Rayyan Alhourani

Atypon

Student grading system

the system's architecture, data model, and interaction flow between components.

A- the system's architecture

1- **Socket-based Project:**

- A multithreaded server listens for incoming user requests using sockets.
- User requests are received as strings and split to extract necessary data.
- Data processing is performed based on the extracted information.
- Server using variables that stores user role and login status.
- Responses are sent back to the client to display messages, status, and errors.

2- **Servlet-based Project:**

- Requests are sent to a server using servlets, often through HTTP requests.
- Request parameters hold the required data.
- The server checks user authorization and role before processing the request.
- Server session stores user role and login status.
- Responses, including success messages or error alerts, are sent back to the client.

3- **Spring Boot Project with REST API:**

- Spring Boot framework is used, and REST APIs are built for communication.
- Clients send HTTP requests to specific endpoints.
- Similar to previous projects, the server ensures user authentication and authorization.
- Server session stores user role and login status.
- Responses containing messages or data are returned to the client.

B-data model

The system's data model is centered around the interaction between users, courses, and their roles.

1. **User Model:**
    - User table with attributes: id, name, password, and role.
    - Roles: admin, teacher, and student.
2. **Course Model:**
    - Course table with attributes: id, name, user_id (teacher's user ID).
3. **Student-Course Model:**
    - student_course table containing user_id (student's user ID), course_id, and student's marks.

C- interaction flow between components.

Here's a simplified explanation of the interaction flow for a typical use case in your system, such as adding a new user:

1. **User Registration:**
    - Client (terminal, HTML form, or API client) initiates a request to add a new user.
    - The request is sent to the server, which checks the user's authorization and role.
    - If authorized, the server processes the request and adds the new user's details to the user table.
2. **Course Creation:**
    - An admin user initiates a request to create a new course.
    - The server validates the user's authorization as an admin.
    - Upon validation, the server adds the new course details to the course table, linking it to the teacher's user ID.
3. **Entering Student Marks:**
    - A teacher user submits a request to enter student marks for a course.
    - The server verifies the user's role and authorization.
    - If validated, the server updates the student_course table with the student's marks for the specified course and user.

4. **Viewing Student Marks:**
    - A student user requests to view their marks.
    - The server validates the user's authorization as a student.
    - Upon validation, the server retrieves the student's marks from the student_course table and sends them as a response.
5. **Conclusion:**
    The student grading system uses different architectural approaches, including socket-based communication, servlet-based interaction, and Spring Boot REST APIs. The data model encompasses users, courses, and student-course relationships. The interaction flow involves user registration, course creation, entering student marks, and viewing student marks. Through these components, your system efficiently manages user roles, authorization, and data

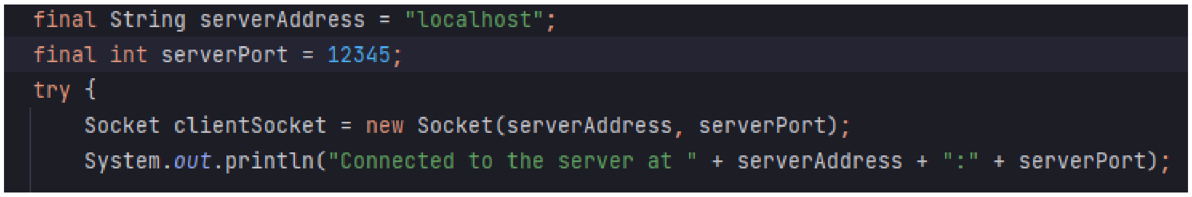
Implementation

There will be an examples of code the present the general idea of implemention

A. **Socket-based Project:**

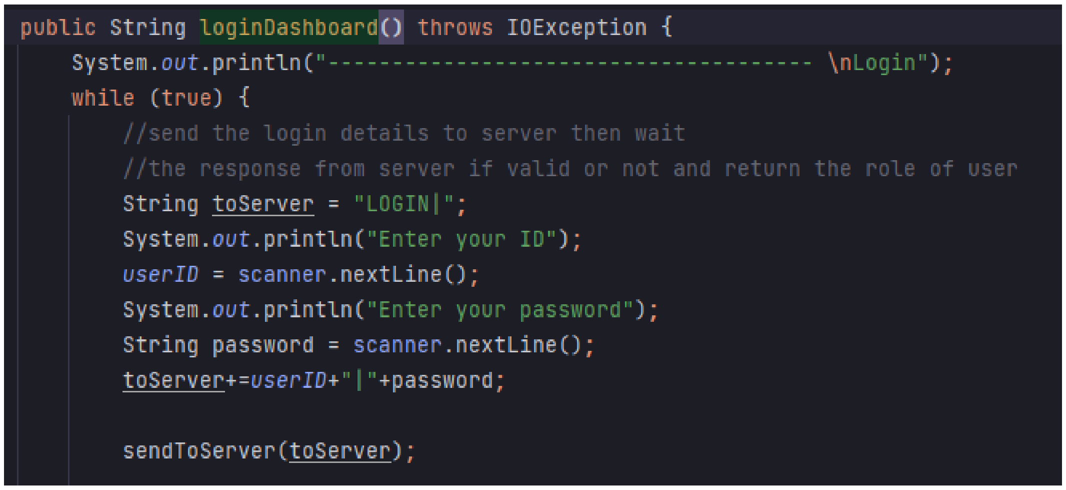
Client implementation:

1. **Initial Connection and User Authentication:**
    - The process commences with the client-side server establishing a connection to the server-side server.
    - The server-side server prompts the user to input their ID and password for authentication.



```
final String serverAddress = "localhost";
final int serverPort = 12345;
try {
    Socket clientSocket = new Socket(serverAddress, serverPort);
    System.out.println("Connected to the server at " + serverAddress + ":" + serverPort);
```

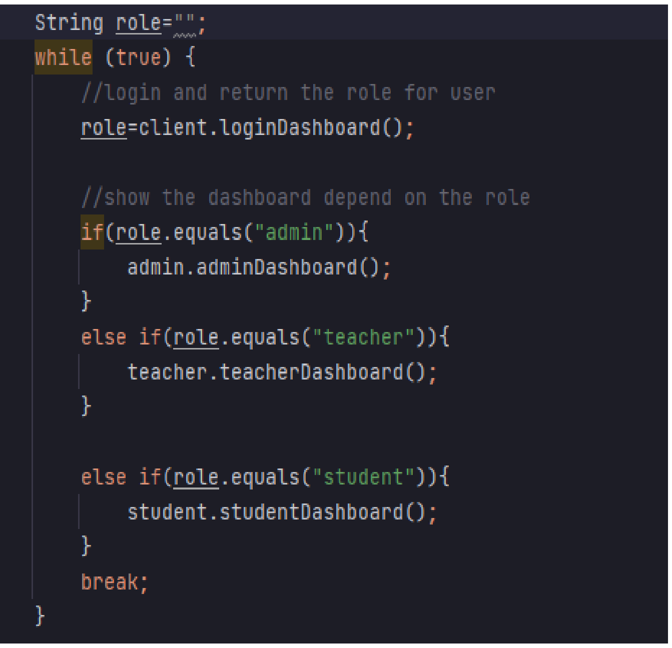
2. **Data Transmission and Action Identification:**
    - User-provided ID and password are combined into a string, separated by a pipe symbol ('|').
    - The string is transmitted to the server-side server for processing.
    - The server-side server inspects the initial part of the string to determine the intended action.



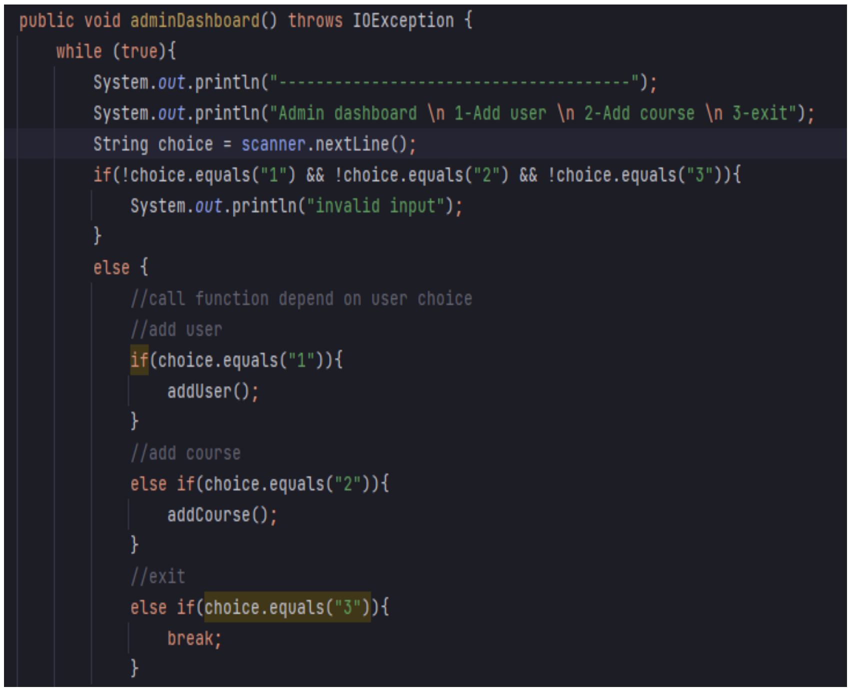
```
public String loginDashboard() throws IOException {
    System.out.println("------------------------------------- \nLogin");
    while (true) {
        //send the login details to server then wait
        //the response from server if valid or not and return the role of user
        String toServer = "LOGIN|";
        System.out.println("Enter your ID");
        userID = scanner.nextLine();
        System.out.println("Enter your password");
        String password = scanner.nextLine();
        toServer+=userID+"|"+password;

        sendToServer(toServer);
```

3. **Role Assignment and Dashboard Display:**
    - If the action is "LOGIN," the server-side server validates the user's credentials against the database.
    - Upon successful validation, the user's role is retrieved from the database.
    - The server-side server communicates the user's role back to the client-side server.
    - The client-side server displays a customized dashboard tailored to the user's role.

4. **Role-specific Functionalities:**
    - Admin Role: The dashboard for admins contains options to "Add User" and "Add Course."
    - Teacher Role: The dashboard for teachers includes options to "Add Student to Course" and "Enter Student Marks."
    - Student Role: Students can view their own marks exclusively.
    - All users have the option to log out from the system.



```java
String role="";
while (true) {
    //login and return the role for user
    role=client.loginDashboard();

    //show the dashboard depend on the role
    if(role.equals("admin")){
        admin.adminDashboard();
    }
    else if(role.equals("teacher")){
        teacher.teacherDashboard();
    }

    else if(role.equals("student")){
        student.studentDashboard();
    }
    break;
}
```

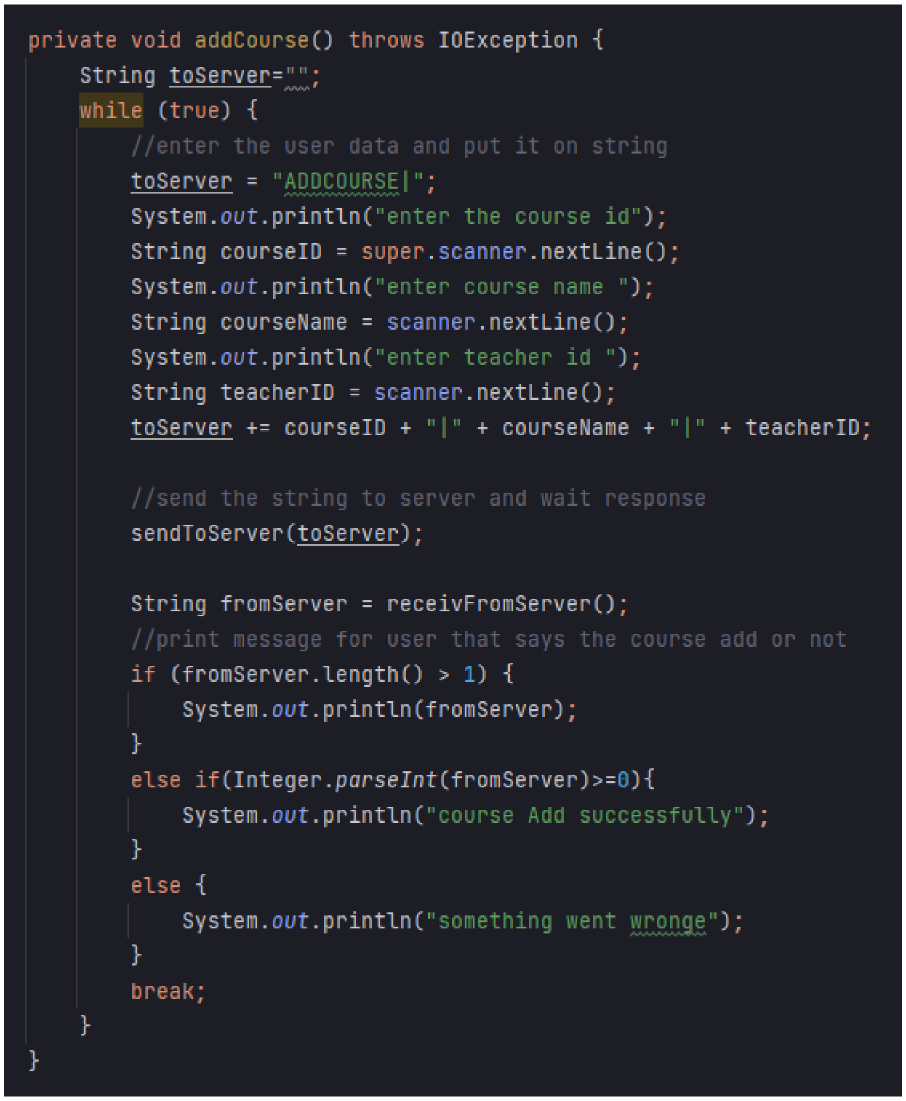


```java
public void adminDashboard() throws IOException {
    while (true){
        System.out.println("------------------------------------");
        System.out.println("Admin dashboard \n 1-Add user \n 2-Add course \n 3-exit");
        String choice = scanner.nextLine();
        if(!choice.equals("1") && !choice.equals("2") && !choice.equals("3")){
            System.out.println("invalid input");
        }
        else {
            //call function depend on user choice
            //add user
            if(choice.equals("1")){
                addUser();
            }
            //add course
            else if(choice.equals("2")){
                addCourse();
            }
            //exit
            else if(choice.equals("3")){
                break;
            }
```

5. **User Action Selection and Data Input:**
    - Users select their desired action from the available options on the dashboard.
    - The client-side server requests relevant data input from the user.

6. **Data Formatting and Action Tagging:**
    - The client-side server consolidates user input into a string, segmented by a pipe symbol ('|').
    - The string's initial content designates the intended action, like "LOGIN," "ADDUSER," or "SETMARKS."

7. **Data Transmission to Server and Response Handling:**
    - The client-side server transmits the formatted string to the server-side server.
    - The server-side server processes the action and data accordingly.
    - Following processing, the server-side server responds to the client-side server, indicating successful execution or providing error details.



```java
private void addCourse() throws IOException {
    String toServer="";
    while (true) {
        //enter the user data and put it on string
        toServer = "ADDCOURSE|";
        System.out.println("enter the course id");
        String courseID = super.scanner.nextLine();
        System.out.println("enter course name ");
        String courseName = scanner.nextLine();
        System.out.println("enter teacher id ");
        String teacherID = scanner.nextLine();
        toServer += courseID + "|" + courseName + "|" + teacherID;

        //send the string to server and wait response
        sendToServer(toServer);

        String fromServer = receivFromServer();
        //print message for user that says the course add or not
        if (fromServer.length() > 1) {
            System.out.println(fromServer);
        }
        else if(Integer.parseInt(fromServer)>=0){
            System.out.println("course Add successfully");
        }
        else {
            System.out.println("something went wronge");
        }
        break;
    }
}
```

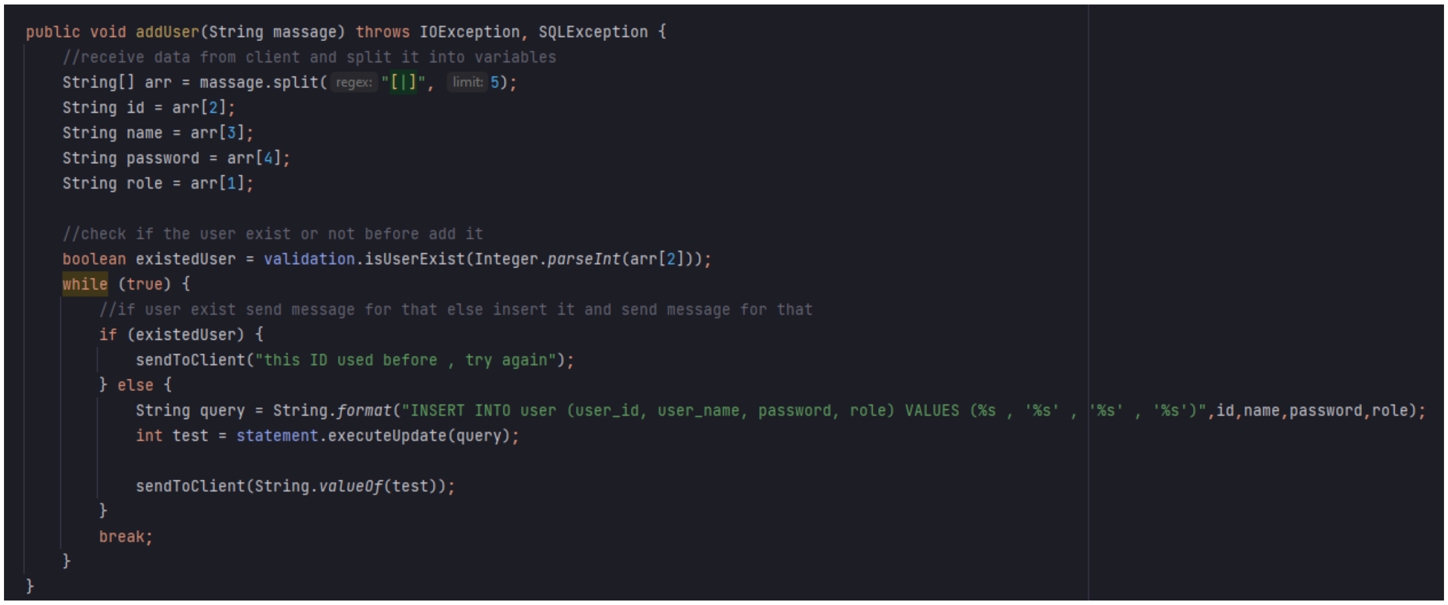
Server implementation:

1. Login Function:

- Responsible for user authentication and role assignment.
- Validates the user's credentials and returns the user's role if valid, or an error if authentication fails.

