VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING


**Web Application Development - IT093IU**

Homework Lab 5: SERVLET & MVC PATTERN

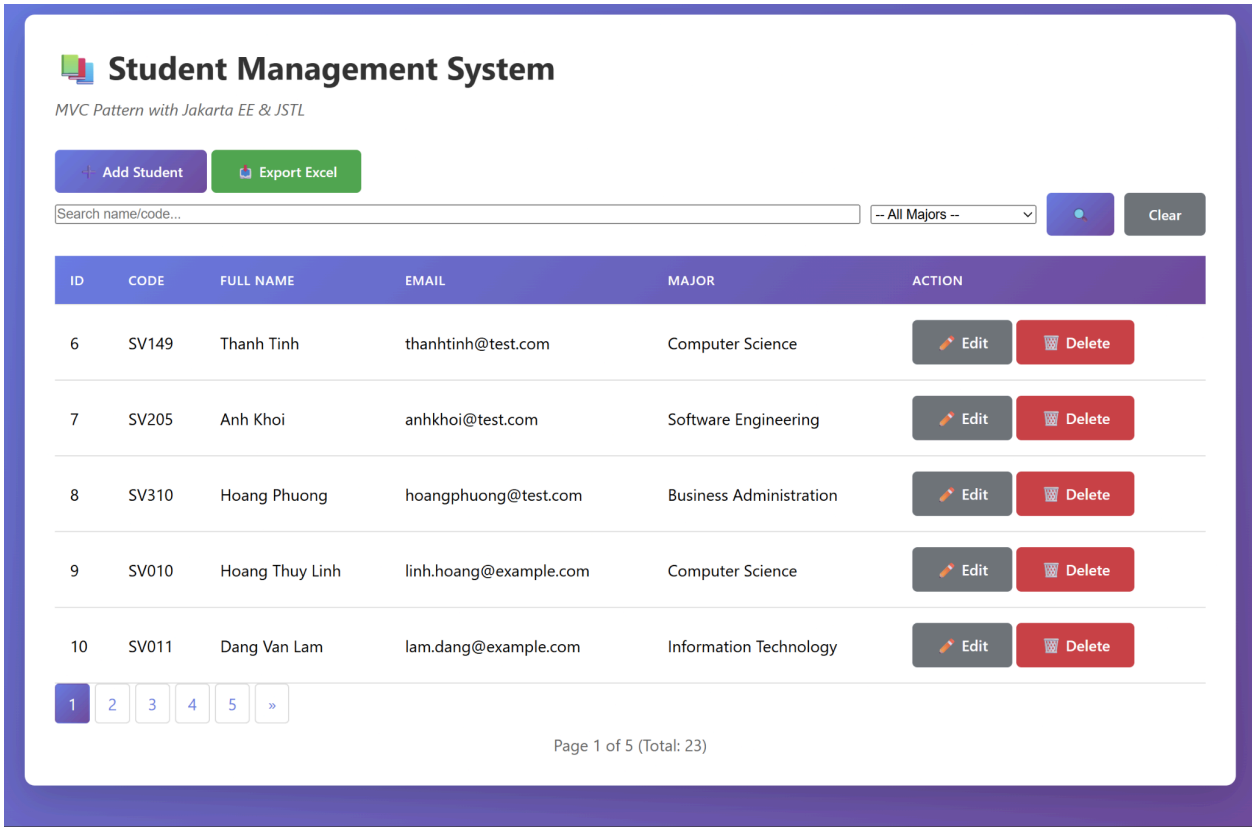
Nguyễn Vũ Thành Tính - ITCSIU23039

**Lab Instructor**: N.T.Nghia

## Github Link

*https://github.com/ngvuthtinh/Web_Lab/tree/main/Lab_5/Homework*

## 1. Home page with student list

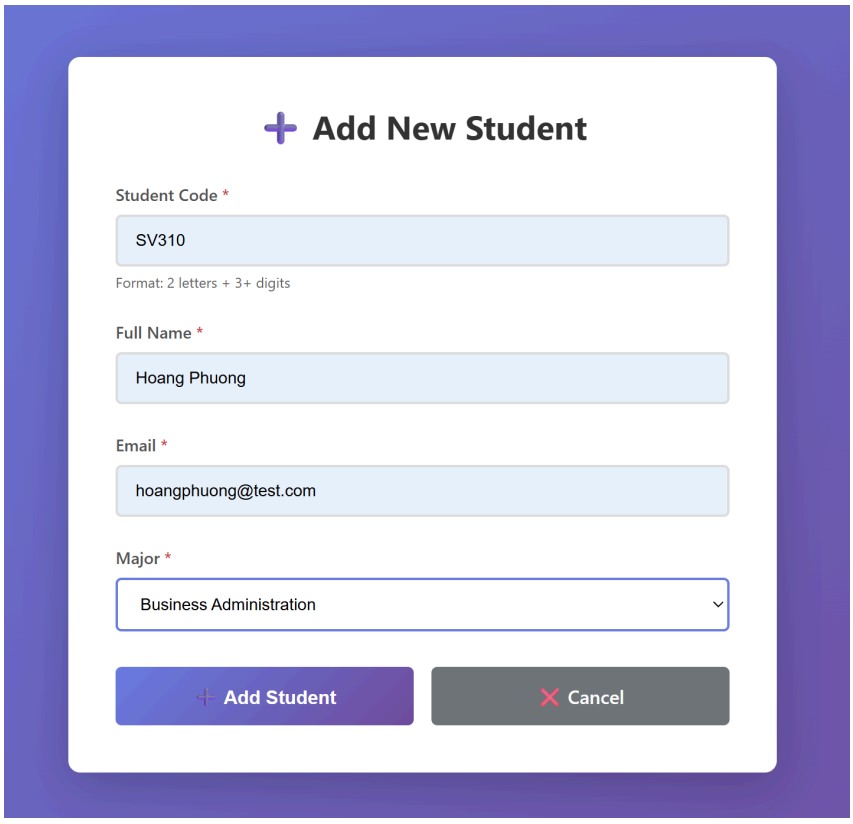



**Explanation:**

1. **Request:** The user navigates to `/student` or clicks a pagination link. Tomcat maps the request to `StudentController`.
2. **Controller (doGet):** The `doGet` method routes the request to the `handleList` method (the central handler for viewing data).
3. **Parameter Extraction:** The controller extracts parameters: `page` (for pagination), `keyword` (search), `major` (filter), and `sortBy` (sorting).
4. **Pagination Calculation:** It sets `recordsPerPage` (e.g., 5) and calculates the SQL offset: `offset = (page - 1) * recordsPerPage`.

5. **DAO Call:** The controller calls `studentDAO.getStudentsCombined(...)`. This "Master Method" builds a dynamic SQL query including `WHERE` clauses for filtering and `LIMIT/OFFSET` for pagination.
6. **Set Attributes:** The controller sets crucial attributes for the view: the `students` list, `totalPages`, `currentPage`, and preserves all filter parameters (e.g., `currentMajor`) to keep the UI state.
7. **Forward to JSP:** The request is forwarded to `/views/student-list.jsp`.
8. **Render View:** The JSP uses `<c:forEach>` to render the student table. It also generates dynamic pagination links (Previous/Next) that retain the current search and filter state.

