INAP
INSTITUTO NACIONAL DE ADMINISTRACIÓN PÚBLICA

Curso online: **Instalación, Configuración y Administración de Apache + Tomcat**

Módulo 1. Aplicaciones Web

Capítulo 2. Aplicaciones Web

Autores

Janine García Morera

Alexandra López de la Oliva Portugués

Julio Villena Román

Octubre de 2014

Índice de contenidos

CAPÍTULO 2 APLICACIONES WEB

2.1. ¿Qué es una aplicación Web?

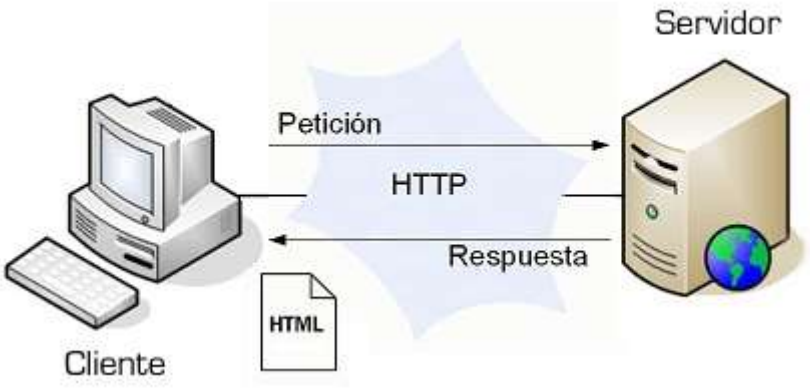

Figura 1.2.1: Aplicación Web

Una aplicación Web es cualquier aplicación que utiliza el Protocolo de Transferencia de Hipertexto o HTTP como principal protocolo de comunicación y de intercambio de información entre un cliente y un servidor.

Tal y como expresa la figura, la arquitectura está formada por:

- **Cliente web**: normalmente es un navegador que interpreta los documentos y las páginas y se las presenta al usuario.
  - → Actualmente el acceso a aplicaciones web se realiza mediante el uso de navegadores tipo MS Internet Explorer, Google Chrome o Mozilla Firefox.
  - → Además, existen otros tipos de clientes como robots, arañas (spiders), rastreadores (crawlers), etc.
- **Servidor web**: es quien presta el servicio. Recibe la petición del cliente y devuelve una respuesta.

HTML es un estándar para representar documentos de hipertexto en formato ASCII permitiendo especificar el formato, las referencias y los enlaces.

La Web se diseñó originalmente como un medio para suministrar contenidos por medio de páginas HTML estáticas. Cuando un navegador web envía una petición a un servidor web, este último se encarga de buscar la página solicitada en su sistema de archivos y devolverla al navegador. Sin embargo, lo que devuelve el servidor Web no tiene por qué ser siempre una

página HTML estática almacenada en el servidor, ya que puede tratarse de la salida de un programa que se ejecuta en el entorno del servidor Web.

Esto es lo que se ha hecho durante mucho tiempo con el uso de la interfaz **CGI** (Common Gateway Interface).

2.2. ¿Qué es un CGI?

CGI es la implementación que permite al usuario ejecutar aplicaciones de servidor desde su navegador.

Se define **CGI** como un **programa ejecutable a nivel de servidor con una interfaz que le permite ser ejecutado por el cliente usando un navegador a través del protocolo HTTP**.

El funcionamiento de esta tecnología es muy sencillo. Los scripts residen en el servidor, se llaman por el cliente a través del navegador, se ejecutan en el servidor y regresan la información de vuelta al usuario.

Esta tecnología tiene la ventaja de correr en el servidor cuando el usuario lo solicita por lo que es dependiente del servidor y no de la máquina del usuario.

La llamada a un programa CGI se realiza mediante una URL como la siguiente:

```
http://www.servidor.com/cgi-bin/programa1
```

Además, se puede enviar información al programa por medio de GET (en la propia URL) o mediante POST (de manera invisible al usuario, típicamente mediante formularios).

GET es recomendable en solicitudes cortas, simples y rápidas y POST para aquellas que requieran una mayor seguridad o un mayor envío de información.

Un ejemplo de una llamada a un script CGI puede ser:

```
http://www.servidor.com/cgi-bin/programa1?param1=valor1&param2=valor2
```

En esta URL, la parte cgi-bin es fija y está definida en el fichero de configuración del servidor web, y le dice al servidor que lo que viene después es una llamada a un ejecutable (en este caso llamado programa1) que se almacena en el directorio que mapea con el alias cgi-bin, y que recibe como información (GET) los parámetros param1 y param2, con valores valor1 y valor2 respectivamente. El servidor ejecuta dicho programa y devuelve el resultado al navegador. Este resultado deberá proporcionarse en un formato entendible por el navegador (normalmente HTML) y debe ser el propio programa el que se encargue de generar dicho formato.

