

**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.

ID	Student Code	Full Name	Email	Major	Created At	Actions
6	SV006	John Doe	john.d@email.com	Computer Science	2025-11-08 13:52:26.0	<a href="#">Edit</a> <a href="#">Delete</a>
5	SV005	David Wilson	david.w@email.com	Computer Science	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>
4	SV004	Sarah Davis	sarah.d@email.com	Data Science	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>
3	SV003	Michael Brown	michael.b@email.com	Software Engineering	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>
2	SV002	Emily Johnson	emily.j@email.com	Information Technology	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>

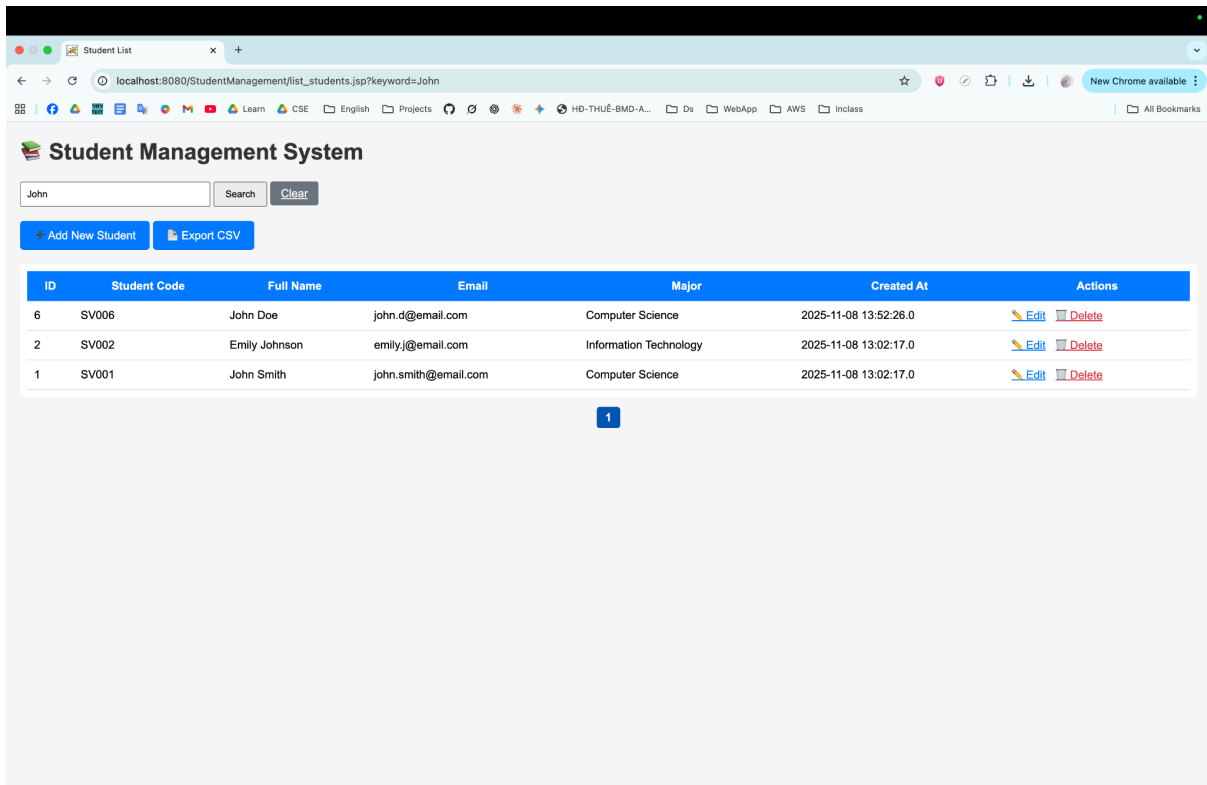
1 2 Next

### 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.

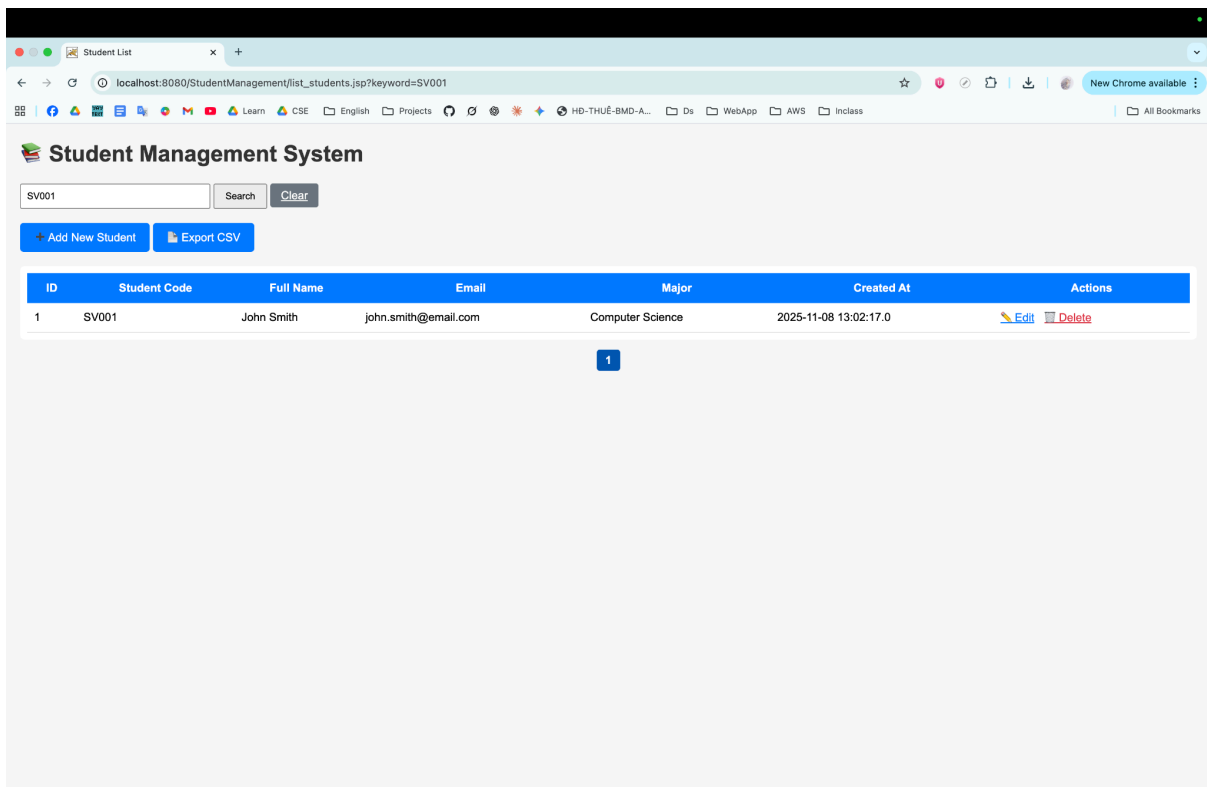
## ● Test Cases 1:



### 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.

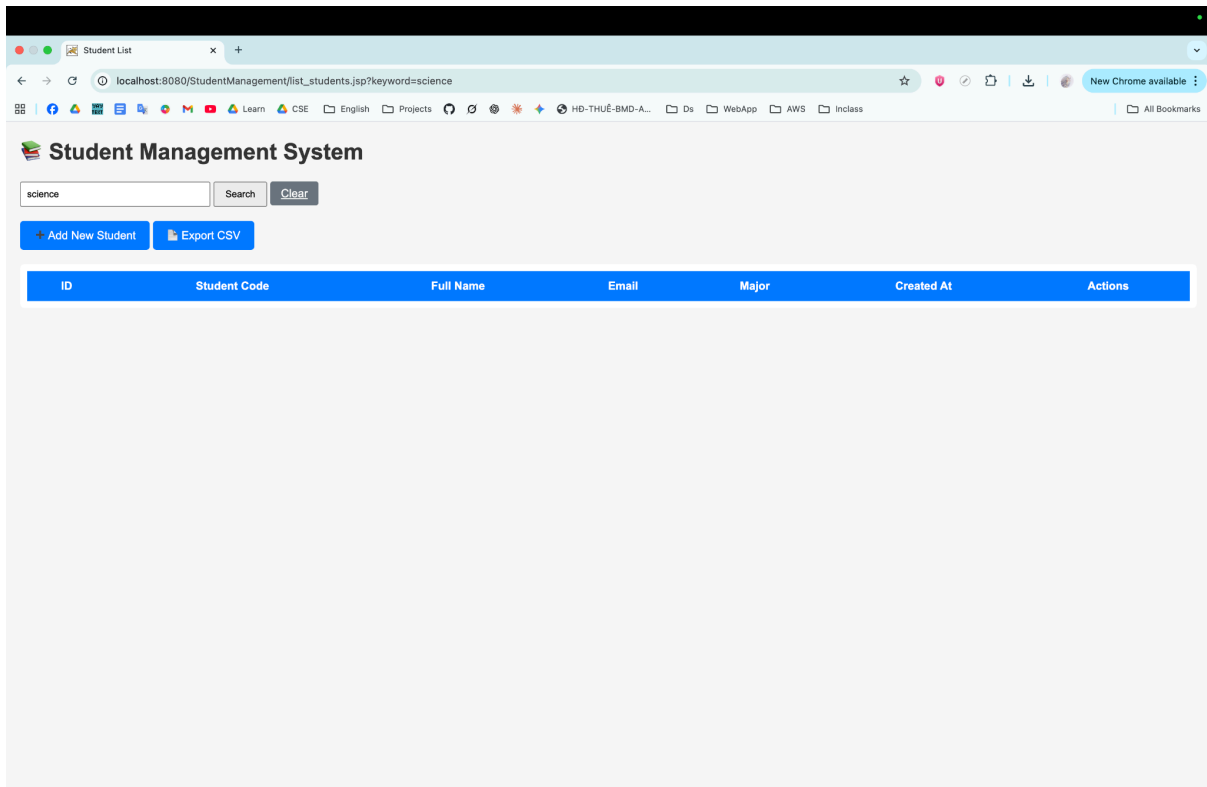
- **Test Cases 2:**



### **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.

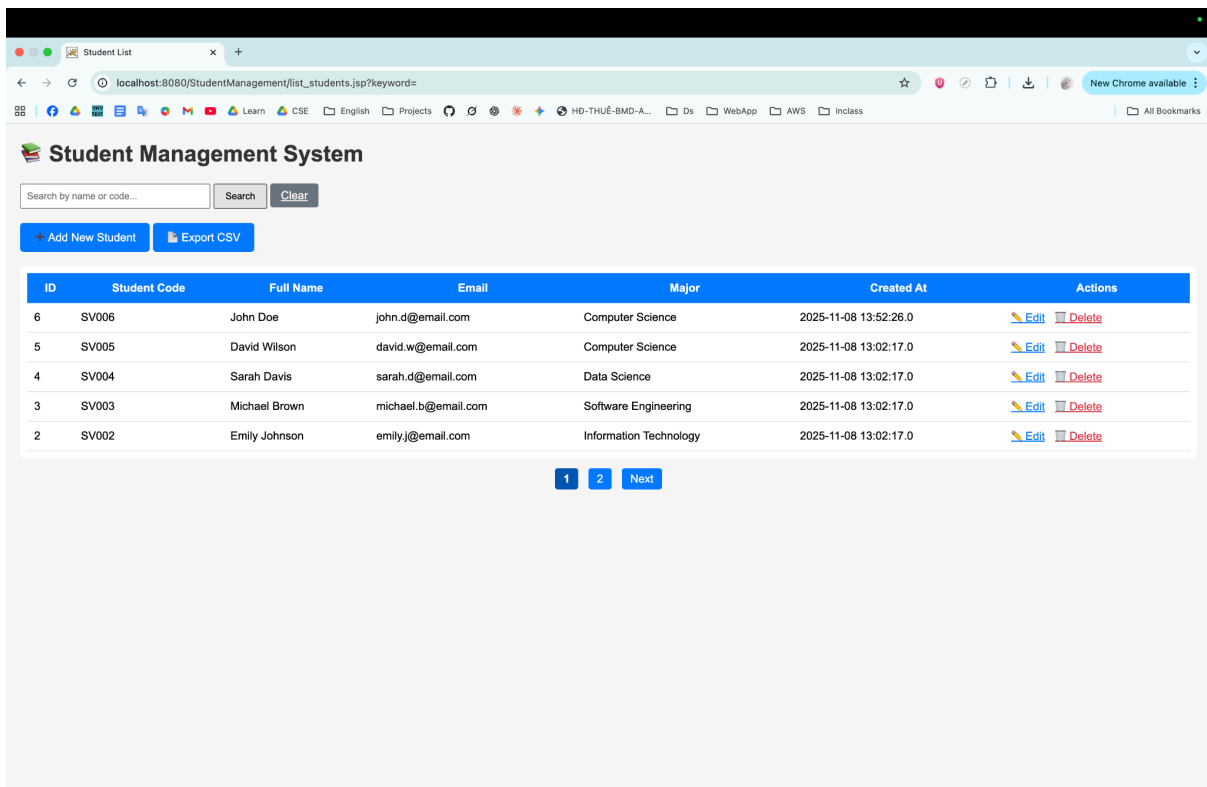
- **Test Cases 3:**



**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:

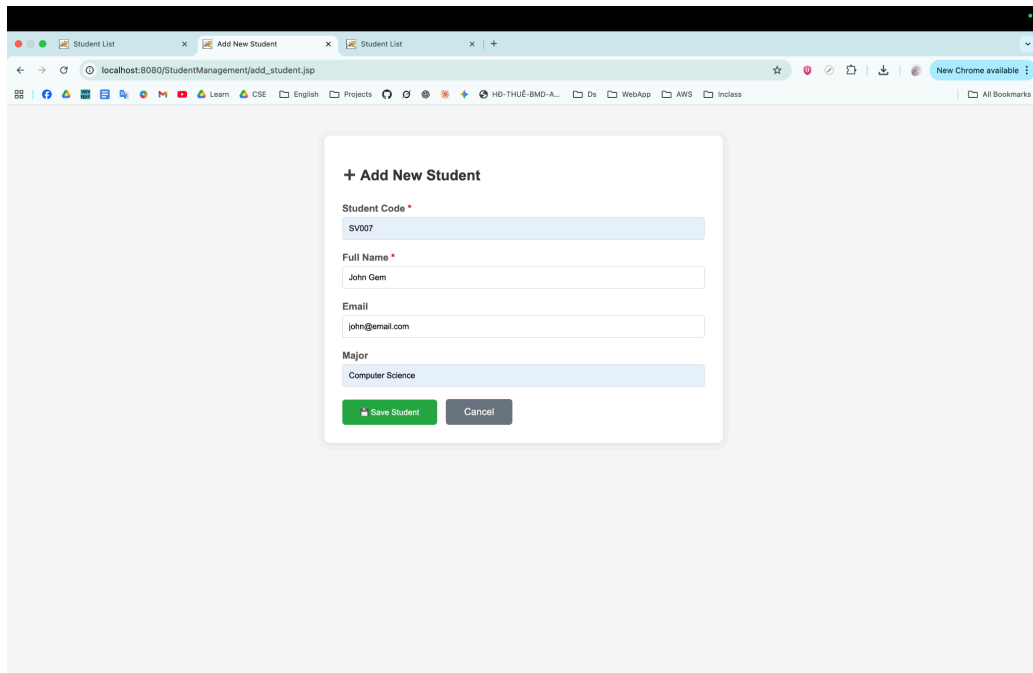
When the search field is left blank and the user clicks Search, the browser still sends a request to list\_students.jsp but with an empty keyword value. The server checks: if (keyword != null && !keyword.trim().isEmpty()) Since this condition is false, the code executes the default SQL: SELECT \* FROM 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:**



**+ Add New Student**

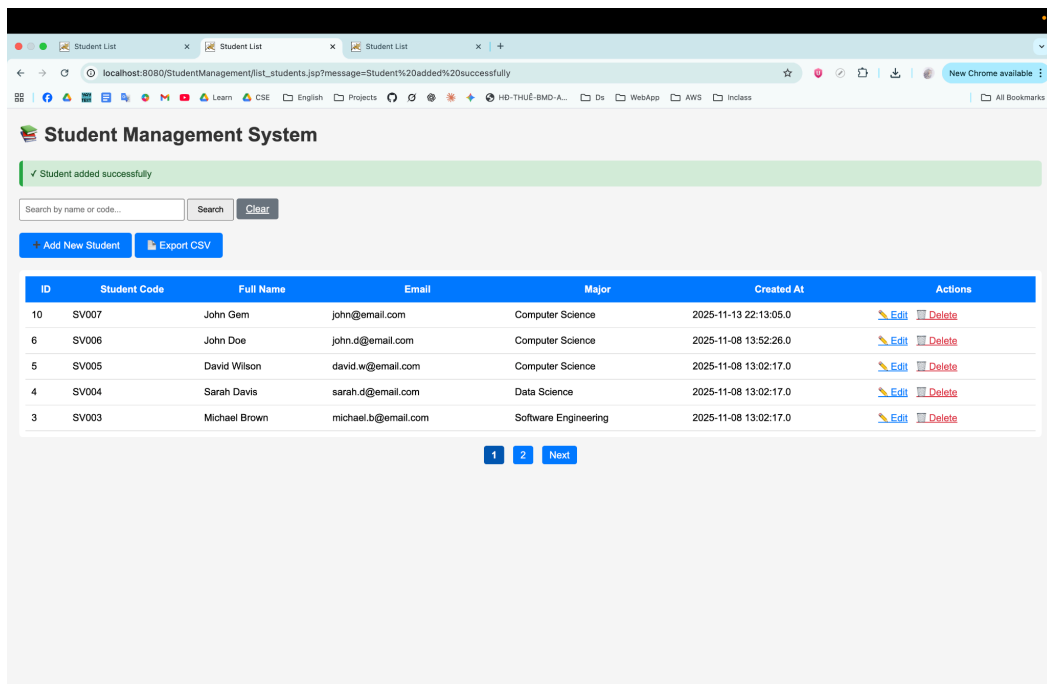
