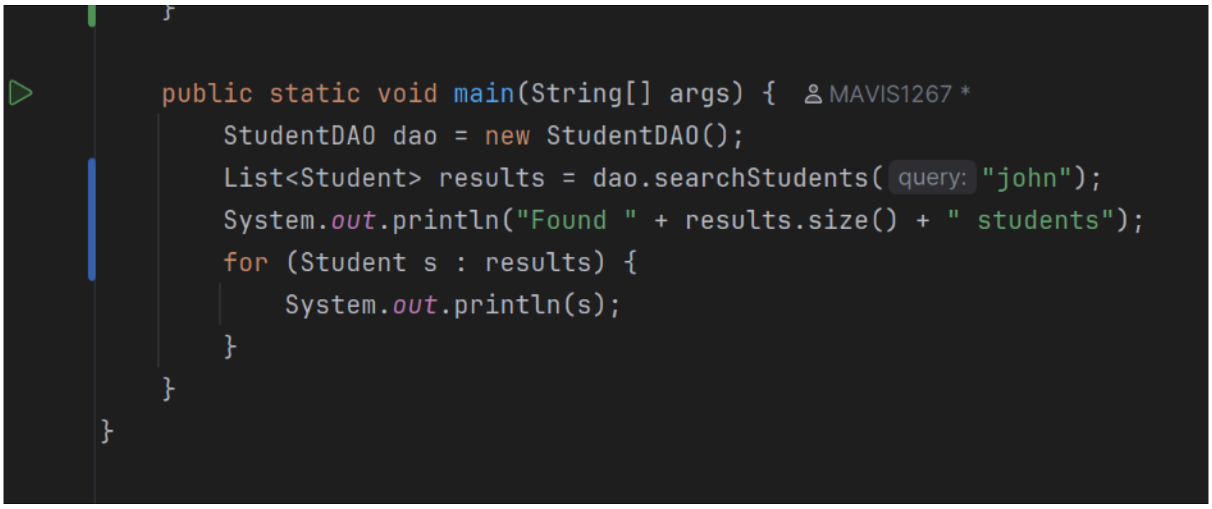
5.1 Update StudentDAO

- Task: Add searchStudents(String keyword) method to StudentDAO.java

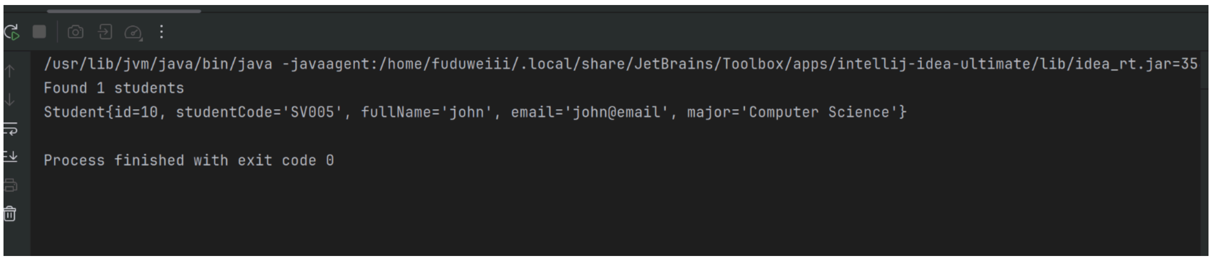
```java
public List<Student> searchStudents(String query) {
  List<Student> result = new ArrayList<>();
  if (query == null || query.trim().length() == 0) {
     return result;
  }

  String sql = "SELECT * FROM students " +
         "WHERE student_code LIKE ? " +
         "OR full_name LIKE ? " +
         "OR email LIKE ? " +
         "ORDER BY id DESC";

  try (Connection c = getConnection();
      PreparedStatement ps = c.prepareStatement(sql)) {

     String val = "%" + query.trim() + "%";

     for (int i = 1; i <= 3; i++) {
        ps.setString(i, val);
     }

     ResultSet rs = ps.executeQuery();
     while (rs.next()) {
        Student s = new Student();
        s.setId(rs.getInt("id"));
        s.setStudentCode(rs.getString("student_code"));
        s.setFullName(rs.getString("full_name"));
        s.setEmail(rs.getString("email"));
        s.setMajor(rs.getString("major"));
        s.setCreatedAt(rs.getTimestamp("created_at"));

        result.add(s);
     }
     rs.close();

  } catch (Exception ex) {
     System.out.println("Search error: " + ex.getMessage());
  }
  return result;
}
```

- Validation: The method first checks the input query. If it is null or empty, an empty list is returned immediately.
- SQL Preparation: A SQL statement is defined using the LIKE operator combined with OR to search across three columns (student_code, full_name, email), ordered by ID.
- Execution:
  + A try-with-resources block is used to securely open the connection.
  + The search keyword is wrapped with wildcards (%keyword%) to enable partial matching.

    + This value is set into the PreparedStatement for all three query parameters.
- Mapping & Return: The method executes the query, iterates through the ResultSet to map database rows into Student objects, adds them to a list, and returns the final collection.



```
    }

    public static void main(String[] args) {  MAVIS1267 *
        StudentDAO dao = new StudentDAO();
        List<Student> results = dao.searchStudents( query: "john");
        System.out.println("Found " + results.size() + " students");
        for (Student s : results) {
            System.out.println(s);
        }
    }
}
```



```
/usr/lib/jvm/java/bin/java -javaagent:/home/fuduweiii/.local/share/JetBrains/Toolbox/apps/intellij-idea-ultimate/lib/idea_rt.jar=35
Found 1 students
Student{id=10, studentCode='SV005', fullName='john', email='john@email', major='Computer Science'}

Process finished with exit code 0
```

