Trần Phúc Quý - ITITDK23027

Lab 5

1. Overview

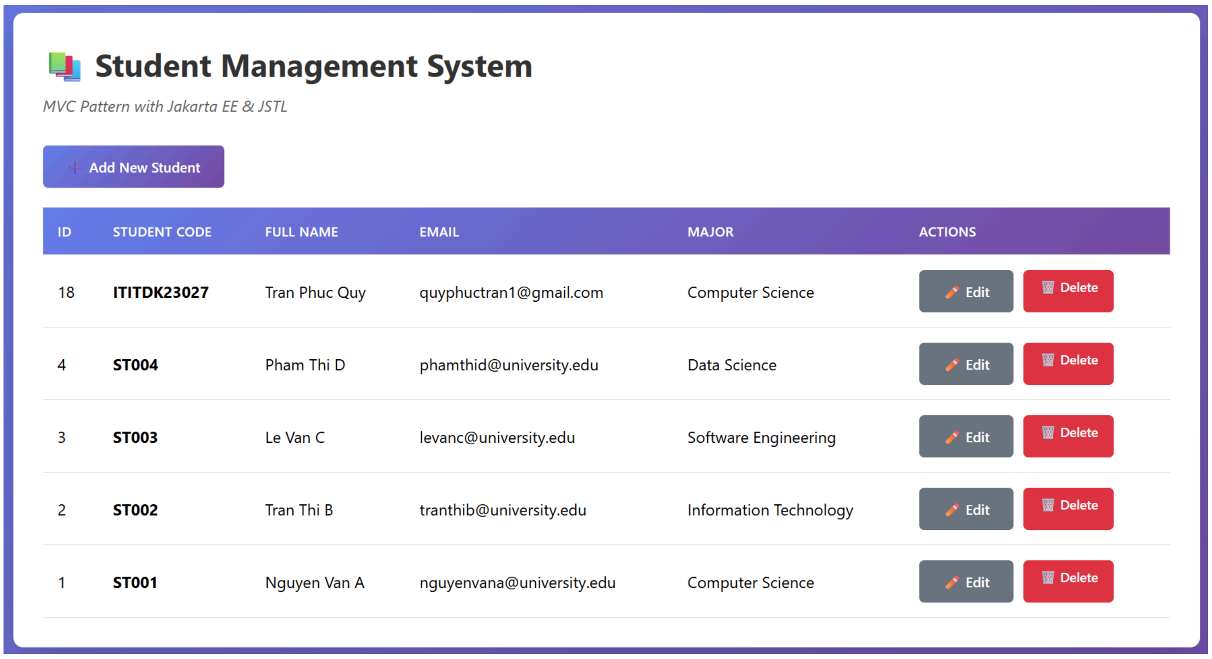
This document describes the **end-to-end workflow** of the Student Management System's CRUD operations using the **MVC pattern**:

- **Controller (Servlet)**: StudentController
- **DAO Layer**: StudentDAO
- **View Layer**: JSP pages (student-list.jsp, student-form.jsp)
- **Database Table**: students

Core operations:

1. **READ** – List all students
2. **CREATE** – Insert a new student
3. **UPDATE** – Edit existing student information
4. **DELETE** – Remove a student record

2. READ OPERATION – LIST STUDENTS



Student Management System

MVC Pattern with Jakarta EE & JSTL

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
18	**ITITDK23027**	Tran Phuc Quy	quyphuctran1@gmail.com	Computer Science	Edit Delete
4	**ST004**	Pham Thi D	phamthid@university.edu	Data Science	Edit Delete
3	**ST003**	Le Van C	levanc@university.edu	Software Engineering	Edit Delete
2	**ST002**	Tran Thi B	tranthib@university.edu	Information Technology	Edit Delete
1	**ST001**	Nguyen Van A	nguyenvana@university.edu	Computer Science	Edit Delete

2.1 Request Flow

HTTP GET: /student OR /student?action=list

↓

StudentController.doGet()

↓

listStudents()

↓
StudentDAO.getAllStudents()
↓
Database Query: SELECT * FROM students ORDER BY id DESC
↓
ResultSet → List<Student>
↓
request.setAttribute("students", students)
↓
Forward to: /views/student-list.jsp
↓
JSTL <c:forEach> renders student table
↓
HTML Response to Browser

2.2 Controller Method

```
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);
}
```

Responsibilities:

- Call DAO to get all students.
- Attach the result to the request scope as students.
- Forward to the JSP view for rendering.

