



■ PROGRAMACION III

Clase Teórica: UNIDAD IV

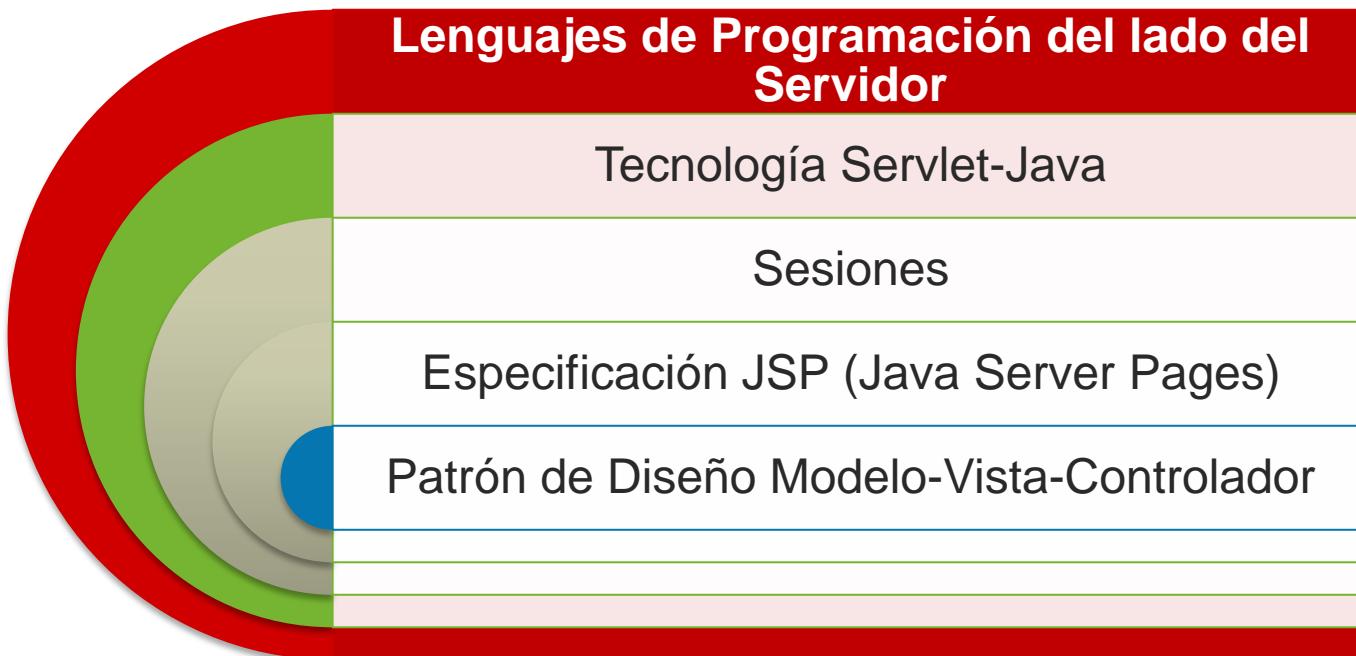
DOCENTES DE CATEDRA:

- ❖ Esp. Cecilia E. Gallardo
- ❖ Esp. Marta Miranda
- ❖ Lic. Oscar Quinteros
- ❖ Julián Acosta Argañaraz





UNIDAD IV: INTRODUCCIÓN A LA PROGRAMACIÓN DEL LADO DEL SERVIDOR

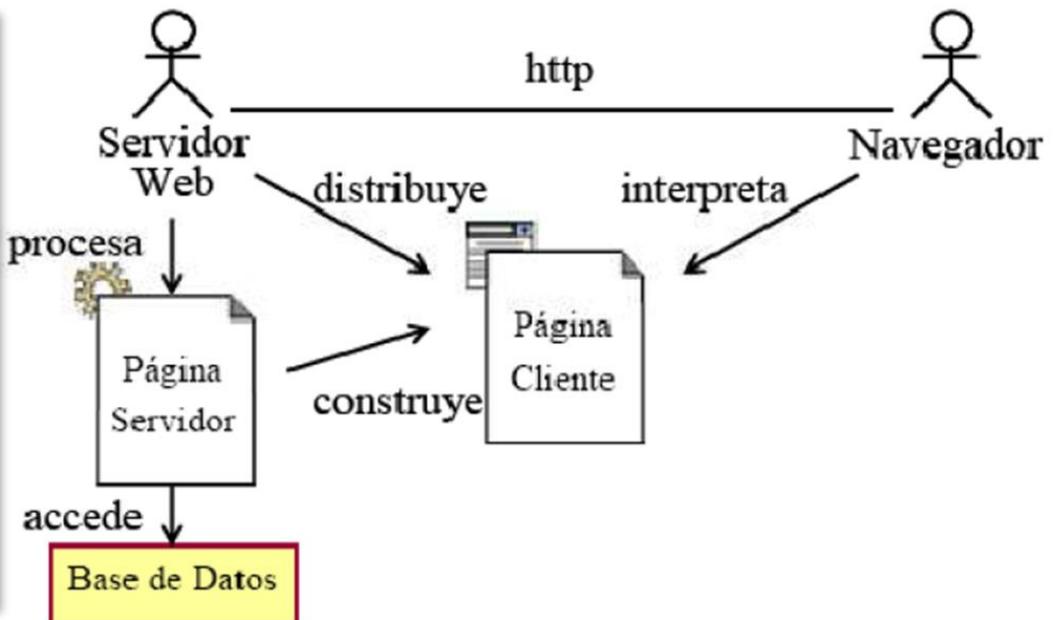


Lenguajes del lado del Servidor. Definición

Un lenguaje del lado del servidor es aquel que se ejecuta en el servidor web, procesando la petición del cliente y generando páginas HTML dinámicamente como respuesta, con lógica de negocio y de arquitectura (persistencia en B.D., transaccionalidad, etc.).

El cliente no puede ver el código fuente, ya que se ejecuta y transforma en doc. HTML antes de enviarlos.

Desde el servidor web también se proporcionan servicios web (SOAP, REST) que pueden ser consumidos por cualquier cliente (navegador, cliente celular, otra aplicación, etc.)





Lenguajes y tecnologías del lado del Servidor. Ejemplos

Algunas tecnologías para el desarrollo web:

- **ASP.NET** (Active Server Pages) de Microsoft. Es el sucesor de la tecnología ASP. Se puede utilizar C#, VB.NET o J#. Los archivos cuentan con la extensión (aspx). Requiere tener instalado el Servidor *Internet Information Server (IIS)* con el Framework .Net
- **PHP** (PHP Hypertext Pre-processor). Es un lenguaje de script interpretado en el lado del servidor utilizado para la generación de páginas web dinámicas, embebidas en páginas HTML y ejecutadas en el servidor. A su vez, **Symfony** es un conjunto de componentes de PHP, un framework de aplicación web, una filosofía y una comunidad de trabajo en conjunto.
<https://symfony.com/>
- **Django**. Es un framework web de alto nivel que utiliza el lenguaje de programación **Python**. Fomenta un desarrollo rápido y un diseño limpio y pragmático. Es gratis y de código abierto. <https://www.djangoproject.com/>



Lenguajes y tecnologías del lado del Servidor. Ejemplos

- **Rails**. Es un framework de desarrollo de aplicaciones web escrito en el lenguaje de programación **Ruby** (lenguaje multiplataforma, dinámico, opensource e interpretado de muy alto nivel y orientado a objetos).
<https://rubyonrails.org/>
- **Trails** representa una generación avanzada del framework **Node.js**. Diseñado en torno al patrón de microservicios, su base de código mejora el rendimiento, la capacidad de prueba, la estabilidad y la seguridad.
<https://trailsjs.io/>
- **JSP** (Java Server Pages). Es un lenguaje multiplataforma para desarrollar páginas web con la tecnología Java. Posee un motor de páginas basado en **servlets**. Para su funcionamiento requiere un servidor contenedor de servlets. Las páginas se compilan en la primera petición. El código JSP puede ser incrustado en código HTML.



Lenguajes del lado del Servidor. Evolución

- Los primeros servidores web permitían visualizar sólo información estática (contenido estático).
- Esto resultó en una limitación: la actividad publicitaria y comercial comenzó a concentrarse también en Internet.
- **Primera solución técnica:** el servidor web ejecutaba programas residentes en él que generaban dinámicamente el contenido → tecnología **Common Gateway Interface (CGI)** que permitía ejecutar programas escritos en C o Perl.
- **Limitaciones de CGI:** **seguridad** y **carga del servidor**.
- **Solución:** desarrollar una tecnología que permita ejecutar, en un único proceso del servidor, todos los pedidos de ejecución de código sin importar la cantidad de clientes que se conecten concurrentemente.

