Atypon training
Java and DevOps (May 2023)
Section 2

DB/Web/Spring Assignment

Assignment 5

Name: Mohammad Al-Qaisy

Date: 6th Aug 2023

Table of Contents

Introduction 3
System Users and Their Features 4
Database Structure 6
JDBC Backend 10
Student Grading System – terminal version 18
Student Grading System – servlet version 23
Student Grading System – MVC spring version 27
Future Updates 31

Introduction

The Student Grading System seeks to streamline the intricate tasks associated with student evaluation and course management. This dynamic system consists of three primary user roles: administrators, teachers, and students. Each user category is equipped with specific functionalities tailored to their responsibilities, allowing for a seamless flow of information and actions.

This report delves into the core features of the Student Grading System. It will provide an in-depth exploration of the distinct capabilities afforded to administrators, teachers, and students, elucidating the ways in which this system simplifies their tasks. The subsequent sections will delve into the comprehensive range of features that define the Student Grading System for each user group, showcasing its adaptability and potential to enhance the educational landscape.

System Users and Their Features

The Student Grading System is a comprehensive platform designed to accommodate the distinct needs and responsibilities of three primary user categories: administrators, teachers, and students. Each user group is granted specific features and functionalities that empower them to effectively manage and engage with the educational ecosystem.

1. Administrators:

Administrators wield the highest level of access within the Student Grading System. The key features available to administrators include:

User Management: Administrators have the authority to add new users to the system. They can create accounts for teachers and students, ensuring that each user is granted appropriate access and privileges.

Course Management: Administrators can add new courses to the system. They can also view the existing courses and manage their details.

Enrollment: Administrators have the capability to enroll students in specific courses, facilitating the organization of classes and academic schedules.

View Teachers and Students: Administrators can access comprehensive lists of teachers and students, making it easier to monitor the overall student-teacher ratio and facilitate communication.

2. Teachers:

Teachers play a pivotal role in the educational process, and the Student Grading System equips them with tools to effectively manage their courses and assess student performance. The features available to teachers include:

Student List and Grades: Teachers can access a list of students enrolled in their courses, along with their corresponding grades. This feature aids in tracking student progress and identifying areas where additional support may be needed.

Mark Submission: Teachers can input student marks for assignments, exams, and other assessments, ensuring accurate and up-to-date grading.

Edit Marks: In the event of errors or adjustments, teachers have the ability to edit and update student marks.

Marks Analysis: The Student Grading System's Marks Analysis feature serves as a comprehensive tool to evaluate student performance across various assessments. This feature offers essential insights for teachers, helping them make informed decisions regarding curriculum enhancements and student support. The Marks Analysis feature calculates several key statistics for assessments, including the total number of marks, median, average, highest score, and lowest score. These metrics allow educators to assess the overall performance distribution, identify trends, and ensure fairness in grading. Whether it's for assignments, exams, or other assessments, the Marks Analysis feature provides a comprehensive overview that aids in maintaining educational excellence and fairness.

3. Students:

Students are the recipients of education and the primary beneficiaries of the Student Grading System. The features available to students include:

Course Access: Students can view the list of courses they are enrolled in.

Grades and Performance: Students can access their grades for assignments, exams, and other assessments. This feature helps them track their academic progress and identify areas of improvement.

In essence, the Student Grading System empowers administrators, teachers, and students with a set of tailored features that align with their roles and responsibilities.

Database Structure

The foundation of any sophisticated software system lies in its database architecture. The provided code snippet outlines the database structure for an Education Management System, designed to support a Student Grading System. This database structure serves as the backbone for managing user accounts, courses, enrollments, and grading within an educational institution. In this report, we will delve into the purpose and relationships of each table, highlighting their significance in facilitating efficient education administration.

1. user Table:

The user table holds essential information about the users of the system, which includes administrators, teachers, and students. Each user is uniquely identified by an id and associated with the following attributes:

- name: The user's full name.
- email: The user's email address.
- password: The user's password for authentication.
- role: Denotes the user's role (admin, teacher, or student).

2. course Table:

The course table captures the details of the various courses offered within the institution. Each course is represented by an id and includes the following attributes:

- name: The name of the course.
- teacher_id: A reference to the id of the teacher responsible for the course, linking to the user table.

