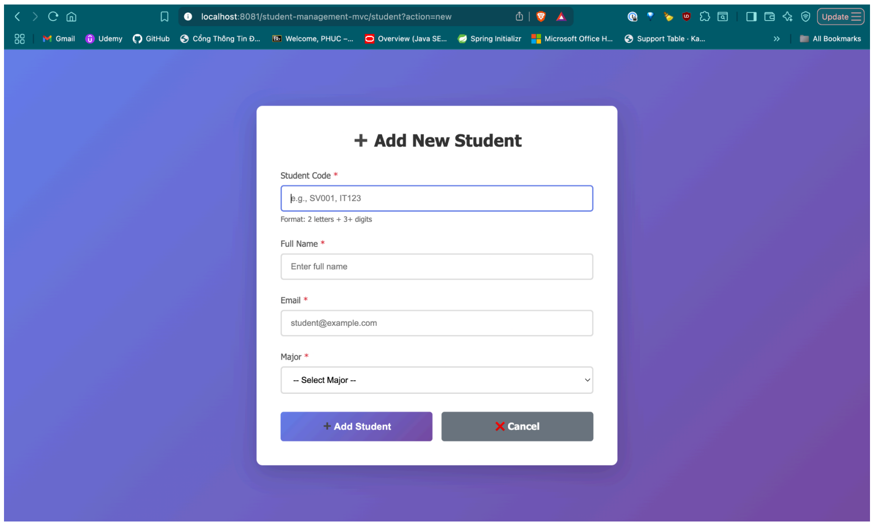
ITITWE23039 - Phan Kim Vĩnh Phúc

Lab 5

Github site & link

https://github.com/alexis0704/web-dev-practice-lab/tree/main/student-management-mvc

Running the demo



+ Add New Student

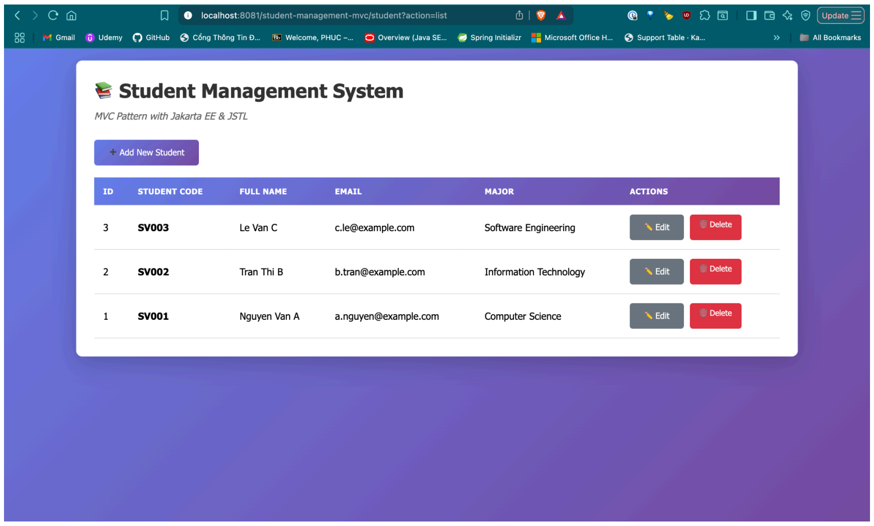
Student Code *
e.g., SV001, IT123
Format: 2 letters + 3+ digits

Full Name *
Enter full name

Email *
student@example.com

Major *
-- Select Major --

+ Add Student | ✕ Cancel



Student Management System

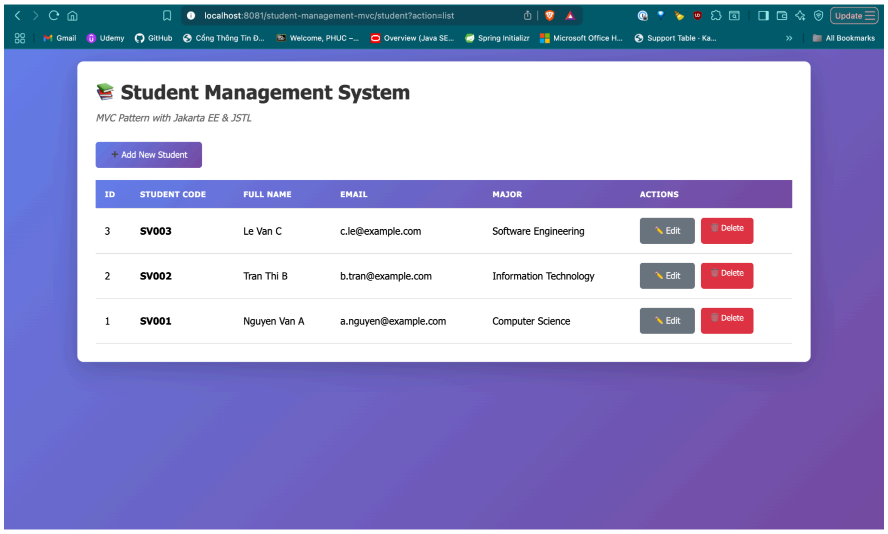
MVC Pattern with Jakarta EE & JSTL

+ Add New Student

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

Code Test and Explanation

1. List Students (/student?action=list or /student)
- What the user does: go to /student (or /student?action=list)



localhost:8081/student-management-mvc/student?action=list

Student Management System

MVC Pattern with Jakarta EE & JSTL

+ Add New Student

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

- What happens in the code
    - Controller (StudentController)
        - doGet() reads the action parameter.
        - If action is null, it defaults to "list":