2.3 DAO Method (getAllStudents)

Behavior:

- Opens a database connection.
- Prepares and executes:

```
SELECT * FROM students ORDER BY id DESC;
```

- Iterates through the ResultSet:
  - For each row, create a Student object.
  - Adds each Student to a List<Student>.
- Returns List<Student> (returns an **empty list** if there are no records).

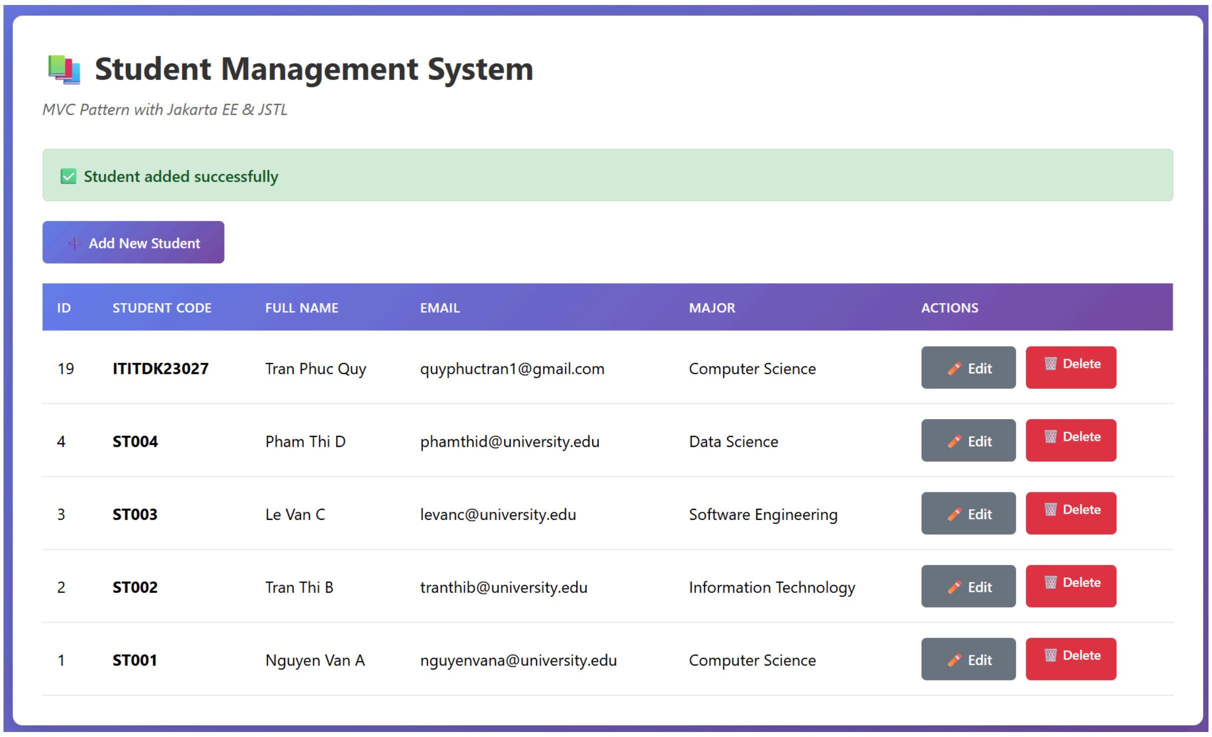
2.4 View Rendering (student-list.jsp)

- Receives the attribute ${students} from the request.
- Uses JSTL <c:forEach> to iterate:

```
<c:forEach var="student" items="${students}">
   <!-- Render table row -->
</c:forEach>
```

- Displays for each student:
  - **ID**
  - **Student Code**
  - **Full Name**
  - **Email**
  - **Major**
  - **Actions**: Edit / Delete buttons (typically links or forms).

3. CREATE OPERATION – INSERT NEW STUDENT



Student Management System

MVC Pattern with Jakarta EE & JSTL

✓ Student added successfully

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
19	**ITITDK23027**	Tran Phuc Quy	quyphuctran1@gmail.com	Computer Science	Edit Delete
4	**ST004**	Pham Thi D	phamthid@university.edu	Data Science	Edit Delete
3	**ST003**	Le Van C	levanc@university.edu	Software Engineering	Edit Delete
2	**ST002**	Tran Thi B	tranthib@university.edu	Information Technology	Edit Delete
1	**ST001**	Nguyen Van A	nguyenvana@university.edu	Computer Science	Edit Delete