2. Add User Function:

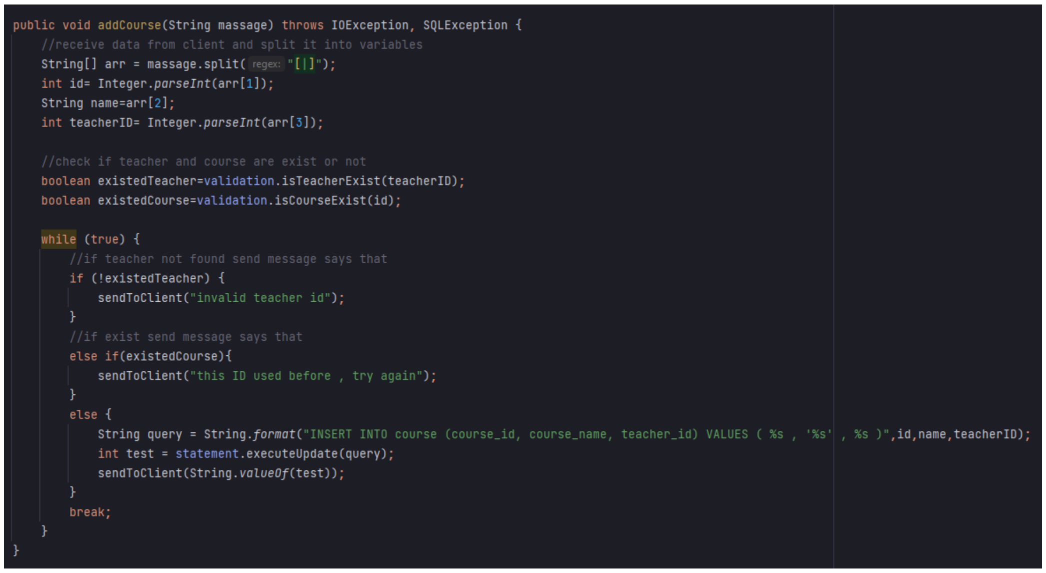
- Allows the admin to add a new user.
- Checks if the entered user ID is available and adds the user if the ID is not in use.
- Returns 1 upon successful user addition, otherwise sends an appropriate error message.



```java
public void addUser(String massage) throws IOException, SQLException {
    //receive data from client and split it into variables
    String[] arr = massage.split( regex: "[|]",  limit: 5);
    String id = arr[2];
    String name = arr[3];
    String password = arr[4];
    String role = arr[1];

    //check if the user exist or not before add it
    boolean existedUser = validation.isUserExist(Integer.parseInt(arr[2]));
    while (true) {
        //if user exist send message for that else insert it and send message for that
        if (existedUser) {
            sendToClient("this ID used before , try again");
        } else {
            String query = String.format("INSERT INTO user (user_id, user_name, password, role) VALUES (%s , '%s' , '%s' , '%s')",id,name,password,role);
            int test = statement.executeUpdate(query);

            sendToClient(String.valueOf(test));
        }
        break;
    }
}
```

3. Add Course Function:

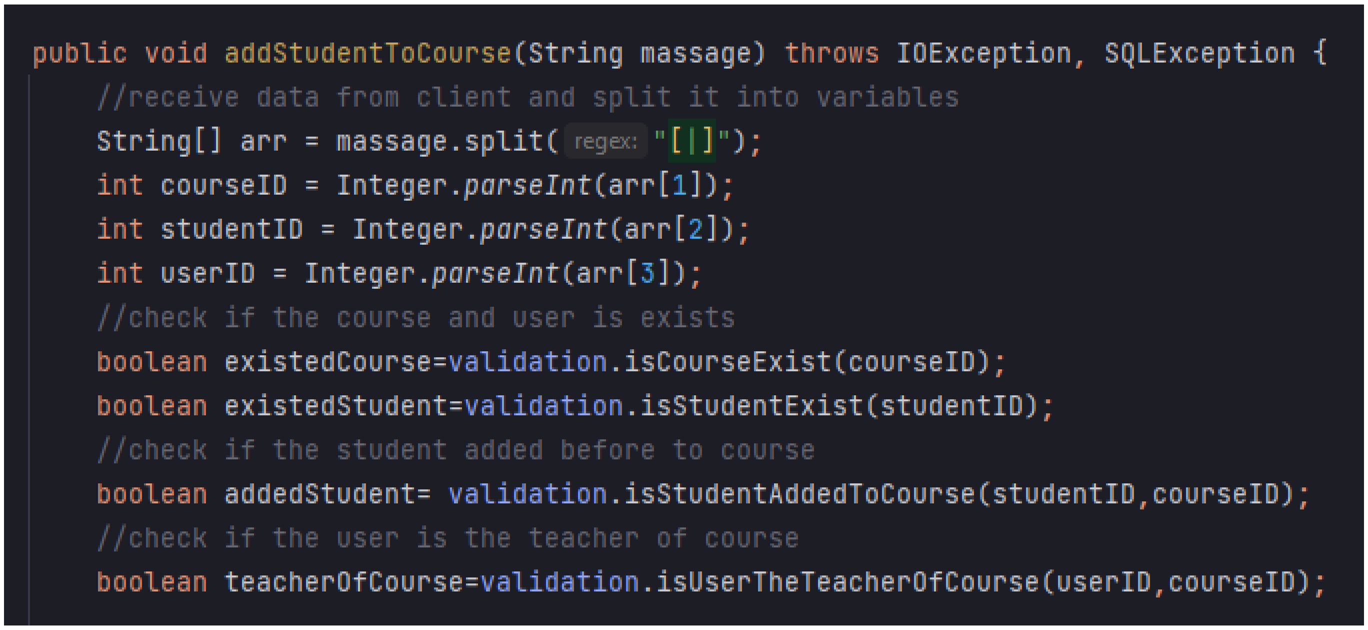
- Enables the admin to add a new course.
- Validates the course ID's availability and checks if the provided teacher ID exists.
- Returns 1 upon successful course addition, otherwise sends an error message.



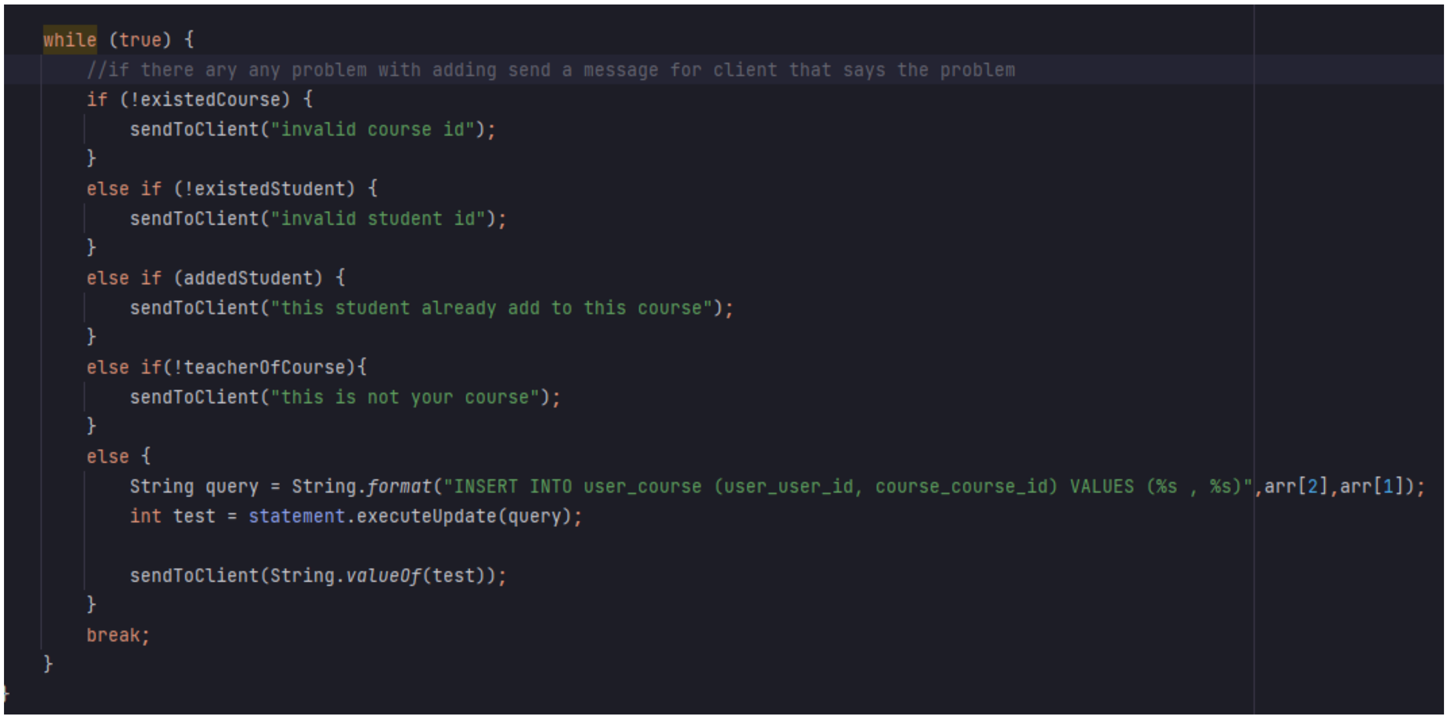
```java
public void addCourse(String massage) throws IOException, SQLException {
    //receive data from client and split it into variables
    String[] arr = massage.split( regex: "[|]");
    int id= Integer.parseInt(arr[1]);
    String name=arr[2];
    int teacherID= Integer.parseInt(arr[3]);

    //check if teacher and course are exist or not
    boolean existedTeacher=validation.isTeacherExist(teacherID);
    boolean existedCourse=validation.isCourseExist(id);

    while (true) {
        //if teacher not found send message says that
        if (!existedTeacher) {
            sendToClient("invalid teacher id");
        }
        //if exist send message says that
        else if(existedCourse){
            sendToClient("this ID used before , try again");
        }
        else {
            String query = String.format("INSERT INTO course (course_id, course_name, teacher_id) VALUES ( %s , '%s' , %s )",id,name,teacherID);
            int test = statement.executeUpdate(query);
            sendToClient(String.valueOf(test));
        }
        break;
    }
}
```

4. Add Student to Course Function:

- Lets teachers add students to their courses.
- Validates course and student IDs, ensures the teacher manages the course, and confirms the student isn't already added.
- Returns a message indicating student addition status and 1 if added successfully.



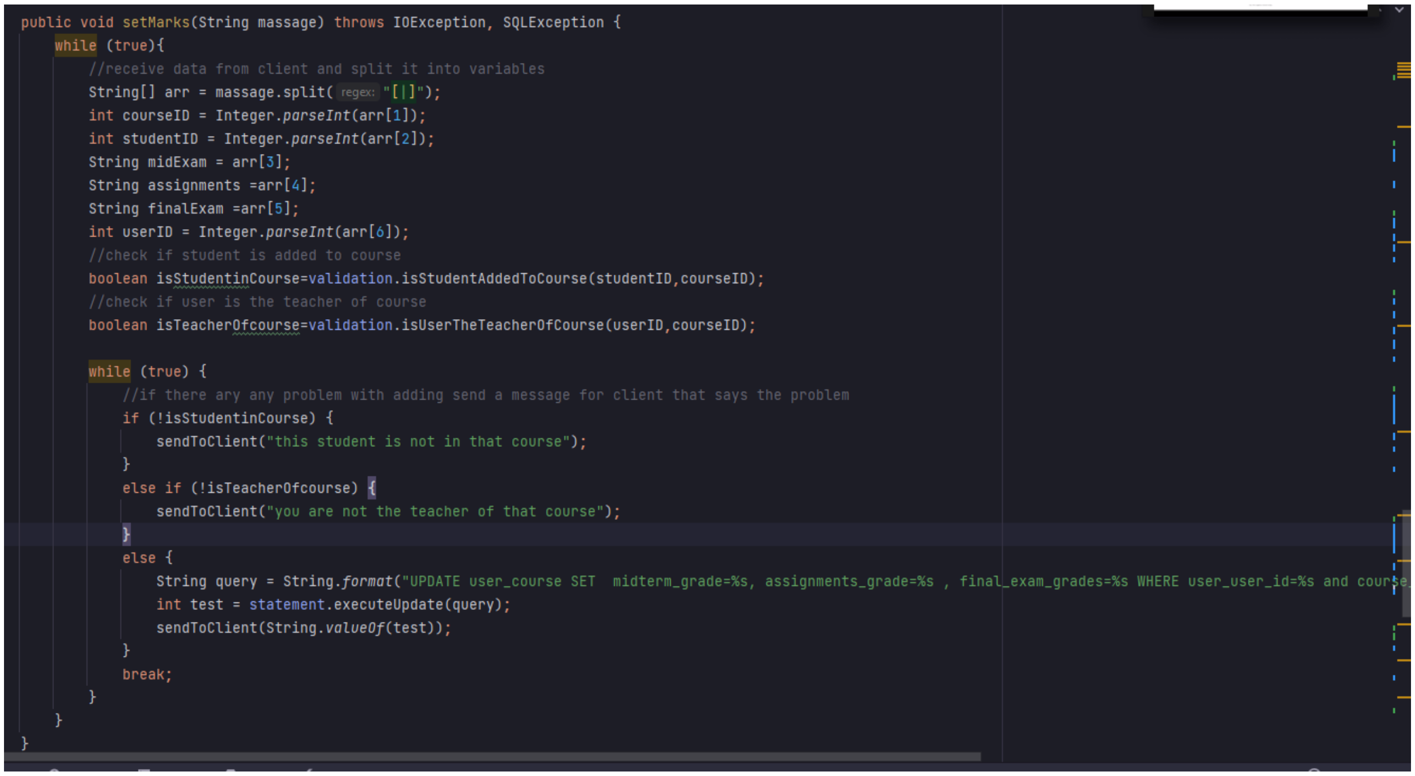
```java
public void addStudentToCourse(String massage) throws IOException, SQLException {
    //receive data from client and split it into variables
    String[] arr = massage.split( regex: "[|]");
    int courseID = Integer.parseInt(arr[1]);
    int studentID = Integer.parseInt(arr[2]);
    int userID = Integer.parseInt(arr[3]);
    //check if the course and user is exists
    boolean existedCourse=validation.isCourseExist(courseID);
    boolean existedStudent=validation.isStudentExist(studentID);
    //check if the student added before to course
    boolean addedStudent= validation.isStudentAddedToCourse(studentID,courseID);
    //check if the user is the teacher of course
    boolean teacherOfCourse=validation.isUserTheTeacherOfCourse(userID,courseID);
```



```java
    while (true) {
        //if there ary any problem with adding send a message for client that says the problem
        if (!existedCourse) {
            sendToClient("invalid course id");
        }
        else if (!existedStudent) {
            sendToClient("invalid student id");
        }
        else if (addedStudent) {
            sendToClient("this student already add to this course");
        }
        else if(!teacherOfCourse){
            sendToClient("this is not your course");
        }
        else {
            String query = String.format("INSERT INTO user_course (user_user_id, course_course_id) VALUES (%s , %s)",arr[2],arr[1]);
            int test = statement.executeUpdate(query);

            sendToClient(String.valueOf(test));
        }
        break;
    }
}
```

