

# **Grading System Web Assignment**

Atypon training program

Done by: Ammar Abuhsukar

# Table of Contents

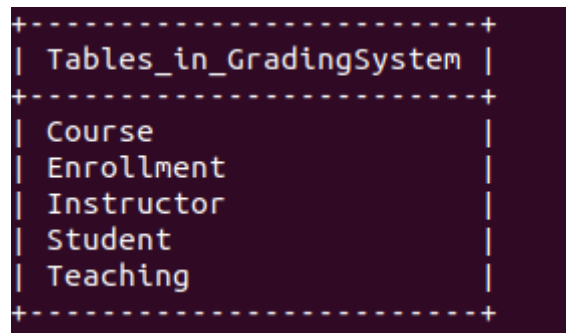
Objective.....	3
Database.....	4
Data Access Objects.....	6
Part one.....	7
GradingServer:.....	7
Main method.....	7
Handling clients.....	7
Student.....	7
Instructor.....	7
Admin.....	7
Communication.....	7
Exception Handling.....	8
Student Client.....	8
Authentication.....	8
Menu.....	8
Viewing enrolled courses.....	8
Input validation.....	8
Instructor Client.....	8
Authentication.....	8
Handling Courses.....	9
Menu.....	9
Adding marks.....	9
Input validation.....	9
Admin Client.....	9
Authentication.....	9
Menu.....	9
Student Registration.....	9
Input validation.....	9
Summary.....	10
Part Two.....	10
Controller(Servlets).....	10
Login Servlets.....	10
Profile servlets.....	10
Action Servlets.....	10
Session handling.....	11
Logout servlets.....	11
Custom error pages.....	11
Model.....	11
Views(JSPS).....	12
Input validation and exception handling.....	12
Summary.....	12
Part Three.....	13
Controllers.....	13
Services.....	13
Model.....	13
Views.....	13
DAOs.....	13
Security.....	13
Summary.....	13
.....	14

## Objective

The objective of this project is to build a comprehensive student grading systems using three technologies to understand the evolution of server client programming. The first stage is building the system using sockets programming, the second is built using servlets and jsps, and the last stage is implemented using spring boot technology. All three stages use the same database therefore this report will start from there.

# Database

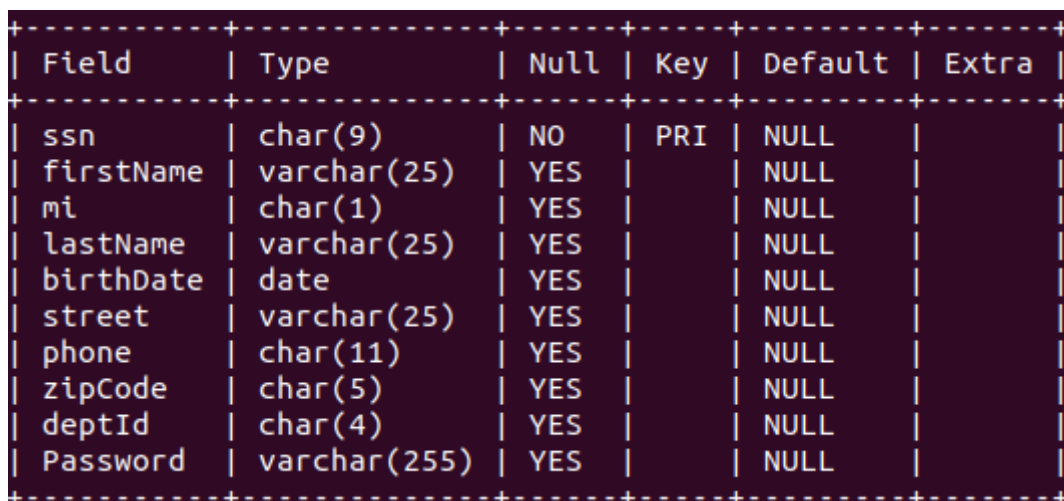
The database for this project is a relational database using MySQL database management system. There are five tables in the system: Course, Enrollment, Instructor, Student, Teaching.



Tables_in_GradingSystem
Course
Enrollment
Instructor
Student
Teaching

Figure 1: Tables

- Student: This table holds information about individual students identified by their student number (ssn).

