International University

School of Computer Science and Engineering

Web Application Development

Laboratory

IT093IU

Lab #4

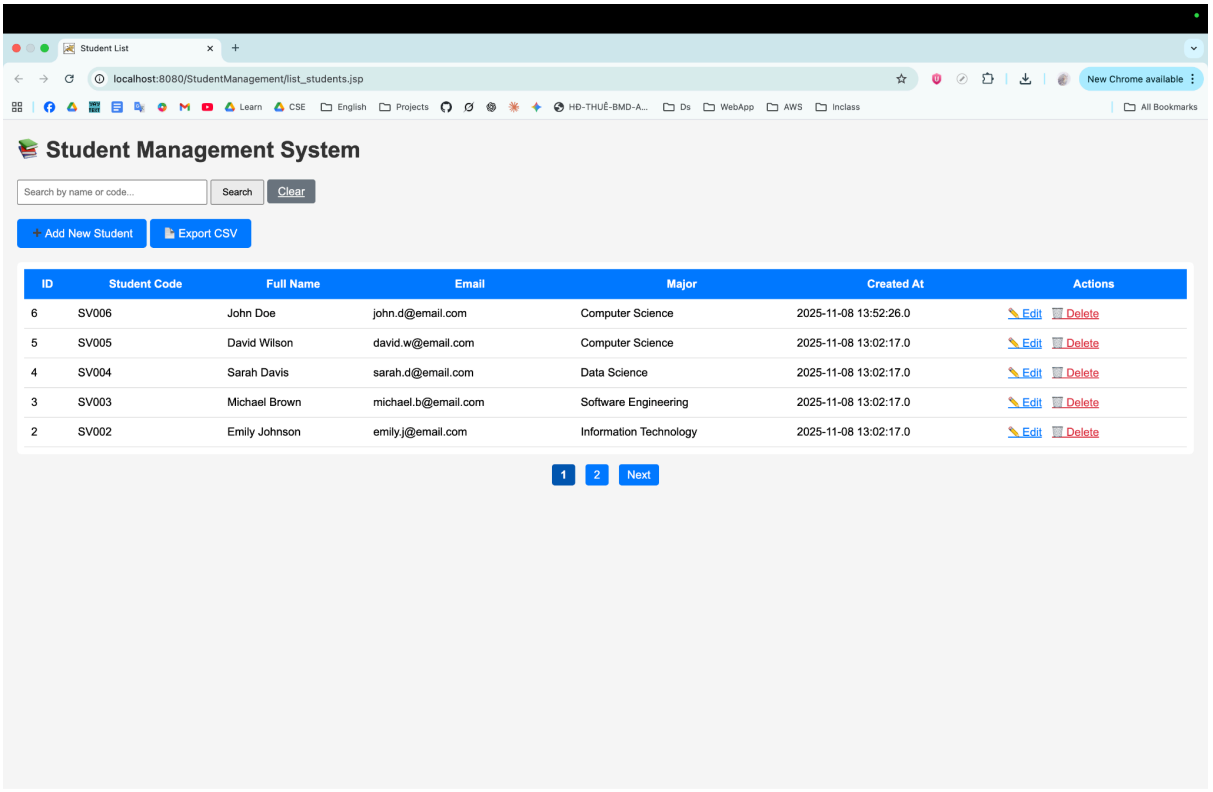
Submitted by

Nguyễn Hồng Ngọc Hân - ITCSIU22229

EXERCISE 5: SEARCH FUNCTIONALITY

5.1: Create Search Form.

5.2: Implement Search Logic.

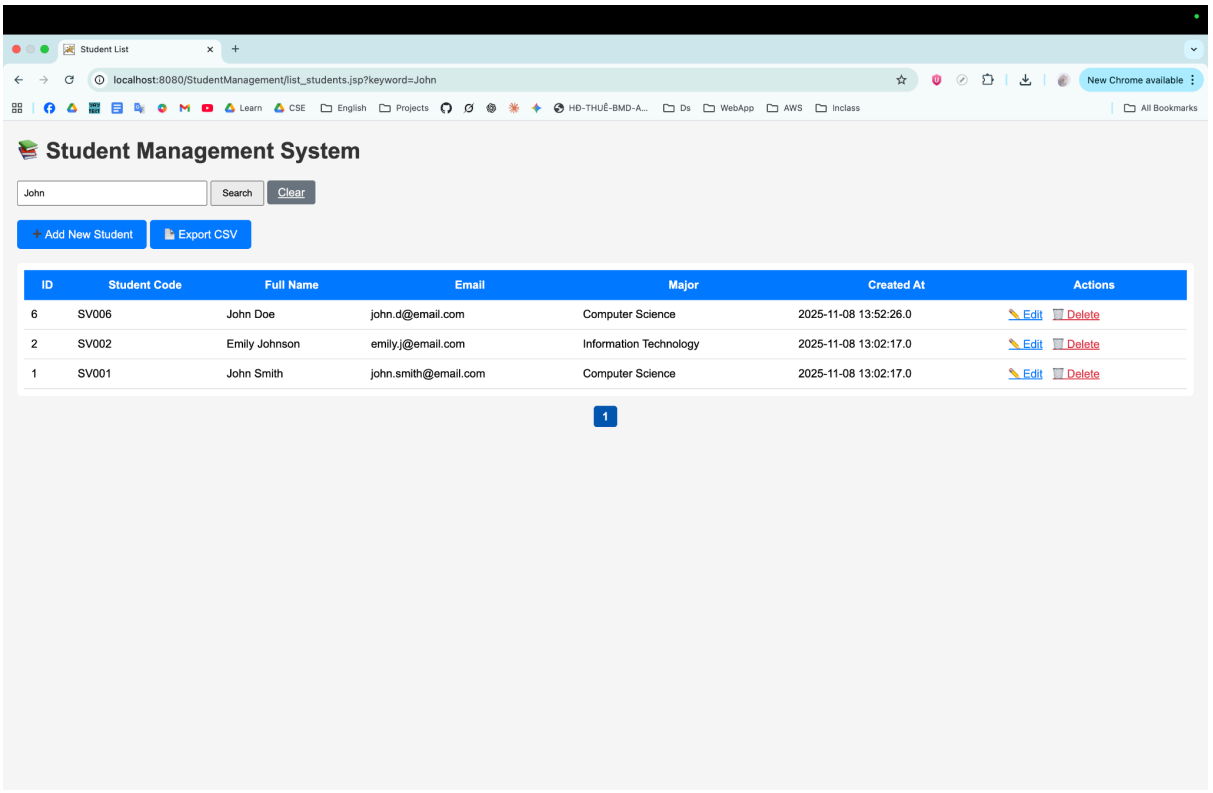


Explain:

This page lists all students and includes a search bar that lets users find records by name or student code. The form uses the GET method so the keyword appears in the URL. When submitted, the server reads the parameter and, if not empty, runs a query with the LIKE operator:

SELECT * FROM students WHERE full_name LIKE ? OR student_code LIKE ? The query uses wildcards (%keyword%) with a PreparedStatement to prevent SQL injection. If no keyword is provided, it defaults to showing all records ordered by ID. Pagination with LIMIT and OFFSET ensures only 10 results per page, and the results appear in a responsive HTML table with edit, delete, and navigation links.