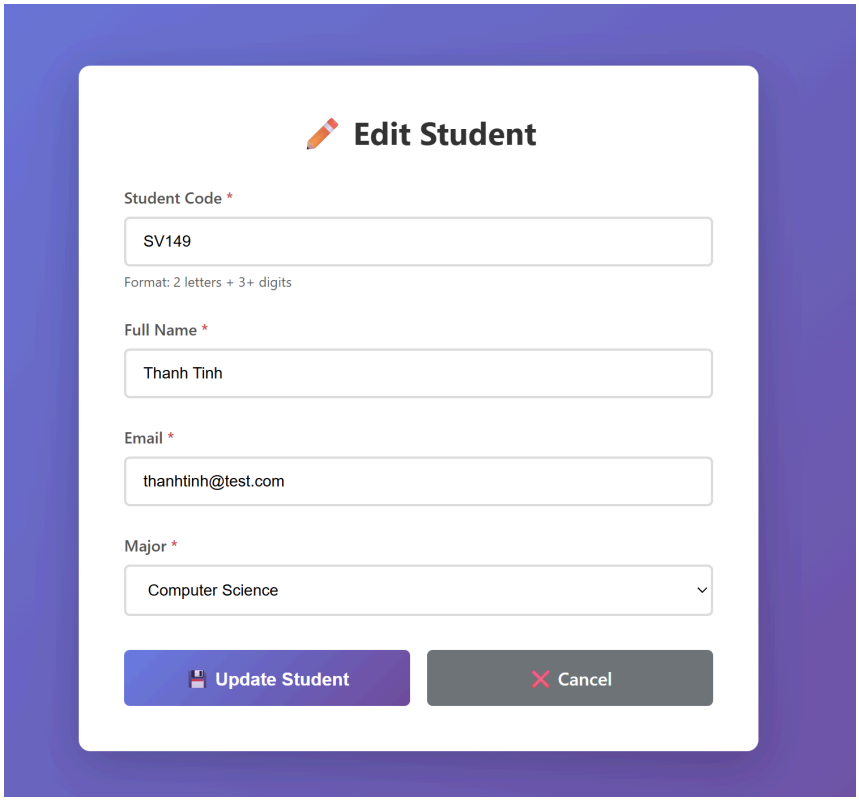
## 2. Add student form



**Explanation:**

1. **Request (GET):** User clicks "Add New Student". URL becomes `/student?action=new`.
2. **Controller (doGet):** The `StudentController` captures `action=new` and calls the `showNewForm` method.
3. **Forward:** The request is forwarded to `/views/student-form.jsp`. The page renders an empty form with the title "Add New Student".
4. **Submission (POST):** User fills data and submits. The form sends a POST request to `/student` with hidden field `action=insert`.
5. **Controller (doPost):** The controller switches on `action=insert` and calls `insertStudent`.
6. **Validation:** The `validateStudent` method checks input rules (Regex, required fields). If invalid, it forwards back to the form with error messages.
7. **DAO Call:** If valid, `studentDAO.addStudent(newStudent)` is called to execute the SQL `INSERT` command.
8. **Redirect:** Upon success, the response redirects to the list page using `response.sendRedirect("student?action=list")`.

