EXERCISE 1: SETUP AND DISPLAY (15 points)

Estimated Time: 20 minutes

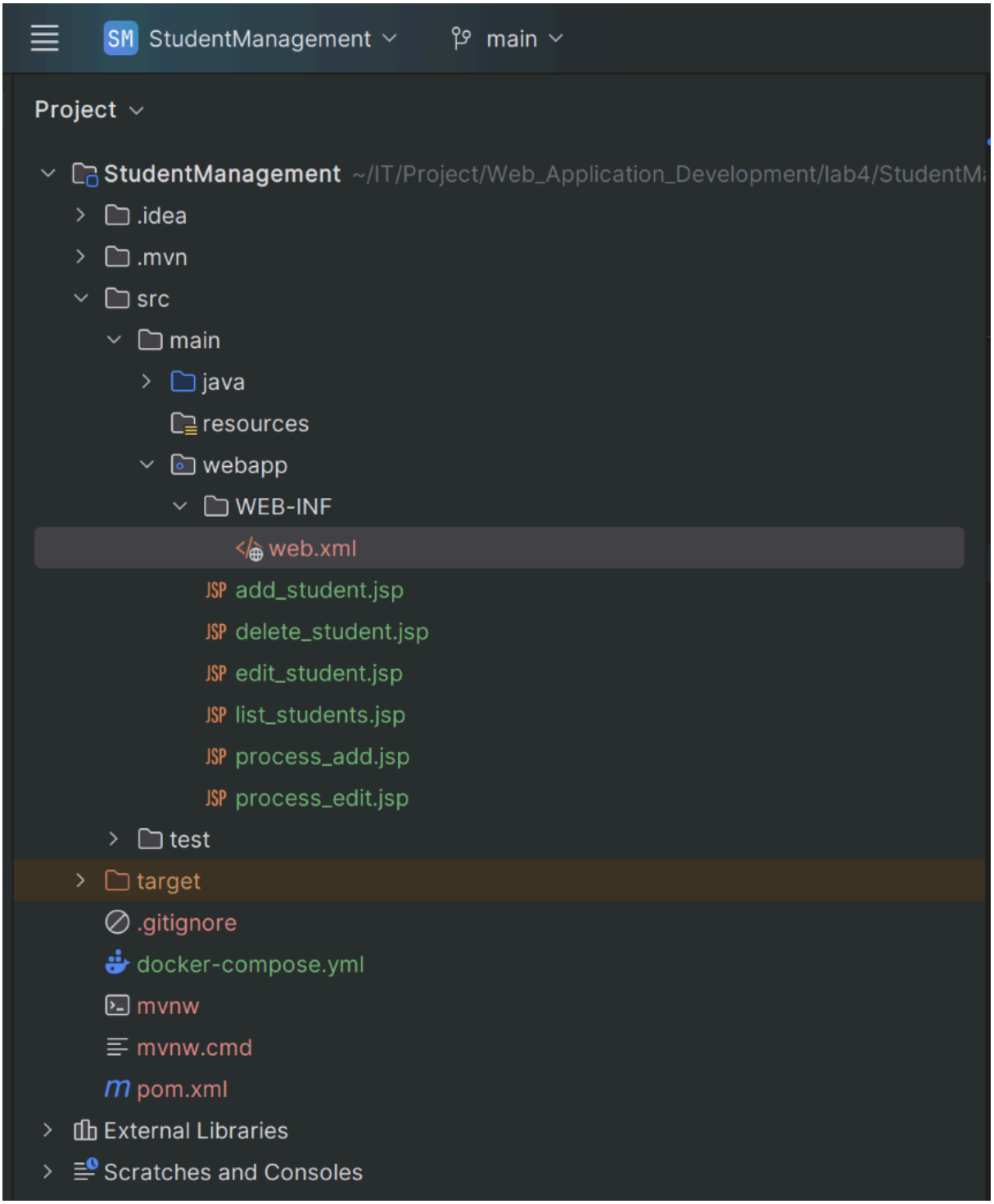
Task 1.1: Project Setup (5 points)

Create a new Web Application project:

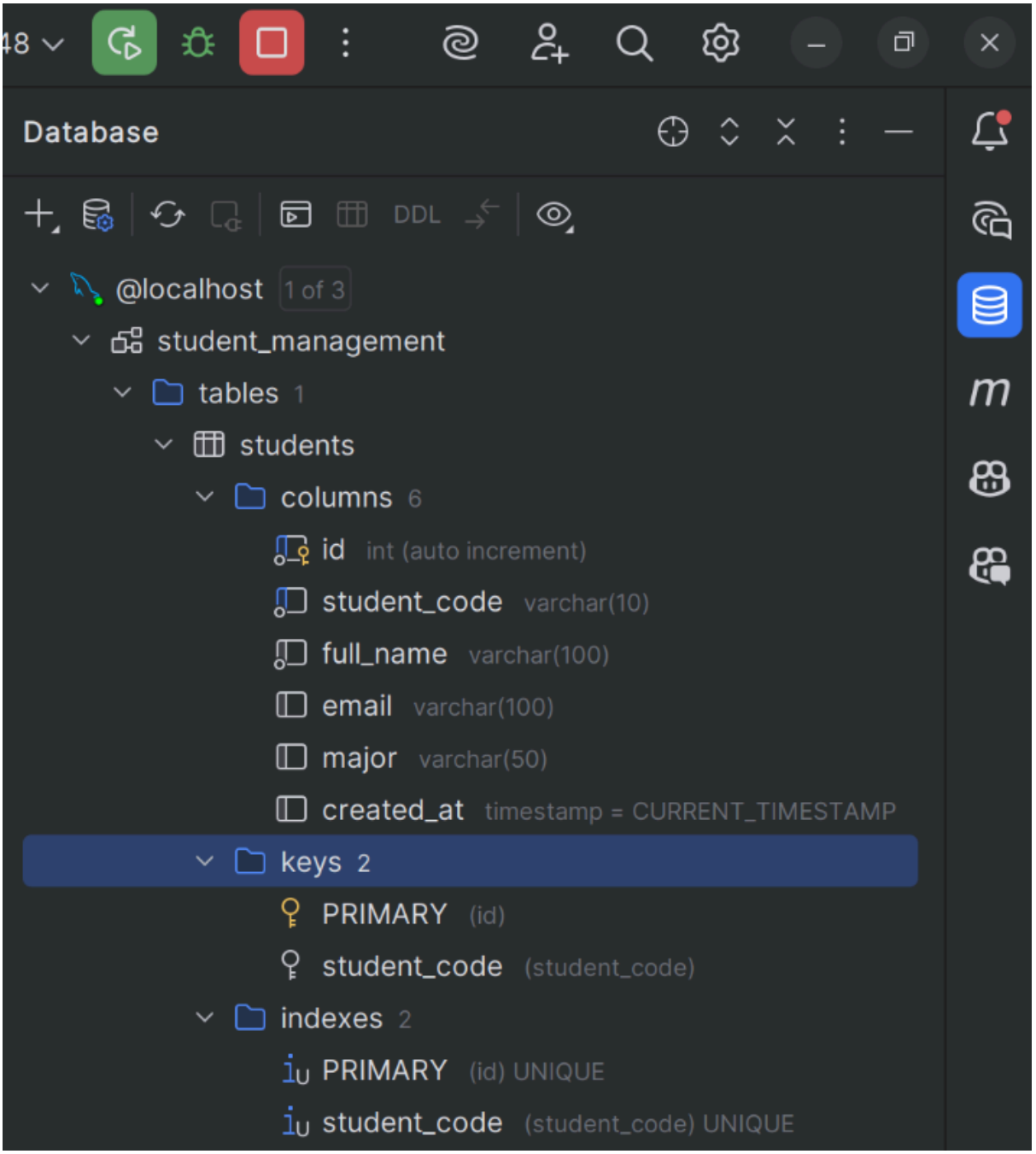
1. Project name: StudentManagement
2. Add MySQL Connector/J library
3. Configure Tomcat server
4. Verify project builds without errors

