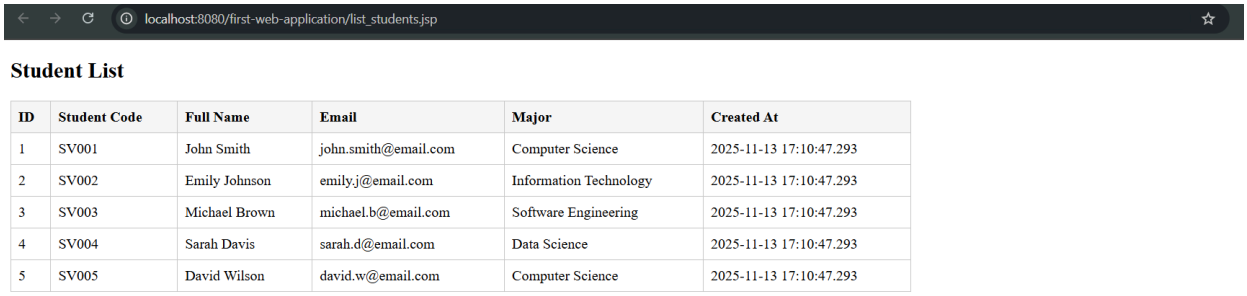
Student Name: Vũ Đình Đức
Student ID: ITCSIU23004
Course: Web-application Development
Semester: I 2025-2026
Instructor: Nguyễn Văn Sinh

Lab 04

Part A:

Exercise 1:



The page works as follow:

-We have a JSP file that contains both HTML and Java code at the top. The Java section does all database connection and querying, then the HTML below renders the table.

-At the very top (the Java part) we define 3 main things:
+jdbcUrl — JDBC connection string:
"jdbc:sqlserver://localhost:1434;databaseName=student_management;encrypt=false;trustServerCertificate=true"
+dbUser and dbPass — the SQL Server username and password.
+query — the SQL statement to get students:

->SELECT id, student_code, full_name, email, major, created_at FROM dbo.students ORDER BY id

-When the JSP is requested by the browser, the server executes the Java code in order:
+First it tries to load the JDBC driver class with Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").
->If the driver class is **not found** (jar missing from WEB-INF/lib) -> show this user-facing message:
JDBC Driver not found. Put the Microsoft JDBC driver JAR in WEB-INF/lib.
->Also the code calls log(...) to record the detailed exception on the server log and then stops processing (return).
->If the driver loads successfully -> continue.

+Next the code uses a **try-with-resources** block to open the database resources:
->Connection conn = DriverManager.getConnection(jdbcUrl, dbUser, dbPass); — opens a connection to SQL Server.
->PreparedStatement ps = conn.prepareStatement(query); — prepares the SQL query.
->ResultSet rs = ps.executeQuery(); — executes the query and returns the results.
+Because try-with-resources is used, conn, ps, and rs are **automatically closed** when the block ends:
->This ensures resources are released properly (no leaks) — meeting the "Resources closed properly" requirement.

-Inside the try block we render the HTML table header first (static HTML with columns: ID, Student Code, Full Name, Email, Major, Created At).

-Then we iterate over the ResultSet to display each student row:
+We use while (rs.next()) to loop through rows.
+For each row:
->rs.getInt("id") prints the ID column.
->rs.getString("student_code") prints Student Code.
->rs.getString("full_name") prints Full Name.
->rs.getString("email") prints Email.
->rs.getString("major") prints Major.
->rs.getTimestamp("created_at") prints Created At as a timestamp.
+We set a boolean flag hasRows to true when at least one row exists.
+After the loop:
->If hasRows is false -> print a single table row with No students found. so the user sees a friendly message instead of an empty table.

-If a **SQLException** occurs while opening the connection or querying:
+The catch block runs:
->Show a friendly message on the page: Database error. Please contact the administrator. so internal details are not exposed to the user.

->Call log("SQL error while fetching students: " + ex.getMessage(), ex); to write the full stack trace and details into the server log for debugging.

-Other important behavior/details:
+The page sets the content type to UTF-8 so characters display correctly.
+The CSS at the top styles the table for readability (borders, padding, header background).
+The JDBC URL currently points to port 1434 — if your SQL Server uses a different port or instance, change jdbcUrl accordingly.
+Credentials are stored in the JSP for this exercise (quick test). For production you should move them into a DataSource (JNDI) or configuration file.
+Because the code uses PreparedStatement (not concatenated SQL strings), it reduces SQL injection risk for queries that include parameters (this query has no parameters, but it's still good practice).

