Lab 5 Exercise
Web Application Development

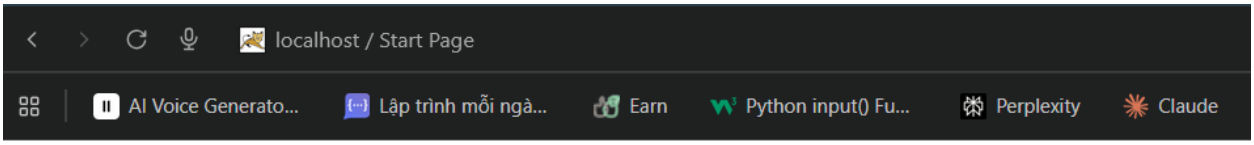
Exercise 1:

Test Cases:

Test Case 1:

```
Scanning for projects...

----------------< com.student:student-management-mvc >----------------
Building Student Management MVC 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ war ]---------------------------------

--- resources:3.3.1:resources (default-resources) @ student-management-mvc ---
Copying 1 resource from src\main\resources to target\classes

--- compiler:3.11.0:compile (default-compile) @ student-management-mvc ---
Changes detected - recompiling the module! :source
Compiling 3 source files with javac [debug target 17] to target\classes
system modules path not set in conjunction with -source 17

--- exec:3.1.0:exec (default-cli) @ student-management-mvc ---
Student{id=5, studentCode='SV005', fullName='David Wilson', email='david.w@email.com', major='Computer Science'}
Student{id=4, studentCode='SV004', fullName='Sarah Davis', email='sarah.d@email.com', major='Data Science'}
Student{id=3, studentCode='SV003', fullName='Michael Brown', email='michael.b@email.com', major='Software Engineering'}
Student{id=2, studentCode='SV002', fullName='Emily Johnson', email='emily.j@email.com', major='Information Technology'}
Student{id=1, studentCode='SV001', fullName='John Smith', email='john.smith@email.com', major='Computer Science'}
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.366 s
Finished at: 2025-11-15T15:18:48+07:00
------------------------------------------------------------------------
```

Test Case 2:



Hello World!

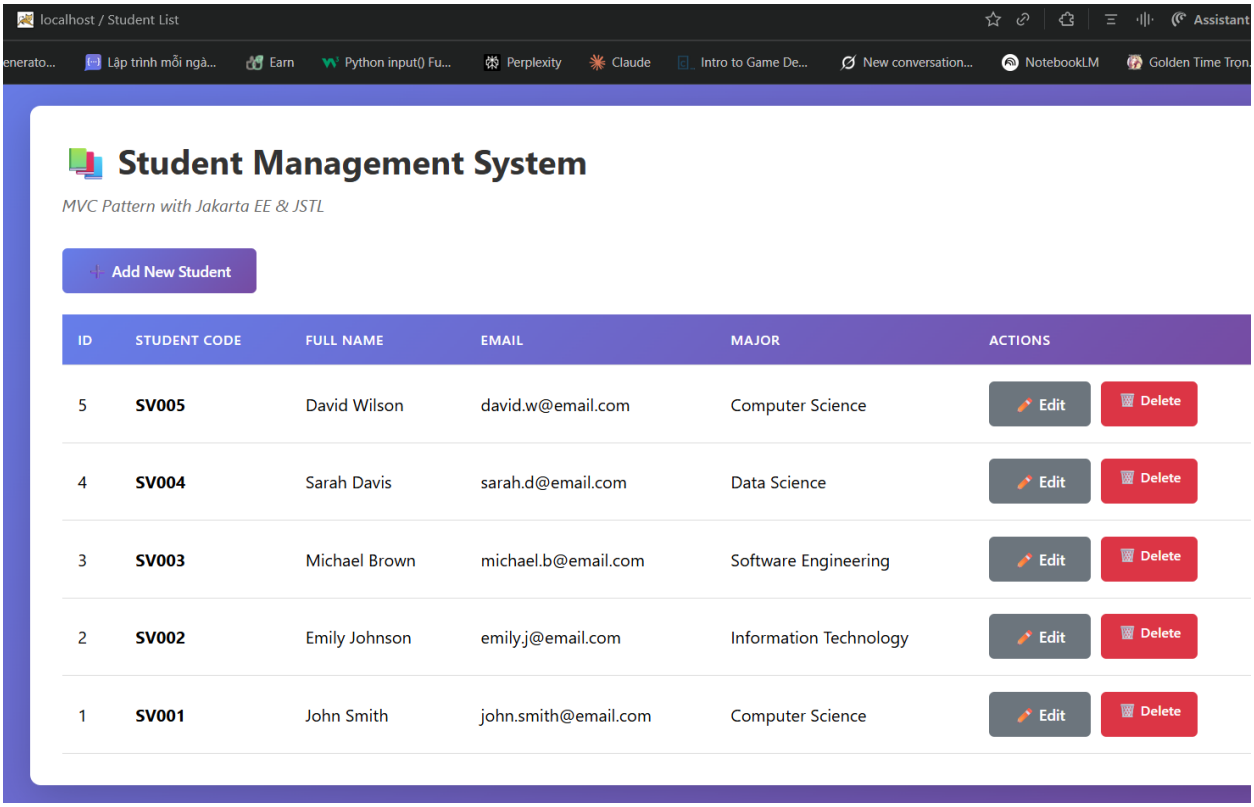
Workflow:

The Student class is designed as a data container that holds student information in memory. The class consists of private attributes—id, studentCode, fullName, email, major, and createdAt—that are accessed through public getters and setters. A no-argument constructor is provided to satisfy JavaBean conventions, and a parameterized constructor is offered to initialize new student instances without requiring the id field.
The toString() method was overridden to facilitate debugging and display operations.

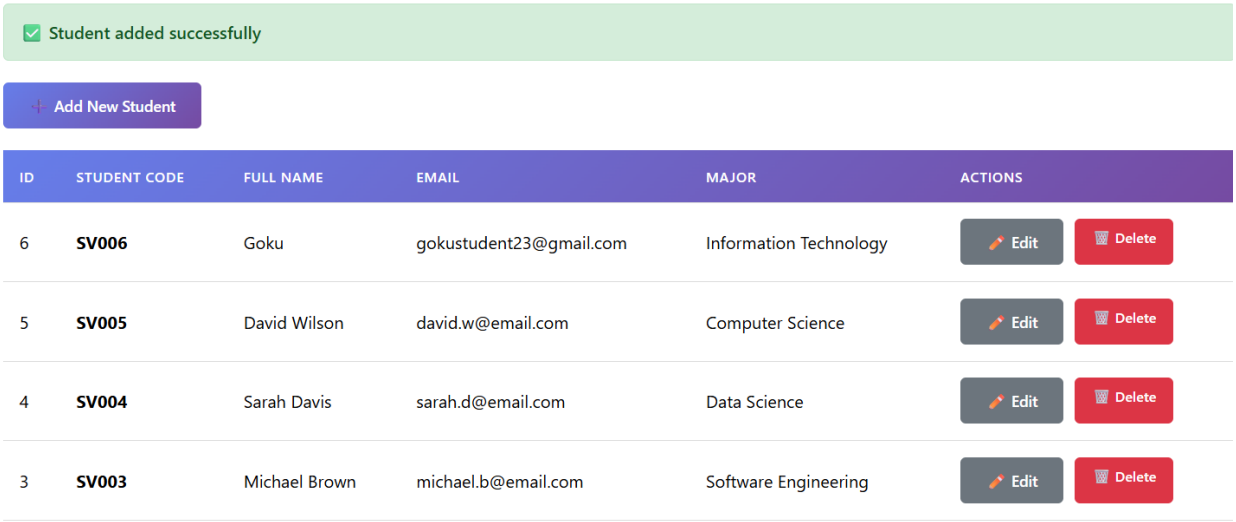
Exercise 2:

Test Cases:

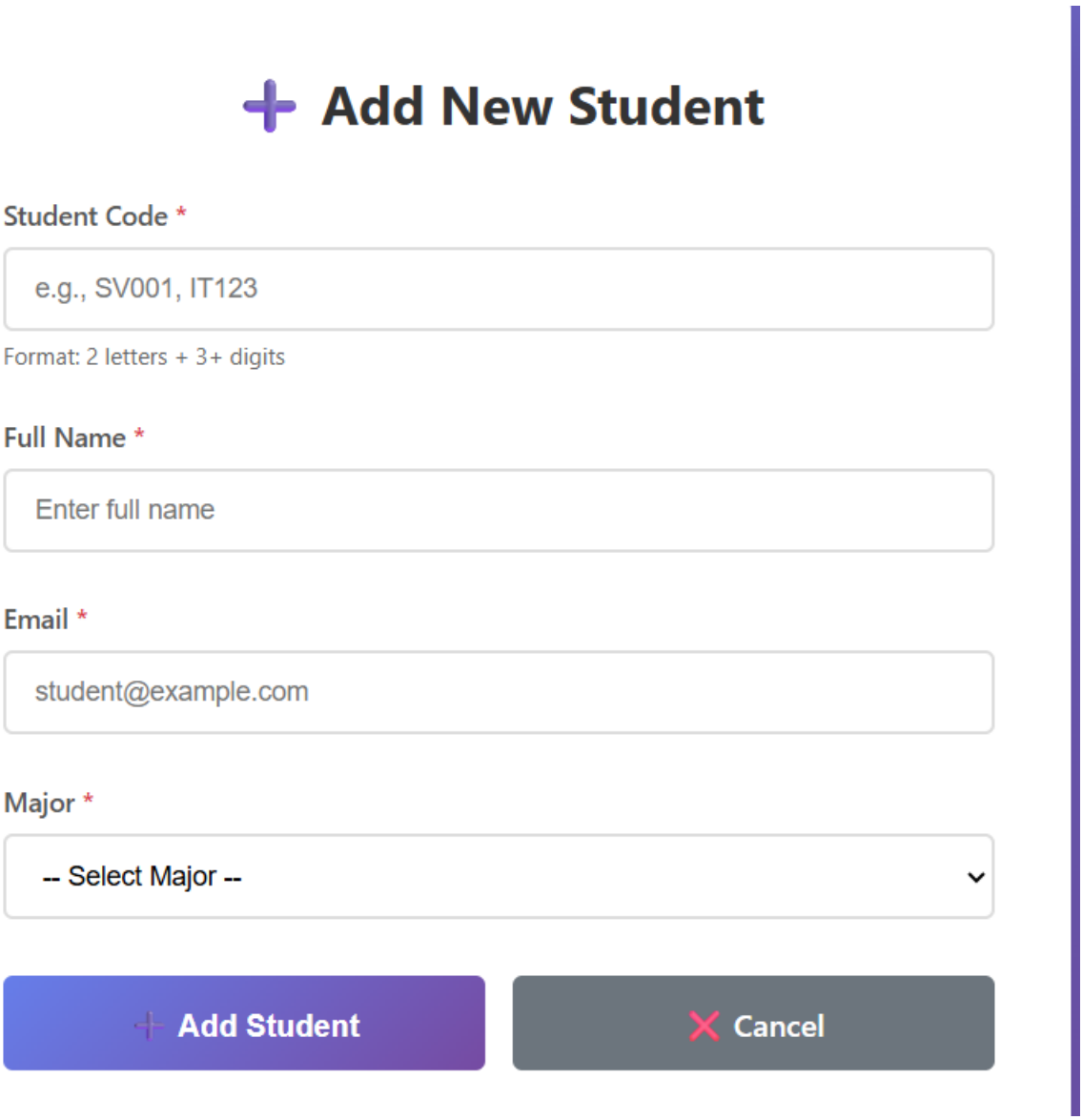
Test case 1:



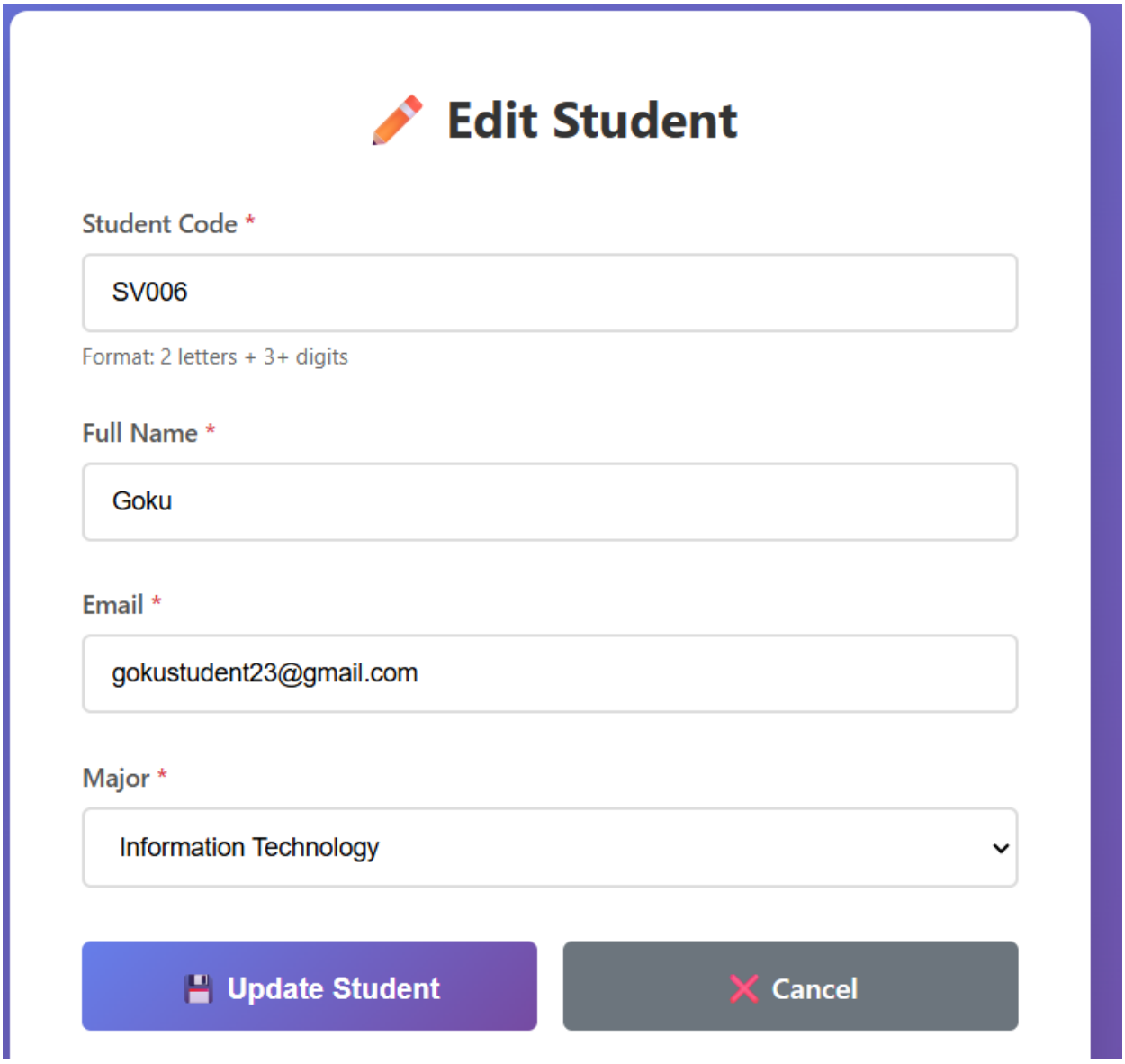
Test case 2: insert student



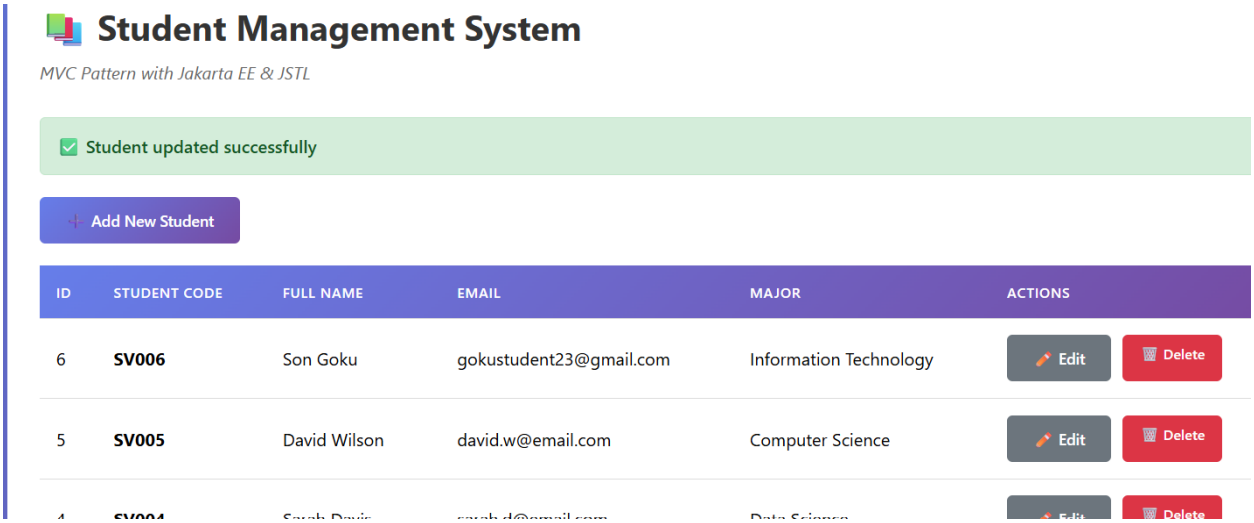
Test case 3: Show Add form



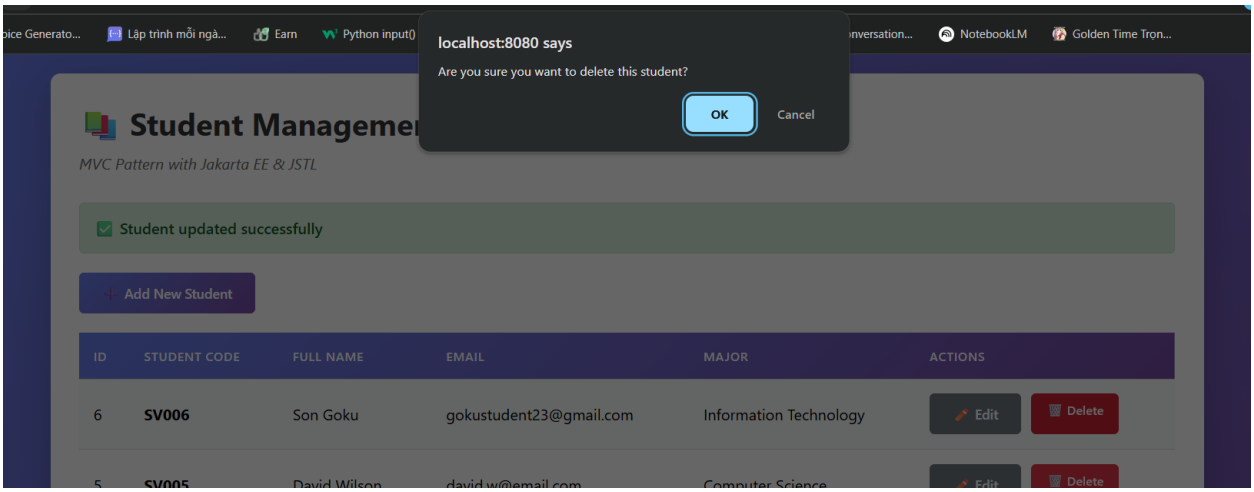
Test case 4: Display edit form

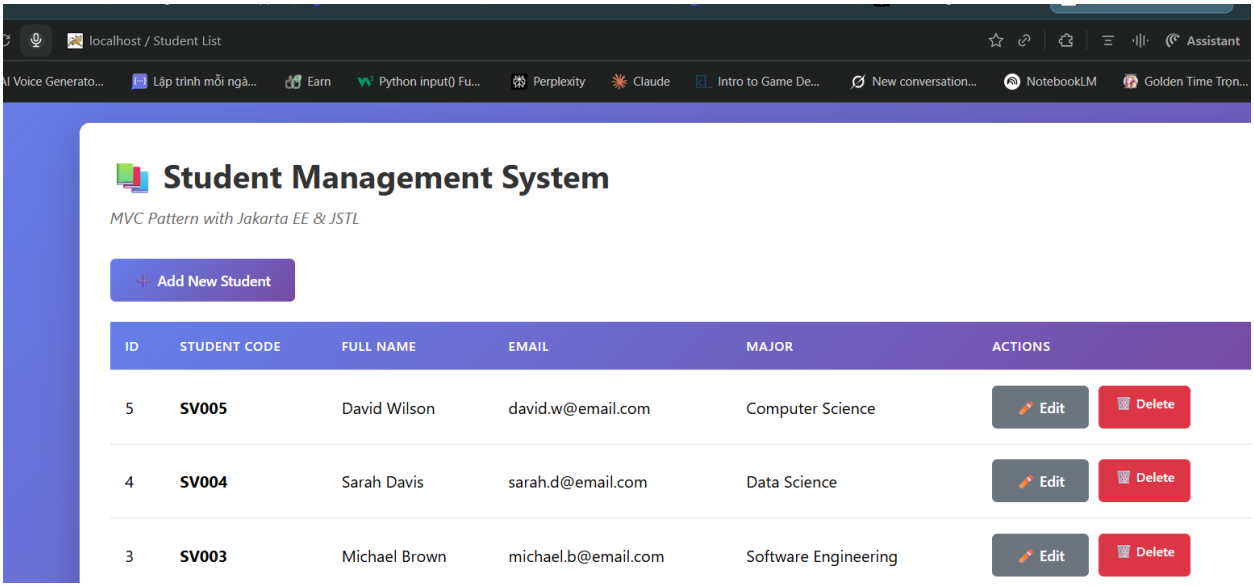


Test case 5: Update student



Test case 6: Delete student





Workflow:

The StudentController servlet is configured via the @WebServlet("/student") annotation to map HTTP requests directed to the /student path to this class. The doGet() method is invoked when GET requests are received, and the action parameter is extracted from the request. A switch statement routes requests based on the action value:

The listStudents() method retrieves all students from the DAO by calling studentDAO.getAllStudents(). The returned list is established as a request attribute and forwarded to student-list.jsp for rendering.

The showNewForm() method is invoked when a user requests a blank form to add a new student. The request is forwarded directly to student-form.jsp without setting any student attribute, causing the form to render in "add mode."

The showEditForm() method is triggered when an edit action is requested. The student id is extracted from the request, the student record is retrieved from the DAO using studentDAO.getStudentById(id), and the student object is established as a request attribute before forwarding to student-form.jsp. This approach permits the form to pre-populate with existing student data.