3. enrollment Table:

The enrollment table facilitates the enrollment of students into courses. Each enrollment is assigned a unique id and consists of the following attributes:

- user_id: References the id of the student being enrolled, linking to the user table.

- course_id: References the id of the course being enrolled in, linking to the course table.
- enrollment_status: Denotes the enrollment status (e.g., enrolled, withdrawn).

4. grade Table:

The grade table captures the grades assigned to students for their performance in courses. Each grade record is identified by an id and contains the following attributes:

- student_id: References the id of the student for whom the grade is assigned, linking to the user table.
- course_id: References the id of the course for which the grade is assigned, linking to the course table.
- grade: The numerical grade assigned to the student.

Relationships:

The “teacher_id” in the **course table** establishes a link between courses and their respective teachers.

The “user_id” and “course_id” in the **enrollment table** create connections between students and their enrolled courses.

The “student_id” and “course_id” in the **grade table** establish the link between students and their course grades.

Routines:

The database routines provided play a pivotal role in managing key aspects of the Education Management System. These procedures, when invoked, interact with the underlying database to carry out essential operations related to grading, enrollment, and grade editing. I will provide a comprehensive understanding of each routine's purpose, parameters, and functionality.

1) add_grade Procedure:

The “add_grade” procedure facilitates the addition of grades for a student in a specific course. It accepts three input parameters: “student_id_param”, “course_id_param”, and “new_grade”. The procedure performs the following actions:

Checks if a grade entry already exists for the specified student and course combination.

Initiates a transaction to ensure data integrity.

If a grade entry exists, updates the existing grade by incrementing it with the “new_grade” value.

If no grade entry exists, inserts a new record into the grade table with the provided student ID, course ID, and new grade value.

Commits the transaction, ensuring that changes are saved.

2) edit_grade Procedure:

The “edit_grade” procedure enables the modification of a student's grade in a specific course. It accepts three input parameters: “student_id_param”, “course_id_param”, and “new_grade”. The procedure performs the following action:

Updates the grade value in the grade table for the specified student and course combination with the provided “new_grade” value.

3) enroll_student Procedure:

The “enroll_student” procedure is responsible for enrolling a student in a course. It takes two input parameters: “s_id” (student ID) and “c_id” (course ID). The procedure executes the following steps:

Checks if the student is already enrolled in the specified course by querying the enrollment table.

If the student is not already enrolled, inserts a new enrollment record into the enrollment table, indicating that the student is now enrolled in the course.

Additionally, inserts a new grade entry into the grade table with an initial grade value of 0 for the enrolled student in the course.

If the student is already enrolled in the course, the procedure returns a message indicating that the enrollment has failed.

In conclusion, the database structure provided forms the backbone of the Education Management System. It captures the vital aspects of user authentication, course management, student enrollment, and grading. This system enables administrators, teachers, and students to interact seamlessly, fostering efficient educational processes and facilitating the assessment of student performance. The intricately woven relationships between tables ensure data integrity and comprehensive tracking of education-related activities.

JDBC Backend

Java Database Connectivity

I've employed two distinct JDBC implementations in my project, each serving a specific purpose:

JDBC for Student Grading System (SGS) - terminal version and Student Grading System (SGS) - servlet version:

The first type of JDBC implementation is utilized in both SGS terminal and SGS servlet of the project. This approach involves using Java Database Connectivity (JDBC) to establish connections and interact with the database. SGS terminal and SGS servlet of the project leverage JDBC for seamless communication between the application and the database. This implementation enables various database operations, such as querying, updating, and managing data, effectively supporting the functionalities required in these project sections.

JDBC Template for Student Grading System (SGS) - MVC spring version:

The second type of JDBC implementation, known as the JDBC Template, is specifically employed in SGS MVC spring of the project. This approach utilizes the JDBC Template provided by the Spring Framework. SGS MVC spring of the project leverages the efficiency and convenience of the JDBC Template to streamline database interactions. By utilizing the JDBC Template, database operations become more concise and maintainable, ultimately enhancing the overall quality of the project.

In summary, my project features two types of JDBC implementations. The first type involves standard JDBC usage for SGS terminal and SGS servlet, enabling essential database interactions throughout these project sections. The second type employs the JDBC Template provided by the Spring Framework, exclusively used in SGS MVC spring to optimize and simplify database interactions within that particular project component. These two JDBC implementations collectively contribute to the project's functionality and efficiency across its distinct parts.

