

# Java Server Page

<b>Java Server Page</b>	<b>1</b>
¿Qué aprenderás?	2
Introducción	2
Conociendo la tecnología JSP	3
Hola mundo JSP	5
Proyecto 1 JSP Básico	10
Proyecto 2 JSP Básico	14
Beneficios para los desarrolladores	16
JSP vs Servlets	17
Relación entre JSP y Servlets	18
Ventajas de los JSP frente a otras tecnologías	18
Elementos de los JSP	19
Scriptlets	19
Tipo 1 <code>&lt;% %&gt;</code>	19
Tipo 2 <code>&lt;%= %&gt;</code> Expresiones	21
Tipo 3 <code>&lt;%! %&gt;</code> Declaraciones	22



**¡Comencemos!**

## ¿Qué aprenderás?

- Conocer la tecnología JSP.
- Entender las diferencias entre JSP v/s Servlets.
- Utilizar los elementos JSP Scriptlets, Expresiones y Declaraciones.

## Introducción

En el mundo del desarrollo JEE, la capa de cliente es un pilar fundamental que sustenta el uso prolongado de los clientes hacia nuestras aplicaciones. En esta capa, se trabaja la tecnología JSP (*Java Server Page*) la cual provee al programador una mecánica de trabajo ágil, rápida y simple para presentar datos al usuario.

En esta unidad se estudiarán los fundamentos de esta tecnología, sus características y sobre todo su uso en el desarrollo de sistemas empresariales.

## Conociendo la tecnología JSP

*Java Server Page (JSP)* es la tecnología que provee una vía rápida y simple para crear contenido web dinámico. Provee herramientas para que los desarrolladores puedan construir páginas web generadas dinámicamente basadas en el lenguaje de marcado HTML, XML u otro tipo de documentos. Fue lanzado en el año 1999 por Sun Microsystems.

JSP tiene muchas similitudes con otras tecnologías como lo son PHP o ASP, pero implementando el lenguaje de programación Java.

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 construir 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. En capítulos posteriores profundizaremos en la arquitectura cliente-servidor.
- Como un componente independiente.

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.

Los JSP se utilizan para generar documentos HTML y XML, pero a través del uso de *OutputStream*, también son capaces de entregar otros tipos de datos.

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).

La imagen 1 muestra el contexto de trabajo de los JSP, los cuales al igual que los servlets viven dentro del contenedor de aplicaciones y tal como se explicó anteriormente pueden implementarse como un elemento independiente, o también como parte de una arquitectura que trabaja junto a uno o varios servlets y clases java.

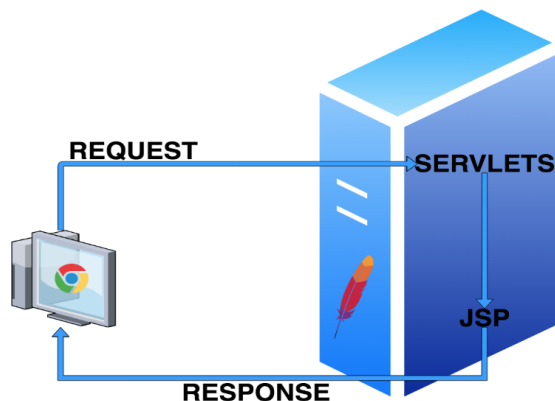


Imagen 1. Java Server Page.  
Fuente: Desafío Latam

La imagen 2 muestra el segundo caso de implementación, el cual solamente consta de una página JSP que recibe el request del usuario, procesa la información y revuelve la respuesta en formato HTML.

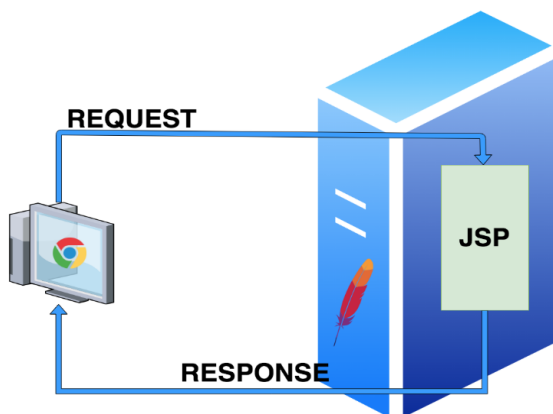


Imagen 2. Flujo de trabajo JSP.  
Fuente: Desafío Latam

Se puede apreciar que el flujo de trabajo comienza con una petición desde el equipo del cliente que solicita información al servidor y el primer elemento que recibe la petición es un servlet, que como se estudió en unidades anteriores es una clase java con características especiales. El *servlet* se encarga de procesar la información (hacer un cálculo, ordenar palabras, conectarse a otras clases, lo que quieras) para luego enviar la información procesada a una página JSP, la cual despliega la información al usuario en formato HTML.

## Hola mundo JSP

Luego de repasar las características de los JSP, se implementará el primer ejemplo de creación de una página JSP para lograr ver su estructura básica. Creamos en eclipse un nuevo *Dynamic Web Project* de nombre *HolaMundo.jsp*.

