

SERVLET & MVC PATTERN

EXERCISE 5: SEARCH FUNCTIONALITY

5.1: Update StudentDAO

Think about:

- How will you reuse the code from `getAllStudents()` to build Student objects?
 - I reuse exact same block of code to populate a Student object:

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

    pstmt.setString(1, searchPattern);
    pstmt.setString(2, searchPattern);
    pstmt.setString(3, searchPattern);

    try (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;
}
```

- What happens if the keyword is null or empty?
 - If the keyword is null: The database searches for the literal text "null". Unless a student named "null", it will return 0 results.

```
Found 0 students:
-----
```

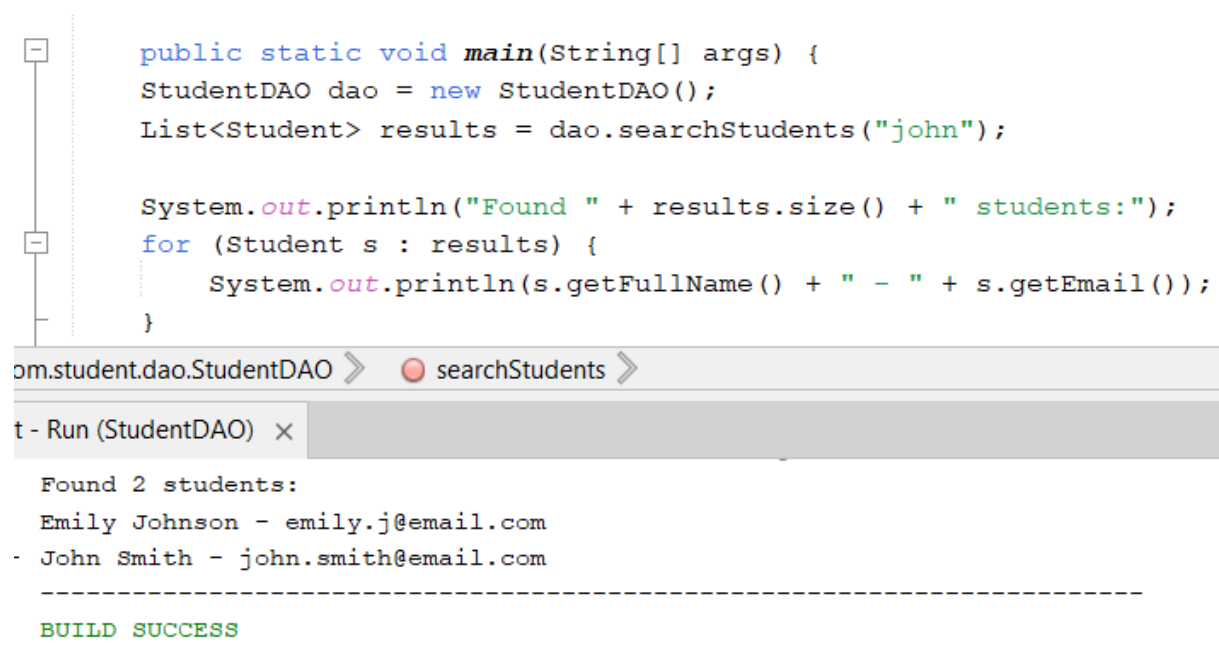
- If the keyword is empty "": The % wild card matches every single record in the database. Therefore, the result usually shows all records.

```
Found 5 students:
Miemie Love - tmy00037@gmail.com
Sarah Davis - sarah.d@email.com
Michael Brown - michael.b@email.com
Emily Johnson - emily.j@email.com
John Smith - john.smith@email.com
-----
```

- Should you validate the keyword before building the SQL?

Yes, absolutely.

Testing:



The screenshot shows an IDE with a Java program and its execution output. The program is a simple test for a StudentDAO class. It searches for students with the keyword "john" and prints the results. The output shows that 2 students were found: Emily Johnson and John Smith.