For standard JDBC I used **DBConnection**

The DBConnection class is designed to centralize and manage database connectivity for the Education Management System. It encapsulates a set of static methods responsible for establishing, maintaining, and closing database connections. This approach promotes code reusability and ensures consistent connection handling across the application.

makeConnection() Function:

This private function is responsible for establishing a connection to the MySQL database. It employs the MySQL JDBC driver by invoking the Class.forName method. If the connection is not already established or is closed, it uses the DriverManager.getConnection method to create a new connection. This function ensures that a valid and open connection is available for database interactions.

getConnection() Function:

This static public function is used to obtain a valid database connection. It calls the makeConnection() function to ensure a connection is established and then returns the connection object. By providing a centralized way to access a database connection, this function promotes consistency and simplifies the process of obtaining connections throughout the application.

getQuery(String query) Function:

This static public function is responsible for executing SQL queries and returning the result set. It ensures that a valid connection is established by invoking the makeConnection() function. A Statement object is created, and the provided SQL query is executed using the executeQuery method. The resulting ResultSet is returned to the caller for further processing.

closeConnection() Function:

This static public function is used to close the database connection if it is not already closed. It performs a check to determine whether the connection is open and, if so, calls the close() method to gracefully close the connection. This function helps manage resources efficiently by releasing connections when they are no longer needed.

User

The User class in the Education Management System forms the foundation for user authentication and encapsulates functionalities related to user roles. This class supports the system's user management by providing methods to verify user credentials and determine their roles.

User Class Overview:

The User class is designed to manage user-related operations across different roles, including administrators, teachers, and students. It provides a basis for user authentication and role-based actions.

User Authentication:

The checkUser() method is responsible for verifying user credentials against the database. It takes the user's name and password as inputs and performs the following steps:

Retrieves user data from the user table using a SQL query.

Iterates through the result set to compare the provided name and password with the stored data.

Utilizes BCrypt verification to validate the password's authenticity.

If both the name and password are verified, identifies the user's role based on the role string stored in the database.

Returns the index corresponding to the user's role within the userTypes array.

Role Handling:

The User class does not directly manage role-specific functionalities, but it establishes the basis for differentiating between roles. The role attribute indicates the user's role (admin, teacher, student, or not a user).

Function Contribution:

The User class provides fundamental authentication and role-handling functionalities essential for the overall system. It ensures that only authorized users can access the system and assigns appropriate roles to users. This foundation enables the system to tailor experiences and permissions based on user roles.

Benefits:

Security: The checkUser() method ensures that only authenticated users with valid credentials can access the system.

Role Differentiation: The user's role is identified, allowing the system to customize user experiences and permissions.

Centralized Authentication: User authentication logic is centralized in the User class, promoting consistency and maintainability.

Conclusion:

The User class's significance lies in its ability to authenticate users and differentiate their roles within the Education Management System. By providing a secure and standardized mechanism for user verification and role identification, this class forms the basis for user interactions, ensuring controlled access and personalized experiences for administrators, teachers, and students.

Admin

The Admin class plays a critical role in managing various administrative tasks within the Education Management System. This class is responsible for interacting with the database to handle user management, course management, and enrollment processes. The provided functions demonstrate how the Admin class leverages database operations to achieve these functionalities. We'll now explore each function's purpose and its contribution to the overall system.

1. readCourses() Function:

This function retrieves course data from the database using a SQL query and populates a courses map. It iterates through the result set, extracting course IDs and names, which are then added to the map. The function allows administrators to access information about available courses for various purposes, such as enrollment or course management.

2. readUsers() Function:

This function reads user data from the database and categorizes users into teachers and students based on their roles. The teachers and students maps are populated with teacher and student IDs and names. This information is crucial for administrators to manage users and assign appropriate roles within the system.

3. addUser() Function:

The addUser() function enables administrators to add new users to the system. It performs user validation, hashes the password using BCrypt, and constructs a SQL query to insert user data into the user table. The function then updates the respective map (teachers or students) with the new user's information.

4. enrollStudent() Function:

This function facilitates student enrollment in courses. It validates the existence of the student and course IDs in the provided maps, prepares a SQL query to call the enroll_student stored procedure, and retrieves the enrollment result message. The function provides a seamless way for administrators to enroll students in courses and receive feedback on the enrollment process.

