Student Grading System Report

Mohannad Al-Tamimi

mohanad.tamimi81@gmail.com

Atypon May 2024

Introduction

This report presents a comprehensive overview of the development of the Student Grading System, which was implemented in three progressive stages:

1. Command-line Application with Sockets and JDBC Backend
2. Web Application using Traditional MVC with Servlets and JSPs
3. Web Application using Spring MVC and Spring REST

Each stage aimed to enhance the functionality, scalability, and user experience of the system. This report details the system's architecture, data model, interaction flow, implementation nuances, critical analysis of technologies used, challenges faced, security considerations, and future enhancements.

1. Design

1.1 System Architecture

1.1.1 Overview

The Student Grading System's architecture evolved with each implementation stage to accommodate increased functionality and improve user experience.

- Stage 1: A client-server architecture using sockets for communication.

- Stage 2: A web-based architecture using Servlets and JSPs following the MVC pattern.

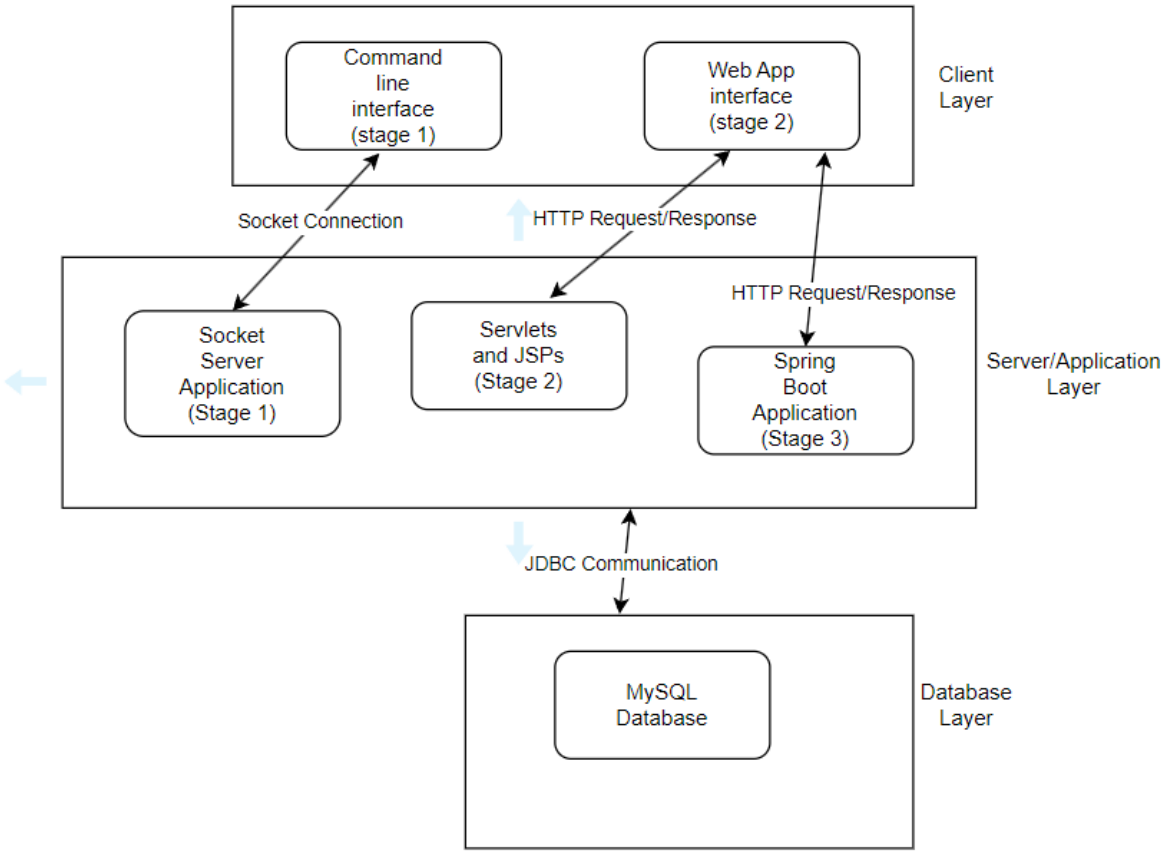
- Stage 3: A modern web application using Spring MVC and Spring REST, adhering to RESTful principles.

1.1.2 Components

- Client Interface: Command-line interface (Stage 1) and web interface (Stages 2 & 3).

