

Introduction to JSP

In Java, JSP stands for Jakarta Server Pages((JSP; formerly JavaServer Pages)). It is a server-side technology which is used for creating web applications. It is used to create dynamic web content. JSP consists of both HTML tags and JSP tags. In this, JSP tags are used to insert JAVA code into HTML pages. It is an advanced version of Servlet Technology i.e. a web-based technology that helps us to create dynamic and platform-independent web pages. In this, Java code can be inserted in HTML/ XML pages or both. JSP is first converted into a servlet by the JSP container before processing the client's request. JSP has various features like JSP Expressions, JSP tags, JSP Expression Language, etc.

How JSP more advantageous than Servlet?

- They are easy to maintain.
- No recompilation or redeployment is required.
- Less coding is required in JSP.
- JSP has access to the entire API of JAVA.
- JSP are extended version of Servlet.

Features of JSP

- *Coding in JSP is easy* : As it is just adding JAVA code to HTML/XML.
- *Reduction in the length of Code* : In JSP we use action tags, custom tags etc.
- *Connection to Database is easier* : It is easier to connect website to database and allows to read or write data easily to the database.
- *Make Interactive websites* : In this we can create dynamic web pages which helps user to interact in real time environment.
- *Portable, Powerful, flexible and easy to maintain* : as these are browser and server independent.
- *No Redeployment and No Re-Compilation* : It is dynamic, secure and platform independent so no need to re-compilation.
- *Extension to Servlet* : as it has all features of servlets, implicit objects and custom tags
 1. *Declaration Tag* : It is used to declare variables.
 2. *Java Scriptlets* : It allows us to add any number of JAVA code, variables and expressions.
 3. *JSP Expression* : It evaluates and convert the expression to a string.

4. *JAVA Comments* : It contains the text that is added for information which has to be ignored.
- Create html page from where request will be sent to server eg try.html.
 - To handle to request of user next is to create .jsp file Eg. new.jsp
 - Create project folder structure.
 - Create XML file eg my.xml.
 - Create WAR file.
 - Start Tomcat
 - Run Application
5. It does not require advanced knowledge of JAVA
6. It is capable of handling exceptions
7. Easy to use and learn
8. It contains tags which are easy to use and understand
9. Implicit objects are there which reduces the length of code
10. It is suitable for both JAVA and non JAVA programmer
11. Difficult to debug for errors.
12. First time access leads to wastage of time
13. It's output is HTML which lacks features.

Creating a simple JSP Page

hello.JSP :

JSP simply puts Java inside HTML pages. You can take any existing HTML page and change its extension to “.jsp” instead of “.html”. In fact, this is the perfect exercise for your first JSP.

Take the HTML file you used in the previous exercise. change its extension from “.html” to “.jsp”. Now load the new file, with the “.jsp” extension, in your browser.

You will see the same output, but it will take longer! But only the first time. If you reload it again, it will load normally.

What is happening behind the scenes is that your JSP is being turned into a Java file, compiled, and loaded. This compilation only happens once, so after the first

load, the file doesn't take long to load anymore. (But every time you change the JSP file, it will be re-compiled again.)

Of course, it is not very useful to just write HTML pages with a .jsp extension! We now proceed to see what makes JSP so useful.

Adding dynamic content via expressions:

As we saw in the previous section, any HTML file can be turned into a JSP file by changing its extension to .jsp . Of course , what makes JSP useful is the ability to embed Java. Put the following text in a file. jsp extension (let us call it hello.jsp) , place it in your JSP directory, and view it in a browser.

```
<HTML>
<BODY>
  Hello! The time is now <%= new java.util.Date() %>
</BODY>
</HTML>
```

Notice that each time you reload the page in the browser, it comes up with the current time. The character sequence.

<%= and %> enclose Java expressions, which are evaluated at run time.

This is what makes it possible to use JSP to generate dynamic HTML pages that change in response to user actions or vary from user to user.

Explain JSP Elements:

We will learn about the various elements available in JSP with suitable examples. In JSP elements can be divided into 4 different types.

These are:

- Expression
- Scriptlets
- Directives
- Declarations

Expression:

We can use this tag to output any data on the generated page. These data are automatically converted to string and printed on the output stream.

Syntax:

JSP Expressions are : `<%= "Anything" %>`

NOTE : JSP Expressions start with Syntax of JSP Scriptlets are with `<%=` and ends with `%>`. Between these, you can put anything that will convert to the String and that will be displayed.

Example:

`<%= "HelloWorld!" %>`

Scriptlets:

In this tag we can insert any amount of valid java code and these codes are placed in the `_jspService` method by the JSP engine.

Syntax:

`<%//java codes%>`

NOTE : JSP Scriptlets begins with `<%` and ends `%>`. We can embed any amount of Java code in the JSP Scriptlets. JSP Engine places these codes in the `_jspService()` method.

Variables available to the JSP Scriptlets are:

- Request
- Response
- Session
- Out

Directives:

A JSP “directive” starts with <%@ characters. In the directives, we can import packages , and define error-handling pages or the session information of the JSP page.

Syntax:

```
<%@directive attribute="value"% >
```

- page
- include
- taglib

Declarations :

This tag is used for defining the functions and variables to be used in the JSP.

Syntax:

```
<%!  
    //java codes  
%>
```

NOTE : JSP Declaratives begins with <%! and ends %> with We can embed any amount of java code in the JSP Declaratives. Variables and functions defined in the declaratives are class-level and can be used anywhere on the JSP page.

Example :

```
<%@page import="java.util.*"%>
<HTML>
<BODY>
<%!
Date theDate = new Date(); // Corrected the unwanted
space in the declaration
Date getDate() {
    System.out.println("In getDate() method");
    return theDate;
}
%>
Hello! The time is now <%=getDate()%>
</BODY>
</HTML>
```

Example of a JSP Web Page:

```
<HTML>
<HEAD>
<TITLE>A Web Page</TITLE>
</HEAD>
<BODY>
<%out.println("Hello there!");%>
</BODY>
</HTML>
```

Run a Simple JSP Page:

Step-1: Save the JSP file using “.jsp” extension (ex- “hello.jsp”)

Step-2: Start the server

Step-3: Place your application inside a folder

Step-4: To execute the JSP script, simply start tomcat server and use a browser to browse an URL of the JSP page i.e.

Servlet and JSP

Servlet technology is used to create a web application. A servlet is a Java class that is used to extend the capabilities of servers that host applications accessed by means of a request-response model. Servlets are mainly used to extend the applications hosted by web services.