5. Set Marks Function:

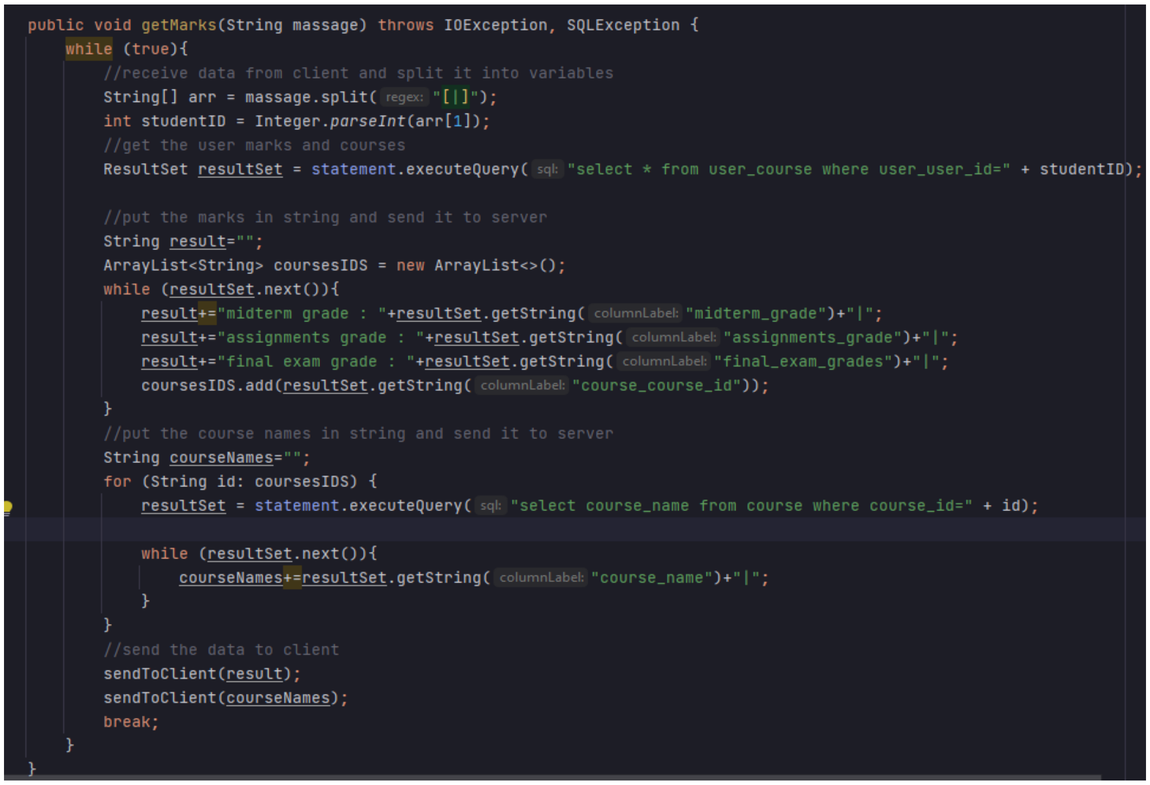
- Allows teachers to set student marks.
- Validates student and course IDs, confirms the user is the teacher for the course, and verifies the student's enrollment.
- Returns 1 upon successful mark update, otherwise sends an error message.



```java
public void setMarks(String massage) throws IOException, SQLException {
    while (true){
        //receive data from client and split it into variables
        String[] arr = massage.split( regex: "[|]");
        int courseID = Integer.parseInt(arr[1]);
        int studentID = Integer.parseInt(arr[2]);
        String midExam = arr[3];
        String assignments =arr[4];
        String finalExam =arr[5];
        int userID = Integer.parseInt(arr[6]);
        //check if student is added to course
        boolean isStudentinCourse=validation.isStudentAddedToCourse(studentID,courseID);
        //check if user is the teacher of course
        boolean isTeacherOfcourse=validation.isUserTheTeacherOfCourse(userID,courseID);

        while (true) {
            //if there ary any problem with adding send a message for client that says the problem
            if (!isStudentinCourse) {
                sendToClient("this student is not in that course");
            }
            else if (!isTeacherOfcourse) {
                sendToClient("you are not the teacher of that course");
            }
            else {
                String query = String.format("UPDATE user_course SET  midterm_grade=%s, assignments_grade=%s , final_exam_grades=%s WHERE user_user_id=%s and course_
                int test = statement.executeUpdate(query);
                sendToClient(String.valueOf(test));
            }
            break;
        }
    }
}
```

6. Get Marks Function:

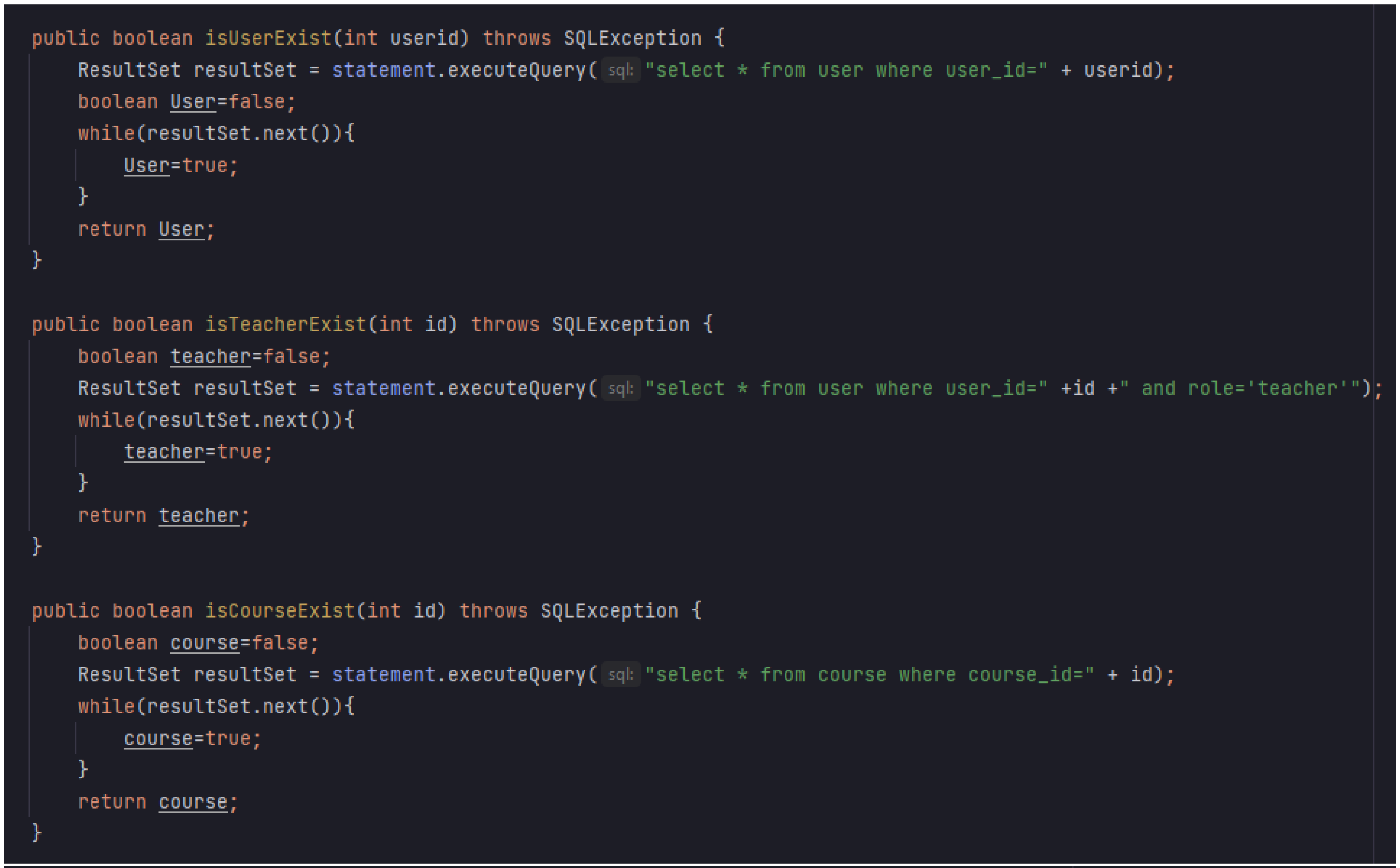
- Displays student marks for the current user (if a student).



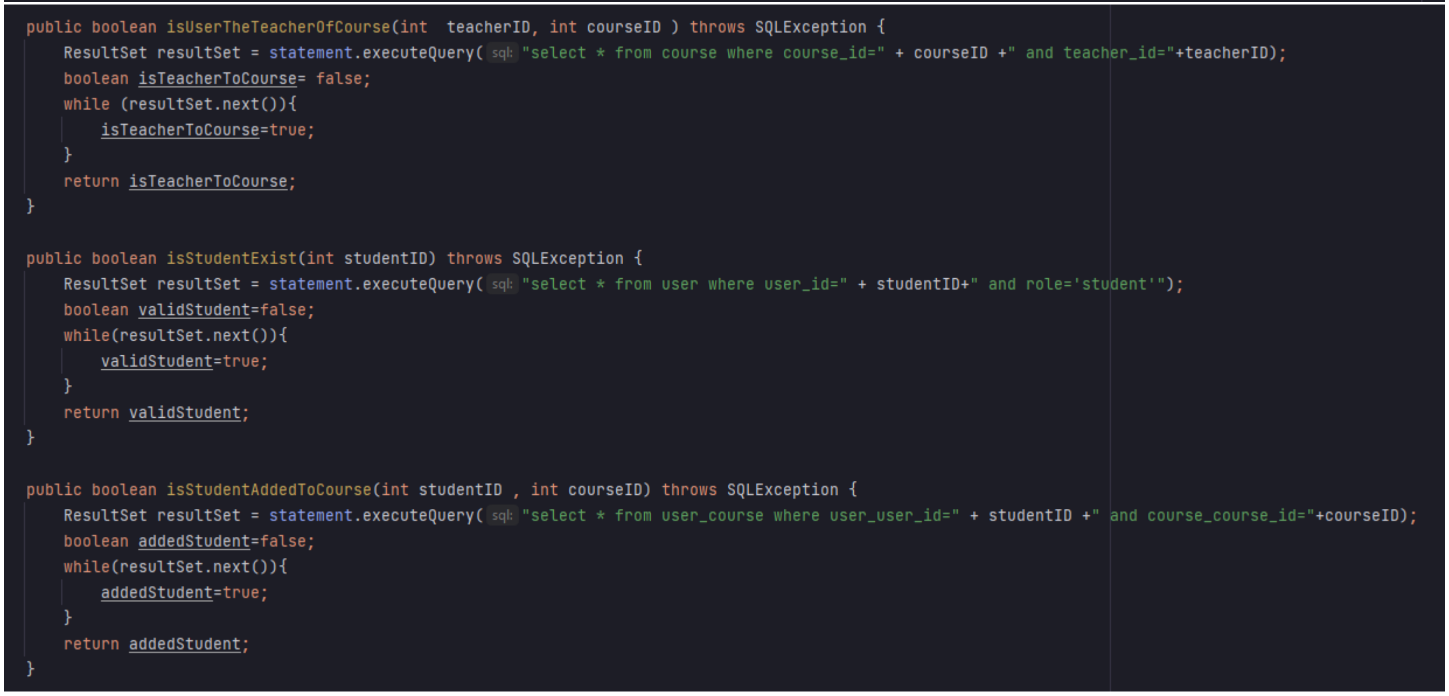
```java
public void getMarks(String massage) throws IOException, SQLException {
    while (true){
        //receive data from client and split it into variables
        String[] arr = massage.split( regex: "[|]");
        int studentID = Integer.parseInt(arr[1]);
        //get the user marks and courses
        ResultSet resultSet = statement.executeQuery( sql: "select * from user_course where user_user_id=" + studentID);

        //put the marks in string and send it to server
        String result="";
        ArrayList<String> coursesIDS = new ArrayList<>();
        while (resultSet.next()){
            result+="midterm grade : "+resultSet.getString( columnLabel: "midterm_grade")+"|";
            result+="assignments grade : "+resultSet.getString( columnLabel: "assignments_grade")+"|";
            result+="final exam grade : "+resultSet.getString( columnLabel: "final_exam_grades")+"|";
            coursesIDS.add(resultSet.getString( columnLabel: "course_course_id"));
        }
        //put the course names in string and send it to server
        String courseNames="";
        for (String id: coursesIDS) {
            resultSet = statement.executeQuery( sql: "select course_name from course where course_id=" + id);

            while (resultSet.next()){
                courseNames+=resultSet.getString( columnLabel: "course_name")+"|";
            }
        }
        //send the data to client
        sendToClient(result);
        sendToClient(courseNames);
        break;
    }
}
```

Validation functions

This section elaborates on the functions responsible for validating user input and executing actions based on the validation results. These files play a pivotal role in ensuring the integrity of data and successful action execution within the system. And the general idea of these functions is to check if the entered is in database or not , and return the true or false depend on result of query that send.



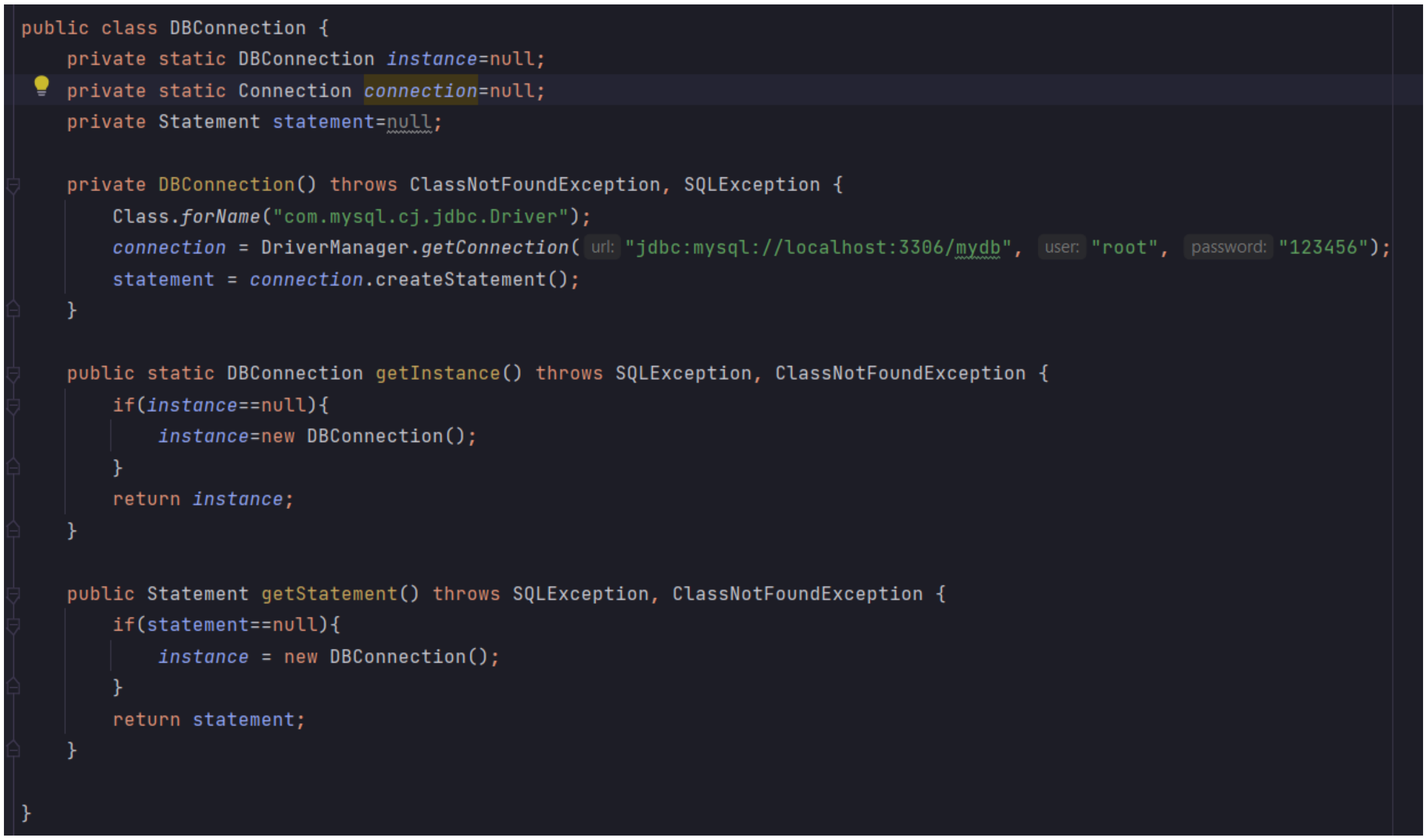
```java
public boolean isUserExist(int userid) throws SQLException {
    ResultSet resultSet = statement.executeQuery( sql: "select * from user where user_id=" + userid);
    boolean User=false;
    while(resultSet.next()){
        User=true;
    }
    return User;
}

public boolean isTeacherExist(int id) throws SQLException {
    boolean teacher=false;
    ResultSet resultSet = statement.executeQuery( sql: "select * from user where user_id=" +id +" and role='teacher'");
    while(resultSet.next()){
        teacher=true;
    }
    return teacher;
}

public boolean isCourseExist(int id) throws SQLException {
    boolean course=false;
    ResultSet resultSet = statement.executeQuery( sql: "select * from course where course_id=" + id);
    while(resultSet.next()){
        course=true;
    }
    return course;
}
```



```java
public boolean isUserTheTeacherOfCourse(int  teacherID, int courseID ) throws SQLException {
    ResultSet resultSet = statement.executeQuery( sql: "select * from course where course_id=" + courseID +" and teacher_id="+teacherID);
    boolean isTeacherToCourse= false;
    while (resultSet.next()){
        isTeacherToCourse=true;
    }
    return isTeacherToCourse;
}

public boolean isStudentExist(int studentID) throws SQLException {
    ResultSet resultSet = statement.executeQuery( sql: "select * from user where user_id=" + studentID+" and role='student'");
    boolean validStudent=false;
    while(resultSet.next()){
        validStudent=true;
    }
    return validStudent;
}

public boolean isStudentAddedToCourse(int studentID , int courseID) throws SQLException {
    ResultSet resultSet = statement.executeQuery( sql: "select * from user_course where user_user_id=" + studentID +" and course_course_id="+courseID);
    boolean addedStudent=false;
    while(resultSet.next()){
        addedStudent=true;
    }
    return addedStudent;
}
```

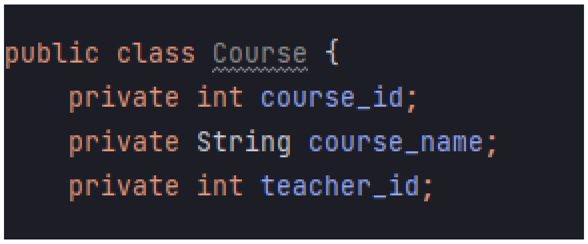
Database connection

we have this function the return instance of DBconnection to use it to send query and use the result of the query

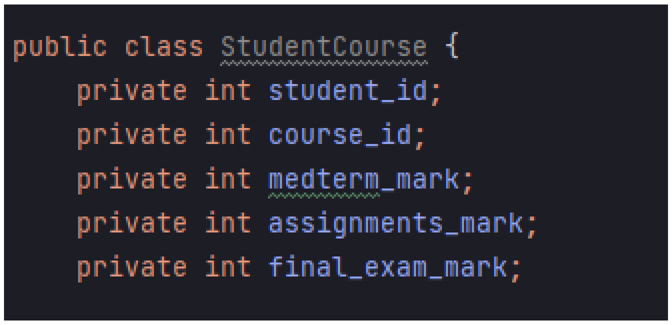


```java
public class DBConnection {
    private static DBConnection instance=null;
    private static Connection connection=null;
    private Statement statement=null;

    private DBConnection() throws ClassNotFoundException, SQLException {
        Class.forName("com.mysql.cj.jdbc.Driver");
        connection = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/mydb",  user: "root",  password: "123456");
        statement = connection.createStatement();
    }

    public static DBConnection getInstance() throws SQLException, ClassNotFoundException {
        if(instance==null){
            instance=new DBConnection();
        }
        return instance;
    }

    public Statement getStatement() throws SQLException, ClassNotFoundException {
        if(statement==null){
            instance = new DBConnection();
        }
        return statement;
    }

}
```

