EXERCISE 5: SEARCH FUNCTIONALITY (15 points)

Estimated Time: 45 minutes

Add search capability to find students by name or code.

Requirements:

5.1: Create Search Form (3 points)

- Add search form at top of list_students.jsp
- Input field for keyword
- Search button
- Method: GET (so search term appears in URL)

HTML Structure:

```
<form action="list_students.jsp" method="GET">
    <input type="text" name="keyword" placeholder="Search by name or code...">
    <button type="submit">Search</button>
    <a href="list_students.jsp">Clear</a>
</form>
```

5.2: Implement Search Logic (12 points)

Modify list_students.jsp to handle search:

```
// Pseudocode:
String keyword = request.getParameter("keyword");

if (keyword != null && !keyword.isEmpty()) {
    // Search query with LIKE operator
    sql = "SELECT * FROM students WHERE full_name LIKE ? OR student_code LIKE ?";
    pstmt = conn.prepareStatement(sql);
    pstmt.setString(1, "%" + keyword + "%");
    pstmt.setString(2, "%" + keyword + "%");
} else {
    // Normal query
    sql = "SELECT * FROM students ORDER BY id DESC";
}
```

Evaluation Criteria:

Criteria	Points
Search form added correctly	3
LIKE operator used properly	4
Searches both name and code	3
Displays search results	3
"Clear" link works	2

Test Cases:

Search Term	Expected Results
"John"	Shows all students with "John" in name
"SV001"	Shows student with code SV001
"science"	Shows students in Computer Science/Data Science
"" (empty)	Shows all students

Bonus (+2 points): Highlight search term in results

Explain code flow:

1. Check for Search Keyword

The first step checks if the user submitted a search term:

- The code retrieves the value of the keyword parameter from the URL.
- A boolean variable, **isSearch**, is set to **True** if a non-empty keyword is found; otherwise, it's **False**.

2. Decision and SQL Execution

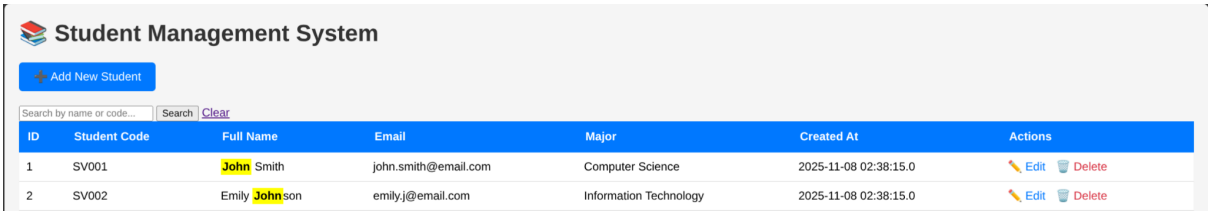
The program uses an **if/else** structure based on the **isSearch** flag to prepare and execute one of two different database queries:

Case	Condition	Description	Optimized SQL Statement
Search	isSearch == True	The program prepares a **PreparedStatement** to search for the keyword in two columns: **full_name** and **student_code**. The LIKE operator is used with wildcards (%) wrapped around the keyword (e.g., "%John%") to find partial matches.	SELECT * FROM students WHERE full_name LIKE ? OR student_code LIKE ?
Normal	isSearch == False	If no keyword is provided, the program prepares a **PreparedStatement** to retrieve **all students**, sorted by the latest added student (id DESC). This ensures the page defaults to showing the full list.	SELECT * FROM students ORDER BY id DESC

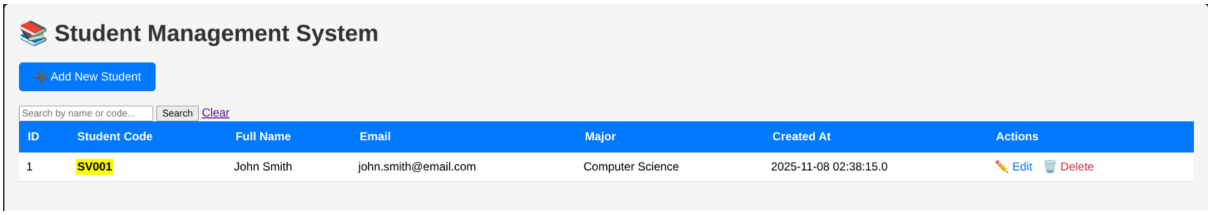
3. Display Results

1. The query is executed, and the results are stored in a **ResultSet**.
2. The code loops through every row in the ResultSet.
3. *(Bonus Logic)*: If isSearch is true, the String.replaceAll() method is used to wrap the matching keyword with an HTML tag (e.g.,) to **highlight the search term** in the displayed name and code fields.
4. The final processed data is rendered into the HTML table rows (<tr> and <td>) displayed on the JSP page.

Test case Shows all students with "John" in name

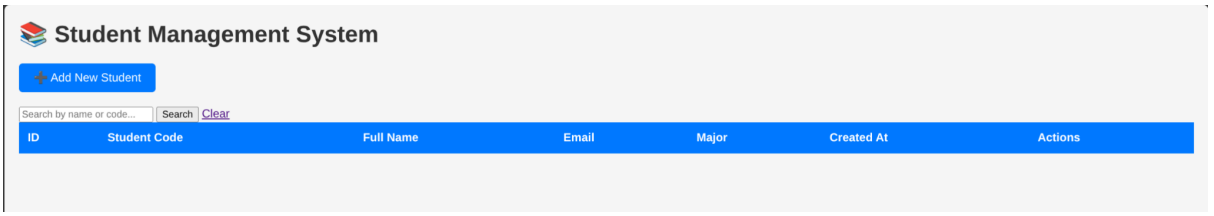


Shows student with code SV001



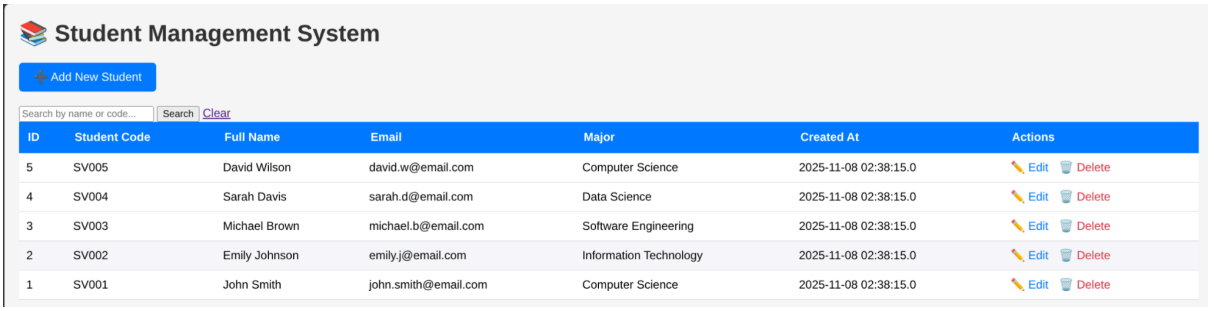
Shows students in Computer Science/Data Science

