1. Write a Servlet Programme to print the current date and time along with the timestamp.

Steps to Create the Servlet:

1. **Create a Dynamic Web Project:**
   - Open Eclipse.
   - Go to File -> New -> Dynamic Web Project.
   - Name your project (e.g., DateTimeServletProject).
   - Select a target runtime (e.g., Apache Tomcat).
   - Click Finish.
2. **Create the Servlet:**
   - Right-click on the src folder of your project.
   - Go to New -> Servlet.
   - Name your servlet (e.g., DateTimeServlet).
   - Click Finish.
3. **Implement the Servlet:** Replace the generated code in DateTimeServlet.java with the following code:

```
import java.io.IOException;

import java.io.PrintWriter;

import java.text.SimpleDateFormat;

import java.util.Date;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


@WebServlet("/datetime")

public class DateTimeServlet extends HttpServlet {

   private static final long serialVersionUID = 1L;

   protected void doGet(HttpServletRequest request, HttpServletResponse
response)

         throws ServletException, IOException {
```

```
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // Get the current date and time
        Date now = new Date();
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
        String formattedDate = formatter.format(now);
        // Print the current date and time
        out.println("<html><body>");
        out.println("<h1>Current Date and Time</h1>");
        out.println("<p>" + formattedDate + "</p>");
        out.println("<p>Timestamp: " + now.getTime() + " milliseconds since
January 1, 1970</p>");
        out.println("</body></html>");


        out.close();
    }
}
```

Configure web.xml: If you're using annotations (like @WebServlet), you don't need to configure web.xml

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
         http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
    <servlet>
        <servlet-name>DateTimeServlet</servlet-name>
        <servlet-class>DateTimeServlet</servlet-class>
    </servlet>
```

```
    <servlet-mapping>
        <servlet-name>DateTimeServlet</servlet-name>
        <url-pattern>/datetime</url-pattern>
    </servlet-mapping>
</web-app>
```

Run the Project:

- Right-click on your project, go to Run As -> Run on Server.
- Choose your server and click Finish.
- Once the server starts, open a web browser and go to http://localhost:8080/DateTimeServletProject/datetime.

Question 2

Create an HTML form with the input of student information using HTTP

Protocol and method, then display the input information using Servlet.

STEP 1: SET UP A DYNAMIC WEB PROJECT

1. Open Eclipse and create a new **Dynamic Web Project** (e.g., StudentInfoForm).
2. Click **Finish**.

STEP 2: CREATE AN HTML FORM FOR STUDENT INFORMATION

1. In the **WebContent** folder, right-click and select **New > HTML File**.
2. Name the file studentForm.html.

```
    <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <title>Student Information Form</title>
</head>
<body>
    <h2>Enter Student Information</h2>
    <form action="StudentServlet" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required><br><br>

        <label for="age">Age:</label>
        <input type="number" id="age" name="age" required><br><br>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required><br><br>

        <label for="course">Course:</label>
        <input type="text" id="course" name="course" required><br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

STEP 3: CREATE THE SERVLET TO PROCESS AND DISPLAY STUDENT INFORMATION

1. In the **src** folder, right-click and select **New > Servlet**.
2. Name the Servlet StudentServlet.
3. Package it as needed (e.g., com.example.student).
4. Click **Finish**.

StudentServlet.java

```
   package com.example.student;


import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


@WebServlet("/StudentServlet")

public class StudentServlet extends HttpServlet {

   private static final long serialVersionUID = 1L;


   protected void doPost(HttpServletRequest request, HttpServletResponse
response)

         throws ServletException, IOException {

      // Set response content type

      response.setContentType("text/html");
```

```
        // Retrieve form parameters
        String name = request.getParameter("name");
        String age = request.getParameter("age");
        String email = request.getParameter("email");
        String course = request.getParameter("course");

        // Output the student information
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h2>Student Information</h2>");
        out.println("<p><strong>Name:</strong> " + name + "</p>");
        out.println("<p><strong>Age:</strong> " + age + "</p>");
        out.println("<p><strong>Email:</strong> " + email + "</p>");
        out.println("<p><strong>Course:</strong> " + course + "</p>");
        out.println("</body></html>");
    }
}
```

EXPLANATION OF SERVLET CODE

- **request.getParameter()** retrieves form data submitted through the POST request.
- The servlet generates HTML to display the student's name, age, email, and course.

STEP 4: RUN THE APPLICATION

1. Right-click the project, select **Run As > Run on Server**.

2. Choose your server (e.g., Apache Tomcat) and click **Finish**.

STEP 5: ACCESS THE FORM AND SUBMIT DATA

1. Open a web browser and navigate to http://localhost:8080/StudentInfoForm/studentForm.html.
2. Enter the student information and submit the form.

Question: 3

Write a servlet program to capture client IPs and display it.

STEP 1: CREATE A NEW DYNAMIC WEB PROJECT

1. Open Eclipse and go to **File > New > Dynamic Web Project**.
2. Name your project, for example, ClientIPServlet.
3. Click **Finish**.

STEP 2: CREATE THE SERVLET TO CAPTURE AND DISPLAY CLIENT IP

1. In the **src** folder of your project, right-click and select **New > Servlet**.
2. Name the servlet ClientIPServlet.
3. Package it as needed (e.g., com.example.client).
4. Click **Finish**.

ClientIPServlet.java

```
package com.example.client;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;


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


    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        // Set response content type
        response.setContentType("text/html");


        // Capture client IP address
        String clientIP = request.getRemoteAddr();


        // Display client IP address
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h2>Client IP Address</h2>");
        out.println("<p><strong>Your IP Address:</strong> " + clientIP +
"</p>");
        out.println("</body></html>");
    }
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)

            throws ServletException, IOException {

        doGet(request, response);

    }

}
```

EXPLANATION OF CODE

- **request.getRemoteAddr()** captures the client's IP address from the request object.
- **doGet()** method outputs the IP address in an HTML response.