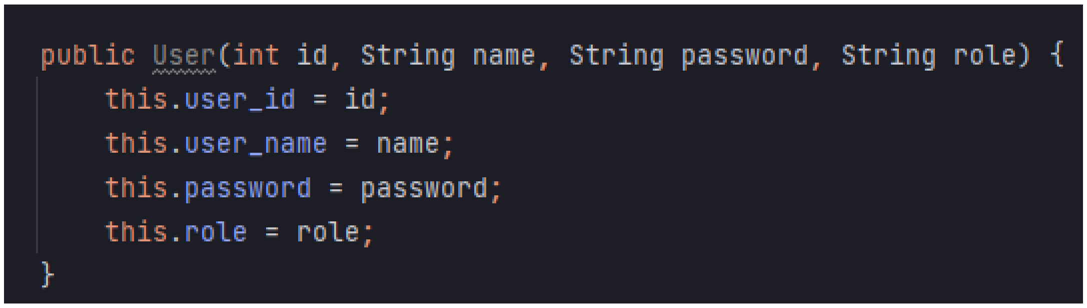
The models



```java
public class Course {
    private int course_id;
    private String course_name;
    private int teacher_id;
```



```java
public class StudentCourse {
    private int student_id;
    private int course_id;
    private int medterm_mark;
    private int assignments_mark;
    private int final_exam_mark;
```



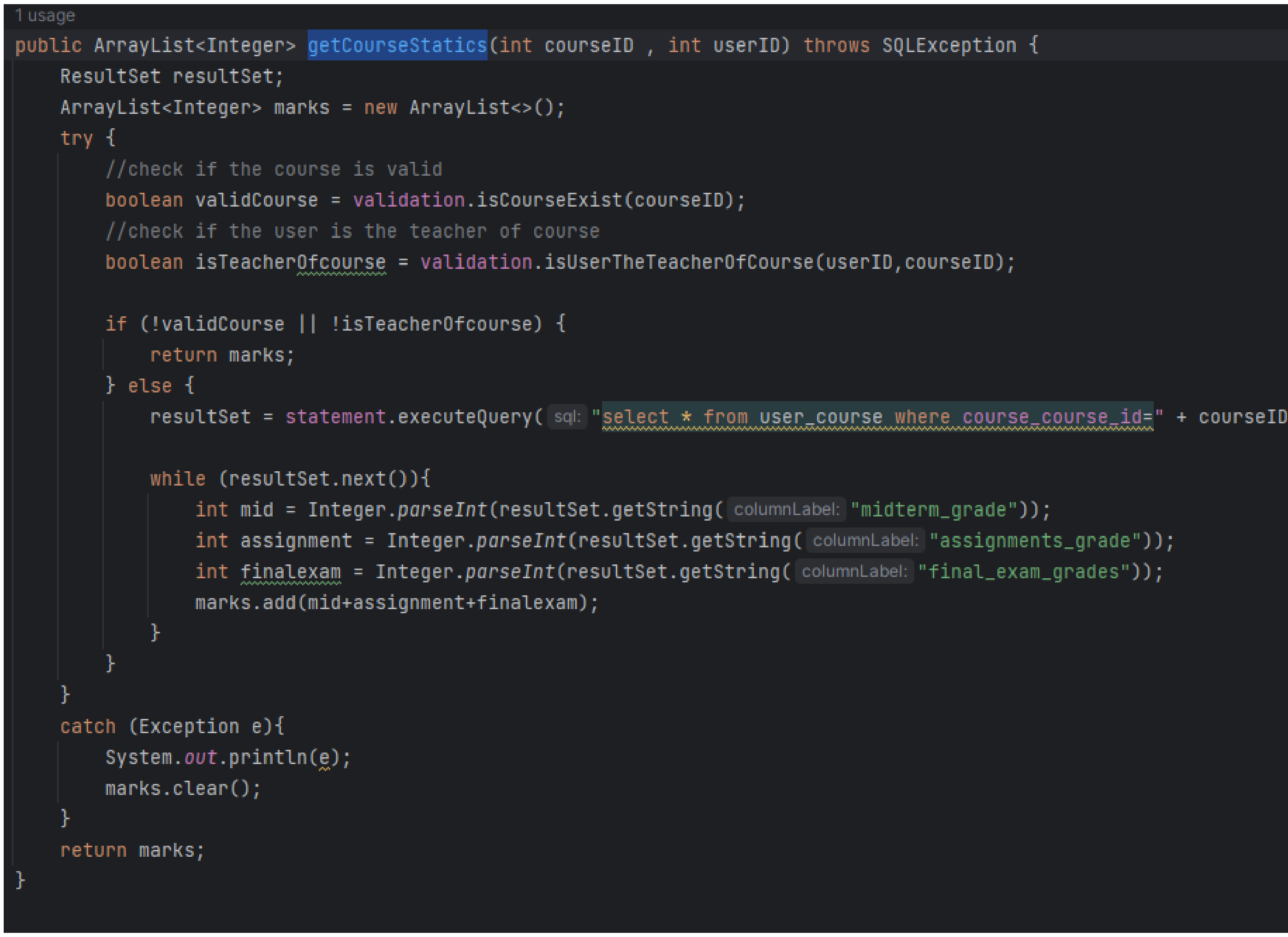
```java
public User(int id, String name, String password, String role) {
    this.user_id = id;
    this.user_name = name;
    this.password = password;
    this.role = role;
}
```

B. **Servlets-based Project:**

The models ,validation functions ,database connection and the actions functions like (add course , add user) that used in sockets , but in servlets some functions instead of send the data in parameters we send object of user or course or student_course , so no need to write about them here, I just will talk about new functions and servlets and jsp.

-extra action functions

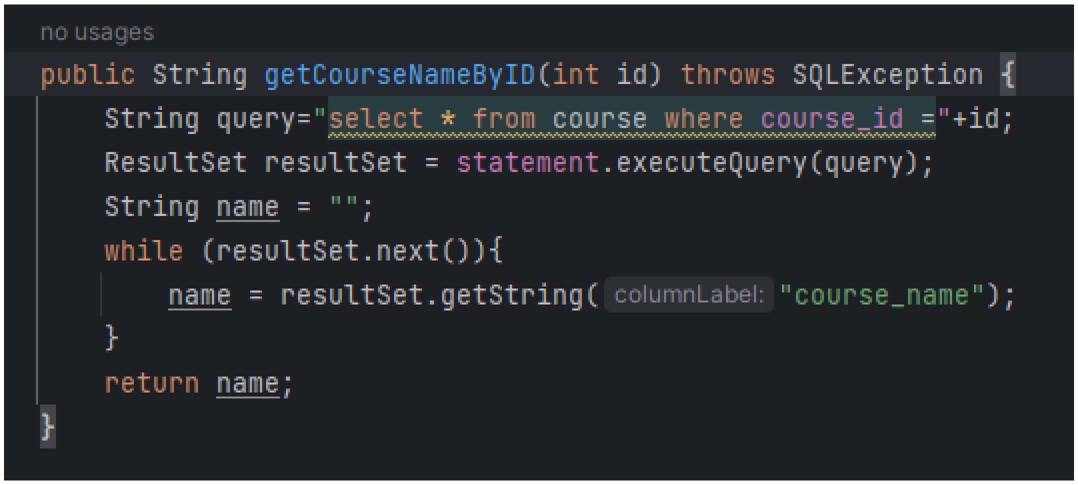
- **getCourseStatistics:**
  - This function retrieves key statistics related to a specific course.
  - It validates the course ID's accuracy and ensures that the user is the assigned teacher.
  - If the data is validated, the function returns array of marks.
  - In cases where the validation fails, an empty array is sent.



```
1 usage
public ArrayList<Integer> getCourseStatics(int courseID , int userID) throws SQLException {
    ResultSet resultSet;
    ArrayList<Integer> marks = new ArrayList<>();
    try {
        //check if the course is valid
        boolean validCourse = validation.isCourseExist(courseID);
        //check if the user is the teacher of course
        boolean isTeacherOfcourse = validation.isUserTheTeacherOfCourse(userID,courseID);

        if (!validCourse || !isTeacherOfcourse) {
            return marks;
        } else {
            resultSet = statement.executeQuery( sql: "select * from user_course where course_course_id=" + courseID

            while (resultSet.next()){
                int mid = Integer.parseInt(resultSet.getString( columnLabel: "midterm_grade"));
                int assignment = Integer.parseInt(resultSet.getString( columnLabel: "assignments_grade"));
                int finalexam = Integer.parseInt(resultSet.getString( columnLabel: "final_exam_grades"));
                marks.add(mid+assignment+finalexam);
            }
        }
    }
    catch (Exception e){
        System.out.println(e);
        marks.clear();
    }
    return marks;
}
```

➢ **getCourseNameByID**:

- This function offers a streamlined approach to acquiring a user's course name.
- It fetches the course name associated with a provided course ID, facilitating efficient display of marks to students.

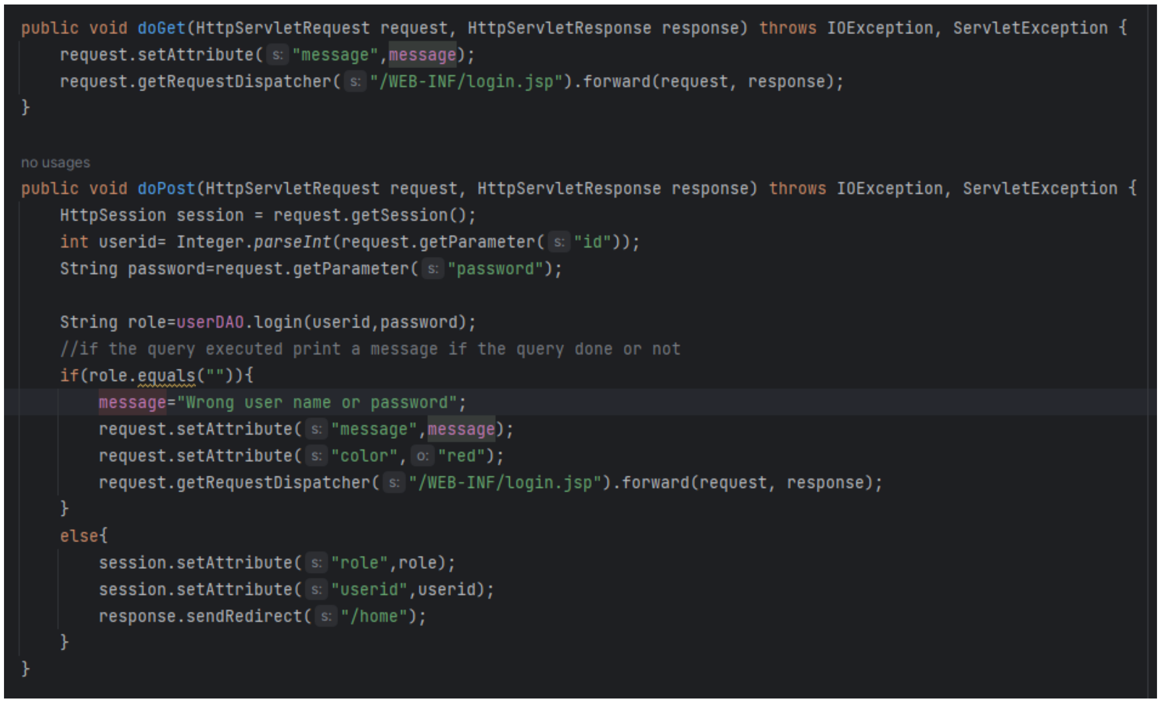


```
no usages
public String getCourseNameByID(int id) throws SQLException {
    String query="select * from course where course_id ="+id;
    ResultSet resultSet = statement.executeQuery(query);
    String name = "";
    while (resultSet.next()){
        name = resultSet.getString( columnLabel: "course_name");
    }
    return name;
}
```

-Servlets

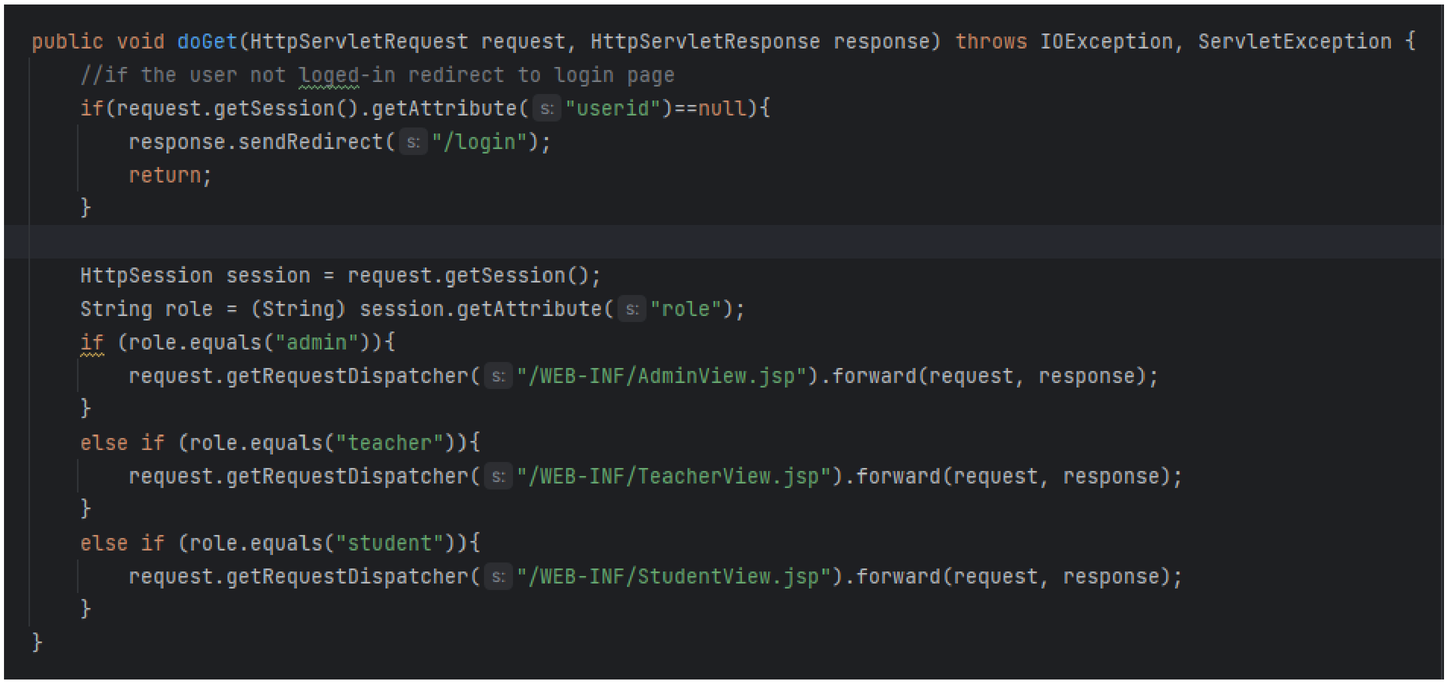
1. Login Servlet:

- The Login Servlet acquires the user's ID and password from URL parameters.
- It forwards these credentials to the login action function.
- If the function returns a valid role, the role and user ID are stored in session attributes for authentication and authorization.
- Upon successful validation, users are redirected to the home page.
- In case of invalid ID or password, the servlet conveys an appropriate message to the JSP indicating the error.