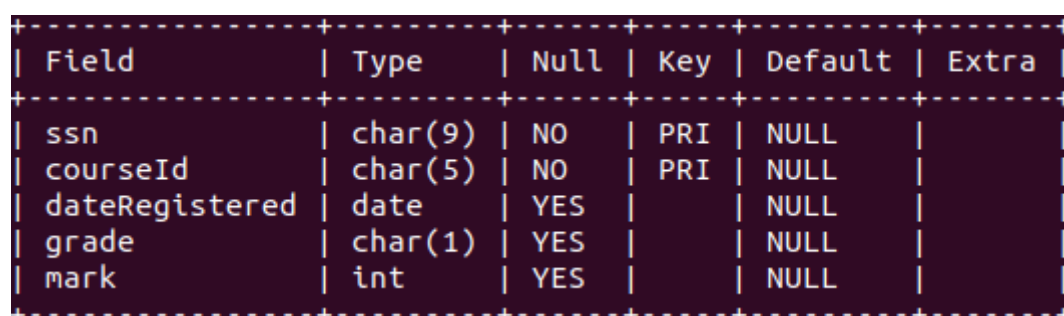


Field	Type	Null	Key	Default	Extra
ssn	char(9)	NO	PRI	NULL	
firstName	varchar(25)	YES		NULL	
mi	char(1)	YES		NULL	
lastName	varchar(25)	YES		NULL	
birthDate	date	YES		NULL	
street	varchar(25)	YES		NULL	
phone	char(11)	YES		NULL	
zipCode	char(5)	YES		NULL	
deptId	char(4)	YES		NULL	
Password	varchar(255)	YES		NULL	

Figure 2: Student table

- Enrollment: This table tracks the enrollment details of students in courses. It has composite primary key consisting of the student's number (ssn) and the Course ID (courseId). The ssn field in the Enrollment table is a foreign key referencing the ssn field in the Students table. The courseId field in the Enrollment table is a foreign key referencing the courseId field in the Course table.

This table establishes a many-to-many relationship between students and courses, as a student can be enrolled in multiple courses, and a course can have multiple enrolled students.



Field	Type	Null	Key	Default	Extra
ssn	char(9)	NO	PRI	NULL	
courseId	char(5)	NO	PRI	NULL	
dateRegistered	date	YES		NULL	
grade	char(1)	YES		NULL	
mark	int	YES		NULL	

Figure 3: Enrollment

- Course: This table contains information about individual courses, identified by the Course ID (courseId) which serves as the primary key.

Field	Type	Null	Key	Default	Extra
courseId	char(5)	NO	PRI	NULL	
subjectId	char(4)	NO		NULL	
courseNumber	int	YES		NULL	
title	varchar(50)	NO		NULL	
numOfCredits	int	YES		NULL	

Figure 4: Course

- Instructor: This table stores information about instructors, identified by their Instructor ID (isn), which serves as the primary key.

Field	Type	Null	Key	Default	Extra
isn	char(9)	NO	PRI	NULL	
firstName	varchar(25)	YES		NULL	
mi	char(1)	YES		NULL	
lastName	varchar(25)	YES		NULL	
birthDate	date	YES		NULL	
phone	char(11)	YES		NULL	
deptId	char(4)	YES		NULL	
instructorRank	varchar(25)	YES		NULL	
Password	varchar(255)	YES		NULL	

Figure 5: Instructor table

- Teaching: This table represents the association between instructors and the courses they teach. It has a composite primary key consisting of the instructor's ID (isn) and the Course ID (courseId). The isn field is a foreign key referencing the isn field in the Instructor table. The courseId field is a foreign key referencing the courseId field in the Course table. This table establishes a many-to-many relationship between instructors and courses, as an instructor can teach multiple courses, and a course can be taught by multiple instructors.

Field	Type	Null	Key	Default	Extra
isn	char(9)	NO	PRI	NULL	
courseId	char(5)	NO	PRI	NULL	

Figure 6: Teaching table

All the passwords stored in the database are stored in a hashed way using SHA-256 algorithm.