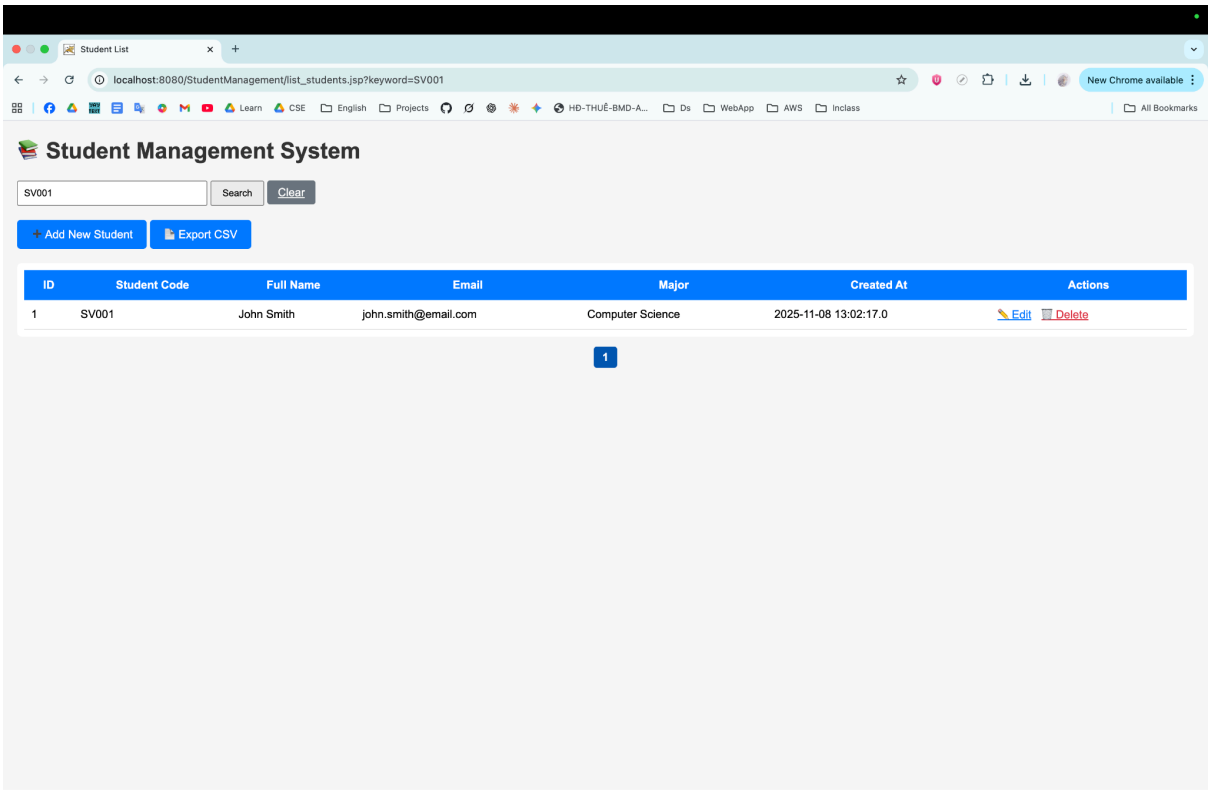
- **Test Cases 1:**



Explain:

In this scenario, the user types “John” into the search box and clicks Search. The form sends a GET request to list_students.jsp?keyword=John. On the server side, the code detects that the keyword is not null and executes the prepared SQL statement with LIKE '%John%' for both the full_name and student_code columns. The database returns all records where the name contains “John,” such as John Doe, Johnny Nguyen, or Peter Johnson. These records are then rendered in the HTML table, while pagination continues to function normally if multiple matches exist.

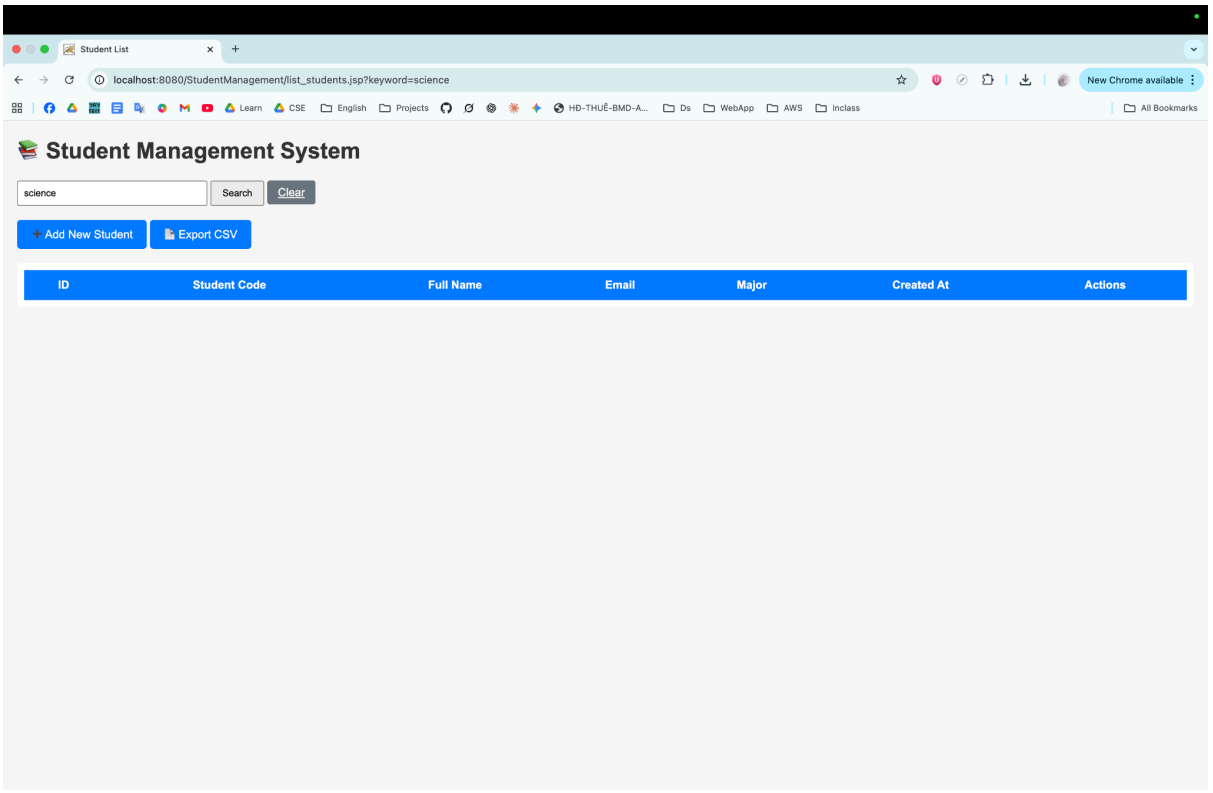
- **Test Cases 2:**



Explain:

Here, the user types “SV001” and presses Search. The same process occurs: the request URL becomes list_students.jsp?keyword=SV001. The SQL query runs with wildcards %SV001%, comparing against both student_code and full_name. Since student_code is unique, the query finds exactly one match, such as SV001 – Nguyen Van A. The result set contains that single record, which is displayed clearly in the table.

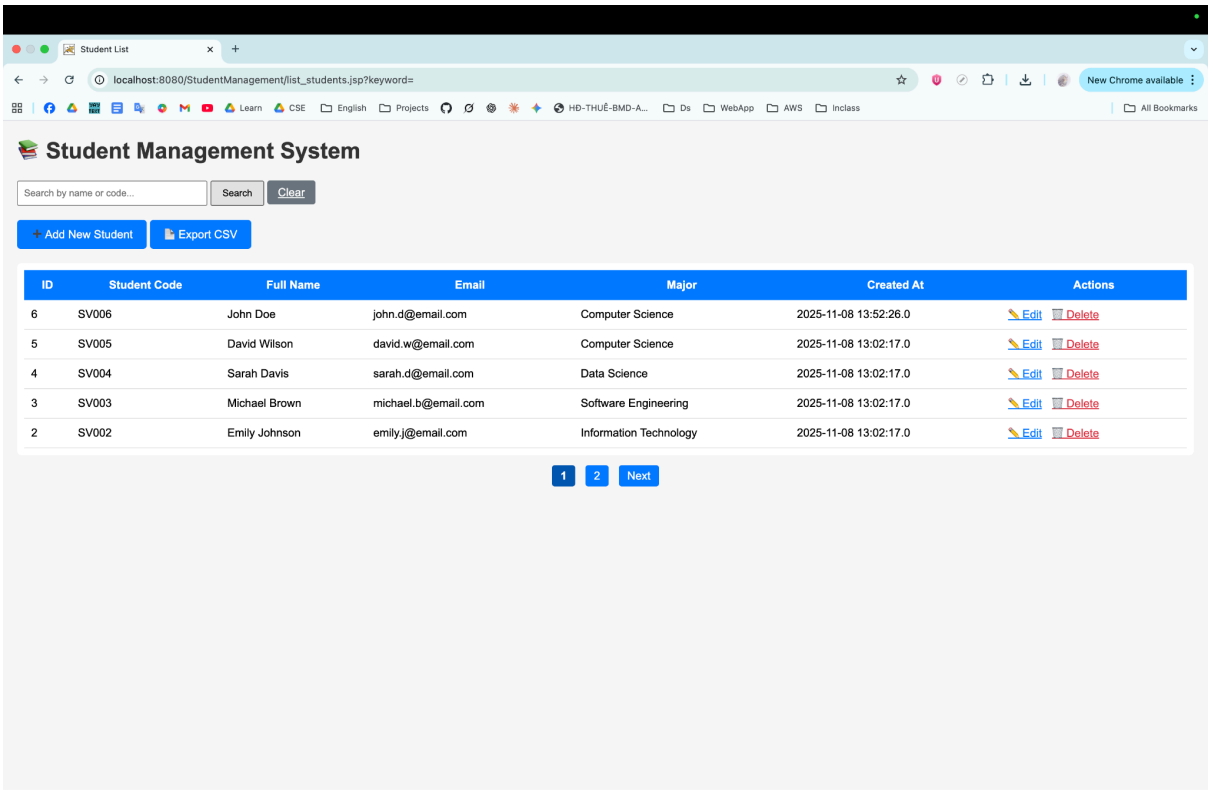
- **Test Cases 3:**



Explain:

In this case, the user enters “science” in the search box to look for students whose major includes “Computer Science” or “Data Science.” The request is sent as list_students.jsp?keyword=science. The server again runs the prepared SQL query with LIKE '%science%'. Since the keyword matches part of the major or full_name field (depending on how data is entered), all students enrolled in majors like “Computer Science,” “Information Science,” or “Data Science” are retrieved.