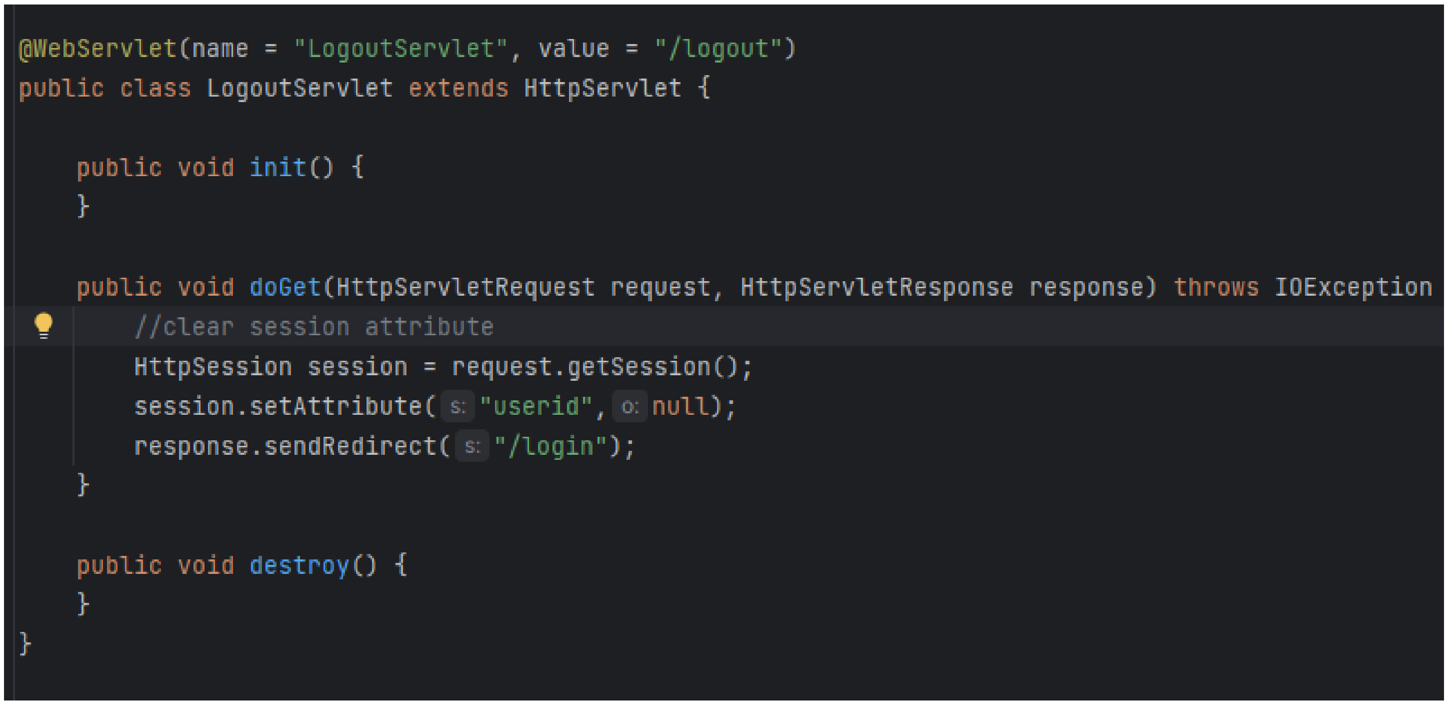
```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    request.setAttribute( s: "message",message);
    request.getRequestDispatcher( s: "/WEB-INF/login.jsp").forward(request, response);
}

no usages
public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    HttpSession session = request.getSession();
    int userid= Integer.parseInt(request.getParameter( s: "id"));
    String password=request.getParameter( s: "password");

    String role=userDAO.login(userid,password);
    //if the query executed print a message if the query done or not
    if(role.equals("")){
        message="Wrong user name or password";
        request.setAttribute( s: "message",message);
        request.setAttribute( s: "color", o: "red");
        request.getRequestDispatcher( s: "/WEB-INF/login.jsp").forward(request, response);
    }
    else{
        session.setAttribute( s: "role",role);
        session.setAttribute( s: "userid",userid);
        response.sendRedirect( s: "/home");
    }
}
```

2. **Home Page Servlet:**
    - The Home Page Servlet's GET method responds with a role-specific home page based on the user's role.



```java
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    //if the user not loged-in redirect to login page
    if(request.getSession().getAttribute("userid")==null){
        response.sendRedirect("/login");
        return;
    }

    HttpSession session = request.getSession();
    String role = (String) session.getAttribute("role");
    if (role.equals("admin")){
        request.getRequestDispatcher("/WEB-INF/AdminView.jsp").forward(request, response);
    }
    else if (role.equals("teacher")){
        request.getRequestDispatcher("/WEB-INF/TeacherView.jsp").forward(request, response);
    }
    else if (role.equals("student")){
        request.getRequestDispatcher("/WEB-INF/StudentView.jsp").forward(request, response);
    }
}
```

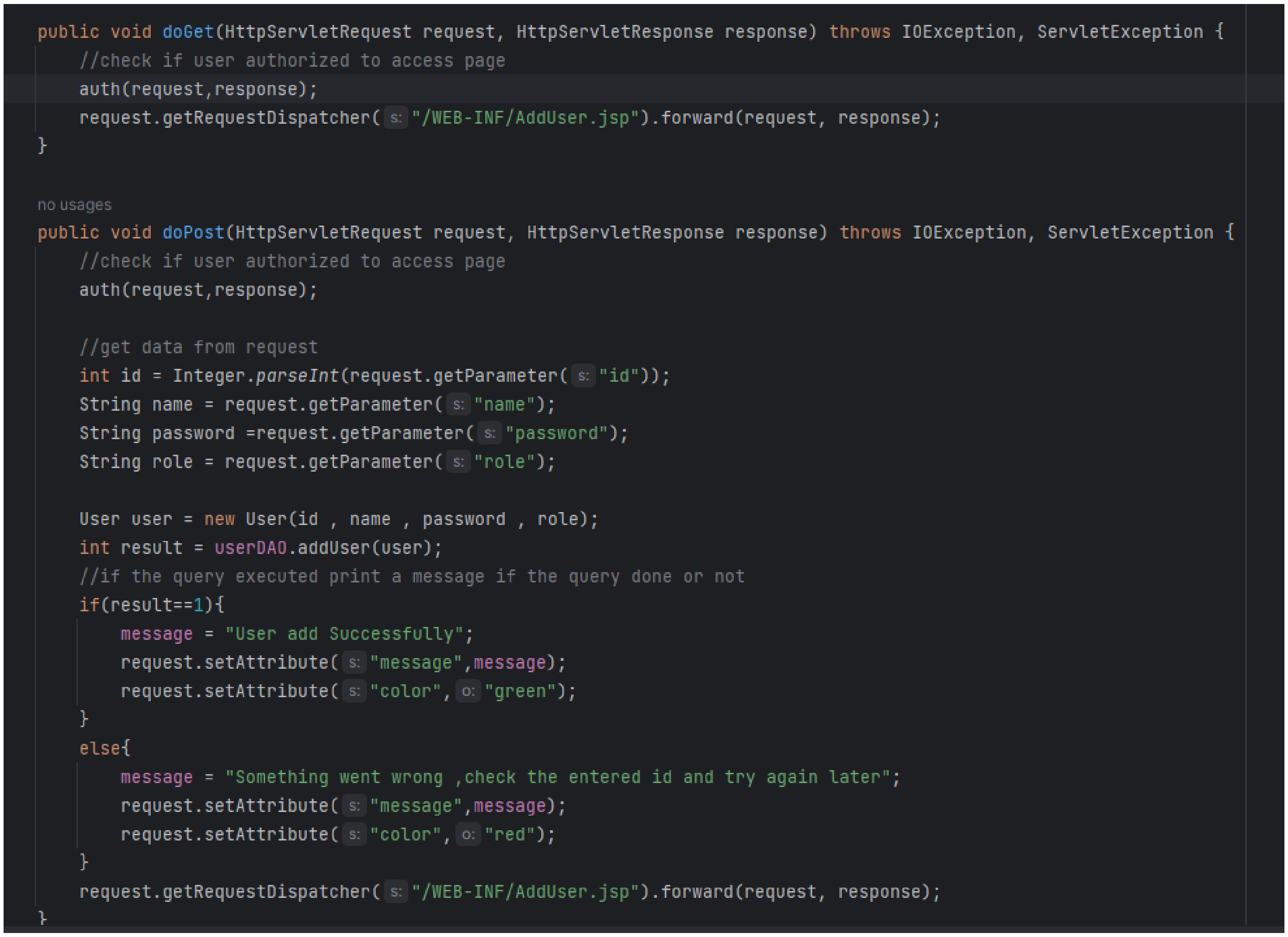
3. **Logout Servlet:**
    - This servlet facilitates secure logout by clearing session attributes.
    - After clearing the attributes, users are redirected to the login page.



```java
@WebServlet(name = "LogoutServlet", value = "/logout")
public class LogoutServlet extends HttpServlet {

    public void init() {
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException
        //clear session attribute
        HttpSession session = request.getSession();
        session.setAttribute("userid", null);
        response.sendRedirect("/login");
    }

    public void destroy() {
    }
}
```

4. Add User Servlet:

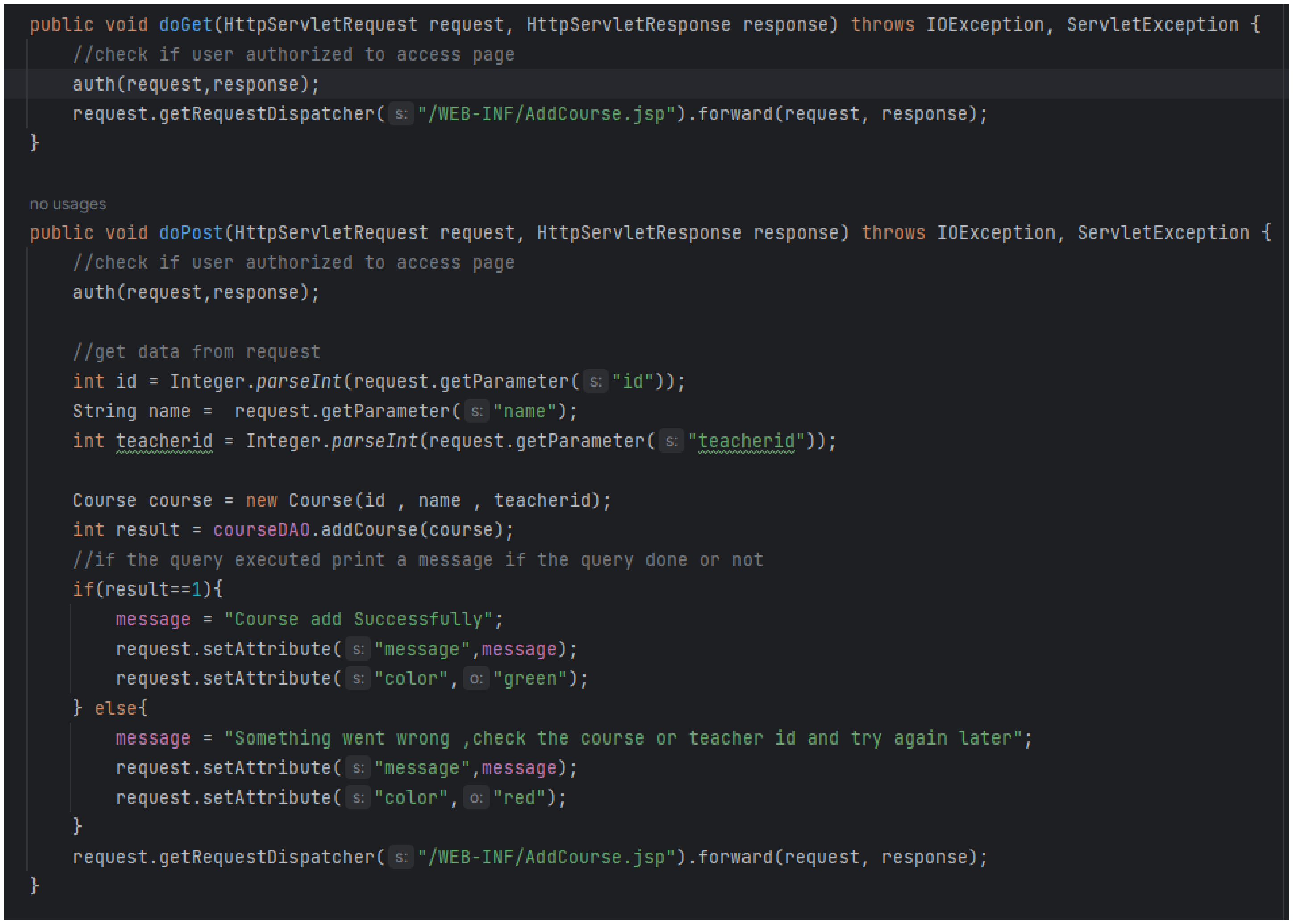
- This servlet serves the purpose of adding users to the system.
- Its GET method provides the add user page.
- During the POST method, user data is extracted from URL parameters, forming a user object.
- The object is then forwarded to the action function.
- If the function returns 0, indicating unsuccessful user addition, or 1 for successful addition, the servlet relays the corresponding message to the JSP.



```java
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    //check if user authorized to access page
    auth(request,response);
    request.getRequestDispatcher( s: "/WEB-INF/AddUser.jsp").forward(request, response);
}

no usages
public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    //check if user authorized to access page
    auth(request,response);

    //get data from request
    int id = Integer.parseInt(request.getParameter( s: "id"));
    String name = request.getParameter( s: "name");
    String password =request.getParameter( s: "password");
    String role = request.getParameter( s: "role");

    User user = new User(id , name , password , role);
    int result = userDAO.addUser(user);
    //if the query executed print a message if the query done or not
    if(result==1){
        message = "User add Successfully";
        request.setAttribute( s: "message",message);
        request.setAttribute( s: "color", o: "green");
    }
    else{
        message = "Something went wrong ,check the entered id and try again later";
        request.setAttribute( s: "message",message);
        request.setAttribute( s: "color", o: "red");
    }
    request.getRequestDispatcher( s: "/WEB-INF/AddUser.jsp").forward(request, response);
}
```

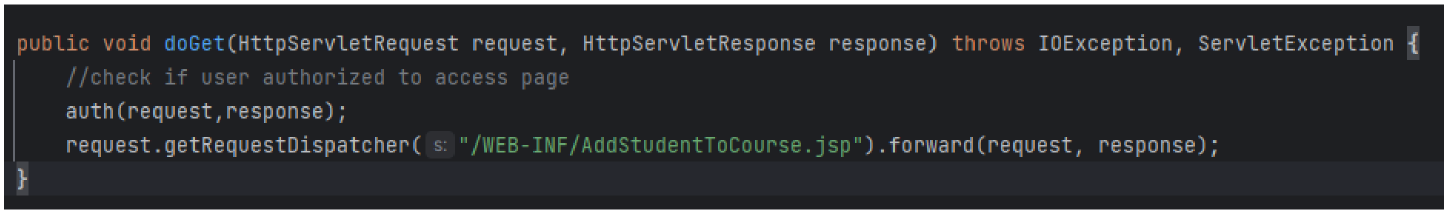
5. Add Course Servlet:

- Similar to the add user servlet, this servlet handles course addition.
- GET method provides the add course page, while the POST method processes URL parameters to create a course object.
- The action function processes the course object, returning 0 for failure or 1 for successful addition.
- The servlet conveys the outcome message to the JSP.

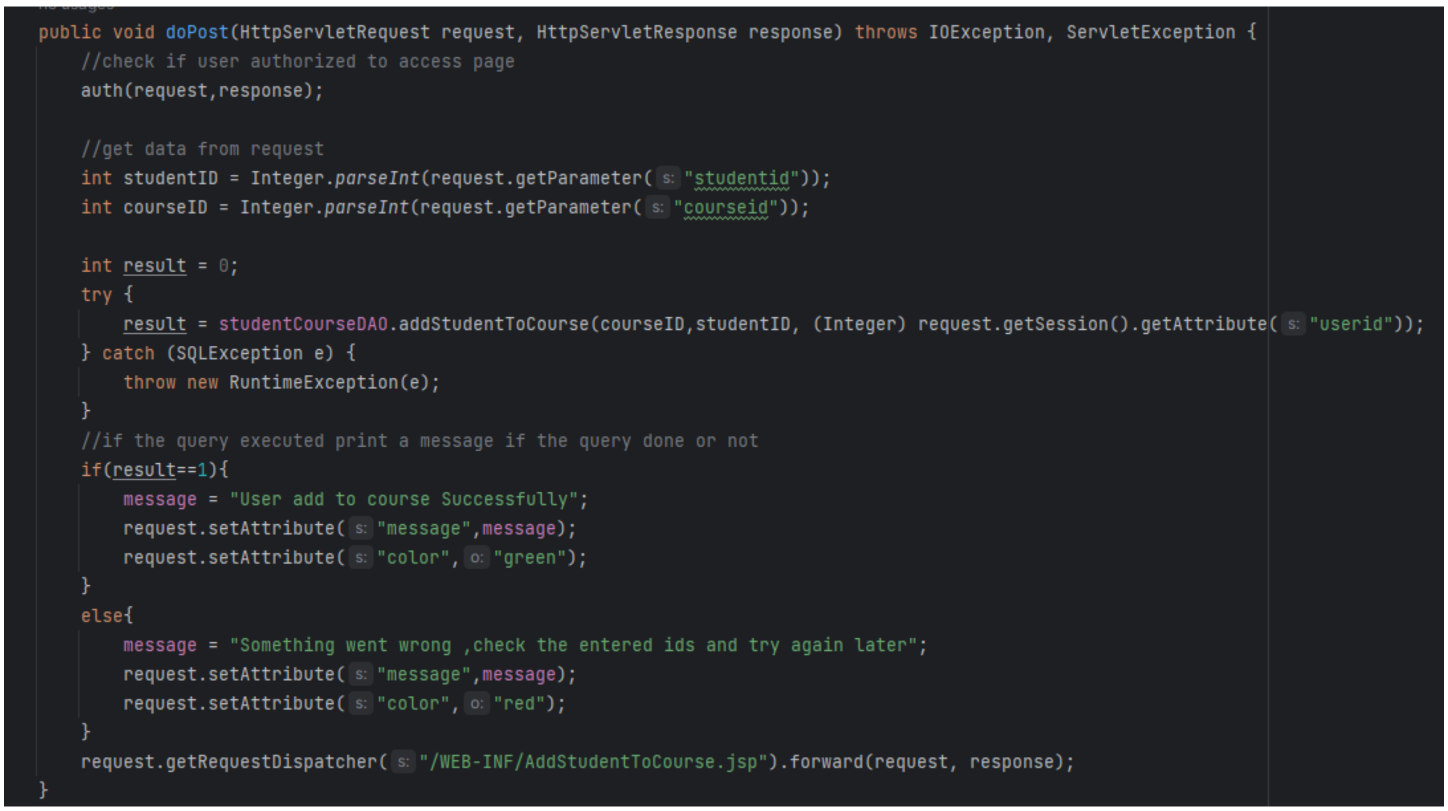


```java
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    //check if user authorized to access page
    auth(request,response);
    request.getRequestDispatcher( s: "/WEB-INF/AddCourse.jsp").forward(request, response);
}

no usages
public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    //check if user authorized to access page
    auth(request,response);

    //get data from request
    int id = Integer.parseInt(request.getParameter( s: "id"));
    String name =  request.getParameter( s: "name");
    int teacherid = Integer.parseInt(request.getParameter( s: "teacherid"));

    Course course = new Course(id , name , teacherid);
    int result = courseDAO.addCourse(course);
    //if the query executed print a message if the query done or not
    if(result==1){
        message = "Course add Successfully";
        request.setAttribute( s: "message",message);
        request.setAttribute( s: "color", o: "green");
    } else{
        message = "Something went wrong ,check the course or teacher id and try again later";
        request.setAttribute( s: "message",message);
        request.setAttribute( s: "color", o: "red");
    }
    request.getRequestDispatcher( s: "/WEB-INF/AddCourse.jsp").forward(request, response);
}
```

6. **Add Student to Course Servlet:**
    - Handling student enrollment in courses, this servlet follows the structure of previous ones.
    - The GET method provides the student to course page, while the POST method extracts URL parameters for a studentCourse object.
    - The action function processes the object, returning 0 or 1 based on success.
    - The servlet forwards the outcome message to the JSP.

