REPORT EXERICSE LAB 5

Student Name: Nguyễn Tiến Anh

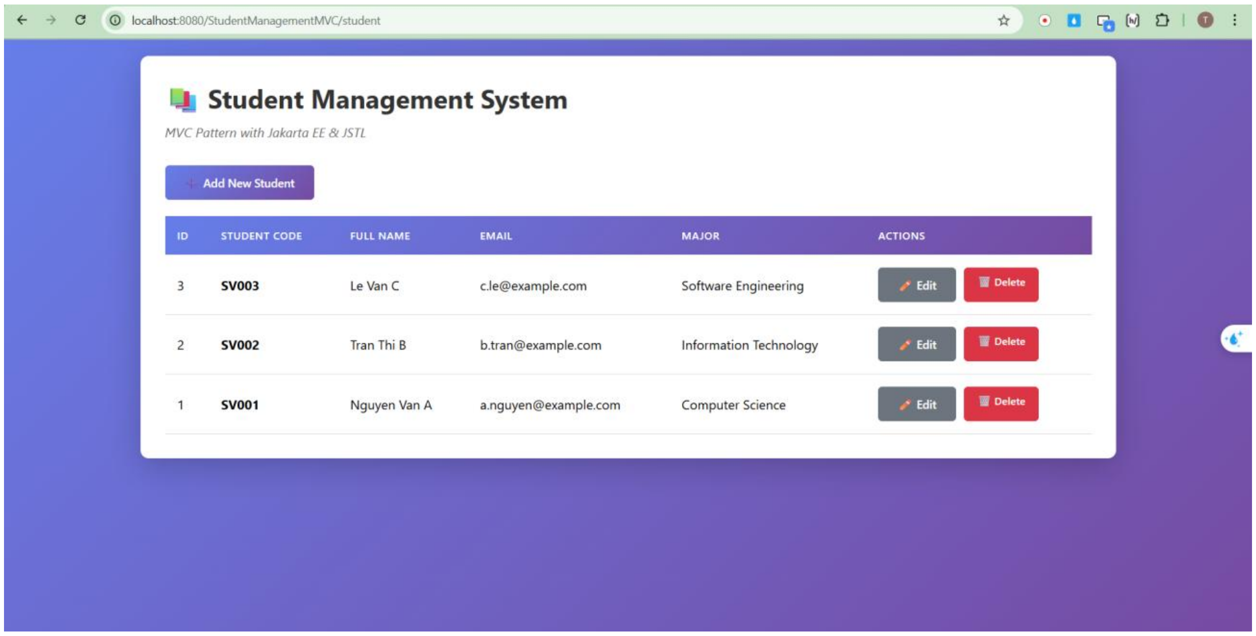
Student ID: ITITDK22128

Course: Web Application Development

Lab 5: SERVLET & MVC PATTERN

Github: https://github.com/TienAnh0108/Lab5_Exercise.git

1. List



Student Management System

MVC Pattern with Jakarta EE & JSTL

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
3	**SV003**	Le Van C	c.le@example.com	Software Engineering	Edit Delete
2	**SV002**	Tran Thi B	b.tran@example.com	Information Technology	Edit Delete
1	**SV001**	Nguyen Van A	a.nguyen@example.com	Computer Science	Edit Delete

- Firstly, the WebServlet("/student") annotation on your **StudentController.java** file tells the Tomcat server that any request ending in /student must be routed to this specific Servlet for processing.
- Secondly, the browser sends an HTTP GET request to the Controller. Inside the StudentController's doGet() method: Since your URL does not contain a ?action=... parameter, the action variable is initially null, the Controller executes the if block, setting the default action to "list".

```java
String action = request.getParameter("action");

if (action == null) {
    action = "list";
}
```

- The switch statement handles the request, directing it to the listStudents(request, response); private method.

```java
switch (action) {
    case "new":
        showNewForm(request, response);
        break;
    case "edit":
        showEditForm(request, response);
        break;
    case "delete":
        deleteStudent(request, response);
        break;
    default:
        listStudents(request, response);
        break;
}
```