Student Code \*  
SV007

Full Name \*  
John Gem

Email  
john@email.com

Major  
Computer Science

Save Student Cancel



**Student Management System**

✓ Student added successfully

Search by name or code... Search Clear

Add New Student Export CSV

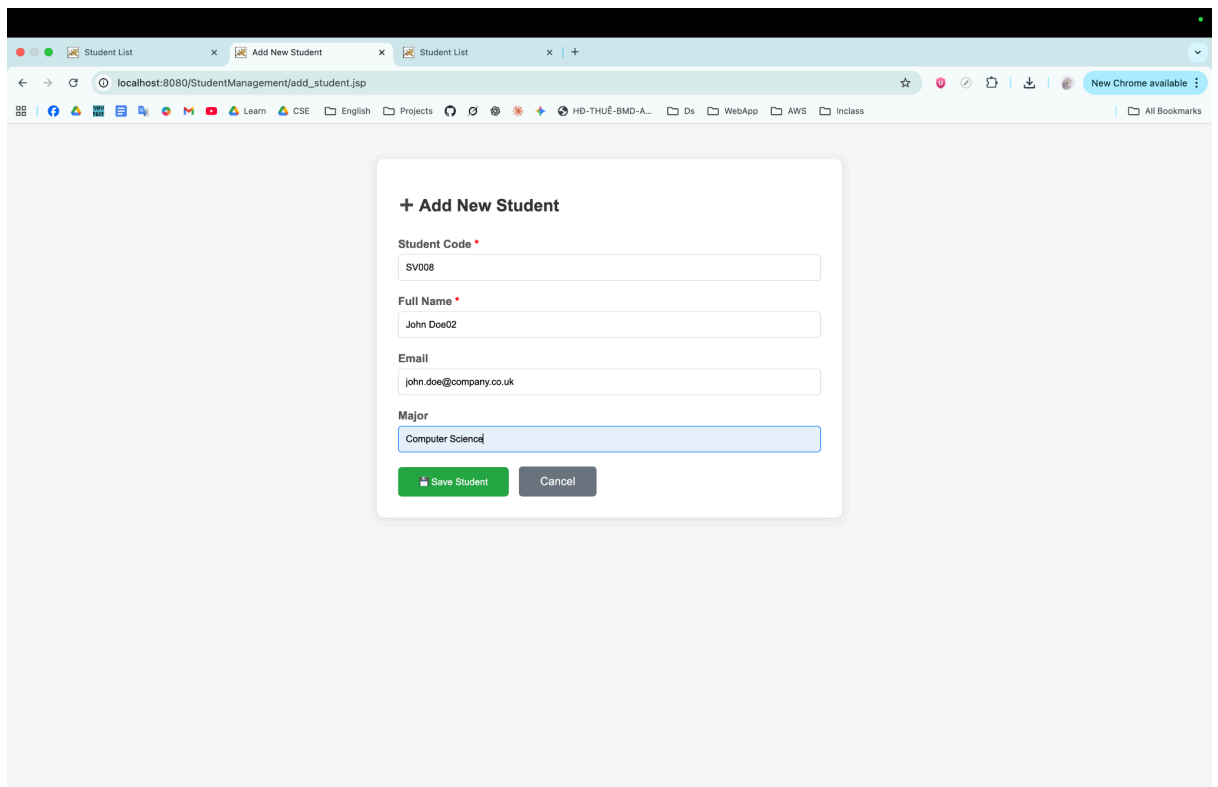
ID	Student Code	Full Name	Email	Major	Created At	Actions
10	SV007	John Gem	john@email.com	Computer Science	2025-11-13 22:13:05.0	Edit Delete
6	SV006	John Doe	john.d@email.com	Computer Science	2025-11-08 13:52:26.0	Edit Delete
5	SV005	David Wilson	david.w@email.com	Computer Science	2025-11-08 13:02:17.0	Edit Delete
4	SV004	Sarah Davis	sarah.d@email.com	Data Science	2025-11-08 13:02:17.0	Edit Delete
3	SV003	Michael Brown	michael.b@email.com	Software Engineering	2025-11-08 13:02:17.0	Edit Delete