## Data Access Objects

Another aspect that is shared between all the three techniques is the DAOs. Each doa has an interface and an implementation in MySQL, this way if the database is changed, it's easy to change other things by simply implementing the interface with the new database. Here is a diagram that describes the various DAOs used and their interfaces:

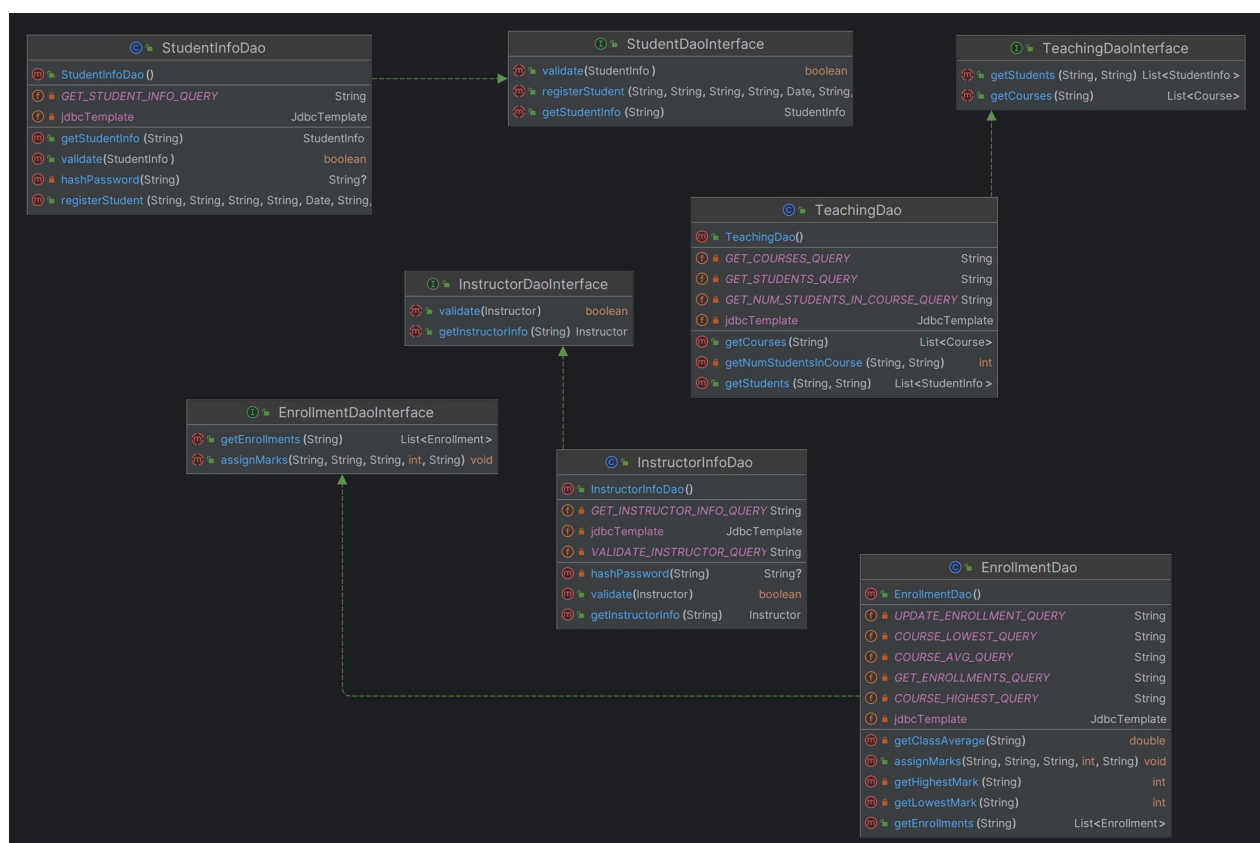


Figure 7: DAOs

## Part one

This part describes the implementation of the first technique that utilizes server client socket programming.

### GradingServer:

Let's begin with the server, The server sets up a ServerSocket on port 8000 to listen for incoming connections. It utilizes multi-threading by initializes a fixed-size thread pool (ExecutorService) to handle multiple client connections concurrently.

### Main method

The main method continuously accepts incoming connections and spawns a new thread to handle each client.

### Handling clients

This method is responsible for processing client requests. It reads the type of client (student, instructor, or administrator) from the client's input stream. It then goes to different methods to handle each user type differently

### Student

If the client is identified as a student, the server expects the student's (SSN) and password. It validates the credentials using a StudentInfoDao and communicates the authentication result back to the client. Upon successful authentication, it provides options to the student, such as retrieving enrollments or exiting.