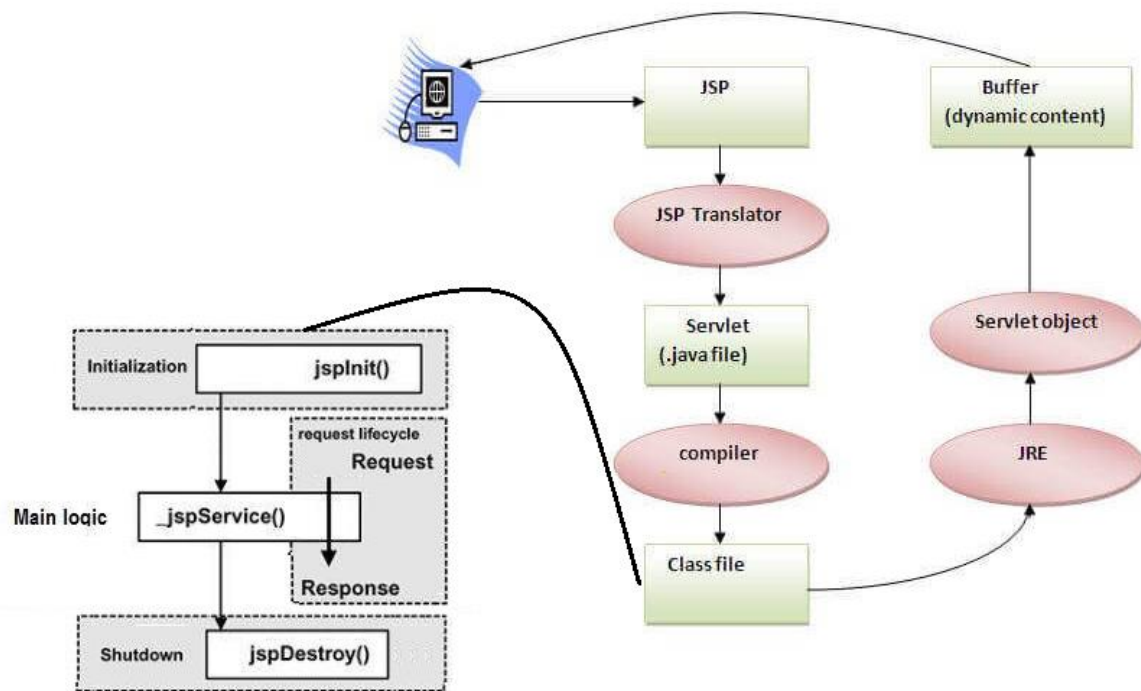
JSP is used to create web applications just like Servlet technology. A JSP is a text document that contains two types of text: static data and dynamic data. The static data can be expressed in any text-based format (like HTML, XML, SVG, and WML), and the dynamic content can be expressed by JSP elements. Difference between Servlet and JSP.

The difference between Servlet and JSP is as follows:

Servlet	JSP
Servlet is a java code.	JSP is a HTML-based compilation code.
Writing code for servlet is harder than JSP as it is HTML in java.	JSP is easy to code as it is java in HTML.
Servlet plays a controller role in the ,MVC approach.	JSP is the view in the MVC approach for showing output.
Servlet is faster than JSP.	JSP is slower than Servlet because the first step in the JSP lifecycle is the translation of JSP to java code and then compile.
Servlet can accept all protocol requests.	JSP only accepts HTTP requests.
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.
In Servlet we have to implement everything like business logic and presentation logic in just one servlet file.	In JSP business logic is separated from presentation logic by using JavaBeansclient-side.
Modification in Servlet is a time-consuming compiling task because it includes reloading, recompiling, JavaBeans and restarting the server.	JSP modification is fast, just need to click the refresh button.
It does not have inbuilt implicit objects.	In JSP there are inbuilt implicit objects.
There is no method for running JavaScript on the client side in Servlet.	While running the JavaScript at the client side in JSP, client-side validation is used.
Packages are to be imported on the top of the program.	Packages can be imported into the JSP program (i.e, bottom , middleclient-side, or top)
It can handle extensive data processing.	It cannot handle extensive data processing very efficiently.
The facility of writing custom tags is not present.	The facility of writing custom tags is present.
Servlets are hosted and executed on Web Servers.	Before the execution, JSP is compiled in Java Servlets and then it has a similar lifecycle as Servlets.

Life cycle of JSP

A Java Server Page life cycle is defined as the process that started with its creation which later translated to a servlet and afterward servlet lifecycle comes into play. This is how the process goes on until its destruction.



Following steps are involved in the JSP life cycle:

1. Translation of JSP page to Servlet
2. Compilation of JSP page(Compilation of JSP into test.java)
3. Classloading (test.java to test.class)
4. Instantiation(Object of the generated Servlet is created)
5. Initialization(jspInit() method is invoked by the container)
6. Request processing(_jspService()is invoked by the container)
7. JSP Cleanup (jspDestroy() method is invoked by the container)

We can override jspInit(), jspDestroy() but we can't override _jspService() method.

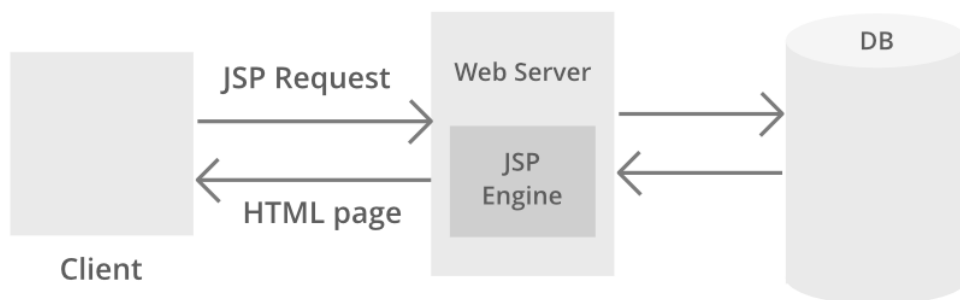
Translation of JSP page to Servlet:

This is the first step of the JSP life cycle. This translation phase deals with the Syntactic correctness of JSP. Here test.jsp file is translated to test.java.

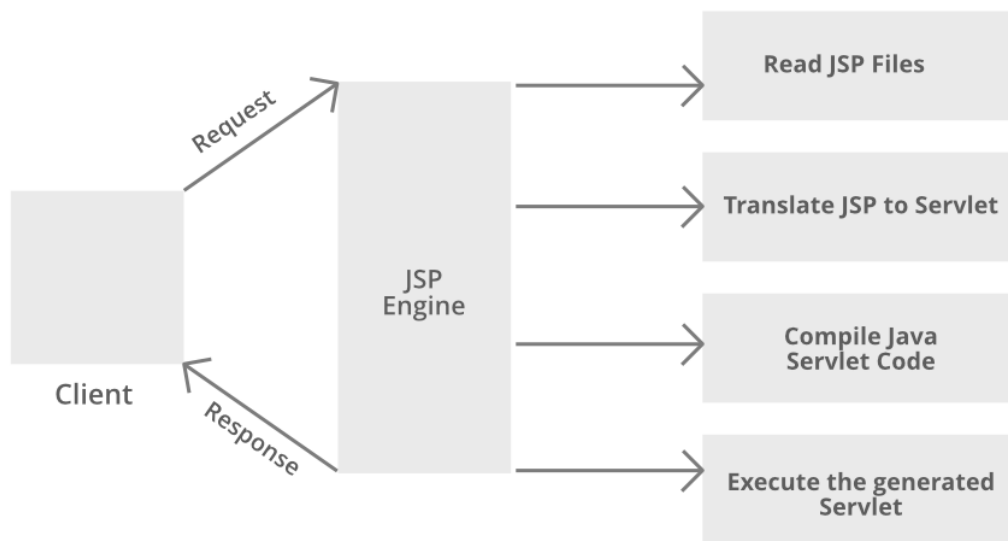
1. *Compilation of JSP page:* Here the generated java servlet file (test.java) is compiled to a class file (test.class).
2. *Classloading:* The classloader loads the Java class file into the memory. The loaded Java class can then be used to serve incoming requests for the JSP page.
3. *Instantiation:* Here an instance of the class is generated. The container manages one or more instances by providing responses to requests.
4. *Initialization:* jspInit() method is called only once during the life cycle immediately after the generation of the Servlet instance from JSP.
5. *Request processing:* _jspService() method is used to serve the raised requests by JSP. It takes request and response objects as parameters. This method cannot be overridden.
6. *JSP Cleanup:* In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy() method is used. This method is called once, if you need to perform any cleanup task like closing open files, or releasing database connections jspDestroy() can be overridden.