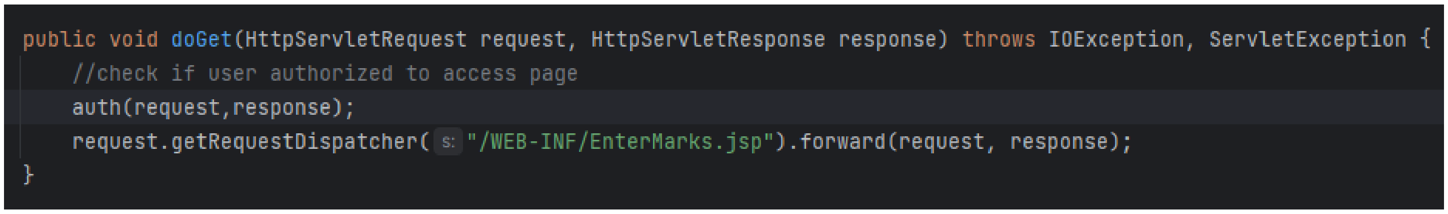


```java
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    //check if user authorized to access page
    auth(request,response);
    request.getRequestDispatcher("/WEB-INF/AddStudentToCourse.jsp").forward(request, response);
}
```

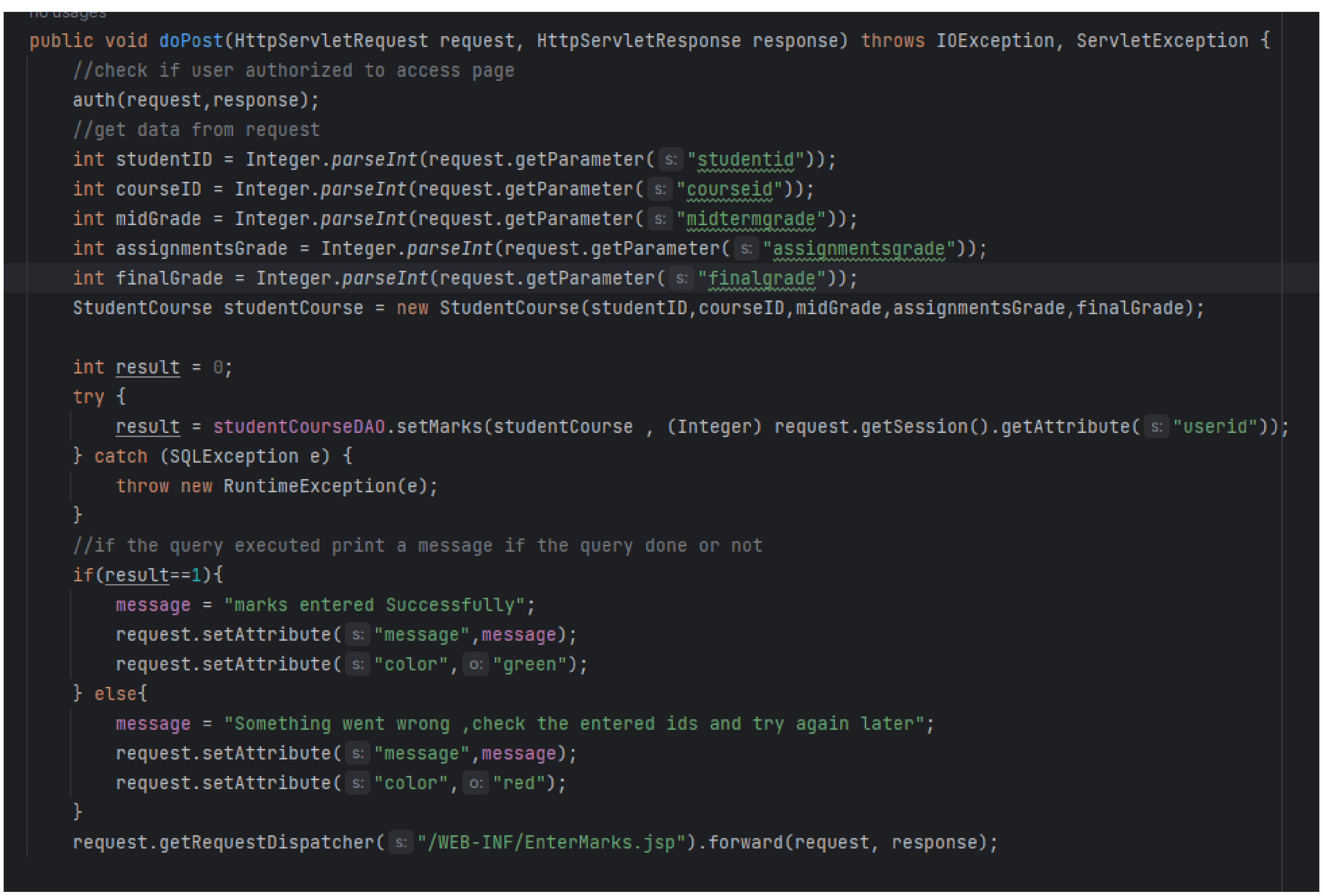


```java
public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    //check if user authorized to access page
    auth(request,response);

    //get data from request
    int studentID = Integer.parseInt(request.getParameter("studentid"));
    int courseID = Integer.parseInt(request.getParameter("courseid"));

    int result = 0;
    try {
        result = studentCourseDAO.addStudentToCourse(courseID,studentID, (Integer) request.getSession().getAttribute("userid"));
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    //if the query executed print a message if the query done or not
    if(result==1){
        message = "User add to course Successfully";
        request.setAttribute("message",message);
        request.setAttribute("color", "green");
    }
    else{
        message = "Something went wrong ,check the entered ids and try again later";
        request.setAttribute("message",message);
        request.setAttribute("color", "red");
    }
    request.getRequestDispatcher("/WEB-INF/AddStudentToCourse.jsp").forward(request, response);
}
```

7. **Enter Marks Servlet:**
   - This servlet handles the addition of marks for students.
   - The GET method provides the set marks page, and the POST method processes URL parameters into a studentCourse object.
   - The action function processes the object, returning 0 or 1 based on the success of mark addition.
   - The servlet sends the outcome message to the JSP.



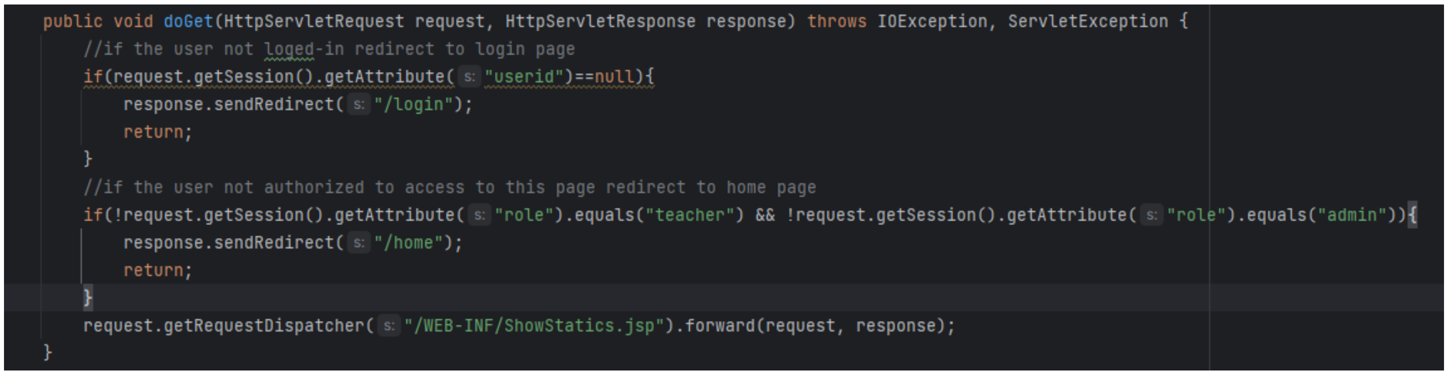
```java
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    //check if user authorized to access page
    auth(request,response);
    request.getRequestDispatcher( s: "/WEB-INF/EnterMarks.jsp").forward(request, response);
}
```



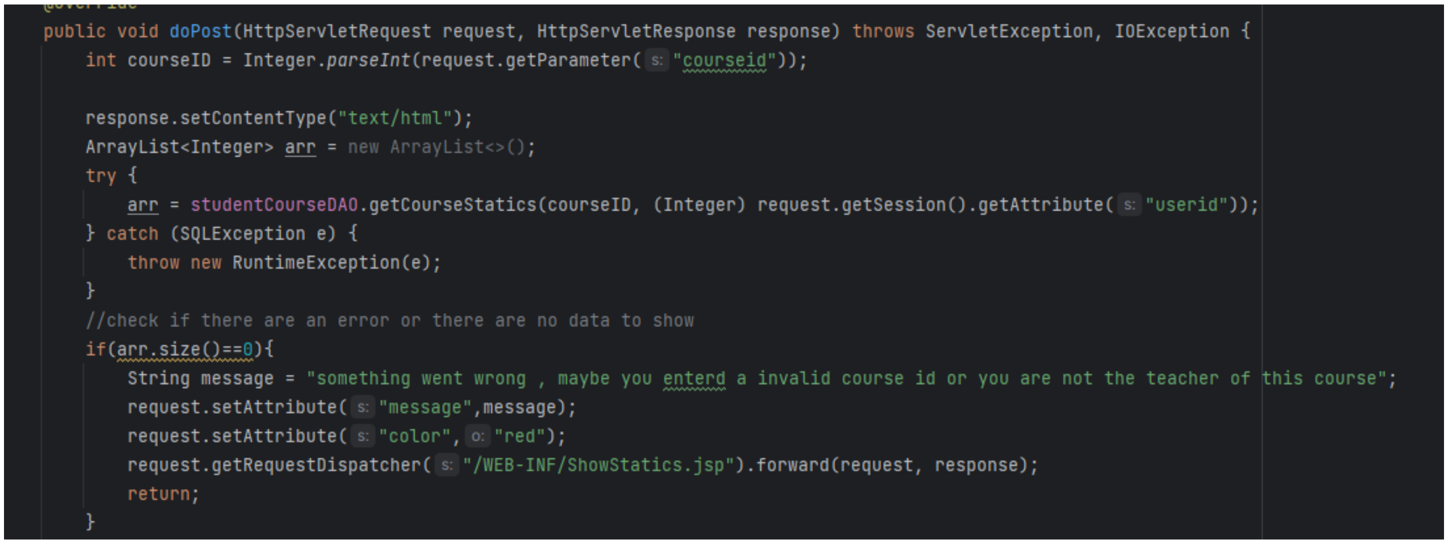
```java
no usages
public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    //check if user authorized to access page
    auth(request,response);
    //get data from request
    int studentID = Integer.parseInt(request.getParameter( s: "studentid"));
    int courseID = Integer.parseInt(request.getParameter( s: "courseid"));
    int midGrade = Integer.parseInt(request.getParameter( s: "midtermgrade"));
    int assignmentsGrade = Integer.parseInt(request.getParameter( s: "assignmentsgrade"));
    int finalGrade = Integer.parseInt(request.getParameter( s: "finalgrade"));
    StudentCourse studentCourse = new StudentCourse(studentID,courseID,midGrade,assignmentsGrade,finalGrade);

    int result = 0;
    try {
        result = studentCourseDAO.setMarks(studentCourse , (Integer) request.getSession().getAttribute( s: "userid"));
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    //if the query executed print a message if the query done or not
    if(result==1){
        message = "marks entered Successfully";
        request.setAttribute( s: "message",message);
        request.setAttribute( s: "color", o: "green");
    } else{
        message = "Something went wrong ,check the entered ids and try again later";
        request.setAttribute( s: "message",message);
        request.setAttribute( s: "color", o: "red");
    }
    request.getRequestDispatcher( s: "/WEB-INF/EnterMarks.jsp").forward(request, response);
```

8. Show Marks Servlet:

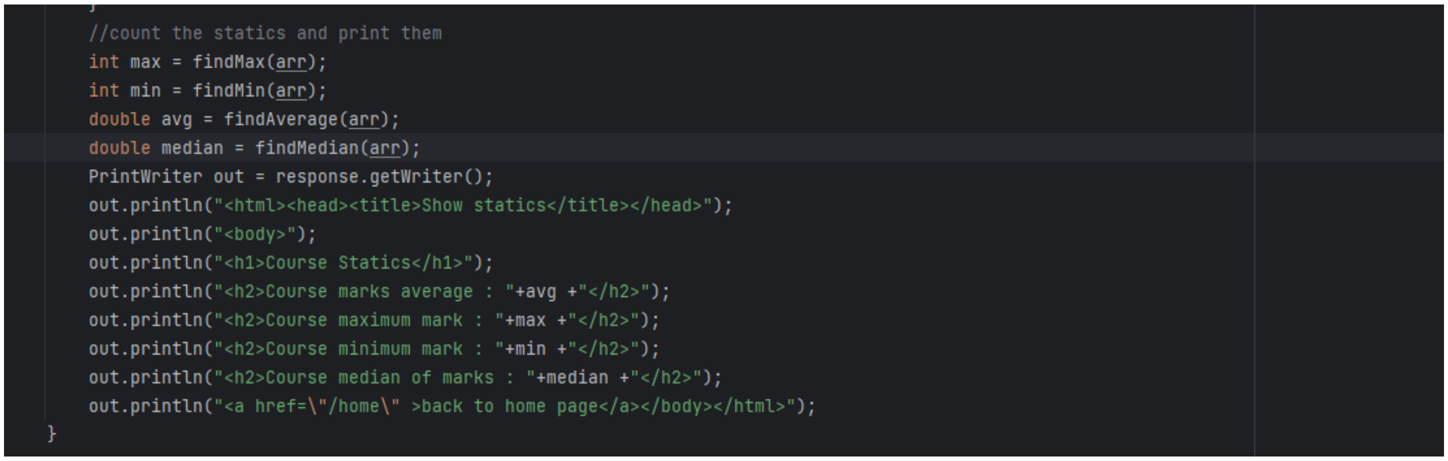
- This servlet retrieves all marks for the logged-in student, with a GET method.



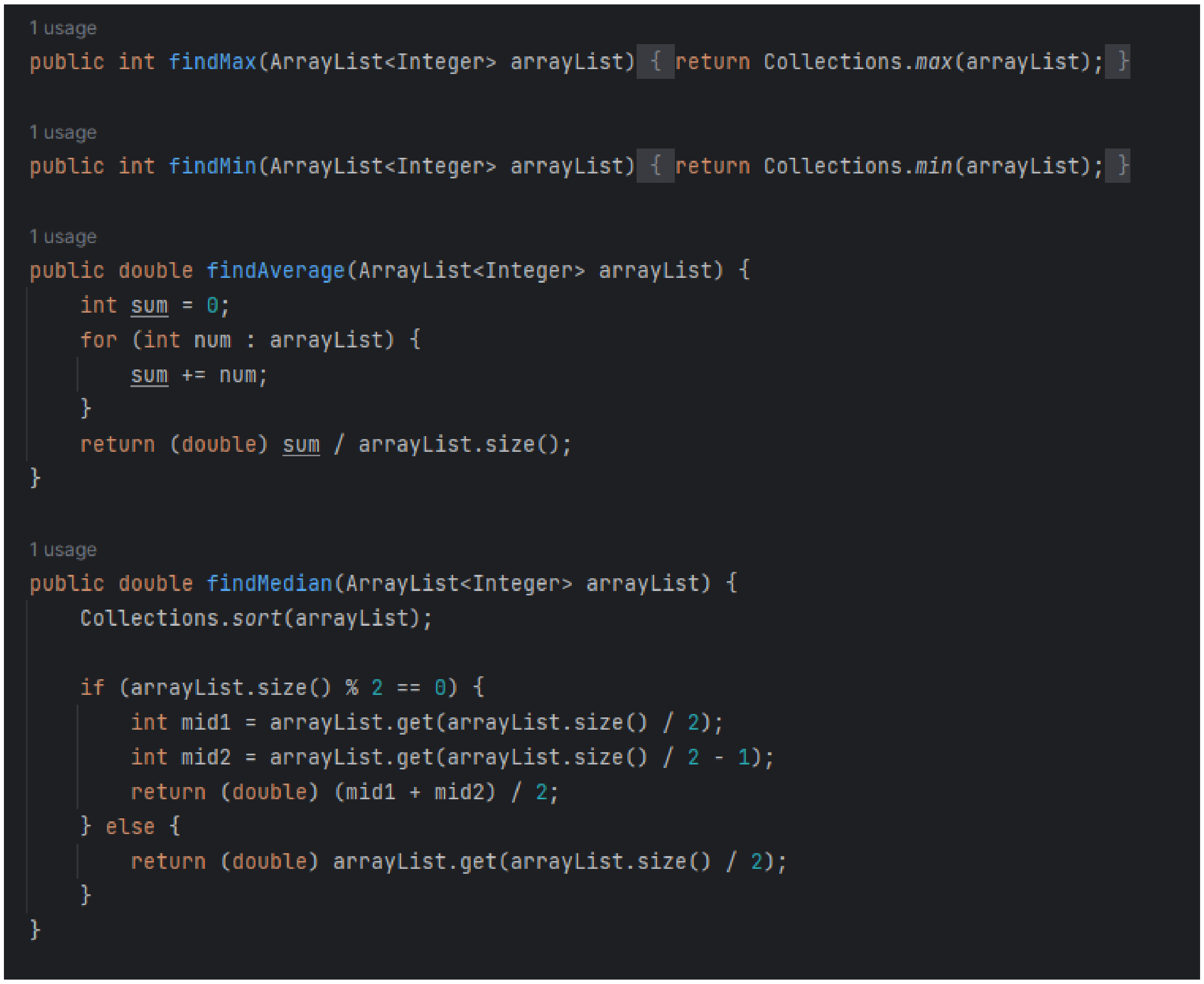
```java
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    //if the user not loged-in redirect to login page
    if(request.getSession().getAttribute("userid")==null){
        response.sendRedirect("/login");
        return;
    }
    //if the user not authorized to access to this page redirect to home page
    if(!request.getSession().getAttribute("role").equals("teacher") && !request.getSession().getAttribute("role").equals("admin")){
        response.sendRedirect("/home");
        return;
    }
    request.getRequestDispatcher("/WEB-INF/ShowStatics.jsp").forward(request, response);
}
```



```java
@Override
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    int courseID = Integer.parseInt(request.getParameter("courseid"));

    response.setContentType("text/html");
    ArrayList<Integer> arr = new ArrayList<>();
    try {
        arr = studentCourseDAO.getCourseStatics(courseID, (Integer) request.getSession().getAttribute("userid"));
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    //check if there are an error or there are no data to show
    if(arr.size()==0){
        String message = "something went wrong , maybe you enterd a invalid course id or you are not the teacher of this course";
        request.setAttribute("message",message);
        request.setAttribute("color","red");
        request.getRequestDispatcher("/WEB-INF/ShowStatics.jsp").forward(request, response);
        return;
    }
```