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

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

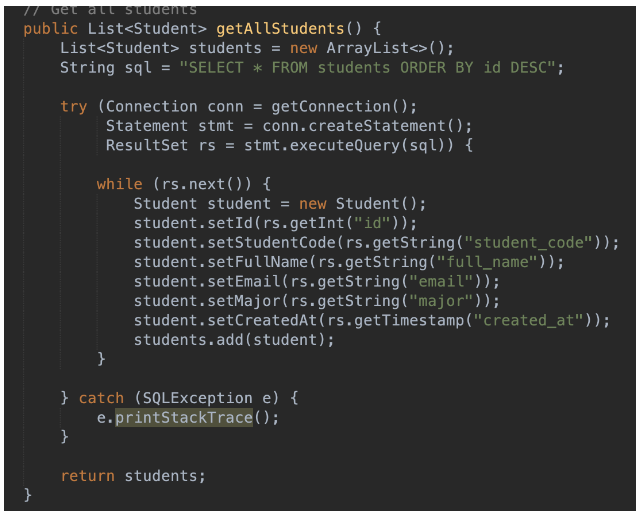
        - For "list", it calls listStudents(request, response).

```
 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;
 }
```

    - DAO (StudentDAO)
        - In the controller, listStudents() calls:

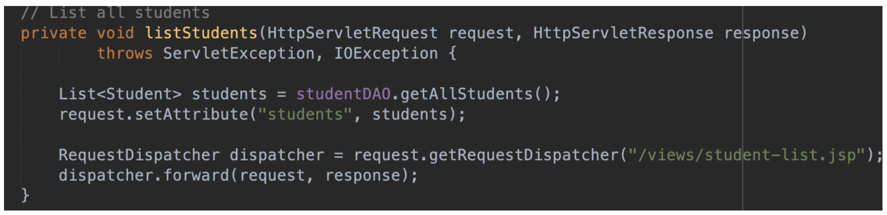
```
// List all students
private void listStudents(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

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

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

        - getAllStudents() opens a DB connection, executes: “SELECT * FROM students ORDER BY id DESC;”. This getAllStudents() function maps each row to a Student object and returns a List<Student>.



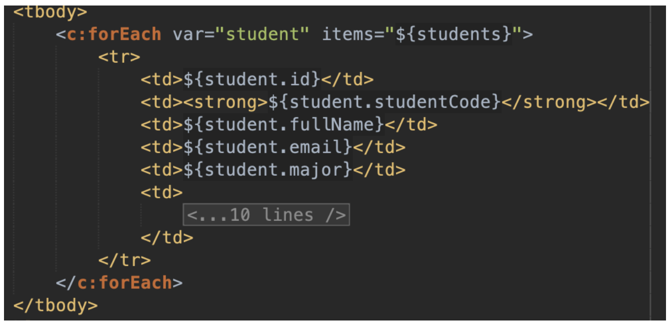
```java
// Get all students
public List<Student> getAllStudents() {
    List<Student> students = new ArrayList<>();
    String sql = "SELECT * FROM students ORDER BY id DESC";

    try (Connection conn = getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            Student student = new Student();
            student.setId(rs.getInt("id"));
            student.setStudentCode(rs.getString("student_code"));
            student.setFullName(rs.getString("full_name"));
            student.setEmail(rs.getString("email"));
            student.setMajor(rs.getString("major"));
            student.setCreatedAt(rs.getTimestamp("created_at"));
            students.add(student);
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return students;
}
```

- View (student-list.jsp)
    - In the controller, listStudents() sets:



```java
// List all students
private void listStudents(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

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

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

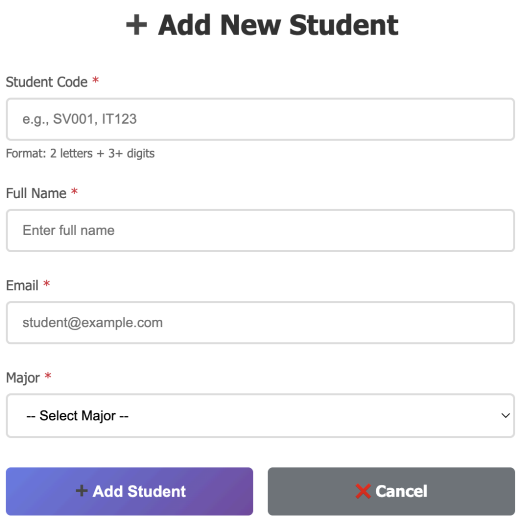
    - In student-list.jsp, JSTL is used to display the list:



```jsp
<tbody>
    <c:forEach var="student" items="${students}">
        <tr>
            <td>${student.id}</td>
            <td><strong>${student.studentCode}</strong></td>
            <td>${student.fullName}</td>
            <td>${student.email}</td>
            <td>${student.major}</td>
            <td>
                <...10 lines />
            </td>
        </tr>
    </c:forEach>
</tbody>
```

2. Add New Student
- What the user does
    - On the student list page, click the button:

- Fill the form with test data and submit.