5. addCourse() Function:

The addCourse() function allows administrators to add new courses to the system. It constructs an SQL query to insert course data into the course table, including the course name and teacher ID. After execution, the function updates the courses map with the new course information.

In summary, the provided functions in the Admin class showcase how administrators interact with the database to manage users, courses, and enrollments in the Education Management System. These functions streamline administrative tasks, enhance data accuracy, and provide a user-friendly interface for administrators to carry out their responsibilities. By leveraging these database interactions, the Education Management System becomes a more efficient and organized platform for education administration.

Teacher

The Teacher class within the Education Management System is responsible for providing teachers with the tools they need to manage their courses, evaluate student performance, and interact with the database. The functions described below demonstrate how the Teacher class interacts with the database to fulfill these responsibilities. This report will provide insights into each function's purpose and how they contribute to efficient course management and grading.

1. updateMyCourses() Function:

This function updates the teacher's list of courses by querying the database for courses associated with the teacher's name. It clears the existing list of myCourses and then populates it with course IDs and names obtained from the query result. This function enables teachers to stay informed about the courses they are responsible for.

2. editMark() Function:

The editMark() function allows teachers to modify student grades for a specific course. The function ensures the teacher has access to the course through the accessCourse() check. It then prepares a call to the edit_grade stored procedure using the provided student ID, course ID, and new grade. The updated grade is directly applied to the database through the procedure.

3. classList() Function:

This function generates a class list for a given course, displaying student IDs and their corresponding grades. The function verifies the teacher's access to the course, constructs a SQL query to retrieve student IDs and grades from the database, and then formats the information into a StringBuilder. This class list serves as a quick reference for teachers to monitor student performance.

4. sizeOfClass() Function:

The sizeOfClass() function calculates the size of a class (number of enrolled students) for a given course. Similar to other functions, it ensures the teacher's course access before executing a SQL query to count the number of enrolled students in the course. The function returns the class size.

5. addMarks() Function:

Teachers can use the addMarks() function to input a list of marks for enrolled students in a specific course. The function confirms the teacher's access to the course, fetches enrolled student IDs from the database, and iterates through the list of marks. For each mark, it calls the add_grade stored procedure to add the mark to the student's record in the grade table.

6. accessCourse() Function:

The accessCourse() function serves as a safeguard to ensure that a teacher is accessing only their assigned courses. It checks whether the provided course ID exists in the teacher's list of courses (myCourses). If not, it throws an IndexOutOfBoundsException, preventing unauthorized access.

In summary, the Teacher class functions facilitate efficient course management and grading through database interactions. These functions empower teachers to update course information, modify grades, access class lists, input marks, and ensure course-specific access. By leveraging these functions, teachers contribute to a more streamlined and effective education management process within the system.

Student

The Student class within the Education Management System encapsulates student-specific functionalities, including the showCourses() function. This function facilitates students' ability to view their enrolled courses and the corresponding grades they have achieved.

showCourses() Function:

The primary objective of the showCourses() function is to empower students by providing them with a comprehensive list of the courses in which they are currently enrolled, along with their corresponding grades. This function enables students to stay informed about their academic progress and performance.

Student Grading System – terminal version

In the terminal version of the Student Grading System, the implementation is divided into two distinct components: one for the server and the other for the client. This separation allows for modular and independent development of the server-side and client-side functionalities, facilitating efficient communication between them. This report provides an overview of both implementations and their roles in enabling a functional terminal-based Student Grading System.

Server-Side Implementation

The provided code snippet presents the server-side implementation of the Student Grading System, demonstrating the interaction between clients and the server. The server-side operation revolves around the Main class, which sets up the server socket, and the ClientHandler class, responsible for managing individual client interactions. This report provides an overview of the server-side architecture, the role of the Main and ClientHandler classes, and their significance in enabling communication between clients and the Student Grading System.

Main Class:

The Main class serves as the entry point for the server-side operations of the Student Grading System. Key features of the Main class include:

Setting up a server socket on a designated port (8000).

Utilizing an ExecutorService to manage thread pooling for handling multiple clients concurrently.

Accepting incoming client connections through the serverSocket.accept() method.

Instantiating a ClientHandler for each client and executing it using the executor.

Handling IOExceptions and gracefully closing the server socket after processing.

