Notes: -

1. What is JSP

- A JSP page is a text document that contains two types of text: static data, which can be expressed in any text-based format (such as HTML, SVG, WML, and XML), and JSP elements, which construct dynamic content.
- The recommended file extension for the source file of a JSP page is .jsp
- It stands for Java Server Pages.
- It is a server side technology.
- It is used for creating web application.
- It is used to create dynamic web content.
- In this JSP tags are used to insert JAVA code into HTML/XML pages or both.
- It is an advanced version of Servlet Technology.
- It is a Web based technology helps us to create dynamic and platform independent web pages.
- JSP is first converted into servlet by JSP container before processing the client's request.

2. Features of JSP

- **Dynamic Content**: JSP allows for the creation of dynamic, server-side web pages using Java code embedded in HTML.
- **Platform Independent**: Runs on any platform with a compatible servlet container, like Apache Tomcat.
- **Separation of Concerns**: Helps separate business logic from presentation by supporting MVC (Model-View-Controller) design patterns.
- **Reusable Components**: Supports the use of JavaBeans and custom tags, promoting reusability and modular design.
- **Session Management**: Built-in support for managing sessions, making it easy to maintain user states across pages.
- **Tag Libraries**: Provides a rich set of tag libraries, such as JSTL (JSP Standard Tag Library), for easier development.

3. Advantages and Disadvantages of JSP

Advantages of JSP

- **Simplified Coding**: Allows embedding Java code directly into HTML, making it easier to create dynamic web content.
- **Separation of Concerns**: Encourages a separation between business logic and presentation, supporting MVC architecture.
- **Platform Independence**: JSP applications are platform-independent and can run on any server with a compatible servlet container.
- **Reusable Components**: Supports JavaBeans, custom tags, and JSTL, allowing code reuse and simplifying complex tasks.

Disadvantages of JSP

- **Harder Debugging**: Debugging JSP pages can be challenging, especially with extensive embedded Java code.
- **Performance Overhead**: Initial page load may be slower due to compilation, especially for larger JSP files.
- **Limited Tooling Support**: Although improved, JSP has less robust tooling support compared to newer technologies like React or Angular.
- **Less Readable Code**: Embedding Java code in HTML can make pages harder to read and maintain, especially for complex applications.

4. **Difference**

Servlets	JSP
Servlet is a java code.	JSP is a html based code.
Servlet is faster than JSP.	JSP is slower than Servlet because the first step in JSP lifecycle is the translation of JSP to java code and then compile.
In Servlet, we can override the service() method.	In JSP, we cannot override its service() method.
In Servlet by default session management is not enabled, user have to enable it explicitly.	In JSP session management is automatically enabled.
Modification in Servlet is a time consuming task because it includes reloading, recompiling and restarting the server.	JSP modification is fast, just need to click the refresh button.

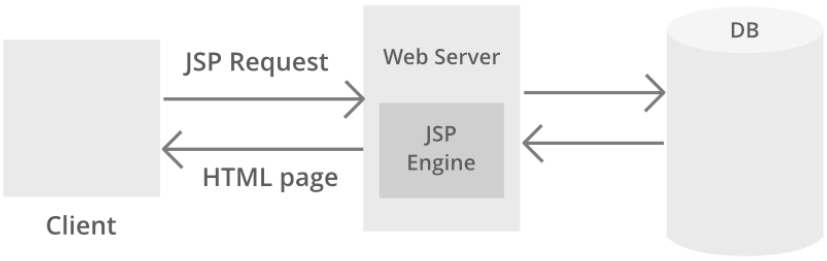
5. **Architecture**

- **Client Request**: A user requests a JSP page by entering a URL in the browser.

