

Introducción a los Formularios y JTSL con JSP

Introducción a los Formularios y JTSL con JSP	1
¿Qué aprenderás?	2
Introducción	2
Introducción a JSP	3
Primer acercamiento a JSP	3
Comentarios	6
Expresiones	6
Scriptlets	7
Un poco de arquitectura	9
JSTL	10
Instalación de JSTL en un proyecto web	10
Descargar las librerías necesarias	13
Usando la librería	15
Tags JSTL	16
Librerías CORE	17
¿Qué ventaja tiene JSTL sobre los scriptlets?	17
Ejercicio guiado 1: Utilización de tag <c: out>	19
¿Qué es un formulario web?	21
Ejercicio guiado 2: Envío de valores entre jsp	22
Ejercicio guiado 3: Envío de datos entre formularios	25



¡Comencemos!

¿Qué aprenderás?

- Entender la función de un formulario.
- Introducir a la tecnología JSP y JSTL.
- Instalar librerías JSTL.
- Aplicar envío de variables entre formularios.

Introducción

JSP surge como una necesidad de introducir código para la generación dinámica de HTML dentro de una página web. Una de sus ventajas es que permite separar la interfaz del usuario de la parte lógica del contenido, facilitando el trabajo de diseñadores y desarrolladores en sus proyectos web. Así, se ha convertido en una de las tecnologías más utilizadas actualmente para el desarrollo web con Java, ya que facilita el trabajo con HTML con código Java incrustado, mediante scriptlets.

Utilizando JSTL es posible fácilmente el envío de valores de formularios mediante métodos post o get. Haremos unos ejemplos que grafican este mecanismo. Haremos un acercamiento a la tecnología JSTL y a los JSP, pero no será muy extenso, ya que esta tecnología se tratará en detalle en unidades posteriores. El objetivo real es entender cómo es el proceso de envío de información entre formularios y servlets.

Introducción a JSP

JSP (Java Server Pages), es una tecnología de Java JEE que provee herramientas para construir páginas web dinámicas, basadas en el lenguaje de marcado HTML, XML u otro tipo de documentos.

JSP tiene muchas similitudes con otras tecnologías como lo son PHP o ASP, pero implementando el lenguaje de programación Java. Para poder utilizar esta tecnología JSP, es necesario contar con un servidor web compatible con un contenedor de servlets como Apache, el cual ya fue utilizado en unidades anteriores.

Primer acercamiento a JSP

Para partir entendiendo de qué se trata todo esto, veamos un pequeño ejemplo de un código JSP sencillo.

En nuestro proyecto anterior, crearemos un archivo llamado Funciones.jsp de la siguiente manera:

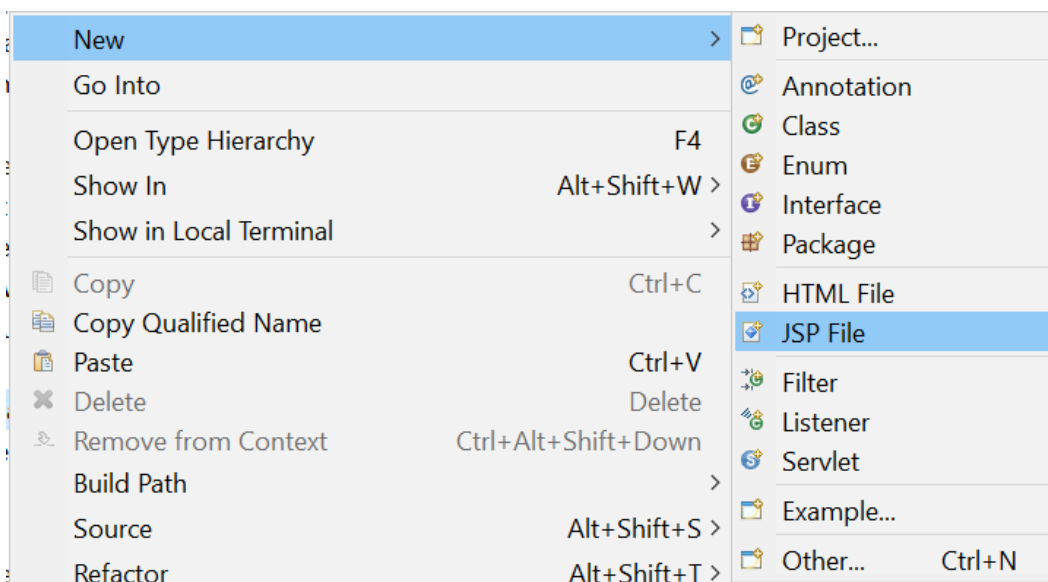


Imagen 1. Ejemplo JSP.
Fuente: Desafío Latam

Veremos que nos genera un archivo que a simple vista posee código HTML sencillo, más un encabezado que llama la atención por los caracteres `<%@ %>`:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
  <meta charset="ISO-8859-1">
  <title>Insert title here</title>
</head>
<body>
</body>
</html>
```

Esas son las llamadas directivas y le indican al contenedor, por ejemplo, que el lenguaje que estamos utilizando es Java.

Algunos atributos de uso habitual en las directivas son:

- **language:** Indica qué lenguaje de programación estamos utilizando (por defecto, Java).
- **contentType:** Indica qué tipo de contenido posee la página (por defecto, text/html).
- **isErrorPage:** Indica si la página que estamos cargando es una página de error (por defecto, false).
- **errorPage:** Define la página a la que debe dirigirse si ocurre una excepción.

Volvamos a nuestro archivo y ejecutamos lo siguiente:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Mi primer JSP</title>
</head>
<body>
  <H1>Algunas expresiones en Java</H1>
  <!-- Mostrar texto en mayúscula -->
```

```
<%= "Hola mundo".toUpperCase() %><br>

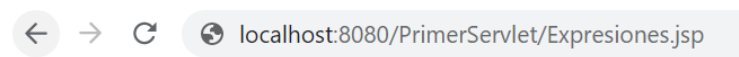
<!-- Imprimir una variable -->
<% String variable = "La hora actual:"; %>
<%= variable %><br>

<!-- Mostrar la fecha y hora -->
<%= new java.util.Date().toString() %><br>

<!-- Suma de dos enteros -->
<% int suma = 1+1; %>
<%= "1+1 = "+suma %><br>

</body>
</html>
```

El resultado de este código Java incrustado, es HTML del lado del cliente:



Algunas expresiones en Java

HOLA MUNDO
La hora actual:
Mon Apr 13 23:39:12 CLT 2020
1+1 = 2

Imagen 2. Ejemplo JSP.
Fuente: Desafío Latam

Si vamos al inspector de elementos, no vemos nada fuera de lo común:

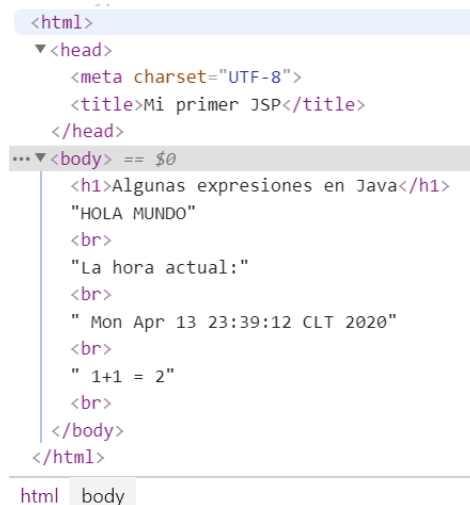


Imagen 3. Inspector de elementos.
Fuente: Desafío Latam

Comentarios

Lo primero que podemos observar del código que acabamos de ejecutar es que la forma de comentar es un tanto distinta a los que hemos visto en HTML o incluso en Java.

```
<!-- comentario HTML -->
<%-- comentario JSP --%>
```

Los comentarios que realicemos en las páginas JSP deben indicarse entre las etiquetas `<%-- --%>`.

La información que incluyamos al interior no será desplegada en el navegador web, como sí ocurre con los comentarios en HTML. Por ejemplo, el comentario `<%-- Mostrar texto en mayúscula --%>` de nuestro ejemplo, no aparece en el inspector de elementos.

Expresiones

Para mostrar una expresión Java, debemos utilizar las etiquetas `<%= %>`. El resultado de la expresión aparecerá como parte del código HTML, tal como lo observamos en el ejemplo:

```
<%= "Hola mundo".toUpperCase() %><br>
```

Que finalmente muestra HOLA MUNDO en el navegador.

Para escribir una expresión, debemos tener en consideración tres aspectos relevantes:

- El resultado de la expresión debe ser válida para el lenguaje de la página.
- El resultado de la expresión, por ende, será convertido a String una vez resuelto.
- No se debe terminar la sentencia en punto y coma.

Scriptlets

Un scriptlet puede manejar declaraciones, expresiones o cualquier otro tipo de fragmento de código válido en el lenguaje script de la página. Dentro de las etiquetas `<% %>` es posible escribir cualquier conjunto de sentencias Java, que serán ejecutadas por el servidor.

En el ejemplo anterior, resolvimos una suma y lo asignamos a una variable:

```
<% int suma = 1+1; %>
```

Este es el aspecto más potente de JSP, ya que podemos ejecutar prácticamente cualquier código Java.

Antes de continuar, intentemos hacer algo un poco más complejo. Usando el siguiente código de base ¿se te ocurre cómo hacer esta página dinámica de acuerdo al horario en que el usuario la visite?:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
  <title>Ejemplo de Scriptlets</title>
</head>
<body>
  <%
    java.util.Calendar now = java.util.Calendar.getInstance();
    int hora = now.get(java.util.Calendar.HOUR_OF_DAY);
  %>
  <b>Hola mundo,
  //si son más de las 6 y menos de las 12, dirá "buenos días"
  //si son más de las 12 y menos de las 20, dirá "buenas tardes"
```

```
//para el resto de los horarios, dirá "buenas noches"  
</b>  
</body>  
</html>
```

Un poco más abajo está la respuesta, pero el resultado que esperamos, es algo como esto:

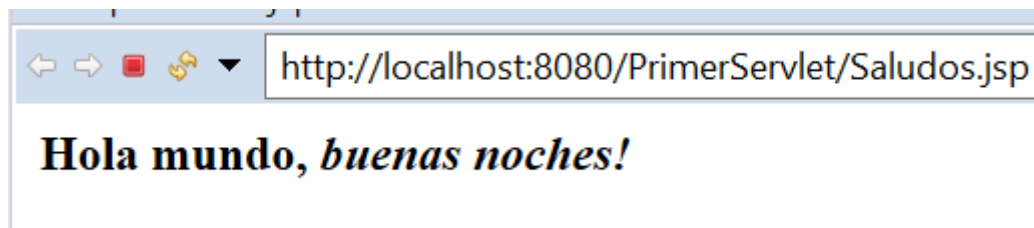


Imagen 4. Saludando con scriptlets.
Fuente: Desafío Latam

Intenta resolverlo, antes de mirar el resultado que viene a continuación:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>  
<!DOCTYPE html>  
<html>  
<head>  
  <meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  <title>Ejemplo de Scriptlets</title>  
</head>  
<body>  
  <%  
    java.util.Calendar now = java.util.Calendar.getInstance();  
    int hora = now.get(java.util.Calendar.HOUR_OF_DAY);  
  %>  
  <b>Hola mundo,  
    <% if ((hora >= 6) && (hora <= 12)) {%>  
      buenos días!  
    <% } else if ((hora > 12) && (hora < 20)) {%>  
      buenas tardes!  
    <% } else {%>  
      buenas noches!  
    <% };%>  
  </b>  
</body>  
</html>
```


Un poco de arquitectura

Si analizamos la arquitectura, JSP está basada en el funcionamiento de los servlets, ya que al ser ejecutados, las páginas JSP se convierten internamente en *servlets* en tiempo de ejecución, por lo tanto se podría decir que una página JSP es en realidad un *servlet*. Una página JSP se puede utilizar utilizando dos mecánicas:

- Como componente de vista de un diseño del lado del servidor, trabajando codo a codo con clases Java Bean como modelo y con *servlets* como controlador. Posteriormente profundizaremos en la arquitectura cliente servidor.

Con JSP podemos incrustar código java y ciertas acciones predefinidas como inicialización de variables e importaciones en una página web con HTML (mezclamos contenido dinámico y contenido estático). La página pasa por una etapa de compilación y ejecución dentro del servidor para poder entregar un documento.