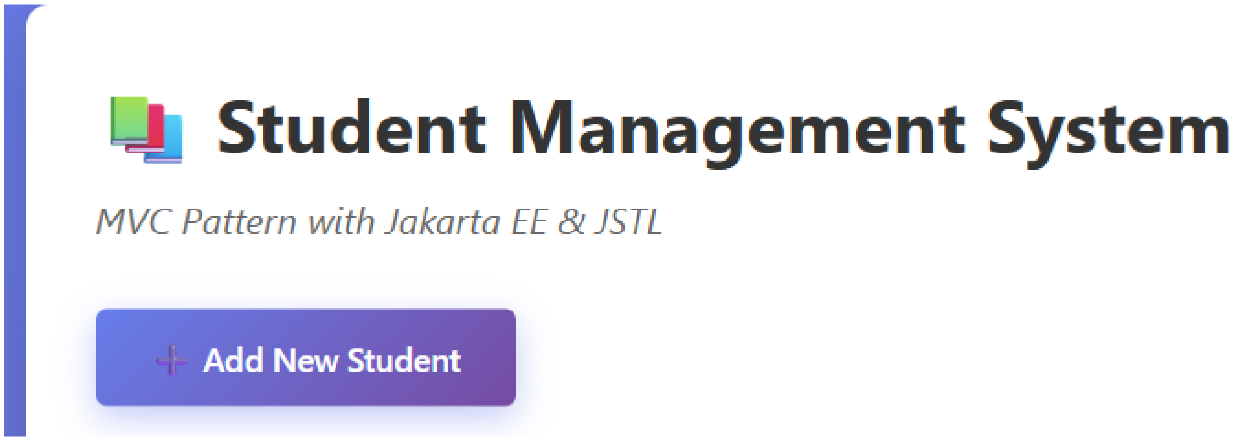
- Thirdly, the listStudents() method initiates the Model/DAO logic, the Controller retrieves the student list from the database and attaches it to the **request object** under the key **"students"**.

```java
List<Student> students = studentDAO.getAllStudents();
request.setAttribute("students", students);
```

- Lastly, the Controller then forwards the request (which now contains the student data) to View. After that, the **student-list.jsp** file receives the request object. It uses JSTL (or scriptlets) to iterate over the student list attached under the "students" key and dynamically generates the HTML table.

```java
RequestDispatcher dispatcher = request.getRequestDispatcher("/views/student-list.jsp");
dispatcher.forward(request, response);
```

