ATYPON

Student Grading System Assignment

Made by: Samer AlHamdan

Instructors:
Motasim Aldiab
Fahed Jubair

1. Introduction

This report offers a comprehensive analysis for the Student Grading System that progresses through three stages of implementation. during Atypon's Training in May 2023. a multi-stage project that transforms a simple command-line application into a sophisticated web-based platform utilizing modern Java technologies. This documentation will guide you through the three key stages of this assignment, each representing a significant step towards creating a more accessible, user-friendly, and efficient system for managing student grades.

2. The Assignment

The assignment encompasses three distinct stages in the development of a comprehensive student grading system.

Stage 1: Command-Line Interface, Sockets, and JDBC Backend

In Stage 1, I constructed the project's foundation by developing a command-line interface (CLI) for the Student Grading System. Here, we delve into the integration of Java Sockets to enable communication between clients and the server. Moreover, the JDBC framework is harnessed to facilitate seamless interaction with a relational database, forming a resilient backend structure. Core features, including student registration, course management, and grade allocation, are meticulously crafted, setting the stage for future enhancements.

Stage 2: Traditional MVC Servlets and JSPs

In Stage 2, I thrust the project into the web realm by adopting the traditional Model-View-Controller (MVC) architecture. Servlets are strategically employed as controllers, intelligently routing requests and responses. Dynamic content presentation is realized through JavaServer Pages (JSPs), introducing an elegant separation of concerns and enhancing the user interface. This stage witnesses the conversion of the system into a web application, reinforcing its accessibility and visual appeal.

Stage 3: Spring MVC and Spring REST Integration

In Stage 3, a transformative shift is orchestrated as I integrate the Spring Framework into the architecture. Spring MVC emerges as the foundational pillar, streamlining the management of controllers and introducing heightened flexibility. The project's backend attains elevated robustness with the assimilation of Spring Boot, facilitating the creation of resilient RESTful services for frictionless communication between the frontend and backend. This stage culminates in the attainment of a contemporary, high-performance web application, poised to adeptly serve a diverse spectrum of users.

3. Database

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

1. **Student Table**:
    - This table stores information about students who use the Grading System.
    - Columns:
        - `Student_ID`: A unique identifier for each student (varchar, max length 10). This is the primary key of the table.
        - `Student_Email`: The email address of the student (varchar, max length 30).
        - `Student_Password`: The password associated with the student's account (varchar, max length 15).
        - `Student_Name`: The name of the student (varchar, max length 50).
        - `Major`: The major or field of study of the student (varchar, max length 35).
2. **Course Table**:
    - This table stores information about the courses offered in the Grading System.
    - Columns:
        - `Course_ID`: A unique identifier for each course (varchar, max length 10). This is the primary key of the table.

        - `Course_Name`: The name of the course (varchar, max length 30).

3. **StudentCourseRelation Table**:
    - This table establishes a many-to-many relationship between students and courses, indicating which student is enrolled in which course and their performance in the course.
    - Columns:
        - `Student_ID`: A foreign key reference to the `Student` table, indicating which student is enrolled in the course (varchar, max length 10).
        - `Course_ID`: A foreign key reference to the `Course` table, indicating which course the student is enrolled in (varchar, max length 10).
        - `Mark`: The student's mark or score in the course (int).
    - Constraints:
        - The combination of `Student_ID` and `Course_ID` forms a unique constraint, ensuring that a student cannot be enrolled in the same course more than once.
        - Foreign key constraints ensure referential integrity by linking the `Student_ID` and `Course_ID` columns to their respective tables.

4. Stage 1

This stage encompasses two primary classes: the "Server" and the "Client," alongside an additional pair of classes dedicated to data management for students and courses. These auxiliary classes are designed to facilitate the retrieval and storage of student and course data, contributing to the overall functionality and organization of the system.

3.1 Server

The `Server` class is a crucial component of the student grading system. It functions as a server-side application that handles client connections, authentication, and data retrieval. The class communicates with clients over sockets, processes instructions, interacts with a MySQL database using JDBC, and provides essential data for student grading and course information.