Las páginas compiladas, al igual que las bibliotecas java contienen el código en formato de *ByteCode* en vez de código de máquina. Al igual que otro programa en java, el archivo debe ejecutarse dentro de una máquina virtual Java (JVM) que interactúa con el sistema operativo host del servidor para proporcionar un entorno neutral para la plataforma.

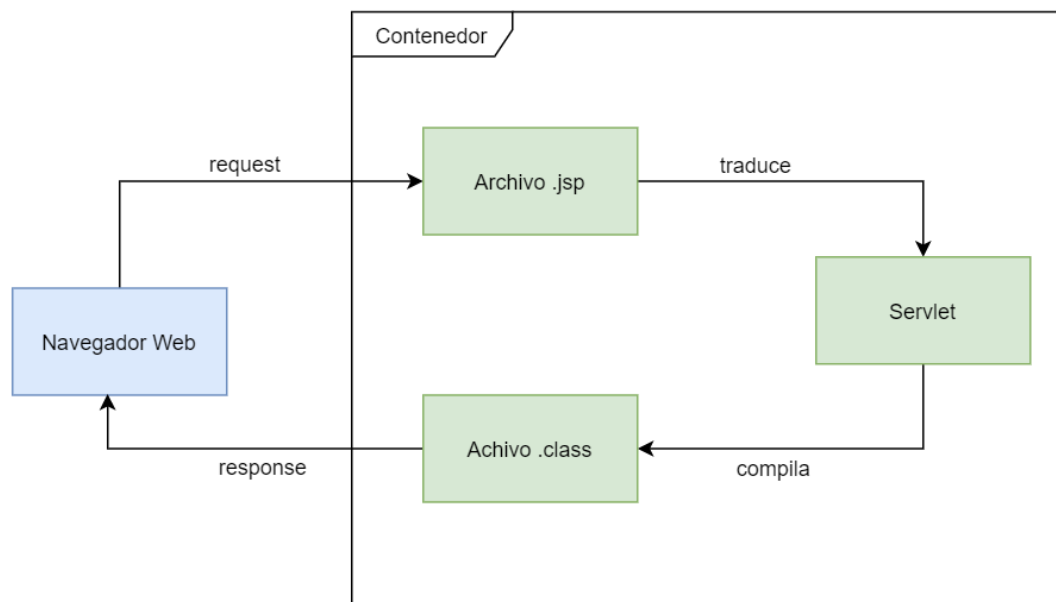


Imagen 5. Arquitectura cliente-servidor.
Fuente: Desafío Latam

El contenedor web crea objetos JSP implícitos como request, response, sesiones, aplicaciones, configuraciones, páginas html, salidas y excepciones. El motor de las *Java Servlets Page* crea estos objetos durante la fase de traducción (al igual que los servlets, las páginas JSP tienen el mismo ciclo de vida).

JSTL

Estos elementos tienen como misión proporcionar una manera fácil de mantener los componentes de una página JSP. El uso de estas etiquetas ha simplificado la tarea de los diseñadores para crear páginas web. Ahora simplemente usan la etiqueta relacionada con la tarea que deben hacer en una página JSP.

Las características más significativas de los JSTL son:

- JSTL también es JSP, siendo un conjunto complementario de este.
- Utiliza 4 librerías estándar: SQL, XML, CORE, INTERNALIZACIÓN
- JSTL define un nuevo lenguaje de expresiones llamado EL.
- Al usar una etiqueta JSTL, lo que hacemos es añadir una acción.
- Una etiqueta JSTL está delimitada por `${ }`.

Instalación de JSTL en un proyecto web

La librería JSTL está compuesta por una serie de clases empaquetadas en forma de jar. Para poder utilizarla es necesario instalar el JAR en el classpath y además importar las librerías a la carpeta lib del proyecto. Al final del procedimiento la estructura de carpetas debe lucir como la imagen siguiente.

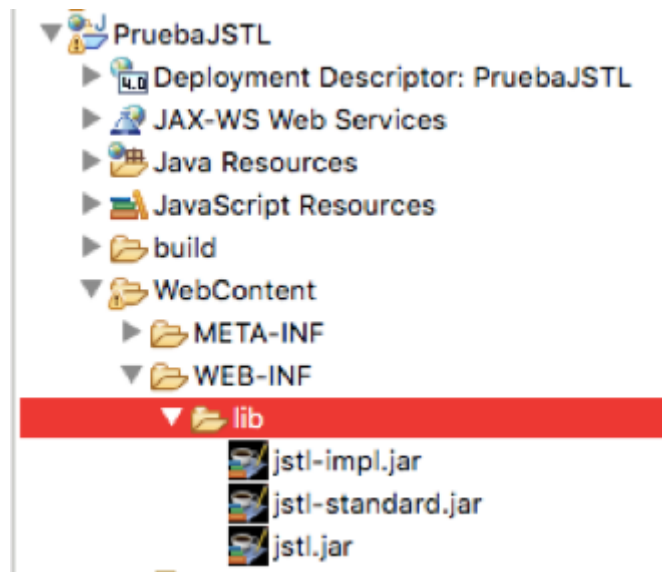


Imagen 6. Incorporando JSTL.

Fuente: Desafío Latam

Para agregar la librería al ClassPath del proyecto, primero debes ir a la opción Build Path y luego a Configure Build Path.

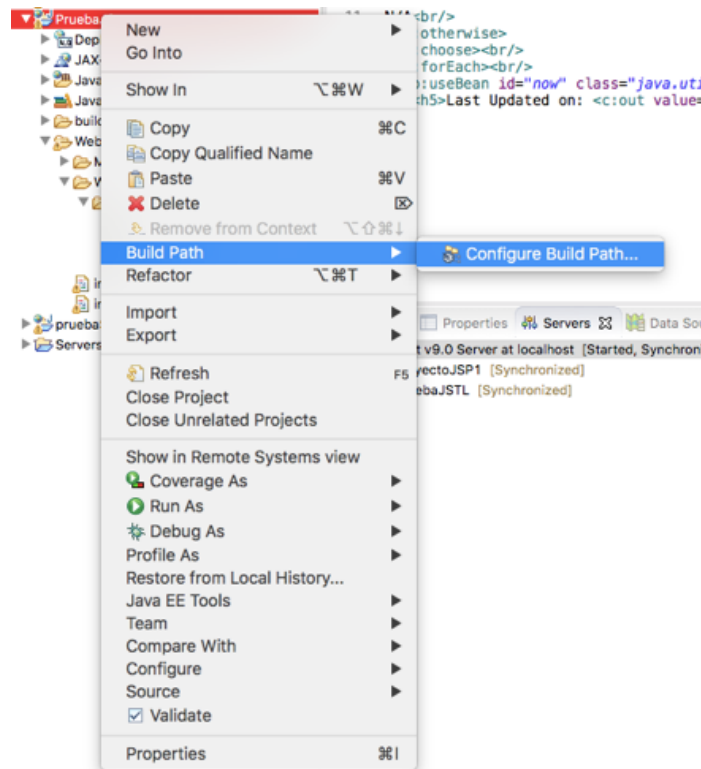


Imagen 7. Agregando la librería JSTL.

