Student Grading System (Report)

Name: Akram Jaghoub

Table of Contents

Table of figures

1.0 Introduction

Welcome to the documentation for the Grading System, a powerful platform designed to simplify the management of student grades, course enrollment, and academic performance analysis. This documentation is your comprehensive guide, providing insights into the system's capabilities, architecture, and usage.

Overview

The Grading System is a vital tool for educational institutions, offering educators an efficient way to record and evaluate student grades while enabling students to track their academic progress. With user-friendly interfaces, robust data security, and insightful analysis, the Grading System streamlines administrative tasks and enhances the learning experience.

Key Features

- **Efficient Grade Management:** Seamlessly record and manage student grades, allowing instructors to input and update scores for assignments, exams, and projects.

- **Comprehensive Course Enrollment:** Simplify course enrollment for students and maintain a central record of course offerings, making academic planning effortless.

- **Insightful Analysis:** Gain valuable insights into student performance and course effectiveness using built-in analysis tools, enabling data-driven decisions.

- **Role-Based Access Control:** Implement role-based access control to ensure data security and privacy, allowing users to access relevant data and functions.

- **User-Friendly Interface:** Intuitive user interface design ensures easy navigation and data entry, accommodating users of all technical levels.

- **Scalability and Integration:** Built for scalability, the system adapts to your institution's needs and integrates seamlessly with existing educational tools.

•

2.0 Role Access

2.1 Admin

Regarding administrators, the way the system worked was that they could immediately access the database by using their database login information. This can be seen in figure 1 below. I'll explain this further: administrators didn't need to indicate a specific role. Instead, they could use certain "magic words" to access and manipulate the database.And what if wrong credentials were entered (as shown in figure 2), users won't realize that an admin page even exists. They would simply encounter a situation where they need to select a role in order to use the system, without any knowledge of the hidden admin page.

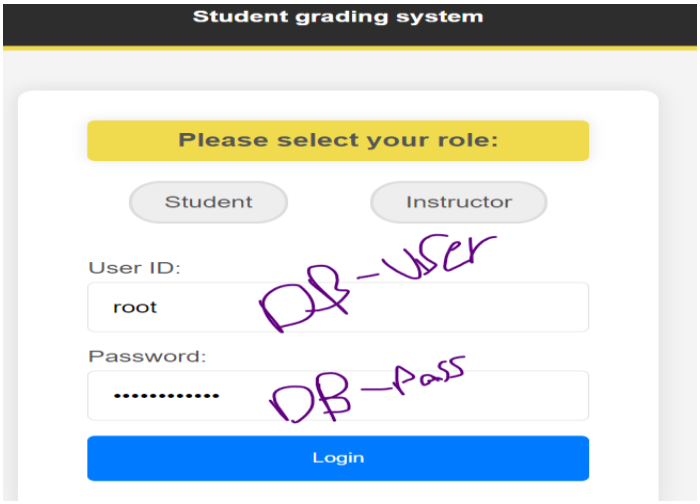


Figure 1: admin-login

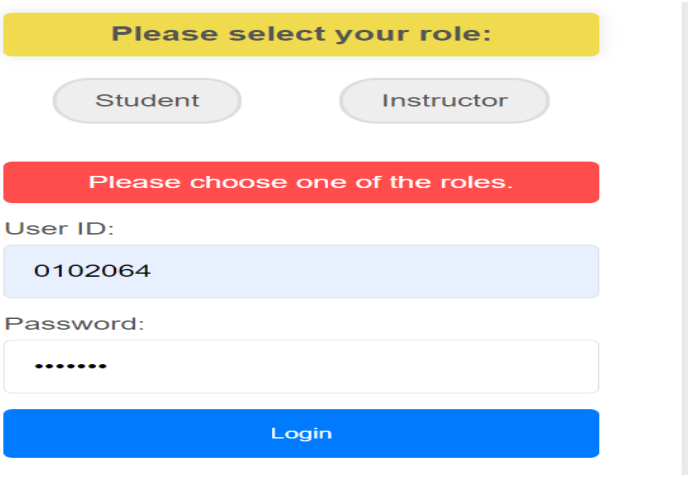


Figure 2: admin-wrong-credentials

```
1 usage
public boolean isAdminCredentials(String userIdStr, String password){
    return userIdStr.equalsIgnoreCase(DB_USERNAME) && password.equalsIgnoreCase(DB_PASSWORD);
}
```

Figure 3: admin-validation

2.2 Instructors

To identify instructors and what they can do, we have a method. They just need to click on the "instructor" button and put in their details. Then, we check these details against the database. For the first check in figure 4, I used a rule where IDs for students and courses are made of 7 numbers, like how the University of Jordan does it.

If someone types in a 7-number ID that's right but the details are wrong as shown in figure 5, the system will show an error saying that the instructor's login information is not correct.

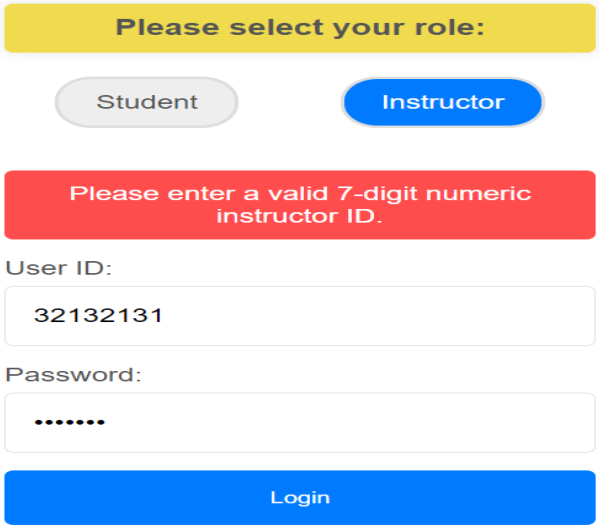


Figure 4: invalid 7-digit

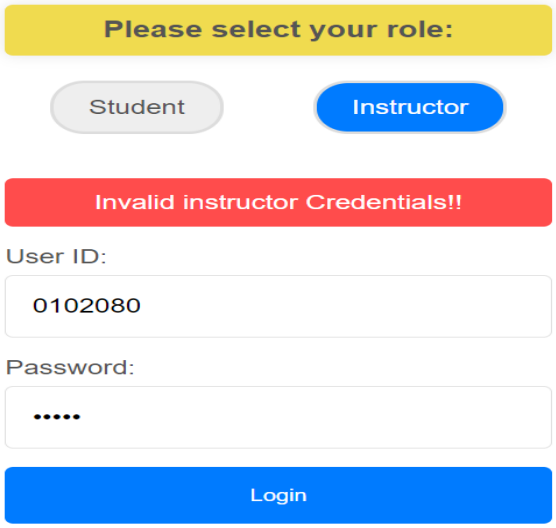


Figure 5: wrong instructor credentials

3.3 Students

As for the students, they really work the same way instructors work so I really won't be talking about it in this section.