Los programas que maneja el CGI pueden estar compilados en diferentes lenguajes de programación como php o Perl.

A continuación presentamos un ejemplo de un programa escrito en php:

```
<html>
<head>
<title>Hola Mundo PHP</title>
</head>
<body>
<p>
<?php
if(isset($_GET["name"]) && !empty($_GET["name"]))
  $name = $_GET["name"];
else
  $name = "Mundo";
echo "Hola ".htmlspecialchars($name);
?>
</p>
</body>
</html>
```

Y su equivalente en Perl.

```
#!/usr/local/bin/perl
use CGI;
$query = new CGI;
$name= $query->param("name");
if ($name eq "") {$name="Mundo";}
print <<"EOF";
<html>
<head>
<title>Hola Mundo CGI (Perl)</title>
</head>
<body>
<p>Hola, $name</p>
</body>
</html>
EOF
```

Ambos programas esperan como parámetro el nombre del usuario para saludarle en su salida, si no se introduce ningún nombre por parámetro imprimirán “Hola Mundo”.

CGI es una opción válida ya que permite generar contenidos dinámicos. Sin embargo presenta una serie de limitaciones, en especial en lo relacionado con la eficiencia: cada vez que se hace una petición a un programa CGI, se arranca un nuevo proceso. Si el programa CGI es relativamente pequeño, el costo de arrancar el proceso puede superar el tiempo de ejecución del mismo.

Por otro lado, CGI no proporciona servicios que faciliten la implementación de sistemas basados en usuarios: el mantenimiento de la identidad del cliente, la información del usuario, etc., debe ser realizado mediante utilidades de programación.

Desde este punto de vista, CGI presenta las siguientes ventajas e inconvenientes:

- **Ventajas**:
  - Permiten generar contenido dinámico.
  - Son sencillos de desarrollar e implementar.
  - Son válidos para pequeños sistemas o con pocos usuarios.
- **Inconvenientes**:
  - No permiten separar la presentación de la lógica de negocio.
  - Cada ejecución de un CGI implica un nuevo proceso.
  - Si un programa falla puede afectar a todo el servidor.
  - CGI no proporciona servicios añadidos (no se trata de una plataforma).

Con la idea de solucionar todos los problemas a la hora de desarrollar aplicaciones web potentes y portables, Sun definió la plataforma J2EE.

2.3. Plataforma J2EE

La plataforma J2EE (Java 2 Enterprise Edition) es la propuesta que ha realizado Sun Microsystems para el desarrollo e implementación de aplicaciones corporativas estructuradas en varios niveles.

La plataforma **J2EE** se apoya por completo en el lenguaje Java, que es **definido por Sun** en su libro blanco de Java **como "sencillo, orientado a objetos, seguro, portable y multitarea"**.

No hay que confundir J2EE (Java 2 Enterprise Edition) con J2SE (Java 2 Standard Edition). J2EE se apoya para funcionar en J2SE, que incluye los binarios de Java (como es la JVM, Java Virtual Machine o el compilador) y las bibliotecas básicas de Java.

J2EE proporciona los siguientes **componentes**:

- La definición, en forma de API de Java, de una estructura de desarrollo de componentes web (Servlets y JSPs) y de componentes activos (EJBs).
- La definición, en forma de API de Java, de un conjunto de servicios que son utilizados como herramientas por los componentes (JDBC, JTA, JNDI, JMS, RMI/IIOP, JavaMail, XML).

- Un modelo de creación de módulos de componentes web (`.war`), de módulos EJB (`.jar`) y de módulos corporativos (`.ear`) asociados cada uno de ellos a descriptores de despliegue en formato XML.
- Contenedores (web y EJB) para la realización y ejecución de los componentes.

Se puede definir **J2EE** como una **norma que define un grupo de API basadas en Java para la realización de aplicaciones basadas en web**. Estas son las API principales:

- **Servlet**: esta API proporciona los elementos necesarios para la creación de componentes Web dinámicos en Java.
- **JSP**: permite crear páginas Web dinámicas con un esqueleto HTML y datos generados dinámicamente por código Java.
- **EJB** (Enterprise Java Bean): proporciona una estructura de componentes en una arquitectura distribuida en n capas proporcionando un conjunto de servicios asociados a los componentes cuya gestión se delega al contenedor de EJBs (transacciones, persistencia, etc.).
- **RMI/IIOP** (Remote Method Invocation/Internet Inter-Orb Protocol): permite el diseño de aplicaciones distribuidas en Java.
- **JMS** (Java Messages Service): permite acceder a un servicio de mensajería para gestionar de forma asíncrona las llamadas a los componentes básicos.
- **JDBC** (Java Database Connectivity): permite la obtención de conexiones para la utilización de fuentes de datos.
- **JTA** (Java Transaction API): permite utilizar la gestión de las transacciones en las aplicaciones distribuidas, dividiendo una misma transacción entre distintos componentes EJBs o compartiendo una misma transacción entre distintas conexiones de bases de datos.
- **JNDI** (Java Naming and Directory Interface): permite el acceso a servicios de datos, como puede ser LDAP. Esta API ha pasado actualmente a la especificación J2SE.
- **JavaMail**, ofrece a las aplicaciones funcionalidades de gestión de correo electrónico.
- **JMX** (Java Management Extensions): proporciona un mecanismo para controlar y gestionar aplicaciones de forma remota en tiempo de ejecución.