- **Test Cases 4:**



Explain:

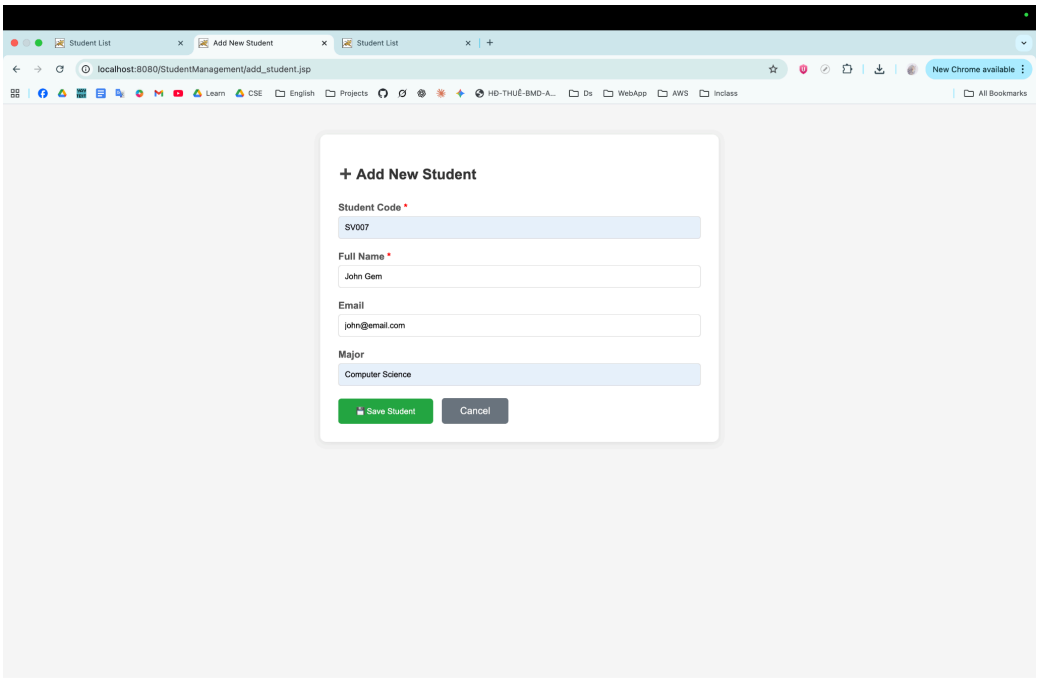
When the search field is left blank and the user clicks Search, the browser still sends a request to list_students.jsp but with an empty keyword value. The server checks: if (keyword != null && !keyword.trim().isEmpty()) Since this condition is false, the code executes the default SQL: SELECT * FROM students ORDER BY id DESC This query retrieves all students from the database, sorted from newest to oldest. Pagination remains active, displaying 10 records per page, and users can navigate using the page links at the bottom.

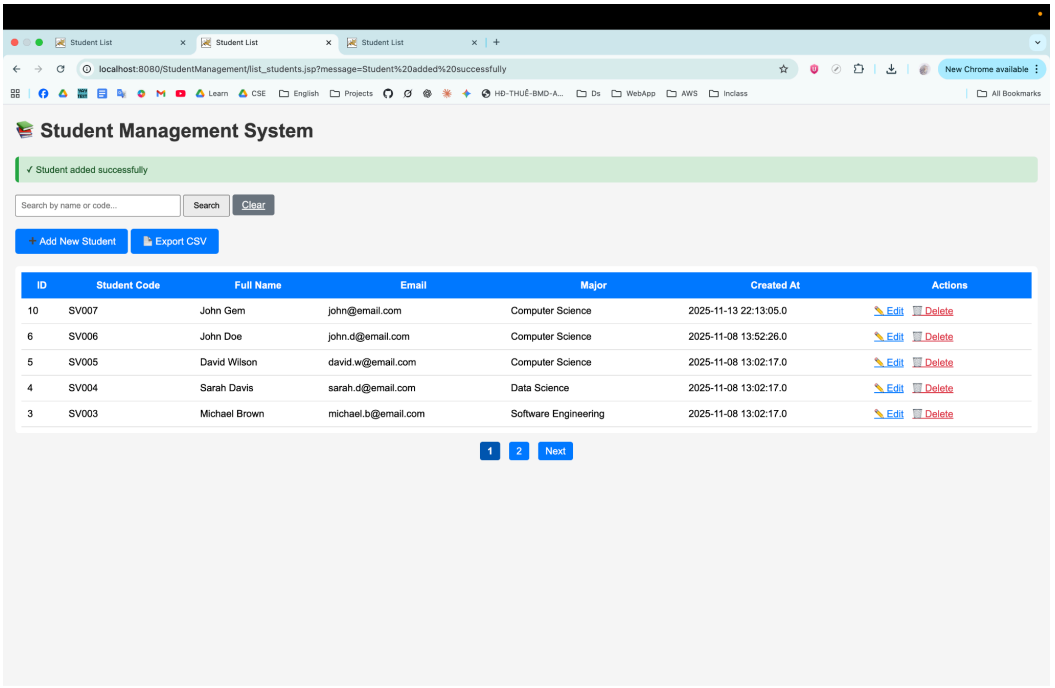
EXERCISE 6: VALIDATION ENHANCEMENT

6.1: Email Validation

Explain:

- **Test Cases 1:**

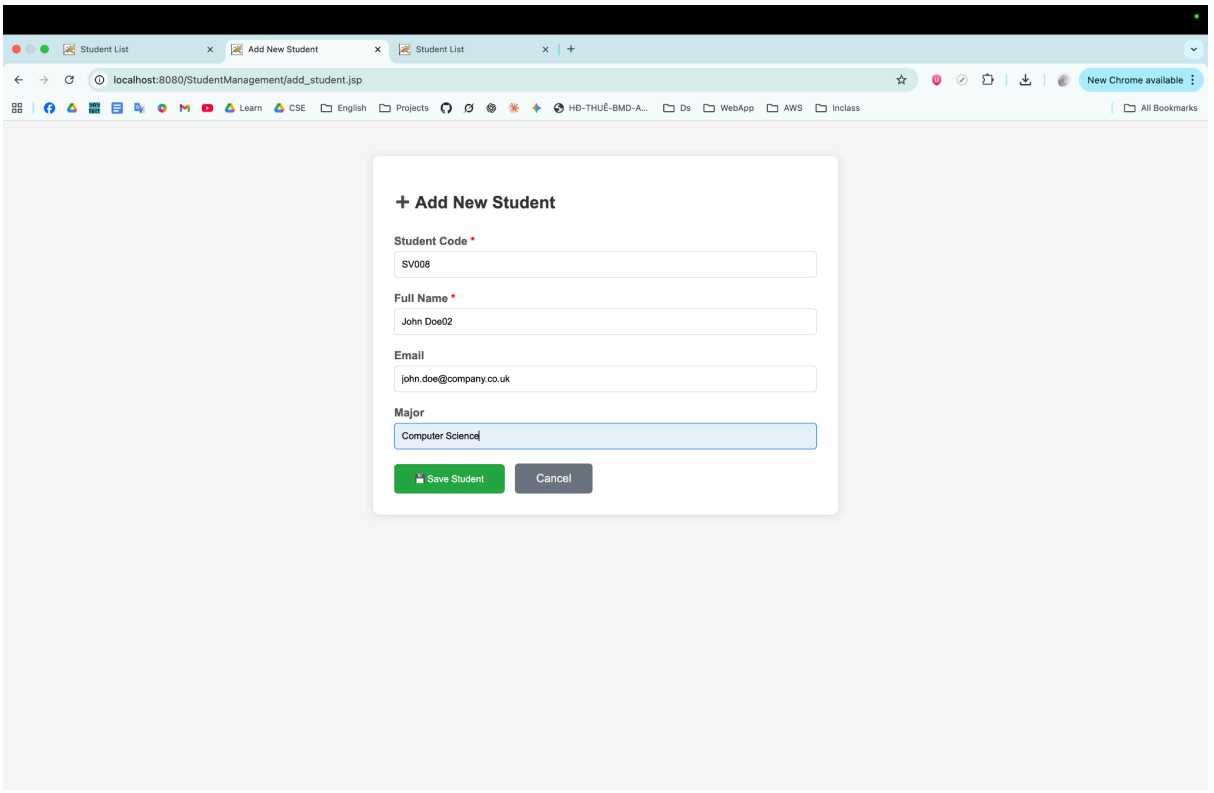


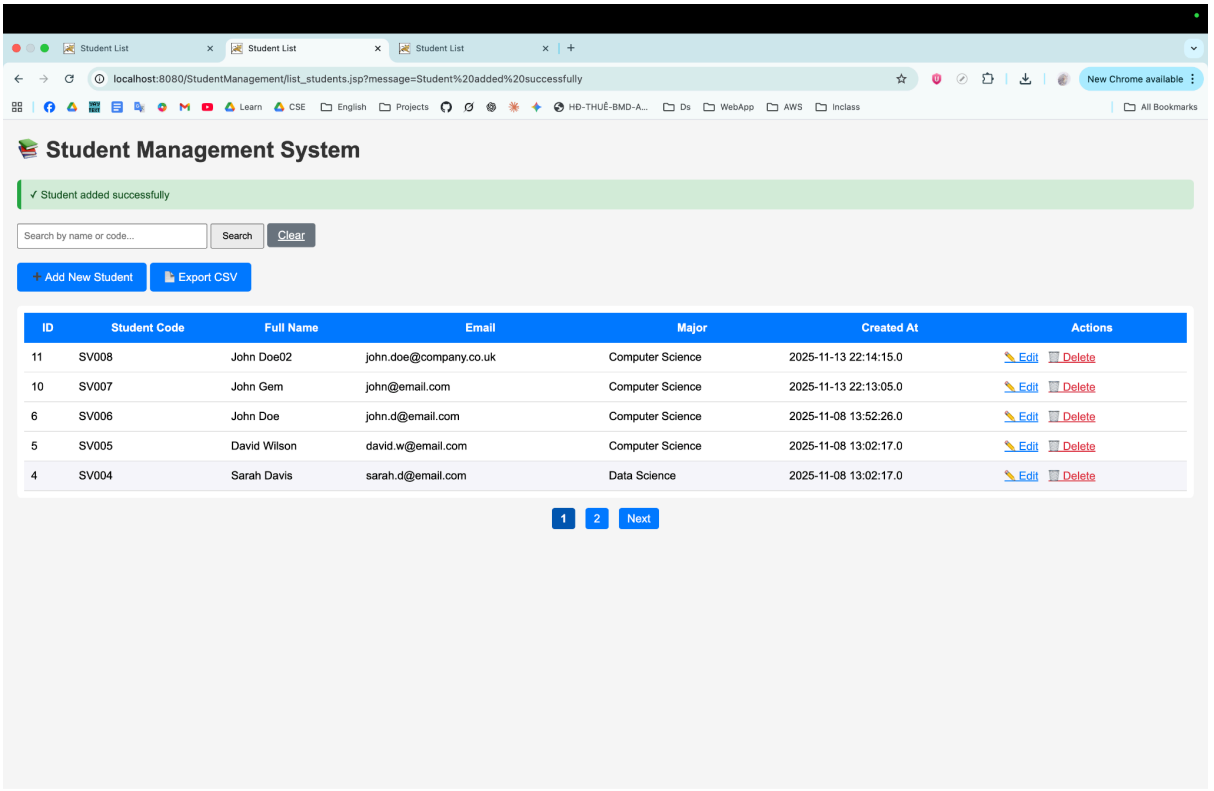


Explain:

When the user enters this email, it matches the regex pattern because it has valid characters before and after the “@” symbol. The system accepts the input, continues the insertion, and displays the message *“Student added successfully.”*

- **Test Cases 2:**

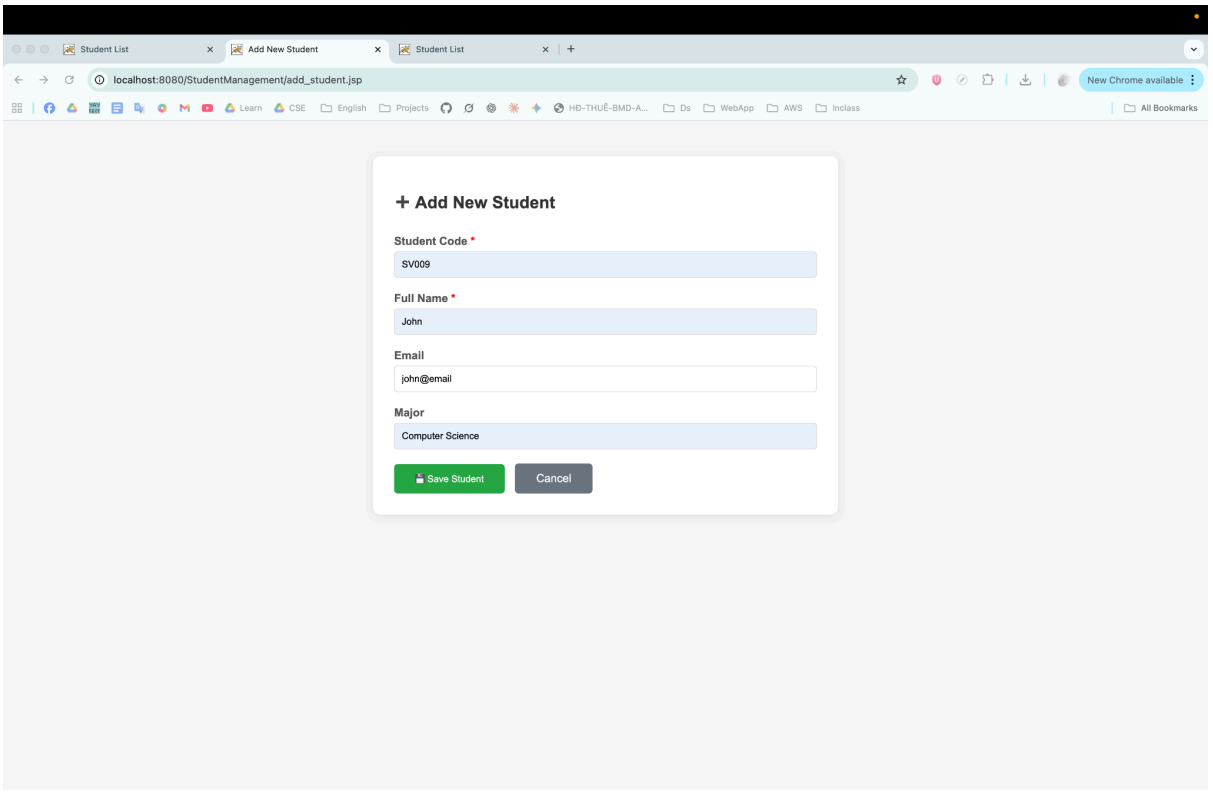


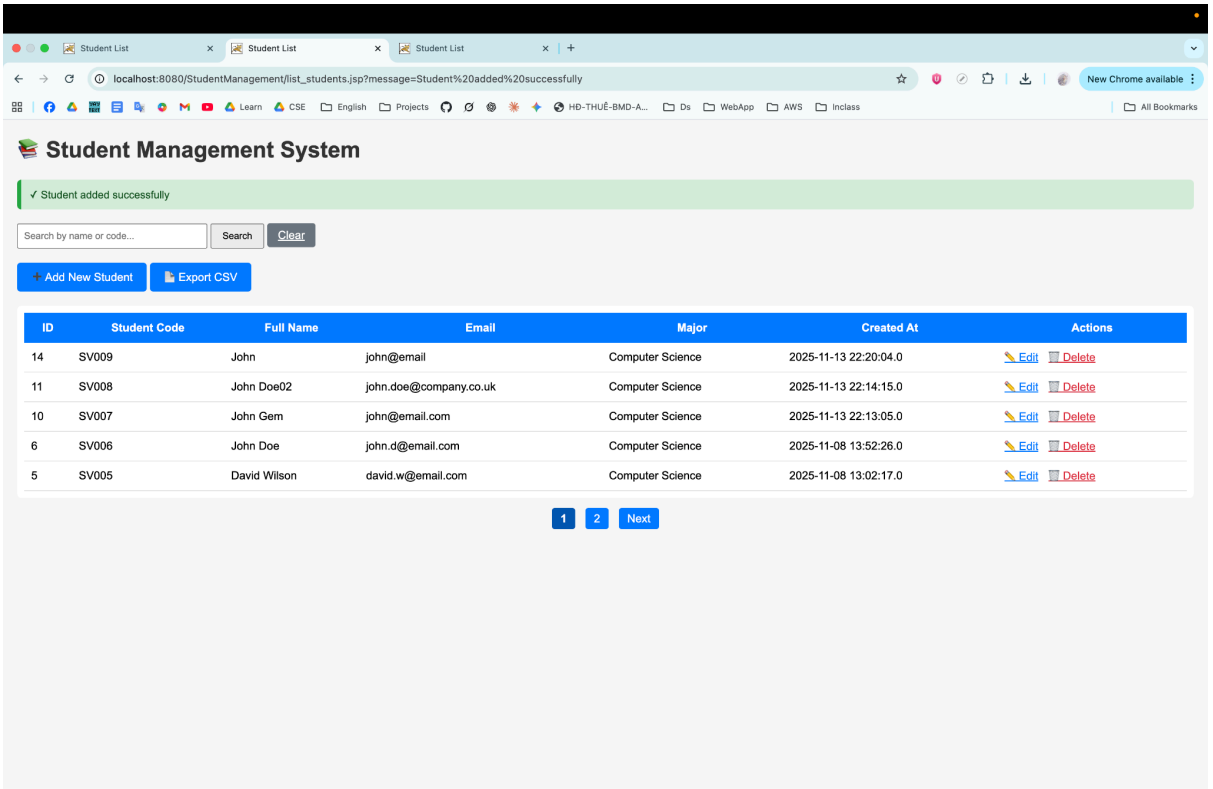


Explain:

This case includes multiple dots in the domain name but still fits the regex structure. The validation passes, and the student record is successfully inserted into the database with no errors.