Para crear un nuevo proyecto en eclipse, basta con seleccionar con botón derecho sobre el área de proyecto y seleccionar *new->dynamic web project*.

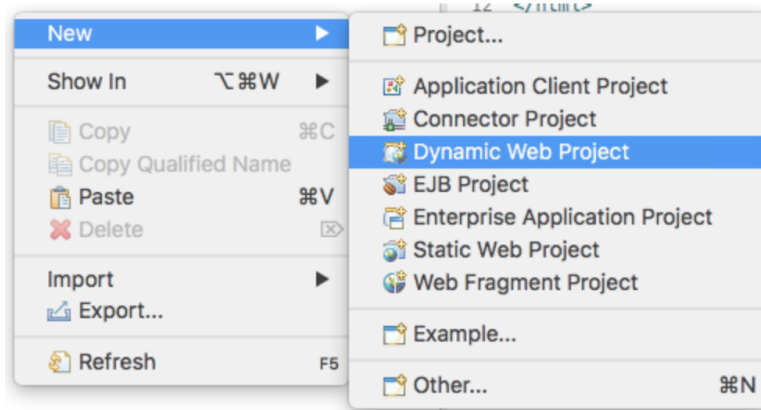


Imagen 3. Creación de un nuevo proyecto con botón derecho.

Fuente: Desafío Latam

La segunda forma de crear un nuevo proyecto es pinchando sobre el menú *File->new->Dynamic Web Project* y seguir las mismas instrucciones del wizard.

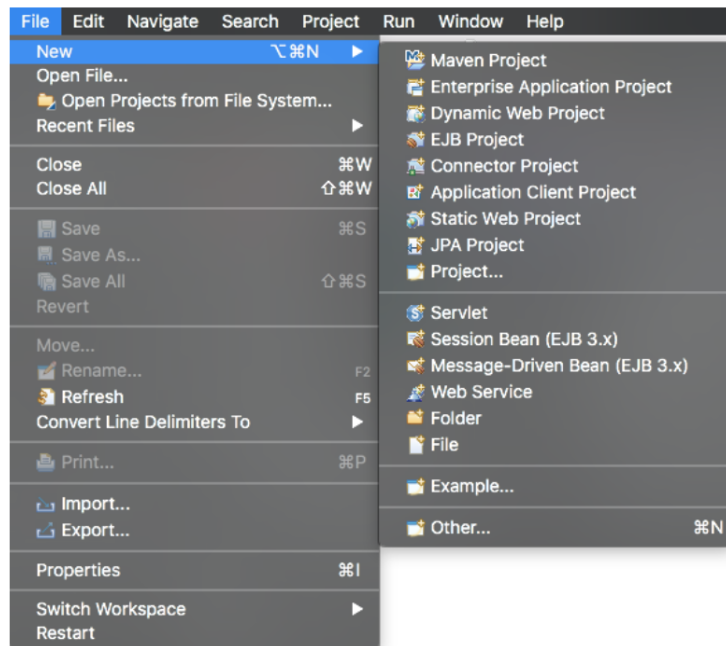


Imagen 4. Crear un nuevo proyecto mediante menú.

Fuente: Desafío Latam

Ahora configuramos el nuevo proyecto web dinámico siguiendo las instrucciones.

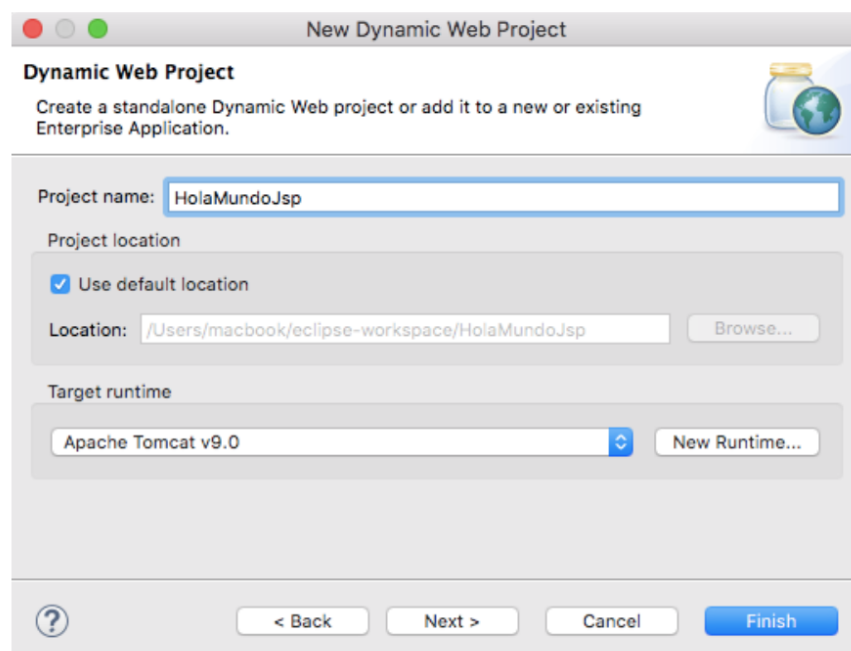


Imagen 5. Crear Dynamic Web Project.