STEP 3: RUN THE SERVLET

1. Right-click on the project, select **Run As > Run on Server**.
2. Choose your server (e.g., Apache Tomcat) and click **Finish**.

STEP 4: ACCESS THE SERVLET TO SEE THE CLIENT IP

Open a web browser and navigate to:

http://localhost:8080/ClientIPServlet/ClientIPServlet

This should display the client's IP address on the webpage.

Question : 4

Write a servlet program for session management using HTTP Session along with tracking and also use a cookie for session tracking.

STEP 1: CREATE A NEW DYNAMIC WEB PROJECT

1. Open Eclipse, go to **File > New > Dynamic Web Project**.
2. Name your project, for example, SessionManagementServlet.
3. Click **Finish**.

STEP 2: CREATE A SERVLET TO MANAGE THE SESSION AND TRACK USERS

1. In the **src** folder of your project, right-click and select **New > Servlet**.
2. Name the servlet SessionServlet.
3. Package it if needed (e.g., com.example.session).
4. Click **Finish**.

SessionServlet.java

```
  package com.example.session;



import java.io.IOException;

import java.io.PrintWriter;

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.servlet.http.HttpSession;



@WebServlet("/SessionServlet")

public class SessionServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;



    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
```

```
    throws ServletException, IOException {
// Set response content type
response.setContentType("text/html");

// Get or create a new session
HttpSession session = request.getSession();

// Check if the session is new or existing
String message;
if (session.isNew()) {
    message = "Welcome, new user!";
} else {
    message = "Welcome back!";
}

// Set a session attribute
session.setAttribute("username", "StudentUser");

// Retrieve or create a new cookie
Cookie[] cookies = request.getCookies();
boolean foundCookie = false;
for (Cookie cookie : cookies) {
    if (cookie.getName().equals("userSessionCookie")) {
```

```
            foundCookie = true;
            break;
        }
    }

    if (!foundCookie) {
        // If cookie not found, create a new one
        Cookie userSessionCookie = new Cookie("userSessionCookie",
session.getId());
        userSessionCookie.setMaxAge(60 * 60); // Set cookie to expire in 1
hour
        response.addCookie(userSessionCookie);
        message += " A new cookie has been created for your session.";
    } else {
        message += " Cookie found, tracking your session.";
    }

    // Output session and cookie information
    PrintWriter out = response.getWriter();
    out.println("<html><body>");
    out.println("<h2>Session Management with HTTP Session and
Cookies</h2>");
    out.println("<p>" + message + "</p>");
    out.println("<p>Session ID: " + session.getId() + "</p>");
```

```
        out.println("<p>Username (from session attribute): " +
session.getAttribute("username") + "</p>");

        out.println("</body></html>");

    }



    protected void doPost(HttpServletRequest request, HttpServletResponse
response)

            throws ServletException, IOException {

        doGet(request, response);

    }

}
```

EXPLANATION OF CODE

- **HttpSession session = request.getSession();** creates a new session or retrieves an existing one.
- **session.isNew()** checks if the session is newly created or an existing one.
- **Session Attribute**: Sets a session attribute (username) to demonstrate attribute handling.

Cookie Management:

- Checks for an existing cookie named userSessionCookie.
- If no such cookie is found, it creates a new cookie with the session ID and sets it to expire in 1 hour.
- The message informs the user if a new cookie was created or if an existing cookie was found.

STEP 3: RUN THE APPLICATION

1. Right-click on the project, select **Run As > Run on Server**.
2. Choose your server (e.g., Apache Tomcat) and click **Finish**.

STEP 4: ACCESS THE SERVLET

Open a web browser and navigate to:

http://localhost:8080/SessionManagementServlet/SessionServlet

ADDITIONAL NOTES

- If you reload the page, the servlet will recognize the session and the existing cookie.
- Closing the browser or waiting for the session to expire will result in the creation of a new session and cookie on the next visit.

Question 5

Write a CRUD (Create/Save, Read, Edit/Update, Delete) application using servlet. Create a Database named IGNOU, create a table named Student which must capture the student information (basics, contact, enrollment details along with courses). Make necessary assumptions required.

To create a CRUD application with Servlet in Eclipse, we'll follow these steps:

1. Set up the project and database.
2. Create the required database table.
3. Create a set of Servlets for each CRUD operation.
4. Use JDBC to connect to the database.
5. Create JSPs (Java Server Pages) for user interaction.

STEP 1: SET UP THE PROJECT IN ECLIPSE

1. Open Eclipse and create a new **Dynamic Web Project** named StudentCRUDApp.
2. Add the MySQL JDBC driver to the project:
    - Right-click on the project, go to **Build Path > Add External Archives...**.
    - Select the MySQL JDBC .jar file.

STEP 2: CREATE THE DATABASE AND TABLE

Open your MySQL client and create the database and table:

CREATE DATABASE IGNOU;

USE IGNOU;

```
CREATE TABLE Student ( id INT AUTO_INCREMENT PRIMARY KEY,
name  VARCHAR(100),

    email VARCHAR(100) UNIQUE,

    phone VARCHAR(15),

    enrollmentNo VARCHAR(50) UNIQUE,

    course VARCHAR(50),

    enrollmentDate DATE

);
```

STEP 3: CREATE THE DATABASE UTILITY CLASS

Create a utility class to manage database connections.

