ALLIANCE UNIVERSITY

Private University established in Karnataka State by Act No.34 of year 2010
Recognized by the University Grants Commission (UGC), New Delhi

Project Report

Master of Computer Application

Semester – II

Java Object-Oriented Design and Programming

Project title: STUDENT MANAGEMENT SYSTEM

Submitted By:

Shivangi Pathak– 2411022250027

G Komala – 2411022250049

Sandra B– 2411022250001

Thanuja M– 2411022250013

Nishanth Ranjan– 2411022250047

Submitted To:

Faculty Name:

Mentor/Faculty Signature:

Department of Computer Application
Alliance University
Chandapura - Anekal Main Road, Anekal
Bengaluru - 562 106

March 2025

Table of Contents

Student Management System

Abstract

The Student Management System (SMS) is a robust software solution designed to efficiently handle the storage, retrieval, and manipulation of student-related data within an academic institution. In the digital age, manual processes of managing student records are time-consuming, error-prone, and inefficient. This project aims to bridge the gap between administrative needs and technological advancement by providing a user-friendly, scalable, and responsive system for educational institutions to manage student information with ease.

This project adopts a dual-interface approach, incorporating both a Java Swing-based desktop application and a web-based interface developed using HTML, CSS, and JavaScript. The desktop version is tailored for offline administrative access, while the web interface allows broader accessibility and remote management capabilities. Both interfaces interact with a centralized MySQL database, ensuring that data remains synchronized, consistent, and up-to-date.

The system enables users to perform a range of **CRUD operations**—Create, Read, Update, and Delete—on student records, which include details such as student ID, name, course, department, semester, contact details, and address. It also supports advanced search functionality and allows administrators to quickly filter and retrieve specific information. The architecture is designed to be **modular**, allowing future enhancements like grade tracking, attendance monitoring, and fee management to be easily integrated

The **technology stack** used in this project includes:

- **Java**: For backend processing and Swing-based GUI.
- **MySQL**: For relational database management and persistent storage of student data.
- **HTML/CSS/JavaScript**: For creating an intuitive and interactive web interface.
- **JDBC (Java Database Connectivity)**: To establish seamless communication between the Java application and the MySQL database.
- **HTTP-based API endpoints**: For interaction between the frontend and backend in the web version.

Introduction

In today's digital era, educational institutions are embracing technology to streamline administrative tasks and improve operational efficiency. One such area that has seen significant transformation is the management of student data. Traditionally, managing student information such as enrollment details, academic records, attendance, and personal information — was a manual process involving physical records and spreadsheets, which are prone to errors, time-consuming, and difficult to maintain.

To address these inefficiencies, many institutions have shifted toward Student Management Systems (SMS) that enable administrators, teachers, and students to access and manage data quickly and securely. These systems not only facilitate easy storage and retrieval of student data but also offer features such as report generation, real-time updates, and multi-user access, which collectively enhance decision-making and overall productivity.

This project aims to design and implement a Student Management System that leverages both a desktop application and a web-based interface. It ensures seamless interaction with a centralized MySQL database and serves as a comprehensive solution for student data management in small to mid-sized educational institutions.

Objective

The primary objectives of the Student Management System project are:

- To build a user-friendly application for managing student records efficiently.
- To integrate a Java Swing-based desktop GUI and a web-based HTML/CSS/JavaScript interface.
- To implement CRUD operations (Create, Read, Update, Delete) securely via a Java backend connected to a MySQL database.
- To provide real-time communication between the front-end and back-end via HTTP APIs.
- To minimize manual data handling and ensure data integrity, accuracy, and accessibility.
- To create a scalable and modular codebase that can be extended in the future to support features like attendance tracking, result management, and SMS/email notifications.

Scope

The scope of the project includes the development of:

- A Java Swing desktop application that allows school or college administrative staff to manage student information directly from their computers.
- A responsive web-based version of the system accessible from any browser, offering basic functionalities for students and faculty.
- A MySQL database to store student details such as name, age, gender, department, year of study, contact details, and academic records.
- Secure API endpoints that connect the front end (both desktop and web) with the database for data operations.
- Code for data validation, error handling, and user authentication to ensure security and reliability.

The system supports multi-user access and can be deployed on a local network or cloud infrastructure, depending on institutional requirements.

