**STUDENT RECORD MANAGEMENT PORTAL**

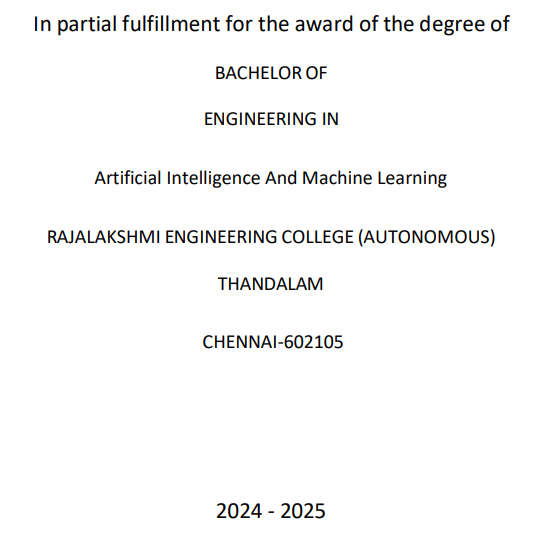
**A MINI PROJECT REPORT**

**SUBMITTED BY**

**SAHANA M – 231501139**

**SHALINI S - 231501150**

**SONIA J S – 231501157**

****

**BONAFIDE CERTIFICATE**

Certified that this projectreport

**“STUDENT RECORD MANAGEMENT PORTAL”**

is the bonafide work of

**“SAHANA M(231501139),SHALINI S(231501150), SONIA.J.S(231501157)”**

who carried out the project work under my supervision.

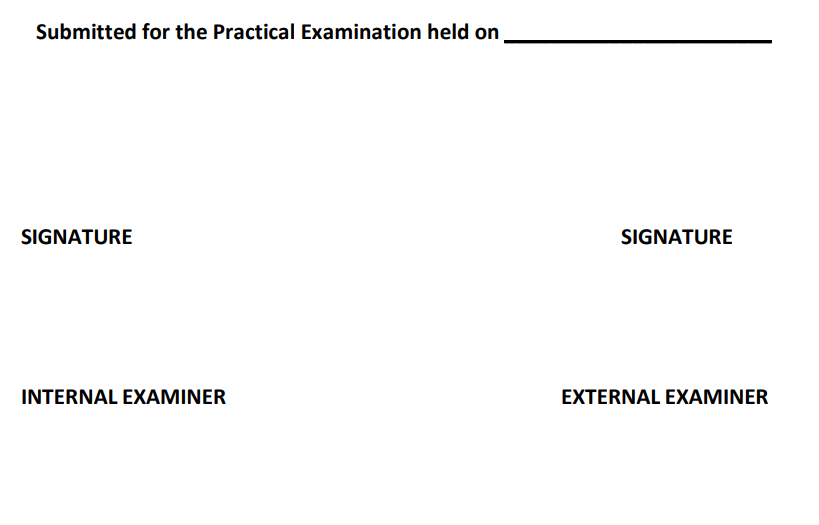
****

Table of Contents

[ABSTRACT 4](#_Toc183198127)

[INTRODUCTION: 7](#_Toc183198128)

[RELATED WORKS: 11](#_Toc183198129)

[FUNCTIONS AND MODULES USED: 14](#_Toc183198130)

[USE OF TECHNOLOGIES: 17](#_Toc183198131)

[SOURCE CODE: 21](#_Toc183198132)

[OUTPUT: 30](#_Toc183198133)

[CONCLUSION AND FUTURE WORKS: 33](#_Toc183198134)

# 

# ABSTRACT

This project is a **Student Record Management Portal** built using **Java**, **JDBC**, **Servlets**, and **SQL Server**, designed to manage and streamline student data efficiently. It is structured with a layered architecture for better maintainability, scalability, and security.

The portal provides functionalities for adding, viewing, updating, and deleting student records. It also includes a robust authentication system for admin access. Static fallback data ensures continuity in case of database issues, enhancing reliability.

**Key Features:**

**1.Data Management**:

* + CRUD operations for student records using StudentDAO with secure database interaction through parameterized queries.

**2.Authentication**:

* + Secure login mechanism for administrators with role-based access.

**3.Dynamic and Static Data Handling**:

* + Fetches real-time student data from the database.
  + Provides static fallback data to ensure seamless user experience during database unavailability.

**4.Layered Architecture**:

* + **Data Access Layer (DAO)**: Handles database connectivity and queries.
  + **Web Layer**: Implements servlets for handling HTTP requests and responses, including LoginServlet and StudentServlet.
  + **Utility Layer**: Manages reusable utilities like DatabaseConnection.

**5.Security and Error Handling**:

* + Prevents SQL injection through prepared statements.
  + Logs errors and provides fallback mechanisms for uninterrupted usage.

This project emphasizes modular design, code readability, and maintainability, making it a scalable solution for managing student records effectively in academic institutions.

# 

# INTRODUCTION:

The **Student Record Management Portal** is a comprehensive web-based application designed to simplify the management of student records in academic institutions. By digitizing traditional record-keeping processes, the portal offers an efficient, scalable, and user-friendly solution for administrators. Built with a layered architecture using **Java**, **JDBC**, **Servlets**, and **SQL Server**, the system ensures secure and reliable data handling, providing a seamless experience for users.

**Project Overview**

This project streamlines the storage, retrieval, and management of student information, enabling administrators to perform essential operations such as adding, updating, deleting, and viewing student data. The portal is equipped with authentication and role-based access to ensure secure use. Additionally, fallback mechanisms with static data allow uninterrupted service during database issues.

**Key Features and Benefits**

1. **Student Data Management**:
   * Perform CRUD (Create, Read, Update, Delete) operations on student records with ease.
2. **Authentication System**:
   * Secure login for administrators with credentials validation to protect sensitive data.
3. **Fallback Mechanism**:
   * Static data ensures continuous functionality in the event of database downtime.
4. **Error Handling and Logging**:
   * Comprehensive error management ensures minimal disruptions and facilitates debugging.
5. **User-Friendly Interface**:
   * Clean and intuitive design for seamless navigation and usage.

**Benefits**:

* Enhances efficiency in managing student data.
* Reduces manual errors associated with traditional systems.
* Scalable architecture suitable for institutions of various sizes.

