Atypon Training Program 2023

STUDENT GRADING SYSTEM: JAVA FRAMEWORKS BASED APPLICATION DEVELOPMENT

BY: YUSSIF ABDALLA, The University of Jordan

I. ABSTRACT

In today's advent of computer development system integration come to be possible enabling past programming experiences to be salvaged. This became possible using Java well developed Frameworks rather than using direct programming language. Frameworks help developers to work more with less time and less effort as here you do not have to write everything from scratch. Java Frameworks offers many functional blocks, which are provided with inbuilt libraries maintaining the integrity and stability without compromising the Java application.

In this work, a Java Student Grading System Framework based project application is fully presented. The outcomes of this application project is to develop a comprehensive progressed development skills. This is implemented to introduce and practice of various technologies, frameworks, and concepts related to backend development, web application development, and Spring MVC/Spring REST.

Finally, the work is presented as three distinguished parts: Part one, introduces Command-line / Sockets and JDBC Backend Implementation, part two, Web App / Traditional MVC Servlets and JSPs Implementation, and part three, Web App / Spring MVC and Spring REST Implementation.

II. INTRODUCTION

Java, a popular programming language, has tremendously developed over the years. Currently, it has shifted to make use of frameworks rather than using direct programming language. Frameworks help developers to work more with less time and less effort as here you do not have to write everything from scratch [1]. Consequently, many functional blocks are provided with inbuilt libraries maintaining the integrity and stability of a Java application. Nowadays, Frameworks are a fundamental aspect of software development and are widely used in the Java ecosystem to improve the productivity and maintainability of software projects [1][2]. Diversity of Java Frameworks types exists; such as, Web, ORM, Testing, Logging …etc. Detailed archived lists are available for the developers to expedite building applications, enhanced security, increased scalability, improved maintainability …etc. [2]. Based on application requirements, performance, learning curve, community support and

many other factors one may choose the right Java Frameworks that are needed to be integrated to their project application.

In this work, a report summary of a Java Student Grading System Framework based project design is fully presented. The outcomes of this application assignment is to develop a comprehensive progressed development to introduce and practice on various technologies, frameworks, and concepts related to backend development, web application development, and Spring MVC/Spring REST [3].

Collectively, the work is presented as three distinguished parts: Part one, introduces Command-line / Sockets and JDBC Backend Implementation, which requires to build the initial version of the Student Grading System using the command-line interface. Part two, Web App / Traditional MVC Servlets and JSPs Implementation, where an enhancement to the Student Grading System is carried out by transforming it into a web application using traditional MVC Servlets and JSPs. sockets, and JDBC in the backend. Finally part three, Web App / Spring MVC and Spring REST Implementation, which require to advance the Student Grading System by migrating it to a web application utilizing Spring MVC and Spring REST [3].

Finally, the report covers the Framework design choices, programming techniques, advanced features, more. It also include a reflection section discussing the lessons learned, difficulties encountered, and potential future improvements.

III. COMMAND LINE/SOCKETS AND JDBC BACKEND IMPLEMENTATION

In this first part, complete overview of command line/sockets and JDBC backend implementation and its design is presented, highlighting the most essential aspects of the design.

3.1 System Design and Architecture

The command-line application is a client-server architecture. The client is responsible for sending user requests, and the server handles these requests. Based on the user's username and password, the server determines if the user has the required privileges for the requested action. This is a simplified explanation of the command-line program's operation.

To fully understand how the system works, you need to comprehend the various classes present in both the client and server components. These classes play crucial roles in the overall functioning and interaction between the client and server and understanding them will provide you with insights into the system's functionality and behavior.

3.2.1 The Client-side Classes

1. The Client Class
   Requests a connection to the server.

2. The Page Navigator class

   This class redirects the user to their corresponding main page based on their role. It then invokes the methods viewPage() to display the user's main page and enterOption() to synchronize the sending and receiving of messages between the client and server.

3. Page Abstract class
   This is the class that PageNavigator invokes the viewPage() and enterOption() methods from. The classes AdminPage, StudentPage, and TeacherPage extend the abstract class and override the viewPage() method to display their respective interfaces. The main logic for enterOption() remains in the abstract class.

3.2.1 The Server-side classes

1. Server Class
   The server class receives the client's connection request and assigns a new thread to handle the client. It starts a new client handler thread and passes a new buffered reader and print writer to it. This setup allows each client to communicate effectively with its corresponding client handler.

2. Client Handler

   This component is responsible for determining the user's role and subsequently instructing the User Factory class to create a user object accordingly. The user's role can be categorized as admin, teacher, or student, and all these classes implement the user's interface.