```
public static void main(String[] args) {
    StudentDAO dao = new StudentDAO();
    List<Student> results = dao.searchStudents("john");

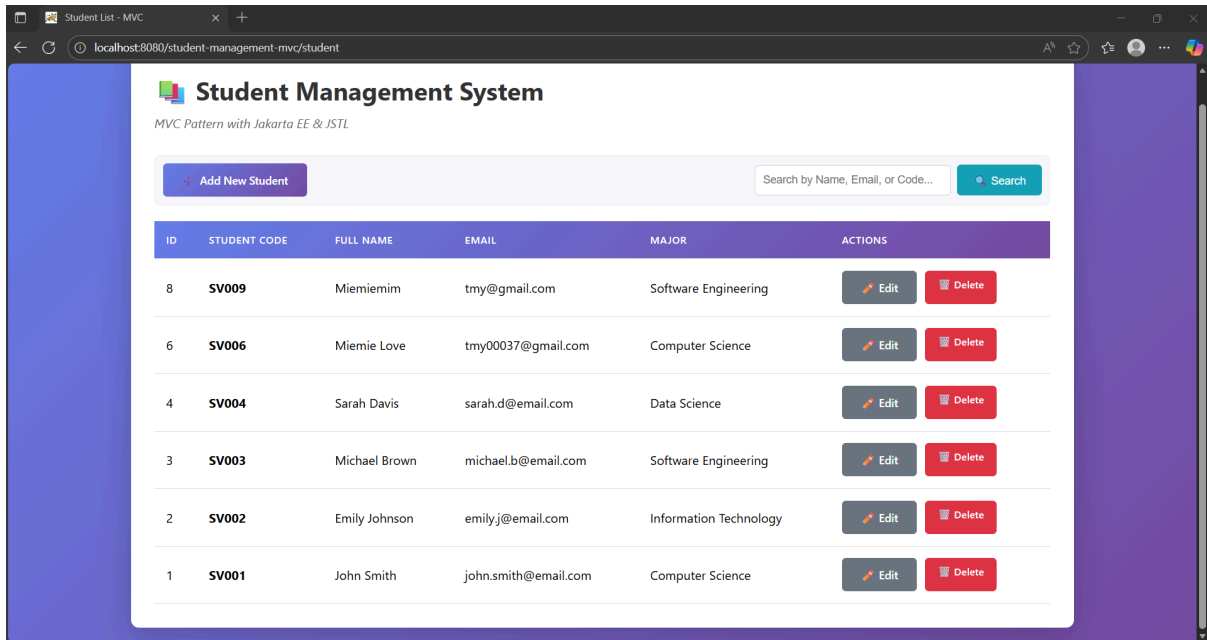
    System.out.println("Found " + results.size() + " students:");
    for (Student s : results) {
        System.out.println(s.getFullName() + " - " + s.getEmail());
    }
}
```

com.student.dao.StudentDAO > searchStudents >

t - Run (StudentDAO) x