Deliverables:

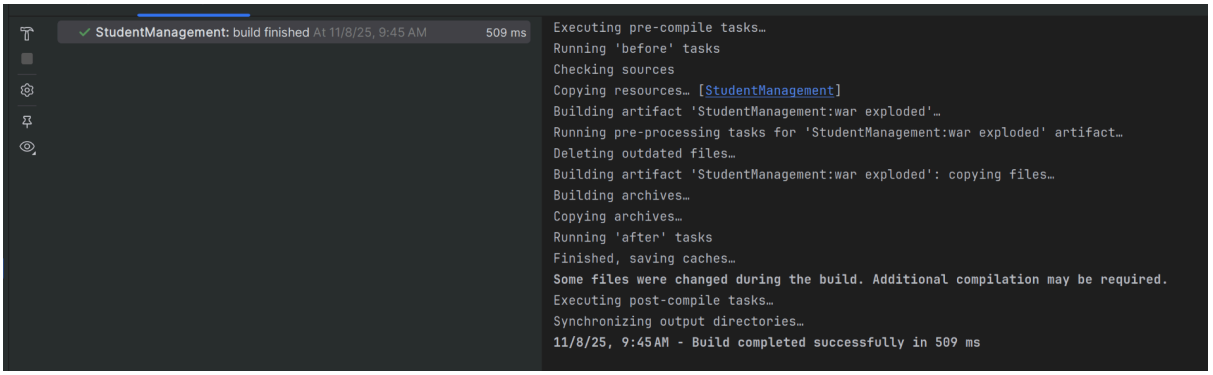
- ✅ Project created successfully
- ✅ Libraries added correctly
- ✅ Test build completes (Green checkmark in NetBeans)



- The project StudentManagement created successfully in IntelliJ



- The MySQL database created and connected successfully in IntelliJ



- Project builded successfully

Task 1.2: Display Student List (10 points)

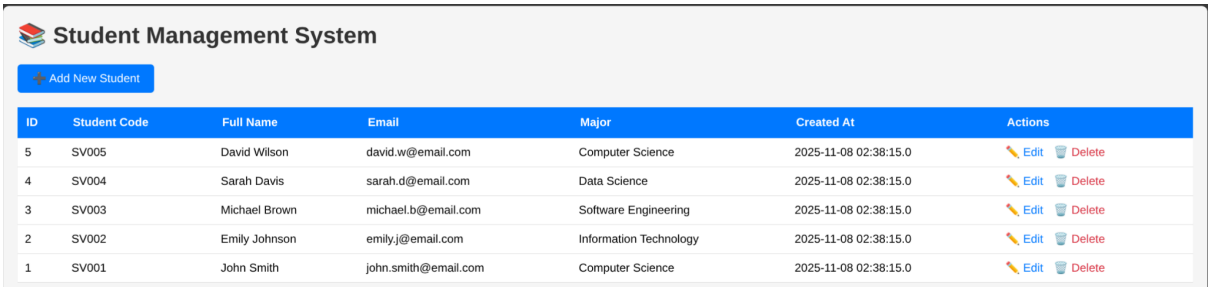
Create list_students.jsp that displays all students in a table.

Requirements:

- Connect to database using JDBC
- Query all students from students table
- Display in HTML table with columns:
  - ID
  - Student Code
  - Full Name
  - Email
  - Major
  - Created At
- Handle database errors gracefully
- Close all database resources properly

Testing:

- Run the page: http://localhost:8080/StudentManagement/list_students.jsp
- Should see 5 sample students
- No error messages displayed



Create the connection with the MySQL Database and query the data in result set (rs)

```
<%
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {
        Class.forName("com.mysql.cj.jdbc.Driver");

        conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3307/student_management",
                "user1",
                "user1"
        );

        stmt = conn.createStatement();
        String sql = "SELECT * FROM students ORDER BY id DESC";
        rs = stmt.executeQuery(sql);

        while (rs.next()) {
            int id = rs.getInt("id");
            String studentCode = rs.getString("student_code");
            String fullName = rs.getString("full_name");
            String email = rs.getString("email");
            String major = rs.getString("major");
            Timestamp createdAt = rs.getTimestamp("created_at");
%>
```

```
Timestamp createdAt = rs.getTimestamp("created_at");
%>
<tr>
    <td><%= id %></td>
    <td><%= studentCode %></td>
    <td><%= fullName %></td>
    <td><%= email != null ? email : "N/A" %></td>
    <td><%= major != null ? major : "N/A" %></td>
    <td><%= createdAt %></td>
    <td>
        <a href="edit_student.jsp?id=<%= id %>" class="action-link">✏ Edit</a>
        <a href="delete_student.jsp?id=<%= id %>"
           class="action-link delete-link"
           onclick="return confirm('Are you sure?')">🗑 Delete</a>
    </td>
</tr>
<%
```

The page loops through the ResultSet using while(rs.next()), which iterates over each student record:

- For each record, it extracts values such as id, student_code, full_name, email, major, and created_at.
- These values are then displayed in an HTML table.
- If email or major is null, the string "N/A" is shown instead.

