Criterion C – Development

Table of contents:

Techniques Used:

(1) Java Servlet API

Used to enable the use of Java Servlets to handle backend business logic.

(1.1) Cookies implementation to remember logins

Cookies are used to implement the feature of remembering logins and subsequently logging out users after 14 days (Success Criteria D).

(2) Java Database Connectivity API (JDBC)

Used to establish a connection to the database and execute SQL queries to retrieve, change and add data.

(3) JavaServer Pages (JSP)

Used to handle the front end, that is the client side viewing. It incorporates the use of Java (scriptlet code) and HTML to generate dynamic web pages.

(3.1) JSP Standard Tag Library (JSTL)

JSTL allows the use of pre-formatted tags instead of Java scriptlet code, which increases readability of code and enables later updates to the code.

(4) Object Oriented Programming (OOP)

While Java is not considered to be a pure object oriented programming language, concepts from OOPs languages can still be implemented. The following concepts are implemented in the development of **'mph_scheduler':**

(4.1) Abstraction

Abstraction means only providing essential information to the user and hiding the background details (Moumita)[1]. In '**mph_scheduler**', the server hides how it processes the data it receives, the teachers and students just click the buttons, and enter input and receive the processed information in its final form.

[1] Moumita. "What is object-oriented programming (OOP)?" *Tutorials Point*, 14 Feb. 2018, www.tutorialspoint.com/What-is-object-oriented-programming-OOP. Accessed 29 Dec. 2020.

(4.2) Encapsulation

Encapsulation is the process of binding variables and properties as well as methods into one unit, and is used as a way to restrict access to certain variables (Moumita)[1]. Here, encapsulation is implemented through the use of classes (especially the data classes used to restrict access to information received from the database, explained in **III**).

(4.3) Polymorphism

Polymorphism is implemented using method overloading and method overriding (shown in **III**).

(4.4) Modularity

Modularity is the concept of writing modules or methods for code that is used frequently to reduce the complexity of code and make it more readable and understandable.

(4.5) Inheritance

"The ability to create a new class from an existing class is called Inheritance" (Moumita)[2]. This is implemented in the servlet class which 'extends' or inherits the HttpServlet Class (**II**).

(5) Dynamic Arrays (ArrayList)

ArrayLists store data in a dynamic array, i.e. an array with no size limit, where elements can be added and removed without knowing the initial size of the Array ("ArrayList in Java")[3]. ArrayLists are used in '**mph_scheduler**' to store the list of lessons that have no known end limit and are constantly changing in number.

(6) Try/Catch blocks to handle exceptions

try/catch and finally blocks are used to handle any exceptions a method or piece of code may throw and to perform any final clean-up of objects, such as closing JDBC java.sql objects.

[2] Moumita. "What is object-oriented programming (OOP)?" *Tutorials Point*, 14 Feb. 2018, www.tutorialspoint.com/What-is-object-oriented-programming-OOP. Accessed 29 Dec. 2020.
[3] "ArrayList in Java - javatpoint." *javatpoint*, www.javatpoint.com/java-arraylist. Accessed 30 Dec. 2020.

(7) Conditional Programming

Conditional programming is used to run a piece of code only when certain conditions are met. In '**mph_scheduler**' if, else if and else statements as well as switch statements to handle multiple conditions.

(8) SQL: Searching, filtering, inserting, editing and deletion

SQL queries are used to retrieve data from the database and filtering by certain columns. They are also used to insert new rows in the database, editing existing data and deleting rows.

(9) HTML & CSS for UI creation

I Database:

Using MySQL, the schema ‘mph_scheduling’ was created for the database with 5 tables. The entries in the table were randomly generated. The tables are as shown:

default_schedule :

Result Grid | Filter Rows: Search | Edit:

lesson_id	t_name	t_class	grade	date	time
93879	Olivia-Mae Barrett	Economics	D1	2021-04-21	08:00
93881	Tulisa Sharma	Sciences	M4	2021-04-21	11:20
93882	Rahul Cullen	Individuals & Societies	M5	2021-04-21	13:40
93884	Manraj Rowe	Individuals & Societies	M3	2021-04-22	08:00
NULL	NULL	NULL	NULL	NULL	NULL

teacher_details :

Result Grid | Filter Rows: Search | Edit:

id_teacher	teacher_name	teacher_class	teacher_grades
93877	Ammar Chadwick	Maths AI HL	D1 D2
93878	Saffron Costa	Maths AI SL	D1 D2
93879	Krystian Diaz	Visual Arts	D1 D2
93880	Lilian Moyer	Maths	M1 M2 M3 M4 M5
93881	Patsy Hancock	Maths AA HL	D1 D2
93882	Iram Robles	Business Management	D1 D2
93883	Dolly Hunter	Design	M1 M2 M3 M4 M5
93884	Olivia-Mae Barrett	Economics	D1 D2
93885	Zaydan Clarke	Maths AA SL	D1 D2
93886	Sameer Derrick	Psychology	D1 D2
93887	Kairon Lawson	Computer Science	D1 D2
93888	Youssef Massey	Geography	D1 D2
93889	Salman Povey	Biology	M1 M2 M3 M4 M5 D1 D2
93890	Abdulahi Roth	Physics	M4 M5 D1 D2
93891	Kaira Ellwood	History	M4 M5 D1 D2
93892	Lex Dickinson	Chemistry	M3 M4 M5 D1 D2
93893	Tessa Davey	Physical Education	M1 M2 M3 M4 M5 D1 D2
93894	Tiana Sharpe	English	M1 M2 M3 M4 M5
93895	Tai Finch	English	D1 D2
93896	Rhia Manning	French	M3 M4 M5
93897	Raya Knapp	French	D1 D2
93898	Cerys Bell	French	M1 M2 M3 M4 M5
93899	Tulisa Sharma	Sciences	M1 M2 M3 M4 M5
93900	Efan Welch	Arts	M1 M2 M3 M4 M5
93901	Rahul Cullen	Individuals & Societies	M3 M4 M5
93902	Manraj Rowe	Individuals & Societies	M1 M2 M3 M4 M5
NULL	NULL	NULL	NULL

student_details :

Result Grid Filter Rows:

student_id	student_name	student_grade
83728	Conall Salgado	M1
83729	Caprice Vazquez	M2
83730	Kaylan Cruz	M3
83731	Casey Brooks	M4
83732	Corrie Wallace	M5
83733	Ioan Hall	D1
83734	Makayla Kearns	D2
83735	Morgan Lim	M1
83736	Abi Webster	M2
83737	Andre Drew	M3
83738	Ayva Houghton	M4
83739	Keaton Fitzpatr…	M5
83740	Izaan Chase	D1
83741	Maya Williams	D2
83742	Connar Horn	M1
83743	Lily-Anne Person	M2
83744	Zakaria Key	M3
83745	Bryony Short	M4
83746	Geoffrey Galindo	M5
83747	Coral Rosario	D1
83748	Isla Mora	D2
83749	Denny Solis	M1
83750	Milena Albert	M2
83751	Matthew Molloy	M3
83752	Harvir Mcgill	D1
83753	Alessio Rubio	D2

teacher_login :

Result Grid Filter Rows:

user_id	username	password
93877	ammarchadwick	pY9DNmBN
93878	saffroncosta	Fw3VBUdC
93879	krystiandiaz	GaFzHDb4
93880	lilianmoyer	YUMbJkuU
93881	patsyhancock	AgjDQj2X
93882	iramrobles	UW2CMssz
93883	dollyhunter	QPZWnbJa
93884	olivia-maebarrett	fy4crsFv
93885	zaydanclarke	pRzrneLT
93886	sameerderrick	U36FyFjF
93887	kaironlawson	e6VPYP5p
93888	youssefmassey	t9V4AVCY
93889	salmanpovey	NvFAuchu
93890	abdulahiroth	xKU4UAhN
93891	kairaellwood	xzEX3EvX
93892	lexdickinson	Jh6CNfgb
93893	tessadavey	dMGaHxZ7
93894	tianasharpe	Dk3bgtXF
93895	taifinch	2HKcF9AW
93896	rhiamanning	MJZFBZNF
93897	rayaknapp	7jggUdWv
93898	cerysbell	FNgd59tT
93899	tulisasharma	WvXRgn8m
93900	efanwelch	rysM22J4
93901	rahulcullen	5RMpSxs2
93902	manrajrowe	M8gcf6GM
NULL	NULL	NULL

student_login :

Result Grid Filter Rows:

user_id	username	password
83728	conallsalgado	9EuWDjYN
83729	capricevazquez	WDBK8sbM
83730	kaylancruz	uSbKWMVv
83731	caseybrooks	AGYqUDFg
83732	corriewallace	pLZm7CTr
83733	ioanhall	Eg7kU6Na
83734	makaylakearns	WjLCk6Wj
83735	morganlim	5tgpJnQB
83736	abiwebster	BW9seycA
83737	andredrew	dALwyjBp
83738	ayvahoughton	6f4udyRb
83739	keatonfitzpatrick	Q5X8mkzt
83740	izaanchase	m8SbarEb
83741	mayawilliams	SrNXUEvY
83742	connarhorn	cL2xyfLK
83743	lily-anneperson	bReCD4Dk
83744	zakariakey	9fPfbkYV
83745	bryonyshort	bdwEPKCQ
83746	geoffreygalindo	3BHn7xYP
83747	coralrosario	ajUpP5tf
83748	islamora	d4zG8GQc
83749	dennysolis	UyN9RByW
83750	milenaalbert	Af7LG3Bj
83751	matthewmolloy	A9dDf97z
83752	harvirmcgill	UWxnJSBk
83753	alessiorubio	GBkjgcED
NULL	NULL	NULL

II The Servlet

The Java Servlet Application programming interface or API is used to handle the backend business logic of the product. The servlet is the 'Controller' in the Model-View-Controller design pattern. The Servlet API provides multiple classes such as the HTTPServlet, HTTPServletRequest and HTTPServletResponse classes which are essential for handling all the HTTP requests coming from the browser and JSP pages as well as sending responses back to them. The class 'MPHScheduleServlet' in the com.mph.scheduler.web package, is used as the servlet class which extends the HTTPServlet class and inherits its functions ((**4.5) Inheritance,** and (**4.4) Modularity**) which allows handling all the business logic such as redirecting to appropriate pages, checking login information, or adding and editing lessons, amongst other essential features.

The Servlet API also has another extremely useful class, Cookie, which is used to store information about whether a user is logged in or not (Success Criteria **A**). All class-wide variables are declared with the **private** keywork so as to deny access from any other classes except the object used to store the user's login details which is declared with a **protected** keyword as it is not threadsafe[4] to do so, in the case that in the future, multiple servlets are implemented in a 'multi-threaded' structure.

III The Data Classes

To allow easy access and transfer of data received and sent to the database, Separate 'data' classes were created, in the com.mph.scheduler.data package, which stored the details of different tables in the database. These class objects were used to construct new data or read data retrieved from the database ((**4.1) Abstraction, (4.2) Encapsulation**).

[4] BalusC. "doGet and doPost in Servlets" #2349741. *Stack Overflow*, 28 Feb. 2010, https://stackoverflow.com/a/2349741. Accessed 20 Oct. 2020.

Commented Screenshots

Data Classes:

Lessons.java:

Lessons.java

```java
package com.mph.scheduler.data;

/*
 *  The Lessons class is used to store information about the Lessons. Such as the index/primary key of
 *  the lesson, name of the teacher, the class being taken, the grade being taught, the date of the
 *  lesson, and the time of the lesson. The Lessons class allows individual access to each column's
 *  data and Lessons objects can be created to change or read this data.
 */
public class Lessons {

    private int index;// Stores the primary key of the lesson as an integer value.
    private String t_name;// Stores name of the teacher.
    private String t_class;// Stores class taken during lesson.
    private String grade;// Stores grade that is being taught during the lesson.
    private String date;// Stores the date of the lesson.
    private String time;// Stores the time of the lesson.


    /* Creating a constructor which does not take the primary key. This constructor
     * is used when the primary key is not required. For example when a new lesson
     * has to be added where the primary key is auto-generated or auto-incremented.
     */
    public Lessons(String t_name, String t_class, String grade, String date, String time) {
        this.t_name = t_name;
        this.t_class = t_class;
        this.grade = grade;
        this.date = date;
        this.time = time;
    }

    /* Creating a constructor which does not take the primary key. This constructor
     * is used when the primary key is required. For example when a specific lesson
     * that already exists must be updated.
     */
    public Lessons(int index, String t_name, String t_class, String grade, String date, String time) {
        this.index = index;
        this.t_name = t_name;
        this.t_class = t_class;
        this.grade = grade;
        this.date = date;
        this.time = time;
    }
```

(4.2) Encapsulation

(4.3) Polymorphism. By implementing method overloading using class constructors

```
46      // Getter and Setter methods for accessing or changing individual variables
47
48      public int getIndex() {
49          return index;
50      }
51
52      public void setIndex(int index) {
53          this.index = index;
54      }
55
56      public String getT_name() {
57          return t_name;
58      }
59
60      public void setT_name(String t_name) {
61          this.t_name = t_name;
62      }
63
64      public String getT_class() {
65          return t_class;
66      }
67
68      public void setT_class(String t_class) {
69          this.t_class = t_class;
70      }
71
72      public String getGrade() {
73          return grade;
74      }
75
76      public void setGrade(String grade) {
77          this.grade = grade;
78      }
79
80      public String getDate() {
81          return date;
82      }
83
84      public void setDate(String date) {
85          this.date = date;
86      }
87
88      public String getTime() {
89          return time;
90      }
91
92      public void setTime(String time) {
93          this.time = time;
94      }
95
96  }
```

LoginDetails.java

LoginDetails.java

```
1  package com.mph.scheduler.data;
2
3  /*
4   *  The LoginDetails class is used to store the login information about the user.
5   *  such as the user id, username and password of the user.
6   *  An object of this class can be used to access these details or store them
7   *  as required.
8   */
9  public class LoginDetails {
10
11      private int user_id;// Stores the user id of the user
12      private String username;// Stores the username of the user.
13      private String password;// Stores the password of the user.
14
15      // Getter and Setter methods for accessing or changing individual variables
16
17      public String getUsername() {
18          return username;
19      }
20      public void setUsername(String username) {
21          this.username = username;
22      }
23      public String getPassword() {
24          return password;
25      }
26      public void setPassword(String password) {
27          this.password = password;
28      }
29      public int getUser_id() {
30          return user_id;
31      }
32      public void setUser_id(int user_id) {
33          this.user_id = user_id;
34      }
35
36  }
```