Design:

1. **Constructor (**`Server(SeverSocket socket)`**)**
    - Initializes essential components for the server, including the server socket, database connection, input/output streams, and statement object.
    - Establishes a connection to the MySQL database using JDBC.
    - Listens and waits for a client connection using the provided server socket.
2. `executeInstructions()` **Method**
    - Executes a continuous loop that processes incoming instructions from clients until the socket is closed.
    - Reads the instruction sent by the client using the `bufferedReader`.
    - Based on the instruction, it invokes the appropriate methods such as `login()` or `getCourseData()`.
3. `isExist(String email, String password)` **Method**
    - Performs a query to check if a given email and password combination exists in the database.
    - Iterates through the result set of the query to compare the provided credentials with stored data.
    - Returns `true` if a matching record is found, indicating successful authentication.
4. `getData(String email)` **Method**
    - Retrieves data related to a specific student from the database, including the student's courses and grades.
    - Constructs a list of `Course` objects based on course-related data retrieved from the database.
    - Creates and returns a `Student` object containing the student's general information and the list of courses.
5. `login()` **Method**
    - Handles the login process by reading the email and password provided by the client.
    - Calls the `isExist()` method to check the validity of the login credentials.
    - Sends a response to the client indicating whether the login was successful.
    - If login is successful, it sends the corresponding student data using the `objectOutputStream`.
6. `getCourseData(String inst)` **Method**
    - Extracts the course ID from the provided instruction.
    - Executes a query to retrieve marks associated with the specified course from the database.

- Calculates various statistics such as minimum, maximum, average, and median marks based on the retrieved data.
- Sends these calculated statistics back to the client using the `printWriter`.

7. `main(String[] args)` **Method**
   - Initializes the server socket on a specified port and creates an instance of the `Server` class.
   - Initiates the `executeInstructions()` method, starting the process of handling client requests.
8. **Database Connection and Communication:**
   - The class establishes a connection to a MySQL database using JDBC, enabling communication with the database for authentication and data retrieval.
   - Utilizes various input/output streams and communication channels to facilitate real-time communication between the server and clients over sockets.

Security Measures:

1. **Database Connection and SQL Injection Prevention**:
   - The code establishes a connection to the database using JDBC. However, it is recommended to use connection pooling for better resource management and security.
   - Prepared statements are not used in the current code. Using prepared statements can help prevent SQL injection attacks by automatically escaping user input.
2. **Authentication**:
   - The `isExist` method checks if a user with a given email and password exists in the database. This helps authenticate users before allowing access.
   - Authentication is a basic security measure, but it's important to implement a more secure authentication mechanism, such as using hashed and salted passwords, to protect sensitive user credentials.
3. **Output Encoding**:
   - The code uses `printWriter` to send data to the client. Ensure that proper output encoding is applied to prevent XSS attacks when rendering user-generated content on the client-side.

3.2 Client

The `Client` class is a significant component of the student grading system, representing the client-side application. It enables users to interact with the server by sending requests, receiving responses, and displaying relevant information to the user. The class communicates with the server using sockets and provides a user-friendly interface for accessing user information, enrolled courses, and course-related statistics.

Design:

1. **Constructor (`Client(Socket socket)`):**
   - Initializes the `Client` object and establishes a connection to the server using the provided socket.
   - Initializes input/output streams and communication channels with the server using `InputStreamReader`, `ObjectInputStream`, `PrintWriter`, and `BufferedReader`.
   - Set up a `Scanner` to read user input from the console.
   - Initializes the `isLoggedIn` flag to `false`.
2. `getInstructions()` **Method:**
   - Enters a loop that displays available instructions to the user and waits for user input.
   - Allows the user to view their user information, enrolled courses, or sign out from their account.
   - Exits the loop when the user chooses to sign out.
3. `login()` **Method:**
   - Handles the login process by repeatedly prompting the user to enter their email and password.
   - Sends the "login" instruction, email, and password to the server for authentication.
   - Receives a response from the server indicating whether the login was successful (`isExist`).
   - If the login is successful, deserializes and stores the received `Student` object.
4. `start()` **Method:**
   - Enters a loop that continues until the client socket is closed.
   - If the user is not logged in, it prompts the user to log in using the `login()` method.
   - Once logged in, allows the user to access different instructions using the `getInstructions()` method.