Each student row also contains two action links:

- **Edit**: A link to edit_student.jsp?id=<student_id> that allows the user to edit the student information.
- **Delete**: A link to delete_student.jsp?id=<student_id> that triggers a confirmation before deleting the student record.

EXERCISE 2: CREATE OPERATION (15 points)

Estimated Time: 30 minutes

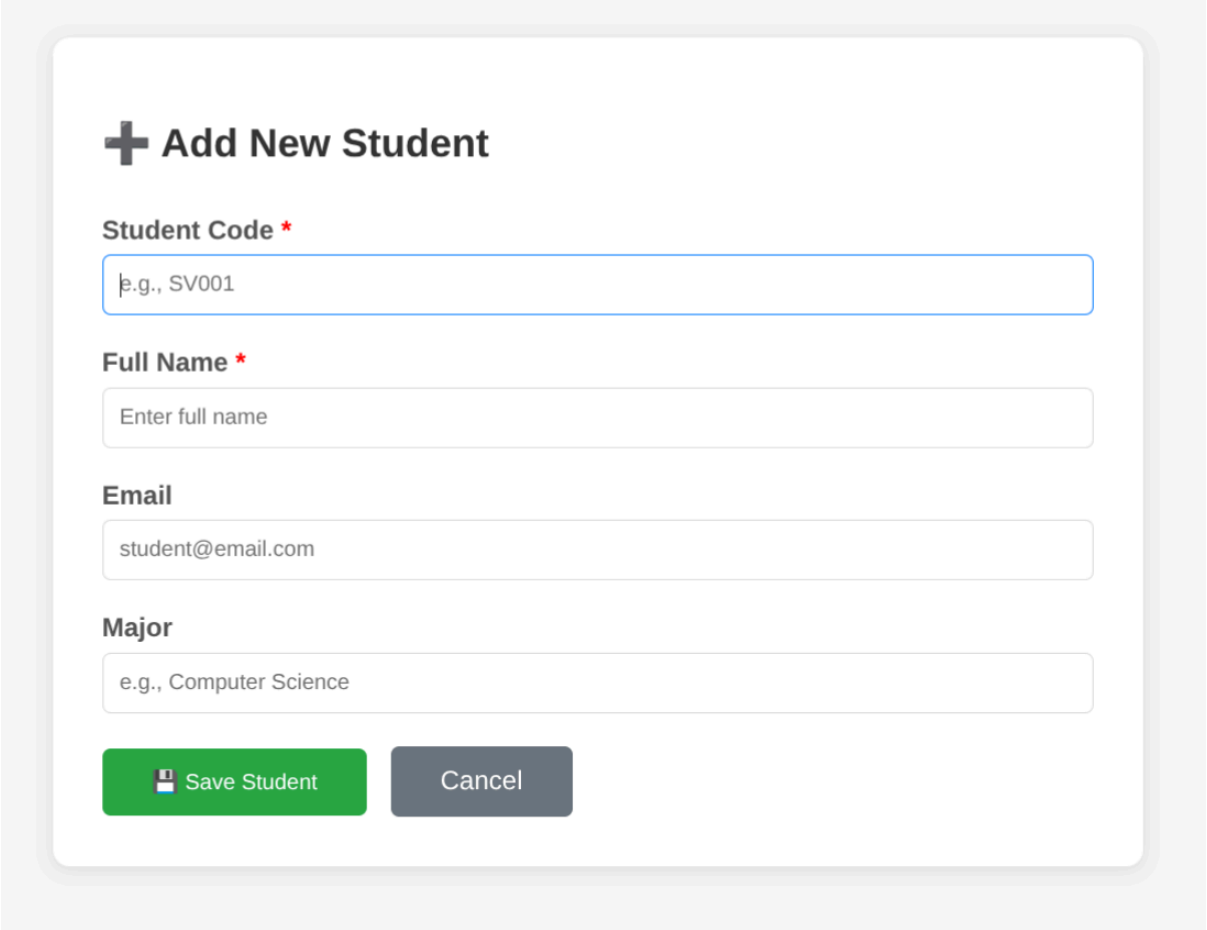
Task 2.1: Create Add Student Form (5 points)

Create add_student.jsp with a form to add new students.

Requirements:

- Form fields:
    - Student Code (text, required)
    - Full Name (text, required)
    - Email (email type, optional)

    - Major (text, optional)
- Submit button
- Cancel button (link back to list)
- Use POST method
- Basic HTML5 validation (required attribute)



- When user click to the button “Add New Student”, it will load the “add_student.jsp” file

---

Task 2.2: Process Add Student (10 points)

Create process_add.jsp to handle form submission.

Requirements:

- Retrieve form parameters using request.getParameter()
- Server-side validation:
    - Student code and full name are required
    - Display error if validation fails
- Insert data using PreparedStatement
- Handle duplicate student code error
- Redirect to list page on success with message

- Display error message on failure

Test Cases:

Test	Input	Expected Result
Valid data	Code: SV006, Name: John Doe	Success, redirects to list
Empty required field	Code: (empty), Name: John	Error: "Required fields missing"
Duplicate code	Code: SV001 (existing)	Error: "Student code already exists"

Input Validation:

- The code retrieves form parameters (student_code, full_name, email, major) from the request.
- It checks if the required fields (student_code and full_name) are provided. If any of them is missing or empty, it redirects the user to add_student.jsp with an error message.

Database Connection:

- It establishes a connection to the MySQL database (student_management) using JDBC with the credentials user1 and user1.

SQL Insert:

- The code prepares an SQL INSERT statement to add a new student to the students table with the provided data.
- The PreparedStatement is used to safely insert the student information.

Execute and Handle Results:

- The executeUpdate() method is called to execute the INSERT statement. It returns the number of affected rows.
- If the insert is successful (rowsAffected > 0), it redirects to list_students.jsp with a success message.
- If the insert fails, it redirects back to add_student.jsp with an error message.