3.0 Next Steps (dashboards)

Once users successfully log in, they are directed to their specific graphical user interfaces:

3.1 Admin Dashboard:

Administrators have a page displaying database tables they can manage. They have the ability to perform actions like creating, reading, updating, and deleting data (CRUD operations). In the student grading system I developed, there are three tables available—students, instructors, and courses—for administrators to choose from. Refer to figures 6 and 7 for distinct interfaces for each part of the program.

```
Hey ADMIN, please enter your username: root
Enter your password: 21232619boom
Successfully logged in
Requesting database tables......
Please choose one of the following tables:
1. students     2. instructors      3. courses
1
| student_id    | username      |
+---------------+---------------+
| 202051        | kmal          |
+---------------+---------------+
| 202061        | noor          |
+---------------+---------------+
| 202063        | mahmod        |
```

Figure 6: admin_dashboard_console

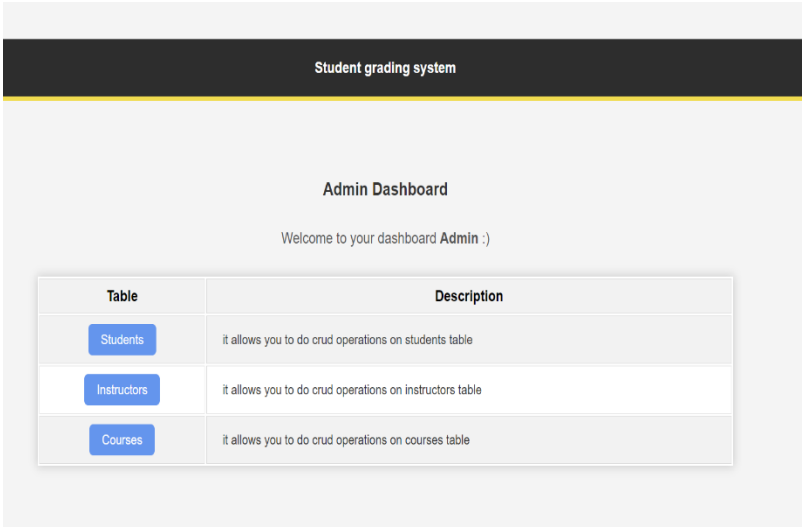


Figure 7: admin_dashboard_frontend

"Please note I will be only using front-end figures for the rest of the documentation."

3.2 Instructor Dashboard

The instructors' dashboard includes four functions, which I will explain in depth for each role mentioned in section 4.0. For now, please refer to figure 8.

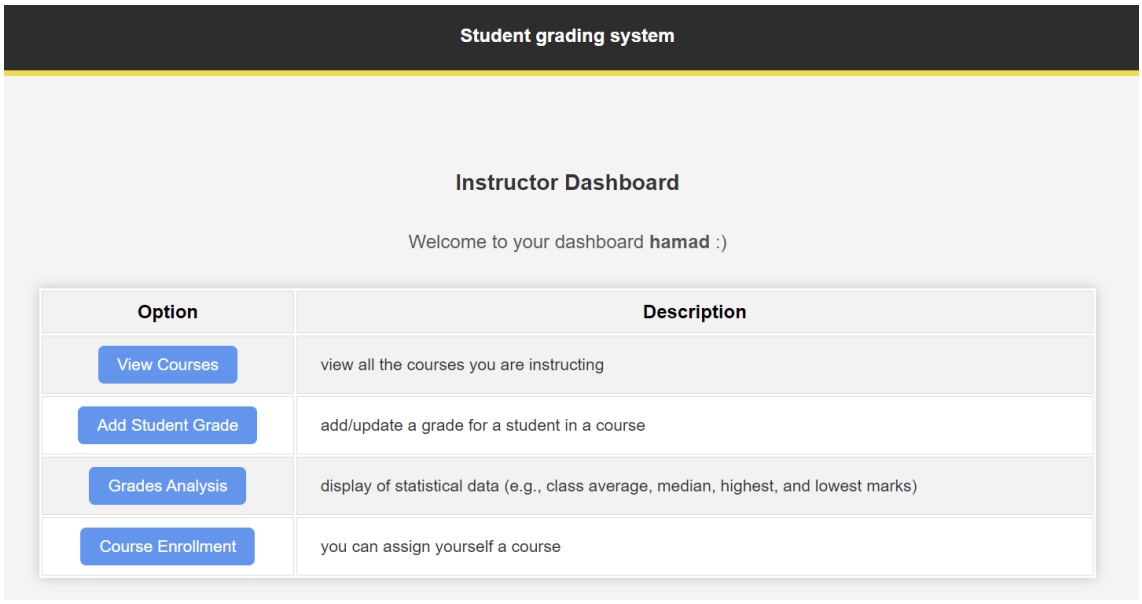


Figure 8: instructor_dashboard

3.3 Instructor Dashboard

The students' dashboard includes four functions, which I will explain in depth for each role mentioned in section 4.0. For now, please refer to figure 8.