Fuente: Desafío Latam

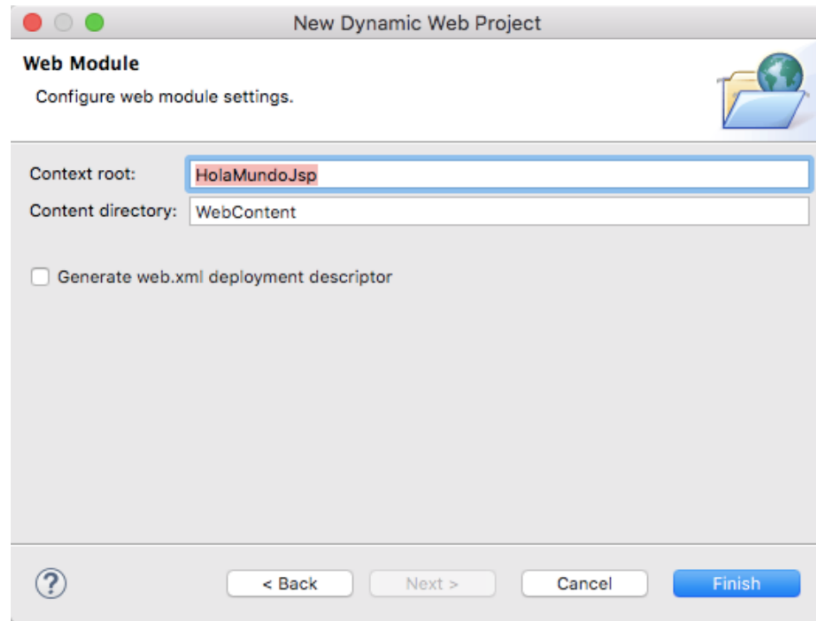


Imagen 6. Creación de Dynamic Web Project.  
Fuente: Desafío Latam

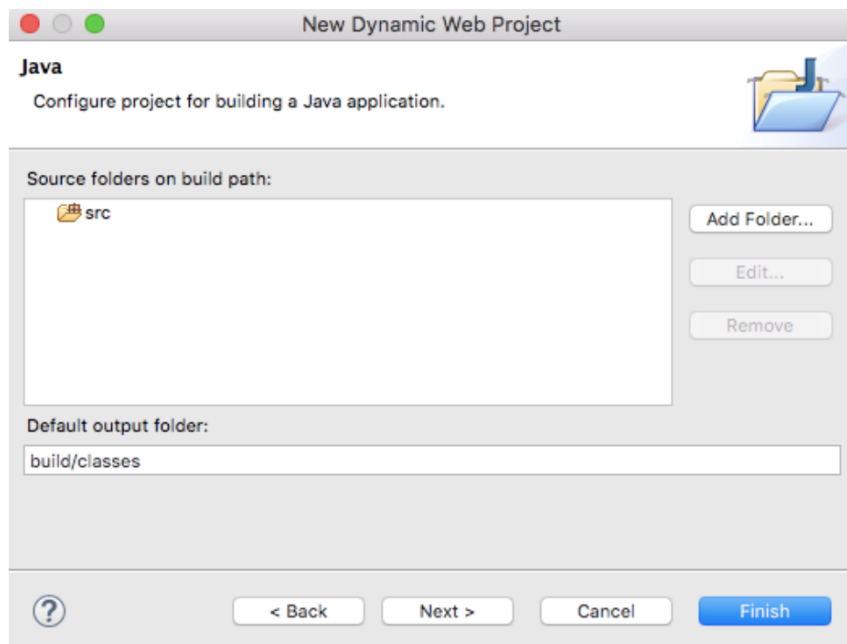


Imagen 7. Configuración Proyecto Dynamic Web Project.  
Fuente: Desafío Latam

Al finalizar el proyecto, crear un nuevo archivo jsp pinchando botón derecho sobre el proyecto.