- **Test Cases 3:**

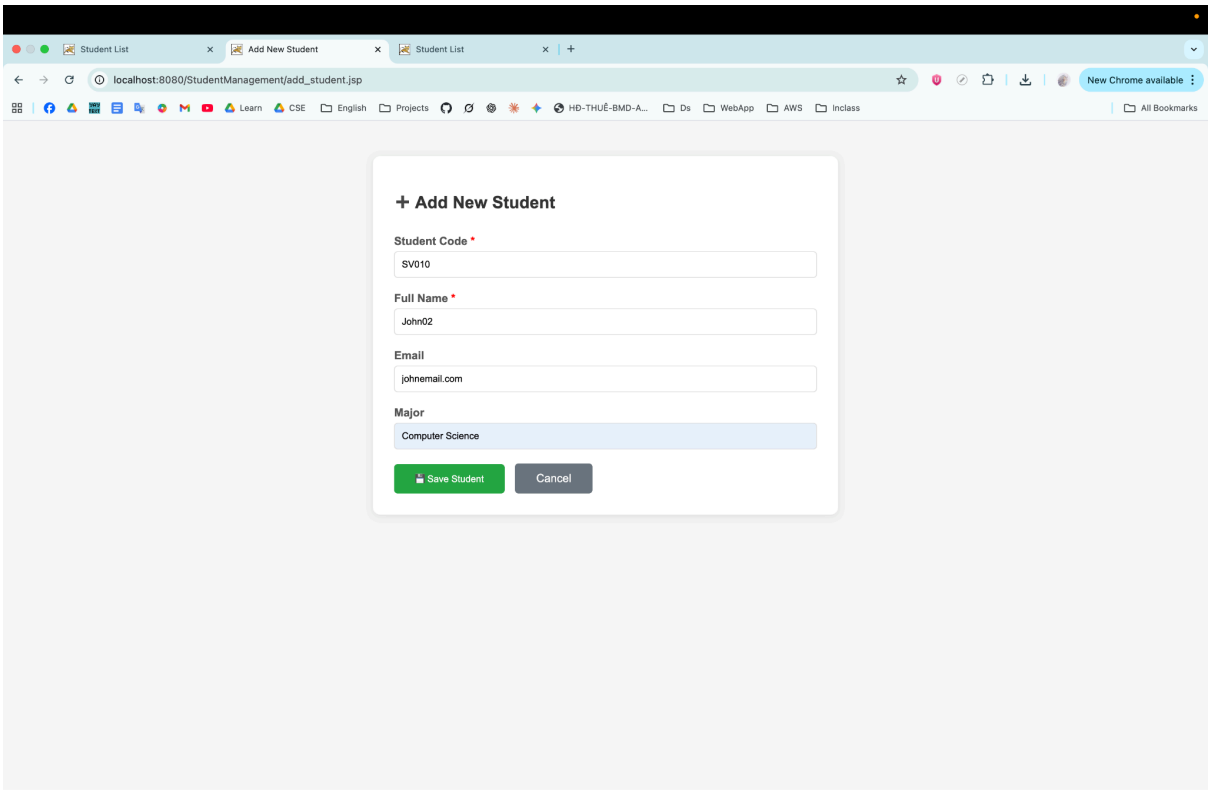


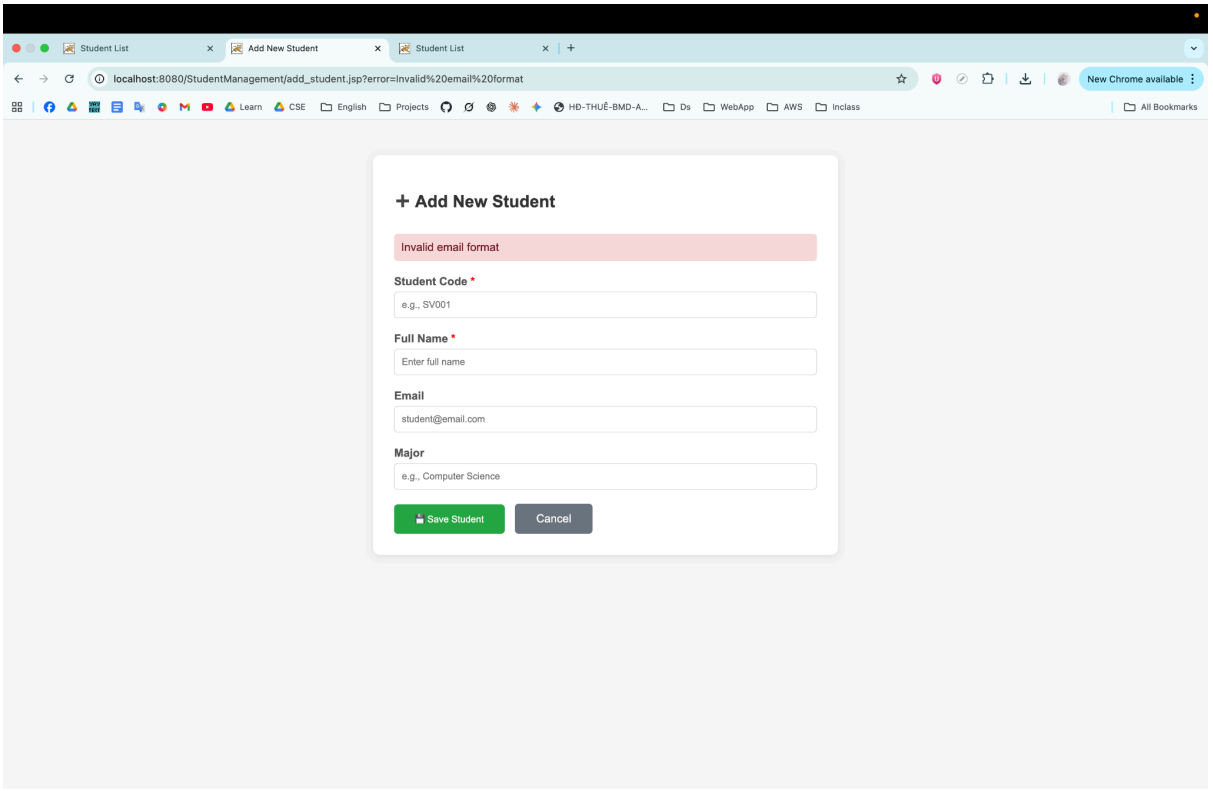


Explain:

Here, the input is missing the top-level domain (e.g., .com), so it sticks to match the regex “[A-Za-z0-9+_.-]+@(.+)$”.