3.1 Request Flow

STEP 1 – Show Empty Form

HTTP GET: /student?action=new
↓
StudentController.doGet() → case "new"
↓
showNewForm()
↓

Forward to: /views/student-form.jsp
↓
Render empty form (no "student" attribute)

STEP 2 – Submit Form

HTTP POST: /student (action=insert)
↓
StudentController.doPost() → case "insert"
↓
insertStudent()
↓
Extract parameters: studentCode, fullName, email, major
↓
Validate: Check not null/empty
↓
Create: new Student(studentCode, fullName, email, major)
↓
StudentDAO.addStudent(student)
↓
Database Execute: INSERT INTO students (...)
↓
Returns: true (success) or false (failure)
↓
Redirect to: /student?action=list&message=success (or error)

3.2 Controller Logic – insertStudent

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

    // 1. Extract form parameters
    String studentCode = request.getParameter("studentCode");
    String fullName    = request.getParameter("fullName");
    String email       = request.getParameter("email");
    String major       = request.getParameter("major");

    // 2. Basic validation (not null / not empty)
    if (studentCode == null || studentCode.isBlank() ||
        fullName == null    || fullName.isBlank()    ||
        email == null       || email.isBlank()       ||
        major == null       || major.isBlank()) {

        response.sendRedirect("student?action=list&message=validationError");
        return;
    }

    // 3. Create Student object
    Student newStudent = new Student(studentCode, fullName, email, major);

    // 4. Call DAO to insert
    boolean success = studentDAO.addStudent(newStudent);

    // 5. Redirect with result message
    if (success) {
        response.sendRedirect("student?action=list&message=insertSuccess");
    } else {
        response.sendRedirect("student?action=list&message=insertError");
    }
}
```

3.3 DAO Method – addStudent

- Uses a PreparedStatement:

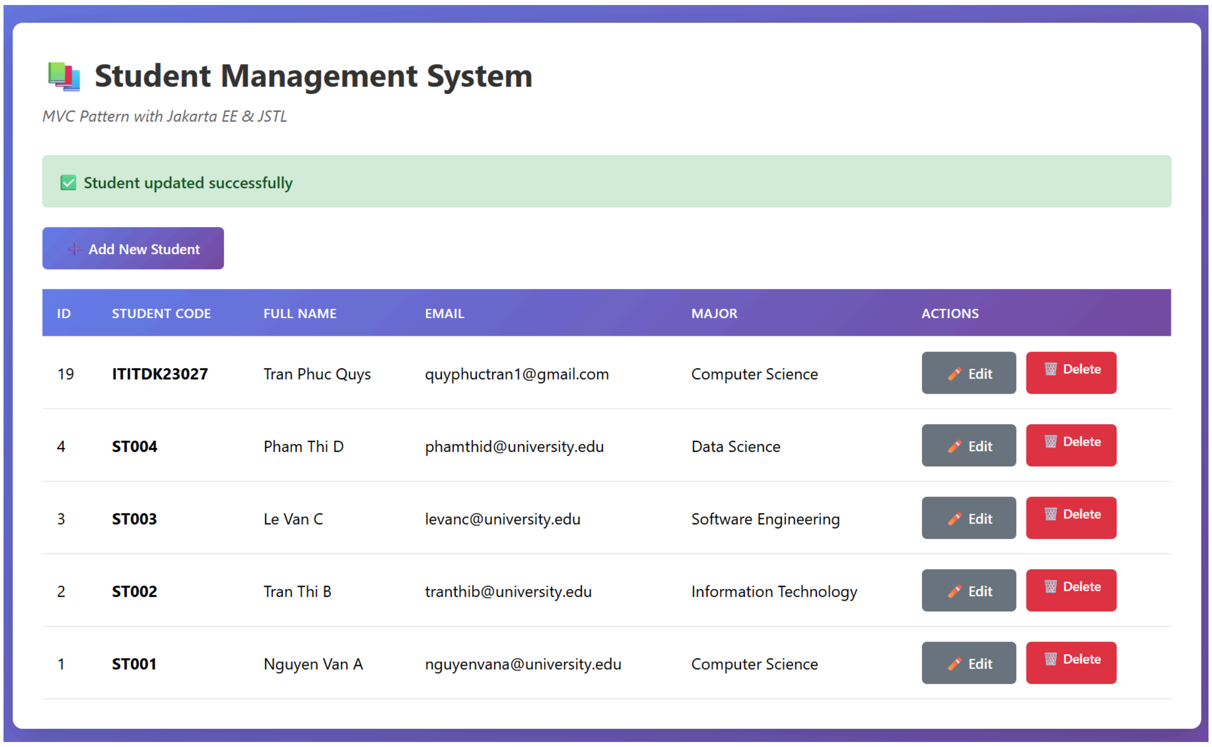
```
INSERT INTO students (student_code, full_name, email, major)
VALUES (?, ?, ?, ?);
```

- Sets 4 parameters from the Student object.
- Calls executeUpdate().
- Returns true if rowsAffected > 0, otherwise false.

3.4 Error Handling (CREATE)

- Catches SQLException, especially for:
    - **Duplicate student_code**
    - **Duplicate email**
      (assuming **UNIQUE constraints** in the database).
- Logs:
    - Error code
    - SQL state
    - Error message
- Returns false so the controller can display an error message to the user.

4. UPDATE OPERATION – EDIT STUDENT


📚 Student Management System

MVC Pattern with Jakarta EE & JSTL

✅ Student updated successfully

+ Add New Student

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
19	**ITITDK23027**	Tran Phuc Quys	quyphuctran1@gmail.com	Computer Science	Edit Delete
4	**ST004**	Pham Thi D	phamthid@university.edu	Data Science	Edit Delete
3	**ST003**	Le Van C	levanc@university.edu	Software Engineering	Edit Delete
2	**ST002**	Tran Thi B	tranthib@university.edu	Information Technology	Edit Delete
1	**ST001**	Nguyen Van A	nguyenvana@university.edu	Computer Science	Edit Delete

4.1 Request Flow

STEP 1 – Show Edit Form

HTTP GET: /student?action=edit&id=1

↓

StudentController.doGet() → case "edit"

↓

showEditForm()

↓

Extract id parameter

↓

StudentDAO.getStudentById(id)

↓

Database Query: SELECT * FROM students WHERE id = ?

↓

ResultSet → Student object

↓

request.setAttribute("student", existingStudent)

↓

Forward to: /views/student-form.jsp

↓

Render form pre-filled with student data

STEP 2 – Submit Edited Form