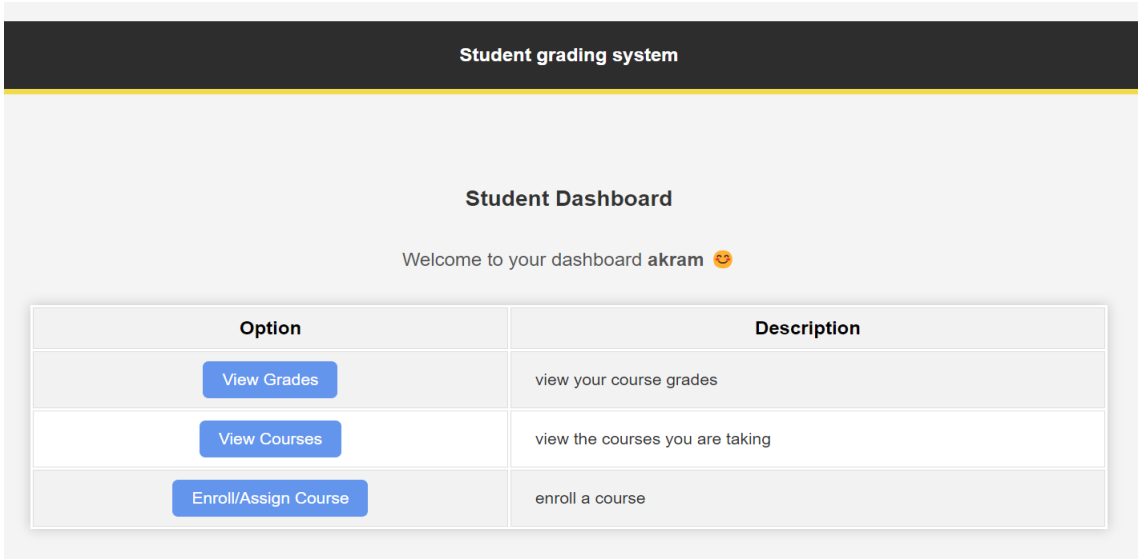


Figure 9: student_dashboard

4.0 Functional Requirements

4.1 Admins

- **CREATE:** Admins have the ability to add new information to the system. This includes adding new students, instructors, and courses by providing essential details like user IDs, usernames, passwords (for instructors and students), as well as course IDs and names. This expands the system's database with new entries look figure 10.

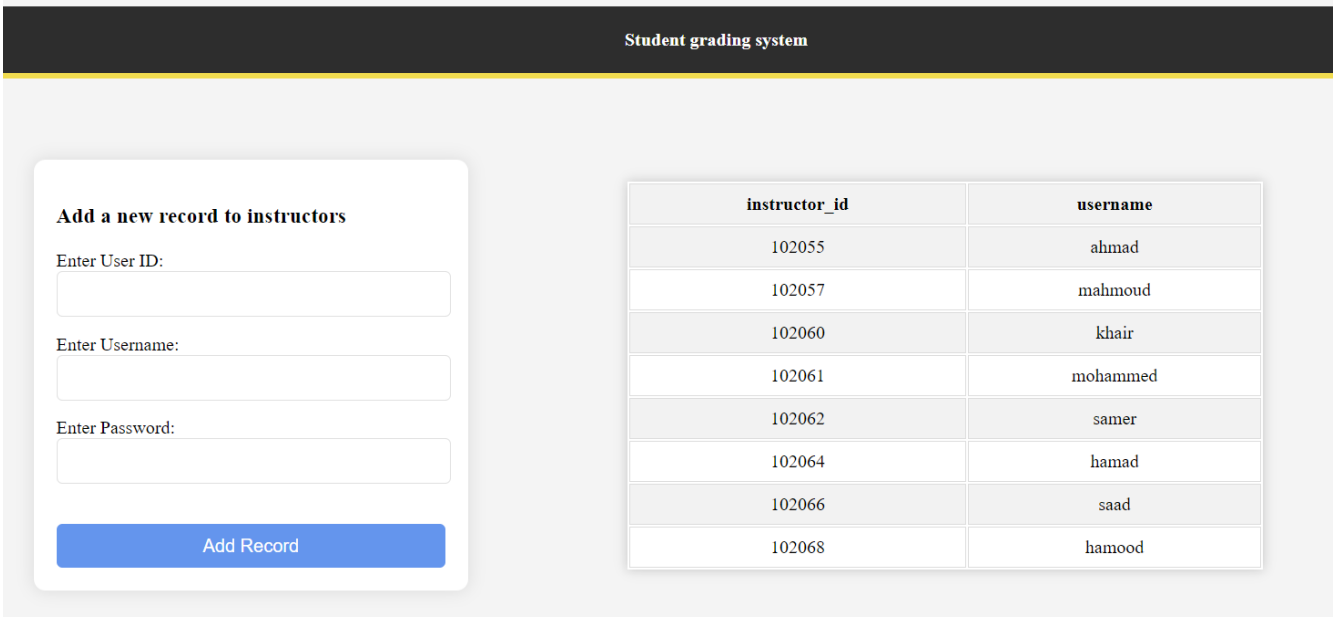


Figure 10: Create Page

- **READ**: Admins can view all records in the database. They have the privilege to access and retrieve information about students, instructors, and courses.