- **Test Cases 4:**

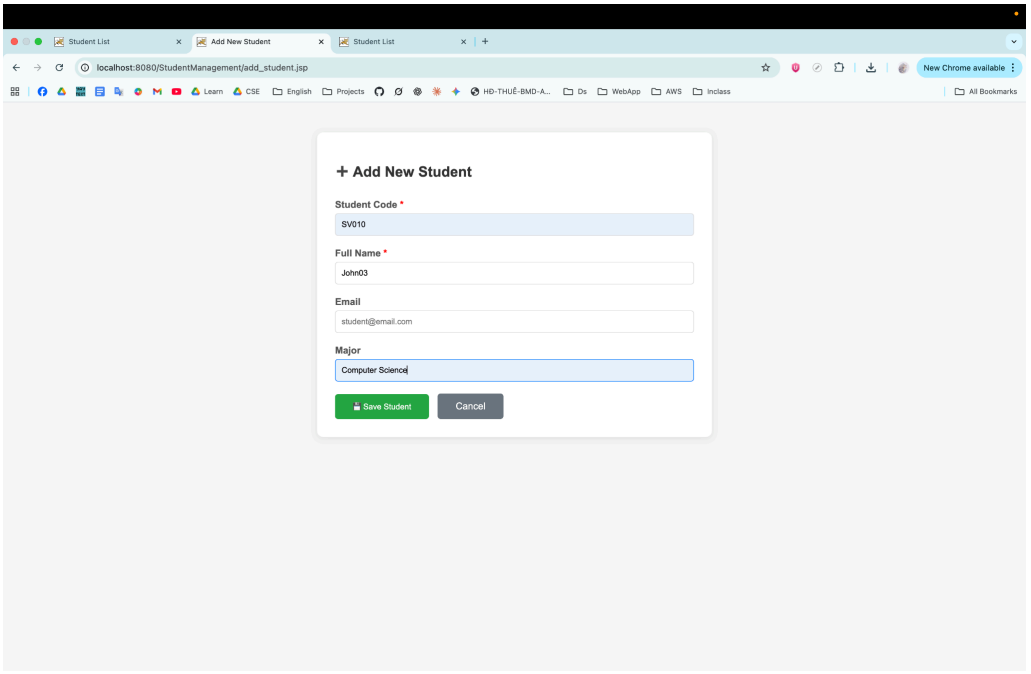


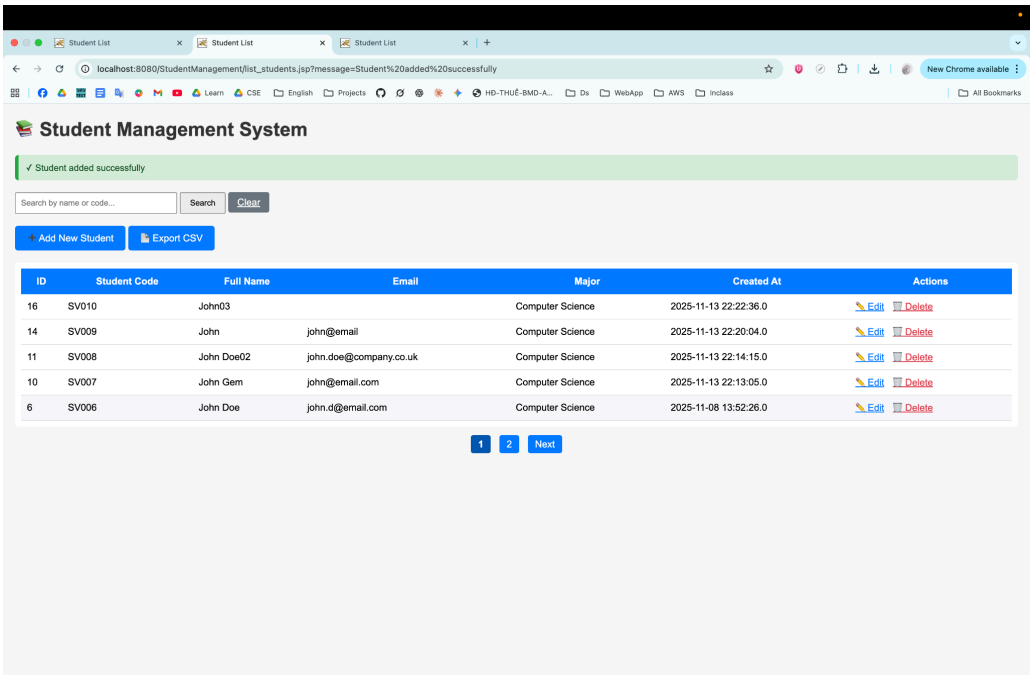


Explain:

Since the ”@” symbol is missing, this input clearly violates the email pattern. The validation fails, and the user is redirected with the same error, *“Invalid email format.”* The database remains unchanged.

- **Test Cases 5:**





Explain:

When the email field is left blank, the code checks the condition:
if (email != null && !email.trim().isEmpty()) { ... }
Because the field is empty, this block is skipped, meaning the system treats the email as optional. The student record is inserted successfully without an email value.

6.2: Student Code Pattern Validation

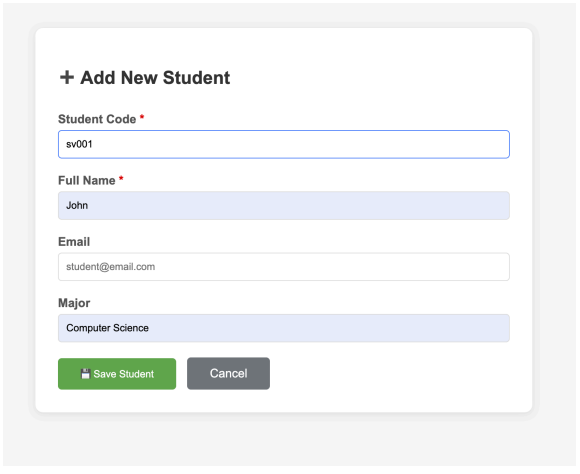
- **Test cases valid:**

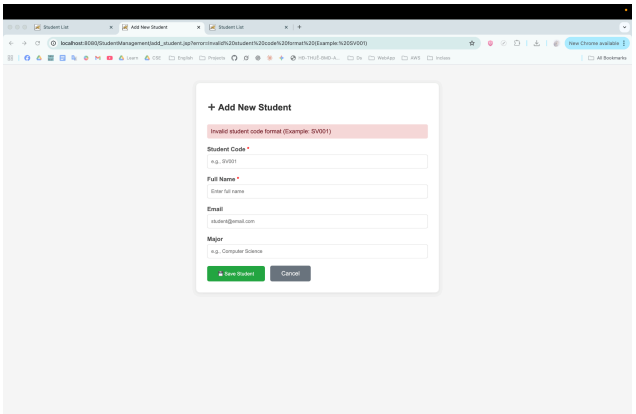
IT005
IT003
IT002
IT001
SV010
SV009
SV008
SV007
SV006
SV005

Explain:

When the user clicks Add, fills in the form with a valid code such as SV001, completes the other fields, and clicks Save, the system checks the pattern using .matches("[A-Z]{2}[0-9]{3,}"). Since the code follows the rule (two capital letters and at least three digits), validation passes. The system proceeds to insert the data into the database using the prepared SQL statement, and the user is redirected to list_students.jsp with a success message — “Student added successfully.”

- **Test cases invalid:**





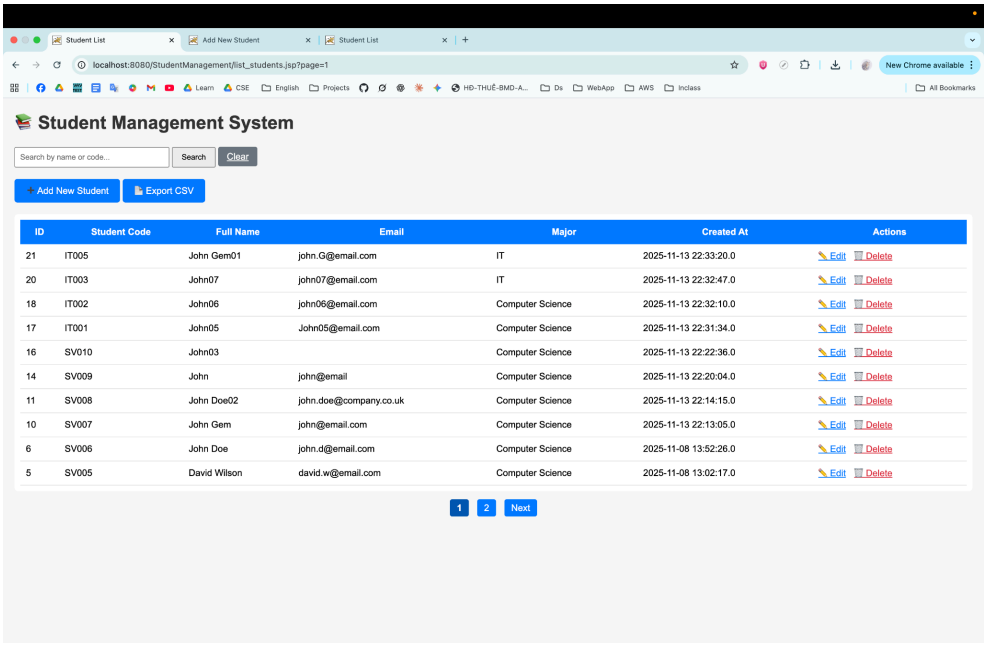
Explain:

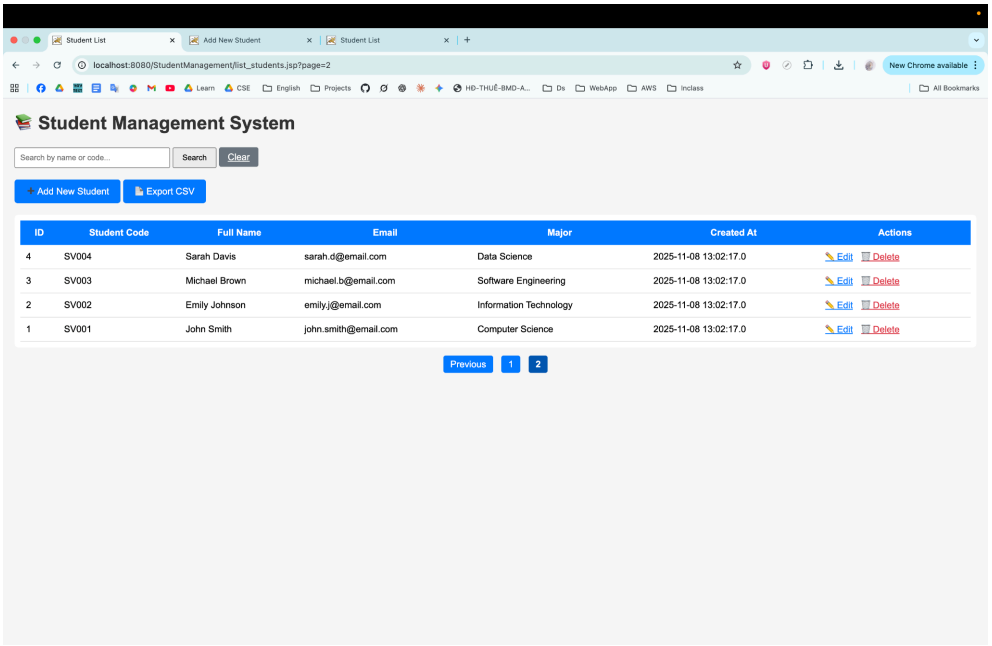
If the user fills the form with lowercase or too-short codes (like sv001, S001, or SV12) and clicks Save, the validation check fails because the input doesn’t match the required uppercase and digit pattern. The code executes the redirect command: response.sendRedirect("add_student.jsp?error=Invalid student code format (Example: SV001)");

The page reloads with a red error message, and no new record is inserted into the database.

EXERCISE 7: USER EXPERIENCE IMPROVEMENTS

7.1: Pagination





Explain:

After the user clicks Add New Student, fills in the form, and presses Save, the new student is inserted into the database. When redirected to list_students.jsp, the page now shows only 10 records per page. The pagination logic retrieves the page parameter from the URL using

String pageParam = request.getParameter("page");

and calculates the offset with

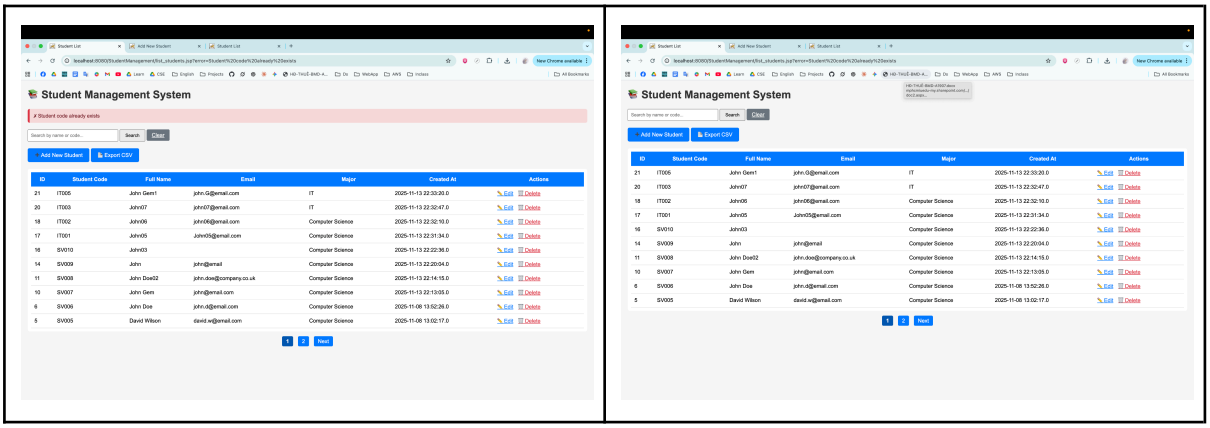
int offset = (currentPage - 1) * recordsPerPage;

The SQL query includes LIMIT ? OFFSET ? to fetch only the students for that page. The code also counts total records with a separate query (SELECT COUNT(*) FROM students) to calculate total pages, then displays numbered page links with Previous and Next buttons at the bottom.
When the user clicks on a page number (e.g., “2”), the browser sends a new GET request like list_students.jsp?page=2, and the system shows the next 10 students accordingly.

7.2: Improved UI/UX

a) Success/Error Message Styling

- **Add distinct colors (green for success, red for error)**
- **Add icons (✓ for success, ✗ for error)**
- **Auto-hide after 3 seconds (JavaScript)**



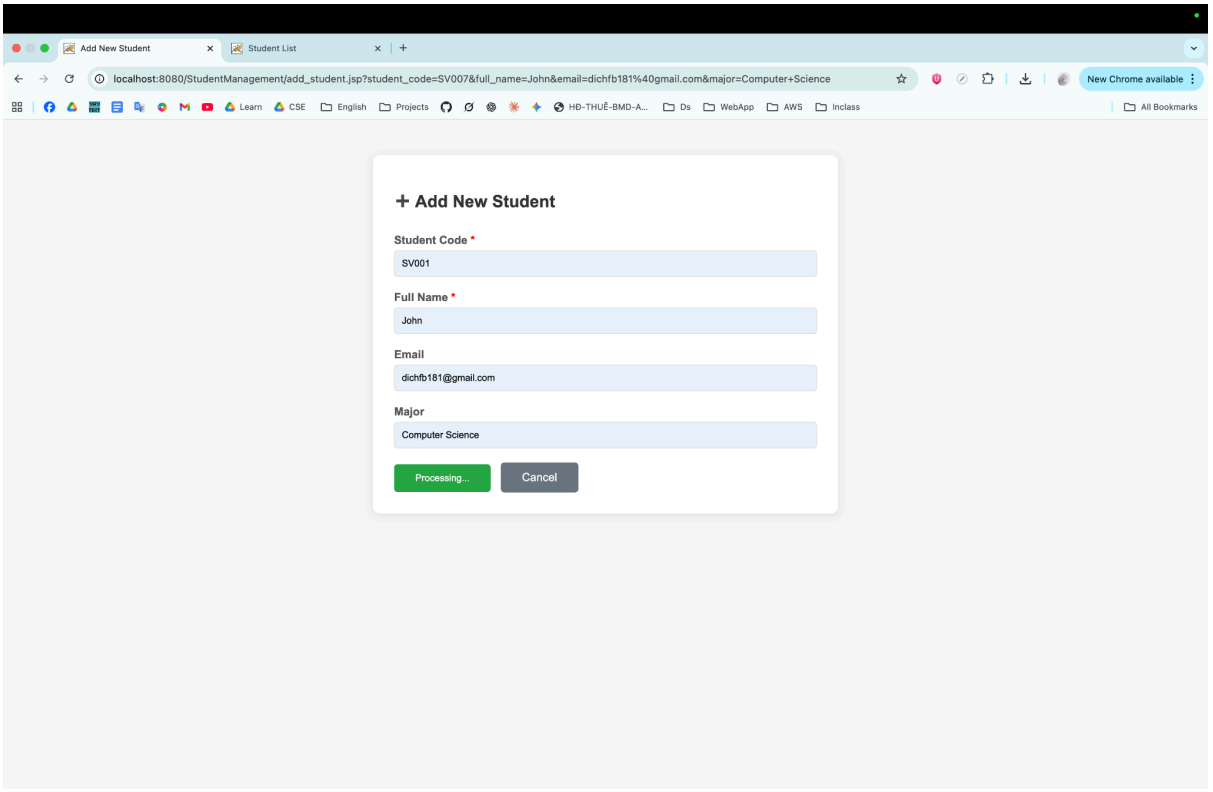
Explain:

When the user clicks Add New Student, fills in the form, and presses Save, the system now responds with smoother visual feedback. If the data is valid, the page redirects to list_students.jsp and shows a green success box with a ✓ icon and the message “Student added successfully.” If the data is invalid (for example, missing required fields or incorrect format), the system instead displays a red error box with a ✗ icon and an appropriate error message such as “Invalid email format.” In both cases, these messages automatically disappear after 3 seconds using JavaScript for a clean interface.

b) Loading States (2 points)