Fuente: Desafío Latam

En la pantalla siguiente correspondiente a la imagen 8, debes seleccionar la opción Add External Jars. Luego debes ubicar la carpeta de instalación de Apache Tomcat (está instalada en tu equipo, dentro de la carpeta Tomcat) y dentro de la carpeta lib agregar el jar de nombre servlet-api.jar. Este jar es necesario para poder utilizar JSTL.

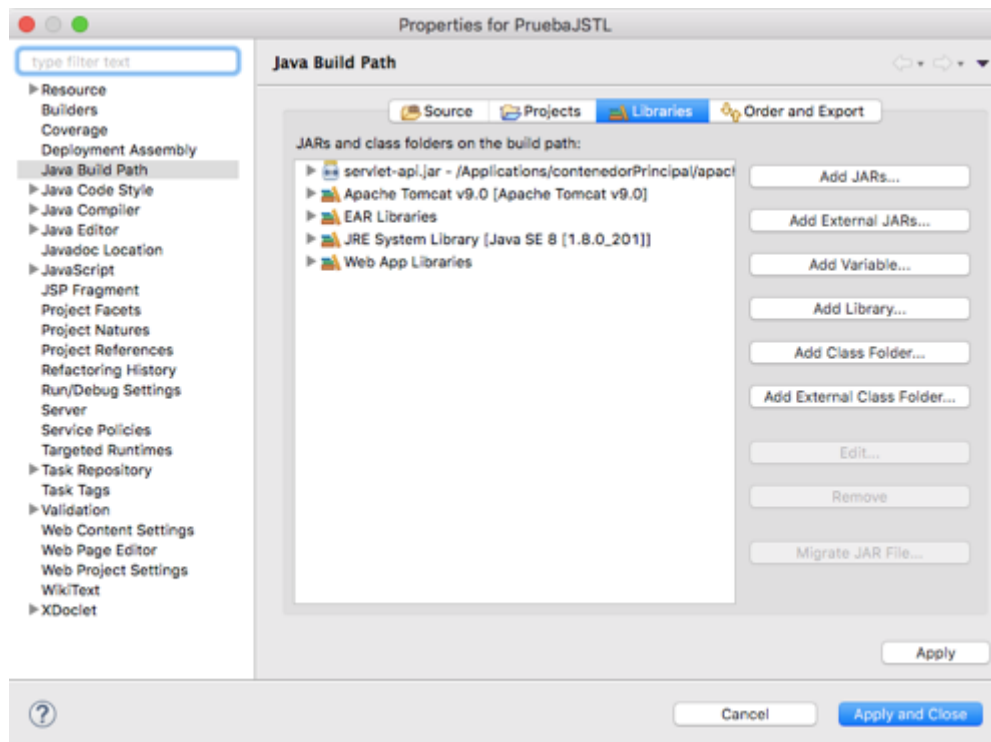


Imagen 8. Add External JARs.

Fuente: Desafío Latam

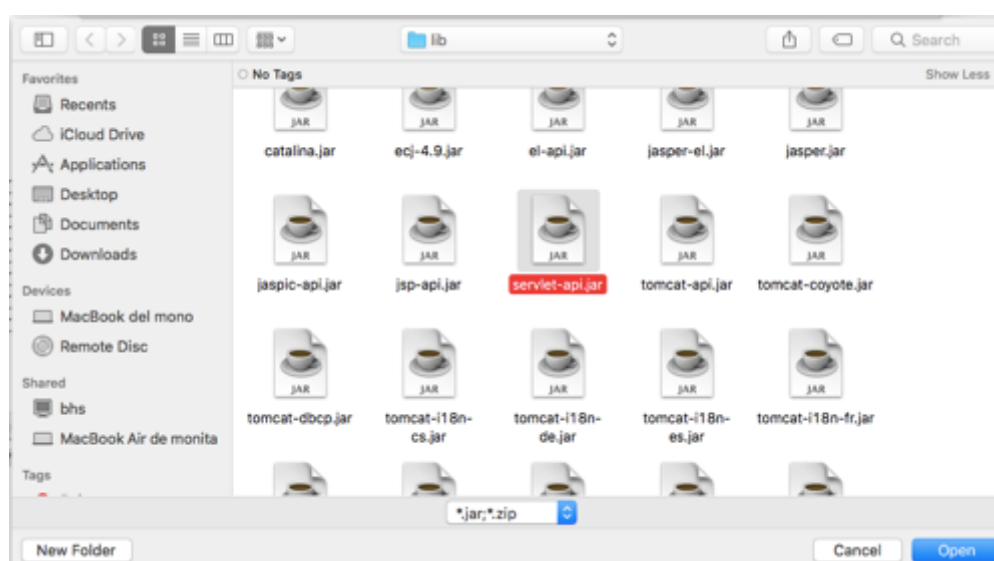


Imagen 9. Agregar la librería.

Fuente: Desafío Latam

Al seleccionar el jar, debe quedar agregado en la pantalla junto a las otras librerías.