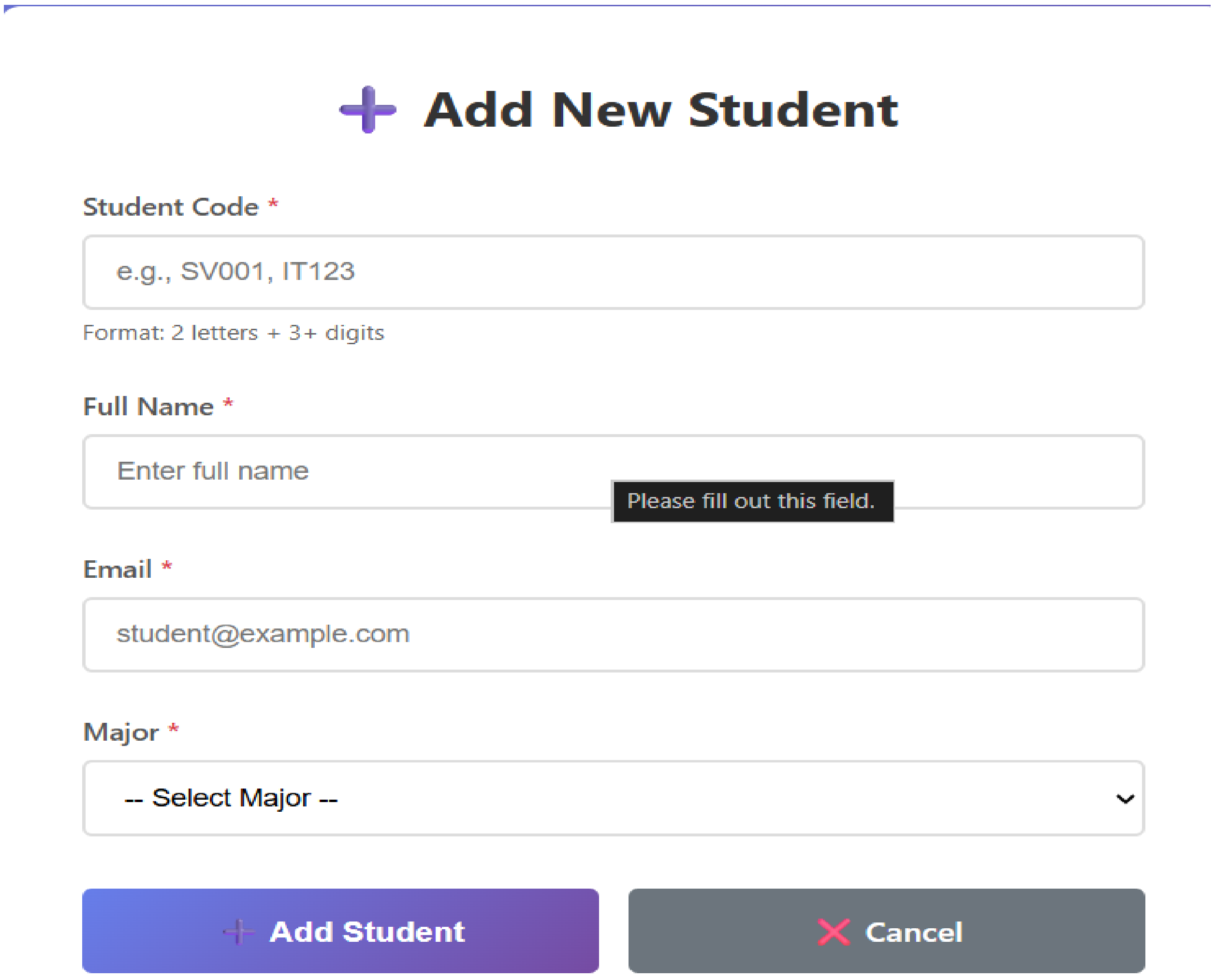
2. **Add**



Student Management System

MVC Pattern with Jakarta EE & JSTL

After click:



Add New Student

Student Code *

e.g., SV001, IT123

Format: 2 letters + 3+ digits

Full Name *

Enter full name

Please fill out this field.

Email *

student@example.com

Major *

-- Select Major --

- When the user clicks the **"Add New Student"** button on the student-list.jsp page, this button is rendered as an HTML anchor tag (<a>) with a specific URL:

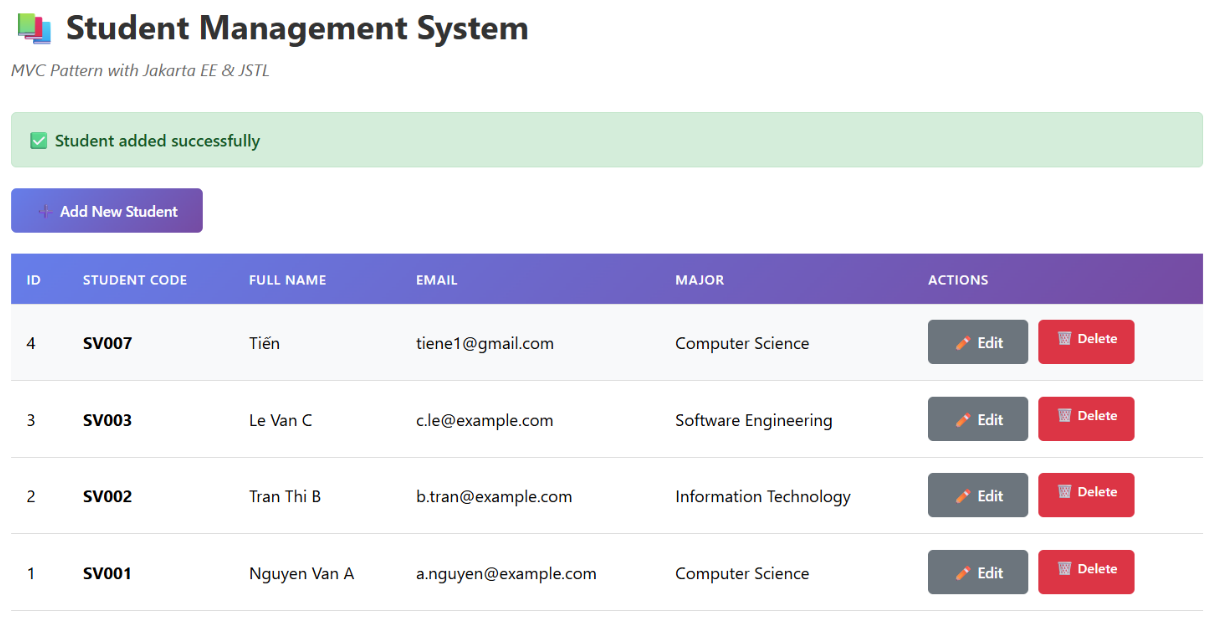
```
<a href="student?action=new" class="btn btn-primary">
    ➕ Add New Student
</a>
```