1 2 Next

## 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:**



**+ Add New Student**

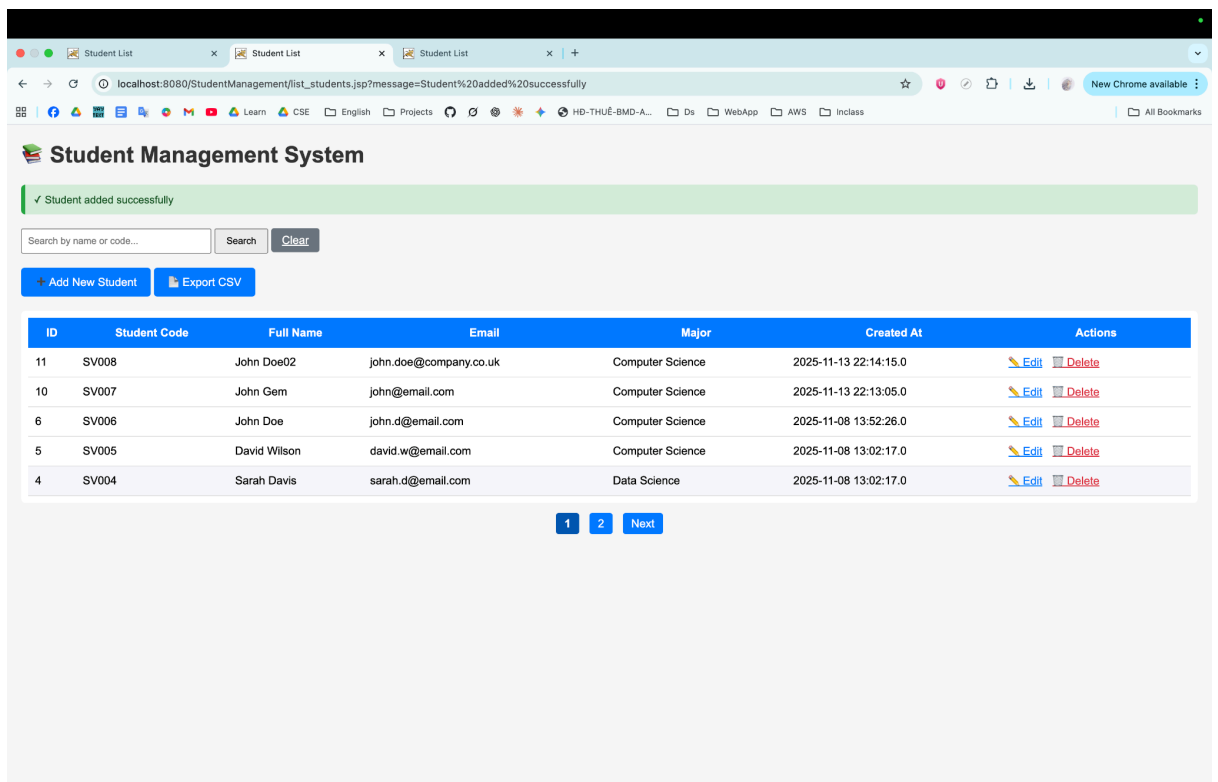
Student Code \*  
SV008

Full Name \*  
John Doe02

Email  
john.doe@company.co.uk

Major  
Computer Science

[Save Student](#) [Cancel](#)



**Student Management System**

✓ Student added successfully

Search by name or code... [Search](#) [Clear](#)

[+ Add New Student](#) [Export CSV](#)

ID	Student Code	Full Name	Email	Major	Created At	Actions
11	SV008	John Doe02	john.doe@company.co.uk	Computer Science	2025-11-13 22:14:15.0	<a href="#">Edit</a> <a href="#">Delete</a>
10	SV007	John Gem	john@email.com	Computer Science	2025-11-13 22:13:05.0	<a href="#">Edit</a> <a href="#">Delete</a>
6	SV006	John Doe	john.d@email.com	Computer Science	2025-11-08 13:52:26.0	<a href="#">Edit</a> <a href="#">Delete</a>
5	SV005	David Wilson	david.w@email.com	Computer Science	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>
4	SV004	Sarah Davis	sarah.d@email.com	Data Science	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>

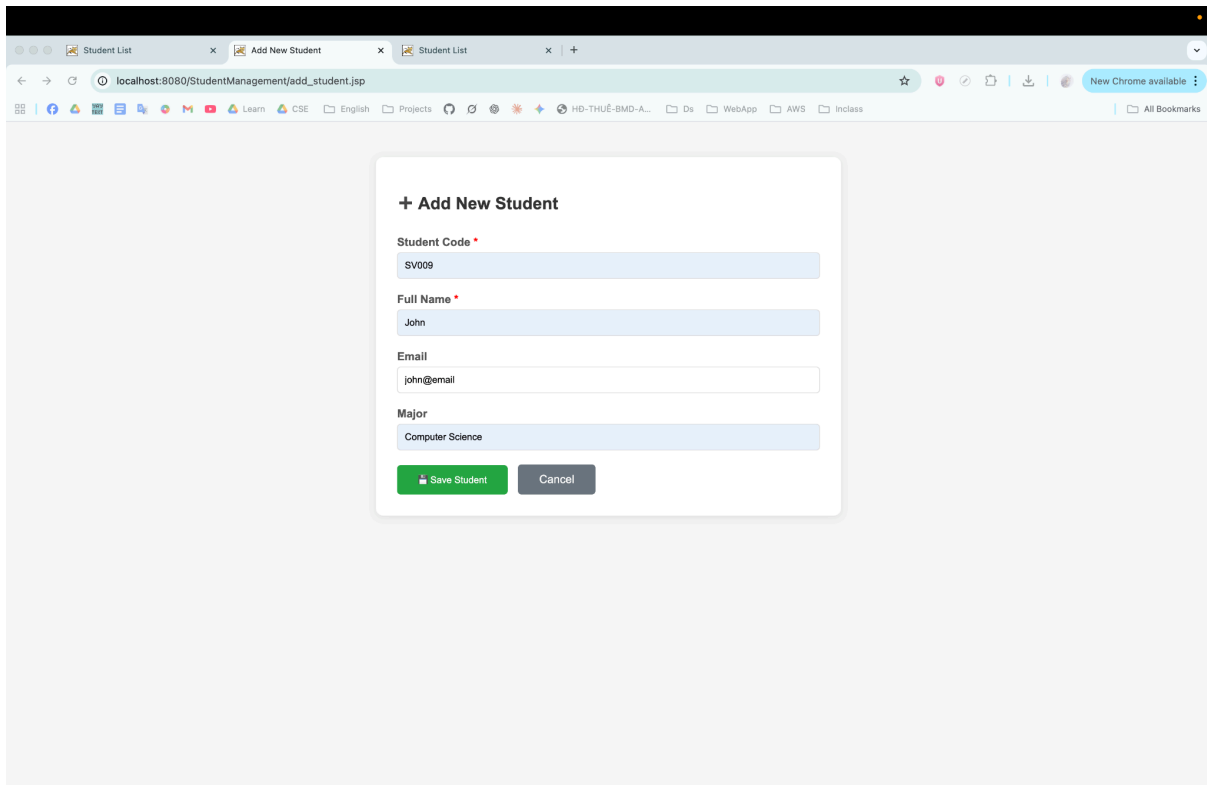
[1](#) [2](#) [Next](#)