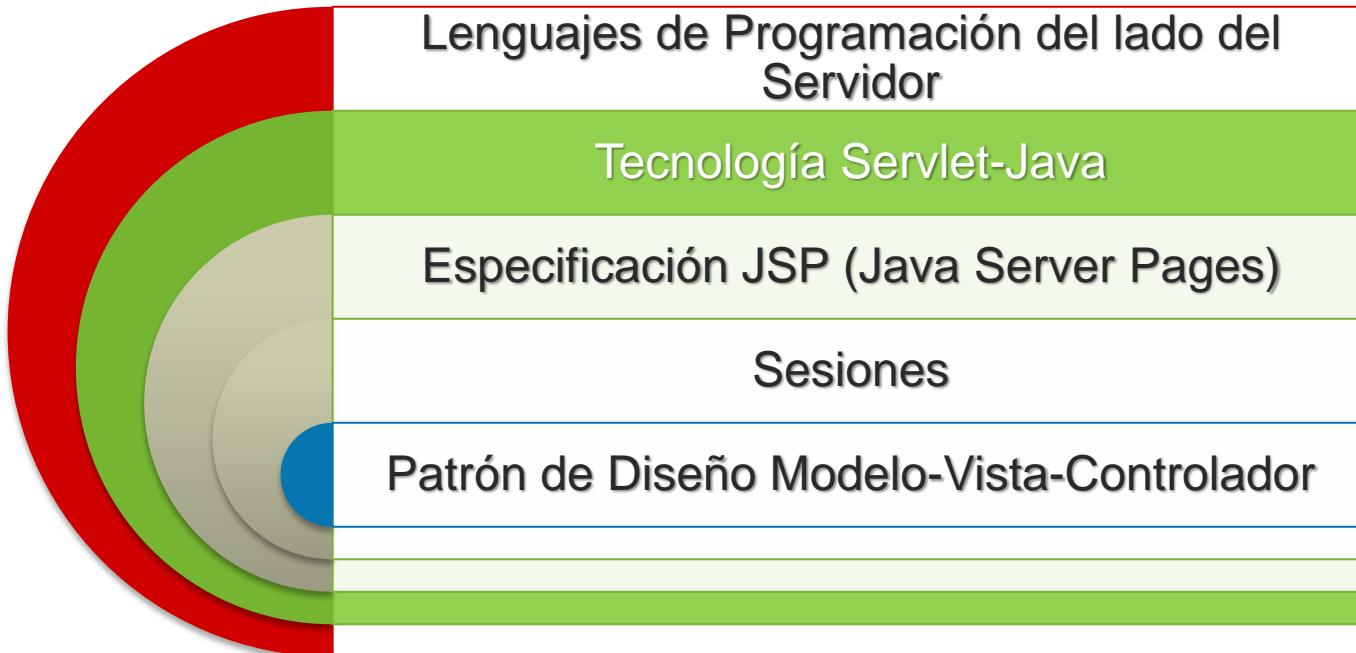


Lenguajes del lado del Servidor. Evolución

- Así surgieron los **servlets**, basados en la tecnología Java y los **filtros ISAPI** de Microsoft. Éstos permitían ejecutar código en un único proceso externo que gestionaba todas las llamadas realizadas por el servidor web, impidiendo al mismo tiempo que el servidor web pueda ejecutar programas del sistema operativo.
- **Limitación:** desarrollo demasiado costoso en términos de tiempo.
- **Solución propuesta:** se permite la inclusión de porciones de código de lenguajes en el interior de archivos HTML. Estos comandos pueden ser **interpretados** (ASP o PHP) o **precompilados** (JSP o ASP.NET).



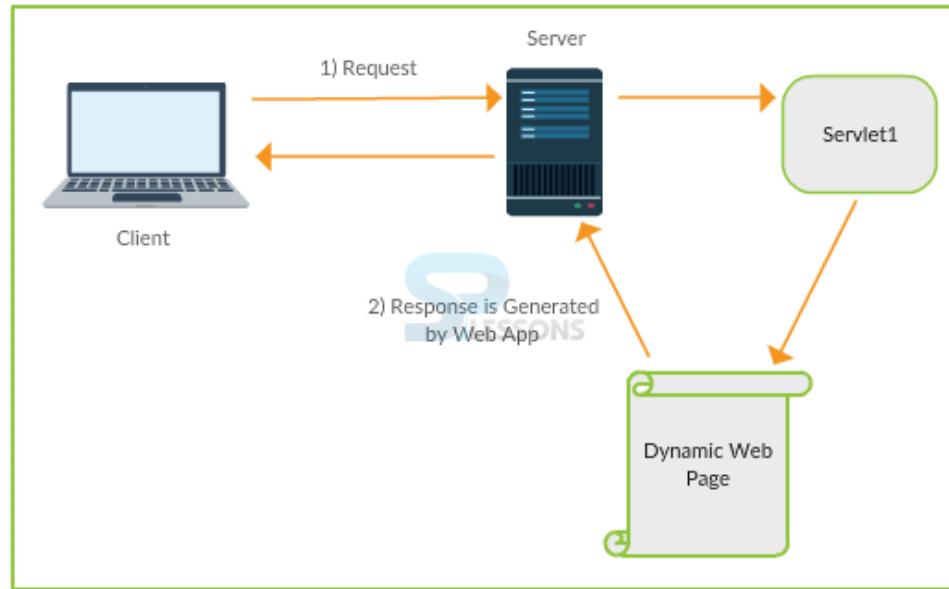
UNIDAD IV: INTRODUCCIÓN A LA PROGRAMACIÓN DEL LADO DEL SERVIDOR



Tecnología Servlet-Java

SERVLET JAVA

- Desde el punto de vista de la programación es una **clase JAVA** que se ejecuta dentro de un servidor web (JEE), como por ejemplo Apache Tomcat.
- Produce contenido dinámico en respuesta a un requerimiento HTTP enviado por el cliente.
- Los **servlets** representan la respuesta de la tecnología Java a la programación CGI.





Tecnología Servlet-Java. Ventajas sobre CGI tradicional

- **Eficiencia.** Con CGI, se inicia un nuevo proceso para cada solicitud HTTP. Con servlets, la MVJ se mantiene activa, y cada petición es manejada por un subproceso de Java liviano (thread), manteniendo una copia única de la clase servlet.
- **Conveniencia.** Los Servlets poseen una gran infraestructura para análisis automático y decodificación de datos de formularios HTML, leer y configurar cabeceras HTTP, manejar cookies, seguimiento de sesiones y muchas otras utilidades.
- **Portabilidad.** Los Servlets están escritos en Java y poseen un API estandarizada.
- **Diversidad de servidores.** Existe un gran número de herramientas y servidores para el desarrollo de aplicaciones Web utilizando tecnología J2EE.



Tecnología Servlet-Java. Funcionamiento Básico

- Para ser un **Servlet**, una clase debe extender la clase **javax.servlet.http.HttpServlet** y sobrescribir los métodos **doGet** o **doPost** (o ambos), dependiendo de si los datos son enviados por los métodos GET o POST de HTTP.

METODOS doGet() y doPost()

reciben como argumento

Objeto **HttpServletRequest**

- Encapsula la comunicación desde el cliente al servidor.
- Tiene métodos que permiten conocer la información entrante, como datos de FORM, encabezados de solicitud HTTP, etc.

Objeto **HttpServletResponse**

- Encapsula la comunicación de vuelta desde el servlet hacia el cliente.
- Tiene métodos que permiten especificar la línea de respuesta HTTP (200, 404, etc.), los encabezados de respuesta (Content-Type, Set-Cookie), obtener un objeto PrintWriter utilizado para enviar resultados de vuelta al cliente, redireccionar pedidos HTTP a otra URL .



Tecnología Servlet-Java. Funcionamiento Básico

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
  
public class HolaMundo extends HttpServlet {  
    protected void doGet(..) throws ServletException, IOException { }  
    protected void doPost(HttpServletRequest request, HttpServletResponse  
                          response) throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
        try (PrintWriter out = response.getWriter()) {  
            out.println("<!DOCTYPE html>");  
            out.println("<html>");  
            out.println("<head>");  
            out.println("<title>Servlet HolaMundo</title>");  
            out.println("</head>");  
            out.println("<body>");  
            out.println("<h1>Servlet HolaMundo</h1>");  
            out.println("</body>");  
            out.println("</html>"); }  
    } }
```



Tecnología Servlet-Java. Recepción de parámetros

Métodos de la clase HttpServletRequest

- El método **String getParameter(String paramName)** devuelve el valor de un parámetro, ya sea enviado mediante la URL en una solicitud GET, o correspondiente al valor de un campo de un formulario que se envía mediante el método POST.
- El método **String[] getParameterValues(String paramName)** retorna todos los valores asociados con el parámetro. Un parámetro puede tener múltiples valores (por ejemplo, una lista de selección múltiple).
- El método **Enumeration getParameterNames()** es útil cuando no se conocen los nombres de los parámetros. Se puede iterar a través de la enumeración de String retornados y por cada elemento llamar a `getParameter()` o `getParameterValues()`.

Tecnología Servlet-Java. Recepción de parámetros

- Ejemplo de recepción de parámetros de una solicitud GET:

```
public class TresParametrosServlet extends HttpServlet {  
    protected void doGet (...) throws ServletException, IOException {  
        try (PrintWriter out = response.getWriter()) {  
            ... // definición de DOCTYPE y etiquetas html, head y body  
            out.println("<h1>Recibiendo tres parámetros de la solicitud</h1>");  
            out.println("<ul>");  
            out.println("<li>Parametro 1:" +request.getParameter('param1')+"</li>");  
            out.println("<li>Parametro 2:" +request.getParameter('param2')+"</li>");  
            out.println("<li>Parametro 3:" +request.getParameter('param3')+"</li>");  
            out.println("</ul>");  
            out.println("</body>");  
            out.println("</html>"); } }
```



Tecnología Servlet-Java. Recepción de parámetros

- Ejemplo de recepción de parámetros de una solicitud POST:

```
<DOCTYPE html>
<html>  <head>...</head>
<body>
    <h1>Envío de Parametros desde Form</h1>
    <form action="ParametrosForm" method="POST">
        Nombre: <input type="text" name="apeynom" ><br>
        DNI: <input type="text" name="dni" ><br><br>
        <input type="submit" value="Enviar Datos" /> método
    </form>
</body></html>
```