5. `userInformation()` **Method:**
    - Displays user information such as name, email, major, and enrolled courses.
    - Uses the stored `student` object to retrieve and display the user's data.
6. `userCourses()` **Method:**
    - Displays the user's enrolled courses and their corresponding marks.
    - Allows the user to select a course by its number to view additional course information.
    - Sends a request to the server for course statistics (minimum, maximum, average, and median marks).
    - Displays the received course statistics to the user.
7. `main(String[] args)` **Method:**
    - Establishes a socket connection to the server running on the specified host and port (localhost, 33333).
    - Creates an instance of the `Client` class, initiating the interaction with the server.
    - Displays a welcome message and starts the interaction by invoking the `start()` method.
8. **Communication with Server:**
    - The class communicates with the server through sockets, sending instructions, data, and receiving responses.
    - Utilizes input/output streams to facilitate communication and data exchange.
    - Manages user interactions by displaying menus, reading user input, and presenting information.

Security Measures:

1. **User Authentication:**
    - The client code implements user authentication by sending the user's email and password to the server for verification.
    - The server responds with a Boolean value indicating whether the user's credentials are valid.
    - This authentication process helps prevent unauthorized access to the application.
2. **Input Validation:**
    - The client code does not directly process user input from the console, which reduces the risk of certain types of attacks like SQL injection and command injection.
    - The input from the console is used to trigger specific actions in the application (e.g., logging in, retrieving course information).
3. **Session Management:**

    - The client maintains a logged-in session status using the `isLoggedIn` variable. This prevents unauthorized access to certain functionalities without proper authentication.

3.3 Models

The `Course` and `Student` classes are essential components of the student grading system, representing the data structure for courses and students, respectively. They are designed to store and manage information related to courses, students, and their relationships. Both classes implement the `Serializable` interface to enable object serialization, which is crucial for transferring these objects between the client and server over network communication.

1. **`Course` Class:**
    - Represents a course within the grading system.
    - Contains instance variables to store the course ID (`id`), course name (`name`), and the student's mark (`mark`) in the course.
    - Provides getter and setter methods for each instance variable.
    - Overrides the `toString()` method to provide a textual representation of the course's attributes.
    - Includes a constructor that initializes the course attributes when creating a `Course` object.
2. **`Student` Class:**
    - Represents a student within the grading system.
    - Contains instance variables to store the student's ID (`id`), email address (`email`), name (`name`), major (`major`), and a list of enrolled courses (`courses`).
    - Provides getter and setter methods for each instance variable.
    - Overrides the `toString()` method to provide a textual representation of the student's attributes, including their enrolled courses.
    - Includes a constructor that initializes the student's attributes, including the list of courses, when creating a `Student` object.
3. **Serialization:**
    - Both `Course` and `Student` classes implement the `Serializable` interface, which allows instances of these classes to be converted into a stream of bytes.
    - Serialization is essential for transmitting these objects between the client and server over network communication.

    - Serialization enables efficient storage and transmission of complex object structures.
4. **Relationship Between Classes:**
    - The `Student` class contains a list of `Course` objects (`courses`), representing the courses in which a student is enrolled.
    - This relationship allows for a student's course information to be stored and retrieved as a part of the student's data.
5. **Usage in the System:**
    - The `Course` and `Student` classes are used in the `Server` class for representing course and student data retrieved from the database.
    - The `Client` class uses the `Course` and `Student` classes to deserialize and display information received from the server.
6. **Benefits:**
    - The `Course` and `Student` classes encapsulate relevant data and provide a structured way to represent and manage course and student information.
    - Serialization enables seamless exchange of course and student data between the client and server, facilitating efficient communication and information sharing.

5. Stage 2

In this stage we augment the grading system's capabilities by leveraging the MVC design pattern through the utilization of Servlets and JavaServer Pages (JSPs). This stage encompasses four key components of significance: the Data Access Objects (DAOs), service layers, servlets, and models.

4.1 DAOs

The `CourseDao` and `LoginDao` classes are DAOs. They encapsulate the database interactions required for retrieving course and student data, as well as handling user authentication. By using the DAO pattern, my application can maintain a clear separation between its business logic and the complexities of working with the database. This separation enhances maintainability, reusability, and testability of my codebase.

1. `CourseDao` **Class:**

- Represents the Data Access Object (DAO) for handling course-related database operations.
- Establishes a database connection to the MySQL server on localhost using the provided credentials.
- Contains a constructor that initializes the database connection and creates a `Statement` object for executing SQL queries.
- Provides two main methods:
  - `getData(int courseId)`: Retrieves the marks of students enrolled in a specific course identified by `courseId`.
  - `getCourseData(int courseId)`: Retrieves all data related to a specific course identified by `courseId`.