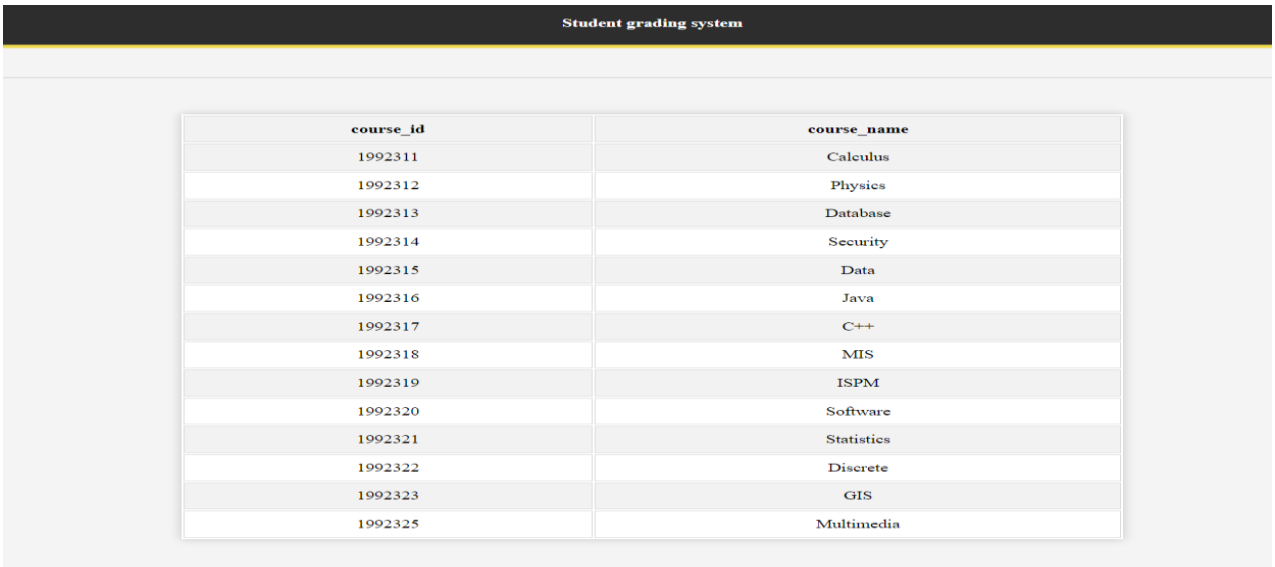
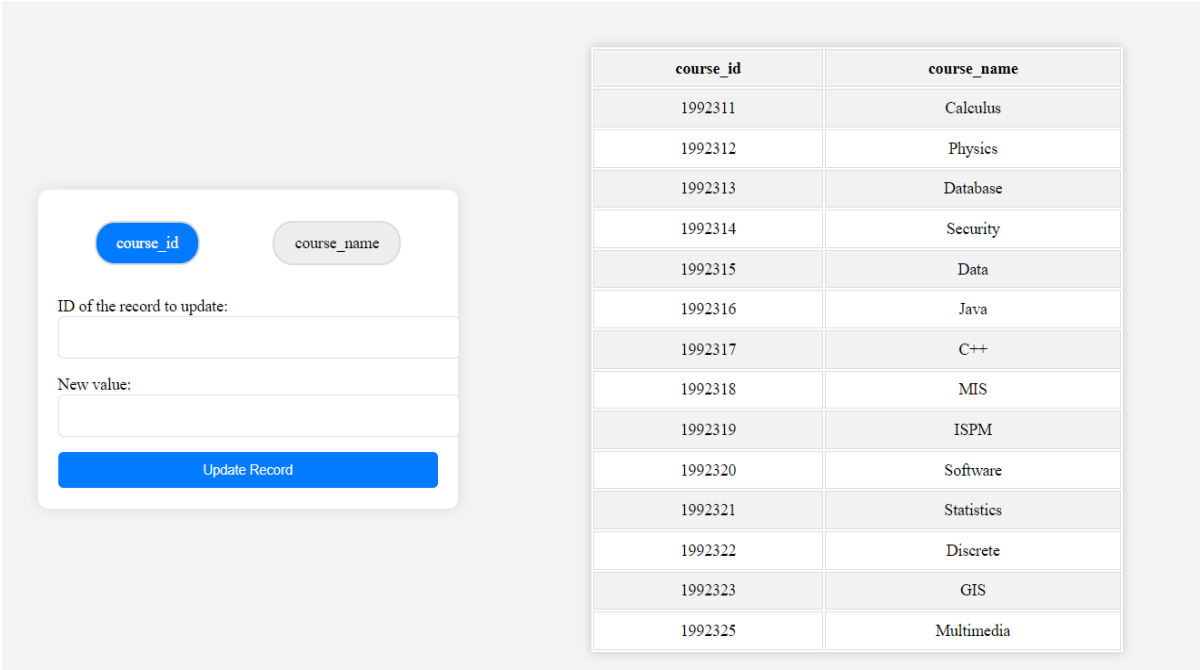


Figure 11: Read Page

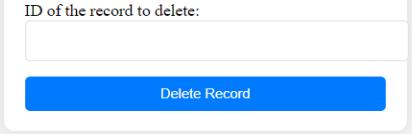
- **UPDATE:** Admins possess the capability to make changes to existing data. This empowers them to alter student particulars, instructor details, and course characteristics. There are two available choices: either modify user or course IDs or adjust user or course names.



course_id	course_name
1992311	Calculus
1992312	Physics
1992313	Database
1992314	Security
1992315	Data
1992316	Java
1992317	C++
1992318	MIS
1992319	ISPM
1992320	Software
1992321	Statistics
1992322	Discrete
1992323	GIS
1992325	Multimedia

Figure 12: Update Page

- **DELETE:** Admins hold the authority to eliminate records from the database. This action might entail the removal of student profiles, instructor data, or course listings. The process involves entering the respective ID of the field that needs to be deleted.



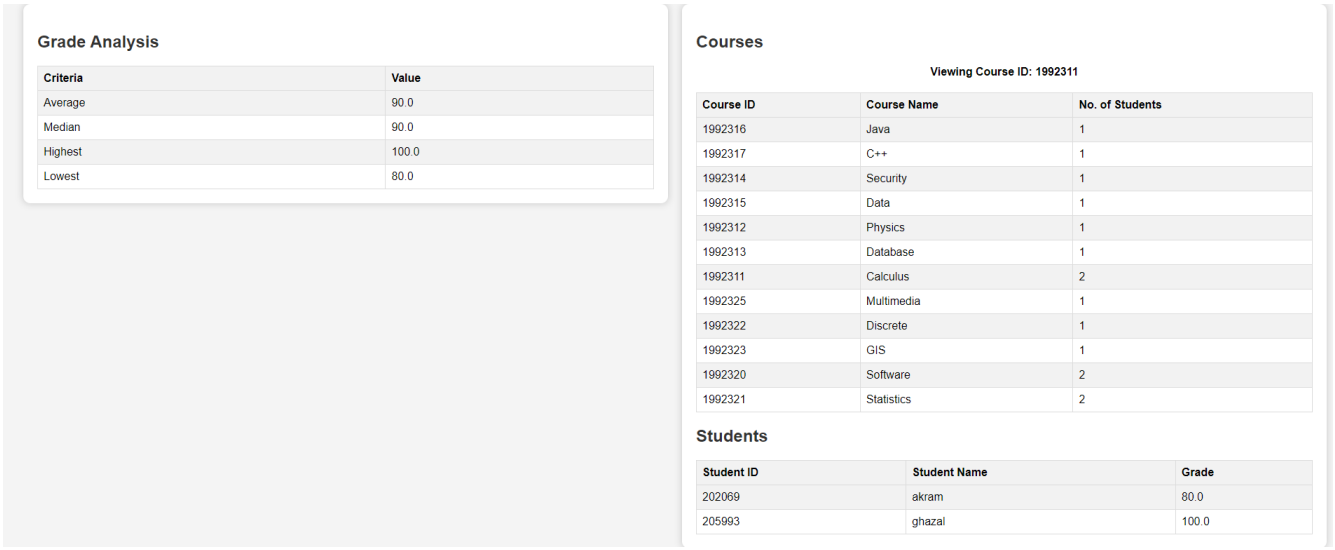
course_id	course_name
1992311	Calculus
1992312	Physics
1992313	Database
1992314	Security
1992315	Data
1992316	Java
1992317	C++
1992318	MIS
1992319	ISPM
1992320	Software
1992321	Statistics
1992322	Discrete
1992323	GIS
1992325	Multimedia

Figure 13: Delete Page

4.2 Instructors

- **View Assigned Courses**: Instructors can see a list of courses they are assigned to teach. This allows them to stay organized and focused on their teaching responsibilities. It's just a simple view courses page.