Error Handling:

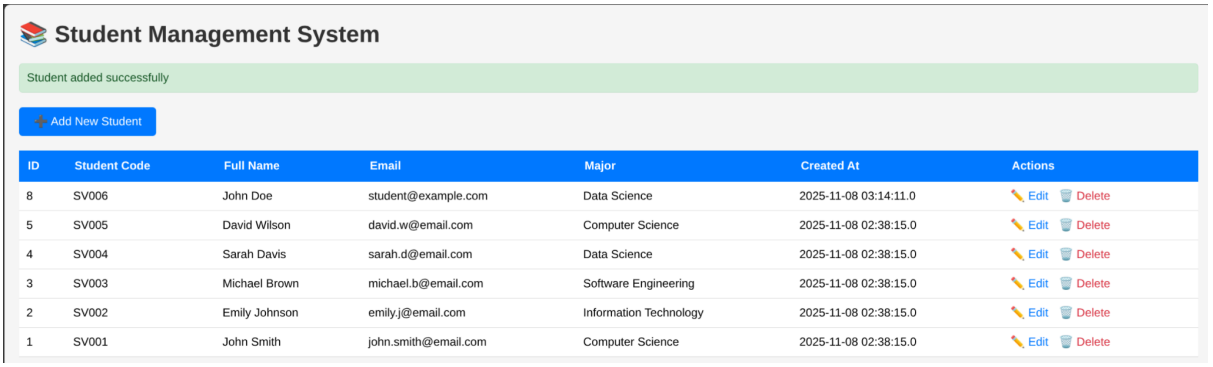
- If a ClassNotFoundException or SQLException occurs, the appropriate error message is displayed on the add_student.jsp page.

- Specific errors like "Duplicate entry" are handled separately, e.g., if the student code already exists.

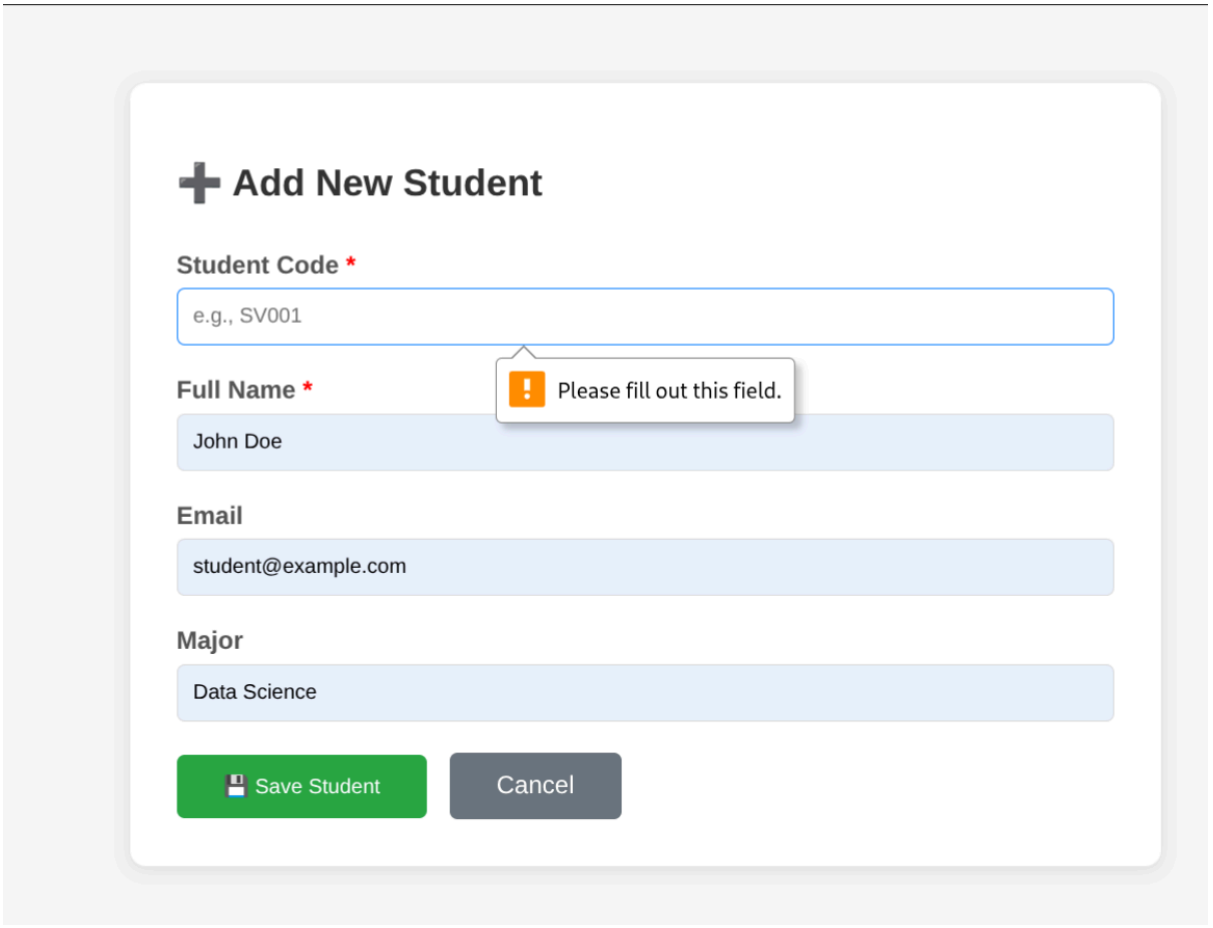
Resource Cleanup:

- Finally, the PreparedStatement and Connection are closed to release resources.