(4.2) Encapsulation

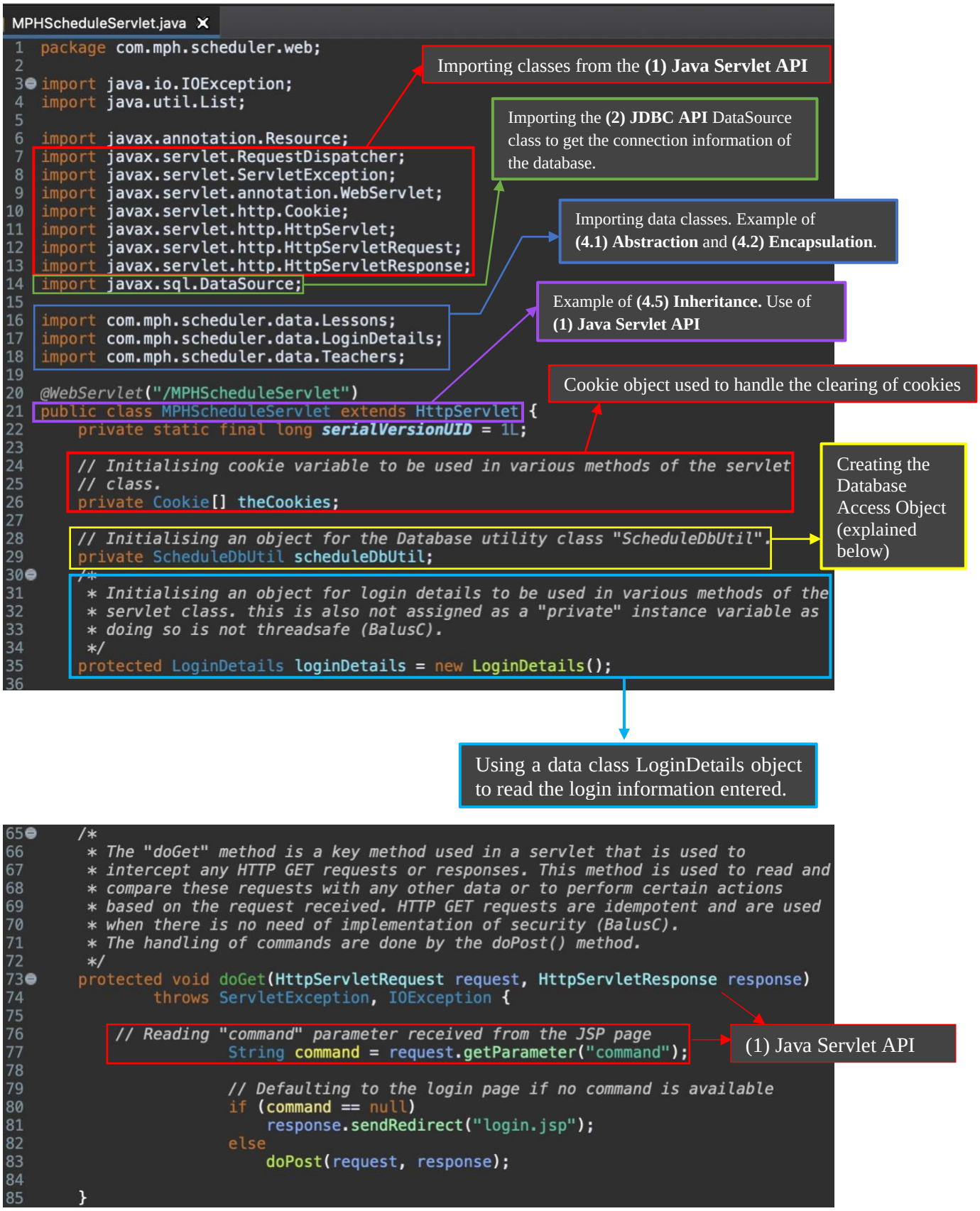
Teachers.java

(4.2) Encapsulation

```java
package com.mph.scheduler.data;


/*
 * The Teachers class is used to store information about teachers. Such as the user id of the teacher
 * teacher name, the class the teacher teaches and the grades the teacher teaches.
 */
public class Teachers {

    private int teacher_id; // Stores the user id of the teacher as an integer value.
    private String teacher_name;// Stores name of the teacher.
    private String teacher_lessons;// Stores the subject the teacher takes.
    private String teacher_grades;// Stores the grades the teacher teaches a String.
    private String[] taught_grades;// Used to store the grades taught by the teacher in a String array.

    //This constructor is used to generate values for every variable other than the grades array.
    public Teachers(int teacher_id, String teacher_name, String teacher_lessons, String teacher_grades) {
        this.teacher_id = teacher_id;
        this.teacher_name = teacher_name;
        this.teacher_lessons = teacher_lessons;
        this.teacher_grades = teacher_grades;
    }

    /* This method is used to take the String value of the grades taught by the teacher and store each
       grade in an array so they can be used as options in a drop-down list. */
    public String[] toGradeArray() {

        /* Splitting the grades String by using the delimiter as a whitespace and then storing the
           split words into the taught_grades array. */
        taught_grades = teacher_grades.split(" ");
        //Returning the array with the grades.
        return taught_grades;
    }

     //Getter and Setter methods for accessing or changing individual variables

     public int getTeacher_id() {
         return teacher_id;
     }

     public void setTeacher_id(int teacher_id) {
         this.teacher_id = teacher_id;
     }

     public String getTeacher_name() {
         return teacher_name;
     }

     public void setTeacher_name(String teacher_name) {
         this.teacher_name = teacher_name;
     }

     public String getTeacher_lessons() {
         return teacher_lessons;
     }

     public void setTeacher_lessons(String teacher_lessons) {
         this.teacher_lessons = teacher_lessons;
     }

     public String getTeacher_grades() {
         return teacher_grades;
     }

     public void setTeacher_grades(String teacher_grades) {
         this.teacher_grades = teacher_grades;
     }


}
```

The Servlet:

MPHScheduleServlet.java




```java
package com.mph.scheduler.web;

import java.io.IOException;
import java.util.List;

import javax.annotation.Resource;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

import com.mph.scheduler.data.Lessons;
import com.mph.scheduler.data.LoginDetails;
import com.mph.scheduler.data.Teachers;

@WebServlet("/MPHScheduleServlet")
public class MPHScheduleServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    // Initialising cookie variable to be used in various methods of the servlet
    // class.
    private Cookie[] theCookies;

    // Initialising an object for the Database utility class "ScheduleDbUtil".
    private ScheduleDbUtil scheduleDbUtil;
    /*
     * Initialising an object for login details to be used in various methods of the
     * servlet class. this is also not assigned as a "private" instance variable as
     * doing so is not threadsafe (BalusC).
     */
    protected LoginDetails loginDetails = new LoginDetails();
```

```java
    /*
     * The "doGet" method is a key method used in a servlet that is used to
     * intercept any HTTP GET requests or responses. This method is used to read and
     * compare these requests with any other data or to perform certain actions
     * based on the request received. HTTP GET requests are idempotent and are used
     * when there is no need of implementation of security (BalusC).
     * The handling of commands are done by the doPost() method.
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        // Reading "command" parameter received from the JSP page
                String command = request.getParameter("command");

                // Defaulting to the login page if no command is available
                if (command == null)
                    response.sendRedirect("login.jsp");
                else
                    doPost(request, response);

    }
```

```java
/*
 * The "doPost" method is a key method used in a servlet that is used to
 * intercept any HTTP POST requests or responses. This method is used when there
 * is a security risk involved, as HTML POST requests are not idempotent and are
 * used to implement security (BalusC)[5]. The HTML POST requests can only be read
 * and will not change even if the servlet explicitly tells it to, this is to
 * retain any original requests to maintain security.
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    try {

        // Reading "command" parameter received from the JSP page
        String command = request.getParameter("command");

        // Defaulting to the login page if no command is available
        if (command == null)
            command = "LOGIN";

        // Checking the command parameter and performing the according processes
        switch (command) {

        case "LOGIN":
            // Redirects the user to the login page
            response.sendRedirect("login.jsp");
            break;

        case "LOGGEDIN":

            // Reading the user input login details received from the JSP Login page
            String username = request.getParameter("username");
            String password = request.getParameter("password");
            int user_id = Integer.parseInt(request.getParameter("userId"));

            // Setting the variables from the loginDetails class to the respective data
            // received from the JSP Login page
            loginDetails.setUsername(username);
            loginDetails.setPassword(password);
            loginDetails.setUser_id(user_id);

            try {
                if (scheduleDbUtil.validateTeacherLogin(loginDetails) || command.equals("LIST_TEACH")) {
                    /*
                     * This conditional statement is used to redirect the user to the default
                     * schedule view if the entered teacher login details are correct or if a
                     * Command from the login page is sent to redirect the user to the teacher
                     * schedule view. This command is sent only if a cookie with a corresponding
                     * value that tells signifies a previously logged in teacher that has not logged
                     * out.
                     */
                    listTeacherLessons(request, response);
                } else {
                    if (scheduleDbUtil.validateStudentLogin(loginDetails) || command.equals("LIST")) {
                        /*
                         * This conditional statement is used to redirect the user to the default
                         * schedule view if the entered teacher login details are incorrect and the
                         * entered student login details are correct or if a Command from the login page
                         * is sent to redirect the user. This command is sent only if a cookie with a
                         * corresponding value that tells signifies a previously logged in student that
                         * has not logged out.
                         */
                        listLessons(request, response);
                    } else {
                        /*
                         * If both student and teacher login details are incorrect, and if there is no
                         * cookie value that indicates a previously logged in user, the user is
                         * redirected back to the login page with a parameter that sends an error
                         * message of "IncorretUserOrPass".
                         */
                        request.setAttribute("IncorrectUserOrPass", "true");
                        RequestDispatcher default_dispatcher = request.getRequestDispatcher("/login.jsp");
                        default_dispatcher.forward(request, response);
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
```

(1) Java Servlet API

(7) Conditional programming using a switch statement that performs the necessary functions based on the request from the JSP page, ex. logging in

Use of Database Utility elaborated below

Using **(7) Conditional Programming** to validate logins (Success Criteria A, D, G)

(6) Try/Catch Blocks to handle exceptions

[5] BalusC. "doGet and doPost in Servlets" #2349741. *Stack Overflow*, 28 Feb. 2010, https://stackoverflow.com/a/2349741. Accessed 20 Oct. 2020.


```
166             case "LOGOUT":
167                 // Clears cookies using the clearCookie method
168                 clearCookie(request, response);
169                 break;
170
171             case "LIST_TEACH":
172                 /*
173                  * Redirects the user to the teacher schedule view and lists the appropriate
174                  * data for teachers
175                  */
176                 listTeacherLessons(request, response);
177                 break;

                case "LIST":
                    /*
                     * Redirects the user to the default schedule view and lists the appropriate
                     * data for students
                     */
                    listLessons(request, response);
                    break;

                case "LOAD_ADD":
                    loadAddLesson(request, response);
                    break;

                case "ADD":
                    addLesson(request, response);
                    break;

                case "LOAD":
                    /*
                     * Calls the method to pre-populate the JSP pages forms with any previously
                     * stored data from the database
                     */
                    editLesson(request, response);
                    break;

                case "UPDATE":
                    // Calls the method to change the values of the lesson in the database
                    updateLesson(request, response);
                    break;

                case "DELETE":
                    /*
                     * Calls the method that 'deletes' the Lesson from the view, setting all values
                     * other than the index to null
                     */
                    clearLesson(request, response);
                    break;

                default:
                    /*
                     * If no command is present or is different than any listed commands, the user
                     * is redirected to the login page
                     */
                    response.sendRedirect("login.jsp");
                }

            } catch (Exception e) {
                throw new ServletException(e);
            }
```

IV The Database Access Object (DAO)

The database access object or DAO is used to retrieve and make changes to the database using the (**2) Java Database Connectivity API**. There are multiple methods created in this class, ScheduleDbUtil in the com.mph.scheduler.web package, that perform the necessary instructions to retrieve, send or edit data in the database. This is an implementation of (**4.1) Abstraction**, (**4.2) Encapsulation** and (**4.4) Modularity**.

These methods are called in the servlet class to perform actions like viewing, adding or deleting lessons from the schedule (database). The following method in the Servlet gets the connection information from the context.xml file that stores the connection details and passes the details to the DAO usnig the constructor.

Commented Screenshots

context.xml

```
context.xml
1  <Context>
2
3      <Resource name="mph_scheduling"
4                auth="Container" type="javax.sql.DataSource"
5                maxActive="20" maxIdle="5" maxWait="10000"
6                username="root" password="wqjvk5um5kuq36e9he"
7                driverClassName="com.mysql.cj.jdbc.Driver"
8                url="jdbc:mysql://localhost:3306/mph_scheduling?useSSL=false&amp;serverTimezone=UTC"/>
9  </Context>
```

MPHScheduleServlet.java

```
37      /*
38       * Getting the resource of the database details from the content.xml file and
39       * using the DataSource object to establish a connection pool to the database
40       * "mph_scheduling".
41       */
42      @Resource(name = "mph_scheduling")
43      private DataSource dataSource;
```

Connecting to the connection resource from the .xml file

```
45      // The "init" method is used to initialise any dependencies for the servlet.
46      @Override
47      /*
48       * This method is set to override it's superclass as it is a 'convenience'
49       * method[6]("GenericServlet (Java)").
50       */
51      public void init() throws ServletException {
52
53          try {
54              /*
55               * Creating a database utility object and passing the connection pool variable
56               * to give the database utility the connection details.
57               */
58              scheduleDbUtil = new ScheduleDbUtil(dataSource);
59          } catch (Exception e) {
60              throw new ServletException(e);
61          }
62
63      }
```

(4.3) Polymorphism
Using method overriding

[6] "GenericServlet (Java EE 6)." *Oracle Java Documentation*, 10 Feb. 2011, docs.oracle.com/javaee/6/api/javax/servlet/GenericServlet.html. Accessed 30 Nov. 2020.

Database Utility: ScheduleDbUtil.java

ScheduleDbUtil.java

```
1  package com.mph.scheduler.web;
2
3  import java.sql.Connection;
4
5  import java.sql.PreparedStatement;
6  import java.sql.ResultSet;
7  import java.sql.SQLException;
8  import java.sql.Statement;
9  import java.util.ArrayList;
10 import java.util.List;
11
12 import javax.sql.DataSource;
13
14 import com.mph.scheduler.data.Lessons;
15 import com.mph.scheduler.data.LoginDetails;
16 import com.mph.scheduler.data.Teachers;
17
18 public class ScheduleDbUtil {
19
20     private DataSource dataSource;
21
22     /* Constructor Method to set the class data source value to the data source
23        received from the MPHScheduleServlet. */
24     public ScheduleDbUtil(DataSource theDataSource) {
25         /* This data source consists of the connection information for the database
26            stored in the context.xml file. */
27         dataSource = theDataSource;
28     }
```

Importing classes from the **(2) JDBC API.**

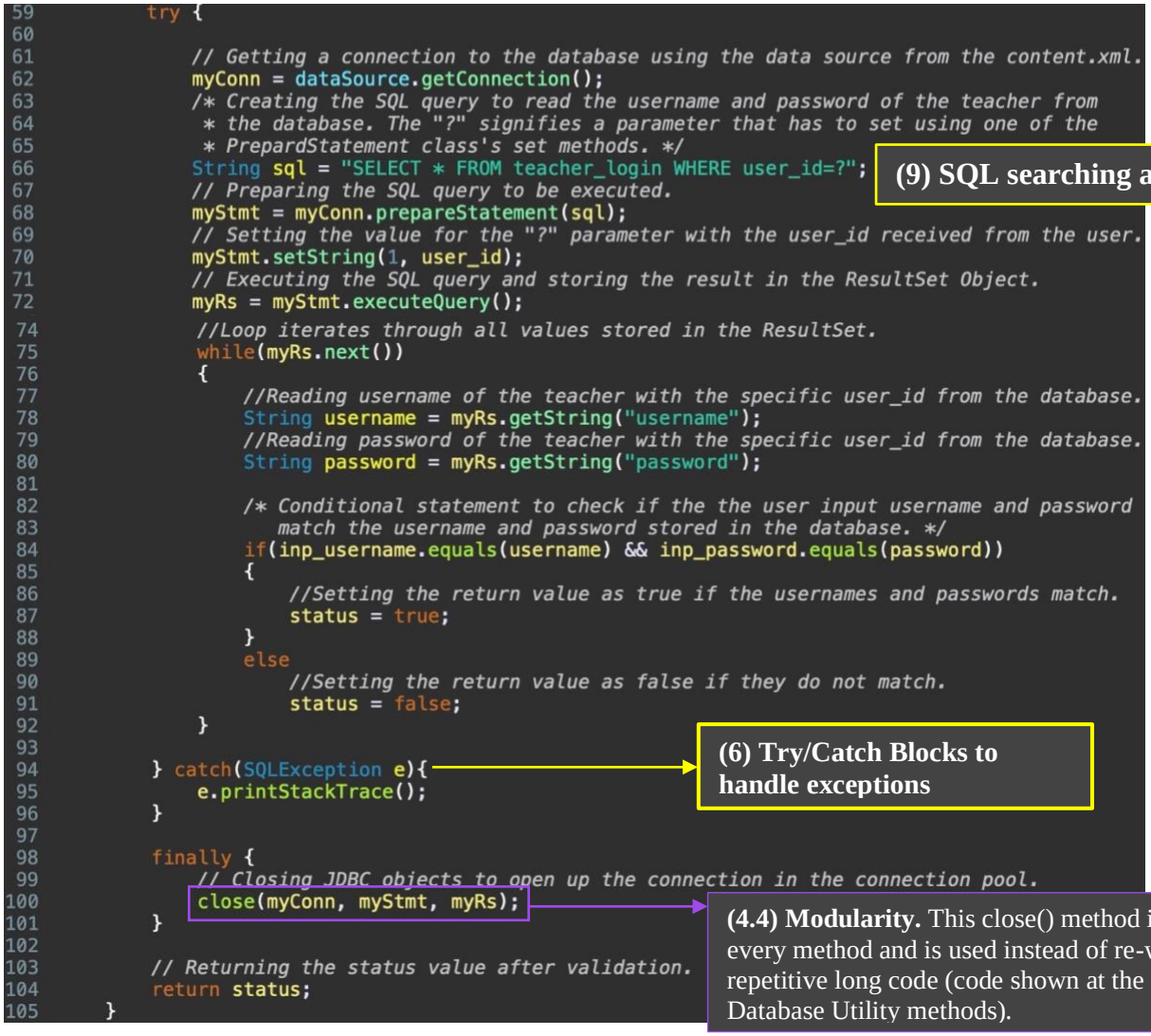
Importing the ArrayList and List classes to use **(5) Dynamic Arrays (ArrayLists).**

Importing data classes. Example of **(4.1) Abstraction** and **(4.2) Encapsulation**.