```
public class ParametrosFormServlet extends HttpServlet {
    protected void doPost (...) throws ServletException, IOException {
        out.println("<h1>Recibiendo parámetros desde Form</h1>");
        out.println("<ul>");
        out.println("<li>Nombre:" +request.getParameter('apeynom')+ "</li>");
        out.println("<li>DNI:" +request.getParameter('dni')+ "</li>");
        out.println("</ul>");
        out.println("</body>");
        out.println("</html>"); }
```





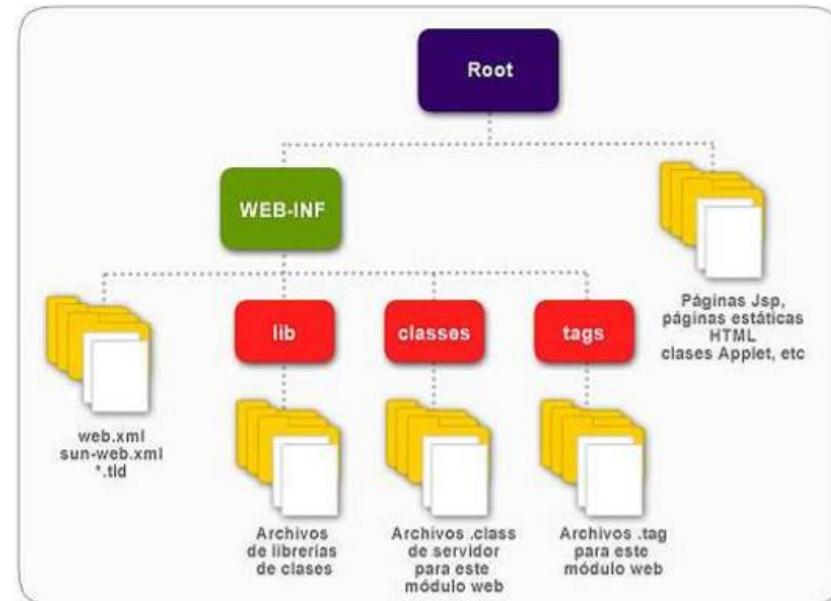
Tecnología Servlet-Java. Descriptor de despliegue

- Es un documento XML llamado web.xml
- Debe ser colocado en el directorio WEB-INF de la aplicación web.
- Contiene opciones de configuración específicas de la aplicación:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
    <servlet>
        <servlet-name>procesamientoParametros</servlet-name>
        <servlet-class>TresParametrosServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>procesamientoParametros</servlet-name>
        <url-pattern>TresParametros</url-pattern>
    </servlet-mapping>
</web-app>
```

Tecnología Servlet-Java. Estructura de una WebApp

- **web.xml:** Descriptor de despliegue de la aplicación web
- Archivos descriptores de librerías de tag
- **classes:** contiene clases del lado del servidor: servlets, clases de utilidad y componentes JavaBeans
- **tags:** directorio que contiene archivos tag, los cuales son implementaciones de librerías de tags
- **lib:** directorio que contiene archivos .jar de librerías utilizadas por las clases del lado servidor





Desarrollo del lado del servidor. Sesiones

¿Por qué decimos que HTTP es un protocolo sin estado?

- Hypertext Transfer Protocol (HTTP) es el protocolo de red empleado por servidores y navegadores de clientes para comunicarse.
- Las conexiones HTTP son iniciadas por el cliente que envía una petición HTTP REQUEST.
- El servidor web responde con un mensaje HTTP RESPONSE y tan pronto como finaliza de enviar una respuesta cierra la conexión.
- Si el mismo cliente solicita nuevamente un recurso del servidor, deberá establecer nuevamente una conexión con el servidor.
- HTTP es un protocolo SIN ESTADO, es decir que cada requerimiento al servidor web, y su correspondiente respuesta, es manejada como una transacción aislada y el servidor no mantiene automáticamente información contextual sobre el cliente.



Desarrollo del lado del servidor. Sesiones

HTTP Sin Estado - Implicancias

Consideremos un ejemplo de un portal de comercio electrónico:

- ✓ Un proceso usual en este tipo de aplicaciones web inicia con el usuario (cliente) buscando uno o varios productos particulares.
- ✓ Si el producto es encontrado, entonces el usuario selecciona la cantidad del producto que desea y lo agrega al “carrito de compras”, y envía dicha información al servidor.
- ✓ El servidor la recibe y envía como respuesta una página para que siga buscando información y “cierra la conexión”.
- ✓ Luego el usuario busca por otro producto, y lo agrega al carrito de compras.
- ✓ Cuando envía el segundo producto al carrito de compras, como el servidor no sabe nada acerca de si la conexión pertenece al usuario anterior o a otro, por lo que no tiene forma de interpretar que estos datos y agregarlos al carrito de compras correspondiente.

Solución provista => Administración de sesiones



Desarrollo del lado del servidor. Sesiones

Seguimiento de Sesiones

- Una **sesión** es un conjunto de interacciones (solicitud y respuesta HTTP) entre un cliente y el servidor web que tienen lugar durante un período de tiempo.
- El cliente y el servidor necesitan intercambiar un identificador que se asocie únicamente (session-id) a un único cliente con el servidor dentro del contexto de una sesión ► *estado conversacional que tiene significado para el servidor.*
- El servidor debe recordar información relacionada con peticiones previas y otras decisiones tomadas.
- Existen dos mecanismos utilizados para intercambiar el ID-Sesion:
 - ✓ Uso de Cookies
 - ✓ Reescritura de URL



Desarrollo del lado del servidor. Sesiones

Seguimiento de Sesiones: Uso de cookies

- Las cookies fueron desarrolladas originalmente por Netscape, para resolver el problema de guardar el contexto de una sesión usando HTTP.
- Las cookies en realidad son pares de “nombres=valor” intercambiados entre cliente y servidor a través de encabezados HTTP especiales
 - ✓ El encabezado **Set-Cookie** envía las cookies del servidor al cliente, cuando la “sesión” es establecida.
 - ✓ El encabezado **Cookie** del cliente al servidor cada vez que se requiere un valor, hasta que expira su tiempo límite de vida.
- El navegador guarda estos datos locamente y los envía de regreso junto con la nueva petición al mismo servidor web.

Desarrollo del lado del servidor. Sesiones

Seguimiento de Sesiones. Ejemplos de uso de cookies

Uso de cookies (Servidor)

- HTTP/1.1 200 OK
- Date: Thu, 20 Sep 2018
16:48:08 GMT
- Transfer-Encoding: chunked
- Server: Apache Tomcat/5.0
(HTTP/1.1 Connector)
- **Set-Cookie:** uid=a3fWa; Max-Age=3600;

[page content]

Uso de cookies (Cliente)

- GET /jspbook/servlet/com.SessionStateHTTP/1.1
- Accept: image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg,
- application/msword, */*
- Accept-Language: en-gb
- Accept-Encoding: gzip, deflate
- User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705)
- Host: localhost:8080 Connection: Keep-Alive
- **Cookie:** uid=a3fWa; Max-Age=3600;



Desarrollo del lado del servidor. Sesiones

Seguimiento de Sesiones: Propagar Session-ID mediante Cookies

- De este modo, es posible “propagar” el identificador único para cada cliente que permita identificar una secuencia de requerimientos del mismo cliente de manera única.
- La especificación de **Servlets** define que es posible propagar un identificador de sesión a través de las cookies.
- El nombre de la cookie que se define para propagar el Session-ID es “**JSESSIONID**”.
- El valor de la cookie que posee el ID de sesión es un número en base hexadecimal.
- Las **ventajas** de cookies es que son fáciles de implementar, personalizables y que persisten ante el reinicio del browser.
- Las **desventajas** de cookies es que se genera carga de trabajo extra para realizar el manejo y administración de cookies y los usuarios pueden desactivar el uso de cookies en el browser.



Desarrollo del lado del servidor. Sesiones

Seguimiento de Sesiones: Propagar Session-ID mediante Reescritura de URL

- La reescritura de URL es un mecanismo para realizar seguimientos de sesiones en donde el cliente agrega información extra que identifica la sesión al final de cada URL, y el servidor asocia este identificador de sesión con el dato que almacena para la sesión.
- Por ejemplo: **<http://host/path/file.html;jsessionid=1234>**
- Esta es una solución eficaz, y tiene la **ventaja** de que es universalmente soportada, funcionando independientemente del soporte de cookies del browser.
- Un **desventaja** importante es que se debe reescribir todas las URL de una página cada vez que la página es generada.
- Si no se incluye el identificador de sesión, se pierde el estado conversacional.



Desarrollo del lado del servidor. Sesiones

Seguimiento de Sesiones: API de Sesiones en servlets

- El contenedor de servlets es el que maneja internamente las sesiones con sus correspondientes identificadores y provee un lugar (objeto session) para guardar los objetos asociados con cada sesión.
- El manejo de sesiones en servlets implica buscar el objeto sesión asociado con el requerimiento actual, crear un nuevo objeto de sesión cuando sea necesario, buscar información asociada con la sesión, almacenar información en la sesión y descartar sesiones completadas o abandonadas.
- El API Servlet abstrae el concepto de sesión a través de la interfaz **javax.servlet.http.HttpSession**, que está construida sobre el mecanismo de **cookies** o el mecanismo de **reescritura de URL**.
- La mayoría de los servidores utilizan cookies si el browser las soporta y cambian automáticamente a utilizar reescritura de URL cuando las cookies no son soportadas o son deshabilitadas.



Desarrollo del lado del servidor. Sesiones

Seguimiento de Sesiones: Uso de API HttpSession de Servlets

1. Recuperar la sesión asociada con una solicitud. Mediante los métodos de la interfaz **HttpServletRequest** para obtener la sesión:

- **HttpSession getSession(boolean create)**: retorna el objeto HttpSession asociado con la solicitud. Si no existe una sesión creada, y el parámetro “create” es true, entonces retorna una sesión nueva.
- **HttpSession getSession()**: es equivalente a getSession(true)

2. Agregar o quitar atributos del tipo nombre-valor a la sesión. Mediante los métodos de la interfaz **HttpSession**:

- **public Object getAttribute(String name)**: recupera un atributo desde el objeto HttpSession.
- **public void setAttribute(String name, Object attribute)**: asigna un objeto a la sesión con un nombre determinado.
- **public void removeAttribute(String name)**: quita un atributo del objeto HttpSession asociado.



