Student Grading System

The student grading system is a web application designed to facilitate and manage student grades, courses. The idea is to develop it using Spring Boot, Servlets, and Sockets to feel the differences.

Part3: **Spring Boot:**

Designing and implementing the student grading system with spring boot using **N-Tier** architecture And **MVC** design pattern, the project is divided into **Models**, **Views**, **Controllers**, **Services**, and **Repositories.**

Model: represents entities that we have such as Student, Admin entities.
Eg: (@Entity(name="admin")).

Views: represents the page that the user will interact with the application through.

Controllers: represent the request handler, let's say that it is the mediator Between the user and the service, which take the request from the user And send it to the service, then send response to the user.
Eg: (@RestController).

Services: which represent the Business logic, which is the layer that contains The logic and implementation and handling of the request coming from the Controller, and it is the one that responsible to communicate with the Data Link Layer.
Eg: (@Service).

Repositories: which represent the Data Link layer, and it is the layer that responsible to communicate with the data base and it send the request to the database then send the response to the Service layer.
Eg: (@Repository).

N-**Tier** architecture represents:

- Application Layer which is the GUI.
- Business Logic Layer which is processing the request.
- Data Link Layer which is communicates with Data Base.

ORM frameworks such as **Hibernate, JPA** was used to map Java classes into Tables in MySQL and the attributes into columns and the objects into records

Spring JPA repository provides a variety of built-in methods to deal with Databases without needs to write the query, it performs it behind the scene. This thing is facilitating interaction with Database, but it limits the user control over the query.

Communication between client and the controller is done via **REST APIs**, through **URL** requests sent from the client side and handled by the controller.
Eg: (PostMapping("/addStudent")), (GetMapping("/getStudentGrades"))

I configured my application setting the right dependencies using Maven **POM** like Spring JPA, MySQL connector jar, and etc.., Maven allows to better and easily way to configure your dependencies and the **resources file** that

contains the Database configuration like the Driver, Root, and the Password.

Lombok is also used by adding it into Maven POM. Lombok writes the boilerplate code behind the scenes such as Constructors, Getters, and the Setters.

And the MySQL Database is deployed and running inside a **Docker** container.

Challenges:

- Error Handling for mapping the project into schema.
- Security.
- Limit control over the query.

Pros:

- Managing requests easily.
- Less programmer involvement in the low-level details.
- Easy setup and configuration.

Cons:

- Limit programmer access to low-level data.
- Can’t control performance.

Part2: **Servlets:**

Servlets is a Java class that take the client request and response back to the client
It acts like a controller in Spring Boot, calling a Servlet force you to Override any of Http Servlet class URLs.

Designing and implementing the servlets done using **N-Tier** arch and the traditional **MVC** pattern, except that the views are the JSPs and controllers are the servlets.

Servlets configurations isn't as easy as Spring Boot configurations, it requires more dependencies, using commands to run **Tomcat**.
Eg: (mvn tomcat7: run).

Connection with Database was using Connection Provider (**JDBC**) and
There are no repositories for dealing with DB via Built-in methods, this gives you the advantage of manipulating and controlling the query more than spring boot, but it implies that you need to write the queries manually.

Lombok was also used to reduce boilerplate code.

Challenges:

- Writing complex queries.
- Error handling with servlets and connection with the Database.
- Naming Convention with attributes to match in queries and Database,
- Creating tables and referencing each other to meet the Relationships concepts.

Pros:

- More access and control over low-level details.
- Easy setting up and configurations but not as same as Spring Boot.

Cons:

- Servlets need JSP's.
- Less flexibility than controllers.

Part1: **Sockets:**

Sockets is a low-level communication between clients and server.

In this project I created 3 clients (Admin, Instructor, Student) and the server to perform client's requests.

The designing of the code is done via Console showing each client menu.

Challenges:

- Managing more than request at a time.
- Setting up the configurations.
- Communication via Sockets.

Pros:

- Higher performance.
- More control over low-level data.

Cons:

- Not easy setup and configurations.
- Managing request simultaneously not easy.
- Applying the concept of multithreading manually.

Differences:

- Sockets can give you more control over low-level details than Spring

  And Servlets.

- Spring Boot offers an easy way to configure and set up.
- Spring Boot generates overhead more than lower levels (Servlets & Sockets).
- Spring Boot more flexible and organized.
- Spring MVC acts better than traditional MVC.
- Spring Boot libraries offer many features over Servlets and Sockets.

Future Enhancement:

- User Authentication.
- UI integration.
- More Analytic Data and Statistics.
- Portals for each role.
- Include more relationships such as: department, Faculty.
- Managing more and more request simultaneously using sockets.