Because the implemented code only search for student_code or name keyword so the result will be empty



Shows all students

When click “search without” empty keyword, it will display all student list



EXERCISE 6: VALIDATION ENHANCEMENT (10 points)

Estimated Time: 30 minutes

Improve validation for better data quality.

6.1: Email Validation (5 points)

Add email format validation in process_add.jsp and process_edit.jsp:

Requirements:

- Check if email is provided
- If provided, validate format using regex
- Regex pattern: ^[A-Za-z0-9+_.-]+@(.+)$
- Display error if invalid

Java Code:

```
String email = request.getParameter("email");
if (email != null && !email.isEmpty()) {
    if (!email.matches("^[A-Za-z0-9+_.-]+@(.+)$")) {
        // Invalid email format
        response.sendRedirect("add_student.jsp?error=Invalid email format");
        return;
    }
}
```

Test Cases:

Email Input	Expected Result
john@email.com	Valid, accepts
john.doe@company.co.uk	Valid, accepts
john@email	Invalid, rejects
johnemail.com	Invalid, rejects
(empty)	Valid, accepts (optional field)

Explain code flow:

1. Retrieve and Check

- **Grab Email:** The code first retrieves the email value submitted by the user from the form.
- **Optional Field Check:** It then checks if the email field is **not empty**. Since the email is an optional field, the code proceeds with validation only if the user actually typed something in.

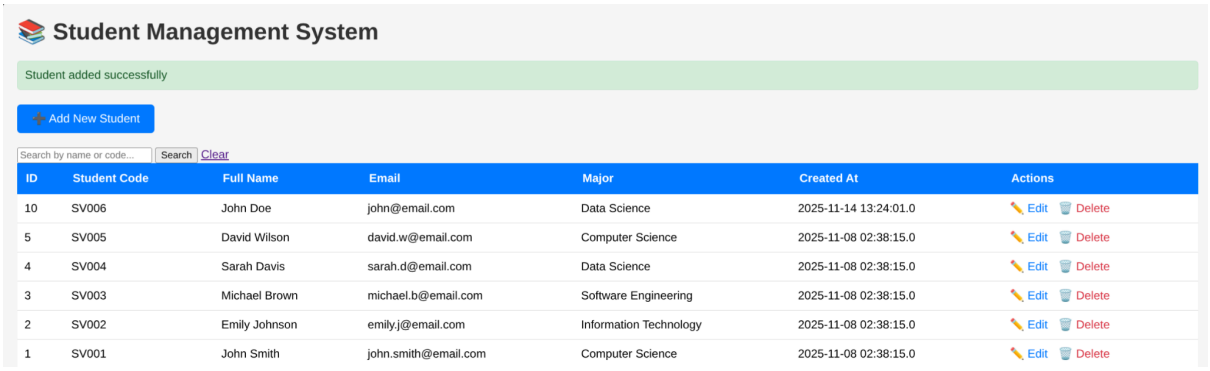
2. Validation Logic

- **Set the Standard:** A specific regular expression (String emailPattern = "^[A-Za-z0-9+_.-]+@.+\\.[A-Za-z]{2,}$";) is defined. This pattern acts as the **standard format** for a valid email (e.g., it requires an @ symbol followed by a domain that includes a dot like .com or .org).
- **The Match Test:** The code uses the email.matches(emailPattern) method to test the user's input against the standard.

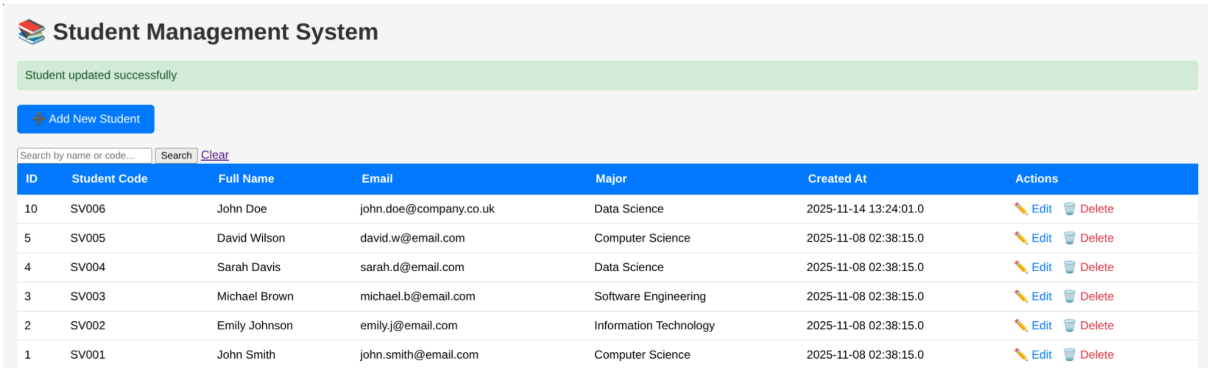
3. Response Decision

- **IF (Invalid Format):** If the email input fails the match test (e.g., john@email or abc), the code immediately executes a **redirect**.
    - It sends the user back to the input form (add_student.jsp or edit_student.jsp).
    - It includes an **error message** in the URL (?error=Invalid email format).
    - The return; statement halts the script, preventing the invalid data from reaching the database connection logic.
- **ELSE (Valid or Empty):** If the email is either empty (which is okay) or matches the required format, the script continues to the next validation checks (like student code) and eventually connects to the database to proceed with the record insertion or update.