- **Grade Student**: Instructors can assign grades to students based on their performance in assignments, exams, and projects. This functionality facilitates accurate and efficient grading. By first selecting the course the row you want to choose it is clickable afterwards the students table keeps and grades analysis tables keep updating based on course selection.



Grade Analysis

Criteria	Value
Average	90.0
Median	90.0
Highest	100.0
Lowest	80.0

Courses

Viewing Course ID: 1992311

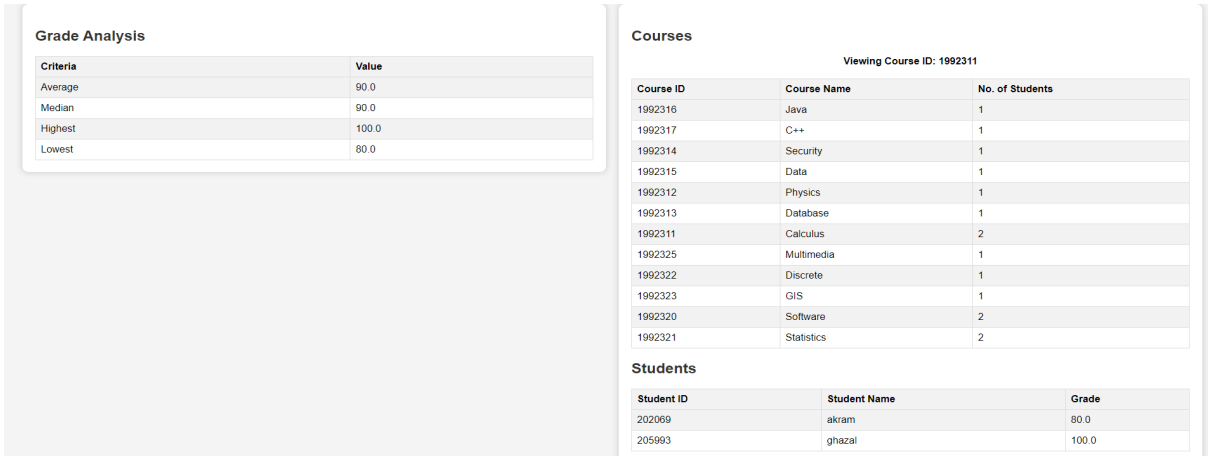
Course ID	Course Name	No. of Students
1992316	Java	1
1992317	C++	1
1992314	Security	1
1992315	Data	1
1992312	Physics	1
1992313	Database	1
1992311	Calculus	2
1992325	Multimedia	1
1992322	Discrete	1
1992323	GIS	1
1992320	Software	2
1992321	Statistics	2

Students

Student ID	Student Name	Grade
202069	akram	80.0
205993	ghazal	100.0

Figure 14: Grade Student Page

- **Grade Analysis**: Grade Analysis: Instructors can analyze student performance by calculating averages, medians, and other statistics. This helps them understand how students are performing. Similar to Grade Student functionality, the analysis page dynamically updates the students and grade analysis tables based on the selected course.



Grade Analysis

Criteria	Value
Average	90.0
Median	90.0
Highest	100.0
Lowest	80.0

Courses

Viewing Course ID: 1992311

Course ID	Course Name	No. of Students
1992316	Java	1
1992317	C++	1
1992314	Security	1
1992315	Data	1
1992312	Physics	1
1992313	Database	1
1992311	Calculus	2
1992325	Multimedia	1
1992322	Discrete	1
1992323	GIS	1
1992320	Software	2
1992321	Statistics	2

Students

Student ID	Student Name	Grade
202069	akram	80.0
205993	ghazal	100.0

Figure 15: Grade Analysis Page

- **Course Enrollment** : Instructors can enroll in or assign courses to themselves, allowing them to begin grading and teaching students. This feature ensures instructors have the necessary access to manage and oversee the courses they are responsible for.

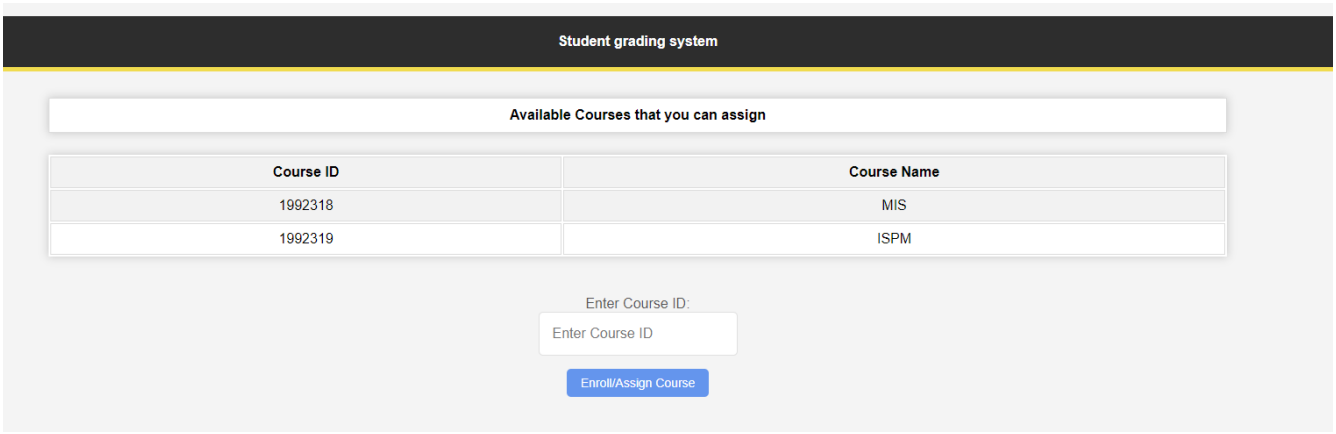


Figure 16: Course Enrollment Page

4.3 Students:

- **View Courses**: Students can see a list of available courses. This helps them make informed decisions about which courses to enroll in will not also provide anything as it is simple.

- **View Grades**: Students can access their grades for various assignments and exams. This allows them to monitor their progress and identify areas for improvement.

