Abstract

This report develops a Student Grading System that is designed for universities and schools to meet their educational goal, which has an integrated system of student grade and statistic management. This system includes technologies such as sockets, JDBC, Servlets, JSPs, Spring REST, Spring MVC, Spring Security, and Thymeleaf, implemented in a phased approach to show the evolution of web apps in Java over the years and. These technologies were used to offer distinct Interfaces for each different user roles, thereby enhancing functionality and user experience. A significant feature of the system is its ability to securely display grades and statistical analyses to users using a different tech on each part, ensuring confidentiality and data integrity.

During the implementation of the third phase especially in spring boot security, Primarily due to deprecated features between the versions.

Despite these challenges, the outcomes of the project were highly positive. The final product is a secure, user-friendly Student Grading System that not only meets its intended functionalities but also showcases significant improvements in security protocols and database interaction, and it was even exceeded in the docker project.

System Design and Implementation

Part 1: CLI/Sockets and JDBC Backend

- **Database Schema Design**

The database schema for part (1,2) of the Student Grading System is designed to efficiently store and manage user data, course information, enrollments, and grades. It consists of five tables, each serving a distinct role within the system:

1. Users Tables (admins, students, and teachers): These tables are responsible for user management subsystem. The tables are similar, there is a unique id, a name and a password.
2. Courses Table: this table has all the available courses, identified by its ID, each table has a name and a teacher_id that represents the teacher for the course.
3. student_courses Table: this table tracks which student is in which course by using the student_id and the course_id, and it also has a mark column for each row to save the student mark in the course.

- **CLI Implementation**

The CLI for the Student Grading System is simple, straightforward and user-friendly. Without the complexities of a designing a graphical user interface, it focuses on functionality, presenting users with a clean, text-based interface that prompts them with clear instructions and reacts to their input efficiently. This minimalistic design ensures the CLI is accessible and easy to use from anyone.

I have hardcoded some data in the database and made an interface only for the students which includes:

1. **View Course Grades**: Students can review their grades for courses they are enrolled in.
2. **View Course:** Shows the current courses the student is involved at.
3. **View Statics**: Shows the student his Stats.
4. **Exit**: Ensures that students can exit their session safely, safeguarding their personal information.

- **JDBC Integration**

  I have used the repository design pattern in this layer, and I have used the same layer in both part one and two.
  The integration of JDBC within the Student Grading System provides an easy way and a reliable link between the Java-based CLI and the database.
  The Repository interfaces and their implementation classes and the services encapsulates all database-related operations.
  This encapsulation not only streamlines the process of database management but also enhances the security and maintainability of the system.
  Prepared statements are utilized to execute SQL commands, which helps in preventing SQL injection attacks and ensures the integrity of the database.

- **Socket Communication**

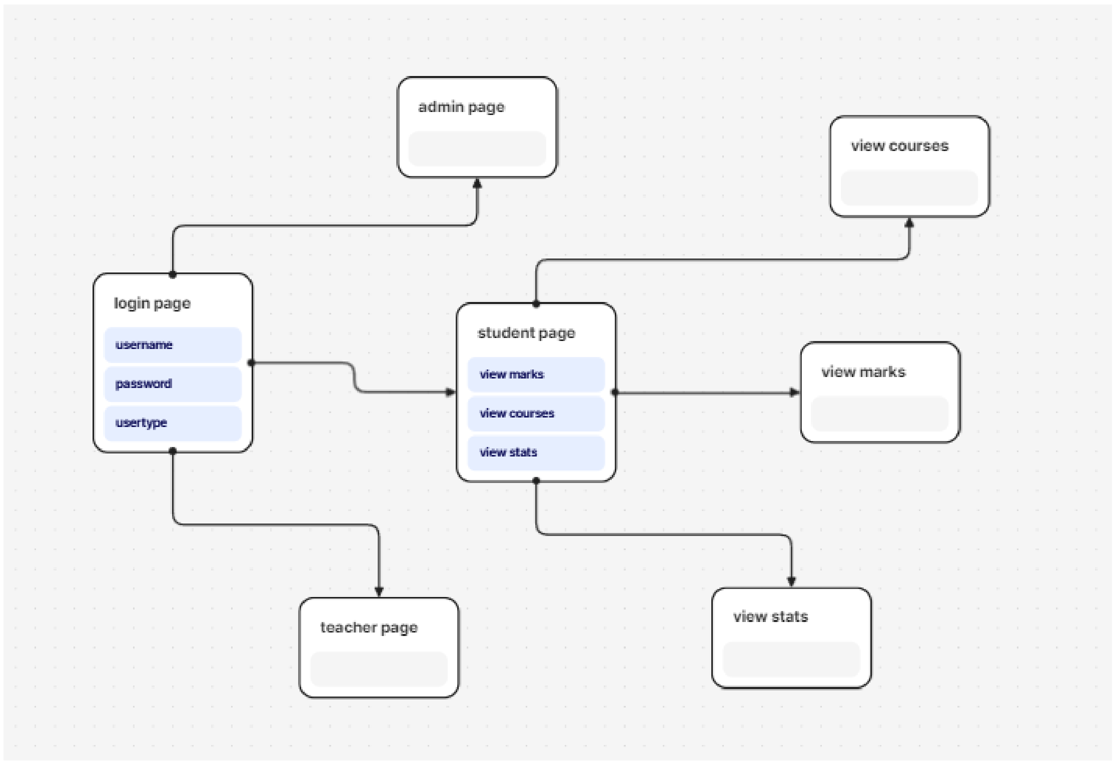
  The system adopts a client-server architecture to create a real-time interaction between the user interface and the server. Each instance running from the CLI class is treated as a client, with a dedicated server responsible for handling incoming messages. In the server side, each time a new client connects a new thread is created to handle his responses.