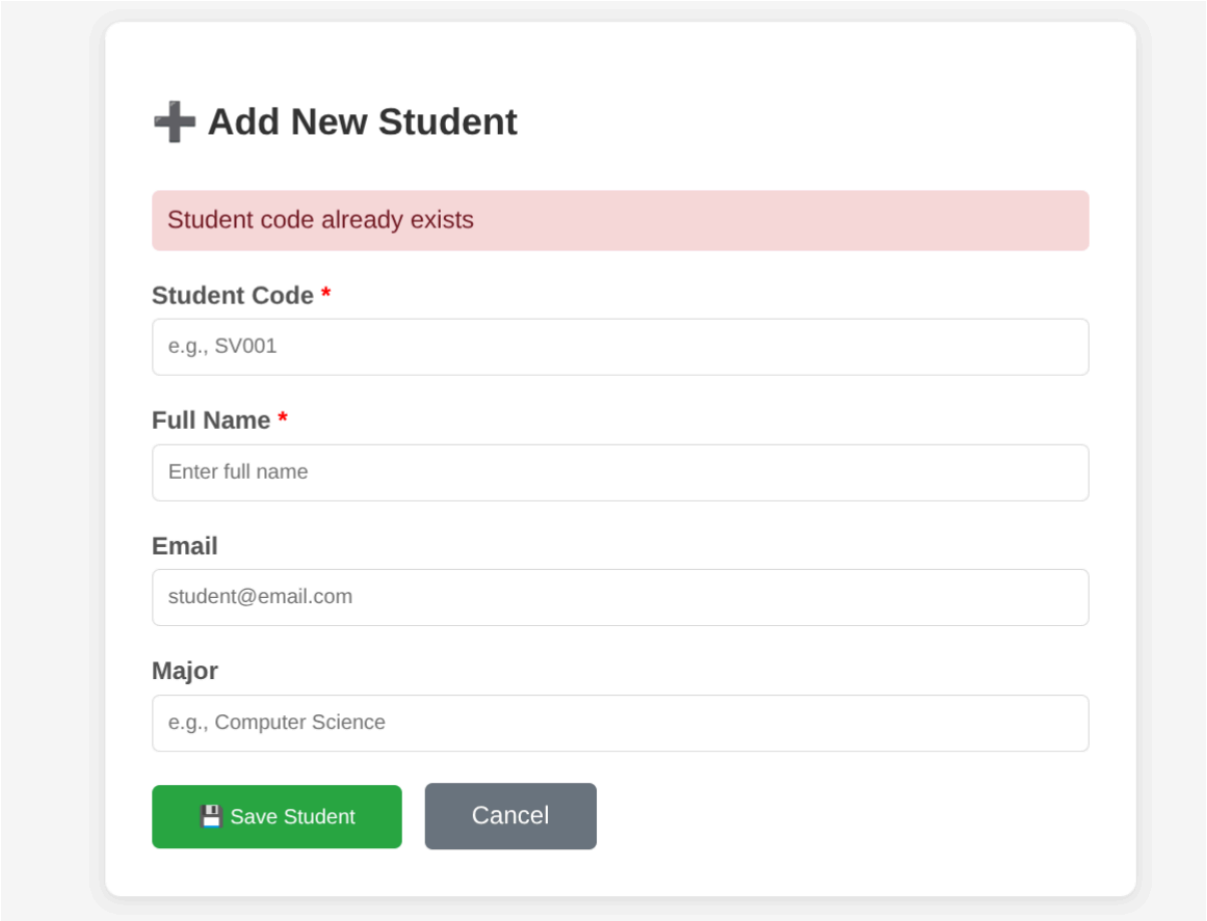
Testing



- Adding student successfully



- Empty required field



- Duplicate code

EXERCISE 3: UPDATE OPERATION (15 points)

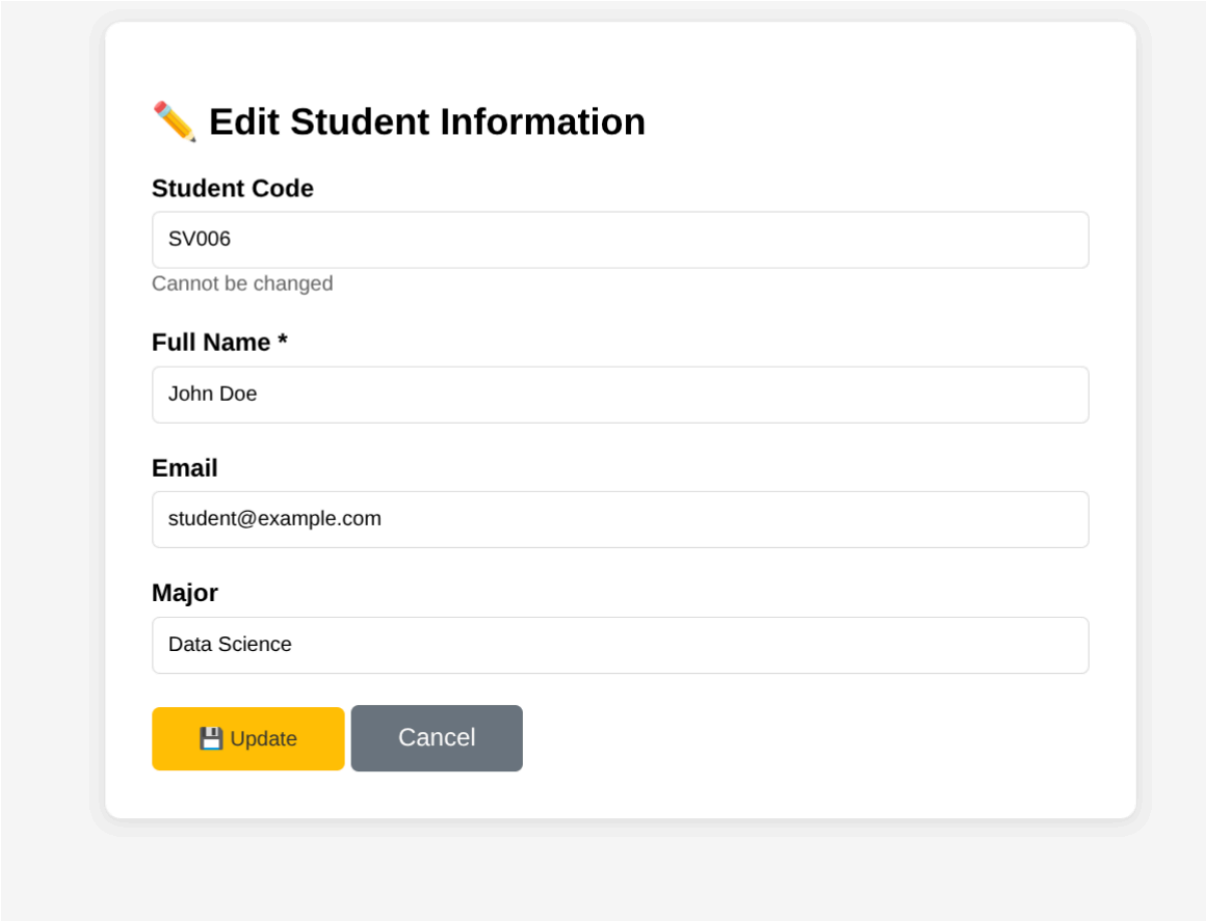
Estimated Time: 35 minutes

Task 3.1: Create Edit Form (7 points)

Create edit_student.jsp that displays a pre-filled form.