2. `LoginDao` **Class:**
   - Represents the DAO responsible for managing user authentication and student data retrieval from the database.
   - Establishes a database connection to the MySQL server on localhost using the provided credentials.
   - Contains a constructor that initializes the database connection and creates a `Statement` object for executing SQL queries.
   - Provides two main methods:
     - `isExist(String email,String password)`: Checks whether a user with a given email and password exists in the `Student` table. Returns a boolean indicating the existence of the user.
     - `getStudent(String email)`: Retrieves detailed information about a student with the given email, including their enrolled courses and associated marks. Constructs and returns a `Student` object.
3. **Method Details:**
   - `CourseDao`:
     - The `getData(int courseId)` method retrieves the marks of students enrolled in a specific course from the `studentcourserelation` table.
     - The `getCourseData(int courseId` method retrieves all data related to a specific course from the `course` table.

   - `LoginDao`:
     - The `isExist(String email,String password` method checks whether a student with the provided email and password

exists in the `Student` table using a prepared statement. It returns a boolean indicating whether the user exists.

        - The `getStudent(String email)` method retrieves detailed student information using a prepared statement. It fetches the student's enrolled courses, marks, and other details from the `studentcourserelation`, `student`, and `course` tables. It constructs and returns a `Student` object.

4. **Key Points:**
    - Both `CourseDao` and `LoginDao` classes encapsulate database operations, shielding the rest of the application from direct interaction with the database.
    - Prepared statements are used to prevent SQL injection and improve security.
    - The `LoginDao` class handles user authentication and gathers student-related information from multiple tables to construct a comprehensive `Student` object.
5. **Security Measures:**
    - Connection Management:Proper connection management is implemented using try-with-resources statements to ensure that database connections are properly closed after usage. This helps prevent resource leaks and ensures efficient use of connections.
    - Parameterized Queries:The DAO classes use parameterized queries for interacting with the database. This prevents SQL injection attacks by properly escaping and sanitizing user inputs before they are used in queries.
    - Sensitive Data Protection:Sensitive data such as passwords is never stored in plain text in the code. Instead, sensitive data is stored securely and retrieved as needed during connection setup.
    - Prepared Statements:Prepared statements are used for executing SQL queries that involve user inputs. This helps prevent SQL injection by separating the query structure from the user input data.
    - Limited Scope of Variables:Variables, such as `ResultSet` and `PreparedStatement`, have limited scopes within try-with-resources blocks. This prevents accidental reuse of these variables and ensures that resources are properly closed.
    - Data Validation:User inputs, such as email and password, are used as query parameters after being validated and sanitized. This helps prevent malicious input from being executed as part of database queries.
    - Use of Object-Oriented Principles:The DAO classes encapsulate database operations within methods, promoting code reusability and maintainability. This reduces the risk of code duplication and potential security flaws.

- Secure Exception Handling:Proper exception handling is implemented, which helps prevent sensitive information leakage and provides appropriate error messages without revealing internal details.

4.2 Services

Service classes in the context of the MVC (Model-View-Controller) architecture are responsible for encapsulating the application's business logic. They act as intermediaries between the web controllers (servlets) and the data access layer (DAOs). The service classes contain methods that implement specific operations required by the application. Let's delve into the details of the `CourseService` and `LoginService` classes:

CourseService: The `CourseService` class is responsible for providing business logic related to courses.

- `getCourse(int courseId)`: Retrieves course data based on the provided `courseId` by delegating to the `CourseDao`. It creates a new `Course` object and returns it.
- `getAvg(int courseId)`: Calculates and returns the average mark of students enrolled in the specified course. It retrieves the data from the database using `CourseDao`.
- `getMin(int courseId)`: Calculates and returns the minimum mark among students enrolled in the specified course. It retrieves the data from the database using `CourseDao`.
- `getMax(int courseId)`: Calculates and returns the maximum mark among students enrolled in the specified course. It retrieves the data from the database using `CourseDao`.
- `getMedian(int courseId)`: Calculates and returns the median mark of students enrolled in the specified course. It retrieves the data from the database using `CourseDao`, sorts the marks, and calculates the median.

LoginService: The `LoginService` class handles business logic related to user authentication and retrieval of user data.

- `validateUser(String user, String password)`: Validates a user's login credentials by checking if a matching record exists in the database using the `LoginDao`. Returns `true` if the user is valid, otherwise `false`.

