# Student Grading System

**Done By:**
Lana Alnimreen
**Supervisor**:
Dr.Motasem Aldiab

# Contents

# Abstract

The Student Grading System assignment aims to assess students' proficiency in backend development, web application frameworks, and full-stack implementation through a comprehensive project that progresses in three stages. The system is developed using command-line interfaces, MVC Servlets, JSPs, and Spring frameworks to manage student grades securely and efficiently. This project not only demonstrates the student’s ability to work with multiple technologies but also showcases key concepts such as user authentication, session management and web application design. Through a series of progressive implementations, the system evolves into a fully functional, secure, and scalable web application.

# Introduction

This report offers a comprehensive analysis for the Student Grading System that progresses through three stages of implementation. A multi-stage project that transforms a simple command-line application into a sophisticated web-based platform utilizing modern Java technologies. This documentation will guide you through the three key stages of this assignment, each representing a significant step towards creating a more accessible, user-friendly, and efficient system for managing student grades.

# Database

The database schema used in my web application models the relationships between students, courses, and their enrollment details, including student information, course information, and student performance in each course. It's designed to support the functionality of a Grading System application by storing and managing data related to students' enrollment and performance in various courses.

1. **Student Table**:
   This table stores information about students who use the Grading System.
   **Columns**:
   - `Student_id`: A unique identifier for each student. This is the primary key of the table.
   - `First_name`: The name of the student (varchar, max length 50).
   - `Password`: The password associated with the student's account (varchar, max length 50).
2. **Course Table**:
   This table stores information about the courses offered in the Grading System.
   **Columns**:
   - `Course_ID`: A unique identifier for each course. This is the primary key of the table.
   - `Course_Name`: The name of the course (varchar, max length 100).
3. **Grade Table**:
   This table establishes a many-to-many relationship between students and courses, indicating which student is enrolled in which course and their performance in the course.
   **Columns**:
   - `grade_ID`: A unique identifier for each grade. This is the primary key of the table.
   - `Student_id`: A foreign key reference to the `Student` table, indicating which student is enrolled in the course.
   - `Course_id`: A foreign key reference to the `Course` table, indicating which course the student is enrolled in.
   - `grade`: The student's mark or score in the course (int).

   **Constraints**:
   - The combination of `Student_ID` and `Course_ID` forms a unique constraint, ensuring that a student cannot be enrolled in the same course more than once.
   - Foreign key constraints ensure referential integrity by linking the `Student_ID` and `Course_ID` columns to their respective tables.

# Part 1

This part demonstrates a simple student login and grade retrieval system using MySQL for database storage and sockets for client-server communication. My project is divided into multiple packages:

- **`com.example.cli`**: Contains command-line-based logic for handling student login and viewing grades.
- **`com.example.database`**: Manages database connectivity with the MySQL database.
- **`com.example.socket`**: Implements socket-based communication between the client and server for grade retrieval and login.

1. **`cli.StudentLogin`** Class

   This class manages student login by interacting with the `Students` table in the database. It performs the following tasks:

   **`login(String firstName, String password)`**:

   - Connects to the database using `DatabaseConnection`.
   - Executes a SQL query to retrieve the password for the student with the provided `firstName`.
   - Compares the stored password with the provided one to authenticate the user.

   **`getStudentId(String firstName)`**:

   - Retrieves the `student_id` for the given student by executing a SQL query against the `Students` table.

2. **`cli.StudentGrades`** Class

   This class is responsible for retrieving and displaying student grades from the database. It interacts with the `Grades` and `Courses` tables in the database.

   **`getGradesForSocket(int studentId)`**:

   - Similar to `viewGrades()`, but instead of printing the grades, it returns them as a formatted string. This method is used by the server to send grades to the client over a socket connection.

3. **`database.DatabaseConnection`** Class

   This class manages the connection to the MySQL database.

   **`getConnection()`**:

   - Establishes a connection to the MySQL database using JDBC.

4. **socket.GradeClient** Class

   This class implements the client-side logic for connecting to the server, sending login credentials, and receiving grades.