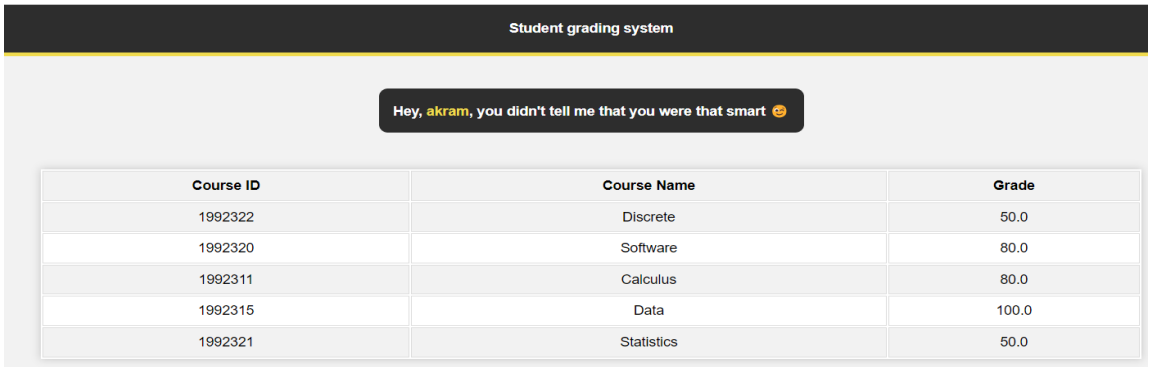


Figure 17: View Grades Page

- **Course Enrollment**: Students can enroll in courses they wish to attend. This feature streamlines the enrollment process and ensures accurate records of student registrations it performs the same operation as the instructor's one doo.

5.0 Design

5.1 System Architecture:

The Grading System employs a three-tier architecture, consisting of Presentation, Business Logic, and Data tiers. This separation of concerns enhances maintainability, scalability, and modularity.

- **Presentation Tier:**

The front-end, utilizing Thymeleaf templates for Spring and JSP for servlets, plays a crucial role in rendering dynamic HTML pages. Designed with user-friendly interfaces, the presentation tier caters to admins, instructors, and students, offering tailored experiences based on their roles. Through HTTP requests, the presentation tier interacts with the business logic tier to initiate actions and retrieve data.

- **Business Logic Tier:**

At the core of the architecture lies the business logic tier, responsible for processing user requests and orchestrating the system's interactions. By utilizing Spring MVC and MVC Servlets, this tier effectively manages controllers, routing requests, and maintaining the application's flow. Role-based access control ensures that users are granted appropriate permissions according to their roles, maintaining data security and privacy. When users perform actions, the corresponding service methods are invoked to engage with the data tier.

- **Data Tier:**

The foundation of the Grading System is the data tier, housing a MySQL database where critical information such as student profiles, instructor details, course data, and grades are stored. Leveraging the power of a relational database, this tier efficiently manages and retrieves data. It offers a structured and organized repository, ensuring data integrity and facilitating seamless interaction between various system components.

5.2 Data Model

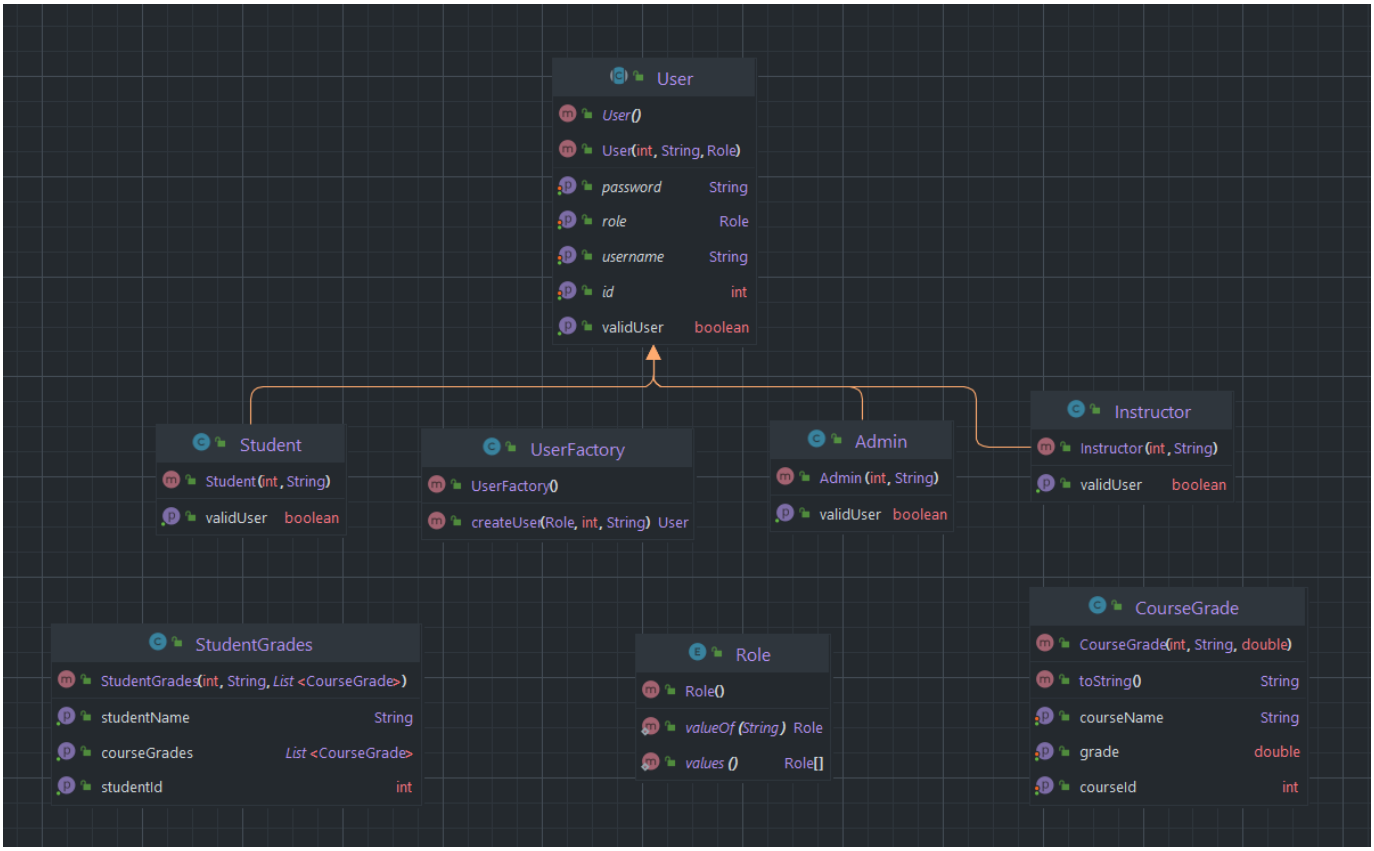


Figure 18: Data Model Diagrams

6.0 Implementation

1. User Authentication and Role-Based Access:

The foundation of system security lies in user authentication and role-based access control. By employing the User abstract class and its subclasses (Admin, Instructor, Student), the system ensures secure login by validating user credentials against the database. This approach grants different roles varying levels of access, maintaining data integrity while accommodating different user responsibilities.