- `getStudent(String email)`: Retrieves student data (including enrolled courses) based on the provided `email` by delegating to the `LoginDao`. Returns a `Student` object representing the user.

4.3 Servlets

These two servlet classes, `CourseServlet` and `LoginServlet` are responsible for receiving user requests, interacting with the appropriate service classes (`CourseService` and `LoginService`) to process and retrieve data, and then directing the response to the appropriate JSP pages for rendering. They play a crucial role in the Model-View-Controller (MVC) architecture, ensuring a clear separation of concerns between handling user interactions, business logic, and UI presentation. Let's dive into more detail about what each class does:

CourseServlet:

Methods:

1. `doGet(HttpServletRequest request, HttpServletResponse response)`: This method is called when an HTTP GET request is made to the `/course` URL. It forwards the request to a JSP page (`/jsp/course.jsp`), which is responsible for rendering the course information to the user interface.
2. `doPost(HttpServletRequest request, HttpServletResponse response)`: This method is called when an HTTP POST request is made to the `/course` URL. It retrieves the `courseId` parameter from the request, fetches course-related data using the `CourseService`, such as course details, average, minimum, maximum, and median marks. Then, it sets these attributes in the request scope and forwards the request to the same JSP page (`/jsp/course.jsp`) for rendering

Security Measures:

1. Input Validation: The `doPost` method validates the input parameter `courseId` before converting it to an integer. This helps prevent potential SQL injection attacks by ensuring that only valid input is used in the SQL query.
2. Exception Handling: Proper exception handling is implemented to catch and handle any `SQLException` that might occur during database operations. This prevents the application from crashing and provides better error messages to users.

3. Request Forwarding: The use of `request.getRequestDispatcher().forward()` to forward the request and response objects to the JSP page is a common practice. This helps maintain separation between servlet logic and presentation, making the code more maintainable.

LoginServlet:

Methods:

1. `init()`: This method is called during servlet initialization and creates an instance of `LoginService` to be used for handling user login and validation.
2. `doGet(HttpServletRequest request, HttpServletResponse response)`: Similar to the `CourseServlet`, this method handles HTTP GET requests to the `/login` URL. It forwards the request to a JSP page (`/jsp/login.jsp`), which is responsible for rendering the login form.
3. `doPost(HttpServletRequest request, HttpServletResponse response)`: This method is called when an HTTP POST request is made to the `/login` URL. It retrieves the `name` (email) and `password` parameters from the request, validates the user's credentials using the `LoginService`, and depending on the result:
    - If the credentials are valid, it fetches student information using the `LoginService`, sets the `student` attribute in the request scope, and forwards the request to a welcome page (`/jsp/welcome.jsp`) to greet the user.
    - If the credentials are invalid, it sets an error message attribute (`errorMessage`) in the request scope and forwards the request back to the login page (`/jsp/login.jsp`).

Security Measures:

1. Input Validation: The `name` and `password` parameters obtained from the request are used in the authentication process. The code should validate and sanitize these inputs before using them to prevent potential security vulnerabilities like SQL injection.
2. Password Validation: The servlet communicates with the `LoginService` to validate user credentials against the database. The use of parameterized queries

(`PreparedStatement`) helps protect against SQL injection by automatically escaping input parameters.
3. Error Handling: The servlet handles potential errors gracefully by forwarding the user to an error page with appropriate error messages. This prevents exposing sensitive information and provides a better user experience.
4. Session Management: The servlets don't directly implement session management, but typically, the web application's session management mechanism ensures that user sessions are managed securely, preventing session fixation and session hijacking attacks.
5. Initialization and Resource Management: The servlets initialize their respective service classes during initialization (`init` method) to ensure that resources are properly managed and connections are established securely.

4.4 Models

These two model classes, `Course` and `Student`, define the structure and attributes of the data entities that your web application will work with. They represent the core data elements and encapsulate the information related to courses and students. Let's explore each class in detail

Course: This class represents a course entity. It has the following attributes:

- `id`: An integer representing the unique identifier of the course.
- `name`: A string representing the name of the course.
- `mark`: An integer representing the mark or grade associated with the course. This attribute might be nullable (`Integer`) to handle cases where marks are not applicable or not yet assigned.

The class also provides constructors, getters, and setters for these attributes. Additionally, it overrides the `toString()` method to provide a human-readable string representation of a `Course` object.

Student: This class represents a student entity. It has the following attributes:

- `id`: An integer representing the unique identifier of the student.
- `email`: A string representing the email address of the student.
- `name`: A string representing the name of the student.