## 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:



**+ Add New Student**

Student Code \*

SV009

Full Name \*

John

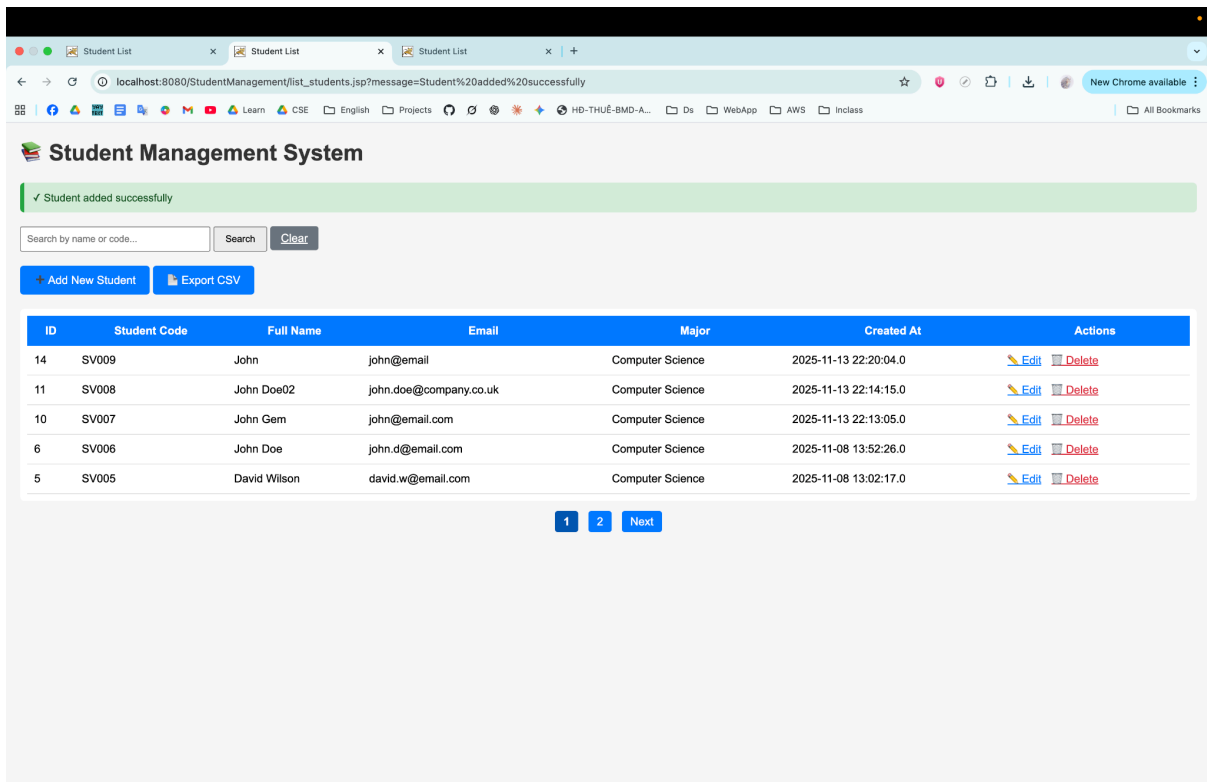
Email

john@email

Major

Computer Science

Save Student Cancel



**Student Management System**

✓ Student added successfully

Search by name or code... Search Clear

+ Add New Student Export CSV

ID	Student Code	Full Name	Email	Major	Created At	Actions
14	SV009	John	john@email	Computer Science	2025-11-13 22:20:04.0	Edit Delete
11	SV008	John Doe02	john.doe@company.co.uk	Computer Science	2025-11-13 22:14:15.0	Edit Delete
10	SV007	John Gem	john@email.com	Computer Science	2025-11-13 22:13:05.0	Edit Delete
6	SV006	John Doe	john.d@email.com	Computer Science	2025-11-08 13:52:26.0	Edit Delete
5	SV005	David Wilson	david.w@email.com	Computer Science	2025-11-08 13:02:17.0	Edit Delete

1 2 Next

## 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:

The screenshot shows a web browser window with the URL `localhost:8080/StudentManagement/add_student.jsp`. The browser has three tabs: 'Student List', 'Add New Student', and 'Student List'. The 'Add New Student' tab is active, displaying a form titled '+ Add New Student'. The form contains the following fields and values:

- Student Code \***: `SV010`
- Full Name \***: `John02`
- Email**: `johnemail.com`
- Major**: `Computer Science`

At the bottom of the form are two buttons: 'Save Student' (green) and 'Cancel' (grey).

The screenshot shows the same web browser window, but the URL is `localhost:8080/StudentManagement/add_student.jsp?error=Invalid%20email%20format`. The 'Add New Student' form is displayed with an error message at the top: 'Invalid email format'. The form fields and their values are:

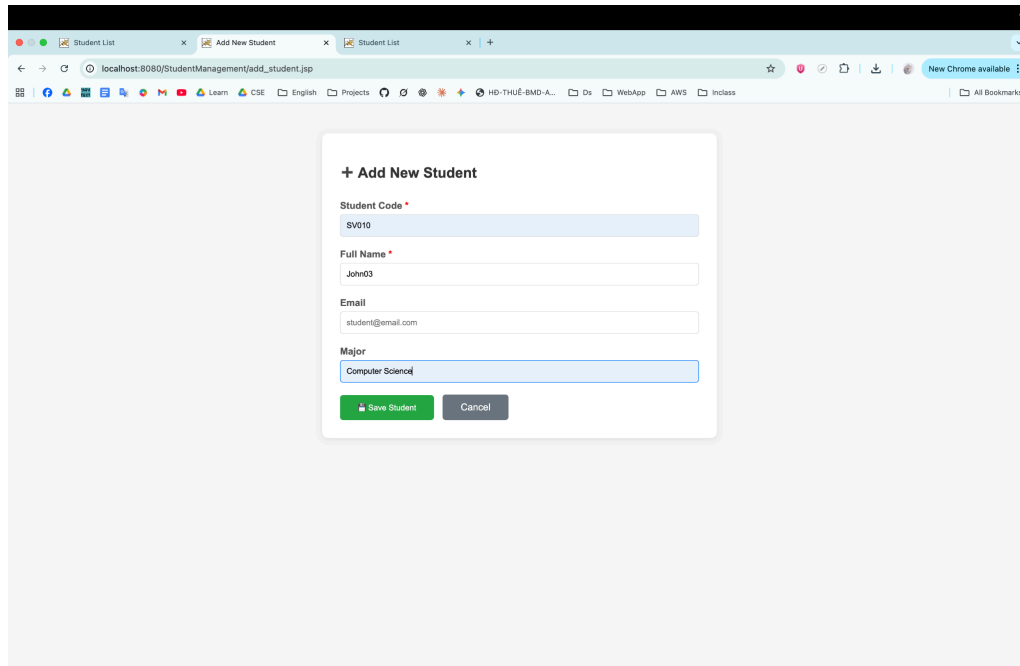
- Student Code \***: `e.g., SV001`
- Full Name \***: `Enter full name`
- Email**: `student@email.com`
- Major**: `e.g., Computer Science`

The 'Save Student' and 'Cancel' buttons are still present at the bottom.