JSP Architecture

JSP architecture gives a high-level view of the working of JSP. JSP architecture is a 3-tier architecture. It has a Client, Web Server, and Database. The client is the web browser or application on the user side. Web Server uses a JSP Engine i.e.; a container that processes JSP. For example, Apache Tomcat has a built-in JSP Engine. JSP Engine intercepts the request for JSP and provides the runtime environment for the understanding and processing of JSP files. It reads, parses, build Java Servlet, Compiles and Executes Java code, and returns the HTML page to the client. The webserver has access to the Database. The following diagram shows the architecture of JSP.



Now let us discuss JSP which stands for Java Server Pages. It is a server-side technology. It is used for creating web applications. It is used to create dynamic web content. In this JSP tags are used to insert JAVA code into HTML pages. It is an advanced version of Servlet Technology. It is a Web-based technology that helps us to create dynamic and platform-independent web pages. In this, Java code can be inserted in HTML/ XML pages or both. JSP is first converted into a servlet by JSP container before processing the client's request. JSP Processing is illustrated and discussed in sequential steps prior to which a pictorial media is provided as a handful pick to understand the JSP processing better which is as follows:



Step 1: The client navigates to a file ending with the .jsp extension and the browser initiates an HTTP request to the webserver. For example, the user enters the login details and submits the button. The browser requests a status.jsp page from the webserver.

Step 2: If the compiled version of JSP exists in the web server, it returns the file. Otherwise, the request is forwarded to the JSP Engine. This is done by recognizing the URL ending with .jsp extension.

Step 3: The JSP Engine loads the JSP file and translates the JSP to Servlet(Java code). This is done by converting all the template text into println() statements and JSP elements to Java code. This process is called translation.

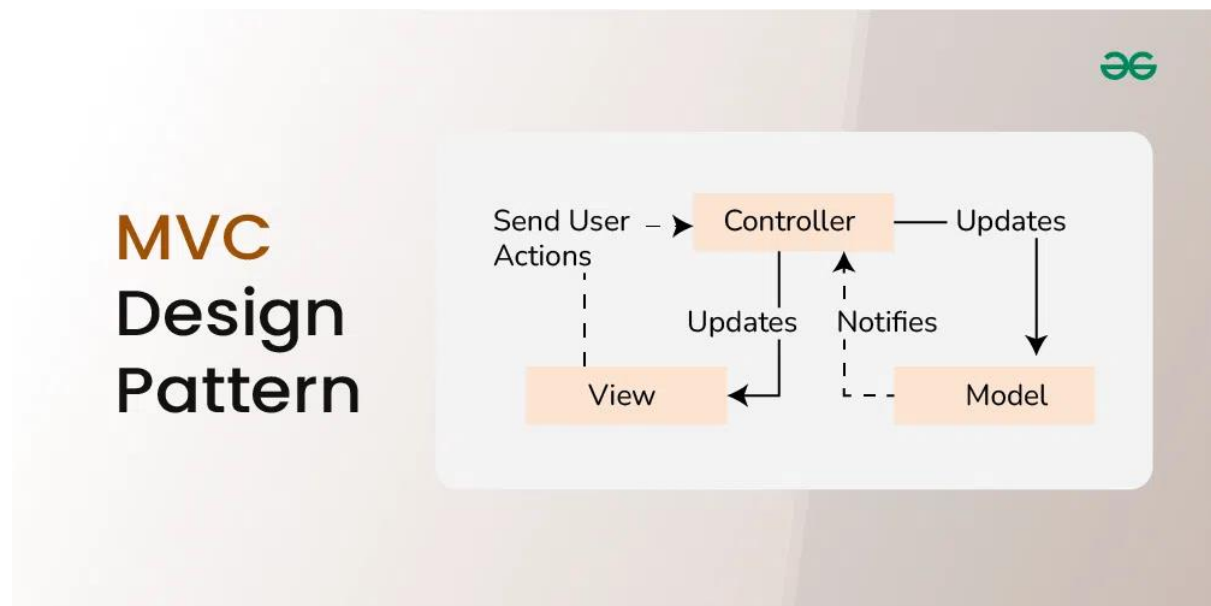
Step 4: The JSP engine compiles the Servlet to an executable .class file. It is forwarded to the Servlet engine. This process is called compilation or request processing phase.

Step 5: The .class file is executed by the Servlet engine which is a part of the Web Server. The output is an HTML file. The Servlet engine passes the output as an HTTP response to the webserver.

Step 6: The web server forwards the HTML file to the client's browser.

MVC Design Pattern

The MVC design pattern is a software architecture pattern that separates an application into three main components: Model, View, and Controller, making it easier to manage and maintain the codebase. It also allows for the reusability of components and promotes a more modular approach to software development.



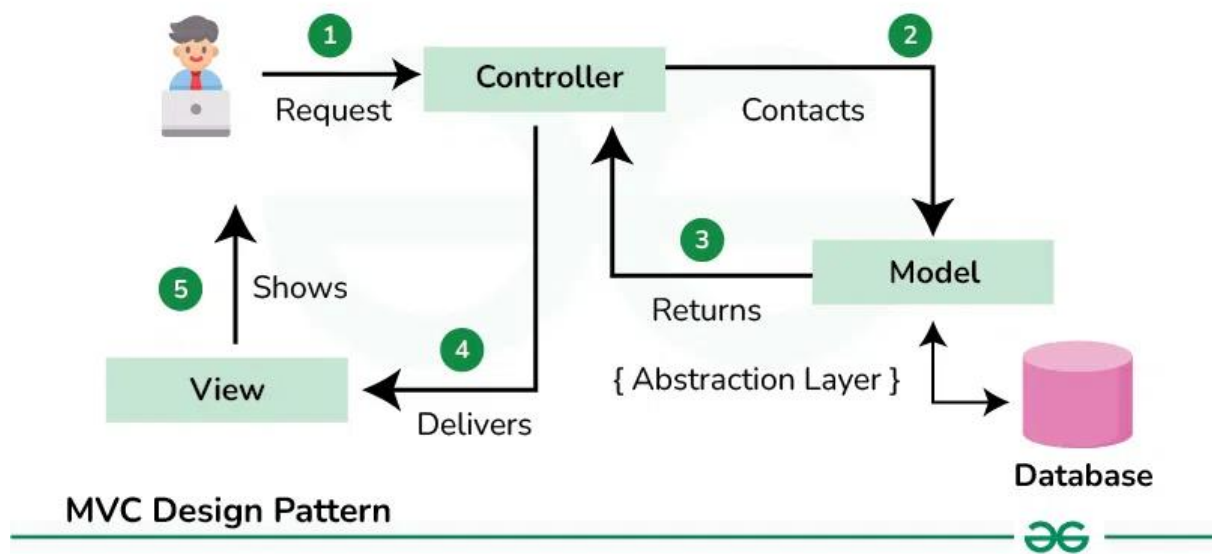
What is the MVC Design Pattern?