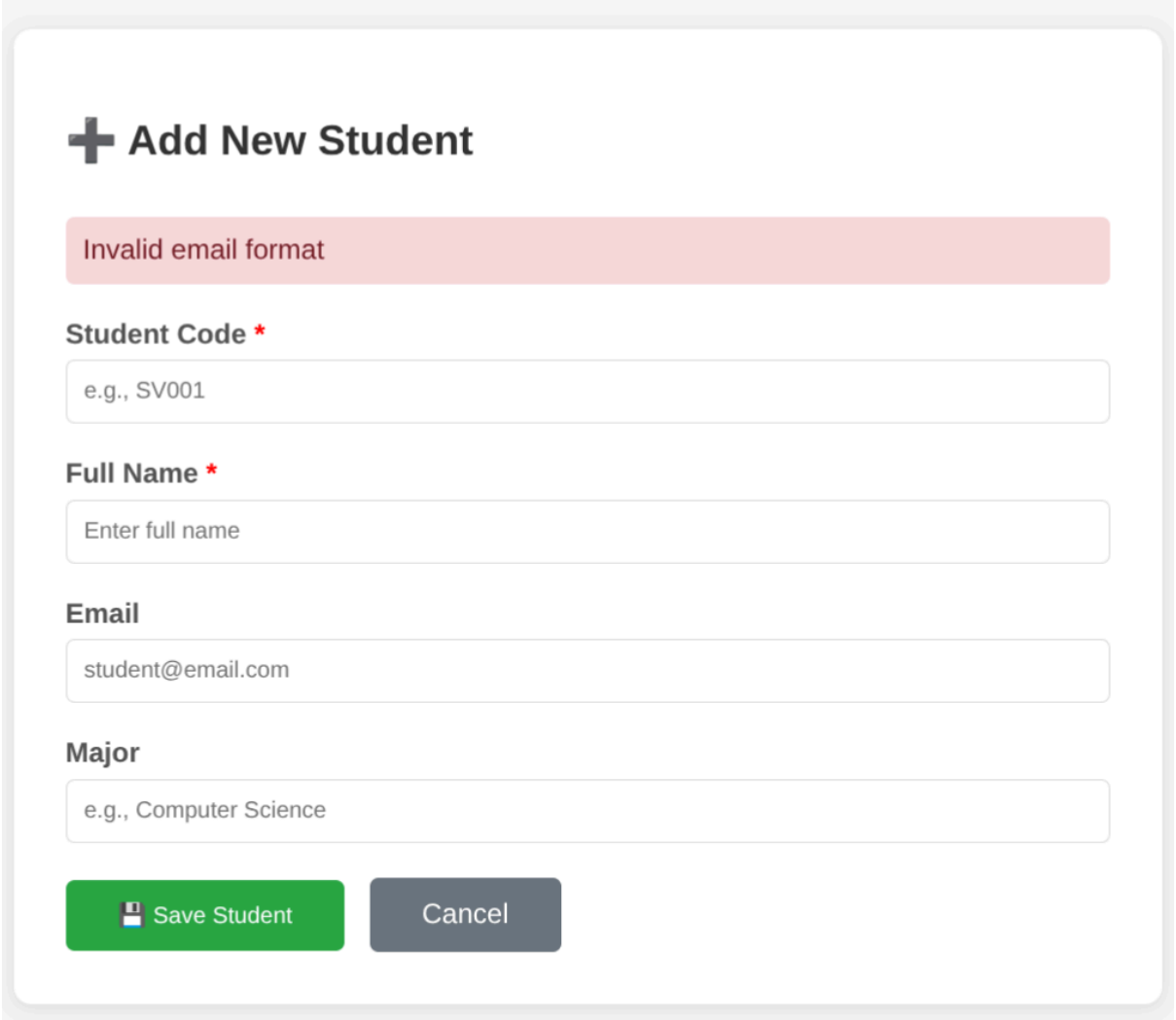
Add student with email john@email.com (valid)



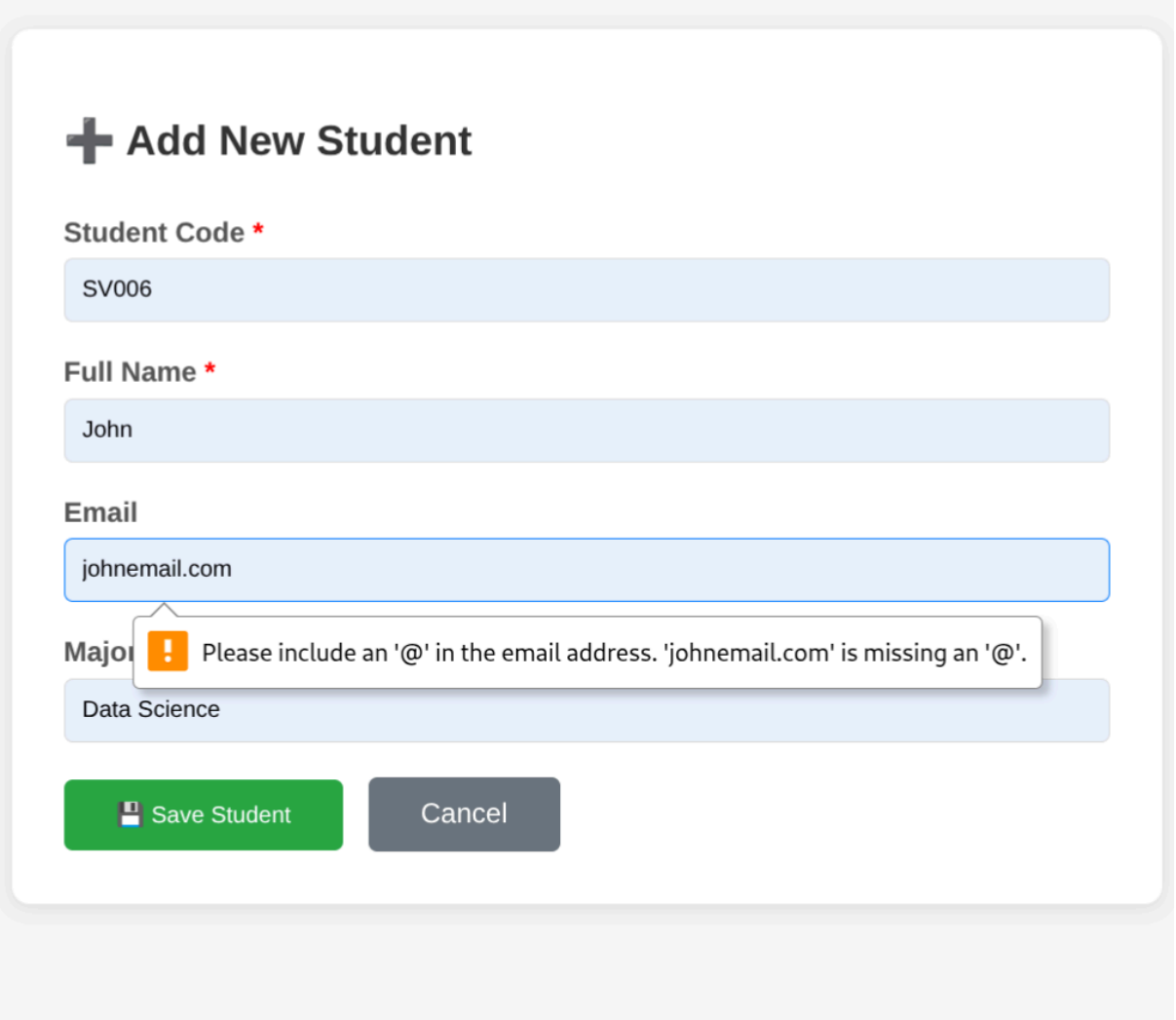
Edit email to john.doe@company.co.uk (valid)