Methods in the Database Utility

```
30     /* This method "validateTeacherLogin" is used to validate the user's login information
31      * with the login information of the teachers that is stored in the database. It takes a
32      * LoginDetails object as an argument and returns a true or false boolean value. A true
33      * value indicates successful validation and a false value indicates incorrect login
34      * information.
35      */
36     public boolean validateTeacherLogin(LoginDetails loginDetails) throws Exception{
37
38         // Variable to be returned as a boolean true or false value.
39         boolean status = false;
40
41         /* Getting the login details input from the user from the LoginDetails object passed
42            from the MPHScheduleServlet. */
43
44         // Converting the user id to a string for easier processing.
45         String user_id = String.valueOf(loginDetails.getUser_id());
46         String inp_username = loginDetails.getUsername();
47         String inp_password = loginDetails.getPassword();
48
49         // Initialising JDBC SQL package objects.
50
51         // JDBC SQL Connection object to establish a connection to the database.
52         Connection myConn = null;
53         /* JDBC SQL PreparedStatement object to execute an SQL statement with varying
54            parameters to be passed.*/
55         PreparedStatement myStmt = null;
56         // JDBC SQL ResultSet object to get the result of the performed SQL query.
57         ResultSet myRs = null;
```

Success Criteria A, G



```
59          try {
60
61              // Getting a connection to the database using the data source from the content.xml.
62              myConn = dataSource.getConnection();
63              /* Creating the SQL query to read the username and password of the teacher from
64               * the database. The "?" signifies a parameter that has to set using one of the
65               * PrepardStatement class's set methods. */
66              String sql = "SELECT * FROM teacher_login WHERE user_id=?";
67              // Preparing the SQL query to be executed.
68              myStmt = myConn.prepareStatement(sql);
69              // Setting the value for the "?" parameter with the user_id received from the user.
70              myStmt.setString(1, user_id);
71              // Executing the SQL query and storing the result in the ResultSet Object.
72              myRs = myStmt.executeQuery();
74              //Loop iterates through all values stored in the ResultSet.
75              while(myRs.next())
76              {
77                  //Reading username of the teacher with the specific user_id from the database.
78                  String username = myRs.getString("username");
79                  //Reading password of the teacher with the specific user_id from the database.
80                  String password = myRs.getString("password");
81
82                  /* Conditional statement to check if the the user input username and password
83                     match the username and password stored in the database. */
84                  if(inp_username.equals(username) && inp_password.equals(password))
85                  {
86                      //Setting the return value as true if the usernames and passwords match.
87                      status = true;
88                  }
89                  else
90                      //Setting the return value as false if they do not match.
91                      status = false;
92              }
93
94          } catch(SQLException e){
95              e.printStackTrace();
96          }
97
98          finally {
99              // Closing JDBC objects to open up the connection in the connection pool.
100             close(myConn, myStmt, myRs);
101         }
102
103         // Returning the status value after validation.
104         return status;
105     }
```

(9) SQL searching and filtering

(6) Try/Catch Blocks to handle exceptions

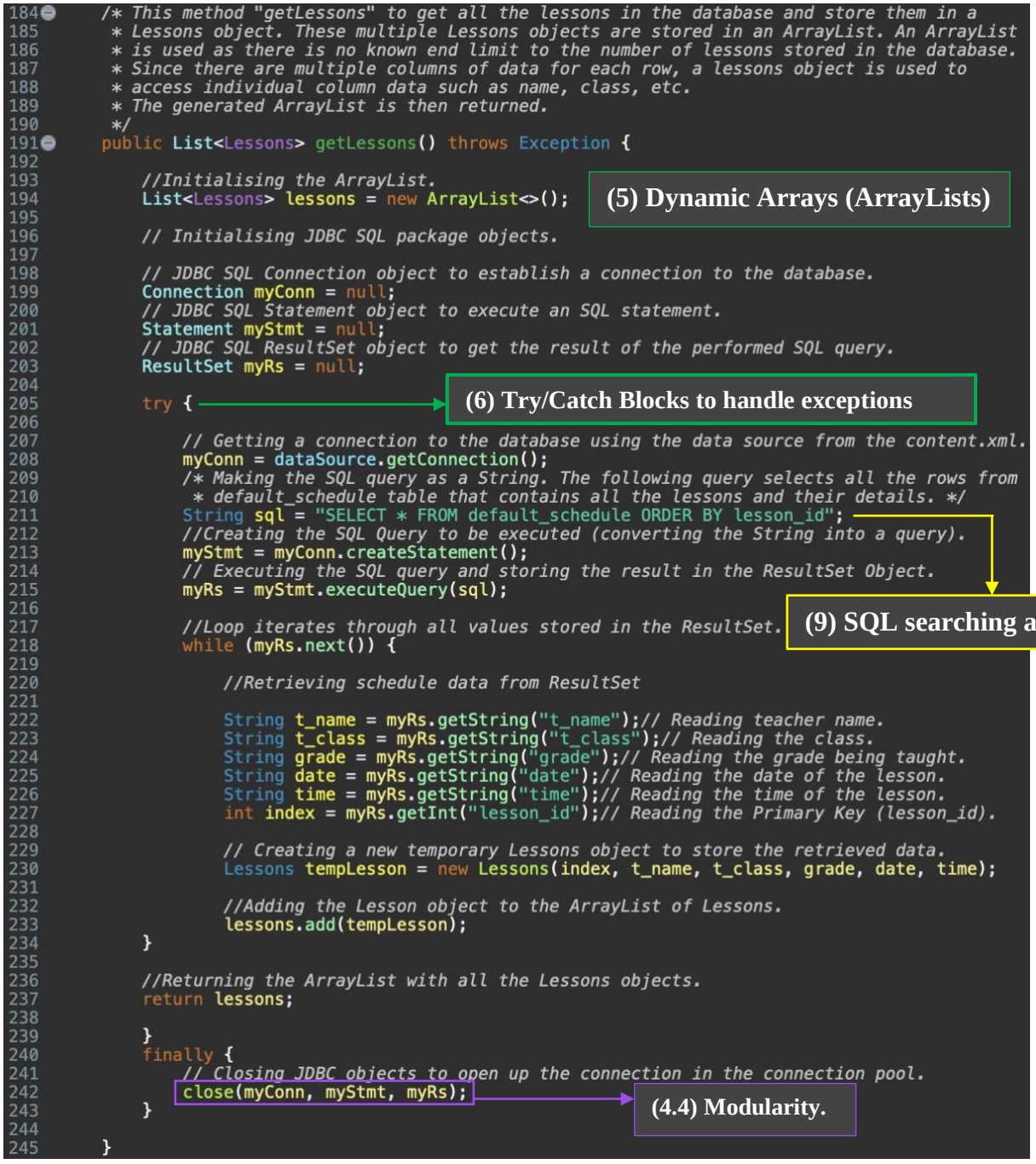
(4.4) Modularity. This close() method is used in every method and is used instead of re-writing repetitive long code (code shown at the end of Database Utility methods).

```java
/* This method "validateStudentLogin" is used to validate the user's login information
 * with the login information of the students that is stored in the database. It takes a
 * LoginDetails object as an argument and returns a true or false boolean value. A true
 * value indicates successful validation and a false value indicates incorrect login
 * information.
 */
public boolean validateStudentLogin(LoginDetails loginDetails) throws Exception{

    // Variable to be returned as a boolean true or false value.
    boolean status = false;

    /* Getting the login details input from the user from the LoginDetails object passed
       from the MPHScheduleServlet. */

    // Converting the user id to a string for easier processing.
    String user_id = String.valueOf(loginDetails.getUser_id());
    String inp_username = loginDetails.getUsername();
    String inp_password = loginDetails.getPassword();

    // Initialising JDBC SQL package objects.

    // JDBC SQL Connection object to establish a connection to the database.
    Connection myConn = null;
    /* JDBC SQL PreparedStatment object to execute an SQL statement with varying
       parameters to be passed.*/
    PreparedStatement myStmt = null;
    // JDBC SQL ResultSet object to get the result of the performed SQL query.
    ResultSet myRs = null;

    try {

        // Getting a connection to the database using the data source from the content.xml.
        myConn = dataSource.getConnection();
        /* Creating the SQL query to read the username and password of the teacher from
         * the database. The "?" signifies a parameter that has to set using one of the
         * PrepardStatement class's set methods. */
        String sql = "SELECT * FROM student_login WHERE user_id=?";
        // Preparing the SQL query to be executed.
        myStmt = myConn.prepareStatement(sql);
        // Setting the value for the "?" parameter with the user_id received from the user.
        myStmt.setString(1, user_id);
        // Executing the SQL query and storing the result in the ResultSet Object.
        myRs = myStmt.executeQuery();

        //Loop iterates through all values stored in the ResultSet.
        while(myRs.next())
        {
            //Reading username of the teacher with the specific user_id from the database.
            String username = myRs.getString("username");
            //Reading password of the teacher with the specific user_id from the database.
            String password = myRs.getString("password");
            /* Conditional statement to check if the the user input username and password
               match the username and password stored in the database. */
            if(inp_username.equals(username) && inp_password.equals(password))
            {
                //Setting the return value as true if the usernames and passwords match.
                status = true;
            }
            else
                //Setting the return value as false if they do not match.
                status = false;
        }

    } catch(SQLException e){
        e.printStackTrace();
    }

    finally {
        // Closing JDBC objects to open up the connection in the connection pool.
        close(myConn, myStmt, myRs);
    }

    // Returning the status value after validation.
    return status;
}
```