- Test function work successfully

5.2 Add Search Controller Method

- Add search handling to StudentController.java

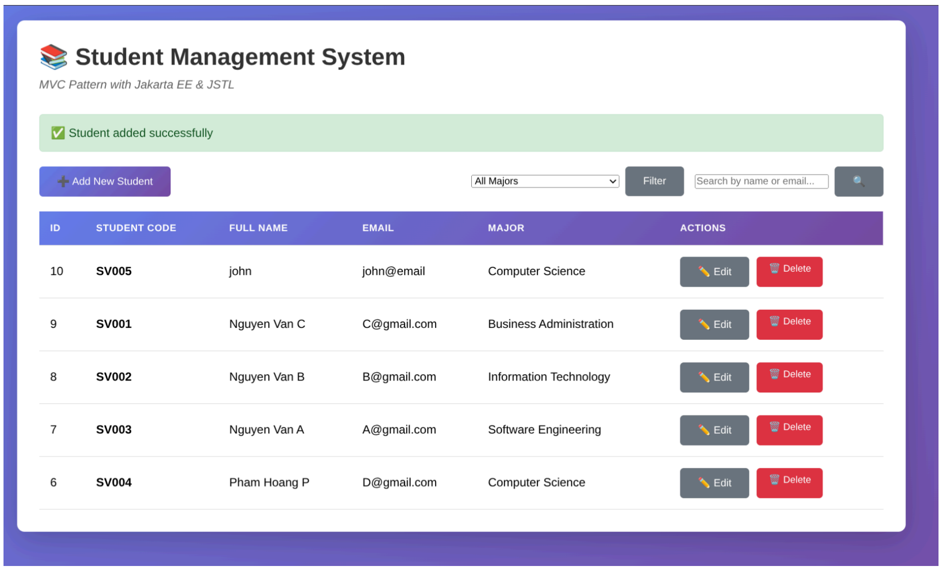
```
private void searchStudents(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
  String searchTxt = request.getParameter("keyword");
  List<Student> students;

  if (searchTxt != null && !searchTxt.trim().isEmpty()) {
    students = studentDAO.searchStudents(searchTxt);
  } else {
    students = studentDAO.getAllStudents();
  }

  request.setAttribute("students", students);
  request.setAttribute("keyword", searchTxt);
  request.getRequestDispatcher("./views/student-list.jsp").forward(request, response);
}
```

- Parameter Retrieval:
    + The method begins by retrieving the user's input from the HTTP request using request.getParameter("keyword"). This value is stored in the searchTxt variable.
- Conditional Logic & Graceful Handling:
    + A conditional check is performed to ensure the keyword is valid: if (searchTxt != null && !searchTxt.trim().isEmpty()).
    + If valid: The controller calls studentDAO.searchStudents(searchTxt) to retrieve filtered results from the database.
    + If null or empty: The controller gracefully handles this by calling studentDAO.getAllStudents(). This ensures the user sees the full list instead of an empty table or an error if they accidentally submit an empty search.
- Attribute Setting:
    + students: The list of result objects is set as a request attribute.
    + keyword: The original search term is also set as a request attribute (request.setAttribute("keyword", searchTxt)). This allows the View (JSP) to preserve the text in the input box after the search is complete.
- View Forwarding:
    + The request is forwarded to the ./views/student-list.jsp page using RequestDispatcher. This preserves the request attributes for display.
- Integration in doGet:
    + To make this method accessible, a new case was added to the switch statement in the doGet method

5.3 Update Student List View

- Task: add search form to student-list.jsp

```
<!-- Toolbar with Add Button and Search Form -->
<div class="toolbar">
  <!-- Left Side: Add Button -->
  <a href="student?action=new" class="btn btn-primary">
    ➕ Add New Student
  </a>

  <!-- Right Side: Search Form -->
  <div class="search-box">
    <form action="student" method="get">
      <!-- Hidden action field -->
      <input type="hidden" name="action" value="search">

      <!-- Keyword Input (Value preserved) -->
      <input type="text"
           name="keyword"
           class="search-input"
           value="${keyword}"
           placeholder="Search by name or email...">

      <!-- Search Button -->
      <button type="submit" class="btn btn-secondary">🔍</button>

      <!-- Conditional Clear Button -->
      <c:if test="${not empty keyword}">
        <a href="student?action=list" class="btn btn-outline">Clear</a>
      </c:if>
    </form>
  </div>
</div>

<!-- Search Feedback Message -->
<c:if test="${not empty keyword}">
  <div class="message info">
    Search results for: <strong>${keyword}</strong>
  </div>
</c:if>
```



📚 Student Management System

MVC Pattern with Jakarta EE & JSTL

✅ Student added successfully

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
10	SV005	john	john@email	Computer Science	✏️ Edit 🗑️ Delete
9	SV001	Nguyen Van C	C@gmail.com	Business Administration	✏️ Edit 🗑️ Delete
8	SV002	Nguyen Van B	B@gmail.com	Information Technology	✏️ Edit 🗑️ Delete
7	SV003	Nguyen Van A	A@gmail.com	Software Engineering	✏️ Edit 🗑️ Delete
6	SV004	Pham Hoang P	D@gmail.com	Computer Science	✏️ Edit 🗑️ Delete

6.1: Create Validation Method (5 points)