➕ Add New Student

Student Code *

e.g., SV001, IT123

Format: 2 letters + 3+ digits

Full Name *

Enter full name

Email *

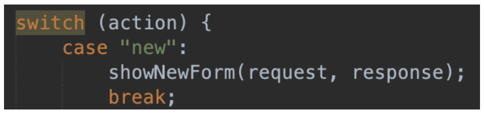
student@example.com

Major *

-- Select Major --

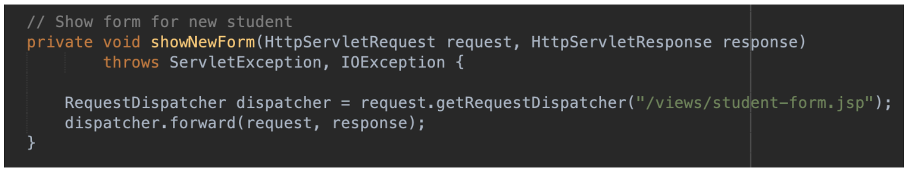
➕ Add Student | ❌ Cancel

- What happens in the code
    - Show Add Form (action=new)
        - Controller:
            - doGet() sees action = "new" and calls:



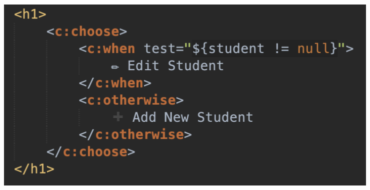
```
switch (action) {
    case "new":
        showNewForm(request, response);
        break;
```

            - showNewForm() just forwards to the form:



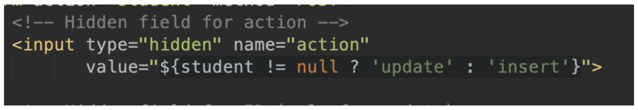
```
// Show form for new student
private void showNewForm(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

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

        - View (student-form.jsp)
            - Since no student attribute is set, ${student} is null.
            - The title and heading use <c:choose> to show “Add New Student”:



```
<h1>
    <c:choose>
        <c:when test="${student != null}">
            ✏ Edit Student
        </c:when>
        <c:otherwise>
            ➕ Add New Student
        </c:otherwise>
    </c:choose>
</h1>
```

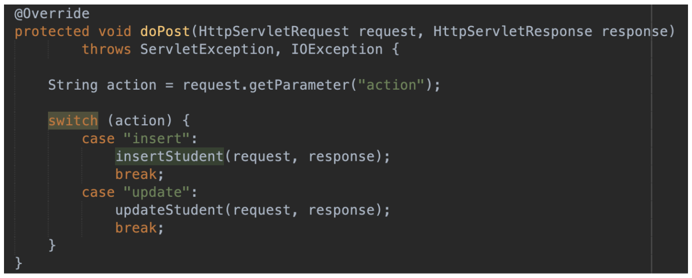
            - The hidden action field is set to "insert":



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

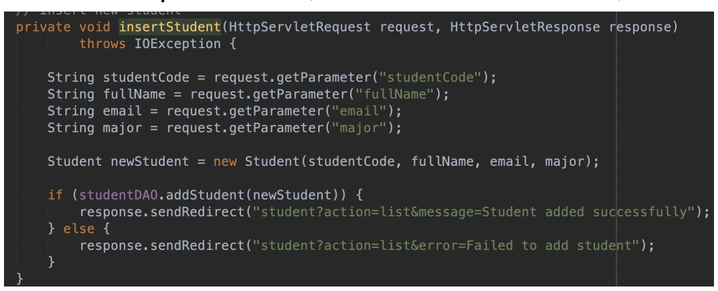
            - Input fields are empty and required.
    - Submit Add Form (action=insert)
        - Controller – doPost()
            - The form posts to action="insert"

- doPost() routes to:



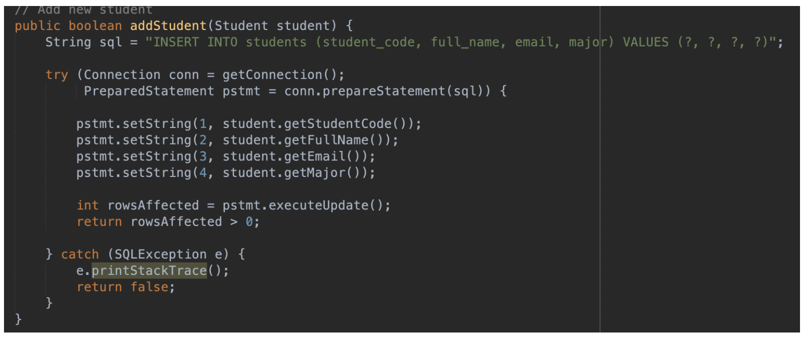
```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