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.

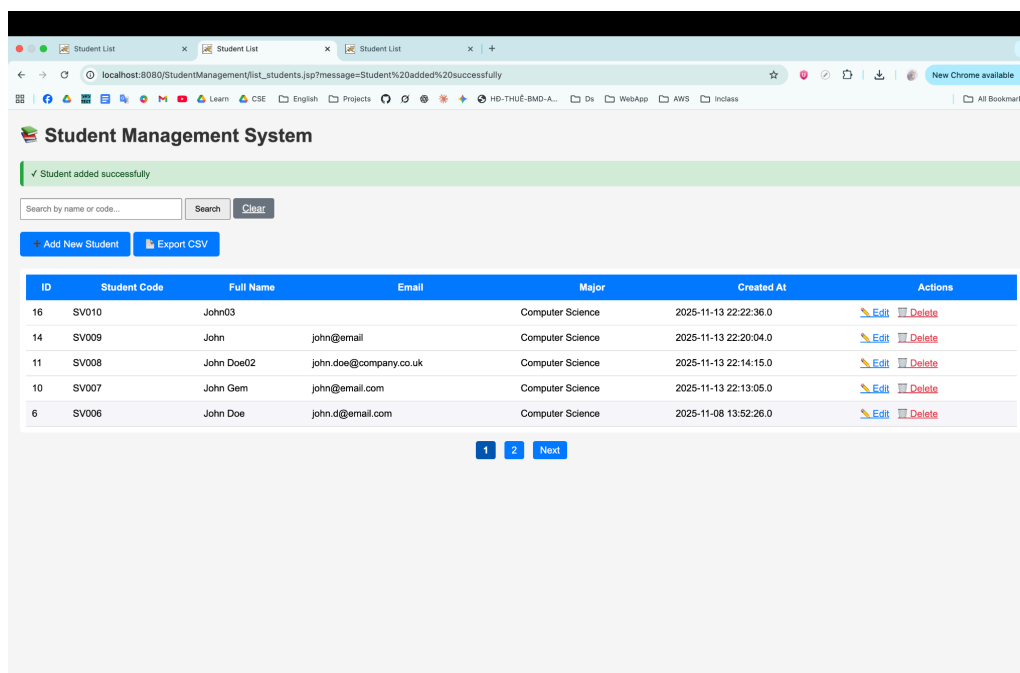
- **Test Cases 5:**



The screenshot shows a web browser window with the URL `localhost:8080/StudentManagement/add_student.jsp`. The page displays a form titled "+ Add New Student". The form contains the following fields and values:

- Student Code: SV010
- Full Name: John03
- Email: student@email.com
- Major: Computer Science

At the bottom of the form are two buttons: "Save Student" (green) and "Cancel" (grey).



The screenshot shows the "Student Management System" dashboard. A green banner at the top indicates "Student added successfully". Below this is a search bar with the text "Search by name or code...". There are two buttons: "Add New Student" and "Export CSV".

The main part of the dashboard is a table with the following columns: ID, Student Code, Full Name, Email, Major, Created At, and Actions. The table contains 5 rows of student data:

ID	Student Code	Full Name	Email	Major	Created At	Actions
16	SV010	John03		Computer Science	2025-11-13 22:22:36.0	<a href="#">Edit</a> <a href="#">Delete</a>
14	SV009	John	john@email	Computer Science	2025-11-13 22:20:04.0	<a href="#">Edit</a> <a href="#">Delete</a>
11	SV008	John Doe02	john.doe@company.co.uk	Computer Science	2025-11-13 22:14:15.0	<a href="#">Edit</a> <a href="#">Delete</a>
10	SV007	John Gem	john@email.com	Computer Science	2025-11-13 22:13:05.0	<a href="#">Edit</a> <a href="#">Delete</a>
6	SV006	John Doe	john.d@email.com	Computer Science	2025-11-08 13:52:26.0	<a href="#">Edit</a> <a href="#">Delete</a>

At the bottom of the table are pagination controls: "1", "2", and "Next".

## 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:

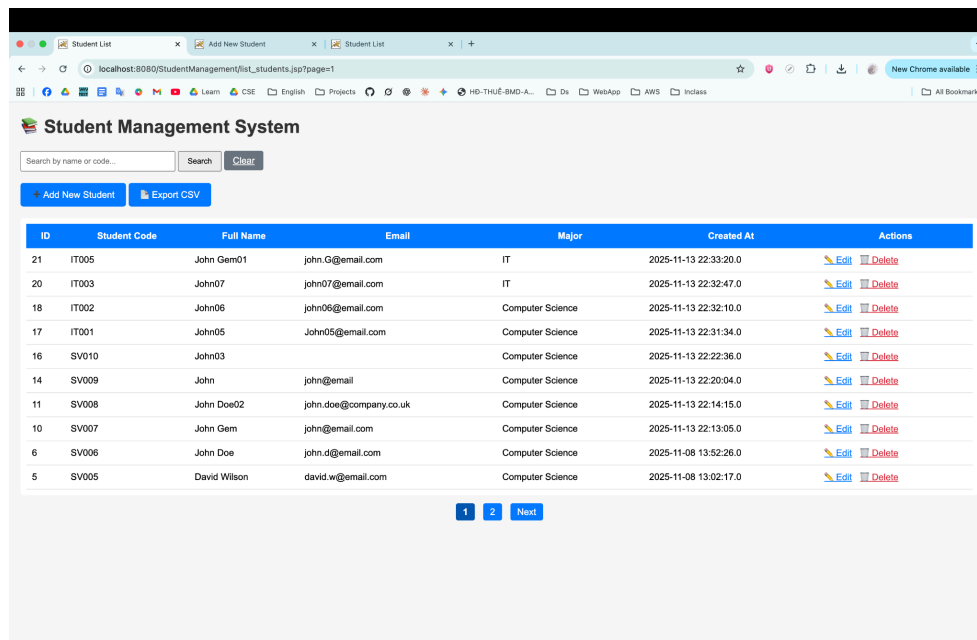
## Explain:

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

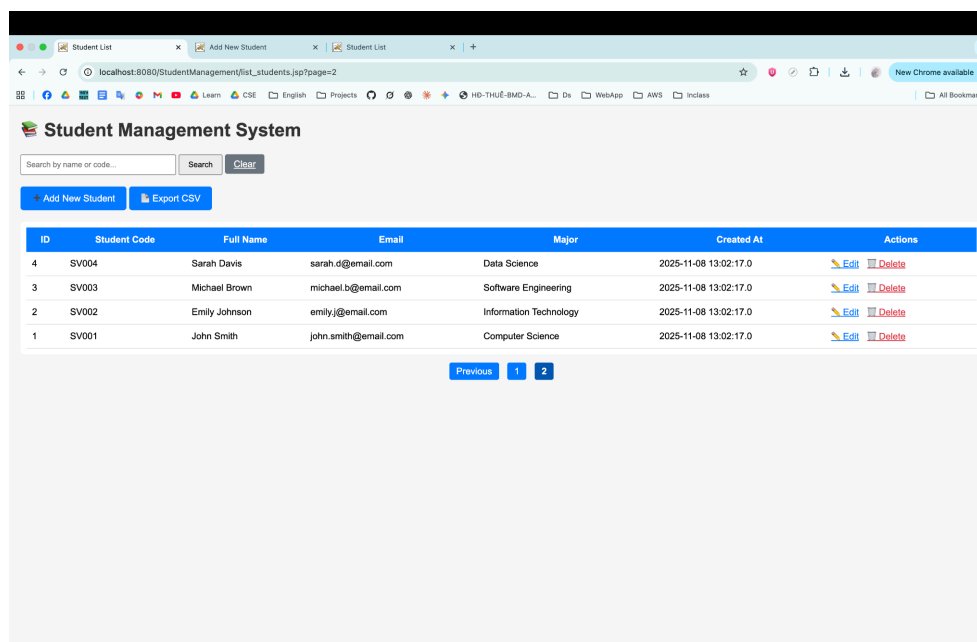


Student Management System

Search by name or code...