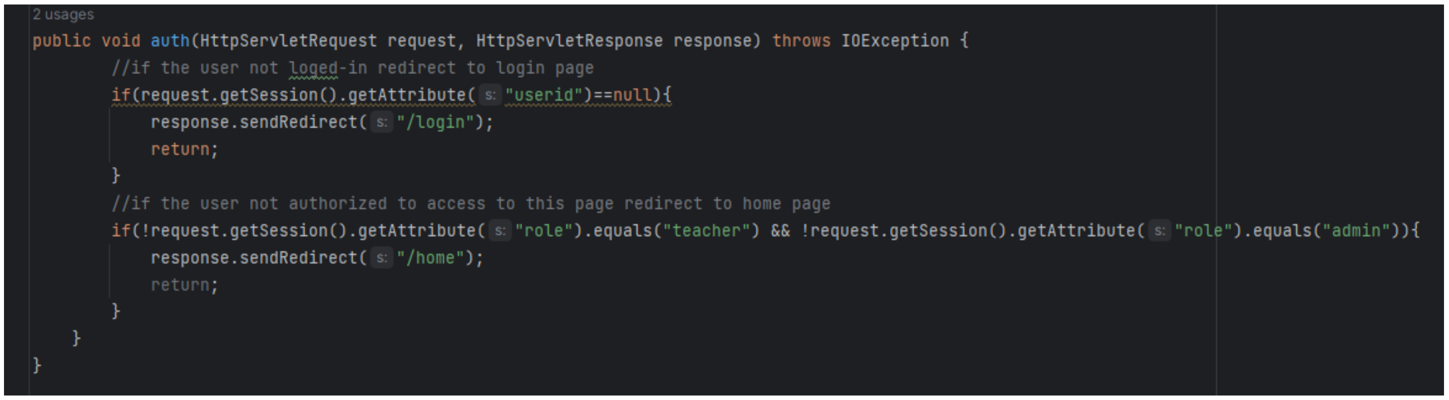
```java
    }
    //count the statics and print them
    int max = findMax(arr);
    int min = findMin(arr);
    double avg = findAverage(arr);
    double median = findMedian(arr);
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>Show statics</title></head>");
    out.println("<body>");
    out.println("<h1>Course Statics</h1>");
    out.println("<h2>Course marks average : "+avg +"</h2>");
    out.println("<h2>Course maximum mark : "+max +"</h2>");
    out.println("<h2>Course minimum mark : "+min +"</h2>");
    out.println("<h2>Course median of marks : "+median +"</h2>");
    out.println("<a href=\"/home\" >back to home page</a></body></html>");
}
```



```java
1 usage
public int findMax(ArrayList<Integer> arrayList) { return Collections.max(arrayList); }

1 usage
public int findMin(ArrayList<Integer> arrayList) { return Collections.min(arrayList); }

1 usage
public double findAverage(ArrayList<Integer> arrayList) {
    int sum = 0;
    for (int num : arrayList) {
        sum += num;
    }
    return (double) sum / arrayList.size();
}

1 usage
public double findMedian(ArrayList<Integer> arrayList) {
    Collections.sort(arrayList);

    if (arrayList.size() % 2 == 0) {
        int mid1 = arrayList.get(arrayList.size() / 2);
        int mid2 = arrayList.get(arrayList.size() / 2 - 1);
        return (double) (mid1 + mid2) / 2;
    } else {
        return (double) arrayList.get(arrayList.size() / 2);
    }
}
```

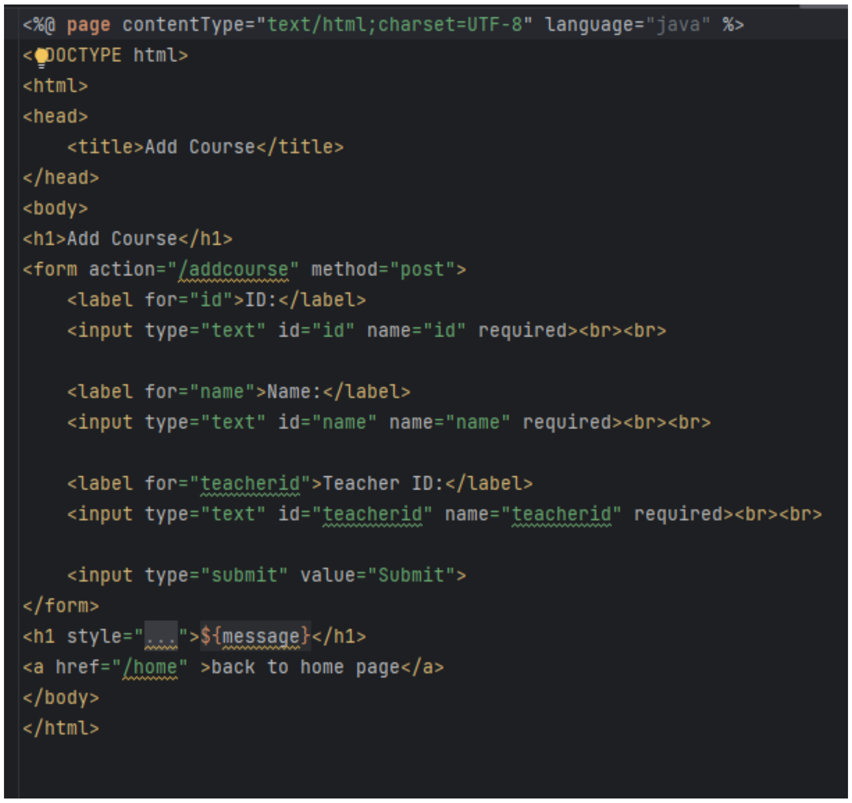
All servlets incorporate checks to ensure user authentication before accessing pages. Unauthorized users are redirected to the login or home page, depending on their status. This comprehensive arrangement ensures secure and seamless user interaction within the system.

Sample of that code for authentication and authorization

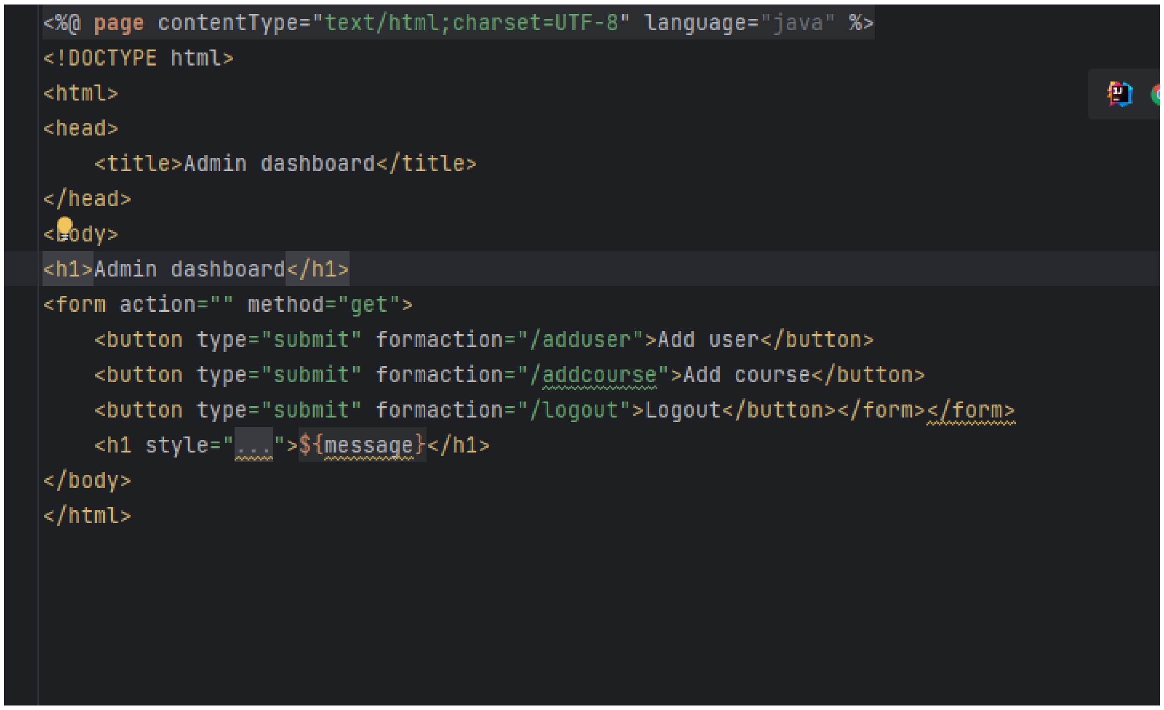


```java
2 usages
public void auth(HttpServletRequest request, HttpServletResponse response) throws IOException {
        //if the user not loged-in redirect to login page
        if(request.getSession().getAttribute( s: "userid")==null){
            response.sendRedirect( s: "/login");
            return;
        }
        //if the user not authorized to access to this page redirect to home page
        if(!request.getSession().getAttribute( s: "role").equals("teacher") && !request.getSession().getAttribute( s: "role").equals("admin")){
            response.sendRedirect( s: "/home");
            return;
        }
    }
}
```

Sample of jsp files



```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>Add Course</title>
</head>
<body>
<h1>Add Course</h1>
<form action="/addcourse" method="post">
    <label for="id">ID:</label>
    <input type="text" id="id" name="id" required><br><br>

    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>

    <label for="teacherid">Teacher ID:</label>
    <input type="text" id="teacherid" name="teacherid" required><br><br>

    <input type="submit" value="Submit">
</form>
<h1 style="...">${message}</h1>
<a href="/home" >back to home page</a>
</body>
</html>
```



```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>Admin dashboard</title>
</head>
<body>
<h1>Admin dashboard</h1>
<form action="" method="get">
    <button type="submit" formaction="/adduser">Add user</button>
    <button type="submit" formaction="/addcourse">Add course</button>
    <button type="submit" formaction="/logout">Logout</button></form></form>
    <h1 style="...">${message}</h1>
</body>
</html>
```

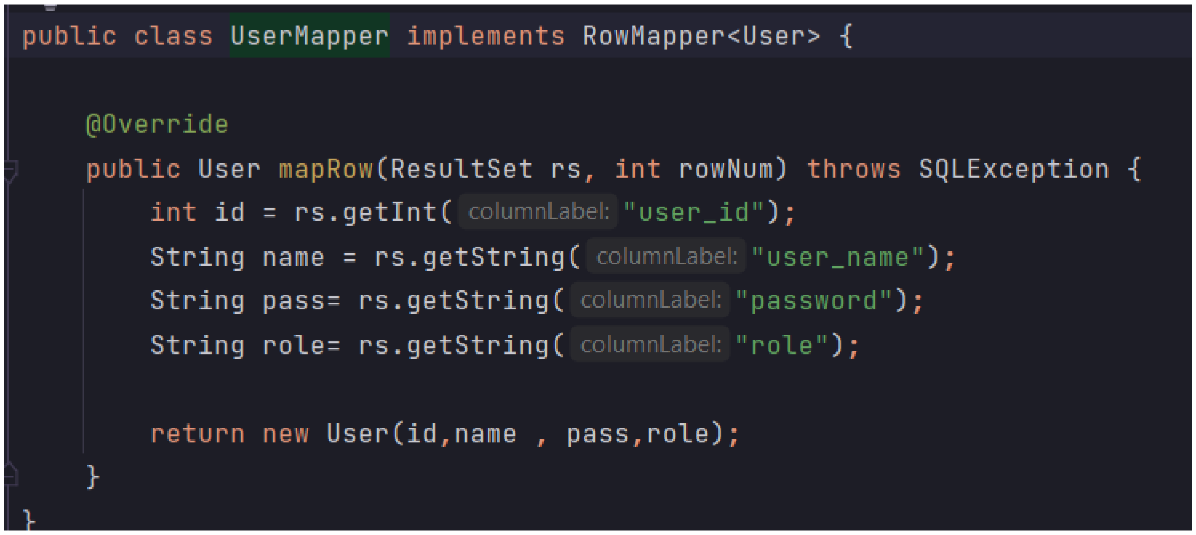
C. **Spring Boot Project with REST API:**

There is a little bit different between servlet and spring boot , so I will talk about duplicate code between them , I only will mention the thing the difference between them, so I will take about the jdbc template and the something that changed between servlet and controller.

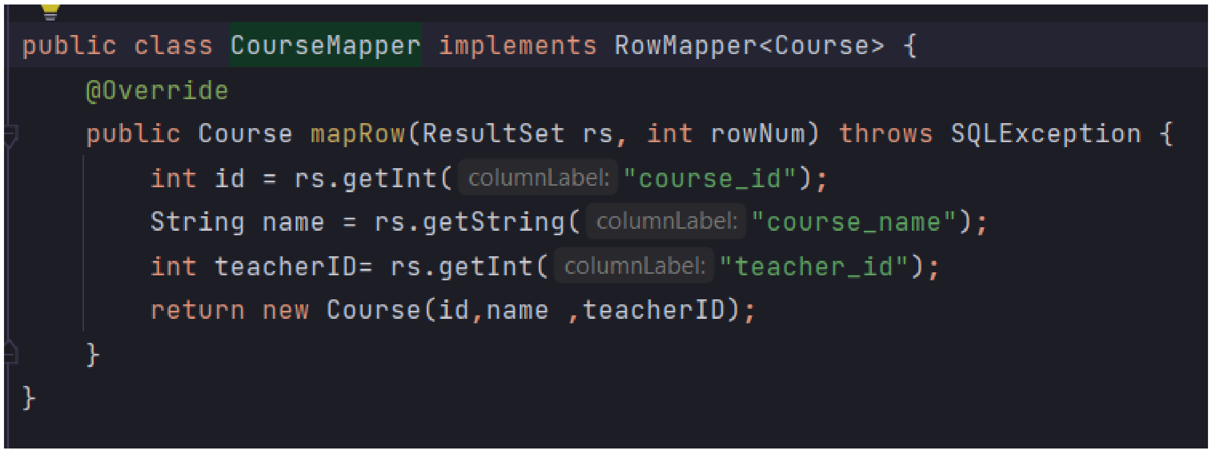
1. Difference between jdbc template and traditional jdbc

   So first the quere will not changed,we have to user mapped to map the data from database to list of object so that we can use it easer

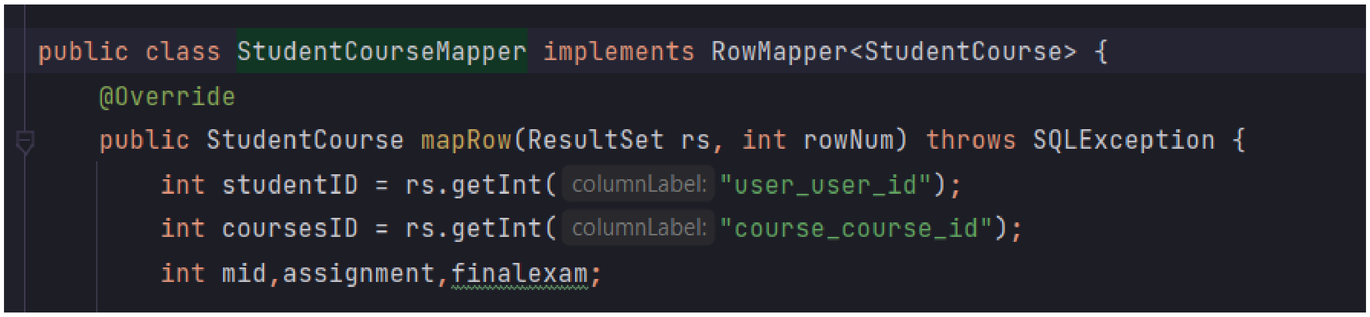
   -the mappers



```
public class UserMapper implements RowMapper<User> {

    @Override
    public User mapRow(ResultSet rs, int rowNum) throws SQLException {
        int id = rs.getInt( columnLabel: "user_id");
        String name = rs.getString( columnLabel: "user_name");
        String pass= rs.getString( columnLabel: "password");
        String role= rs.getString( columnLabel: "role");

        return new User(id,name , pass,role);
    }
}
```



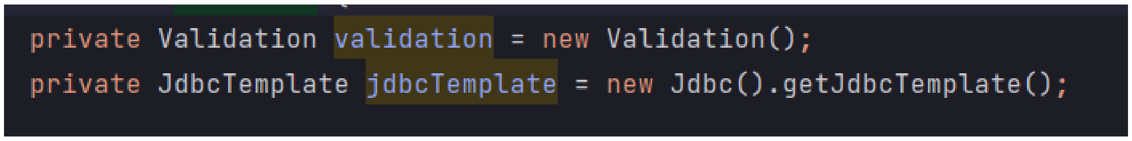
```
public class CourseMapper implements RowMapper<Course> {
    @Override
    public Course mapRow(ResultSet rs, int rowNum) throws SQLException {
        int id = rs.getInt( columnLabel: "course_id");
        String name = rs.getString( columnLabel: "course_name");
        int teacherID= rs.getInt( columnLabel: "teacher_id");
        return new Course(id,name ,teacherID);
    }
}
```



```
public class StudentCourseMapper implements RowMapper<StudentCourse> {
    @Override
    public StudentCourse mapRow(ResultSet rs, int rowNum) throws SQLException {
        int studentID = rs.getInt( columnLabel: "user_user_id");
        int coursesID = rs.getInt( columnLabel: "course_course_id");
        int mid,assignment,finalexam;
```

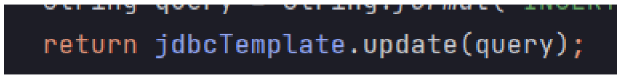
-In the action functions the execute of query only changed , so it will be like that :

- 1-first we will create a instance of jdbcTemplate.