Requirements:

- Get student ID from URL parameter (?id=1)
- Query database for student with that ID
- Display form with pre-filled values
- Student code field should be readonly
- Handle case where student ID doesn't exist
- Include hidden input for student ID



Task 3.2: Process Update (8 points)

Create process_edit.jsp to handle update.

Requirements:

- Retrieve student ID and form data
- Validate inputs
- Update database using PreparedStatement
- Redirect to list on success
- Display error on failure

Test Cases:

Test	Action	Expected Result
Valid update	Change name from "John Doe" to "John Smith"	Success, list shows updated name
Invalid ID	Access edit page with id=999	Error: "Student not found"

Empty name	Submit with blank name	Error: "Required field missing"

Input Validation:

- The code retrieves the student id, full_name, email, and major parameters from the request.
- It checks if the id and full_name are provided. If either is missing or invalid (e.g., empty full_name), it redirects to list_students.jsp with an error message.

Parsing Student ID:

- The id parameter is parsed into an integer (studentId) for use in the database query.

Database Connection:

- The code establishes a connection to the MySQL database (student_management) using JDBC with the credentials user1 and user1.

SQL Update Statement:

- It prepares an SQL UPDATE statement to modify the student's full_name, email, and major based on the provided id.
- The PreparedStatement is used to safely insert the user input values into the query.

Execute and Handle Results:

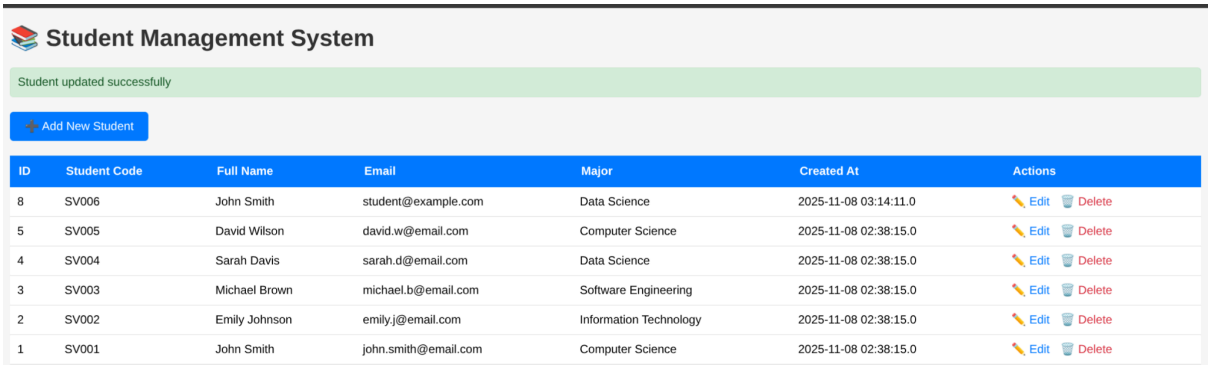
- The executeUpdate() method is called to execute the UPDATE statement. It returns the number of affected rows.
- If the update is successful (rowsAffected > 0), it redirects to list_students.jsp with a success message.
- If the update fails (no rows affected), it redirects back to edit_student.jsp with an error message.

Error Handling:

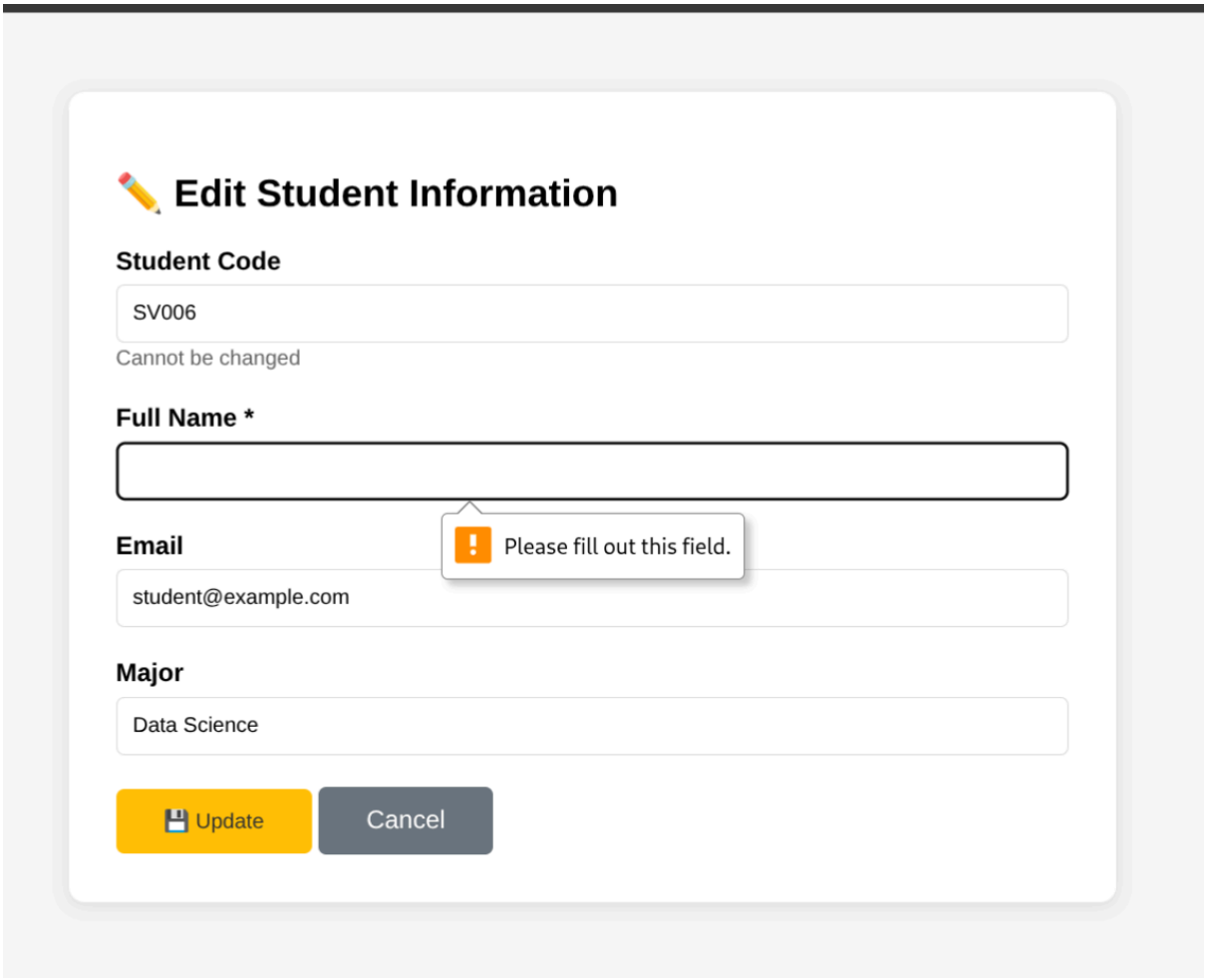
- If any exception occurs (e.g., SQLException or ClassNotFoundException), an error message is sent back to the edit_student.jsp page.
- The exception details are printed for debugging purposes.