```
Found 2 students:
Emily Johnson - emily.j@email.com
John Smith - john.smith@email.com
-----
BUILD SUCCESS
-----
```

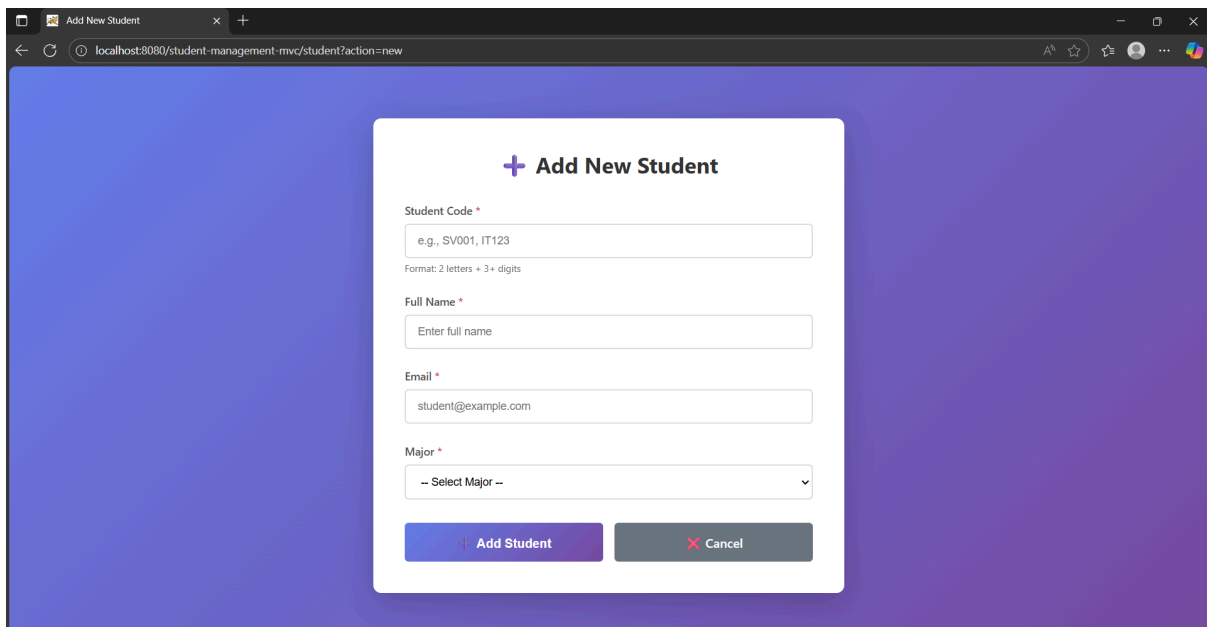
1. Home page with student list



The screenshot shows the 'Student Management System' home page. It features a header with the system name and a subtitle 'MVC Pattern with Jakarta EE & JSTL'. Below the header is a navigation bar with an 'Add New Student' button and a search bar. The main content area displays a table of students with columns for ID, Student Code, Full Name, Email, Major, and Actions. The table contains 8 rows of student data.

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
8	SV009	Miemiemim	tmy@gmail.com	Software Engineering	Edit Delete
6	SV006	Miemie Love	tmy00037@gmail.com	Computer Science	Edit Delete
4	SV004	Sarah Davis	sarah.d@email.com	Data Science	Edit Delete
3	SV003	Michael Brown	michael.b@email.com	Software Engineering	Edit Delete
2	SV002	Emily Johnson	emily.j@email.com	Information Technology	Edit Delete
1	SV001	John Smith	john.smith@email.com	Computer Science	Edit Delete

2. Add student form