Conceptos importantes
La plataforma J2EE es 100 % Java, está definido por Sun como “sencillo, orientado a objetos, seguro, portable y multitarea”.
Se puede definir J2EE como una norma que define un grupo de API basadas en Java para la realización de aplicaciones basadas en web. Sus principales API son las siguientes: Servlet, JSP, EJB, RMI/IIOP, JMS, JDBC, JTA, JNDI, JavaMail y JMX.

2.4. Aplicaciones Web J2EE

Una **aplicación Web** es un **conjunto de recursos Web que participan en el funcionamiento de la propia aplicación**. Se compone de:

- Componentes de servidor dinámicos: Servlets y JSPs.
- Bibliotecas de clases Java.
- Elementos Web estáticos: páginas HTML, imágenes, sonidos, etc.
- Componentes de cliente dinámicos: applets, JavaBeans y clases.
- Un descriptor de desarrollo y de configuración de la aplicación Web en forma de uno o varios archivos en formato XML (fundamentalmente `web.xml`), que contiene información que permite definir el entorno de ejecución de la aplicación Web, así como relacionar entre sí los componentes.

El despliegue de una aplicación Web sigue la siguiente estructura de directorios:

- **Carpeta `webapp`** (aplicación). Representa la raíz de la aplicación Web. El nombre de este directorio es el nombre del contexto de la aplicación Web. En esta carpeta se ubica, por lo general:
  - La página de inicio de la aplicación (`index.html` o `index.jsp`), otras páginas HTML, las imágenes e iconos que se usen en la aplicación, páginas JSPs, etc. que es contenido público del servidor de aplicaciones.
  - La carpeta **`META-INF`**, que contiene el **archivo `manifest.mf`**. Ambos son generados por la utilidad jar.exe que permite realizar el fichero comprimido de distribución `.war` y no son necesarios para el servidor de aplicaciones (en la imagen siguiente no se muestra).
  - La carpeta **`WEB-INF`** que representa la parte privada de la aplicación Web conteniendo los recursos que no son descargables para el cliente. Contiene el descriptor de despliegue de la aplicación Web (fichero `web.xml`) y las **subcarpetas `classes`** y **`lib`** con los archivos `.class` y `.jar` de los servlets de la aplicación Web y otras clases java que son utilizadas por la aplicación. Ésta puede usar además clases java almacenadas en los directorios comunes.

```
aplicación\
   index.html
   login.jsp
   images\ *.gif
   doc\ *.pdf
   META-INF\MANIFEST.MF
   WEB-INF\
      web.xml   <- Deployment Descriptor
      classes\ *.class   <- clases ejecutables de la aplicación,
los servlets, los javaBeans…
      lib\ *.jar
```

Conceptos importantes	
Componentes de una aplicación Web	Servlets y JSPs
	Bibliotecas de clases Java
	HTML, imágenes, sonidos, etc.
	Applets, JavaBeans y clases.
	Uno o varios archivos en formato XML (fundamentalmente `web.xml`), que contiene información que permite definir el entorno de ejecución de la aplicación Web, así como relacionar entre sí los componentes.
Estructura de directorios del despliegue de una aplicación Web	Carpeta `webapp` (aplicación). Es de acceso público.
	La carpeta `META-INF`.
	La carpeta `WEB-IN` (`fichero web.xml`). Es de acceso privado.
	Las subcarpetas `classes` y `lib` de `WEB-INF`.

2.5. Modelo Vista Controlador (MVC)

Es un **modelo de implementación de aplicaciones Web** cuya base es la separación de la parte lógica de la de presentación.

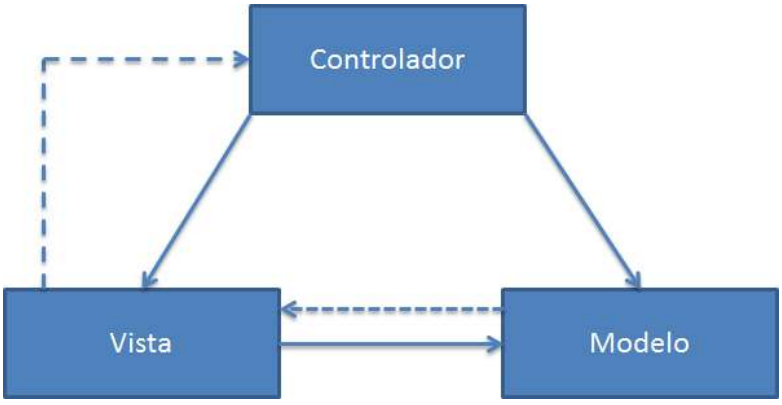

Figura 1.2.2: Esquema Modelo Vista Controlador

- **Modelo**: objeto que maneja los datos del programa y controla todas sus transformaciones. El Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo. En resumen, el modelo encapsula el estado de la aplicación.