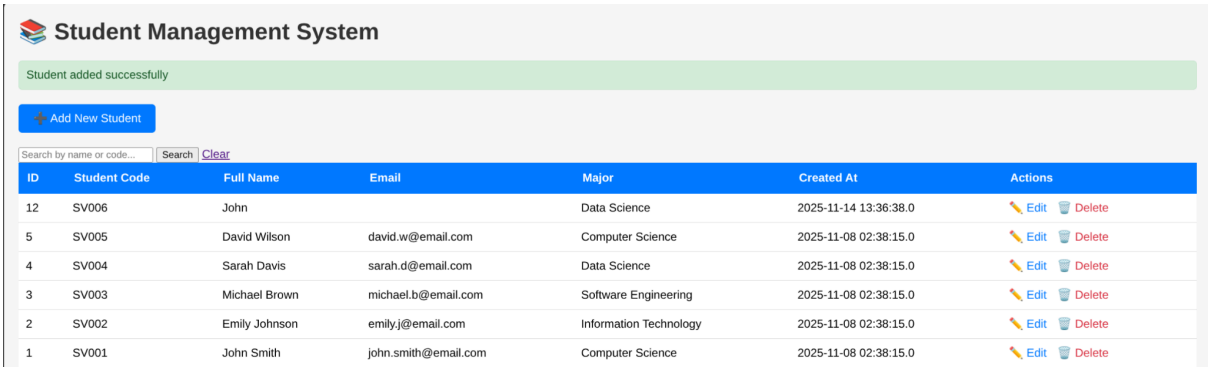
Edit email to john@email (invalid)



Add a student with email johnemail.com (invalid)



Add student with empty email field (valid)



6.2: Student Code Pattern Validation (5 points)

Validate student code follows pattern: 2 uppercase letters + 3+ digits

Requirements:

- Pattern: [A-Z]{2}[0-9]{3,}
- Examples: SV001, IT123, CS9999 (valid)
- Examples: sv001, S001, SV12 (invalid)
- Check in server-side code
- Display clear error message

Evaluation Criteria:

Criteria	Points
Email validation regex correct	3
Error displayed for invalid email	2
Student code pattern validated	3
Error displayed for invalid code	2

Explain code flow:

1. Retrieve and Define the Standard

- **Grab Code:** The script retrieves the student_code value submitted by the user.
- **Define Pattern:** A specific regular expression is defined to enforce the rule: **two uppercase letters followed by at least three digits**.
    - String codePattern = "[A-Z]{2}[0-9]{3,}";

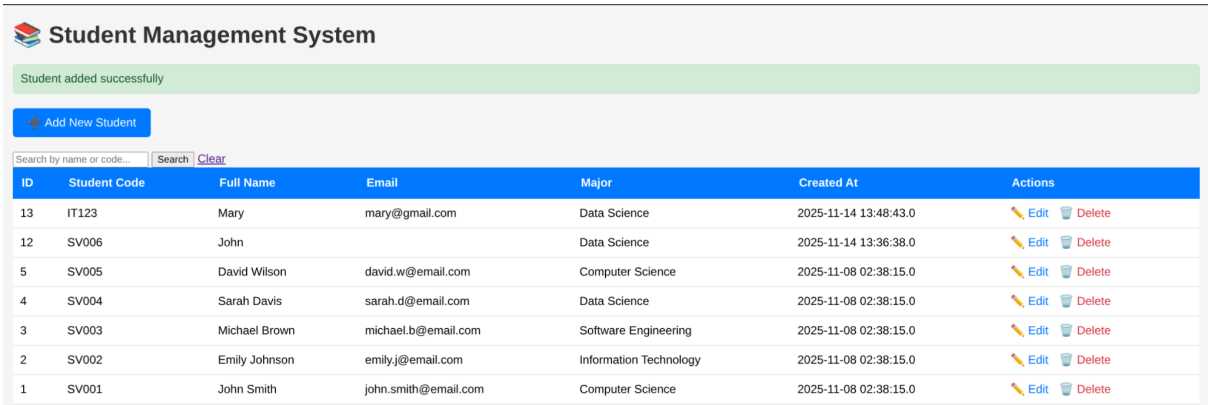
2. The Validation Test

- **The Match Test:** The code uses the studentCode.matches(codePattern) method to test the user's input against the strict format requirement.
    - *Example Valid:* CS101, IT9999
    - *Example Invalid:* cs101 (lowercase), C101 (only one letter), SV12 (only two digits).