Target Users

This Student Management System is designed for the following user groups:

1. Administrative Staff
    - Responsible for entering, updating, and maintaining student records.
    - Will primarily use the Java Swing desktop version.
2. Faculty Members
    - Can view student data, generate academic reports, and possibly manage attendance or internal marks.
    - Will have access via the web interface.
3. Students
    - Can log in to view their personal records, academic details, and announcements (in extended versions).
    - Will access the system through the web-based platform.

3. IT Administrators
    - In charge of database management, system deployment, user access control, and overall system health monitoring

System Overview

System Architecture

The Student Management System is designed using a hybrid architecture that supports both a desktop-based Java Swing application and a web-based interface using HTML, CSS, JavaScript, and Java back-end services. Both interfaces communicate with a common MySQL database using HTTP APIs and JDBC connections.

This dual-interface system architecture provides flexibility, allowing different user types (admin, faculty, students) to interact with the application based on their roles and preferences.

Java Swing Desktop Application

- Built using Java Swing for the GUI.
- Designed for internal administrative users who manage and maintain student records.
- Connects to the MySQL database using **JDBC** for performing CRUD operations.
- Offers direct interaction with the database — ideal for faster, heavy-duty data entry and manipulation.

Web-Based Interface

- Developed using **HTML**, **CSS**, and **JavaScript** for the frontend.
- Backend written in **Java (Servlets/REST APIs)**, handling data operations via HTTP.
- Designed for students and faculty to view or interact with the system remotely.
- Communicates with the backend through **AJAX calls or fetch APIs**, which in turn perform JDBC-based database interactions.

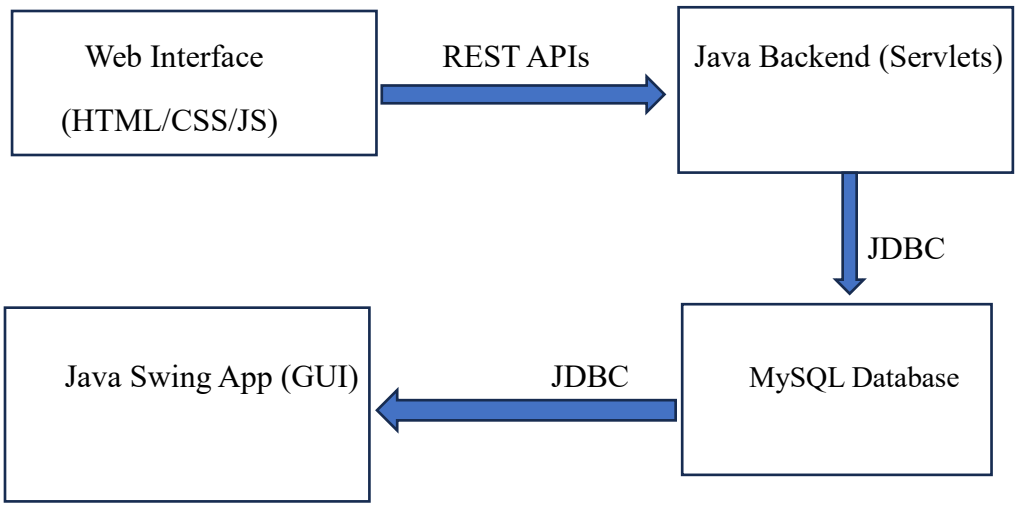
MySQL Database

- A centralized database that stores all student-related data such as:
    - Personal information
    - Academic details
    - Course enrollments
    - Contact details, etc.
- Ensures data consistency across both desktop and web platforms.

Communication Layer

- The Java backend exposes **RESTful APIs** to allow the frontend (web interface) to send and receive data using **HTTP GET/POST/PUT/DELETE** methods.
- The desktop app interacts directly with the database using JDBC, while the web app uses REST APIs for indirect interaction.

Block Diagram



Requirements Specification

This section outlines both the software and hardware requirements needed to develop, run, and maintain the Student Management System. Ensuring the appropriate configuration helps achieve better performance, reliability, and user experience.