- `major`: A string representing the major or field of study of the student.
- `courses`: A list of `Course` objects representing the courses that the student is enrolled in.

The class provides constructors, getters, and setters for these attributes. It also overrides the `toString()` method to provide a human-readable string representation of a `Student` object.

4.5 JSPs

The three JSP (JavaServer Pages) files I have are responsible for rendering the user interface of my Grading System application. JSP files allow you to embed Java code within HTML content, enabling dynamic generation of web pages. Let's take a closer look at each JSP file:

1. **course.jsp**:
   - This JSP file is used to display information about a specific course.
   - It imports the `Course` model class to access its attributes.
   - The page layout consists of a header with the title "Grading System" and a logout button. The course information is displayed in a structured format.
   - The course's name, average, median, maximum, and minimum values are displayed using the values retrieved from the servlet's attributes.
   - A "Go Back" button allows the user to navigate back to the previous page.
   - JavaScript code is used to implement the `goBack()` function, which utilizes the browser's history to navigate back.
2. **login.jsp**:
   - This JSP file is responsible for rendering the login page of the Grading System application.
   - It imports the `Course` and `Student` model classes, but it seems that these imports are not used in this JSP.
   - The layout includes a "Grading System" header, a login form with fields for email and password, and a "Login" button.
   - If there is an error message (e.g., invalid credentials), it's displayed below the login form in red.
   - The styling provides a modern and clean appearance for the login page.
3. **welcome.jsp**:
   - This JSP file displays a welcome page for the logged-in student.
   - It imports the `Course` and `Student` model classes.

- The page layout includes a "Grading System" header, the student's name, email, major, and a list of courses they are enrolled in.
- For each course, a button displays the course name, and the associated mark is displayed below.
- The information is retrieved from the servlet's attributes and displayed in a visually appealing format.

6. Stage 3

5.1 DAOs

They are identical to the DAOs used in 4.1.

5.2 Services

Both service classes are similar to the classes used in 4.2, they encapsulate the business logic of their respective functionalities and interact with the DAO classes (`CourseDao` and `LoginDao`) to access and manipulate data from the database. The use of Spring annotations (`@Service`) indicates that these classes are intended to be managed as Spring beans, making them suitable candidates for dependency injection and inversion of control. The Lombok annotations are used for constructor injection and logging, which simplifies the code and improves readability.

CourseService:

- This service class is responsible for providing methods related to courses, such as retrieving course information and calculating statistics like average, minimum, maximum, and median marks for a given course.
- The class is annotated with `@Service`, indicating that it's a Spring service component.
- It uses constructor injection (`@RequiredArgsConstructor`) to inject an instance of `CourseDao` into the service.
- The constructor initializes the `CourseDao` instance, and if an exception occurs during initialization, it throws a `RuntimeException` wrapping the caught exception.

- `getCourse(int courseId)`: Retrieves course data based on the provided course ID. It uses `CourseDao` to fetch course details from the database.
- `getAvg(int courseId)`: Calculates and returns the average mark for the specified course. It uses `CourseDao` to retrieve marks and then calculates the average.
- `getMin(int courseId)`: Calculates and returns the minimum mark for the specified course. It uses `CourseDao` to retrieve marks and iterates through the results to find the minimum.
- `getMax(int courseId)`: Calculates and returns the maximum mark for the specified course. It uses `CourseDao` to retrieve marks and iterates through the results to find the maximum.
- `getMedian(int courseId)`: Calculates and returns the median mark for the specified course. It uses `CourseDao` to retrieve marks, sorts the marks, and then calculates the median.

LoginService:

- This service class handles user authentication and retrieval of student information.
- It is also annotated with `@Service` and uses constructor injection to inject an instance of `LoginDao`.
- Similar to `CourseService`, the constructor initializes the `LoginDao` instance and handles potential exceptions.
- `validateUser(String user, String password)`: Validates user credentials by checking if the provided email and password match an entry in the database. It uses the `LoginDao` to perform the database query.
- `getStudent(String email)`: Retrieves student information based on the provided email. It uses the `LoginDao` to fetch student details from the database.

5.3 Controllers

In the context of the Grading System application, the controllers play a crucial role in handling incoming HTTP requests, processing the requested actions, and returning appropriate responses. Controllers serve as the bridge between the user interface (front-end) and the business logic (back-end) of my application. Controllers are a key component of the Model-View-Controller (MVC) design pattern, which is widely used in web development to separate concerns and ensure a clear structure for building applications.