### Instructor

If the client is identified as an instructor, the server expects the instructor's ISN and password. It validates the credentials using an InstructorInfoDao and communicates the authentication result back to the client. Upon successful authentication, it provides options to the instructor, such as managing courses, assigning grades, or exiting.

### Admin

If the client is identified as an administrator, the server expects a username and password. It validates the credentials and communicates the authentication result back to the client. Upon successful authentication, it provides options to the administrator, such as registering students or exiting.

### Communication

The communication with clients is done through DataInputStream, DataOutputStream and Object Streams for complex objects like dates.

## Exception Handling

The code includes exception handling for `IOException`, `SQLException`, and `ClassNotFoundException`, logging errors or throwing a runtime exception in case of an issue, and informing the client about errors in the clients input for example.

## Student Client

The client establishes a socket connection with the server running on localhost at port 8000. It then sends a type identifier "1" to the server, indicating that it's a student client.

### Authentication

The client then prompts the user to enter their student number (SSN) and password, hashes the password using SHA-256, and sends the credentials to the server. The client receives a boolean value indicating the success or failure of the authentication. If the authentication is successful, the client proceeds to interact with the server. It displays a welcome message from the server and presents a menu to the user with options such as viewing marks or exiting.

### Menu

The client reads the user's choice, validates it, and sends the choice to the server. Depending on the choice, the client performs actions such as retrieving and displaying course enrollments and marks or exiting the application.

### Viewing enrolled courses

If the user chooses to view marks (option 1), the client receives a list of Enrollment objects from the server using `ObjectInputStream`. It then iterates through the enrollments, displaying details such as course title, grade, mark, class average, highest mark, and lowest mark.

### Input validation

The user input is validated via multiple methods like the `readIntWithValidation`, which keeps prompting the user to enter an int until a valid integer is inputted.

## Instructor Client

The client establishes a socket connection with the server running on localhost at port 8000. It sends a type identifier "2" to the server, indicating that it's an instructor client.

### Authentication

The client then prompts the user to enter their instructor ID and password, hashes the password using SHA-256, and sends the credentials to the server. The client receives a boolean value indicating the success or failure of the authentication. If the authentication is successful, the client proceeds to interact with the server. It displays a welcome message from the server, presents a list of courses, and provides options for the instructor to add marks or exit.



## Handling Courses

The client receives a list of Course objects from the server using ObjectInputStream. It then iterates through the courses, displaying details such as course ID, title, and the number of registered students.

## Menu

The client reads the user's choice, validates it, and sends the choice to the server. Depending on the choice, the client performs actions such as adding marks to students or exiting the application.

## Adding marks

If the user chooses to add marks (option 1), the client prompts the instructor to enter the course ID for the course they want to update. It then displays a list of students enrolled in that course and prompts the instructor to enter the student's SSN, mark, and grade. The client sends this information to the server for processing.

## Input validation

The user input is validated via multiple methods like the readIntWithRangeValidation, which prompts the instructor to enter a valid mark between 0 and 100.

## Admin Client

The client establishes a socket connection with the server running on localhost at port 8000. The client sends a type identifier "3" to the server, indicating that it's an admin client.

## Authentication

The client then prompts the user to enter their username and password, sends the credentials to the server, and receives a boolean value indicating the success or failure of the authentication. If the authentication is successful, the client proceeds to interact with the server. It displays a welcome message from the server and presents options for the administrator to register a new student or exit.

## Menu

The client reads the user's choice, validates it, and sends the choice to the server. Depending on the choice, the client performs actions such as registering a new student or exiting the application.

## Student Registration

If the user chooses to register a student (option 1), the client prompts the administrator to enter various details for the new student, such as SSN, name, birth date, address, and contact information. The client sends this information to the server for processing.

## Input validation

The client performs input validation for date format and ensures that the user input for menu choices is a valid integer.

## Summary

The server and its corresponding client applications collectively form a distributed system for managing a grading system. The server facilitates communication and database interactions, while clients provide user interfaces tailored to specific roles (student, instructor, or administrator). The use of password hashing and role-based access control enhances the security of the system, ensuring secure authentication and authorization for each user type.

## Part Two

This part describes the second technique for the grading system, mainly using java servlets and jsps.