Software Requirements

The following software components are essential for the development and execution of the Student Management System

Component	Details
Java (JDK)	Version 8 or above. Required for developing the Java Swing application and backend
MySQL Database	Version 5.7 or higher. Stores all student-related information
Java Swing	Used to design the desktop GUI interface. Integrated within Java SDK.
Web Browser	Chrome, Firefox, or Edge for accessing the web

	version.
XAMPP / MySQL Workbench	For managing the MySQL server and database visually.
HTML/CSS/JavaScript	For designing and managing the web interface.
Eclipse	IDEs for Java development.

Hardware Requirements

To develop and run the Student Management System smoothly, the following hardware specifications are recommended:

Component	Minimum Specification	Recommended Specification
Processor (CPU)	Intel i3 7th Gen or AMD Ryzen 3	Intel i5 10th Gen or AMD Ryzen 5 or higher
RAM	4 GB	8 GB or more
Hard Disk	250 GB HDD	512 GB SSD or higher
Operating System	Windows 7/8/10, Linux (Ubuntu), or macOS	Windows 10/11 or Ubuntu 20.04+
Network	Internet Connection (for web-based access)	Stable Broadband or LAN

Technology Stack

The Student Management System is developed using a modern full-stack approach, combining both desktop-based and web-based interfaces. The chosen technologies are robust, open-source, and well-supported, ensuring scalability, maintainability, and cross-platform compatibility.

1. **Java (Backend + Swing GUI)**

Java is used as the core programming language for both the backend logic and the development of the desktop application using the Swing framework.

- **Why Java?**
  Java is platform-independent, secure, and widely used for enterprise-level applications. Its object-oriented nature makes it suitable for modular and scalable software
- **Swing for GUI:**
  Java Swing is a part of Java Foundation Classes (JFC) and is used to build the

desktop application interface. It provides rich components like buttons, tables, forms, and dialog boxes.

Backend Role:
Java handles form validation, business logic, and interaction with the MySQL database using JDBC (Java Database Connectivity).

2. MySQL (Database System)
MySQL is a relational database management system (RDBMS) that stores all student records, login credentials, and related data securely.

- **Why MySQL?**
  It is free, open-source, and supports complex queries and transactions, making it ideal for handling structured data.
- **Database Usage:**
  - Student information is stored in structured tables.
  - Data is retrieved using SELECT queries and manipulated using INSERT, UPDATE, and DELETE operations.

3. HTML / CSS / JavaScript (Frontend Web Interface)
The web version of the Student Management System is built using core web technologies — HTML, CSS, and JavaScript.

- **HTML** structures the content of the web pages.
- **CSS** styles the interface for better user experience.
- **JavaScript** handles interactivity, form validation, and API communication.

3.JavaScript for Form Submission:

```
document.getElementById('studentForm').addEventListener('submit', function(e) {
  e.preventDefault();
  // Fetch API call here
});
```

4. HTTP Communication (APIs)
To connect the frontend and backend, RESTful HTTP APIs are used. The backend Java code exposes endpoints that accept requests from the frontend and return appropriate responses.

- **Technology Used:**
  Java Servlets / Java HTTP Handlers (or optionally Spring Boot in advanced versions)
- **How It Works:**
  - The web form sends data to the backend using POST requests.
  - The backend processes the data and updates the MySQL database.
  - GET requests are used to fetch student data for display.

API Call in JavaScript:

```
fetch('http://localhost:8080/api/students', {
method: 'POST',
```

```
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ name: 'John', course: 'Maths' })
});
```

Database Design

The database is the backbone of the Student Management System, responsible for securely storing and retrieving all student-related data. It is designed using the principles of relational database management, implemented in MySQL. The design ensures data consistency, integrity, and ease of querying.

1. Entity-Relationship (ER) Diagram
The system includes a primary entity:

- **Student**
  Each student record is associated with basic personal and academic details. The ER model is simple, yet scalable for future extensions like adding "Courses", "Grades", or "User Roles".