ID	Student Code	Full Name	Email	Major	Created At	Actions
21	IT005	John Gem01	john.g@email.com	IT	2025-11-13 22:33:20.0	<a href="#">Edit</a> <a href="#">Delete</a>
20	IT003	John07	john07@email.com	IT	2025-11-13 22:32:47.0	<a href="#">Edit</a> <a href="#">Delete</a>
18	IT002	John06	john06@email.com	Computer Science	2025-11-13 22:32:10.0	<a href="#">Edit</a> <a href="#">Delete</a>
17	IT001	John05	John05@email.com	Computer Science	2025-11-13 22:31:34.0	<a href="#">Edit</a> <a href="#">Delete</a>
16	SV010	John03		Computer Science	2025-11-13 22:22:36.0	<a href="#">Edit</a> <a href="#">Delete</a>
14	SV009	John	john@email	Computer Science	2025-11-13 22:20:04.0	<a href="#">Edit</a> <a href="#">Delete</a>
11	SV008	John Doe02	john.doe@company.co.uk	Computer Science	2025-11-13 22:14:15.0	<a href="#">Edit</a> <a href="#">Delete</a>
10	SV007	John Gem	john@email.com	Computer Science	2025-11-13 22:13:05.0	<a href="#">Edit</a> <a href="#">Delete</a>
6	SV006	John Doe	john.d@email.com	Computer Science	2025-11-08 13:52:26.0	<a href="#">Edit</a> <a href="#">Delete</a>
5	SV005	David Wilson	david.w@email.com	Computer Science	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>

[1](#) [2](#) [Next](#)



Student Management System

Search by name or code...

ID	Student Code	Full Name	Email	Major	Created At	Actions
4	SV004	Sarah Davis	sarah.d@email.com	Data Science	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>
3	SV003	Michael Brown	michael.b@email.com	Software Engineering	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>
2	SV002	Emily Johnson	emily.j@email.com	Information Technology	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>
1	SV001	John Smith	john.smith@email.com	Computer Science	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>

[Previous](#) [1](#) [2](#)