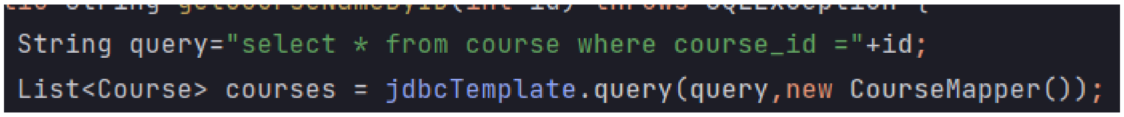


```java
private Validation validation = new Validation();
private JdbcTemplate jdbcTemplate = new Jdbc().getJdbcTemplate();
```

- 2-then we will use it execute the query



```java
return jdbcTemplate.update(query);
```



```java
String query="select * from course where course_id ="+id;
List<Course> courses = jdbcTemplate.query(query,new CourseMapper());
```

-Controllers

The controller will be the same code of servlet , the change will that many request handlers in same controller. And the request handler will return a string.

UserController:

1. Endpoint: /login
   - Handles user login.
   - Returns an appropriate string.
2. Endpoint: /logout
   - Facilitates user logout.
   - Returns a relevant string.
3. Endpoint: /adduser
   - Manages user addition.
   - Returns a corresponding string.

CourseController:

1. Endpoint: /addcourse
    - Takes care of course addition.
    - Returns a suitable string.

TeacherController:

2. Endpoint: /showstatics
    - Presents course statistics for teachers.
    - Returns the required string.

StudentController:

3. Endpoint: /showmarks
    - Provides students with their marks.
    - Returns the appropriate string.

StudentCourseController:

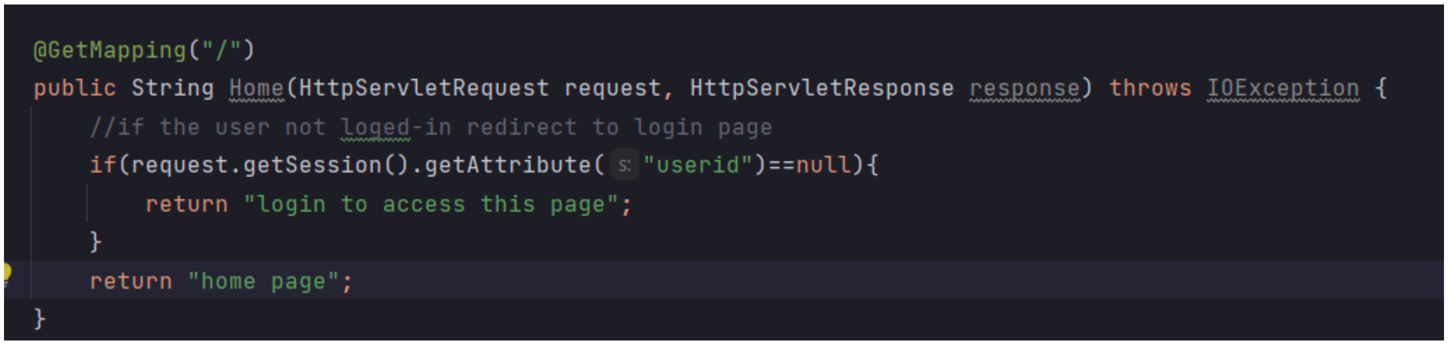
Endpoint: /addusercourse

- Handles the addition of students to courses.
- Returns the expected string.

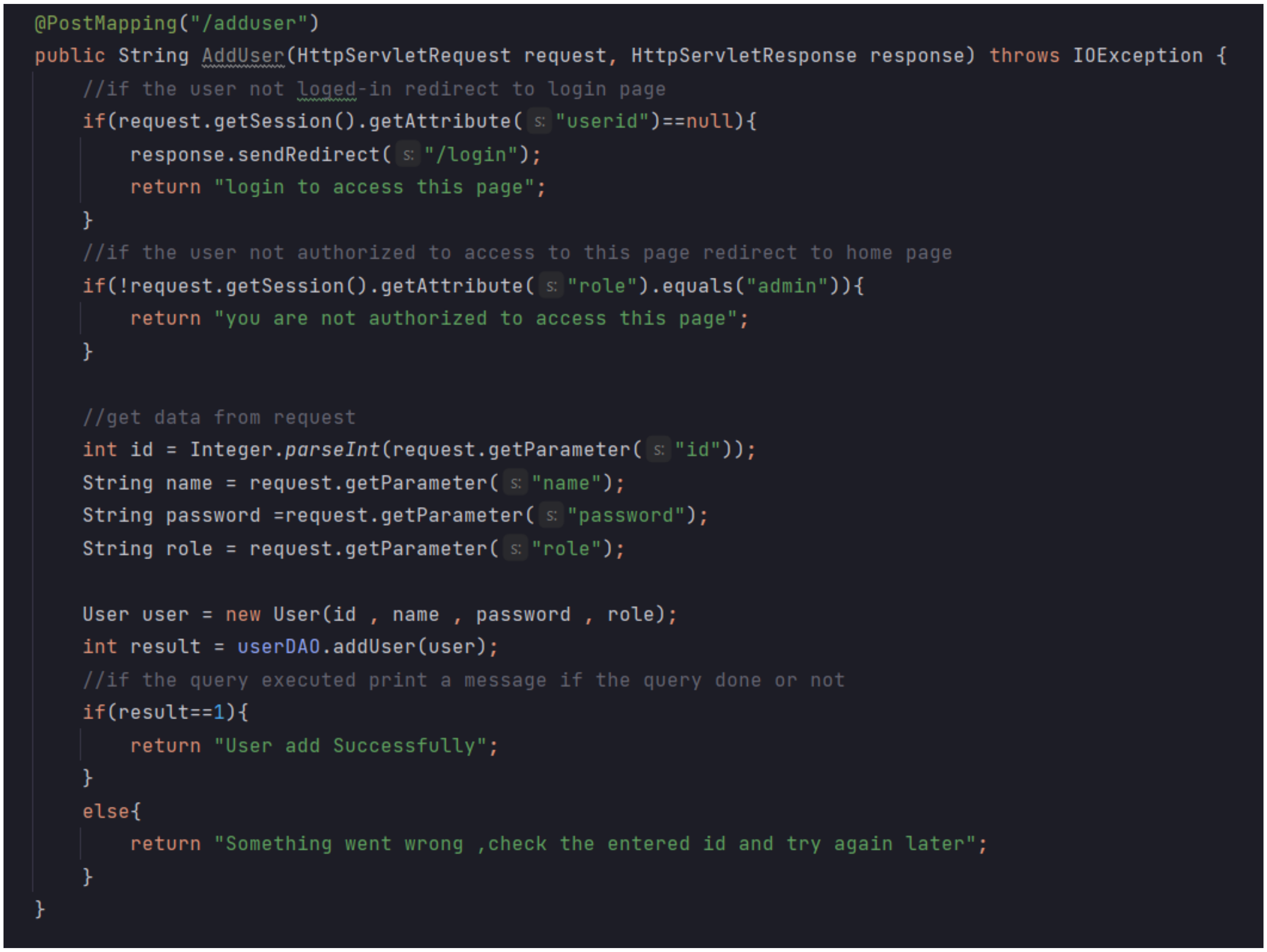
Endpoint: /entermarks

- Manages the input of marks for students.
- Returns a corresponding string.

Some examples of controllers

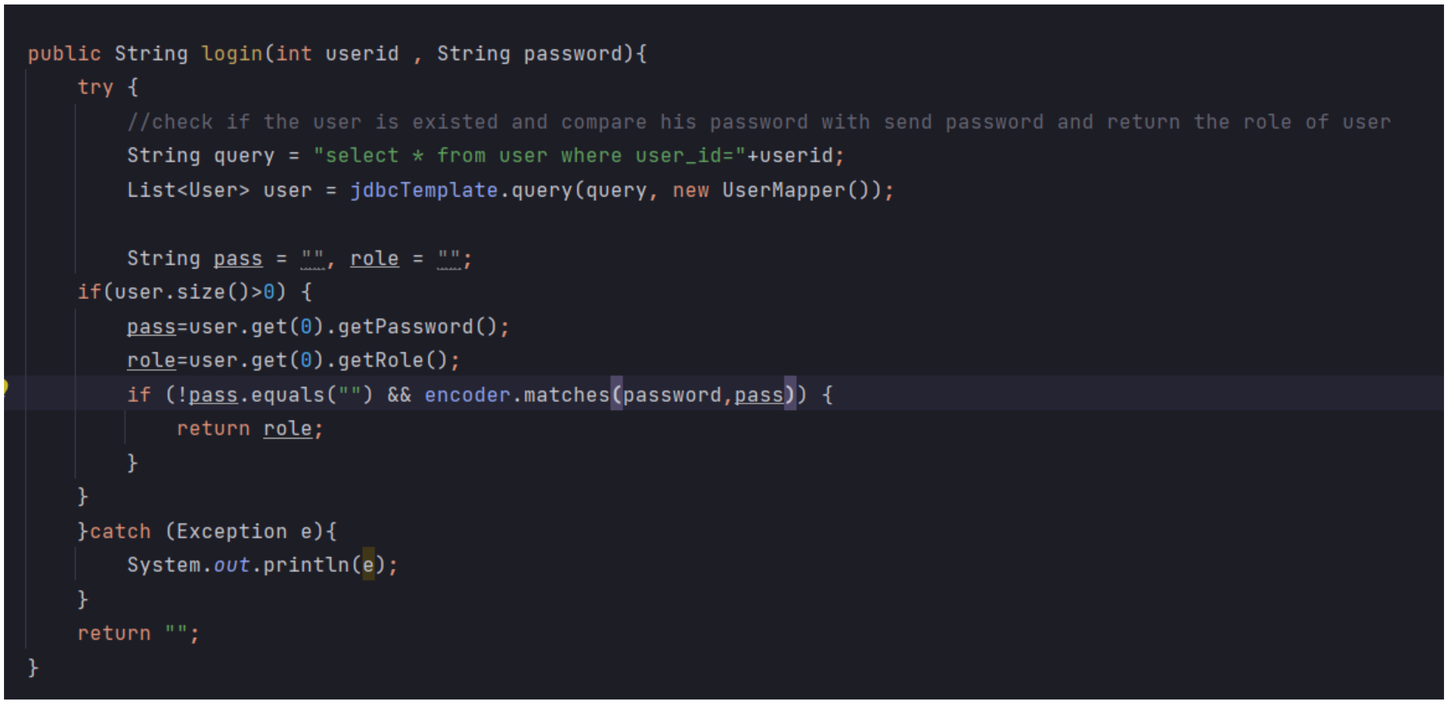


```java
@GetMapping("/")
public String Home(HttpServletRequest request, HttpServletResponse response) throws IOException {
    //if the user not loged-in redirect to login page
    if(request.getSession().getAttribute( s: "userid")==null){
        return "login to access this page";
    }
    return "home page";
}
```

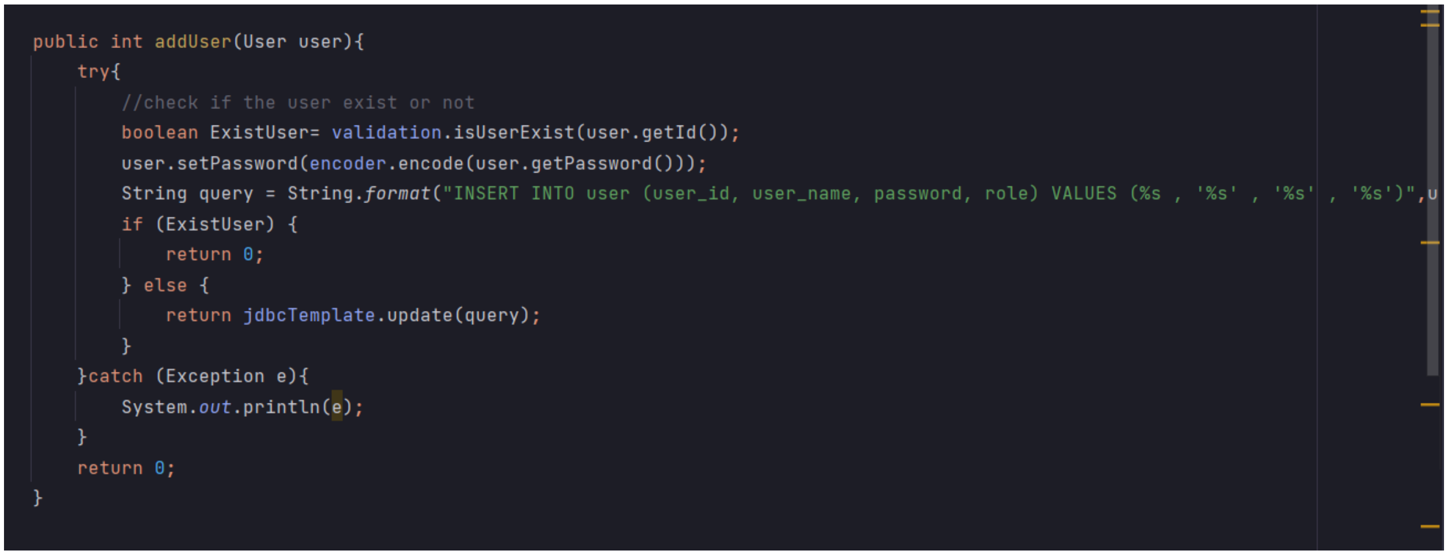


```java
@PostMapping("/adduser")
public String AddUser(HttpServletRequest request, HttpServletResponse response) throws IOException {
    //if the user not loged-in redirect to login page
    if(request.getSession().getAttribute( s: "userid")==null){
        response.sendRedirect( s: "/login");
        return "login to access this page";
    }
    //if the user not authorized to access to this page redirect to home page
    if(!request.getSession().getAttribute( s: "role").equals("admin")){
        return "you are not authorized to access this page";
    }

    //get data from request
    int id = Integer.parseInt(request.getParameter( s: "id"));
    String name = request.getParameter( s: "name");
    String password =request.getParameter( s: "password");
    String role = request.getParameter( s: "role");

    User user = new User(id , name , password , role);
    int result = userDAO.addUser(user);
    //if the query executed print a message if the query done or not
    if(result==1){
        return "User add Successfully";
    }
    else{
        return "Something went wrong ,check the entered id and try again later";
    }
}
```

And in the userDAO the change that happened that the password will be encrypted using spring security



```java
public String login(int userid , String password){
    try {
        //check if the user is existed and compare his password with send password and return the role of user
        String query = "select * from user where user_id="+userid;
        List<User> user = jdbcTemplate.query(query, new UserMapper());

        String pass = "", role = "";
    if(user.size()>0) {
        pass=user.get(0).getPassword();
        role=user.get(0).getRole();
        if (!pass.equals("") && encoder.matches(password,pass)) {
            return role;
        }
    }
    }catch (Exception e){
        System.out.println(e);
    }
    return "";
}
```



```java
public int addUser(User user){
    try{
        //check if the user exist or not
        boolean ExistUser= validation.isUserExist(user.getId());
        user.setPassword(encoder.encode(user.getPassword()));
        String query = String.format("INSERT INTO user (user_id, user_name, password, role) VALUES (%s , '%s' , '%s' , '%s')",u
        if (ExistUser) {
            return 0;
        } else {
            return jdbcTemplate.update(query);
        }
    }catch (Exception e){
        System.out.println(e);
    }
    return 0;
}
```

All controllers incorporate checks to ensure user authentication before accessing pages. Unauthorized users are redirected to the login or home page, depending on their status. This comprehensive arrangement ensures secure and seamless user interaction within the system.

Challenges and Solutions

My main challenges came while learning about sockets and servlets. These concepts seemed complex at first, but I tackled them by reading the documentation, watching videos, and searching online. Spring Boot wasn't as tough because I was already familiar with it. However, I struggled with spring security the most. It took a lot of time and searching, but eventually, I managed to understand and overcome that challenge too.

Security Considerations

In the context of sockets, I employed a simple authenticated variable to determine user authentication, which didn't offer extensive security. However, in servlets, I leveraged the HTTP session, integrated with Spring Security, to enhance the application's security. For Spring Boot applications, I took steps to bolster security by utilizing the Spring Security Crypto Module for password encryption and decryption. This multifaceted approach contributes to a more secure system overall.

Future Enhancements:

1. User-Friendly Interface: Enhance the app's interface for improved usability.
2. Security Enhancement: Strengthen security with measures like JWT implementation.
3. New Features: Introduce assignments and exams for enhanced functionality.
4. Teacher Visibility and Course Selection: Provide teacher profiles and streamline course registration.

Analytical Thinking

Spring :

Advantages: Comprehensive ecosystem, dependency injection, AOP, and data access. Simplified development with Spring Boot. Strong security through Spring Security.

Disadvantages: Complexity due to extensive features. Potential performance overhead in smaller projects.

Servlets:

Advantages: Foundation of Java web apps. Robust request handling, session management, and server-side processing. Scalability and platform independence.

Disadvantages: Complexity as apps grow. Session management challenges leading to memory leaks and concurrency issues.

Sockets:

Advantages: Real-time communication, low-latency interactions. Direct control over data exchange. Suitable for gaming and chat applications.

Disadvantages: Requires managing connection states, error handling, and security vulnerabilities. Not ideal for standardized communication.