Attributes of the Student entity:

- StudentID (Primary Key)
- Name
- Age
- course
- Phone
- password
- Email

---

2. Database Schema
Database Name: student_management

```
CREATE DATABASE student_management;
USE student_management;
```

---

3. Table Structure
Table: students

```
CREATE TABLE students (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   age INT NOT NULL,
   course VARCHAR(100) NOT NULL,
   phone VARCHAR(15),
   password TEXT,
   email VARCHAR(100)
);
```

This table reflects the fields used in both the **Java Swing GUI** and **Web Frontend** forms. Each field corresponds directly to form inputs and backend handling:

- name, age, and course are mandatory fields, as enforced in both JavaScript and Java validations.
- phone, address, and email are optional but included to allow comprehensive student data storage.
- id is auto-incremented and serves as a **unique identifier** for each student, used during **update** or **delete** operations.

---

4. Integration with Code

- In the **Java backend**, the POST /addStudent and GET /getStudents endpoints interact directly with this table using SQL queries.
- In the **JavaScript frontend**, form data is collected and sent to the backend as a JSON object, which is then inserted into the MySQL table.

Code Implementation

Java code:

```
package Project_1;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.util.ArrayList;

class Student {

   String name;

   int age;

   String course;

   String phone;

   String address;

   String email;

   public Student(String name, int age, String course, String phone, String address, String email) {
```

```
        this.name = name;
        this.age = age;
        this.course = course;
        this.phone = phone;
        this.address = address;
        this.email = email;
    }
    public String toString() {
        return "<html><b>" + name + "</b> | Age: " + age + " | Course: " + course + "<br>📞 " +
phone + " | ✉ " + email + "<br>🏠 " + address + "</html>";
    }
}
public class StudentManagementGUI {
    private JFrame frame;
    private DefaultListModel<String> studentListModel;
    private JList<String> studentList;
    private ArrayList<Student> students = new ArrayList<>();

    public StudentManagementGUI() {
        frame = new JFrame("🎓 Student Management System");
        frame.setSize(600, 600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());

        // Set main background color
        frame.getContentPane().setBackground(new Color(240, 248, 255));
```

```
// Title Label
JLabel title = new JLabel("🗄 Manage Student Information", SwingConstants.CENTER);
title.setFont(new Font("Segoe UI", Font.BOLD, 24));
title.setForeground(new Color(25, 25, 112));
frame.add(title, BorderLayout.NORTH);

// Student List Panel
studentListModel = new DefaultListModel<>();
studentList = new JList<>(studentListModel);
studentList.setFont(new Font("Segoe UI", Font.PLAIN, 14));
studentList.setSelectionBackground(new Color(173, 216, 230));
frame.add(new JScrollPane(studentList), BorderLayout.CENTER);

// Buttons Panel
JPanel panel = new JPanel();
panel.setBackground(new Color(230, 240, 255));
JButton addButton = createButton("➕ Add Student", new Color(60, 179, 113));
JButton editButton = createButton("✏ Edit Student", new Color(255, 165, 0));
JButton deleteButton = createButton("🗑 Delete", new Color(255, 99, 71));

addButton.setToolTipText("Add a new student");
editButton.setToolTipText("Edit selected student");
deleteButton.setToolTipText("Delete selected student");

panel.add(addButton);
```

```
    panel.add(editButton);
    panel.add(deleteButton);
    frame.add(panel, BorderLayout.SOUTH);

    // Button Actions
    addButton.addActionListener(e -> addStudent());
    editButton.addActionListener(e -> editStudent());
    deleteButton.addActionListener(e -> deleteStudent());

    frame.setVisible(true);
}

private JButton createButton(String text, Color color) {
    JButton button = new JButton(text);
    button.setBackground(color);
    button.setForeground(Color.WHITE);
    button.setFont(new Font("Segoe UI", Font.BOLD, 14));
    button.setFocusPainted(false);
    button.setPreferredSize(new Dimension(150, 40));
    return button;
}
private void addStudent() {
    JTextField nameField = new JTextField();
    JTextField ageField = new JTextField();
    JTextField courseField = new JTextField();
    JTextField phoneField = new JTextField();
```

```
        JTextField addressField = new JTextField();
        JTextField emailField = new JTextField();