Resource Cleanup:

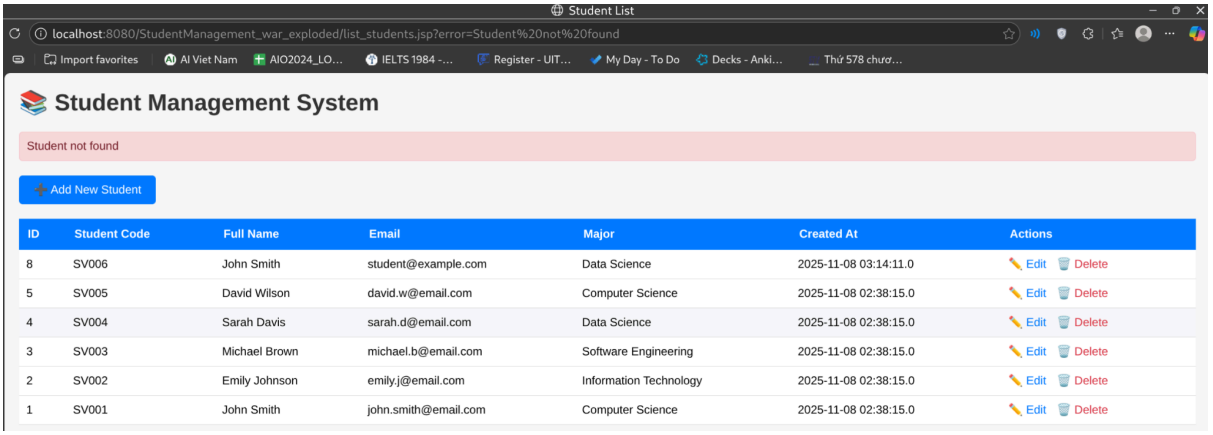
- The PreparedStatement and Connection are closed in the finally block to release resources.



- Update “John Doe” to “John Smith” successfully



- Submit with blank name



- Access edit page with id=999

EXERCISE 4: DELETE OPERATION (15 points)

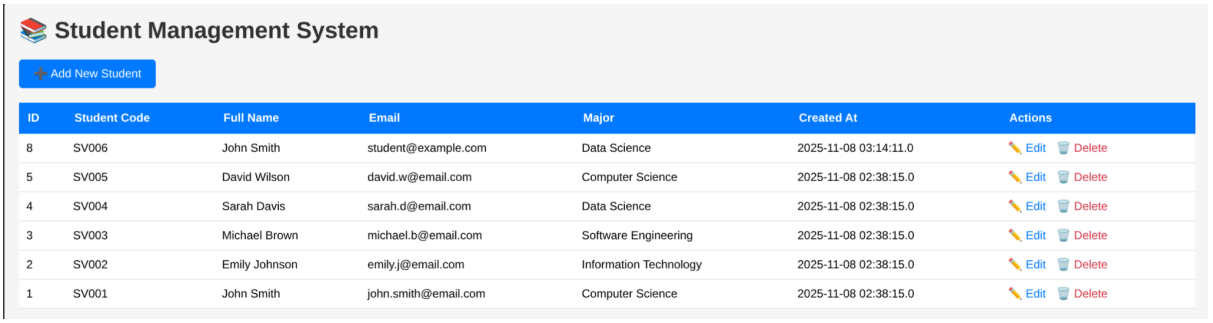
Estimated Time: 25 minutes

Task 4.1: Implement Delete (10 points)

Create delete_student.jsp to delete a student.

Requirements:

- Get student ID from URL parameter
- Delete record from database
- Redirect to list with message
- Handle errors (ID not found, database errors)



Get student ID from URL parameter:

- The student ID is retrieved from the URL using request.getParameter("id"). This ID is used to identify the student record that needs to be deleted.

Delete record from database:

- A DELETE SQL query is prepared and executed using the student ID. The query deletes the student record where the id matches the provided parameter.

Redirect to list with message:

- After the deletion, the page redirects to list_students.jsp. If the deletion was successful, a success message is passed via the query string (message=Student deleted successfully). If no record was found for the given ID, an error message is passed (error=Student not found).

Handle errors (ID not found, database errors):

- If the student ID is missing or invalid, the code redirects to the list page with an error message (error=Invalid student ID).
- If any database issues occur (e.g., SQL errors), an error message is shown (error=Database error: <error_message>).
- If the student ID does not exist in the database, it redirects with error=Student not found.

Task 4.2: Add Delete Links and Confirmation (5 points)

Modify list_students.jsp to add delete functionality.

Requirements:

- Add "Delete" link for each student in table
- Link format: delete_student.jsp?id=[student_id]
- Add JavaScript confirmation dialog
- Style delete link in red color

Code Example:

```
<a href="delete_student.jsp?id=<%= student.getId() %>"
   class="delete-link"
   onclick="return confirm('Are you sure you want to delete this student?')">
   Delete
</a>
```

Testing:

- Click delete on test student
- Confirm in dialog → Student should be deleted
- Cancel in dialog → Student should remain

Get the ID Parameter:

- The code retrieves the id parameter from the URL using request.getParameter("id").
- If the id is missing or empty, it redirects to list_students.jsp with an error message: Invalid student ID.

Validate and Parse the ID:

- The code tries to convert the id parameter to an integer using Integer.parseInt().
- If the id cannot be parsed (i.e., it's not a valid integer), the page redirects with an error message: Invalid student ID format.

