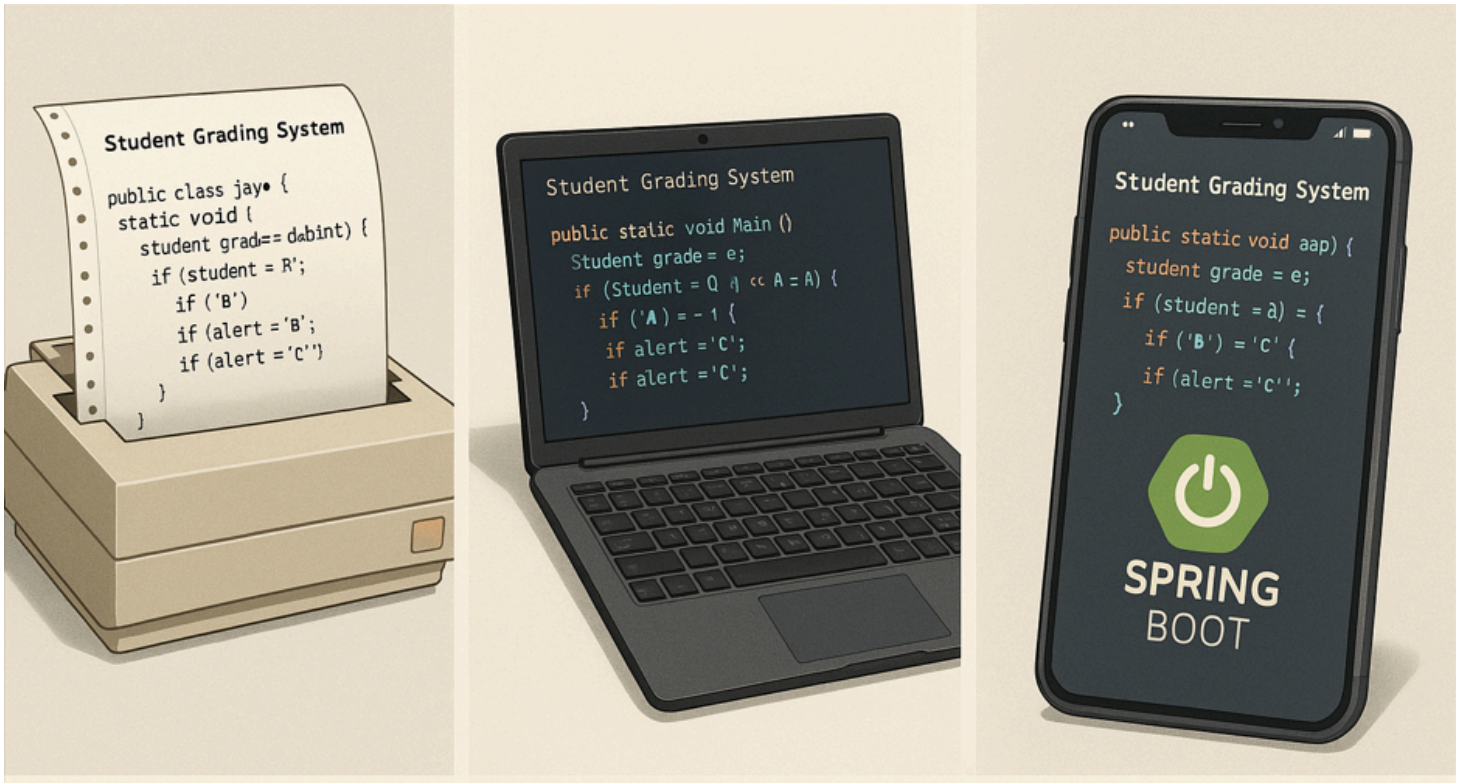


Student Grading System Report



By Hamza Alhalabi

Table of Contents

- [Student Grading System Report](#)
 - [Table of Contents](#)
 - [Introduction](#)
 - [System Design](#)
 - [Database Schema](#)
 - [System Architecture](#)
 - [Implementation](#)
 - [Command-line / Sockets and JDBC Backend](#)
 - [Web App \(Traditional MVC - Servlets and JSPs\)](#)
 - [Web App \(Spring MVC and Spring REST\)](#)
 - [Challenges and Solutions](#)
 - [Future Enhancements](#)
 - [Conclusion](#)

Introduction

This Student Grading System is an educational project designed to facilitate the management of student grades across multiple courses with multiple teachers. The system aims to provide a seamless experience for three distinct user roles: students, instructors, and administrators. Students can view their enrolled courses and grades, instructors can manage course grades, and administrators can perform system-wide management tasks such as user and course creation.