- **Vista**: objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa con el Modelo a través de una referencia al propio Modelo (solicita actualizaciones de los modelos). En resumen, la vista representa el modelo de datos.
- **Controlador**: objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el Modelo a través de una referencia al propio Modelo. En resumen, el controlador define el comportamiento de la aplicación.

En la **arquitectura MVC:**

- El modelo está representado por los EJBs y/o los JavaBeans.
- La vista está representada por los JSPs.
- El controlador está representado por los servlets.

El funcionamiento de la arquitectura MVC es el siguiente:

1. El cliente envía una petición HTTP con destino a un Servlet.
2. El servlet recupera los datos enviados en la petición HTTP y delega el proceso de los datos a componentes EJB y/o JavaBean.
3. El componente EJB o JavaBean puede acceder a una fuente de datos (base de datos o directorio LDAP) si se requiere.

   Retoma el control el **paso 2**. Una vez concluidos los procesos, los componentes devuelven el resultado al servlet que los almacena en un entorno concreto (sesión, petición,...).
4. El servlet envía el conjunto del proceso de la petición al JSP.
5. El JSP recupera los datos almacenados por el servlet, genera la respuesta HTTP y la envía al cliente.

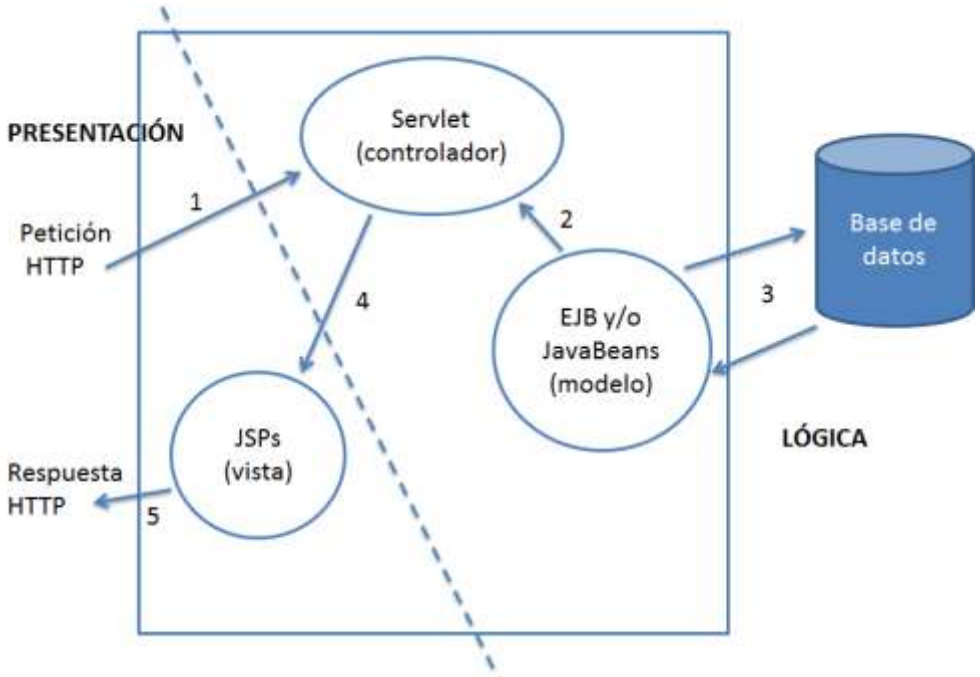

Figura 1.2.3: Arquitectura MVC

Conceptos importantes	
Es un modelo de implementación de aplicaciones Web cuyo objetivo es la separación de la parte lógica de la de presentación.	
Arquitectura MVC	El modelo encapsula el estado de la aplicación: EJBs y/o JavaBeans.
	La vista representa el modelo de datos: JSPs.
	El controlador define el comportamiento de la aplicación: servlets.

2.6. API Servlet y API JSP

Un **servlet** es un **programa Java usado para extender las capacidades de un servidor** a cuyas aplicaciones se accede mediante un modelo petición-respuesta (request-response).

Un servlet es una clase Java que permite realizar la gestión de flujo HTTP en entrada/salida. Un servlet debe devolver respuesta a una pregunta enviada por un cliente, una vez realizado el proceso que corresponda.

A pesar de que un servlet podría utilizarse en cualquier servidor de este tipo, actualmente su uso se ha extendido básicamente a las aplicaciones que se ejecutan bajo un servidor web. En este caso, los servlets se ejecutan dentro del entorno del servidor web, y por lo tanto amplían la funcionalidad del mismo al permitir la generación de páginas HTML de manera dinámica. Estos servlets reciben y responden a peticiones de clientes web a través del protocolo HTTP, el Protocolo de Transferencia de Hipertexto.