The servelts serve different tasks, there are the login servelts, the logout servlets, the servlets responsible for showing the profile pages, and servlets for handling actions like assigning marks and registering students.

## Controller(Servlets)

### Login Servlets

These are the studentlogin, instructorlogin and admin login servlets that handle authentication. Each have a doGet method that directs the user to the jsp that shows a login form, and a doPost method that handles the form submission. This is done using the same doa interfaces discussed before. If the login is valid, the user directed using the response.sendRedirect to the appropriate url for the profile page. Otherwise the user is sent back to the login page with the message telling them they have given the wrong credentials. The passwords submitted through the form are hashed before processed by the daos.

### Profile servlets

These are the servlets that are used to handle the viewing of the profile page of the logged in user. These include the StudentProfile, InstructorProfile, and AdminProfile servlets. The student profile, and instructor profile servlets have only doGet method responsible for retrieving the courses that the student is enrolled in and storing them inside a list that is set inside an attribute of “enrollments” that can be accessed in the jsp. The instructor servlet’s doGet gets the courses the student teaches via the instructorInfoDao and sets an attribute so that the jsp can view them.

### Action Servelts

These contain servlets to handle various actions like assigning marks if the user logged in is an instructor, or registering a student if an admin is logged in. The instructor reaches the assign marks by clicking on the course he wants to update marks for his profile page. The assign marks has

doGet and doPost methods, the doGet stores the students that are enrolled in the selected course, and the doPost handles the form submission. The enrollmentDoa has an assign marks method that is called after the form is submitted and the input validated. The register students in the Admin works in a similar manner.

## Session handling

In the entirety of this part, logging in and logging out is handled through sessions from HttpSession, before accessing a specific profile page, the code checks if the user is logged in or not through the this session, if he is not logged in, he is taken to the login page with an error message.

## Logout servlets

These servlets use the session to logout users by invalidating their sessions and redirecting them to the login page.

## Custom error pages

The code includes a servlet for showing error pages with appropriate messages.

## Model

The model in this project is the various classes that translate the tables in the database into java classes, with fields equivalent to those in the database tables, and getters and setters.

<b>Instructor</b> <ul style="list-style-type: none"> <li>password String</li> <li>isn String</li> <li>deptId String</li> <li>lastName String</li> <li>mi String</li> <li>firstName String</li> <li>instructorRank String</li> <li>getFirstName() String</li> <li>setLastName(String) void</li> <li>setMi(String) void</li> <li>setDeptId(String) void</li> <li>getMi() String</li> <li>setIsn(String) void</li> <li>setInstructorRank(String) void</li> <li>getLastName() String</li> <li>setPassword(String) void</li> <li>getDeptId() String</li> <li>getInstructorRank() String</li> <li>getIsn() String</li> <li>setFirstName(String) void</li> <li>getPassword() String</li> </ul>	<b>Enrollment</b> <ul style="list-style-type: none"> <li>courseId String</li> <li>ssn String</li> <li>average double</li> <li>title String</li> <li>dateRegistered Date</li> <li>grade String</li> <li>highest int</li> <li>lower int</li> <li>mark int</li> <li>getDateRegistered() Date</li> <li>setSsn(String) void</li> <li>getDateRegistered(Date) void</li> <li>getGrade() String</li> <li>setMark(int) void</li> <li>getTitle() String</li> <li>getSsn() String</li> <li>getLower() int</li> <li>setCourseId(String) void</li> <li>setHighest(int) void</li> <li>getCourseId() String</li> <li>setGrade(String) void</li> <li>getAverage() double</li> <li>setTitle(String) void</li> <li>getHighest() int</li> <li>getMark() int</li> <li>setLower(int) void</li> <li>setAverage(double) void</li> </ul>	<b>Course</b> <ul style="list-style-type: none"> <li>title String</li> <li>subjectId String</li> <li>students List&lt;StudentInfo&gt;</li> <li>courseNumber int</li> <li>numOfStudents int</li> <li>numOfCredits int</li> <li>courseId String</li> <li>getNumOfStudents() int</li> <li>setCourseId(String) void</li> <li>setSubjectId(String) void</li> <li>setTitle(String) void</li> <li>setNumOfCredits(int) void</li> <li>setCourseNumber(int) void</li> <li>getTitle() String</li> <li>setStudents(List&lt;StudentInfo&gt;) void</li> <li>getStudents() List&lt;StudentInfo&gt;</li> <li>getSubjectId() String</li> <li>setNumOfStudents(int) void</li> <li>getCourseId() String</li> <li>getNumOfCredits() int</li> <li>getCourseNumber() int</li> </ul>	<b>StudentInfo</b> <ul style="list-style-type: none"> <li>firstName String</li> <li>lastName String</li> <li>phone String</li> <li>mi String</li> <li>zipcode String</li> <li>ssn String</li> <li>street String</li> <li>password String</li> <li>birthDate Date</li> <li>deptId String</li> <li>setLastName(String) void</li> <li>getBirthDate() Date</li> <li>getPassword() String</li> <li>getSsn() String</li> <li>getLastName() String</li> <li>setSsn(String) void</li> <li>setFirstName(String) void</li> <li>getMi() String</li> <li>setMi(String) void</li> <li>setStreet(String) void</li> <li>getDeptId() String</li> <li>getStreet() String</li> <li>setPassword(String) void</li> <li>setPhone(String) void</li> <li>getZipcode() String</li> <li>setBirthDate(Date) void</li> <li>getFirstName() String</li> <li>getPhone() String</li> <li>setDeptId(String) void</li> <li>setZipcode(String) void</li> </ul>	<b>Teaching</b> <ul style="list-style-type: none"> <li>isn String</li> <li>CourseId String</li> <li>getIsn() String</li> <li>getCourseId() String</li> <li>setIsn(String) void</li> <li>setCourseId(String) void</li> </ul>
---	--	--	--	---