```
HTTP POST: /student (action=update, id=1)
  ↓
StudentController.doPost() → case "update"
  ↓
updateStudent()
  ↓
Extract parameters: id, studentCode, fullName, email, major
  ↓
Create Student object, setId(id)
  ↓
StudentDAO.updateStudent(student)
  ↓
Database Execute: UPDATE students SET ... WHERE id = ?
  ↓
Returns: true (success) or false (failure)
  ↓
Redirect to: /student?action=list&message=updated
```

4.2 Controller Logic – updateStudent

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

    // 1. Extract parameters
    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");

    // (Optional) validate updated fields similar to insert

    // 2. Create Student object and set ID
    Student student = new Student(studentCode, fullName, email, major);
    student.setId(id);  // used in WHERE clause

    // 3. Call DAO to update
    boolean success = studentDAO.updateStudent(student);

    // 4. Redirect with result
    if (success) {
        response.sendRedirect("student?action=list&message=updateSuccess");
    } else {
        response.sendRedirect("student?action=list&message=updateError");
    }
}
```

4.3 DAO Method – updateStudent

- Uses a PreparedStatement:

```
UPDATE students
SET student_code = ?, full_name = ?, email = ?, major = ?
WHERE id = ?;
```

- Sets 5 parameters:
    1. student_code
    2. full_name
    3. email
    4. major
    5. id (for the WHERE clause)
- Executes executeUpdate().
- Returns true if rowsAffected > 0 (record exists and is updated).

4.4 Form Behavior (student-form.jsp)

- JSP checks if a student attribute exists:

```
 <c:if test="${student != null}">
    <!-- Edit mode -->
</c:if>
<c:if test="${student == null}">
    <!-- Create mode -->
</c:if>
```