## 3. Edit student form (pre-filled)

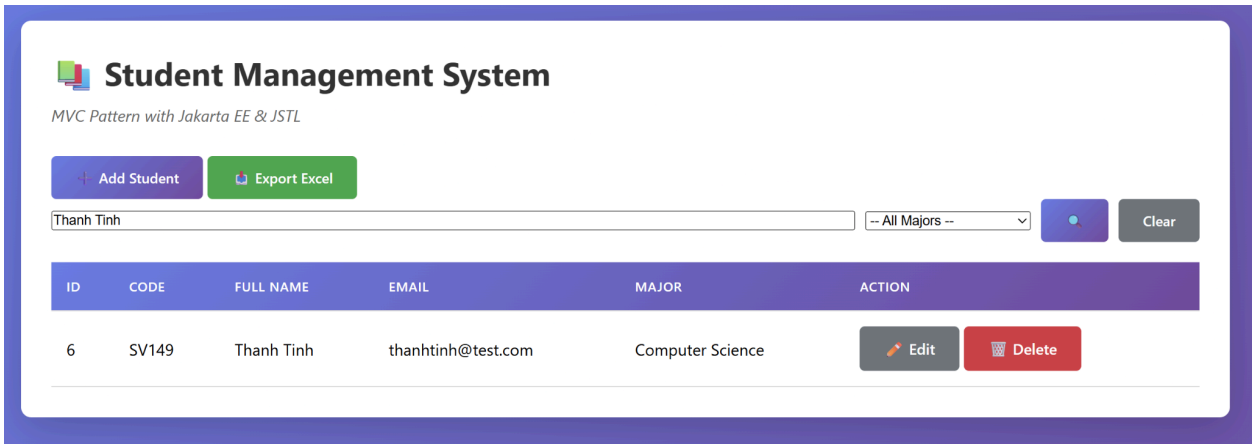



**Explanation:**

1. **Request (GET):** User clicks "Edit" on a student row. URL becomes `/student?action=edit&id=5`.
2. **Controller:** Captures `action=edit` and gets the `id` parameter from the request.
3. **DAO Call:** Calls `studentDAO.getStudentById(id)`. The DAO executes `SELECT * FROM students WHERE id = ?` and returns a specific `Student` object.
4. **Set Attribute:** The found student object is attached to the request: `request.setAttribute("student", existingStudent)`.
5. **Render View:** Forwards to `student-form.jsp`. The JSP checks if `${student}` exists and pre-fills all input fields (e.g., `value="${student.fullName}"`) for editing.

6. **Update (POST):** When submitted, `action=update` triggers the `updateStudent` method in Controller, which calls DAO to run the SQL `UPDATE` command.

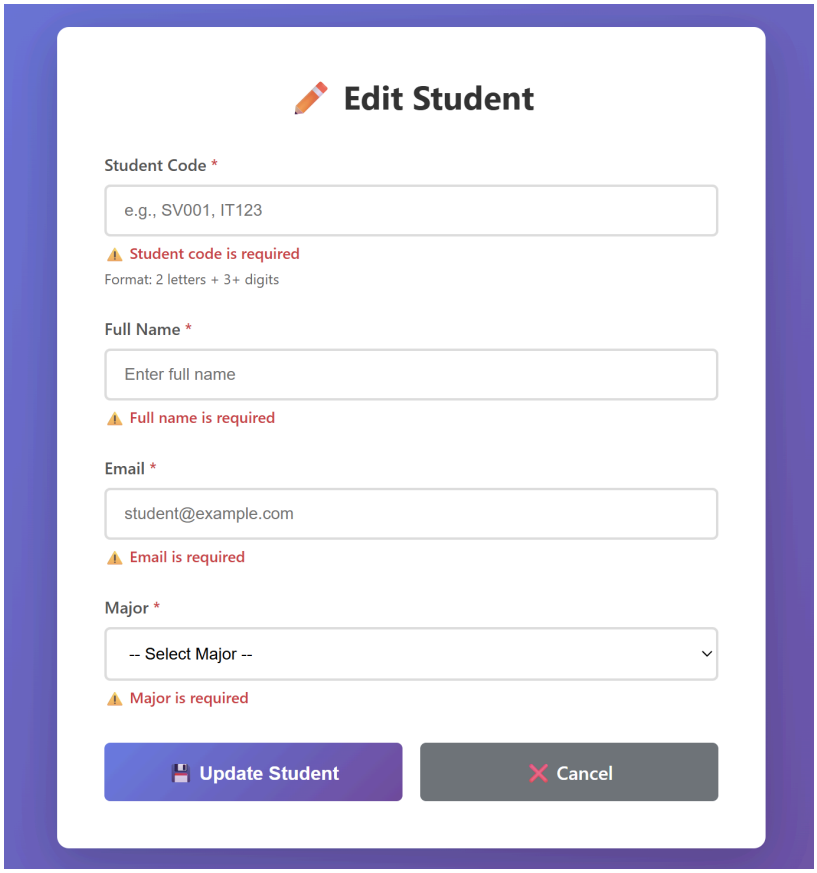
## 4. Search results




**Explanation:**

1. **Request:** User types a keyword and clicks Search. URL becomes `/student?action=list&keyword=John`.
2. **Controller:** The `handleList` method retrieves the `keyword` parameter.
3. **DAO Call:** Calls `studentDAO.getStudentsCombined(...)`. The DAO dynamically builds a SQL query using `LIKE`: `WHERE full_name LIKE ? OR student_code LIKE ?`.
4. **Wildcards:** The keyword is wrapped with `%` (e.g., `%John%`) to allow partial matching.
5. **Set Attribute:** The filtered list is set as `students`. Crucially, the `keyword` is also set as an attribute to keep it inside the search box after reloading.
6. **Render View:** The JSP displays only the students matching the criteria.