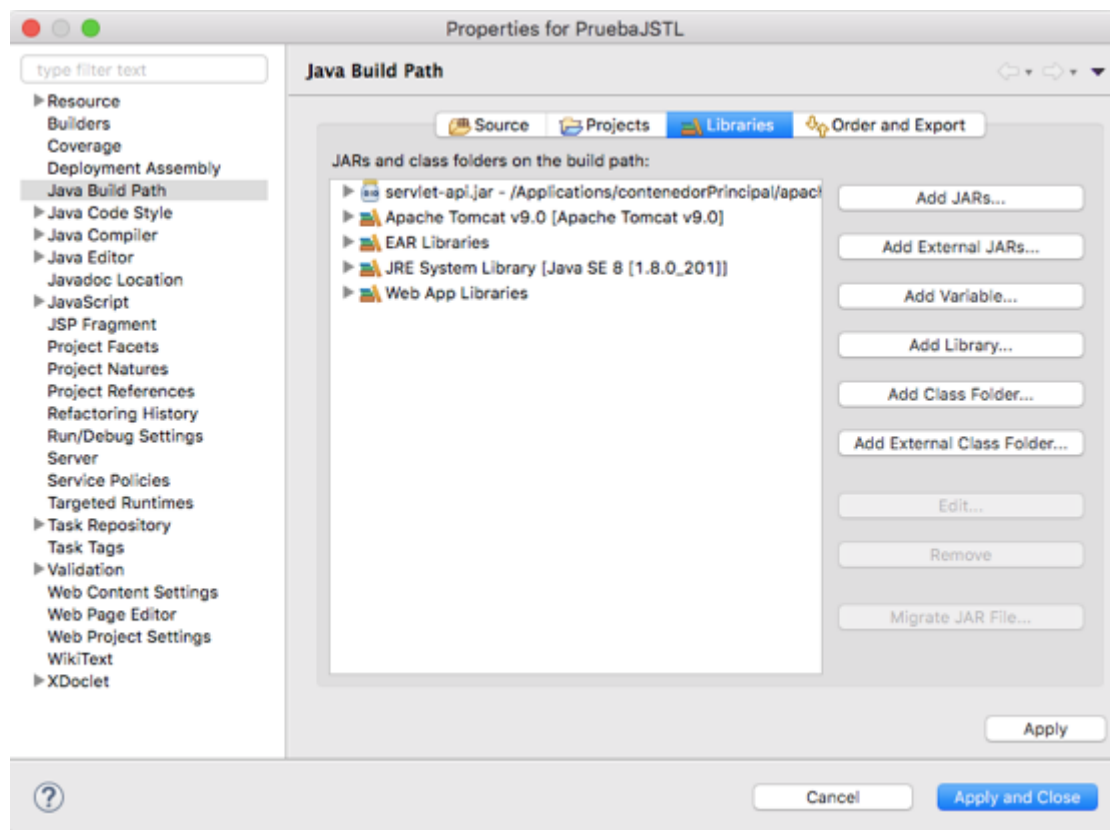


Imagen 10. Aplicar los cambios.

Fuente: Desafío Latam

Descargar las librerías necesarias

Ahora es tiempo de descargar las librerías necesarias. Para poder instalar los jar primero debemos crear una carpeta con un nombre, y dentro de ella debes alojar los siguientes jar:



Imagen 11. jar a incorporar.

Fuente: Desafío Latam

Para descargarlos, debemos entrar a la siguiente [url](#) y dentro de la página, buscar los jar:

- jstl-impl.jar
- jstl-standar.jar
- jstl.jar


40.	Download jstl-impl.jar	
41.	Download jstl-standard.jar	
42.	Download jstl.jar	

Imagen 12. Descarga de librerías.
Fuente: Desafío Latam

Debemos descomprimir los archivos y guardar los 3 jar en una carpeta, para luego volver a Eclipse y agregarlos al ClassPath. Después de instalar las librerías, se deben agregar los .jar dentro de la carpeta lib.

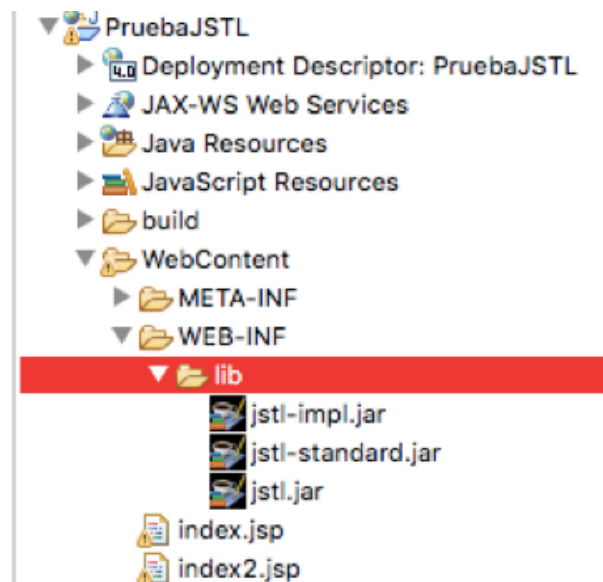


Imagen 13. Agregar los .jar a la carpeta lib.
Fuente: Desafío Latam

Al tener los archivos tanto en el classPath y en la carpeta lib, ya es posible utilizar JSTL con total libertad.

Ahora dentro de un proyecto *Dynamic Web Project* de nombre *JstlEjemplo1*, genera un archivo JSP de nombre *PrimerUsoJstl.jsp*

Lo primero que debemos hacer para utilizar JSTL, es importar las librerías necesarias, en este caso exportamos las 4 bibliotecas básicas en el archivo jsp.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

Usando la librería

Ya con las librerías importadas podemos utilizar los *tags* que queramos, en este caso utilizaremos el tag estándar `<c:out>` solamente para generar un string con los números **"1+2+3"**.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Usando JSTL</title>
</head>
<body>
  <p>Cadena de caracteres: <strong><c:out value="1+2+3"/>
</strong></p>
</body>
</html>
```

Al ejecutar el jsp nos dará como resultado:



Imagen 14. Resultado usando la librería.
Fuente: Desafío Latam

Tags JSTL

Para referenciar la librería JSTL Core en una página JSP debemos declarar la cabecera del documento como a continuación:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

Lo más significativo es el `prefix="c"`, que especifica el tipo de etiqueta y el contexto que se necesita utilizar mediante JSTL.

JSTL incluye una gran variedad de tags que engloban distintas áreas funcionales. Para segmentar estas áreas, se utilizan *namespaces*.

JSTL expone múltiples tags en estas url:

- **Core:** <http://java.sun.com/jsp/jstl/core>
- **XML:** <http://java.sun.com/jsp/jstl/xml>
- **Internationalization:** <http://java.sun.com/jsp/jstl/fmt>
- **SQL:** <http://java.sun.com/jsp/jstl/sql>
- **Functions:** <http://java.sun.com/jsp/jstl/functions>

Area	Subfunction	Prefix
Core	Variable support	c
	Flow control	
	URL management	
	Miscellaneous	
XML	Core	x
	Flow control	
	Transformation	
I18N	Locale	fmt
	Message formatting	
	Number and date formatting	
Database	SQL	sql
Functions	Collection length	fn
	String manipulation	

Imagen 15. Tabla con los múltiples tags disponibles.
Fuente: Desafío Latam

Librerías CORE

Las librerías core incluyen variables para el control de flujo, manejo de variables, administración de URLs, y algunas funciones misceláneas.

El tag de salida estándar de código es el **c:out**. Se encarga de mostrar por pantalla el valor que se le asigne en la etiqueta. Cumple la misma función que el *System.out.println* de java. En el ejercicio de configuración y prueba de JSTL ya se implementó para generar por pantalla la salida de un string.

¿Qué ventaja tiene JSTL sobre los scriptlets?

Anteriormente observamos cómo implementar un JSP y estamos empezando a comprender la sintaxis de JSTL, por lo que podemos preguntarnos ¿cuál es la ventaja real de utilizar JSTL?

Lo explicaremos a través de un sencillo ejemplo:

```
<html>
<head>
  <title>Contando de 1 a 10 en JSP con Scriptlet </title>
</head>
<body>
  <%
    for (int i = 1; i <= 10; i ++){%>
```

```
<% = i%> <br/>
<%}%>
</body>
</html>
```

Imaginemos que este sencillo código nos dio algunas dificultades de legibilidad. Si bien es cierto que los scriptlets son potentes y le agregan funcionalidades a la programación con Java, a simple vista es mucho más complejo escribir y depurar el código con scriptlets.

Otra de las grandes debilidades que viene a solventar JSTL es la reutilización de código: en el uso de scriptlets, el código no puede ser reutilizados por otros JSP, por lo que la lógica común termina siendo replicada en múltiples páginas.

Veamos cómo resolvemos este mismo desafío con JSTL:

```
<% @ taglib uri = "http://java.sun.com/jstl/core" prefix = "c"%>
<html>
  <head>
    <title>Contando de 1 a 10 en JSP con JSTL</title>
  </head>
  <body>
    <c:forEach var = "i" begin = "1" end = "10" step = "1">
      <c:out value="${i}" />
      <br />
    </c:forEach>
  </body>
</html>
```

Ejercicio guiado 1: Utilización de tag <c: out>

En este ejercicio generamos una página jsp que desplegará dos valores por pantalla, por un lado el texto "1+2+3" y debajo de él implementaremos la suma de esos valores. El resultado esperado es similar a esto:

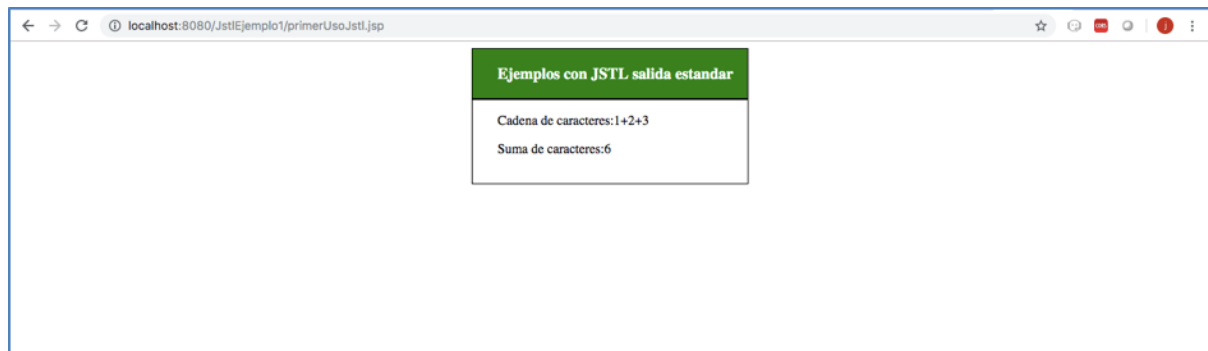


Imagen 16. Ejemplo JSTL.
Fuente: Desafío Latam

El código para implementar este ejemplo es el siguiente, archivo *primerUsoJstl.jsp*:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Usando JSTL</title>
<style>
#texto{
    margin:0 auto;
    width:300px;
    height:100px;
    border:solid 1px black;
    padding-left:30px;
}
#cabecera{
    margin:0 auto;
    width:300px;
    border:solid 1px black;
    padding-left:30px;
    background-color:green;
    color:white;
}
</style>
</head>
<body>
    <div id="cabecera">
        <h3>Ejemplos con JSTL salida estandar </h3>
    </div>
    <div id="texto">
        <p>Cadena de caracteres:<c:out value="1+2+3"/></p>
        <p>Suma de caracteres:<c:out value="\${1+2+3}"/></p>
    </div>

</body>
</html>
```

El código anterior tiene un poco de css para dar algo de formato, pero lo más importante está en div de nombre texto, donde implementamos las dos etiquetas c: para imprimir por pantalla la cadena de caracteres y en la línea siguiente, utilizando la expresión **$\${1+2+3}$** sumamos los valores.

¿Qué es un formulario web?

Pensemos un momento en una página web, con estilos y estructuras que permiten al usuario enterarse de los productos o servicios que una empresa puede ofrecer. Tal página es solo descriptiva, ya que se encarga de desplegar información y mostrarla de manera ordenada y atractiva.

Pero hay un tema pendiente el cual es la interacción con el usuario. La persona que usa la web no tiene ninguna forma de alimentar con datos al sitio web, y por consiguiente no hay forma de procesar datos ni tampoco de generar conocimientos. Aquí entran en juego los formularios web, que son el primer puente entre el usuario y las bases de datos, o la lógica real de la empresa.

Estos elementos web se componen de componentes que capturan los datos del usuario mediante cajas de texto, listas desplegables, combos de opciones y una serie de componentes html, y gracias a ellos una simple página web estática se puede convertir en un real sistema web (acompañado de algún lenguaje de programación por el lado del servidor y una base de datos obviamente).

Ejercicio guiado 2: Envío de valores entre jsp

En este ejercicio crearemos un simple formulario con la entrada de dos valores, nombre y apellido, que luego de ser enviados serán recibidos por un jsp con las etiquetas **c: out**.