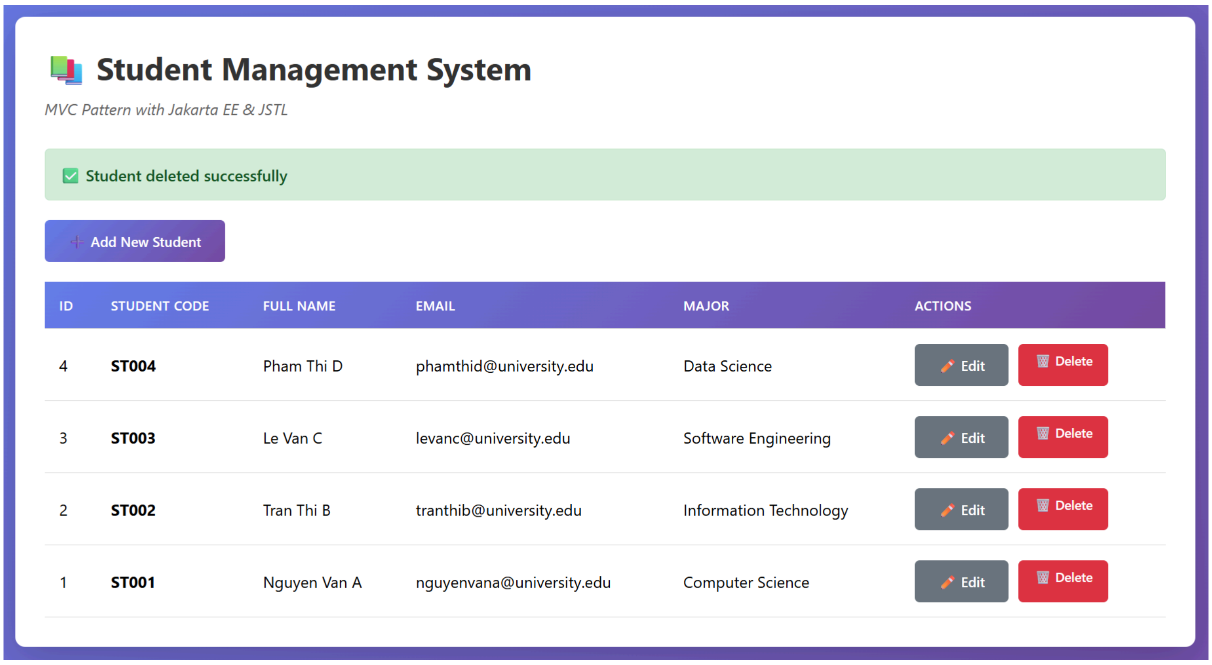
- **Edit mode:**
    - Shows title: **"Edit Student"**
    - Pre-fills form fields using ${student.studentCode}, ${student.fullName}, etc.
    - Often sets studentCode as readonly (if it is a unique identifier).
    - Includes hidden fields:

```
<input type="hidden" name="action" value="update">
<input type="hidden" name="id" value="${student.id}">
```

- **Create mode:**

- Title: **"New Student"**
- All fields are blank.
- Hidden field: action=insert.

5. DELETE OPERATION – REMOVE STUDENT



Student Management System

MVC Pattern with Jakarta EE & JSTL

✓ Student deleted successfully

Add New Student

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
4	**ST004**	Pham Thi D	phamthid@university.edu	Data Science	Edit Delete
3	**ST003**	Le Van C	levanc@university.edu	Software Engineering	Edit Delete
2	**ST002**	Tran Thi B	tranthib@university.edu	Information Technology	Edit Delete
1	**ST001**	Nguyen Van A	nguyenvana@university.edu	Computer Science	Edit Delete

5.1 Request Flow

HTTP GET: /student?action=delete&id=1
↓
StudentController.doGet() → case "delete"
↓
deleteStudent()
↓
Extract id parameter
↓
StudentDAO.deleteStudent(id)
↓
Database Execute: DELETE FROM students WHERE id = ?
↓
Returns: true (success) or false (failure)
↓
Redirect to: /student?action=list&message=deleted

5.2 Controller Logic – deleteStudent

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

```
    // 1. Extract id parameter
    int id = Integer.parseInt(request.getParameter("id"));

    // 2. Call DAO to delete
    boolean success = studentDAO.deleteStudent(id);

    // 3. Redirect with result
    if (success) {
        response.sendRedirect("student?action=list&message=deleteSuccess");
    } else {
        response.sendRedirect("student?action=list&message=deleteError");
    }
}
```

5.3 DAO Method – deleteStudent

- Uses a PreparedStatement:

  ```
  DELETE FROM students WHERE id = ?;
  ```
- Sets 1 parameter: id.
- Executes executeUpdate().
- Returns true if rowsAffected > 0 (record existed and was removed).

5.4 UI Trigger (student-list.jsp)

- Each table row includes a **Delete** button or link.
- Typical HTML:

```
          <a href="student?action=delete&id=${student.id}"
   onclick="return confirm('Delete this student?');">
    Delete
</a>
```

- JavaScript confirm() provides a basic confirmation dialog before sending the delete request.