This project has been implemented in three progressive stages (three versions), each building upon the previous one to showcase different architectural approaches and technologies:

1. Command-line interface with Socket-based client-server architecture and `JDBC` for database operations
2. Traditional Web Application using `Servlets` and `JSPs` with `MVC` pattern
3. Modern Web Application utilizing `Spring MVC` and `REST APIs`

Welcome, Ruzain

Menu:

[View Courses](#)

[View Course Grades](#)

[Edit Student Grade](#)

[View Course Statistics](#)

[Logout](#)

Instructor Dashboard

Welcome to your instructor dashboard. Use the menu to select an option.

Select Course to View Grades

Course Name	Action
Introduction to Computer Science	View Grades

System Design

Database Schema

The system uses a relational database MySQL with the following key entities:

- **Users:** Stores authentication details and role information (Student, Instructor, Admin)
- **Courses:** Contains course information including name and assigned instructor
- **Enrollments:** Manages **many-to-many** relationships between students and courses
- **Grades:** Stores grade information for each student in each enrolled course

```
12 > CREATE TABLE users ( ...
17 );
18
19 > CREATE TABLE courses ( ...
24 );
25
26 > CREATE TABLE enrollments ( ...
32 );
33
34 > CREATE TABLE grades ( ...
40 );
```

The database schema supports the relationships needed to track all users operations across all implementation stages.

System Architecture

Each implementation stage follows a distinct architectural approach:

1. CLI Implementation:

- Client-Server architecture using Java Sockets

```
5 public class Client {
9     public static void main(String[] args) {
16         try {
17             socket = new Socket(HOST, PORT);
18             out = new PrintWriter(socket.getOutputStream(), autoFlush: true);
19             in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
20             userInput = new Scanner(System.in);
21
22             System.out.println("Connected to server at " + HOST + ":" + PORT);
```

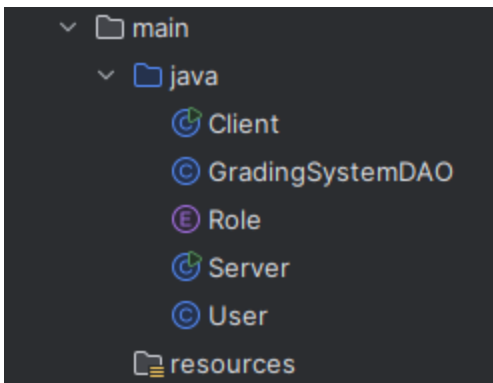
- Multi-threaded server using ExecutorService for concurrent client connections

```

7   public class Server {
11  ▶   public static void main(String[] args) {
12      try (ServerSocket serverSocket = new ServerSocket(PORT)) {
13          System.out.println("Server is running on port " + PORT);
14
15          while (true) {
16              Socket clientSocket = serverSocket.accept();
17              System.out.println("New client connected: " + clientSocket.getInetAddress());
18
19              // New thread to handle each client connection
20              threadPool.submit(new ClientHandler(clientSocket));
21          }