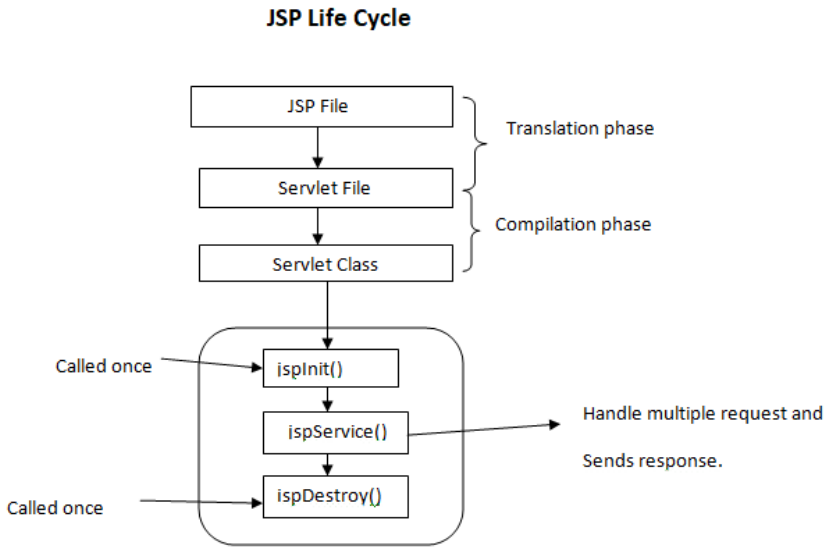
 Web Server: The request reaches the web server hosting the JSP container (like Tomcat).
 JSP Engine: The web server passes the request to the JSP engine, which handles JSP processing.
 Translation Phase: The JSP engine translates the JSP page into a servlet by converting JSP tags and Java code.
 Compilation Phase: The translated servlet code is compiled into a Java bytecode .class file.
 Class Loading: The compiled servlet class is loaded into memory by the classloader.
 Initialization: The servlet's init() method is called once, initializing the servlet.
 Request Handling: The servlet's service() method is called to process the client request.
 Response Generation: The servlet generates an HTML response, combining dynamic data with static HTML.
 Response Sent: The generated HTML is sent back to the client's browser.

- **Caching (Optional)**: The compiled servlet is cached for future requests, improving performance.

 Subsequent Requests: For repeated requests, only the service() method runs, bypassing translation and compilation steps, until the JSP page is modified.



6. JSP Life-Cycle



a.

- The JSP (JavaServer Pages) life cycle involves the various stages a JSP page goes through from creation to destruction. Here's an overview of each stage:
- **Translation**:
  - The JSP file is translated into a Java servlet by the JSP engine.
  - This step occurs only once, unless the JSP file is modified.
- **Compilation**:
  - The translated servlet code is compiled into a .class file.
  - This .class file represents the servlet that the server will execute.
- **Class Loading**:
  - The compiled servlet class is loaded into memory by the classloader.
  - This allows the JSP page to be available for processing requests.
- **Instantiation**:
  - An instance of the servlet class is created.
  - This instance serves as the JSP page's runtime object.
- **Initialization (jspInit() method)**:
  - The jspInit() method is called once when the JSP is first loaded.
  - It is used for resource allocation or other one-time setup tasks.
- **Request Processing (_jspService() method)**:
  - For each client request, the JSP container calls the _jspService() method.
  - This method contains the main logic for handling requests and generating dynamic content.
  - HTML content and Java code are combined to create the response, which is sent back to the client.
- **Destruction (jspDestroy() method)**:
  - When the JSP is no longer needed, the jspDestroy() method is called.
  - This method is used to release resources, close connections, or perform other cleanup tasks.

7. Components Of JSP

1. Directives

Directives provide instructions to the JSP engine regarding page processing. There are three main types: page, include, and taglib.

- **Syntax**: <%@ directive attribute="value" %>
- **Types**:
  - page: Defines attributes like language and error handling.
  - include: Includes content from other files.
  - taglib: Declares a library of custom tags.
- **Example (Page Directive)**:

  <%@ page contentType="text/html" language="java" %>

- **Page Directive Example**:

  <%@ page language="java" contentType="text/html" errorPage="error.jsp" %>

---

2. Comments

Comments in JSP can either be HTML or JSP comments. JSP comments are not visible in the HTML source code.

- **Syntax**:
  - HTML Comment: <!-- HTML Comment -->
  - JSP Comment: <%-- JSP Comment --%>
- **Purpose**:
  - HTML comments are sent to the client.
  - JSP comments are only for developers and not sent to the client.
- **Example (JSP Comment)**:

  <%-- This is a JSP comment and won't appear in the HTML output --%>

---

3. Expression

Expressions in JSP are used to output dynamic values to the client. The value of the expression is automatically added to the output.

- **Syntax**: <%= expression %>
- **Automatic Output**: Evaluated and added to the response directly.
- **No Semicolon Required**: Just write the expression.
- **Example**:

  Current time: <%= new java.util.Date() %>

  This outputs the current date and time to the page.