- Server/Application Layer: Handles business logic, user authentication, and database interactions.

- Database Layer: MySQL database storing student records, courses, grades, and user credentials.



1.2 Data Model

1.2.1 Entity Relationships

The system's data model consists of four primary entities:

- Student
- Course
- Grade
- UserCredentials

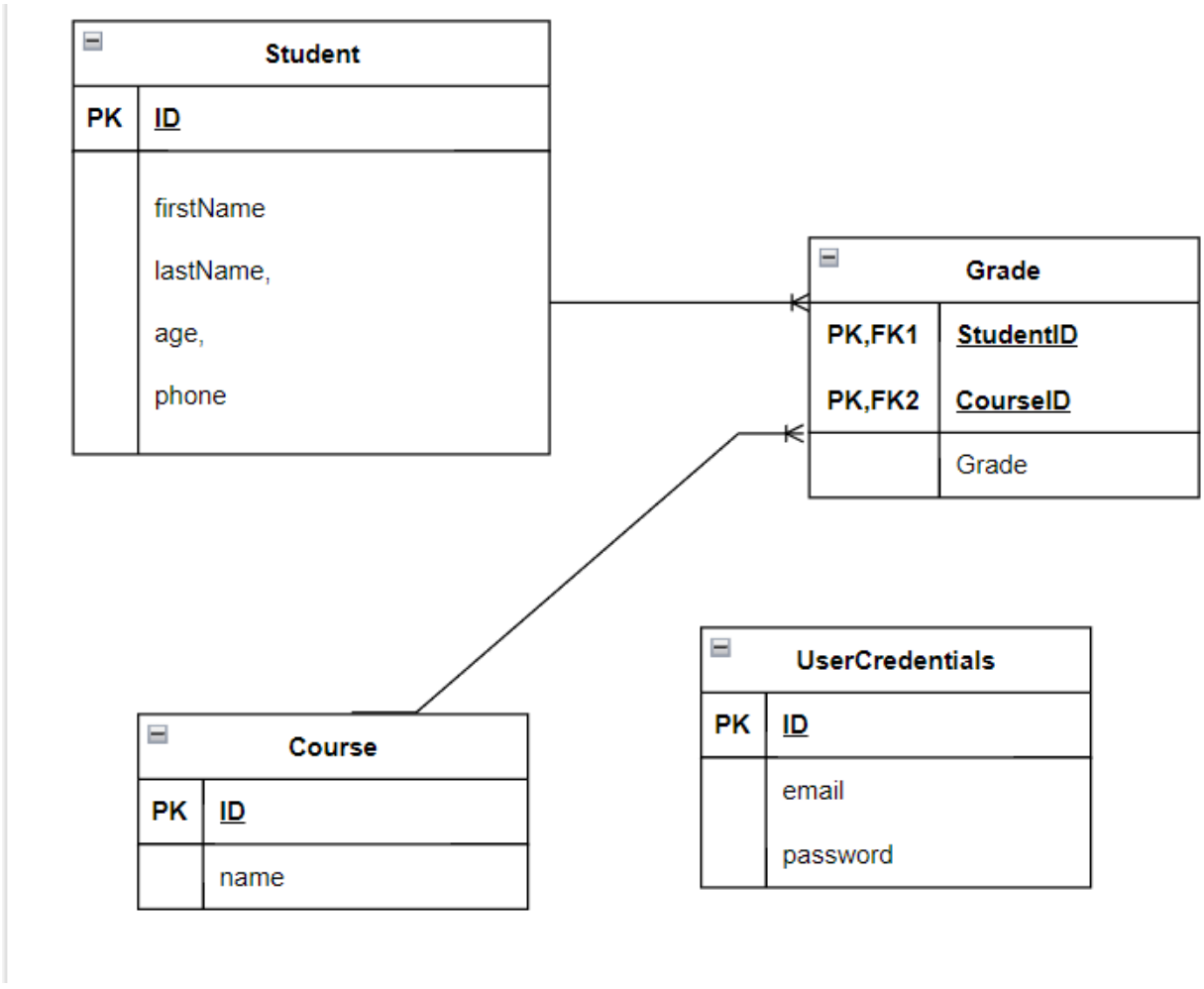
These entities are interconnected as follows:

- A Student can enroll in multiple Courses.

- A Course can have multiple Students.