The Model View Controller (MVC) design pattern specifies that an application consists of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

The MVC pattern separates the concerns of an application into three distinct components, each responsible for a specific aspect of the application's functionality.

This separation of concerns makes the application easier to maintain and extend, as changes to one component do not require changes to the other components.

Components of the MVC Design Pattern



1. Model

The Model component in the MVC (Model-View-Controller) design pattern represents the data and business logic of an application. It is responsible for managing the application's data, processing business rules, and responding to requests for information from other components, such as the View and the Controller.

2. View

Displays the data from the Model to the user and sends user inputs to the Controller. It is passive and does not directly interact with the Model. Instead, it receives data from the Model and sends user inputs to the Controller for processing.

3. Controller

Controller acts as an intermediary between the Model and the View. It handles user input and updates the Model accordingly and updates the View to reflect changes in the Model. It contains application logic, such as input validation and data transformation.

Communication between the components

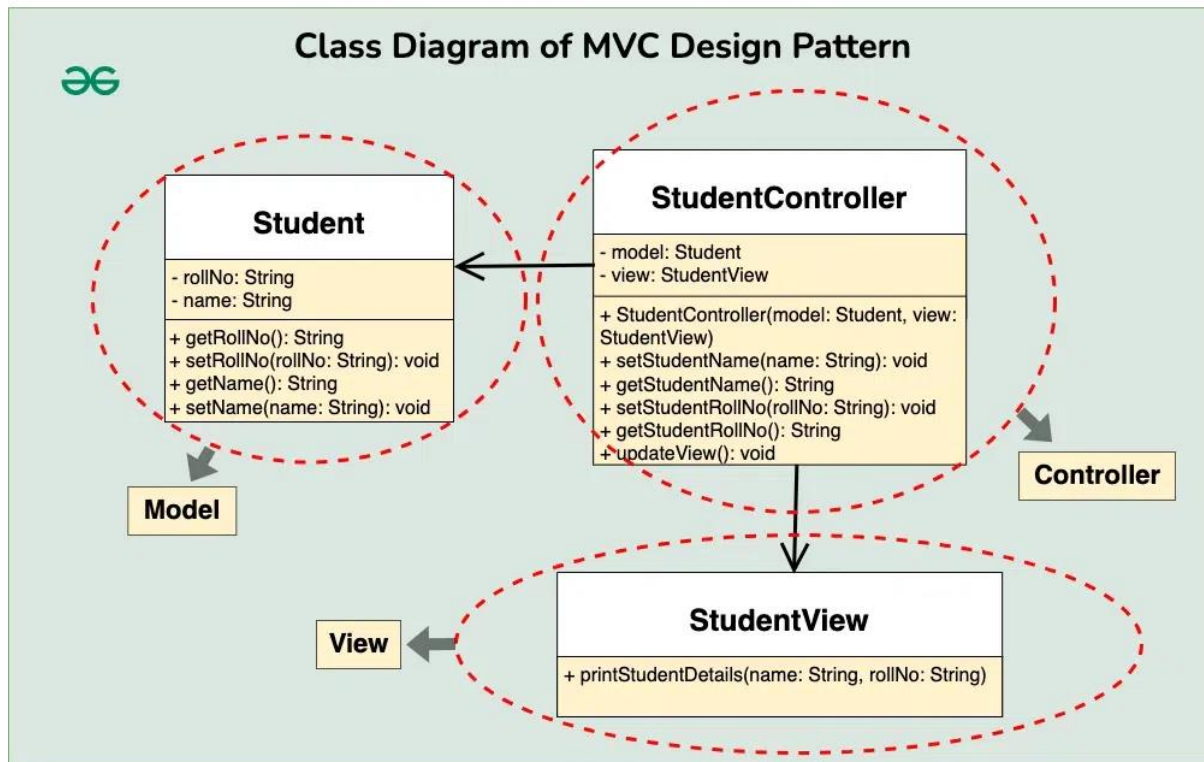
This below communication flow ensures that each component is responsible for a specific aspect of the application's functionality, leading to a more maintainable and scalable architecture

- **User Interaction with View:**
 - The user interacts with the View, such as clicking a button or entering text into a form.
- **View Receives User Input:**
 - The View receives the user input and forwards it to the Controller.
- **Controller Processes User Input:**
 - The Controller receives the user input from the View.
 - It interprets the input, performs any necessary operations (such as updating the Model), and decides how to respond.
- **Controller Updates Model:**
 - The Controller updates the Model based on the user input or application logic.
- **Model Notifies View of Changes:**
 - If the Model changes, it notifies the View.
- **View Requests Data from Model:**
 - The View requests data from the Model to update its display.
- **Controller Updates View:**
 - The Controller updates the View based on the changes in the Model or in response to user input.
- **View Renders Updated UI:**
 - The View renders the updated UI based on the changes made by the Controller.

Example of the MVC Design Pattern

Below is the code of above problem statement using MVC Design Pattern:

Let's break down into the component wise code:



1. Model (Student class)

Represents the data (student's name and roll number) and provides methods to access and modify this data.

2. View (StudentView class)

Represents how the data (student details) should be displayed to the user. Contains a method (`printStudentDetails`) to print the student's name and roll number.

3. Controller (Student Controller class)

Acts as an intermediary between the Model and the View. Contains references to the Model and View objects. Provides methods to update the Model (e.g., `setStudentName`, `setStudentRollNo`) and to update the View (`updateView`).

Example:

```
class Student {  
    private String rollNo;  
    private String name;  
  
    public String getRollNo() {  
        return rollNo;  
    }  
  
    public void setRollNo(String rollNo) {  
        this.rollNo = rollNo;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}  
  
class StudentView {  
    public void printStudentDetails(String  
studentName, String studentRollNo) {  
        System.out.println("Student:");  
        System.out.println("Name: " + studentName);  
    }  
}
```

```
        System.out.println("Roll No: " +  
studentRollNo);
```

```
    }  
}
```

```
class StudentController {
```

```
    private Student model;  
    private StudentView view;
```

```
    public StudentController(Student model,  
StudentView view) {
```

```
        this.model = model;  
        this.view = view;  
    }
```

```
    public void setStudentName(String name) {  
        model.setName(name);  
    }
```

```
    public String getStudentName() {  
        return model.getName();  
    }
```

```
    public void setStudentRollNo(String rollNo) {  
        model.setRollNo(rollNo);  
    }
```

```
    public String getStudentRollNo() {
```

```

        return model.getRollNo();
    }

    public void updateView() {
        view.printStudentDetails(model.getName(),
model.getRollNo());
    }
}

public class MVCPattern {
    public static void main(String[] args) {
        Student model = retrieveStudentFromDatabase();

        StudentView view = new StudentView();

        StudentController controller = new
StudentController(model, view);

        controller.updateView();

        controller.setStudentName("Vikram Sharma");

        controller.updateView();
    }

    private static Student
retrieveStudentFromDatabase() {
        Student student = new Student();
        student.setName("Lokesh Sharma");
    }
}

```

```
        student.setRollNo("15UCS157");  
        return student;  
    }  
}
```

Output: -

Student:

Name: Lokesh Sharma

Roll No: 15UCS157

Student:

Name: Vikram Sharma

Roll No: 15UCS157

Advantages of the MVC Design Pattern

- **Separation of Concerns:** MVC separates the different aspects of an application (data, UI, and logic), making the code easier to understand, maintain, and modify.
- **Modularity:** Each component (Model, View, Controller) can be developed and tested separately, promoting code reusability and scalability.
- **Flexibility:** Since the components are independent, changes to one component do not affect the others, allowing for easier updates and modifications.
- **Parallel Development:** Multiple developers can work on different components simultaneously, speeding up the development process.
- **Code Reusability:** The components can be reused in other parts of the application or in different projects, reducing development time and effort.