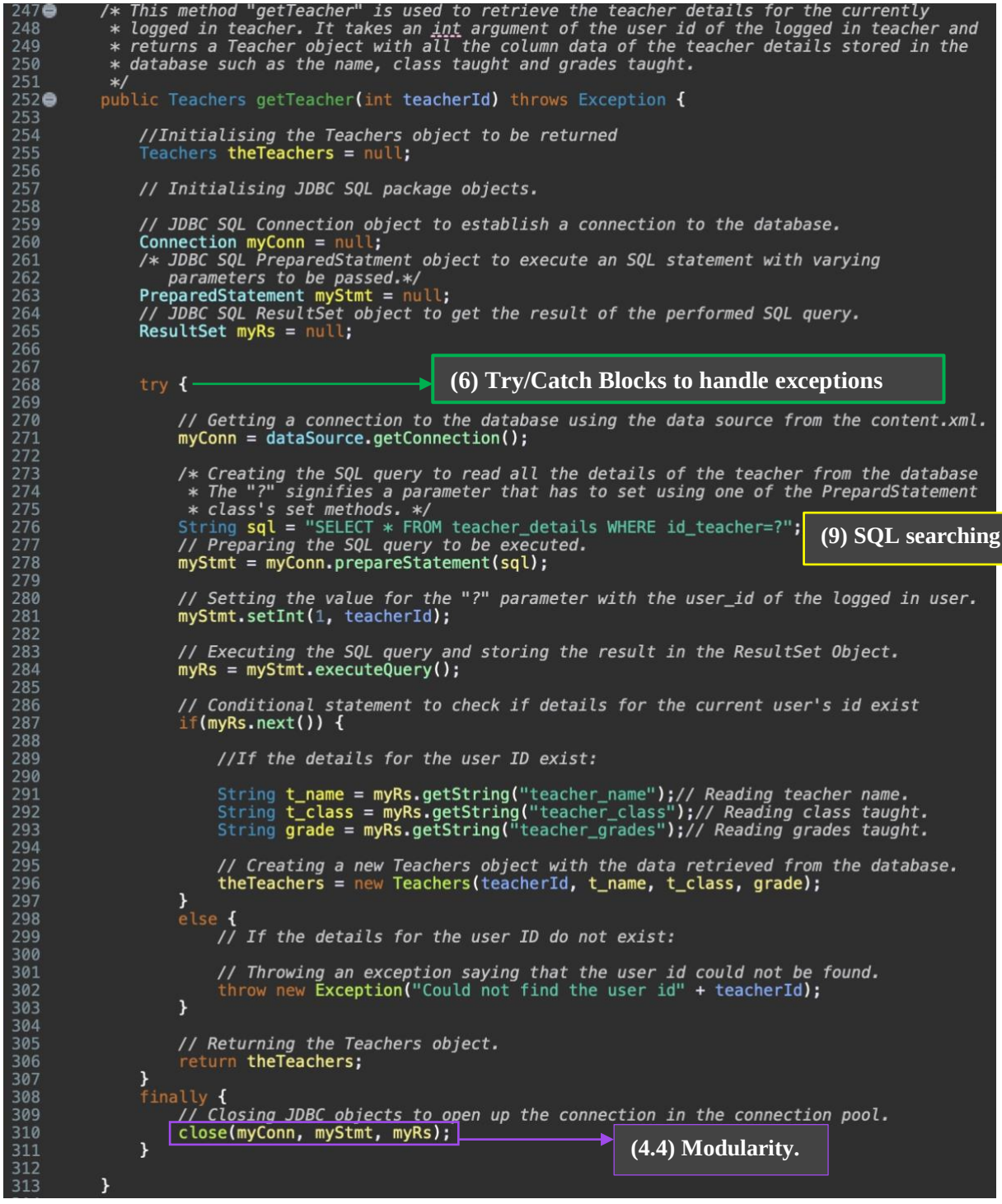
(9) SQL searching and filtering

(6) Try/Catch Blocks to handle exceptions

(4.4) Modularity.




```java
/* This method "getLessons" to get all the lessons in the database and store them in a
 * Lessons object. These multiple Lessons objects are stored in an ArrayList. An ArrayList
 * is used as there is no known end limit to the number of lessons stored in the database.
 * Since there are multiple columns of data for each row, a lessons object is used to
 * access individual column data such as name, class, etc.
 * The generated ArrayList is then returned.
 */
public List<Lessons> getLessons() throws Exception {

    //Initialising the ArrayList.
    List<Lessons> lessons = new ArrayList<>();

    // Initialising JDBC SQL package objects.

    // JDBC SQL Connection object to establish a connection to the database.
    Connection myConn = null;
    // JDBC SQL Statement object to execute an SQL statement.
    Statement myStmt = null;
    // JDBC SQL ResultSet object to get the result of the performed SQL query.
    ResultSet myRs = null;

    try {

        // Getting a connection to the database using the data source from the content.xml.
        myConn = dataSource.getConnection();
        /* Making the SQL query as a String. The following query selects all the rows from
         * default_schedule table that contains all the lessons and their details. */
        String sql = "SELECT * FROM default_schedule ORDER BY lesson_id";
        //Creating the SQL Query to be executed (converting the String into a query).
        myStmt = myConn.createStatement();
        // Executing the SQL query and storing the result in the ResultSet Object.
        myRs = myStmt.executeQuery(sql);

        //Loop iterates through all values stored in the ResultSet.
        while (myRs.next()) {

            //Retrieving schedule data from ResultSet

            String t_name = myRs.getString("t_name");// Reading teacher name.
            String t_class = myRs.getString("t_class");// Reading the class.
            String grade = myRs.getString("grade");// Reading the grade being taught.
            String date = myRs.getString("date");// Reading the date of the lesson.
            String time = myRs.getString("time");// Reading the time of the lesson.
            int index = myRs.getInt("lesson_id");// Reading the Primary Key (lesson_id).

            // Creating a new temporary Lessons object to store the retrieved data.
            Lessons tempLesson = new Lessons(index, t_name, t_class, grade, date, time);

            //Adding the Lesson object to the ArrayList of Lessons.
            lessons.add(tempLesson);
    }

    //Returning the ArrayList with all the Lessons objects.
    return lessons;

    }
    finally {
        // Closing JDBC objects to open up the connection in the connection pool.
        close(myConn, myStmt, myRs);
    }

}
```




```java
/* This method "getTeacher" is used to retrieve the teacher details for the currently
 * logged in teacher. It takes an int argument of the user id of the logged in teacher and
 * returns a Teacher object with all the column data of the teacher details stored in the
 * database such as the name, class taught and grades taught.
 */
public Teachers getTeacher(int teacherId) throws Exception {

    //Initialising the Teachers object to be returned
    Teachers theTeachers = null;

    // Initialising JDBC SQL package objects.

    // JDBC SQL Connection object to establish a connection to the database.
    Connection myConn = null;
    /* JDBC SQL PreparedStatment object to execute an SQL statement with varying
       parameters to be passed.*/
    PreparedStatement myStmt = null;
    // JDBC SQL ResultSet object to get the result of the performed SQL query.
    ResultSet myRs = null;


    try {

        // Getting a connection to the database using the data source from the content.xml.
        myConn = dataSource.getConnection();

        /* Creating the SQL query to read all the details of the teacher from the database
         * The "?" signifies a parameter that has to set using one of the PrepardStatement
         * class's set methods. */
        String sql = "SELECT * FROM teacher_details WHERE id_teacher=?";
        // Preparing the SQL query to be executed.
        myStmt = myConn.prepareStatement(sql);

        // Setting the value for the "?" parameter with the user_id of the logged in user.
        myStmt.setInt(1, teacherId);

        // Executing the SQL query and storing the result in the ResultSet Object.
        myRs = myStmt.executeQuery();

        // Conditional statement to check if details for the current user's id exist
        if(myRs.next()) {

            //If the details for the user ID exist:

            String t_name = myRs.getString("teacher_name");// Reading teacher name.
            String t_class = myRs.getString("teacher_class");// Reading class taught.
            String grade = myRs.getString("teacher_grades");// Reading grades taught.

            // Creating a new Teachers object with the data retrieved from the database.
            theTeachers = new Teachers(teacherId, t_name, t_class, grade);
        }
        else {
            // If the details for the user ID do not exist:

            // Throwing an exception saying that the user id could not be found.
            throw new Exception("Could not find the user id" + teacherId);
        }

        // Returning the Teachers object.
        return theTeachers;
    }
    finally {
        // Closing JDBC objects to open up the connection in the connection pool.
        close(myConn, myStmt, myRs);
    }

}
```

```java
    /* This method "getLesson" is used to read the information of the particular lesson the
     * user has accessed through the edit button on the TeacherScheduleView JSP page and
     * takes a String argument of the lesson id chosen. It returns a Lessons object with the
     * details of the lesson. This method is used to pre-populate the fields of the
     * update-class-form JSP page using the returned Lessons object.
     */
    public Lessons getLesson(String theLessonId) throws Exception{

        // Initialising the Lessons object to be returned.
        Lessons theLesson = null;
        // Initialising an int variable that will be used to store the id of the lesson
        int lessonId;

        // Initialising JDBC SQL package objects.

        // JDBC SQL Connection object to establish a connection to the database.
        Connection myConn = null;
        /* JDBC SQL PreparedStatment object to execute an SQL statement with varying
           parameters to be passed.*/
        PreparedStatement myStmt = null;
        // JDBC SQL ResultSet object to get the result of the performed SQL query.
        ResultSet myRs = null;

        try {

            /* Converting the String value of the lesson id to an int (as the Lessons object
             * Constructor takes the user id as in integer value. */
            lessonId = Integer.parseInt(theLessonId);

            // Getting a connection to the database using the data source from the content.xml.
            myConn = dataSource.getConnection();

            /* Making the SQL query as a String. The following query selects the row from the
             * default_schedule table that contains details of the specified lesson_id. */
            String sql = "SELECT * FROM default_schedule WHERE lesson_id=?";
            // Preparing the SQL query to be executed.
            myStmt = myConn.prepareStatement(sql);
            // Setting the value for the "?" parameter with the user_id of the logged in user.
            myStmt.setInt(1, lessonId);
            // Executing the SQL query and storing the result in the ResultSet Object.
            myRs = myStmt.executeQuery();

            // Conditional statement to check if details for the current user's id exist
            if(myRs.next()) {

                // If the details for the user ID exist:

                String t_name = myRs.getString("t_name");// Reading teacher name.
                String t_class = myRs.getString("t_class");// Reading class taught.
                String grade = myRs.getString("grade");// Reading grade taught.
                String date = myRs.getString("date");// Reading the date of lesson.
                String time = myRs.getString("time");// Reading the time of lesson.

                // Constructing a Lessons object with the data retrieved from the database.
                theLesson = new Lessons(lessonId, t_name, t_class, grade, date, time);
            }
            else {
                // If the details for the user ID do not exist:

                // Throwing an exception saying that the user id could not be found.
                throw new Exception("Could not find the class id" + lessonId);
            }

            // Returning the Lessons object.
            return theLesson;
        }
        finally {

            // Closing JDBC objects to open up the connection in the connection pool.
            close(myConn, myStmt, myRs);
        }
    }
```

(6) Try/Catch Blocks to handle exceptions

(9) SQL searching and filtering

(4.4) Modularity.

```java
/* This method "addLesson" is used to add a new lesson and store it in the database in the
 * default_schedule table. It takes a Lessons object as an argument. The intended use of
 * this method is for a Lessons object that contains user input data to be passed as the
 * argument.
 */
public void addLesson(Lessons theLesson) throws Exception{

    // Initialising JDBC SQL package objects.

    // JDBC SQL Connection object to establish a connection to the database.
    Connection myConn = null;
    /* JDBC SQL PreparedStatment object to execute an SQL statement with varying
       parameters to be passed.*/
    PreparedStatement myStmt = null;

    try {

        // Getting a connection to the database using the data source from the content.xml.
        myConn = dataSource.getConnection();
        /* Creating the SQL query to insert the user input values into the default_schedule
         * table in the database. The "?" signifies a parameter that has to set using one
         * of the PrepardStatement class's set methods. */
        String sql = "INSERT INTO default_schedule "
                   + "(t_name, t_class, grade, date, time) "
                   + "values (?, ?, ?, ?, ?)";
        // Preparing the SQL query to be executed.
        myStmt = myConn.prepareStatement(sql);

        // Setting the values for the "?" parameters with the data input by the user.
        myStmt.setString(1, theLesson.getT_name());
        myStmt.setString(2, theLesson.getT_class());
        myStmt.setString(3, theLesson.getGrade());
        myStmt.setString(4, theLesson.getDate());
        myStmt.setString(5, theLesson.getTime());

        // Executing the SQL statement
        myStmt.execute();
    }
    finally{
        // Closing JDBC objects to open up the connection in the connection pool.
        close(myConn, myStmt, null);
    }

}
```

Success Criteria B, H

(9) SQL inserting

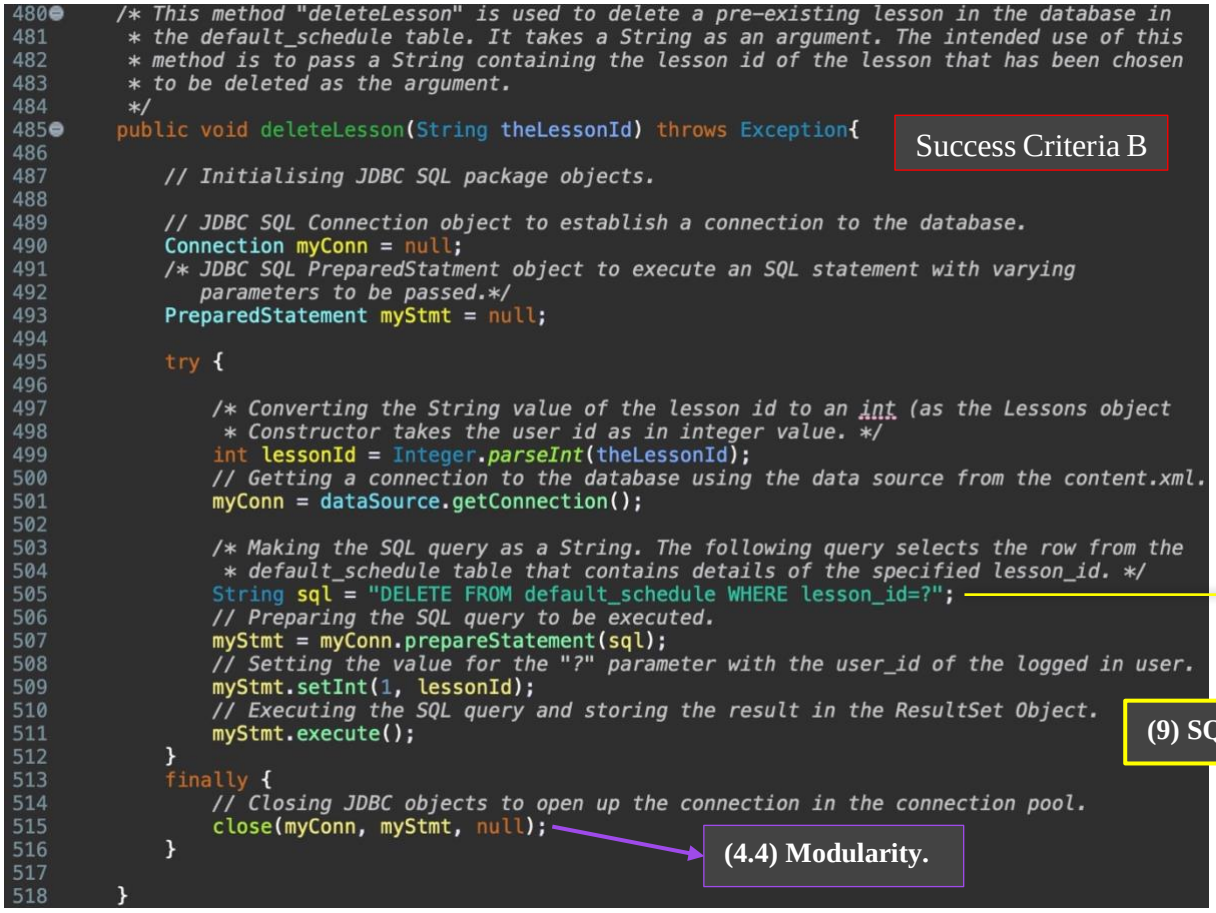
(4.4) Modularity.

```java
/* This method "updateLesson" is used to edit a pre-existing lesson in the database in the
 * default_schedule table. It takes a Lessons object as an argument. The intended use of
 * this method is for a Lessons object that contains user input data to be passed as the
 * argument.
 */
public void updateLesson(Lessons theLesson) throws Exception {

    // Initialising JDBC SQL package objects.

    // JDBC SQL Connection object to establish a connection to the database.
    Connection myConn = null;
    /* JDBC SQL PreparedStatment object to execute an SQL statement with varying
       parameters to be passed.*/
    PreparedStatement myStmt = null;

    try {

        // Getting a connection to the database using the data source from the content.xml.
        myConn = dataSource.getConnection();
        /* Creating the SQL query to insert the user input values into the default_schedule
         * table in the database for the specific lesson that the user has chosen to
         * update using the lesson_id. The "?" signifies a parameter that has to set using
         * one of the PrepardStatement class's set methods. */
        String sql = "UPDATE default_schedule "
                   + "SET t_name=?, t_class=?, grade=?, date=?, time=?"
                   + "WHERE lesson_id=?";
        // Preparing the SQL query to be executed.
        myStmt = myConn.prepareStatement(sql);

        // Setting the values for the "?" parameters with the data input by the user.
        myStmt.setString(1, theLesson.getT_name());
        myStmt.setString(2, theLesson.getT_class());
        myStmt.setString(3, theLesson.getGrade());
        myStmt.setString(4, theLesson.getDate());
        myStmt.setString(5, theLesson.getTime());
        myStmt.setInt(6, theLesson.getIndex());

        // Executing the SQL statement
        myStmt.execute();
    }
    finally{
        // Closing JDBC objects to open up the connection in the connection pool.
        close(myConn, myStmt, null);
    }
```

Success Criteria B

(9) SQL updating

(4.4) Modularity.




```java
/* This method "deleteLesson" is used to delete a pre-existing lesson in the database in
 * the default_schedule table. It takes a String as an argument. The intended use of this
 * method is to pass a String containing the lesson id of the lesson that has been chosen
 * to be deleted as the argument.
 */
public void deleteLesson(String theLessonId) throws Exception{

    // Initialising JDBC SQL package objects.

    // JDBC SQL Connection object to establish a connection to the database.
    Connection myConn = null;
    /* JDBC SQL PreparedStatment object to execute an SQL statement with varying
       parameters to be passed.*/
    PreparedStatement myStmt = null;

    try {

        /* Converting the String value of the lesson id to an int (as the Lessons object
         * Constructor takes the user id as in integer value. */
        int lessonId = Integer.parseInt(theLessonId);
        // Getting a connection to the database using the data source from the content.xml.
        myConn = dataSource.getConnection();

        /* Making the SQL query as a String. The following query selects the row from the
         * default_schedule table that contains details of the specified lesson_id. */
        String sql = "DELETE FROM default_schedule WHERE lesson_id=?";
        // Preparing the SQL query to be executed.
        myStmt = myConn.prepareStatement(sql);
        // Setting the value for the "?" parameter with the user_id of the logged in user.
        myStmt.setInt(1, lessonId);
        // Executing the SQL query and storing the result in the ResultSet Object.
        myStmt.execute();
    }
    finally {
        // Closing JDBC objects to open up the connection in the connection pool.
        close(myConn, myStmt, null);
    }

}
```

```java
/* This method "close" is used to close connections used by JDBC objects by opening up the
 * used connection to the connection pool so it can be used by other users. */
private void close(Connection myConn, Statement myStmt, ResultSet myRs) {

    /* This does not technically does not close any connections but instead makes the
     * connection available to other users for them to use. */
    try {
        if(myRs != null)
            myRs.close();
        if(myStmt != null)
            myStmt.close();
        if(myConn != null)
            myConn.close();
    }
    catch(Exception e) {
        e.printStackTrace();
    }

}
}
```

(4.4) Modularity. Allows use of one method to execute frequently used code through one statement and reduces writing longer redundant code.

V Use of the DAO and Servlet methods

The database access object is used in the Servlet in the following methods. These DAO methods are used to implement **(4.1) Abstraction** and **(4.2) Encapsulation**.

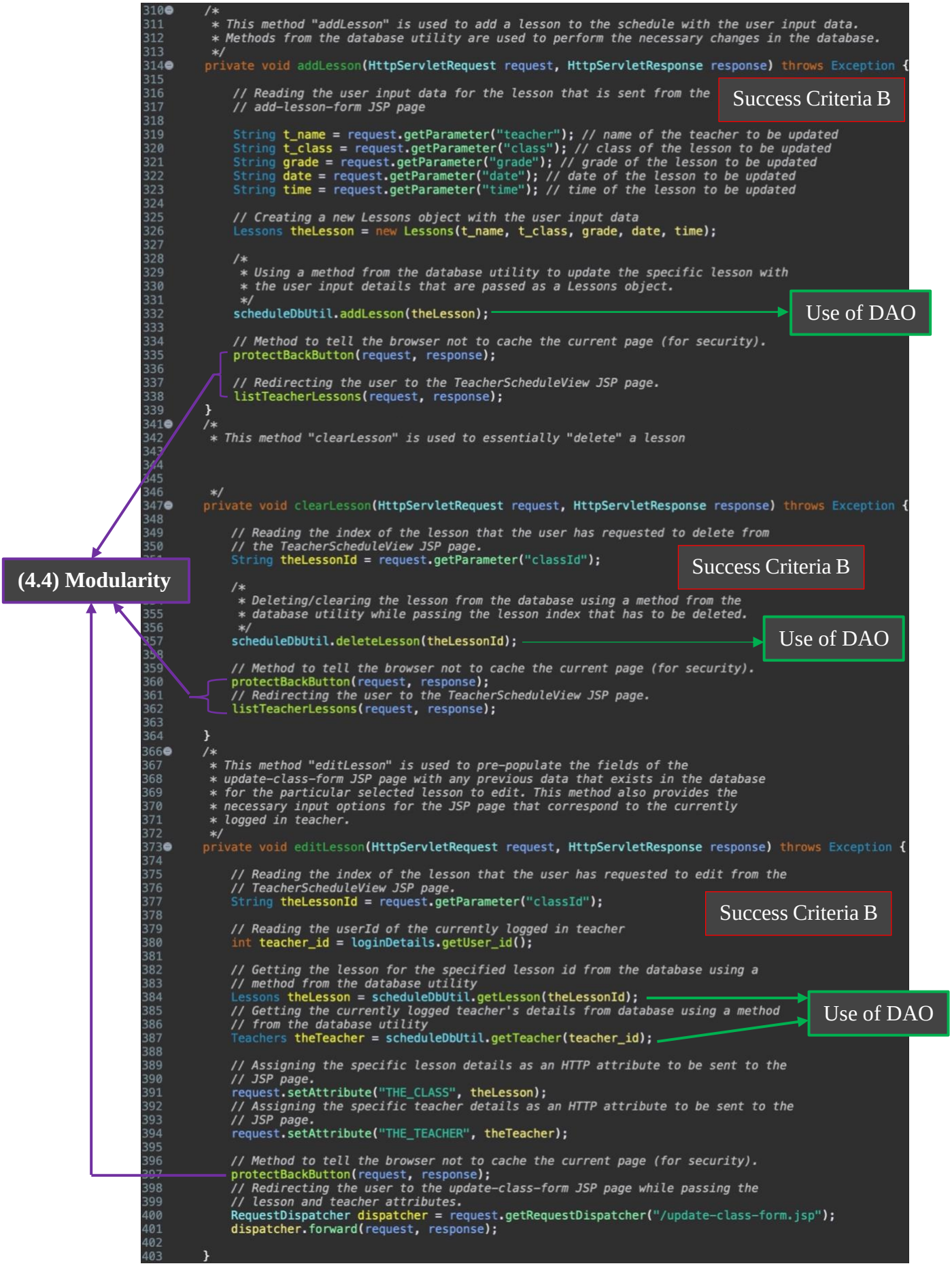
The functions in the servlet are used to implement **(4.5) Modularity** to reduce redundancy and to avoid repeating long lines of code throughout the class. These also improve the readability

of code especially in the doPost() method above in **(I) & (II)** , where the methods have clear name descriptions of what their purpose is, ex. listLessons() or addLesson(). These methods are shown below.