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

    switch (action) {
        case "insert":
            insertStudent(request, response);
            break;
        case "update":
            updateStudent(request, response);
            break;
    }
}
```

- Controller – insertStudent()
    - Reads form parameters, creates a new Student, and calls DAO:



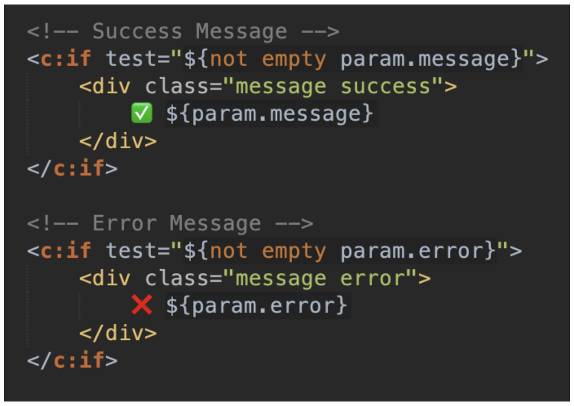
```
private void insertStudent(HttpServletRequest request, HttpServletResponse response)
        throws IOException {

    String studentCode = request.getParameter("studentCode");
    String fullName = request.getParameter("fullName");
    String email = request.getParameter("email");
    String major = request.getParameter("major");

    Student newStudent = new Student(studentCode, fullName, email, major);

    if (studentDAO.addStudent(newStudent)) {
        response.sendRedirect("student?action=list&message=Student added successfully");
    } else {
        response.sendRedirect("student?action=list&error=Failed to add student");
    }
}
```

- DAO – addStudent()
    - Executes an INSERT statement:



```
// Add new student
public boolean addStudent(Student student) {
    String sql = "INSERT INTO students (student_code, full_name, email, major) VALUES (?, ?, ?, ?)";

    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, student.getStudentCode());
        pstmt.setString(2, student.getFullName());
        pstmt.setString(3, student.getEmail());
        pstmt.setString(4, student.getMajor());

        int rowsAffected = pstmt.executeUpdate();
        return rowsAffected > 0;

    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
```

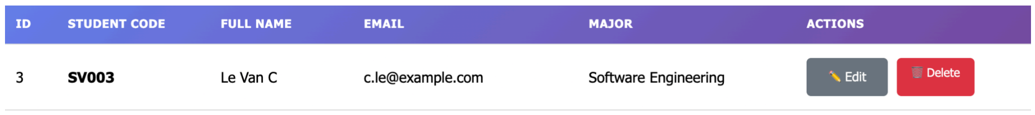
- View
    - After redirection, student-list.jsp is loaded again. The message is displayed within the form as follows:



```
<!-- Success Message -->
<c:if test="${not empty param.message}">
    <div class="message success">
        ✅ ${param.message}
    </div>
</c:if>