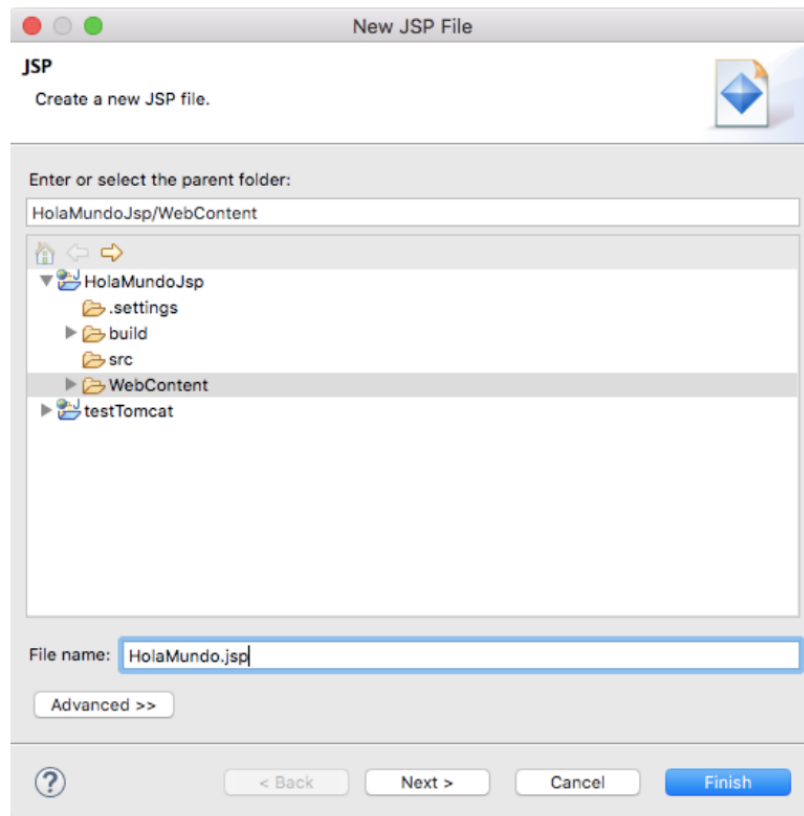


Imagen 8. Creación del nuevo archivo JSP.  
Fuente: Desafío Latam



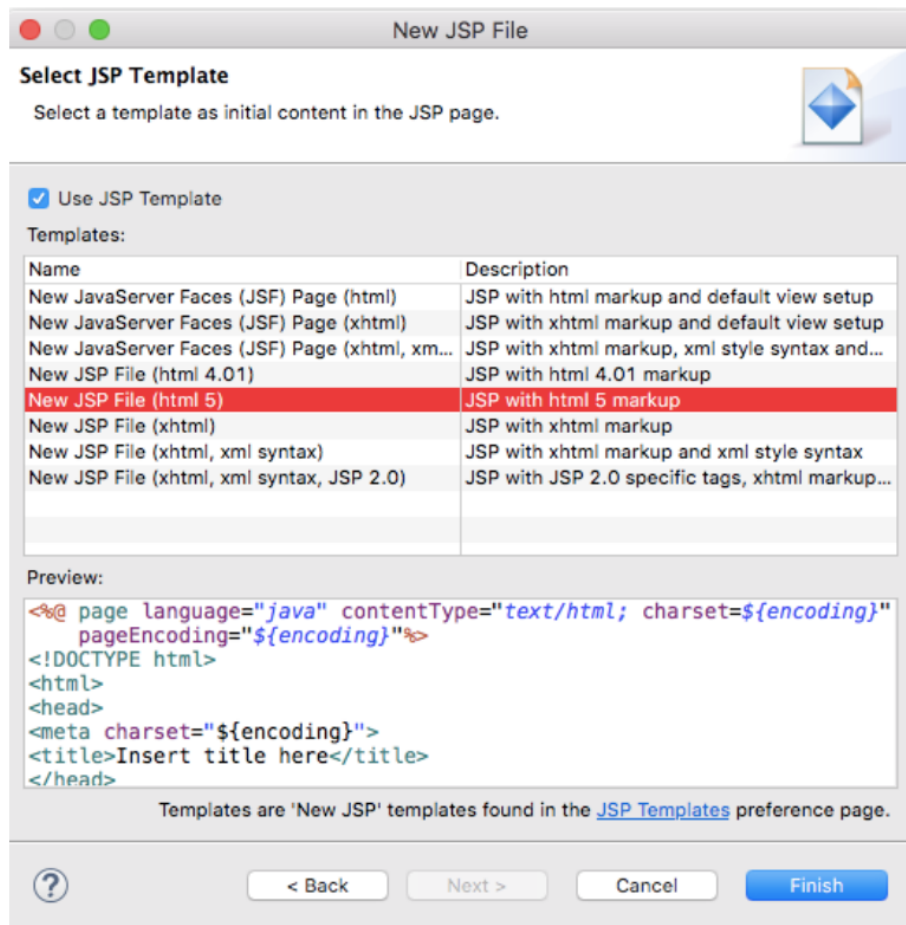


Imagen 9. Creación de archivo JSP.

Fuente: Desafío Latam

Al finalizar el procedimiento, se obtiene el proyecto web ya configurado.

## Proyecto 1 JSP Básico

Vamos a generar nuestra primera página jsp en un nuevo proyecto *Dynamic Web Project* de nombre *Patrones\_Diseño\_Sesion1\_ejemplo001*. Una página JSP en esencia es una página con código HTML, pero tiene una pequeña diferencia en su declaración. Si observamos el código veremos que:

### Declaración de la página 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>Hola Mundo JSP</title>
</head>
<%@ page import="java.util.Date"%>
<body>
    <h3>Bienvenidos</h3>
    <br>
    <strong>La fecha actual es:</strong>
    <%=new Date()%>
</body>
</html>
```

En la primera línea vemos una directiva que indica el lenguaje java en la etiqueta del mismo nombre, el *content type* en html y la codificación de caracteres mientras el resto del documento es solo html puro.

La página de ejemplo imprime por salida HTML la fecha actual, utilizando la conocida clase *Date* de java. Una página JSP es capaz de desplegar código java de forma dinámica lo que significa que no está sólo destinada a mostrar elementos estáticos, sino que su función es permitir al programador generar lógicas complejas y variadas formas de comunicación entre capas.

Veamos el ejemplo de una página JSP que realmente está cumpliendo su función. Crear un proyecto web dinámico de nombre *Patrones\_Diseño\_Sesion1\_ejemplo002*. Si en este ejemplo ve tags o información que no entiendes es normal, solo es para demostrar el funcionamiento real de un JSP.

### Contenido Dinámico.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>The real power of JSP</title>
</head>
<body>
    <h2>Bienvenido al desafio :P</h2>
    <%
        out.println("<br/>Tu dirección IP es " +
request.getRemoteAddr());
        String userAgent = request.getHeader("user-agent");
        String browser = "desconocido";
        String protocolo = "Estas usando el protocolo " +
request.getProtocol();
        String saltoLinea = "<br>";
        out.print("<br/> estas navegando con ");
        if (userAgent != null) {
            if (userAgent.indexOf("MSIE") > -1) {
                browser = "MS Internet Explorer";
            } else if (userAgent.indexOf("Firefox") > -1) {
                browser = "Mozilla Firefox";
            } else if (userAgent.indexOf("Opera") > -1) {
                browser = "Opera";
            } else if (userAgent.indexOf("Chrome") > -1) {
                browser = "Google Chrome";
            } else if (userAgent.indexOf("Safari") > -1) {
                browser = "Apple Safari";
            }
        }
        out.println(browser);
        out.println(saltoLinea);
        out.println(protocolo);

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

El código anterior corresponde al archivo *ContenidoDinamico.jsp* el cual tiene ciertas particularidades:

1. Cuenta con los tags `<% %>` que delimitan el sector que puede ser utilizado para trabajar con código java puro. Estos tags son conocidos como scriptlets (luego se hablará de ellos).
2. Se genera la implementación con código java que indica qué tipo de navegador está usando el cliente y la dirección *ip* de la máquina.

Internamente el intérprete del contenedor tomcat toma todo el código e internamente genera un *servlet* para que sea capaz de compilar y generar el resultado del código. La buena noticia es que nosotros no tenemos que preocuparnos del *servlet*, solo programamos en java puro.

No se detalla el código a fondo aunque es autoexplicativo y no necesita mucha introducción. Lo importante aquí es que se utilizó una página JSP la cual contiene en su interior código java usado para dar lógica dinámica al sitio. No fue necesario implementar ninguna otra clase java en el proyecto ni tampoco servlets, lo cual es una de las características de esta tecnología, y además es posible diferenciar bien el código de lógica java del código html.

Para obtener el tipo de navegador y la ip del ejemplo anterior se utilizaron los métodos de la clase *HttpServletRequest*, *getHeader* y *getRemoteAddr*. Estos métodos están explicados en la API de java servlets y para poder conocerlas hay que ir a esa dirección y mirarlas.

La dirección es:

<https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpServletRequest.html>

The screenshot shows the Java API documentation for the `HttpServletRequest` interface. The page title is "Interface HttpServletRequest" under the package `javax.servlet.http`. It lists superinterfaces as `ServletRequest` and implementing classes as `HttpServletRequestWrapper`. The interface is defined as `public interface HttpServletRequest extends ServletRequest`. A description states: "Extends the `ServletRequest` interface to provide request information for HTTP servlets. The servlet container creates an `HttpServletRequest` object and passes it as an argument to the servlet's service methods (`doGet`, `doPost`, etc)."

Metadata includes: Version: 5, Author: Various.

Field Summary	
static java.lang.String	<code>BASIC_AUTH</code> String identifier for Basic authentication.
static java.lang.String	<code>CLIENT_CERT_AUTH</code> String identifier for Client Certificate authentication.
static java.lang.String	<code>DIGEST_AUTH</code> String identifier for Digest authentication.
static java.lang.String	<code>FORM_AUTH</code> String identifier for Form authentication.

Imagen 10. API Java.  
Fuente: Desafío Latam

En la sección *Field Summary* se listan los distintos métodos de *HttpServletRequest*. Intenta encontrar en la lista los métodos que se utilizaron en el ejemplo, lee su descripción y comprende que nadie se sabe todo de memoria, para esto está la documentación. Si se necesitaba saber cual es el navegador que utiliza el usuario, basta con consultar esta api, buscar el método que obtiene la dirección *ip* y utilizarlo.

Si ya encontraste los métodos y sus descripciones, intenta imprimir en la misma pantalla debajo del nombre del navegador el protocolo de comunicación utilizado.

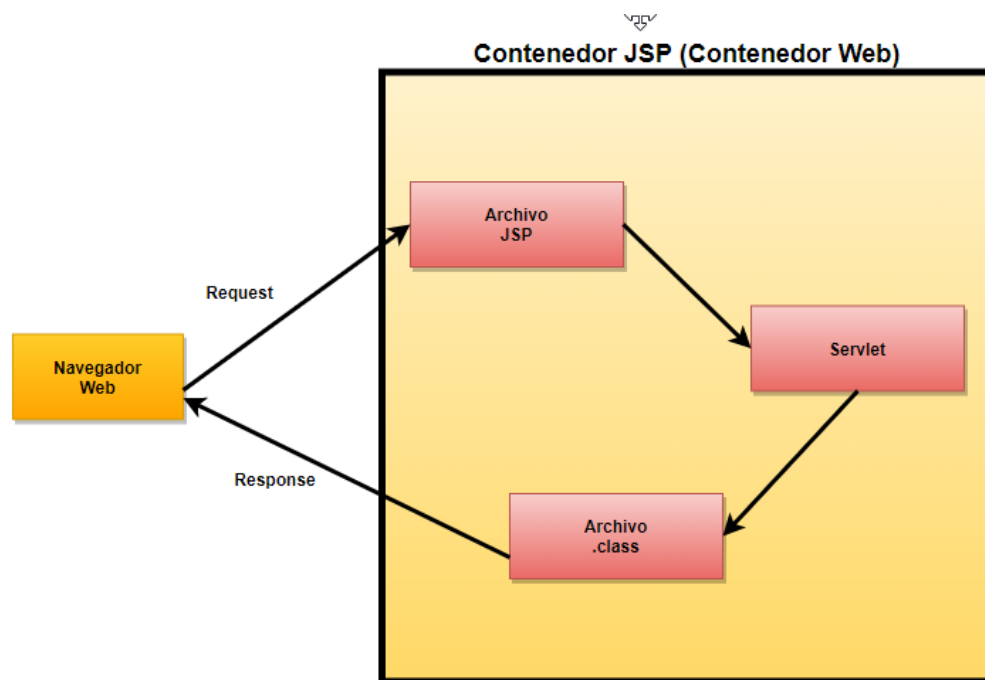


Imagen 11. Arquitectura JSP.  
Fuente: Desafío Latam

## Proyecto 2 JSP Básico

Continuando con la introducción a los sitios JSP, se implementará una página básica que simplemente imprimirá en una tabla una lista de marcas de autos, la cual previamente se creará mediante código java. Para implementar este ejercicio en HTML puro habría que generar la tabla y en cada celda imprimir por separado cada marca de auto. Si bien funcionará bien tenemos dos problemas:

- Repetición de código.
- Datos "Hardcoreados" (Código escrito en html que es estático, nunca variará y no representa ninguna información relevante.

Si se utiliza un lenguaje de programación que se encargue de entregar todos los elementos listos para que la tabla se genere de forma automática reducimos las líneas de código html y permite que la tabla se alimente de forma dinámica.

Crear un nuevo proyecto *Dynamic Web Project* de nombre *Patrones\_Diseño\_Sesion1\_ejemplo003* y un jsp de nombre *HolaMundo.jsp*. El código a continuación implementa la funcionalidad.

### ListadoAutos.jsp