- **Disable submit button after clicking to prevent double submission**

- **Show "Processing…" text**

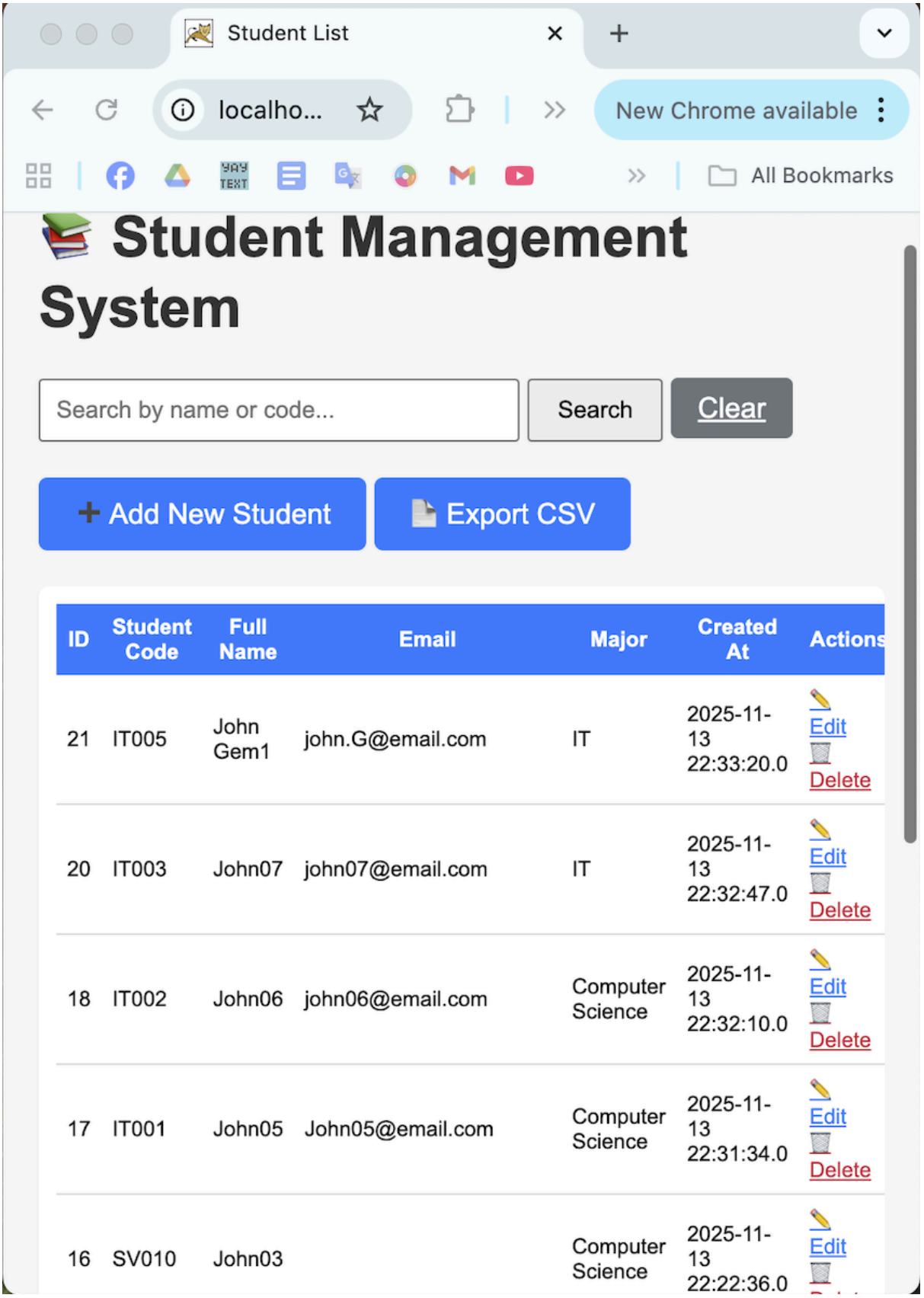


Explain:

When the user submits any form, JavaScript runs the submitForm() function. The Save button becomes disabled and its text changes to “Processing…”. This prevents double-clicks or repeated inserts while waiting for the response. Once redirected, the button returns to normal.

c) Responsive Table (3 points)

- **Table scrollable on small screens**
- **Better mobile layout**

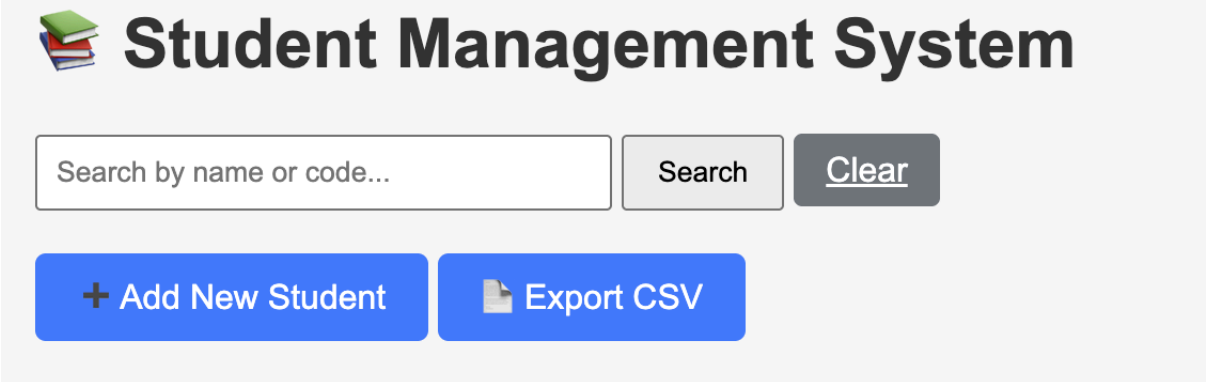


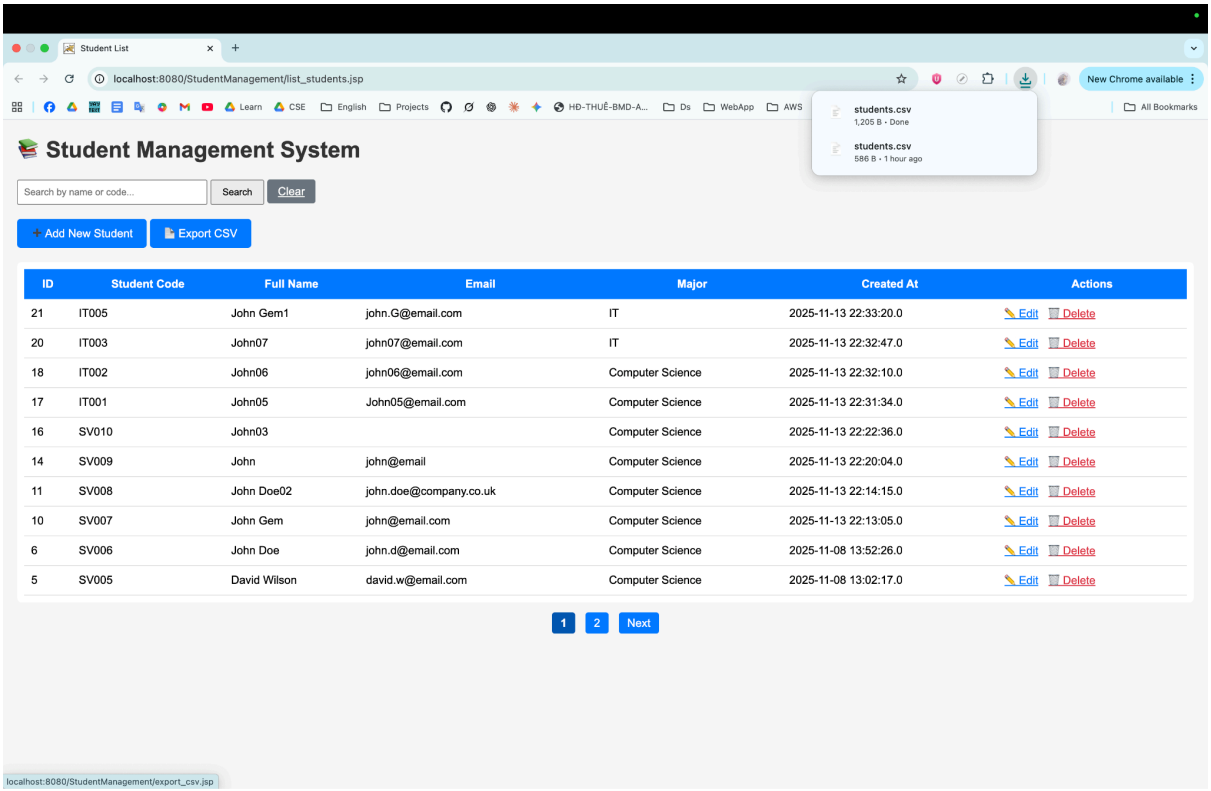
Explain:

When viewing on smaller screens, the .table-responsive CSS makes the table horizontally scrollable, while font size and padding adjust for visibility.

BONUS EXERCISES

BONUS 1: Export to CSV





Explain:

When the user clicks Add New Student, fills in the form, and saves, the student record is successfully stored in the database. On the list_students.jsp page, the user can then click the Export CSV button. This action sends a request to export_csv.jsp, which dynamically generates a CSV file from the database and automatically downloads it to the user's computer as students.csv.

The JSP sets the response headers using

response.setContentType("text/csv");

response.setHeader("Content-Disposition", "attachment; filename=\"students.csv\"");

so the browser recognizes it as a downloadable CSV file. The code then connects to the database using JDBC, retrieves all student records (or filtered results if a

keyword search is active), and prints each record as a CSV line using out.println(). The output includes columns such as ID, Student Code, Full Name, Email, Major, and Created At.