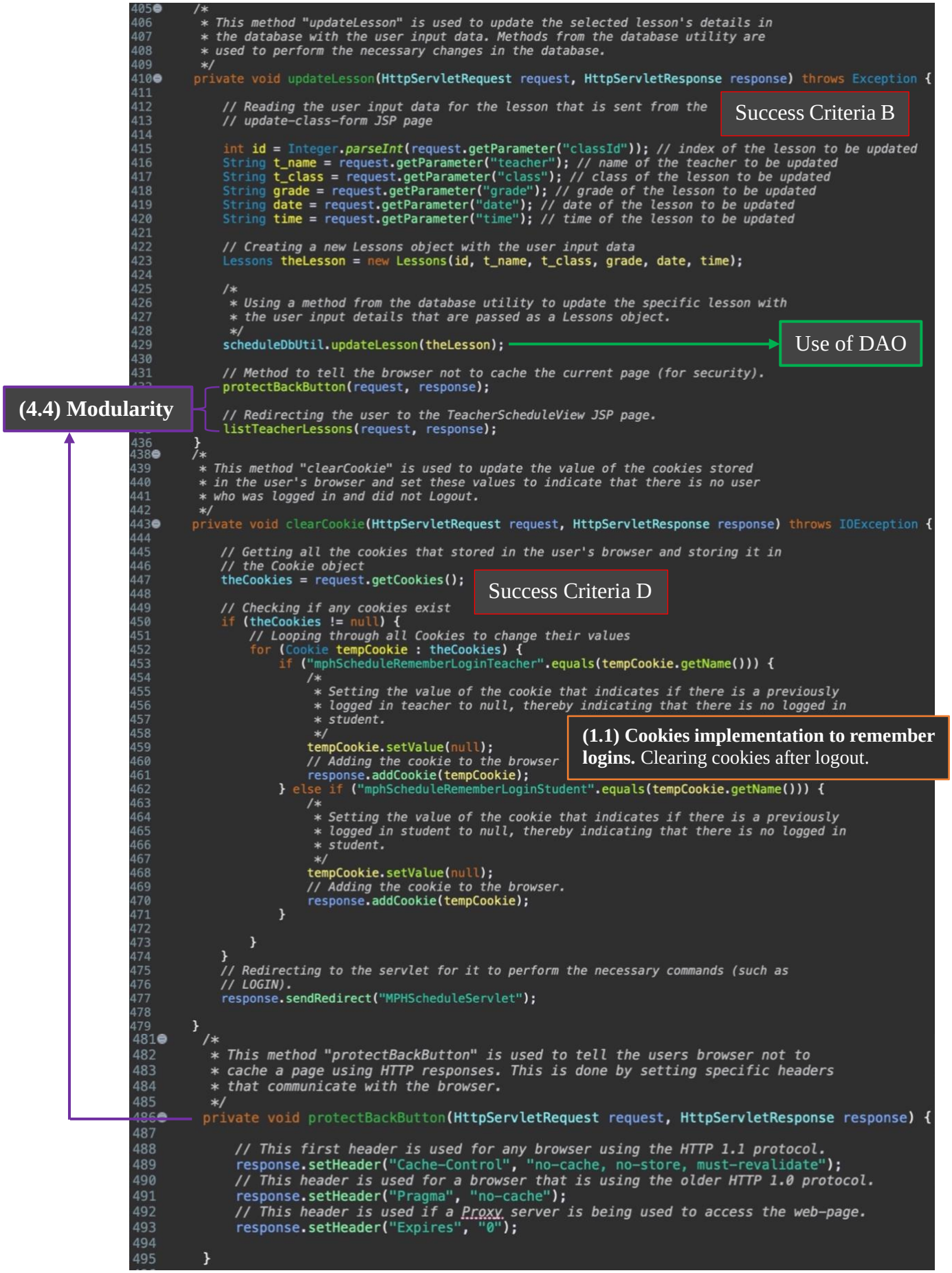
Commented Screenshots


```java
/*
 * This method "listLessons" uses the database utility to get the array list of
 * lessons and sends the data to the DefaultScheduleView JSP page. This page is
 * for the students.
 */
private void listLessons(HttpServletRequest request, HttpServletResponse response) throws Exception {

    /*
     * Getting the Array list of lessons from the database using the database
     * utility. An Array List is used as each lesson has multiple columns and there
     * are a total of 5 Lessons that exist in the database. An Array List provides
     * access to the data of each column as it is stored as a list. The multiple (5)
     * lists make up the Array.
     */
    List<Lessons> lessons = scheduleDbUtil.getLessons();

    // Assigning the Array List as an HTTP attribute to send to the JSP page.
    request.setAttribute("CLASS_LIST", lessons);

    // Method to tell the browser not to cache the current page (for security).
    protectBackButton(request, response);

    // Redirecting the user to the DefaultScheduleView JSP page while passing the
    // Array List attribute.
    RequestDispatcher default_dispatcher = request.getRequestDispatcher("/DefaultScheduleView.jsp");
    default_dispatcher.forward(request, response);
}
/*
 * This method "listTeacherLessons" uses the database utility to get the array
 * list of lessons and sends the data to the TeacherScheduleView JSP page. This
 * page is for the teachers and contains the "Edit" and "Delete" buttons.
 */
private void listTeacherLessons(HttpServletRequest request, HttpServletResponse response) throws Exception {

    // Getting the Array list of lessons from the database using the database
    // utility.
    List<Lessons> lessons = scheduleDbUtil.getLessons();

    // Assigning the Array List as an HTTP attribute to send to the JSP page.
    request.setAttribute("CLASS_LIST", lessons);

    // Method to tell the browser not to cache the current page (for security).
    protectBackButton(request, response);

    // Redirecting the user to the TeacherScheduleView JSP page while passing the
    // Array List attribute.
    RequestDispatcher teacher_dispatcher = request.getRequestDispatcher("/TeacherScheduleView.jsp");
    teacher_dispatcher.forward(request, response);

}
```

```java
/*
 * This method "loadAddLesson" is used to pre-populate the fields of the add-lesson-form JSP Page with
 * the Teacher data that exists in the database, such as the teacher name, class and grades taught.
 * This method also provides the necessary input options for the JSP page that correspond to the
 * currently logged in teacher.
 */
private void loadAddLesson(HttpServletRequest request, HttpServletResponse response) throws Exception {

    // Reading the userId of the currently logged in teacher
    int teacher_id = loginDetails.getUser_id();

    // Getting the currently logged teacher's details from database using a method
    // from the database utility
    Teachers theTeacher = scheduleDbUtil.getTeacher(teacher_id);

    // Assigning the specific teacher details as an HTTP attribute to be sent to the
    // JSP page.
    request.setAttribute("THE_TEACHER", theTeacher);

    // Method to tell the browser not to cache the current page (for security).
    protectBackButton(request, response);

    // Redirecting the user to the add-lesson-form JSP page while passing the
    // teacher details attribute.
    RequestDispatcher dispatcher = request.getRequestDispatcher("/add-lesson-form.jsp");
    dispatcher.forward(request, response);
}
```




```java
    /*
     * This method "addLesson" is used to add a lesson to the schedule with the user input data.
     * Methods from the database utility are used to perform the necessary changes in the database.
     */
    private void addLesson(HttpServletRequest request, HttpServletResponse response) throws Exception {

        // Reading the user input data for the lesson that is sent from the
        // add-lesson-form JSP page

        String t_name = request.getParameter("teacher"); // name of the teacher to be updated
        String t_class = request.getParameter("class"); // class of the lesson to be updated
        String grade = request.getParameter("grade"); // grade of the lesson to be updated
        String date = request.getParameter("date"); // date of the lesson to be updated
        String time = request.getParameter("time"); // time of the lesson to be updated

        // Creating a new Lessons object with the user input data
        Lessons theLesson = new Lessons(t_name, t_class, grade, date, time);

        /*
         * Using a method from the database utility to update the specific lesson with
         * the user input details that are passed as a Lessons object.
         */
        scheduleDbUtil.addLesson(theLesson);

        // Method to tell the browser not to cache the current page (for security).
        protectBackButton(request, response);

        // Redirecting the user to the TeacherScheduleView JSP page.
        listTeacherLessons(request, response);
    }
    /*
     * This method "clearLesson" is used to essentially "delete" a lesson



     */
    private void clearLesson(HttpServletRequest request, HttpServletResponse response) throws Exception {

        // Reading the index of the lesson that the user has requested to delete from
        // the TeacherScheduleView JSP page.
        String theLessonId = request.getParameter("classId");

        /*
         * Deleting/clearing the lesson from the database using a method from the
         * database utility while passing the lesson index that has to be deleted.
         */
        scheduleDbUtil.deleteLesson(theLessonId);

        // Method to tell the browser not to cache the current page (for security).
        protectBackButton(request, response);
        // Redirecting the user to the TeacherScheduleView JSP page.
        listTeacherLessons(request, response);

    }
    /*
     * This method "editLesson" is used to pre-populate the fields of the
     * update-class-form JSP page with any previous data that exists in the database
     * for the particular selected lesson to edit. This method also provides the
     * necessary input options for the JSP page that correspond to the currently
     * logged in teacher.
     */
    private void editLesson(HttpServletRequest request, HttpServletResponse response) throws Exception {

        // Reading the index of the lesson that the user has requested to edit from the
        // TeacherScheduleView JSP page.
        String theLessonId = request.getParameter("classId");

        // Reading the userId of the currently logged in teacher
        int teacher_id = loginDetails.getUser_id();

        // Getting the lesson for the specified lesson id from the database using a
        // method from the database utility
        Lessons theLesson = scheduleDbUtil.getLesson(theLessonId);
        // Getting the currently logged teacher's details from database using a method
        // from the database utility
        Teachers theTeacher = scheduleDbUtil.getTeacher(teacher_id);

        // Assigning the specific lesson details as an HTTP attribute to be sent to the
        // JSP page.
        request.setAttribute("THE_CLASS", theLesson);
        // Assigning the specific teacher details as an HTTP attribute to be sent to the
        // JSP page.
        request.setAttribute("THE_TEACHER", theTeacher);

        // Method to tell the browser not to cache the current page (for security).
        protectBackButton(request, response);
        // Redirecting the user to the update-class-form JSP page while passing the
        // lesson and teacher attributes.
        RequestDispatcher dispatcher = request.getRequestDispatcher("/update-class-form.jsp");
        dispatcher.forward(request, response);

    }
```




```
    /*
     * This method "updateLesson" is used to update the selected lesson's details in
     * the database with the user input data. Methods from the database utility are
     * used to perform the necessary changes in the database.
     */
    private void updateLesson(HttpServletRequest request, HttpServletResponse response) throws Exception {

        // Reading the user input data for the lesson that is sent from the
        // update-class-form JSP page

        int id = Integer.parseInt(request.getParameter("classId")); // index of the lesson to be updated
        String t_name = request.getParameter("teacher"); // name of the teacher to be updated
        String t_class = request.getParameter("class"); // class of the lesson to be updated
        String grade = request.getParameter("grade"); // grade of the lesson to be updated
        String date = request.getParameter("date"); // date of the lesson to be updated
        String time = request.getParameter("time"); // time of the lesson to be updated

        // Creating a new Lessons object with the user input data
        Lessons theLesson = new Lessons(id, t_name, t_class, grade, date, time);

        /*
         * Using a method from the database utility to update the specific lesson with
         * the user input details that are passed as a Lessons object.
         */
        scheduleDbUtil.updateLesson(theLesson);

        // Method to tell the browser not to cache the current page (for security).
        protectBackButton(request, response);

        // Redirecting the user to the TeacherScheduleView JSP page.
        listTeacherLessons(request, response);
    }
    /*
     * This method "clearCookie" is used to update the value of the cookies stored
     * in the user's browser and set these values to indicate that there is no user
     * who was logged in and did not Logout.
     */
    private void clearCookie(HttpServletRequest request, HttpServletResponse response) throws IOException {

        // Getting all the cookies that stored in the user's browser and storing it in
        // the Cookie object
        theCookies = request.getCookies();

        // Checking if any cookies exist
        if (theCookies != null) {
            // Looping through all Cookies to change their values
            for (Cookie tempCookie : theCookies) {
                if ("mphScheduleRememberLoginTeacher".equals(tempCookie.getName())) {
                    /*
                     * Setting the value of the cookie that indicates if there is a previously
                     * logged in teacher to null, thereby indicating that there is no logged in
                     * student.
                     */
                    tempCookie.setValue(null);
                    // Adding the cookie to the browser
                    response.addCookie(tempCookie);
                } else if ("mphScheduleRememberLoginStudent".equals(tempCookie.getName())) {
                    /*
                     * Setting the value of the cookie that indicates if there is a previously
                     * logged in student to null, thereby indicating that there is no logged in
                     * student.
                     */
                    tempCookie.setValue(null);
                    // Adding the cookie to the browser.
                    response.addCookie(tempCookie);
                }

            }
        }
        // Redirecting to the servlet for it to perform the necessary commands (such as
        // LOGIN).
        response.sendRedirect("MPHScheduleServlet");

    }
    /*
     * This method "protectBackButton" is used to tell the users browser not to
     * cache a page using HTTP responses. This is done by setting specific headers
     * that communicate with the browser.
     */
    private void protectBackButton(HttpServletRequest request, HttpServletResponse response) {

        // This first header is used for any browser using the HTTP 1.1 protocol.
        response.setHeader("Cache-Control", "no-cache, no-store, must-revalidate");
        // This header is used for a browser that is using the older HTTP 1.0 protocol.
        response.setHeader("Pragma", "no-cache");
        // This header is used if a Proxy server is being used to access the web-page.
        response.setHeader("Expires", "0");

    }
```

VI The (3) JavaServer Pages

The **(3) JSP Pages** are used to dynamically generate HTML UI and display the data sent from the Servlet (which retrieves the data from the database using the DAO). The JSP pages use minimal scriptlet code (other than cookie implementation) and instead use the **(3.1) JSP Standard Tag Library** for easier readability of code and allows easier editing of code for any necessary updates to the product. The code is shown below.