- The request is received by the StudentController.java Servlet. doGet() Invoked: Since the request is a GET, the doGet() method is executed. Action Parameter Read: The Controller reads the action parameter, which is found to be "new”. Dispatching: The switch statement executes the case "new" block, which calls the showNewForm(request, response) method. Forwarding: Inside showNewForm():

```
RequestDispatcher dispatcher = request.getRequestDispatcher("/views/student-form.jsp");
dispatcher.forward(request, response);
```

- View Execution (Rendering the Form): Form View Loads: The server executes the code inside the student-form.jsp file. Rendering HTML: The JSP generates the HTML for the student entry form, including input fields for Student Code, Full Name, Email, and Major. Display: The generated HTML is sent back to the browser as the HTTP Response, and the user sees the blank form ready for data input.

After adding student’s information:



Student Management System

MVC Pattern with Jakarta EE & JSTL

✅ Student added successfully

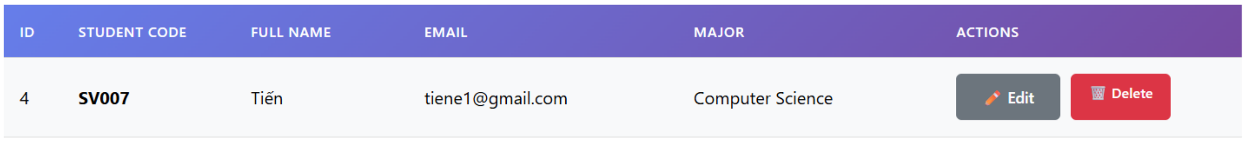
ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
4	**SV007**	Tiến	tiene1@gmail.com	Computer Science	Edit Delete
3	**SV003**	Le Van C	c.le@example.com	Software Engineering	Edit Delete
2	**SV002**	Tran Thi B	b.tran@example.com	Information Technology	Edit Delete
1	**SV001**	Nguyen Van A	a.nguyen@example.com	Computer Science	Edit Delete

- The mechanism is executed by the Controller and the DAO to insert the data into the database. Form Action: The HTML form in student-form.jsp is configured with method="POST" and action="student". The form also contains a hidden field specifying the action:

```
<form action="student" method="POST">
    <!-- Hidden field for action -->
    <input type="hidden" name="action"
           value="${student != null ? 'update' : 'insert'}">
```

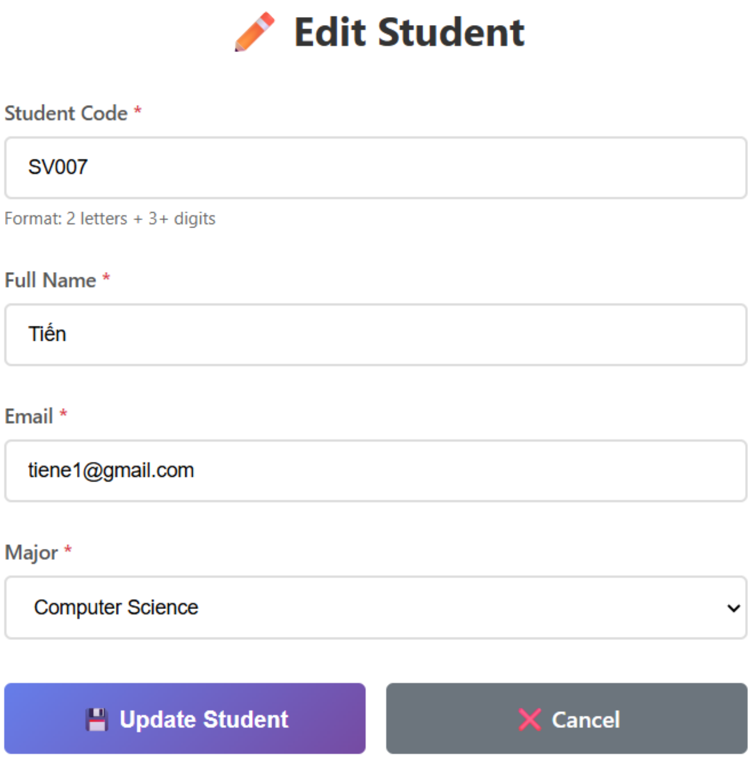
- The browser packages all the input field values (studentCode, fullName, email, major, and action=insert) and sends an **HTTP POST request** to the /student endpoint.
  Controller Action: The request is received by the **StudentController.java** Servlet.**doPost() Invoked:** The server executes the doPost(request, response) method. **Dispatching:** The Controller reads the action="insert" parameter and calls the **insertStudent(request, response)** method. **Parameter Retrieval:** Inside insertStudent():The Controller retrieves all input values using request.getParameter(). **Model Instantiation:** The Controller creates a new **Student object (Model)** and populates its fields with the retrieved data. **DAO Call (Delegation):** The Controller delegates the responsibility of saving data to the persistence layer:
- **DAO Execution: The StudentDAO.java class executes the database logic in its addStudent(Student student) method:**