- **Task:** Add validateStudent() method to StudentController.java

```
private boolean validateStudent(Student student, HttpServletRequest request)
{
   boolean isValid = true;

   String code = student.getStudentCode();
   String codePattern = "[A-Z]{2}[0-9]{3,}";

   if (code == null || code.trim().isEmpty()) {
       request.setAttribute("errorCode", "Student code is required");
       isValid = false;
   } else if (!code.matches(codePattern)) {
       request.setAttribute("errorCode", "Invalid format. Use 2 letters + 3+
digits (e.g., SV001)");
       isValid = false;
   }

   String name = student.getFullName();
   if (name == null || name.trim().isEmpty()) {
       request.setAttribute("errorName", "Full name is required");
       isValid = false;
   } else if (name.trim().length() < 2) {
       request.setAttribute("errorName", "Name must be at least 2
characters");
       isValid = false;
   }

   String email = student.getEmail();
   String emailPattern = "^[A-Za-z0-9+_.-]+@(.+)$";

   if (email != null && !email.trim().isEmpty()) {
       if (!email.matches(emailPattern)) {
           request.setAttribute("errorEmail", "Invalid email format");
           isValid = false;
       }
   }

   String major = student.getMajor();
   if (major == null || major.trim().isEmpty()) {
       request.setAttribute("errorMajor", "Major is required");
       isValid = false;
   }

   return isValid;
}
```

- Initialization:
  + The method initializes a boolean flag isValid to true. This flag will be switched to false if any validation rule is violated.
- Student Code Validation:

    + The method retrieves the student code and defines a regex pattern ([A-Z]{2}[0-9]{3,}).
    + Required Check: It first checks if the code is null or empty. If so, it sets the errorCode request attribute.
    + Format Check: If the code exists, it checks against the regex pattern (ensuring 2 uppercase letters followed by at least 3 digits). If the format is incorrect, a specific error message is set in errorCode.
- Full Name Validation:
    + Required Check: It checks if the name is null or empty.
    + Length Check: It verifies that the name contains at least 2 characters.
    + If either fails, the errorName attribute is set, and isValid becomes false.
- Email Validation (Conditional):
    + The logic checks the email only if the user has provided input (email != null and not empty).
    + If input exists, it validates against a standard email regex pattern (^[A-Za-z0-9+_.-]+@(.+)$).
    + If the format is invalid, the errorEmail attribute is set.
- Major Validation:
    + It performs a simple check to ensure the Major field is selected (not null or empty). If missing, the errorMajor attribute is set.
- Return Result:
    + The method returns the final state of isValid. If it returns true, the controller proceeds with the DAO operation; if false, the controller will reload the form with error messages.

6.2: Integrate Validation into Insert/Update (3 points)

- **Task:** Use validation in insertStudent() and updateStudent() methods

```java
private void updateStudent(HttpServletRequest request, HttpServletResponse
response)

       throws ServletException, IOException {


   int id = Integer.parseInt(request.getParameter("id"));

   String studentCode = request.getParameter("studentCode");

   String fullName = request.getParameter("fullName");

   String email = request.getParameter("email");

   String major = request.getParameter("major");


   Student student = new Student(studentCode, fullName, email, major);

   student.setId(id);


   if (!validateStudent(student, request)) {

       request.setAttribute("student", student);

       RequestDispatcher dispatcher =
request.getRequestDispatcher("/views/student-form.jsp");

       dispatcher.forward(request, response);

       return;

   }



   if (studentDAO.updateStudent(student)) {

       response.sendRedirect("student?action=list&message=Student updated
successfully");

   } else {

       response.sendRedirect("student?action=list&error=Failed to update
student");

   }
}
```

```java
private void insertStudent(HttpServletRequest request, HttpServletResponse
response)

       throws ServletException, IOException {


   String studentCode = request.getParameter("studentCode");

   String fullName = request.getParameter("fullName");

   String email = request.getParameter("email");

   String major = request.getParameter("major");


   Student newStudent = new Student(studentCode, fullName, email, major);


   if (!validateStudent(newStudent, request)) {

       request.setAttribute("student", newStudent);

       RequestDispatcher dispatcher =
request.getRequestDispatcher("/views/student-form.jsp");

       dispatcher.forward(request, response);

       return;

   }


   if (studentDAO.addStudent(newStudent)) {

       response.sendRedirect("student?action=list&message=Student added
successfully");

   } else {

       response.sendRedirect("student?action=list&error=Failed to add
student");

   }

}
```

- Data Binding:
  + Both methods start by extracting form parameters (studentCode, fullName, etc.) from the HTTP request.
  + A Student object is instantiated and populated with these values.
- Validation Guard Clause:
  + The code calls validateStudent(student, request).
  + IF Validation Fails (!valid):

    Preserve Input: The student object (containing the data the user just typed) is saved back into the request scope using request.setAttribute("student", student). This ensures the form fields remain filled when the page reloads.

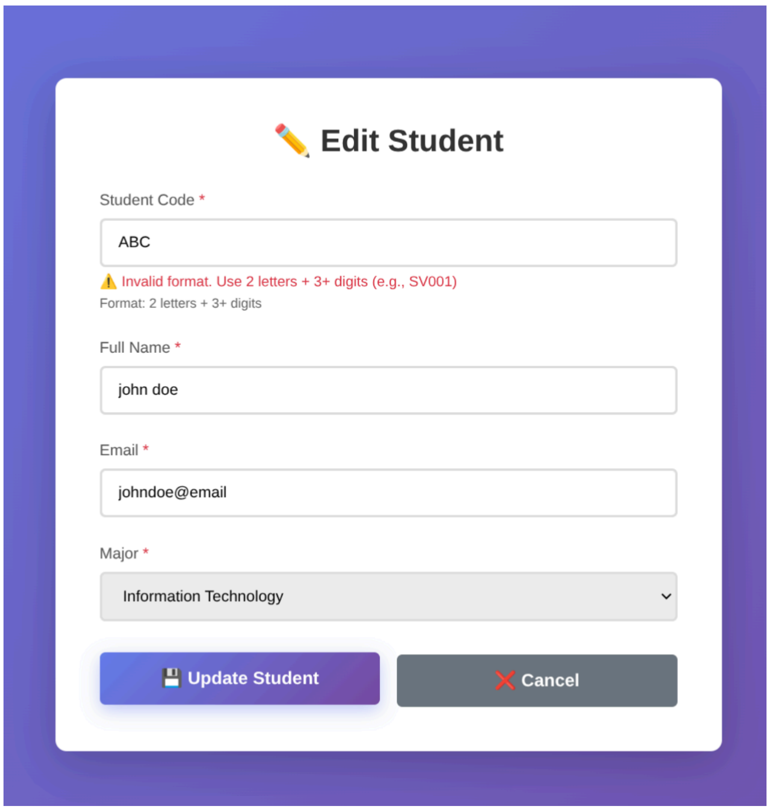
    Forwarding: The request is forwarded back to student-form.jsp. Since validateStudent has already set the error message attributes (e.g., errorEmail), the JSP will display them.