```

- DAO pattern for database operations
- Console-based UI



1. Web Application (MVC with Servlets/JSPs):

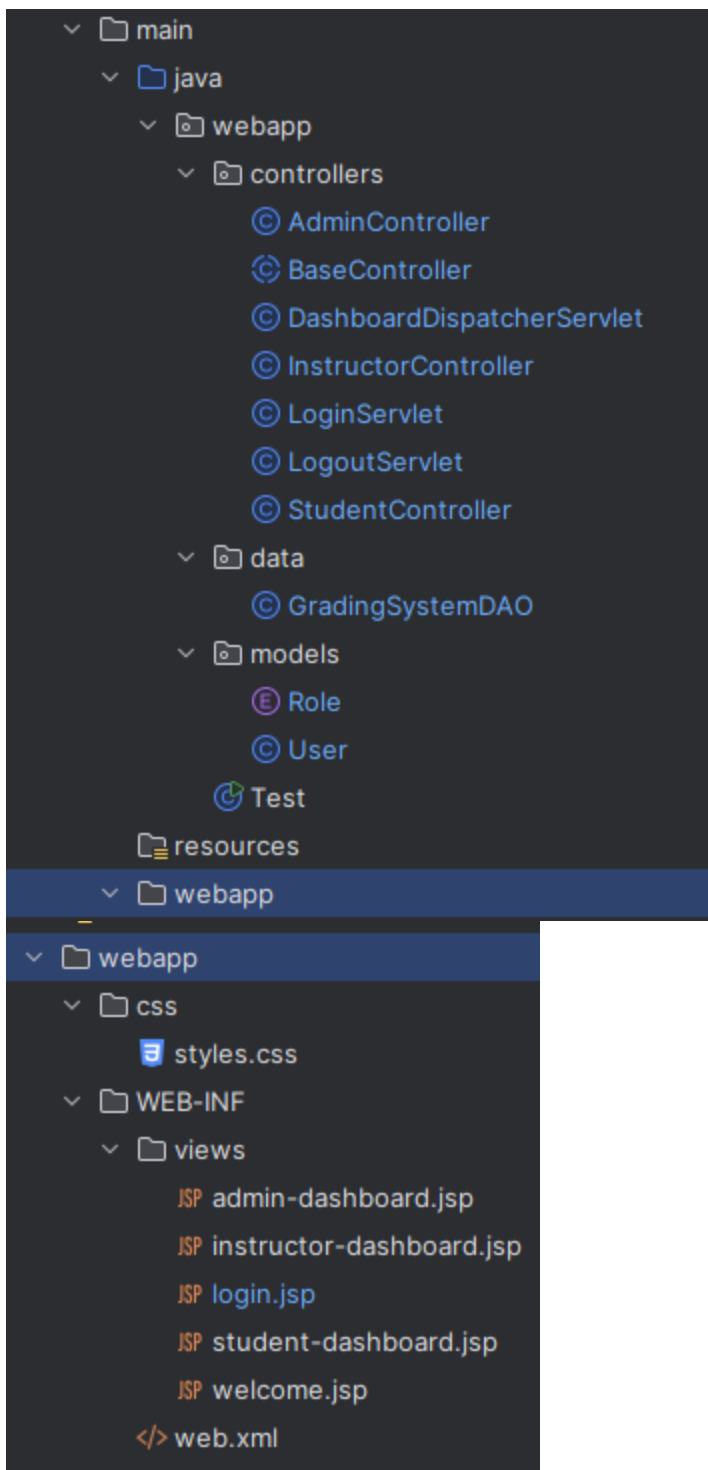
- Traditional MVC architecture

```

13   @WebServlet("/login")
14   public class LoginServlet extends HttpServlet {
15       private GradingSystemDAO gradingDAO;
16
17       public void init() {
18           gradingDAO = new GradingSystemDAO();
19       }

```

- Servlets as controllers, JSPs as views



- DAO pattern for data access

```
14 > public GradingSystemDAO(){...}
23
24 @ > private DataSource getDataSource() throws SQLException {...}
35
```

- HTTP Session-based authentication

```

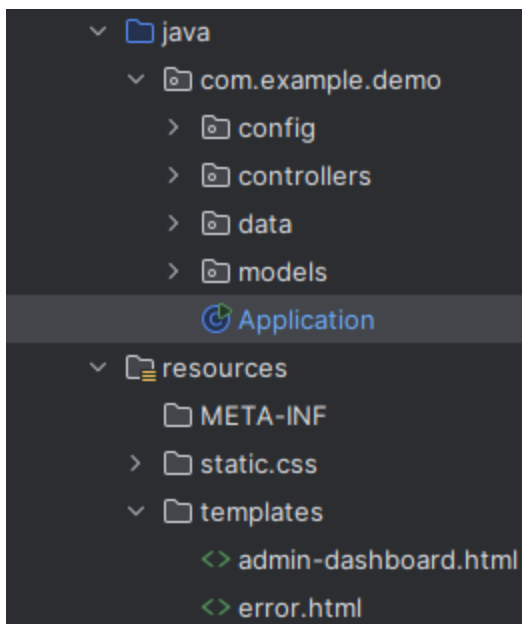
32
33         if (gradingDAO.validateUser(username, password)) {
34             HttpSession session = request.getSession(b: true);
35
36             // 30 minutes session timeout
37             session.setMaxInactiveInterval(30 * 60);
38
39             // Get the User object
40             webapp.models.User user = gradingDAO.getUser(username, password);
41
42             // Store user information in session
43             session.setAttribute(s: "username", username);
44             session.setAttribute(s: "userRole", user.getRole().toString());