---

4. Scriptlets

Scriptlets contain Java code embedded within a JSP page. The code is executed during the request processing phase.

- **Syntax**: <% code %>
- **Can Contain Any Java Code**: Limited only by the servlet container.
- **Directly Inserted**: Code is inserted directly into the servlet's _jspService() method.
- **Example**:

  ```
  <%
    String message = "Hello, World!";
    out.println(message);
  %>
  ```

  This outputs "Hello, World!" to the page.

---

5. Declarations

Declarations are used to declare variables or methods that can be reused within the JSP page.

- **Syntax**: <%! declaration %>
- **Scope**: Variables or methods are accessible throughout the JSP page.
- **Placement**: Placed outside the _jspService() method.
- **Example**:

  ```
  <%! int counter = 0; %>
  ```

  This declaration creates a counter variable with a scope across the entire JSP page.

6. Implicit Objects

- ▶ These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared.
- ▶ JSP Implicit Objects are also called pre-defined variables.
- ▶ JSP supports 9 implicit objects.
- ▶ request
  - ▶ This is the HttpServletRequest object associated with the request
- ▶ response
  - ▶ This is the HttpServletResponse object associated with the response to the client.
- ▶ out
  - ▶ This is the PrintWriter object used to send output to the client.
- ▶ session
  - ▶ This is the HttpSession object associated with the request.

- **config**
  - This is the **ServletConfig** object associated with the page.
- application
  - This is the ServletContext object associated with the application context.
- pageContext
  - This encapsulates use of server-specific features like higher performance JspWriters.
- page
  - This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.
- Exception
  - The Exception object allows the exception data to be accessed by designated JSP.

(See example from ppt)

7. Action Tags

- The action tags are used to control the flow between pages and to use Java Bean
- We can dynamically insert a file, reuse the beans components, forward user to another page, etc. through JSP Actions like include and forward.
- Unlike directives, actions are re-evaluated each time the page is accessed.

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

(See example from ppt)

8. Tag Extension

- JSP supports the authors to add their own custom tags that perform custom actions.
- This is performed by using JSP tag extension API.
- A java class is written by developers which implements the tag interface and provides a tag library XML description file.

- This file specifies the tags and java classes that are used to implement the tags.
- The JSP tag extensions create new tags that can be inserted into the JavaServer Page directly.
- To write a code just extend SimpleTagSupport class and override the doTag() method, where the code can be placed to generate content for the tag.
- Consider you want to define a custom tag named <ex:Hello> and you want to use it in the following fashion without a body
- <ex:Hello>
- To create a custom JSP tag, you must first create a Java class that acts as a tag handler. Let us now create the HelloTag class as follows

(See example from ppt)

Elements Created with the <input> Tag

The <input> tag is used to create various types of form elements for user input. Its behavior and appearance depend on the type attribute.

1. **Different Input Types**: The type attribute defines the type of input, such as text, password, checkbox, radio, submit, etc.
2. **Text Input**: <input type="text"> creates a single-line text field for user input.
3. **Password Field**: <input type="password"> creates a text field where characters are obscured.
4. **Checkboxes and Radios**: <input type="checkbox"> and <input type="radio"> create checkboxes and radio buttons, respectively.
5. **Submit Button**: <input type="submit"> creates a button to submit the form data.
6. **File Upload**: <input type="file"> allows users to select and upload files from their device.
7. **Default Value**: The value attribute sets a default value, which can be displayed in fields like text or hidden.
8. **Attributes for Validation**: Attributes like required, minlength, and maxlength add client-side validation.

- **Example**:

  ```
  <input type="text" name="username" placeholder="Enter your username">
  <input type="password" name="password" placeholder="Enter your password">
  <input type="submit" value="Submit">
  ```

---

2. Elements Created with <select> and <option> Tags

The <select> and <option> tags create a drop-down list, allowing users to select from a list of predefined options.

1. **Dropdown Menu**: The <select> tag defines a dropdown menu for selecting one (or multiple) options.
2. **Options Defined by <option>**: Each <option> tag inside a <select> represents an individual choice.
3. **Preselecting Options**: Adding selected to an <option> makes it the default choice when the page loads.
4. **Multi-Selection Support**: Adding the multiple attribute to <select> allows users to select multiple options.
5. **Naming and Data Submission**: The name attribute in <select> allows it to be submitted with form data.
6. **Disable Options**: <option disabled> prevents specific options from being selectable.
7. **Group Options**: <optgroup> groups related options, improving readability.
8. **Value Attribute**: The value attribute in <option> specifies the data submitted when that option is selected.