```
<%@page import="java.util.ArrayList"%>
<%@page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.util.*" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<style>
    table,thead,tbody,tr,td{
        border: 1px solid;
        collapse:collapse;
        text-align: center;
        background-color:orange;
    }

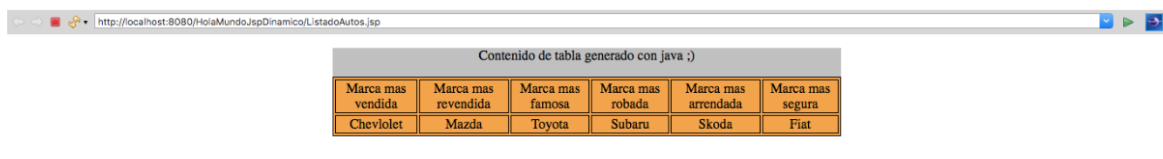
    #contenido{
        width: 600px;
```

```
        height: 300px;
        margin: 0 auto;
        background-color: silver;
        text-align: center;
    }
</style>
</head>
<body>
    <%
        //declaracion de una lista de marcas de vehiculos
        List<String> autos = new ArrayList<String>();
        autos.add("Chevrolet");
        autos.add("Mazda");
        autos.add("Toyota");
        autos.add("Subaru");
        autos.add("Skoda");
        autos.add("Fiat");
    %>
    <!-- comienza el HTML -->
    <div id="contenido">
        <p>Contenido de tabla generado con java ;</p>
        <table>
            <thead>
                <tr>
                    <td>Marca mas vendida</td>
                    <td>Marca mas revendida</td>
                    <td>Marca mas famosa</td>
                    <td>Marca mas robada</td>
                    <td>Marca mas arrendada</td>
                    <td>Marca mas segura</td>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <%
                        for(int i = 0; i < autos.size(); i
                    ++){
                        out.println("<td>" +
                            autos.get(i)+ "</td>");
                    }
                <%>
            </tr>
        </tbody>
    </table>
```

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

Como se aprecia, el área de la tabla html se alimenta mediante una lista, y es generada con el ciclo for.

El resultado es simplemente una tabla, pero la gran diferencia con una página estática es su implementación y su naturaleza dinámica.



Marca mas vendida	Marca mas revendida	Marca mas famosa	Marca mas robada	Marca mas arrendada	Marca mas segura
Chevrolet	Mazda	Toyota	Subaru	Skoda	Fiat

Imagen 12. Resultado de clase.  
Fuente: Desafío Latam

## Beneficios para los desarrolladores

Las ventajas de utilizar JSP en una aplicación empresarial para los desarrolladores se pueden enumerar en:

- JSP no es solo una página HTML, sino que es una completa implementación de todas las características de un servlet. En una página JSP se pueden utilizar todas las funcionalidades de la biblioteca de clases java, además de usar tags de funcionalidades y desplegar información al usuario.
- Con JSP es mucho más fácil hacer la separación entre código de programación java y código de presentación HTML (En servlets, generar código html es difícil).
- Usando JSP ya no es necesario recompilar y construir el proyecto cada vez que se haga un cambio. En términos prácticos esto significa que si modificas un título a tu JSP, basta con guardar el cambio y refrescar el navegador. Esto aumenta la rapidez de desarrollo en comparación con los servlets.
- Con JSP se utiliza menos código para hacer cosas similares que en un servlet.



## JSP vs Servlets

Estos dos componentes de la tecnología java, generan contenido dinámicamente desde un servidor con contenedor de servlets a una página web, pero de una manera distinta. Ambas tecnologías se complementan entre sí, así que no es posible elegir un camino u otro, ya que juntas alcanzan su mayor potencial. La mayor diferencia que encontramos es que un servlet es un programa java simple con una herencia de clases característica (HttpServlet) que se incrusta en una página web simple.

El programa java se ejecuta en el lado del servidor y muestra el resultado en el navegador incrustado html que se envía al navegador. El servlet incrusta HTML en el código java a través de las declaraciones `out.println`.

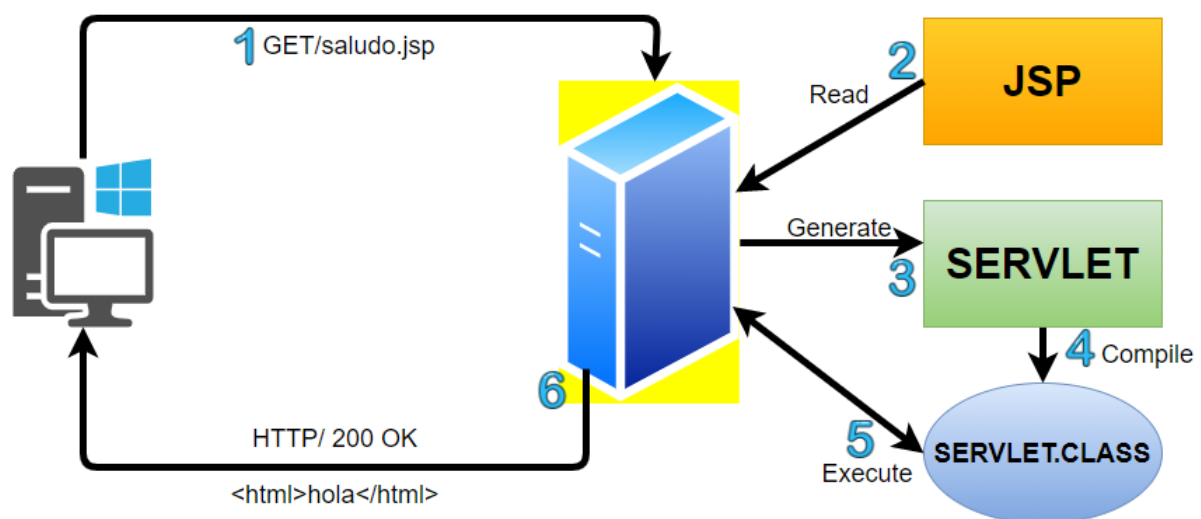


Imagen 13. Interacción Cliente-jsp-servlet.  
Fuente: Desafío Latam

## Relación entre JSP y Servlets

Por regla general, las páginas JSP y los archivos Servlets están muy relacionados entre sí. Cada JSP se compila primero en un servlet antes de poder hacer cualquier cosa. Cuando se utiliza una primera llamada a JSP, se traduce al código fuente de java servlet y luego, utilizando el compilador, se compila en un archivo de clase Servlet, convirtiéndose en la salida para el cliente.