```

- HTML / CSS for frontend

2. Spring Implementation:

- Spring MVC architecture



- Annotation-based configuration

```

7
8     @SpringBootApplication
9     public class Application extends SpringBootServletInitializer {
10

```

```

15     @Repository
16     public class GradingSystemDAO {
17         private final JdbcTemplate jdbcTemplate;
18
19         @Autowired
20         public GradingSystemDAO(JdbcTemplate jdbcTemplate) {
21             this.jdbcTemplate = jdbcTemplate;
22         }

```

- Controller-Service-Repository pattern
- Thymeleaf templates for views

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <title>Student Dashboard</title>
5     <link rel="stylesheet" th:href="@{/css/styles.css}">
6 </head>
7 <body>
8     <div class="container">
9         <h2 th:text="'Welcome, ' + ${session.username}"></h2>
10
11         <div class="menu">
12             <h3>Menu:</h3>
13             <ul>
14                 <li><a th:href="@{/student/courses}">View Courses</a></li>
15                 <li><a th:href="@{/student/grades}">View Grades</a></li>
16                 <li><a th:href="@{/student/statistics}">View Course Statistics</a></li>
17                 <li><a th:href="@{/logout}">Logout</a></li>
18             </ul>
19         </div>
```

- RESTful API endpoints

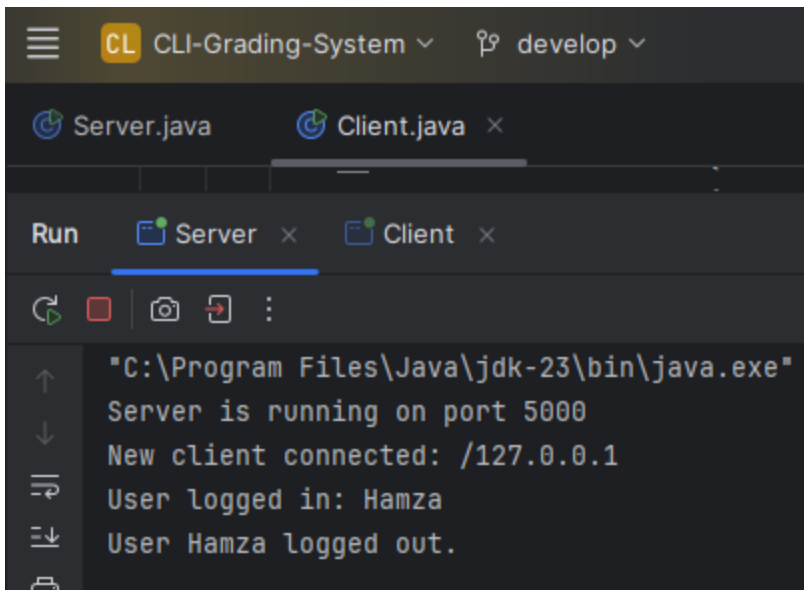
```
56
57 // API endpoint for login
58 @PostMapping("/login/api")
59 @ > public ResponseEntity<Map<String, Object>> loginApi(@RequestBody Map<String, String> credentials) {...}
60 }
```

Implementation

Command-line / Sockets and JDBC Backend

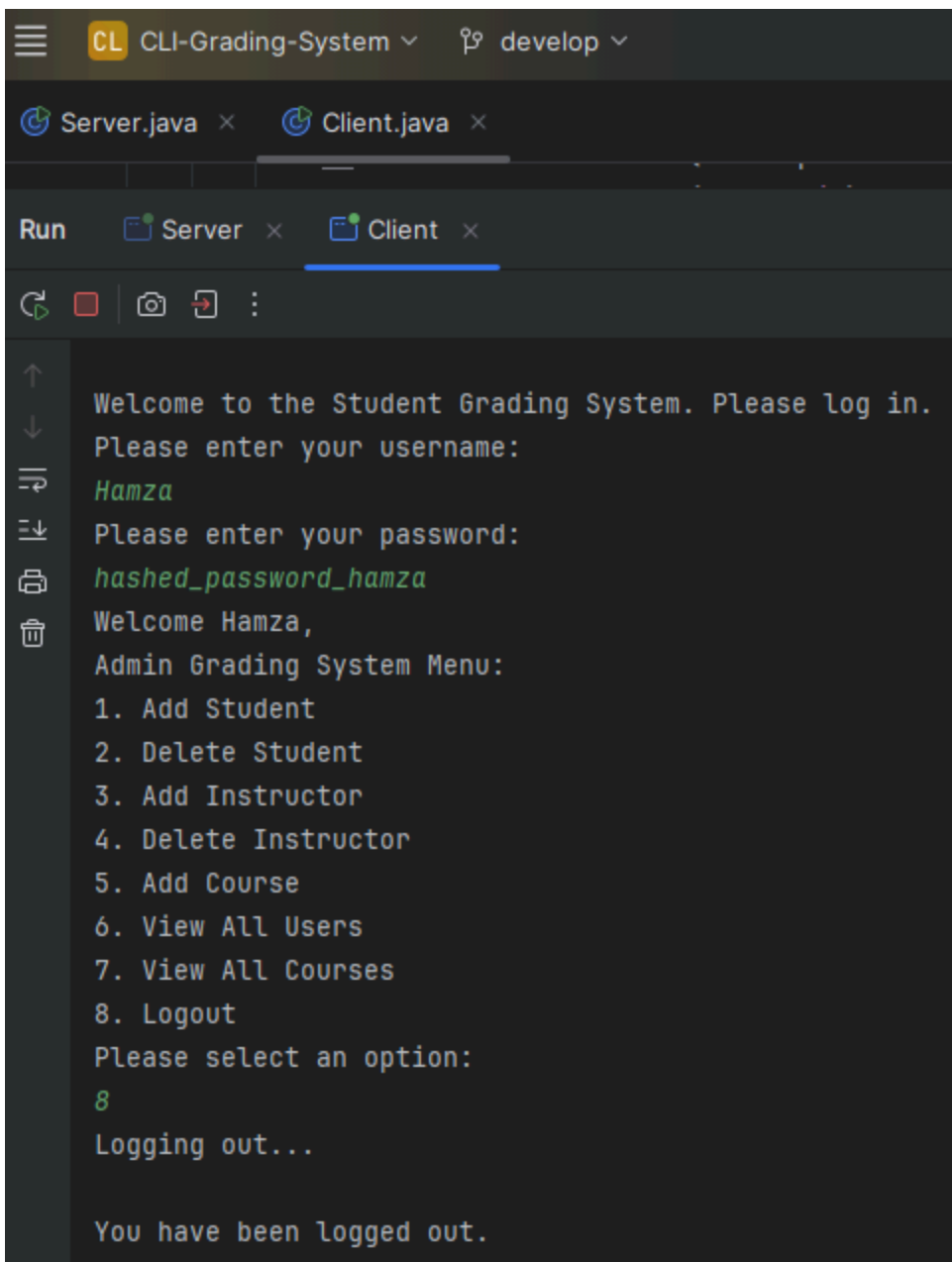
The first implementation uses a client-server architecture with Java Sockets for communication:

- **Server:** A multi-threaded server that listens for client connections and handles requests through a thread pool