- **Example**:

```
<select name="country">
   <option value="us">United States</option>
   <option value="ca">Canada</option>
   <option value="uk" selected>United Kingdom</option>
</select>
```

---

3. <textarea> Element

The <textarea> element is used for multi-line text input, suitable for longer entries.

1. **Multi-Line Input**: <textarea> allows users to input multiple lines of text, unlike <input type="text">.
2. **Rows and Columns**: The rows and cols attributes specify the initial visible height and width of the text area.
3. **Default Text**: Any content between the opening and closing <textarea> tags appears as default text.
4. **Resizeable**: By default, most browsers allow users to resize the <textarea> manually.
5. **Placeholder Support**: The placeholder attribute provides a hint to the user about the expected input.
6. **Form Submission**: Text in <textarea> is submitted with the form, using the name defined in the name attribute.
7. **Client-Side Validation**: Attributes like maxlength and required provide client-side validation.
8. **Styling**: The size, font, and other style aspects can be customized using CSS.

- **Example**:

```
<textarea name="comments" rows="4" cols="50" placeholder="Enter your comments here..."></textarea>
```

JSP Form Validation

Form validation in JSP ensures that the data submitted by a user is correct, complete, and secure. While client-side validation (using JavaScript) provides a better user experience, server-side validation using JSP ensures that data integrity is maintained even if client-side validation is bypassed.

JSP form validation involves two main steps:

1. **Client-Side Validation**: Performed using JavaScript for a better user experience.
2. **Server-Side Validation**: Performed on the server using JSP to ensure data integrity before processing.

Below is an example of both client-side and server-side validation in JSP.

Example: JSP Form Validation

1. HTML Form (Client-Side Validation using JavaScript)

The form contains basic input fields (like name, email, and age), and JavaScript is used to validate the form before submission.

```
<!DOCTYPE html>
<html>
<head>
   <title>JSP Form Validation Example</title>
   <script>
      // Client-side validation using JavaScript
      function validateForm() {
         var name = document.forms["myForm"]["name"].value;
         var email = document.forms["myForm"]["email"].value;
         var age = document.forms["myForm"]["age"].value;

         if (name == "") {
            alert("Name must be filled out");
            return false;
         }
         if (email == "") {
            alert("Email must be filled out");
            return false;
         }
```

```
        if (age == "" || isNaN(age) || age < 18) {
            alert("Please enter a valid age (18 or older)");
            return false;
        }
        return true;
    }
  </script>
</head>
<body>

  <h2>Form Validation in JSP</h2>
  <form name="myForm" action="processForm.jsp" onsubmit="return validateForm()"
method="post">
    Name: <input type="text" name="name"><br><br>
    Email: <input type="text" name="email"><br><br>
    Age: <input type="text" name="age"><br><br>
    <input type="submit" value="Submit">
  </form>

</body>
</html>
```

- **Explanation**:
  - The form includes fields for name, email, and age.
  - The onsubmit="return validateForm()" attribute triggers the JavaScript function validateForm() before the form is submitted.
  - If validation fails (empty or invalid fields), an alert message is shown and the form is not submitted.

---

2. JSP Processing Page (Server-Side Validation)

The server-side validation ensures that even if the client bypasses JavaScript validation, the form data is properly validated before processing.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%!
  // Server-side validation function
  public boolean isValidForm(String name, String email, String age) {
    if (name == null || name.trim().isEmpty()) {
      return false;
    }
    if (email == null || email.trim().isEmpty()) {
      return false;
    }
    try {
      int parsedAge = Integer.parseInt(age);
```

```
            if (parsedAge < 18) {
                return false;
            }
        } catch (NumberFormatException e) {
            return false;
        }
        return true;
    }
%>
<%
    String name = request.getParameter("name");
    String email = request.getParameter("email");
    String age = request.getParameter("age");

    boolean isValid = isValidForm(name, email, age);