Figure 8: Model

## Views(JSPS)

The views in this project are the jsp, these are html pages with java code embedded in them. For example a java list is defined to get the attribute that has the courses enrolled and a loop to put them in an unordered list for the student to see.

## Input validation and exception handling

The input provided by the user is handled on both the client side and the back-end. The client side handling is done by the jsp form inputs attributes, and on the back-end through the servlets, if an error is found, the user is taken to an error page with a message on it informing him what is the problem. Sql statements that can lead to errors are handled in a similar manner.

## Summary

In summary, this part utilizes java servlets and jsp to provide dynamic web pages and user authentication, care has been put to provide input validation and exception handling.

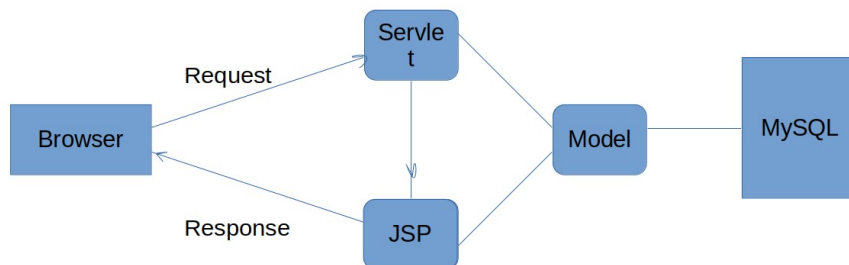


Figure 9: Servlets summary

## Part Three

In this part, we utilize spring boot framework to build the grading system. This framework follows the MVC pattern, we will start with the controller

### Controllers

The controllers in this framework are very similar to the servlets in part two, however these controllers differ in certain aspects, first these controllers don't have doPost and doGet methods, instead they use mappings to achieve the same effect. Second, these controllers don't directly invoke the daos, instead they invoke services, and from these services daos are invoked, to achieve dependency injection, we annotate them with controller and autowiring, and the services are annotated with component with type of service.

### Services

The services layer is the layer that calls the dao objects, in this project we have a service for each model class we have, ie for each table in the database, with the addition of a session service we will discuss later on.

### Model

The model classes are the same as in the part before.

### Views

The views here are html pages, and to achieve dynamic webpages with java code inside the html pages, we use thymeleaf. This is similar but also different than the jsp's of the previous part.

### DAOs

The data access objects are similar to the previous parts with the exception that they use a JDBC template instead of normal jdbc programming used before.

### Security

Spring boot can have a spring security dependency, however due to the simplicity of this project, I opted for using a custom session service that stores user logins inside a session, this way if a user tries to access a page that he is not authorized to view, the session service will redirect him to the login page depending on his user role. Also I used session services to have a higher control of what happens when a user is not authenticated.