Part 2: MVC Servlets and JSPs Web App

- **UI Design**

The UI Design of the Student Grading System's web application uses JSPs which is a jump in technology.

After login, users reach interfaces specific to their roles— students access their courses and statistics, teachers manage grades, and admins handle user registration (I have only designed an interface for the student).
The included sitemap visually outlines this streamlined flow, showcasing the direct and distraction-free paths to each role's essential functions.



- **Servlets and JSP**

Integration Login

Servlet

The login servlet will manage user authentication and the session.
After submission of login credentials, it utilizes the Authenticate method from the userservice
To validate the user credentials, if the authentication is successful a new session get creates with some data of the user, which will enable user-specific navigation, the user will get redirected to his interface depending on his role.

Student Servlet

The student servlet will handle all the student interactions, it utilizes the user, student, course repositories and the student and user services to retrieve what the user asked for from the database.

Upon receiving a request, the servlet determines the nature of the user's request through session validation and the request parameters. If a user identified as a "student" initiates a GET request, the servlet directs them to the student-specific interface. For POST requests, the servlet processes the command specified by the user—such as viewing marks, courses, or obtaining course-specific statistics—and fetches the required data from the database using the appropriate service.

The servlet is designed to handle various student requests:

1. **Viewing Marks**: If the option to view marks is selected, the servlet retrieves the student's grades for all enrolled courses, formatting this information for display on the student marks page.
2. **Viewing Courses**: When opting to view courses, it lists all courses the student is enrolled in, providing detailed information such as course IDs and names.
3. **Getting Course Maxima**: A specialized option may direct the student to a page showing maximum marks or other statistics for a course.

Errors or unauthorized access attempts are handled gracefully, redirecting users to an error page or a welcome page as appropriate.

Max in Course Servlet

The Max In Course Servlet is specifically designed to handle requests related to fetching the states from a specific course. This servlet leverages the capabilities of the StudentService, which interacts with both the StudentRepository and CourseRepository to access the required data. Upon receiving a POST request, this servlet extracts the course ID from the request parameters and invokes the **getMaxInCourse** method from the StudentService, then forwards the request to a JSP page (**maxInCourseResult.jsp**) for displaying the result.

Part 3: Spring MVC and Spring REST Web App

- **Spring Configuration**

The Spring configuration for the Student Grading System is built around the Spring Boot framework, which provides a comprehensive platform to develop Java-based applications with minimal setup.

Maven Setup

The project is managed using Maven**,** with spring-boot-starter-parent as the parent dependency in the POM file ensuring standardized dependency management and plugin configurations. The key dependencies includes Spring Boot Starters for security, JPA, Theymleaf, mysql connector, Lombok,

Web and Devtools.

These starters bundle the necessary dependencies to integrate security, database operations, view templates, and web functionality into the application.

Application Properties

The properites file specifies the database connection and the ddl-auto settings.

Project Structure

The project follows a typical Spring Boot structure with packages organized by feature and type, including:

config, controllers, services, entity, dto, repository and security.