As the system evolved, the authentication mechanism evolved too. Initially, individual logins were used for socket communication. However, as the system migrated to servlets and Spring, a unified login page was adopted, allowing a more streamlined and consistent authentication process across different components of the system.

```
public class Student extends User {
    1 usage
    DAO dao = new DAO();
    public Student(int id, String password) { super(id, password, Role.STUDENT); }
    1 usage
    @Override
    public boolean isValidUser() {
        try (Connection connection = dao.getDatabaseConnection()) {
            String query = "SELECT student_id FROM students WHERE student_id = ? AND password = ?";
            PreparedStatement preparedStatement = connection.prepareStatement(query);
            preparedStatement.setInt( parameterIndex: 1, id);
            preparedStatement.setString( parameterIndex: 2, password);
            ResultSet resultSet = preparedStatement.executeQuery();
            if(!resultSet.next()) {
                System.out.println("The student entered wrong credentials");
                return false;
            }
            System.out.println("The Student had Successfully Connected");
            return true;
        } catch (Exception e) {
            System.err.println("Error with the database operation: " + e.getMessage());
            return false;
        }
    }
}
```

Figure 19: imp-user-auth

2. Dynamic Front-End Web Interaction:

Dynamic front-end interfaces were employed for servlets and Spring as shown in the 4.0 section (functional programming), prioritizing user engagement. This approach generated live user interfaces, enhancing interaction compared to alternatives like Postman. Servlets utilized JSP pages, while Spring integrated Thymeleaf templates. Both strategies offered intuitive views, enabling user actions, and providing a user-friendly experience. This dynamic front-end design ensures user engagement and satisfaction.

3. **Controller Logic and Request Handling:**

In the Grading System, controllers act as guides for user interactions. They use tools like Spring MVC and MVC Servlets to direct requests and responses. This keeps the system organized, like having different teams for different tasks.

Controllers understand what users want, while tools like Spring MVC handle the technical side. It's like a conductor leading an orchestra, making sure everyone plays the right notes at the right time.

This setup ensures smooth communication between users and the system, making interactions effective and efficient.

Model:

```java
@Setter
@Getter
@NoArgsConstructor
public abstract class User {
    protected int id;
    protected String username;
    protected String password;
    protected Role role;

    public User(int id, String password, Role role) {
        this.id = id;
        this.password = password;
        this.role = role;
    }

    1 usage  3 implementations
    public abstract boolean isValidUser();
}
```

View:

```html
<body>
<div class="top-bar">Student grading system</div>
<form action="/login" method="POST">
  <div class="login-section">
    <div class="role-message">
      Please select your role:
    </div>
    <div class="role-selection">
      <div class="role-option">
        <input type="radio" id="student" name="role" value="STUDENT">
        <label for="student">Student</label>
      </div>
      <div class="role-option">
        <input type="radio" id="instructor" name="role" value="INSTRUCTOR">
        <label for="instructor">Instructor</label>
      </div>
    </div>
    <div class="login-fields">
      <div class="error-message" th:if="${errorMessage != null}">
        <span th:text="${errorMessage}"></span>
      </div>
      <label for="user_id">User ID:</label>
      <input id="user_id" name="user_id" type="text" placeholder="Enter ID" required />
      <label for="password">Password:</label>
      <input id="password" name="password" type="password" placeholder="Enter Password" required/>
      <input type="submit" value="Login"/>
```

Controller:

```java
@GetMapping("/login")
public String showLoginPage() { return "login"; }

no usages
@PostMapping("/login")
public String login(@RequestParam("user_id") String userIdStr,
                    @RequestParam("password") String password,
                    @RequestParam(value = "role", required = false) String roleStr, // Note the required = false
                    Model model) {
    Role role;
    if (roleStr == null || roleStr.isEmpty()) {
        if (dao.isAdminCredentials(userIdStr, password)) {
            role = Role.ADMIN;
        } else {
            model.addAttribute( attributeName: "errorMessage",  attributeValue: "Please choose one of the roles.");
            return "login";
        }
    } else {
        role = Role.valueOf(roleStr);
        String errorMessage = Validation.validateUserInput(role, userIdStr, password);
        if (errorMessage != null) {
            model.addAttribute( attributeName: "errorMessage", errorMessage);
            return "login";
        }
    }
```

4. The Data Access Object (DAO):

is like a helper that manages communication with the database. It keeps the database-related stuff separate from the main code, making things easier to manage. Think of it as a specialized tool that does the dirty work behind the scenes, so the rest of the code can stay neat and organized. By using DAO, we can make changes to the database without disrupting the whole system.

```java
@Service
public class DAO {
    1 usage
    private static final String HOST = "jdbc:mysql://localhost:3306/";
    2 usages
    private static final String DB_USERNAME = "root";
    2 usages
    private static final String DB_PASSWORD = "21232619boom";
    1 usage
    private static final String DB_DATABASE = "student_system";
    6 usages
    private final Map<String, List<String>> tableColumns = new HashMap<>();

    public DAO() {
        tableColumns.put("students", Arrays.asList("student_id", "username"));
        tableColumns.put("instructors", Arrays.asList("instructor_id", "username"));
        tableColumns.put("courses", Arrays.asList("course_id", "course_name"));
    }

    6 usages
    public List<String> getTableColumns(String tableName) { return tableColumns.get(tableName); }
```

5. Usage of helper methods in database operations:

My system employed two utility methods, executeUpdate and executeQuery, to streamline database interactions. These methods simplify database operations by handling connection establishment, query preparation, parameter setting, and execution. executeUpdate is used for updates like inserts and updates, while executeQuery retrieves data and returns a ResultSet. These utilities enhance code readability and maintainability by encapsulating repetitive database tasks, promoting efficient and organized code.

```
8 usages
private int executeUpdate(String query, Object... params) throws SQLException {
    try (Connection connection = getDatabaseConnection();
         PreparedStatement preparedStatement = connection.prepareStatement(query)) {
        for (int i = 0; i < params.length; i++) {
            preparedStatement.setObject( parameterIndex: i + 1, params[i]);
        }
        return preparedStatement.executeUpdate();
    }
}

8 usages
private ResultSet executeQuery(String query, Object... params) throws SQLException {
    Connection connection = getDatabaseConnection();
    PreparedStatement preparedStatement = connection.prepareStatement(query);
    for (int i = 0; i < params.length; i++) {
        preparedStatement.setObject( parameterIndex: i + 1, params[i]);
    }
    return preparedStatement.executeQuery();
}
```

6. Securing User's Data:

To ensure the security of user data, the system employs various measures, including password hashing and input validation.