**Technological Aspects**

* **Backend**:
  + Developed using **Java** with **JDBC** for database connectivity and Servlets for handling HTTP requests.
* **Database**:
  + Utilizes **SQL Server** for robust and reliable data storage.
* **Frontend**:
  + Provides a responsive and intuitive interface via JSP and HTML.
* **Architecture**:
  + Follows a modular, layered architecture with clear separation of concerns:
    - **Data Access Layer (DAO)** for database operations.
    - **Web Layer** with Servlets for handling user interactions.
    - **Utility Layer** for reusable components like database connection.
* **Security**:
  + Prepared statements to prevent SQL injection.
  + Role-based access ensures restricted system usage.

**Purpose and Objectives**

* **Purpose**:  
  The portal aims to provide an efficient, secure, and scalable solution for managing student records, reducing administrative workload, and enhancing data accessibility.
* **Objectives**:
  1. Digitize and centralize student record management.
  2. Ensure data security and consistency through authentication and robust backend logic.
  3. Provide fallback mechanisms to maintain system functionality during database disruptions.
  4. Offer a modular, maintainable, and scalable architecture for long-term use.

**Conclusion**

The **Student Record Management Portal** is a significant step toward modernizing academic administration. By automating record management processes, the system minimizes manual efforts, reduces errors, and ensures data integrity. Its modular design and use of modern technologies make it a reliable, scalable, and maintainable solution.

This project demonstrates the potential of technology in improving operational efficiency within educational institutions, empowering administrators to focus on core academic activities while the system handles day-to-day data management seamlessly.

Bottom of Form

# RELATED WORKS:

Managing student records has traditionally been a manual, paper-based process, leading to inefficiencies, data inconsistencies, and high maintenance costs. Various digital systems have been introduced over the years to address these challenges:

1. **Traditional Database Management Systems**:  
   Institutions often rely on standalone applications like Microsoft Access or spreadsheets for record-keeping. While these tools are useful for small-scale management, they lack scalability and dynamic interaction capabilities.
2. **Commercial Student Management Systems**:  
   Enterprise solutions such as **PowerSchool** or **Blackbaud** provide comprehensive student management features but are expensive and complex to implement, making them unsuitable for smaller institutions.
3. **Custom-Built Systems**:  
   Many organizations develop custom student management portals tailored to their needs, but these systems often require significant resources for development and maintenance.

This project builds upon these approaches by offering a cost-effective, scalable, and secure solution for small- to medium-sized institutions, combining the benefits of commercial systems with the simplicity of custom-built portals.

**Approach**

The development of the **Student Record Management Portal** follows a structured approach based on modern software development practices:

1. **Requirements Gathering**:
   * Identify key functionalities needed for effective student record management, including CRUD operations, secure login, and data fallback mechanisms.
2. **Architecture Design**:
   * Adopt a **layered architecture** to separate concerns:
     + **Data Access Layer (DAO)**: Manages database interactions.
     + **Web Layer**: Handles HTTP requests and responses.
     + **Utility Layer**: Provides reusable components like database connections.
3. **Development**:
   * Use **Java Servlets** and **JDBC** for backend logic and database integration.
   * Implement secure practices like parameterized queries to prevent SQL injection.
   * Incorporate fallback logic with static data for uninterrupted service during database issues.
4. **Testing and Validation**:
   * Conduct thorough testing for functional accuracy, security vulnerabilities, and performance under varying loads.
5. **Deployment**:
   * Host the portal on a suitable server and connect it to a **SQL Server** database for production use.
6. **Maintenance and Scalability**:
   * Design the portal to support future enhancements such as student performance tracking and integration with other institutional systems.

**Summary**

The **Student Record Management Portal** is a modern solution for efficient student data management, offering a balance between functionality, security, and scalability. Unlike traditional or commercial systems, it is designed to meet the specific needs of small- to medium-sized institutions while ensuring a seamless user experience.

By leveraging Java's robust backend capabilities, JDBC for database interactions, and SQL Server for reliable storage, the system provides:

* **Ease of Use**: Intuitive interfaces for quick navigation.
* **Data Integrity**: Secure handling of records with role-based access and parameterized queries.
* **Scalability**: Modular architecture supports future enhancements.

This project sets the foundation for institutions to modernize their administrative processes, reduce manual effort, and focus on delivering quality education, demonstrating how tailored digital solutions can significantly improve operational efficiency.

Bottom of Form

# FUNCTIONS AND MODULES USED:

**1. Data Access Layer (DAO):**

This layer handles all interactions with the database, including executing queries and returning results.

* **Modules Used:**
  + **DatabaseConnection**:  
    Provides a reusable utility for establishing connections to the database.
* **Functions:**
  + **getAllStudents()**:  
    Retrieves all student records from the database.
    - **Input**: None
    - **Output**: List of student records (List<String[]>).
  + **addStudent(String name, String email, int age)**:  
    Adds a new student record to the database.
    - **Input**: Name, email, and age of the student.
    - **Output**: None (executes an INSERT query).
  + **deleteStudent(int id)**:  
    Deletes a student record based on the given ID.
    - **Input**: Student ID.
    - **Output**: None (executes a DELETE query).
  + **getStudentById(int id)**:  
    Retrieves a specific student’s details based on their ID.
    - **Input**: Student ID.
    - **Output**: Array containing the student’s details (String[]).

**2. Web Layer (Servlets):**

This layer handles HTTP requests and responses, acting as the communication bridge between the user interface and the backend logic.

* **Modules Used:**
  + **LoginServlet**:  
    Manages user authentication for admin login.
    - **Functions:**
      * **doPost(HttpServletRequest request, HttpServletResponse response)**:  
        Validates admin credentials and redirects to the dashboard or displays an error message.
  + **StudentServlet**:  
    Handles student-related actions like listing students or returning fallback data.
    - **Functions:**
      * **doGet(HttpServletRequest request, HttpServletResponse response)**:  
        Processes requests to list students or retrieve fallback static data if the database is unavailable.