    if (isValid) {
%>
        <h2>Form Submitted Successfully</h2>
        <p>Name: <%= name %></p>
        <p>Email: <%= email %></p>
        <p>Age: <%= age %></p>
<%
    } else {
%>
        <h2>Error: Invalid Input</h2>
        <p>Please check the details and try again.</p>
        <a href="form.html">Go back to form</a>
<%
    }
%>
```

- **Explanation**:
  - The JSP page processes the form data submitted by the user.
  - It uses the getParameter() method to retrieve form values for name, email, and age.
  - The isValidForm() method is called to validate the form data:
    - It checks if name and email are non-empty.
    - It parses age and ensures that it is a number and that the user is 18 or older.
  - If the form data is valid, a success message is displayed. Otherwise, an error message prompts the user to check their input.

Key Points

1. **Client-Side Validation (JavaScript)**: Provides immediate feedback to users but can be bypassed.
2. **Server-Side Validation (JSP)**: Ensures data is checked securely before processing and submission, even if JavaScript is bypassed.
3. **Form Handling**: Data is validated first on the client side and then validated again on the server side.
4. **Security**: Server-side validation is crucial for security, as client-side validation can be manipulated.

By combining both client-side and server-side validation, you ensure that form data is validated efficiently and securely.

Cookies

- Cookies are text files stored on the client computer and they are kept for various information tracking purposes.
- JSP transparently supports HTTP cookies using underlying servlet technology.
- There are three steps involved in identifying and returning users:
    - Server script sends a set of cookies to the browser. For example, name, age, or identification number, etc.
    - Browser stores this information on the local machine for future use.
    - When the next time the browser sends any request to the web server then it sends those cookies information to the server and server uses that information to identify the user or may be for some other purpose as well.

1. Setting a Cookie (Storing Username)

In this example, we create a form where a user enters their username. Upon submitting the form, we store the username in a cookie.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="javax.servlet.http.Cookie" %>
<!DOCTYPE html>
<html>
<head>
   <title>Set Username Cookie</title>
</head>
<body>

   <h2>Enter your Username</h2>
   <form action="setCookie.jsp" method="post">
      Username: <input type="text" name="username" required><br><br>
      <input type="submit" value="Submit">
   </form>

   <%
```

```
        // Check if the username is provided and set the cookie
        String username = request.getParameter("username");
        if (username != null && !username.isEmpty()) {
            // Create a new cookie
            Cookie usernameCookie = new Cookie("username", username);
            usernameCookie.setMaxAge(60 * 60 * 24); // Set cookie to expire in 1 day (60 sec *
60 min * 24 hrs)
            response.addCookie(usernameCookie); // Add the cookie to the response
            out.println("<p>Cookie has been set with your username: " + username + "</p>");
        }
    %>

</body>
</html>
```

- **Explanation**:
  - When the form is submitted, the username value is retrieved from the request.
  - A new Cookie object is created with the name username and the user's entered value.
  - The cookie's expiration time is set to 1 day (setMaxAge(60 * 60 * 24)).
  - The cookie is then added to the response using response.addCookie(), and a success message is shown.

---

2. Deleting a Cookie

To delete a cookie, we set its max age to 0, which tells the browser to immediately expire the cookie.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="javax.servlet.http.Cookie" %>
<!DOCTYPE html>
<html>
<head>
    <title>Delete Username Cookie</title>
</head>
<body>

    <h2>Delete Username Cookie</h2>
    <form action="deleteCookie.jsp" method="post">
        <input type="submit" value="Delete Cookie">
    </form>

    <%
        // Deleting the username cookie
        Cookie[] cookies = request.getCookies();
```

```
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if ("username".equals(cookie.getName())) {
                    // Set the cookie's max age to 0 to delete it
                    cookie.setMaxAge(0);
                    response.addCookie(cookie); // Add the updated cookie to the response
                    out.println("<p>Username cookie has been deleted.</p>");
                }
            }
        }
    %>

</body>
</html>
```

- **Explanation**:
  - We retrieve all cookies using request.getCookies().
  - If a cookie named username exists, its maxAge is set to 0, effectively deleting it.
  - The updated cookie (with max age 0) is added back to the response, instructing the browser to delete it.

Sessions

Maintaining Sessions and Tracking Session ID in JSP

In JSP, sessions are used to store user-specific information across multiple requests. The HttpSession object is used to maintain the session, and each session is uniquely identified by a **session ID**. This session ID is automatically generated by the server and can be tracked either through cookies or URL rewriting.

---

1. Creating a Session in JSP

When a user accesses a JSP page, the server creates a session and assigns a unique **session ID**. You can use the HttpSession object to store and retrieve data within the session.

Example: Creating a Session and Storing Data

jsp
Copy code

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="javax.servlet.http.HttpSession" %>
<!DOCTYPE html>
```

```
<html>
<head>
   <title>Session Example - Store Data</title>
</head>
<body>

   <h2>Welcome to the Session Tracking Page</h2>

   <form action="storeSession.jsp" method="post">
      Name: <input type="text" name="username" required><br><br>
      <input type="submit" value="Store in Session">
   </form>

   <%
      // Store session data if the form is submitted
      String username = request.getParameter("username");
      if (username != null && !username.isEmpty()) {
         HttpSession session = request.getSession(); // Get the current session
         session.setAttribute("username", username); // Store the username in the session
         out.println("<p>Session created and username stored: " + username + "</p>");
      }
   %>

</body>
</html>
```