Con los servlets, la Máquina Virtual de Java permanece en ejecución y administra cada petición mediante un ligero subproceso de Java, no con un pesado proceso del sistema operativo. En resumen, **los servlets son la respuesta de la tecnología Java a la programación CGI tradicional**, y nos proporciona un ambiente de desarrollo más eficiente, seguro, portable y robusto del lado del servidor, a través del API Java Servlet.

La inicialización de un servlet se realiza mediante el método **`init()`** que se ejecuta una vez por cada instancia de servlet y carga todos los recursos que necesita ese servlet y realiza cualquier inicialización que este precise.

Posteriormente la invocación del servlet se realiza mediante el método **`service()`**, que se invoca por cada solicitud del cliente. Cada servlet puede procesar varias solicitudes de varios clientes.

Por último, la interfaz Servlet proporciona el método **`destroy()`** para eliminar la instancia del servlet, liberando todos los recursos que se adquirieron mediante la invocación del método `init()`.

Para que se puedan ejecutar los servlets es preciso tener un contenedor de servlets que trabaje en conjunción con el servidor web. Este contenedor se encarga, entre otras cosas, de administrar la carga (`init()`) y descarga (`destroy()`) de los servlets dentro del servidor web. Es muy común usar el término servlet engine (algo así como "motor de servlets"), para referirse a dicho contenedor.

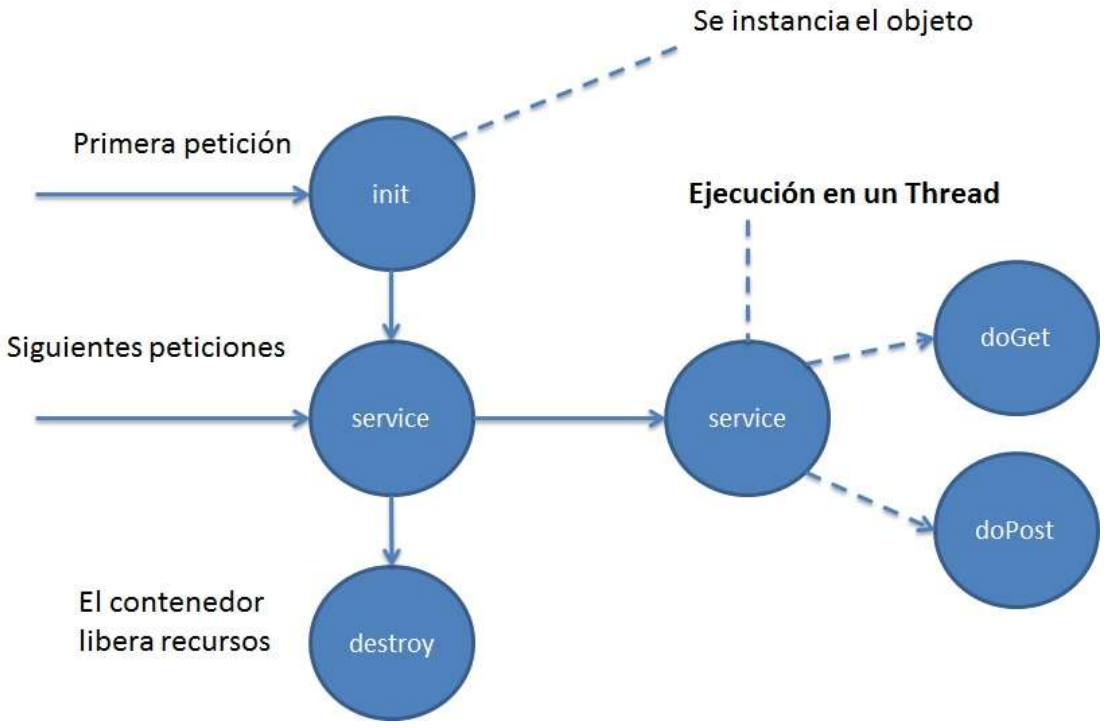

Figura 1.2.4: Ciclo de vida de un Servlet

Un contenedor de servlets debe proporcionar al servlet los servicios de carga y descarga, además de otros servicios que permitan completar el ciclo de vida de éste, como es el caso del acceso a bases de datos y a otros recursos de la aplicación. Pero, además, debe proporcionar los recursos necesarios que permitan al servlet el mantenimiento de la sesión: una conexión HTTP carece de estado, y una vez devuelta la respuesta al cliente el servidor no puede reconocer la identidad de éste cuando realice otra solicitud.

El contenedor provee de los mecanismos necesarios para que el sistema reconozca la identidad del cliente (previamente identificado) mediante el uso de cookies o mediante la reescritura del código HTML que devuelve al cliente.