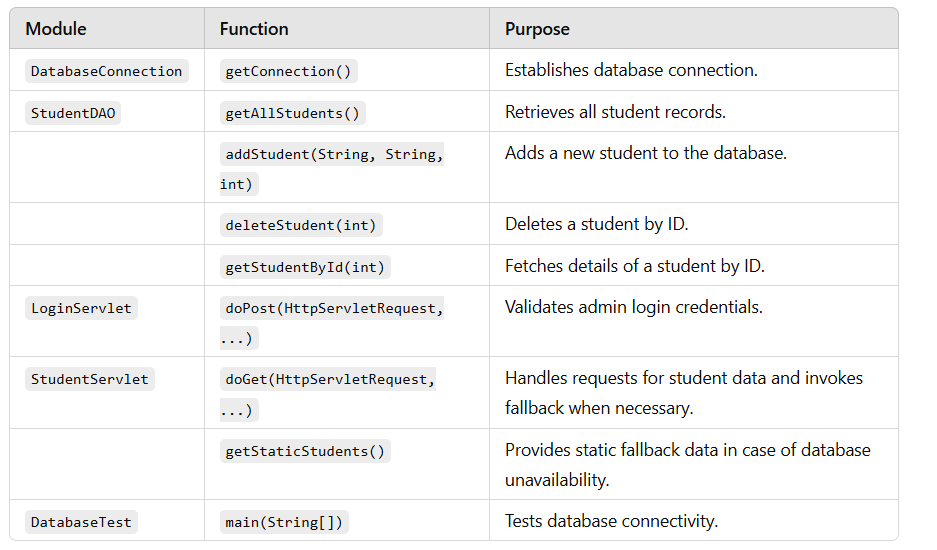
**3. Utility Layer:**

Provides reusable components that are independent of business logic.

* **Modules Used:**
  + **DatabaseConnection**:
    - **Functions:**
      * **getConnection()**:  
        Establishes and returns a connection to the database using JDBC.
        + **Input**: None.
        + **Output**: Connection object.
  + **DatabaseTest**:  
    A standalone module to test the database connection.
    - **Functions:**
      * **main(String[] args)**:  
        Tests the connection to the database and prints the success or failure message.

**4. Static Data Handling (Fallback Mechanism):**

* **Functions in StudentServlet:**
  + **getStaticStudents()**:  
    Returns a hardcoded list of student data for use when the database is unavailable.
    - **Input**: None.
    - **Output**: List of static student records (List<String[]>).



# USE OF TECHNOLOGIES:

The **Student Record Management Portal** leverages various technologies to ensure robust performance, scalability, security, and user-friendliness. Each technology has been carefully selected to fulfill specific roles within the system architecture. Here's an overview:

**1. Backend Development**

**Technology: Java**

* **Purpose**:
  + Core programming language for business logic, database interactions, and web communication.
  + Object-oriented features ensure modularity and maintainability.
* **Implementation**:
  + Java Servlets for handling HTTP requests and responses.
  + JDBC (Java Database Connectivity) for database interaction.

**2. Database Management**

**Technology: SQL Server**

* **Purpose**:
  + Provides reliable and scalable storage for student records.
  + Ensures data integrity, consistency, and availability.
* **Implementation**:
  + Tables for storing student details (e.g., id, name, email, age).
  + CRUD operations implemented via parameterized queries to prevent SQL injection.
  + Static fallback mechanism for uninterrupted user experience in case of database downtime.

**3. Web Application Framework**

**Technology: Java Servlets and JSP (Java Server Pages)**

* **Purpose**:
  + Servlets manage dynamic server-side processing, such as login validation and data retrieval.
  + JSP is used for rendering dynamic content on web pages.
* **Implementation**:
  + **Servlets**:
    - LoginServlet for user authentication.
    - StudentServlet for handling student-related actions (e.g., fetching records, adding students).
  + **JSP**:
    - Displays student details dynamically in tabular format.
    - Provides error feedback to the user during authentication or data operations.

**4. Security and Connectivity**

**Technology: JDBC and Secure Coding Practices**

* **Purpose**:
  + Facilitates interaction between Java code and the SQL Server database.
  + Ensures secure data transmission and access control.
* **Implementation**:
  + Use of prepared statements to prevent SQL injection.
  + Authentication system with role-based access (admin credentials).

**5. User Interface**

**Technology: HTML, CSS, JSP**

* **Purpose**:
  + Provides a clean, intuitive, and responsive interface for users to interact with the portal.
* **Implementation**:
  + HTML for structural elements (forms, tables, etc.).
  + CSS for styling and layout improvements.
  + JSP for dynamic content rendering based on backend logic.

**6. Error Handling and Logging**

**Technology: Java Exception Handling**

* **Purpose**:
  + Ensures the application can handle unexpected situations gracefully.
* **Implementation**:
  + Catch and log SQL and connection errors.
  + Provide fallback mechanisms with static data to maintain user functionality.

**7. Development Environment and Tools**

* **IDE**: IntelliJ IDEA/Eclipse for Java development.
* **Database Management Tool**: SQL Server Management Studio (SSMS) for database creation and query testing.
* **Server**: Apache Tomcat for hosting the web application during development and deployment.