- The Grade entity represents the many-to-many relationship between Students and Courses, including the grade awarded.

- UserCredentials store login information for academic staff and, in future iterations, students.



1.3 Interaction Flow Between Components

1.3.1 Stage 1: Command-line Interaction

- Client-Server Communication: Clients connect to the server via sockets.
- Authentication: Academic staff authenticate using credentials.
- Operations:
  - Add new students and courses.
  - Assign grades.
  - View students, courses, and grades.

1.3.2 Stage 2: Web Interaction with Servlets and JSPs

- Client Requests: Users interact via web forms.
- Controller Servlets: Handle requests and responses.
- JSP Views: Render dynamic content.
- Session Management: Maintains user sessions after authentication.

1.3.3 Stage 3: Spring MVC and RESTful Services

- Spring Controllers: Manage web requests using annotations.
- Thymeleaf Templates: Serve as the view layer for rendering HTML.

- RESTful Endpoints: Expose APIs for data retrieval and manipulation.

- Dependency Injection: Utilizes Spring's IoC container for managing components.

2. Implementation

2.1 Key Functions and Algorithms

2.1.1 Data Access Object (DAO)

- Purpose: Abstracts database interactions.

- Functions:

  - `addStudent(Student student)`

  - `addCourse(Course course)`

  - `addGrade(Long studentId, Long courseId, int grade)`

  - `allStudents()`, `allCourses()`

  - `getStudentById(Long id)`, `getCourseById(Long id)`

  - `getFinishedCoursesForStudent(Long studentId)`

  - `getStudentsInCourse(Long courseId)`

  - `courseAvg(Long courseId)`

```
public void addStudent(Student student) {  1 usage
    try (PreparedStatement ps = conn.prepareStatement(INSERT_STUDENT)) {
        ps.setString( parameterIndex: 1, student.getFirstName());
        ps.setString( parameterIndex: 2, student.getLastName());
        ps.setInt( parameterIndex: 3, student.getAge());
        ps.setString( parameterIndex: 4, String.valueOf(student.getPhone()));
        ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

2.1.2 Authentication Service

- Function: Validates user credentials against the database.

- Implementation: Compares input credentials with stored credentials in `AcademicStaffCredentials`.

```
public static boolean authenticate(UserCredentials userCredentials) {  1 usage
    // Returning whether the user is one of the academic staff members.
    List<UserCredentials> academicStaff = dao.allAcademicStaff();
    for (UserCredentials currentCredentials : academicStaff) {
        if (currentCredentials.getEmail().equals(userCredentials.getEmail()) &&
                currentCredentials.getPassword().equals(userCredentials.getPassword())) {
            return true;
        }
    }
    return false;
}
```

2.1.3 Controllers

- Servlets (Stage 2): Handle HTTP requests and forward responses to JSPs.

- Spring Controllers (Stage 3): Use annotations like `@Controller`, `@GetMapping`, and `@PostMapping` to map URLs to methods.