La invocación a un servlet se puede realizar de dos formas: mediante una URL que incluya la llamada al nombre del servlet (que puede ser completo o no) o mediante una ruta de contexto que se asigna al servlet en los ficheros de configuración.

Tomcat proporciona el contenedor de servlets necesario para el trabajo con los Java Servlets, y además puede ser usado en conjunción con algunos otros servidores web, tales como **Apache**, aunque también permite una instalación independiente (standalone) que es perfectamente capaz de servir servlets en instalaciones pequeñas, sin mucha carga estática o, simplemente, en entornos de desarrollo.

La versión Tomcat que utilizaremos en este curso (7.0.56) soporta la especificación Servlet 3.0. de Sun Microsystems.

La especificación JSP (JavaServer Pages) se refiere a documentos HTML que contienen código Java y que Tomcat (y los demás contenedores de servlets) compilan como servlets la primera vez que se invocan, aunque se almacenan como código fuente.

Los servlets se utilizan sobre todo para procesar lógica, pero es una mala práctica usarlos para crear contenido (presentación en HTML), ya que el incluir tags HTML en el código de un servlet obliga a tener que modificar dicho código y recompilar el servlet ante cualquier cambio en la presentación. Para separar la lógica de la presentación se creó la especificación JSP. Como en el caso de los servlets, precisan de un contenedor para ser ejecutados, que realiza el paso previo de conversión a servlet y compilación. A partir de la versión 5.5.17, Tomcat proporciona ese contenedor y la versión 8.0 soporta las especificaciones JSP 2.3 de Sun Microsystems.

A continuación mostramos un ejemplo de un servlet:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse
res)throws ServletException, IOException {
```

```
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    String name= req.getParameter("name");
    if (name == null) name = "Mundo";
    out.println("<html>");
    out.println("<head><title>Hola Mundo
Servlet</title></head>");
    out.println("<body>");
    out.println("<p>Hola" + name +  "</p>");
    out.println("</body></html>");
  }
}
```

Y su equivalente en JSP. Ambos funcionan como los ejemplos vistos anteriormente en CGI, esperan recibir el nombre del usuario por parámetro y si no lo obtienen imprimen “Hello World”.