Disadvantages of the MVC Design Pattern

- **Complexity:** Implementing the MVC pattern can add complexity to the code, especially for simpler applications, leading to overhead in development.
- **Learning Curve:** Developers need to understand the concept of MVC and how to implement it effectively, which may require additional time and resources.
- **Overhead:** The communication between components (Model, View, Controller) can lead to overhead, affecting the performance of the application, especially in resource-constrained environments.
- **Potential for Over-Engineering:** In some cases, developers may over-engineer the application by adding unnecessary abstractions and layers, leading to bloated and hard-to-maintain code.
- **Increased File Count:** MVC can result in a larger number of files and classes compared to simpler architectures, which may make the project structure more complex and harder to navigate.

Directives in JSP

JSP directives are the elements of a JSP source code that guide the web container on how to translate the JSP page into its respective servlet.

Syntax :

```
<%@ directive attribute = "value"%>
```

Directives can have a number of attributes that you can list down as key-value pairs and separate by commas. The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

Different types of JSP directives:

There are three different JSP directives available. They are as follows:

1. **Page Directives :** JSP page directive is used to define the properties applying the JSP page, such as the size of the allocated buffer, imported packages, and classes/interfaces, defining what type of page it is, etc. The syntax of JSP page directive is as follows:

```
<%@page attribute = "value"%>
```

Different properties/attributes:

The following are the different properties that can be defined using page directive:

- **import:** This tells the container what packages/classes are needed to be imported into the program.

Syntax:

```
<%@page import = "value"%>
```

- **contentType:** This defines the format of data that is being exchanged between the client and the server. It does the same thing as the `setContentType` method in servlet used to.

Syntax:

```
<%@page contentType="value"%>
```

- **info:** Defines the string which can be printed using the `'getServletInfo()'` method.

Syntax:

```
<%@page info="value"%>
```

- **buffer:** Defines the size of the buffer that is allocated for handling the JSP page. The size is defined in Kilo Bytes.

Syntax:

```
<%@page buffer = "size in kb"%>
```

- **language:** Defines the scripting language used in the page. By default, this attribute contains the value `'java'`.
- **isELIgnored:** This attribute tells if the page supports expression language. By default, it is set to false. If set to true, it will disable expression language.

Syntax:

```
<%@page isELIgnored = "true/false"%>
```

- **errorPage:** Defines which page to redirect to, in case the current page encounters an exception.

Syntax:

```
<%@page errorPage = "true/false"%>
```

- **isErrorPage:** It classifies whether the page is an error page or not. By classifying a page as an error page, it can use the implicit object `'exception'` which can be used to display exceptions that have occurred.

Syntax:


```
<%@page isErrorPage="true/false"%>
```

➤ **Include directive :**

JSP include directive is used to include other files into the current jsp page. These files can be html files, other sp files etc. The advantage of using an include directive is that it allows code re-usability.

The syntax of an include directive is as follows:

```
<%@include file = "file location"%>
```

➤ **Taglib Directive :** The taglib directive is used to mention the library whose custom-defined tags are being used in the JSP page. It's major application is JSTL(JSP standard tag library).

Syntax:

```
<%@taglib uri = "library url" prefix="the prefix to  
identify the tags of this library with"%>
```

JSP – Action Tags

In Java, JavaServer Pages can provide action tags that can enable the developers to perform specific tasks including the other files, forwarding the requests, and manipulating the session data within the JSP documents. Action tags are denoted by the `<jsp: ..>` syntax and it is used to perform the dynamic actions on the server side of the JSP applications.

Understanding of the JSP Action Tags

JSP action tags are the special tags that can be used to provide instructions to the JSP container on how to manage the server-side actions. It can be enclosed with the `<jsp: .. >` tags and it can allow the developers to perform various tasks such as including other files, forwarding the requests, and manipulating the session attributes.

List of the Commonly used JSP Action Tags in JSP

Tag Name	Syntax	Description	Example
Include Tag	<pre><jsp:include page="filename.jsp" /></pre>	It can be used to include the content of the other resources like the JSP, HTML, or servlet in the current JSP page.	<pre><jsp:include page="header.jsp" /></pre>
Forward Tag	<pre><jsp:forward page="destination.jsp" /></pre>	This tag can be used to forward the current request to another resource like the JSP, HTML, or servlet without the client's knowledge.	<pre><jsp:forward page="success.jsp" /></pre>
setProperty Tag	<pre><jsp:setProperty name="beanName" property="propertyName" value="propertyValue" /></pre>	This tag can be used to set the properties of the JavaBean	<pre><jsp:setProperty name="user" property="username" value="John" /></pre>

Tag Name	Syntax	Description	Example
		component of the JSP pages.	
getProperty	<pre><jsp:getProperty name="beanName" property="propertyName" /></pre>	This tag can be used to retrieve the properties of the JavaBean component of the JSP pages.	<pre><jsp:getProperty name="user" property="username" /></pre>
useBean Tag	<pre><jsp:useBean id="beanId" class="packageName.class Name" /></pre>	This tag can be used to instantiate the JavaBean component or retrieve the existing instance of the JSP pages.	<pre><jsp:useBean id="user" class="com.example. User" /></pre>
plugin Tag	<pre><jsp:plugin type="pluginType" code="pluginCode" /></pre>	It can be used to generate the	<pre><jsp:plugin type="applet" code="MyApplet.class" /></pre>

Tag Name	Syntax	Description	Example
		HTML code for the browser specific plugin.	
attribute tag	<pre><jsp:attribute name="attributeName" value="attributeValue" /></pre>	<p>This tag can be used to defines the attributes values for the custom actions of the JSP pages.</p>	<pre><jsp:attribute name="color" value="blue" /></pre>
Body Tag	<pre><jsp:body> <!-- Body Content --> </jsp:body></pre>	<p>This tag can be used to defines the body content for the custom actions of the JSP pages.</p>	<pre><jsp:body> <p>This is the body content.</p> </jsp:body></pre>

JSP | Implicit Objects – request and response

JSP stands for Java Server Pages, and it's a server side technology. It's used for creating web applications and dynamic web content. The main property of JSP is that we can insert our java code inside our HTML page using JSP tag. JSP provides you platform-independent pages.