El resultado del ejercicio será el siguiente:



Imagen 17. Resultado envio de valores entre jsp.
Fuente: Desafío Latam

Y al procesar el formulario con un nombre y un apellido, el segundo jsp será el siguiente:



Imagen 18. Resultado al recibir los parámetros.
Fuente: Desafío Latam

El código es el siguiente para el archivo formulario.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Usando JSTL</title>
<style>
//codigo de estilo
</style>
</head>
<body>
  <div id="cabecera">
    <h3>FORMULARIO DE ENVIO</h3>
  </div>
  <div id="texto">
    <form action="primerUsoJstl.jsp" method="POST">
      <label>Nombre</label>
      <input type="text" placeholder="Ingrese nombre" id="nombre"
name="nombre">
      <label>Apellido</label>
      <input type="text" placeholder="Ingrese apellido"
id="apellido" name="apellido">
      <input type="submit" value="enviar">
    </form>
  </div>
</body>
</html>
```

Vemos la implementación de un formulario con un atributo *action="primerUsoJstl.jsp"* y con método POST. *Action* indica a donde se irán los datos al momento de presionar el botón de envío, y el método indica el tipo de envío, en este caso se envía por POST para que los datos no se vean en la url.

El css está para dar algo de formato a la pantalla.

El código para el *primerUsoJstl.jsp* que recibe los datos es el siguiente:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Usando JSTL</title>
<style>
//codigo de style
</style>
</head>
<body>
  <div id="cabecera">
    <h3>FORMULARIO DE ENVIO</h3>
  </div>
  <div id="texto">
    <p>Nombre: <c:out value="${param.nombre}"/></p>
    <p>Apellido: <c:out value="${param.apellido}"/></p>
  </div>
</body>
</html>
```

Como se ve en el código, para recibir valores mediante jstl y la etiqueta **c:out** se utiliza el formato **\${param.nombre}** en donde param es palabra reservada y .nombre es el nombre del name del input del primer formulario.

Ejercicio guiado 3: Envío de datos entre formularios

Para completar las prácticas de envío de valores entre formularios, vamos a generar un pequeño formulario en un archivo jsp, el cual simplemente enviará sus valores a otro formulario jsp y los desplegará en la pantalla. En esta oportunidad como un “extra” añadiremos un framework de presentación llamado bootstrap, solo para que los formularios sean más atractivos.

Para utilizar bootstrap, dentro de la etiqueta **<head></head>** pegar las siguientes líneas, corresponden a los archivos necesarios para utilizar bootstrap, el css y el js.

```
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min
.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy60rQ6VrjIEaFf/nJGzIxFDsf4x0
xIM+B07jRM" crossorigin="anonymous"></script>
```



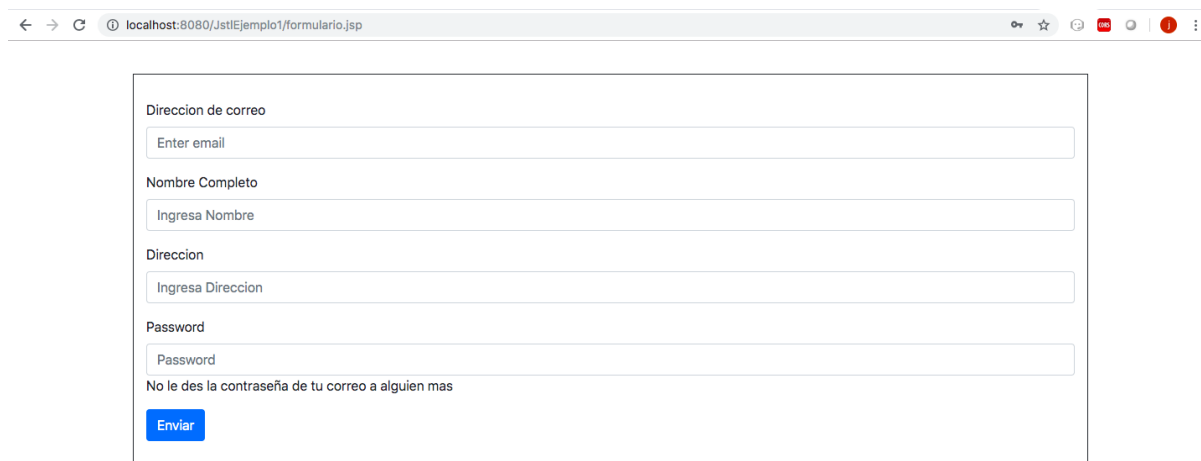
```
<link  
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.m  
in.css" rel="stylesheet"  
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQU0hcWr7x9Jv  
oRxT2MZw1T" crossorigin="anonymous">
```

Ya estás en condiciones de utilizar bootstrap en tus páginas web. Como se puede ver, es perfectamente compatible con jsp, ya que al final es html.

A continuación, veremos el ejemplo funcionando. Los formularios en JSP como habíamos dicho son hechos con html, y gracias a los tags jstl es posible aplicar estilos de forma más fácil que con los antiguos scriptlet. En este caso generamos un formulario con los campos de nombre completo, correo, dirección, celular y un botón enviar.

Este formulario enviará los datos ingresados por el usuario a un segundo jsp de nombre procesa.jsp, que se encargará de recopilar los valores ingresados por el usuario y desplegarlos en la pantalla de forma ordenada. Todo esto utilizando tags jstl de la librería core.

El primer jsp de nombre formulario debe lucir de esta forma:



← → ↻ local:localhost:8080/JstlEjemplo1/formulario.jsp

Direccion de correo
Enter email

Nombre Completo
Ingresa Nombre

Direccion
Ingresa Direccion

Password
Password

No le des la contraseña de tu correo a alguien mas

Enviar

Imagen 19. Formulario de datos.
Fuente: Desafío Latam

Como se ve es un formulario común y corriente. Al enviar los valores mediante el botón, se desplegará otro jsp que recibirá el formulario y lo desplegará en pantalla:

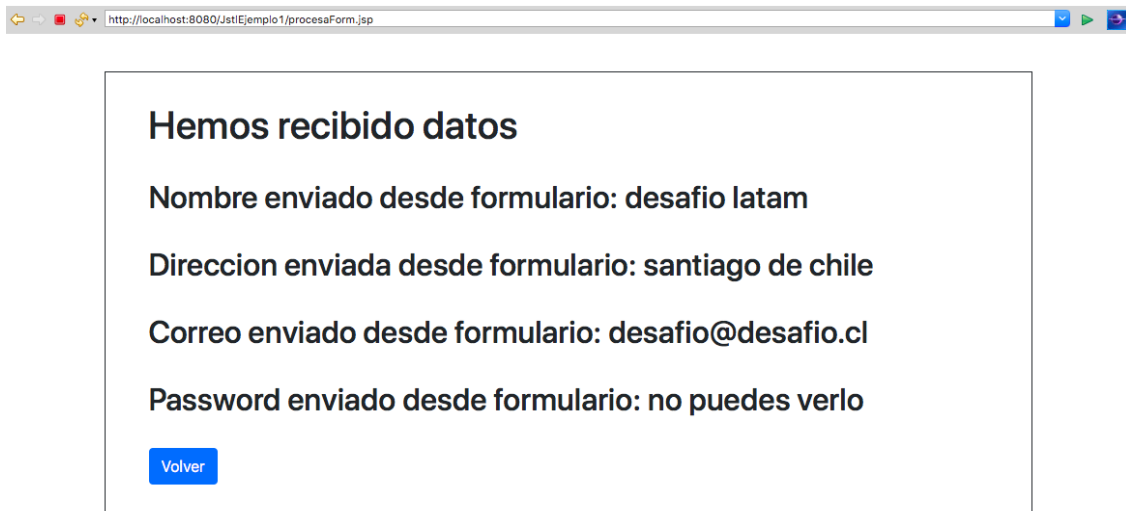


Imagen 20. Resultado del envío de datos.

Fuente: Desafío Latam

El código del formulario.jsp es el siguiente:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Usando JSTL</title>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min
.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy60rQ6VrjIEaFf/nJGzIxFDs4x0
xIM+B07jRM" crossorigin="anonymous"></script>
<link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.m
in.css" rel="stylesheet"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQU0hcWr7x9Jv
oRxT2MZw1T" crossorigin="anonymous">
<style>
.container{
margin-top: 39px;
```

```
        border: solid 1px;
        padding-bottom: 30px;
    }
</style>
</head>
<body>
<div class="container">
    <div class="row">
        <div class="col">
            <form action="procesaForm.jsp" method="POST">
                <div class="form-group">
                    <label for="exampleInputEmail">Direccion de
correo</label>

                    <input type="email" class="form-control"
id="correo" name="correo" aria-describedby="emailHelp"
placeholder="Enter email" required>
                </div>
                <div class="form-group">
                    <label for="exampleInputEmail">Nombre
Completo</label>

                    <input type="text" class="form-control"
id="nombre" name="nombre" placeholder="Ingresa Nombre" required>
                </div>
                <div class="form-group">
                    <label
for="exampleInputEmail">Direccion</label>

                    <input type="text" class="form-control"
id="direccion" name="direccion" placeholder="Ingresa Direccion"
required>
                </div>
                <div class="form-group">
                    <label
for="exampleInputPassword">Password</label>

                    <input type="password" class="form-control"
id="pass" name="pass" placeholder="Password" required>
                    No le des la contraseña de tu correo a
alguien mas

                </div>
                <button type="submit" class ="btn btn-primary">Enviar
</button>

            </form>
        </div>
    </div>
</div>
```

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

Importante recordar que de todo el código que se ve, lo importante para enviar un formulario a algún otro archivo que lo procese es el atributo action del form, que declara el archivo que recibirá los valores. El botón input se encarga de ejecutar las acciones necesarias para llegar al archivo.

El archivo procesa.jsp solo se encarga de recibir los valores enviados desde el formulario.jsp mediante las sentencias JSTL marcadas con amarillo. Como se puede ver la mecánica de envío de parámetros es bastante simple.

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3 <%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
4 <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
5 <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="UTF-8">
10 <title>Usando JSTL</title>
11 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy60rQ6VrjIEa="
12 <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-ggOyR0iXCbMQv3Xipma3=
13 <style>
14 .container{
15     margin-top:39px;
16     border:solid 1px; black;
17     padding-top : 30px;
18     padding-bottom: 30px;'
19 }
20 #datos{
21     padding-left:30px;
22 }
23 </style>
24 </head>
25 <body>
26 <div class="container">
27 <div id="datos">
28 <h1>Hemos recibido datos</h1>
29 <br>
30 <h2>Nombre enviado desde formulario: <c:out value="${param.nombre}" /></h2>
31 <br>
32 <h2>Direccion enviada desde formulario: <c:out value="${param.direccion}" /></h2>
33 <br>
34 <h2>Correo enviado desde formulario: <c:out value="${param.correo}" /></h2>
35 <br>
36 <h2>Password enviado desde formulario: no puedes verlo</h2>
37 <br>
38 <a href="formulario.jsp"><button type="button" class="btn btn-primary">Volver</button></a>
39 </div>
40 </div>
41 </body>
42 </html>
```

Imagen 21. Código del archivo procesa, donde se reciben los datos.

Fuente: Desafío Latam

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<title>Usando JSTL</title>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min
.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy60rQ6VrjIEaFf/nJGzIxFDsf4x0
xIM+B07jRM" crossorigin="anonymous"></script>
<link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.m
in.css" rel="stylesheet"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9Jv
oRxT2MZw1T" crossorigin="anonymous">
<style>
.container{
    margin-top: 39px;
    border: solid 1px;
    padding-bottom: 30px;
}
#datos{
    padding-left:30px;
}
</style>
</head>
<body>
<div class="container">
    <div id="datos">
        <h1>Hemos recibido datos</h1>
        <br>
        <h2>Nombre enviado desde formulario: <c:out
value="${param.nombre}"/></h2>
        <br>
        <h2>Direccion enviada desde formulario: <c:out
value="${param.direccion}"/></h2>
        <br>
        <h2>Correo enviado desde formulario: <c:out
value="${param.correo}"/></h2>
        <br>
        <h2>Password enviado desde formulario: no puedes verlo</h2>
        <br>
        <a href="formulario.jsp"><button type="button" class="btn
btn-primary">Volver</button></a>
    </div>
</div>
</body>
```

```
</html>
```

Como se ha visto en las prácticas, con el uso de jstl y jsp es muy simple generar una comunicación fluida entre elementos de una aplicación web. Por el momento solo se reviso lo más básico de los tags JSTL, ya que además de permitir el envío de datos entre formularios jsp, también cuentan con iteraciones, decisiones, tags de manejo de información y más, pero para comenzar a entender las arquitecturas y la comunicación entre componentes es un muy buen comienzo.