-How the user sees results (runtime flow):
+Open http://localhost:8080/StudentManagement/list_students.jsp.
+Tomcat compiles the JSP and runs the Java code at the top.
+If driver and connection succeed -> page displays a table with the 5 sample students (one row per student) including the created_at timestamp.
+If driver is missing or DB error occurs -> the page shows a short friendly error message and the server log contains the full exception details.

-Finally, how this meets the exercise criteria:
+Database connection successful: the code attempts DriverManager.getConnection(...) using the provided JDBC URL and credentials.
+Query executes correctly: ps.executeQuery() runs the SELECT ... FROM dbo.students.
+All data displayed in table: the while (rs.next()) loop prints ID, student_code, full_name, email, major, created_at.
+Proper error handling: user-friendly messages on the page and detailed logs via log(...).
+Resources closed properly: try-with-resources guarantees ResultSet, PreparedStatement, and Connection are closed.

Exercise 2:

Add New Student

Student Code (required)

123

Full Name (required)

DucVu

Email (optional)

dinhducvu87@gmail.com

Major (optional)

Computer Science

Add Student Cancel

Student List

ID	Student Code	Full Name	Email	Major	Created At
1	SV001	John Smith	john.smith@email.com	Computer Science	2025-11-13 17:10:47.293
2	SV002	Emily Johnson	emily.j@email.com	Information Technology	2025-11-13 17:10:47.293
3	SV003	Michael Brown	michael.b@email.com	Software Engineering	2025-11-13 17:10:47.293
4	SV004	Sarah Davis	sarah.d@email.com	Data Science	2025-11-13 17:10:47.293
5	SV005	David Wilson	david.w@email.com	Computer Science	2025-11-13 17:10:47.293
6	123	DucVu	dinhducvu87@gmail.com	Computer Science	2025-11-13 18:02:38.043

The add student form works as follow:

- We have a total of 4 inputs: student_code, full_name, email, major.

- We set each of the inputs to have attributes as follow:

- For student_code field:
  +The input is <input type="text" name="student_code" required maxlength="10">.
  +The browser will enforce that it is filled (because of required) and limit length to 10.
  ->If it is empty -> browser blocks submit and shows native required error.
  ->Otherwise, nothing happens on the client side.
  +On submit the form sends data by POST to process_add.jsp.

- For full_name field:
  +The input is <input type="text" name="full_name" required maxlength="100">.
  +The browser will enforce that it is filled (because of required) and limit length to 100.
  ->If it is empty -> browser blocks submit and shows native required error.
  ->Otherwise, nothing happens on the client side.
  +On submit the form sends data by POST to process_add.jsp.

- For email field:
  +The input is <input type="email" name="email"> (optional).
  +The browser will validate format when the user tries to submit (because of type="email").
  ->If it is present but not a valid email -> browser blocks submit and shows native format error.
  ->If it is empty or valid -> nothing happens on the client side.
  +On submit the form sends data by POST to process_add.jsp.

- For major field:
  +The input is <input type="text" name="major" maxlength="50"> (optional).
  +The browser only enforces maxlength.
  ->If it exceeds maxlength -> browser stops input / shows error depending on browser.
  ->Otherwise, nothing happens on the client side.
  +On submit the form sends data by POST to process_add.jsp.

- Buttons:
  +Submit button posts the form to process_add.jsp.
  +Cancel link navigates back to list_students.jsp.

Once the form is submitted -> process_add.jsp runs and works as follow:

- When process_add.jsp receives the POST request:
  +It first calls request.setCharacterEncoding("UTF-8") to ensure correct encoding.
  +It retrieves parameters using request.getParameter(...) for student_code, full_name, email, major and trims them.

->If retrieval returns null or empty strings, they will be handled by validation below.