<!-- Error Message -->
<c:if test="${not empty param.error}">
    <div class="message error">
        ❌ ${param.error}
    </div>
</c:if>
```

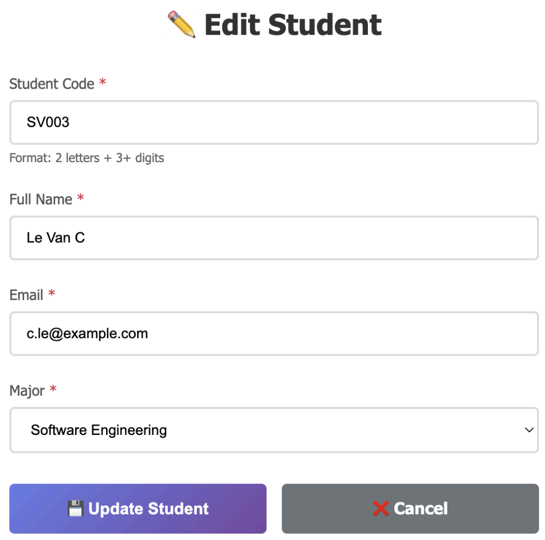
3. Edit Existing Student
- What the user does

- On the list page, click Edit for a specific student:



ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
3	**SV003**	Le Van C	c.le@example.com	Software Engineering	✏️ Edit 🗑 Delete

- The form opens with fields pre-filled. Modify data and submit.



✏️ **Edit Student**

Student Code *

SV003

Format: 2 letters + 3+ digits

Full Name *

Le Van C

Email *

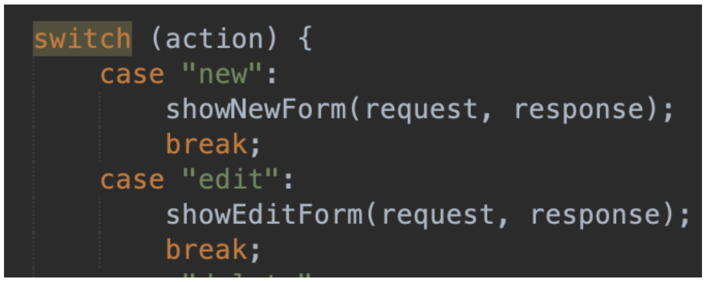
c.le@example.com

Major *

Software Engineering

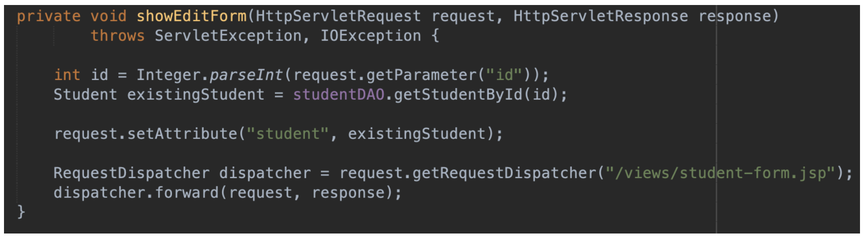
💾 Update Student ❌ Cancel

- What happens in the code
  - Show Edit Form (action=edit)
    - Controller – doGet(): Reads action = "edit" and id from the query string. Calls showEditForm()



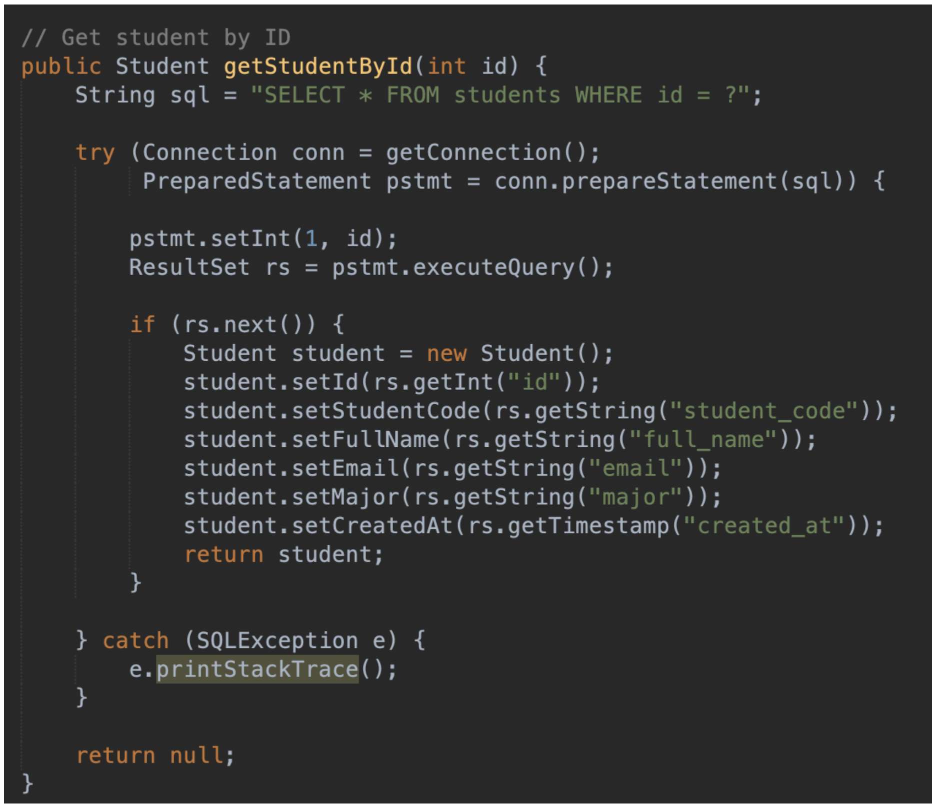
```java
switch (action) {
    case "new":
        showNewForm(request, response);
        break;
    case "edit":
        showEditForm(request, response);
        break;
```

    - Controller – showEditForm(): Parses the id and forwards to student-form.jsp



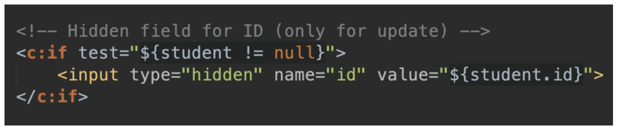
```java
private void showEditForm(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    int id = Integer.parseInt(request.getParameter("id"));
    Student existingStudent = studentDAO.getStudentById(id);

    request.setAttribute("student", existingStudent);

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

- DAO – getStudentById(): Executes “SELECT * FROM students WHERE id = ?;”



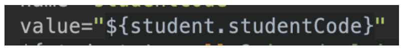
```java
// Get student by ID
public Student getStudentById(int id) {
    String sql = "SELECT * FROM students WHERE id = ?";

    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();

        if (rs.next()) {
            Student student = new Student();
            student.setId(rs.getInt("id"));
            student.setStudentCode(rs.getString("student_code"));
            student.setFullName(rs.getString("full_name"));
            student.setEmail(rs.getString("email"));
            student.setMajor(rs.getString("major"));
            student.setCreatedAt(rs.getTimestamp("created_at"));
            return student;
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}
```

