Java Project Test - 1

Project: Student Management System (Console-Based)

Project Overview

Create a Student Management System console application in Java to perform CRUD (Create, Read, Update, Delete) operations on student records stored in a relational database using JDBC.

Functional Requirements

1. **Add New Student**
   - Input student details: ID, Name, Age, Email, and Course.
   - Save student information into the database.
2. **View All Students**
   - Retrieve and display all student records from the database.
   - Show fields: ID, Name, Age, Email, Course.
3. **Search Student by ID**
   - Input student ID.
   - Retrieve and display the student details if present.
4. **Update Student Details**
   - Input student ID.
   - Allow updating Name, Age, Email, and Course.
   - Save changes in the database.
5. **Delete Student**
   - Input student ID.
   - Remove the student record from the database.
6. **Exit**
   - Exit the application.

Non-Functional Requirements

- Use JDBC to connect and interact with the database (MySQL, PostgreSQL, or any relational DB).
- Use PreparedStatement to prevent SQL injection.
- Handle exceptions properly with meaningful messages.
- Validate user inputs (e.g., positive age, valid email format).
- Menu-driven console UI with options for each function.
- Modular design: separate classes for:
  - Database operations
  - Student entity
  - Main UI logic

Java Project Test - 1

Database Design

Table Name: students

Column Name	Data Type	Constraints
student_id	INT	PRIMARY KEY
name	VARCHAR(100)	NOT NULL
age	INT	NOT NULL
email	VARCHAR(100)	UNIQUE, NOT NULL
course	VARCHAR(100)	NOT NULL

Core Java Concepts to Use

- Classes and Objects
- Methods (with overloading if needed)
- Exception Handling (try-catch)
- JDBC API: Connection, PreparedStatement, ResultSet
- Input validation
- Loops and conditionals for menu navigation
- String manipulation
- Static utility methods (optional)
- Encapsulation (private fields + getters/setters)

===== Student Management System =====

1. Add New Student
2. View All Students
3. Search Student by ID
4. Update Student Details
5. Delete Student
6. Exit

Enter your choice:

Sample Flow

- User selects option 1 to add a student.
- Application asks for student details one by one.
- Validates input.
- Inserts record into the database.
- Displays confirmation and returns to menu.

Additional Enhancements (Optional)

- Use regex for better email validation.
- Implement pagination for large data sets.
- Add search functionality by student name (partial match).
- Log operations to a text file.
- Store DB credentials in a .properties file.

Java Project Test - 1

Student Management System – Folder Structure

```
StudentManagementSystem/
├── src/
│   └── com/student/
│       ├── model/                          # Data model (POJO)
│       │   └── Student.java
│       ├── dao/                            # DAO Layer (Database operations)
│       │   ├── interfaces/
│       │   │   └── StudentDAO.java
│       │   └── impl/
│       │       └── StudentDAOImpl.java
│       ├── service/                        # Business logic layer
│       │   ├── interfaces/
│       │   │   └── StudentService.java
│       │   └── impl/
│       │       └── StudentServiceImpl.java
│       ├── util/                           # Utility classes
│       │   ├── DBConnection.java
│       │   └── InputValidator.java
│       └── app/                            # Main application and menu
│           ├── StudentApp.java
│           └── MenuHandler.java
├── resources/
│   └── db.properties                       # DB connection configuration
├── sql/
│   └── schema.sql                          # SQL script for table creation
├── logs/
│   └── app.log                             # Optional log file
├── README.md                               # Project documentation
├── build.gradle / pom.xml                  # Gradle or Maven config
└── .gitignore                              # Ignore compiled files and settings
```