    Stop Execution: The return; statement is crucial here. It immediately terminates the method, preventing any Database access code from running.

- Success Path (DAO Execution):
  + If the validation returns true (passes), the code proceeds to the next block.
  + It calls studentDAO.addStudent (or updateStudent).
  + Upon success, it sends a sendRedirect to the list page with a success message.

6.3: Display Validation Errors in Form (2 points)

- **Task:** Update student-form.jsp to show validation errors



✏️ Edit Student

Student Code *

ABC

⚠️ Invalid format. Use 2 letters + 3+ digits (e.g., SV001)

Format: 2 letters + 3+ digits

Full Name *

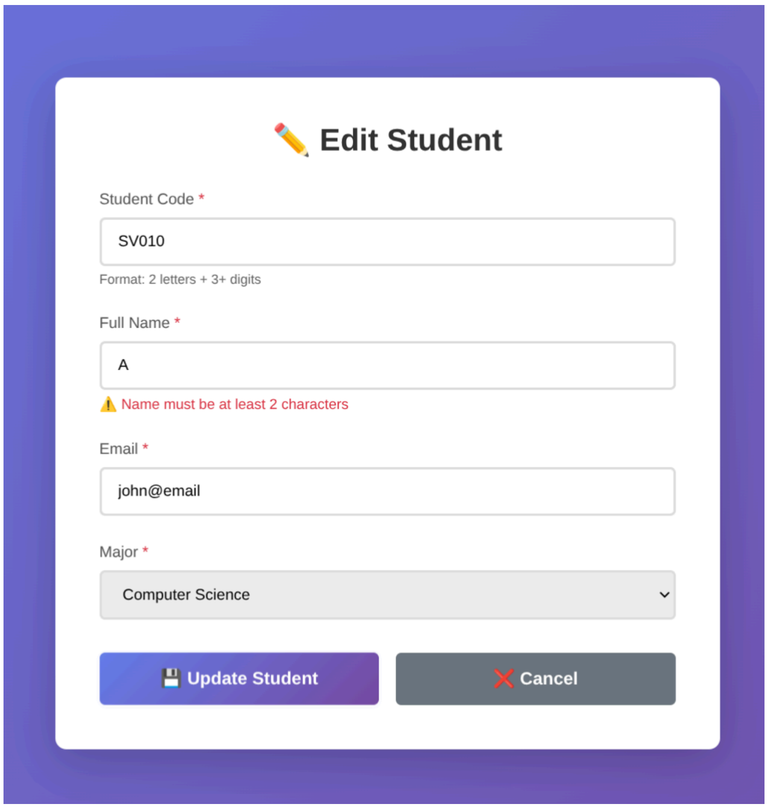
john doe

Email *

johndoe@email

Major *

Information Technology



✏️ Edit Student

Student Code *

SV010

Format: 2 letters + 3+ digits

Full Name *

A

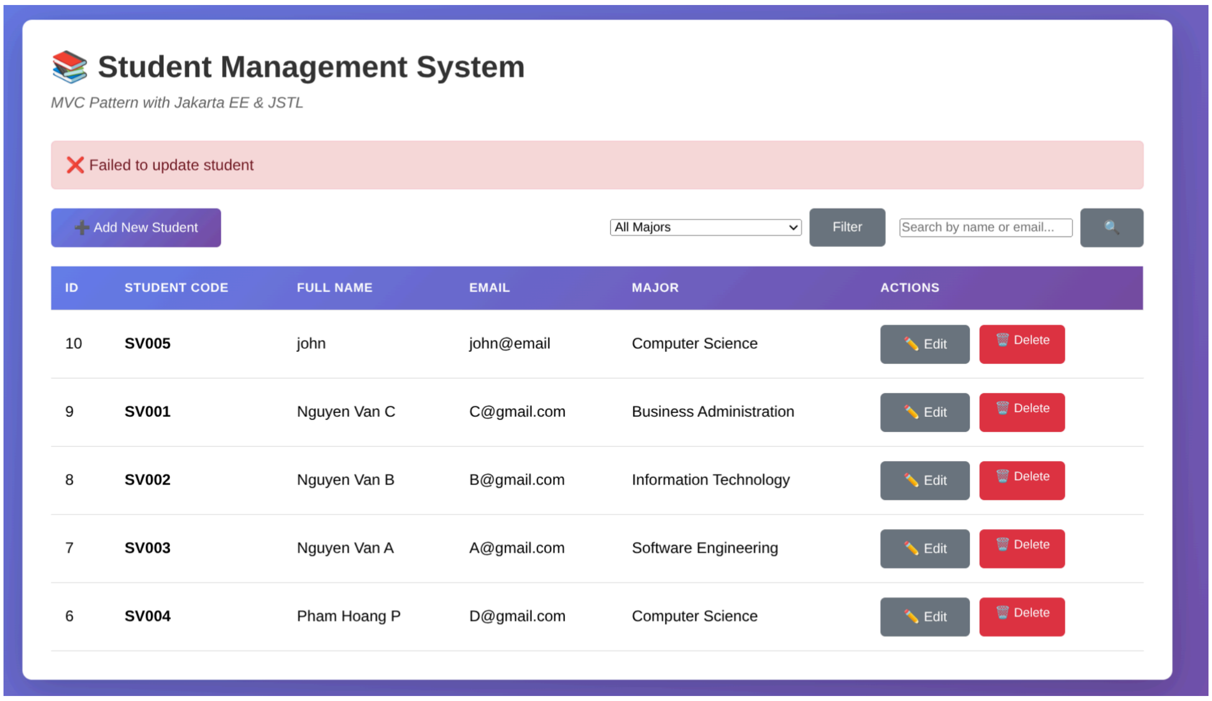
⚠️ Name must be at least 2 characters

Email *

john@email

Major *

Computer Science



📚 Student Management System

MVC Pattern with Jakarta EE & JSTL

❌ Failed to update student

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
10	**SV005**	john	john@email	Computer Science	✏️ Edit 🗑️ Delete
9	**SV001**	Nguyen Van C	C@gmail.com	Business Administration	✏️ Edit 🗑️ Delete
8	**SV002**	Nguyen Van B	B@gmail.com	Information Technology	✏️ Edit 🗑️ Delete
7	**SV003**	Nguyen Van A	A@gmail.com	Software Engineering	✏️ Edit 🗑️ Delete
6	**SV004**	Pham Hoang P	D@gmail.com	Computer Science	✏️ Edit 🗑️ Delete

7.1: Add Sort & Filter Methods to DAO (4 points)

- **Task:** Add two new methods to StudentDAO.java

Method 1: Sort Students

```
public List<Student> getStudentsSorted(String sortBy, String order) {
   List<Student> students = new ArrayList<>();


   String validSort = "id";
   if (sortBy != null) {
       if (sortBy.equals("student_code") || sortBy.equals("full_name") ||
               sortBy.equals("email") || sortBy.equals("major")) {
           validSort = sortBy;
       }
   }


   String validOrder = "ASC";
   if (order != null && order.equalsIgnoreCase("desc")) {
       validOrder = "DESC";
```

```
    }


    String sql = "SELECT * FROM students ORDER BY " + validSort + " " +
validOrder;


    try (Connection conn = getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {


        while (rs.next()) {
            Student student = new Student();
            student.setId(rs.getInt("id"));
            student.setStudentCode(rs.getString("student_code"));
            student.setFullName(rs.getString("full_name"));
            student.setEmail(rs.getString("email"));
            student.setMajor(rs.getString("major"));
            student.setCreatedAt(rs.getTimestamp("created_at"));
            students.add(student);
        }


    } catch (SQLException e) {
        e.printStackTrace();
    }


    return students;
}
```

- Input Validation (Security):
    + The method accepts sortBy (column name) and order (ASC/DESC).

    + Whitelist Check: It validates sortBy against a strict list of allowed column names (student_code, full_name, etc.). If the input doesn't match (or is null), it defaults to "id". This prevents SQL Injection attacks where users might inject malicious SQL commands into the ORDER BY clause.
    + Order Check: It checks if order is "desc" (case-insensitive). If not, it defaults to "ASC".
- Query Construction:
    + It constructs a dynamic SQL string: "SELECT * FROM students ORDER BY " + validSort + " " + validOrder.
- Execution:
    + It executes the query using a Statement, iterates through the ResultSet, and maps the rows to a list of Student objects.