Implicit objects are a set of Java objects that the JSP Container makes available to developers on each page. These objects may be accessed as built-in variables via scripting elements and can also be accessed programmatically by JavaBeans and Servlets. JSP provide you Total 9 implicit objects which are as follows

1. **request:** This is the object of `HttpServletRequest` class associated with the request.
2. **response:** This is the object of `HttpServletResponse` class associated with the response to the client.
3. **config:** This is the object of `ServletConfig` class associated with the page.
4. **application:** This is the object of `ServletContext` class associated with the application context.
5. **session:** This is the object of `HttpSession` class associated with the request.
6. **page context:** This is the object of `PageContext` class that encapsulates the use of server-specific features. This object can be used to find, get or remove an attribute.
7. **page object:** The manner we use the keyword `this` for current object, page object is used to refer to the current translated servlet class.
8. **exception:** The exception object represents all errors and exceptions which is accessed by the respective jsp. The exception implicit object is of type `java.lang.Throwable`.
9. **out:** This is the `PrintWriter` object where methods like `print` and `println` help for displaying the content to the client.

request Object

The JSP request is an implicit object which is provided by `HttpServletRequest`. In the servlet, we have to first import `javax.servlet.http.HttpServletRequest` then we have to create its object for taking input from any HTML form as.

Syntax :

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

public class LoginServlet extends HttpServlet
{
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException,
        IOException
    {
        HttpSession session =
        request.getSession()
    }
}
```

In JSP, the request object is implicitly defined so that you don't have to create an object. JSP request object is created by the web container for each request of the client. It's used for getting the parameter value, server name, server port, etc.

response object

This is the `HttpServletResponse` object associated with the response to the client. The response object also defines the interfaces that deal with creating new HTTP headers. Through this object the JSP programmer can add new cookies or date stamps, HTTP status codes, etc.

In JSP the response object is implicitly defined, so you don't have to create an object. JSP response object is created by the web container for each request of the client. It basically is used for redirecting to any other resource.

Advantage of JSP over servlet:

- Servlets are difficult to code than JSP. In another way, we can say, JSP is the replacement for Servlets.
- In Servlets, both static code and dynamic code are put together. In JSP, they are separated.
- The objects of PrintWriter, ServletConfig, ServletContext, HttpSession and RequestDispatcher etc. are created by the Programmer in Servlets. But in JSP, they are built-in and are known as implicit objects.

Disadvantage:

- JSP pages require more memory to hold the page.
- The output is in HTML which is not rich for viewers.

JSP Session

In JSP, the session is the most regularly used implicit object of type HttpSession. It is mainly used to approach all data of the user until the user session is active.

Methods used in session Implicit Object are as follows:

Method 1: isNew(): This method is used to check either the session is new or not. Boolean value (true or false) is returned by it. Primarily it used to trace either the cookies are enabled on client side or not. If cookies are not enabled then the session.isNew() method should return true all the time.

Method 2: getId(): While creating a session, the servlet container allots a distinctive string identifier to the session. This distinctive string identifier is returned by getId method.

Method 3: getAttributeNames(): All the objects stored in the session are returned by getAttributeNames method. Fundamentally, this method results in an enumeration of objects.

Method 4: getCreationTime(): The session creation time (the time when the session became active or the session began) is returned by getCreationTime method.

Method 5: getAttribute(String name): Using getAttribute method, the object which is stored by the setAttribute() method is retrieved from the session. For example, We need to store the “userid” in session using the setAttribute() method if there is the requirement of accessing userid on every jsp page until the session is active and when needed it can be accessed using the getAttribute() method.

Method 6: setAttribute(String, object): The setAttribute method is used to store an object in session by allotting a unique string to the object. Later, By using the same string this object can be accessed from the session until the session is active.

In JSP, while dealing with session `setAttribute()` and `getAttribute()` are the two most regularly used methods.

Method 7: `getMaxInactiveInterval()`: The `getMaxInactiveInterval` return session's maximum inactivates time interval in seconds.

Method 8: `getLastAccessedTime`: The `getLastAccessedTime` method is mostly used to notice the last accessed time of a session.

Method 9: `removeAttribute(String name)`: Using `removeAttribute(String name)` method, the objects that are stored in the session can be removed from the session.

Method 10: `invalidate()`: The `invalidate()` method ends a session and breaks the connection of the session with all the stored objects.

Cookies Handling in JSP

When do we need to use Cookies?

A cookie is a small object; it can be used to represent a single name-value pair and which will be maintained permanently at the client machine. In HttpSession Session Tracking Mechanism, all the client's conversations will be maintained at the server machine in the form of HttpSession objects. In HttpSession Session Tracking Mechanism, if we increase the number of users then automatically a number of HttpSession objects will be created at the server. So that HttpSession Tracking Mechanism will increase the burden to the server machine. To overcome the above problem, we have to use an alternative mechanism, where we have to manage all the client's conversations at the respective client machines only. To achieve the above requirement, we have to use Cookies Session Tracking Mechanism.

What are Cookies?

Cookies are used to transfer the data between multiple form-based applications. Simply a cookie is a small piece of information sent by the server to any client. Every cookie will store the data in the form of key-value pair. The server will add the cookie to the response header by using the SetCookie header and send it to the client. Cookies will be stored in the browser by using the domain name of the website. The browser can contain cookies from multiple domains. The client needs to use HeaderCookie to get all the cookies available in the browser.

Types of Cookies in JSP

1. *Persistent Cookie*: Persistent cookies means the cookie data will be available even though we close the browser up to a specified time. If we want to get persistent cookies we have to use the `setMaxAge()` method.
2. *Non-Persistent Cookie*: Non-persistent cookies means the cookie data will be deleted when we close the browser immediately. By default, we get non-persistent cookies.

How do the Cookies Work in JSP?

Every cookie contains a name and allows one string value. Every cookie contains a cookie ID and also remembers the web application/domain name to which it belongs to. Server-side web resource programs of web applications create cookies

and add them to the response to send to the client. So, cookies allocate memory as String information at the client-side. Cookies go back to the web application along with the request when the browser window gives a request back to the web application for which these cookies belong to. Cookies come to the client from the server/web application along with the response, as the values of the special response header called “set-cookie”. Cookies go back to the web application along with the request given by the browser window, as the values of the special request header called “cookie”. At the client-side or browser window, we can see multiple cookies belonging to different domains.

Advantages of Cookies:

- Compared to hidden variables cookies are better to transfer the data because once we store the cookie, we can get the data on any page of the website directly.
- Cookies work with all server-side technologies and all servers.
- Persistent cookies can remember client data even after the session having expiry time.

Disadvantages of Cookies:

- If the client disables the cookies in the browser we can work with cookies.
- For security reasons, it is not recommended to use cookies.
- If we want to a huge amount of data between server and client it will be complicated.

How to handle Cookie?

Handling cookie involves below three steps:

1. Creating a cookie object
2. Setting the maximum age of cookie
3. Sending cookie into the HTTP response headers

JSP Cookies API

JSP provides a cookie class in `javax.servlet.http` package. We can call Cookie with a cookie name and a cookie value as follows:

1. *Cookie()*: It will construct a cookie.
2. *Cookie(String name, String value)*: It will construct a cookie with mentioned name and value.

Cookies Methods

1. *setDomain(String pattern)*: It defines the domain of the cookie.
2. *getDomain()*: It gets the domain of the cookie.
3. *setMaxAge(int expiry)*: It sets the time for which the cookie will remain. It expires automatically when the session ends if nothing is set.
4. *getMaxAge()*: It gets the age of the cookie in seconds. It sends -1 when the browser closes if the age is not set.
5. *getName()*: It will return the name of the cookie.
6. *setValue(String newValue)*: It will set the value of the cookie.
7. *getValue()*: It will get the value of the cookie.
8. *setPath(String uri)*: It will set the path of the cookie. If the path is not defined, then the cookie gets applied to all the URLs of that directory.
9. *getPath()*: It will get the path of the associated cookie.
10. *setSecure(Boolean flag)*: It tells whether the cookie needs to be sent over encrypted connections or not.
11. *setComment(String purpose)*: It gives a message to a cookie so that the user can understand the use of it.
12. *getComment()*: It returns the message of the cookie and returns null if it is not there.

Restricted Characters in Cookie

Following is the list of restricted characters which we cannot use in name and value of a cookie.

, (comma)

= (equals)

() (parenthesis)

” (double quotes)

[] (square brackets)

/ (slash)

: (colon)

@

Sending Cookies to the client / How to set Cookie?

1. Creating a cookie: `Cookie cookie = new Cookie("key","value");`
2. Setting the maximum age: `Cookie.setMaxAge(60*60*24);`
3. Sending the cookie into the HTTP response headers:
`response.addCookie(cookie);`

Cookies Handling Example in JSP

In this example, we are setting the cookie. In `setCookie.html` we are taking a form that has to be processed in `main.jsp`. We are also taking two fields "First name" and "Last name" which have to be taken as input from the user with a submit button. In `main.jsp` we are creating two cookie objects of "First name" and "Last name" using `request.getParameter`. We are also adding age to both the cookies. Cookies added to the session of `firstname` and `lastname` can be fetched by `getParameter()`.

setCookie.html