3. User Factory class
   The User Factory class is responsible for creating and returning a user object based on the user's role. The user can be assigned one of the roles: admin, teacher, or student. The User Factory ensures that the appropriate user object is instantiated according to the specified role.

4. GradingSystemDb Class

   Handles all database operations, including finding, deleting, creating, modifying various courses, users, roles, and grades, and serves as a central hub for interacting with the database and managing all relevant data within the application.

5. Admin, Teacher, Student Classes

All of these classes implement the User interface, which includes a getPermissions() method. Each class has a set of predefined permissions, and the getPermissions() method is used to retrieve these permissions. The permissions are then compared with the user's commands to determine if they align with the user's object permissions.

6. Permission Enumeration

   This enumeration contains different permissions that are assigned to admin, teacher, and student classes. Each class is associated with a specific set of permissions.

3.3 Class Interaction

Now that we understand the different classes in the program, it becomes easier to comprehend the flow between the components and their interactions during the program's execution process. Let's take a look at how it unfolds:

1. System Initialization
   - The **Client** requests a connection from the server
   - The **Server** waits for incoming client connections

2. Client-Server Connection
   - The **Server** receives and accepts the client's connection request.
   - Upon receiving a connection, the Server assigns a new thread, creating a unique **Client Handler** instance for each client.
   - The **Client Handler** establishes communication with the client using a buffered reader and print writer for seamless data exchange.

3. . User Authentication
   - The Client Handler sends messages to the client, prompting the user to enter a username and password.
   - The user responds by sending the requested username and password back to the Client Handler.
   - The Client Handler uses GradingSystemDb to check if the user is authorized.

4. User Creation
   - Once the user is shown to be authorized, the Client Handler uses GradingSystemDb to fetch the role of the user, and then creates a user object with the specified role (admin/teacher/student).

5. Request Handling
   - The Client Handler is now prepared to receive commands from the user.

- Upon receiving a command from the user, the Client Handler checks the user's privileges using the getPermission() method from the user object.
- If the user is granted permission to perform the operation, the Client Handler forwards the command to the Request Handler.
- The Request Handler matches the command and executes the necessary operations accordingly.

IV. WEB APP / TRADITIONAL MVC SERVLETS AND JSP IMPLEMENTATION

4.1 System Design and Architecture

An The Model-View-Controller (MVC) is a design pattern in web applications used to separate the application logic from the user interface. As the name implies, the MVC pattern has three layers: the model layer, the view, and the controller layer. Let's take a look at each layer in the program:

4.1.1 The Model Layer

This is the data layer that contains business logic of the system, and also represents the state of the application. It is independent of the presentation layer, the controller fetches the data from the model layer and sends it to the view layer.

Here are the models of the web application:

Course: The Course model embodies the courses undertaken by students and instructed by teachers. It comprises essential attributes such as the course name, course ID, along with their respective getters and setters.

Course Grade

The CourseGrade model represent the student's performance in their individual courses. Each CourseGrade includes the course ID, user ID, and the achieved score. The CourseGrade model facilitates access to and modification of grade information through its provided getters and setters.

Grading System DAO

The grading system dao model facilitates interactions with the database and encompasses various methods for querying and modifying data. These methods include checking user credentials, fetching grades, retrieving course information, and creating new records.

4.1.2 The Controllers Layer

The controller layer acts as an interface between the view and model. It receives requests from the view layer and processes them. Here are the controllers of the web application:

The Login Servlet

The login servlet's primary responsibility is to redirect users to their respective main login pages. It achieves this by authenticating users through their username and password, utilizing the grading system dao. Additionally, the servlet retrieves the user's role based on their username. Upon successful authentication, the controller redirects users to the appropriate pages, such as the admin page, student page, or teacher page, based on their assigned role.

If the logged-in user is a student, the system will retrieve the user's grades from the database and direct them to the student page. On the other hand, if the user is an admin or teacher, they will be redirected to their respective main pages.

The Logout Servlet

The logout servlet allows users to terminate their active sessions effectively by removing the "username" and "password" attributes. This action prompts users to re-enter their credentials for subsequent logins, thereby enhancing security and preventing unauthorized access.

Create Course Servlet

This servlet caters to admin users and handles requests when they choose the "Create a course" option from their main page. The servlet forwards the admin's request to the "create-course-page.jsp" for course creation.

Create Course Action

This servlet is requested from the “create-course-page” after filling a form with information and clicking the subcmit button. The servlet gets the necessary information and creates a course using the GradingSystemDao.