Desarrollo del lado del servidor. Sesiones

Seguimiento de Sesiones: Uso de API HttpSession de Servlets

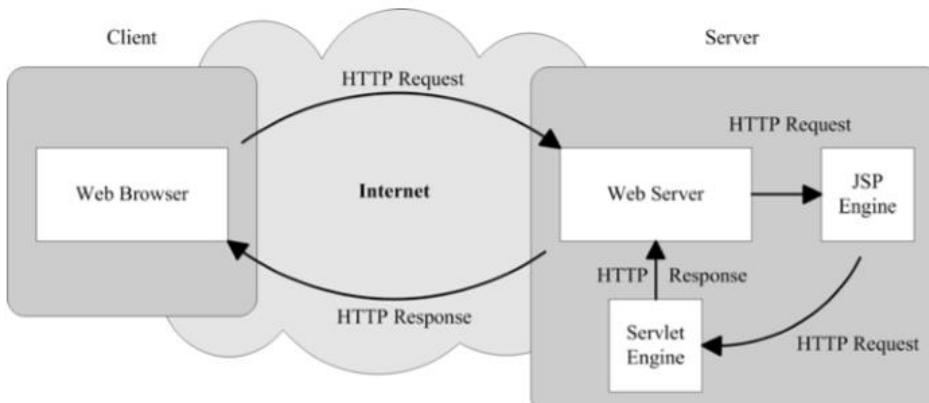
3. Gestionar el ciclo de vida de la sesión. Mediante los métodos de la interfaz **HttpSession**:

- **getCreationTime()**: devuelve el instante en que se creó la sesión
- **getID()**: devuelve el ID de la sesión
- **getLastAccessedTime()**
- **setMaxInactiveInterval()**: Establece el tiempo en segundos que la sesión permanecerá inactiva entre peticiones antes de invalidarla
- **isNew()**: es true si la sesión ha sido creada pero el cliente no lo sabe

4. Invalidar la sesión si es necesario. Mediante el método de la interfaz **HttpSession**: **public void invalidate()** que invalida la sesión y la desvincula del objeto con el que estaba relacionada.

Especificación JSP (Java Server Pages)

- **JSP** es una especificación de la capa Web que complementa la especificación de Servlet y es útil en el desarrollo de interfaces Web para aplicaciones empresariales.
- **JSP** es un documento basado en texto combina lenguajes de marcado HTML/XML y elementos del lenguaje de programación Java para retornar contenido Web dinámico a clientes Web.
- **JSP** usa las mismas técnicas usadas en la programación con Servlet. Por ejemplo, JSP trabaja con solicitudes y respuesta HTTP, parámetros y atributos de la solicitud, administración de sesiones, etc.





Especificación JSP (Java Server Pages)

■ Ejemplo archivo holamundo.jsp

```
<html>
  <body>
    <%@ page language="java" %>
    <%! int count = 0; %>
    <% count++; %>
    Usted es el visitante nro: <%= count %>
  </body>
</html>
```



Especificación JSP. Relación entre JSP y Servlet

- JSP es una extensión de Servlet.
- Las páginas JSP son traducidas a código Servlet cuando son desplegadas en un contenedor Web.
- Si se desea incluir tags HTML en un servlet, los mismos deben ser embebidos dentro de una variable de tipo String utilizando código Java, lo cual lo torna en un trabajo tedioso y poco flexible.
- Mediante JSP, la página HTML puede ser presentada utilizando directamente tags HTML e intercalando con código java.
- En una típica aplicación Web se utilizan tecnologías **Serlvet** y **JSP** juntas en lo que se conoce como **patrón MVC** (Model-View-Controller), en el cual los serlvet juegan el papel de **controlador** mientras las páginas JSP realizan la tarea de **presentación (vista)**.



Especificación JSP. Relación entre JSP y Servlet

Diferencias entre Servlet y JSP

Servlet

Código html en Java

Dificultad en la
confección de la
pagina web

JSP

Código Java en
html

Facilidad en la
confección de la
página web

El código es compilado en
un servlet



Especificación JSP. Elementos de sintaxis

- JSP posee una gramática bien definida e incluye elementos de sintaxis
- Estos elementos de sintaxis son llamados **tags JSP**, los cuales están clasificados en seis categorías:

Tipo de tag JSP	Descripción	Sintaxis
Directiva	Especifica instrucciones en tiempo de traducción para el motor JSP	<%@ Directives %>
Declaración	Declara y define variables y métodos	<%! Declaraciones Java %>
Scriptlet	Permite al desarrollador escribir código Java en una página JSP	<% código Java %>
Expresión	Usado como un atajo para imprimir valores en la salida HTML de una página JSP	<%= Una expresión %>
Acción	Provee instrucciones en tiempo de solicitud al motor JSP	<jsp:nombreAccion />
Comentario	Usado para documentación y para comentar partes del código JSP	<%-- Un comentario --%>



Especificación JSP. Elementos de sintaxis

■ Ejemplo de elementos de sintaxis JSP

```
<html>
  <body>
    <%@ page language="java" %> DIRECTIVA
    <%! int count = 0; %> DECLARACION
    <% count++; %> SCRIPTLET
    Usted es el visitante nro: <%= count %> EXPRESION
  </body>
</html>
```



Especificación JSP. Elementos de sintaxis

DIRECTIVAS

- Proveen información general acerca de la página JSP al motor JSP
- Una directiva siempre empieza con %@ y termina con %>
- Hay tres tipos de directivas: page, include y taglib.
 - **Page:** informa al motor acerca de todas las propiedades de una página JSP `<%@ page import="java.util.Date"%>`
 - **Include:** informa al motor JSP que incluya el contenido de otro archivo (HTML, JSP, etc.) en la página actual
`<%@ include file="copyright.html" %>`
 - **Taglib:** es usada para asociar un prefijo con una librería de tags. `<%@ taglib prefix="test" uri="taglib.tld" %>`



Especificación JSP. Elementos de sintaxis

DIRECTIVA: “Page” Atributo “Import”

- Similar a la declaración import en una clase Java.
- Se utiliza para incluir el contenido de un paquete para evitar escribir el “path” de una clase cada vez que se utiliza.
- Por ejemplo, la directiva `<%@ page import="java.util.Date"%>` permite referenciar directamente a la clase Date sin prefijar con el nombre del paquete.
- En tiempo de traducción, el motor JSP inserta una declaración import en el servlet generado por cada paquete declarado usando este atributo.



Especificación JSP. Elementos de sintaxis

DIRECTIVA: “Page” Atributo “Session”

- El atributo “session” indica si la página JSP forma parte en la sesión HTTP.
- El valor por defecto es true, lo que significa que el motor JSP declara la variable **session** implícitamente.
- Si no se desea que la página participe en una sesión se debe explicitar agregando la siguiente línea:

```
<%@ page session="false" %>
```



Especificación JSP. Elementos de sintaxis

DIRECTIVA: “Page” Atributo “errorPage”

- El atributo “errorPage” permite delegar el tratamiento de una excepción en otra página JSP
- Ejemplo: `<%@ page errorPage="errorHandler.jsp" %>`
- En caso de generarse un excepción, y la misma no es tratada dentro de la pagina, entonces el motor de jsp delega el mismo en la página `errorHandler.jsp`



Especificación JSP. Elementos de sintaxis

DIRECTIVA: “Page” Atributo “isErrorHandler”

- Comunica al motor de jsp que la página actual puede actuar como un manejador de errores para otra página JSP
- El valor por defecto del atributo “isErrorHandler” es false.
- Ejemplo de página para el manejo de errores:

```
<%@ page isErrorPage="true" %>
<html>
<body>
    No se pudo procesar la petición:
    <%=exception.getMessage() %> <br>
    Intente nuevamente mas tarde.
</body> </html>
```



Especificación JSP. Elementos de sintaxis

DECLARACIONES

- Declaran y definen variables y métodos que pueden ser utilizados en la página de JSP
- Siempre empiezan con <%! y terminan con %>
- Puede contener cualquier número de declaraciones Java.
- Cada declaración de una variable debe terminar con un punto y coma. Ejemplo:

```
<%! String color[] = {"red", "green", "blue"};  
    String getColor(int i)  
    {return color[i];}  
%>
```



Especificación JSP. Elementos de sintaxis

SCRIPTLET

- Son fragmentos de código de Java que se insertan en la página de JSP
- Un scriptlet siempre empieza con <% y termina con %>
- Se ejecutan cada vez la página es accedida
- Utilizados para embeber lógica dentro de una página de JSP
- El código dentro de un scriptlet debe ser válido en el lenguaje de programación Java.
- Ejemplo:
`<% if (condition) { %>
 <p>This is only shown if the condition is satisfied</p>
<% } %>`



Especificación JSP. Elementos de sintaxis

EXPRESIONES

- Actúan como contenedores para expresiones Java
- Se evalúa cada vez la página es accedida, y su valor es embebido en la salida HTML
- Siempre empiezan con <%= y terminan con %>
- No deben terminar con un punto y coma
- Se puede imprimir el valor de cualquier objeto o tipo de dato primitivo (int, booleano, char, etc.) al flujo de salida utilizando una expresión. También se puede imprimir el valor de cualquier expresión aritmética o booleana o el valor returned por una llamada de un método.
- Ejemplo: <%= aFloatObj.toString() %> Un método que retorna un objeto String