1. In the src folder, create a package named com.example.utils.
2. Add a class DatabaseConnection.java:

```
package com.example.utils;



import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;



public class DatabaseConnection {

    private static final String URL = "jdbc:mysql://localhost:3306/IGNOU";

    private static final String USER = "root";

    private static final String PASSWORD = "yourpassword";
```

```
    public static Connection initializeDatabase() throws SQLException,
ClassNotFoundException {

        Class.forName("com.mysql.cj.jdbc.Driver");

        return DriverManager.getConnection(URL, USER, PASSWORD);

    }

}
```

STEP 4: CREATE THE CRUD SERVLETS

4.1 CREATE A SERVLET FOR ADDING A NEW STUDENT (CREATE)

1. Create a new Servlet called AddStudentServlet in the package com.example.servlet.

```
package com.example.servlet;



import com.example.utils.DatabaseConnection;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;
```

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

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String name = request.getParameter("name");
        String email = request.getParameter("email");
        String phone = request.getParameter("phone");
        String enrollmentNo = request.getParameter("enrollmentNo");
        String course = request.getParameter("course");
        String enrollmentDate = request.getParameter("enrollmentDate");

        try (Connection conn = DatabaseConnection.initializeDatabase()) {
            String query = "INSERT INTO Student (name, email, phone, enrollmentNo, course, enrollmentDate) VALUES (?, ?, ?, ?, ?, ?)";
            PreparedStatement statement = conn.prepareStatement(query);
            statement.setString(1, name);
            statement.setString(2, email);
            statement.setString(3, phone);
            statement.setString(4, enrollmentNo);
            statement.setString(5, course);
            statement.setString(6, enrollmentDate);
```

```
            statement.executeUpdate();
            response.sendRedirect("listStudents.jsp");  // Redirect to list page after adding
        } catch (Exception e) {
            e.printStackTrace();
            response.getWriter().write("Error occurred while adding student.");
        }
    }}
```

4.2 CREATE SERVLET FOR VIEWING ALL STUDENTS (READ)

1. Create a Servlet called ListStudentsServlet.

```
package com.example.servlet;

import com.example.utils.DatabaseConnection;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

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

    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        try (Connection conn = DatabaseConnection.initializeDatabase();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery("SELECT * FROM Student")) {

            request.setAttribute("resultSet", rs);
            request.getRequestDispatcher("listStudents.jsp").forward(request,
response);
        } catch (Exception e) {
            e.printStackTrace();
            response.getWriter().write("Error occurred while retrieving students.");
        }
    }
}
```

4.3 CREATE SERVLET FOR EDITING STUDENT (UPDATE)

1. Create a Servlet called UpdateStudentServlet.

```
package com.example.servlet;
```

```
import com.example.utils.DatabaseConnection;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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

    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        int id = Integer.parseInt(request.getParameter("id"));
        String name = request.getParameter("name");
        String email = request.getParameter("email");
        String phone = request.getParameter("phone");
        String enrollmentNo = request.getParameter("enrollmentNo");
        String course = request.getParameter("course");
```

```
        String enrollmentDate = request.getParameter("enrollmentDate");


        try (Connection conn = DatabaseConnection.initializeDatabase()) {
            String query = "UPDATE Student SET name = ?, email = ?, phone = ?,
enrollmentNo = ?, course = ?, enrollmentDate = ? WHERE id = ?";
            PreparedStatement statement = conn.prepareStatement(query);
            statement.setString(1, name);
            statement.setString(2, email);
            statement.setString(3, phone);
            statement.setString(4, enrollmentNo);
            statement.setString(5, course);
            statement.setString(6, enrollmentDate);
            statement.setInt(7, id);


            statement.executeUpdate();
            response.sendRedirect("listStudents.jsp");
        } catch (Exception e) {
            e.printStackTrace();
            response.getWriter().write("Error occurred while updating student.");
        }
    }
}
```

4.4 CREATE SERVLET FOR DELETING STUDENT (DELETE)

1. Create a Servlet called DeleteStudentServlet.

```
package com.example.servlet;


import com.example.utils.DatabaseConnection;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


@WebServlet("/DeleteStudentServlet")

public class DeleteStudentServlet extends HttpServlet {

   private static final long serialVersionUID = 1L;


   protected void doGet(HttpServletRequest request, HttpServletResponse
response)

         throws ServletException, IOException {

      int id = Integer.parseInt(request.getParameter("id"));


      try (Connection conn = DatabaseConnection.initializeDatabase()) {

         String query = "DELETE FROM Student WHERE id = ?";
```

```
            PreparedStatement statement = conn.prepareStatement(query);
            statement.setInt(1, id);

            statement.executeUpdate();
            response.sendRedirect("listStudents.jsp");
        } catch (Exception e) {
            e.printStackTrace();
            response.getWriter().write("Error occurred while deleting student.");
        }
    }}
```

STEP 5: CREATE JSP PAGES FOR THE CRUD INTERFACE

5.1 ADDSTUDENT.JSP

A form to add a new student.

5.2 LISTSTUDENTS.JSP

A page to display the list of students with edit and delete links.

5.3 EDITSTUDENT.JSP

A form to edit existing student details.

Each of these JSP pages will use the respective servlet to perform the action.

SESSION 2: JSP

Question : 1

Write JSP Programme to print current date and time along with timestamp, implement auto-refresh of a page.

STEP 1: SET UP A DYNAMIC WEB PROJECT

1. Open Eclipse and create a new **Dynamic Web Project** (e.g., `DateTimeAutoRefresh`).
2. Click **Finish**.

STEP 2: CREATE A JSP PAGE TO DISPLAY CURRENT DATE AND TIME WITH AUTO-REFRESH

1. In the **WebContent** folder, right-click and select **New > JSP File**.
2. Name the file `dateTime.jsp`.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

    <title>Current Date and Time</title>

    <!-- Set the page to refresh every 5 seconds -->

    <meta http-equiv="refresh" content="5">

</head>

<body>

    <h2>Current Date and Time with Timestamp</h2>

    <%

        // Get the current date and time

        java.util.Date currentDate = new java.util.Date();



        // Print current date and time along with timestamp

        out.println("<p><strong>Date and Time:</strong> " +
currentDate.toString() + "</p>");
```

```
        out.println("<p><strong>Timestamp:</strong> " + currentDate.getTime()
+ "</p>");

    %>

</body>

</html>
```