CourseController:

- This controller is responsible for handling requests related to courses, such as displaying course information and statistics.
- It contains methods that correspond to different actions related to courses, such as retrieving course data and rendering the course page.
- The methods are annotated with `@GetMapping` and `@PostMapping` annotations to specify the HTTP methods (GET and POST) that they handle.
- Constructor injection is used to inject an instance of the `CourseService` class, which contains the business logic for handling course-related operations.
- The methods extract data from the HTTP request (parameters or form data), invoke methods from the `CourseService` to perform necessary calculations or data retrieval, and then set attributes in the request for use in the view (JSP page).
- After processing the request, the methods forward the request to the appropriate JSP page using `request.getRequestDispatcher().forward()`.
- Security Measures:
    - Input Validation**:** In the `getCourse` method, the input parameter `courseId` obtained from the request is used. Similar to the previous code, this input should be validated and sanitized to prevent potential security vulnerabilities like SQL injection.
    - Exception Handling: Proper exception handling is implemented to catch and handle any `SQLException` that might occur during database operations. This prevents the application from crashing and provides better error messages to users.
    - Request Forwarding: The use of `request.getRequestDispatcher().forward()` to forward the request and response objects to the JSP page is a common practice. This helps maintain separation between controller logic and presentation, making the code more maintainable.

LoginController:

- This controller manages user authentication and interactions related to user login.
- It contains methods for displaying the login page and processing login attempts.
- Similar to the `CourseController`, methods are annotated with `@GetMapping` and `@PostMapping` to handle different types of requests.
- The `LoginService` is injected through constructor injection to handle user authentication and retrieval of student information.
- The `showLoginPage` method displays the login page by forwarding the request to the corresponding JSP page.

- The `login` method processes login attempts by extracting user credentials from the request, validating them using the `LoginService`, and responding accordingly based on the validation result. It sets attributes in the request to provide feedback to the user (success or error messages) and forwards the request to appropriate JSP pages.
- Security Measures:
  - Input Validation: The `name` and `password` parameters obtained from the request are used in the authentication process. input validation and sanitization are crucial to prevent security vulnerabilities.
  - Password Validation: The controller communicates with the `LoginService` to validate user credentials against the database. The use of parameterized queries (`validateUser` method) helps protect against SQL injection.
  - Error Handling: The controller handles potential errors gracefully by forwarding the user to an error page with appropriate error messages. This prevents exposing sensitive information and provides a better user experience.
  - Logging: The `log` instance (from Lombok's `@Slf4j` annotation) can be used to log relevant information, errors, and warnings. Proper logging is essential for security monitoring and debugging.

5.4 Models

These two model classes are also similar to the model classes used in 4.4, `Course` and `Student`, define the structure and attributes of the data entities in my Grading System application. They represent the core data objects that your application works with.

Course:

- This class represents a course in your grading system.
- It implements the `Serializable` interface, which allows instances of this class to be serialized and deserialized, making them suitable for network communication and storage.
- The `@Getter` annotation generates getter methods for the private fields, providing access to their values.
- It has the following attributes:
  - `id`: An `Integer` representing the unique identifier of the course.
  - `name`: A `String` representing the name of the course.
  - `mark`: An `Integer` representing the mark or score associated with the course (can be `null`).

- The constructor takes three arguments: the course ID, name, and mark. These values are used to initialize the corresponding attributes.
- The `toString()` method provides a human-readable string representation of a `Course` object, including its ID, name, and mark.

Student:

- This class represents a student in your grading system.
- Like the `Course` class, it implements the `Serializable` interface and uses the `@Getter` annotation to generate getter methods for its fields.
- It has the following attributes:
    - `id`: An `int` representing the unique identifier of the student.
    - `email`: A `String` representing the email address of the student.
    - `name`: A `String` representing the name of the student.
    - `major`: A `String` representing the major or field of study of the student.
    - `courses`: A `List<Course>` representing the list of courses that the student is enrolled in.
- The constructor takes five arguments: the student's ID, email, name, major, and a list of courses. These values are used to initialize the corresponding attributes.
- The `toString()` method provides a human-readable string representation of a `Student` object, including its ID, email, name, major, and the list of courses.

5.5 JSPs

They are identical to the JSPs used in 4.5.