Method 2: Filter Students by Major

```java
public List<Student> getStudentsByMajor(String major) {
   List<Student> students = new ArrayList<>();
   String sql = "SELECT * FROM students WHERE major = ? ORDER BY id DESC";

   try (Connection conn = getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

       pstmt.setString(1, major);
       ResultSet rs = pstmt.executeQuery();

       while (rs.next()) {
           Student student = new Student();
           student.setId(rs.getInt("id"));
           student.setStudentCode(rs.getString("student_code"));
           student.setFullName(rs.getString("full_name"));
           student.setEmail(rs.getString("email"));
           student.setMajor(rs.getString("major"));
           student.setCreatedAt(rs.getTimestamp("created_at"));
```

```
            students.add(student);

        }



    } catch (SQLException e) {

        e.printStackTrace();

    }



    return students;

}
```

- Query Construction:
    + It defines a parameterized SQL query: "SELECT * FROM students WHERE major = ? ORDER BY id DESC".
- Execution:
    + It uses a PreparedStatement to safely bind the major parameter to the ? placeholder.
    + It executes the query and maps the results to a list of Student objects.

Method 3: Create ONE method that handles BOTH sorting AND filtering:

```
public List<Student> getStudentsFiltered(String major, String sortBy, String
order) {
    List<Student> students = new ArrayList<>();

    String validSort = "id";
    if (sortBy != null) {
        if (sortBy.equals("student_code") || sortBy.equals("full_name") ||
                sortBy.equals("email") || sortBy.equals("major")) {
            validSort = sortBy;
        }
    }

    String validOrder = "ASC";
    if (order != null && order.equalsIgnoreCase("desc")) {
        validOrder = "DESC";
    }

    StringBuilder sql = new StringBuilder("SELECT * FROM students ");
    boolean hasMajor = (major != null && !major.trim().isEmpty());

    if (hasMajor) {
        sql.append("WHERE major = ? ");
    }

```

```java
    sql.append("ORDER BY ").append(validSort).append(" ").append(validOrder);

    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql.toString())) {

        if (hasMajor) {
            pstmt.setString(1, major);
        }

        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            Student student = new Student();
            student.setId(rs.getInt("id"));
            student.setStudentCode(rs.getString("student_code"));
            student.setFullName(rs.getString("full_name"));
            student.setEmail(rs.getString("email"));
            student.setMajor(rs.getString("major"));
            student.setCreatedAt(rs.getTimestamp("created_at"));
            students.add(student);
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return students;
}
```

- Validation:
  + It performs the same whitelist validation for sortBy and order as Method 1.
  + It checks if the major parameter is present (not null and not empty).
- Dynamic SQL Construction:
  + It starts with a base query using StringBuilder: "SELECT * FROM students ".
  + Conditional Filtering: If hasMajor is true, it appends "WHERE major = ? ".
  + Sorting: It appends "ORDER BY " + validSort + " " + validOrder.
- Execution:
  + It prepares the statement based on the constructed SQL string.
  + Conditional Parameter Binding: If hasMajor is true, it sets the major string into the first parameter index (1).
  + It executes the query and returns the mapped list.