Las grandes ventajas de una página JSP vs un Servlet son las siguientes:

- Los servlets utilizan sentencias de impresión para imprimir un documento HTML, que resulta muy difícil de leer y mantener. JSP no tiene esta característica y mejora sustancialmente la legibilidad del código.
- JSP no necesita compilación, ni configuración de variables de entorno ni tampoco de empaquetado.
- En una página JSP, el contenido visual y la lógica están separados, lo cual con un servlet no se puede lograr.
- Con JSP contamos con la implementación automática. Cuando se genera un cambio en el código no es necesario volver a compilar o construir el elemento, a diferencia de los servlets que al más mínimo cambio es necesario recompilar el código y por ende el servidor debe levantarse nuevamente.

## Ventajas de los JSP frente a otras tecnologías

- **Páginas Active Server Pages (ASP):** La tecnología web de Microsoft. La parte dinámica de los JSP está escrita en java, por lo que es más potente y fácil de usar. En segundo lugar, JSP es una plataforma independiente, mientras que ASP no lo es.
- **Servlets Puros:** Es más conveniente escribir HTML normal en vez de tener declaraciones de impresión o expresiones de servlets. Permite generar la separación de contenido. En un jsp el diseñador web puede enfocarse solamente en el html de la página y los programadores pueden insertar código dinámico por separado, evitando problemas.
- **Server Side include (SSI):** SSI es una tecnología ampliamente soportada para incluir piezas definidas externamente en una página web estática. JSP es mejor porque permite usar servlets en lugar de un programa separado para generar esa parte

dinámica. Además, SSI sólo está destinado a inclusiones simples, no a programas reales que usan datos de formularios, hacen una conexión de base de datos.

- **JavaScript:** Javascript también puede generar HTML dinámicamente en el cliente, pero está limitado solo a la capa del cliente por lo que no puede acceder a los recursos del lado del servidor como base de datos, catálogos, información de lógica de negocios, etc.
- **HTML estático:** La gran diferencia está a la vista, el html puro solamente presenta información y no provee ningún mecanismo de comunicación entre capas, por lo que no existe la colaboración ni el envío de mensajes.

## Elementos de los JSP

Una página JSP está hecha sobre una plantilla HTML, que está compuesta por etiquetas html y elementos java server page como scriptlets, elementos de directivas y elementos de action. En el capítulo anterior se dio una introducción a la implementación de páginas JSP a nivel básico, en el presente se analizarán los detalles de la construcción.

### Scriptlets

Estos elementos en una página JSP delimitan el área en que el código java es interpretado y por consiguiente permite que se pueda utilizar lógica utilizando las características del lenguaje.

Dentro del área de scriptlet es posible usar y manipular clases java, generar excepciones, conectar a bases de datos, validar variables y utilizar todos los métodos disponibles de la API de java. En el ejemplo anterior se utilizaron los tags `<% %>` los cuales cuentan con un tipo de scriptlet, ya que existen 3 a nuestra disposición. El resto de scriptlets son declaraciones y expresiones.

*Tipo 1* `<% %>`

Para incluir código java en una página web JSP se debe delimitar el área de código de programación con los tags `<% %>`.

Por ejemplo, en el siguiente fragmento de código, vamos a mostrar por pantalla el párrafo sólo si una condición se cumple.

Crear un proyecto Dynamic Web Project de nombre *Patrones\_Diseño\_Sesion1\_ejemplo004* y un archivo *index.jsp*.

### Implementación Scriptlets

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Ejemplo de condición</title>
</head>
<body>
<!-- Codigo java -->
<%
    Date date = new Date();
    date.getMonth();
    if(date.getMonth() != 0){
%>
<!-- Codigo html -->
    <h1>Solo me mostrare si el dia es distinto a 0</h1>
<%
    }//codigo java
%>
</body>
</html>
```