ClientHandler Class:

The ClientHandler class is responsible for managing individual client interactions. It implements the Runnable interface to operate in a separate thread for each client. The ClientHandler class features:

Establishing communication streams (inputFromClient and outputToClient) with the client socket.

Sending an initial welcome message to the client upon connection.

Continuously prompting the client for their choice (login or exit).

Authenticating the user's credentials using the User.checkUser() method.

Creating appropriate user control instances (Admin, Teacher, or Student) based on the user's role and initiating user interactions.

Sending messages back to the client, guiding them through the process.

Closing the client socket after interaction completion.

Functionality and Significance:

The server-side implementation facilitates interactions between clients and the Student Grading System. Clients can log in as Admin, Teacher, or Student users, based on which control instance is created. Each control instance manages user-specific interactions and functionalities. The server handles multiple clients concurrently using thread pooling, providing scalability and responsiveness.

Benefits:

Concurrency: The use of thread pooling allows the server to handle multiple clients concurrently without overwhelming system resources.

User Authentication: Clients are authenticated using the User.checkUser() method, ensuring secure access to the system.

Modularity: The ClientHandler class encapsulates client-specific logic, promoting code modularity and maintainability.

Scalability: Thread pooling enables the server to efficiently handle a growing number of clients without degradation in performance.

Conclusion:

The server-side implementation of the Student Grading System demonstrates effective client-server communication. The Main class sets up the server socket and thread pooling, while the ClientHandler class manages client interactions and user authentication. Together, these classes ensure smooth, concurrent, and secure access to the Student Grading System for administrators, teachers, and students, providing an efficient and interactive user experience.

Client-Side Implementation

The provided code snippet showcases the client-side implementation of the terminal version within the Student Grading System. This implementation establishes communication between the client and the server, enabling users to interact with the system through the terminal interface. This report offers an overview of the client-side architecture, the role of the Main class, and its significance in facilitating user interactions and communication with the server.

Main Class:

The Main class serves as the entry point for the client-side operations of the terminal version. Key features of the Main class include:

Establishing a socket connection with the server at the specified hostname ("localhost") and port (8000).

Creating communication streams (inputFromServer and outputToServer) for data exchange with the server.

Receiving and displaying an initial welcome message from the server.

Displaying options to the user (login or exit) and prompting for user input.

Sending user input to the server and receiving responses.

Authenticating users and initializing the appropriate user control instance based on the user's role.

Initiating the corresponding user control (Admin, Teacher, or Student) based on authentication.

The Main class effectively connects users to the Student Grading System server, allowing them to interact with the system through the terminal interface.

User Control Classes:

The Main class initializes user control instances based on user authentication. The user control classes (AdminControl, TeacherControl, StudentControl) are expected to manage specific user interactions and system functionalities for each role. They play a pivotal role in guiding users and handling data exchange between the client and server.

Functionality and Significance:

The client-side implementation serves as the interface between the user and the Student Grading System server. It provides users with options, prompts, and authentication processes. The client-side architecture supports user authentication and initiates role-specific functionalities based on the received user type.

Benefits:

User Interaction: The client implementation offers a user-friendly terminal interface for users to interact with the Student Grading System.

Secure Authentication: User authentication ensures that only authorized individuals can access the system.

Role-Specific Functionality: Different user control instances facilitate tailored interactions and functionalities based on user roles.

Conclusion:

The client-side implementation of the terminal version in the Student Grading System enables users to interact with the system through the terminal interface. The Main class establishes communication with the server, handles user authentication, and initializes role-specific user control instances. This implementation fosters a seamless user experience and efficient communication between clients and the Student Grading System server, contributing to the system's overall functionality and usability.

Student Grading System – servlet version

The servlet version of the Student Grading System amalgamates Java servlets, Java Server Pages (JSP), and JDBC connectivity to deliver a dynamic web-based interface. Servlets handle user requests, interact with the database using JDBC, and generate responses reflected through JSP pages. JSP files seamlessly integrate Java with HTML, ensuring personalized dashboard views. This integration fosters an immersive user experience, enabling real-time interactions, role-based actions, and efficient data management through a web browser interface.

Login Servlet Implementation

The provided code snippet presents the implementation of the LoginServlet class in the servlet version of the Student Grading System. This servlet handles user authentication and directs users to their respective dashboards based on their roles.