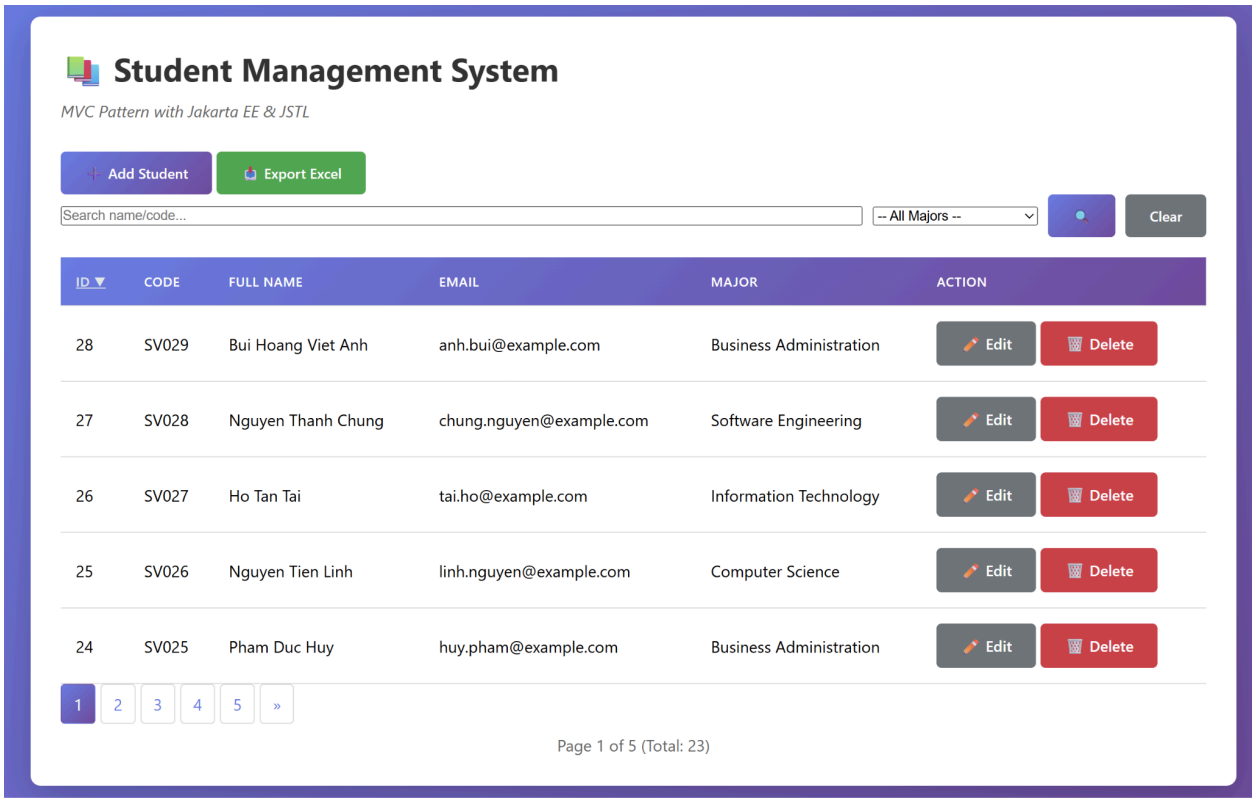
## 5. Validation errors




**Explanation:**

1. **Trigger:** Occurs inside `insertStudent` or `updateStudent` methods before calling the DAO.
2. **Validation Logic:** The `validateStudent()` method checks inputs. For example, it uses `matches("[A-Z]{2}[0-9]{3,}")` to validate the Student Code format.
3. **Error Handling:** If a rule is violated, an error message is added to the request (e.g., `request.setAttribute("errorCode", "Invalid format")`).
4. **Flow Interruption:** The controller stops processing and immediately **forwards** back to `student-form.jsp` (instead of redirecting).
5. **Feedback:** The JSP uses `<c:if test="${not empty errorCode}">` to display the red error message next to the specific field.