```
<html>
<head>
<title>Hola Mundo JSP</title>
</head>
<body>
<p>
<% String name = request.getParameter("name");
if (name == null) name = "Mundo";
%>
Hola, <%= name %>!
</p>
</body>
</html>
```

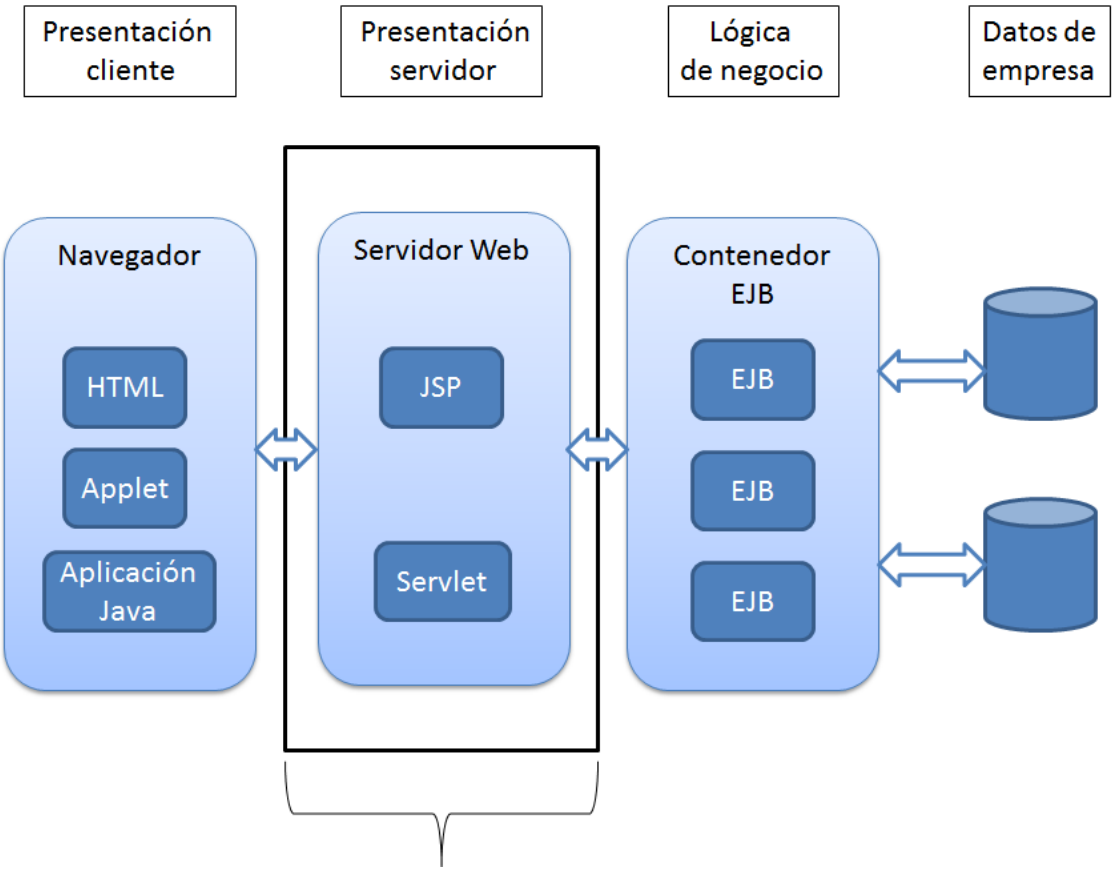

Figura 1.2.5: Escenario de uso JSP y Servlet

2.6.1. Capa Web

Dentro de la especificación J2EE, nuestra instalación cubrirá la llamada "Capa Web": podremos servir contenido estático (páginas HTML, imágenes, ficheros,...), desde Apache, y podremos ejecutar servlets y JSPs desde Tomcat.

Una implementación de API J2EE se define como un Servidor de Aplicaciones. Tomcat no es capaz de proporcionar la capa EJB ni otras APIs J2EE, por lo que no es propiamente un servidor de aplicaciones, sino un contenedor de servlets con soporte para otras APIS como JNDI y JMX. Sun Java System Application Server, Bea Weblogic Server, etc., sí operan en esta capa.

2.6.2. ¿Cuál es el rol de un servidor de aplicaciones?

El rol de un servidor de aplicaciones es hacer funcionar aplicaciones distribuidas a base de componentes Java (Servlet, JSP, EJB), haciéndolas accesibles a los clientes Web (navegadores) y a las aplicaciones corporativas escritas en Java.

El servidor debe encargarse de la creación y de la carga en memoria de las instancias de los componentes, así como de la gestión de una cola de espera para satisfacer las peticiones de los clientes.

En un servidor de aplicaciones físico (un servidor) se pueden instalar y configurar varios servidores de aplicación lógicos, cada uno de los cuales puede contener dos elementos: un contenedor Web (llamado comúnmente "motor de Servlet/JSP") y un contenedor de EJB.

El rol de los contenedores es poner al servicio de las peticiones de los clientes aquellos servicios que les corresponden.

La mayor parte de los servidores de aplicación están compuestos de un contenedor Web y de un contenedor de EJB, como es el caso de IBM WebSphere Application Server y BEA WebLogic. Las soluciones open source normalmente sólo ofrecen uno de los tipos de contenedor, como es el caso del contenedor Web Apache Tomcat, de JBoss y del contenedor de EJB Evidian JOnAS.

Conceptos importantes	
Un servlet es un programa Java usado para extender las capacidades de un servidor a cuyas aplicaciones se acceden mediante un modelo petición-respuesta (request-response).	
Un servlet es una clase Java que permite realizar la gestión de flujo HTTP en entrada / salida. Un servlet debe devolver respuesta a una pregunta enviada por un cliente, una vez realizado el proceso que corresponda.	
En resumen, los servlets son la respuesta de la tecnología Java a la programación CGI tradicional, y nos proporciona un ambiente de desarrollo más eficiente, seguro, portable y robusto del lado del servidor, a través del API Java Servlet.	
API servlet	Inicialización: `init()`.
	Invocación: `service()`.
	Eliminación de una instancia: `destroy()`.
El rol de un servidor de aplicaciones es hacer funcionar aplicaciones distribuidas a base de componentes Java (Servlet, JSP, EJB), haciéndolas accesibles a los navegadores y a las aplicaciones escritas en Java.	

2.7. ¿Por qué Apache y Tomcat?