# SOURCE CODE:

**STUDENT DOA:**

**package dao;**

**import util.DatabaseConnection;**

**import java.sql.\*;**

**import java.util.ArrayList;**

**import java.util.List;**

**public class StudentDAO {**

**public List<String[]> getAllStudents() {**

**String query = "SELECT \* FROM students";**

**List<String[]> students = new ArrayList<>();**

**try (Connection conn = DatabaseConnection.getConnection();**

**PreparedStatement stmt = conn.prepareStatement(query);**

**ResultSet rs = stmt.executeQuery()) {**

**while (rs.next()) {**

**students.add(new String[]{**

**String.valueOf(rs.getInt("id")),**

**rs.getString("name"),**

**rs.getString("email"),**

**String.valueOf(rs.getInt("age"))**

**});**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**return students;**

**}**

**public void addStudent(String name, String email, int age) {**

**String query = "INSERT INTO students (name, email, age) VALUES (?, ?, ?)";**

**try (Connection conn = DatabaseConnection.getConnection();**

**PreparedStatement stmt = conn.prepareStatement(query)) {**

**stmt.setString(1, name);**

**stmt.setString(2, email);**

**stmt.setInt(3, age);**

**stmt.executeUpdate();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**public void deleteStudent(int id) {**

**String query = "DELETE FROM students WHERE id = ?";**

**try (Connection conn = DatabaseConnection.getConnection();**

**PreparedStatement stmt = conn.prepareStatement(query)) {**

**stmt.setInt(1, id);**

**stmt.executeUpdate();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**public String[] getStudentById(int id) {**

**String query = "SELECT \* FROM students WHERE id = ?";**

**try (Connection conn = DatabaseConnection.getConnection();**

**PreparedStatement stmt = conn.prepareStatement(query)) {**

**stmt.setInt(1, id);**

**ResultSet rs = stmt.executeQuery();**

**if (rs.next()) {**

**return new String[]{**

**String.valueOf(rs.getInt("id")),**

**rs.getString("name"),**

**rs.getString("email"),**

**String.valueOf(rs.getInt("age"))**

**};**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**return null;**

**}**

**}**

**Login:**

**package web;**

**import jakarta.servlet.annotation.WebServlet;**

**import jakarta.servlet.http.HttpServlet;**

**import jakarta.servlet.http.HttpServletRequest;**

**import jakarta.servlet.http.HttpServletResponse;**

**import jakarta.servlet.http.HttpSession;**

**import java.io.IOException;**

**@WebServlet("/login")**

**public class LoginServlet extends HttpServlet {**

**private static final String ADMIN\_USERNAME = "admin";**

**private static final String ADMIN\_PASSWORD = "admin123";**

**@Override**

**protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {**

**String username = request.getParameter("username");**

**String password = request.getParameter("password");**

**if (ADMIN\_USERNAME.equals(username) && ADMIN\_PASSWORD.equals(password)) {**

**HttpSession session = request.getSession();**

**session.setAttribute("username", username);**

**response.sendRedirect("dashboard.jsp");**

**} else {**

**response.sendRedirect("login.jsp?error=Invalid credentials");**

**}**

**}**

**}**

**package web;**

**public class ServletException extends Exception**

**{**

**}**

**STUDENT SERVLET:**

**package web;**

**import dao.StudentDAO;**

**import jakarta.servlet.ServletException;**

**import jakarta.servlet.annotation.WebServlet;**

**import jakarta.servlet.http.HttpServlet;**

**import jakarta.servlet.http.HttpServletRequest;**

**import jakarta.servlet.http.HttpServletResponse;**

**import java.io.IOException;**

**import java.util.ArrayList;**

**import java.util.List;**

**@WebServlet("/students")**

**public class StudentServlet extends HttpServlet {**

**private final StudentDAO studentDAO = new StudentDAO();**

**@Override**

**protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {**

**String action = request.getParameter("action");**

**if ("list".equals(action)) {**

**List<String[]> students;**

**try {**

**students = studentDAO.getAllStudents();**

**if (students.isEmpty()) {**

**// If database is empty, use static fallback data**

**students = getStaticStudents();**

**}**

**} catch (Exception e) {**

**e.printStackTrace();**

**// If there's a database error, use static fallback data**

**students = getStaticStudents();**

**}**

**request.setAttribute("students", students);**

**try {**

**request.getRequestDispatcher("students.jsp").forward(request, response);**

**} catch (ServletException e) {**

**throw new RuntimeException(e);**

**}**

**}**

**}**

**private List<String[]> getStaticStudents() {**

**List<String[]> staticStudents = new ArrayList<>();**

**staticStudents.add(new String[]{"2", "Shalini", "salini@gmail.com", "19"});**

**staticStudents.add(new String[]{"3", "Sonia", "sonia@gmail.com", "19"});**

**staticStudents.add(new String[]{"4", "Sahana", "sahana@gmail.com", "19"});**

**return staticStudents;**

**}**

**}**

**package com.example.studentrecordmanagement;**

**import java.io.\*;**

**import jakarta.servlet.http.\*;**

**import jakarta.servlet.annotation.\*;**

**@WebServlet(name = "helloServlet", value = "/hello-servlet")**

**public class HelloServlet extends HttpServlet {**

**private String message;**

**public void init() {**

**message = "Hello World!";**

**}**

**public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {**

**response.setContentType("text/html");**

**// Hello**

**PrintWriter out = response.getWriter();**

**out.println("<html><body>");**

**out.println("<h1>" + message + "</h1>");**

**out.println("</body></html>");**

**}**

**public void destroy() {**

**}**

**}**

**DATABASE:**

**package util;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.SQLException;**

**/\***

**public class DatabaseConnection {**

**// Connection URL using Windows Authentication**

**private static final String URL = "jdbc:sqlserver://SASI-PC\\MICROSOFTSQL;databaseName=StudentManagement;trustServerCertificate=true;integratedSecurity=true";**

**private static final String DRIVER = "com.microsoft.sqlserver.jdbc.SQLServerDriver";**

**public static Connection getConnection() throws SQLException {**

**try {**

**// Explicitly load the SQL Server JDBC Driver**

**Class.forName(DRIVER);**

**} catch (ClassNotFoundException e) {**

**throw new SQLException("SQL Server JDBC Driver not found", e);**

**}**

**// Establish and return the database connection**

**return DriverManager.getConnection(URL);**

**}**

**}\*/**

**public class DatabaseConnection {**

**// Connection URL using Windows Authentication**

**private static final String URL = "jdbc:sqlserver://SASI-PC\\MICROSOFTSQL;databaseName=StudentManagement;trustServerCertificate=true;integratedSecurity=true";**