```
<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Setting Cookies</title>

</head>

<body>

<h1>Setting Cookies</h1>

  <form action="main.jsp" method="GET">

    First Name: <input type="text" name="first_name">
<br /><br /> Last

    Name: <input type="text" name="last_name"
/><br /><br /> <input type="submit" value="Submit" />

  </form>
```

```
</body>
</html>
```

main.jsp

```
<%
// Create cookies for first and last names.
Cookie firstName = new Cookie("first_name",
request.getParameter("first_name"));

Cookie lastName = new Cookie("last_name",
request.getParameter("last_name"));

// Set expiry date after one hour for both the
cookies.

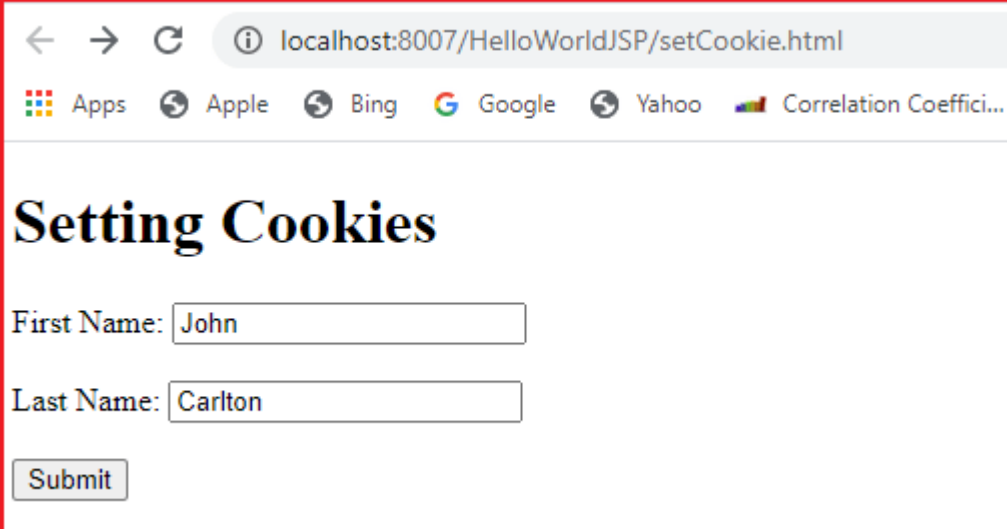
firstName.setMaxAge(60 * 60);
lastName.setMaxAge(60 * 60);

// Add both the cookies in the response header.
response.addCookie(firstName);
response.addCookie(lastName);
%>
<html>
<body>
<h1>Setting Cookies</h1>
  <ul>
    <b>First
Name:</b><%=request.getParameter("first_name")%><br
/><br/>
    <b>Last
Name:</b><%=request.getParameter("last_name")%>
```

```
</body>
```

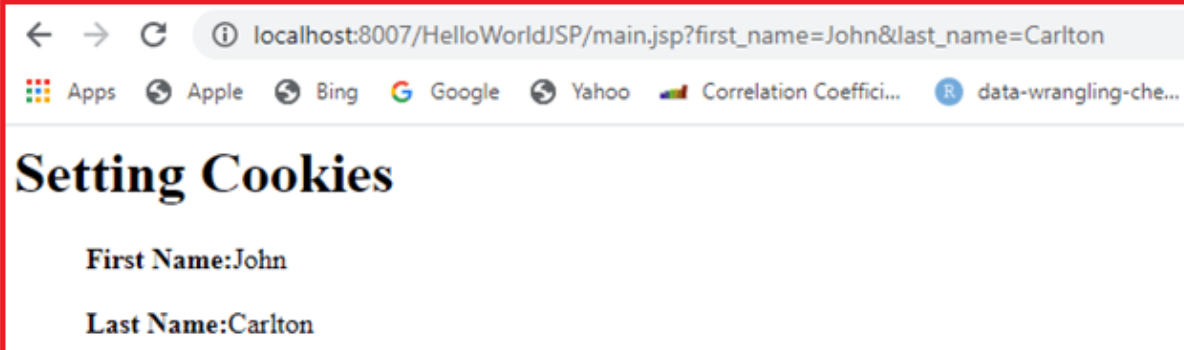
```
</html>
```

Output:



A screenshot of a web browser window. The address bar shows 'localhost:8007/HelloWorldJSP/setCookie.html'. The browser's toolbar includes icons for Apps, Apple, Bing, Google, Yahoo, and Correlation Coeffici... Below the toolbar, the page title 'Setting Cookies' is displayed in a large, bold, black serif font. Underneath the title, there are two text input fields. The first is labeled 'First Name:' and contains the text 'John'. The second is labeled 'Last Name:' and contains the text 'Carlton'. At the bottom left of the form area, there is a 'Submit' button with a light gray background and a thin black border.

Once you click on the Submit button, you will get the following output.



A screenshot of a web browser window. The address bar shows 'localhost:8007/HelloWorldJSP/main.jsp?first_name=John&last_name=Carlton'. The browser's toolbar includes icons for Apps, Apple, Bing, Google, Yahoo, Correlation Coeffici..., and data-wrangling-che... Below the toolbar, the page title 'Setting Cookies' is displayed in a large, bold, black serif font. Underneath the title, the text 'First Name:John' is displayed in a smaller, bold, black serif font. Below that, the text 'Last Name:Carlton' is displayed in a smaller, bold, black serif font.

How to Read Cookies in JSP?

Create an array `javax.servlet.http.Cookie` objects by calling `getCookies()` method from `HttpServletResponse` to read the cookies. Then use `getName()` and `getValue()` methods to access each cookie and associated value using if loop.

JSP Cookies Read Example

In this example, we will create an object of Cookie type and an array of Cookie type which will get all the cookies using request.getCookie() method of HttpServletRequest. Then we will get the name and value of these cookies using the getName() and getValue() method by applying the if loop.

readCookie.jsp

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

    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Read Cookies</title>
</head>
<body>
<h1>Read Cookies </h1>
<%
    Cookie cookie = null;
    Cookie[] cookies = null;
    cookies = request.getCookies();
    if (cookies != null) {
        for (int i = 0; i < cookies.length; i++) {
            cookie = cookies[i];
            out.print("Name : " + cookie.getName() + ",  ");
            out.print("Value: " + cookie.getValue() + "
<br/>");
        }
    }
```

```
} else {  
    out.println("<h2>No cookies founds</h2>");  
}  
%>  
</body>  
</html>
```

Output



How to Delete Cookies?