7.2: Add Controller Methods (3 points)

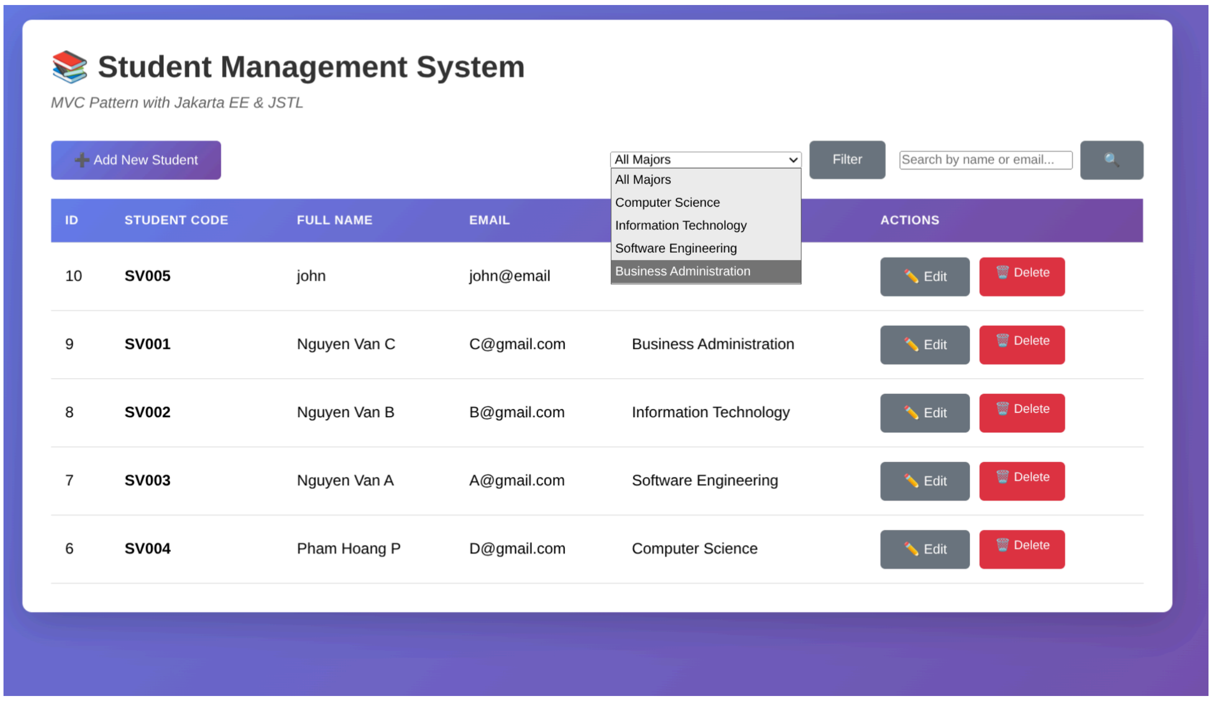
- **Task:** Add sorting and filtering to StudentController.java

```
private void sortStudents(HttpServletRequest request, HttpServletResponse
response)

       throws ServletException, IOException {

   String sortBy = request.getParameter("sortBy");

   String order = request.getParameter("order");



   List<Student> students = studentDAO.getStudentsSorted(sortBy, order);



   request.setAttribute("students", students);

   request.setAttribute("sortBy", sortBy);

   request.setAttribute("order", order);



   request.getRequestDispatcher("/views/student-list.jsp").forward(request,
response);

}



private void filterStudents(HttpServletRequest request, HttpServletResponse
response)

       throws ServletException, IOException {

   String major = request.getParameter("major");

   List<Student> students;



   if (major != null && !major.isEmpty()) {

       students = studentDAO.getStudentsByMajor(major);

   } else {

       students = studentDAO.getAllStudents();
```

```
    }



   request.setAttribute("students", students);

   request.setAttribute("currentMajor", major);



   request.getRequestDispatcher("/views/student-list.jsp").forward(request,
response);

}
```

7.3: Update View with Sort & Filter UI (3 points)