Commented Screenshots

login.jsp – Login page JSP code

(3.1) JSP Standard Tag Library tags used to handle login cookies.

```
login.jsp
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
4  <!DOCTYPE html>
5  <html>
6  <head>
7  <meta charset="UTF-8">
8  <title>MPH Scheduler | User Login</title>
9  <link type ="text/css" rel="stylesheet" href="css/style.css">
10 <link type ="text/css" rel="stylesheet" href="css/add-class-style.css">
11 </head>
12 <body>
13
14     <div id ="wrapper">
15         <div id="header">
16             <h2>MPH Schedule</h2>
17         </div>
18     </div>
19
20     <div ID="container">
21         <h3>Student/Teacher Login</h3>
22
23     <!-- Variable for indicating any incorrect login information. -->
24         <c:set var="loginError" value="true"/>
25
26     <!-- Variables for storing Teacher and Student cookies. -->
27         <c:set var="theTeacherCookie" scope="session" value="${cookie.mphScheduleRememberLoginTeacher.value}"/>
28         <c:set var="theStudentCookie" scope="session" value="${cookie.mphScheduleRememberLoginStudent.value}"/>
29
30     <!-- Variables for storing the Teacher and Student cookie values. -->
31         <c:set var="teacherCookieValue" value="rememberTeacher"/>
32         <c:set var="studentCookieValue" value="rememberStudent"/>
33
34     <!-- URL variable for redirecting to the servlet with a LIST_TEACH command to display the Teacher Schedule. -->
35         <c:url var="servletRedirectTeacher" value="MPHScheduleServlet">
36             <c:param name="command" value="LIST_TEACH"/>
37         </c:url>
38
39     <!-- URL variable for redirecting to the servlet with a LIST command to display the Default Schedule. -->
40         <c:url var="servletRedirectStudent" value="MPHScheduleServlet">
41             <c:param name="command" value="LIST"/>
42         </c:url>
43
44     <!-- Conditional statement to test whether either cookie indicates a logged in user and redirecting to the
45          appropriate schedule view using the JSTL URL variables.     -->
46         <c:choose>
47             <c:when test="${theTeacherCookie == teacherCookieValue}">
48                 <c:redirect url="${servletRedirectTeacher}"/>
49             </c:when>
50             <c:when test="${theStudentCookie == studentCookieValue}">
51                 <c:redirect url="${servletRedirectStudent}"/>
52             </c:when>
53         </c:choose>
```

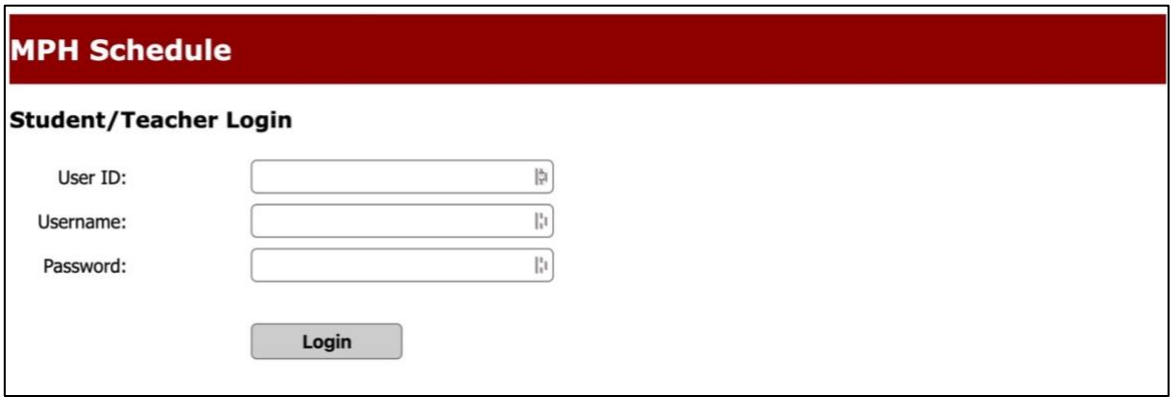
(9) HTML & CSS for UI Creation

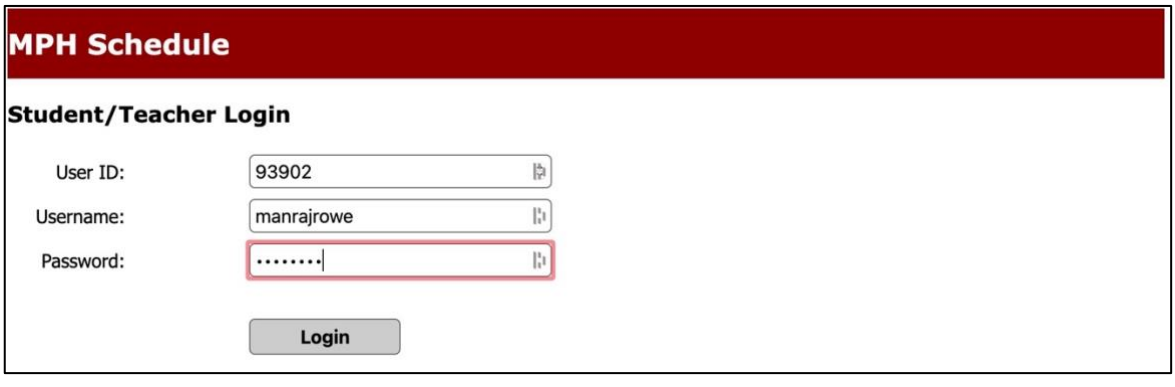
Contd.

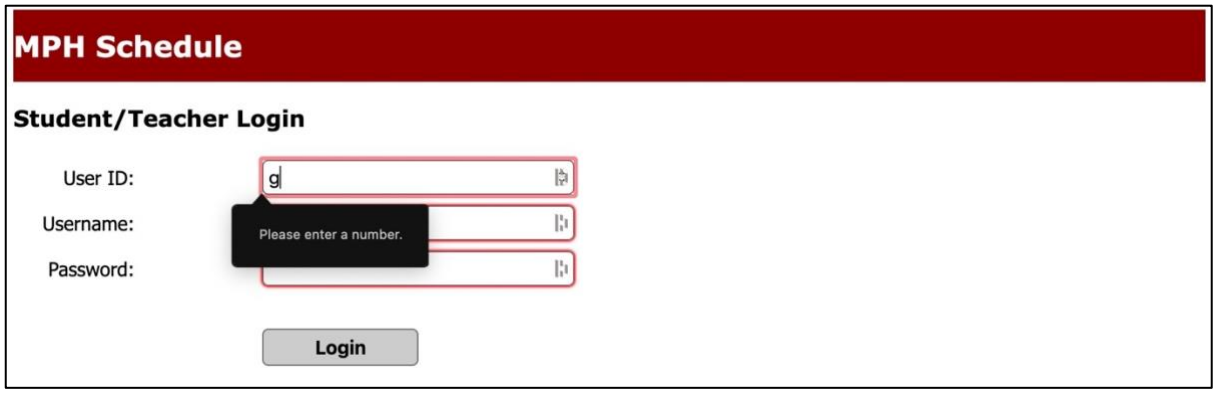
(9) HTML & CSS for UI Creation

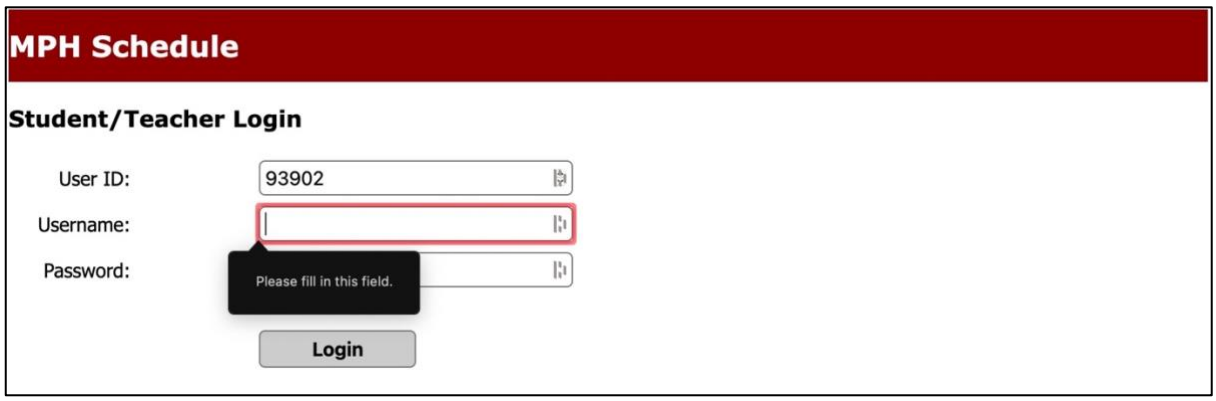
```
55      <!-- Form details are sent to the default POST method of the servlet (in this case doPost). -->
56          <form action="MPHScheduleServlet" method='POST'>
57      <!-- Adding a hidden input field that holds the LOGGEDIN command to be sent to the MPHScheduleServlet -->
58          <input type="hidden" name="command" value="LOGGEDIN"/>
59                  <table>
60                      <tbody>
61
62                          <tr>
63                          <!-- User inputs user id as a number. -->
64                              <td><label>User ID:</label></td>
65                              <td><input type="number" name="userId" required/></td>
66                          </tr>
67
68                          <tr>
69                          <!-- User inputs username as text. -->
70                              <td><label>Username:</label></td>
71                              <td><input type="text" name="username"  required/></td>
72                          </tr>
73
74                          <tr>
75                          <!-- User inputs password as password (text is replaced by bullets ●). -->
76                              <td><label>Password:</label></td>
77                              <td><input type="password" name="password"  required/></td>
78                          </tr>
79                          <tr>
80                          <!-- Displaying an incorrect details error if the servlet parameter received indicates
81                               incorrect login information. -->
82                              <c:if test="${IncorrectUserOrPass == loginError}">
83                                  <p style="color:Red;">Incorrect Details!</p>
84                              </c:if>
85                          </tr>
86                          <tr>
87                              <td><label></label></td>
88                              <td><input type="submit" value="Login" class="save" /></td>
89                          </tr>
90
91                      </tbody>
92                  </table>
93              </form>
94
95              <div style="clear: both;"></div>
96
97      </div>
98
99  </body>
100 </html>
```

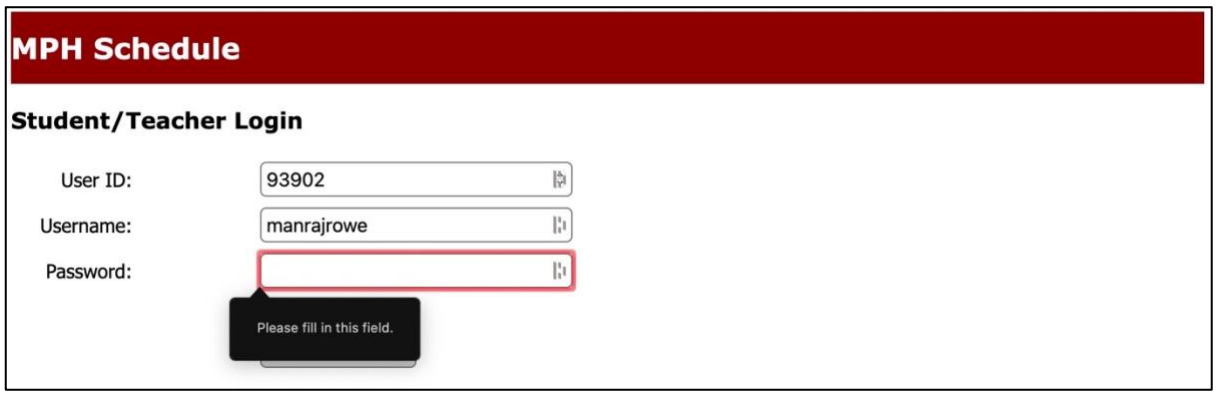
Screenshots of Live Login Page:

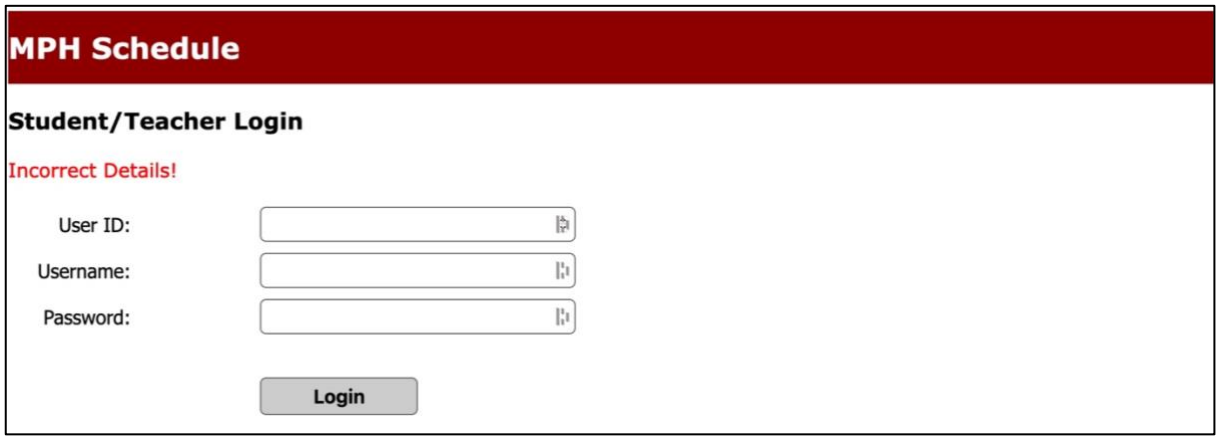





MPH Schedule
Student/Teacher Login
User ID:
g
Username:
Please enter a number.
Password:
Login


MPH Schedule
Student/Teacher Login
User ID:
93902
Username:
Password:
Please fill in this field.
Login


MPH Schedule
Student/Teacher Login
User ID:
93902
Username:
manrajrowe
Password:
Please fill in this field.


MPH Schedule
Student/Teacher Login
Incorrect Details!
User ID:
Username:
Password:
Login

TeacherScheduleView.jsp – JSP Page for the teacher schedule which contains Add, Edit and Delete buttons (Success Criteria B, G, H).

Success Criteria A, B, C, D, G

(1.1) Cookies Implementation to remember logins

Java Scriptlet code

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>MPH Scheduler | Teacher Schedule</title>
<link type="text/css" rel="stylesheet" href="css/style.css">
</head>

<body>

<%
    /*
     * This scriplet is used to store a cookie in the user's browser with
     * a value that indicates that a teacher is currently logged in.
     */

    // String to store the value of the Cookie.
    String remember = "rememberTeacher";

    /* Creating a new Cookie called mphScheduleRememberLoginTeacher and storing
       the previously created Cookie value in this Cookie. */
    Cookie theCookie = new Cookie("mphScheduleRememberLoginTeacher", remember);

    /*
     * Setting the expiry for the Cookie as 14 days. This is done as per the
     * request of the client. A user must be remembered for 14 days till they
     * are automatically logged after this period. The age is set in seconds,
     * therefore the argument is passed as:
     * 14 (days) X 24 (hours) X 60 (minutes) X 60 (seconds).
     */
    theCookie.setMaxAge(14*24*60*60);

    //Adding the Cookie to the user's browser.
    response.addCookie(theCookie);

%>
```

(9) HTML & CSS for UI Creation

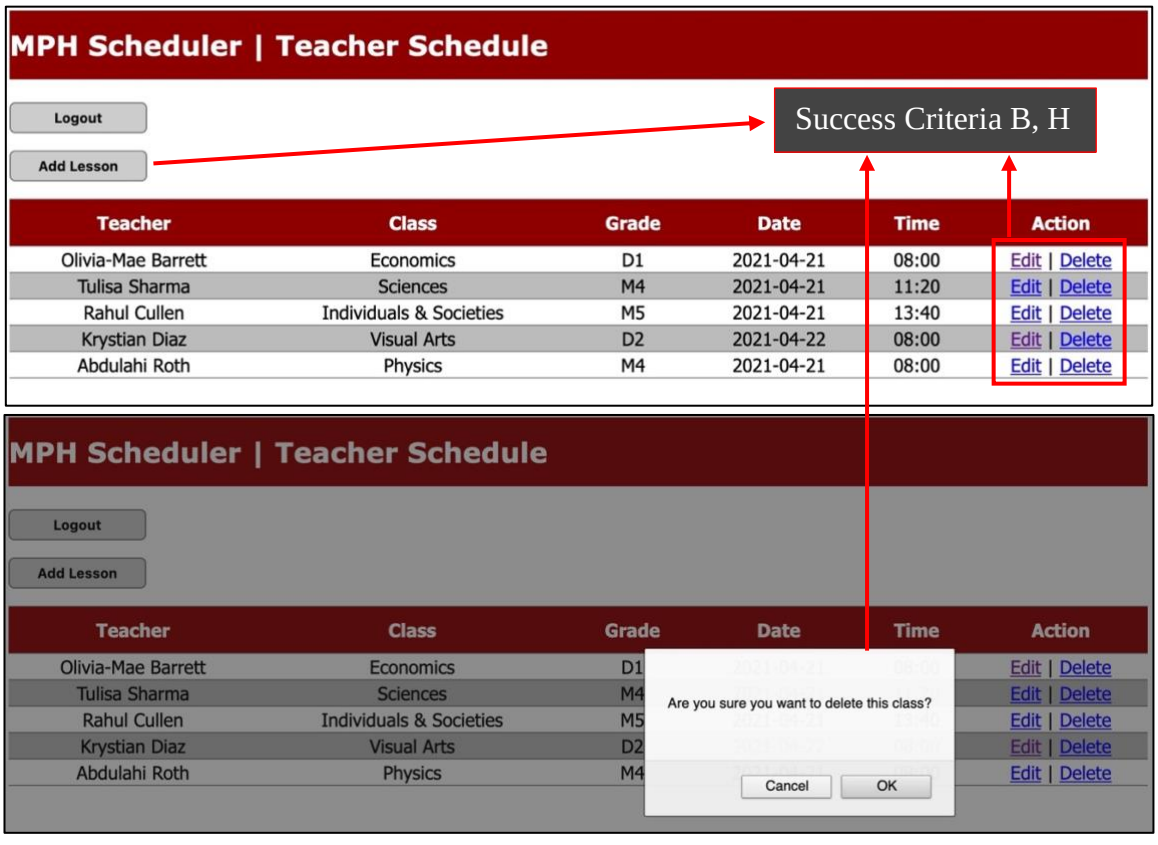
```jsp
<div id="wrapper">
<div id="header">
  <h2>MPH Scheduler | Teacher Schedule</h2>
  </div>
  </div>