To delete cookies, set the age of the cookie to zero using the `setMaxAge()` method.

Delete Cookies Example in JSP

In this example, we have set the cookie to zero to delete the cookie using the `setMaxAge()` method. After that add it to the HTTP Response Headers to delete this cookie.

deleteCookies.jsp

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

    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Delete Cookies</title>

</head>

<body>

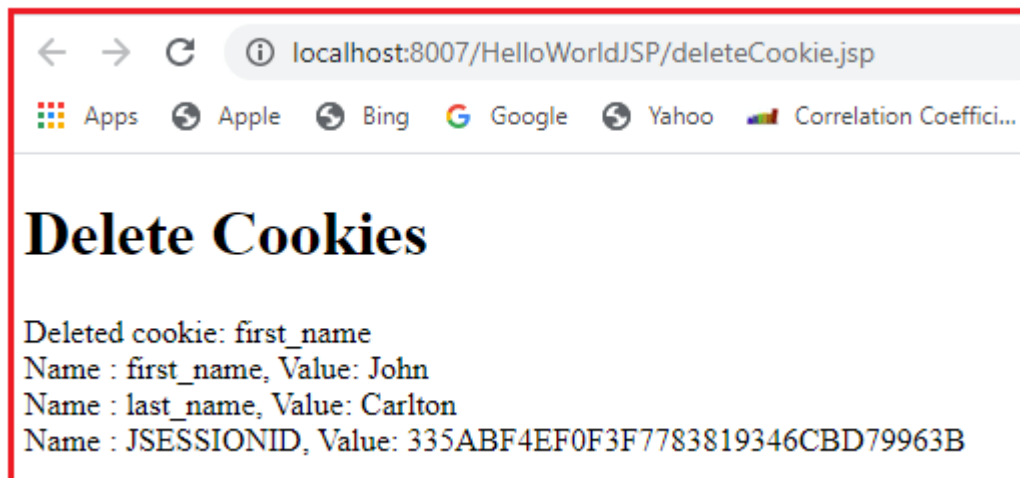
<h1>Delete Cookies</h1>

<%

    Cookie cookie = null;
    Cookie[] cookies = null;
    cookies = request.getCookies();
    if (cookies != null) {
        for (int i = 0; i < cookies.length; i++) {
            cookie = cookies[i];
            if ((cookie.getName()).compareTo("first_name") ==
0) {
                cookie.setMaxAge(0);
                response.addCookie(cookie);
                out.print("Deleted cookie: " + cookie.getName() +
"<br/>");
            }
            out.print("Name : " + cookie.getName() + ",  ");
            out.print("Value: " + cookie.getValue() + "
<br/>");
        }
    }
}
```

```
    }  
  }  
  %>  
</body>  
</html>
```

Output:



Remember Username and Password Example in JSP

In this example, we are using cookies to remember username and password features. In the RememberMe.jsp file, we have created username and password fields with a Remember Me checkbox, which will check for a cookie. If the cookie is found it will set the value of fields with the cookie values. In the HomePage.jsp file we will display username and password. If the user has checked the “Remember Me” check box, the JSP will add a username and password as a cookie.

RememberMe.jsp

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

    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Login Form</title>

</head>

<body>

<%

    Cookie[] cookies = request.getCookies();
    String username = "";
    String password = "";
    if (cookies != null) {
        for (int i = 0; i < cookies.length; i++) {
            Cookie cookie = cookies[i];
            if (cookie.getName().equals("username-cookie")) {
                username = cookie.getValue();
            } else if (cookie.getName().equals("password-
cookie")) {
                password = cookie.getValue();
            }
        }
    }

    %>

    <form name="logonform" action="HomePage.jsp"
method="POST">
```

```

    Username: <input type="text" name="username"
value="<%=username%>" />

    <br /> <br/>Password:<input type="password"
name="password"

    value="<%=password%>" /> <br /> <br/>Remember
Me<input type="checkbox"

    name="rememberMe" value="true" /> <br/><br/><input
type="submit" value="Submit" />

</form>

</body>

</html>

```

HomePage.jsp

```

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

    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Display Details</title>

</head>

<body>

    <%

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

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

        String message = "Username is : " + username +
"<br/> Password is : " + password;

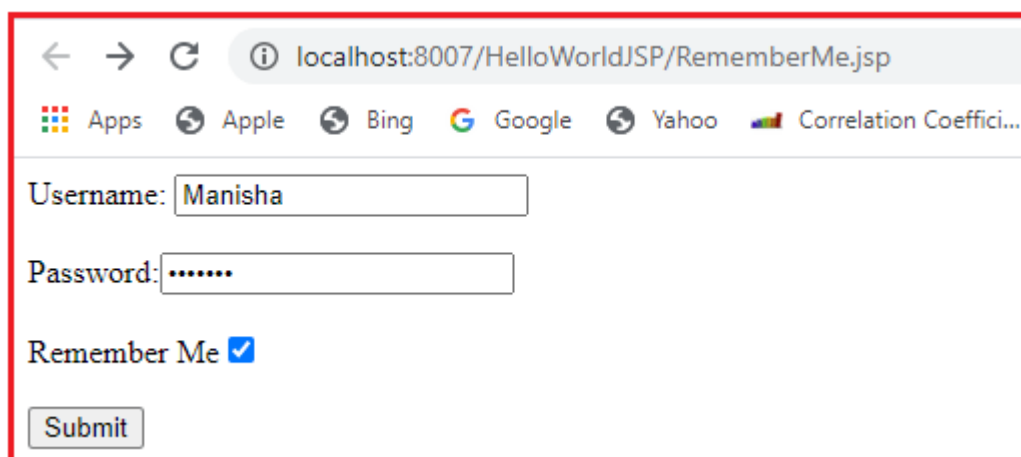
        String rememberMe =
request.getParameter("rememberMe");

```

```
if (rememberMe != null) {  
    Cookie usernameCookie = new Cookie("username-  
cookie", username);  
    Cookie passwordCookie = new Cookie("password-  
cookie", username);  
    usernameCookie.setMaxAge(24 * 60 * 60);  
    passwordCookie.setMaxAge(24 * 60 * 60);  
    response.addCookie(usernameCookie);  
    response.addCookie(passwordCookie);  
}  
%>  
<strong> <%=message%>  
</strong>  
</body>  
</html>
```

Output

Enter username and password and check the “Remember Me” check box and click submit. Here, HomePage.JSP page will add cookies. If you do not check the “Remember Me” check box, the cookies will not be added.



← → ↻ ⓘ localhost:8007/HelloWorldJSP/RememberMe.jsp

Apps Apple Bing Google Yahoo Correlation Coeffici...

Username:

Password:

Remember Me ☒

You will get the following output after clicking on Submit button.