        Object[] fields = {
            "👤 Name:", nameField,
            "🎂 Age:", ageField,
            "📖 Course:", courseField,
            "📞 Phone:", phoneField,
            "🏠 Address:", addressField,

            "✉️ Email:", emailField
        };
        int option = JOptionPane.showConfirmDialog(frame, fields, "➕ Add Student",
JOptionPane.OK_CANCEL_OPTION);
        if (option == JOptionPane.OK_OPTION) {
            try {
                String name = nameField.getText().trim();
                int age = Integer.parseInt(ageField.getText().trim());
                String course = courseField.getText().trim();
                String phone = phoneField.getText().trim();
                String address = addressField.getText().trim();
                String email = emailField.getText().trim();

                if (name.isEmpty() || course.isEmpty()) {
                    throw new IllegalArgumentException("Name and course cannot be empty.");
                }
```

```
            Student student = new Student(name, age, course, phone, address, email);
            students.add(student);
            studentListModel.addElement(student.toString());
            try {
                Connection conn = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/studentinfodb", "root", "123456789"
                );

                String sql = "INSERT INTO student (name, age, course, phone, address, email) VALUES (?, ?, ?, ?, ?, ?)";
                java.sql.PreparedStatement stmt = conn.prepareStatement(sql);
                stmt.setString(1, name);
                stmt.setInt(2, age);
                stmt.setString(3, course);
                stmt.setString(4, phone);
                stmt.setString(5, address);
                stmt.setString(6, email);

                stmt.executeUpdate();
                stmt.close();
                conn.close();
            } catch (SQLException ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog(frame, "❌ Database insert failed: " + ex.getMessage());
```

```
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(frame, "Invalid input. Please enter all fields correctly.");
        }
    }
}

private void editStudent() {
    int index = studentList.getSelectedIndex();
    if (index != -1) {
        Student student = students.get(index);

        JTextField nameField = new JTextField(student.name);
        JTextField ageField = new JTextField(String.valueOf(student.age));
        JTextField courseField = new JTextField(student.course);
        JTextField phoneField = new JTextField(student.phone);
        JTextField addressField = new JTextField(student.address);
        JTextField emailField = new JTextField(student.email);

        Object[] fields = {
            "👤 Name:", nameField,
            "🎂 Age:", ageField,
            "📖 Course:", courseField,
            "📞 Phone:", phoneField,
            "🏠 Address:", addressField,
```

```
            "✉ Email:", emailField
        };

        int option = JOptionPane.showConfirmDialog(frame, fields, "✏ Edit Student",
JOptionPane.OK_CANCEL_OPTION);
        if (option == JOptionPane.OK_OPTION) {
            try {
                student.name = nameField.getText().trim();
                student.age = Integer.parseInt(ageField.getText().trim());
                student.course = courseField.getText().trim();
                student.phone = phoneField.getText().trim();
                student.address = addressField.getText().trim();
                student.email = emailField.getText().trim();

                studentListModel.set(index, student.toString());
            } catch (Exception e) {
                JOptionPane.showMessageDialog(frame, "Invalid input. Please enter all fields
correctly.");
            }
        }
    } else {
        JOptionPane.showMessageDialog(frame, "Please select a student to edit.");
    }
  }

  private void deleteStudent() {
```

```
        int index = studentList.getSelectedIndex();
        if (index != -1) {
            int confirm = JOptionPane.showConfirmDialog(frame, "Are you sure you want to delete this student?", "Delete Confirmation", JOptionPane.YES_NO_OPTION);
            if (confirm == JOptionPane.YES_OPTION) {
                students.remove(index);
                studentListModel.remove(index);
            }
        } else {
            JOptionPane.showMessageDialog(frame, "Please select a student to delete.");
        }
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(StudentManagementGUI::new);
    }
}
```