<div id="container">
<div id="content">

<!-- Button that is used to Logout by redirecting to the MPHScheduleServlet with a
      "LOGOUT" command as a parameter. -->
<form action="MPHScheduleServlet" method="POST">
      <input type="hidden" name="command" value="LOGOUT">
      <input type='submit' value='Logout' class="add-class-button" formnovalidate/>
  </form>

<!-- Button that is used to pre-populate the form data in the add-lesson-form JSP page
      and redirecting to it by redirecting to the MPHScheduleServlet with a "LOAD_ADD"
      command as a parameter. -->
<form action="MPHScheduleServlet" method="POST">
      <input type="hidden" name="command" value="LOAD_ADD">
      <input type='submit' value='Add Lesson' class="add-class-button" formnovalidate/>
  </form>

<table>

<tr>

      <th>Teacher</th>
      <th>Class</th>
      <th>Grade</th>
      <th>Date</th>
      <th>Time</th>
      <th>Action</th>

  </tr>
```

```jsp
<!-- Using JSTL tags to loop through the data stored in the ArrayList sent by the MPHScheduleServlet -->
    <c:forEach var="tempTeacher" items="${CLASS_LIST}">

    <!-- JSTL URL that is used to pre-populate the form data in the update-class-form JSP page and
         and redirecting to it by redirecting to the MPHScheduleServlet with a "LOAD"
         command as a parameter. The index of the class clicked is also sent as a parameter. -->
        <c:url var = "tempLink" value="MPHScheduleServlet">
            <c:param name="command" value="LOAD" />
            <c:param name="classId" value="${tempTeacher.index}" />
        </c:url>

        <!-- JSTL URL that is used to delete the selected lesson from the schedule by redirecting to the
             MPHScheduleServlet with a "DELETE" command and the index of the lesson that has chosen to be
             deleted as parameters. -->
        <c:url var = "deleteLink" value="MPHScheduleServlet">
            <c:param name="command" value="DELETE" />
            <c:param name="classId" value="${tempTeacher.index}" />
        </c:url>

        <tr>
            <!-- Displaying teacher name -->
            <td> ${tempTeacher.getT_name()}</td>

            <!-- Displaying class taught -->
            <td> ${tempTeacher.getT_class()}</td>

            <!-- Displaying grade taught -->
            <td> ${tempTeacher.getGrade()}</td>

            <!-- Displaying date of the lesson -->
            <td> ${tempTeacher.getDate()}</td>

            <!-- Displaying time of the lesson -->
            <td> ${tempTeacher.getTime()}</td>

            <!-- Link to edit the particular lesson using the "tempLink" JSTL URL variable -->
            <td> <a href="${tempLink}">Edit</a>
                 |
            <!-- Link to delete the particular lesson using the "deleteLink" JSTL URL variable.
                 An additional JavaScript element is added to confirm the deletion of the lesson. -->
                <a href="${deleteLink}"
                onclick="if (!(confirm('Are you sure you want to delete this class?'))) return false">Delete</a>
            </td>
        </tr>

    </c:forEach>

</table>
</div>
</div>
</body>
</html>
```

Success Criteria B, H

(9) HTML & CSS for UI Creation

Screenshots of Live Teacher Schedule View Page:



add-lesson-form.jsp – Page/Form to accept Teacher input for adding a new lesson (Success Criteria B, E, F).

add-lesson-form.jsp

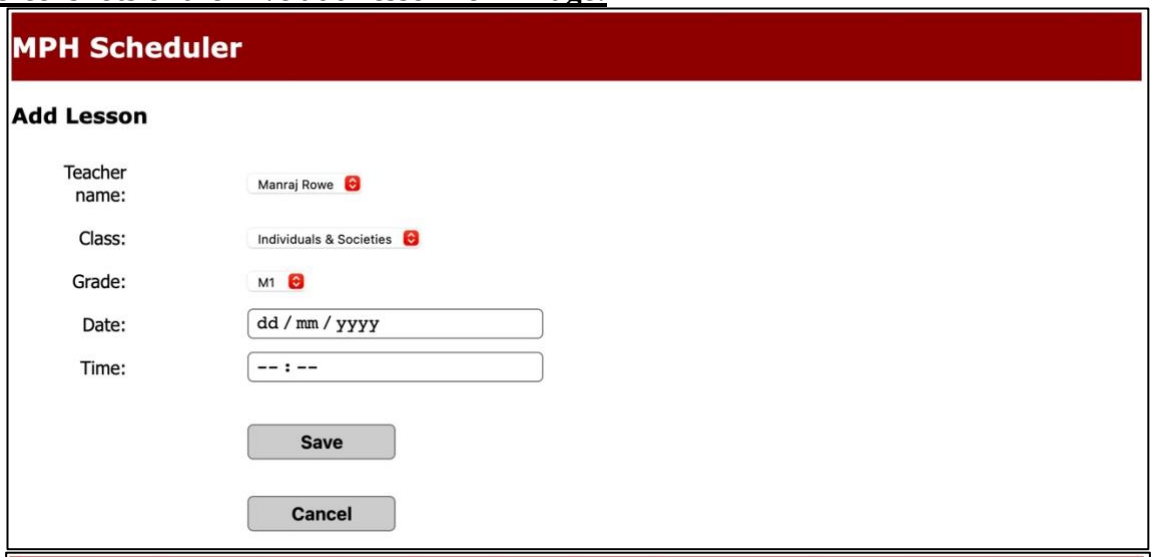
```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>MPH Scheduler | Teacher Schedule</title>
<link type ="text/css" rel="stylesheet" href="css/style.css">
<link type ="text/css" rel="stylesheet" href="css/add-class-style.css">
</head>
<body>

    <div id ="wrapper">
        <div id="header">
            <h2>MPH Scheduler</h2>
        </div>
    </div>

    <div ID="container">
        <h3>Add Lesson</h3>

        <!-- Form details are sent to the default POST method of the servlet (in this case doPost). -->
        <form action="MPHScheduleServlet" method='POST'>

            <!-- Adding a hidden input field that holds the ADD command to be sent to the MPHScheduleServlet -->
                <input type="hidden" name="command" value='ADD' />
            <!-- JSTL URL that is used to redirect to the TeacherSchedule View JSP Page by sending a
                 "LIST_TEACH" command to the MPHScheduleServlet -->
                <c:url var = "outLink" value="MPHScheduleServlet">
                    <c:param name="command" value="LIST_TEACH" />
                </c:url>
            <table>
                <tbody>

                    <tr>
                        <td><label>Teacher name:</label></td>
                        <td>
                            <select name="teacher" class ="select" required>
                                <option value = "${THE_TEACHER.teacher_name}" selected>${THE_TEACHER.teacher_name}</option>
                            </select>
                        </td>
                    </tr>

                    <tr>
                        <td><label>Class:</label></td>
                        <td>
                            <select name="class" class ="select" required>
                                <option value = "${THE_TEACHER.teacher_lessons}" selected>${THE_TEACHER.teacher_lessons}</option>
                            </select>
                        </td>
                    </tr>

                    <tr>
                        <td><label>Grade:</label></td>
                        <td>
                            <select name="grade" class ="select" required>
                            <!-- Using JSTL to loop through the array of grades taught by using the toGradeArray()
                                    method to split the String of the grades taught by the teacher and display each
                                    grade as a separate drop-down list option. -->
                                <c:forEach var = "gradeTaught" items="${THE_TEACHER.toGradeArray()}">
                                    <option value = "${gradeTaught}" >${gradeTaught}</option>
                                </c:forEach>
                            </select>
                        </td>
                    </tr>

                    <tr>
                        <td><label>Date:</label></td>
                        <td><input type="date" name="date" min=2021-01-01 max=2040-01-01 required/></td>
                    </tr>

                    <tr>
                        <td><label>Time:</label></td>
                        <td><input type="time" name="time" required/></td>
                    </tr>

                    <tr>
                        <td><label></label></td>
                        <td><input type="submit" value="Save" class="save" /></td>
                    </tr>

                    <tr>
                        <td><label></label></td>
                        <!-- Cancel button to redirect back to the TeacherScheduleView -->
                        <td><a href='${outLink}'><input type="button" value="Cancel" class="save" formnovalidate/></a></td>
                    </tr>

                </tbody>
            </table>
        </form>

        <div style="clear: both;"></div>

    </div>

</body>
</html>
```

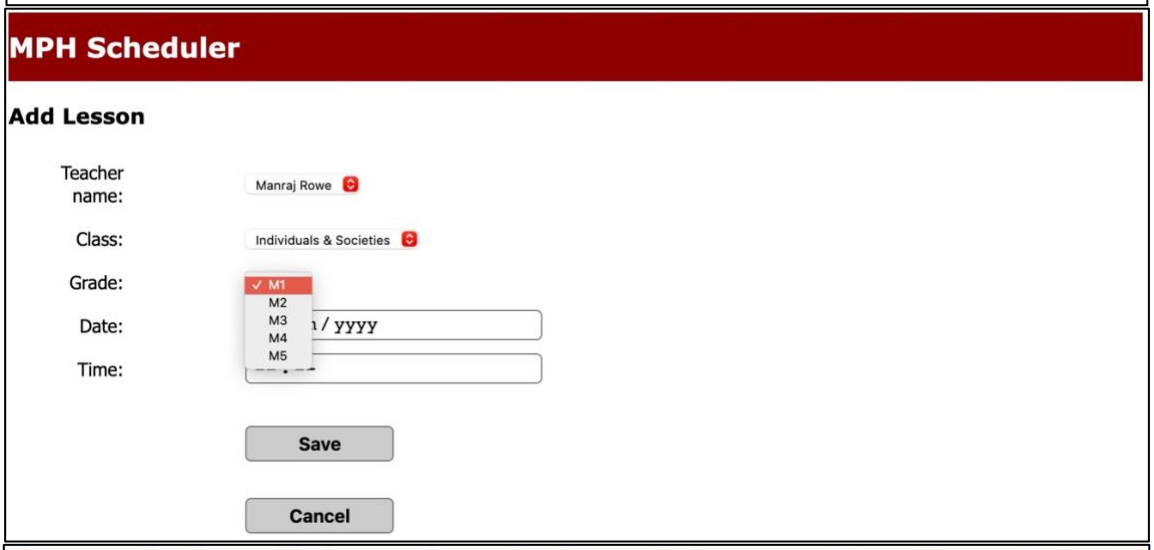
(9) HTML & CSS for UI Creation

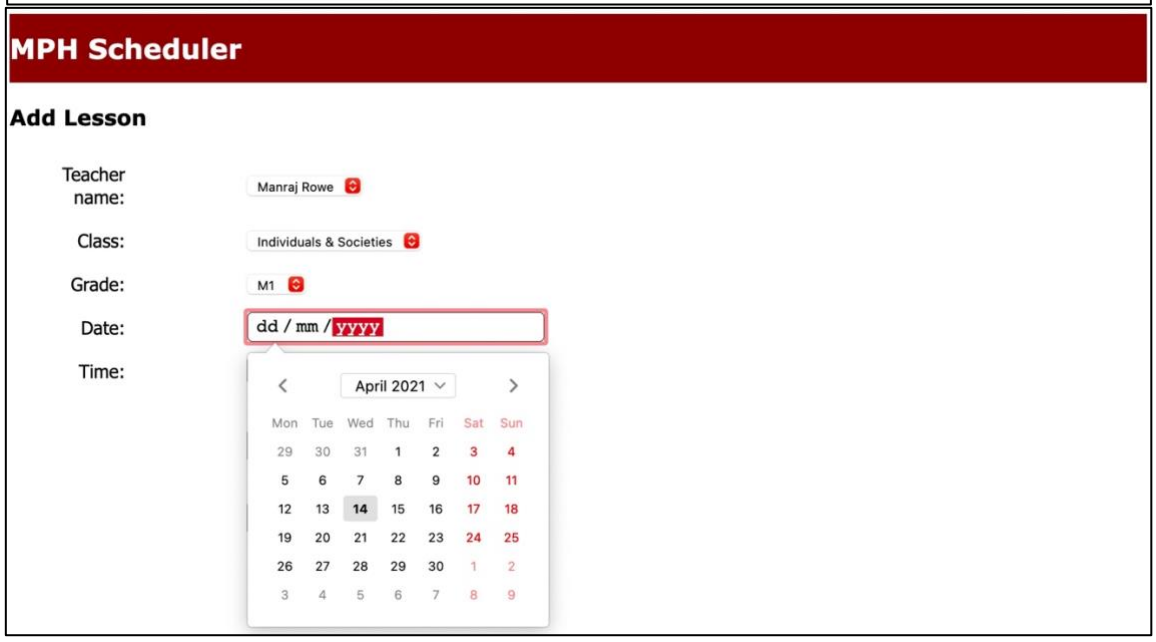
Success Criteria E, F

(9) HTML & CSS for UI Creation

Screenshots of the Live add-lesson-form Page:







93901	Rahul Cullen	Individuals & Societies	M3 M4 M5
93902	Manraj Rowe	Individuals & Societies	M1 M2 M3 M4 M5

Currently Logged in User's details in the database table

update-class-form.jsp – Page to accept Teacher input for the Lesson details to be edited.

update-class-form.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>MPH Scheduler | Teacher Schedule</title>
<link type ="text/css" rel="stylesheet" href="css/style.css">
<link type ="text/css" rel="stylesheet" href="css/add-class-style.css">
</head>
<body>

    <div id ="wrapper">
        <div id="header">
            <h2>MPH Scheduler</h2>
        </div>
    </div>

    <div ID="container">
        <h3>Update Class</h3>

        <!-- Form details are sent to the default POST method of the servlet (in this case doPost). -->
        <form action="MPHScheduleServlet" method='POST'>

            <!-- Adding a hidden input field that holds the UPDATE command to be sent to the MPHScheduleServlet -->
                <input type="hidden" name="command" value='UPDATE' />
            <!-- A hidden input field that holds the current lesson's id to be sent to the MPHScheduleServlet -->
                <input type="hidden" name="classId" value="${THE_CLASS.index}" />

               <table>
                   <tbody>

                       <tr>
                           <td><label>Teacher name:</label></td>
                           <td>
                               <select name="teacher" class ="select" required>
                               <!-- The first selected option is selected by default and is the previous data of the lesson
                                    that is pre-populated using the "THE_CLASS" attribute. The second option is the name of
                                    the currently logged in teacher. -->
                                   <option value = "${THE_CLASS.t_name}" selected>${THE_CLASS.t_name}</option>
                                   <option value = "${THE_TEACHER.teacher_name}" >${THE_TEACHER.teacher_name}</option>
                               </select>
                           </td>
                       </tr>

                       <tr>
                           <td><label>Class:</label></td>
                           <td>
                               <select name="class" class ="select" required>
                               <!-- The first selected option is selected by default and is the previous data of the lesson
                                    that is pre-populated using the "THE_CLASS" attribute. The second option is the class
                                    taught by the currently logged in teacher. -->
                                   <option value = "${THE_CLASS.t_class}" selected>${THE_CLASS.t_class}</option>
                                   <option value = "${THE_TEACHER.teacher_lessons}" >${THE_TEACHER.teacher_lessons}</option>
                               </select>
                           </td>
                       </tr>

                       <tr>
                           <td><label>Grade:</label></td>
                           <td>
                               <select name="grade" class ="select" required>
                               <!-- The first selected option is selected by default and is the previous data of the lesson
                                    that is pre-populated using the "THE_CLASS" attribute.-->
                                   <option value = "${THE_CLASS.grade}" selected>${THE_CLASS.grade}</option>
                                   <!-- Using JSTL to loop through the array of grades taught by using the toGradeArray()
                                        method to split the String of the grades taught by the teacher and display each
                                        grade as a separate drop-down list option. -->
                                   <c:forEach var = "gradeTaught" items="${THE_TEACHER.toGradeArray()}">
                                   <option value = "${gradeTaught}" >${gradeTaught}</option>
                                   </c:forEach>
                               </select>
                           </td>
                       </tr>

                       <tr>
                           <td><label>Date:</label></td>
                           <!-- The previous date of the lesson is pre-populated using the "THE_CLASS" attribute. The maximum
                                and minimum dates to select are also set. -->
                           <td><input type="date" name="date" value="${THE_CLASS.date}"
                                   min=2021-01-01 max=2040-01-01 required/></td>
                       </tr>
```