The screenshot shows the 'Add New Student' form. It is a modal form with a title '+ Add New Student'. The form contains several input fields: 'Student Code' (with a hint 'e.g., SV001, IT123' and a format note 'Format: 2 letters + 3+ digits'), 'Full Name' (with a hint 'Enter full name'), 'Email' (with a hint 'student@example.com'), and 'Major' (a dropdown menu with the option '-- Select Major --'). At the bottom of the form are two buttons: 'Add Student' and 'Cancel'.

+ Add New Student

Student Code *
e.g., SV001, IT123
Format: 2 letters + 3+ digits

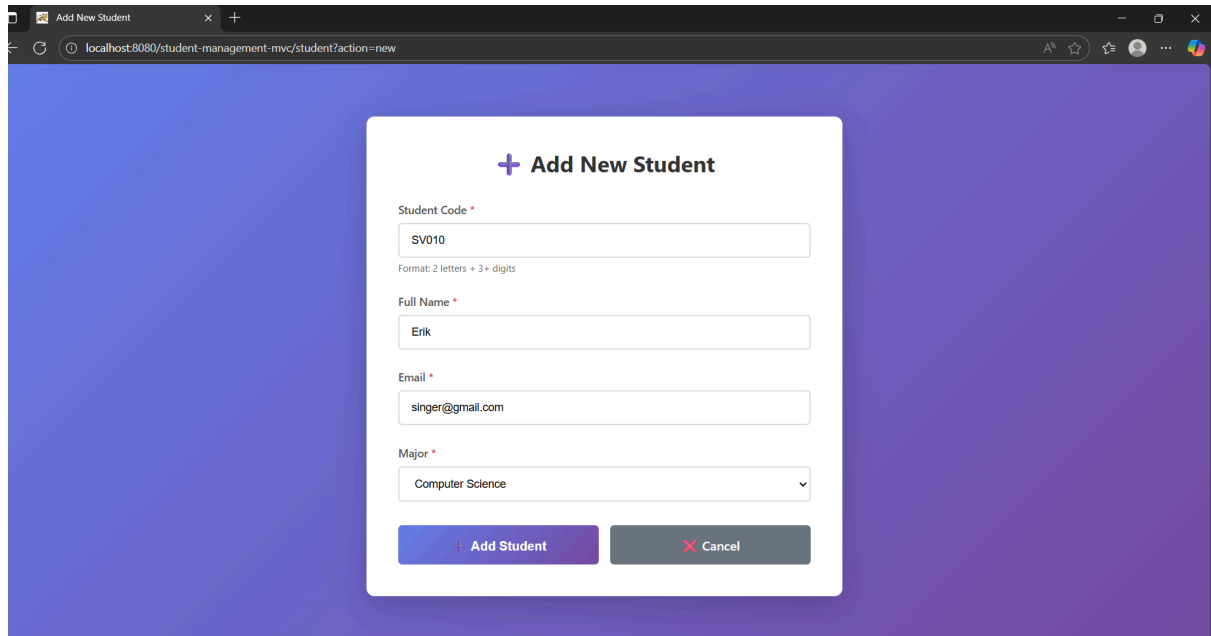
Full Name *
Enter full name

Email *
student@example.com

Major *
-- Select Major --

[Add Student](#) [Cancel](#)

3. Edit student form (pre-filled)



+ Add New Student

Student Code *
SV010
Format: 2 letters + 3+ digits

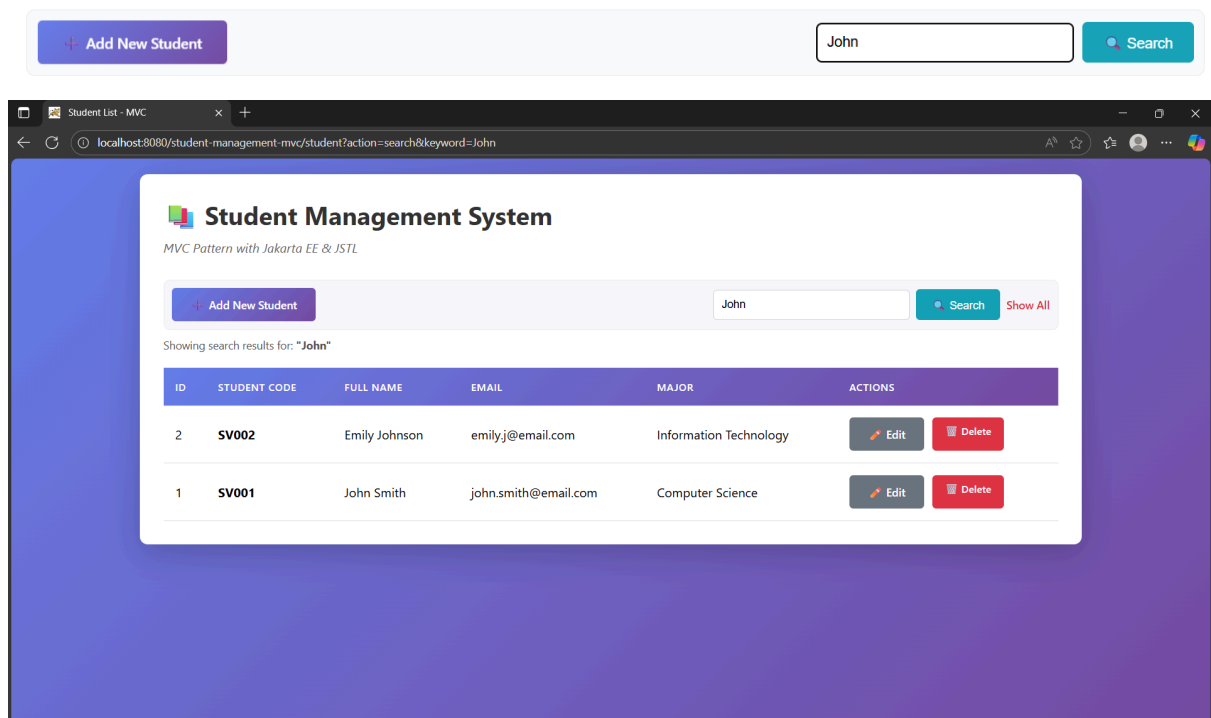
Full Name *
Erik

Email *
singer@gmail.com

Major *
Computer Science

+ Add Student **✕ Cancel**

4. Search results



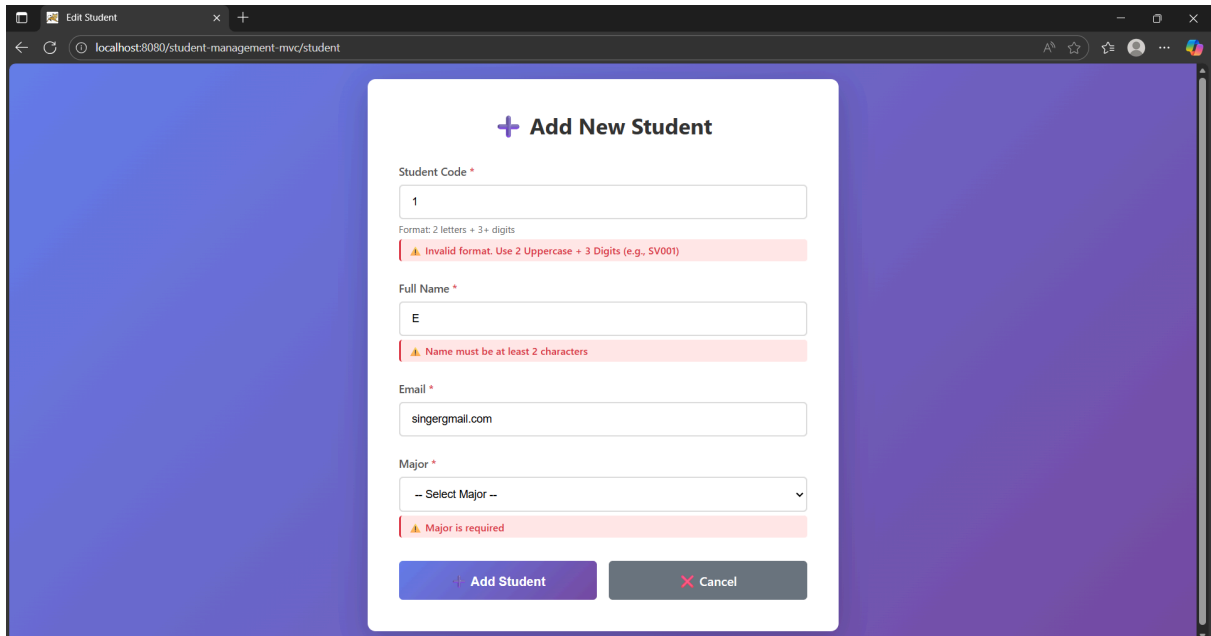
Student Management System
MVC Pattern with Jakarta EE & JSTL

+ Add New Student John **Search** **Show All**

Showing search results for: "John"

ID	STUDENT CODE	FULL NAME	EMAIL	MAJOR	ACTIONS
2	SV002	Emily Johnson	emily.j@email.com	Information Technology	Edit Delete
1	SV001	John Smith	john.smith@email.com	Computer Science	Edit Delete

5. Validation errors



The screenshot shows a web browser window with the address bar displaying `localhost:8080/student-management-mvc/student`. The page features a modal form titled "+ Add New Student" with the following fields and validation errors:

- Student Code ***: Contains the value "1". Below the input, a message states "Format: 2 letters + 3+ digits". A red error bar below the input reads: "Invalid format. Use 2 Uppercase + 3 Digits (e.g., SV001)".
- Full Name ***: Contains the value "E". A red error bar below the input reads: "Name must be at least 2 characters".
- Email ***: Contains the value "singer@gmail.com".
- Major ***: A dropdown menu showing "-- Select Major --". A red error bar below the dropdown reads: "Major is required".

At the bottom of the form are two buttons: "Add Student" (with a plus icon) and "Cancel" (with a red X icon).