- **Explanation**:
  - When the user submits the form, the username is retrieved and stored in the session using session.setAttribute("username", username).
  - The session ID is automatically generated and assigned to the session when request.getSession() is called.

---

2. Retrieving and Tracking the Session ID

You can access the session ID and use it to track the user's session across multiple requests. The session ID is automatically passed between the client and server using cookies (by default) or URL rewriting.

Example: Retrieving Session ID and Session Data

jsp
Copy code

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="javax.servlet.http.HttpSession" %>
<!DOCTYPE html>
<html>
```

```
<head>
   <title>Session Example - Retrieve Data</title>
</head>
<body>

   <h2>Retrieve Session Data</h2>

   <%
      HttpSession session = request.getSession(false); // Get the current session, don't create a new one if it doesn't exist
      if (session != null) {
         // Retrieve the session ID
         String sessionID = session.getId();
         // Retrieve the username from session
         String username = (String) session.getAttribute("username");

         if (username != null) {
            out.println("<p>Welcome back, " + username + "!</p>");
            out.println("<p>Your session ID is: " + sessionID + "</p>");
         } else {
            out.println("<p>No session data found!</p>");
         }
      } else {
         out.println("<p>No active session. Please log in first.</p>");
      }
   %>

</body>
</html>
```

- **Explanation**:
  - The request.getSession(false) method checks for an existing session without creating a new one if it doesn't exist.
  - If the session exists, we retrieve the **session ID** using session.getId().
  - We also retrieve the **username** stored in the session using session.getAttribute("username").
  - If there is no active session, an appropriate message is shown.

---

3. Tracking the Session ID

The session ID is usually stored in a cookie by default, and it's sent to the server with each request. You can check the session ID by accessing the session.getId() method, which will return the unique session ID for the user.

- **Session ID via Cookies**:

    - The session ID is stored as a cookie named JSESSIONID in the browser, and this cookie is sent to the server with each request to associate the request with the correct session.
- **Session ID via URL Rewriting**:
    - If cookies are disabled, the session ID can be passed as part of the URL using URL rewriting. You can get the session ID using response.encodeURL("yourJSPpage.jsp"), which appends the session ID to the URL.

---

4. Deleting a Session

You can invalidate the session and delete the stored session data using the session.invalidate() method.

Example: Deleting the Session

jsp
Copy code

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="javax.servlet.http.HttpSession" %>
<!DOCTYPE html>
<html>
<head>
    <title>Session Example - Delete Data</title>
</head>
<body>

    <h2>Delete Session Data</h2>

    <form action="deleteSession.jsp" method="post">
        <input type="submit" value="Delete Session Data">
    </form>

    <%
        // Delete the session and its data
        HttpSession session = request.getSession(false); // Get the session, don't create a new
one
        if (session != null) {
            session.invalidate(); // Invalidate the session
            out.println("<p>Session has been deleted.</p>");
        } else {
            out.println("<p>No active session to delete.</p>");
        }
    %>

</body>
```

</html>

- **Explanation**:
  - session.invalidate() removes the session and all its data.
  - If there's no session, a message indicating that no session exists is displayed.

Key Points:

1. **Session Creation**: Use request.getSession() to create a session and store user data.
2. **Session ID**: The server automatically generates a unique session ID to track the user.
3. **Tracking Session**: You can retrieve the session ID with session.getId() to track the session.
4. **Session Deletion**: Use session.invalidate() to delete a session and remove stored data.
5. **Session Management**: Sessions are usually tracked using cookies or URL rewriting if cookies are disabled.