Summary of the Code

This code defines a **graphical user interface (GUI)** application for managing student information using **Java Swing**. It allows the user to:

- **Add** new students
- **Edit** existing students
- **Delete** students
- **View** a list of students in a scrollable panel

The GUI is styled with colors, emojis, and clean fonts to make the application more user-friendly and visually engaging.

Detailed Explanation

1. **Student Class:**

- Acts as a data model for each student.
- Contains attributes like name, age, course, phone, address, and email.
- The `toString()` method formats student details using HTML for display in the GUI.

2. **Main GUI Class — StudentManagementGUI:**

- **`JFrame`**: Main window of the application.
- **`DefaultListModel`** & **`JList`**: Used to display the list of students.
- **`ArrayList<Student>`**: Stores the actual student objects in memory.

3. **GUI Components Setup:**

- Sets up the main window with a title, layout, size, and background color.
- A title label is added at the top of the window.

4. **Buttons and Panel:**

- A bottom panel with three buttons: Add, Edit, Delete.
- Each button is styled and has tooltips.

5. **Edit Student Logic:**

- If a student is selected, a pre-filled dialog appears.
- On confirmation, the selected `Student` object is updated, and the list is refreshed.

6. **Delete Student Logic:**

- Asks for confirmation before removing a student.
- Removes from both the `ArrayList` and `ListModel`

7. **Main Method:**

- Launches the GUI on the **Event Dispatch Thread (EDT)**—the correct way to run Swing apps

MySQL Queries:

Purpose: Store and manage student data.

What to Include:

- **SQL table creation code:**

```
create database studentinfodb;
use studentinfodb;
CREATE TABLE student (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100),
   age INT,
   course VARCHAR(100),
   phone VARCHAR(20),
   address VARCHAR(255),
   email VARCHAR(100)
);
select* from student;
SHOW DATABASES;
```

- Sample INSERT, UPDATE, DELETE, and SELECT queries.
- Explain database normalization (if used), foreign keys (if applicable), and indexing for performance.

HTML/CSS/JS Frontend Code

Purpose: Web interface for accessing the system.

What to Include:

- index.html with form layout to enter student data.
- CSS styling (e.g., layout, fonts, responsive design).
- JavaScript for:
    - Fetch API calls to the backend.
    - Form validation.
    - DOM manipulation to display results dynamically.

HTML Frontend Code

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Student Management</title>

  <link rel="stylesheet" href="style.css">
```

```
</head>
<body>
  <div class="container">
    <h1>🎓 Student Management System</h1>

    <form id="student-form">
      <input type="text" placeholder="Name" id="name" required />
      <input type="number" placeholder="Age" id="age" required />
      <input type="text" placeholder="Course" id="course" required />
      <input type="text" placeholder="Phone" id="phone" required />
      <input type="text" placeholder="Address" id="address" required />
      <input type="email" placeholder="Email" id="email" required />
      <button type="submit">Add Student</button>
    </form>

    <div id="student-list"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

HTML Code Explanation:

- This code defines the **frontend (web interface)** of a basic student management system. It allows users to enter student details and displays the list of added students.

Key Components:

<!DOCTYPE html>:

- Declares the document type as HTML5.

<html lang="en">:

- Starts the HTML document with language set to English.

<head> Section:

- <meta charset="UTF-8">: Ensures proper character encoding.
- <title>: Sets the page title (shown in the browser tab).
- <link rel="stylesheet" href="style.css">: Links to an external CSS file for styling.

<body> Section:

Main Wrapper:

Acts as the main layout container for the content.

Heading: Displays the page title visually

<h1>🎓 Student Management System</h1>

Student Form:

<form id="student-form"> ... </form>

A form that collects student information using input fields:

- Name
- Age
- Course
- Phone
- password
- Email
- Add Student (to add the student)

Student List Container:

This empty div will be used by JavaScript (script.js) to display a list of students dynamically.

JavaScript Link:

<script src="script.js"></script>

Connects the HTML with an external JavaScript file (script.js), which will contain the logic to handle form submissions, save/display student data, and update the list in real time.

CSS Frontend Code:

```
body {
    font-family: 'Segoe UI', sans-serif;
    background: #f0f8ff;
    margin: 0;
    padding: 20px;
  }
  .container {
    max-width: 700px;
    margin: auto;
    background: white;
    padding: 30px;
    border-radius: 12px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
  }
  h1 {
    text-align: center;
    color: #1e3a8a;
  }
  form {
    display: grid;
    grid-gap: 12px;
    margin-bottom: 20px;
  }
```

```
input {
  padding: 12px;
  border: 1px solid #ccc;
  border-radius: 8px;
}
button {
  background-color: #2563eb;
  color: white;
  padding: 12px;
  border: none;
  border-radius: 8px;
  font-weight: bold;
  cursor: pointer;
  transition: background 0.3s ease;
}
button:hover {
  background-color: #1d4ed8;
}
#student-list {
  margin-top: 20px;
}
.student-card {
  background: #f9fafb;
  padding: 15px;
  margin-bottom: 10px;
  border: 1px solid #e5e7eb;
```

```
  border-radius: 8px;
}
```

CSS Code Explanation

- This CSS file styles the student management webpage for a clean, modern, and user-friendly appearance.