3. **Edit**



ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
4	**SV007**	Tiến	tiene1@gmail.com	Computer Science	Edit Delete

After clicking edit button:



✏️ Edit Student

Student Code *	SV007
	Format: 2 letters + 3+ digits
Full Name *	Tiến
Email *	tiene1@gmail.com
Major *	Computer Science

- I'd be glad to explain the mechanism when you click the Edit button in your MVC application. This process involves a two-stage workflow:
  - Stage 1: GET Request (Displaying the Form)
  - Stage 2: POST Request (Executing the Update)
- Here is a detailed explanation of the mechanism, starting from the Edit button on the student list page.
- Stage 1: Mechanism for Displaying the Edit Form
  - The goal here is to load the existing student's data into the student-form.jsp interface.
  - From the View (List Page)
  - Action: On the student-list.jsp page, you click the Edit button next to a specific student (e.g., SV007).
  - GET Request: The Edit button is an anchor tag (<a>) constructed to send an HTTP GET request to the Controller, crucially including the student's ID:
  - HTML
  - <a href="student?action=edit&id=123" ...>Edit</a>
  - Controller Processing (StudentController.java)
  - doGet(): The Controller receives the GET request.

    - Dispatch: It identifies action=edit and calls the showEditForm(request, response) method.
    - DAO Call: The showEditForm() method retrieves the ID from the request, then calls the DAO method:
    - Java
    - Student existingStudent = studentDAO.getStudentById(id);
    - request.setAttribute("student", existingStudent); // Attaches Model to Request
    - Forwarding: The Controller forwards the request to the View (student-form.jsp), carrying the populated Student object (SV007).
    - View Rendering (student-form.jsp)
    - Data Population: student-form.jsp receives the Student object from the request.
    - Display: The View uses JSP Expression Language (EL) to pre-populate the input fields with the existing student's data (e.g., value="${student.fullName}").
    - Result: The browser displays the form filled with old data, ready for the user to make changes.
- Stage 2: Mechanism for Data Update
    - After the user modifies the data on the form (e.g., changes the name) and clicks the Update Student button.
    - View Submits Request (student-form.jsp)
    - Action: The Update Student button (Submit button) is clicked.
    - POST Request: The form is configured to send an HTTP POST request to the Controller, including all the new form fields and a hidden field action=update:
    - HTML
    - <form action="student" method="POST">
    - <input type="hidden" name="action" value="update">
    - <input type="hidden" name="id" value="123"> </form>
    - Controller Processing (StudentController.java)
    - doPost(): The Controller receives the POST request.
    - Dispatch: It identifies action=update and calls the updateStudent(request, response) method.
    - Model Creation: The updateStudent() method retrieves *all* the new data (including the mandatory ID), creates a new Student object, and sets the updated fields.
    - DAO Call: The Controller delegates the saving logic:

- Java
- studentDAO.updateStudent(student); // Calls DAO to execute UPDATE SQL
- DAO Execution (StudentDAO.java)
- SQL Execution: The updateStudent() method in the DAO executes the UPDATE SQL query in the database, using the student's ID in the WHERE clause to target the specific record for modification.
- Final Response
- Redirection: Upon successful UPDATE, the Controller executes the Post-Redirect-Get (PRG) pattern by using response.sendRedirect("student?action=list&message=...") to send the user back to the list page, showing the success message and the updated data.