Servlet Overview:

The LoginServlet is a Java servlet designed to handle user authentication requests and manage user redirection to appropriate dashboards based on their roles. It follows the HTTP POST method to process user login credentials, interact with the User class to verify credentials, and then direct users accordingly.

doGet(HttpServletRequest request, HttpServletResponse response) Method:

This method is responsible for handling HTTP GET requests to the /login URL pattern. It forwards the request to the login.jsp view, where users can input their credentials. This is the initial view presented to the user when accessing the login page.

doPost(HttpServletRequest request, HttpServletResponse response) Method:

This method handles the HTTP POST requests sent when users submit their login credentials. Key functionalities of this method include:

Retrieving the name and password parameters from the HTTP request.

Using the User.checkUser() method to verify user credentials and obtain the user's role.

Based on the user's role, the servlet performs the following actions:

For "Admin" users, it constructs a redirect URL to the AdminDashboard servlet.

For "Teacher" users, it constructs a redirect URL to the TeacherDashboard servlet.

For "Student" users, it initializes a new instance of the StudentControl class, which handles student-specific functionalities.

If the user's credentials are invalid (userType == 3), an error message is set as an attribute and the user is redirected back to the login page to re-enter their credentials.

Benefits and Significance:

The LoginServlet plays a crucial role in the servlet version of the Student Grading System by ensuring secure user authentication and efficient navigation. By processing login credentials and interacting with the User class, it provides an effective means of verifying user access rights. Additionally, it helps ensure that users are directed to the appropriate dashboard based on their roles.

Conclusion:

The LoginServlet is a fundamental component within the servlet version of the Student Grading System. It enables secure user authentication and role-based redirection, enhancing the user experience and ensuring that users are directed to the relevant sections of the system based on their roles. The servlet's functionality contributes to the overall effectiveness and usability of the Student Grading System's web-based interface.

Controls Implementation

After the initial authentication process handled by the servlet version of the Student Grading System, user interactions are seamlessly directed to the appropriate control modules: StudentControl, TeacherControl, and AdminControl, each tailored to their respective roles. These control modules encapsulate the specific functionalities and actions that users can perform within the system, creating a cohesive and role-based experience.

StudentControl empowers students to view their enrolled courses, grades, and progress. Through user input, they can access course details, view their grades, and potentially initiate grade improvement processes.

TeacherControl equips teachers with tools to manage their courses, update marks. Teachers can input grades, modify existing marks, and access class lists through intuitive user interfaces.

AdminControl empowers administrators to manage users, courses, and overall system functionalities. Administrators can create new users, add courses, and enroll students in courses based on the provided inputs.

These control modules harness the servlet architecture to efficiently process user inputs, initiate relevant actions, and display outputs. Through a well-designed interface, users are prompted to input data and navigate through the functionalities of their respective roles. The system then orchestrates the backend operations, executes the requested actions, and presents users with meaningful outputs, contributing to a seamless and productive experience within the Student Grading System.

Student Grading System – MVC spring version

In the MVC Spring version of the Student Grading System, the architecture is organized into two distinct packages: the controllers package and the models package. This design follows the Model-View-Controller (MVC) architectural pattern, promoting separation of concerns, modularity, and efficient code management. This report provides an overview of these packages and their roles within the system, highlighting how they contribute to the overall functionality and structure of the application.

User interactions are seamlessly facilitated through HTML pages that connect to the controllers, allowing for a user-friendly and intuitive experience. Where HTML pages serve as the view layer, and controllers handle the business logic and data interactions. This approach ensures a clear separation between the presentation layer and the underlying functionalities of the application. Here's how this process unfolds:

HTML Pages (View Layer):

The HTML pages represent the user interface that users interact with. These pages are responsible for collecting user input, displaying data, and conveying information. They are designed to be visually appealing and intuitive, providing users with an accessible means to interact with the Student Grading System.

Benefits:

User-Friendly Interface: HTML pages provide an intuitive and familiar environment for users to input data and interact with the system.

Modular Design: The clear separation between the view and the business logic promotes modularity, making it easier to maintain and extend the system.

Scalability: As new functionalities or features are added, new HTML pages can be created, and controllers can be designed to handle them without disrupting existing components.