```
CL CLI-Grading-System develop
Server.java Client.java
Run Server Client
"C:\Program Files\Java\jdk-23\bin\java.exe"
Server is running on port 5000
New client connected: /127.0.0.1
User logged in: Hamza
User Hamza logged out.
```

- **Client:** A simple console application that connects to the server and handles user input/output



```
CL CLI-Grading-System develop
Server.java Client.java
Run Server Client
Welcome to the Student Grading System. Please log in.
Please enter your username:
Hamza
Please enter your password:
hashed_password_hamza
Welcome Hamza,
Admin Grading System Menu:
1. Add Student
2. Delete Student
3. Add Instructor
4. Delete Instructor
5. Add Course
6. View All Users
7. View All Courses
8. Logout
Please select an option:
8
Logging out...
You have been logged out.
```


- **Authentication:** Basic username and password authentication
- **DAO Layer:** Uses traditional JDBC with prepared statements for secure database operations
- **User Interface:** Text-based menus with role-specific functionality
- **Key Features:**
 - Role-based access control

```

7      public class Server {
30          private static class ClientHandler implements Runnable {
40              public void run() {
44                  // ...
45                  do {
46                      while (user == null){
47                          loginMenu(in, out);
48                      }
49                      switch (user.getRole()) {
50                          case STUDENT:
51                              out.println("Welcome " + user.getUsername() + ",");
52                              studentMenu(in, out);
53                              break;
54                          case INSTRUCTOR:
55                              out.println("Welcome " + user.getUsername() + ",");
56                              instructorMenu(in, out);
57                              break;
58                          case ADMIN:
59                              out.println("Welcome " + user.getUsername() + ",");
60                              adminMenu(in, out);
61                              break;
62                      }

```

- Student course and grade viewing
- Instructor grade management
- Administrator user and course management

Web App (Traditional MVC - Servlets and JSPs)

The second implementation evolves the system into a web application:

- **Controllers:** Servlet-based controllers for handling HTTP requests

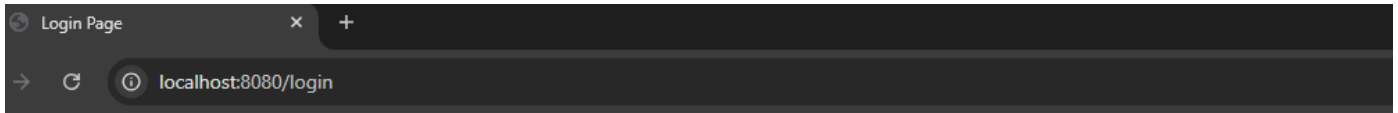
```

117 @      Shows form to add a new student
118      private void showAddStudentForm(HttpServletRequest request, HttpServletResponse response)
119          throws ServletException, IOException {
120          request.setAttribute("operation", "addStudent");
121          request.getRequestDispatcher("/WEB-INF/views/admin-dashboard.jsp").forward(request, response);
122      }

```

- **Views:** JSP pages with JSTL for dynamic content
- **Authentication:** Session-based authentication with role-based access control

- **DAO Layer:** Enhanced JDBC implementation with connection pooling
- **User Interface:** Responsive web UI



Web-App Login

Username:

Password:

Login

- **Key Features:**
 - Web-based interface accessible by browser
 - Enhanced UI with tables and forms
 - Session management (a session is 30 minutes)
 - More intuitive navigation (as user interface)
- We can **run** the app using this command: `mvn tomcat7:run`

Web App (Spring MVC and Spring REST)

The third implementation leverages the Spring framework for a modern approach:

- **Controllers:** Spring MVC controllers with annotation-based mappings