Es cierto que la condición no tiene mucha complejidad, pero la idea es ver cómo es posible incluir código java dentro de la página jsp. Si se ejecuta la página jsp desde el navegador podremos ver que siempre se verá el párrafo ya que la lógica está totalmente manipulada, pero es cuestión de cambiar la condición del if para ver que no se mostrará nada en la página.

El resultado de la sentencia anterior es el siguiente:



**Solo me mostrare si el dia es distinto a 0**

Imagen 14. Resultado condicionEjemplo.jsp.  
Fuente: Desafío Latam.

## Tipo 2 <%= %> Expresiones

Las expresiones son un tipo de scriptlets que se encargan de insertar en la página el resultado de una expresión java encerrada entre los tags <%= %>. Por ejemplo en el siguiente código, la expresión insertará la fecha desde la clase date a un tag html. Crear un proyecto Dynamic Web Project de nombre *Patrones\_Diseño\_Sesion1\_ejemplo005* y un jsp de nombre *index.jsp*.

### ExpresionEjemplo.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@page import="java.util.Date" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Expresion Ejemplo</title>
</head>
<body>
    Fecha actual y tiempo: <%= new Date() %>
</body>
</html>
```

### Tipo 3 <%! %> Declaraciones

Las declaraciones son los tags que permiten declarar variables dentro de una página JSP. Cuando se declara una variable entre tags de declaración, la misma es compartida en todo el documento.

Crearemos un pequeño ejemplo para graficar su uso. Crear un proyecto Dynamic Web Project de nombre *Patrones\_Diseño\_Sesion1\_ejemplo006* y un jsp de nombre *index.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>Insert title here</title>
</head>
<body>
    <%! String nombre = "INVITADO"; %>
    <%
        out.println("Bienvenido: " + nombre);
    %>
</body>
</html>
```

El valor de la variable *nombre*, es visible por todo el documento y puede ser usado en cualquier parte del mismo.