Delete Course Servlet

Dedicated to admin role users, this servlet processes requests when they choose the "Delete a course" option on their main page. The servlet redirects the admin's request to the "delete-course-page.jsp" for course deletion.

Delete Course Action

This servlet is requested from the “delete-course-page” after filling a form with information and clicking the submit button. The servlet gets the necessary information and deletes a course using the GradingSystemDao.

Create User Servlet

This servlet caters to admin users and handles requests when they choose the "Create a user" option from their main page. The servlet forwards the admin's request to the "create-user-page.jsp" for course creation.

Create User Action

This servlet is requested from the “create-user-page” after filling a form with information and clicking the submit button. The servlet gets the necessary information and creates a user using the GradingSystemDao.

Delete User Servlet

Dedicated to admin role users, this servlet processes requests when they choose the "Delete a user" option on their main page. The servlet redirects the admin's request to the "delete-user-page.jsp" for course deletion.

Delete Course Action

This servlet is requested from the “delete-user-page” after filling a form with information and clicking the submit button. The servlet gets the necessary information and deletes a user using the GradingSystemDao.

Modify Servlet

Dedicated to teacher role users, this servlet processes requests when they choose the "Modify a user" option on their main page. The servlet redirects the teacher’s request to the “modify-page.jsp" for grade modification.

Modify Action

This servlet is requested from the “modify-page” after filling a form with information and clicking the submit button. The servlet gets the necessary information and modify the student’s grade using the GradingSystemDao.

View Classes Servlet

This servlet is triggered when the "teacher-page_1" sends a request after the teacher clicks a button to view their classes. The servlet retrieves all the courses taught by the teacher and the corresponding student grades using the GradingSystemDao. The retrieved data is then presented in the "view-classes.jsp" page for the teacher to view.

4.1.3 The Views

The view layer portrays the application's output via a user interface (UI), showcasing the model data retrieved by the controller. Here is the representation of the web application's view:

1. login.jsp

This view exhibits a login form, allowing users to input their username and password. Upon form submission, the login servlet is engaged for authentication. Upon successful validation, users are directed to a corresponding view aligned with their role (such as "student-page.jsp" "teacher-page-1.jsp" or "admin-page.jsp"). In instances of incorrect credentials, the login page will refresh, granting users another attempt.

2. admin-page.jsp

This is the primary page visible to administrators, featuring buttons for various tasks. Admins can perform actions such as creating or removing a course and adding or deleting a user. When an admin clicks a button for a specific task, it may trigger different servlets like CreateUserServlet or DeleteUserServlet, as well as CreateCourseServlet or DeleteCourseServlet, responsible for handling the intended action.

3. create-user-page.jsp

In this view, the admin can easily generate a new user by completing a form containing essential details: username, first name, last name, role name, and password. When the user taps the submit button, a POST request is dispatched to CreateUserAction controller. This action employs the Grading System Dao class to confirm the absence of an existing user before initiating the creation of the new user.

4. delete-user-page.jsp

This view empowers the admin to eliminate a user by supplying the username and password of the intended deletion. Upon submission, the view triggers a POST request directed to the DeleteUserAction controller. The Grading System Dao is then utilized to validate the user's presence before authorizing the deletion process.

5. create-course-page.jsp

This view simplifies the process of generating a new course. The admin provides a course name along with the teacher's ID. Subsequently, the request is transmitted to the CreateCourseAction controller. Here, an examination by GradingSystemDao class is conducted to ensure that a course with the same name doesn’t exist. If such a course is not found, a new course is generated.

6. delete-course-page.jsp

This view simplifies the course deletion procedure. The admin fills a form with the course's ID, and upon clicking the submit button, the request is forwarded to the DeleteCourseAction controller, leading to the removal of the designated course.

7. student-page.jsp

This view is designed solely for students, granting them access to their grades across different courses. The displayed content in this view features the data from the Course Grade model, which has been retrieved and organized by the controller.

8. teacher-page-main.jsp

This serves as the main page accessible to teachers, showcasing buttons for a range of tasks. Teachers can execute actions like modifying grades, as well as viewing classes. Upon a teacher's selection of a task-specific button, it could activate servlets such as ModifyServlet or ViewClassesServlet. These servlets are tasked with managing the desired actions.

9. `teacher-page-modify.jsp`

In this view, the teacher is empowered to change a student's grade by inputting the course ID, student ID, and the new score for the student. Upon submitting this information, a POST request is dispatched to the ModifyAction controller. Here, the controller ensures the teacher's assignment to the course and the student's enrollment in the class before implementing the grade adjustment.