   **`main(String[] args)`**:

   - Establishes a socket connection to the server (`localhost:5050`).
   - Sends login credentials (`firstName` and `password`) to the server.
   - Reads the server's response to check if the login was successful.
   - If login succeeds, it reads and prints the student's grades sent by the server.

5. **socket.GradeServer** Class

   This class is responsible for handling client requests over a socket connection. It processes login requests and, upon successful login, sends the student's grades to the client.

   **`main(String[] args)`**:

   - Listens for client connections on port `5050`.
   - Upon client connection, it reads the command (`LOGIN` in this case).
   - Handles the login request by calling `StudentLogin.login()` and sends a success or failure response.
   - If login is successful, retrieves the `studentId` using `StudentLogin.getStudentId()` and sends the grades to the client using `StudentGrades.getGradesForSocket()`.
   - Closes the socket connection after processing each client.

# Part 2

In this part we augment a simple Student Grading System implemented using Java Servlets and JSPs, and it interacts with a MySQL database to manage and display student grades.

### 1. Servlets

- **`LoginServlet`**:
  - Handles user login by verifying the provided credentials against the database.
  - If authentication is successful, it sets the `student_id` in the session and redirects the user to the `grades` page.
  - If authentication fails, it sets an error message and forwards the request back to `login.jsp`.
- **`GradesServlet`**:
  - Retrieves the grades for the authenticated student from the database.
  - Checks if the session contains a valid `student_id`. If not, it redirects to `login.jsp`.
  - If valid, it fetches the student's grades, sets them as a request attribute, and forwards the request to `grades.jsp`.

### 2. Web.xml

The web.xml file is a core part of Java EE web applications when using traditional MVC architecture. It provides configuration for servlets, URL mappings, and other components like filters or listeners.

### 3. JSP Pages

- **`login.jsp`**:
  - This page serves as the login form where students can enter their `first_name` and `password` to authenticate themselves.
  - The form submits the credentials to the `LoginServlet` via a POST request.
  - If there is an error, a message is displayed in red.
- **`welcome.jsp`**:
  - This page welcomes users to the Student Grading System and provides a link to the login page.
- **`grades.jsp`**:
  - This page displays the grades of the authenticated student.
  - It uses JSTL (`<c:forEach>` and `<c:if>`) to iterate over the grades and display them in a table.
  - If no grades are available, a message indicating "No grades available" is shown.

# Part 3

This part represents a Student Grading System built using Spring Boot with MVC architecture, Thymeleaf templates for views, and a MySQL database for storing data.

**1. StudentController**

- The `StudentController` class handles user login and retrieving the grades for authenticated students.

**Login Logic**

- The /`login` endpoint is mapped to the `login`() method, where the student enters their first name and password.
- The method interacts with the `StudentService` to check if a student with the given first name exists.
- If authentication is successful, the student ID is stored in the session, and the user is redirected to view their grades.
- If the login fails, an error message is shown.

**Grades Retrieval**

- The /`grades` endpoint is mapped to the `getGrades`() method.
- This method fetches the grades for the authenticated student using their `studentId` from the session and returns a list of grades.
- The grades.html template displays the student's courses and grades in a table.

**2. Models**

My project contains three model classes: Student, Grade, and Course, which represent the entities in the database.

**3. Repositories**

The repository interfaces extend JpaRepository, allowing access to CRUD operations without writing custom SQL queries.

**4. Services**

The service layer handles the business logic and interacts with the repository layer.

**5. Thymeleaf Views**

**Login Page**

- The login form allows users to input their `firstName` and `password`. If authentication fails, an error message is displayed.

**Grades Page**

- The grades page shows the courses and grades for the authenticated student in a table format. The data is dynamically populated using Thymeleaf's `th:each` attribute.

# Conclusion

The combination of these three parts are challenges and contributes to a deeper understanding of core backend technologies, web application development principles, and enterprise-level frameworks. The project helps me to gain both technical and architectural skills.