```
@PostMapping("done-adding-new-student")  no usages
public String doneAddingNewStudent(@RequestParam String firstName,
                                   @RequestParam String lastName,
                                   @RequestParam int age,
                                   @RequestParam long phone) {
    dao.addStudent(new Student(firstName, lastName, age, phone));
    return "add-new-student";
}
```

2.2 Design Decisions

- Separation of Concerns: DAO for database logic, services for business logic, controllers for handling HTTP requests.

- Use of Prepared Statements: Prevents SQL injection attacks.

- Transition to Spring Framework: For better scalability and ease of development.

3. Analytical Thinking

3.1 Strengths and Weaknesses of Each Implementation Stage

3.1.1 Stage 1: Command-line with Sockets and JDBC

- Strengths:
  - Simple and straightforward for initial development.
  - Direct control over client-server communication.
- Weaknesses:
  - Limited user interface; not user-friendly.
  - Scalability issues with socket management.
  - Difficult to maintain and extend.

3.1.2 Stage 2: Web App with Servlets and JSPs

- Strengths:
  - Improved user experience with web interface.
  - Follows MVC pattern, enhancing maintainability.
- Weaknesses:
  - Manual handling of boilerplate code.

- Less efficient session management.
- JSPs can become cluttered with Java code.

3.1.3 Stage 3: Web App with Spring MVC and REST

- Strengths:
  - Streamlined development with Spring Boot.
  - Enhanced scalability and maintainability.
  - RESTful APIs enable easy integration with other services.
  - Dependency Injection reduces tight coupling.
- Weaknesses:
  - Steeper learning curve for Spring Framework.
  - Initial setup complexity.

3.2 Technology Choices Justification

- Sockets to Web Transition: Moving from sockets to web applications meets modern user expectations and accessibility.
- Spring Framework Adoption: Provides robust features like IoC, AOP, and RESTful web services, reducing boilerplate code and improving efficiency.
- Thymeleaf over JSPs: Offers better integration with Spring and cleaner HTML templates.

4. Challenges and Solutions

4.1 Database Connection Issues

- Challenge: Encountered `SQLSyntaxErrorException` due to incorrect database naming and driver deprecation warnings.

- Solution: Updated JDBC driver class to `com.mysql.cj.jdbc.Driver`, ensured database names matched case-sensitively, and included proper connection parameters.

4.2 Managing Dependencies

- Challenge: Managing and updating dependencies in `pom.xml`, dealing with deprecated or conflicting libraries.

- Solution: Regularly reviewed and updated dependencies, used Maven scopes appropriately, and removed unused libraries.

4.3 Session Management

- Challenge: Maintaining user sessions and authentication state across requests.

- Solution: Implemented session handling using `HttpSession` in Servlets and leveraged Spring Security for robust session management in Spring Boot.

4.4 Learning Curve with Spring Framework

- Challenge: Steep learning curve when transitioning to Spring MVC and REST.

- Solution: Invested time in studying Spring documentation and tutorials, and incrementally refactored the application to adopt Spring features.

5. Security Considerations

5.1 Authentication and Authorization