- View – student-form.jsp: This time, ${student} is not null. Dynamic title and heading show “Edit Student”.
    - Hidden fields:

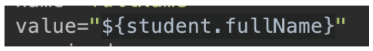


```jsp
<!-- Hidden field for ID (only for update) -->
<c:if test="${student != null}">
    <input type="hidden" name="id" value="${student.id}">
</c:if>
```

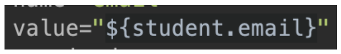
    - All inputs are pre-filled:



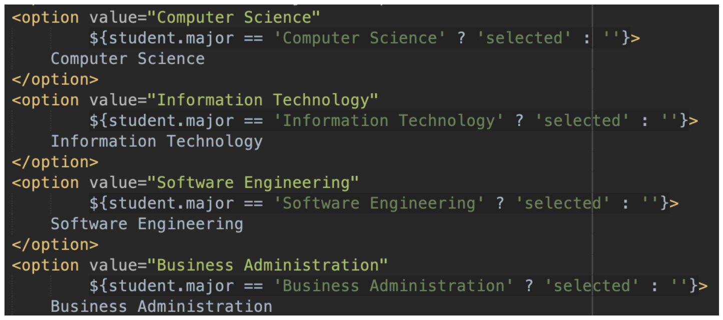
```jsp
value="${student.studentCode}"
```



```jsp
value="${student.fullName}"
```

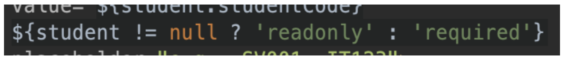


```jsp
value="${student.email}"
```



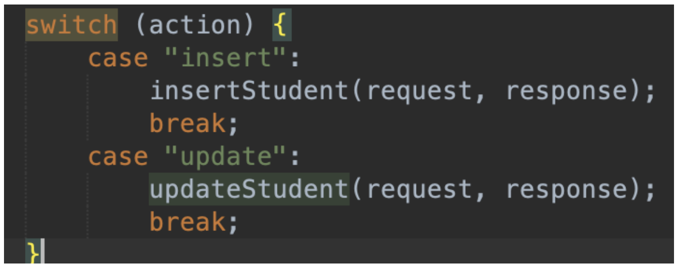
```jsp
<option value="Computer Science"
        ${student.major == 'Computer Science' ? 'selected' : ''}>
    Computer Science
</option>
<option value="Information Technology"
        ${student.major == 'Information Technology' ? 'selected' : ''}>
    Information Technology
</option>
<option value="Software Engineering"
        ${student.major == 'Software Engineering' ? 'selected' : ''}>
    Software Engineering
</option>
<option value="Business Administration"
        ${student.major == 'Business Administration' ? 'selected' : ''}>
    Business Administration
```

            - Student code is set to readonly



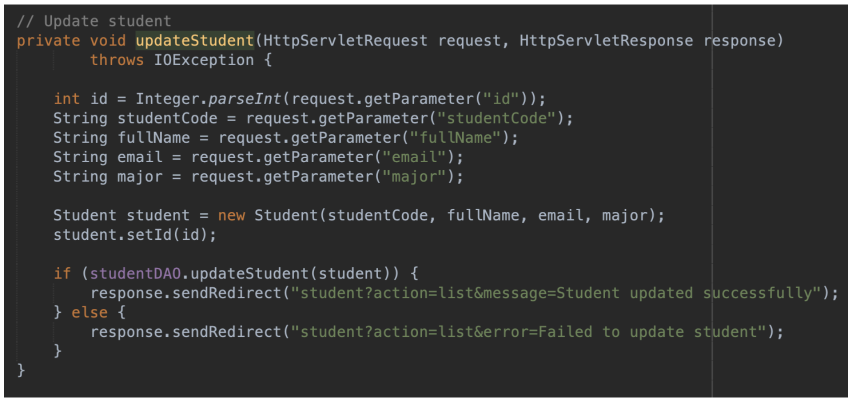
```
${student != null ? 'readonly' : 'required'}
```

        - Submit Edit Form (action=update)
            - Controller – doPost(): Routes action="update" to updateStudent(request, response)



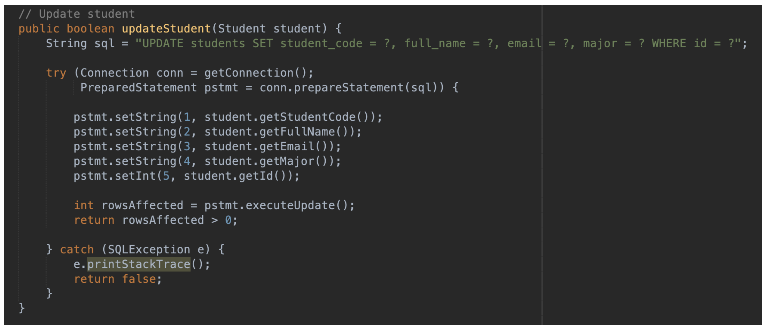
```
switch (action) {
    case "insert":
        insertStudent(request, response);
        break;
    case "update":
        updateStudent(request, response);
        break;
}
```