Global Styles:

- Sets the font for the whole page.
- Applies a light blue background.
- Removes default margins and adds padding**.**

Container Style:

- Centers the container on the page.
- Gives it a white background, rounded corners, and a drop shadow for a card-like effect.

Heading Style:

- Centers the heading and gives it a deep blue color.

Form Layout:

- Organizes form fields in a grid layout with space between inputs.

Input Fields:

- Adds padding and soft rounded borders to input fields.

Submit Button:

- Styled with a blue background, rounded corners, and hover effect to enhance interactivity.

JS Frontend Code:

```
const form = document.getElementById('student-form');
const studentList = document.getElementById('student-list');

form.addEventListener('submit', async (e) => {
```

```
  e.preventDefault();

  const studentData = {
    name: document.getElementById('name').value,
    age: document.getElementById('age').value,
    course: document.getElementById('course').value,
    phone: document.getElementById('phone').value,
    address: document.getElementById('address').value,
    email: document.getElementById('email').value,
  };

  // Call Java backend via POST
  const response = await fetch('http://localhost:8080/addStudent', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(studentData)
  });

  if (response.ok) {
    alert("✅ Student added!");
    loadStudents(); // refresh the list
    form.reset();
  } else {
    alert("❌ Error adding student");
  }
});
```

```
// Fetch all students from Java backend
async function loadStudents() {
  const res = await fetch('http://localhost:8080/getStudents');
  const students = await res.json();

  studentList.innerHTML = '';
  students.forEach((s) => {
    const div = document.createElement('div');
    div.className = 'student-card';
    div.innerHTML = `
      <strong>${s.name}</strong> | Age: ${s.age} | Course: ${s.course} <br>
      Phone: ${s.phone} | Email: ${s.email} <br>
      Address: ${s.address}
    `;
    studentList.appendChild(div);
  });
}
loadStudents(); // Load on page start
```

JS code Explanation

Form Submission: form.addEventListener('submit', async (e) => { ... });

- Prevents the default form behavior.
- Gathers student details from the form.
- Sends a POST request to http://localhost:8080/addStudent with JSON data.
- Shows an alert if successful, clears the form, and reloads the student list.

Fetching Students: async function loadStudents() { ... }

- Sends a GET request to http://localhost:8080/getStudents
- Gets a list of students as JSON.
- Clears the current list and creates a styled card for each student using innerHTML.

Initial Load: loadStudents();

- Automatically loads the student list when the page opens.

Summary: This JavaScript file connects the frontend with the Java backend API, handles adding students, and dynamically updates the UI with student data from the server.

Screenshots:

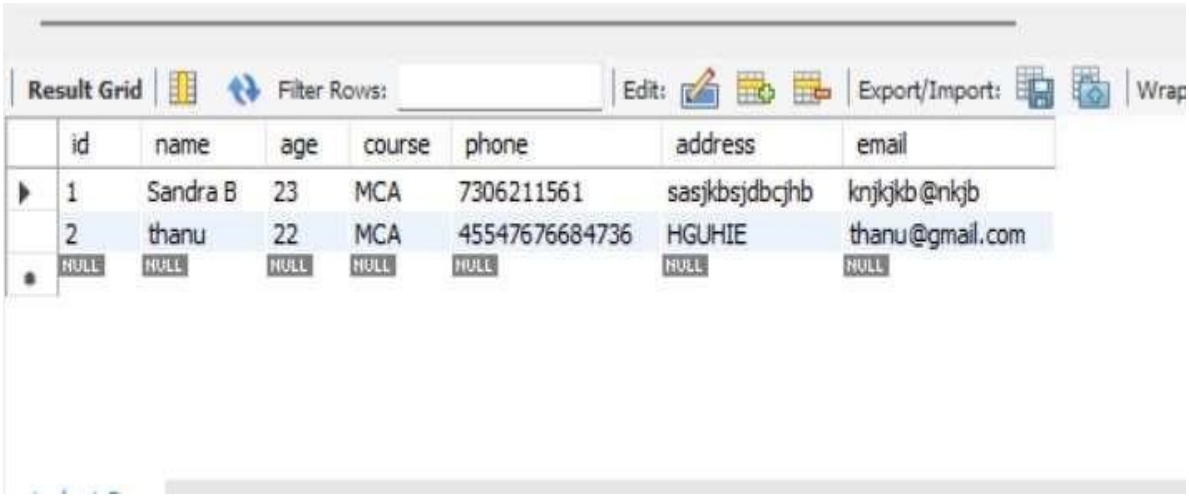
Desktop Application UI (Java Swing)