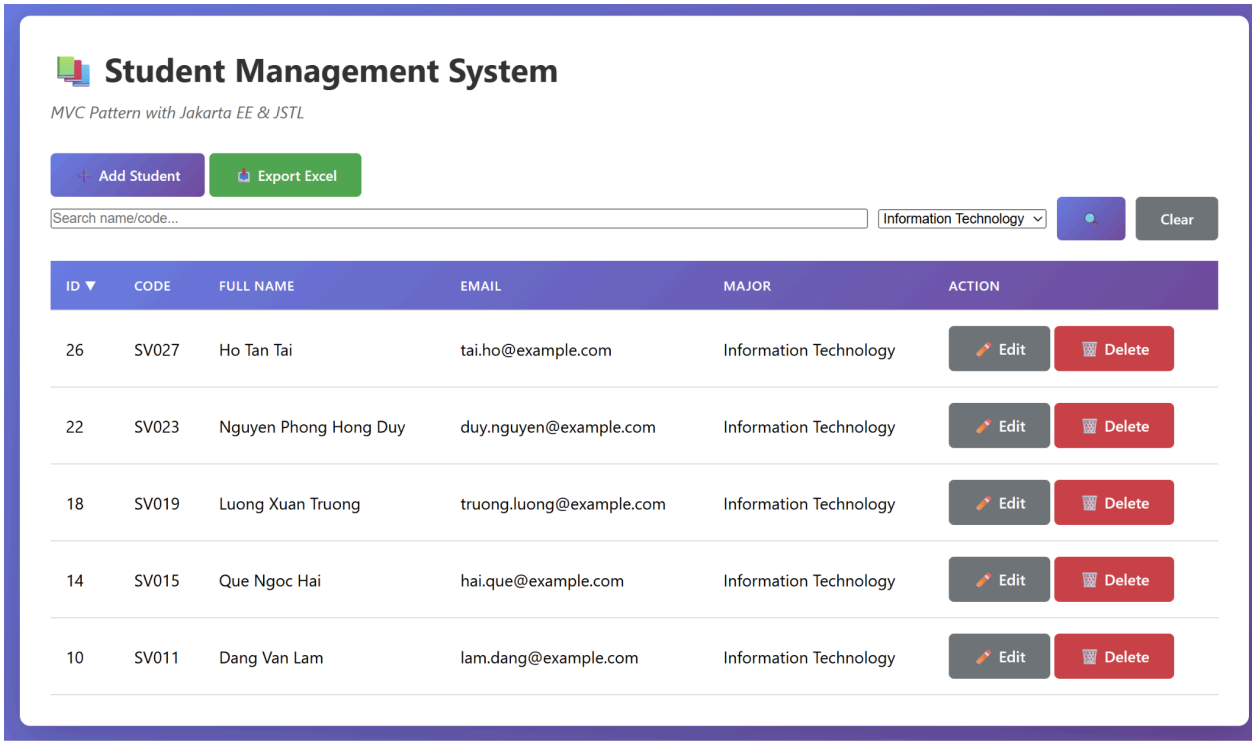
# 6. Sorted list




**Explanation:**

1. **Request:** User clicks a column header (e.g., "Full Name"). URL becomes `/student?...&sortBy=full_name&order=asc`.
2. **Controller:** The `handleList` method captures `sortBy` and `order` parameters.
3. **Security Check:** The DAO's `validateSortBy` method checks the column name against a whitelist to prevent SQL Injection.
4. **DAO Call:** The SQL query appends the sorting clause: `ORDER BY full_name ASC`.
5. **Render View:** The JSP renders the sorted list. It also toggles the link for the next click (if current is `asc`, the link generates `desc`).

# 7. Filtered list




**Explanation:**

1. **Request:** User selects "Computer Science" from the dropdown. URL becomes `/student?...&major=Computer Science`.
2. **Controller:** The `handleList` method captures the `major` parameter.
3. **DAO Call:** The DAO adds a `WHERE` clause to the SQL: `AND major = ?`.
4. **Combined Logic:** Since we use the "Master Method" in DAO, this filter works simultaneously with Search and Sort parameters.
5. **Render View:** The dropdown in JSP uses a conditional check (`${currentMajor == 'CS' ? 'selected' : ''}`) to show the active filter.