10. `teacher-view-page.jsp`

This view shows the various courses being taught by the teacher, as well as the grades of each student for their respective courses. The displayed content in this view features the data from the Course Grade model, which has been retrieved and organized by the controller.

V. WEB APP / Spring MVC Implementation

5.1 System Design and Architecture

The Spring web application adheres to the Model-View-Controller (MVC) design pattern and is primarily used for developing web applications. In this architectural pattern, Spring MVC encompasses distinct interactive components, namely models, views, and controllers.

The user begins by sending a request to the front controller, which, in this project, is the login controller. The front controller analyzes the username and password entered by the user and checks for the user's role, which can be an admin, student, or teacher. Based on the user's role, the front controller directs the request to the appropriate controller. The controller then handles the user's requests and generates a model—a container holding the requested data of the application. This model is then integrated into a view, which represents the user interface, and finally returned to the user. For a visual representation of this process, please refer to Figure 1.

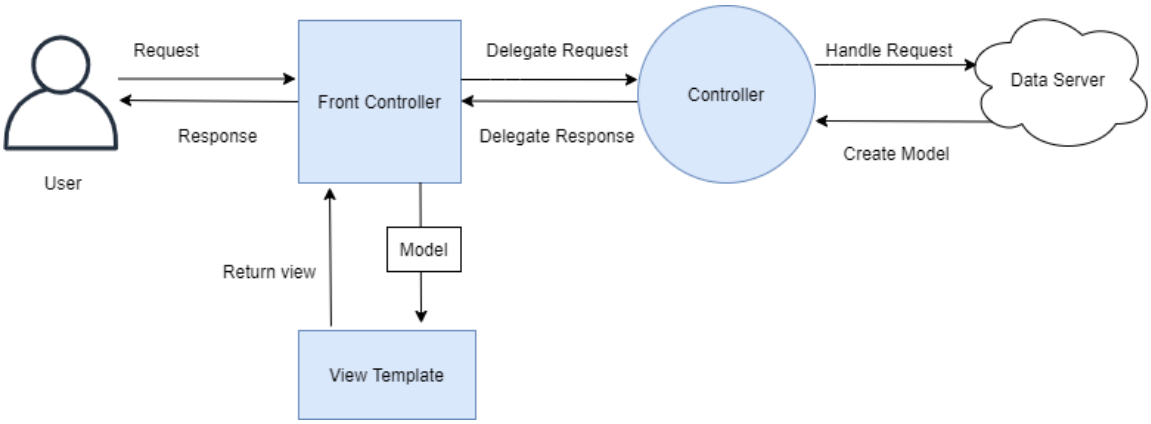

Figure 1: Web application's architecture and design

5.1.1 The Models

Models are components that represent classes and are used to encapsulate business logic and define getter and setter methods. The web application includes five essential models: user, course, grade, and role.

Course

Course represents the courses taken by students and taught by teachers. Each course is uniquely identified by an auto-incrementing ID and includes the course name, teacher's ID, and corresponding getter and setter methods.

Grade

Grades correspond to the students' grades in their respective courses. Each grade is linked to a course through its auto-incrementing ID, contains the student's ID, and stores the obtained score. The Grade model provides the necessary getters and setters for accessing and updating grade information.

User

User represents individual users within the system. Each user is uniquely identified by a user ID and is associated with a first name and last name. The user model class includes appropriate getter and setter methods to handle user data.

Role

Role defines the roles of registered users in the system. It includes an auto-incrementing role ID and links each role to the ID of the corresponding user. Users are assigned role names such as "admin," "teacher," or "student." Additionally, the Role model stores the password associated with each role.

5.1.2 The Controllers

The controller handles user requests and delegates the necessary data to views for rendering. In the web application, the following controllers were utilized:

Login Controller

The login controller is responsible for presenting the main login page to users. It also interacts with the role service to retrieve the user's role based on their username. After authentication, the controller redirects the user to the appropriate page, such as the admin page, student page, or teacher page, depending on their role.
The controller follows the principles of the front controller design pattern. It centralizes the request handling logic and handles different request and dispatches them to appropriate destination based on the user role.

Logout Controller

The logout controller enables users to terminate their active sessions, requiring them to re-enter their username and password for subsequent logins. This enhances security and prevents unauthorized access.

Admin Controller

The admin controller facilitates the management of the admin's page and handles various user demands. Upon accessing the admin's page, the user is presented with a dashboard that provides access to different services, such as creating and deleting user accounts or managing courses. The controller ensures that the appropriate views are displayed based on the actions taken by the admin.

Teacher Controller

The teacher controller views the teacher's page and depending on the teacher's demands it will show a view. The teacher gets a main page where he can choose a service such as viewing his courses and grades of his students and modifying grades.