EXPLANATION OF CODE

- **`<meta http-equiv="refresh" content="5">`**: This tag refreshes the page every 5 seconds. You can adjust the value if you want a different refresh interval.
- **`java.util.Date`**: This Java class gets the current date and time.
- **`currentDate.getTime()`**: Prints the timestamp (milliseconds since January 1, 1970, 00:00:00 GMT).

STEP 3: RUN THE APPLICATION

1. Right-click on the project, select **Run As > Run on Server**.
2. Choose your server (e.g., Apache Tomcat) and click **Finish**.

STEP 4: ACCESS THE JSP PAGE

Open a web browser and navigate to:

http://localhost:8080/DateTimeAutoRefresh/dateTime.jsp

Question : 2

Create a JSP page and implement a Scripting Tag, Expression tag and Declaration tag.

STEP 1: SET UP YOUR ECLIPSE ENVIRONMENT

1. **Open Eclipse**.
2. **Create a new Dynamic Web Project**:
    - Go to `File > New > Dynamic Web Project`.
    - Enter a project name, for example, `JSPDemo`.
    - Select the appropriate `Target Runtime` for your server, like Apache Tomcat.
    - Click `Finish`.
3. **Create a JSP Page**:
    - Right-click on the `WebContent` folder within your project.
    - Select `New > JSP File`.
    - Name the file `example.jsp` and click `Finish`.

STEP 2: IMPLEMENT THE JSP TAGS

example.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>JSP Tags Example</title>

</head>

<body>

    <h2>JSP Tags Example</h2>


    <!-- Declaration Tag -->

    <%!

        int counter = 0; // Declares a variable for the entire JSP page

        public String getWelcomeMessage() {

            return "Welcome to JSP!";

        }

    %>


    <p><strong>Welcome Message:</strong> <%= getWelcomeMessage()
%></p>
```

```
    <!-- Expression Tag -->
    <p><strong>Current Counter Value:</strong> <%= ++counter %></p>

    <!-- Scripting Tag -->
    <%
        String name = "Ashitha"; // A local variable within this scriptlet
        out.println("<p><strong>User:</strong> " + name + "</p>");
    %>
</body>
</html>
```

Explanation of Tags

Declaration Tag (<%! ... %>):

- Used to declare variables or methods that are accessible throughout the JSP page.
- In this example, int counter and the getWelcomeMessage method are declared with <%! ... %>.

Expression Tag (<%= ... %>):

- Used to output values directly to the response, equivalent to out.println(...).
- Here, <%= getWelcomeMessage() %> displays the welcome message, and <%= ++counter %> increments and displays the counter.

Scripting Tag (<% ... %>):

- Used to embed Java code directly in the JSP, typically for performing actions or initializing variables.
- Here, <% String name = "Ashitha"; %> initializes a name variable, and out.println(...) outputs it.

Run the JSP Page

1. Right-click on example.jsp.
2. Select Run As > Run on Server.
3. Choose your server and click Finish to deploy and run the JSP page on the server.

Question :3

Import JSTL library in JSP Page and use its following tags:

a. out

b. if

c. forEach

d. choice, when and otherwise

e. url and redirect

STEP 1: SET UP JSTL IN YOUR PROJECT

1. **Download the JSTL Library** (if not already available):
   - If you're using Apache Tomcat, you may already have the JSTL library (`javax.servlet.jsp.jstl.jar` and `javax.servlet.jsp.jstl-api.jar`).
   - If not, you can download it from the Maven Repository.
2. **Add JSTL to Your Project**:
   - Place the `.jar` files in your project's `WebContent/WEB-INF/lib` directory, or add them to your build path if you're using Maven.

STEP 2: IMPORT THE JSTL LIBRARY IN YOUR JSP PAGE

To use JSTL tags in your JSP page, include the JSTL core library directive at the top of the page.

STEP 3: CREATE THE JSP PAGE WITH JSTL TAGS

Now, open or create a new JSP file (e.g., `jstlExample.jsp`) and add the following code, which demonstrates the `out`, `if`, `forEach`, `choose` (choice), `when`, `otherwise`, `url`, and `redirect` tags.

```
jsp
Copy code
<%@ 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>JSTL Tags Example</title>
</head>
<body>
    <h2>JSTL Tags Example</h2>

    <!-- a. out Tag -->
    <c:set var="username" value="Ashitha"/>
    <p><strong>Username:</strong> <c:out value="${username}"/></p>

    <!-- b. if Tag -->
    <c:set var="isLoggedIn" value="true"/>
    <c:if test="${isLoggedIn}">
        <p>Welcome back, <c:out value="${username}"/>!</p>
    </c:if>

    <!-- c. forEach Tag -->
    <h3>Shopping Cart Items:</h3>
    <c:set var="items" value="${['Laptop', 'Smartphone', 'Tablet']}"/>
    <ul>
        <c:forEach var="item" items="${items}">
            <li><c:out value="${item}"/></li>
        </c:forEach>
    </ul>

    <!-- d. choose, when, and otherwise Tags -->
    <c:set var="role" value="admin"/>
    <c:choose>
        <c:when test="${role == 'admin'}">
            <p>Access Level: Administrator</p>
        </c:when>
        <c:when test="${role == 'user'}">
            <p>Access Level: Regular User</p>
        </c:when>
        <c:otherwise>
            <p>Access Level: Guest</p>
        </c:otherwise>
    </c:choose>