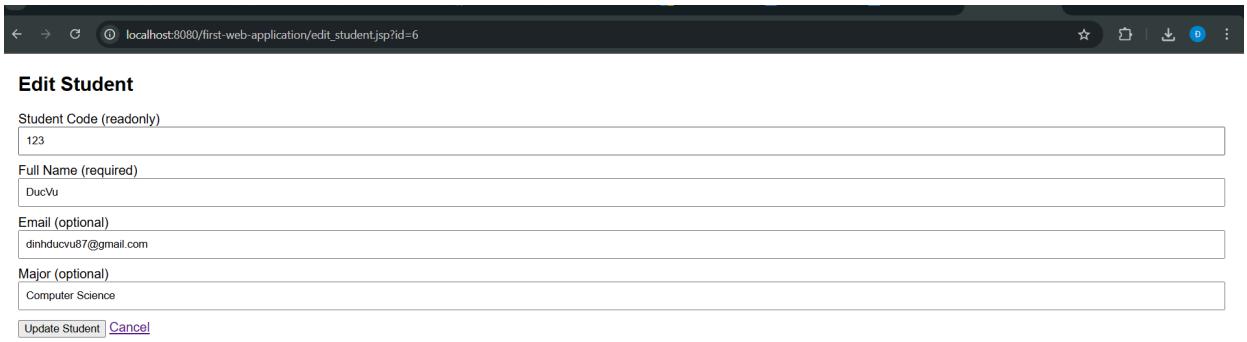
- Server-side validation:
  +It checks required fields: student_code and full_name.
  ->If either is null or empty -> show an error message "Required fields missing" and provide a link back to add_student.jsp.
  ->Otherwise, continue to insertion.

- Database insertion (safe flow):
  +Load JDBC driver com.microsoft.sqlserver.jdbc.SQLServerDriver (shows friendly message and logs if missing).
  +Open DB resources in a try-with-resources block: Connection, PreparedStatement.
  +Prepare SQL with placeholders:
  ->INSERT INTO dbo.students (student_code, full_name, email, major) VALUES (?, ?, ?, ?)
  +Set parameters:
  ->ps.setString(1, studentCode) and ps.setString(2, fullName).
  ->For email and major: if empty -> ps.setNull(..., Types.VARCHAR) else ps.setString(...).
  +Execute ps.executeUpdate().

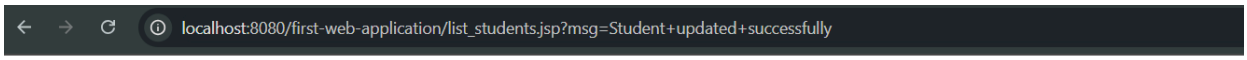
- On success:
  +If executeUpdate() returns > 0 -> the insert succeeded.
  ->The code redirects to list_students.jsp with a URL-encoded success message using response.sendRedirect("list_students.jsp?msg=...").

- On failure / duplicate:
  +Catch SQLException and inspect ex.getErrorCode().
  ->If error code is 2627 or 2601 -> treat as duplicate student code and show "Student code already exists" with a link back to add_student.jsp.
  ->Otherwise -> show a generic "Database error. Please contact the administrator." and log(...) the full exception for debugging.

Exercise 3:


localhost:8080/first-web-application/edit_student.jsp?id=6
Edit Student
Student Code (readonly)
123
Full Name (required)
DucVu
Email (optional)
dinhducvu87@gmail.com
Major (optional)
Computer Science
Update Student
Cancel


localhost:8080/first-web-application/edit_student.jsp?id=6
Edit Student
Student Code (readonly)
123
Full Name (required)
John
Email (optional)
dinhducvu87@gmail.com
Major (optional)
Computer Science
Update Student
Cancel



Student List

ID	Student Code	Full Name	Email	Major	Created At
1	SV001	John Smith	john.smith@email.com	Computer Science	2025-11-13 17:10:47.293
2	SV002	Emily Johnson	emily.j@email.com	Information Technology	2025-11-13 17:10:47.293
3	SV003	Michael Brown	michael.b@email.com	Software Engineering	2025-11-13 17:10:47.293
4	SV004	Sarah Davis	sarah.d@email.com	Data Science	2025-11-13 17:10:47.293
5	SV005	David Wilson	david.w@email.com	Computer Science	2025-11-13 17:10:47.293
6	123	John	dinhducvu87@gmail.com	Computer Science	2025-11-13 18:02:38.043

The edit form (edit_student.jsp) works as follow:

- We have a total of 4 inputs: student_code, full_name, email, major.

- We set each of the inputs and hidden id as follow:

- For student_code field:
  +When the page loads, we get id from URL (?id=...) and query the database for that student's data.
  +The returned student_code is placed in <input name="student_code" ... readonly>.
  ->Because it is readonly -> the user cannot change the student code.
  ->The student_code is still submitted with the form for clarity but cannot be edited.

- For full_name field:
  +When the page loads, the full_name from the database is pre-filled into <input name="full_name" required maxlength="100">.
  +The input has required so the browser will block empty submit.
  ->If it is empty on submit -> browser shows native required error.
  ->Server also checks it is not empty before updating.