Student Controller

The Student Controller is responsible for handling requests related to student views and their academic scores in different courses.

5.1.3 The Views

A view is a crucial component of the MVC architecture responsible for representing an HTML/JSP file or any other template generated by the controller. It plays a vital role in providing the user interface output in response to a user's request. The view's responsibility is to render the data received from the controller into a format that is suitable for presentation to the end-user.

Here are the web application views:

1. `Login-page.html (admin/create-user)`

This view presents a login form where users can enter their username and password. When the user submits the form, a request is sent to the login controller for validation. If the user is successfully verified, they will be redirected to a specific page based on their role (e.g., "student," "teacher," or "admin"). In case of invalid credentials, the login page will reload, allowing the user to try again.

2. `admin-page-main.html (admin/create-user)`

This is the main page view for the admin, providing various options accessible through buttons. The admin can perform actions like creating/deleting a course or creating/deleting a user. When the admin clicks a button corresponding to their desired action, the request will be sent to the admin controller to process the chosen task.

3. `create-user.html (admin/create-user)`

This view allows the admin to create a new user by filling in a form with the following information: username, first name, last name, role name, and password. Once the user clicks the submit button a post request is sent to the admin controller and uses both the role service and user service to ensure the user doesn't already exist before creating a new user.

4. `delete-user.html (admin/delete-user)`

This view enables the admin to delete a user by providing the username and password of the user to be deleted. The view sends a post request to the admin controller and uses the role service and user service to verify the existence of the user before proceeding with the deletion.

5. `create-course.html (admin/create-course)`

This view facilitates the creation of a new course. The admin enters a course name and the teacher ID. The request is sent to the admin controller which checks if a course with the same name doesn’t exist it will create a course.

6. `delete-course.html (admin/delete-course)`

This view facilitates the deletion of a course. The admin fills a form with course’s ID and once the submit button is clicked, the request reaches the admin controller and specified course is deleted.

7. `student-page.html (student/)`

This view is exclusively available to the student, allowing them to access their grades in various courses.

8. `teacher-page-main.html (teacher/)`

This is the main page view for the teachers, providing various options accessible through buttons. The teacher can perform actions such as modifying student grades and viewing their classes, the request will be sent to the teacher controller to process the chosen task.

9. `teacher-page-modify.html (teacher/modify)`

This view allows the teacher to modify a student's grade by providing the course ID, student ID, and the new score for the student. Upon submitting the information, a POST request is sent to the teacher controller. The controller verifies that the teacher is assigned to the course and the student is enrolled in the class before applying the grade modification.

10. `teacher-view-page.html (teacher/view-page)`
This view shows the various courses being taught by the teacher, as well as the grades of each student for their respective courses.

5.2 Implementation

5.2.1 JPA and Hibernates

The application utilizes ORM (Object-Relational Mapping) as its data querying and manipulation method, allowing seamless interaction with the relational database without the need for direct SQL usage. JPA (Java Persistence API) serves as the Java specification responsible for managing persistent data in the application, operating at the ORM layer and leveraging Hibernate as the chosen ORM framework.

5.2.2 Application Repositories

The application consists of four repositories, each extending the JpaRepository, enabling CRUD (Create, Read, Update, Delete) operations and JPA-specific functionalities. A dedicated repository has been created for each entity within the program. Here are the repositories and their methods used in the application:

1. **Course Repository**

- **findByTeacherID (int teacherId)**
  Find courses based on the given teacher ID. It returns a list of courses associated with that teacher.

- **findByCourseIdAndTeacherId (inte courseId, int teacherId)**
  Find courses with a specific course ID given by a teacher's ID. It returns a list of courses that match the provided criteria.

- **findByCourseName (String courseName)**
  Find courses with a specific course name.

- **findByCourseId (int courseId)**
  Find courses with a specific course ID.

- **deleteByCourseId (int courseId)**
  Delete courses with a specific course ID.

2. **Grade Repository**

- **findGradeByCourseId (int courseId)**
  Find grades based on the given course ID. It returns a list of grades associated with that course.

- **findGradeByUserId (int userId)**
  Find grades with a specific student given by a user ID. It returns a list of grades that match the provided criteria.

- **findGradeByCourseIdAndUserId (int courseId, int userId)**
  Find grade by specifying a course ID and user ID. It returns a list of grades that match the provided criteria.

3. **Role Repository**

- **findByUserIdAndPassword (int userId, String password)**
  Find roles based on the given user ID and password. It returns a user with the specified password.