    <!-- e. url and redirect Tags -->
    <p><a href="<c:url value='/newPage.jsp'/>">Go to New Page</a></p>

    <c:if test="${role == 'guest'}">
        <c:redirect url="/login.jsp"/>
    </c:if>
</body>
</html>
```

EXPLANATION OF JSTL TAGS

1. **`<c:out>` Tag**:
   - Outputs content to the page. In this example, `<c:out value="${username}"/>` displays the value of `username`.
2. **`<c:if>` Tag**:
   - Used to conditionally render content. Here, it checks if `isLoggedIn` is `true` before displaying a welcome message.

3. **`<c:forEach>` Tag**:
   - Used for iteration, typically with collections or arrays. In this example, it iterates over `items` and displays each item in a list.
4. **`<c:choose>`, `<c:when>`, and `<c:otherwise>` Tags**:
   - These work similarly to an `if-else` structure.
   - `<c:choose>` contains multiple `<c:when>` conditions, with an optional `<c:otherwise>` for the default case.
5. **`<c:url>` and `<c:redirect>` Tags**:
   - `<c:url>` constructs a URL relative to the context root.
   - `<c:redirect>` redirects the page to a new URL. Here, it checks if the user role is `guest` and, if so, redirects to `login.jsp`.

RUN THE JSP PAGE

1. Right-click on `jstlExample.jsp`.
2. Select `Run As > Run on Server`.
3. Choose your server and click `Finish` to deploy and run the JSP page on the server.

Question : 4

Create a JSP Page for database connectivity using JDBC and show the students details from the database created during exercise no 5 in session 1.

Question : 5

Write a CRUD (Create/Save, Read, Edit/Update, Delete) application using servlet. Create a Database named IGNOU, create a table named Student which must capture the student information (basics, contact, enrollment details along with courses). Make necessary assumptions required. in eclipse

- **Database Setup**: You need to create a database called `IGNOU` and a table called `Student` with the necessary fields to store student information.
- **Java Servlet Setup**: You'll use servlets to interact with the database (via JDBC).
- **JSP Pages**: To display and interact with the data from the user.

- **Configuration in Eclipse**: We'll use an IDE (Eclipse) for development.

1. **Database Setup** (MySQL)

   CREATE DATABASE IGNOU;

   USE IGNOU;

   CREATE TABLE Student (

   student_id INT PRIMARY KEY AUTO_INCREMENT,

   first_name VARCHAR(50) NOT NULL,

   last_name VARCHAR(50) NOT NULL,

```
    date_of_birth DATE,
    gender ENUM('Male', 'Female', 'Other'),
    email VARCHAR(100),
    phone_number VARCHAR(15),
    enrollment_number VARCHAR(50) NOT NULL,
    course VARCHAR(100),
    enrollment_date DATE
);
```

2. Set up Eclipse Environment

- **Step 1**: Create a Dynamic Web Project.
  - Go to `File -> New -> Dynamic Web Project`.
  - Name it `StudentCRUDApp`.
- **Step 2**: Add a MySQL JDBC driver (or any other DB driver you're using) in the `lib` folder of the `WebContent/WEB-INF` folder.
  - You can download the MySQL JDBC driver from here.
- **Step 3**: Create the necessary servlets and JSP pages for the CRUD operations.

3. Servlet Code for CRUD Operations

3.1. DATABASE CONNECTION UTILITY

Create a `DBConnection` class to manage database connections:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {
    public static Connection getConnection() throws SQLException {
        try {
```

```
            // Register JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/IGNOU", "root", "password"); // replace with your DB username and password
        } catch (Exception e) {
            throw new SQLException("Database connection failed", e);
        }
    }
}
```

3.2. **Student Model Class**

```
public class Student {
    private int studentId;
    private String firstName;
    private String lastName;
    private String email;
    private String phoneNumber;
    private String enrollmentNumber;
    private String course;
    private String enrollmentDate;

    // Getters and Setters for each field
}
```

3.3. Servlets for CRUD Operations

3.3.1. Create Student (Add Student)

Create a servlet called `AddStudentServlet.java` to handle the Create operation.

```
import javax.servlet.*;

import javax.servlet.http.*;

import java.io.*;

import java.sql.*;


public class AddStudentServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        String firstName = request.getParameter("firstName");

        String lastName = request.getParameter("lastName");

        String email = request.getParameter("email");

        String phoneNumber = request.getParameter("phoneNumber");

        String enrollmentNumber = request.getParameter("enrollmentNumber");

        String course = request.getParameter("course");

        String enrollmentDate = request.getParameter("enrollmentDate");


        try (Connection conn = DBConnection.getConnection()) {

            String query = "INSERT INTO Student (first_name, last_name, email,
phone_number, enrollment_number, course, enrollment_date) " +

                    "VALUES (?, ?, ?, ?, ?, ?, ?)";

            try (PreparedStatement stmt = conn.prepareStatement(query)) {

                stmt.setString(1, firstName);

                stmt.setString(2, lastName);
```

```
            stmt.setString(3, email);
            stmt.setString(4, phoneNumber);
            stmt.setString(5, enrollmentNumber);
            stmt.setString(6, course);
            stmt.setString(7, enrollmentDate);
            stmt.executeUpdate();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    response.sendRedirect("listStudents");
  }
}
```