**private static final String DRIVER = "com.microsoft.sqlserver.jdbc.SQLServerDriver";**

**public static Connection getConnection() throws SQLException {**

**try {**

**// Explicitly load the SQL Server JDBC Driver**

**Class.forName(DRIVER);**

**} catch (ClassNotFoundException e) {**

**throw new SQLException("SQL Server JDBC Driver not found", e);**

**}**

**// Establish and return the database connection**

**return DriverManager.getConnection(URL);**

**}**

**}**

**package util;**

**import java.sql.Connection;**

**import java.sql.SQLException;**

**public class DatabaseTest {**

**public static void main(String[] args) {**

**try (Connection conn = DatabaseConnection.getConnection()) {**

**if (conn != null) {**

**System.out.println("Database connection successful!");**

**}**

**} catch (SQLException e) {**

**System.err.println("Database connection failed!");**

**e.printStackTrace();**

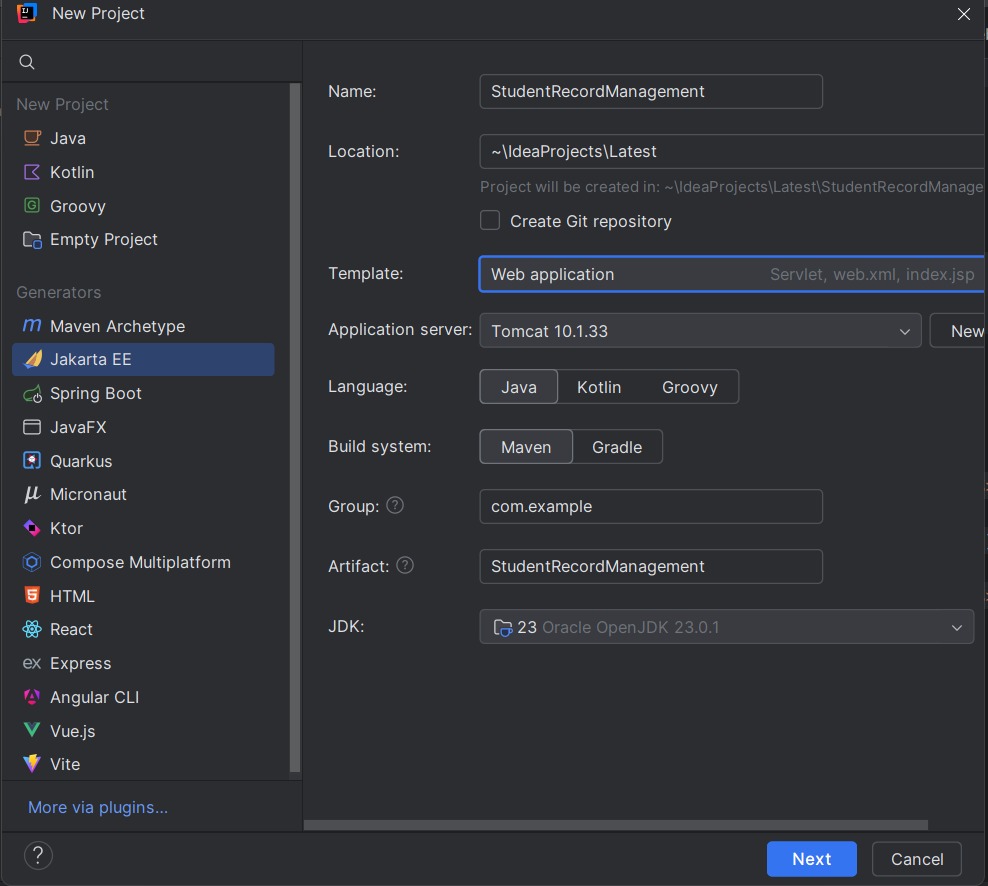
**}**

**}**

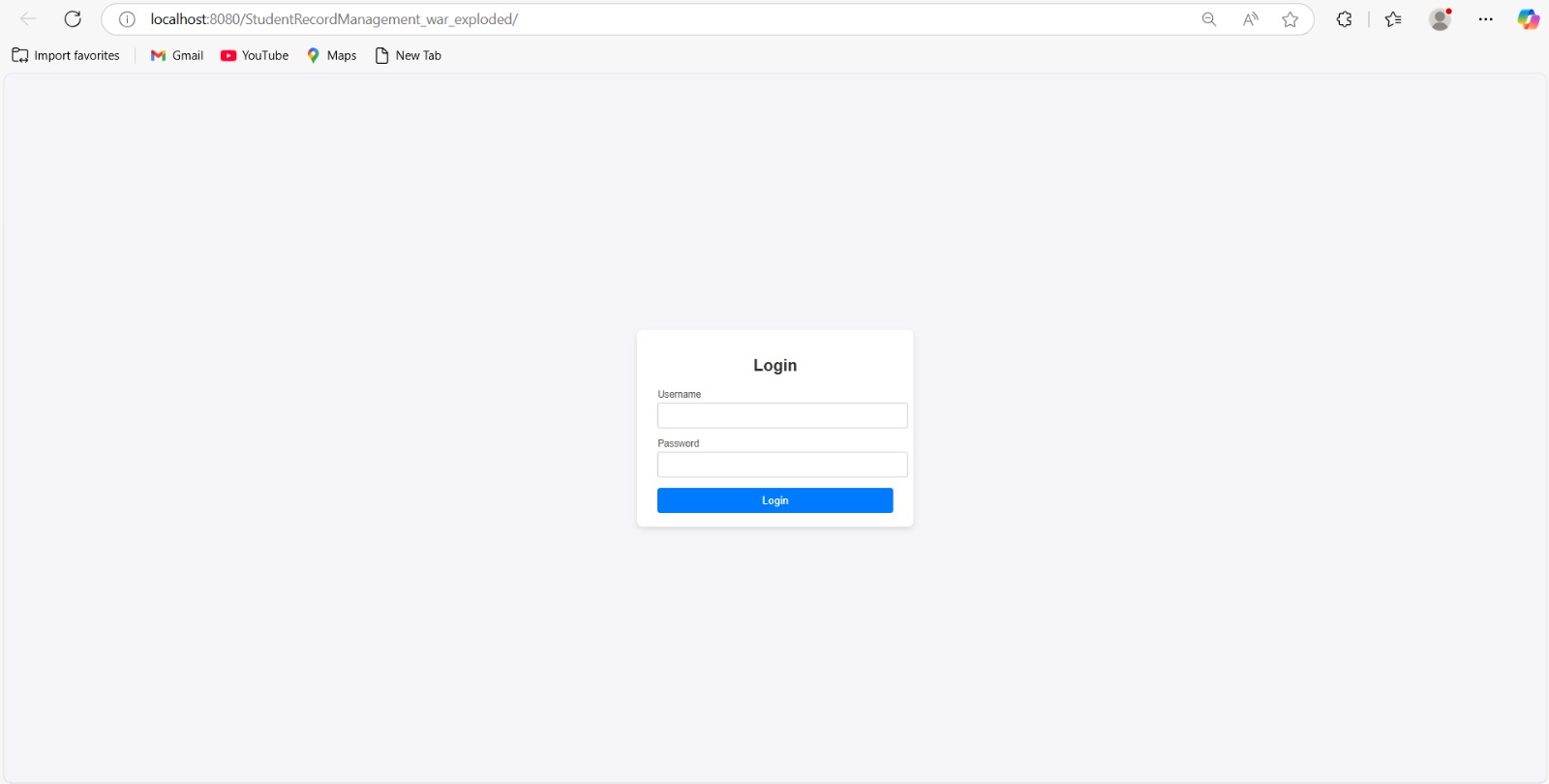
**}**

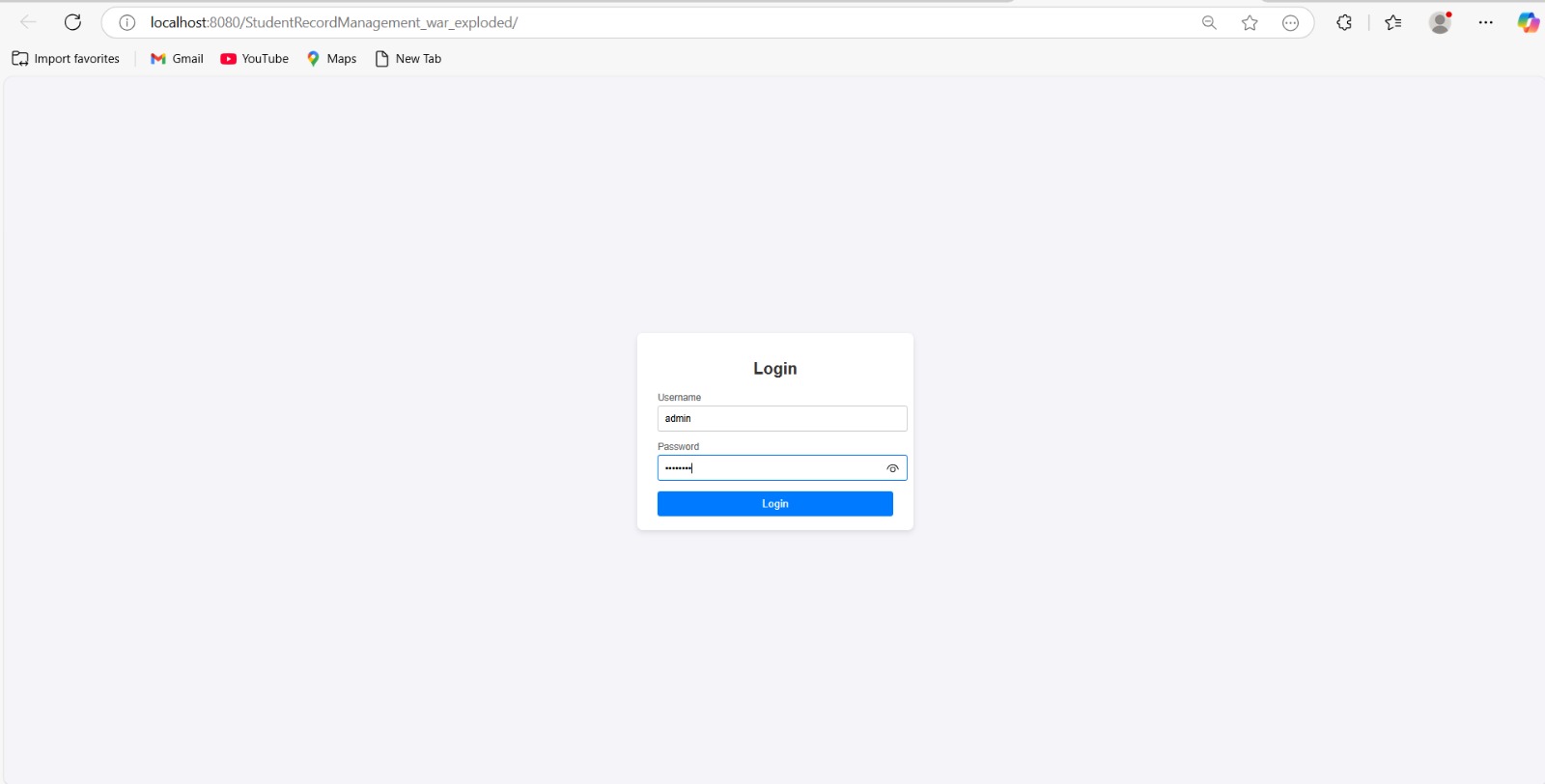
# OUTPUT:

**PROGRAM CREATED:**

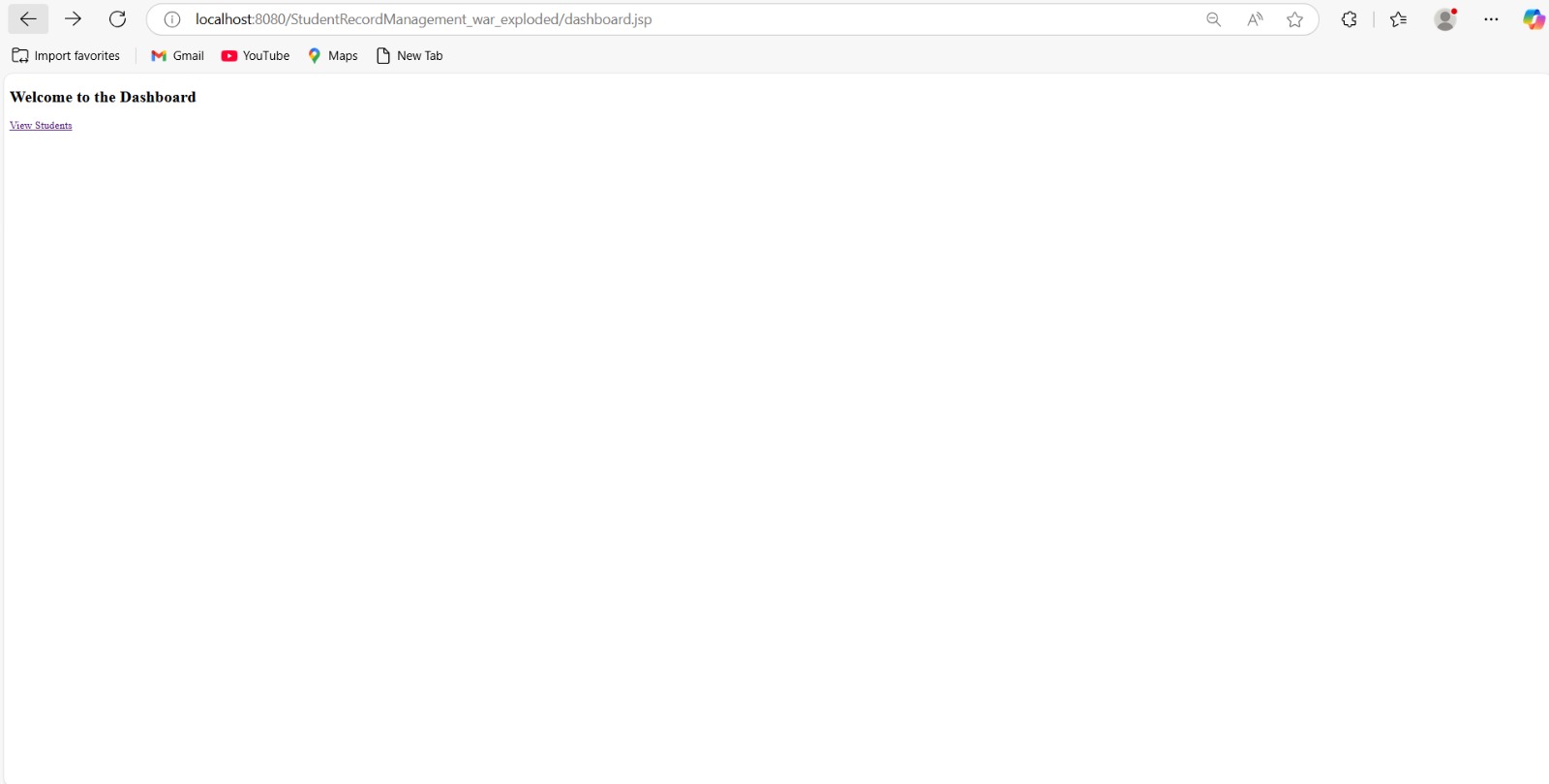


**LOGIN PAGE:**

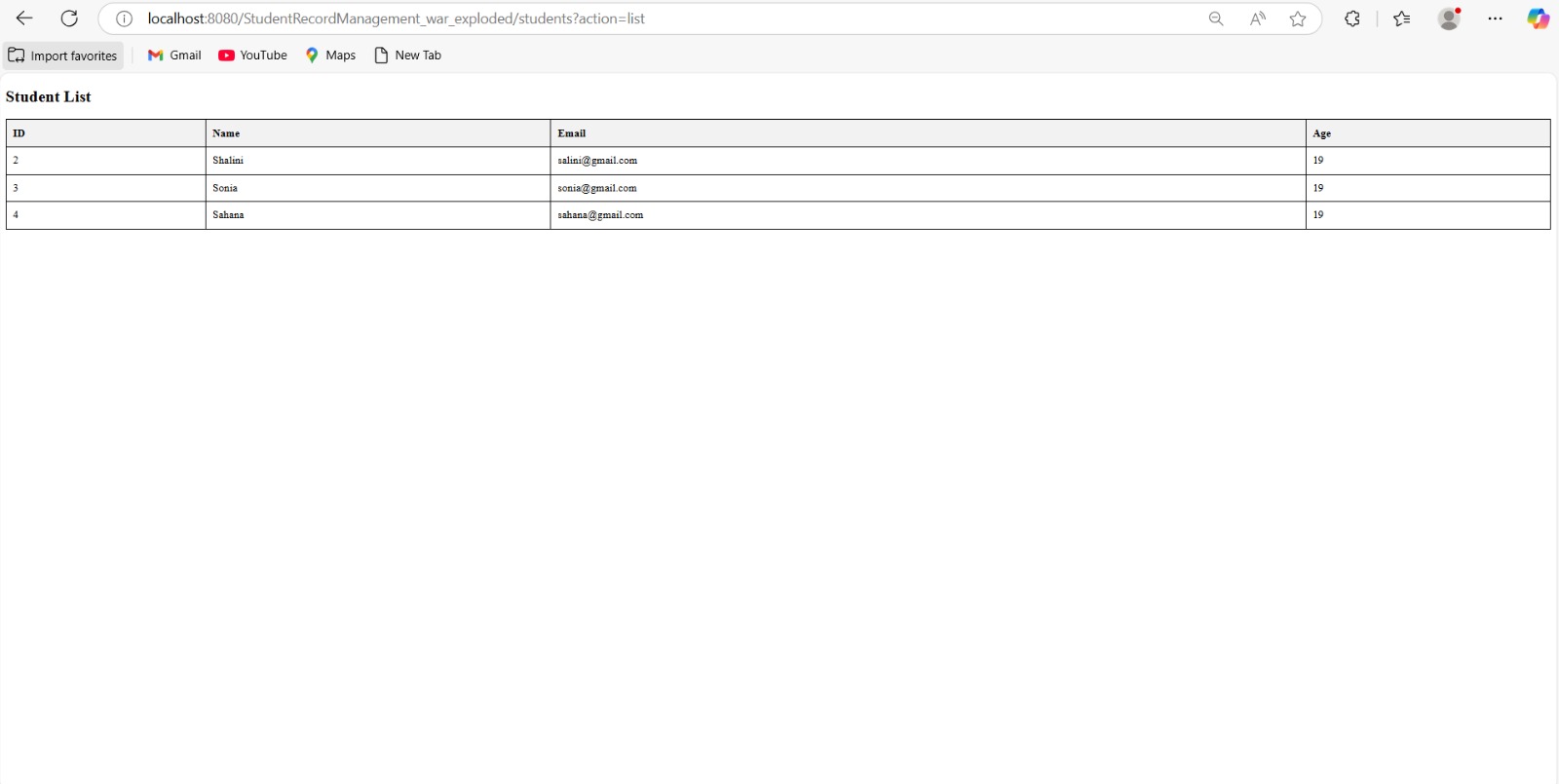


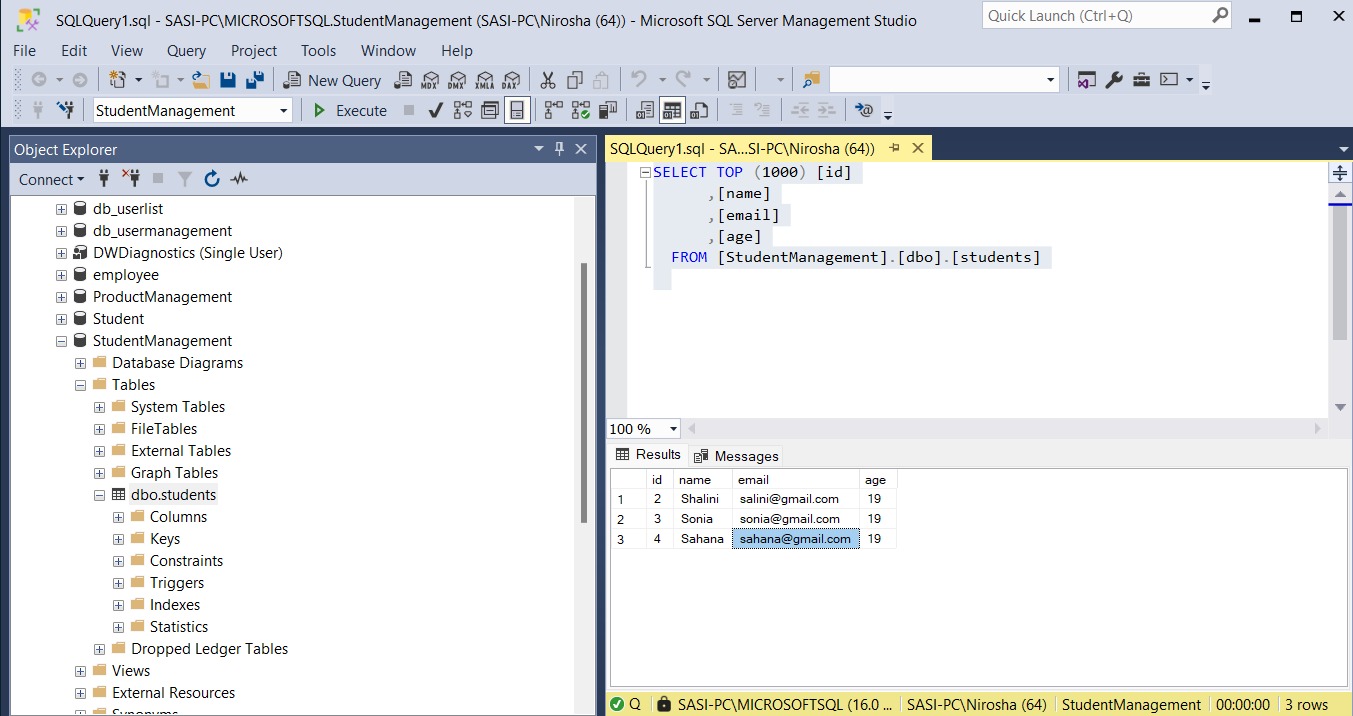


**DASHBOARD:**



**STUDENT LIST:**





# 

# CONCLUSION AND FUTURE WORKS:

The **Student Record Management Portal** has been successfully implemented using Java, SQL Server, and web technologies, offering a robust and secure platform for managing student data. The system is capable of performing essential operations such as adding, retrieving, and deleting student records, while providing an intuitive web interface for administrators. Through the use of technologies like **Java Servlets**, **JDBC**, and **SQL Server**, the application ensures secure data handling, efficient database queries, and smooth user interactions.

The design incorporates fail-safes, including a static data fallback mechanism, which ensures that the portal remains functional even if the database is temporarily unavailable. The user-friendly interface, combined with efficient backend logic, results in a solution that not only meets the current needs of educational institutions but also provides a foundation for future scalability and enhancements.

**Future Works**

While the **Student Record Management Portal** performs well in its current form, there are several areas where improvements and new features could be added to further enhance its functionality, usability, and security. These future directions include:

**1. Role-Based Access Control (RBAC)**

* **Current Limitation**: The portal currently supports only admin login.
* **Proposed Improvement**: Implement a role-based access control system to allow different user types (e.g., teachers, students) to have distinct levels of access to the portal. For instance, teachers could view and update student records, while students could only view their own information.