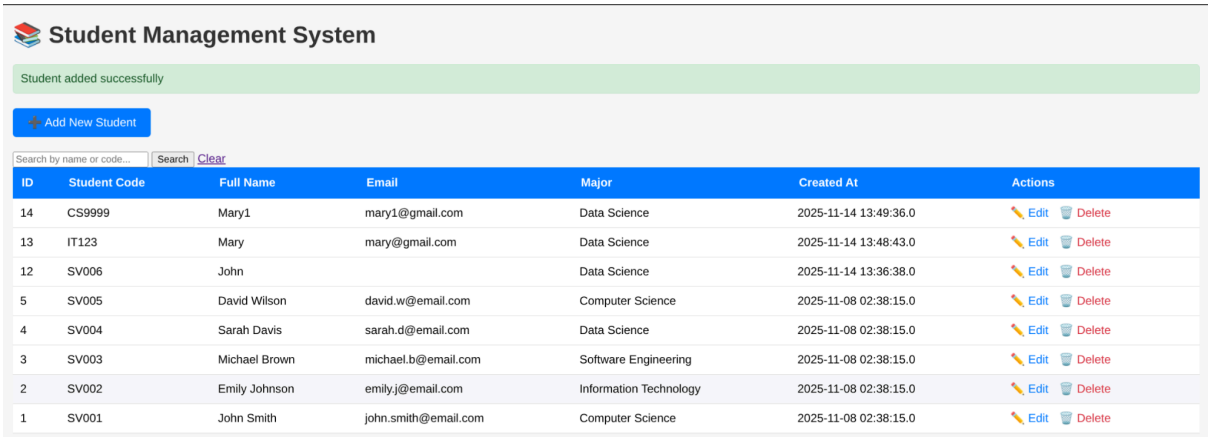
3. Response Decision

- **IF (Invalid Pattern):** If the input code fails the match test (e.g., the code is lowercase or has too few digits), the code immediately executes a **redirect**.
    - It sends the user back to the input form (add_student.jsp).
    - It includes a **clear error message** in the URL (?error=Student code must be 2 uppercase letters followed by 3+ digits...).
    - The return; statement stops the script, preventing the incorrectly formatted data from proceeding to the database.
- **ELSE (Valid Pattern):** If the student code adheres to the defined pattern, the script continues to the next validation checks (like email format) and eventually proceeds to the database insertion.

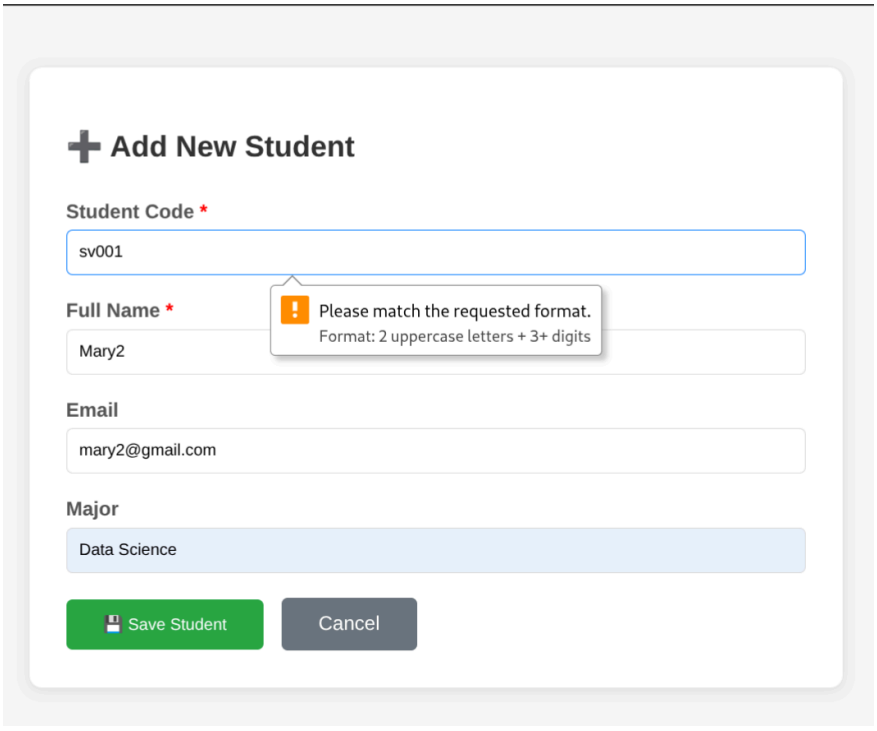
New student with id = IT123 (valid)



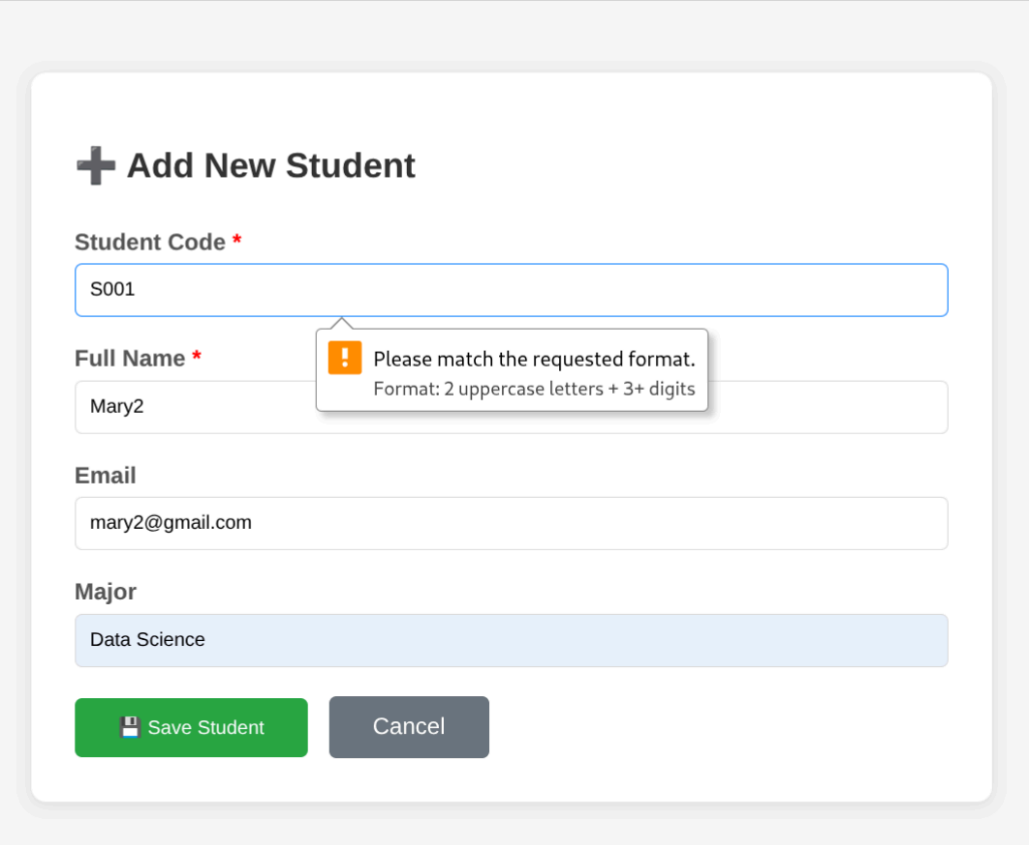
New student with id = CS9999 (valid)