```

27     @GetMapping("/login")
28     > public String showLoginPage() { return "login"; }
31
32     @PostMapping("/login")
33     public String login(@RequestParam String username,
34                        @RequestParam String password,
35                        HttpSession session,
36     >                        Model model) {...}

```

- **Views:** Thymeleaf templates with Spring integration
- **Authentication:** With role-based authorization
- **DAO Layer:** Spring JDBC Template for simplified data access

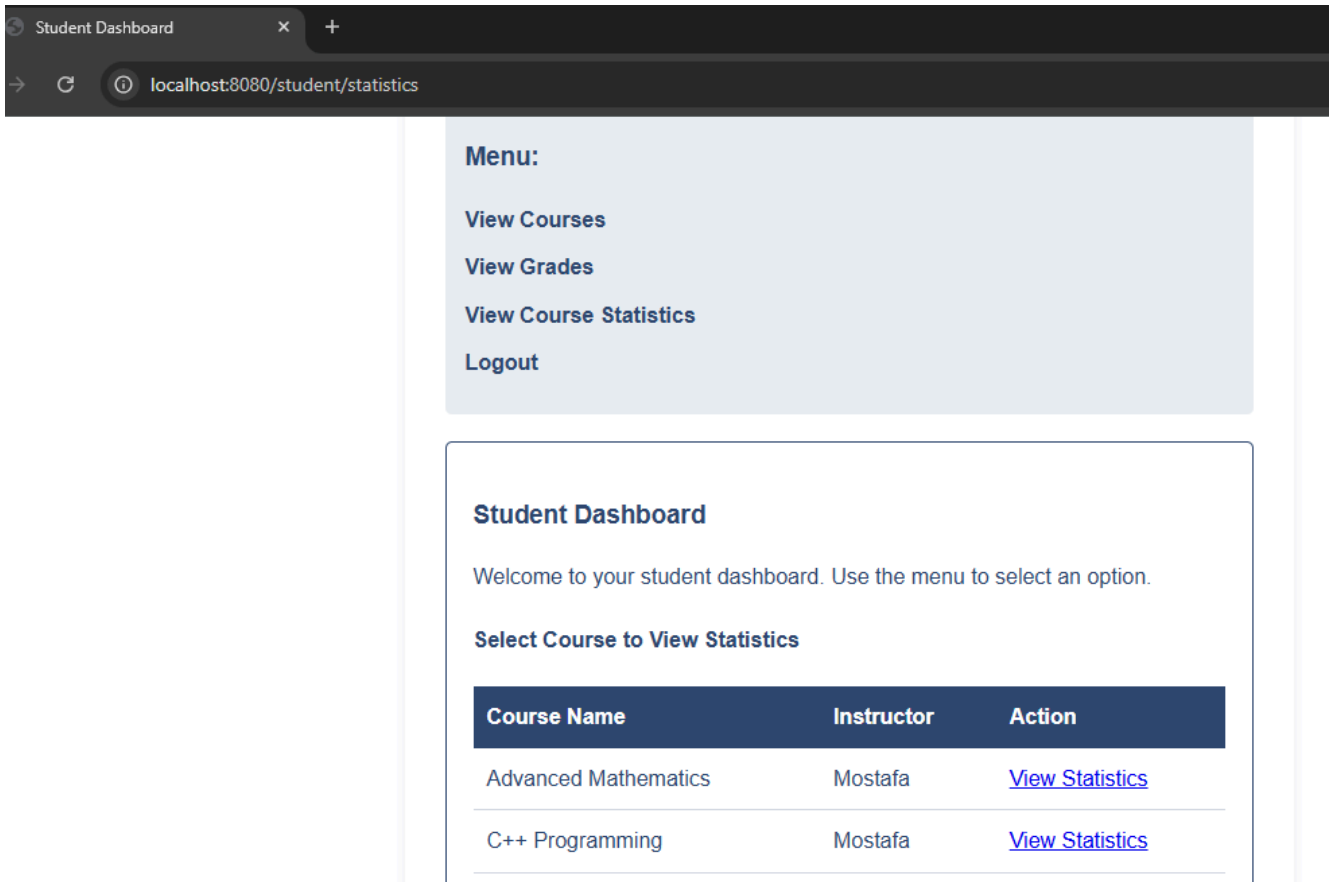
- **REST API:** RESTful endpoints for programmatic access
- **User Interface:** Enhanced responsive UI with CSS
- **Key Features:**
 - Dependency injection

```

21     @Autowired
22     public LoginController(GradingSystemDAO gradingDAO) {
23         this.gradingDAO = gradingDAO;
24     }

```

- Declarative transaction management
- Stronger separation of concerns
- API access alongside web interface



- We can **run** the app by running the `Application.java` file

Challenges and Solutions

1. Concurrency Management

- **Challenge:** Handling multiple client connections in the Socket implementation
- **Solution:** Implemented thread pool with `ExecutorService` to manage concurrent client sessions

2. Session Management

- **Challenge:** Maintaining user state across HTTP requests
- **Solution:** Implemented session-based authentication in web implementations

3. Managing Dependencies

- **Challenge:** Setting up the dependencies of JSP in Spring Boot
- **Solution:** Migrated to Thymeleaf as it is more modern and reliable

4. Surprising Errors

- **Challenge:** For example, I faced an error related to missing a dependency when running Server class in the first version
- **Solution:** I was running it with **Maven** not from the command line alone

5. Changing Controllers

- **Challenge:** Changing controllers type between the Servlet web app and Modern MVC controllers
- **Solution:** Used another example and followed it step by step

Future Enhancements

1. Enhanced Authentication

- Implement password hashing

2. Make DAO Singleton

- I felt like the DAO instance in all of the three projects needs to be a singleton instance?

3. Advanced Features

- Assignment submission and grading
- Grade analytics and reporting
- Course material management

4. Technical Improvements

- Microservices architecture for better scalability
- More Containerization with Docker
- CI/CD pipeline for automated deployment
- Comprehensive test suite with both unit and integration tests

5. UI/UX Enhancements

- Real-time updates with WebSockets
- Enhanced data visualization

Conclusion

This project demonstrates a progressive approach to software development, evolving from a simple command-line application to a fully-featured web application with modern architecture. Each implementation stage builds upon the previous one, applying new technologies and patterns while maintaining the core functionality.

This evolution shows the benefits of modern frameworks in reducing boilerplate code and enhancing maintainability. The project also highlights the importance of proper software design principles across different architectural styles.

It was my first time dealing with databases in Java and using Spring framework, and it was easier than what I expected.