Student Grading System Assignment

RAWAN AFANEH

Part 1: Command-line / Sockets and JDBC Backend Implementa&on

Task:

Build the ini8al version of the Student Grading System using the command-line interface,

sockets, and JDBC in the backend.

```
import java.sql.*;
import java.util.ArrayList;

public class Database {
    private static Connection connection = null;
    private static ResultSet resultSet = null;
    private static PreparedStatement preparedStatement = null;

    private Database() {
    }

    public static void initializeConnection() {
        if (connection == null) {
            try {
                Class.forName("com.mysql.cj.jdbc.Driver");
                connection = DriverManager.getConnection("jdbc:mysql://localhost/grading_system", "root", "Rawan");
            } catch (SQLException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
    }

    public static boolean isUser(String username, String password, String role) {
        initializeConnection();
        try {
            String tableName = (role.equals("student")) ? "students" : "instructors";
            String sqlStatement = "SELECT * FROM " + tableName + " WHERE LOWER(username) = LOWER(?) AND password = ?";

            preparedStatement = connection.prepareStatement(sqlStatement);
            preparedStatement.setString(1, username);
            preparedStatement.setString(2, password);
            resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {
                return true;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }



    public static void updateGrade(String username, String courseName, float newGrade) {
        initializeConnection();
        try {
            String updateQuery = "UPDATE grades SET grade = ? WHERE username = ? AND course_name = ?";
            preparedStatement = connection.prepareStatement(updateQuery);
            preparedStatement.setFloat(1, newGrade);
            preparedStatement.setString(2, username);
            preparedStatement.setString(3, courseName);
            int rowsAffected = preparedStatement.executeUpdate();

            if (rowsAffected == 0) {
                System.out.println("No matching row found for the given student and course.");
            } else {
                System.out.println("Grade updated successfully.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            // Handle the exception and provide a meaningful error message to the user
            System.err.println("An error occurred while updating the grade.");
        }
    }


    public static ArrayList<String> getCoursesAndMarks(String username) {
        initializeConnection();
        ArrayList<String> courses = new ArrayList<>();

        try {
            String sql_statement =
                    "SELECT c.course_name, g.grade " +
                            "FROM students s " +
                            "JOIN students_courses sc ON s.student_id = sc.student_id " +
                            "JOIN courses c ON sc.course_id = c.course_id " +
                            "LEFT JOIN grades g ON s.student_id = g.student_id AND c.course_id = g.course_id " +
                            "WHERE LOWER(s.username) = LOWER(?)";  // Use the username column here
            preparedStatement = connection.prepareStatement(sql_statement);
            preparedStatement.setString(1, username);
            resultSet = preparedStatement.executeQuery();

            while (resultSet.next()) {
                String courseName = resultSet.getString("course_name");
                float grade = resultSet.getFloat("grade");
                String courseInfo = courseName + " " + (grade != 0 ? grade : "N/A");
                courses.add(courseInfo);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return courses;
    }

    public static ArrayList<String> getStudentGradesForCourse(String courseName) {
        initializeConnection();
        ArrayList<String> studentGrades = new ArrayList<>();

        try {
            String sql_statement =
                    "SELECT s.username, g.grade " +
                            "FROM students s " +
                            "JOIN students_courses sc ON s.student_id = sc.student_id " +
                            "JOIN courses c ON sc.course_id = c.course_id " +
                            "LEFT JOIN grades g ON s.student_id = g.student_id AND c.course_id = g.course_id " +
                            "WHERE LOWER(c.course_name) = LOWER(?)";

            preparedStatement = connection.prepareStatement(sql_statement);
            preparedStatement.setString(1, courseName);
            resultSet = preparedStatement.executeQuery();

            while (resultSet.next()) {
                String username = resultSet.getString("username");
                float grade = resultSet.getFloat("grade");
                String gradeInfo = username + " " + (grade != 0 ? grade : "N/A");
                studentGrades.add(gradeInfo);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return studentGrades;
    }
    public static ArrayList<String> getInstructorCourses(String instructorUsername) {
        initializeConnection();
        ArrayList<String> instructorCourses = new ArrayList<>();

        try {
            String sql_statement =
                    "SELECT c.course_name " +
                            "FROM instructors i " +
                            "JOIN instructors_courses ic ON i.instructor_id = ic.instructor_id " +
                            "JOIN courses c ON ic.course_id = c.course_id " +
                            "WHERE LOWER(i.username) = LOWER(?)";

            preparedStatement = connection.prepareStatement(sql_statement);
            preparedStatement.setString(1, instructorUsername);
            resultSet = preparedStatement.executeQuery();

            while (resultSet.next()) {
                String courseName = resultSet.getString("course_name");
                instructorCourses.add(courseName);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return instructorCourses;
    }
```