Screenshot Ideas:

- Main window with student list visible.
- Add Student Dialog showing input fields.

Descriptions (to place below or next to each screenshot):

- *Main Interface:* The home screen displays the list of students in a formatted manner using JList and allows users to interact via buttons.
- *Add/Edit Forms:* Users can input or update student information like name, age, course, phone, address, and email.
- *Responsive UI:* The Swing UI is designed for usability and clean layout using JPanel, BorderLayout, and custom colors.



This screenshot appears to be from MySQL Workbench or a similar SQL database management tool. Here's a breakdown of what is shown:

Table Overview

The table shown in the "Result Grid" seems to contain student-related data with the following columns:

Column Name	Description
id	A unique identifier for each student (probably the primary key).
name	The name of the student.
age	The age of the student.
course	The course the student is enrolled in (e.g., MCA).
phone	The student's phone number.
address	The student's address (though values are inconsistent).
email	The student's email address.

Purpose

This is likely a sample or test database, possibly for a **student management system** or similar academic project. It might be used to demonstrate database operations like **CRUD (Create, Read, Update, Delete)**.

Future Scope:

Summary of What Was Accomplished

The Student Management System project aimed to create an efficient and user-friendly platform to manage student records. The system was designed with both a Java Swing desktop application and a web-based interface using HTML, CSS, and JavaScript, connected to a MySQL database. The following key functionalities were successfully implemented:

Seamless Integration:

- The **backend Java APIs** handle data manipulation and storage in the MySQL database.
- The **frontend (HTML/CSS/JS)** interacts with the backend using HTTP requests to send and receive data in real-time.

- The **database** effectively stores all student information and supports CRUD operations efficiently.

User-Friendly Interface: Both the **desktop application** and **web interface** were designed to ensure an intuitive experience for the users, with clear forms, feedback on submission, and styled cards to present student data.

Real-time Data Display: The web version allows users to see updated student records below the form after submitting, thanks to dynamic rendering with JavaScript.

Ideas for Improvement

While the system is functional and fulfills its purpose, there are several areas where enhancements can be made to improve its usability, performance, and scalability:

1. **Advanced Search and Filtering**:
    - Implement a search bar or filter options for users to quickly find students by specific criteria like name, course, or age.
    - This would improve the user experience, especially with larger datasets.
2. **Authentication and Authorization**:
    - Add **login functionality** to ensure that only authorized users can access or modify student records.
    - Implement roles such as **admin** and **user**, where admins can manage student data, and users can only view it.
3. **Data Validation**:
    - Further enhance the input validation both on the frontend and backend to ensure that the data entered is accurate and in the correct format.
    - For instance, validating email formats, phone number formats, and ensuring age is a positive integer.
4. **Responsive Design**:
    - Improve the **web interface** to be fully responsive across different devices (mobile, tablet, etc.), ensuring that it's usable on various screen sizes.
5. **Cloud Integration and Backup**:
    - Implement cloud storage to save and back up the student data. This will allow for better scalability and security.

- Provide users with the ability to export the student records to CSV or PDF formats.

6. **Enhanced User Interface**:
    - Add more **interactive UI elements** like tooltips, modals for editing, and confirmation dialogs.
    - Implement animations to improve user interactions, providing a more modern and fluid experience.
7. **Performance Optimization**:
    - Optimize database queries, especially if the system scales up to manage larger datasets.
    - Use **pagination** for the student list to avoid rendering large amounts of data at once.

Conclusion

The Student Management System project has achieved its primary goal of creating a user-friendly platform for managing student information. However, with the suggested improvements, the system can be further enhanced to offer more advanced features, security, and scalability for future use in educational institutions.


THANK YOU