Especificación JSP. Elementos de sintaxis

ACCIONES

- Son comandos dados al motor JSP. Existen seis acciones JSP estándares:
 - **jsp:include** y **jsp:forward**: Habilitan al contenedor a reusar otros componentes web
 - **jsp:useBean**, **jsp:setProperty**, **jsp:getProperty**: Relacionadas al uso de Java Beans.
 - **jsp:plugin**, le indica al motor JSP que debe generar código HTML apropiado para embeber componentes del lado cliente.
- Sintaxis general de una acción JSP: **<jsp:actionName attribute-list />**
- Una pagina JSP puede tener acciones definidas por el usuario (Tags personalizados o “Custom Tags”)



Especificación JSP. Objetos Implícitos

OBJETOS IMPLICITOS

- Ciertos objetos de uso común, están disponibles para su uso automático en la pagina JSP, tales como:
 - **session** = javax.servlet.http.HttpSession
 - **request** = javax.servlet.http.HttpServletRequest
 - **response** = javax.servlet.http.HttpServletResponse
 - **out** = javax.servlet.jsp.JspWriter
 - **application** = javax.servlet.ServletContext
 - **exception** = java.lang.Throwable
 - **page** = javax.servlet.jsp.HttpJspPage



Especificación JSP. Objetos Implícitos

OBJETOS IMPLICITOS: Ejemplo

<HTML>

```
<HEAD> <TITLE>Página Simple</TITLE></HEAD>
<BODY>
    <% String nombre = request.getParameter("nombre");
      out.println("Nombre: " + nombre); %>

    <h1>Un mensaje</h1>
    <% out.println("Con JSP es mas fácil"); %>
</BODY>
</HTML>
```



Especificación JSP. Reutilización de recursos web

- Reutilización de recursos Web significa incluir el **contenido o la salida de otro componente Web** en una página JSP
- Existen dos maneras de incluir el contenido de un recurso web en una página jsp:
 - **Inclusión Estática:** el contenido del componente Web es incluido en el archivo JSP en tiempo de traducción
 - **Inclusión Dinámica:** la salida de otro componente es incluido en la salida de una página JSP cuando la página JSP es solicitada



Especificación JSP. Reutilización de recursos web

INCLUSION ESTATICA

- El contenido de otro archivo es incluido con el archivo JSP actual en tiempo de traducción para producir un único servlet
- Los nombres de archivos incluidos pueden tener cualquiera extensión (txt, html, inc, jsp...)
- Sintaxis para realizar una inclusión:

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

```
<jsp:directive.include file="URLrelativa"/>
```



Especificación JSP. Reutilización de recursos web

INCLUSION DINAMICA

- La acción **<jsp:include page="relativeURL" flush="true" />** delega el control del procesamiento de la solicitud al componente incluido de manera temporal. Y una vez que el componente incluido finaliza su procesamiento, el control es transferido de regreso a la página que hizo la inclusión.
- La acción **<jsp:forward page="relativeURL"/>** delega el procesamiento de la solicitud al componente indicado en esta acción, quien posteriormente envía la respuesta al cliente, sin cambiar el URL de origen.
- La acción **<jsp:param name="nombre" value="valor"/>** adjunta un valor de parámetro a una consulta pasada a otro servlet o JSP utilizando <jsp:include> o <jsp:forward>



Especificación JSP. Java Beans

- Un **bean** es un objeto de **clase Java** que encapsula los datos en forma de variables de instancia.
- Cualquier clase que sigue estas dos convenciones se puede utilizar como un JavaBean en páginas JSP:
 - La clase debe tener a un constructor público sin argumentos. Esto permite que la clase sea instanciada a medida que el motor de JSP lo necesite.
 - Para cada propiedad, la clase debe tener dos métodos públicamente accesibles, referidos como getter y setter, esto permite al motor de JSP acceder o mutar las propiedades de bean. Sintaxis: **public property-type getXXX();**
public void setXXX(property-type);



Especificación JSP. Java Beans

Ejemplo de interacción con un Java Beans

- Crear un objeto **AddressBean** que almacene información de direcciones postales y un formulario para administrarlas.
- Cuando el usuario llena el formulario y envía la pagina, se deben realizar las siguientes tareas en el archivo “address.jsp” que procesará los datos desde el servidor:
 - Chequear si un objeto AddressBean ya existe en la sesión
 - Si no existe, crear un nuevo objeto AddressBean y agregarlo a la sesión.
 - Llamar a `request.getParameter()` por todos los campos del formulario HTML.
 - Setear los respectivos valores en el objeto AddressBean.



Especificación JSP. Java Beans

Ejemplo Java Beans

```
public class AddressBean {  
    private String street;  
    private String city;  
    public void setStreet(String street) {  
        this.street = street; }  
    public void setCity(String city) {  
        this.city = city; }  
    public String getStreet() {  
        return this.street; }  
    public String getCity() {  
        return this.city; }  
}
```



Especificación JSP. Java Beans

Formulario HTML

```
<html>
  <body>
    Por favor ingrese su domicilio:<br>
    <form action="address.jsp">
      Calle: <input type="text" name="street">
      <br>
      Ciudad: <input type="text" name="city"><br>
      <input type="submit">
    </form>
  </body>
</html>
```



Especificación JSP. Java Beans

Archivo “address.jsp”. Opción 1: utilizando scriplet

```
<%@ page import="AddressBean" %>
<%
    AddressBean address = null;
    address=(AddressBean) session.getAttribute("address");
    if (address==null) {
        address = new AddressBean();
        session.setAttribute("address", address);
    }
    address.setStreet(request.getParameter("street"));
    address.setCity(request.getParameter("city"));
%>
```



Especificación JSP. Java Beans

Archivo “address.jsp”. Opción 2: utilizando tags jsp

```
<%@ page import="AddressBean" %>
<jsp:useBean id="address" class="AddressBean" scope="session"/>
<jsp:setProperty name="address" property="street" />
<jsp:setProperty name="address" property="city"/>
<html><body>
    <table>
        <tr> <td>Street</td>
            <td> <jsp:getProperty name="address"
                property="street"/> </td> </tr>
        <tr> <td>City</td>
            <td><jsp:getProperty name="address"
                property="city"/> </td> </tr>
    </table> </body> </html>
```



Especificación JSP. Java Beans

- **<jsp:useBean>** declara el uso de una instancia JavaBean en una página JSP y puede contener los siguientes atributos:
 - Id: el nombre por el cual el bean es identificado en la pagina JSP
 - Scope: el alcance de la instancia del bean y que puede contener los valores: **page, request, session, application.**
 - Class: la clase Java del bean
 - Type: tipo de la variable a usar como referencia
 - beanName: el nombre de la Clase
- **<jsp:setProperty>** Asigna un nuevo valor en las propiedades del bean que tienen el mismo nombre que un parámetro del formulario enviado. Es posible asignar todas las propiedades de un bean en una sola acción:
<jsp:setProperty name="address" property="*" />
- **<jsp:getProperty>** Obtiene el valor actual de una propiedad del bean



Especificación JSP. Desarrollo de una WebApp

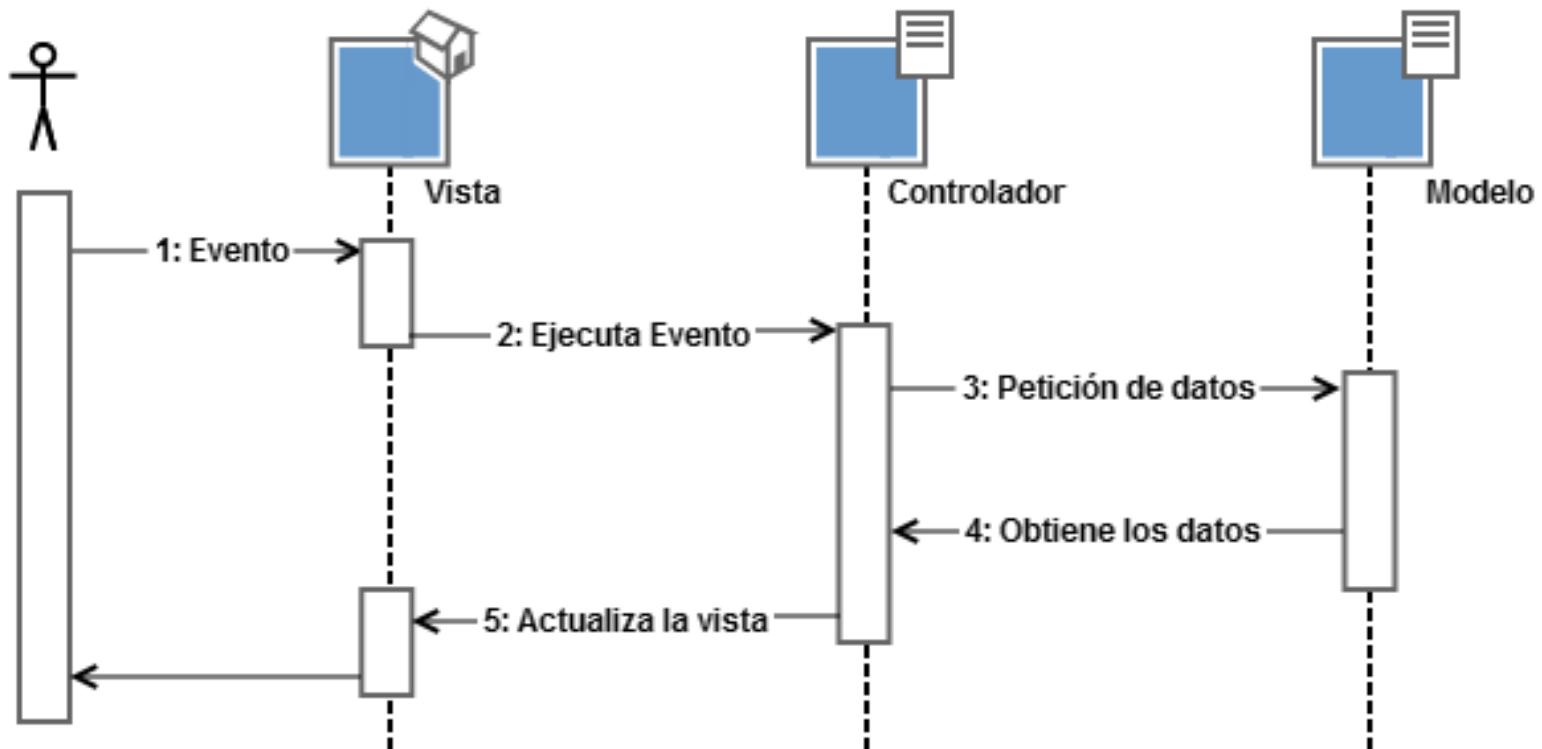
Aplicación Web JSP

Pasos que hay que seguir para desarrollar y desplegar una WebApp:

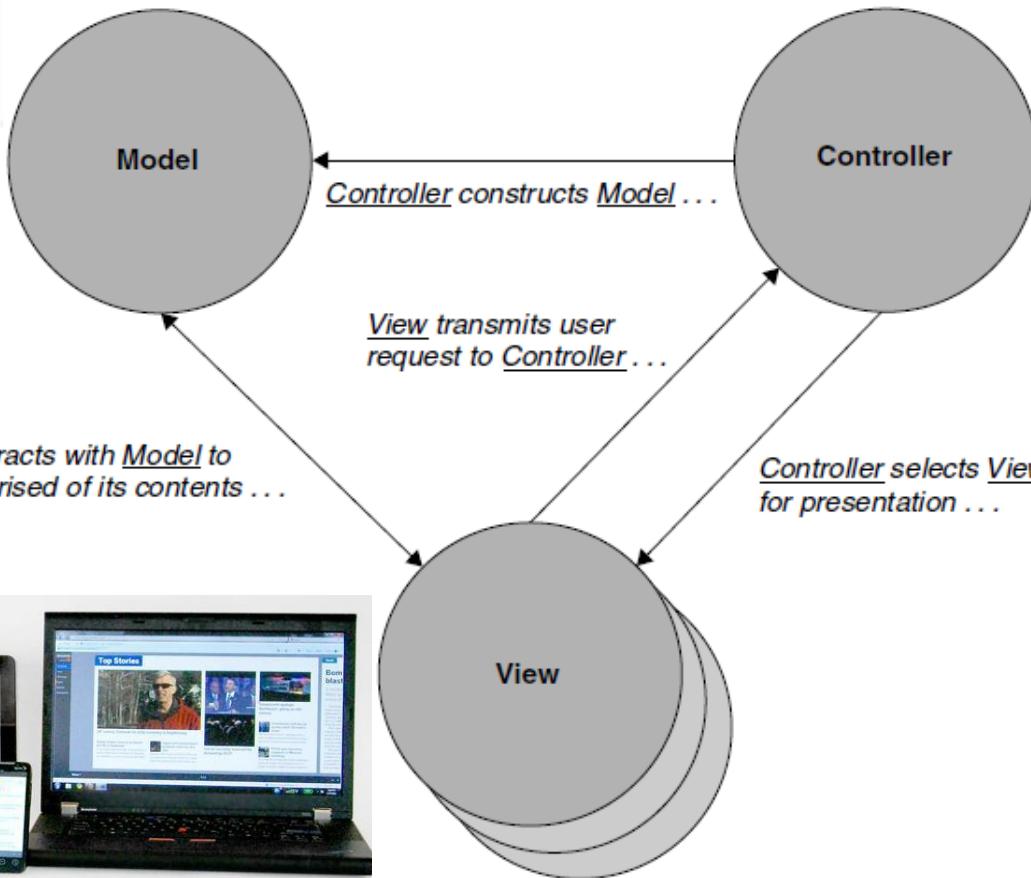
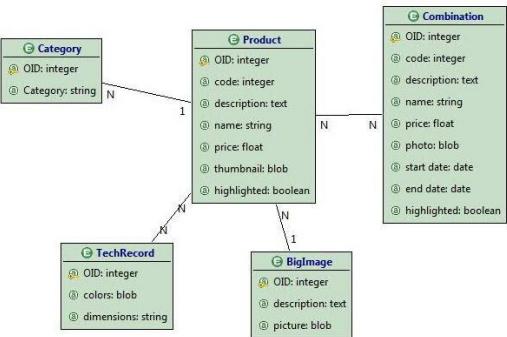
- Escribir y compilar el código de los componentes Web (servlet y JSP) y las clases referenciadas por el código de los componentes Web.
- Crear algún recurso estático (por ejemplo, imágenes o páginas HTML)
- Crear un descriptor de despliegue (web.xml)
- Construir la aplicación Web (archivo .war o directorio de despliegue)
- Instalar de desplegar la aplicación Web en el contenedor Web

Especificación JSP. Patrón Modelo-Vista-Controlador

FLUJO DE CONTROL:



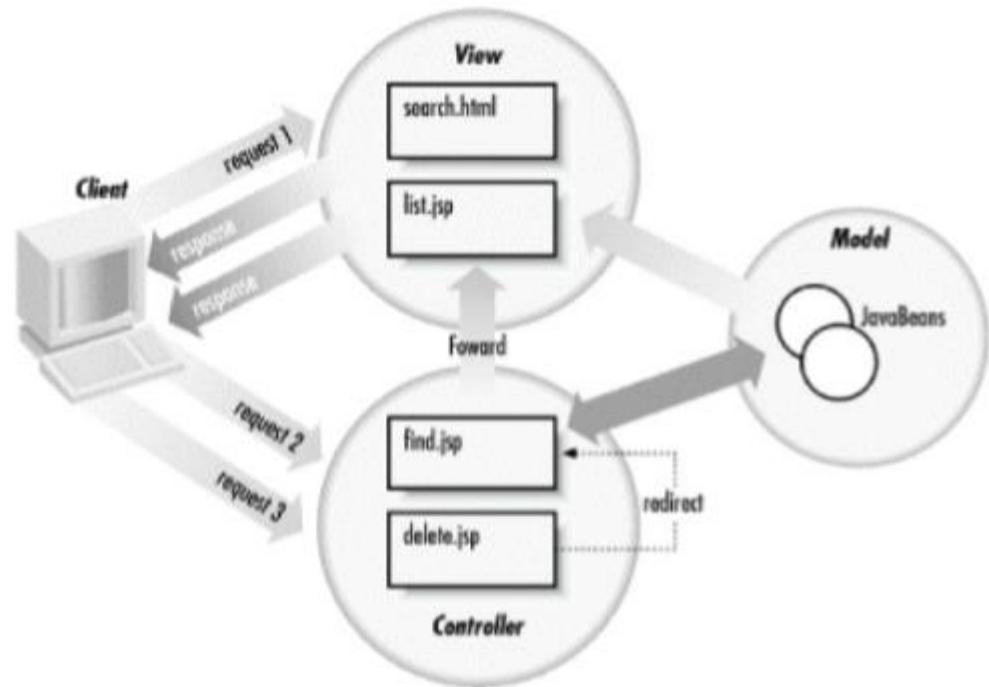
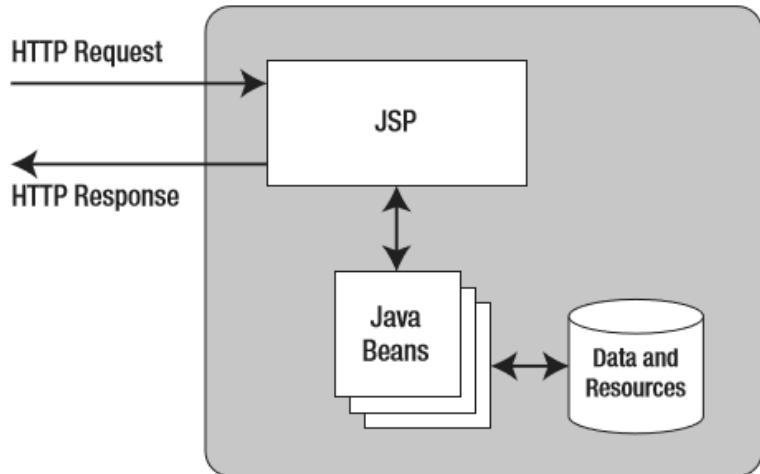
Especificación JSP. Patrón Modelo-Vista-Controlador