- For email field:
  +When the page loads, the email from the database is pre-filled into <input type="email" name="email">.
  +This field is optional; browser will validate format if filled (type="email").

->If left empty -> nothing on client side; server will store NULL.

- For major field:
  +When the page loads, the major from the database is pre-filled into <input name="major" maxlength="50">.
  +This field is optional.
  ->If left empty -> server will store NULL.

- Hidden id:
  +We include <input type="hidden" name="id" value="..."> so the processing page knows which record to update.

- If id is missing, invalid, or the query returns no row:
  +The page shows a friendly error Invalid student id or Student not found and a link back to the list.
  ->No form is shown in that case.

Once the form is submitted -> process_edit.jsp runs and works as follow:

- When the page receives POST:
  +It calls request.setCharacterEncoding("UTF-8").
  +It retrieves parameters with request.getParameter(...): id, student_code, full_name, email, major.

- ID validation:
  +It checks id exists and can be parsed to an integer.
  ->If missing or not numeric -> show Invalid student id and link back to list.

- Server-side validation:
  +It checks full_name is not null/empty.
  ->If full_name is empty -> show Required field missing: Full Name is required and link back to edit page.

- Database update (safe flow):
  +Load JDBC driver com.microsoft.sqlserver.jdbc.SQLServerDriver (friendly message + log if missing).
  +Open Connection and PreparedStatement inside try-with-resources (so resources close automatically).
  +Use PreparedStatement with SQL and WHERE clause:
  ->UPDATE dbo.students SET full_name = ?, email = ?, major = ? WHERE id = ?
  +Set parameters:
  ->ps.setString(1, fullName.trim())
  ->If email empty -> ps.setNull(2, Types.VARCHAR) else ps.setString(2, email.trim())
  ->If major empty -> ps.setNull(3, Types.VARCHAR) else ps.setString(3, major.trim())

->ps.setInt(4, id)
+Execute ps.executeUpdate().

- On success:
  +If affected rows > 0 -> redirect to list_students.jsp?msg=Student+updated+successfully using response.sendRedirect(...).
  ->User sees the updated values in the list.

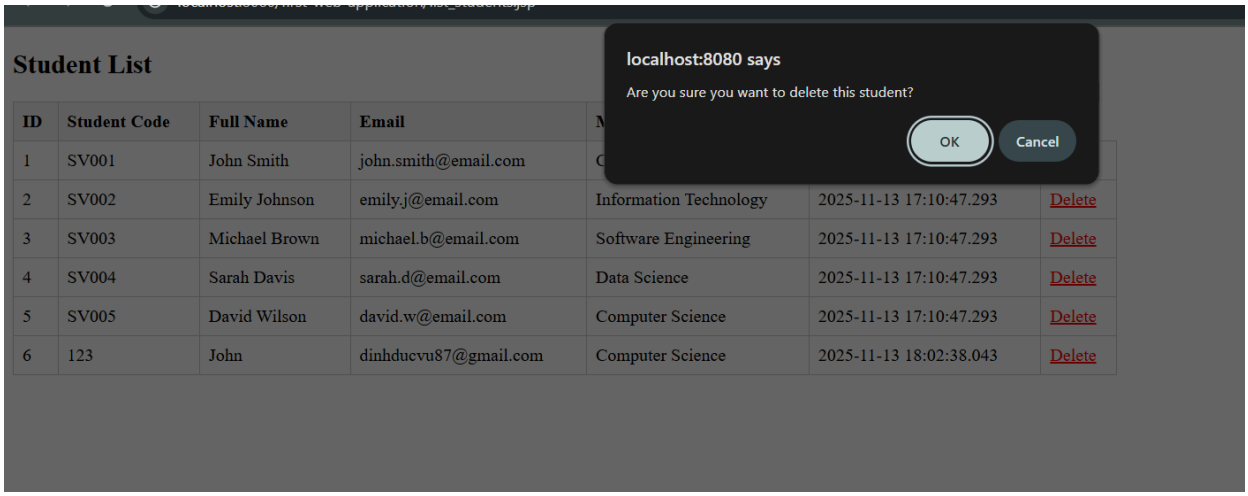
- On failure:
  +If affected rows == 0 -> show Update failed: student not found. and link back to list.
  +If SQLException occurs -> show friendly Database error. Please contact administrator. and log(...) the detailed exception.