### 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;
```

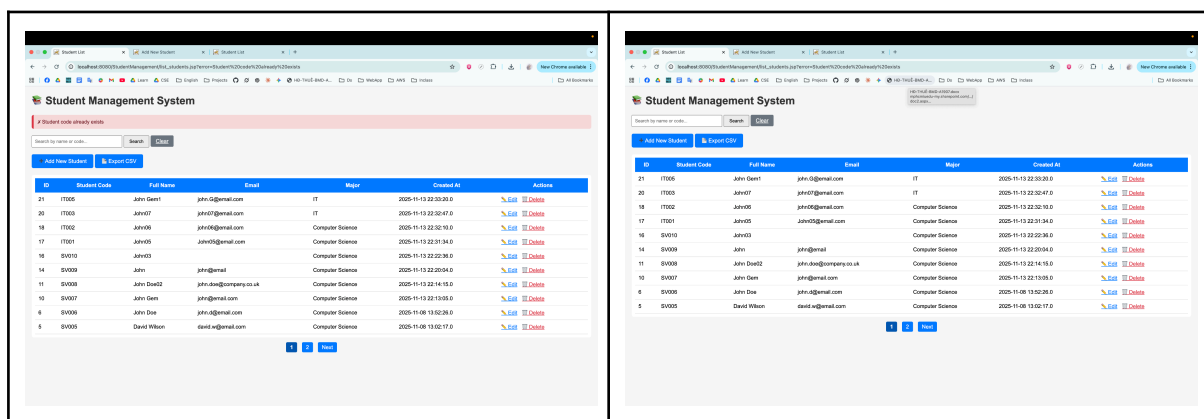
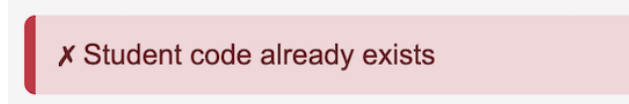
The SQL query includes LIMIT ? OFFSET ? to fetch only the students for that page. The code also counts total records with a separate query (SELECT COUNT(\*) FROM students) to calculate total pages, then displays numbered page links with Previous and Next buttons at the bottom.

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

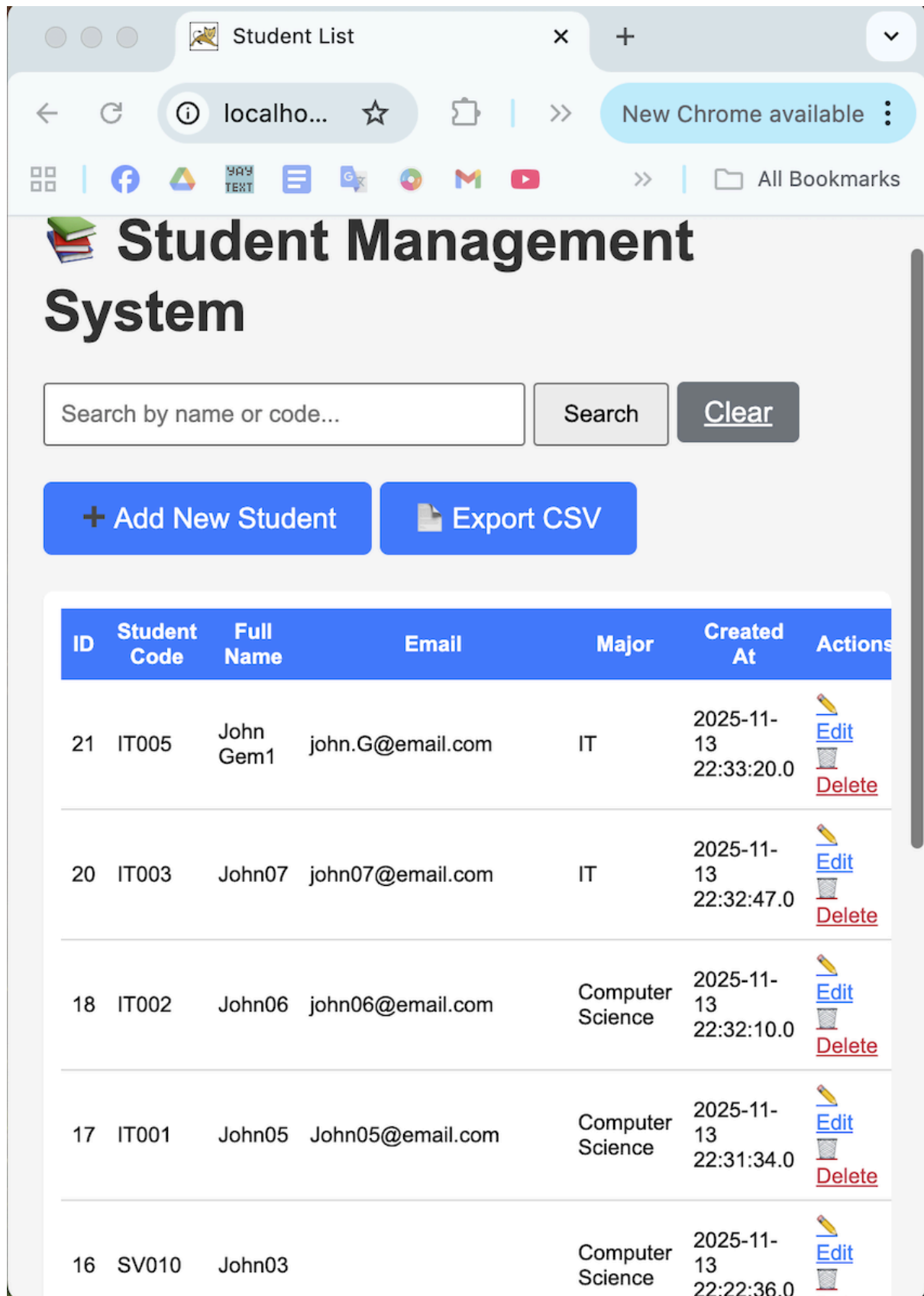
The screenshot shows a web browser window with two tabs: 'Add New Student' and 'Student List'. The active tab is 'Add New Student', which displays a form titled '+ Add New Student'. The form contains four input fields: 'Student Code' (with the value 'SV001'), 'Full Name' (with the value 'John'), 'Email' (with the value 'dichfb181@gmail.com'), and 'Major' (with the value 'Computer Science'). Below the input fields are two buttons: a green button labeled 'Processing...' and a grey button labeled 'Cancel'. The browser's address bar shows the URL: 'localhost:8080/StudentManagement/add\_student.jsp?student\_code=SV007&full\_name=John&email=dichfb181%40gmail.com&major=Computer+Science'. The browser's bookmarks bar shows several bookmarks, including 'Learn', 'CSE', 'English', 'Projects', 'HD-THUE-BMD-A...', 'Ds', 'WebApp', 'AWS', and 'Inclass'.

### 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.

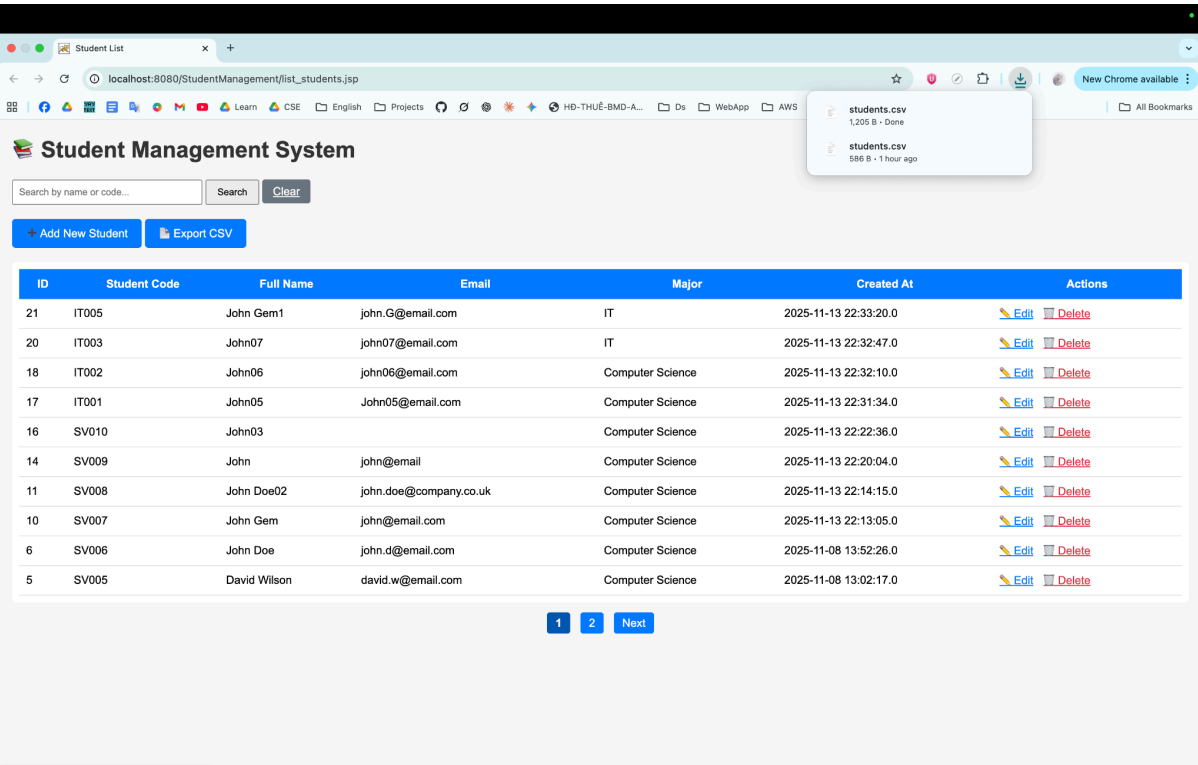


## BONUS EXERCISES

### BONUS 1: Export to CSV



# Student Management System



ID	Student Code	Full Name	Email	Major	Created At	Actions
21	IT005	John Gem1	john.G@email.com	IT	2025-11-13 22:33:20.0	<a href="#">Edit</a> <a href="#">Delete</a>
20	IT003	John07	john07@email.com	IT	2025-11-13 22:32:47.0	<a href="#">Edit</a> <a href="#">Delete</a>
18	IT002	John06	john06@email.com	Computer Science	2025-11-13 22:32:10.0	<a href="#">Edit</a> <a href="#">Delete</a>
17	IT001	John05	John05@email.com	Computer Science	2025-11-13 22:31:34.0	<a href="#">Edit</a> <a href="#">Delete</a>
16	SV010	John03		Computer Science	2025-11-13 22:22:36.0	<a href="#">Edit</a> <a href="#">Delete</a>
14	SV009	John	john@email	Computer Science	2025-11-13 22:20:04.0	<a href="#">Edit</a> <a href="#">Delete</a>
11	SV008	John Doe02	john.doe@company.co.uk	Computer Science	2025-11-13 22:14:15.0	<a href="#">Edit</a> <a href="#">Delete</a>
10	SV007	John Gem	john@email.com	Computer Science	2025-11-13 22:13:05.0	<a href="#">Edit</a> <a href="#">Delete</a>
6	SV006	John Doe	john.d@email.com	Computer Science	2025-11-08 13:52:26.0	<a href="#">Edit</a> <a href="#">Delete</a>
5	SV005	David Wilson	david.w@email.com	Computer Science	2025-11-08 13:02:17.0	<a href="#">Edit</a> <a href="#">Delete</a>

### 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.