

## List students

The screenshot shows a web application titled "Student Management System" with a subtitle "MVC Pattern with Jakarta EE & JSTL". A purple header bar contains a "Add New Student" button. Below it is a table with columns: ID, STUDENT CODE, FULL NAME, EMAIL, MAJOR, and ACTIONS. The table lists six student entries:

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
27	SV021	Truong Huy Hoang	Hoang@email.com	CS	<button>Edit</button> <button>Delete</button>
26	SV020	ABC	john@email.com	CS	<button>Edit</button> <button>Delete</button>
24	SV013	QWERT	john@email.com	CS	<button>Edit</button> <button>Delete</button>
23	SV012	QWER	john@email.com	CS	<button>Edit</button> <button>Delete</button>
22	SV011	QWE	john@email.com	CS	<button>Edit</button> <button>Delete</button>
21	SV010	QW	john@email.com	CS	<button>Edit</button> <button>Delete</button>

## Add student

The screenshot shows a modal dialog titled "+ Add New Student". It contains four input fields with validation messages and a dropdown menu for selecting a major. At the bottom are two buttons: "Add Student" and "Cancel".

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

## Edit student

The screenshot shows a modal dialog titled "Edit Student". It contains four input fields: "Student Code" with value "SV021" (with a note "Format: 2 letters + 3+ digits"), "Full Name" with value "Truong Huy Hoang", "Email" with value "Hoang@email.com", and "Major" with a dropdown menu showing "Select Major". At the bottom are two buttons: "Update Student" (blue) and "Cancel" (grey).

## Messages display

The screenshot shows a "Student Management" application. A confirmation dialog box is displayed in the center, asking "Are you sure you want to delete this student?". It has "OK" and "Cancel" buttons. Below the dialog is a table of student data:

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
27	<b>SV021</b>	Truong Huy Hoang	Hoang@email.com	CS	<button>Edit</button> <button>Delete</button>
26	<b>SV020</b>	ABC	john@email.com	CS	<button>Edit</button> <button>Delete</button>
24	<b>SV013</b>	QWERT	john@email.com	CS	<button>Edit</button> <button>Delete</button>
23	<b>SV012</b>	QWER	john@email.com	CS	<button>Edit</button> <button>Delete</button>
22	<b>SV011</b>	QWE	john@email.com	CS	<button>Edit</button> <button>Delete</button>
21	<b>SV010</b>	QW	john@email.com	CS	<button>Edit</button> <button>Delete</button>

The URL at the bottom left is "localhost:8080/student-management-mvc/student?action=delete&id=26".

## Delete student

The screenshot shows a web application titled "Student Management System" with a subtitle "MVC Pattern with Jakarta EE & JSTL". A green success message at the top states "✓ Student deleted successfully". Below is a table of student data:

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
27	SV021	Truong Huy Hoang	Hoang@email.com	CS	<button>Edit</button> <button>Delete</button>
24	SV013	QWERT	john@email.com	CS	<button>Edit</button> <button>Delete</button>
23	SV012	QWER	john@email.com	CS	<button>Edit</button> <button>Delete</button>
22	SV011	QWE	john@email.com	CS	<button>Edit</button> <button>Delete</button>
21	SV010	QW	john@email.com	CS	<button>Edit</button> <button>Delete</button>

## Code flow

**Model:** Student.java (the data object) and StudentDAO.java (the data access logic).

**View:** student-list.jsp and student-form.jsp (the user interface).

**Controller:** StudentController.java (the servlet that manages all traffic).

### Scenario: Viewing the Student List (The Default Action)

1. Browser → GET /student?action=list (<http://localhost:8080/student-management-mvc/student?action=list>)
2. Controller (StudentController.java)
  - Server Tomcat routes this request to StudentController servlet (StudentController.java) @WebServlet("/student")
  - The doGet() method is executed.
  - It reads the action parameter. Since it's the first visit, action is null.
  - The if (action == null) check sets action = "list".
  - The switch statement calls the listStudents() method.
3. Controller calls DAO (listStudents() method)
  - The controller calls studentDAO.getAllStudents().
4. Model/DAO (StudentDAO.java) executes SQL
  - The getAllStudents() method connects to MySQL database.

- It executes the SQL query: `SELECT * FROM students ORDER BY id DESC.`
  - It loops through the `ResultSet`, creates a `List<Student>` (a list of Model objects), and returns it.
5. DAO returns to Controller
    - The `listStudents()` method receives the `List<Student>`.
    - It stores this list in the request object: `request.setAttribute("students", students);`.
  6. Controller forwards to View sets data as request attribute
    - The controller gets a `RequestDispatcher` for `/views/student-list.jsp`.
    - It uses `dispatcher.forward(request, response)`. This is a server-side forward
  7. View (`student-list.jsp`) displays using JSTL
    - The JSTL tag `<c:forEach var="student" items="${students}">` finds the `students` attribute in the request.
  8. Browser receives HTML response