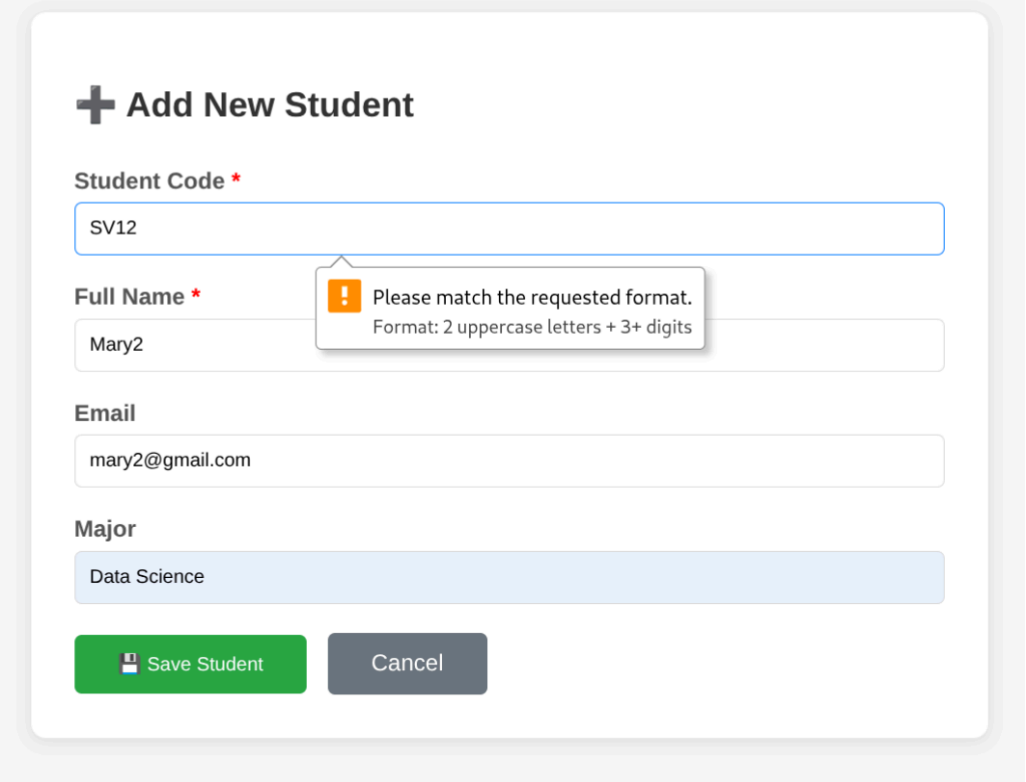
New student with id = sv001 (invalid)



New student with id = S001 (invalid)



New student with id = SV12 (invalid)



EXERCISE 7: USER EXPERIENCE IMPROVEMENTS (15 points)

Estimated Time: 60 minutes

7.1: Pagination (8 points)

Add pagination to display 10 students per page.

Requirements:

Database Query with LIMIT:

```
SELECT * FROM students
ORDER BY id DESC
LIMIT ? OFFSET ?
```

Calculate Pagination:

```
// Get page number from URL (default = 1)
String pageParam = request.getParameter("page");
int currentPage = (pageParam != null) ? Integer.parseInt(pageParam) : 1;

// Records per page
int recordsPerPage = 10;

// Calculate offset
int offset = (currentPage - 1) * recordsPerPage;

// Get total records for pagination
int totalRecords = getTotalRecords(); // You need to implement this
int totalPages = (int) Math.ceil((double) totalRecords / recordsPerPage);
```

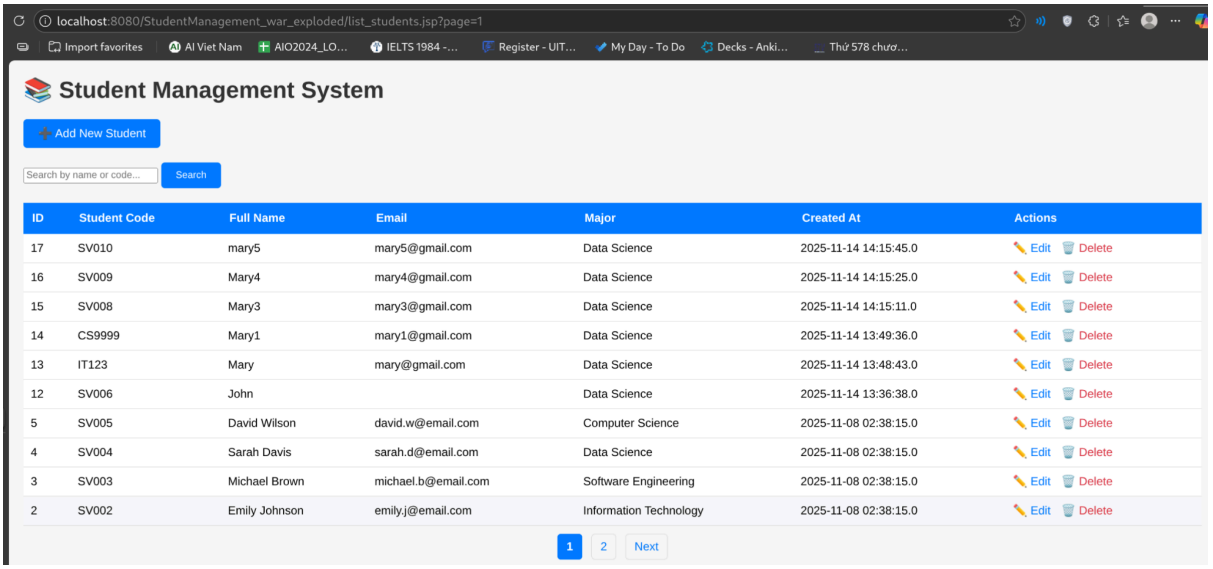
Display Pagination Links:

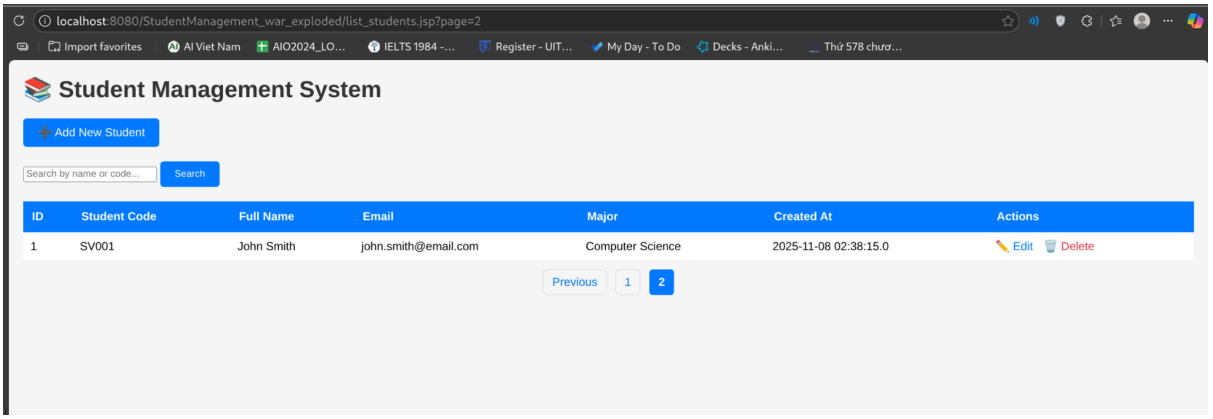
```
<div class="pagination">
   <% if (currentPage > 1) { %>
      <a href="list_students.jsp?page=<%= currentPage - 1 %>">Previous</a>
   <% } %>

   <% for (int i = 1; i <= totalPages; i++) { %>
      <% if (i == currentPage) { %>
         <strong><%= i %></strong>
      <% } else { %>
         <a href="list_students.jsp?page=<%= i %>"><%= i %></a>
```

```
        <% } %>
    <% } %>

    <% if (currentPage < totalPages) { %>
        <a href="list_students.jsp?page=<%= currentPage + 1 %>">Next</a>
    <% } %>
</div>
```

Evaluation Criteria:

Criteria	Points
LIMIT/OFFSET query implemented	3
Page number from URL parameter	2
Total pages calculated correctly	2
Pagination links display	1





7.2: Improved UI/UX (7 points)

Enhance the visual design and user experience.

Requirements:

a) **Success/Error Message Styling (2 points)**

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

```
<script>
setTimeout(function() {
    var messages = document.querySelectorAll('.message');
    messages.forEach(function(msg) {
        msg.style.display = 'none';
    });
}, 3000);
</script>
```

b) **Loading States (2 points)**

- Disable submit button after clicking to prevent double submission
- Show "Processing…" text

```
<script>
function submitForm(form) {
    var btn = form.querySelector('button[type="submit"]');
    btn.disabled = true;
    btn.textContent = 'Processing...';
    return true;
}
</script>

<form onsubmit="return submitForm(this)">
```

c) **Responsive Table (3 points)**

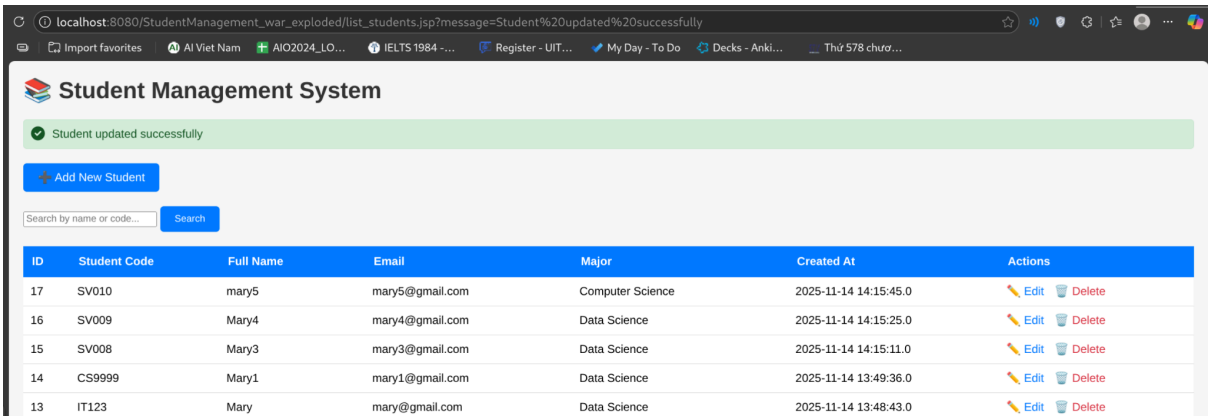
- Table scrollable on small screens
- Better mobile layout