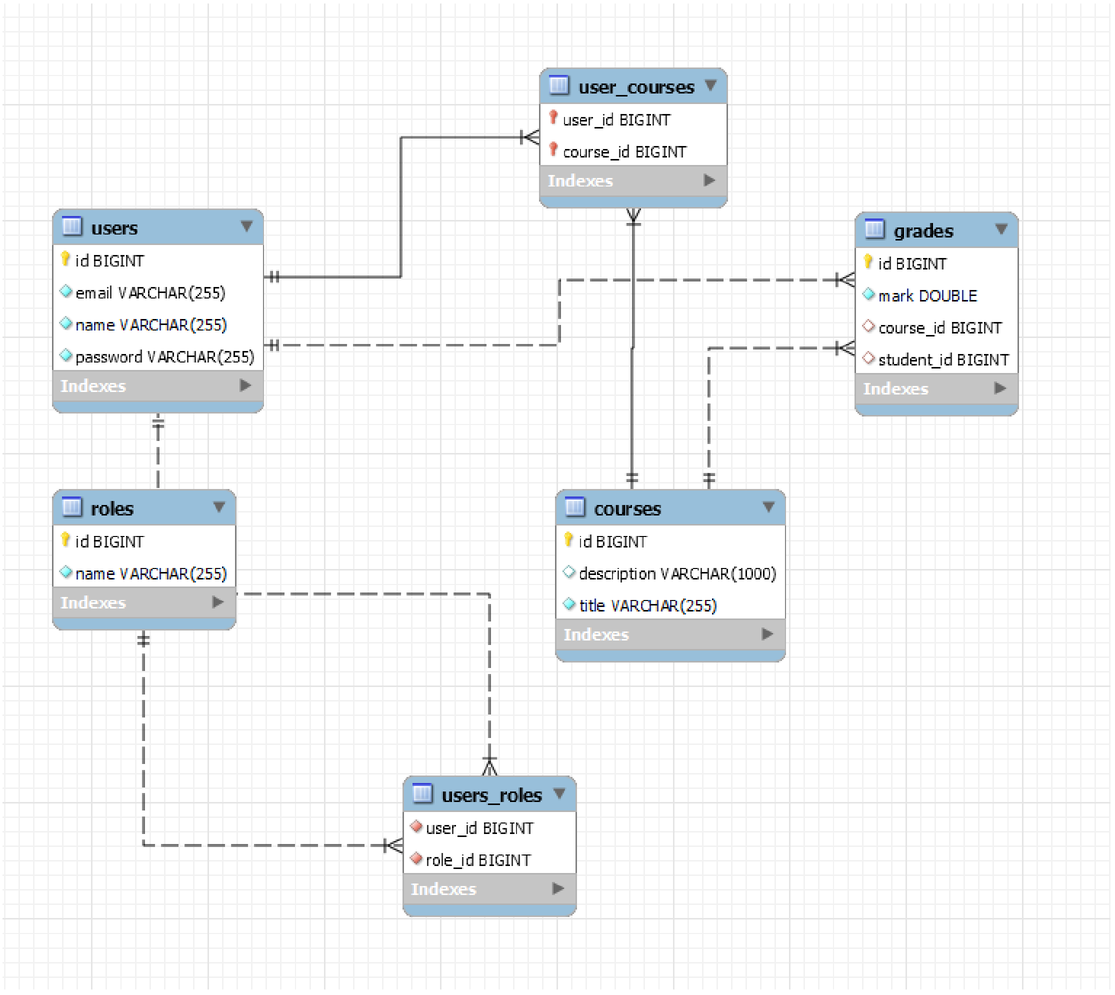
This design pattern enhances separation of concerns.

Controller Implementation

Jumping from servlets to Spring boot controllers involved the use of spring boot approach of handling HTTP requests and responses. Spring controllers use annotations to map specific URLs to Java methods and rely on Dependency Injection to handle business logic, leading to a more modular and declarative style of web application development.
This jump also achieved a next level of security and performance.

Changes in the database

In this part I have redesigned the database to a more flexible designed, the designed is well shown in the next pic:



Dependency Injection

I have used spring's Dependency Injection to access the repositories and services which reduce the coding time and simplify the look of the code.

Views and Thymeleaf Integration

The shift from JSP to Thymeleaf for rendering views is also facilitated
By spring boot MVC. Thymeleaf templates are returned by the controllers as the html file name, and the data get passed to them throw the model attribute

Enhanced Database Interactions

Moving to Spring Data JPA has refined the way controllers communicate with the database. By using repositories, we've adopted a more sophisticated method for database interaction. This approach not only streamlines the process but also reduces the likelihood of errors, offering a marked improvement over the direct JDBC usage in servlet-based setups.