Server class :

```
import java.io.*;
import java.net.*;


public class Server {
    public static void main(String[] args) {
        int port = 8000;

        try {

            ServerSocket serverSocket = new ServerSocket(port);
            System.out.println("Server listening on port " + port);

            while (true) {
                Socket socket = serverSocket.accept();
                new ClientHandler(socket).start();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}

class ClientHandler extends Thread {
    private Socket socket;
    private DataInputStream in;
    private DataOutputStream out;

    public ClientHandler(Socket socket) {
        this.socket = socket;
        try {

            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public void run() {
        try {

            String role = in.readUTF();
            String username = in.readUTF();
            String password = in.readUTF();

            boolean isUser = Database.isUser(username.toLowerCase(), password, role.toLowerCase());

            out.writeBoolean(isUser);

            socket.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

User class:

```
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
public class User extends Application {



    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Login Application");

        Label roleLabel = new Label("Are you a student or instructor?");
        TextField roleField = new TextField();

        Label usernameLabel = new Label("Username:");
        TextField usernameField = new TextField();

        Label passwordLabel = new Label("Password:");
        PasswordField passwordField = new PasswordField();

        Button loginButton = new Button("Login");
        loginButton.setOnAction(e -> onLogin(roleField.getText(), usernameField.getText(), passwordField.getText()));
        VBox layout = new VBox(10);
        layout.setPadding(new Insets(20));
        layout.getChildren().addAll(roleLabel, roleField, usernameLabel, usernameField, passwordLabel, passwordField,
loginButton);

        Scene scene = new Scene(layout, 500, 500);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void onLogin(String role, String username, String password) {
        try {
            Socket socket = new Socket("localhost", 8000);
            DataInputStream in = new DataInputStream(socket.getInputStream());
            DataOutputStream out = new DataOutputStream(socket.getOutputStream());
            out.writeUTF(role);
            out.writeUTF(username);
            out.writeUTF(password);
            boolean isUser = in.readBoolean();
            if (isUser) {
                showMainApplicationStage(username, role);
            } else {
                Alert alert = new Alert(AlertType.ERROR);
                alert.setTitle("Login Error");
                alert.setHeaderText(null);
                alert.setContentText("Invalid username or password or the role.");
                alert.showAndWait();
            }
            socket.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

        private void showMainApplicationStage(String username, String role) {
            Stage mainStage = new Stage();
            mainStage.setTitle("Main Application");

            Label welcomeLabel = new Label("Welcome!");

            VBox mainLayout = new VBox(10);
            mainLayout.getChildren().add(welcomeLabel);

            if (role.equalsIgnoreCase("student")) {
                Button seeGradeButton = new Button("See Grade for Specific Course");
                seeGradeButton.setOnAction(e -> Courses(username));
                mainLayout.getChildren().addAll(seeGradeButton);
            } else if (role.equalsIgnoreCase("instructor")) {
                Button updateGradeButton = new Button("Courses");
                updateGradeButton.setOnAction(e -> handleUpdateGradeButton(username));
                mainLayout.getChildren().addAll(updateGradeButton);
            }
            Scene mainScene = new Scene(mainLayout, 500, 500);
            mainStage.setScene(mainScene);
            mainStage.show();
        }
    private void Courses(String username) {
        ArrayList<String> coursesAndMarks = Database.getCoursesAndMarks(username);

        Stage courseStage = new Stage();
        courseStage.setTitle("Courses and Grades");

        VBox courseLayout = new VBox(10);

        for (String courseInfo : coursesAndMarks) {
            String[] parts = courseInfo.split(" ");
            if (parts.length == 2) {
                String courseName = parts[0];
                String grade = parts[1];

                Button courseButton = new Button(courseName);
                courseButton.setOnAction(e -> showCourseGradeDetails(courseName, grade));

                courseLayout.getChildren().add(courseButton);
            }
        }

        Scene courseScene = new Scene(courseLayout, 500, 500);
        courseStage.setScene(courseScene);
        courseStage.show();
    }
    private void showCourseGradeDetails(String courseName, String grade) {
        Stage courseDetailStage = new Stage();
        courseDetailStage.setTitle(courseName + " Grade Details");

        Label courseLabel = new Label("Course: " + courseName);
        Label gradeLabel = new Label("Grade: " + grade);

        VBox courseDetailLayout = new VBox(10);
        courseDetailLayout.getChildren().addAll(courseLabel, gradeLabel);