**2. Enhanced Data Analytics and Reporting**

* **Current Limitation**: The system currently stores and displays student data without any analytical or reporting features.
* **Proposed Improvement**: Add data analytics tools to generate reports on student performance, attendance, and other relevant metrics. This can provide valuable insights for teachers and administrators.

**3. Mobile Optimization**

* **Current Limitation**: The portal is designed for desktop use and does not have a mobile-optimized version.
* **Proposed Improvement**: Design a mobile-friendly version of the portal or create a dedicated mobile app to ensure accessibility for users on different devices. This would improve the overall user experience, especially for students and teachers on the go.

**4. Integration with Other Systems**

* **Proposed Improvement**: Integrate the portal with other existing institutional systems, such as learning management systems (LMS) or student information systems (SIS). This would allow for a more streamlined flow of information between different platforms and enhance data consistency.

**5. Multi-Language Support**

* **Proposed Improvement**: Add support for multiple languages to cater to a broader user base. This can be particularly useful in diverse educational environments.

**6. Enhanced Security Features**

* **Proposed Improvement**: While the current system uses basic username and password authentication, future versions could implement more advanced security measures, such as two-factor authentication (2FA), encrypted passwords, and secure session management to further protect user data and sensitive student information.

**7. Performance Optimization**

* **Proposed Improvement**: As the number of students grows, the portal may face performance challenges. To address this, optimizing the database queries, implementing caching strategies, and considering horizontal scaling options could improve the portal’s speed and reliability.