### Summary

In this part, the grading system is developed using the Spring Boot framework, adhering to the MVC (Model-View-Controller) pattern. The controllers, similar to servlets but with distinctions, use

mappings instead of doGet and doPost methods. These controllers interact with services, which, in turn, invoke DAOs to achieve dependency injection. The services layer corresponds to each model class, representing tables in the database. Thymeleaf is employed for dynamic web pages with Java code embedded. DAOs, while resembling previous parts, use JDBC templates for data access. Security is handled through a custom session service, storing user logins and redirecting unauthorized users based on their roles, ensuring controlled access and authentication management for the project's simplicity, avoiding the use of Spring Security dependency.

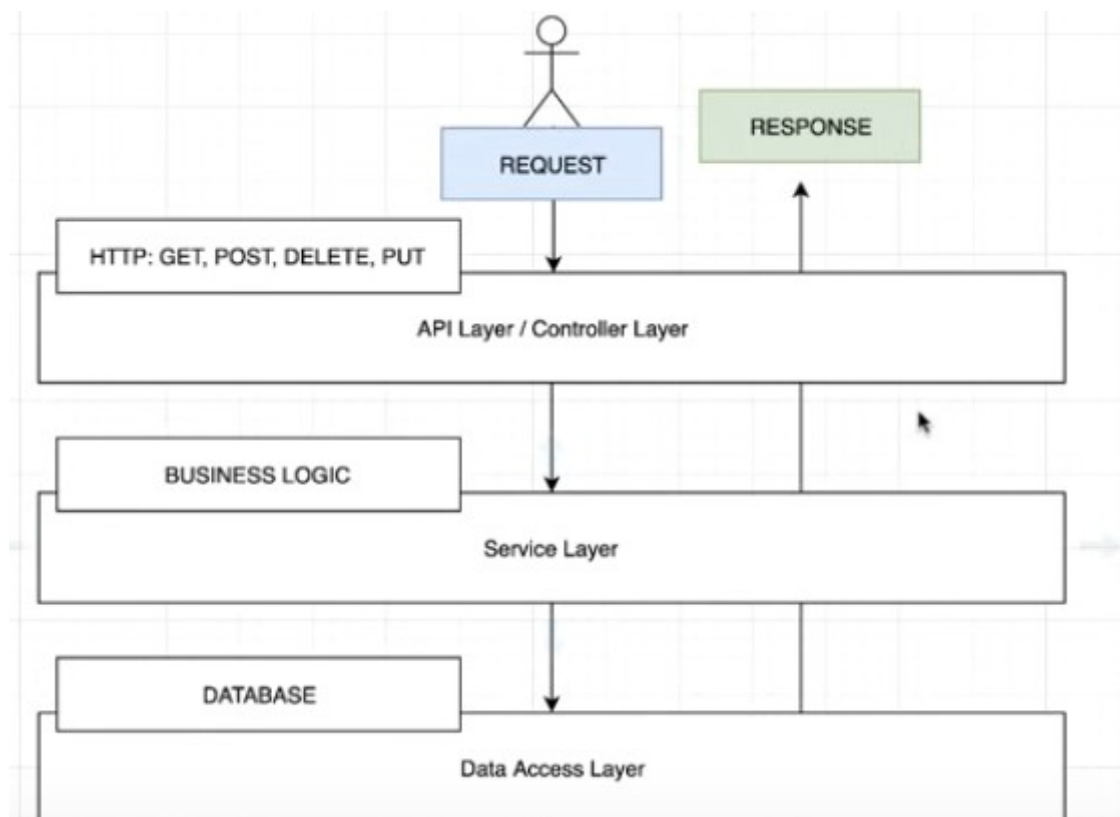


Figure 10: Spring boot summary

## Table of Figures

Figure 1: Tables.....	4
Figure 2: Student table.....	4
Figure 3: Enrollment.....	4
Figure 4: Course.....	5
Figure 5: Instructor table.....	5
Figure 6: Teaching table.....	6
Figure 7: DAOs.....	6
Figure 8: Model.....	11
Figure 9: Servlets summary.....	12
Figure 10: Srping boot summary.....	14