JSTL

- The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.
- There are many tags in JSTL library. Few of them are
  - Core tags
  - SQL tags
  - XML tags
  - JSTL Functions

Core Tags

- The core group of tags are the most commonly used JSTL tags.
- Following is the syntax to include the JSTL Core library in your JSP
- <%@ taglib prefix = "c" uri = " https://www.oracle.com/java/technologies/?er=221886 " %>
- A few core tags

Tag	Description
<c:out>	To write something in JSP page
<c:import>	Same as jsp:include or include directive
<c:redirect>	redirect request to another resource
<c:catch>	To catch the exception and wrap it into an object.
<c:if>	Simple conditional logic, used with EL and we can use it to process the exception from <c:catch>
<c:forEach>	For iteration over a collection

SQL Tags

- The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as Oracle, mySQL, or Microsoft SQL Server.
- Following is the syntax to include JSTL SQL library in your JSP
    - <%@ taglib prefix = "sql" uri = "http://java.sun.com/jsp/jstl/sql" %>

Tag	Desciption
<sql:setDataSource>	Creates a simple DataSource suitable only for prototyping
<sql:query>	Executes the SQL query defined in its body or through the sql attribute.
<sql:update>	Executes the SQL update defined in its body or through the sql attribute
<sql:param>	Sets a parameter in an SQL statement to the specified value

XML Tags

- The JSTL XML tags provide a JSP-centric way of creating and manipulating the XML documents.
- Following is the syntax to include the JSTL XML library in your JSP.
- <%@ taglib prefix = "x"
- uri = "http://java.sun.com/jsp/jstl/xml" %>

Tag	Description
<x:out>	Like <%= ... >, but for XPath expressions.
<x:parse>	Used to parse the XML data specified either via an attribute or in the tag body
<x:if >	Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored
<x:forEach>	To loop over nodes in an XML document

JSTL Functions

- JSTL includes a number of standard functions, most of which are common string manipulation functions.
- Following is the syntax to include JSTL Functions library in your JSP
- <%@ taglib prefix = "fn"
- uri = "http://java.sun.com/jsp/jstl/functions" %>

Tag	Description
fn:contains()	Tests if an input string contains the specified substring
fn:containsIgnoreCase()	Tests if an input string contains the specified substring in a case insensitive way
fn:join()	Joins all elements of an array into a string
fn:length()	Returns the number of items in a collection, or the number of characters in a string
fn:toLowerCase()	Converts all of the characters of a string to lower case
fn:toUpperCase()	Converts all of the characters of a string to upper case
fn:trim()	Removes white spaces from both ends of a string

JSP: Creating Table and Performing SQL Operations (SELECT, INSERT, DELETE, and UPDATE)

In this example, we'll demonstrate how to create a table in a database and perform basic SQL operations (SELECT, INSERT, DELETE, and UPDATE) using JSP. We'll use JDBC (Java Database Connectivity) to connect to the database, perform SQL operations, and display the results.

---

1. Creating the Table

Assuming you're using a MySQL database, here’s how you can create a simple table:

sql
Copy code

```
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    email VARCHAR(100),
    age INT
);
```

---

2. JSP Page for SELECT, INSERT, DELETE, and UPDATE Operations

Below is the structure of JSP pages that demonstrate how to use JDBC in JSP to connect to a database and perform the various operations.

JSP Page for Selecting Data (SELECT)

This JSP page will retrieve and display the data from the users table

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="java.sql.*, javax.naming.*, javax.sql.*" %>
<!DOCTYPE html>
<html>
<head>
    <title>View Users</title>
</head>
<body>
    <h2>Users List</h2>

    <table border="1">
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Email</th>
            <th>Age</th>
            <th>Action</th>
        </tr>

        <%
            Connection conn = null;
```

```
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // Establishing the connection
            Class.forName("com.mysql.cj.jdbc.Driver");
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/yourdb",
"username", "password");

            // SQL Query to select all users
            String sql = "SELECT * FROM users";
            stmt = conn.createStatement();
            rs = stmt.executeQuery(sql);