- Implementation: User credentials are verified against stored passwords.

- Rationale: Protects user data and prevents unauthorized access.

5.3 Input Validation

- Implementation: All user inputs are validated both on the client side (using HTML5 validation) and server side.

- Rationale: Prevents SQL injection, XSS attacks, and ensures data integrity.

5.4 Use of Prepared Statements

- Implementation: Database queries use `PreparedStatement` to handle user inputs safely.

- Rationale: Safeguards against SQL injection attacks by treating inputs as parameters.

5.5 Session Management

- Implementation: Sessions are managed securely, and sensitive data is not exposed in URLs or client-side scripts.

- Rationale: Ensures session hijacking is mitigated and user data remains secure.

6. Future Enhancements

6.1 Implementing Student Login and Profiles

- Description: Allow students to have individual accounts to log in and view their own grades and profiles.

- Benefits: Enhances user experience and data privacy.

6.2 Role-Based Access Control

- Description: Introduce roles (e.g., Admin, Instructor, Student) with specific permissions.

- Benefits: Improves security and allows for more granular control over system functionalities.

6.3 Enhanced User Interface

- Description: Improve the front-end using modern JavaScript frameworks (e.g., React or Angular) and responsive design.

- Benefits: Provides a better user experience across different devices.

6.4 API Documentation and Testing

- Description: Utilize tools like Swagger for API documentation and JUnit for unit testing.

- Benefits: Improves maintainability and facilitates integration with other systems.