3.3.2. READ STUDENTS (LIST ALL STUDENTS)

Create a servlet called `ListStudentsServlet.java` to display all students:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
import java.util.*;

public class ListStudentsServlet extends HttpServlet {
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        List<Student> students = new ArrayList<>();
        try (Connection conn = DBConnection.getConnection()) {
            String query = "SELECT * FROM Student";
            try (Statement stmt = conn.createStatement();
                 ResultSet rs = stmt.executeQuery(query)) {
                while (rs.next()) {
                    Student student = new Student();
                    student.setStudentId(rs.getInt("student_id"));
                    student.setFirstName(rs.getString("first_name"));
                    student.setLastName(rs.getString("last_name"));
                    student.setEmail(rs.getString("email"));
                    student.setPhoneNumber(rs.getString("phone_number"));
                    student.setEnrollmentNumber(rs.getString("enrollment_number"));
                    student.setCourse(rs.getString("course"));
                    student.setEnrollmentDate(rs.getString("enrollment_date"));
                    students.add(student);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        request.setAttribute("students", students);
```

```
        RequestDispatcher dispatcher =
request.getRequestDispatcher("studentList.jsp");

        dispatcher.forward(request, response);

    }

}
```

3.3.3. UPDATE STUDENT

To update a student's information, create a servlet `UpdateStudentServlet.java`:

```
import javax.servlet.*;

import javax.servlet.http.*;

import java.io.*;

import java.sql.*;


public class UpdateStudentServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        int studentId = Integer.parseInt(request.getParameter("studentId"));

        String firstName = request.getParameter("firstName");

        String lastName = request.getParameter("lastName");

        String email = request.getParameter("email");

        String phoneNumber = request.getParameter("phoneNumber");

        String enrollmentNumber = request.getParameter("enrollmentNumber");

        String course = request.getParameter("course");

        String enrollmentDate = request.getParameter("enrollmentDate");
```

```
        try (Connection conn = DBConnection.getConnection()) {

            String query = "UPDATE Student SET first_name=?, last_name=?,
email=?, phone_number=?, course=?, enrollment_date=? WHERE
student_id=?";

            try (PreparedStatement stmt = conn.prepareStatement(query)) {

                stmt.setString(1, firstName);

                stmt.setString(2, lastName);

                stmt.setString(3, email);

                stmt.setString(4, phoneNumber);

                stmt.setString(5, course);

                stmt.setString(6, enrollmentDate);

                stmt.setInt(7, studentId);

                stmt.executeUpdate();

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }


        response.sendRedirect("listStudents");

    }

}
```

3.3.4. DELETE STUDENT

Create a servlet `DeleteStudentServlet.java` for the delete operation:

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;

import java.io.*;

import java.sql.*;


public class DeleteStudentServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        int studentId = Integer.parseInt(request.getParameter("studentId"));


        try (Connection conn = DBConnection.getConnection()) {

            String query = "DELETE FROM Student WHERE student_id=?";

            try (PreparedStatement stmt = conn.prepareStatement(query)) {

                stmt.setInt(1, studentId);

                stmt.executeUpdate();

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }


        response.sendRedirect("listStudents");

    }

}
```

4. JSP PAGES

Create a JSP page `studentList.jsp` to display the list of students:

```
<%@ page import="java.util.*, com.example.Student" %>
<%@ page contentType="text/html; charset=UTF-8" %>
<html>
<head>
    <title>Student List</title>
</head>
<body>
    <h2>List of Students</h2>
    <table border="1">
        <tr>
            <th>Student ID</th>
            <th>Name</th>
            <th>Email</th>
            <th>Course</th>
            <th>Actions</th>
        </tr>
        <c:forEach var="student" items="${students}">
            <tr>
                <td>${student.studentId}</td>
                <td>${student.firstName} ${student.lastName}</td>
                <td>${student.email}</td>
                <td>${student.course}</td>
                <td>
```

```
                <a
href="updateStudentForm?studentId=${student.studentId}">Edit</a>
                <a
href="deleteStudent?studentId=${student.studentId}">Delete</a>
            </td>
        </tr>
    </c:forEach>
</table>
</body>
</html>
```

4. **Configure `web.xml`**

```
<web-app>
    <servlet>
        <servlet-name>AddStudentServlet</servlet-name>
        <servlet-class>AddStudentServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AddStudentServlet</servlet-name>
        <url-pattern>/addStudent</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>ListStudentsServlet</servlet-name>
        <servlet-class>ListStudentsServlet</servlet-class>
```

```
</servlet>
<servlet-mapping>
    <servlet-name>ListStudentsServlet</servlet-name>
    <url-pattern>/listStudents</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>DeleteStudentServlet</servlet-name>
    <servlet-class>DeleteStudentServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DeleteStudentServlet</servlet-name>
    <url-pattern>/deleteStudent</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>UpdateStudentServlet</servlet-name>
    <servlet-class>UpdateStudentServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>UpdateStudentServlet</servlet-name>
    <url-pattern>/updateStudent</url-pattern>
</servlet-mapping>
```

</web-app>

6. RUN THE APPLICATION

You can now run your application in Eclipse using a servlet container like Tomcat.

- To **add a student**, go to `/addStudent` with the necessary parameters.
- To **list all students**, visit `/listStudents`.
- To **edit or delete** a student, use the links in the table.

Question :

Create a user table into the database and bind the user entity with Spring Security for Login.

STEP 1: CREATE THE DATABASE AND USER TABLE

Create a `users` table in your database. This table will store user credentials and roles.