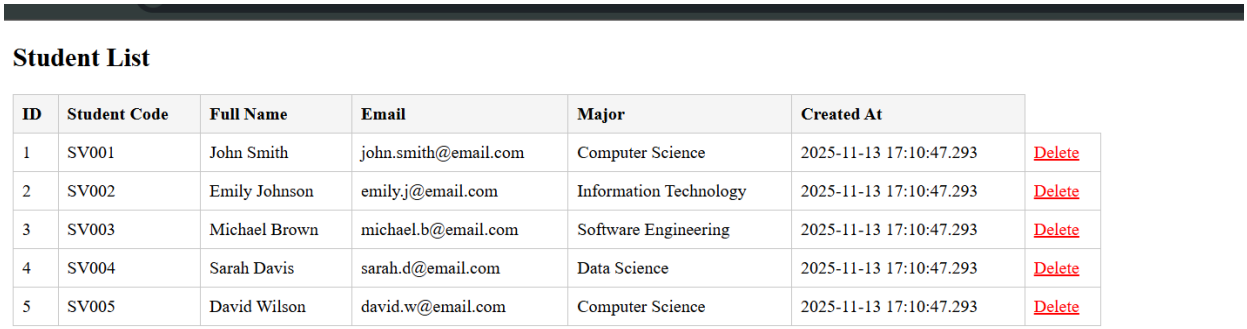
- Security and resource notes:
  +Using PreparedStatement prevents SQL injection and handles parameter binding safely.
  +try-with-resources ensures Connection and PreparedStatement are closed (no leaks).
  +student_code is readonly in the form so unique code is preserved and cannot be changed by user.

Exercise 4:

Student List

ID	Student Code	Full Name	Email	Major	Created At	
1	SV001	John Smith	john.smith@email.com	Computer Science	2025-11-13 17:10:47.293	Delete
2	SV002	Emily Johnson	emily.j@email.com	Information Technology	2025-11-13 17:10:47.293	Delete
3	SV003	Michael Brown	michael.b@email.com	Software Engineering	2025-11-13 17:10:47.293	Delete
4	SV004	Sarah Davis	sarah.d@email.com	Data Science	2025-11-13 17:10:47.293	Delete
5	SV005	David Wilson	david.w@email.com	Computer Science	2025-11-13 17:10:47.293	Delete
6	123	John	dinhducvu87@gmail.com	Computer Science	2025-11-13 18:02:38.043	Delete





Student List

ID	Student Code	Full Name	Email	Major	Created At	
1	SV001	John Smith	john.smith@email.com	Computer Science	2025-11-13 17:10:47.293	Delete
2	SV002	Emily Johnson	emily.j@email.com	Information Technology	2025-11-13 17:10:47.293	Delete
3	SV003	Michael Brown	michael.b@email.com	Software Engineering	2025-11-13 17:10:47.293	Delete
4	SV004	Sarah Davis	sarah.d@email.com	Data Science	2025-11-13 17:10:47.293	Delete
5	SV005	David Wilson	david.w@email.com	Computer Science	2025-11-13 17:10:47.293	Delete

The delete function works as follow:

- We have a hidden ID parameter in the URL: ?id=....

- When we click the **Delete** link:
  +A confirmation dialog pops up using onclick="return confirm(...)".
  ->If user clicks **Cancel** -> nothing happens.
  ->If user clicks **OK** -> browser navigates to delete_student.jsp?id=[student_id].

- In delete_student.jsp:

- We first get id from URL parameter using request.getParameter("id").
  +If id is missing or invalid -> show an error and link back to list.

- We prepare a DELETE SQL using PreparedStatement:
  +DELETE FROM dbo.students WHERE id = ?
  +Set the parameter id with ps.setInt(1, id).

- Execute the query with ps.executeUpdate().
  +If affected rows > 0 -> redirect to list_students.jsp?msg=Student deleted successfully.
  +If affected rows == 0 -> show Delete failed: student not found. and link back.
  +If SQLException occurs -> show friendly Database error and log full details.

- In list_students.jsp:
  +Each student row has a Delete link.
  +Link is red (style="color:red") and has a confirmation dialog.
  +Clicking OK calls delete_student.jsp?id=... for that student.
  +Clicking Cancel prevents accidental deletion.

- Resource and security notes:
  +PreparedStatement prevents SQL injection.
  +try-with-resources ensures DB resources are closed automatically.

Link github repo: https://github.com/ducvu01/web_lab_04_exercise1234.git