- **findByUserId (int userId)**
  Find a role by specifying a user ID. It returns an optional list of grades that match the provided criteria.

- **deleteByUserId (int userId)**
  Delete a role from the role table by giving a user ID.

- **findByUserIdAndRoleName (int userId, String roleName)**
  Find a user by specifying a user ID and role name. It returns an optional list of roles that match the provided criteria.

4. **User Repository**

- **deleteByUserId (int userId)**
  Delete a user from the user table by giving a user ID.

For the detailed code implementation of all the repositories, please refer to the Appendices

5.3 Application Services

The application services act as a bridge between the controllers and repositories. They utilize the repositories and combine them to create specific methods with more complex functionalities than the repositories alone. The controllers rely on these services to handle conditions and retrieve data, enabling them to send it back to the view.gfHere are the services provided by the application:

1. **Course Service**

   - **getTeacherCourseList (int teacherId)**
     Uses the findCourseByTeacherID method from the course repository to return a list of courses taught by a specific teacher.

   - **isValidCourse (int courseId, int teacherId)**
     Uses the findCourseByCourseIdAndTeacherId method from the course repository to check if a specific course is being taught by the logged-in teacher.

   - **courseExistsByName (String courseName)**
     Uses the findCourseByCourseName method from the course repository to check if a course with a given name exists.

   - **courseExistsById (int courseId)**
     Uses the findCourseByCourseId method from the course repository to check if a course with a given ID exists.

   - **createNewCourse (String courseName, String teacherId)**
     Takes a course name and teacher ID, and uses the save() method from JpaRepository to create a new course.

   - **deleteCourseById (String courseId)**
     Uses the deleteCourseById method to delete a course with a given ID.

2. **Grade Services**

   - **getGradesByCourseId (int courseId)**

Uses the findGradeByCourseId method from the grade repository to return a list of grades from a specified course.

- **getGradesByStudentId (int userId)**
  Uses the findGradeByUserId method from the grade repository to retrieve all the grades of a particular student.

- **isValidGrade (int courseId, int userId)**
  Uses the findGradeByCourseIdAndUserId method from the grade repository to check if a course with a given student exists.

- **modifyGrade (int courseId, int userId, double newScore)**
  Uses the findGradeByCourseIdAndUserId method from the grade repository to get a student's grade from a course and then modifies it using the save() method from JpaRepository

3. **Role Service**

- **getRole (String userId, String password)**
  Uses the findByUserIdAndPassword method from the role repository to return the role of the user based on the given username and password.

- **userExists (String userId)**
  Uses the findByUserId method from the role repository to check if a user exists based on their userId.

- **createNewRole (String userId, String roleName, String password)**
  Creates a new role by creating a role object and setting its parameters, then uses the save() method from JpaRepository to save it into the database.

- **deleteByUserId (int userId)**
  Uses the deleteByUserId method from the role repository to delete a role entry based on the user ID.

- **isRoleName (int userId)**
  Uses the findByUserIdAndRoleName method from the role repository to check if a user with a given user ID and role name exists in the table.

4. **User Service**

- **createNewUser (String userId, String roleName, String password)**
  Creates a new user by creating a user object and setting its parameters such as user ID, first name, and last name then uses the save() method from JpaRepository to save it into the database.

- **deleteUserById (String userId)**
  Uses the deleteByUserId method from the user repository to delete a user entry based on the user ID.

VII ANALYTICAL THINKING, CHALLENGES, SECURITY CONSIDERATION AND FUTURE WORK

Complete design, optimization, and benchmarking of a web application that performs tasks efficiently is successfully coded and presented. The work focuses on different technologies, system designs, applying security standards, and creating a user-friendly experience. Collectively, the work is presented in detailed as distinguished parts: advantages and limitations, challenges and solutions.

6.1 Assessment of Implementations: Advantages and Limitations

Various implementations come with distinct strengths and weaknesses. The choice among these implementations depends on a range of factors, including the project's deployment timeline, environment setup and configurations, complexity, desired flexibility, level of abstraction, and whether you prefer starting from scratch or seeking simplicity.