CREATE DATABASE user_management;

USE user_management;

```
CREATE TABLE users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(20) NOT NULL,
    enabled BOOLEAN DEFAULT TRUE
);
```

STEP 2: SET UP A SPRING BOOT PROJECT IN ECLIPSE

1. Open Eclipse and create a new Spring Boot project.
2. Include dependencies in `pom.xml`:
    - **Spring Web**
    - **Spring Security**
    - **Spring Data JPA**
    - **MySQL Driver**

`pom.xml` dependencies:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
```

```
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
</dependencies>
```

STEP 3: CONFIGURE `APPLICATION.PROPERTIES`

Add the database connection properties in `src/main/resources/application.properties`:

```
spring.datasource.url=jdbc:mysql://localhost:3306/user_management
spring.datasource.username=root
spring.datasource.password=your_password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

STEP 4: CREATE THE USER ENTITY

`User.java`:

```
package com.example.demo.entity;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
```

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int userId;

    private String username;
    private String password;
    private String role;
    private boolean enabled;

    // Getters and Setters
    public int getUserId() {
        return userId;
    }

    public void setUserId(int userId) {
        this.userId = userId;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
```

```
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    public boolean isEnabled() {
        return enabled;
    }
    public void setEnabled(boolean enabled) {
        this.enabled = enabled;
    }}
```

STEP 5: CREATE THE USER REPOSITORY

`UserRepository.java`:

```
package com.example.demo.repository;

import com.example.demo.entity.User;
```

```
import org.springframework.data.jpa.repository.JpaRepository;


public interface UserRepository extends JpaRepository<User, Integer> {
    User findByUsername(String username);
}
```

STEP 6: CONFIGURE SPRING SECURITY

`SecurityConfig.java`:

```
package com.example.demo.config;


import com.example.demo.service.CustomUserDetailsService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;


@Configuration
public class SecurityConfig {


    @Bean
    public UserDetailsService userDetailsService() {
        return new CustomUserDetailsService();
    }


    @Bean
```

```
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/login", "/register").permitAll()
            .anyRequest().authenticated()
            .and()
            .formLogin()
            .loginPage("/login")
            .defaultSuccessUrl("/dashboard", true)
            .permitAll()
            .and()
            .logout()
            .permitAll();
        return http.build();
    }

    @Bean
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService()).passwordEncoder(passwordEncoder());
    }
}
```

Step 7: Implement CustomUserDetailsService

CustomUserDetailsService.java:

```
package com.example.demo.service;


import com.example.demo.entity.User;
import com.example.demo.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.core.userdetails.User.UserBuilder;
import org.springframework.security.core.userdetails.User;


public class CustomUserDetailsService implements UserDetailsService {


    @Autowired
    private UserRepository userRepository;


    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException("User not found");
        }
        UserBuilder builder = org.springframework.security.core.userdetails.User.withUsername(username);
        builder.password(user.getPassword());
        builder.roles(user.getRole());
        builder.disabled(!user.isEnabled());
        return builder.build();
    }
```

```
}
```

STEP 8: CREATE THE LOGIN PAGE

`login.html` (Thymeleaf template):

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
   <title>Login</title>
</head>
<body>
   <h2>Login</h2>
   <form th:action="@{/login}" method="post">
      <label for="username">Username:</label>
      <input type="text" id="username" name="username" required><br><br>
      <label for="password">Password:</label>
      <input type="password" id="password" name="password" required><br><br>
      <button type="submit">Login</button>
   </form>
</body>
</html>
```

TEP 9: RUN YOUR APPLICATION

- Run the Spring Boot application and access it at `http://localhost:8080/login`.

Question : jsp database

Write a program using JSP and JDBC create CRUD (Create/Save, Read, Edit/Update, Delete) application for students attendance management in MCSL-222 counselling classes. in eclipse

Step 1: Set Up the Database

Create a database and table to store student attendance data.

SQL script to create the database and table:

CREATE DATABASE student_management;

USE student_management;

CREATE TABLE student_attendance (id INT PRIMARY KEY AUTO_INCREMENT, student_name VARCHAR(100) NOT NULL, date_of_attendance DATE NOT NULL,status VARCHAR(10) NOT NULL

);

Step 2: Configure the Project in Eclipse

Create a Dynamic Web Project in Eclipse.

Add the MySQL JDBC driver (e.g., mysql-connector-java-X.X.X.jar) to your project's WEB-INF/lib folder.

Set up your project structure:

- `src` (Java code)
- `WebContent` (HTML, JSP files)
- `WEB-INF` (web.xml and libraries)

STEP 3: CREATE THE DATABASE CONNECTION CLASS

`DBConnection.java`:

```
package com.example.utils;



import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;



public class DBConnection {

    private static final String URL = "jdbc:mysql://localhost:3306/student_management";

    private static final String USER = "root";

    private static final String PASSWORD = "your_password";
```

```
    private static final String DRIVER = "com.mysql.cj.jdbc.Driver";


    public static Connection getConnection() throws ClassNotFoundException, SQLException
{
        Class.forName(DRIVER);
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

STEP 4: CREATE A DAO CLASS FOR CRUD OPERATIONS

`AttendanceDAO.java`:

```
package com.example.dao;


import com.example.model.StudentAttendance;
import com.example.utils.DBConnection;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;


public class AttendanceDAO {


    public void saveAttendance(StudentAttendance attendance) throws SQLException,
ClassNotFoundException {
        String query = "INSERT INTO student_attendance (student_name, date_of_attendance,
status) VALUES (?, ?, ?)";
        try (Connection con = DBConnection.getConnection(); PreparedStatement stmt =
con.prepareStatement(query)) {
            stmt.setString(1, attendance.getStudentName());
```