        Scene courseDetailScene = new Scene(courseDetailLayout, 500, 500);
        courseDetailStage.setScene(courseDetailScene);
        courseDetailStage.show();
    }
    private void handleUpdateGradeButton(String instructorUsername) {
        Stage courseStage = new Stage();
        courseStage.setTitle("Select a Course to Update Grades");

        VBox courseLayout = new VBox(10);

        ArrayList<String> instructorCourses = Database.getInstructorCourses(instructorUsername);

        for (String courseName : instructorCourses) {
            Button courseButton = new Button(courseName);
            courseButton.setOnAction(e -> showStudentGradesForCourse(courseName));

            courseLayout.getChildren().add(courseButton);
        }

        Scene courseScene = new Scene(courseLayout, 500, 500);
```

```
        courseStage.setScene(courseScene);
        courseStage.show();
    }
    private void showStudentGradesForCourse(String courseName) {
        ArrayList<String> studentGrades = Database.getStudentGradesForCourse(courseName);

        Stage studentGradeStage = new Stage();
        studentGradeStage.setTitle("Student Grades for " + courseName);

        VBox studentGradeLayout = new VBox(10);

        for (String studentGradeInfo : studentGrades) {
            String[] parts = studentGradeInfo.split(" ");
            if (parts.length == 2) {
                String studentName = parts[0];
                String oldGrade = parts[1];

                Label studentLabel = new Label("Student: " + studentName);
                Label gradeLabel = new Label("Grade: " + oldGrade);

                TextField newGradeField = new TextField();
                newGradeField.setPromptText("Enter new grade");

                Button updateGradeButton = new Button("Update Grade");
                updateGradeButton.setOnAction(e -> {
                    String newGradeText = newGradeField.getText();
                    if (!newGradeText.isEmpty()) {
                        float newGrade;
                        try {
                            newGrade = Float.parseFloat(newGradeText);
                            if (newGrade < 0 || newGrade > 100) {
                                throw new NumberFormatException("Grade must be a decimal number between 0 and 100.");
                            }
                            Database.updateGrade(studentName, courseName, newGrade);
                            gradeLabel.setText("Grade: " + newGrade); // Update displayed grade
                        } catch (NumberFormatException ex) {
                            Alert alert = new Alert(AlertType.ERROR);
                            alert.setTitle("Input Error");
                            alert.setHeaderText(null);
                            alert.setContentText(ex.getMessage());
                            alert.showAndWait();
                        }
                    }
                });

                studentGradeLayout.getChildren().addAll(studentLabel, gradeLabel, newGradeField, updateGradeButton, new 
Separator());
            }
        }