Here's a table that illustrates the diverse strengths and weaknesses associated with each implementation:

Implementation	Strengths	Weaknesses
Command Line / Sockets / JDBC Backend Implementation	Extremely flexible and powerful	• Very Complex • Tedious amount of coding • Having to implement your own synchronization criteria

		• Having to implement and create your own protocol for facilitating communication between the client and server
Traditional MVC Servlets Implementation	Suitable for rapid and temporary production needs, without any intention of future extension	• Have to handle requests and responses manually • Time consuming due to reloading , recompiling, and restarting the server
Spring MVC Implementation	• Projects are easier to extend • Smoother coding experience • No need to handle requests and responses manually • Better structured way of using Servlets and JSP	• High setting up environment and configuration time

6.2 Challenges and Solutions

Implementation of Command Line, Sockets, and JDBC Backend

During the implementation, I encountered a challenge in the communication between the client and the client handler. After the client handler responds to a message from the client, the client continues listening, unaware that the message has concluded. To address this, I devised a protocol between the client and client handler. Now, the client handler appends a notifier to its message, signaling to the client that the message has concluded. For a more ilustration of this solution, please refer to the appendices.

Traditional MVC Servlets

While working on the project, I faced a challenge in achieving proper functionality with the Tomcat server. While everything appeared to be in order and error-free, requests were not being successfully processed. After an extensive investigation, I identified the root cause: the deployed Tomcat server was incompatible with the Spring version I was utilizing. To resolve this issue, I successfully resolved the compatibility conflict by updating the Tomcat server.

Spring MVC Implementation

Among the different implementations, this one was the easiest to execute but proved challenging when it came to finding online resources. I overcame this challenge by carefully studying articles and tutorials, investing a significant amount of time and effort. However, the implementation's simplicity and straightforward approach justified the effort, highlighting the effectiveness and excellence of the Spring framework.

6.3 Security Considerations

Robust User Authentication: Utilizing Salted Hashing for Enhanced Security

Passwords must be securely stored to prevent the retrieval of the original password. One-way cryptographic hash functions are employed for this purpose, as passwords cannot be deduced from the hash alone.

Numerous hashing functions exist, yet not all are suitable for password hashing due to vulnerabilities to brute-force attacks. After thorough research, PBKDF2 and bcrypt emerged as excellent hashing algorithms, with the added advantage of being among the NIST-recommended algorithms. Consequently, the system has implemented PBKDF2 bcrypt.

When an administrator creates a user, a password is specified. This password is then hashed and stored in the role database. However, a challenge arises when users have identical passwords, resulting in the same hash codes. Exploiting this weakness, hackers can launch rainbow table attacks. To counter this threat, we employ a salt—a random string—appended to each password before hashing. This ensures unique hashes even for identical passwords, effectively mitigating the risk of rainbow table attacks.

For user authentication, once a user provides their username and password, we search for the username in the database. If the user is located, we retrieve their unique salt and then hash their password using this salt. We subsequently compare the resulting hashed password with the stored hashed password in the database. If the two match, the user is successfully authenticated. If they do not

match, the user is not authenticated. For a visual representation of this process, please refer to Figure 2.

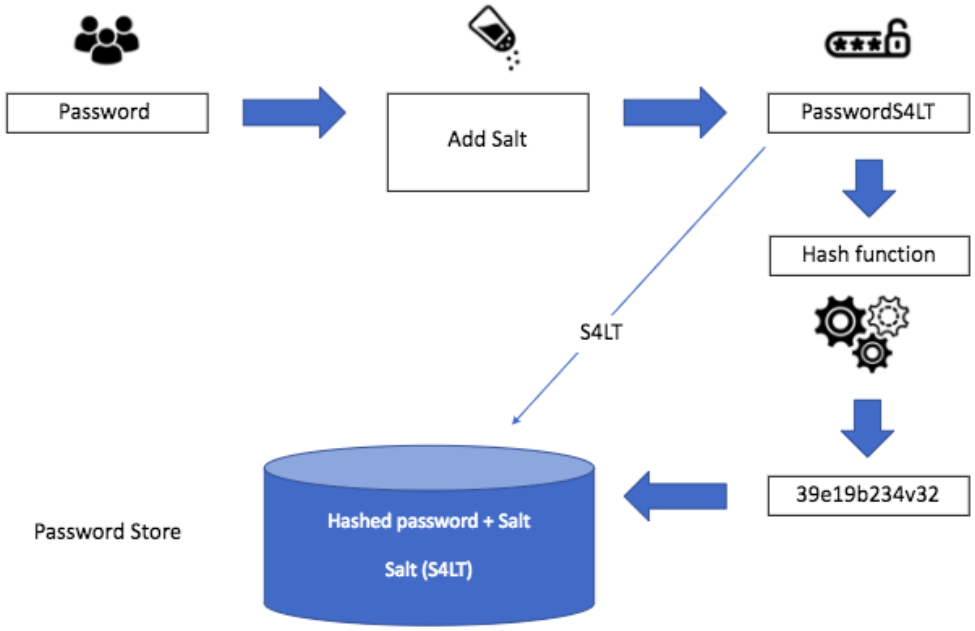

Figure 2: Hashed Passwords

Clearing Session Attributes upon Logging Out

An essential security practice involves the removal of session attributes when a user clicks the sign-out button. This action is crucial to prevent the retention of these attributes, as leaving them accessible could potentially enable unauthorized access by other users. For a detailed overview of the implementation process, please refer to the appendices.