Especificación JSP. Patrón Modelo-Vista-Controlador

MVC con JSP y JavaBeans

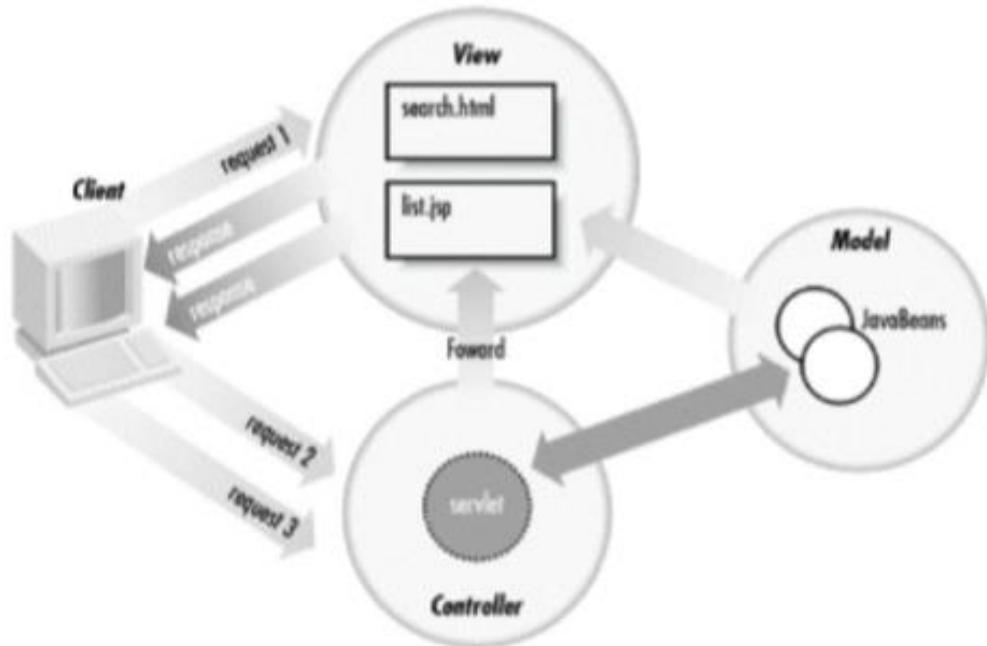
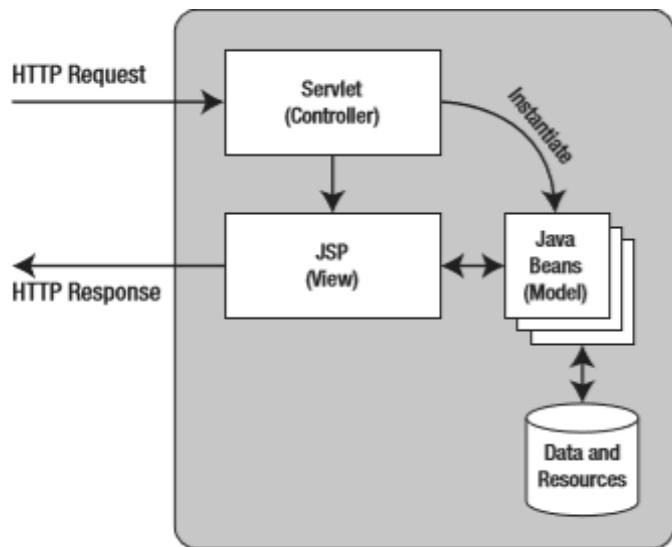
- El modelo MVC se puede implementar con páginas JSP (**presentación y controlador**) y JavaBeans (**modelo y lógica de negocio**).
- Es bueno utilizarlo para aplicaciones pequeñas o prototipo de aplicaciones grandes; y una vez que la futura aplicación esté estable, utilizar Servlets y EJB.



Especificación JSP. Patrón Modelo-Vista-Controlador

MVC con JSP, JavaBeans y servlet

Utilizar archivos **JSP** para la presentación, los **servlets** para el manejo de la **lógica del negocio** y proceso de peticiones (**controladores**) y los **Javabeans** para los **datos (modelo)**, es una manera poderosa de desarrollar una aplicación bien estructurada, fácil de mantener y lista para evolucionar.





JSP. Actividad 4.1 – Caso estudio: Búsqueda de Libros



Se solicita una aplicación web que permita buscar los libros de una biblioteca. Cada libro está representado con la siguiente información:

* Título, * Autor, * Editorial, * Año publicación, * Resumen, * Url Foto

La vista de la página web inicial debe ser similar al siguiente:

Titúlo	Autor	Editorial	Publicación
BEGINNING JSP, JSF AND TOMCAT	GIULIO ZAMBON	SPRINGER SCIENCE + BUSINESS MEDIA NEW YORK	2012

El usuario debe poder buscar un libro por un título que coincida con las palabras de búsqueda.

Al presionar el botón Buscar se debe mostrar el resultado en una tabla que contendrá:

- ✓ el título del libro
- ✓ el apellido y nombre del autor del libro
- ✓ La editorial
- ✓ año publicación

Además:

Al seleccionar un libro de la tabla, la webapp debe mostrar la información completa de dicho libro.

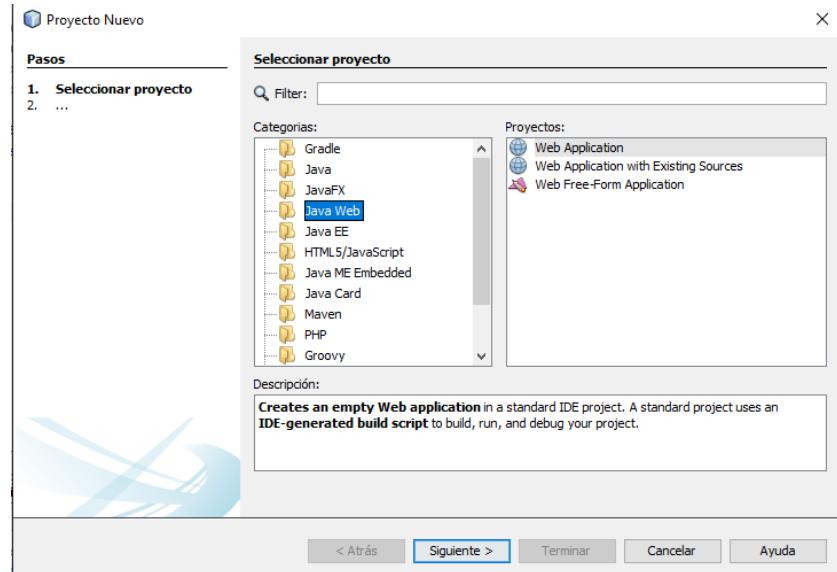


JSP. Actividad 4.1 – Desarrollo Paso a paso



1. Crear el proyecto de Aplicación Web Java

- En el IDE Netbeans 8.2 o posterior, dirigirse a la opción de menú: Archivo → Nuevo Proyecto y seleccionar Java Web → Web Application y luego hacer clic en el botón “Siguiente”



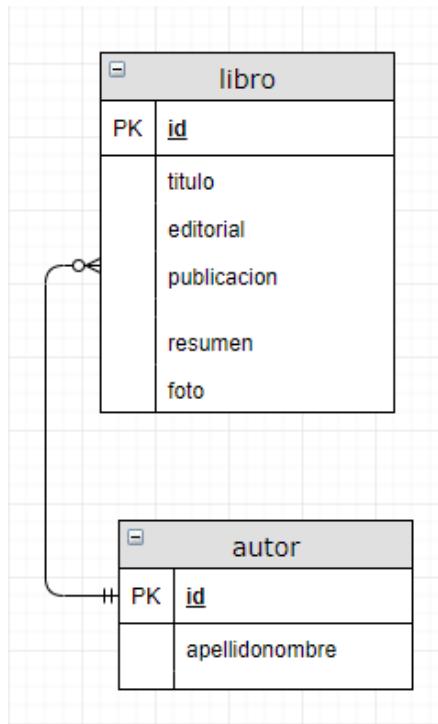
- En la ventana que se muestra a continuación ingresar el Nombre del Proyecto, en este caso “libros” y seleccionar el directorio de ubicación deseado.

JSP. Actividad 4.1 – Desarrollo Paso a paso



2. Crear Base de Datos

- Mediante el gestor de B.D. PosgreSQL, crear la base de datos “libros” y las tablas “autor” y “libro”, de acuerdo al siguiente diagrama ER:
- Los campos “id” deben ser definidos como clave primaria y a su vez, se debe definir la clave foránea correspondiente sobre el campo “id_autor” de la tabla “libro”

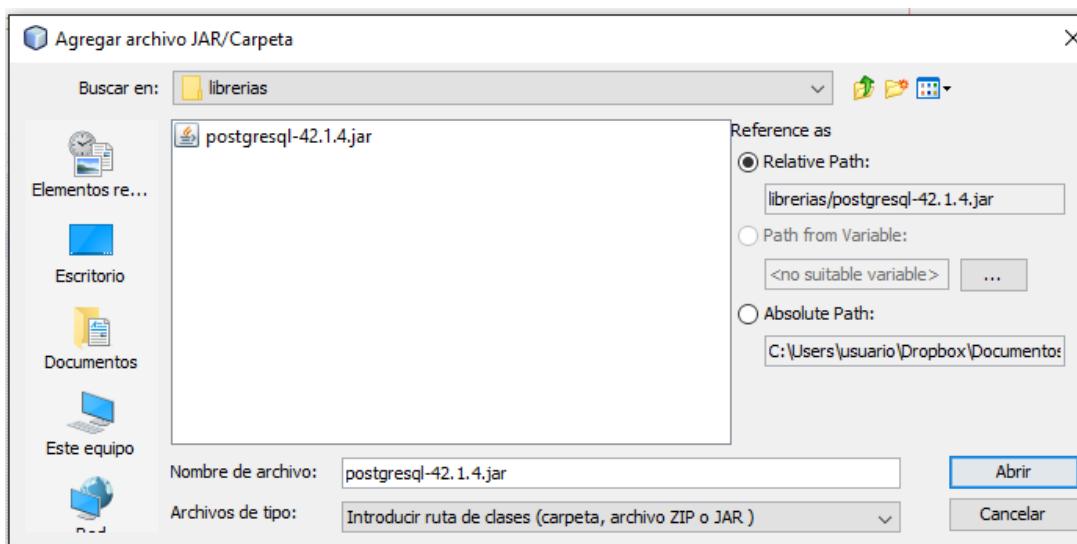


JSP. Actividad 4.1 – Desarrollo Paso a paso



3. Especificar la conexión a la B.D. desde el proyecto Java Web

- Descargar el Driver JDBC PostgreSQL desde la url:
<https://jdbc.postgresql.org/download.html>
- Agregar el driver a la librería del proyecto. Hacer clic derecho sobre la carpeta del proyecto “Libraries” y elegir la opción: “Agregar archivo Jar/Carpeta”.
- Luego seleccionar la ubicación del archivo Jar descargado y hacer un clic en el botón “Abrir”.