            - Controller – updateStudent(): Reads parameters including id, calls studentDAO.updateStudent(student), and redirect to student-list.jsp



```
// Update student
private void updateStudent(HttpServletRequest request, HttpServletResponse response)
        throws IOException {

    int id = Integer.parseInt(request.getParameter("id"));
    String studentCode = request.getParameter("studentCode");
    String fullName = request.getParameter("fullName");
    String email = request.getParameter("email");
    String major = request.getParameter("major");

    Student student = new Student(studentCode, fullName, email, major);
    student.setId(id);

    if (studentDAO.updateStudent(student)) {
        response.sendRedirect("student?action=list&message=Student updated successfully");
    } else {
        response.sendRedirect("student?action=list&error=Failed to update student");
    }
}
```

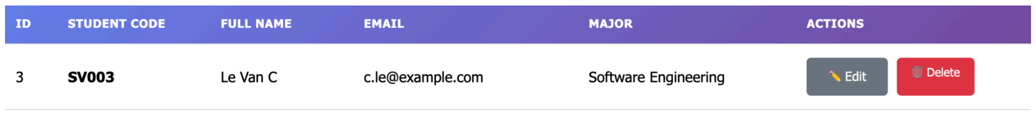
            - DAO – updateStudent(): Executes “UPDATE students SET student_code = ?, full_name = ?, email = ?, major = ? WHERE id = ?;”



```
// Update student
public boolean updateStudent(Student student) {
    String sql = "UPDATE students SET student_code = ?, full_name = ?, email = ?, major = ? WHERE id = ?";

    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, student.getStudentCode());
        pstmt.setString(2, student.getFullName());
        pstmt.setString(3, student.getEmail());
        pstmt.setString(4, student.getMajor());
        pstmt.setInt(5, student.getId());

        int rowsAffected = pstmt.executeUpdate();
        return rowsAffected > 0;

    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
```

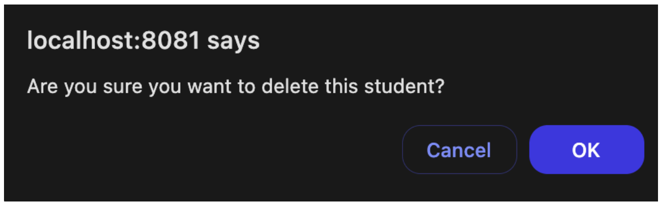
            - View: After redirection, student-list.jsp shows the updated data and the update success message.

4. Delete Student
- What the user does
    - On the list page, click Delete for a student:



ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
3	**SV003**	Le Van C	c.le@example.com	Software Engineering	✏️ Edit 🗑 Delete

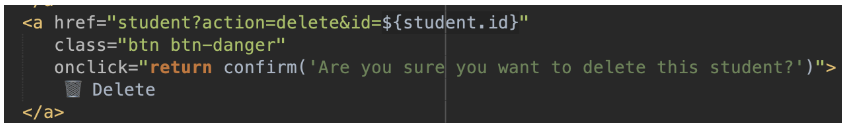
  - The browser shows a confirmation dialog. If the user confirms, the request continues; if they cancel, nothing happens.



localhost:8081 says

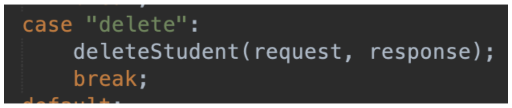
Are you sure you want to delete this student?

- What happens in the code
  - Browser: onclick="return confirm(...)" uses JavaScript:
    - true → continue navigation to /student?action=delete&id=...
    - false → cancel



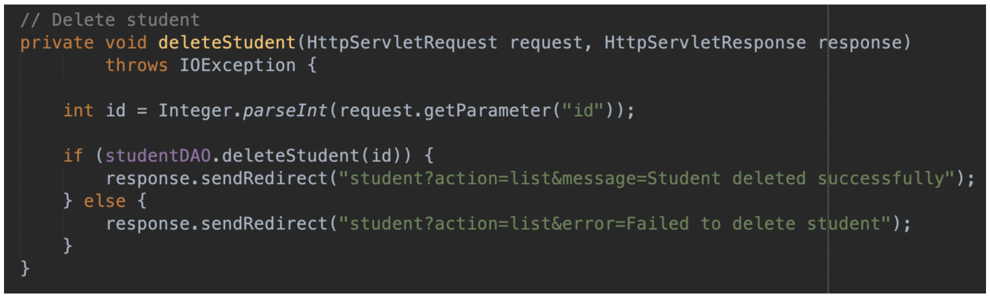
```
<a href="student?action=delete&id=${student.id}"
   class="btn btn-danger"
   onclick="return confirm('Are you sure you want to delete this student?')">
    🗑️ Delete
</a>
```

  - Controller – doGet(): For action="delete", calls deleteStudent(request, response)