Spring Access Control Based on Assigned Roles

A crucial security measure involves allowing access to specific pages only for users with designated roles. This is achieved by examining the user's role name before processing their request. If the role name does not correspond to the required authority, the user is redirected to the login page.

6.4 Potential Future Improvement

While the web application's structure has been established, various modifications and future enhancements are taken into consideration. These include expanding options for administrators, teachers, and students, such as allowing teachers to remove students below a certain grade from a class or enabling teachers to

request admin intervention for student removal. Another feature could be keeping track of students' marks to visualize their grades over time.

Additionally, another potential enhancement is allowing teachers the flexibility to determine how student grades are distributed within their courses. For instance, teachers could opt for a configuration involving a midterm exam, a final exam, or a combination of first, second, and final assessments. This customization empowers teachers to incorporate additional grading components like homework assignments, projects, and quizzes, offering a personalized assessment.

Furthermore, there is the possibility of introducing a parent role that permits parents to monitor their children's progress and grades. An additional enhancement involves adding a note field to student marks, exclusively viewable by parents. This feature ensures that parents can access important information without the students themselves having access.

6.5 Conclusions

Complete design, optimization, and implementation of a Student Grading System using Java Framework is successfully accomplished. The program takes the advantage of frameworks rather than using direct programming language. It turns out that these well-developed and tested Frameworks help developers to work more with less time and less effort as one does not have to write everything from scratch.

Java Frameworks offers many functional blocks that are provided with inbuilt libraries maintaining the integrity and stability of a Java application. Nowadays, Frameworks are a fundamental aspect of software development and are widely used in the Java ecosystem to improve the productivity and maintainability of software projects.

In this work, a successful development of a progressed work of a Java Student Grading System is achieved using Java Frameworks. During the course programming of the project various technologies, frameworks, and concepts that are related to backend development, web application development, and Spring MVC/Spring REST are studied, understood and fully utilized.

The successfully produced work is presented as three distinguished parts: part one, introduces Command-line / Sockets and JDBC Backend Implementation, which requires to build the initial version of the Student Grading System using the command-line interface. Part two, Web App / Traditional MVC Servlets and JSPs

Implementation, where an enhancement to the Student Grading System is carried out by transforming it into a web application using traditional MVC Servlets and JSPs. sockets, and JDBC in the backend. Finally part three, Web App / Spring MVC and Spring REST Implementation, which require to advance the Student Grading System by migrating it to a web application utilizing Spring MVC and Spring REST.

V. REFERENCES

[1] Tutorial Points, https://www.tutorialspoint.com, *What are Java Frame Works*, Aug. 2023

[2] Wikipedia , https://en.wikipedia.org, *List of Java Frame Works*, Aug. 2023.

[3] Dr. Motasem Aldiab , Student Grading System: Java Frameworks Based Application Development, Atypon Training Program 2023.

VIII. APPENDICES

```
public interface CourseRepo extends JpaRepository<Course, Integer> {

    List<Course> findCourseByTeacherId(int teacherId);

    List<Course> findCourseByCourseIdAndTeacherId(int courseId, int
teacherId);

    Optional<Course> findCourseByCourseName(String courseName);

    Optional<Course> findCourseByCourseId(int courseId);

    void deleteByCourseId(int courseId);

}

public interface GradeRepo extends JpaRepository<Grade, Integer> {

    List<Grade> findGradeByCourseId(int courseId);

    List<Grade> findGradeByUserId(int userId);

    List<Grade> findGradeByCourseIdAndUserId(int courseId, int userId);

}

public interface RoleRepo extends JpaRepository<Role, Integer> {

    Optional<Role> findByUserIdAndAndPassword(int userId, String password);

    Optional<Role> findByUserId(int userId);

    void deleteByUserId(int userId);

    Optional<Role> findByUserIdAndRoleName(int userId, String roleName);

}

public interface UserRepo extends JpaRepository<User, Integer> {

    void deleteByUserId(int userId);

}
```

```
@WebServlet("/LogoutServlet")
public class LogoutServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
IOException {
        HttpSession session = req.getSession();
        session.removeAttribute("username");
        session.removeAttribute("password");
        session.invalidate();
        res.sendRedirect("login.jsp");
    }
}
```