JSP. Actividad 4.1 – Desarrollo Paso a paso



3. Especificar la conexión a la B.D. desde el proyecto Java Web

- Editar el archivo “Web Pages -> META-INF -> context-xml” y agregar el siguiente elemento XML:

```
<Context antiJARLocking="true" path="/libros">  
    <Resource auth="Container"  
        driverClassName="org.postgresql.Driver" maxActive="100"  
        maxIdle="30" name="jdbc/libros" password="xxxxpasword"  
        type="javax.sql.DataSource"  
        url="jdbc:postgresql://127.0.0.1:5432/libros"  
        username="postgres"/>  
</Context>
```

- Mediante esta configuración se crea en el contexto del Servidor Web un pool de conexiones con nombre “jdbc/libros” a la B.D. “libros” ubicada en el localhost



JSP. Actividad 4.1 – Desarrollo Paso a paso



4. Crear las clases del Modelo

- Dentro de la carpeta “Source Packages”, crear la carpeta “model”
- Luego hacer un clic derecho en la carpeta creada y elegir la opción “New -> Java Class...”. Mediante esta opción crear las clases:
 - Clase **Autor** con los atributos: int id, String apellidoNombre
 - Clase **Libro** con los atributos: int id, String titulo, Autor autor, String editorial, int publicacion, String resumen, String foto
- Generar para todos los atributos los métodos Setter y Getter.



JSP. Actividad 4.1 – Desarrollo Paso a paso



5. Crear las clases DAO que se conectarán a la B.D.

- Dentro de la carpeta “Source Packages”, crear la carpeta “dao”
- Luego hacer un clic derecho en la carpeta creada y elegir la opción “New -> Java Class...”. Mediante esta opción crear las clases:
 - ConexionDAO
 - LibroDAO



JSP. Actividad 4.1 – Desarrollo Paso a paso



5. Crear las clases DAO que se conectarán a la B.D.

- La clase ConexionDAO debe ser similar a:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

public class ConexionDAO {
    private DataSource jeeDs;
    protected DataSource getDs() throws Exception{
        if (jeeDs == null) {
            Context ctx = new InitialContext();
            jeeDs = (DataSource)
ctx.lookup("java:comp/env/jdbc/libros");
            return jeeDs;
        }
    }
    public void cierra_todo(ResultSet rs, Connection
conn, PreparedStatement ps) throws Exception{
        if(rs != null) {rs.close();}
        if(conn != null) {conn.close();}
        if(ps != null) {ps.close();}
    }
}
```



JSP. Actividad 4.1 – Desarrollo Paso a paso



5. Crear las clases DAO que se conectarán a la B.D.

- La clase LibroDAO debe ser similar a:

```

import java.sql.Connection; import java.sql.PreparedStatement;
import java.sql.ResultSet; import java.util.ArrayList;
import java.util.List; import model.Libro; import model.Autor;

public class LibroDAO extends ConexionDAO {
    public List<Libro> getLibrosPorTitulo(String titulo) throws Exception{
        Connection conn = null; ResultSet rs=null;
        PreparedStatement ps=null; Libro libro = null;
        List<Libro> lista=new ArrayList<Libro>();
        titulo = "%" + titulo.trim().toUpperCase() + "%";

        try {
            conn = this.getDs().getConnection();
            String vsql="SELECT libro.*,autor.id as idAutor,autor.apellidonombre FROM libro LEFT
OUTER JOIN autor on (libro.id_autor=autor.id) where libro.titulo like ? ORDER BY libro.titulo";
            ps = conn.prepareStatement(vsql);
            ps.setString(1, titulo);
            rs = ps.executeQuery();
            while (rs.next()) {
                libro = new Libro();
                libro.setId(rs.getInt("id"));
                libro.setTitulo(rs.getString("titulo"));
                libro.setAutor(new Autor(rs.getInt("idAutor"),rs.getString("apellidonombre")));
                libro.setEditorial(rs.getString("editorial"));
                libro.setPublicacion(rs.getInt("publicacion"));
                lista.add(libro);
            }
            rs.close(); rs=null; ps.close(); ps=null; conn.close(); conn=null;
        } catch (Exception e) {throw e;} finally {cierra_todo(rs, conn, ps);}
    return lista; } }
```

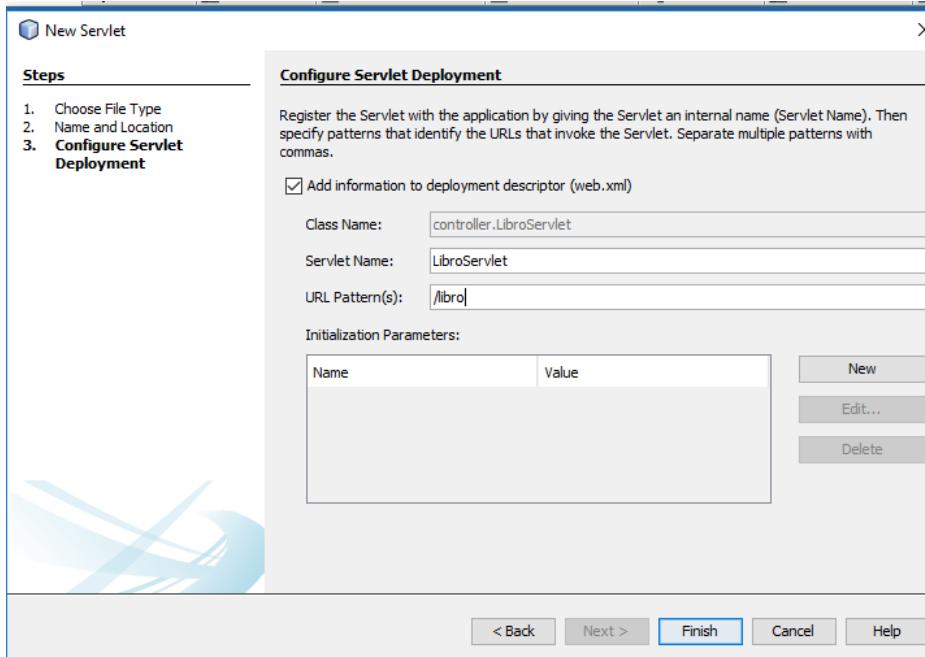


JSP. Actividad 4.1 – Desarrollo Paso a paso



6. Crear el Servlet que actuará de Controlador

- Dentro de la carpeta “Source Packages”, crear la carpeta “controller” y luego hacer un clic derecho y elegir la opción “New -> Servlet...”
- En la pantalla que se muestra, ingresar el nombre de la clase Servlet, en este caso “LibroServlet”. Luego hacer un clic en el botón “Next>”
- A continuación, se muestra la pantalla para configurar el despliegue del Servlet. Para ello se debe ingresar un nombre para el servlet y un patrón de URL para convocarlo. Finalmente hacer un clic en el botón “Finish”:





JSP. Actividad 4.1 – Desarrollo Paso a paso



6. Crear el Servlet que actuará de Controlador

- Verificar que se haya creado la clase **controller.LibroServlet.java**, como así también el archivo “WEB-INF -> web.xml”, con la configuración del nombre y URL mapping del Servlet.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi
3      <servlet>
4          <servlet-name>LibroServlet</servlet-name>
5          <servlet-class>controller.LibroServlet</servlet-class>
6      </servlet>
7      <servlet-mapping>
8          <servlet-name>LibroServlet</servlet-name>
9          <url-pattern>/libro</url-pattern>
10     </servlet-mapping>
11     <session-config>
12         <session-timeout>
13             30
14         </session-timeout>
15     </session-config>
16 </web-app>
17
```



JSP. Actividad 4.1 – Desarrollo Paso a paso



6. Crear el Servlet que actuará de Controlador

- **El método doPost() del servlet debe:**
 - Recibir el parámetro “titulo” que se envía desde el formulario de búsqueda.
 - Crear un objeto del tipo LibroDAO y convocar el metodo getLibrosPorTitulo(titulo), cuyo resultado hay que almacenar en una variable tipo Lista.
 - Agregar al objeto “request” el atributo "listaLibros“, con el valor de la variable resultante del punto anterior.
 - Redireccionar a la vista "index.jsp“ mediante un forward.
- **El método doGet() del servlet debe:**
 - Recibir el parámetro “id” que se envía mediante el link del libro.
 - Crear un objeto del tipo LibroDAO y convocar el metodo getLibroById(id), cuyo resultado hay que almacenar en una variable tipo Libro.
 - Agregar al objeto “request” el atributo “libro“, con el valor de la variable resultante del punto anterior.
 - Redireccionar a la vista " showBook.jsp" mediante un forward.



JSP. Actividad 4.1 – Desarrollo Paso a paso



7. Crear las vistas index.jsp y showBook.jsp

- Dentro de la carpeta “Source Packages”, crear la carpeta “Web Pages” y generar los archivos index.jsp y showBook.jsp
- **Desde la vista index.jsp se debe:**
 - Obtener el objeto (atributo) "listaLibros" enviado desde el servlet mediante el objeto “request” y guardarlo en una variable de tipo Lista de libros.
 - Iterar la variable resultante del punto anterior, y generar dinámicamente las filas de la tabla con los datos de los libros.
- **Desde la vista showBook.jsp se debe :**
 - Obtener el objeto (atributo) "listaLibros" enviado desde el servlet mediante el objeto “request” y guardarlo en una variable de tipo Libro.
 - Completar la ficha del libro con los datos específicos del libro seleccionado.