```
            stmt.setDate(2, Date.valueOf(attendance.getDateOfAttendance()));
            stmt.setString(3, attendance.getStatus());
            stmt.executeUpdate();
        }
    }

    public List<StudentAttendance> getAllAttendances() throws SQLException,
ClassNotFoundException {
        List<StudentAttendance> attendances = new ArrayList<>();
        String query = "SELECT * FROM student_attendance";
        try (Connection con = DBConnection.getConnection(); PreparedStatement stmt =
con.prepareStatement(query); ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                StudentAttendance attendance = new StudentAttendance(
                    rs.getInt("id"),
                    rs.getString("student_name"),
                    rs.getDate("date_of_attendance").toLocalDate(),
                    rs.getString("status")
                );
                attendances.add(attendance);
            }
        }
        return attendances;
    }

    public StudentAttendance getAttendanceById(int id) throws SQLException,
ClassNotFoundException {
```

```
        String query = "SELECT * FROM student_attendance WHERE id = ?";

        try (Connection con = DBConnection.getConnection(); PreparedStatement stmt =
con.prepareStatement(query)) {

            stmt.setInt(1, id);

            try (ResultSet rs = stmt.executeQuery()) {

                if (rs.next()) {

                    return new StudentAttendance(

                        rs.getInt("id"),

                        rs.getString("student_name"),

                        rs.getDate("date_of_attendance").toLocalDate(),

                        rs.getString("status")

                    );

                }

            }

        }

        return null;

    }



    public void updateAttendance(StudentAttendance attendance) throws SQLException,
ClassNotFoundException {

        String query = "UPDATE student_attendance SET student_name = ?,
date_of_attendance = ?, status = ? WHERE id = ?";

        try (Connection con = DBConnection.getConnection(); PreparedStatement stmt =
con.prepareStatement(query)) {

            stmt.setString(1, attendance.getStudentName());

            stmt.setDate(2, Date.valueOf(attendance.getDateOfAttendance()));

            stmt.setString(3, attendance.getStatus());
```

```
            stmt.setInt(4, attendance.getId());
            stmt.executeUpdate();
        }
    }

    public void deleteAttendance(int id) throws SQLException, ClassNotFoundException {
        String query = "DELETE FROM student_attendance WHERE id = ?";
        try (Connection con = DBConnection.getConnection(); PreparedStatement stmt =
con.prepareStatement(query)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        }
    }
}
```

STEP 5: CREATE THE MODEL CLASS

`StudentAttendance.java`:

```
package com.example.model;

import java.time.LocalDate;

public class StudentAttendance {
    private int id;
    private String studentName;
    private LocalDate dateOfAttendance;
    private String status;
```

```
    public StudentAttendance() {}

    public StudentAttendance(int id, String studentName, LocalDate dateOfAttendance, String
status) {
        this.id = id;
        this.studentName = studentName;
        this.dateOfAttendance = dateOfAttendance;
        this.status = status;
    }

    // Getters and Setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getStudentName() {
        return studentName;
    }

    public void setStudentName(String studentName) {
        this.studentName = studentName;
```

```
    }

    public LocalDate getDateOfAttendance() {
        return dateOfAttendance;
    }

    public void setDateOfAttendance(LocalDate dateOfAttendance) {
        this.dateOfAttendance = dateOfAttendance;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }
}
```

STEP 6: CREATE JSP PAGES

`addAttendance.jsp`:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>
```

```
    <title>Add Attendance</title>
</head>
<body>
    <h2>Add Student Attendance</h2>
    <form action="saveAttendance" method="post">
        <label for="studentName">Student Name:</label>
        <input type="text" id="studentName" name="studentName" required><br><br>
        <label for="dateOfAttendance">Date:</label>
        <input type="date" id="dateOfAttendance" name="dateOfAttendance"
required><br><br>
        <label for="status">Status:</label>
        <select id="status" name="status">
            <option value="Present">Present</option>
            <option value="Absent">Absent</option>
        </select><br><br>
        <button type="submit">Save</button>
    </form>
</body>
</html>
```

`listAttendance.jsp`:

```
<%@ page language="java" import="java.util.*, com.example.dao.AttendanceDAO,
com.example.model.StudentAttendance" %>
<%
    AttendanceDAO dao = new AttendanceDAO();
    List<StudentAttendance> attendanceList = dao.getAllAttendances();
%>
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Attendance List</title>
</head>
<body>
    <h2>Student Attendance List</h2>
    <table border="1">
        <tr>
            <th>ID</th>
            <th>Student Name</th>
            <th>Date of Attendance</th>
            <th>Status</th>
            <th>Actions</th>
        </tr>
        <%
            for (StudentAttendance att : attendanceList) {
        %>
        <tr>
            <td><%= att.getId() %></td>
            <td><%= att.getStudentName() %></td>
            <td><%= att.getDateOfAttendance() %></td>
            <td><%= att.getStatus() %></td>
            <td>
                <a href="editAttendance.jsp?id=<%= att.getId() %>">Edit</a>
```

```
            <a href="deleteAttendance?id=<%= att.getId() %>">Delete</a>
        </td>
    </tr>
    <%
        }
    %>
  </table>
</body>
</html>
```

STEP 7: CREATE SERVLET FOR CRUD OPERATIONS

Create servlets for handling CRUD operations like saving, updating, and deleting attendance records.

`SaveAttendanceServlet.java`:

```
@WebServlet("/saveAttendance")
public class SaveAttendanceServlet extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String studentName = request.getParameter("studentName");
        LocalDate dateOfAttendance = LocalDate.parse(request.getParameter("dateOfAttendance"));
        String status = request.getParameter("status");

        StudentAttendance attendance = new StudentAttendance();
        attendance.setStudentName(studentName);
        attendance.setDateOfAttendance(dateOfAttendance);
        attendance.setStatus(status);
```

```
try {
    AttendanceDAO dao = new AttendanceDAO();
    dao.saveAttendance(attendance);
    response.sendRedirect("listAttendance.jsp");
} catch (Exception e) {
    e.printStackTrace();
}   }}
```

STEP 8: RUN THE APPLICATION

- Deploy the project on a server like Tomcat