(String npm, String name, double gpa);
```

   b. tambahkan data di StudentServiceDatabase yang meng-override fungsi delete pada interface StudentService.

```
@Override
public void updateStudent(String npm, String nama, double gpa)
{
    log.info("Student " + npm + "updated");
    studentMapper.updateStudent(npm, nama, gpa);
}
```

c. Menambahkan baris kode pada Interface StudentMapper untuk menghapus data di database.

```
@Update("Update student set name = #{name}, gpa = #{gpa} where npm = #{npm}")
void updateStudent(@Param("npm") String npm, @Param("name") String name, @Param("gpa") double gpa);
```

d. Lalu tambahkan baris kode di controller

```
@RequestMapping("/student/update/{npm}")
public String update (Model model,
                      @PathVariable(value = "npm") String npm)
{

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

method ini berfunsi untuk melakukan pengecekan terhadap npm dari mahasiswa. Apabila npm nya ada maka akan langsung diarahkan ke form-update, jika kosong maka akan tampil halaman error.

e. Tambahkan di view

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

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

field berfungsi untuk menangkap data yang dilempar dari database. Action /student/update/submit merupakan fungsi yang akan dipanggil apabila tombol submit ditekan.

f. Buatlah fungsi untuk memproses perubahan data di controller

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (@ModelAttribute("student") StudentModel student, ModelMap model)
{
    if(student.getName() == null || student.getName() == "") {
        return "error";
    }

    StudentModel studentValid = studentDAO.selectStudent(student.getNpm());
    if (studentValid != null) {
        studentDAO.updateStudent(student.getNpm(), student.getName(), student.getGpa());
        return "success-update";
    }else {
        model.addAttribute ( attributeName: "npm", student.getNpm());
        return "not-found";
    }
}
```

Fungsi ini akan memanggil service update yang telah dibuat sebelumnya. Dan jika npm nya kosong atau salah maka akan mengembalikan halaman error.

3. Method apa saja yang Anda tambahkan pada latihan menambahkan object sebagai parameter, jelaskan

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (@ModelAttribute("student") StudentModel student, ModelMap model)
{
    if(student.getName() == null || student.getName() == "") {
        return "error";
    }

    StudentModel studentValid = studentDAO.selectStudent(student.getNpm());
    if (studentValid != null) {
        studentDAO.updateStudent(student.getNpm(), student.getName(), student.getGpa());
        return "success-update";
    }else {
        model.addAttribute ( attributeName: "npm", student.getNpm());
        return "not-found";
    }
}
```

Pada latihan menambahkan object sebagai parameter kita mengganti parameter RequestParameter yang sebelumnya digunakan pada latihan menjadi StudentModel. Cara ini dirasa lebih praktis dibanding dengan cara sebelumnya.