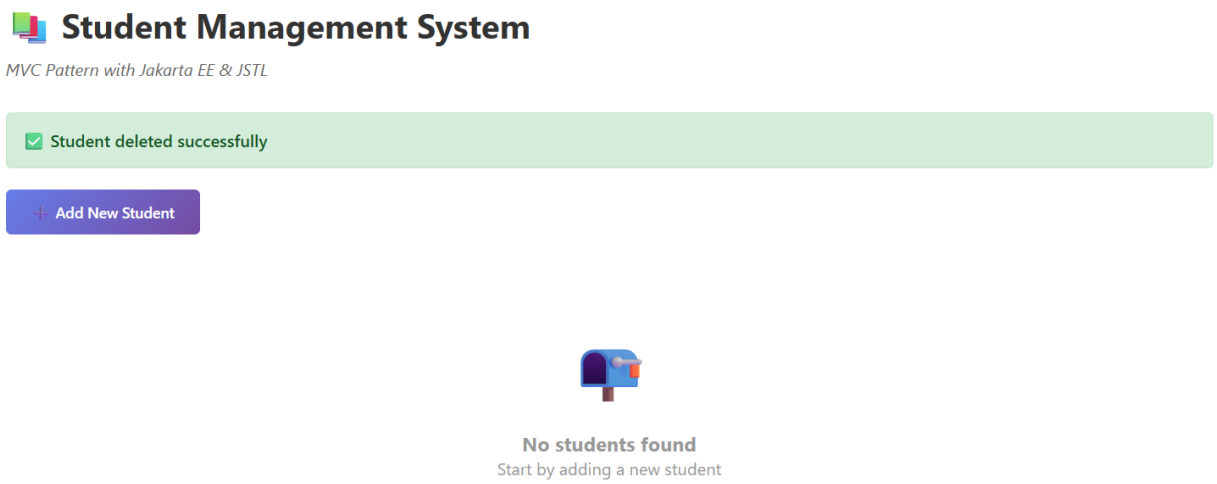
The deleteStudent() method is called when a delete action is requested. The student id is extracted and passed to the DAO's delete method. Upon deletion, the response is redirected to the list view with a success or failure message appended as a URL parameter.

The doPost() method handles form submissions from the view. Form data is extracted from the request, and a new Student object is constructed from the extracted values. For insert operations, insertStudent() is called, which invokes studentDAO.addStudent(newStudent) and redirects to the list with a confirmation message. For update operations, updateStudent() is called, which establishes the student's id, invokes studentDAO.updateStudent(student), and redirects to the list with an update confirmation message.

Exercise 3:

Test Cases:

Test case 1: Empty student



Workflow:

The JSTL taglib is declared at the beginning of the JSP file to enable tag-based rendering without scriptlets. Messages from the redirect URL are checked using <c:if test="${not empty param.message}">, and when present, success or error messages are presented to the user. The "Add New Student" button is rendered as a link to student?action=new, which triggers the servlet to display the blank form.

The student list is rendered using <c:forEach var="student" items="${students}"> to iterate over the students attribute that was established by the controller. Each student row is populated with data accessed through Expression Language: ${student.id}, ${student.studentCode}, ${student.fullName}, ${student.email}, and ${student.major}. Edit links are constructed with href="student?action=edit&id=${student.id}" to pass the student id back to the controller. Delete links are similarly constructed with href="student?action=delete&id=${student.id}".

An empty-state message is rendered using <c:choose> and <c:otherwise> tags to display when no students are present in the list.

The form title and button text are dynamized using <c:choose> tags that assess whether the student attribute is null. When the student attribute is not null, the form renders in "edit mode" with the title "Edit Student" and a submit button labeled "Update Student." When the student attribute is null, the form renders in "add mode" with the title "Add New Student" and a button labeled "Add Student."

Hidden input fields are configured to store the action (insert or update) and, for edit operations, the student id. The <c:if test="${student != null}"> tag is used to conditionally render the hidden id field exclusively during edit operations.

Form fields are pre-populated with student data using Expression Language: value="${student.studentCode}", value="${student.fullName}", and similar patterns. The student code field is configured to be readonly during edit mode using a conditional attribute: ${student != null ? 'readonly' : 'required'}.

Exercise 4:

Workflow:

Database connection parameters are established as static constants to facilitate connectivity with the MySQL server. The getConnection() method is used to retrieve a database connection, and exception handling is implemented to catch ClassNotFoundException if the MySQL driver fails to load.

The getAllStudents() method performs a SELECT query against the database. Each row returned is mapped to a Student object, and the aggregate result is returned as a List<Student> to the invoking code. The getStudentById(int) method is used to retrieve a single student record by id employing a prepared statement.
The addStudent(Student) method inserts a new record into the database containing the student's code, name, email, and major. The updateStudent(Student) method modifies an existing student record based on the provided id. The deleteStudent(int) method removes a student record from the database.