The Password Hashing utility class employs the SHA-256 hashing algorithm to secure user passwords. This hashing process converts the user's password into a unique hash, making it significantly more secure than storing plain passwords. This hashing function transforms the password into a format that is unreadable, protecting the actual passwords in the database. The generated hash is stored and compared during authentication.

The Validation utility class handles the validation of user inputs, enhancing data integrity. It ensures that entered user IDs are in the correct format and that the corresponding passwords are properly hashed. The class performs the following steps:

- Validates that the user ID is in the correct 7-digit numeric format.
- Hashes the user's password using the PasswordHashing utility.
- Creates a user object based on the provided role, user ID, and hashed password.
- Verifies the user's validity by checking the user's credentials against the database.

7. 0 Analytical Thinking

1. Sockets:

Strengths:

Real-time Updates: Immediate data transmission for instant interaction.

Direct Control: Low-level data flow control for specific scenarios.

Weaknesses:

Steep Learning Curve: Complex implementation requiring meticulous programming.

Debugging Challenges: Troubleshooting socket-related issues could be time-consuming.

Scalability Concerns: Scaling socket-based systems for larger user bases posed difficulties.

2. Servlets and JSPs:

Strengths:

Improved Communication: Easier communication handling compared to sockets.

Server-side Rendering: JSPs enabled dynamic content rendering on the server.

Weaknesses:

Configuration Complexity: Setting up servlet containers could be intricate and time-consuming.

Mixing Logic and Presentation: Logic and presentation intertwined in JSPs, making maintenance harder.

Learning Curve: Servlet lifecycle and container interactions required time for mastery.

3. Spring and Thymeleaf:

Strengths:

Robust Framework: Comprehensive tools for application development.

Encapsulates logic into one class not like servlets as we had to create multiple servlets....

Streamlined Configuration: Simplified configuration for streamlined development.

Modular Components: Thymeleaf's templating system for clear separation of concerns.

Weaknesses:

Complexity of Choices: Spring's flexibility sometimes required selecting from various options.

Overkill for Simple Apps: Might be excessive for small applications with minimal requirements.

Overall Technological Choices:

From the initial use of sockets to the eventual adoption of the Spring framework, the progression reflects a strategic response to the system's increasing complexity. Challenges faced with sockets and servlets underscored their limitations, while Spring's simplicity and Thymeleaf's user-friendly templates proved effective in addressing these issues.

Strategic Adaptation:

The shift to Spring represents a well-considered move to enhance development efficiency and system scalability. This adaptation reflects a proactive approach to technology selection, aiming to align with the project's growth and development goals. The transition showcases a commitment to delivering a robust and effective solution by leveraging technologies that streamline development and improve overall project performance.

8.0 challenges and solutions

1. **Socket Implementation Complexity:**

Challenge: Implementing socket-based communication was intricate and time-consuming, leading to debugging challenges.

Solution: Meticulous programming, extensive testing, and leveraging online resources helped address socket-related complexities.

2. **Servlet Configuration Complexity:**

Challenge: Configuring servlet containers for communication required detailed understanding and time investment.

Solution: In-depth research, documentation referencing, and seeking multiple resources and YouTube videos.

3. **Maintaining Logic-Presentation Separation in JSPs:**

Challenge: In JSPs, mixing logic with presentation led to maintenance difficulties.

Solution: Implementing the Model-View-Controller

4. **Managing User Authentication:**

Challenge: Ensuring secure user authentication while maintaining user-friendly interfaces.

Solution: Implementing password hashing, role-based access control, and user input validation ensured data security and a smooth user experience.

9.0 Security Considerations

1. **Password Hashing:**

Implementation: Passwords are hashed using the SHA-256 algorithm before storing them in the database.

Rationale: Hashing passwords ensures that even if the database is compromised, the actual passwords remain hidden. Hashed passwords are extremely difficult to reverse-engineer.

2. **Role-Based Access:**

Implementation: Users are assigned roles (Admin, Instructor, Student), each with specific permissions.

Rationale: Role-based access ensures that users can only perform actions relevant to their roles. Admins can manage the system, instructors can grade students, and students can view their grades and courses.

3. **Input Validation:**

Implementation: User inputs are validated and sanitized to prevent SQL injection and other malicious attacks.

Rationale: Input validation prevents attackers from inserting harmful code through user inputs, reducing the risk of unauthorized access or data manipulation.

4. Limited Database Access:

Implementation: Database access is restricted to specific CRUD (Create, Read, Update, Delete) operations required for the system's functionality.

Rationale: Limiting database access ensures that only authorized actions are allowed, reducing the attack surface, and protecting sensitive data.

5. Secure Error Handling:

Implementation: Error messages do not reveal system details and are designed to provide minimal information to users.

Rationale: Secure error handling prevents attackers from gaining insights into the system's structure, making it harder for them to exploit vulnerabilities.

10.0 Future Enhancements

As the Grading System continues to evolve, there are several avenues for potential improvements and feature additions. Integrating advanced security measures, such as Spring Security, can significantly enhance the system's authentication and authorization mechanisms. Spring Security offers a robust toolkit for implementing role-based access, fortified authentication providers, and customizable login processes. Moreover, the system could benefit from adopting modern database interaction techniques like JPA (Java Persistence API) and JDBC Template. Integrating JPA could streamline data management through object-relational mapping, while JDBC Template simplifies raw JDBC operations, optimizing database interactions. These enhancements lay the groundwork for advanced capabilities, such as data visualization tools that present grade analytics in a more comprehensible manner. Additionally, real-time notifications or alerts could enhance user experience, keeping students informed about grade updates and instructors apprised of new assignments or course enrollment changes.