(9) HTML & CSS for UI Creation

Success Criteria B

Success Criteria E, F

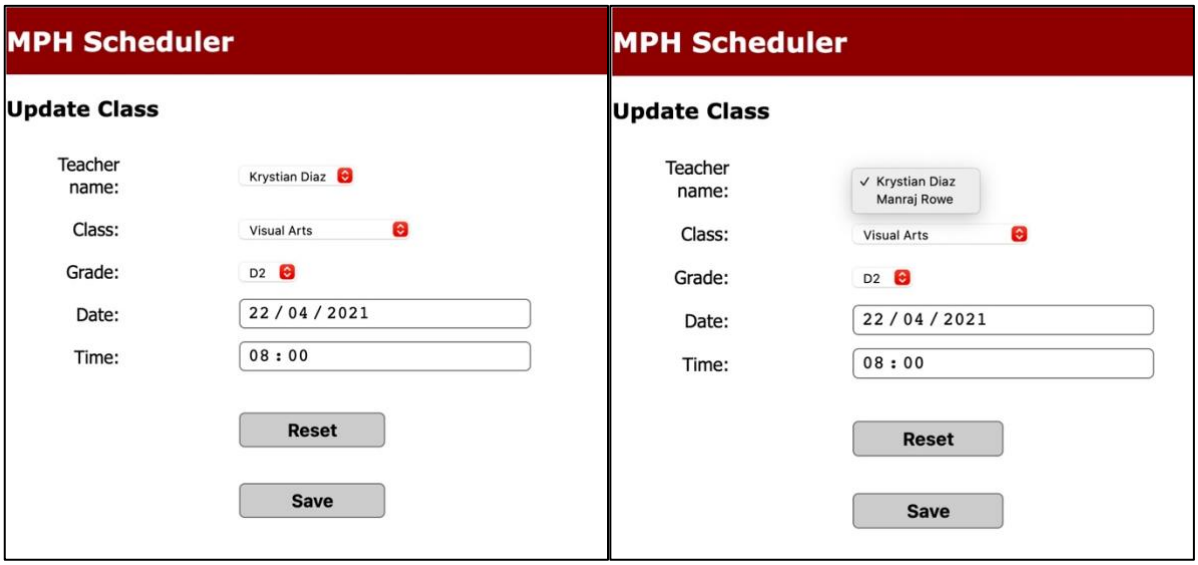
```
                        <tr>
                        <!-- The previous time of the lesson is pre-populated using the "THE_CLASS" attribute. -->
                            <td><label>Time:</label></td>
                            <td><input type="time" name="time"
                                       value="${THE_CLASS.time}" required/></td>
                        </tr>

                        <tr>
                            <td><label></label></td>
                            <td><input type="reset" value="Reset" class="save" /></td>
                        </tr>

                        <tr>
                            <td><label></label></td>
                            <td><input type="submit" value="Save" class="save" /></td>
                        </tr>

                    </tbody>
                </table>
            </form>

            <div style="clear: both;"></div>

    </div>

</body>
</html>
```

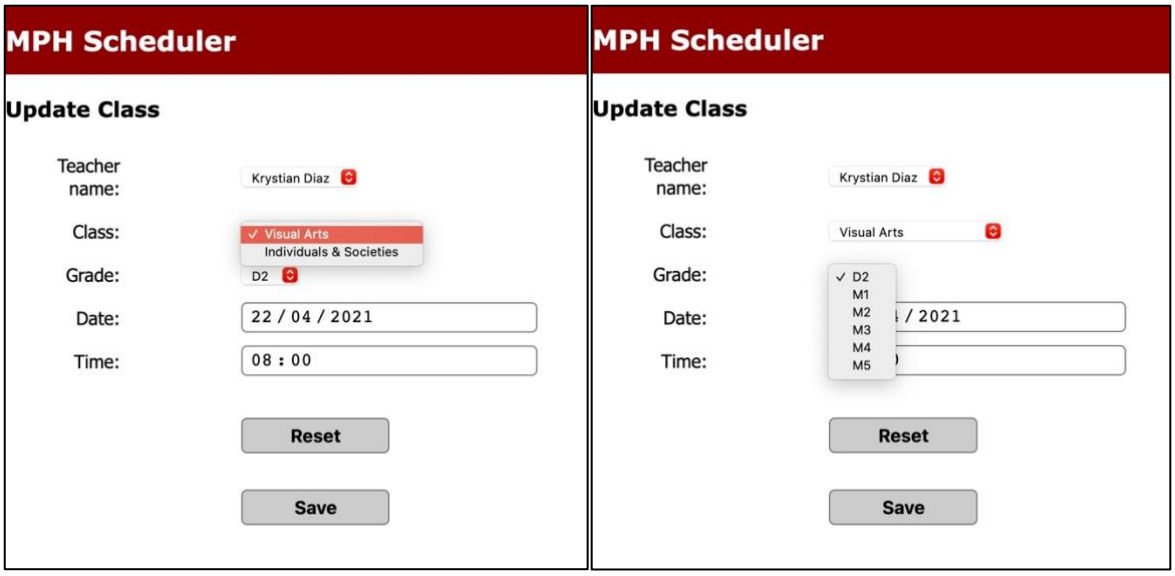
Screenshots of the Live update-class-form Page:

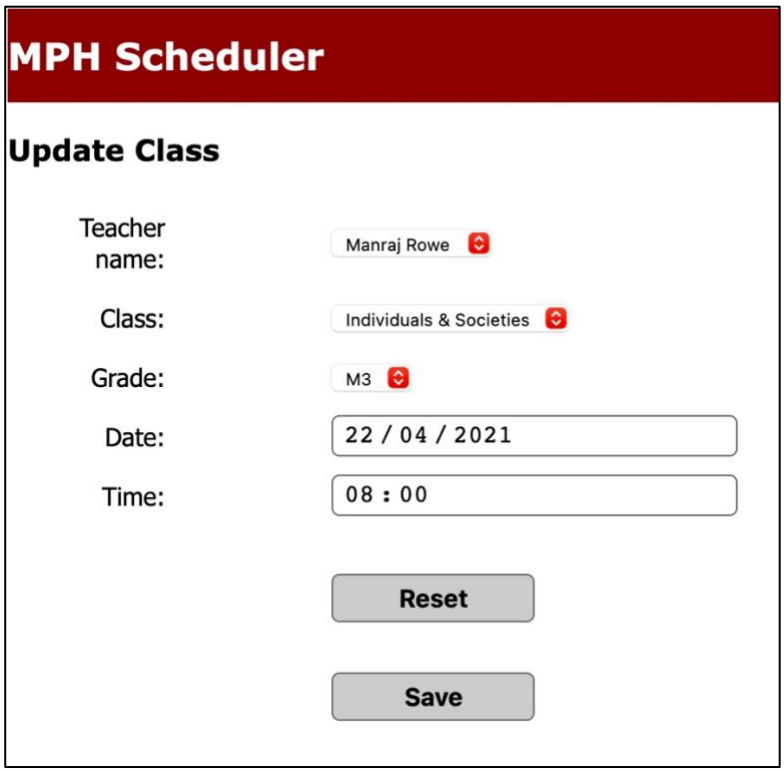
'Edit' button clicked on the TeacherScheduleView Page for the particular lesson:

Teacher	Class	Grade	Date	Time	Action
Olivia-Mae Barrett	Economics	D1	2021-04-21	08:00	Edit | Delete
Tulisa Sharma	Sciences	M4	2021-04-21	11:20	Edit | Delete
Rahul Cullen	Individuals & Societies	M5	2021-04-21	13:40	Edit | Delete
Krystian Diaz	Visual Arts	D2	2021-04-22	08:00	Edit | Delete

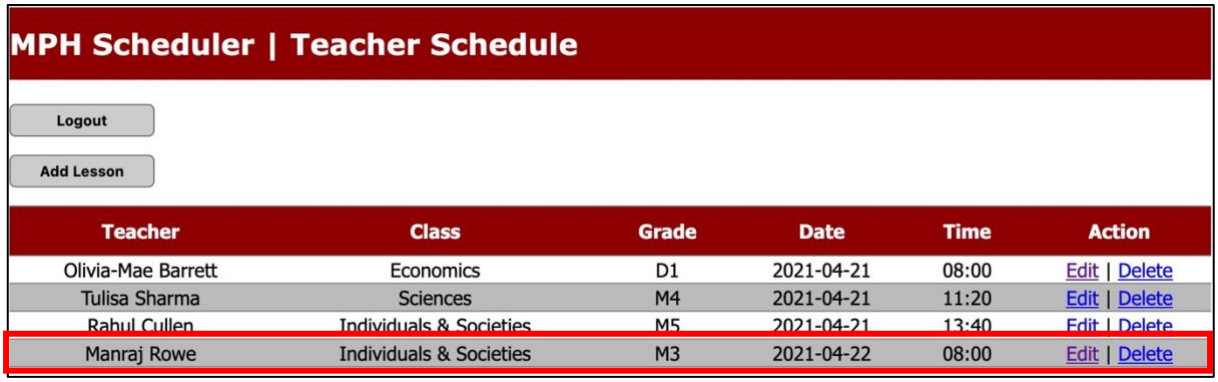
update-class-form.jsp page:







Updated TeacherScheduleView Page:



DefaultScheduleView.jsp – JSP Page for the student schedule which only allows viewing of the lessons (Success Criteria C, G)

DefaultScheduleView.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>MPH Scheduler | Student Schedule</title>
<link type="text/css" rel="stylesheet" href="css/style.css">
</head>

<body>
<%
    /*
     * This scriptlet is used to store a cookie in the user's browser with
     * a value that indicates that a student is currently logged in.
     */

    // String to store the value of the Cookie.
    String remember = "rememberStudent";

    /* Creating a new Cookie called mphScheduleRememberLoginStudent and storing
    the previously created Cookie value in this Cookie. */
    Cookie theCookie = new Cookie("mphScheduleRememberLoginStudent", remember);

    /*
     * Setting the expiry for the Cookie as 14 days. This is done as per the
     * request of the client. A user must be remembered for 14 days till they
     * are automatically logged after this period. The age is set in seconds,
     * therefore the argument is passed as:
     * 14 (days) X 24 (hours) X 60 (minutes) X 60 (seconds).
     */
    theCookie.setMaxAge(14*24*60*60);

    //Adding the Cookie to the user's browser.
    response.addCookie(theCookie);
%>
```

Success Criteria A, C, D, G

(1.1) Cookies Implementation to remember logins

Java Scriptlet code

```jsp
<div id="wrapper">
<div id="header">

<h2>MPH Scheduler | Student Schedule</h2>

</div>
</div>

<div id="container">
<div id="content">

<!-- Button used to Logout -->
<form action="MPHScheduleServlet" method="POST">
    <input type="hidden" name="command" value="LOGOUT">
    <input type='submit' value='Logout' class="add-class-button" formnovalidate/>
</form>

<table>

<tr>

    <th>Teacher</th>
    <th>Class</th>
    <th>Grade</th>
    <th>Date</th>
    <th>Time</th>

</tr>
```

(9) HTML & CSS for UI Creation

```
<!-- Using JSTL tags to loop through the data stored in the ArrayList sent by the
     MPHScheduleServlet -->
    <c:forEach var="tempTeacher" items="${CLASS_LIST}">

        <tr>
            <!-- Displaying teacher name -->
            <td> ${tempTeacher.getT_name()}</td>

            <!-- Displaying class taught -->
            <td> ${tempTeacher.getT_class()}</td>

            <!-- Displaying grade taught -->
            <td> ${tempTeacher.getGrade()}</td>

            <!-- Displaying date of the lesson -->
            <td> ${tempTeacher.getDate()}</td>

            <!-- Displaying time of the lesson -->
            <td> ${tempTeacher.getTime()}</td>
        </tr>

    </c:forEach>
```

(9) HTML & CSS for UI Creation

Screenshot of the Live DefaultScheduleView Page:

MPH Scheduler | Student Schedule

Logout

Teacher	Class	Grade	Date	Time
Olivia-Mae Barrett	Economics	D1	2021-04-21	08:00
Tulisa Sharma	Sciences	M4	2021-04-21	11:20
Rahul Cullen	Individuals & Societies	M5	2021-04-21	13:40
Manraj Rowe	Individuals & Societies	M3	2021-04-22	08:00

VII HTML & CSS UI Creation

All the UI elements were creating using HTML and using cascading style sheets. Tables, forms, links, and labels were used in the HTML and was integrated into the JSP pages. CSS was used to apply the required styles for the HTML tags. The CSS files are shown below.

style.css

```
html, body{
    margin-left:15px; margin-right:15px;
    padding:0px;
    font-family:Verdana, Arial, Helvetica, sans-serif;
}

table {
    border-collapse:collapse;
    border-bottom:1px solid gray;
    font-family: Tahoma,Verdana,Segoe,sans-serif;
    width:72%;
}


th {
    border-bottom:1px solid gray;
    background:none repeat scroll 0 0 #810C09;
    padding:10px;
    color: #FFFFFF;
}

tr {
    border-top:1px solid gray;
    text-align:center;
```

```
}

tr:nth-child(even) {background: #FFFFFF}
tr:nth-child(odd) {background: #BBBBBB}

#wrapper {width: 100%; margin-top: 0px; }
#header {width: 72%; background: #810C09; margin-top: 0px; padding:15px 0px 15px 0px;}
#header h2 {width: 100%; margin:auto; color: #FFFFFF;}
#container {width: 100%; margin:auto}
#container h3 {color: #000;}
#container #content {margin-top: 20px;}

.add-class-button {
    border: 1px solid #666;
    border-radius: 5px;
    padding: 4px;
    font-size: 12px;
    font-weight: bold;
    width: 120px;
    padding: 5px 10px;

    margin-bottom: 15px;
    background: #cccccc;
}
```

add-class-style.css

```
form {
    margin-top: 10px;
}

label {
    font-size: 16px;
    width: 100px;
    display: block;
    text-align: right;
    margin-right: 10px;
    margin-top: 8px;
    margin-bottom: 8px;
}

input {
    width: 250px;
    border: 1px solid #666;
    border-radius: 5px;
    padding: 4px;
    font-size: 16px;
}

.save {
    font-weight: bold;
    width: 130px;
    padding: 5px 10px;
    margin-top: 30px;
    background: #cccccc;
}

table {
    border-style:none;
    width:50%;
}

tr:nth-child(even) {background: #FFFFFF}
tr:nth-child(odd) {background: #FFFFFF}

tr {
    border-style:none;
    text-align:left;
}
```

Word Count: 1074

Works Cited

"ArrayList in Java - javatpoint." *javatpoint*, www.javatpoint.com/java-arraylist. Accessed 30 Dec. 2020.

BalusC. "doGet and doPost in Servlets" #2349741. *Stack Overflow*, 28 Feb. 2010, https://stackoverflow.com/a/2349741. Accessed 20 Oct. 2020.

"GenericServlet (Java EE 6)." *Oracle Java Documentation*, 10 Feb. 2011, docs.oracle.com/javaee/6/api/javax/servlet/GenericServlet.html. Accessed 30 Nov. 2020.

Moumita. "What is object-oriented programming (OOP)?" *Tutorials Point*, 14 Feb. 2018, www.tutorialspoint.com/What-is-object-oriented-programming-OOP. Accessed 29 Dec. 2020.