- **Task:** Add sorting and filtering controls to student-list.jsp



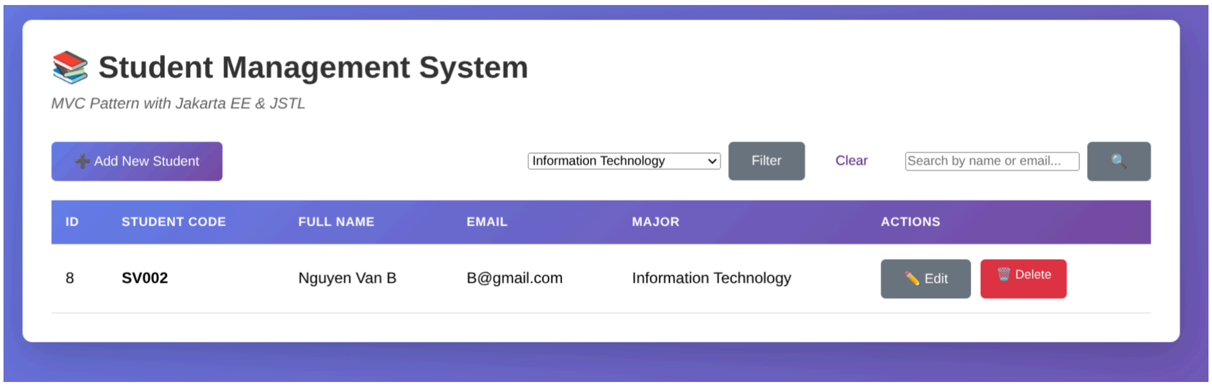
📚 Student Management System

MVC Pattern with Jakarta EE & JSTL

➕ Add New Student | All Majors | Filter | Search by name or email... | 🔍

- All Majors
- Computer Science
- Information Technology
- Software Engineering
- Business Administration

ID	STUDENT CODE	FULL NAME	EMAIL		ACTIONS
10	**SV005**	john	john@email		✏️ Edit 🗑️ Delete
9	**SV001**	Nguyen Van C	C@gmail.com	Business Administration	✏️ Edit 🗑️ Delete
8	**SV002**	Nguyen Van B	B@gmail.com	Information Technology	✏️ Edit 🗑️ Delete
7	**SV003**	Nguyen Van A	A@gmail.com	Software Engineering	✏️ Edit 🗑️ Delete
6	**SV004**	Pham Hoang P	D@gmail.com	Computer Science	✏️ Edit 🗑️ Delete

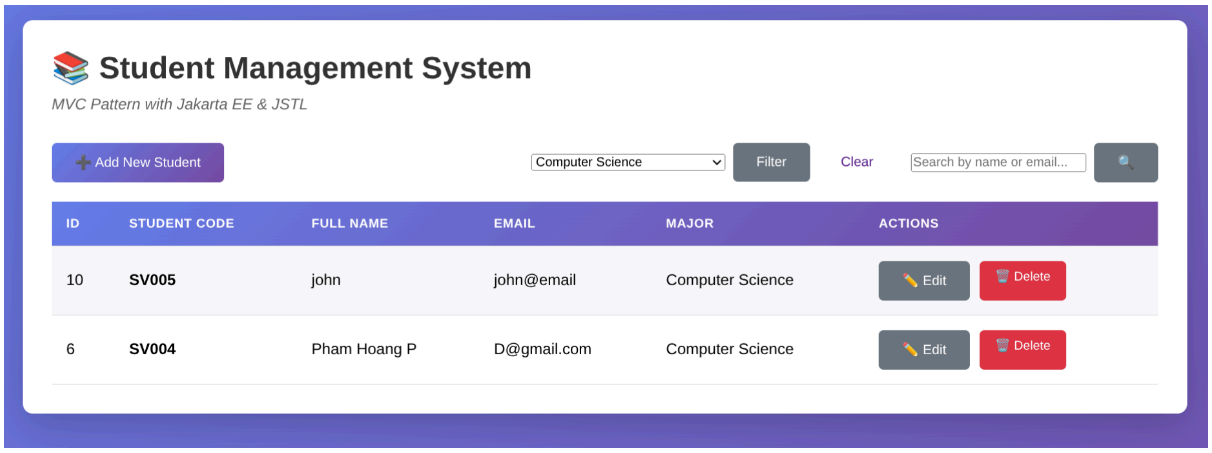


📚 Student Management System

MVC Pattern with Jakarta EE & JSTL

➕ Add New Student | Information Technology | Filter | Clear | Search by name or email... | 🔍

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
8	**SV002**	Nguyen Van B	B@gmail.com	Information Technology	✏️ Edit 🗑️ Delete

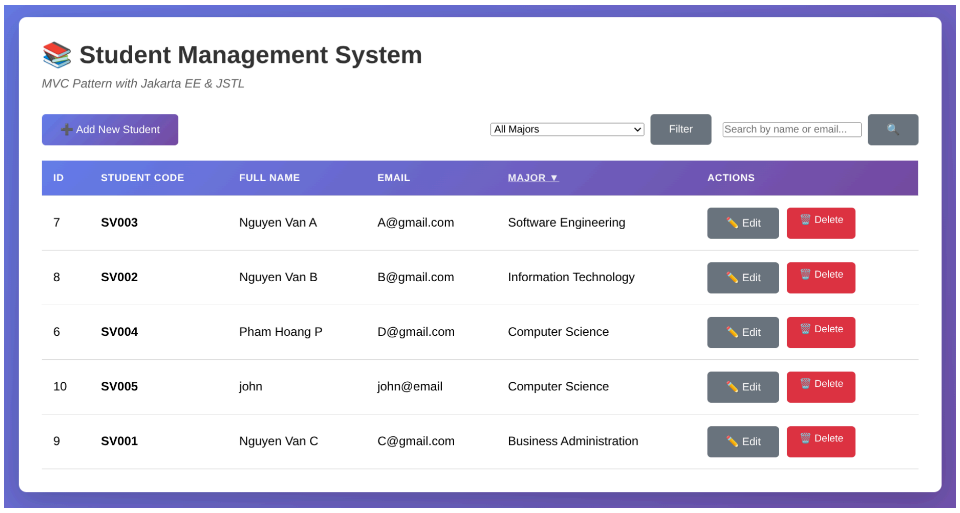


📚 Student Management System

MVC Pattern with Jakarta EE & JSTL

➕ Add New Student | Computer Science | Filter | Clear | Search by name or email... | 🔍

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
10	**SV005**	john	john@email	Computer Science	✏️ Edit 🗑️ Delete
6	**SV004**	Pham Hoang P	D@gmail.com	Computer Science	✏️ Edit 🗑️ Delete