Apache Tomcat, en sus últimas versiones (de la 4.0.x en adelante), es perfectamente capaz de servir el contenido dinámico (su función principal como motor de servlets) y también el estático (haciendo de servidor web). Puede haber muchas razones para integrar Apache con Tomcat, pero depende del uso que se vaya a dar a la instalación podría ser conveniente o no: en un entorno de pruebas o desarrollo, por ejemplo, podría no tener sentido.

Algunas de las razones que se pueden valorar para realizar la integración son las siguientes:

- Para servir más rápido el contenido estático de la webapp. Apache es un servidor web específico, desarrollado en C y por lo tanto de ejecución más rápida que Tomcat, que está desarrollado en Java. Sin embargo, esto es cada vez menos cierto: los motores http estáticos de Tomcat cada vez son más rápidos y en instalaciones con poca carga tienen un rendimiento similar.
- Para permitir servir, desde la misma máquina, otros tipos de aplicaciones web (por ejemplo, PHP, ASP), que pueden colgar de nuestro mismo Apache, sin estorbar a las aplicaciones Java con Tomcat.
- Para instalar un cluster de servidores Tomcat (veremos un ejemplo): un servidor web Apache hace de frontal y reparte peticiones de servlets y JSPs entre dos (o más) motores Tomcat, en la misma o en otra(s) máquina(s). Si una de las instancias de Tomcat falla, Apache remite las peticiones a las otras instancias activas, ignorando la instancia caída. Este tipo de configuración se podría conseguir con componentes específicos de balanceo (hardware o software).
- Apache tiene gran cantidad de módulos (**`mod_rewrite`**, **`mod_proxy`**) de los que Tomcat carece. Si se quiere hacer uso de su potencialidad en una aplicación web, es necesario realizar la integración.
- Para montar servidores virtuales (Tomcat permite los servidores virtuales en modo standalone, pero son más potentes y configurables los de Apache).

El **funcionamiento** de un sistema integrado **Apache-Tomcat** se define en el siguiente esquema:

1. El cliente hace una petición al puerto 80 (por defecto) del servidor.
2. Si ha pedido una página estática, la servirá directamente Apache.
3. Si ha pedido una página dinámica (JSP) o un Servlet, Apache, mediante el módulo de cooperación, le pasará la petición a Tomcat, que a su vez, llamará a la máquina virtual Java, que compila la página (si es una JSP) o ejecutará el servlet.
4. Si es necesario acceder a la base de datos, lo hará la máquina virtual Java mediante JDBC.
5. Hecho esto, Tomcat, de nuevo a través del conector, le devuelve los resultados (ya estáticos) a Apache, que los enviará al cliente.

En forma gráfica:

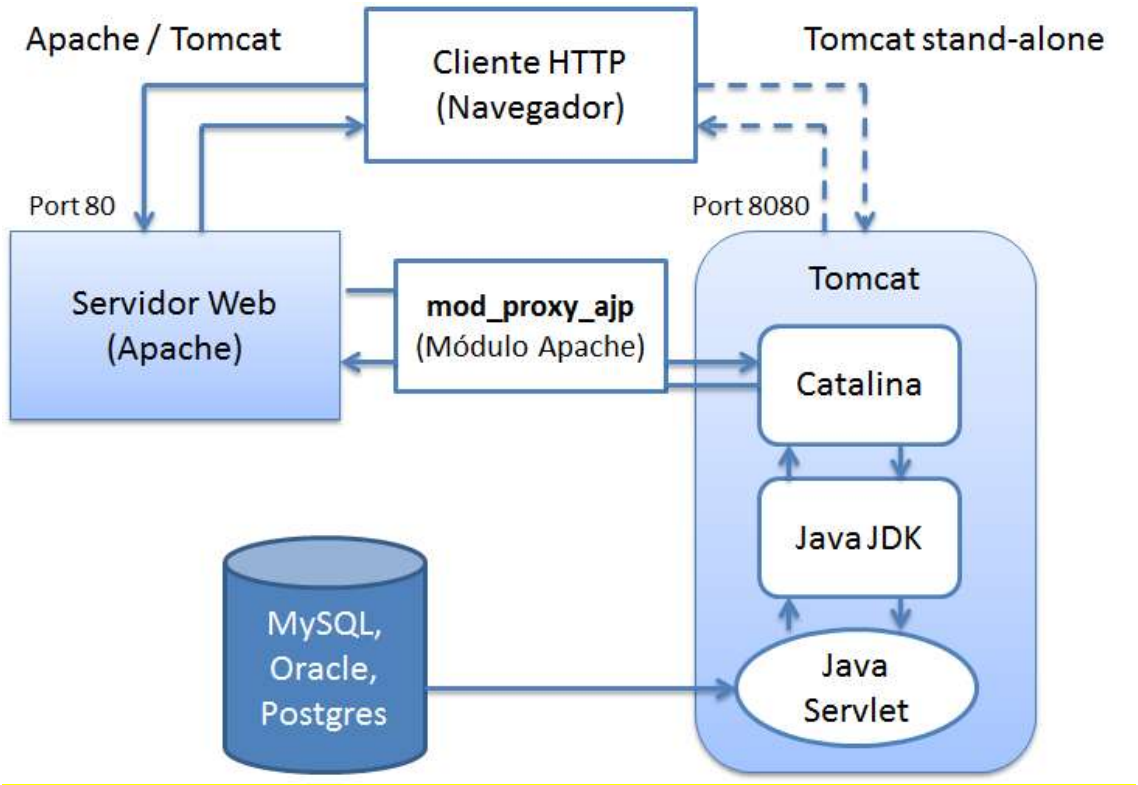

Figura 1.2.6: Proceso de un sistema integrado Apache - Tomcat

El cliente podría haber accedido al contenido dinámico (y al estático) de la web-application directamente, "saltándose" a Apache, si hubiera llamado al puerto 8080 (y si las reglas de control de acceso le permiten hacer esto).

Por tanto, "el mismo" Tomcat puede actuar, simultáneamente como servidor independiente "standalone" y como plug-in del servidor web Apache.