        Scene studentGradeScene = new Scene(studentGradeLayout, 500, 500);
        studentGradeStage.setScene(studentGradeScene);
        studentGradeStage.show();
    }
    }
```

Architecture Overview

The Grading System Application is designed to efficiently manage student and instructor data, course enrollment, grades, and user authentication. It follows a client-server architecture, where clients interact with the server to access and manipulate the database. The application consists of three main components: the client-side user interface, the server-side logic, and the MySQL database.

Client-Side User Interface

The client-side interface is developed using JavaFX, providing an intuitive and user-friendly experience for both students and instructors. The interface includes login functionality and displays relevant options based on the user's role (student or instructor). Students can view their grades and course details, while instructors have the ability to update student grades and manage course information.

Server-Side Logic

The server-side logic is implemented using Java and involves two key classes: **Server** and **ClientHandler**. The **Server** class establishes a socket-based server that listens for incoming connections from clients. When a connection is accepted, a new **ClientHandler** thread is spawned to handle the client's requests concurrently. The **ClientHandler** communicates with the database and responds to client queries regarding user authentication and data retrieval.

MySQL Database

The MySQL database, named "grading_system," stores all the application's data. It comprises tables for student and instructor information, course enrollment, grades, and course details. The database schema is designed to efficiently support various types of queries required by the application.

Implementation Details

The **Database** class is the core component responsible for database access and interactions. It encapsulates methods to initialize the database connection, authenticate users, retrieve grades and course information, and update student grades. The class employs prepared statements to mitigate SQL injection vulnerabilities and uses JDBC to execute queries and retrieve results from the database.

- **User Authentication (isUser):** This method checks the user's role, username, and password against the appropriate table (students or instructors) in a case-insensitive manner to authenticate users.
- **Grade Retrieval (getgrade):** The **getgrade** method retrieves a student's grade for a specific course by employing SQL joins across the **grades**, **students**, and **courses** tables.
- **Grade Update (updateGrade):** Instructors can update student grades for specific courses using this method. It constructs and executes an SQL update query to modify the grade entry in the **grades** table.
- **Course and Grade Information Retrieval:** The **getCoursesAndMarks** method retrieves a student's enrolled courses along with their corresponding grades, and **getStudentCoursesAndMarksForInstructor** retrieves specific student course and grade details for instructors. Similarly, **getStudentGradesForCourse** fetches student grades for a given course, and **getInstructorCourses** retrieves courses taught by a specific instructor.

Analytical Thinking and Challenges

The application's design prioritizes security and data integrity by utilizing prepared statements, minimizing SQL injection risks. The use of JDBC ensures efficient and reliable database interactions. The architecture promotes concurrent handling of client requests through the multithreaded **ClientHandler** approach, enhancing the application's responsiveness.

Challenges faced during development included handling concurrent connections, ensuring thread safety, and designing efficient database queries for various data retrieval scenarios. These challenges were addressed through proper synchronization mechanisms and thoughtful query optimization.

Security Considerations

The application addresses security concerns through several measures. Prepared statements prevent SQL injection attacks, ensuring user inputs are properly sanitized. Role-based authentication restricts unauthorized access. However, additional security measures such as password hashing and encryption could be considered for enhanced data protection.

Future Enhancements:

Future enhancement could involve implementing a role-based access control mechanism, providing more fine-grained control over user permissions and actions. This would enhance security and allow administrators to define different levels of access for students, instructors, and potentially other roles.

Part 2: Web App / Tradi&onal MVC Servlets and JSPs Implementa&on

Task:

Enhance the Student Grading System by transforming it into a web applica8on using tradi8onal

MVC Servlets and JSPs.

Architecture Overview

The system's architecture follows the Model-View-Controller (MVC) design pattern. The Model is the database, which contains tables for storing user information, course information, and grades. The View is the user interface, implemented using JSP and Bootstrap. The Controller is the collection of Servlets that handle requests, communicate with the database, and manage the business logic. The data model is composed of six tables - students, instructors, courses, students_courses, instructors_courses, and grades. These tables are related through keys, allowing for efficient retrieval of information based on relationships. For example, the grades table references the students and courses tables, allowing for easy retrieval of a student's grade for a particular course. The interaction flow between components is as follows: a user submits a request to the server, which is handled by a Servlet. The Servlet communicates with the database to retrieve or update information, and then renders a JSP page as a response. The JSP page contains dynamically generated content based on the information retrieved by the Servlet.

The architecture of the system follows a client-server model, where clients (web browsers) interact with the server through HTTP requests and responses. The main components include:

1. **Front-End (JSP Pages):**
    - Users access the system through different JSP pages, such as login.jsp, student-dashboard.jsp, instructor-dashboard.jsp, student-grades.jsp, instructor-grades.jsp.
    - The JSP pages use Java code to retrieve and display data from the server.
2. **Servlets:**
    - Java servlets handle HTTP requests and manage the business logic of the application.

- Key servlets include **LoginServlet**, which handles user authentication, and **DataServlet**, which retrieves and updates data based on user roles (student or instructor) and specific courses.

3. **Service Layer:**
    - The **DataService** class encapsulates the logic for retrieving and updating data. It interacts with the database and prepares data for presentation.
4. **Database Layer:**
    - The **Database** class manages the connection to the MySQL database.
    - It provides methods to query user data, grades, courses, and instructor data.

Data Model:

The application involves several data entities:

- Students and instructors, each with their own username, password, and other attributes.
- Courses, associated with students and instructors.
- Grades, representing student performance in courses.

Implementation Details:

- User authentication is performed using the **LoginServlet**, which validates user credentials against records in the database.
- Upon successful login, users are redirected to their respective dashboards (**student-dashboard.jsp** or **instructor-dashboard.jsp**), where they can view their courses.
- Instructors can access course-specific grade information through the **instructor-grades.jsp** page. They can update student grades using forms that submit data to the **DataServlet**.

Analytical Thinking:

- The application design appears to follow a modular structure, separating concerns and promoting code reuse.
- Usage of servlets and JSP pages separates presentation logic from business logic.
- The system handles different user roles (students and instructors) and provides role-based access control.

Challenges and Solutions:

- Handling user authentication securely, which is addressed through the **LoginService** and hashed passwords in the database.
- Ensuring data consistency and concurrency when updating grades, which is managed by the **updateGrade** method in **DataService**.

Security Considerations:

- User authentication is implemented to prevent unauthorized access to sensitive data.
- The use of hashed passwords in the database enhances security.

Future Enhancements:

Potential future enhancements could include the addition of data visualization and analytics, more advanced user management, and integration with external systems such as Learning Management Systems. Additionally, the system could be scaled to handle larger datasets or higher traffic.

Part 3: Web App / Spring MVC and Spring REST Implementa&on

Task:

Advance the Student Grading System by migra8ng it to a web applica8on u8lizing Spring MVC

and Spring REST.

Architecture Overview

The Student Grading System follows a three-tier architecture, separating presentation, business logic, and data storage layers.

Presentation Layer

The presentation layer is built using JavaServer Pages (JSP) for rendering dynamic content. The **login.jsp**, **instructor.jsp**, **instructor-information.jsp**, **student.jsp**, and **student-information.jsp** files handle user interface components.

Controller Layer

The controller layer is managed by the Spring MVC framework. The **LoginController** and **DisplayInformationController** classes handle user authentication, student/instructor dashboard rendering, and grade management.

Service Layer

The service layer encapsulates business logic and data retrieval. The **LoginService** class handles user validation, while the **DataRetrievingService** class retrieves student/course information. The **Database** class manages interactions with the MySQL database.

Database Layer

The MySQL database stores user accounts, courses, and grade information. Tables include **students**, **instructors**, **courses**, and **grades**, with appropriate relationships established.

Implementation Details

User Authentication and Dashboard

The **LoginController** authenticates users and determines their roles. Students and instructors are directed to their respective dashboards, where course information is displayed.

Grade Management

The **DisplayInformationController** manages grade information for instructors. The **instructor-information.jsp** page allows instructors to view and update student grades.

Analytical Thinking

Modularity and Reusability

The system's modular design enables code reusability. The Spring MVC framework's components ensure clear separation of concerns and maintainability.

Role-Based Access

Role-based access control ensures that students and instructors have access to relevant features and data based on their roles.

User Interface

The system utilizes JSP templates for a consistent and intuitive user interface.

Challenges and Solutions

Security Concerns

Challenge: Implementing secure user authentication. Solution: Passwords are securely hashed before storage in the database.

Grade Updates

Challenge: Ensuring consistent and accurate grade updates. Solution: The **instructor-information.jsp** page allows instructors to update grades for students and provides immediate feedback.

Security Considerations

The system employs hashed passwords to enhance user account security. Role-based access control restricts user actions based on their roles, preventing unauthorized access to sensitive functionalities.

Future Enhancements

In the future, the system could benefit from the integration of data visualization and analytics for comprehensive insights into student performance. Advanced user management features, including role-based access control, could be implemented to enhance administrative control and security. Furthermore, seamless integration with external Learning Management Systems (LMS) would facilitate a cohesive learning environment. Architectural optimizations for scalability would prepare the system to efficiently handle larger datasets and increased user activity. These potential enhancements collectively contribute to a more sophisticated and adaptable Student Grading System that aligns with evolving educational requirements and technological advancements.

In conclusion, the presented system, the "Students Grading System," demonstrates a robust and well-structured architecture for managing student and instructor interactions within an educational environment. Through the utilization of Spring MVC framework and Java technologies, the system effectively facilitates user authentication, course management, and grade updates.

The architectural overview highlighted the system's clear separation of concerns, achieved through the Model-View-Controller (MVC) design pattern. This design promotes modularity, maintainability, and extensibility, enabling the system to accommodate future enhancements with ease. The data model encompasses comprehensive user profiles, course details, and grade records, ensuring accurate representation of key entities and their relationships.

The interaction flow among components has been meticulously established. The login process seamlessly validates user credentials and directs them to their respective dashboards, whether as students or instructors. Navigational controls within each dashboard facilitate intuitive movement between course information, grade updates, and the home page. Furthermore, the integration of JSP templates allows for dynamic content rendering, enhancing user experience.

The implementation documentation sheds light on critical code segments, including the login validation, course information retrieval, and grade update functionalities. These implementations reflect prudent design choices, such as session management and error handling, contributing to a reliable and user-friendly system.

While the current system is well-structured and functional, several future enhancements could further elevate its utility. These include the integration of data visualization tools for enhanced analytics, advanced user management capabilities, and potential integration with external systems like Learning Management Systems. Additionally, considering scalability to accommodate larger datasets and increased traffic is imperative for sustained performance.

Throughout the development journey, challenges were met with innovative solutions, demonstrating the analytical thinking and problem-solving skills applied during the process. Security considerations were embedded in the design, ensuring the protection of sensitive student data and system integrity.