The architecture is divided into two packages: controllers and models. These packages facilitate the seamless interaction between users and the underlying system functionalities, promoting a structured, maintainable, and user-friendly experience.

Controllers Package:

The controllers package encapsulates the logic of handling user requests, processing inputs, and rendering appropriate outputs. It consists of distinct controller classes tailored to different user roles:

LoginController: This controller handles the authentication process initiated from the login.html page with the URL "/login". It verifies user credentials and determines the user type to redirect them to their respective dashboard controllers.

AdminController: Catering to administrative users, this controller manages inputs from the AdminDashboard.html page (URL: "/admin"). It processes administrative actions and returns relevant outputs.

TeacherController: Serving teachers, this controller receives inputs from the TeacherDashboard.html page (URL: "/teacher"). It processes teacher-specific tasks and responds with relevant outputs.

StudentController: Designed for students, this controller handles inputs from the StudentDashboard.html page (URL: "/student"). It processes student-related actions and returns suitable outputs.

Models Package:

The models package employs the JDBC Template to interact with the database and contains classes representing distinct user roles:

User: This class provides functionality to authenticate users and determine their roles. It interfaces with the database to verify user credentials and retrieve their roles. The user's role is then used to route them to the appropriate controller and dashboard. This model serves as the bridge between the authentication process and the respective user actions.

Admin: This class encapsulates administrative functionalities, allowing interactions with user data, course management, and other administrative tasks.

Teacher: Representing teacher functionalities, this class facilitates course management, grade updates, and interactions with enrolled students.

Student: Catering to students, this class provides the means to view enrolled courses, grades, and academic progress.

Interaction Flow:

Upon accessing the system, users input their credentials through the login.html page.

The User class verifies the provided credentials using the JDBC template and database interactions.

Based on the authentication results, the User identifies the user's role (Admin, Teacher, or Student).

The user's role is then communicated to the respective controller, determining the route to their dashboard (AdminDashboard, TeacherDashboard, or StudentDashboard).

Benefits:

Structured Architecture: The separation of controllers and models promotes modular development and enhances code organization.

Role-Based Design: Each controller serves a specific user role, ensuring tailored functionalities and a personalized experience.

Maintainability: The clear separation of concerns allows for easier maintenance and updates to specific components without affecting others.

Conclusion:

In the MVC Spring version of the Student Grading System, the controllers package handles user interactions and redirects requests to the respective controllers, while the models package employs the JDBC template and user-specific classes to execute database interactions and manage functionalities. This structured design ensures a smooth flow of data, user inputs, and outputs, creating an effective and user-centered educational management system.

Future Updates

The Student Grading System is poised for exciting enhancements that will elevate its functionality, security, and user experience. Some of the planned future updates include:

1. Spring Security Implementation:

Enhancing the system's security, the implementation of Spring Security will provide robust authentication and authorization mechanisms. This addition will ensure that user data remains confidential and that only authorized individuals can access specific functionalities. Spring Security will bolster the system's defense against unauthorized access and potential vulnerabilities.

2. Expanded Features:

The upcoming updates will introduce an array of new features that enrich the user experience and streamline administrative and academic processes. Some of the additions include:

Assignment Management: Teachers will be able to create, manage, and distribute assignments to students through the system. Students can then submit their assignments electronically for evaluation and grading.

File Upload Capability: To facilitate seamless interaction, teachers can upload course-related materials, lecture notes, assignments, and resources. Students will have access to these resources, enhancing their learning experience.

Detailed Assignment and Exam Marking: Teachers can assign marks for individual assignments and exams. The system will maintain a comprehensive record of student performance across different assessments, offering insights into areas of improvement and achievement.

Grade Calculation Flexibility: The system will offer the flexibility to customize grading components, allowing teachers to assign different weightages to assignments, exams, and other evaluations. This customization ensures a fair and representative grading system.

Enhanced Reporting: Users, including administrators, teachers, and students, will have access to comprehensive reports that showcase student progress, course performance, and overall system analytics. These reports will facilitate data-driven decision-making and insight generation.

Conclusion:

With the implementation of Spring Security and the introduction of an array of new features, the Student Grading System is poised to become a comprehensive, secure, and efficient tool for educational management. These updates will contribute to a seamless, user-friendly experience while maintaining data security and accuracy. As the system evolves, it will continue to support educators, administrators, and students in their pursuit of academic excellence.