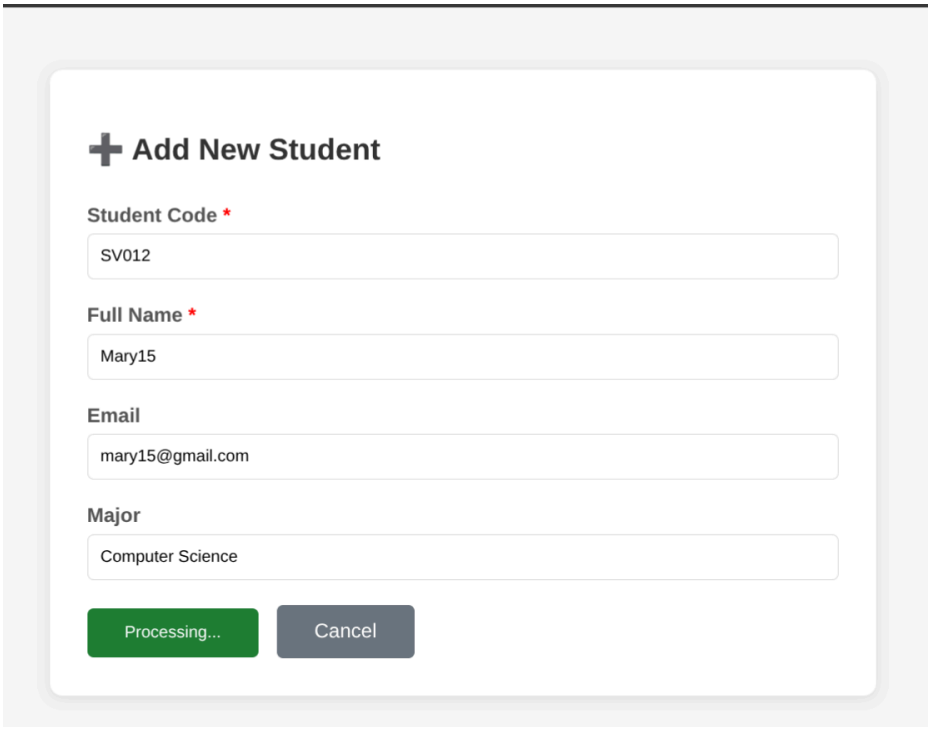
```
.table-responsive {
```

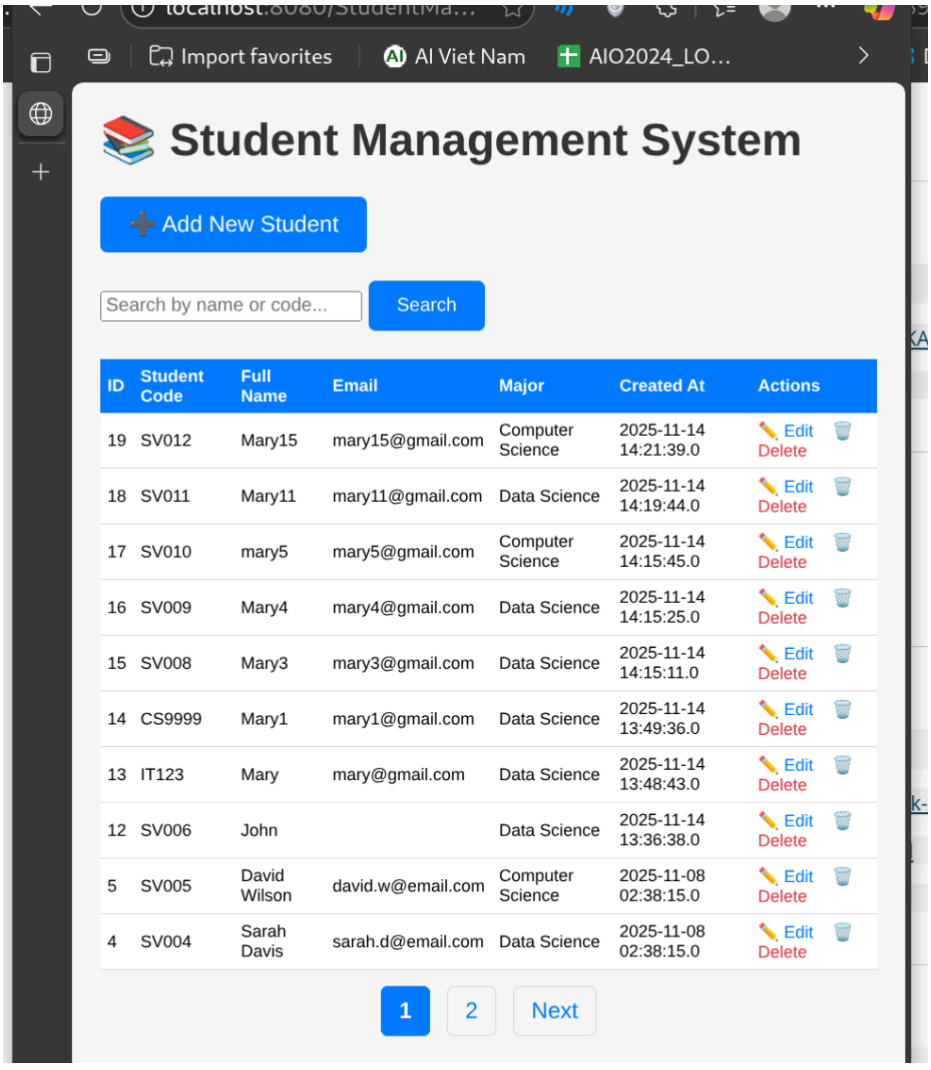
```
    overflow-x: auto;
}

@media (max-width: 768px) {
    table {
        font-size: 12px;
    }
    th, td {
        padding: 5px;
    }
}
```

Evaluation Criteria:

Criteria	Points
Message styling improved	2
Button loading state	2
Responsive design	3




Add New Student
Student Code *
SV012
Full Name *
Mary15
Email
mary15@gmail.com
Major
Computer Science
Processing...
Cancel


Import favorites
AI Viet Nam
AIO2024_LO...
Student Management System
Add New Student
Search by name or code...
Search
ID Student Code Full Name Email Major Created At Actions
19 SV012 Mary15 mary15@gmail.com Computer Science 2025-11-14 14:21:39.0 Edit Delete
18 SV011 Mary11 mary11@gmail.com Data Science 2025-11-14 14:19:44.0 Edit Delete
17 SV010 mary5 mary5@gmail.com Computer Science 2025-11-14 14:15:45.0 Edit Delete
16 SV009 Mary4 mary4@gmail.com Data Science 2025-11-14 14:15:25.0 Edit Delete
15 SV008 Mary3 mary3@gmail.com Data Science 2025-11-14 14:15:11.0 Edit Delete
14 CS9999 Mary1 mary1@gmail.com Data Science 2025-11-14 13:49:36.0 Edit Delete
13 IT123 Mary mary@gmail.com Data Science 2025-11-14 13:48:43.0 Edit Delete
12 SV006 John Data Science 2025-11-14 13:36:38.0 Edit Delete
5 SV005 David Wilson david.w@email.com Computer Science 2025-11-08 02:38:15.0 Edit Delete
4 SV004 Sarah Davis sarah.d@email.com Data Science 2025-11-08 02:38:15.0 Edit Delete
1
2
Next