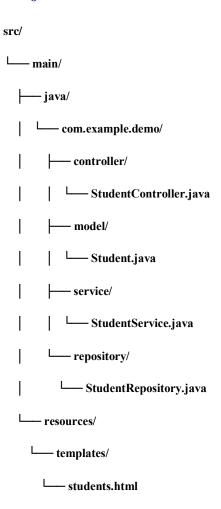
Practical Work 4: Understanding the flaw: Services, Repositories, Models, and Controllers in Spring Boot

Objective:

- Create a Spring Boot application that demonstrates the use of **Services**, **Repositories**, **Models**, and **Controllers**.
- Implement CRUD operations to simulate interaction with a database.

Extend the previous Spring Boot application (PW03) by adding a Service layer that handles business logic, and a Repository that manages student data (without persistence).

Project Structure:



Implementation:

1. Model Class (Student.java), Same as the previous example.

```
public class Student {
```

```
name; email; age; + the constructor + Getters and Setters \}
```

2. Repository Class (StudentRepository.java)

This class will simulate a repository with an in-memory list.

- 1. In the src/main/java/MySecondProject directory, create a new folder "Reposiroty".
- 2. Create a new file class "StudentRepository.java".
- 3. Use @Repository. For defining this class as a repository class.
- 4. Create a list of some Students:

```
private List<Student> students = new ArrayList<>();
```

5. Add some initial students,

```
public StudentRepository() {
    students.add(new Student("mohamed", "med@example.com", 25));
    .
    .
    .
    .
}
```

6. Add this method to retrieve all the existed students:

```
public List<Student> findAll() {
    return students;
}
```

7. We want Add a new student, like this:

```
public void addStudent(Student student) {
   students.add(student);
}
```

3. Service Class (StudentService.java)

This class contains the business logic and interacts with the repository.

1. In the src/main/java/MySecondProject directory, create a new folder "Service".

- 2. Create a new file class "StudentService.java".
- 3. Use @Service. For defining this class as a service class.
- 4. In this case, this class **retrieves** and **adds students** using the repository:

Add this code to use the repository class:

• private final **StudentRepository** studentRepository;

```
public StudentService(StudentRepository studentRepository) {
this.studentRepository = studentRepository;
}
```

• Retrieve all students:

```
public List<Student> getAllStudents() {
return studentRepository.findAll();
```

• Add a new student:

```
public void addStudent(Student student) {
studentRepository.addStudent(student);
```

4. Controller Class (StudentController.java)

The controller handles incoming requests and interacts with the service layer.

1. For delegating the logic to the service, inside the class add:

```
private final StudentService studentService;

public StudentController(StudentService studentService) {
   this.studentService = studentService;
   }
```

2. Inside the Getmethod, Fetch all students using the service:

```
List<Student> students = studentService.getAllStudents(); model.addAttribute("students", students);
```

- 3. Return the view "name"
- 4. Add this method: @PostMapping("/students/add")

public String addStudent(@RequestParam String name, @RequestParam String email,
@RequestParam int age) {}

5. Create a new student and add it via the service

```
Student student = new Student(name, email, age);
studentService.addStudent(student);
```

6. Rendering the view by redirecting back to the student list page, to see the added student:

```
return "redirect:/students";
```

5. View Template (students.html)

This view allows the user to see the list of students and also add a new student.

- 1. The first part is the same
- 2. Add another tilte: Add a New Student
- 3. Add a new form containing 03 text fields and a submit bottom

```
<form action="/students/add" method="post">
Name: <input type="text" name="name"><br>
Email:
Age:
<input type="submit" value="Add Student">
</form>
```

6. Running the Project:

- 1. Run the Spring Boot application.
- 2. Open http://localhost:8080/students to view the list of students.
- 3. Add a new student using the form at the bottom of the page.