            // Displaying data
            while (rs.next()) {
                out.println("<tr>");
                out.println("<td>" + rs.getInt("id") + "</td>");
                out.println("<td>" + rs.getString("name") + "</td>");
                out.println("<td>" + rs.getString("email") + "</td>");
                out.println("<td>" + rs.getInt("age") + "</td>");
                out.println("<td><a href='deleteUser.jsp?id=" + rs.getInt("id") + "'>Delete</a> |
"
                        + "<a href='updateUser.jsp?id=" + rs.getInt("id") + "'>Update</a></td>");
                out.println("</tr>");
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // Closing resources
            if (rs != null) rs.close();
            if (stmt != null) stmt.close();
            if (conn != null) conn.close();
        }
    %>
  </table>
</body>
</html>
```

- **Explanation**: This page establishes a connection to the MySQL database, executes a SELECT query to retrieve all rows from the users table, and displays the results in an HTML table.

JSP Page for Inserting Data (INSERT)

This page allows a user to insert a new record into the users table.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="java.sql.*, javax.naming.*, javax.sql.*" %>
<!DOCTYPE html>
<html>
<head>
   <title>Insert User</title>
</head>
<body>
   <h2>Insert New User</h2>

   <form action="insertUser.jsp" method="post">
      Name: <input type="text" name="name" required><br><br>
      Email: <input type="email" name="email" required><br><br>
      Age: <input type="number" name="age" required><br><br>
      <input type="submit" value="Insert">
   </form>

   <%
      String name = request.getParameter("name");
      String email = request.getParameter("email");
      String age = request.getParameter("age");

      if (name != null && email != null && age != null) {
         try {
            // Establishing the connection
            Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/yourdb", "username",
"password");

            // SQL Query for insertion
            String sql = "INSERT INTO users (name, email, age) VALUES (?, ?, ?)";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, name);
            pstmt.setString(2, email);
            pstmt.setInt(3, Integer.parseInt(age));

            // Executing the insert query
            pstmt.executeUpdate();
            out.println("<p>User inserted successfully!</p>");
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
   %>
</body>
</html>
```

- **Explanation**: This form takes user input (name, email, age) and submits it to insertUser.jsp. The INSERT INTO SQL statement is used to add the new record to the users table.

JSP Page for Deleting Data (DELETE)

This page handles the deletion of a user record.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="java.sql.*, javax.naming.*, javax.sql.*" %>
<!DOCTYPE html>
<html>
<head>
    <title>Delete User</title>
</head>
<body>

    <%
        String userId = request.getParameter("id");
        if (userId != null) {
            try {
                // Establishing the connection
                Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/yourdb", "username",
"password");

                // SQL Query for deletion
                String sql = "DELETE FROM users WHERE id = ?";
                PreparedStatement pstmt = conn.prepareStatement(sql);
                pstmt.setInt(1, Integer.parseInt(userId));

                // Executing the delete query
                int rowsAffected = pstmt.executeUpdate();
                if (rowsAffected > 0) {
                    out.println("<p>User deleted successfully!</p>");
                } else {
                    out.println("<p>User not found!</p>");
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    %>

</body>
```

</html>

- **Explanation**: This JSP page retrieves the user ID from the URL parameter, executes the DELETE SQL statement to remove the user, and then confirms the deletion.

JSP Page for Updating Data (UPDATE)

This page allows updating the details of a user.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="java.sql.*, javax.naming.*, javax.sql.*" %>
<!DOCTYPE html>
<html>
<head>
    <title>Update User</title>
</head>
<body>

    <%
        String userId = request.getParameter("id");
        String name = request.getParameter("name");
        String email = request.getParameter("email");
        String age = request.getParameter("age");

        if (userId != null && name != null && email != null && age != null) {
            try {
                // Establishing the connection
                Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/yourdb", "username",
"password");

                // SQL Query for update
                String sql = "UPDATE users SET name = ?, email = ?, age = ? WHERE id = ?";
                PreparedStatement pstmt = conn.prepareStatement(sql);
                pstmt.setString(1, name);
                pstmt.setString(2, email);
                pstmt.setInt(3, Integer.parseInt(age));
                pstmt.setInt(4, Integer.parseInt(userId));

                // Executing the update query
                int rowsAffected = pstmt.executeUpdate();
                if (rowsAffected > 0) {
                    out.println("<p>User updated successfully!</p>");
                } else {
                    out.println("<p>User not found!</p>");
                }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Fetch user data for pre-population in form
    if (userId != null) {
        try {
            Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/yourdb", "username",
"password");
            String sql = "SELECT * FROM users WHERE id = ?";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, Integer.parseInt(userId));
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                name = rs.getString("name");
                email = rs.getString("email");
                age = String.valueOf(rs.getInt("age"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
%>

<h2>Update User</h2>
<form action="updateUser.jsp" method="post">
    <input type="hidden" name="id" value="<%= userId
```