6.5 Implementing Notifications

- Description: Send email or SMS notifications to students about grade updates or announcements.

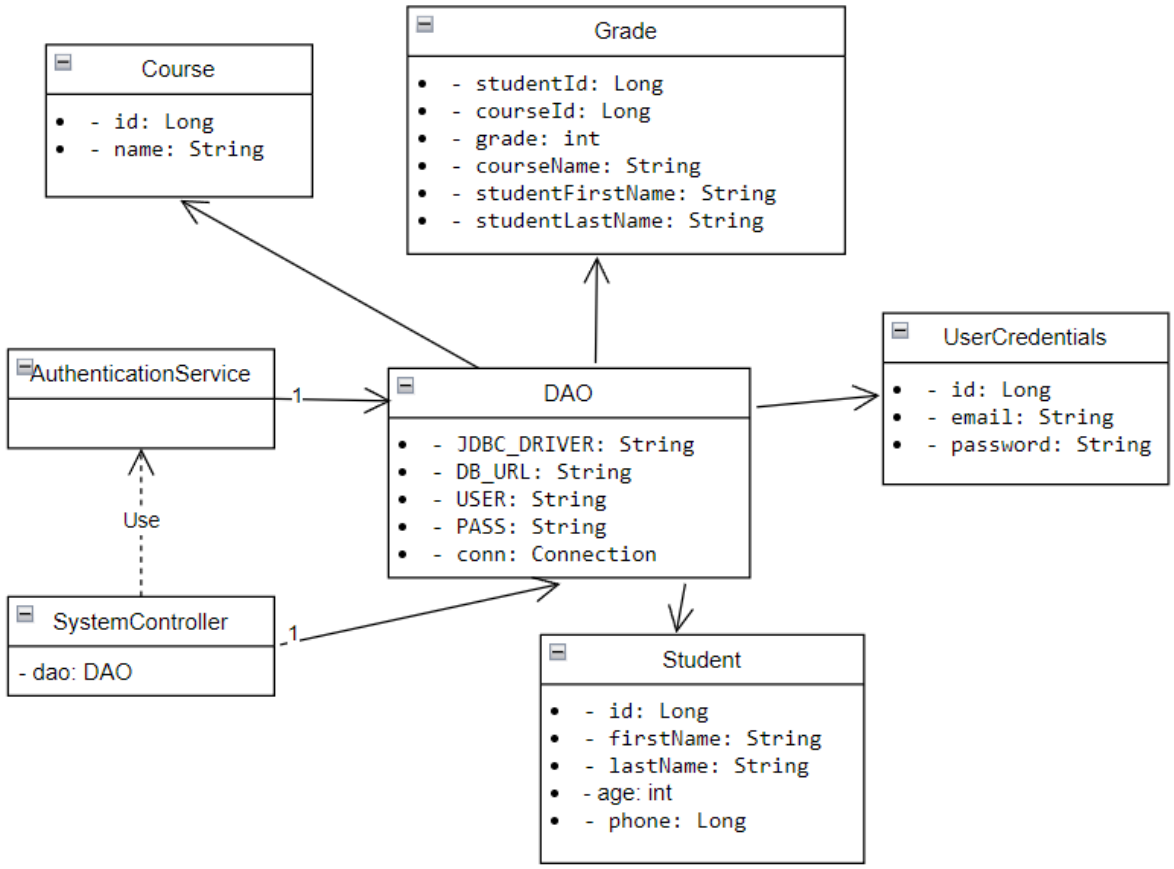
- Benefits: Keeps users engaged and informed.

7. Conclusion

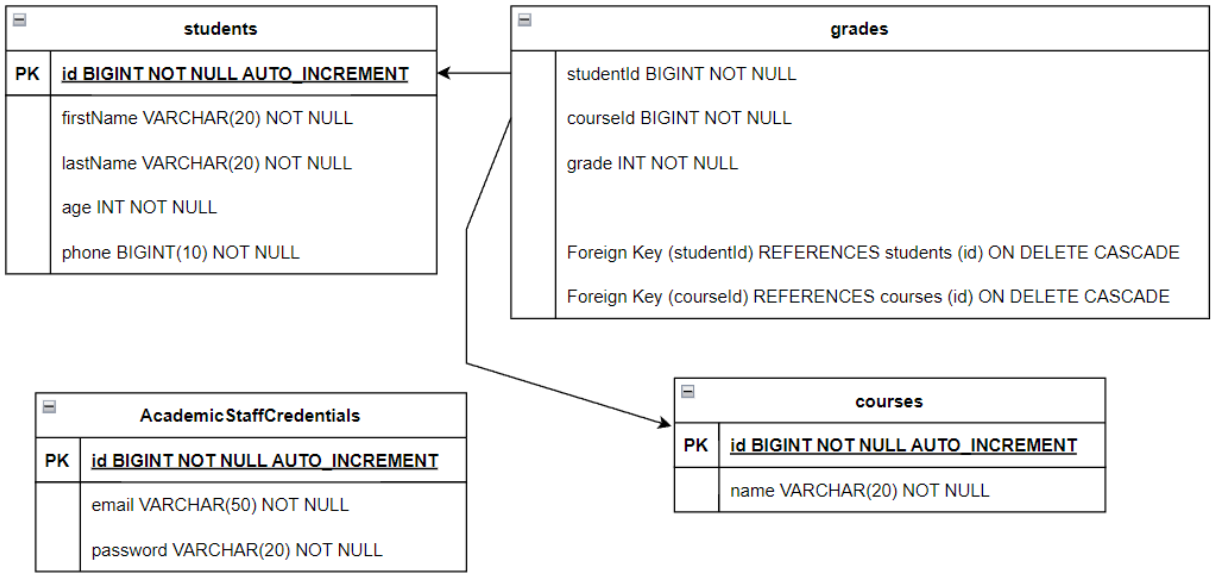
The development of the Student Grading System showcased the evolution of a software application from a simple command-line tool to a robust, web-based platform utilizing modern frameworks. Each stage brought improvements in usability, scalability, and maintainability. While challenges were encountered, they provided learning opportunities that strengthened the final product. Security considerations were paramount throughout the development process, ensuring user data remains protected. Future enhancements aim to further refine the system, keeping it aligned with user needs and technological advancements.

Appendices

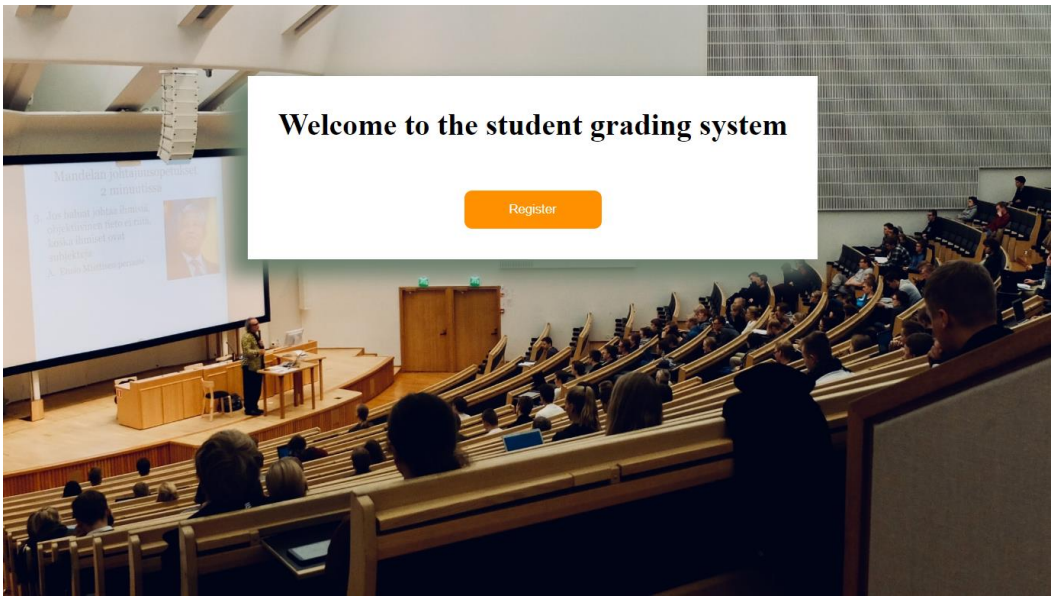
Appendix A: Class Diagram

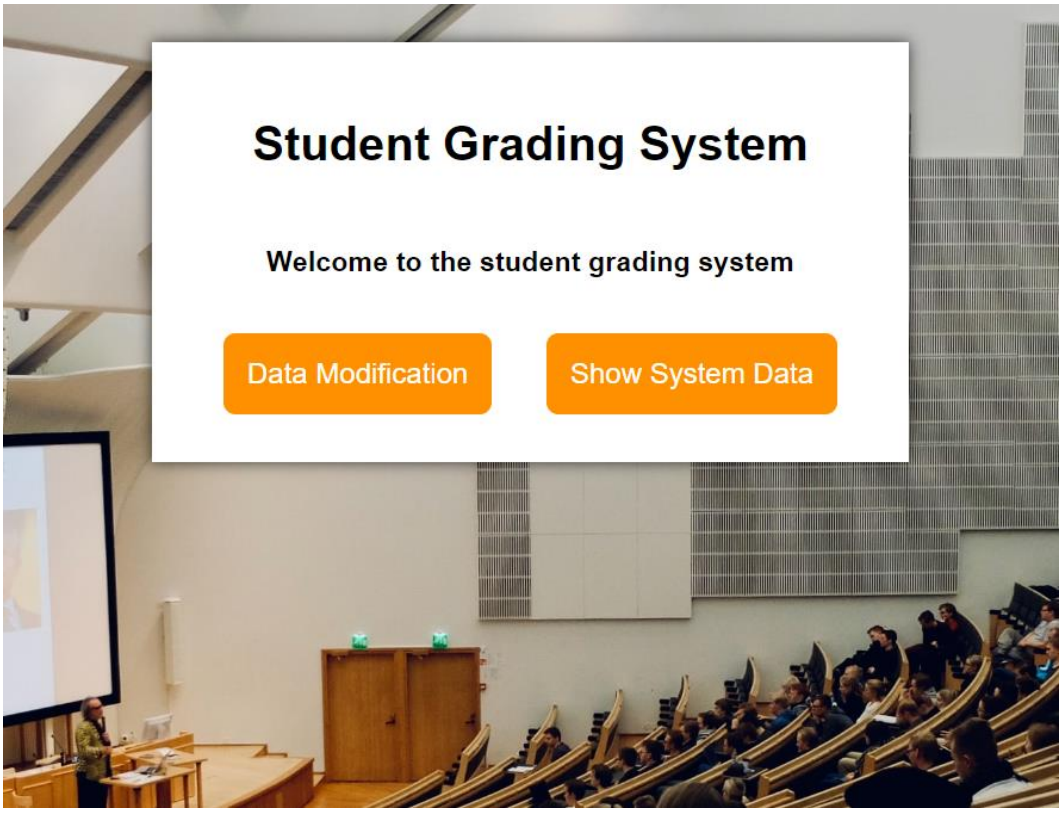


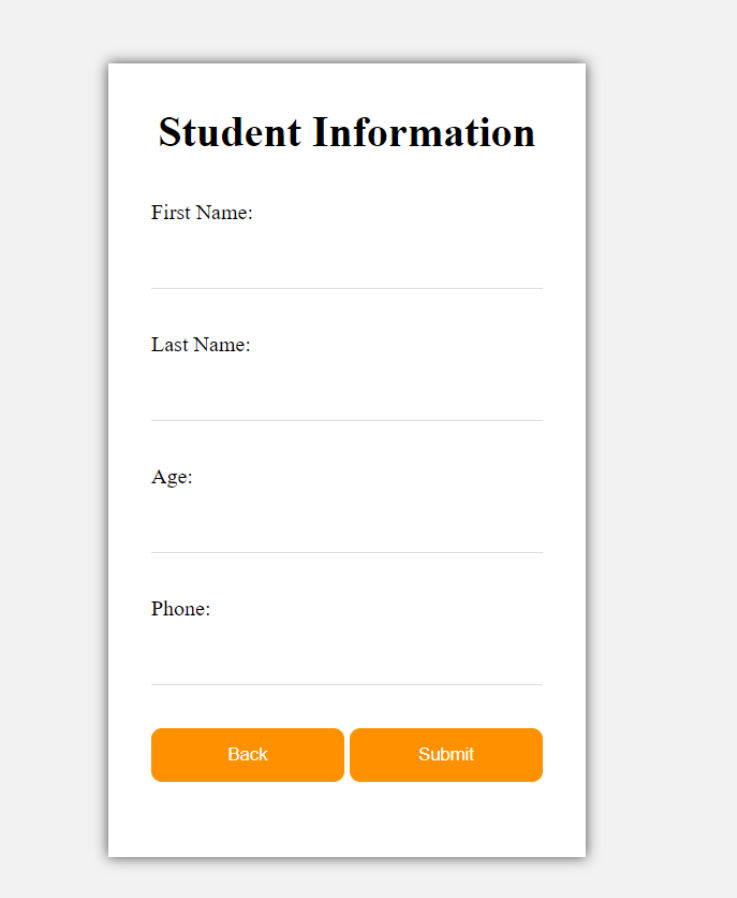
Appendix B: Database Schema



Appendix C: User Interface Screenshots






Student Information
First Name:
Last Name:
Age:
Phone:
Back
Submit

All Students

Student ID	First Name	Last Name	Age	Phone
1	John	Doe	20	5551234567
2	Jane	Doe	19	5559876543
3	Bob	Smith	21	5555551212
4	Alice	Johnson	18	5555551234
5	Mike	Jones	22	5557778888
6	Emily	Davis	20	5555554321
7	Alex	Garcia	19	5559990000
8	Maria	Rodriguez	21	5551234567
9	David	Wilson	18	5555557890
10	Sarah	Lee	22	5554443333
11	John	Doe	20	5551234567
12	Jane	Doe	19	5559876543
13	Bob	Smith	21	5555551212
14	Alice	Johnson	18	5555551234
15	Mike	Jones	22	5557778888
16	Emily	Davis	20	5555554321