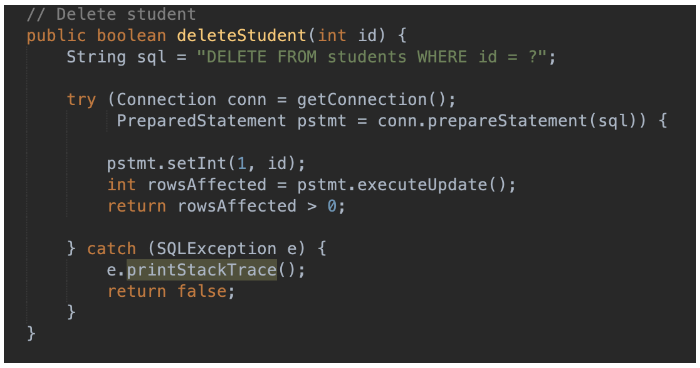
```
case "delete":
    deleteStudent(request, response);
    break;
```

  - Controller – deleteStudent(): Reads the id, calls DAO and redirect:



```
// Delete student
private void deleteStudent(HttpServletRequest request, HttpServletResponse response)
        throws IOException {

    int id = Integer.parseInt(request.getParameter("id"));

    if (studentDAO.deleteStudent(id)) {
        response.sendRedirect("student?action=list&message=Student deleted successfully");
    } else {
        response.sendRedirect("student?action=list&error=Failed to delete student");
    }
}
```

  - DAO – deleteStudent(): Executes "DELETE FROM students WHERE id = ?;"

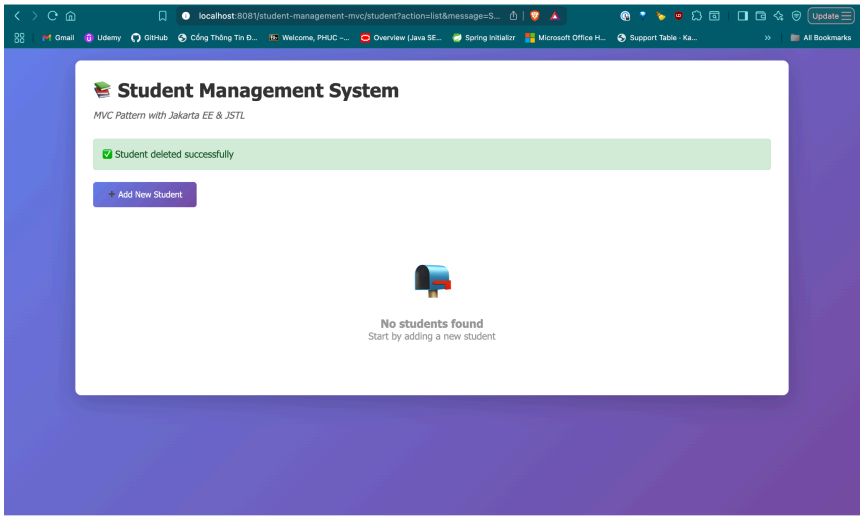


```
// Delete student
public boolean deleteStudent(int id) {
    String sql = "DELETE FROM students WHERE id = ?";

    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setInt(1, id);
        int rowsAffected = pstmt.executeUpdate();
        return rowsAffected > 0;

    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
```

  - View: After redirection, student-list.jsp shows remaining students and the delete success message.

5. Empty State (No Students)

- What the user does: Repeatedly delete students until no rows remain in the students table, then go to /student.



localhost:8081/student-management-mvc/student?action=list&message=S...

📚 Student Management System

MVC Pattern with Jakarta EE & JSTL

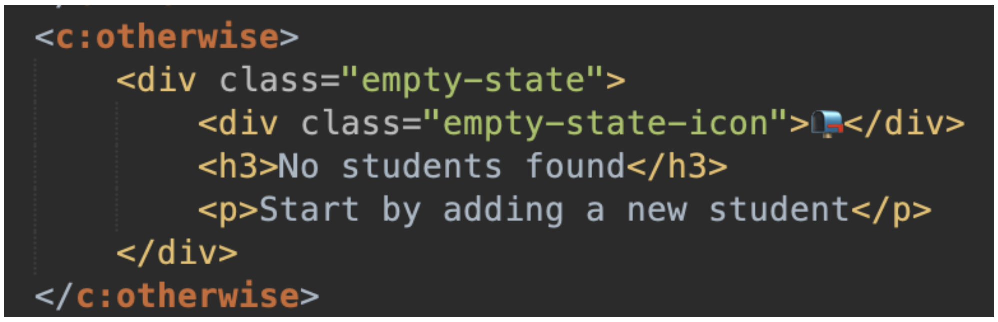
✅ Student deleted successfully

📭

No students found

Start by adding a new student

- What happens in the code
    - DAO: getAllStudents() returns an empty List<Student>
    - Controller: listStudents() puts this empty list into the request as "students" and forwards to student-list.jsp.
    - View – student-list.jsp: Uses <c:choose> to differentiate between non-empty and empty:



```
<c:otherwise>
    <div class="empty-state">
        <div class="empty-state-icon">📭</div>
        <h3>No students found</h3>
        <p>Start by adding a new student</p>
    </div>
</c:otherwise>
```