Database Connection:

- A JDBC connection to the MySQL database is established with the provided credentials (user1 / user1).

Prepare and Execute the DELETE Query:

- A DELETE query is prepared with a PreparedStatement to remove the student record where the id matches the provided student ID.
- The setInt(1, studentId) method sets the student ID as the parameter for the DELETE query.

Handle Query Execution:

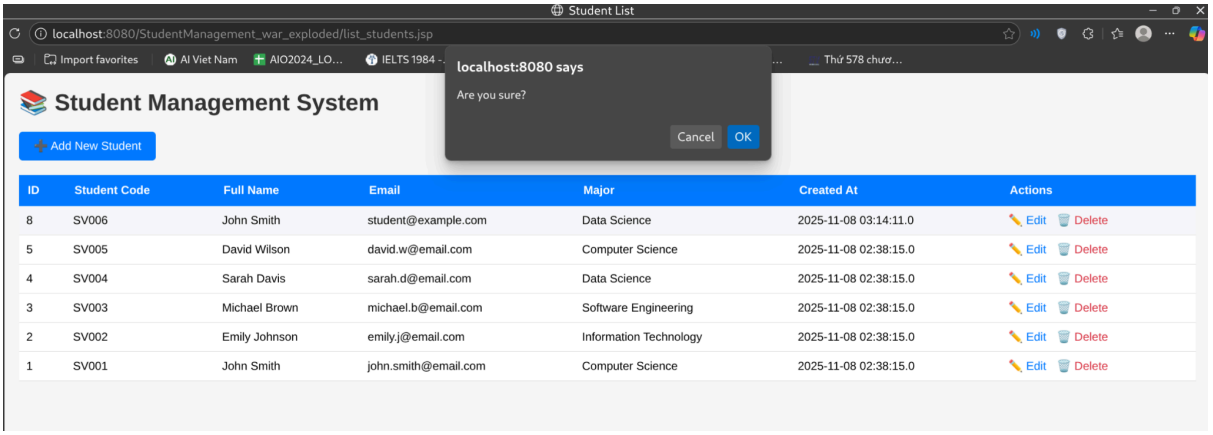
- The executeUpdate() method executes the DELETE query. It returns the number of rows affected.
- If the query successfully deletes the student (i.e., rowsAffected > 0), the page redirects to list_students.jsp with a success message.
- If no student with the given ID is found (i.e., rowsAffected == 0), it redirects with an error message: Student not found.

Error Handling:

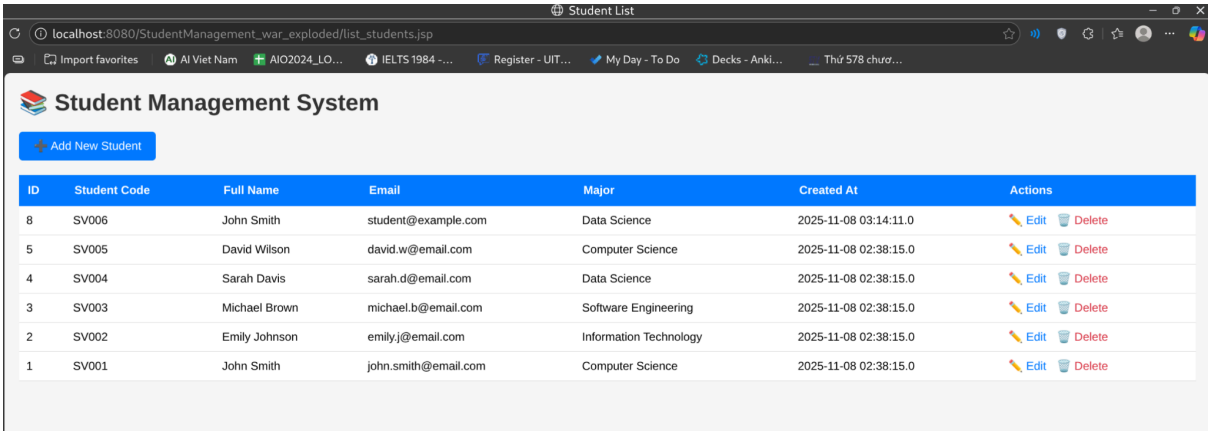
- **JDBC Driver not found**: If Class.forName() fails, it indicates that the JDBC driver is missing.
- **SQL Exception**: Catches any SQL errors, such as database connection issues or query execution problems.
- All exceptions are logged, and the page redirects to list_students.jsp with an appropriate error message.

Resource Cleanup:

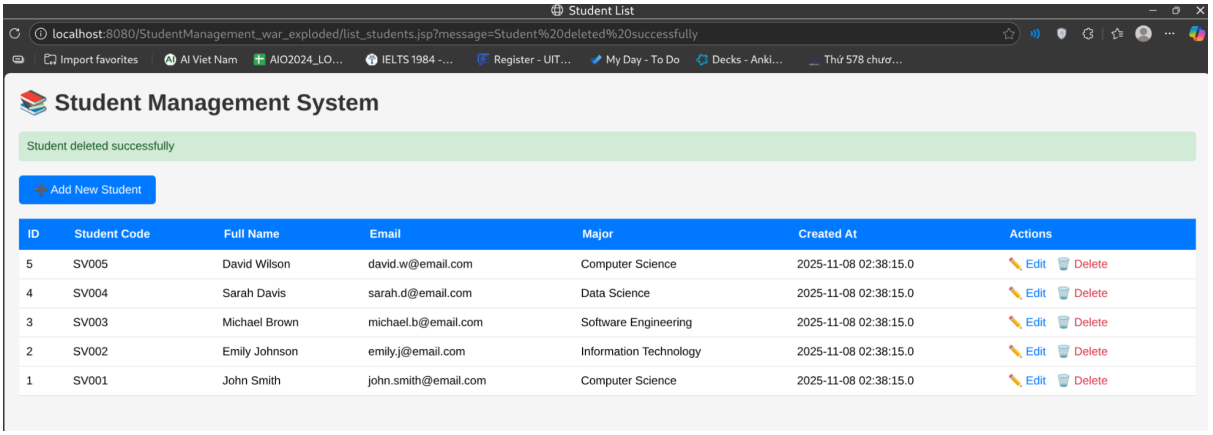
- The PreparedStatement and Connection objects are closed in the finally block to release the resources.



- Click on “Delete” button



- Click “Cancel”, the student information remain



- Click “Ok”, delete student successfully