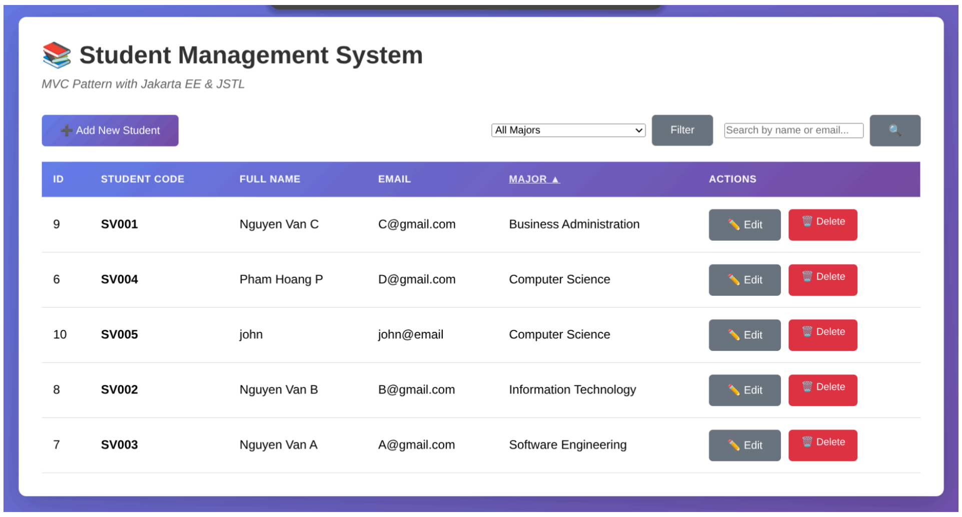


📚 Student Management System

MVC Pattern with Jakarta EE & JSTL

➕ Add New Student | All Majors | Filter | Search by name or email... | 🔍

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR ▼	ACTIONS
7	**SV003**	Nguyen Van A	A@gmail.com	Software Engineering	✏️ Edit 🗑️ Delete
8	**SV002**	Nguyen Van B	B@gmail.com	Information Technology	✏️ Edit 🗑️ Delete
6	**SV004**	Pham Hoang P	D@gmail.com	Computer Science	✏️ Edit 🗑️ Delete
10	**SV005**	john	john@email	Computer Science	✏️ Edit 🗑️ Delete
9	**SV001**	Nguyen Van C	C@gmail.com	Business Administration	✏️ Edit 🗑️ Delete

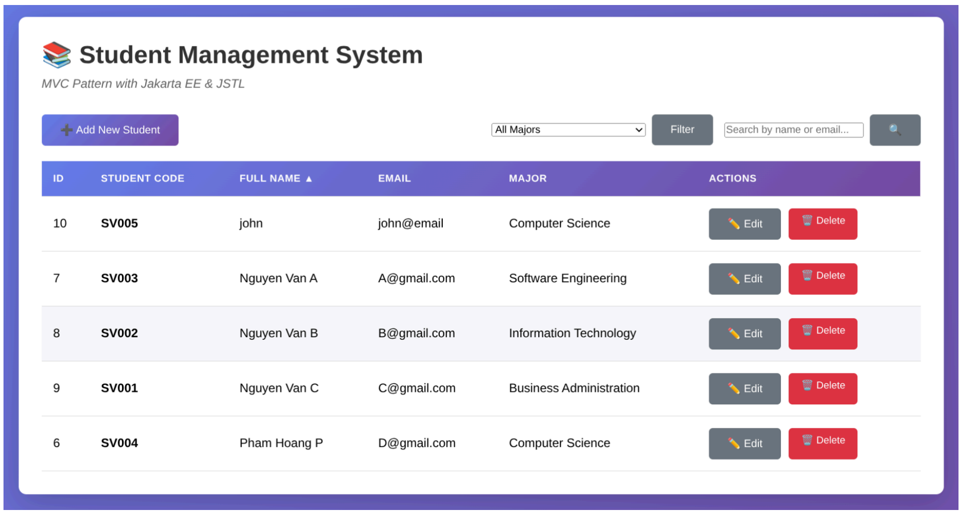


📚 Student Management System

MVC Pattern with Jakarta EE & JSTL

➕ Add New Student | All Majors | Filter | Search by name or email... | 🔍

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR ▲	ACTIONS
9	**SV001**	Nguyen Van C	C@gmail.com	Business Administration	✏️ Edit 🗑️ Delete
6	**SV004**	Pham Hoang P	D@gmail.com	Computer Science	✏️ Edit 🗑️ Delete
10	**SV005**	john	john@email	Computer Science	✏️ Edit 🗑️ Delete
8	**SV002**	Nguyen Van B	B@gmail.com	Information Technology	✏️ Edit 🗑️ Delete
7	**SV003**	Nguyen Van A	A@gmail.com	Software Engineering	✏️ Edit 🗑️ Delete



📚 Student Management System

MVC Pattern with Jakarta EE & JSTL

➕ Add New Student | All Majors | Filter | Search by name or email... | 🔍

ID	STUDENT CODE	FULL NAME ▲	EMAIL	MAJOR	ACTIONS
10	**SV005**	john	john@email	Computer Science	✏️ Edit 🗑️ Delete
7	**SV003**	Nguyen Van A	A@gmail.com	Software Engineering	✏️ Edit 🗑️ Delete
8	**SV002**	Nguyen Van B	B@gmail.com	Information Technology	✏️ Edit 🗑️ Delete
9	**SV001**	Nguyen Van C	C@gmail.com	Business Administration	✏️ Edit 🗑️ Delete
6	**SV004**	Pham Hoang P	D@gmail.com	Computer Science	✏️ Edit 🗑️ Delete